HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF INTERNATIONAL EDUCATION

FINAL PROJECT

**DESIGNING A PERIPHERAL COMMUNICATION SYSTEM USING EXPANDING EXTERNAL MEMORY ATMEGA 128**

**MAJOR OF COMPUTER ENGINGEETING TECHNOLOGY**

Students: **NGUYỄN THANH HUY**

ID: 20119115

HO CHI MINH CITY – 05/24

HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF INTERNATIONAL EDUCATION

FINAL PROJECT

**DESIGNING A PERIPHERAL COMMUNICATION SYSTEM USING EXPANDING EXTERNAL MEMORY ATMEGA 128**

**MAJOR OF COMPUTER ENGINGEETING TECHNOLOGY**

Students: **NGUYỄN THANH HUY**

ID: 20119115

Supervisor**: TRƯƠNG NGỌC SƠN (PhD/ Assoc. Prof.)**

HO CHI MINH CITY – 05/24

**ACKNOWLEDGMENTS**

From the bottom of my heart, I would like to express my attitude to my supervisor PhD/ Assoc. Prof. Truong Ngoc Son, who had wonderful guidance in our class in the right way to write a technical document of the technical project in the last 15 weeks. His dedication in giving us precious knowledge is the best thing we have appreciated all the time we spend together.

**ABSTRACT**

The rapid evolution of embedded systems in automation and control necessitates the continuous development of sophisticated and multifunctional platforms. This project addresses the growing demand for versatile embedded systems capable of managing multiple devices with real-time user interaction. The core challenge lies in designing a reliable and user-friendly system that integrates diverse hardware components such as LEDs, displays, relays, and motors, using the ATMEGA128 microcontroller.

To solve this problem, a comprehensive embedded system was designed and simulated, featuring a robust menu-driven interface implemented on a 20x4 LCD display. The system utilizes linked lists for menu navigation, enabling dynamic user interaction with various hardware components. The design approach emphasizes modularity and scalability, allowing for future expansions and modifications. This project significantly departs from traditional static control systems by offering a flexible, user-configurable platform facilitated through a real-time graphical user interface.

Key results of the project include successful hardware integration and seamless software execution, demonstrating the system's capability to perform multiple tasks simultaneously, such as real-time clock adjustments, motor direction control, and LED pattern manipulation, all navigable through an intuitive LCD-based menu system. The system not only meets its design objectives but also provides a foundation for further research into more integrated and autonomous embedded systems for industrial applications. This work showcases significant advancements over previous efforts by enhancing user interaction, system integration, and providing a scalable solution for complex control tasks in embedded systems.

**TABLE OF CONTENTS**

[Chapter 1 oVERVIEW 1](#_Toc166634278)

[This chapter presents an overview of the embedded system designed as part of the course requirements for Embedded System Design. It serves as the foundation for the detailed exploration that follows in subsequent chapters. The embedded system revolves around the use of the ATMEGA128 microcontroller to control a variety of peripheral devices through a user-friendly interface. This project not only demonstrates the practical application of embedded system theories but also introduces innovations in device control and user interaction. 1](#_Toc166634279)

[1.1 INTRODUCTION 1](#_Toc166634280)

[1.2 OBJECTIVES 3](#_Toc166634281)

[1.3 SCOPE 4](#_Toc166634282)

[1.4 OUTLINE 5](#_Toc166634283)

[Chapter 2 background 7](#_Toc166634284)

[2.1 Software Architecture and Techniques 7](#_Toc166634285)

[2.2 Major Hardware Components 8](#_Toc166634286)

[2.3 Supporting Technologies 9](#_Toc166634287)

[Chapter 3 design and implementation 10](#_Toc166634288)

[3.1 SYSTEM REQUIREMENTS 10](#_Toc166634289)

[3.2 BLOCK DIAGRAM 11](#_Toc166634290)

[3.3 HARDWARE DESIGNS 13](#_Toc166634291)

[3.4 SOFTWARE DESIGNS 16](#_Toc166634292)

[Chapter 4 CONCLUSIONS 44](#_Toc166634293)

# oVERVIEW

This chapter presents an overview of the embedded system designed as part of the course requirements for Embedded System Design. It serves as the foundation for the detailed exploration that follows in subsequent chapters. The embedded system revolves around the use of the ATMEGA128 microcontroller to control a variety of peripheral devices through a user-friendly interface. This project not only demonstrates the practical application of embedded system theories but also introduces innovations in device control and user interaction.

## INTRODUCTION

Embedded systems are specialized computing systems that combine hardware and software to perform dedicated functions within larger systems. These systems are ubiquitous in modern technology, found in devices ranging from simple digital watches and home appliances to complex automotive systems and industrial machinery. The design of embedded systems involves intricate planning and integration of hardware and software components to ensure functionality, reliability, and efficiency.

The evolution of embedded systems has been marked by significant advancements in microcontroller technology, miniaturization, and integration capabilities. Key studies and developments in this field have focused on improving computational power, reducing power consumption, and enhancing real-time operational capabilities. Literature reveals a trend towards the development of more user-friendly interfaces and increased connectivity, such as Internet of Things (IoT) integration. However, despite considerable advancements, many systems remain limited by rigid software architectures and constrained user interaction models.

One of the persistent challenges in embedded systems design is achieving a balance between complexity and user-friendliness. Specifically, many systems do not provide dynamic interaction capabilities, making it difficult for users to modify system behavior or settings without comprehensive technical knowledge. Additionally, integrating a diverse set of peripheral devices often complicates the design and can reduce system stability and reliability.

This project proposes the design and implementation of a versatile, menu-driven embedded system using the ATMEGA128 microcontroller. The system aims to control multiple devices, including LEDs, LCDs, relays, and motors, through a cohesive user interface. The approach involves:

* Developing a modular software architecture that allows for easy updates and modifications.
* Utilizing a linked list to create a dynamic and navigable menu system on the LCD.
* Integrating real-time controls and adjustments through user input via pushbuttons.

Compared to existing works, this project introduces several enhancements:

* User Interface: A significant focus on user experience with a graphical menu system that simplifies interaction, making the system accessible to users with limited technical expertise.
* Modularity: The software and hardware design emphasize modularity, allowing for easy expansion or modification of the system's capabilities without extensive redesign.
* Integration of Peripheral Devices: The project demonstrates a robust method for integrating and managing multiple types of hardware devices, showcasing a scalable model suitable for various applications.

The anticipated outcomes of this project include a fully functional embedded system that demonstrates advanced control and interaction capabilities. The system is expected to showcase:

* Effective management of diverse hardware components through a single control unit.
* Enhanced user interaction with the system through a real-time, menu-driven interface.

