VJTI — SY BTECH EXTC

OPTICAL CHARACTER RECOGNITION INTEGRATED NUMERICAL ARITHMETIC USING ESP8266

Digital Logic Design (DLD) Project Report

Submitted By:

1. Zaid Faruqui	241090024
2. Ananya Rane	241091062
3. Arnav Shelke	241090070
4. Mayuresh Surve	241090076
5. Vedant Malkar	241090080

Course Instructor: Sonal Gedam

Academic Year: 2025–2026

Contents

Abstract

This project implements a real-time handwritten number recognition and digital BCD arithmetic system using Python (OpenCV + MediaPipe + PyTesseract), ESP8266 (serial communication), and hardware-based addition using two 4-bit binary adders (74LS83), BCD correction logic and 7448 seven-segment decoders. The user draws digits in the air using hand gestures detected via MediaPipe, which are processed through Optical Character Recognition (OCR) to extract two decimal digits. These values are transmitted to ESP8266 over serial communication and converted into binary outputs to drive ICs implementing a hardware BCD adder. If the sum exceeds 9, a correction (0110) is added using a second 74LS83, and the result is decoded using 7448 drivers connected to Common Cathode 7-segment displays. This pipeline integrates software intelligence with hardware digital logic, demonstrating real-time physical computation using binary arithmetic and logic design.

Chapter 1

Introduction

Gesture-based digit recognition is an emerging interaction technique replacing conventional input hardware. By combining computer vision, OCR, embedded microcontrollers, and digital hardware arithmetic, the system provides an advanced demonstration of real-time hybrid computation.

The motivation for this project was to build an end-to-end pipeline where numbers drawn by a human hand are recognized, processed, transmitted to hardware, added using pure digital logic ICs, error-corrected as BCD, decoded, and finally displayed on 7-segment displays. Instead of performing addition in software, decimal addition is executed using standard digital logic adders to illustrate ripple-carry propagation, hardware timing, propagation delay, and correction logic.

This project demonstrates:

- Integration of software-based computer vision and real-time hardware computation
- Understanding and implementation of 4-bit adder ICs and BCD correction
- Seven-segment driving through BCD decoders
- Serial interfacing and voltage-level digital control

Chapter 2

Components Used

Sr.No	Component	Specification	Quantity
1	ESP8266 NodeMCU	Wi-Fi Microcontroller, 3.3V logic	1
2	74LS83	4-bit Binary Full Adder	2
3	7448 BCD to 7-Segment Decoder	Common Cathode compatible	2
4	7432	OR Gate	1
5	7408	AND Gate	1
6	7-Segment Display	Common Cathode	2
7	Resistors (220)	Current limiting for display	14
8	Jumper Wires	M-M	-
9	Breadboards	Full and Mini	2
10	5V Power Supply	Power Supply (12V - 5V Buck Converter)	1

IC Pin Diagrams :

