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

HUFFMAN ENCODING FOR LOSSLESS DATA COMPRESSION

GROUP PA 23

JEASYA DAVID GAMALAEL N. P.	**2306161965**
MUHAMMAD BRYAN FARRAS	**2306230975**
ALFONSUS TANARA GULTOM	**2306267126**

PREFACE

Kompresi data merupakan salah satu bidang penelitian yang memiliki keberlanjutan tinggi di era digital, di mana pertumbuhan data eksponensial membutuhkan metode yang efisien untuk transmisi. Dalam konteks aplikasi seperti pemrosesan gambar, algoritma “lossless compresion” menjadi sangat penting untuk memastikan keutuhan data tetap terjaga selama proses kompresi dan dekompresi.

Huffman Encoding, yang diperkenalkan oleh David A. Huffman pada tahun 1952, adalah salah satu algoritma pengkodean lossless paling terkenal dan banyak digunakan. Keunggulan utama algoritma ini adalah kemampuannya untuk menghasilkan kode optimal berdasarkan distribusi frekuensi simbol, sehingga memungkinkan pengurangan ukuran data secara signifikan tanpa kehilangan informasi.

Jurnal ini bertujuan untuk mengeksplorasi penerapan Huffman Encoding untuk kompresi data lossless menggunakan VHDL programming. Dengan kombinasi teori kompresi data dan implementasi perangkat keras, diharapkan jurnal ini dapat memberikan kontribusi signifikan terhadap pengembangan solusi kompresi data yang lebih cepat dan efisien.

Depok, December 16, 2024

Group PA-23

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Huffman Encoding bekerja dengan membangun struktur pohon biner berdasarkan frekuensi kemunculan karakter dalam data. Simbol yang lebih sering muncul diberikan kode biner yang lebih pendek, sementara simbol yang jarang muncul diberi kode lebih panjang. Dengan demikian, algoritma ini memaksimalkan efisiensi penyimpanan dan transmisi data, terutama pada dataset dengan distribusi frekuensi yang tidak merata.

Namun, implementasi algoritma ini pada perangkat lunak saja seringkali tidak cukup cepat untuk menangani ukuran data yang besar dalam waktu nyata. Untuk mengatasi keterbatasan ini, implementasi berbasis perangkat keras menggunakan VHDL (VHSIC Hardware Description Language) dan platform FPGA (Field-Programmable Gate Array) menawarkan solusi yang lebih cepat dan hemat daya.

FPGA memungkinkan pengkodean paralel dan pengelolaan pohon Huffman secara langsung pada level perangkat keras, yang secara signifikan meningkatkan efisiensi pemrosesan.

Proyek ini bertujuan untuk mengimplementasikan algoritma Huffman Encoding berbasis struktur pohon untuk proses kompresi data teks dengan memanfaatkan VHDL sebagai bahasa desain perangkat keras. Proyek ini memberikan solusi untuk menyimpan data besar secara efisien sambil memastikan kecepatan pemrosesan yang optimal melalui teknologi FPGA.

1.2 PROJECT DESCRIPTION

Proyek ini berfokus pada implementasi algoritma Huffman Encoding untuk kompresi data lossless menggunakan pendekatan perangkat keras berbasis VHDL. Tujuan utama dari proyek ini adalah untuk menciptakan solusi yang efisien dalam menyimpan dan mengkompres data teks besar dengan memanfaatkan keunggulan FPGA untuk meningkatkan kecepatan dan efisiensi proses.

Alur kerja Algoritma:

1. Input data

   Program menerima masukan berupa file teks (.txt) yang berisi string atau data teks. Data ini dianalisis untuk menghitung frekuensi kemunculan setiap karakter yang ada dalam file tersebut.
2. Analisis Frekuensi

   Frekuensi kemunculan setiap karakter dihitung dan diurutkan dalam urutan menaik. Informasi ini digunakan untuk membangun struktur pohon Huffman.
3. Pembuatan tree structure

   Berdasarkan frekuensi yang telah diurutkan, pohon biner Huffman dibuat. Proses ini melibatkan penggabungan dua node dengan frekuensi terendah menjadi satu node baru, yang frekuensinya merupakan jumlah dari kedua node tersebut. Proses ini berlanjut hingga terbentuk pohon tunggal yang merepresentasikan seluruh karakter dalam data.
4. Programming karakter

   Setelah pohon Huffman terbentuk, setiap karakter diberi kode biner berdasarkan jalur yang dilalui dari akar pohon ke node daun. Simbol yang lebih sering muncul diberi kode yang lebih pendek, sementara simbol yang lebih jarang diberi kode lebih panjang. Pendekatan ini memastikan efisiensi dalam pengkodean data.
5. Menyimpan data terkompres

   Data yang telah dikompresi, bersama dengan metadata (informasi tentang kode biner untuk setiap karakter), disimpan kembali ke dalam file teks baru. Metadata ini diperlukan untuk proses dekompresi data di kemudian hari.

Untuk meningkatkan efisiensi proses kompresi, algoritma ini diimplementasikan menggunakan VHDL pada FPGA. FPGA dipilih karena kemampuannya untuk memproses data secara paralel dan secara real-time, yang sangat penting untuk menangani string data yang besar. Proses implementasi mencakup:

1. Pemodelan Algoritma Huffman Encoding VHDL
2. Simulasi fungsionalitas dan akurasi algoritma sesuai dengan input data
3. Sintesis dan pengujian di platform FPGA guna evaluasi kerja perangkat keras dibandingkan perangkat lunak.

1.3 OBJECTIVES

Tujuan dari proyek ini yaitu:

1. Mengimplementasikan Algoritma Huffman Encoding pada Perangkat Keras
2. Meningkatkan Efisiensi Kompresi Data
3. Mengoptimalkan Kinerja dengan FPGA
4. Menyediakan Solusi untuk Aplikasi Skala Besar
5. Menyediakan Metadata untuk Proses Dekompresi

1.4 ROLES AND RESPONSIBILITIES

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

Roles	Responsibilities	Person
Code (utama)	Melakukan pemrograman dari hal mendasar hingga lanjutan serta melakukan testing untuk pendataan.	Jesaya David
Laporan (utama)	Menulis laporan serta testing program untuk pendataan dalam	Bryan Farras
Presentasi Powerpoint	Membuat file PPT untuk presentasi kelompok	Jesaya David, Bryan Farras

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT SOFTWARE

The tools that are going to be used in this project are as follows:

- **Modelsim**
- **Quartus Prime Lite**
- **VSCode**

2.2 IMPLEMENTATION

Proyek Huffman Encoding ini memilili beberapa modul terpisah yang digunakan untuk program-nya, diantaranya:

HuffmanNode.vhd

Program ini adalah implementasi algoritma Huffman Encoding berbasis perangkat keras menggunakan VHDL, yang dirancang untuk memproses data teks dan menghasilkan tree structure Huffman. HuffmanProcessor mengontrol alur proses yang terdiri dari dua tahap: pembuatan node dan pengurutan node berdasarkan frekuensi kemunculan karakter. Komponen NodeGenerator bertugas menghasilkan node dari data input berupa karakter dan frekuensi, sementara NodeSorter mengurutkan node tersebut dalam urutan menaik berdasarkan frekuensi.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity HuffmanProcessor is
    Port (
        clk         : in  STD_LOGIC;
        reset       : in  STD_LOGIC;
        sort_ready  : out STD_LOGIC;
        sorted_char : out STD_LOGIC_VECTOR(7 downto 0);
        sorted_freq : out INTEGER;
        done        : out STD_LOGIC
    );
end HuffmanProcessor;

architecture Behavioral of HuffmanProcessor is
    -- NodeGenerator component
    component NodeGenerator is
        Port (
            clk       : in  STD_LOGIC;
```

```
        reset      : in  STD_LOGIC;
        node_ready : out STD_LOGIC;
        node_char  : out STD_LOGIC_VECTOR(7 downto 0);
        node_freq  : out INTEGER;
        done       : out STD_LOGIC
    );
end component;

-- NodeSorter component
component NodeSorter is
    Port (
        clk        : in  STD_LOGIC;
        reset      : in  STD_LOGIC;
        node_ready  : in  STD_LOGIC;
        node_char   : in  STD_LOGIC_VECTOR(7 downto 0);
        node_freq   : in  INTEGER;
        input_done  : in  STD_LOGIC;
        sort_ready  : out STD_LOGIC;
        sorted_char : out STD_LOGIC_VECTOR(7 downto 0);
        sorted_freq : out INTEGER;
        sort_done   : out STD_LOGIC
    );
end component;

