VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering (SENSE)

PROJECT REPORT

COURSE CODE / TITLE	BECE204L– Microprocessors and Microcontrollers	
PROGRAM / YEAR/ SEM	B.Tech (BEC)/II Year/ WINTER 2023-2024	
DATE OF SUBMISSION	26/04/2024	
TEAM MEMBERS DETAILS	**REGISTER NO.**	**NAME**
	22BEC1392	Trivickram
	22BEC1483	Manikanta
	22BEC1395	Rahul krishna
	22BEC1226	Satwik
	22BEC1434	Mallikarjuna
	22BEC1372	Seshank
	22BEC1477	Vamsi krishna
PROJECT TITLE	RFID BASED PASSPORT VERIFICATION SYSTEM USING 8051 MICROCONTROLLER	

COURSE HANDLING FACULTY	**Dr. S. REVATHI Associate Professor, SENSE**	**REMARKS**	
FACULTY SIGNATURE			

OBJECTIVE:

Design and implement an RFID-based passport verification system that securely authenticates passport holders. The system reads passport details from an RFID tag, compares them with stored data in the 8051micro controller, and displays either a confirmation or denial message on an LCD screen.

INTRODUCTION:

Radio Frequency Identification (RFID) uses radio frequency to read information stored in a RFID card or tag. In this project we are going to Interface EM-18 RFID reader with 8051 microcontroller and display RFID card number on 16*2 LCD display.

The RFID-based passport verification system aims to enhance passport security and streamline the verification process. Existing manual methods are often time-consuming and prone to human error. Our motivation is to create an automated solution that ensures accurate and efficient passport verification during travel and border crossings.

Existing Scenario:

- Currently, passport verification relies on manual checks by immigration officers.
- Paper-based passport booklets are susceptible to forgery and tampering.
- The process lacks real-time access control and can lead to delays.

Our Approach:

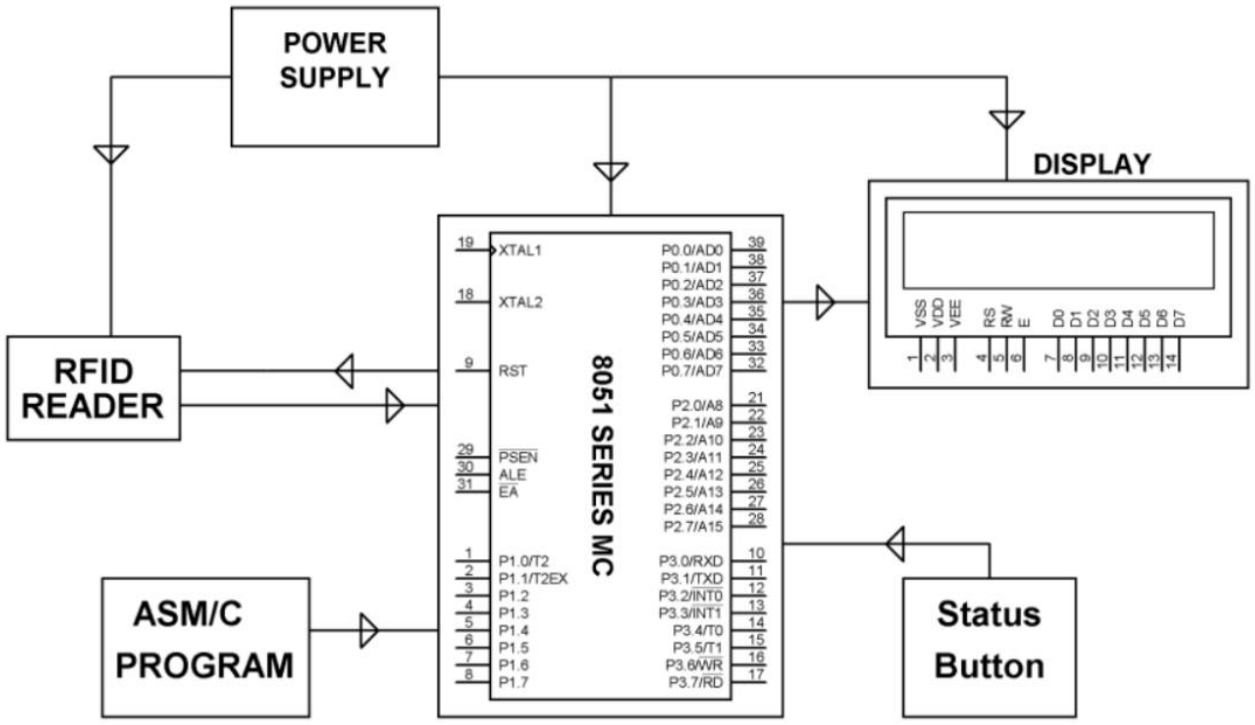
- We propose an RFID-based system that integrates with e-passports (electronic passports).
- Each passport holder receives an RFID tag containing passport details (name, number, nationality, etc.).
- When swiped over an RFID reader, the system retrieves the information.
- An 8051 microcontroller compares the data with stored records.
- If the details match, a confirmation message is displayed; otherwise, a denial message appears on an LCD screen.