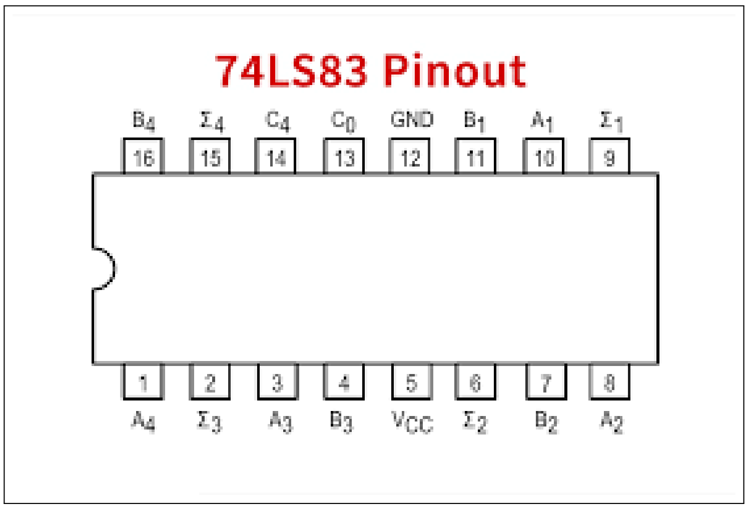


Figure 2.1: 74LS83 Pinout Diagram

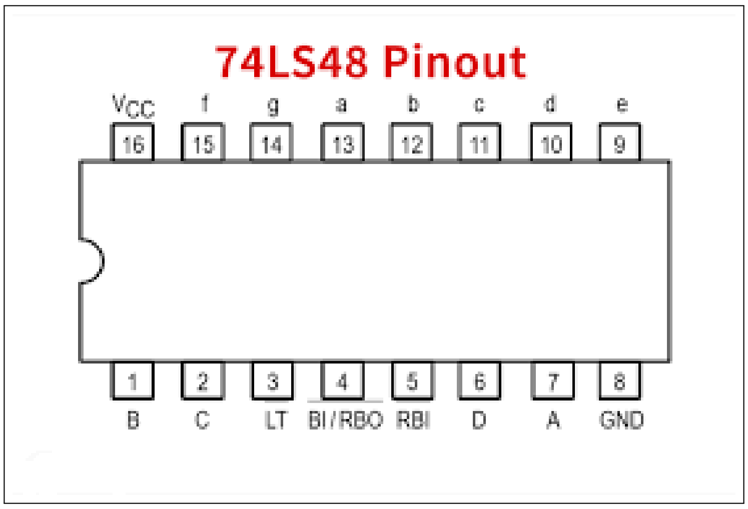


Figure 2.2: 7448 BCD Decoder Diagram

Chapter 3

Theory and Pipeline Flow

3.1 Overall Pipeline

1. Webcam captures frames using OpenCV
2. MediaPipe Hands extracts fingertip path
3. Drawn digit processed as image canvas
4. PyTesseract OCR extracts numbers → output string “num1, num2”
5. ESP8266 receives values over Serial
6. Converts decimal numbers to binary and sets GPIO pins
7. First 74LS83 computes 4-bit addition
8. Correction logic: if Sum ¿ 9, add 0110
9. Second 74LS83 adds correction and generates BCD output
10. 7448 decodes to 7-segment display

$$\begin{tikzcd}
\boxed{\text{Webcam (OpenCV)}} \arrow[d] \\
\boxed{\text{MediaPipe Hands}} \arrow[d] \\
\boxed{\text{PyTesseract OCR}} \arrow[d] \\
\boxed{\text{ESP8266 Serial}} \arrow[d] \\
\boxed{\text{74LS83 Adder 1}} \arrow[d] \\
\boxed{\text{BCD Correction Logic}} \arrow[d] \\
\boxed{\text{74LS83 Adder 2}} \arrow[d] \\
\boxed{\text{7448 Decoders}} \arrow[d] \\
\boxed{\text{7-Segment Displays}}
\end{tikzcd}$$

Figure 3.1: System Pipeline

3.2 Binary Addition and BCD Correction Theory

A 4-bit adder can represent sum values 0–15. However, decimal digits require 0–9 only. When the raw sum $S = A + B$ exceeds 9, an **invalid BCD state** occurs. Therefore a correction value 6 (0110) must be added.

$$S = A + B + C_{in}$$

$$\text{If } S > 9 \text{ then } S_{BCD} = S + 6$$

3.2.1 Condition for Correction Generation

$$CC = C_4 + S_3S_2 + S_3S_1$$

Where:

- C_4 = carry out from first 74LS83
- S_3, S_2, S_1 = MSBs of computed sum

  However, since our inputs were constrained between 0-7, maximum sum possible was within 0-14 and therefore due to the impossibility of any sum greater than 15, C_4 was kept unconnected due to redundancy (as C_4 will always remain 0 ; 0 + A = A).

Table 3.1: BCD Correction Truth Table

Decimal Sum	Binary Sum	Valid BCD?	CC	Add 6?
0–9	0000–1001	Yes	0	No
10–15	1010–1111	No	1	Yes

3.3 7448 BCD to Seven Segment Decoder Operation

The 7448 is a BCD (Binary Coded Decimal) to seven-segment decoder driver IC designed to convert 4-bit binary inputs (D, C, B, A) into seven outputs (a, b, c, d, e, f, g) which control the illumination of a seven-segment display. The 7448 produces **active-LOW outputs**, meaning a logic 0 on any segment output terminal activates (lights) that segment when used with a **Common Cathode** seven-segment display.

