Hamming Distance based Optical Character Recognition in FPGA

By

AD.Vikram - 22BEC1018
Surya Thangaraju - 22BEC1098

A project report submitted to

Dr.Velmathi G
School of Electronics Engineering

in partial fulfilment of the requirements for the course of

BECE406E - FPGA based System Design

in

B Tech - Electronics and Communication Engineering

Vandalur – Kelambakkam Road
Chennai – 600127

APRIL 2025

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Optical Character Recognition"** is a bonafide work of A D VIKRAM -22BEC1018, SURYA THANGARAJU-22BEC1098,who carried out the Project work under my supervision and guidance for BECE406E - FPGA Based System Design.

Dr. Velmathi G
Professor-Higher Academic Grade
School of Electronics Engineering (SENSE),
VIT University, Chennai
Chennai – 600 127.

Abstarct

In the modern healthcare environment, **digital transformation** is crucial for improving efficiency, accuracy, and communication. This project presents a **smart medical information system** that integrates **Optical Character Recognition (OCR)**, **ModelSim-based character detection**, and a **user-friendly MATLAB GUI** to automate and simplify the process of extracting and displaying patient disease information from handwritten prescriptions.

The system initially focuses on recognizing individual handwritten characters such as **A, B, and C** using OCR, serving as a **proof of concept**. The extracted characters are then stored in a .mem file and processed using **Verilog in ModelSim**, where a **Hamming distance algorithm** matches the input with predefined templates. Building upon this foundation, the system is extended to detect **full disease names** such as "Malaria" or "Typhoid" from prescription images, enhancing the application's **practical utility**.

Instead of relying on full prescription OCR—which often results in **errors** due to messy handwriting and varied formats—the system smartly focuses on extracting **only the disease name**. This targeted approach significantly **improves accuracy** and reduces misinterpretation. Once the disease is identified, the user can input it into a **custom MATLAB GUI**, which instantly displays related information including **symptoms**, **causes**, and **treatment suggestions**.

To further enhance communication, the GUI includes an **email feature** that allows this information to be sent directly to the patient's email, thereby supporting **remote healthcare access** and **patient education**. The system is designed to be **simple and intuitive**, ensuring even non-technical users like **receptionists** and **medical staff** can operate it with ease.

Table of Contents

Introduction

1.1 Objectives

1)To implement a basic OCR system capable of detecting individual handwritten characters (e.g., A, B, C) using MATLAB and MODELSIM

2)To integrate the OCR output with Verilog in ModelSim, enabling character recognition using memory (.mem) files and Hamming distance-based matching.

3)To extend OCR functionality for recognizing complete handwritten **disease names** from prescription images.

4)To design a MATLAB GUI that allows users to manually input the detected disease and retrieve relevant medical information (symptoms, causes, treatment).

5)To enable email communication, sending disease-related information to the patient for improved engagement and digital healthcare support.

6)To ensure the system is simple and accessible, so that even non-technical users like receptionists or nurses can operate it effectively.

Goals

✔ Improve the reliability of disease recognition from handwritten prescriptions by avoiding full prescription OCR and focusing on disease name only.

✔ Provide a practical digital interface for medical staff to view, verify, and share disease-related information.

✔ Reduce manual errors in interpreting handwritten prescriptions, especially in critical diagnosis fields.

✔ Digitize patient interaction with minimal effort, enhancing both record-keeping and communication.

✔ Bridge hardware (ModelSim) and software (MATLAB GUI) to create an end-to-end automated medical data handling system.

✔ Demonstrate the application of character-level OCR in real-world medical scenarios with emphasis on disease identification and patient care.

1.2 Applications

- **Digital Prescription Management Systems**

Automatically extract the disease name from handwritten prescriptions and convert it into structured, digital data for hospitals and clinics.

- **Smart Hospital Reception Desks**

Help receptionists quickly retrieve patient-specific disease information by entering just the disease name, without reading the full prescription.

- **Patient Education Platforms**

Generate instant information (symptoms, causes, treatments) about a diagnosed disease and send it to patients via email for better understanding and compliance.

1.3 Features

- **Handwritten Character Recognition using OCR**

Detects handwritten characters from scanned prescription images using MATLAB's OCR capabilities.

- **Verilog-based Disease Name Detection in ModelSim**

Uses .mem files and compares character data with predefined templates using the Hamming distance algorithm for accurate disease identification.

- **Focused Disease Name Extraction**

Instead of parsing the entire messy prescription, only the disease name is extracted, improving accuracy and reducing errors.

- **MATLAB App Designer GUI**

A clean, intuitive interface that allows medical staff or receptionists to input the disease name manually and view related medical data.

- **Instant Display of Medical Information**

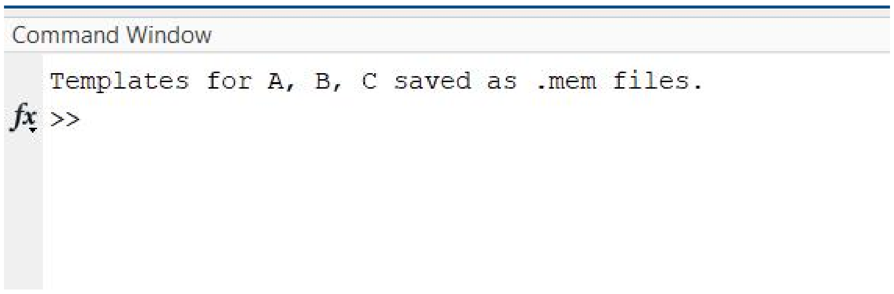
Once the disease name is entered, the GUI shows symptoms, causes, and treatment options instantly for quick reference.

Design and Implementation

2.1 Implementation of OCR for Letters(A,B,C,,)

Creating Predefined Templates for Characters

The MATLAB script generates predefined 20x20 binary templates for the letters A, B, and C. It first creates an image of each letter using a bold font and converts it to grayscale. The image is then resized and thresholded to produce a binary matrix (0s and 1s). The script saves each letter's binary representation into a .mem file, which can be used for comparison in an OCR system. Finally, the generated templates are displayed for verification.

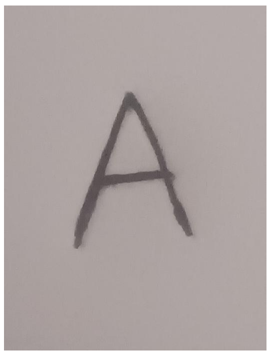
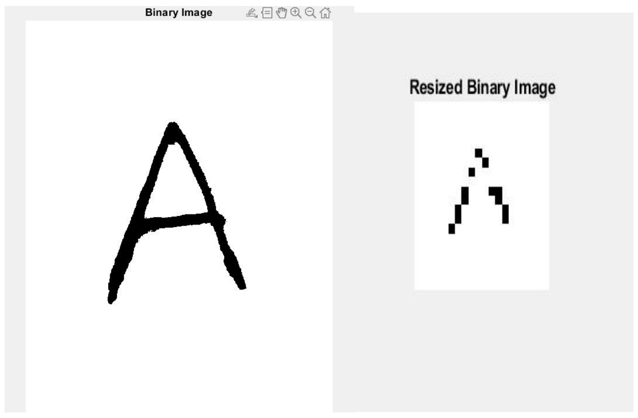


```
Templates for A, B, C saved as .mem files.
>>
```

Scanned Letters used for OCR

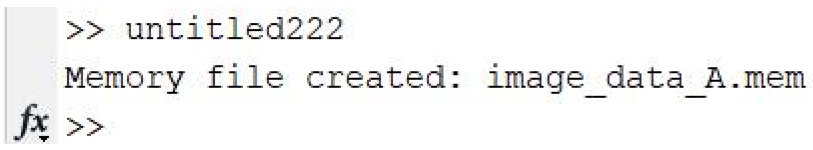
The MATLAB script processes a handwritten letter image for OCR matching. It first loads the image and converts it to grayscale if needed. Then, it applies thresholding to create a binary image, which is resized to 20x20 pixels for standardization. The pixel values are extracted and reshaped into a vector for feature extraction. Finally, the processed binary data is saved as a text file (processed_image_A.txt), making it compatible for further use in ModelSim.

Scanned Image : **Binary Image and Resized Binary Image :**

Converting the Processed Text to .mem file

The MATLAB script converts a processed image into a memory file (.mem) for ModelSim. It first opens a new file (image_data_A.mem) for writing. Then, it loads binary pixel values from a previously saved text file (processed_image_A.txt). Using a loop, it writes each pixel value into the memory file in a sequential manner. After writing all data, the file is closed to finalize the saving process. Finally, a confirmation message is displayed indicating successful memory file creation.



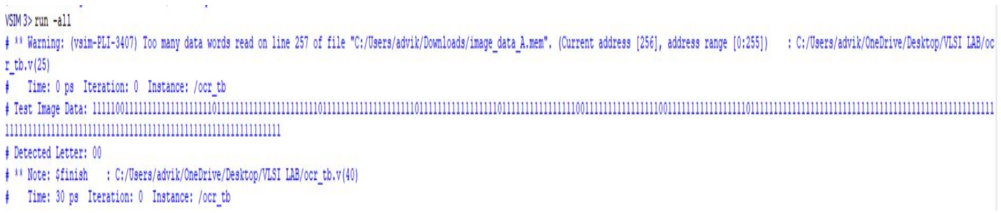
```
>> untitled222
Memory file created: image_data_A.mem
>>
```

Verilog Implementation of OCR Matching :

The Verilog module implements an OCR matcher using Hamming distance to compare a handwritten image with predefined templates of letters A, B, and C. It first loads the template data from .mem files into memory registers (template_A, template_B, template_C). A function compares the input image with each template, counting the number of matching pixels. These match scores are computed on each clock cycle, determining the similarity between the input image and stored templates. The letter with the highest match score is selected as the detected character (detected_letter). The module outputs a 2-bit encoded result representing A (00), B (01), or C (10). This simple and efficient design makes it suitable for FPGA-based character recognition systems.

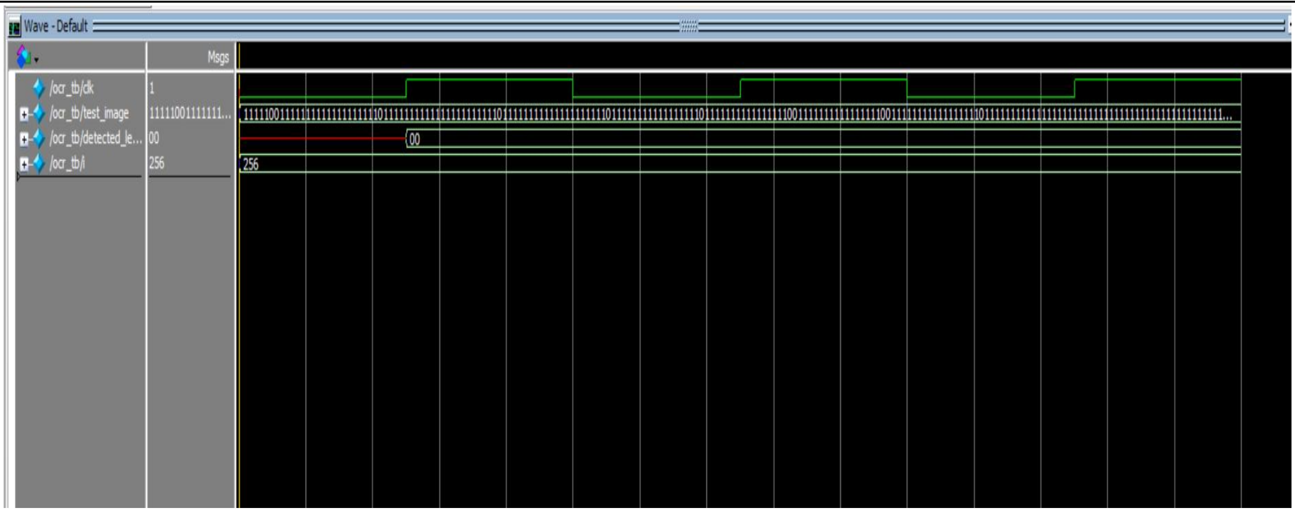
Scanned Image Recognition in MODELSIM :

The Verilog testbench (ocr_tb) is designed to verify the functionality of the OCR matcher module. It initializes a clock signal with a 50% duty cycle, ensuring proper timing for the module. The test image data is loaded from a .mem file (image_data_A.mem) using $readmemb. The binary pixel values are concatenated into a 256-bit test_image register for processing. The OCR matcher module (DUT) processes the test image and outputs the detected letter as a 2-bit encoded value. The testbench displays the input image data and the detected letter for verification.



```
VSIM 3> run -all
# ** Warning: (vsim-PLI-3407) Too many data words read on line 257 of file "C:/Users/advik/Downloads/image_data_A.mem". (Current address [256], address range [0:255])    : C:/Users/advik/OneDrive/Desktop/VLSI LAB/oc
r_tb.v(25)
#    Time: 0 ps  Iteration: 0  Instance: /ocr_tb
# Test Image Data: 111100111111111111111111011111111111111111111011111111111111111101111111111111111101111111111111111001111111111111111001111111111111111011111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111
# Detected Letter: 00
# ** Note: $finish    : C:/Users/advik/OneDrive/Desktop/VLSI LAB/ocr_tb.v(40)
#    Time: 30 ps  Iteration: 0  Instance: /ocr_tb
```

Detected Letter : 00(A)



```
/ocr_tb/clk              1
/ocr_tb/test_image       11111001111111...   11111001111111111111111111101111111111111111111111110111111111111111111111110111111111111111111111101111111111111111100111111111111111111001111111111111111111011111111111111111111111111111111111111...
/ocr_tb/detected_le...   00                  00
/ocr_tb/i                256                 256
```

Test Image (test_image) : The input image is successfully loaded as a 256-bit binary value, meaning the .mem file was read properly.

Detected Letter (detected_letter) : The output is "00", indicating that the system has identified the input as letter "A" (assuming 00 = A from the design).

2.2 Implementation of OCR for Disease

Process of OCR used :

1)**Image Preprocessing** :
RGB, it converts to grayscale.
Applies thresholding to create a binary (black-and-white) image if needed.
Applying deskewing and noise reduction to clean up the input.

2)Segmentation :
It breaks the image into individual lines, words, and characters using connected component analysis and whitespace spacing.

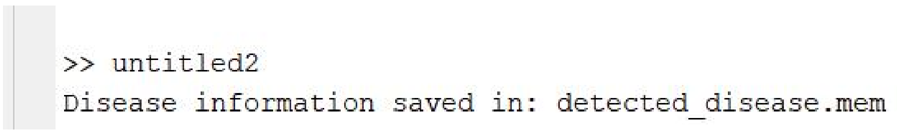
3)Character Recognition
Each character image is matched against known character templates using a neural network classifier trained on thousands of fonts and handwritings.
It assigns a character code (like 'A', 'b', '3', etc.)

Along with a confidence score (how sure it is that it read the correct letter)

<u>Matlab Code for Converting the image to Memory file</u>

```
% Step 1: Read the image (change to your image file path if necessary)
img = imread('prescription.jpg'); % Change to your image file path

% Step 2: Perform OCR on the image to extract the text
ocrResult = ocr(img);

% Step 3: Extract the disease name (assuming the OCR result is the disease name)
disease = strtrim(ocrResult.Text); % Clean up the text, remove any leading/trailing spaces

% Step 4: Check if a disease was detected
if isempty(disease)
error('Could not detect disease name. Please ensure the image is clear and contains only
the disease name.');
end

% Step 5: Convert the disease name into binary format (8-bit binary for each character)
diseaseBin = ''; % Initialize a variable to hold the binary values
for i = 1:length(disease)
diseaseBin = [diseaseBin; dec2bin(disease(i), 8)]; % Convert each character into 8-bit
binary and stack them
end

% Step 6: Save the binary values into a .mem file
memFile = 'detected_disease.mem'; % Name of the output .mem file
fileID = fopen(memFile, 'w'); % Open the .mem file for writing
for i = 1:size(diseaseBin, 1)
fprintf(fileID, '%s\n', diseaseBin(i, :)); % Write each binary value on a new line
end
fclose(fileID); % Close the .mem file

% Display confirmation message that the .mem file has been saved
disp(['Disease information saved in: ', memFile]);
```

First, the image is loaded using the imread function, and then the function is used to identify and extract the characters present in the image. The extracted text is cleaned using strtrim to remove any extra spaces, ensuring that only the disease name is considered. If no text is detected, the program displays an error message. Once the disease name is obtained, each character is converted into its corresponding 8-bit binary ASCII representation using the dec2bin function in a loop. These binary values are stored in a matrix format, with each line representing a character in binary. Finally, this binary data is written into a .mem file named detected_disease.mem using file handling functions like fopen and fprintf. This .mem file acts as the interface for the Verilog system, allowing the detected disease name to be processed and displayed during simulation.


```
>> untitled2
Disease information saved in: detected_disease.mem
```

The .mem file is saved in the same folder as matlab files,copying this file into verilog folder(Work) for simulation

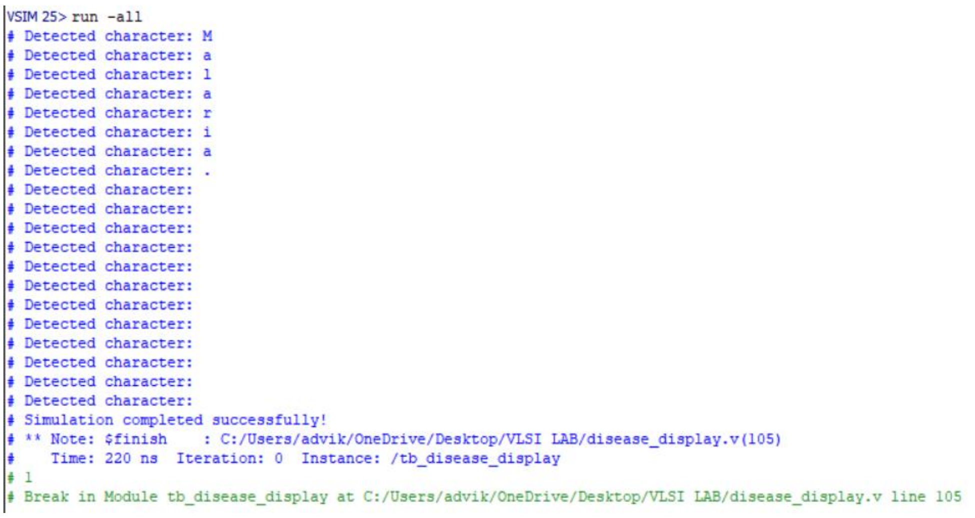
Verilog Code for Detecting the Characters in the Handwritten Disease name

```
module disease_display (
   input clk,               // Clock input
   input rst,               // Reset input
   output [7:0] char_out,     // 8-bit output for the character to display
   output reg [7:0] disease    // Register to store the detected disease name (one character at a
time)
);

   // Parameters for memory size
   integer i;
   integer mem_file;
   reg [7:0] mem [0:255];  // Memory array to hold binary values (max 256 characters)

   // Register to track the position of the character in the .mem file
   reg [7:0] index;

   initial begin
      // Initialize the index to 0
      index = 0;

      // Open the .mem file containing the disease name in binary
      mem_file = $fopen("detected_disease.mem", "r");
      if (mem_file == 0) begin
         $display("Error: Could not open .mem file.");
         $finish;
      end

      // Read the .mem file into the memory array
      for (i = 0; i < 256; i = i + 1) begin
         if ($fscanf(mem_file, "%b\n", mem[i]) == 0) begin
            $display("Error: Unable to read data from the .mem file.");
            $finish;
         end
      end
```

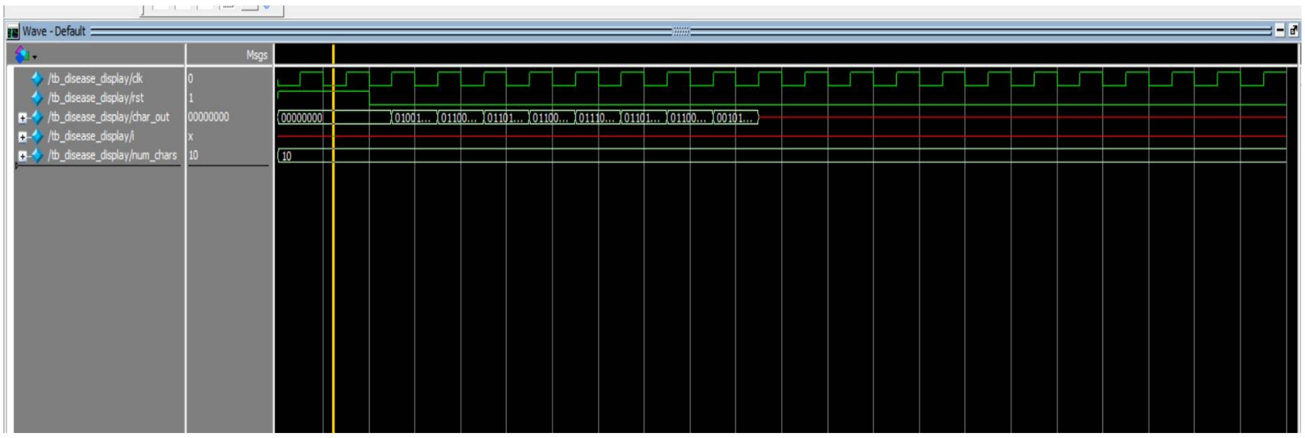
```
    // Close the file after reading
    $fclose(mem_file);
  end

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      // Reset logic: Reset the index and disease character
      index <= 0;
      disease <= 8'b0;
    end else begin
      // Output the current character from memory
      disease <= mem[index];

      // Display the detected character for the disease (for debugging)
      $display("Detected character: %c", mem[index]);

      // If the character has been displayed, move to the next one
      if (index < 255) begin
        index <= index + 1;
      end else begin
        // If we have reached the end of the disease name, loop back
        index <= 0;
      end
    end
  end

  // Output the current character to be displayed
  assign char_out = disease;

Endmodule
```

The Verilog code is designed to read the .mem file generated from the image processing phase. This file contains 8-bit binary representations of each character in the disease name. The code uses a memory array to store these binary values. On each clock cycle, one character is fetched from the memory and assigned to the output. This character is interpreted using its ASCII value, allowing the detected disease name to be displayed sequentially. A testbench is also used to simulate clock pulses and verify the output. This simulation helps ensure the correct interpretation of the disease name in ModelSim, validating the communication between the image processing output and representation.

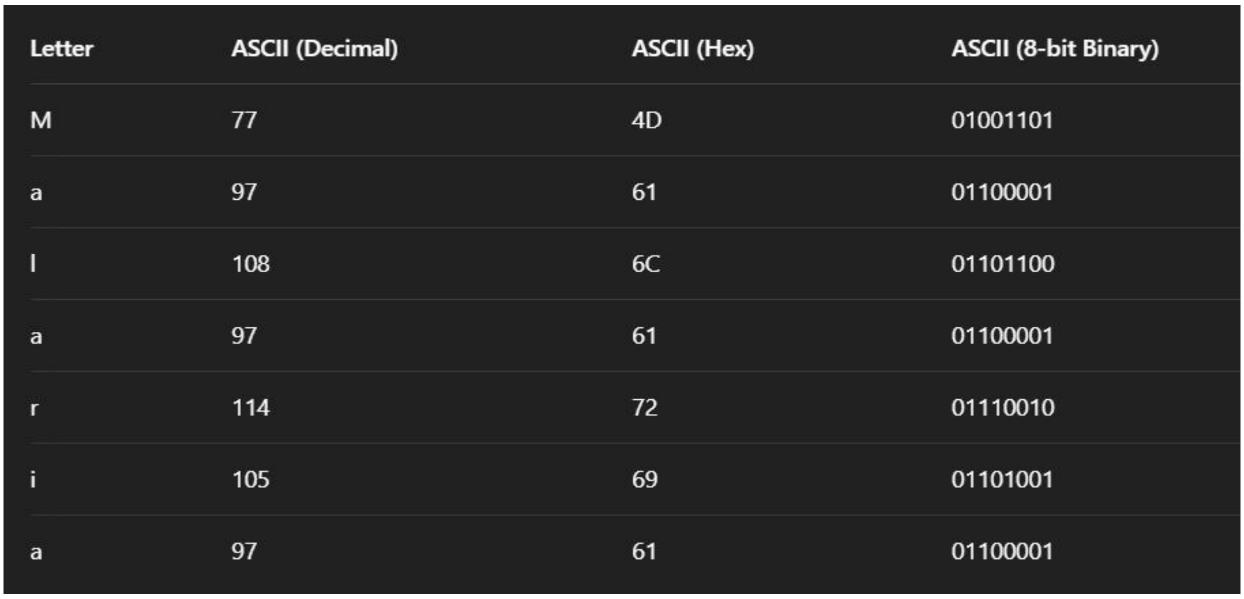
Output of ModelSim displaying the detected letters



```
VSIM 25> run -all
# Detected character: M
# Detected character: a
# Detected character: l
# Detected character: a
# Detected character: r
# Detected character: i
# Detected character: a
# Detected character: .
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Detected character:
# Simulation completed successfully!
# ** Note: $finish    : C:/Users/advik/OneDrive/Desktop/VLSI LAB/disease_display.v(105)
#    Time: 220 ns  Iteration: 0  Instance: /tb_disease_display
# 1
# Break in Module tb_disease_display at C:/Users/advik/OneDrive/Desktop/VLSI LAB/disease_display.v line 105
```



The detected letters are being displayed in the transcript window and the ASCII Values of the letters are being displayed in the Waveform



Letter	ASCII (Decimal)	ASCII (Hex)	ASCII (8-bit Binary)
M	77	4D	01001101
a	97	61	01100001
l	108	6C	01101100
a	97	61	01100001
r	114	72	01110010
i	105	69	01101001
a	97	61	01100001

Application - Medical Prescription(Smart Health App)

Creating a GUI design for medical Staff or receptionists :

- **User Input for Disease** :

A field to manually enter the detected disease name (extracted from handwritten prescription via MATLAB and processed in ModelSim).

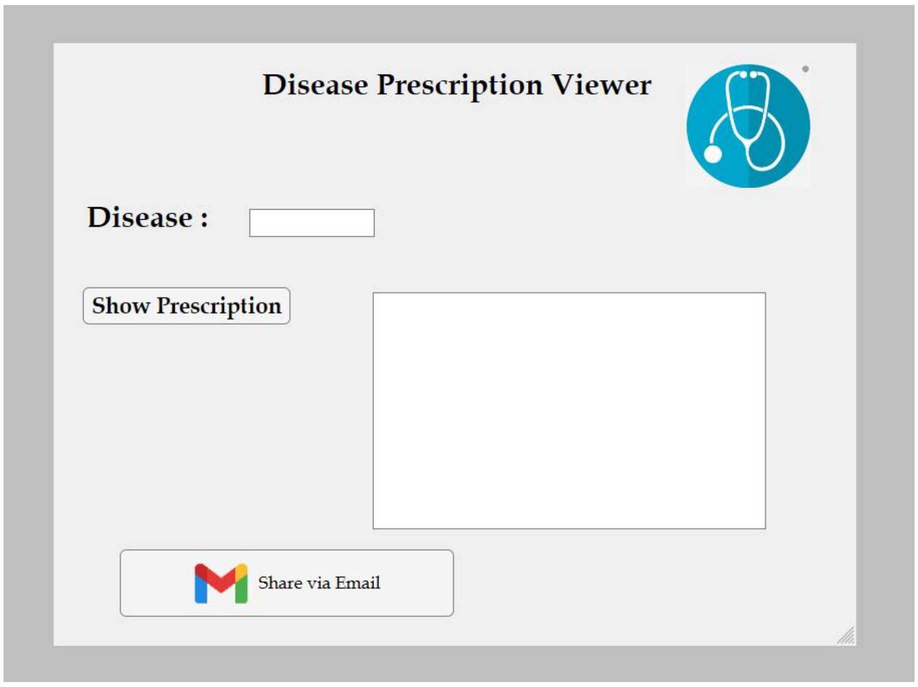
- **Display of Disease Information :**

Once the disease is entered, the GUI automatically displays related information, including:
Symptoms ; Causes ; Treatment suggestions

- **Email Integration :**

An option to send the displayed disease-related information directly to the patient's email.

GUI Design



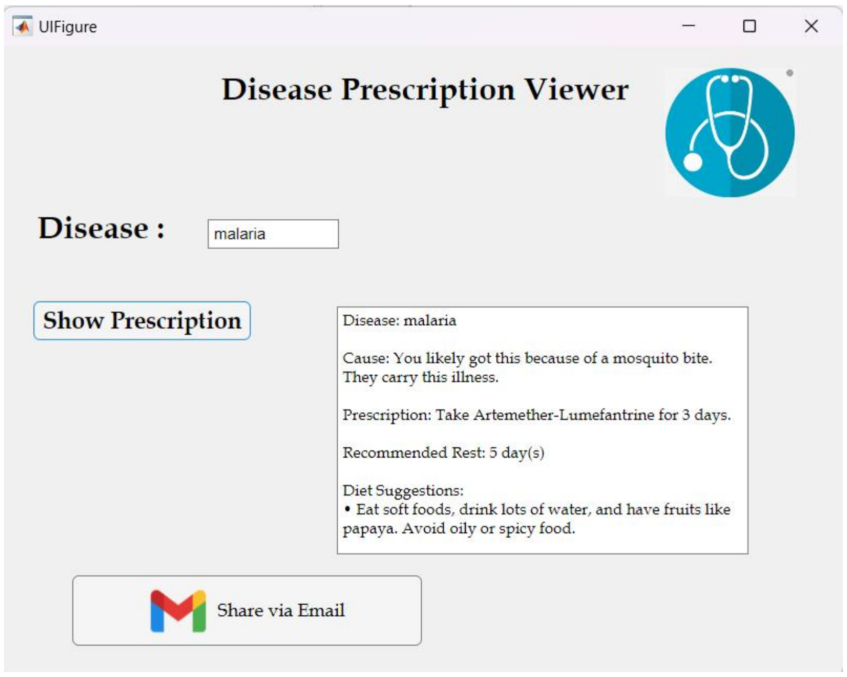
1)Entering the disease obtained through OCR in the EditField of the Disease Box.

2)When Clicked on the Show Prescription button it displays the cause , Prescription , Recommended days and Diet to be followed.

3)All these will be displayed in the TextArea as shown there.

4) We can send these Information Directly to the patient via there email.

Output of the Prescription :



malaria

Disease: malaria

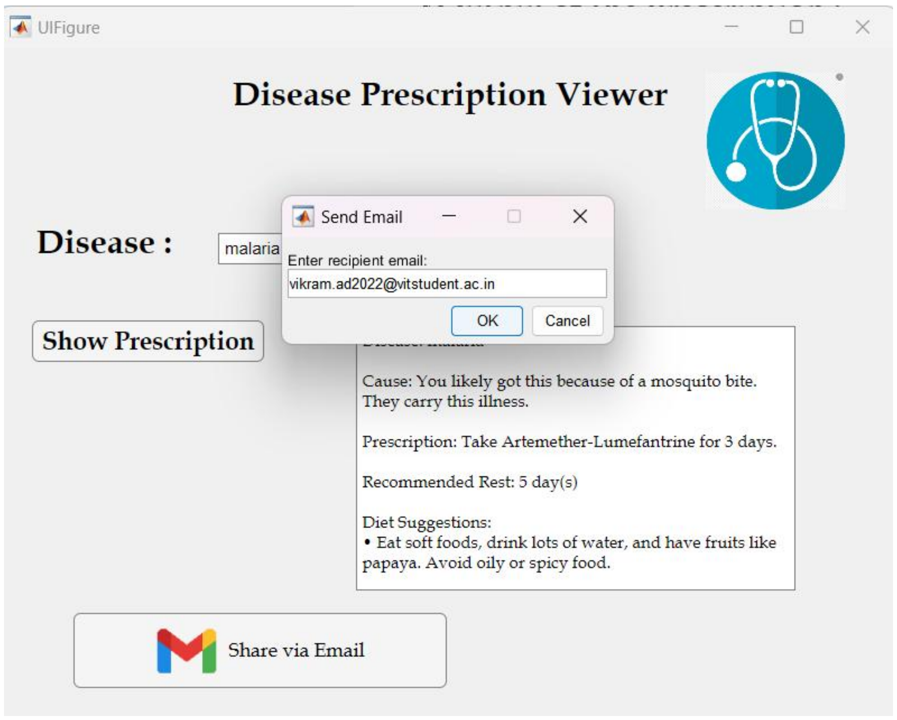
Cause: You likely got this because of a mosquito bite. They carry this illness.

Prescription: Take Artemether-Lumefantrine for 3 days.

Recommended Rest: 5 day(s)

Diet Suggestions:
• Eat soft foods, drink lots of water, and have fruits like papaya. Avoid oily or spicy food.

Sending the data to patient's mail



malaria

Enter recipient email:

vikram.ad2022@vitstudent.ac.in

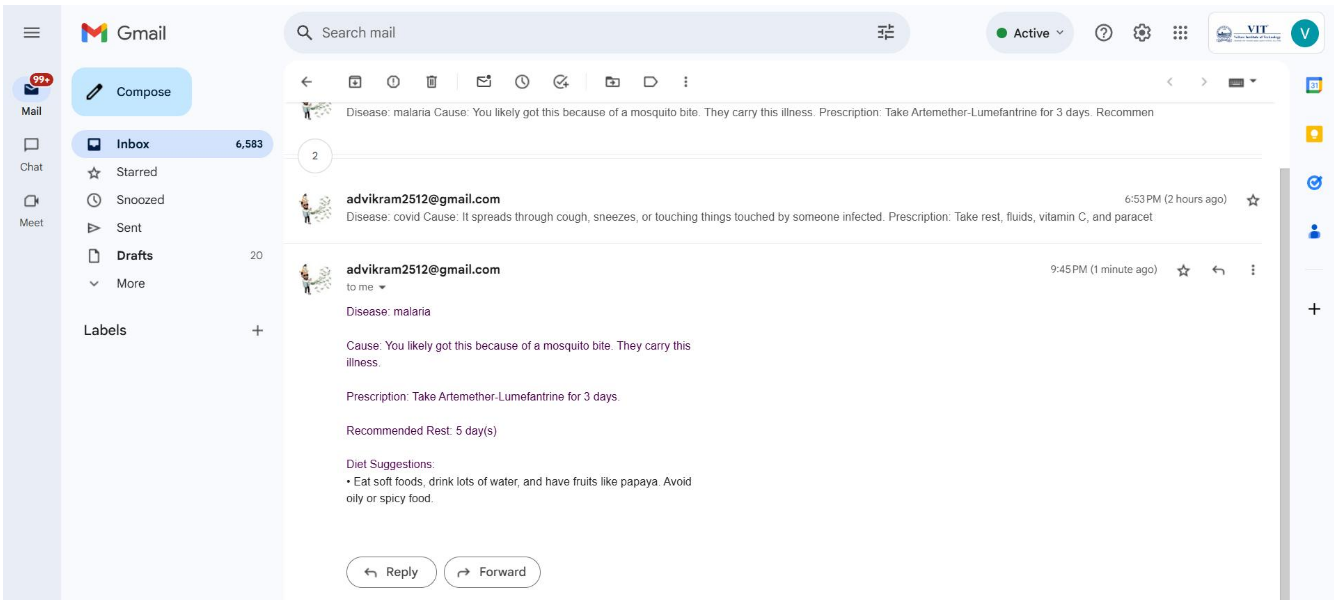
Cause: You likely got this because of a mosquito bite. They carry this illness.

Prescription: Take Artemether-Lumefantrine for 3 days.

Recommended Rest: 5 day(s)

Diet Suggestions:
• Eat soft foods, drink lots of water, and have fruits like papaya. Avoid oily or spicy food.

Email Sent Successfully



Disease: malaria Cause: You likely got this because of a mosquito bite. They carry this illness. Prescription: Take Artemether-Lumefantrine for 3 days. Recommen

advikram2512@gmail.com 6:53 PM (2 hours ago)
Disease: covid Cause: It spreads through cough, sneezes, or touching things touched by someone infected. Prescription: Take rest, fluids, vitamin C, and paracet

advikram2512@gmail.com 9:45 PM (1 minute ago)
to me

Disease: malaria

Cause: You likely got this because of a mosquito bite. They carry this illness.

Prescription: Take Artemether-Lumefantrine for 3 days.

Recommended Rest: 5 day(s)

Diet Suggestions:
• Eat soft foods, drink lots of water, and have fruits like papaya. Avoid oily or spicy food.

Conclusion and Future Work

3.1 **Result,Conclusion and Inference**

Conclusion :

Limitations of Using OCR on Full Handwritten Prescriptions (Real-time Data)

➢ **Character-by-Character Recognition :**

OCR reads text one character at a time, making it prone to errors in context (e.g., "Dengue" might be read as "Dengu3" or "DenguC").

This results in inaccurate or meaningless outputs when trying to extract information from entire prescriptions.

➢ **Complex Layouts :**

Prescriptions include mixed content : patient details, multiple medications, dosages, doctor signatures.

OCR lacks the intelligence to understand structure or context in such complex handwritten documents.

Inference :

- **Focus on What Matters** :

Instead of processing the whole prescription, just extract and manually enter the disease name.

The disease name is usually short and distinct (e.g., "Typhoid", "Malaria"), making it easier for OCR to detect accurately.

- **Combine with GUI for Digitization** :

Once the disease name is typed into the GUI, the system can instantly fetch:

Symptoms
Causes
Treatments
And even email the information to patients

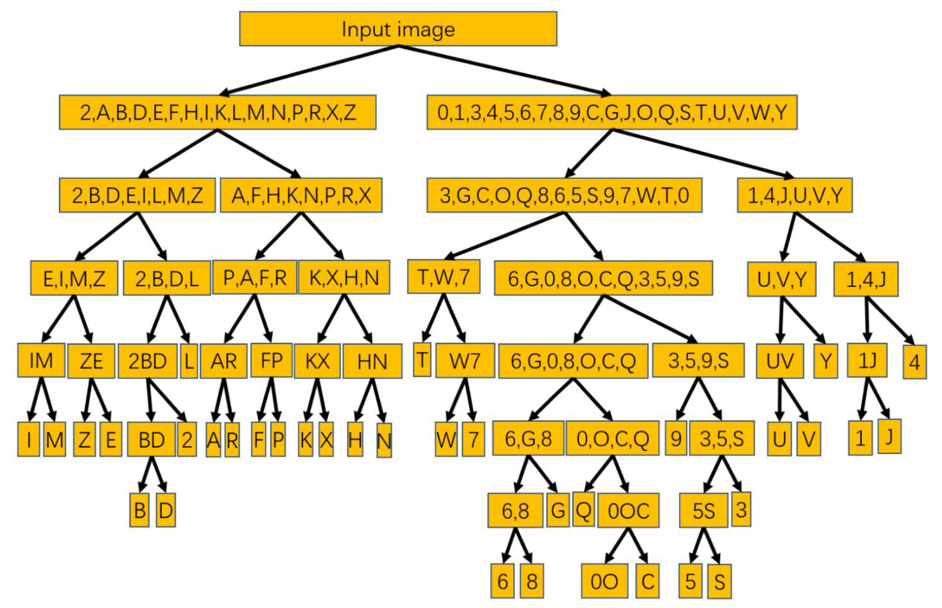
This approach ensures precision, speed, and clarity without depending on messy OCR interpretations.

3.2 Future Work

The current system effectively demonstrates character-level disease recognition and information display through a hybrid OCR and GUI-based approach. In future work, the project can be expanded to support **multi-language OCR** to detect disease names written in regional scripts (e.g., Hindi, Malayalam). Additionally, **deep learning models** like CNNs or RNNs can be integrated to enhance recognition accuracy for complex handwriting styles. The GUI can be upgraded with features like **voice input**, **QR code generation**, and **database integration** for storing patient records securely. Moreover, deploying the system as a **mobile app or web platform** would extend its accessibility, making it a scalable solution for rural clinics, telemedicine platforms, and home healthcare providers.

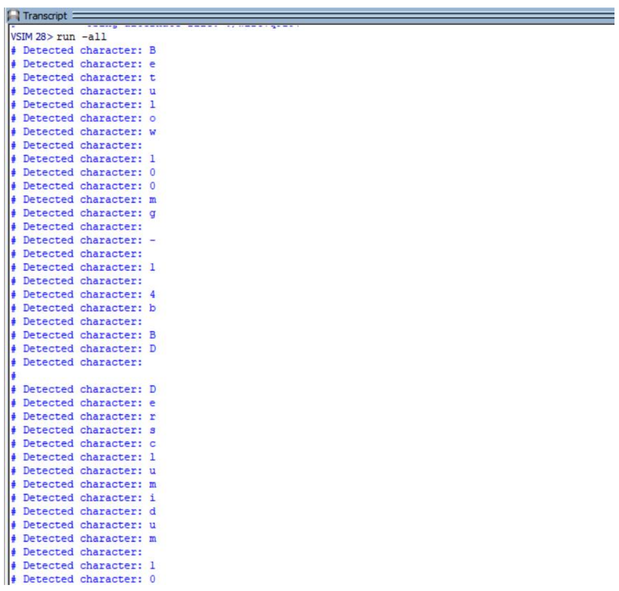
Comparsion of Results with Referred Paper

Reference Paper Results :



The image represents the output of a memory-based algorithm used for **Optical Character Recognition (OCR)**, specifically designed for recognizing alphanumeric characters from input images. The approach utilizes a hierarchical decision tree structure, where the input image is passed through successive levels of classification based on shared visual features among characters. At each node in the tree, the character set is narrowed down using predefined memory patterns, allowing the system to efficiently distinguish between similar-looking symbols. This structured method significantly reduces the complexity of the recognition process by eliminating unlikely options early on and focusing only on a smaller subset of characters at each step. The final recognized character is determined when the traversal reaches a leaf node, which stores the exact match. This memory-based decision tree approach is particularly useful for low-resource environments, as it emphasizes speed and accuracy while minimizing computational overhead.

Betterment of this Results :



```
VSIM 28> run -all
# Detected character: B
# Detected character: e
# Detected character: t
# Detected character: u
# Detected character: l
# Detected character: o
# Detected character: w
# Detected character:
# Detected character: 1
# Detected character: 0
# Detected character: 0
# Detected character: m
# Detected character: g
# Detected character:
# Detected character: -
# Detected character:
# Detected character: 1
# Detected character:
# Detected character: 4
# Detected character: b
# Detected character:
# Detected character: B
# Detected character: D
# Detected character:
#
# Detected character: D
# Detected character: e
# Detected character: r
# Detected character: s
# Detected character: c
# Detected character: l
# Detected character: u
# Detected character: m
# Detected character: i
# Detected character: d
# Detected character: u
# Detected character: m
# Detected character:
# Detected character: 1
# Detected character: 0
```

The result shows the output of our OCR system, which uses a **template matching approach** based on **Hamming distance** to identify characters from an input image. In this method, each segmented character from the input is compared against a set of predefined binary templates stored in memory. By calculating the Hamming distance—the number of differing bits between the input character and each template—the system selects the closest match. Once the best match is found, its corresponding **ASCII value** is retrieved and used to display the detected character. This process continues for each character in the input image, as seen in the ModelSim output log.

Additionally, our project includes a **MATLAB GUI** that enhances functionality by allowing the user to input a disease name (detected through the OCR system), and then view related prescription or disease information. **This complete pipeline—from image input and character recognition using Hamming distance, to GUI-based**

disease information retrieval—demonstrates an efficient and user-friendly application of OCR in the medical domain.

References

1)Memory-Tree Based Design of Optical Character Recognition in FPGA

by Ke Yu , Minguk Kim and Jum Rim Choi ,
https://doi.org/10.3390/electronics12030754