Key Differentiators:

1. **Automation**: Our system automates the verification process, reducing reliance on manual checks.
2. **Security**: RFID technology enhances security, preventing unauthorized access and minimizing forgery risks.
3. **Efficiency**: Real-time access control ensures swift verification, improving overall travel experience.

By combining RFID technology with the 8051 microcontroller, our project aims to revolutionize passport verification, making it more secure, efficient, and reliable

BLOCK DIAGRAM:



The block diagram of the **RFID-Based Passport Verification System using 8051 Microcontroller** depicts the interconnections among various components. Let’s delve into each connection:

1. **Power Supply**:

- Provides regulated power to all system components.
- Ensures stable operation.
- Connects to the microcontroller, RFID reader, and display.

2. **RFID Reader**:
    - Communicates wirelessly with RFID tags embedded in passports.
    - Receives power from the power supply.
    - Data lines connect to the microcontroller for data exchange.
3. **8051 Microcontroller**:
    - Central processing unit (CPU) that controls the system.
    - Receives power from the power supply.
    - Data lines connect to the RFID reader, display, and status button.
    - Control lines manage system behavior (e.g., reset, enable).
4. **ASM/C Program**:
    - Stored in the microcontroller's memory.
    - Executes passport verification logic.
    - Interacts with the RFID reader and database.
5. **Display (LCD)**:
    - Receives power from the power supply.
    - Data lines connect to the microcontroller for displaying passport status.
    - Provides visual feedback to the operator.
6. **Status Button**:
    - Receives power from the power supply.
    - Connects to the microcontroller.
    - Allows manual interaction (e.g., override, reset).

COMPONENTS/ SOFTWARE REQUIRED:

8051 Microcontroller:

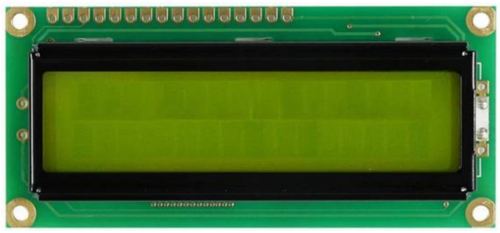
8051microcontroller is a 8 bit microcontroller which has 128 bytes of on chip RAM, 4K bytes of on chip ROM, two timers, one serial port and four 8bit ports. 8052 microcontroller is an extension of microcontroller. The table below shows the comparison of 8051 family members.

EM-18 RFID:

EM-18 RFID reader operates at 125 KHz and it comes with an on-chip antenna and it can be powered with 5V power supply. It provides serial output along with weigand output.

16*2 LCD DISPLAY:

16*2 lcd is connected to 8051 micro controller. RS pin of 16*2 lcd is connected to P3.7, RW pin of 16*2 lcd is connected to P3.6 and E pin of 16*2 lcd is connected to P3.5. Data pins are connected to port 1 of 8051 microcontroller.



RFID TAGS:

RFID (Radio Frequency Identification) tags work by using radio frequency signals to transmit data between a tag (also called a transponder) and an RFID reader.

- Potentiometer And Jumper wires.

PROJECT DESCRIPTION:

1. **Initialization**:
    - The system powers up, initializing the microcontroller and peripherals.
    - The RFID reader is ready to detect RFID tags.
2. **Passport Presentation**:
    - A passport holder places their passport (with an embedded RFID tag) near the RFID reader.
    - The RFID reader detects the tag and establishes communication.
3. **Data Retrieval**:
    - The RFID reader reads the data stored in the passport’s RFID tag.
    - The data includes passport details such as name, passport number, nationality, and other relevant information.
4. **Microcontroller Processing**:
    - The 8051 microcontroller receives the data from the RFID reader.
    - It extracts the relevant information (e.g., passport number) for verification.
5. **Data Comparison**:
    - The microcontroller compares the received passport details with stored records.
    - The stored records can be in an internal memory or an external database.