-- Internal signals
signal gen_node_ready : STD_LOGIC;
signal gen_node_char  : STD_LOGIC_VECTOR(7 downto 0);
signal gen_node_freq  : INTEGER;
signal gen_done       : STD_LOGIC;
signal sort_done_int  : STD_LOGIC;

type state_type is (gen_phase, sort_phase, done_state);
signal state : state_type := gen_phase;

begin
-- Instantiate NodeGenerator
node_gen: NodeGenerator port map (
    clk        => clk,
    reset      => reset,
    node_ready => gen_node_ready,
    node_char  => gen_node_char,
    node_freq  => gen_node_freq,
    done       => gen_done
);

-- Instantiate NodeSorter
node_sort: NodeSorter port map (
    clk        => clk,
    reset      => reset,
    node_ready  => gen_node_ready,
```

```
        node_char   => gen_node_char,
        node_freq   => gen_node_freq,
        input_done  => gen_done,
        sort_ready  => sort_ready,
        sorted_char => sorted_char,
        sorted_freq => sorted_freq,
        sort_done   => sort_done_int
    );

    -- Control process
    process(clk, reset)
    begin
        if reset = '1' then
            state <= gen_phase;
            done <= '0';
        elsif rising_edge(clk) then
            case state is
                when gen_phase =>
                    if gen_done = '1' then
                        state <= sort_phase;
                    end if;

                when sort_phase =>
                    if sort_done_int = '1' then
                        state <= done_state;
                        done <= '1';
                    end if;

                when done_state =>
                    done <= '1';

                when others =>
                    state <= done_state;
            end case;
        end if;
    end process;

end Behavioral;
```

NodeGen.vhd

Modul NodeGenerator membaca data pasangan karakter dan frekuensi dari file teks, menyimpannya dalam array internal, lalu mengeluarkan node satu per satu melalui port output dengan sinyal node_ready. Menggunakan state machine, modul ini mengontrol alur dari pembacaan file hingga selesai, ditandai oleh sinyal done.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;
use IEEE.NUMERIC_STD.ALL;

entity NodeGenerator is
    Port (
        clk          : in  STD_LOGIC;
        reset        : in  STD_LOGIC;
        node_ready   : out STD_LOGIC;
        node_char    : out STD_LOGIC_VECTOR(7 downto 0);
        node_freq    : out INTEGER;
        done         : out STD_LOGIC
    );
end NodeGenerator;

architecture Behavioral of NodeGenerator is
    type node_type is record
        character    : character;
        frequency    : integer;
        left_child   : integer;
        right_child  : integer;
        merged       : boolean;
    end record;

    constant MAX_NODES : integer := 256;
    type node_array is array (0 to MAX_NODES - 1) of node_type;

    signal nodes          : node_array := (others => (character => ' ', frequency => 0, left_child
=> -1, right_child => -1, merged => false));
    signal total_nodes    : integer := 0;
    signal node_index     : integer := 0;

    type state_type is (read_file, parse_line, create_node, done_state);
    signal state : state_type := read_file;

    file input_file : text open read_mode is "Output";

    function char_to_slv(c : character) return STD_LOGIC_VECTOR is
    begin
        return std_logic_vector(to_unsigned(character'pos(c), 8));
    end function;

begin
    process(clk, reset)
        variable input_line  : line;
        variable char_read   : character;
        variable freq_read   : integer;
    begin
        if reset = '1' then
            -- Reset signal
```

```
        state       <= read_file;
        total_nodes <= 0;
        node_index  <= 0;
        node_ready  <= '0';
        done        <= '0';
    elsif rising_edge(clk) then
        case state is
            when read_file =>
                if not endfile(input_file) then
                    -- Read line
                    readline(input_file, input_line);
                    state <= parse_line;
                else
                    state <= done_state; -- No more line
                end if;

            when parse_line =>
                -- Parse character
                read(input_line, char_read);
                read(input_line, freq_read);

                -- Store data
                nodes(total_nodes).character <= char_read;
                nodes(total_nodes).frequency <= freq_read;
                nodes(total_nodes).left_child <= -1;
                nodes(total_nodes).right_child <= -1;
                nodes(total_nodes).merged <= false;

                -- Log
                report "Character read: " & char_read & ", Frequency read: " &
integer'image(freq_read);

                -- Increment
                total_nodes <= total_nodes + 1;

                -- Create_node state
                state <= create_node;

            when create_node =>
                -- Output data
                if node_index < total_nodes then
                    node_char <= char_to_slv(nodes(node_index).character);
                    node_freq <= nodes(node_index).frequency;
                    node_ready <= '1';

                    report "Outputting node: Char = " & nodes(node_index).character &
                        ", Freq = " & integer'image(nodes(node_index).frequency);

                    node_index <= node_index + 1;
```

```
                -- Transition back to read_file to read next line
                state <= read_file;
            else
                node_ready <= '0';
                state <= done_state;
            end if;

        when done_state =>
            -- Mark process as completed
            node_ready <= '0';
            done <= '1';
            report "Node generation completed. Total nodes: " &
integer'image(total_nodes);

        when others =>
            state <= done_state;
      end case;
    end if;
  end process;
end Behavioral;
```

NodeSort.vhd

Modul NodeSorter mengurutkan node berdasarkan frekuensi secara menurun untuk mendukung pembuatan tree Huffman. Modul ini menerima input berupa karakter dan frekuensi melalui port, menyimpan data dalam array, lalu mengurutkannya menggunakan algoritma bubble sort. Setelah selesai, node yang telah diurutkan dikeluarkan satu per satu melalui port output dengan sinyal sort_ready, hingga seluruh data diproses dan sinyal sort_done diaktifkan.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity NodeSorter is
  Port (
    clk         : in  STD_LOGIC;
    reset       : in  STD_LOGIC;
    -- Input interface
    node_ready   : in  STD_LOGIC;
    node_char    : in  STD_LOGIC_VECTOR(7 downto 0);
    node_freq    : in  INTEGER;
    input_done   : in  STD_LOGIC;
    -- Output interface
    sort_ready   : out STD_LOGIC;
    sorted_char  : out STD_LOGIC_VECTOR(7 downto 0);
    sorted_freq  : out INTEGER;
    sort_done    : out STD_LOGIC
```

```
    );
end NodeSorter;

architecture Behavioral of NodeSorter is
    type node_type is record
        character   : STD_LOGIC_VECTOR(7 downto 0);
        frequency   : integer;
    end record;

    constant MAX_NODES : integer := 256;
    type node_array is array (0 to MAX_NODES - 1) of node_type;

    signal nodes       : node_array;
    signal node_count   : integer := 0;
    signal output_index : integer := 0;

    type state_type is (receiving, sorting, outputting, done_state);
    signal state : state_type := receiving;

begin
    process(clk, reset)
        variable temp_nodes : node_array;
        variable i, j : integer;
        variable temp_node : node_type;
        variable found : boolean;
    begin
        if reset = '1' then
            report "Reset activated";
            state <= receiving;
            node_count <= 0;
            output_index <= 0;
            sort_ready <= '0';
            sort_done <= '0';
        elsif rising_edge(clk) then
            case state is
                when receiving =>
                    if node_ready = '1' then
                        report "Receiving new node - Char: " &
                               character'val(to_integer(unsigned(node_char))) &
                               " Freq: " & integer'image(node_freq);

                        if node_count = 0 then
                            report "First node being added";
                            nodes(0).character <= node_char;
                            nodes(0).frequency <= node_freq;
                            node_count <= 1;
                        else
                            found := false;
                            for i in 0 to node_count-1 loop
                                if nodes(i).character = node_char then
```