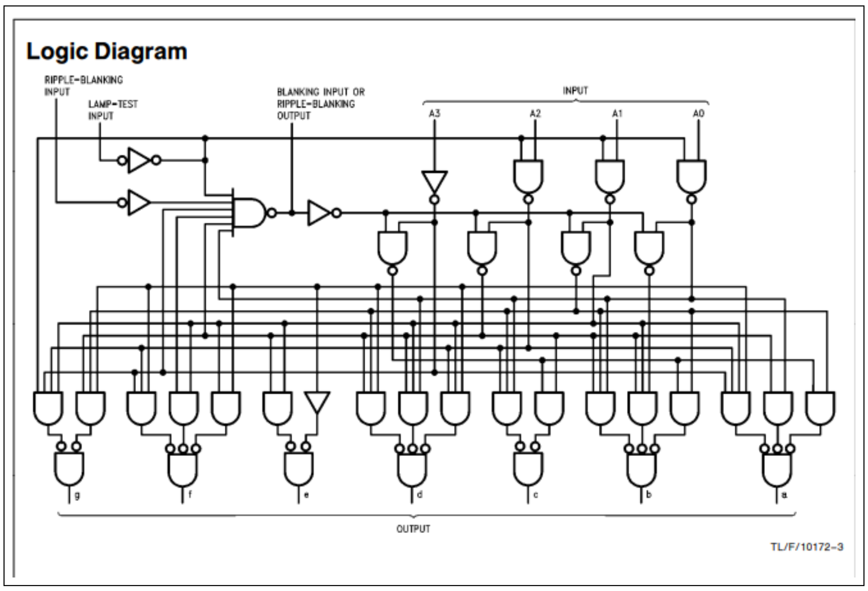


Figure 3.2: Internal functional representation of 7448 decoder

Input Line Interpretation

The decoder accepts a 4-bit BCD input from 0000 (0 decimal) to 1001 (9 decimal). The inputs correspond to:

$$A = \text{LSB}, \quad B, \quad C, \quad D = \text{MSB}$$

Inputs from 1010 to 1111 are invalid in BCD and produce blank or undefined segment outputs.

Segment Output Logic

Each output line controls one display segment. The segments are labelled:

$$a, b, c, d, e, f, g$$

For a seven-segment CC display, each segment LED lights when:

$$V_{segment} = 5V$$

Thus, the 7448 must generate LOW for segments that should turn ON. For example, for displaying digit 0, segments a, b, c, d, e, and f are ON while segment g is OFF.

Segment Boolean Equations

Each segment output is derived using combinational logic inside the decoder. Example Boolean expressions:

$$a = \overline{D + B + (C \cdot A) + (\overline{C} \cdot \overline{A})}$$

$$b = \overline{\overline{C} + \overline{B} \cdot \overline{A} + (B \cdot A)}$$

$$c = \overline{\overline{B} + A + C}$$

Similar expressions exist for remaining segments. The output is inverted through built-in transistor drivers to sink current when the segment must turn ON.

Why Common Cathode Display is Required

- 7448 outputs are active-HIGH, meaning ON = 5V
- Common Cathode display shares ground reference at cathode pins
- Segment turns ON when anode receives HIGH from decoder
- If Common Anode were used, outputs (LOW ON) would short the display supply

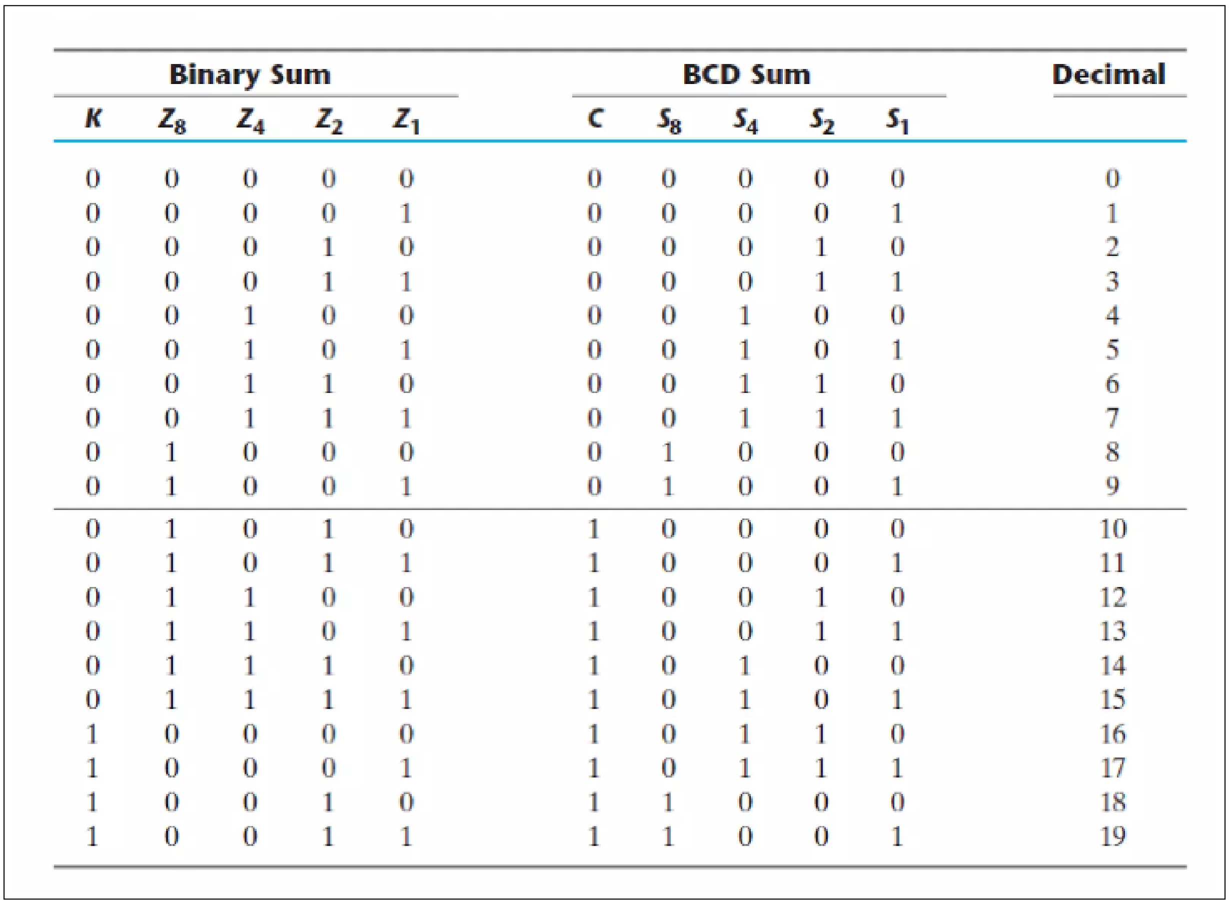
Display Driving and Current Limiting

Each segment must include a series resistor (typically 220–330) to limit LED current:

$$R = \frac{V_{cc} - V_f}{I_f}$$

Thus 220 resistor was selected to ensure brightness and safe operation.

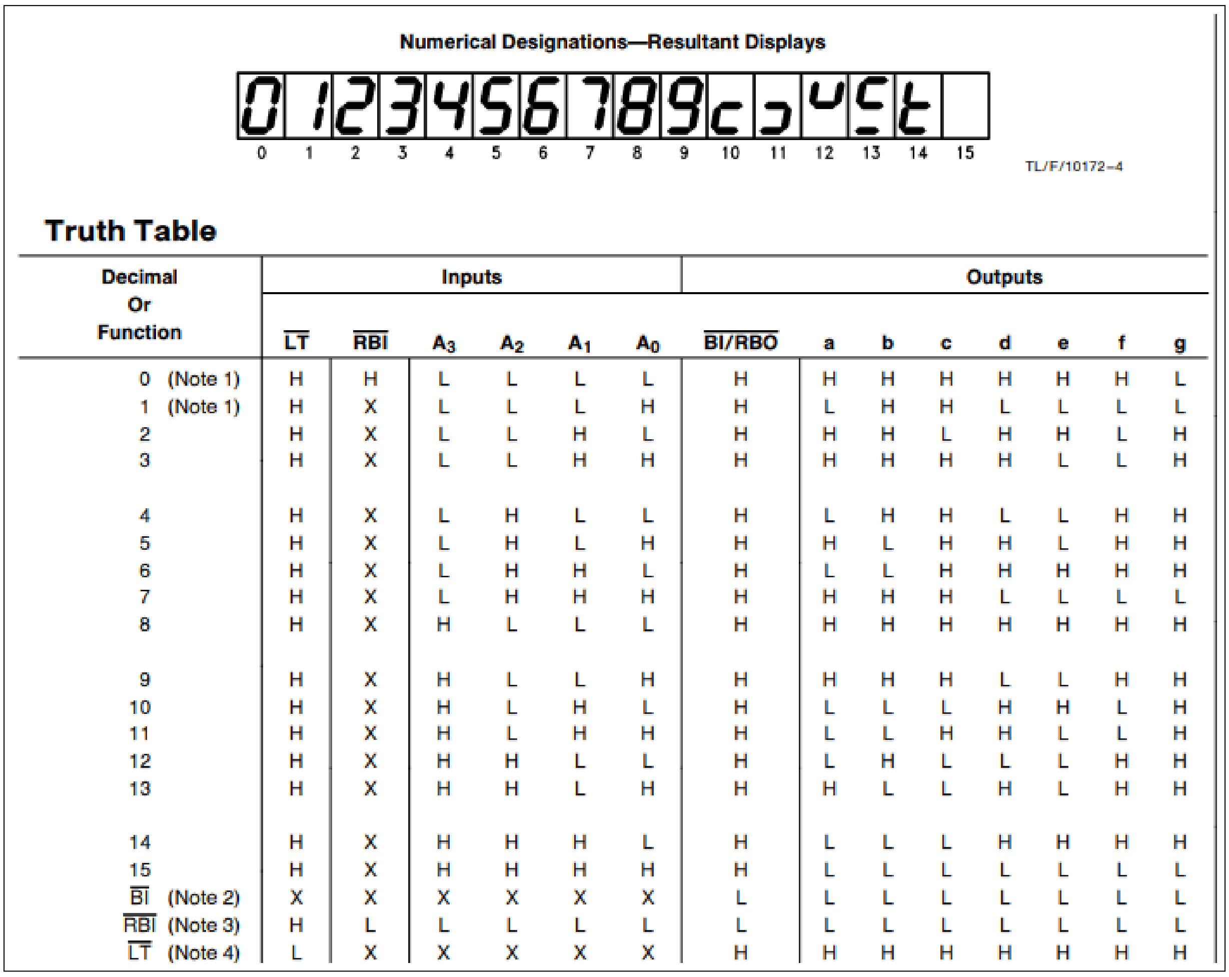
3.4 Truth Tables



Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Figure 3.3: BCD Adder Truth Table

.



Numerical Designations—Resultant Displays

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

TL/F/10172–4

Truth Table

Decimal Or Function	Inputs							Outputs						
	$\overline{LT}$	$\overline{RBI}$	A_3	A_2	A_1	A_0	$\overline{BI}/\overline{RBO}$	a	b	c	d	e	f	g
0 (Note 1)	H	H	L	L	L	L	H	H	H	H	H	H	H	L
1 (Note 1)	H	X	L	L	L	H	H	L	H	H	L	L	L	L
2	H	X	L	L	H	L	H	H	H	L	H	H	L	H
3	H	X	L	L	H	H	H	H	H	H	H	L	L	H
4	H	X	L	H	L	L	H	L	H	H	L	L	H	H
5	H	X	L	H	L	H	H	H	L	H	H	L	H	H
6	H	X	L	H	H	L	H	L	L	H	H	H	H	H
7	H	X	L	H	H	H	H	H	H	H	L	L	L	L
8	H	X	H	L	L	L	H	H	H	H	H	H	H	H
9	H	X	H	L	L	H	H	H	H	H	L	L	H	H
10	H	X	H	L	H	L	H	L	L	L	H	H	L	H
11	H	X	H	L	H	H	H	L	L	H	H	L	L	H
12	H	X	H	H	L	L	H	L	H	L	L	L	H	H
13	H	X	H	H	L	H	H	H	L	L	H	L	H	H
14	H	X	H	H	H	L	H	L	L	L	H	H	H	H
15	H	X	H	H	H	H	H	L	L	L	L	L	L	L
$\overline{BI}$ (Note 2)	X	X	X	X	X	X	L	L	L	L	L	L	L	L
$\overline{RBI}$ (Note 3)	H	L	L	L	L	L	L	L	L	L	L	L	L	L
$\overline{LT}$ (Note 4)	L	X	X	X	X	X	H	H	H	H	H	H	H	H

Figure 3.4: 7-Segment Decoder Truth Table

Chapter 4

Circuit Diagram

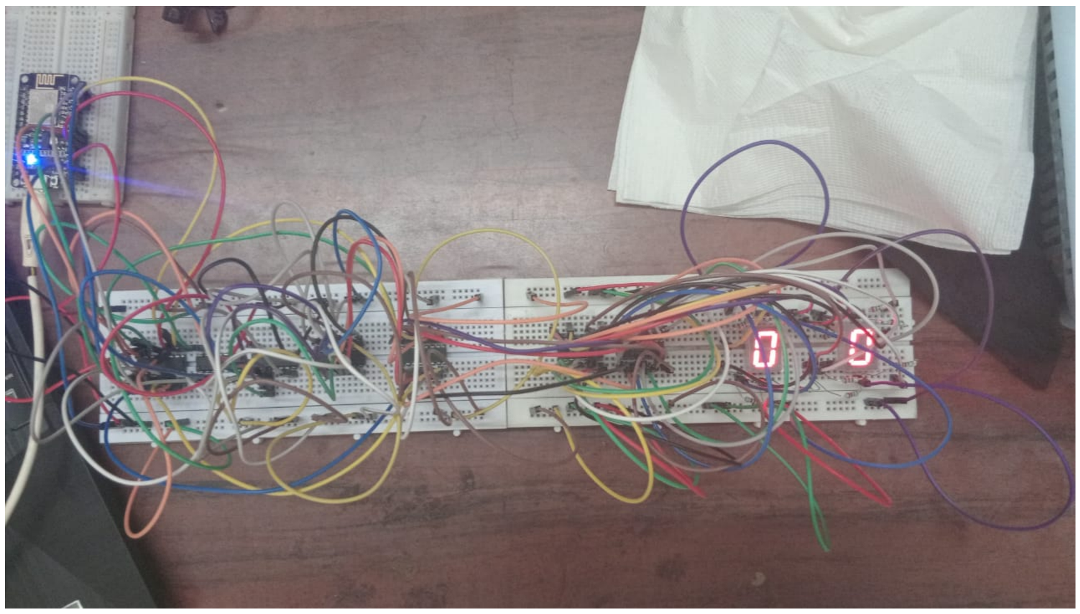


Figure 4.1: Full Hardware Circuit

Explanation of key wiring:

- ESP GPIO D5,D6,D7 drive num1 bits
- ESP GPIO D0,D1,D2 drive num2 bits
- The first 74LS83 $A0 - A3$ $B0 - B3$ receive these signals
- 74LS32 results in $S1$ OR $S2$
- 74LS08 results in $C_C =$ $S1$ AND ($S1$ OR $S2$)
- 74LS83 adds $S0 - S3$ and $0, C_C, C_C, 0$
- The first 74LS48 decodes $0, 0, 0, C_o$

- The first 74LS48 decodes $S0 - S3$
- The first 7-Segment displays output of first 74LS48 decoder
- The second 7-Segment displays output of second 74LS48 decoder

Chapter 5

Simulation (Multisim Results)

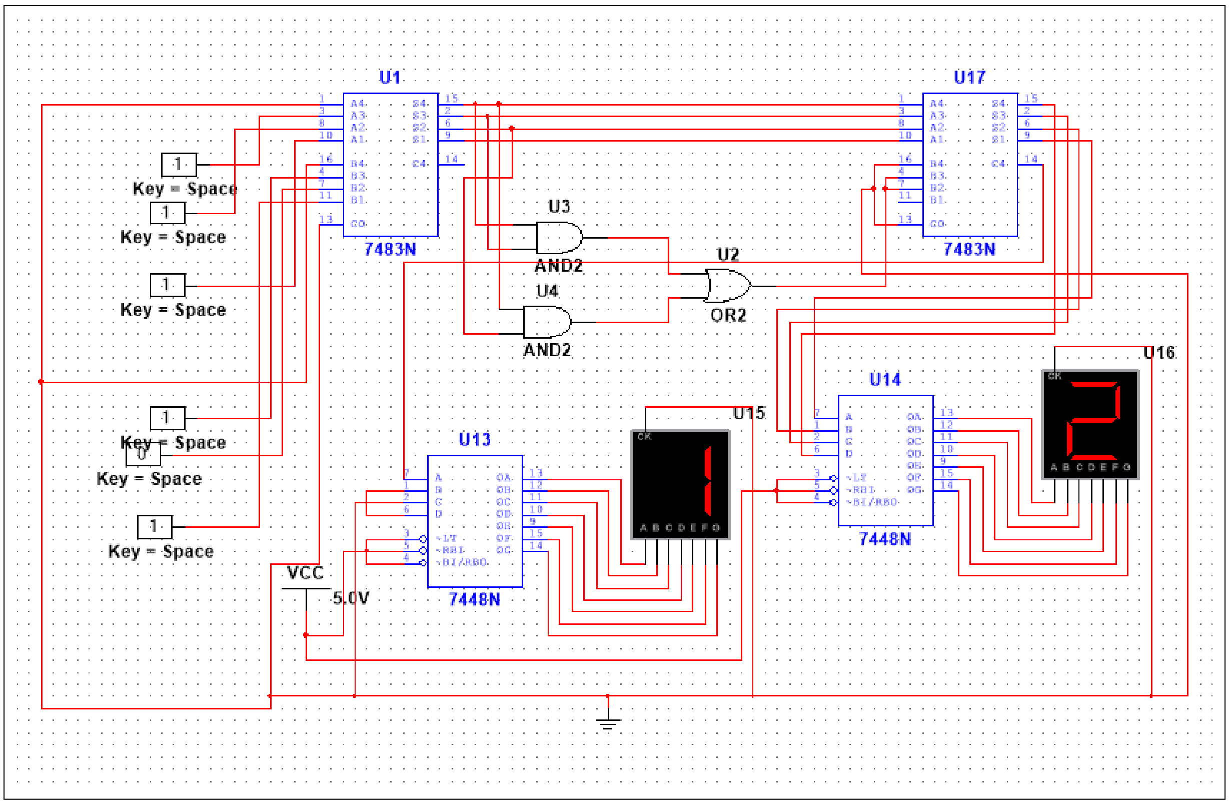


Figure 5.1: Simulated Circuit Output

Chapter 6

Outputs

Input (num1,num2)	Raw Sum	Final BCD	Display Output
2,5	0111 (7)	0000 0111	07
7,5	1100 (12)	0001 0010	12

Chapter 7

Challenges

During the development of this project, challenges were encountered and resolved through debugging, experimentation, and iterative design improvements:

- Achieving stable and reliable OCR accuracy was difficult due to variations in handwriting thickness, movement speed, background noise, and finger-tracking jitter from MediaPipe.
- Serial communication between Python and ESP8266 initially caused data synchronization problems, including partial or broken string reads, requiring inclusion of acknowledgement ("OK") and buffer clearing.
- Ripple-carry delay in the first 74LS83 resulted in unstable intermediate output states, leading to flickering display output until proper wiring and timing stabilization was performed.
- Noise introduced through breadboard wiring and jumper lengths caused bit-level fluctuations in adder inputs, which required improved grounding, shorter routing and tighter physical wiring.
- Debugging BCD correction was challenging because the CC signal influenced the second adder output timing; tracing signals with multimeter/logic timing observation was necessary.
- Connection of special pins on 7448 (LT, BI, RBI) was necessary to be proper as any inappropriate connection can result in wrongly displayed outputs.
- Multisim simulation produced ideal values, but real hardware propagation delay and wiring parasitics resulted in differences that needed correction through physical testing.

Chapter 8

Future Scope

- Multi-digit numbers and cascading BCD arithmetic
- Wireless transfer and PCB fabrication
- Replace OCR with CNN MNIST handwritten recognizers
- 4-digit multiplexed display driving
- Integration of multiple arithmetic operations

Chapter 9

Source Code

The full project repository is provided at:
GitHub: `https://github.com/vedantmalkar/OCRina`

Chapter 10

Conclusion

This project successfully integrates computer vision input, OCR recognition, embedded interfacing, and hardware-level BCD arithmetic. The implementation demonstrates practical understanding of binary adders, ripple carry propagation, BCD correction, and 7-segment decoding and display. The combination of Python recognition and hardware computation proves the potential of hybrid computing for real digital logic system design.