- If the data matches, the passport is considered valid.

6. **Decision Making**:
   - If the passport details match, the system displays a confirmation message on the connected LCD display.
   - If the data does not match (e.g., forged passport or incorrect details), the system displays a denial message.
   - The decision is based on the comparison results.
7. **Visual Feedback**:
   - LED lights may indicate the verification status (e.g., green for valid, red for invalid).
   - The LCD display shows relevant messages (e.g., "Passport Verified" or "Invalid Passport").
8. **Repeat Process**:
   - The system waits for the next passport presentation.
   - The entire process repeats for each verification request.

SIMULATION

1. **Creating a New Project:**
   - Open Proteus.
   - Create a new project by clicking on File > New Project.
   - Save the project with an appropriate name.
2. **Adding Components to the Schematic:**
   - Click on the Component Mode button (looks like a resistor) from the left toolbar.
   - Select components from the libraries:
     - 8051 Microcontroller (e.g., AT89C51)
     - RFID Reader/Writer Module
     - LCD Display (16x2 or 20x4)
     - Resistors, Capacitors, Diodes
     - Voltage Regulator (e.g., LM7805)
   - Place the components on the workspace by left-clicking**.**
3. **Wiring Up the Circuit:**
   - Connect the components using wires.
   - Click on the Wire Mode button (looks like a zigzag line) from the toolbar.
   - Left-click to start a wire and connect the pins of components.
4. **Setting Component Properties:**

- Double-click on each component to open its properties.
- Set parameters such as resistance values, capacitor values, etc.

5. **Microcontroller Programming:**
   - Write the 8051 microcontroller program in embedded C (using Keil).
   - Configure the microcontroller pins for interfacing with the RFID reader, LCD, LED lights, and other peripherals.
   - Implement the logic for comparing RFID data and displaying messages on the LCD.
6. **Simulation Setup:**
   - Click on the Play Button (looks like a triangle) on the bottom left to start the simulation.
   - Configure simulation settings (e.g., time duration, step size).
7. **Running the Simulation:**
   - The simulation will run, and you can observe the behavior of the circuit.

CONCEPT LEARNED:

1. **RFID Technology**:
   - Understanding how RFID (Radio Frequency Identification) works.
   - Interfacing RFID readers with microcontrollers.
   - Reading data from RFID tags (e.g., embedded in passports).
2. **Microcontroller Interfacing**:
   - Connecting external devices (e.g., RFID reader, LCD) to the 8051 microcontroller.
   - Configuring microcontroller pins for input/output.
3. **LCD Interfacing:**

   RS (Register Select) pin to a digital pin on the microcontroller

   R/W (Read/Write) pin to ground (for write-only operation)

   E (Enable) pin to a digital pin on the microcontroller

   Data pins (D0-D7) to digital pins on the microcontroller

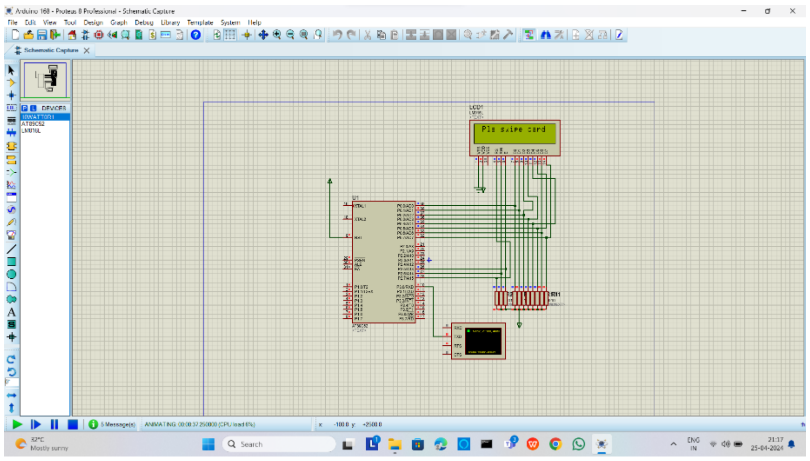
   V0 (Contrast) pin to the potentiometer