Demonstrated improvements in system flexibility and user accessibility

## OBJECTIVES

The primary objective of this project is to design and implement an embedded system using the ATMEGA128 microcontroller that enhances user interaction and device control capabilities. This objective is further delineated into several specific goals, each addressing distinct facets of the project:

**1.2.1 Develop an Interactive User Interface**

Objective: Design and implement a menu-driven interface on a 20x4 LCD display, using a linked list for navigation. This interface aims to simplify user interaction, allowing for real-time control and configuration adjustments.

Action: Implement graphical menu elements that enable users to control peripheral devices such as LEDs, relays, and motors, and view real-time system statuses.

**1.2.2 Integrate Multiple Peripheral Devices**

Objective: Achieve seamless integration of various hardware components including 64 single LEDs, four seven-segment displays, LCD, two relays, and two DC motors to demonstrate comprehensive system control.

Action: Design a robust hardware setup that allows each device to function in concert with others, ensuring stability and efficiency in operations.

**1.2.3. Enhance System Modularity and Scalability**

Objective: Create a modular system architecture that facilitates future expansions or modifications without extensive redesigns.

Action: Structure the software and hardware design to allow easy addition or removal of components and functionalities, supporting scalability and flexibility.

**1.2.4. Optimize Real-Time System Performance**

Objective: Ensure that the system responds swiftly and reliably to user inputs and operates effectively under real-time constraints.

Action: Implement efficient code and use appropriate microcontroller programming techniques to minimize response times and maximize reliability during operation.

**1.2.5. Demonstrate System Usability and Effectiveness**

Objective: Validate the system design through comprehensive testing to confirm that it meets all specified requirements and is user-friendly.

Action: Conduct systematic testing phases to assess the functionality, user interface, and integration of the embedded system, making adjustments based on feedback to optimize performance and usability.

## SCOPE

The scope of this project outlines the boundaries and parameters within which the embedded system design and implementation are conducted. It defines what the project will focus on, the conditions under which the system operates, and areas that are outside the purview of this study.

**Parameters and Conditions**

* Microcontroller Utilization: The project is built around the ATMEGA128 microcontroller, chosen for its versatility and adequate resource availability, including memory and I/O ports suitable for handling multiple peripheral devices.
* Periheral Devices: The system integrates specific peripheral components:
  + LEDs: 64 single LEDs and 4 seven-segment displays.
  + LCD Display: A 20x4 character LCD for user interface display.
  + Relays and Motors: 2 relays and 2 DC motors, showcasing the system's capability to handle mechanical actions.
  + User Input: 3 push buttons (up, down, enter) for navigating the menu and altering system settings.
* Software Environment: Development will be conducted using C programming language, appropriate for low-level hardware interaction and performance efficiency on the chosen microcontroller.
* Operational Conditions: The system is designed to operate under standard room temperatures and power conditions typical to laboratory environments. It is assumed that the system will have a stable power supply and operate free from extreme environmental conditions.

## OUTLINE

This outline provides a roadmap of the report's structure, detailing the content of each chapter to guide readers through the project's documentation.

**Chapter 1. Overview**

This chapter introduces the project, defining the objectives, scope, and the structure of the report. It provides the foundational context for the embedded system design and highlights the key elements that will be addressed throughout the report.

**Chapter 2. Background**

This chapter presents a thorough review of the relevant literature and historical context of embedded systems. It discusses the evolution of microcontroller technology, previous works in the field, and identifies the gaps that this project aims to fill.

**Chapter 3. Design and Implementation**

Detailed description of the system design and implementation process. This chapter breaks down the hardware configurations, software development, and the integration of the two. It explains the rationale behind design decisions and how they contribute to overall system functionality.

**Chapter 4. Conclusions**

Summarizes the achievements of the project and assesses its impact on the field of embedded systems. This chapter discusses the lessons learned, suggests areas for further research, and outlines potential improvements for future versions of the system.

# background

This chapter delves into the technical foundations and components crucial to the embedded system developed in this project. It covers the methodologies, software architectures, and major hardware components that are integral to the design and functionality of the system. This section aims to provide a clear understanding of the technological choices and their relevance to the objectives of the project.

## Software Architecture and Techniques

The software driving the embedded system is structured around a modular architecture, which allows for flexibility in updates and scalability in integrating additional functionalities. The key techniques and methods employed in the software development include:

Modular Programming: The software is divided into distinct modules corresponding to each hardware component and functionality. This approach simplifies debugging, enhances maintainability, and eases the process of adding new features.

Interrupt-Driven Programming: Utilizing the microcontroller's interrupt capabilities ensures that the system can respond immediately to user inputs or changes in peripheral devices without unnecessary polling.

State Machine Implementation: The system utilizes a state machine for menu navigation, ensuring a responsive and intuitive interface experience for the user.

## Major Hardware Components

Each hardware component selected for this project plays a crucial role in the system's functionality. Below are the key components and their specific contributions to the project.

**2.2.1 ATMEGA128 Microcontroller**

- Core Features: The ATMEGA128 is central to the project, selected for its 128 KB of flash memory, 4 KB of SRAM, and 4 KB of EEPROM. These specifications support the complex control algorithms required for managing multiple devices.

- I/O Capabilities: It provides extensive I/O options with 53 digital I/O pins, 8 of which are used for analog inputs, making it versatile for interfacing with various sensors and actuators.

- Clock Speed: Operating up to 16 MHz, it offers fast processing capabilities that are essential for real-time control and interfacing with high-speed peripherals.

**2.2.2 DC Motors**

- Operational Parameters: Two DC motors are used, each capable of delivering the necessary torque and speed for demonstrating dynamic mechanical control.

- Integration with Motor Drivers: Motor drivers are implemented to handle the high current requirements of the DC motors, providing precise control over their operational states (forward, reverse, stop).

**2.2.3 Power Supply**

- Specifications: The power supply is designed to provide a stable 5V DC output, critical for the safe and reliable operation of the microcontroller and peripheral devices.

- Regulation Features: It includes over-current and over-voltage protection to ensure the longevity and safety of the system components.

**2.2.4 LCD Display (20x4)**

- User Interface: The LCD display acts as the primary interface for user interaction, displaying menus and system statuses.

- Characteristics: It supports up to 20 characters in 4 rows, providing ample space for clear and effective communication of options and outputs.

**2.2.5 LED Arrays**

- Functionality: 64 individual LEDs are used for status indication and for demonstrating pattern control capabilities.

- Control Mechanisms: These are directly driven by the microcontroller via digital I/O pins, showcasing basic digital output handling.