```
                        report "Character already exists at index " & integer'image(i);
                        found := true;
                        exit;
                    end if;
                end loop;

                if not found then
                    report "Adding new node at index " & integer'image(node_count);
                    nodes(node_count).character <= node_char;
                    nodes(node_count).frequency <= node_freq;
                    node_count <= node_count + 1;
                end if;
            end if;
        elsif input_done = '1' then
            report "Input complete. Total nodes: " & integer'image(node_count);
            state <= sorting;
        end if;

    when sorting =>
        report "Starting sort phase";
        report "Initial array:";
        for i in 0 to node_count-1 loop
            report "Node[" & integer'image(i) & "]: Char=" &
                character'val(to_integer(unsigned(nodes(i).character))) &
                " Freq=" & integer'image(nodes(i).frequency);
        end loop;

        temp_nodes := nodes;
        for i in 0 to node_count-2 loop
            for j in 0 to node_count-i-2 loop
                if temp_nodes(j).frequency < temp_nodes(j+1).frequency then
                    report "Swapping nodes " & integer'image(j) & " and " &
integer'image(j+1);
                    temp_node := temp_nodes(j);
                    temp_nodes(j) := temp_nodes(j+1);
                    temp_nodes(j+1) := temp_node;
                end if;
            end loop;
        end loop;

        report "Sort complete. Final array:";
        for i in 0 to node_count-1 loop
            report "Node[" & integer'image(i) & "]: Char=" &
                character'val(to_integer(unsigned(temp_nodes(i).character))) &
                " Freq=" & integer'image(temp_nodes(i).frequency);
        end loop;

        nodes <= temp_nodes;
        state <= outputting;
```

```
            when outputting =>
               if output_index < node_count then
                  report "Outputting node " & integer'image(output_index) &
                        ":                                    Char="                                    &
character'val(to_integer(unsigned(nodes(output_index).character))) &
                        " Freq=" & integer'image(nodes(output_index).frequency);
                  sorted_char <= nodes(output_index).character;
                  sorted_freq <= nodes(output_index).frequency;
                  sort_ready <= '1';
                  output_index <= output_index + 1;
               else
                  report "Output complete";
                  sort_ready <= '0';
                  state <= done_state;
               end if;

            when done_state =>
               report "Sorting process complete";
               sort_ready <= '0';
               sort_done <= '1';

            when others =>
               state <= done_state;
         end case;
      end if;
   end process;
end Behavioral;
```

Reader.vhd

Program Reader.vhd membaca karakter dari file input, menghitung frekuensi kemunculannya, mengurutkan karakter berdasarkan frekuensi secara menurun, dan menyimpan hasil ke file output. Data yang diurutkan juga dikirim melalui port output dengan sinyal data_ready. Prosesnya dikendalikan oleh state machine yang meliputi tahapan pembacaan file, penghitungan frekuensi, pengurutan data, penulisan ke file, dan pengeluaran data hingga selesai ditandai oleh sinyal done.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ReadSort is
   Port (
      clk         : in  STD_LOGIC;
      reset       : in  STD_LOGIC;
      data_ready  : out STD_LOGIC;
      sorted_char : out STD_LOGIC_VECTOR(7 downto 0);
      sorted_freq : out INTEGER;
```

```
        done         : out STD_LOGIC
    );
end ReadSort;

architecture Behavioral of ReadSort is
    constant MAX_CHARS   : integer := 256;
    constant ASCII_RANGE : integer := 128;

    type char_array is array (0 to MAX_CHARS - 1) of character;
    type freq_array is array (0 to MAX_CHARS - 1) of integer;

    signal sorted_chars    : char_array := (others => ' ');
    signal sorted_freqs    : freq_array := (others => 0);
    signal data_index      : integer := 0;
    signal total_chars     : integer := 0;

    type state_type is (read_file, count_freq, sort_data, write_output, output_data,
done_state);
    signal state : state_type := read_file;

    file input_file  : text open read_mode is "Input";
    file output_file : text open write_mode is "Output";

    function char_to_slv(c : character) return STD_LOGIC_VECTOR is
    begin
        return std_logic_vector(to_unsigned(character'pos(c), 8));
    end function;

begin
    process(clk, reset)
        variable input_line     : line;
        variable temp_char      : character;
        variable unsorted_chars : char_array := (others => ' ');
        variable frequencies    : freq_array := (others => 0);
        variable char_count     : integer := 0;
        variable i, j, k        : integer;
        variable temp_freq      : integer;
        variable temp_char_var  : character;
        variable output_line    : line;
    begin
        if reset = '1' then
            state       <= read_file;
            data_index  <= 0;
            data_ready  <= '0';
            total_chars <= 0;
            done        <= '0';
        elsif rising_edge(clk) then
            case state is
                when read_file =>
                    if not endfile(input_file) then
```

```
                readline(input_file, input_line);
                for idx in input_line'range loop
                    if char_count < MAX_CHARS then
                        temp_char := input_line(idx);
                        unsorted_chars(char_count) := temp_char;
                        char_count := char_count + 1;
                    end if;
                end loop;
            else
                total_chars <= char_count;
                state <= count_freq;
            end if;
        when count_freq =>
            for i in 0 to total_chars - 1 loop
                frequencies(character'pos(unsorted_chars(i))) :=
frequencies(character'pos(unsorted_chars(i))) + 1;
            end loop;
            state <= sort_data;
        when sort_data =>
            j := 0;
            for i in 0 to ASCII_RANGE - 1 loop
                if frequencies(i) > 0 then
                    sorted_chars(j) <= character'val(i);
                    sorted_freqs(j) <= frequencies(i);
                    j := j + 1;
                end if;
            end loop;

            for i in 0 to j - 2 loop
                for k in 0 to j - i - 2 loop
                    if sorted_freqs(k) < sorted_freqs(k + 1) then
                        temp_freq := sorted_freqs(k);
                        sorted_freqs(k) := sorted_freqs(k + 1);
                        sorted_freqs(k + 1) := temp_freq;
                        temp_char_var := sorted_chars(k);
                        sorted_chars(k) := sorted_chars(k + 1);
                        sorted_chars(k + 1) := temp_char_var;
                    end if;
                end loop;
            end loop;
            total_chars <= j;
            data_index  <= 0;
            state       <= write_output;
        when write_output =>
            for i in 0 to total_chars - 1 loop
                output_line := null;
                write(output_line, sorted_chars(i));
                write(output_line, String(" "));
                write(output_line, sorted_freqs(i));
                writeline(output_file, output_line);
```

```
                end loop;
                state <= output_data;
            when output_data =>
                if data_index < total_chars then
                    sorted_char <= char_to_slv(sorted_chars(data_index));
                    sorted_freq <= sorted_freqs(data_index);
                    data_ready  <= '1';
                    data_index  <= data_index + 1;
                else
                    data_ready <= '0';
                    state      <= done_state;
                end if;
            when done_state =>
                data_ready <= '0';
                done       <= '1';
            when others =>
                state <= done_state;
        end case;
    end if;
  end process;
end Behavioral;
```

ReadSort.vhd

Modul ReadSort membaca string dari file masukan, menghitung frekuensi kemunculan setiap karakter, mengurutkan data berdasarkan frekuensi secara menurun, dan menulis hasilnya ke file keluaran. Setelah itu, modul mengeluarkan pasangan karakter dan frekuensi satu per satu melalui port output dengan sinyal data_ready.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ReadSort is
  Port (
    clk         : in  STD_LOGIC;
    reset       : in  STD_LOGIC;
    data_ready  : out STD_LOGIC;
    sorted_char : out STD_LOGIC_VECTOR(7 downto 0);
    sorted_freq : out INTEGER;
    done        : out STD_LOGIC
  );
end ReadSort;

architecture Behavioral of ReadSort is
  constant MAX_CHARS  : integer := 256;
  constant ASCII_RANGE : integer := 128;
```

```
    type char_array is array (0 to MAX_CHARS - 1) of character;
    type freq_array is array (0 to MAX_CHARS - 1) of integer;

    signal sorted_chars    : char_array := (others => ' ');
    signal sorted_freqs    : freq_array := (others => 0);
    signal data_index      : integer := 0;
    signal total_chars     : integer := 0;

    type state_type is (read_file, count_freq, sort_data, output_data, write_file, done_state);
    signal state : state_type := read_file;

    file input_file : text open read_mode is "Input";
    file output_file : text open write_mode is "Output";

    function char_to_slv(c : character) return STD_LOGIC_VECTOR is
    begin
        return std_logic_vector(to_unsigned(character'pos(c), 8));
    end function;

begin
    process(clk, reset)
        variable input_line     : line;
        variable temp_char      : character;
        variable unsorted_chars : char_array := (others => ' ');
        variable frequencies    : freq_array := (others => 0);
        variable char_count     : integer := 0;
        variable i, j, k        : integer;
        variable temp_freq      : integer;
        variable temp_char_var  : character;
        variable output_line    : line;
    begin
        if reset = '1' then
            state       <= read_file;
            data_index  <= 0;
            data_ready  <= '0';
            total_chars <= 0;
            done        <= '0';
        elsif rising_edge(clk) then
            case state is
                when read_file =>
                    if not endfile(input_file) then
                        readline(input_file, input_line);
                        for idx in input_line'range loop
                            if char_count < MAX_CHARS then
                                temp_char := input_line(idx);
                                unsorted_chars(char_count) := temp_char;
                                char_count := char_count + 1;
                            end if;
                        end loop;
                    else
```

```
            total_chars <= char_count;
            state <= count_freq;
        end if;

    when count_freq =>
        for i in 0 to total_chars - 1 loop
            frequencies(character'pos(unsorted_chars(i))) :=
                frequencies(character'pos(unsorted_chars(i))) + 1;
        end loop;
        state <= sort_data;

    when sort_data =>
        j := 0;
        -- First collect non-zero frequency characters
        for i in 0 to ASCII_RANGE - 1 loop
            if frequencies(i) > 0 then
                sorted_chars(j) <= character'val(i);
                sorted_freqs(j) <= frequencies(i);
                j := j + 1;
            end if;
        end loop;

        -- Then sort them by frequency (bubble sort)
        for i in 0 to j - 2 loop
            for k in 0 to j - i - 2 loop
                if sorted_freqs(k) < sorted_freqs(k + 1) then
                    -- Swap frequencies
                    temp_freq := sorted_freqs(k);
                    sorted_freqs(k) <= sorted_freqs(k + 1);
                    sorted_freqs(k + 1) <= temp_freq;
                    -- Swap characters
                    temp_char_var := sorted_chars(k);
                    sorted_chars(k) <= sorted_chars(k + 1);
                    sorted_chars(k + 1) <= temp_char_var;
                end if;
            end loop;
        end loop;
        total_chars <= j;
        data_index <= 0;
        state <= output_data;

    when output_data =>
        if data_index < total_chars then
            sorted_char <= char_to_slv(sorted_chars(data_index));
            sorted_freq <= sorted_freqs(data_index);
            data_ready <= '1';
            data_index <= data_index + 1;
        else
            data_ready <= '0';
            state <= write_file;
```

```
                end if;

            when write_file =>
                for i in 0 to total_chars - 1 loop
                    write(output_line, sorted_chars(i));
                    write(output_line, string'(" "));
                    write(output_line, sorted_freqs(i));
                    writeline(output_file, output_line);
                end loop;
                state <= done_state;

            when done_state =>
                data_ready <= '0';
                done <= '1';

            when others =>
                state <= done_state;
        end case;
    end if;
  end process;
end Behavioral;
```

NodeMerge.vhd

NodeMerge.vhd adalah modul yang membangun pohon Huffman dari daftar karakter yang telah diurutkan berdasarkan frekuensi. Modul ini bekerja dengan menggabungkan dua node dengan frekuensi terkecil secara berulang hingga terbentuk satu node root, yang merepresentasikan pohon Huffman. Proses ini dikontrol oleh state machine dengan tahap utama: memuat node, mengurutkan, dan menggabungkan node, hingga proses selesai. Output berupa indeks root pohon Huffman (root_index) dan sinyal selesai (merge_done).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity NodeMerger is
  Port (
    clk          : in  std_logic;
    reset        : in  std_logic;
    sorted_char  : in  std_logic_vector(7 downto 0);
    sorted_freq  : in  integer;
    sort_ready   : in  std_logic;
    sort_done    : in  std_logic;
    merge_done   : out std_logic;
    root_index   : out integer
  );
```

```
end NodeMerger;

architecture Behavioral of NodeMerger is
    -- Node type definition (unchanged)
    type node_type is record
        character    : std_logic_vector(7 downto 0);
        frequency    : integer;
        left_child   : integer;
        right_child  : integer;
        is_leaf      : boolean;
    end record;

    type state_type is (idle_state, load_state, sort_state, merge_state, done_state);

    constant NUM_LEAF_NODES : integer := 8;
    constant MAX_NODES      : integer := (2 * NUM_LEAF_NODES) - 1;

    type node_array is array(0 to MAX_NODES - 1) of node_type;
    type index_array is array(0 to MAX_NODES - 1) of integer;

    -- Initialize all signals
    signal nodes : node_array := (others => (
        character => (others => '0'),
        frequency => 0,
        left_child => -1,
        right_child => -1,
        is_leaf => false
    ));

    signal sorted_indices : index_array := (others => 0);
    signal state : state_type := idle_state;  -- Now properly typed
    signal sorted_size : integer := 0;
    signal next_node_index : integer := 0;
    signal root_index_sig : integer := 0;
    signal received_first_node : boolean := false;

    constant SHOW_DEBUG : boolean := true;

begin
    root_index <= root_index_sig;

    process(clk, reset)
        variable temp_index : integer;
```

```
    variable min1_index, min2_index : integer;
    variable i, j : integer;
    variable temp_indices : index_array;
    variable temp_node : node_type;
    variable temp_nodes : node_array;

begin
    if reset = '1' then
        state <= idle_state;
        merge_done <= '0';
        sorted_size <= 0;
        next_node_index <= 0;
        root_index_sig <= 0;
        received_first_node <= false;
        -- Clear arrays
        nodes <= (others => (
            character => (others => '0'),
            frequency => 0,
            left_child => -1,
            right_child => -1,
            is_leaf => false
        ));
        sorted_indices <= (others => 0);

    elsif rising_edge(clk) then
        case state is
            when idle_state =>
                -- Always accept nodes in idle state
                if sort_ready = '1' then
                    if SHOW_DEBUG then
                        report "Idle state received node: Char = " &
                            character'val(to_integer(unsigned(sorted_char))) &
                            ", Freq = " & integer'image(sorted_freq);
                    end if;

                    nodes(next_node_index).character <= sorted_char;
                    nodes(next_node_index).frequency <= sorted_freq;
                    nodes(next_node_index).is_leaf <= true;
                    sorted_indices(next_node_index) <= next_node_index;

                    next_node_index <= next_node_index + 1;
                    received_first_node <= true;
                    state <= load_state;
```

```
                end if;

            when load_state =>
                if sort_ready = '1' then
                    if SHOW_DEBUG then
                        report "Loading node " & integer'image(next_node_index) &
                            ": Char = " & character'val(to_integer(unsigned(sorted_char))) &
                            ", Freq = " & integer'image(sorted_freq);
                    end if;

                    nodes(next_node_index).character <= sorted_char;
                    nodes(next_node_index).frequency <= sorted_freq;
                    nodes(next_node_index).is_leaf <= true;
                    sorted_indices(next_node_index) <= next_node_index;
                    next_node_index <= next_node_index + 1;
                end if;

                if sort_done = '1' then
                    sorted_size <= next_node_index;
                    state <= sort_state;

                    if SHOW_DEBUG then
                        report "Total nodes loaded: " & integer'image(next_node_index);
                    end if;
                end if;

            when sort_state =>
                -- Use variable for sorting
                temp_indices := sorted_indices;

                -- Bubble sort using variables
                for i in 0 to sorted_size - 2 loop
                    for j in 0 to sorted_size - i - 2 loop
                        if nodes(temp_indices(j)).frequency > nodes(temp_indices(j + 1)).frequency or
                        (nodes(temp_indices(j)).frequency = nodes(temp_indices(j + 1)).frequency and
                            temp_indices(j) > temp_indices(j + 1)) then
                            temp_index := temp_indices(j);
                            temp_indices(j) := temp_indices(j + 1);
                            temp_indices(j + 1) := temp_index;
                        end if;
                    end loop;
```

```
                end loop;

                if SHOW_DEBUG then
                   report "----------------------------------------";
                   report "After initial sorting:";
                   report "Array contents:";
                   for i in 0 to sorted_size - 1 loop
                      report "Index " & integer'image(i) & ": Node " &
                             integer'image(temp_indices(i)) &
                             " (Char = " &
character'val(to_integer(unsigned(nodes(temp_indices(i)).character))) &
                             ", Freq = " & integer'image(nodes(temp_indices(i)).frequency) &
                             ", Leaf = " & boolean'image(nodes(temp_indices(i)).is_leaf) & ")";
                   end loop;
                   report "----------------------------------------";
                end if;

                sorted_indices <= temp_indices;
                state <= merge_state;

                when merge_state =>
                if sorted_size > 1 then
                   -- Get indices of two smallest frequency nodes
                   min1_index := sorted_indices(0);
                   min2_index := sorted_indices(1);

                   -- Create temporary node array to track current frequencies
                   temp_nodes := nodes;

                   -- Calculate new merged node
                   temp_node.frequency := temp_nodes(min1_index).frequency +
temp_nodes(min2_index).frequency;
                   temp_node.left_child := min1_index;
                   temp_node.right_child := min2_index;
                   temp_node.is_leaf := false;
                   temp_node.character := (others => '0');

                   -- Update temp array with new node
                   temp_nodes(next_node_index) := temp_node;

                   if SHOW_DEBUG then
                      report "Merging: Node " & integer'image(min1_index) &
                         " (Freq=" & integer'image(temp_nodes(min1_index).frequency) &
```

```
                        ") and Node " & integer'image(min2_index) &
                        " (Freq=" & integer'image(temp_nodes(min2_index).frequency) &
                        ") -> New Node " & integer'image(next_node_index) &
                        " (Total Freq=" & integer'image(temp_node.frequency) &
                        ", Left=" & integer'image(min1_index) &
                        ", Right=" & integer'image(min2_index) & ")";
                end if;

                -- Assign to actual nodes array
                nodes(next_node_index) <= temp_node;

                -- Update indices array
                temp_indices := sorted_indices;

                -- Shift remaining nodes left
                for i in 0 to sorted_size - 3 loop
                   temp_indices(i) := sorted_indices(i + 2);
                end loop;

                -- Add new merged node
                temp_indices(sorted_size - 2) := next_node_index;

                -- Sort using frequencies from temp_nodes
                for i in 0 to sorted_size - 3 loop
                   for j in 0 to sorted_size - i - 3 loop
                      if                    temp_nodes(temp_indices(j)).frequency                    >
temp_nodes(temp_indices(j + 1)).frequency then
                         temp_index := temp_indices(j);
                         temp_indices(j) := temp_indices(j + 1);
                         temp_indices(j + 1) := temp_index;
                      end if;
                   end loop;
                end loop;

                if SHOW_DEBUG then
                   report "After merge and sort:";
                   for i in 0 to sorted_size - 2 loop
                      report "Index " & integer'image(i) & ": Node " &
                         integer'image(temp_indices(i)) &
                         " (Freq = " & integer'image(temp_nodes(temp_indices(i)).frequency)
& ")";
                   end loop;
                end if;
```

```
                sorted_indices <= temp_indices;
                sorted_size <= sorted_size - 1;
                next_node_index <= next_node_index + 1;

            else
                if SHOW_DEBUG then
                    report "Merging complete. Root index = " &
integer'image(sorted_indices(0));
                end if;

                root_index_sig <= sorted_indices(0);
                merge_done <= '1';
                state <= done_state;
            end if;

        when done_state =>
            merge_done <= '1';

        when others =>
            state <= done_state;
      end case;
    end if;
  end process;

end Behavioral;
```

HuffmanTranslations.vhd

Program HuffmanTranslator ini dirancang untuk mengimplementasikan algoritma Huffman secara penuh, dari membaca data pohon Huffman hingga menerjemahkan teks masukan menjadi representasi biner. Alur kerjanya dimulai dengan reset untuk inisialisasi, dilanjutkan dengan membaca file masukan yang memuat informasi pohon Huffman. Program menghitung jumlah node, mengisi struktur data pohon, dan menentukan node akar. Proses ini diatur melalui state machine dengan lima kondisi utama: idle, counting, reading, translating, dan done. Output memastikan semua karakter dimasukkan sesuai kode Huffman yang valid.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use STD.TEXTIO.ALL;

entity HuffmanTranslator is
  Port (
```

```
        clk         : in  std_logic;
        reset       : in  std_logic;
        start       : in  std_logic;
        done        : out std_logic
    );
end HuffmanTranslator;

architecture Behavioral of HuffmanTranslator is
    -- Node type definition
    type node_type is record
        character    : std_logic_vector(7 downto 0);
        frequency    : integer;
        left_child   : integer;
        right_child  : integer;
        is_leaf      : boolean;
    end record;

    -- Dynamic arrays
    type node_array is array (natural range <>) of node_type;
    type path_array is array (natural range <>) of std_logic;

    -- State machine with preprocessing
    type state_type is (idle, counting, reading, translating, done_state);
    signal state : state_type := idle;

    -- Storage with initial sizes
    signal nodes : node_array(0 to 511);  -- Allow up to 256 characters (512 total nodes)
    signal path : path_array(0 to 511);   -- Max possible path length
    signal path_length : integer := 0;
    signal root_index : integer := 0;
    signal num_nodes : integer := 0;
    signal max_depth : integer := 0;

    -- File handling
    file tree_file : text;
    file orig_file : text;
    file output_file : text open write_mode is "FinalOutput";

    function maximum(a, b: integer) return integer is
    begin
        if a > b then
            return a;
        else
```

```
            return b;
        end if;
    end function;

    -- Helper function to calculate tree height
    function calculate_tree_height(
        nodes_arr: in node_array;
        node_idx: in integer) return integer is
        variable left_height, right_height : integer;
    begin
        -- Bounds checking
        if node_idx < 0 or node_idx > 511 then
            report "Warning: Node index " & integer'image(node_idx) & " out of bounds" severity warning;
            return 0;
        end if;

        -- Base cases
        if node_idx = -1 then
            return 0;
        elsif nodes_arr(node_idx).is_leaf then
            return 0;
        else
            -- Check child nodes exist within bounds
            if nodes_arr(node_idx).left_child >= 0 and nodes_arr(node_idx).left_child <= 511 then
                left_height := calculate_tree_height(nodes_arr, nodes_arr(node_idx).left_child);
            else
                left_height := 0;
            end if;

            if nodes_arr(node_idx).right_child >= 0 and nodes_arr(node_idx).right_child <= 511 then
                right_height := calculate_tree_height(nodes_arr, nodes_arr(node_idx).right_child);
            else
                right_height := 0;
            end if;

            -- Protect against overflow
            if left_height < 0 or right_height < 0 then
                report "Warning: Height calculation overflow detected" severity warning;
                return 0;
```

```
        end if;

        return 1 + maximum(left_height, right_height);
    end if;
end function;

-- Helper procedure to write translation
procedure write_translation(
    file f: text;
    char: in std_logic_vector(7 downto 0);
    path: in path_array;
    length: in integer) is
    variable l: line;
    variable temp_char: character;
begin
    temp_char := character'val(to_integer(unsigned(char)));
    write(l, temp_char);
    write(l, string'(": "));

    for i in 0 to length-1 loop
        if path(i) = '0' then
            write(l, character('0'));
        else
            write(l, character('1'));
        end if;
    end loop;
    writeline(f, l);
end procedure;

procedure translate_string(
    file input_file: text;
    file output_file: text;
    nodes_arr: in node_array;
    root: in integer) is
    variable l_in, l_out: line;
    variable in_char: character;
    variable curr_node: integer;
    variable result: string(1 to 1024);
    variable binary: string(1 to 4096);
    variable result_len: integer := 0;
    variable binary_len: integer := 0;
    variable temp_path: path_array(0 to 511);
    variable path_len: integer;
```

```
        variable found: boolean;
    begin
        readline(input_file, l_in);
        write(l_out, string'("Original text translation: "));

        while l_in'length > 0 loop
            read(l_in, in_char);
            result_len := result_len + 1;
            result(result_len) := in_char;

            -- Reset for new character search
            curr_node := root;
            path_len := 0;
            found := false;

            -- Find path to character
            while not found loop
                if nodes_arr(curr_node).is_leaf then
                    if      character'val(to_integer(unsigned(nodes_arr(curr_node).character)))      =
in_char then
                        found := true;
                    else
                        -- Backtrack - remove last path bit and try right path
                        while path_len > 0 and temp_path(path_len-1) = '1' loop
                            path_len := path_len - 1;
                            curr_node := root; -- Reset to root for backtracking
                            -- Rebuild path up to this point
                            for i in 0 to path_len-1 loop
                                if temp_path(i) = '0' then
                                    curr_node := nodes_arr(curr_node).left_child;
                                else
                                    curr_node := nodes_arr(curr_node).right_child;
                                end if;
                            end loop;
                        end loop;
                        if path_len > 0 then
                            -- Try right path
                            path_len := path_len - 1;
                            curr_node := root;
                            -- Rebuild path
                            for i in 0 to path_len-1 loop
                                if temp_path(i) = '0' then
                                    curr_node := nodes_arr(curr_node).left_child;
```

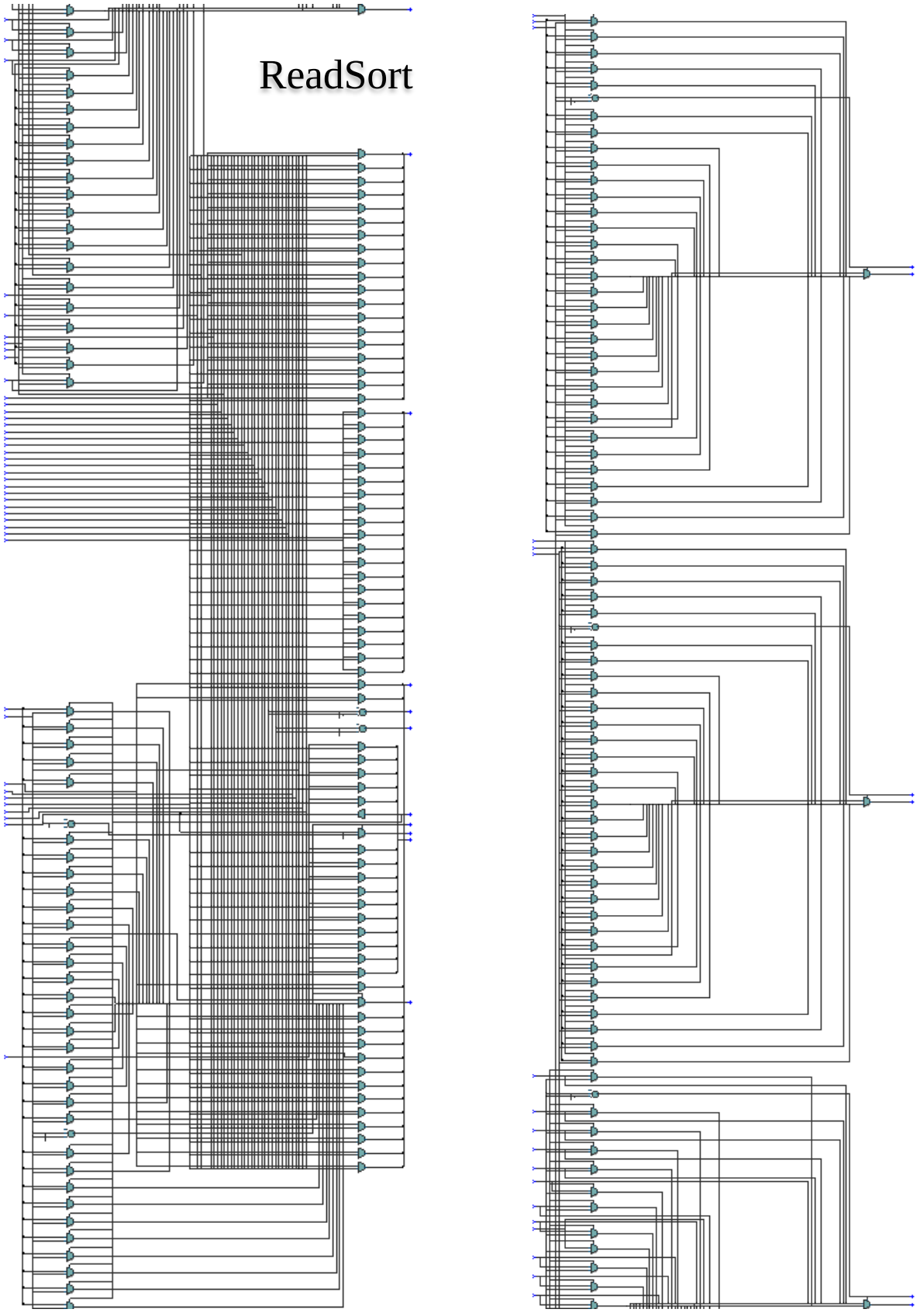
```
                        else
                            curr_node := nodes_arr(curr_node).right_child;
                        end if;
                    end loop;
                    -- Add right path
                    temp_path(path_len) := '1';
                    path_len := path_len + 1;
                    curr_node := nodes_arr(curr_node).right_child;
                end if;
            end if;
        else
            -- Try left path first
            temp_path(path_len) := '0';
            path_len := path_len + 1;
            curr_node := nodes_arr(curr_node).left_child;
        end if;
    end loop;

    -- Add found path to binary result
    for i in 0 to path_len-1 loop
        binary_len := binary_len + 1;
        if temp_path(i) = '0' then
            binary(binary_len) := '0';
        else
            binary(binary_len) := '1';
        end if;
    end loop;
end loop;

-- Write results
write(l_out, result(1 to result_len));
writeline(output_file, l_out);

write(l_out, string'("Binary translation: "));
write(l_out, binary(1 to binary_len));
writeline(output_file, l_out);
end procedure;

-- Recursive traversal procedure
procedure traverse_tree(
    node_index: in integer;
    curr_path: inout path_array;
    curr_length: inout integer;
```

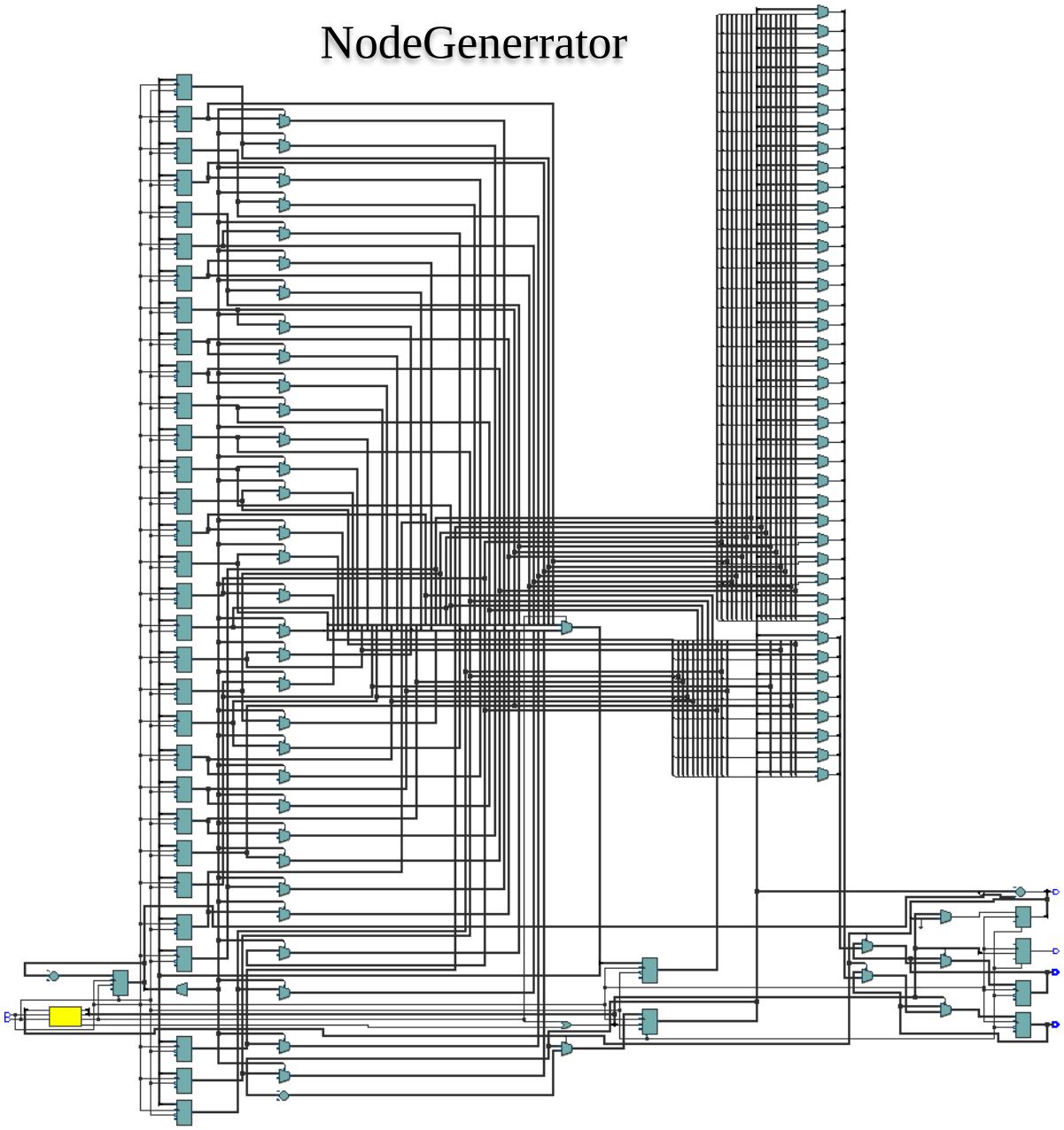
```
        nodes_arr: in node_array) is
    begin
        if nodes_arr(node_index).is_leaf then
            -- Found leaf node, write translation
            write_translation(output_file, nodes_arr(node_index).character, curr_path,
curr_length);
            report "Found translation for " &
                character'val(to_integer(unsigned(nodes_arr(node_index).character))) &
                " at depth " & integer'image(curr_length);
        else
            -- Traverse left (0)
            if nodes_arr(node_index).left_child /= -1 then
                curr_path(curr_length) := '0';
                curr_length := curr_length + 1;
                traverse_tree(nodes_arr(node_index).left_child, curr_path, curr_length,
nodes_arr);
                curr_length := curr_length - 1;
            end if;

            -- Traverse right (1)
            if nodes_arr(node_index).right_child /= -1 then
                curr_path(curr_length) := '1';
                curr_length := curr_length + 1;
                traverse_tree(nodes_arr(node_index).right_child, curr_path, curr_length,
nodes_arr);
                curr_length := curr_length - 1;
            end if;
        end if;
    end procedure;

begin
    process(clk, reset)
        variable line_in : line;
        variable char : character;
        variable node_index, freq, left, right : integer;
        variable is_leaf : boolean;
        variable temp_path : path_array(0 to 511);
        variable temp_length : integer;
        variable node_count : integer := 0;
    begin
        if reset = '1' then
            state <= idle;
            done <= '0';
```

```
            path_length <= 0;
            root_index <= 0;
            num_nodes <= 0;

        elsif rising_edge(clk) then
            case state is
                when idle =>
                    if start = '1' then
                        -- First count nodes
                        file_open(tree_file, "HuffmanArray", read_mode);
                        state <= counting;
                    end if;

                when counting =>
                    -- Count total nodes in file
                    while not endfile(tree_file) loop
                        readline(tree_file, line_in);
                        node_count := node_count + 1;
                    end loop;

                    num_nodes <= node_count;
                    file_close(tree_file);
                    file_open(tree_file, "HuffmanArray", read_mode);
                    state <= reading;

                when reading =>
                    while not endfile(tree_file) loop
                        readline(tree_file, line_in);

                        -- Parse node index
                        read(line_in, node_index);
                        read(line_in, char); -- Skip comma

                        -- Parse character/dash
                        read(line_in, char);
                        if char = '-' then
                            nodes(node_index).character <= (others => '0');
                            nodes(node_index).is_leaf <= false;
                        else
                            nodes(node_index).character                                     <=
std_logic_vector(to_unsigned(character'pos(char), 8));
                            nodes(node_index).is_leaf <= true;
                        end if;
```

```
                -- Parse frequency, left child, right child
                read(line_in, char); -- Skip comma
                read(line_in, freq);
                read(line_in, char); -- Skip comma
                read(line_in, left);
                read(line_in, char); -- Skip comma
                read(line_in, right);
                read(line_in, char); -- Skip comma
                read(line_in, is_leaf);

                -- Assign values
                nodes(node_index).frequency <= freq;
                nodes(node_index).left_child <= left;
                nodes(node_index).right_child <= right;
            end loop;

            -- Find root (last node)
            file_close(tree_file);
            root_index <= num_nodes - 1;
            max_depth <= calculate_tree_height(nodes, num_nodes - 1);
            state <= translating;

            when translating =>
            temp_length := 0;
            traverse_tree(root_index, temp_path, temp_length, nodes);

            file_open(orig_file, "Input", read_mode);
            translate_string(orig_file, output_file, nodes, root_index);
            file_close(orig_file);

            state <= done_state;

          when done_state =>
            done <= '1';

          when others =>
            state <= idle;
        end case;
      end if;
    end process;
end Behavioral;
```

Berikut adalah hasil sintesis rangkaian dari program yang telah tertulis menggunakan Quartus Prime Lite:





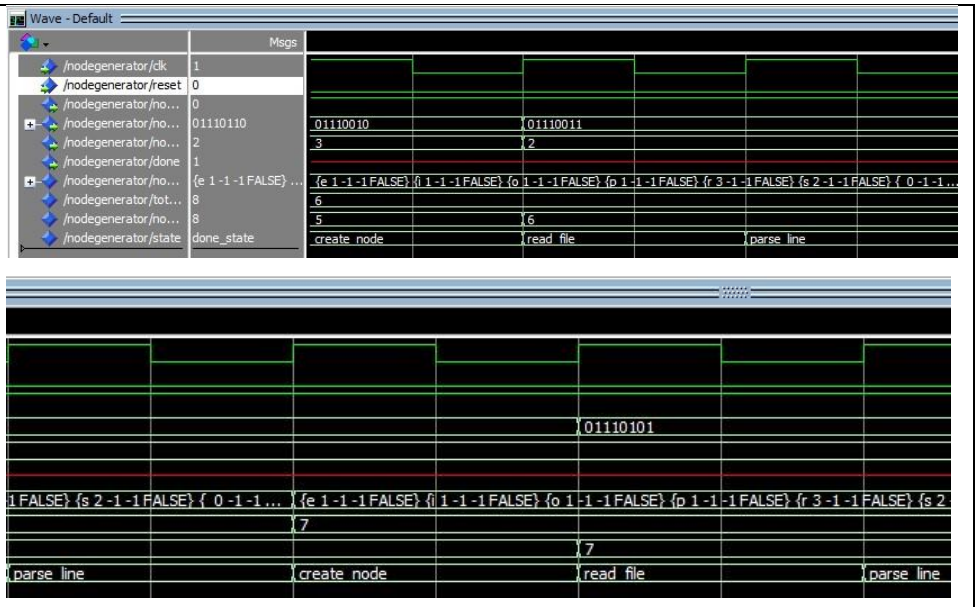
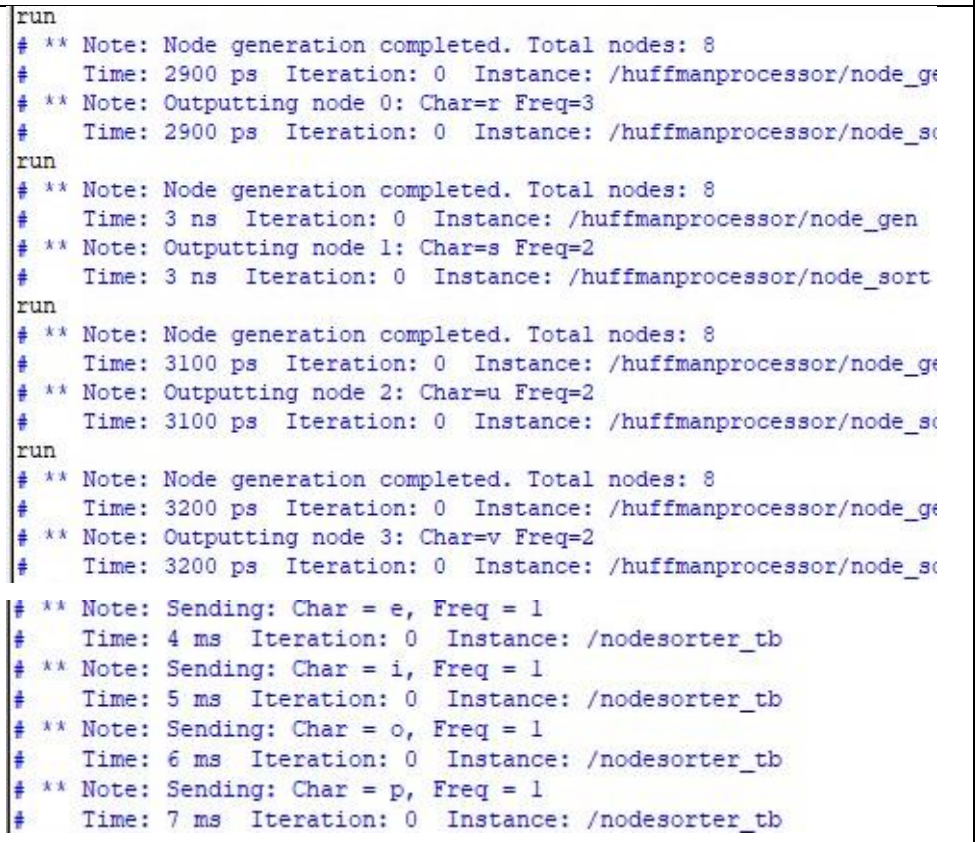
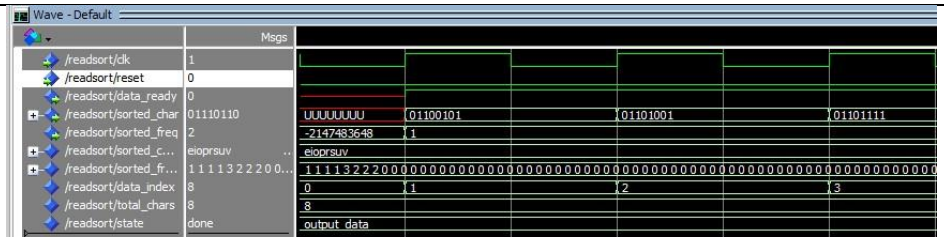
CHAPTER 3

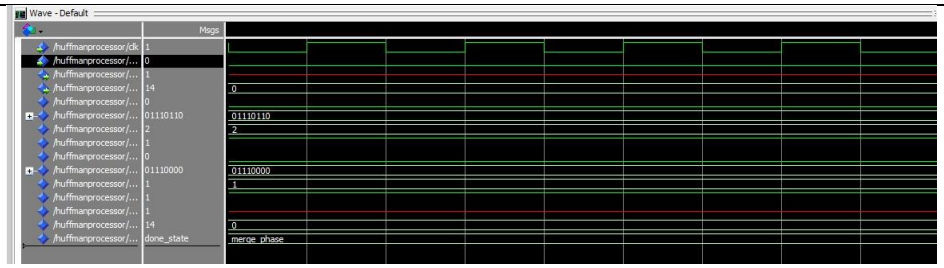
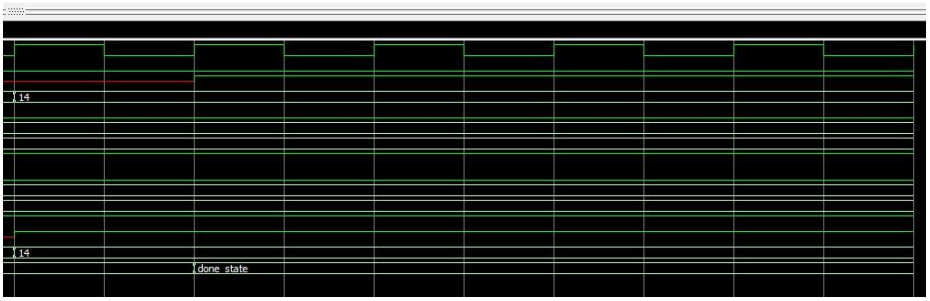
TESTING AND ANALYSIS

3.1 TESTING

Berikut adalah hasil dari tes run untuk beberapa pekerjaan yang berbeda sebelum:

Pekerjaan	Run Test

Tes NodeGen	
Testbench untuk NodeSort	
Membaca isi file text	

NodeMerge untuk membuat node dari 2 node yang lebih kecil.	
	 
	`# Time: 365 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 1: Node 8 (Freq = 2)`
	`# Time: 365 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 2: Node 9 (Freq = 2)`
	`# Time: 365 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 3: Node 0 (Freq = 3)`
	`# Time: 365 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 4: Node 10 (Freq = 4)`
	`# Time: 365 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Merging: Node 3 (Freq=2) and Node 8 (Freq=2) -> New Node 11 (Total Freq=4, Left=3, Right=8`
	`# Time: 375 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: After merge and sort:`
	`# Time: 375 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 0: Node 9 (Freq = 2)`
	`# Time: 375 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 1: Node 0 (Freq = 3)`
	`# Time: 375 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 2: Node 10 (Freq = 4)`
	`# Time: 375 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Index 3: Node 11 (Freq = 4)`
	`# Time: 375 ns Iteration: 1 Instance: /nodemerger_tb/uut`
	`# ** Note: Merging: Node 9 (Freq=2) and Node 0 (Freq=3) -> New Node 12 (Total Freq=5, Left=9, Right=0`

3.2 RESULT

```
o: 000
p: 001
r: 01
s: 100
u: 101
v: 110
e: 1110
i: 1111
Original text translation: supersurvivor
Binary translation: 10010100111100110010101110111111000001
```

Snipping tersebut adalah hasil run program. Input file text berisi tulisan supersurvivor yang kemudian akan dipecah menjadi karakter individu. Karakter-karakter unik dalam string dihitung frekuensinya. Contoh: s, u, p, e, r, v, i, dan o dihitung berapa kali muncul. Data ini digunakan untuk membangun node pada pohon Huffman.

Setiap karakter unik dijadikan *node*, dengan informasi karakter, frekuensi, dan status sebagai daun (*leaf*). Dua node dengan frekuensi terendah digabungkan menjadi node baru dengan frekuensi total dari keduanya. Node baru ini memiliki anak kiri (frekuensi lebih kecil) dan anak kanan (frekuensi lebih besar). Proses ini diulang sampai terbentuk satu node akar yang mewakili pohon Huffman lengkap. Setiap cabang kiri pada pohon ditandai dengan 0, dan cabang kanan ditandai dengan 1. Traversal pohon menghasilkan kode Huffman untuk setiap karakter. Sebagai contoh:

- o: 000
- p: 001
- r: 01
- dan seterusnya.

String asli supersurvivor diterjemahkan berdasarkan kode Huffman yang telah dibuat. Setiap karakter digantikan oleh kodenya:

- s → 100
- u → 101
- p → 001
- e → 1110
- r → 01
- dan seterusnya. Hasil translasi biner lengkap adalah: 100101001111100110010101110111111100001.

3.3 ANALYSIS

Alur program ini dimulai dengan membaca file Input yang berisi string asli. Proses diawali dengan ReadSort, yang memecah string menjadi karakter-karakter individu dan menghitung frekuensi kemunculan setiap karakter unik.

Hasil ini diteruskan ke NodeGen, di mana setiap karakter unik diubah menjadi *node* yang berisi informasi karakter, frekuensi, dan statusnya sebagai daun (*leaf*). Node-node ini kemudian disortir berdasarkan frekuensi dalam NodeSorter. Setelah itu, masuk ke proses NodeMerger, di mana dua node dengan frekuensi terendah digabungkan menjadi node baru.

Node baru ini menyimpan data anak kiri (frekuensi lebih kecil) dan anak kanan (frekuensi lebih besar).

Node yang sudah digabung tidak akan diproses ulang, dan daftar node terus disortir ulang setelah setiap penggabungan. Proses ini berulang sampai hanya tersisa satu node, yaitu akar pohon Huffman. Data pohon yang sudah lengkap kemudian ditulis ke dalam file HuffmanArray.

Pada tahap berikutnya, HuffmanTranslator membaca file HuffmanArray dan memparse data menjadi array node di memori. Program menghitung tinggi pohon berdasarkan data anak kiri dan kanan dari setiap node, lalu menjelajahi seluruh kombinasi yang mungkin dalam pohon.

Untuk setiap langkah, anak kiri ditandai dengan "0" dan anak kanan dengan "1". Ketika menemukan node daun, translasi karakter disimpan. Proses ini diulangi sampai semua karakter mendapatkan translasi Huffman-nya.

Terakhir, HuffmanTranslator menulis translasi karakter dan translasi string lengkap (biner) ke file FinalOutput. File ini berisi representasi Huffman dari string masukan serta hasil kompresi final.

CHAPTER 4

CONCLUSION

algoritma Huffman berhasil diterapkan untuk mengompresi string secara efisien dengan menghasilkan representasi biner yang lebih hemat. Proses dimulai dengan menghitung frekuensi setiap karakter unik dalam string, membangun pohon Huffman dari node-node berdasarkan frekuensi tersebut, dan memberikan kode biner ke setiap karakter berdasarkan posisi di pohon. String asli kemudian diterjemahkan menggunakan kode Huffman ini, menghasilkan representasi biner yang lebih ringkas. Program juga memastikan hasil translasi biner dan kode karakter ditulis ke file output untuk dokumentasi. Metode ini membuktikan efisiensinya dalam mengurangi ukuran data tanpa kehilangan informasi, yang sangat berguna untuk kompresi tanpa hilang data.

REFERENCES

[1] Z. Shao et al., "A High-Throughput VLSI Architecture Design of Canonical Huffman Encoder," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 1, pp. 209-213, Jan. 2022, doi: 10.1109/TCSII.2021.3091611.

[2] X. Kavousianos, E. Kalligeros and D. Nikolos, "Test Data Compression Based on Variable-to-Variable Huffman Encoding With Codeword Reusability," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 7, pp. 1333-1338, July 2008, doi: 10.1109/TCAD.2008.923100.

[3] GeeksforGeeks. “Huffman Coding”. [Online]. Available: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/.