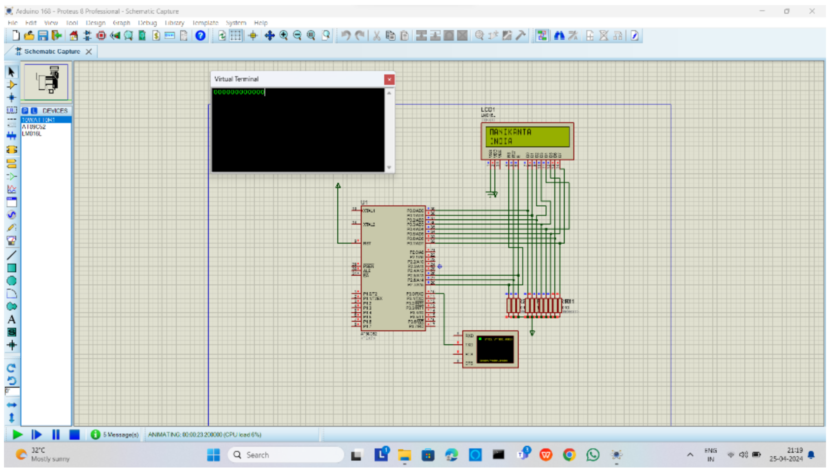
   VCC and GND pins to the power supply
4. **Data Processing and Comparison**:
   - Extracting relevant information from RFID data.
   - Comparing received data with stored records.

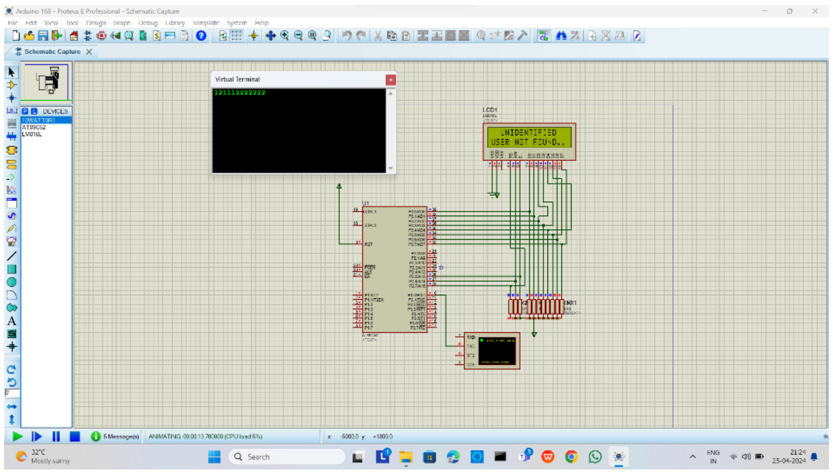
- Decision-making based on data matching.

5. **Simulation and Prototyping**:
    - Simulating the system behavior using tools like Proteus.
    - Translating the schematic into a physical Proto type.

6.Simulation and results:





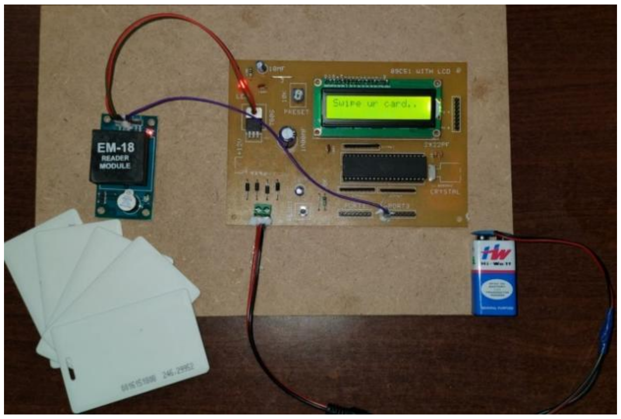


```
121112222222
```

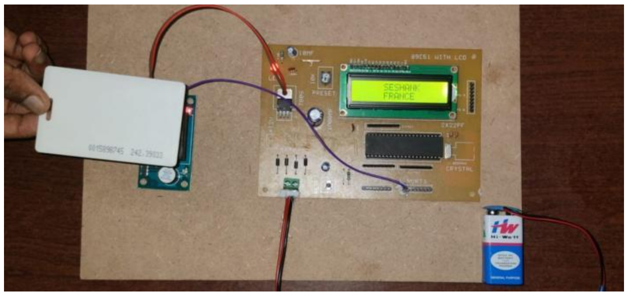
```
UNIDENTIFIED
USER NOT FOUND..
```

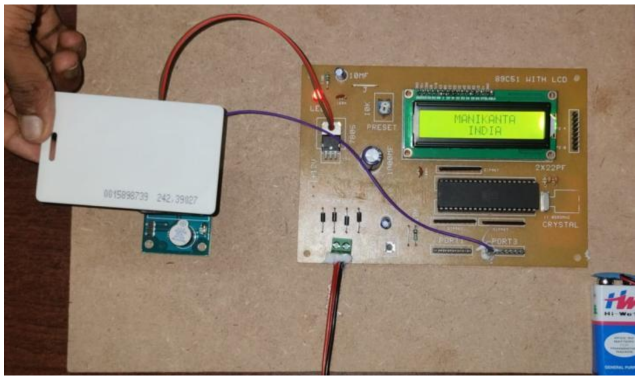
IMPLEMENTATION:

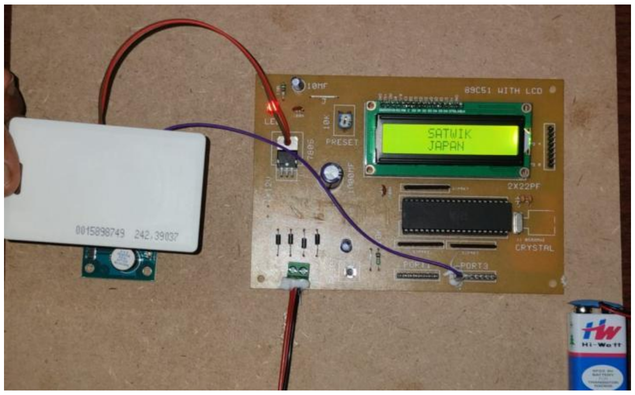
1. Asking for swipe the rfid tag.



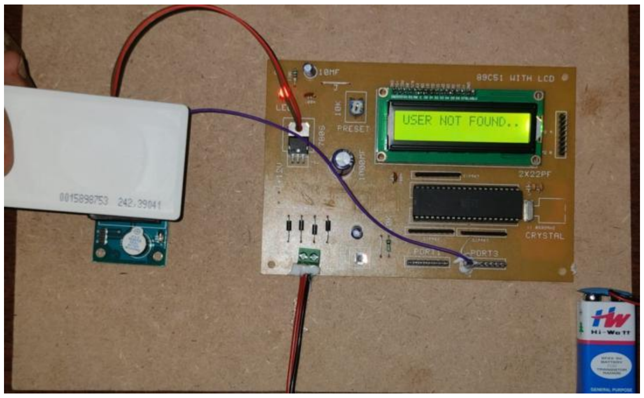
2. displaying information the rfid tag.







3. Display the user not found.



Youtube link:

https://youtu.be/hMiJ4grABlE?si=hGYyuiTm8o5KktKP

CHALLENGES FACED:

1. **RFID Tag Compatibility**:
   - Dealing with different RFID tag standards (e.g., ISO 14443, ISO 15693).
   - Ensuring that the chosen RFID tags are compatible with the reader and microcontroller.
2. **Microcontroller Programming**:
   - Writing efficient and error-free code for the 8051 microcontroller.
   - Handling interrupts, timers, and I/O ports correctly.
3. **Data Parsing and Verification**:
   - Extracting relevant information from the RFID data (e.g., passport number, nationality).

- Implementing logic to compare the received data with stored records.

4. **LCD Display Integration**:
    - Properly interfacing the LCD display with the microcontroller.
    - Formatting and displaying messages clearly on the limited screen space.
5. **Power Supply Stability**:
    - Ensuring stable power supply to all components (5v).
    - Dealing with voltage fluctuations or noise that could affect system behavior.
6. **Testing and Debugging**:
    - Rigorous testing to verify correct functionality.
    - Debugging issues related to communication, data mismatch, or unexpected behavior.

APPLICATIONS:

The RFID-based passport verification system using an 8051 microcontroller has broader applications beyond passport control.