**2.2.6 Seven-Segment Displays**

- Application: Four seven-segment displays are utilized to show numerical data, such as time or count values, enhancing the system's feedback capabilities.

- Control: Controlled via multiplexing techniques to minimize the use of I/O pins and simplify the wiring complexity.

**2.2.7 Push Buttons**

- Function: Three push buttons (up, down, enter) provide the input interface for navigating the menu system.

- Debouncing: Proper debouncing techniques are implemented in software to ensure reliable input recognition without false triggering.

## Supporting Technologies

The development environment AVR MPLAB IDE and simulation tools (Proteus) play supportive roles by providing the necessary platforms for programming and preliminary testing of circuit designs and logic, ensuring that the system is robust and error-free before deployment.

# design and implementation

This chapter elaborates on the detailed design and implementation of the embedded system project. It covers the systematic approach taken from conceptualization to the physical realization of the system, including hardware setup, software development, and the integration of both components. The design process is informed by the objectives laid out in Chapter 1, with each design decision aimed at optimizing functionality and user interaction.

## SYSTEM REQUIREMENTS

The system requirements are specifically defined to ensure that the project meets all the objectives as outlined in Chapter 1. These requirements are divided into functional and non-functional criteria, ensuring comprehensive coverage of all aspects necessary for a successful implementation.

**3.1.1 Functional Requirements**

- Functional requirements detail what the system must do. They are directly tied to the actions and operations that the embedded system needs to perform:

- Device Control:

* LED Control: Ability to individually manage 64 LEDs with various patterns such as blinking, chasing, and fading.
* Motor Control: Capability to operate two DC motors with controls for direction, speed, and on/off states.
* Relay Operation: Manage two relays for turning on/off connected devices or systems.
* Display Management: Utilize a 20x4 LCD to display system status, menus, and user inputs effectively.
* User Interface:
* Provide a menu-driven interface on the LCD display, navigable through three push buttons (up, down, enter).
* Display real-time status updates and feedback in response to user inputs or system changes.

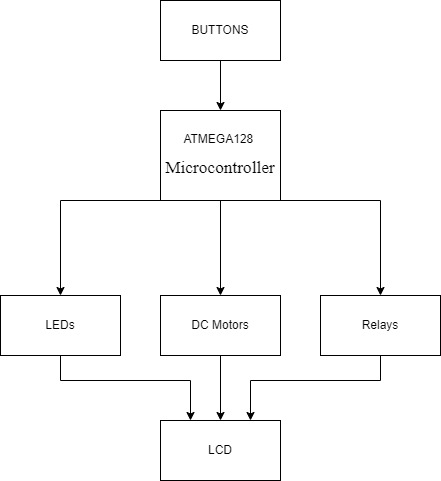
**3.1.2 Non-Functional Requirements**

- Non-functional requirements specify how the system performs certain tasks, focusing on the quality and attributes of the system:

-The system should have minimal boot and operational delays, with real-time processing capabilities to handle concurrent inputs and outputs effectively.

- Optimize resource utilization to ensure that the system operates within the memory and processing constraints of the ATMEGA128 microcontroller.

## BLOCK DIAGRAM

. 

**ATMEGA128 Microcontroller:**

* Central Control Unit: The microcontroller serves as the central processing unit, managing all interactions between the user and the peripheral devices.
* Data and Control Signals: It sends control signals to the peripheral devices and processes input signals from the push buttons.

**64 Individual LEDs:**

* Output Device: These LEDs are used to display various patterns such as blinking, chasing, and fading.
* Connection: Controlled directly by the digital I/O pins of the microcontroller.

**4 Seven-Segment Displays:**

* Numeric Display: Used to show numerical data like time or count values.
* Connection: Managed through multiplexing techniques to minimize the number of required I/O pins.

**20x4 LCD Display:**

* User Interface: Displays menus, system statuses, and user inputs.
* Connection: Interfaced with the microcontroller for real-time updates and feedback.

**2 Relays:**

* Control Switches: Used to turn on/off external devices or systems.
* Connection: Controlled by the microcontroller to manage power supply to connected devices.

**2 DC Motors:**

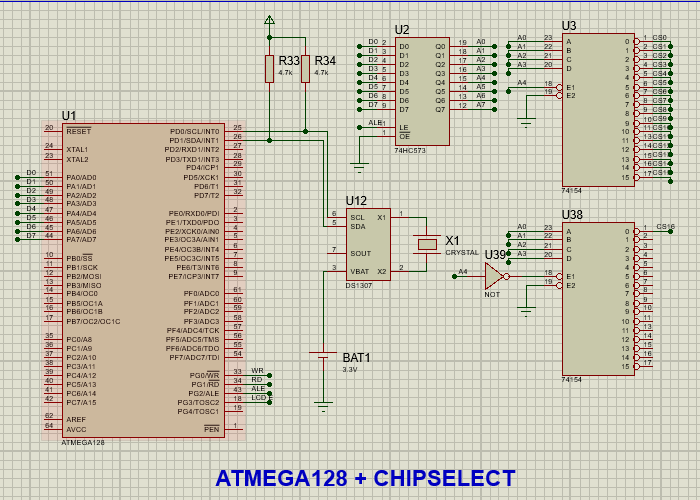
* Mechanical Actuation: Provide physical movement, controlled in terms of direction, speed, and on/off states.
* Connection: Interfaced through motor driver circuits to handle higher current requirements.

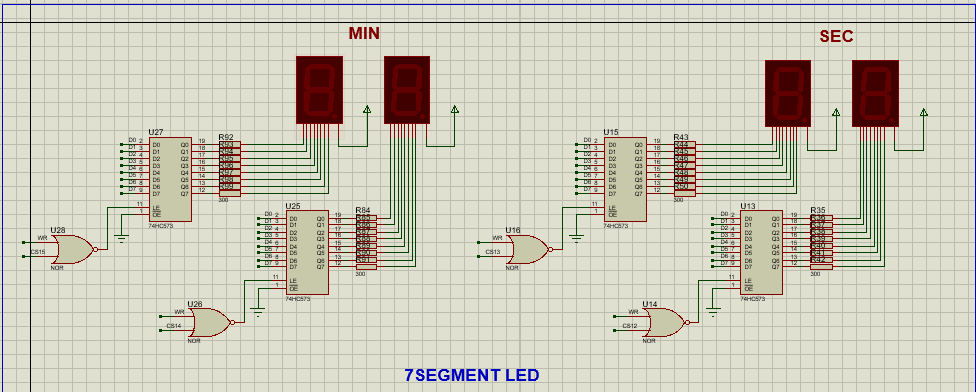
**3 Push Buttons (Up, Down, Enter):**

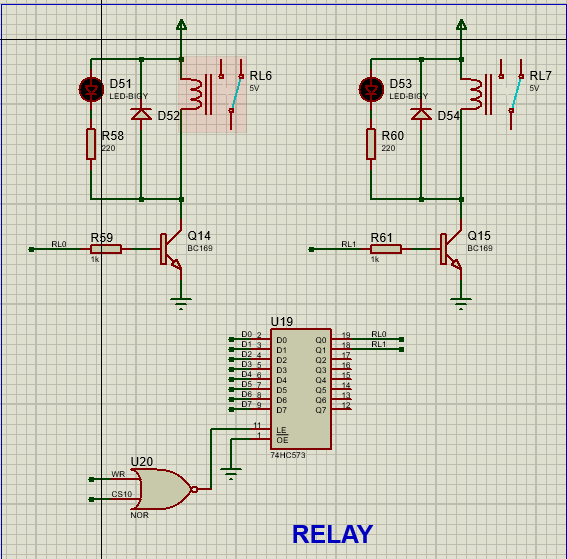
* User Input: Allow the user to navigate the menu and select options.
* Connection: Inputs are debounced and processed by the microcontroller to ensure reliable operation.

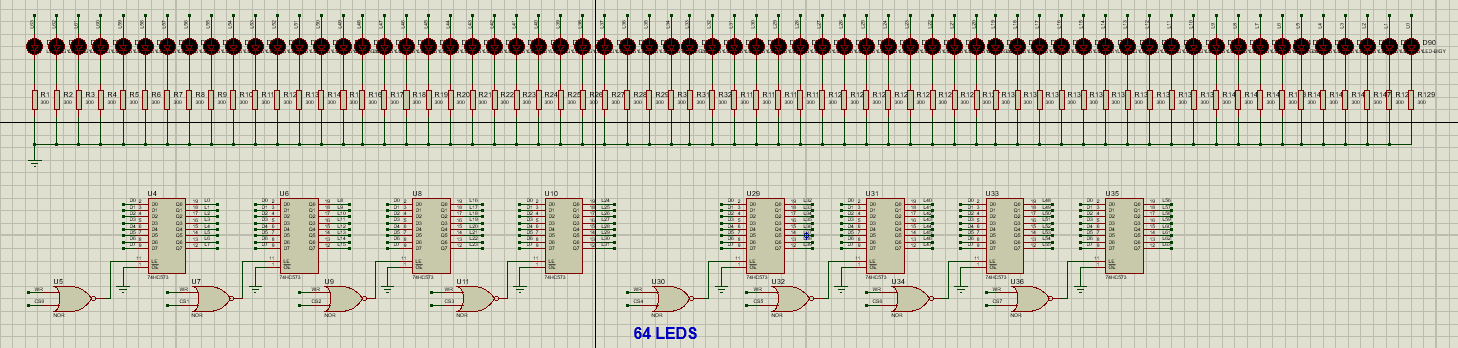
## HARDWARE DESIGNS

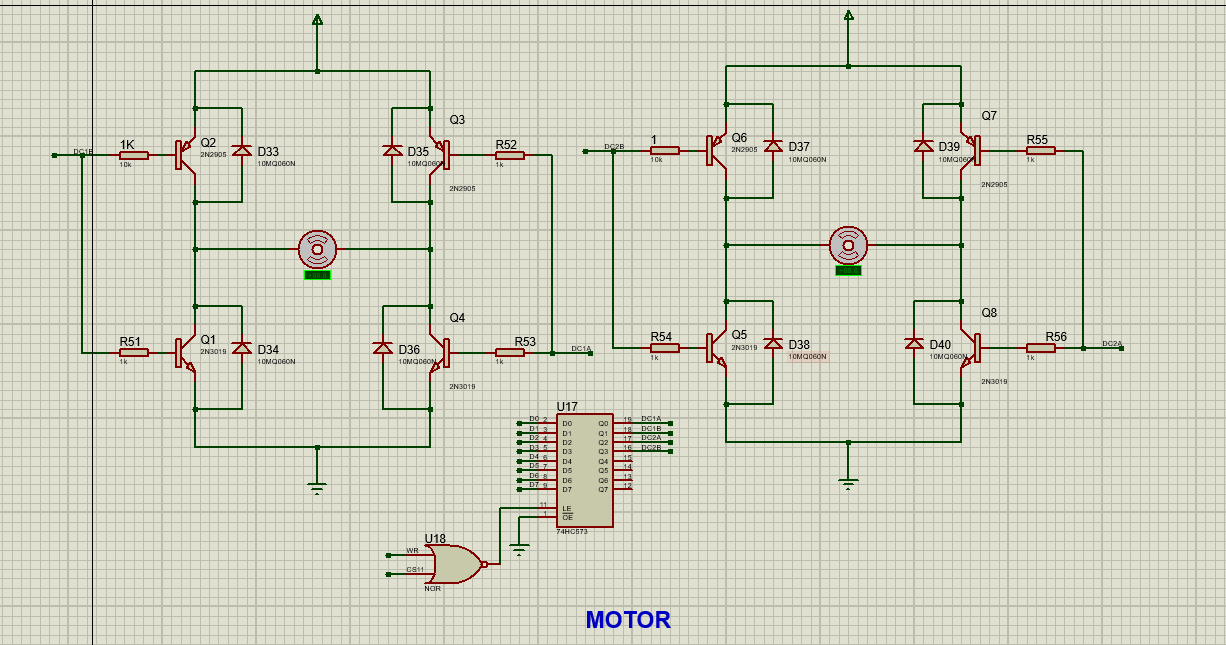
|  |  |
| --- | --- |
| **CHIP SELECT** | **DEVICES** |
| CS0 | 8 LEDS L0-L7 |
| CS1 | 8 LEDS L8-L15 |
| CS2 | 8 LEDS L16-L23 |
| CS3 | 8 LEDS L24-L31 |
| CS4 | 8 LEDS L32-L39 |
| CS5 | 8 LEDS L40-L47 |
| CS6 | 8 LEDS L48-L55 |
| CS7 | 8 LEDS L56-L63 |
| CS8 | INSTRUCTION LCD |
| CS9 | BUTTON |
| CS10 | RELAY |
| CS11 | DATA LCD |
| CS12 | LED7D1 |
| CS13 | LED7D2 |
| CS14 | LED7D3 |
| CS15 | LED7D4 |
| CS16 | DC MOTOR |

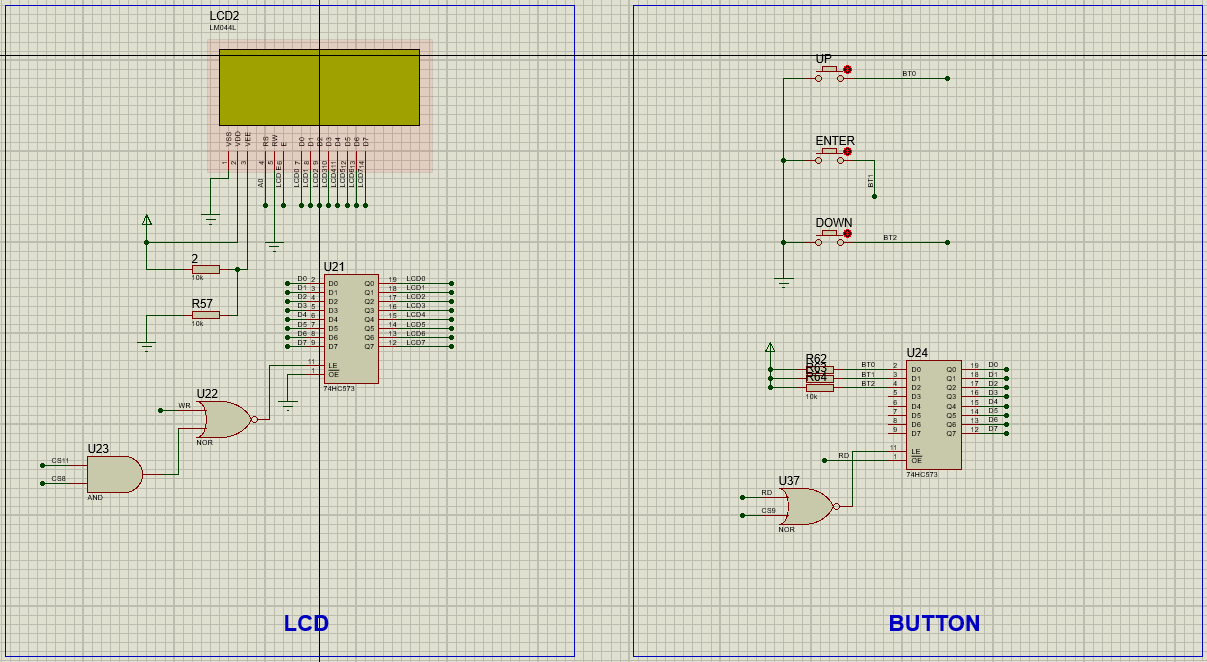












## SOFTWARE DESIGNS

**3.4.1 CODE DEFINE:**

#define EXMEM\_ADD 0x1100

//CS OFFSET

#define CS0 0

#define CS1 1

#define CS2 2

#define CS3 3

#define CS4 4

#define CS5 5

#define CS6 6

#define CS7 7

#define CS8 8

#define CS9 9

#define CS10 10

#define CS11 11

#define CS12 12

#define CS13 13

#define CS14 14

#define CS15 15

#define CS16 16

//define external device

#define LED0 \*(volatile unsigned char \*) (EXMEM\_ADD + CS0)

#define LED1 \*(volatile unsigned char \*) (EXMEM\_ADD + CS1)

#define LED2 \*(volatile unsigned char \*) (EXMEM\_ADD + CS2)

#define LED3 \*(volatile unsigned char \*) (EXMEM\_ADD + CS3)

#define LED4 \*(volatile unsigned char \*) (EXMEM\_ADD + CS4)

#define LED5 \*(volatile unsigned char \*) (EXMEM\_ADD + CS5)

#define LED6 \*(volatile unsigned char \*) (EXMEM\_ADD + CS6)

#define LED7 \*(volatile unsigned char \*) (EXMEM\_ADD + CS7)

//define button

#define READKEY (\*(unsigned char \*)(&Buttons) = \*(unsigned char \*) (EXMEM\_ADD+CS9) )

//define LCD

#define LCD\_DATA \*(unsigned char \*)(EXMEM\_ADD+CS11) //data

#define LCD\_INS \*(unsigned char \*)(EXMEM\_ADD+CS8) //instruction

#define LCDE\_H (PORTG |= (1<<3))

#define LCDE\_L (PORTG &= ~(1<<3))

//define 7seg

#define LED7D1 \*(unsigned char \*) (EXMEM\_ADD+CS12)

#define LED7D2 \*(unsigned char \*) (EXMEM\_ADD+CS13)

#define LED7D3 \*(unsigned char \*) (EXMEM\_ADD+CS14)

#define LED7D4 \*(unsigned char \*) (EXMEM\_ADD+CS15)

//define relay

#define RELAYMAP \*(unsigned char \*)(EXMEM\_ADD+CS10)

#define RELAY\_ACTIVATION (\*(volatile unsigned char \*)(EXMEM\_ADD+CS10) = \*( unsigned char \*)(&relays))

//define motor

#define MOTOR\_ACTIVATION (\*(volatile unsigned char \*)(EXMEM\_ADD+CS16) = \*(unsigned char \*)(&Motors))

#define STOP 0

#define FORWARD 1

#define REWARD 2

// LIBRARY FOR NULL CHARACTER IN LINKED LIST EXERCISE

#include<stddef.h> // for NULL popinter

struct Button{

unsigned char UP : 1 ;

unsigned char ENTER : 1 ;

unsigned char DOWN : 1 ;

};

struct Button Buttons;

#define ON 1

#define OFF 0

struct Motor {

unsigned char DC1 : 2 ;

unsigned char DC2: 2;

unsigned char DCFree: 4;};

struct Motor Motors ;

struct bits {

unsigned char RL0 : 1 ;

unsigned char RL1 : 1 ;

};

struct bits relays;

union RL {unsigned char relayall ;

struct bits relay ;

};

union RL Relays;

/\*------------------------------LINKED LIST----------------------------\*/

#define Function\_Alternate 0

#define Function\_Increment 1

#define Function\_Chasing 2

#define Function\_PushBack 3

#define Device\_Relay1 4

#define Device\_Relay2 5

#define Device\_Motor1 6

#define Device\_Motor2 7

#define Function\_Time 8

void Activation (char Devices, char Status){

switch(Devices){

case Function\_Alternate: if(Status) sangchoptatxenke();

else LedReset();

break;

case Function\_Increment: if(Status) sangdan();

else LedReset();

break;

case Function\_Chasing : if(Status) sangchay();

else LedReset();

break;

case Function\_PushBack : if(Status) sangdon();

else LedReset();

break;

case Device\_Relay1 : if(Status) Relays.relay.RL0 = 1;

else Relays.relay.RL0 = 0;

RELAY\_ACTIVATION;

break;

case Device\_Relay2 : if(Status) relays.RL1 = 1;

else relays.RL1 = 0;

RELAY\_ACTIVATION;

break;

case Device\_Motor1 : Motors.DC1 = Status;

MOTOR\_ACTIVATION;

break;

case Device\_Motor2 : Motors.DC2 = Status;

MOTOR\_ACTIVATION;

break;

case Function\_Time: if(Status) Timer();

else SevenDisplay(0000);

break;

}

}

typedef struct linker {

char MenuID;

struct linker \*pre ;

char Title[20];

char List1[20];

struct linker \*MenuList1; void (\*ActivationON)(char,char);

char List2[20];

struct linker \*MenuList2; void (\*ActivationOFF)(char,char);

char List3[20];

struct linker \*MenuList3;

char List4[20];

struct linker \*MenuList4;

} Menu ;

Menu

MainMenu,LedMenu,FUNCTION\_1,FUNCTION\_2,AcuatorMenu,TimeMenu,Increment,Chasing,PushBack,Alternate,

MotorMenu,Motor1Menu,Motor2Menu,RelaysMenu,Relays1Menu,Relays2Menu;

Menu MainMenu;

//MAIN MENU--------------------------------------------------------

Menu MainMenu = {

NULL,

NULL,

" Main Menu " ,

" LEDS " , &LedMenu,NULL,

" Actuators " , &AcuatorMenu,NULL,

" Time " , &TimeMenu,NULL,

};

// LED\_MENU--------------------------------------------------------

Menu LedMenu = {

NULL,

&MainMenu,

" LEDS ",

" FUNCTION\_1 " ,&FUNCTION\_1,NULL,

" FUNCTION\_2 " ,&FUNCTION\_2,NULL,

" BACK ",&MainMenu,NULL,

};

//-----------------------------------------------------------------------

Menu FUNCTION\_1 = {

NULL,

&LedMenu,

" FUNCTION\_1\_LED ",

" Alternate " ,&Alternate,NULL,

" Increment " ,&Increment,NULL,

" BACK ",&LedMenu,NULL,

};

Menu Alternate = {

Function\_Alternate,

&FUNCTION\_1,

" Alternate ",

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&LedMenu,NULL,

};

Menu Increment = {

Function\_Increment,

&FUNCTION\_1,

" Increment ",

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&FUNCTION\_1,NULL,

};

//-----------------------------------------------------------------------

Menu FUNCTION\_2 = {

NULL,

&LedMenu,

" FUNCTION\_2\_LED ",

" Chasing " ,&Chasing,NULL,

" PushBack " ,&PushBack,NULL,

" BACK ",&LedMenu,NULL,

};

Menu Chasing = {

Function\_Chasing,

&FUNCTION\_2,

" Chasing ",

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&FUNCTION\_2,NULL,

};

Menu PushBack = {

Function\_PushBack,

&FUNCTION\_2,

" PushBack ",

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&FUNCTION\_2,NULL,

};

//-----------------------------------------------------------------------

Menu TimeMenu = {

Function\_Time,

&MainMenu,

" Time " ,

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&MainMenu,NULL,

};

//-----------------------------------------------------------------------

// ACTUATOR\_MENU

Menu AcuatorMenu = {

NULL,

&MainMenu,

" ACTUATOR ",

" DC Motors " ,&MotorMenu,NULL,

" Relays " ,&RelaysMenu,NULL,

" BACK ",&MainMenu,NULL,

};

//MORTOR MENU

Menu MotorMenu = {

NULL,

&AcuatorMenu,

" DC Motors " ,

" Motors 01 " ,&Motor1Menu,NULL,

" Motors 02 " ,&Motor2Menu,NULL,

" BACK ",&AcuatorMenu,NULL,

};

//MOTOR\_SUB MENU 1

Menu Motor1Menu = {

Device\_Motor1,

&MotorMenu,

" Motors 1 " ,

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&MotorMenu,NULL,

};

Menu Motor2Menu = {

Device\_Motor2,

&MotorMenu,

" Motors 2 " ,

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&MotorMenu,NULL,

};

//RELAY MENU

Menu RelaysMenu = {

NULL,

&AcuatorMenu,

" Relays " ,

" Relay 1 " ,&Relays1Menu,NULL,

" Relay 2 " ,&Relays2Menu,NULL,

" BACK ",&AcuatorMenu,NULL,

};

Menu Relays1Menu = {

Device\_Relay1,

&RelaysMenu,

" Relay 1 " ,

" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&RelaysMenu,NULL,

};

Menu Relays2Menu = {

Device\_Relay2,

&RelaysMenu,

" Relay 2 " ,

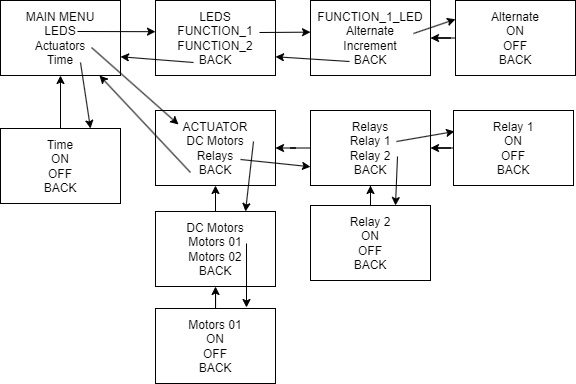
" ON " ,NULL,&Activation,

" OFF " ,NULL,&Activation,

" BACK ",&RelaysMenu,NULL,

};

**3.4.2 LINKED LIST MENU**



**Main Menu Structure**

* **Main** **Menu**: The top level of the menu. From here, users can navigate to submenus for LEDs, Actuators, or Time settings.
* **LEDs**: This leads to a further submenu where LED functions can be selected.
* **Actuators**: Leads to a submenu to control actuators, such as DC Motors or Relays.
* **Time**: Directly allows toggling time settings ON or OFF, or returning to the Main Menu.

**LEDs Submenu**

* FUNCTION\_1 and FUNCTION\_2: These are categories or types of LED operations. For example:
* **FUNCTION\_1\_LED**: Might include operations like:
* **Alternate**: Toggle LEDs in an alternating pattern.
* **Increment**: Incrementally light up LEDs.
* **Back**: Returns to the previous LEDs menu.

**Actuators Submenu**

* **DC Motors:** Opens a submenu to control specific motors.
* Motors 01 and Motors 02: Control individual motors with options to turn ON or OFF.
* **Relays**: Allows control of relay modules.
* Relay 1 and Relay 2: Each has options to turn ON or OFF.
* **Back:** Returns to the Actuators menu or further up to the Main Menu.

**Time Submenu**

* Direct actions to turn time-related settings ON or OFF without navigating into deeper submenus.

**Navigation**

Each submenu typically has a "Back" option, allowing the user to return to the higher-level menu. This is critical for usability, ensuring that users can easily navigate back and forth across different levels of the menu structure without needing to reset the system or lose their place.

**Linked List Implementation**

* Each menu item (Menu) likely holds data about the menu text (Title), function pointers for actions (ActivationON, ActivationOFF), and pointers to the next menu items (MenuList1, MenuList2, etc.).
* Navigational actions (like pressing "up" or "down" buttons) traverse these pointers to update the current menu display on the LCD or perform the selected actions.

**3.4.3 MAIN PROGRAM**

**Function LedDisplay**

void LedDisplay(unsigned long long led) {

LED0 = (unsigned char)(led & 0xFF);

LED1 = (unsigned char)((led >> 8) & 0xFF);

...

LED7 = (unsigned char)((led >> 56) & 0xFF);

}

This function displays a 64-bit pattern on an array of 8 LEDs. Each byte of the led variable controls a corresponding LED, which means each bit in a byte can turn on or off a specific segment or LED, depending on the hardware configuration.

**Function LedReset**

void LedReset() {

led = 0;

LedDisplay(led);

\_delay\_ms(100);

}

LedReset resets all the LEDs to an off state. It sets the led variable to 0 and then calls LedDisplay to apply this state, effectively turning off all the LEDs. It also introduces a 100-millisecond delay after turning off the LEDs to ensure the state change is noticeable.

**Function sangdan**

void sangdan() {

int i;

led = 0;

for (i=0; i<64; i++) {

led = (led << 1) | 1;

LedDisplay(led);

\_delay\_ms(100);

}

}

sangdan gradually turns on LEDs from the first to the last, creating a left-to-right incremental light-up effect. Each iteration of the loop shifts the led bits left and sets the least significant bit to 1, then displays this new LED state.

**Function sangchay**

sangchay creates a "chasing" light effect by lighting up one LED at a time across the LED array. Each LED lights up sequentially and then turns off as the next LED lights up.

void sangchay() {

int i;

led = 0;

for (i=0; i<64; i++) {

led = (unsigned long long)1 << i;

LedDisplay(led);

\_delay\_ms(100);

}

}

**Function sangdon**

void sangdon() {

unsigned char i, j;

unsigned long long run = 1;

unsigned long long led = 0x0000;

unsigned long long save\_state = 0x0000;

for (i=0; i<64; i++) {

for (j=0; j<64-i; j++) {

led = (run << j) | save\_state;

LedDisplay(led);

\_delay\_ms(100);

}

save\_state = led;

}

}

sangdon simulates a "push back" effect where each LED turns on one after the other until all are lit, then the pattern reverses. It progressively moves a lit LED through the array while maintaining previously lit LEDs in their state.

**Function sangchoptatxenke**

void sangdon() {

unsigned char i, j;

unsigned long long run = 1;

unsigned long long led = 0x0000;

unsigned long long save\_state = 0x0000;

for (i=0; i<64; i++) {

for (j=0; j<64-i; j++) {

led = (run << j) | save\_state;

LedDisplay(led);

\_delay\_ms(100);

}

save\_state = led;

}

}

sangchoptatxenke toggles LEDs on and off in an alternating pattern, creating a blinking effect. The LED states are shifted left and then right, causing the pattern to "dance" back and forth.

**Function TIMER**

void Timer()

{

unsigned char seconds = 0, minutes = 0;

unsigned char SegCode[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};

Variable Initialization: seconds and minutes are initialized to 0. They will store the current time in seconds and minutes, respectively.

Segment Code Array: SegCode contains hexadecimal codes for displaying digits 0-9 and some additional symbols on a seven-segment display. These codes are mapped to the segments that need to be lit up to represent each digit.

TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00); // Prescaler = 1024

TCNT0 = 0; // Initialize counter

Timer Setup: The TCCR0 register is configured to set the prescaler for the timer to 1024. This setting slows down the timer tick rate, making it more suitable for clock applications.

Counter Initialization: TCNT0 is the timer counter register that is set to 0 to start counting from a known state

while(1)

{

LED7D1 = SegCode[seconds % 10]; // Display units of seconds

LED7D2 = SegCode[seconds / 10]; // Display tens of seconds

LED7D3 = SegCode[minutes % 10]; // Display units of minutes

LED7D4 = SegCode[minutes / 10]; // Display tens of minutes

Infinite Loop: The main part of the function is an infinite loop that continuously updates the display based on the seconds and minutes variables.

Display Update: Each of the seven-segment display units (LED7D1 to LED7D4) is assigned a value from SegCode based on the current time. This conversion ensures that the correct segments are lit to display the current time accurately.

for(int i = 0; i < 4; i++) { // Wait for 4 timer overflows

while (!(TIFR & (1 << TOV0))); // Wait for overflow

TIFR |= (1 << TOV0); // Clear overflow flag

}

Timing Control: This block of code waits for the timer to overflow four times. Each overflow represents a fixed time period, so waiting for four overflows gives a one-second interval (adjustable based on the clock frequency and prescaler settings).

Overflow Handling: Checks the timer overflow flag (TOV0) and clears it after it's set. This ensures each wait is due to an actual timer overflow.

if (++seconds == 60) { // Increment seconds, reset if 60

seconds = 0;

if (++minutes == 60) { // Increment minutes, reset if 60

minutes = 0;

}

}

}

}

Time Increment: Each cycle of the loop increments the seconds variable. If seconds reaches 60, it is reset to 0 and minutes is incremented. Similarly, if minutes reaches 60, it is also reset to 0.

Resetting Time: This simple mechanism allows the function to keep track of time accurately and reset the counters when they exceed their bounds, mimicking a typical clock's behavior.

**FULL MAIN PROGRAM:**

#include <xc.h>

#include "userdef.h"

#define F\_CPU 1000000

#include<util/delay.h>

//unsigned long long led = 0x5555555555555555;// Use `unsigned long long` for 64-bit variable

void LedDisplay (unsigned long long led)

{

LED0 = (unsigned char)(led & 0xFF);

LED1 = (unsigned char)((led >> 8) & 0xFF);

LED2 = (unsigned char)((led >> 16) & 0xFF);

LED3 = (unsigned char)((led >> 24) & 0xFF);

LED4 = (unsigned char)((led >> 32) & 0xFF);

LED5 = (unsigned char)((led >> 40) & 0xFF);

LED6 = (unsigned char)((led >> 48) & 0xFF);

LED7 = (unsigned char)((led >> 56) & 0xFF);

}

unsigned long long led ;

void LedReset(){

led = 0;

LedDisplay(led);

\_delay\_ms(100);

}

//Increment

void sangdan(){

int i;

led = 0;

for (i=0;i<64;i++)

{

led =(led<<1)|1;

LedDisplay(led);

\_delay\_ms(100);

}

}

//Chasing

void sangchay(){

int i;

led = 0;

for (i=0;i<64;i++)

{

(led = (unsigned long long)1<<i);

LedDisplay(led);

\_delay\_ms(100);

}

}

//PushBack

void sangdon(){

unsigned char i,j;

unsigned long long run = 1;//0000 0000 0000 0001

unsigned long long led = 0x0000;

unsigned long long save\_state = 0x0000;

for (i=0;i<64;i++)

{

for (j=0;j<64-i;j++)

{

led = (run<<j)|save\_state;

LedDisplay(led);

\_delay\_ms(100);

}

save\_state = led;

LedDisplay(led);

\_delay\_ms(100);

}

}

//Alternate

void sangchoptatxenke(){

int i;

led = 0x5555555555555555;

for (i=0;i<64;i++)

{

(led = (led<<1));

LedDisplay(led);

\_delay\_ms(100);

(led = (led>>1));

LedDisplay(led);

\_delay\_ms(100);

}

}

void SevenDisplay (unsigned int value) {

unsigned char SegCode[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,

0x88,0x83,0xC6,0xA1,0x86,0x8E};

LED7D1=SegCode[value%10];

LED7D2=SegCode[(value/10)%10];

LED7D3=SegCode[(value/100)%10];

LED7D4=SegCode[(value/1000)%10];

}

void Relay(){

relays.RL0=0;

relays.RL1=0;

RELAY\_ACTIVATION;

}

void Motor(){

Motors.DC1 = STOP;

Motors.DC2 = STOP ;

MOTOR\_ACTIVATION;

}

/\*void Button(){

READKEY;

if(!Buttons.UP)

{while(!Buttons.UP)READKEY;

Motors.DC1 = FORWARD ;}

if(!Buttons.DOWN)

{while(!Buttons.DOWN)READKEY;

Motors.DC2 = REWARD ;}

MOTOR\_ACTIVATION;

}\*/

// 7 Seg Timer initialize

// 7 Seg Timer Display

// 7 Seg Timer Display

void Timer()

{

unsigned char seconds = 0, minutes = 0;

unsigned char SegCode[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};

TCCR0 |= (1 << CS02) | (1 << CS01) | (1 << CS00); // Prescaler = 1024

TCNT0 = 0; // Initialize counter

while(1)

{

LED7D1 = SegCode[seconds % 10]; // Display units of seconds

LED7D2 = SegCode[seconds / 10]; // Display tens of seconds

LED7D3 = SegCode[minutes % 10]; // Display units of minutes

LED7D4 = SegCode[minutes / 10]; // Display tens of minutes

for(int i = 0; i < 4; i++) { // Wait for 4 timer overflows

while (!(TIFR & (1 << TOV0))); // Wait for overflow

TIFR |= (1 << TOV0); // Clear overflow flag

}

if (++seconds == 60) { // Increment seconds, reset if 60

seconds = 0;

if (++minutes == 60) // Increment minutes, reset if 60

minutes = 0;

}

}

}

//LCD

void LCDInit()

{

LCD\_INS =0x38 ;LCDE\_H ;\_delay\_us(1); LCDE\_L; \_delay\_us(1);

\_delay\_us(200) ;

LCD\_INS =0x0c ; LCDE\_H ;\_delay\_us(1); LCDE\_L; \_delay\_us(1);

\_delay\_us(200) ;

LCD\_INS =0x06 ; LCDE\_H ;\_delay\_us(1); LCDE\_L; \_delay\_us(1);

\_delay\_us(200) ;

LCD\_INS =0x01 ;LCDE\_H ;\_delay\_us(1); LCDE\_L; \_delay\_us(1);

\_delay\_ms(20) ;

}

void PrintL (const char \*str,unsigned char line, unsigned char col) {

unsigned char add ;

switch (line){

case 0: add = 0x80; break;

case 1: add = 0xC0; break;

case 2: add = 0x94; break;

case 3: add = 0xD4; break;

}

LCD\_INS =add + col ;

LCDE\_H ;\_delay\_us(1); LCDE\_L; \_delay\_us(50); // move to the position

while(\*(str)!='\0' ) {

LCD\_DATA = \*str++ ;

LCDE\_H ;\_delay\_us(1); LCDE\_L;

\_delay\_us(10);

}

}

//Display LCD with LINKED LIST options

void MenuDisplay(Menu \*menu, unsigned char select)

{

PrintL(menu->Title,0,0);

PrintL(menu->List1,1,0);

PrintL(menu->List2,2,0);

PrintL(menu->List3,3,0);

PrintL(">", select, 0);

}

void main(void) {

MCUCR = 0x80 ;// Assume setup SRE (external memory enable)

Menu \*menu;

unsigned char select = 1;

XMCRA = 0;

DDRG =0x0f;

//LCD PRINTING

menu = &MainMenu;

LCDInit();

MenuDisplay(menu, select);

while(1){

if (!Buttons.UP) {

while(!Buttons.UP) READKEY;

select = (select==1) ?3:select-1;

MenuDisplay(menu, select);

}

READKEY;

if (!Buttons.DOWN) {

while(!Buttons.DOWN) READKEY;

switch (select)

{

case 1:

if(menu->ActivationON!=NULL)menu->ActivationON(menu->MenuID,ON);

break;

case 2:

if(menu->ActivationOFF!=NULL)menu->ActivationOFF(menu->MenuID,OFF);

break;

}

}

if (!Buttons.ENTER)

{

while(!Buttons.ENTER) READKEY;

switch (select)

{

case 1:menu=(menu->MenuList1==NULL)?menu:menu->MenuList1;

break;

case 2:menu=(menu->MenuList2==NULL)?menu:menu->MenuList2;

break;

case 3:menu=(menu->MenuList3==NULL)?menu:menu->MenuList3;

break;

}

MenuDisplay(menu,select) ;

}

// Delay for a short period to allow display refresh

\_delay\_ms(100);

}

return;

}

# CONCLUSIONS

This project successfully designed and implemented an embedded system using the ATmega128 microcontroller, which controls various hardware components such as LEDs, LCDs, DC motors, and relays. The system was developed to address specific needs for device control and user interaction, providing a practical example of embedded system capabilities.