1. **Access Control Systems**:
    - The same system can be adapted for secure access control in buildings, offices, or restricted areas.
    - Replace passport data with employee IDs or access cards.
    - Enhance security by verifying authorized personnel.
2. **Library Management**:
    - Replace passports with library cards or student IDs.
    - Automate book borrowing and return processes.
    - Ensure accurate tracking of borrowed items.
3. **Inventory Management**:
    - Use RFID tags for tracking inventory items (e.g., retail products, warehouse stock).
    - Streamline inventory checks and reduce manual effort.
4. **Vehicle Identification and Toll Collection**:
    - Integrate RFID tags into vehicle registration.

- Enable automatic toll collection at highways or parking lots.
- Enhance traffic flow and reduce congestion.

5. **Hospital Patient Management**:
    - Replace passports with patient IDs.
    - Automate patient check-in, appointment scheduling, and record keeping.
    - Improve hospital workflow and patient experience.
6. **Event Management and Ticketing**:
    - Issue RFID wristbands or cards for event access.
    - Validate tickets, control entry, and manage crowd flow.
    - Enhance event security and prevent counterfeiting.
7. **Smart Agriculture**:
    - Attach RFID tags to livestock or farm equipment.
    - Monitor animal health, track equipment usage, and manage inventory.
    - Optimize farming processes.

CONCLUSION:

The RFID-based passport verification system developed using an 8051 microcontroller and assembly language programming provides a secure and efficient solution for authenticating passports. The Proteus simulation demonstrates the effective integration of RFID technology, microcontroller-based processing, and assembly language programming to create a reliable and optimized passport verification system. This project showcases the potential of low-level control and customized programming to enhance security and efficiency in border control and immigration management applications.

APPENDIX:

Complete program with comments

ORG 0000H	; Set the origin to address 0000H
SJMP 0030H	; Short jump to address 0030H
ORG 0030H	; Set the origin to address 0030H

MOV TMOD,# 20H	; Load 20H into TMOD register (Timer 1 in Mode 2 - 8-bit auto-reload)
MOV TH1,#0FDH	; Load 0FDH into TH1 (Timer 1 high byte)
MOV SCON,#50H	; Load 50H into SCON (Serial mode 1, receive enabled)
SETB TR1	; Start Timer 1
MOV A, #38H	; Load 38H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV A, #0EH	; Load 0EH into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV A, #01H	; Load 01H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV A, #080H	; Load 080H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#SIN	; Load address of SIN into data pointer
ACALL String	; Call subroutine String
Again:	; Label for loop start
MOV R5, #12D	; Load 12D into register R5
MOV R0, #20H	; Load 20H into register R0
rfi:	; Label for inner loop start

Monitor: JNB RI, Monitor	; Jump to Monitor if RI (Receive Interrupt Flag) is not set
ACALL Serial	; Call subroutine Serial
INC R0	; Increment register R0
DEC R5	; Decrement register R5
CJNE R5, #00H, rfi	; Compare R5 with 00H, if not equal, jump to label rfi
MOV R0, #20H	; Load 20H into register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0

INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
CJNE @R0,#"0",n0	; Compare indirect @R0 with "0", if not equal, jump to label n0
INC R0	; Increment register R0
MOV A, #080H	; Load 080H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St1	; Load address of St1 into data pointer
ACALL String	; Call subroutine String
MOV A, #0C0H	; Load 0C0H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St2	; Load address of St2 into data pointer
ACALL String	; Call subroutine String
JMP Again	; Jump to label Again
n0:	; Label for loop start
MOV R0, #20H	; Load 20H into register R0

CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1

INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
CJNE @R0,#"1",n1	; Compare indirect @R0 with "1", if not equal, jump to label n1
INC R0	; Increment register R0
MOV A, #080H	; Load 080H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St3	; Load address of St3 into data pointer
ACALL String	; Call subroutine String
MOV A, #0C0H	; Load 0C0H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St4	; Load address of St4 into data pointer
ACALL String	; Call subroutine String
JMP Again	; Jump to label Again
n1:	; Label for loop start
MOV R0, #20H	; Load 20H into register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0

CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
CJNE @R0,#"2",n2	; Compare indirect @R0 with "2", if not equal, jump to label n2
INC R0	; Increment register R0
MOV A, #080H	; Load 080H into accumulator

ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St5	; Load address of St5 into data pointer
ACALL String	; Call subroutine String
MOV A, #0C0H	; Load 0C0H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St6	; Load address of St6 into data pointer
ACALL String	; Call subroutine String
JMP Again	; Jump to label Again
n2:	; Label for loop start
MOV R0, #20H	; Load 20H into register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0

CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
CJNE @R0,#"3",n3	; Compare indirect @R0 with "3", if not equal, jump to label n3
INC R0	; Increment register R0
MOV A, #080H	; Load 080H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St7	; Load address of St7 into data pointer
ACALL String	; Call subroutine String
MOV A, #0C0H	; Load 0C0H into accumulator

ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#St8	; Load address of St8 into data pointer
ACALL String	; Call subroutine String
JMP Again	; Jump to label Again
n3:	; Label for loop end
MOV A, #080H	; Load 080H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#StD	; Load address of StD into data pointer
ACALL String	; Call subroutine String
MOV A, #0C0H	; Load 0C0H into accumulator
ACALL Command	; Call subroutine Command
ACALL Delay	; Call subroutine Delay
MOV DPTR,#StE	; Load address of StE into data pointer
ACALL String	; Call subroutine String
JMP Again	; Jump to label Again
Command:	; Subroutine Command
CLR P2.7	; Clear bit 7 of port 2
CLR P2.6	; Clear bit 6 of port 2
SETB P2.5	; Set bit 5 of port 2
MOV P0, A	; Move accumulator content to port 0
ACALL Delay	; Call subroutine Delay
CLR P2.5	; Clear bit 5 of port 2
RET	; Return from subroutine

Datas:	; Subroutine Datas
SETB P2.7	; Set bit 7 of port 2
CLR P2.6	; Clear bit 6 of port 2
SETB P2.5	; Set bit 5 of port 2
MOV P0, A	; Move accumulator content to port 0
ACALL Delay	; Call subroutine Delay
CLR P2.5	; Clear bit 5 of port 2
RET	; Return from subroutine
Delay:	; Subroutine Delay
MOV R6, #0FFH	; Load 0FFH into register R6
Her1:	; Label for outer loop start
MOV R7, #0AH	; Load 0AH into register R7
Here:	; Label for inner loop start
DJNZ R7, Here	; Decrement R7 and jump to Here if R7 is not zero
DJNZ R6, Her1	; Decrement R6 and jump to Her1 if R6 is not zero
RET	; Return from subroutine
String:	; Subroutine String
L1:	; Label for loop start
CLR A	; Clear accumulator
MOVC A,@A+DPTR	; Move code byte relative to DPTR and accumulator into accumulator
JZ Finish	; Jump to Finish if accumulator is zero
ACALL Datas	; Call subroutine Datas
ACALL Delay	; Call subroutine Delay

INC DPTR	; Increment data pointer
SJMP L1	; Short jump to L1
Finish:	; Label for loop end
RET	; Return from subroutine
Serial:	; Subroutine Serial
CLR RI	; Clear receive interrupt flag
MOV A, SBUF	; Move serial buffer into accumulator
MOV @R0, A	; Move accumulator into indirect @R0
RET	; Return from subroutine
SIN:DB ' Pls Swipe Card ',0	; Define byte array SIN with string ' Pls Swipe Card ' and null terminator
St1:DB 'Manikanta ',0	; Define byte array St1 with string 'Manikanta ' and null terminator
St2:DB 'India ',0	; Define byte array St2 with string 'India ' and null terminator
St3:DB 'Vamsi ',0	; Define byte array St3 with string 'Vamsi ' and null terminator
St4:DB 'Germany ',0	; Define byte array St4 with string 'Germany ' and null terminator
St5:DB 'Rahul ',0	; Define byte array St5 with string 'Rahul ' and null terminator
St6:DB 'Canada ',0	; Define byte array St6 with string 'Canada ' and null terminator
St7:DB 'Satwik ',0	; Define byte array St7 with string 'Satwik ' and null terminator
St8:DB 'Japan ',0	; Define byte array St8 with string 'Japan ' and null terminator
St9:DB 'Trivickram ',0	; Define byte array St9 with string 'Trivickram ' and null terminator

StA:DB 'USA ',0	; Define byte array StA with string 'USA ' and null terminator
StB:DB 'Seshank ',0	; Define byte array StB with string 'Seshank ' and null terminator
StC:DB 'France ',0	; Define byte array StC with string 'France ' and null terminator
StD:DB ' Unidentified ',0	; Define byte array StD with string ' Unidentified ' and null terminator
StE:DB 'User Not Found..',0	; Define byte array StE with string 'User Not Found..' and null terminator
END	; End of the assembly program

Photo of hardware: