EMBEDDED C

PIC16F877A Driven Automotive Cooling System

Abstract

This Project describes the design and simulation of an automotive cooling system, built around the use of the PIC16F877A Microcontroller. Modelling the simplified version of the engine cooling system and outputting the system status onto an LCD display and a UART virtual terminal.

Jacob C (2420842)

2420842@student.uwtsd.ac.uk

Contents

Introduction 2
1.1 Project Aim and Objectives 2
2. Background and Literature Review 2
2.1 Overview of Automotive Cooling Systems 2
2.2 Embedded Systems in Automotive Control 2
2.3 PIC16f877A Microcontroller Overview 3
3. System Overview 3
3.1 System Description 3
3.2 Input and Output Requirements 3
3.3 Operational Sequence Summary 3
4. Hardware Design and Circuit 4
4.1 Microcontroller Selection 4
4.2 Switch Inputs and LED Indicators 4
4.3 LCD Interface 4
4.4 Motor Driver 4
5. Software Design and Program Logic 4
5.1 Program Structure 4
5.2 Control Logic 5
5.3 Switch Debouncing 5
5.4 LCD and UART message Handling 5
6. System Flowchart 5
6.1 Flowchart 5
6.2 State Transitions 6
7. Simulation Testing and Results 6
7.1 Testing Methodology 6
7.2 Test Cases and Observations 6
8. Discussion and Evaluation 6
9. Conclusion 7
9.1 Summary and Achievements 7
9.2 Areas for Improvement 7
References ..... **Error! Bookmark not defined.**
Appendix 7

Introduction

This project presents the design and simulation of an automotive cooling system, built around the PIC16f877A microcontroller, will take simulated input from a user and output the results onto an LCD display and also into a virtual terminal found within the Proteus design software.

The following sections will outline the background research, the systems implemented and the overall design of the circuitry. As well as the stages involved in testing the circuit and highlighting key areas for improvement revealed through the testing process.

1.1 Project Aim and Objectives

The overall aim of this project is to design and simulate a microcontroller-base automotive cooling system using embedded C.

The key objectives within this project include:

- To implement state-based control.
- Interfacing with multiple peripherals, including 3 input switches, an output motor, an LCD, a terminal output, and 3 LEDs
- Producing well optimized code, to minimize program size and the load on the microprocessor.
- Include debounce protection on the input switches

2. Background Research

2.1 Overview of Automotive Cooling Systems

The purpose of Automotive cooling systems has always been to keep the temperature of the engine within safe operating limits [1], this helps to preserve the integrity of the engine, ensuring a reliable product lifetime, whilst also assisting to improve the efficiency of the engine.

From a system perspective, cooling systems work through passing fluid, coolant, through the engine block to pull out heat, and then into a radiator where this fluid is cooled [2]. These radiators typically have a fan adjoined to them, which cools the fluid faster, providing better cooling for the engine.

2.2 Embedded Systems in Automotive Control

Automotive cooling systems regulate the internal temperature of the engine, to do this a microcontroller, or similar embedded system is used to drive the cooling system. These micro controllers allow for reliable real-time control of the system through the use of sensors and that feed to a central control unit, also known as the ECU [3].

This also allows for full control over the functions and systems of the engine, as well as the ability to communicate between individual modules within the engine, meaning certain modules can cascade instructions between one another [3], such as engaging the cooling system when the temperature spikes too an unacceptable level.

2.3 PIC16F877A Microcontroller Overview

The PIC16F877A is an 8-bit microcontroller, commonly used within embedded applications due to its simplicity and its wide range of peripheral support. And its Harvard architecture allows for efficient execution of instructions and storage of data without affecting the program memory [4].

The device features 33 digital I/O pins across multiple ports allowing for the ability to interface with multiple components simultaneously, including switches, LEDs, LCDs and motor drivers [4]. Integrated within the chip are peripherals such as timers and a Universal Synchronous Asynchronous Receiver Transmitter (USART), which allows for serial communication, often used for system monitoring and debugging [4].

Its compatibility with embedded C compilers makes it well suited for demonstrating structured programming and fine-tuned peripheral control [5] [6].

3. System Overview

3.1 System Description

The system is a simulated automotive cooling system controller, driven by a PIC16F877A microcontroller, and programmed in C using CCS. This microcontroller acts as the central control unit, processing user inputs and then controlling the system outputs. This is programmed using a state-machine, managing the system and forcing the program to only function in sequence.

3.2 Input and Output

User inputs are read using three pushbuttons, which control a series of predefined states which operate in sequence. These buttons cannot be pressed outside of the set order, which would normally disrupt the system.

The current system status is communicated through LEDs, a 16x2 LCD display, and through a UART virtual terminal.

A motor output is used to simulate the radiator fan of the system, which is driven through a motor driver, controlled through a logic output from the microcontroller.

3.3 Operational Sequence

The overall system is driven and managed through a state-machine, ensuring the system only operates in sequence, and preventing invalid state transitions.

The system begins in a ready state, awaiting an input, and then progresses through a sequence of:

Engine Start -> Coolant Confirmation -> Fan activation -> Alert -> Shutdown

After entering the shutdown state for a set time, the system resets, allowing for the system to be tested again.

4. Hardware Design and Circuit

4.1 Microcontroller Selection

The PIC16F877A was selected to act as the central processing unit for its suitability for control-based applications [4]. It provides a significant number of I/O pins to interface with multiple peripherals.

Its integrated peripherals support the functional requirements of the system, allowing for smooth and reliable operation.

4.2 Switch Inputs and LED Indicators

Three push button switches are used as digital inputs and are used to control the progression of the system through the various operational stages. These allow for the user to initialise processes such as the engine start, the coolant confirmation and the fan activation.

LED indicators are connected to digital output pins and are used to display the system status visually, providing feedback to the user of the current status of the system.

Software and hardware debouncing measures are in place to ensure reliable input detection.

4.3 LCD Interface

An LM016L 16x2 LCD is configured in a 4-bit data configuration to minimize pin usage. This is used to provide real time feedback to the user regarding the system status, displaying messages and alerts, informing the user of the current status of the circuit. This allows for system behaviour to be actively monitored by the user.

4.4 Motor Driver

A DRV8816 motor driver was used to control the motor used in the simulated radiator fan. This is due to the fact that a microcontroller cannot directly drive a motor without risking damage from feedback currents. Using a motor driver ensures safe and reliable operation whilst protecting the microcontroller.

5. Software Design and Program Logic

5.1 Program Structure

The software is written in CCS C and follows a structured approach to assist in maintaining the software should the need for adjustments arise.

The main program loop begins by awaiting user input, which enters into a switch function which handles the current state of the program, which is stored in a global variable. A second global variable is used to handle which pushbutton is allowed to execute the next set of instructions, called statepermit.

Additional functions are also used to handle repeated tasks, such as LED blinking, reducing code duplication.

5.2 Control Logic

System behaviour is controlled through a state-machine, wherein each operational stage is represented by a defined state. This is kept within a switch-case function, to ensure that only the currently active state is active. State transitions are sequential and are initially driven by user input. This prevents any potential invalid transitions.

5.3 Switch Debouncing

Software debouncing is implemented to prevent any unintentional triggers that can potentially be caused by a mechanical switch bouncing. A short delay introduced after the detection of an input, followed by a second read to ensure the input is not read multiple times.

5.4 LCD and UART message Handling

System status messaged are output to both the LCD and the UART terminal, this provides both visual feedback to the user, whilst also supporting system monitoring and debugging during the simulation. This improves system transparency and ease of testing.

6. System Flowchart

6.1 Flowchart

Figure B.1 in Appendix B illustrates the overall control flow of the cooling system. Showcasing the sequential progression of the system from system start, through until the shutdown.

6.2 State Transitions

The state table provided in Appendix D.1 outlines the state structure of the system.

The ‘statepermit’ variable works to ensure that a state cannot be advanced due to user input without the previous state having been completed. After the input reliant states are completed, states are able to be forcefully advanced by the program.

7. Simulation Testing and Results

7.1 Testing Methodology

The system was tested within the Proteus design software, where each push button input was activated individually under different conditions in order to observe a different interaction. System responses were observed using the UART interface, LCD display, the LED indicators, and the motor output. Tests were repeated to ensure consistent results.

7.2 Test Cases and Observations

Evident in Appendix E.1, initial testing confirmed that the system initialises and remains in a ready state prior to any user input.

Later tests showed that the states were progressing in the correct sequence and that the statepermit condition was operating as intended, blocking inputs that have already been pressed, and only allowing for the next button in the sequence to be input.

The other states also progressed as intended following the third and final user input.

8. Discussion and Evaluation

The simulation results demonstrate that the system operates correctly and reliably, following the intended operational structure. The use of a system states improved control clarity, allowing for clearly defined instructions to be written, providing clear and predictable functionality.

However, the system is limited by the reliance on simulated inputs and polling-based control, in a real-world scenario, sensor driven inputs would be able to provide much finer control over the system, and introducing interrupts would improve the responsiveness, allowing for the system to react to real world variables.

Despite these limitations, the design meets the project objectives and demonstrates sound programming practices.

9. Conclusion

9.1 Summary and Achievements

The project successfully implemented a simulated automotive cooling system using a PIC16F877A. The system demonstrated structured control logic, proper peripheral interfacing and control, and reliable state sequencing.

All project objectives were achieved, and the system performed admirably within the simulated environment.

9.2 Areas for Improvement

Future improvements could include the integration of proper temperature sensors, to replace the manual inputs and introduce autonomy to the system. The use of interrupts within the system could also improve efficiency, allowing for the system to react to changes within the system, and allow for fault detection to be implemented.

References

[1] Bosch, Automotive Handbook, 11th Edition, Stuttgart: Society of Automotive Engineers, 2022.

[2] J. E. &. R. Thompson, Automotive Technology: A Systems Approach (7th ed.), Boston, MA: Cengage Learning, 2019.

[3] Taylor & Francis Group, Automotive Embedded Systems Handbook, Boca Raton, Florida: CRC Press, 2009.

[4] Microchip, “Microchip.com,” 2013. [Online]. Available: https://www.microchip.com/en-us/product/pic16f877a?utm_source=chatgpt.com#Documentation. [Accessed 18 12 2025].

[5] H. H. Ward, C Programming for the PIC Microcontroller, New York: Apress, 2010.

[6] T. Williamhurst, Designing Embedded Systems with PIC Microcontrollers: Principles and Applications, Oxford: Newnes, 2010.

Appendix

APPENDIX A – Source Code

```c
// tells the compiler what chip we are using by adding a library,
// includes the necessary pins and functions
#include <16f877a.h>

// sets the clock speed of the microcontroller using the internal oscillator
#use    delay(clock=20000000)
#define delay     delay_ms

// sets the pins for UART for use with the virtual terminal
#use    rs232(baud=9600, xmit=pin_c6, rcv=pin_c7)

// Defines the locations of the three switches for refence later in the code
#define  sw1       pin_b4 // Defines pin b4 as sw1
#define  sw2       pin_b5 // Defines pin b5 as sw2
#define  sw3       pin_b6 // Defines pin b6 as sw3

// Defines the LED outputs for later reference
#define  ledGr     pin_b0 // Defines pin b0 as the green led
#define  ledYel    pin_b1 // Defines pin b1 as the yellow led
#define  ledRed    pin_b2 // Defines pin b2 as the red led

// Defines the pints for the motor output
#define  m1        pin_c0 // Defines pin c0 as Motor input 1
#define  m2        pin_c1 // Defines pin c1 as Motor input 2

// Provides named states for the switch state, with 0 being an 'idle' state
#define  started   1
#define  coolant   2
#define  fan       3
#define  alert     4
#define  shutdown  5
#define  sysready  6

// Defines the LCD Pins found in Port D, in line with the LCD.c library
#define  LCD_RS_PIN     PIN_D0
#define  LCD_RW_PIN     PIN_D1
#define  LCD_ENABLE_PIN PIN_D2
#define  LCD_DATA4      PIN_D3
#define  LCD_DATA5      PIN_D4
#define  LCD_DATA6      PIN_D5
#define  LCD_DATA7      PIN_D6
#include <lcd.c>

// Creates an integar called 'state' and assigns it a value
int state = 0;

/*
Creates an integar called 'state permit and assigns it a value
State permit acts as a permission variable
This is to stop states being called too early through the switches
*/
int statepermit = 0;

/*
This function handles the blinking of the LEDS, depending on the current state
within the program. This function is intended to act as a delay, whilst running
another process simultaneously, in this case, blinking the LEDs.
*/
void ledblink(){
   if(state == 1){
   /*
   loops the code within the for loop whilst incrementing the value of 'i'
   until the desired value is reached, overall delay is
   (([Maximum value of 'i'] * 2) * 100) Milliseconds
   */
      for(int i=0/* defines i as 0*/;
         i<3/* max value of i is 3*/;
         i++/*increment i for each loop*/){
         //Blinks the green LED
         output_high(ledGr);
         delay(100);
         output_low(ledGr);
         delay(100);
      }
```

Figure A.1 – Source Code 1/4

```c
    }
    if(state == 3){
        for(int i=0; i<15; i++){//loops 15 times, creating a 3s delay
            //Blinks the Yellow LED
            output_high(ledYel);
            delay(100);
            output_low(ledYel);
            delay(100);
        }
    }
    if(state == 4){
        for(int i=0; i<5; i++){// loops 5 times creating a 1s delay
            //Blinks the Red LED
            output_high(ledRed);
            delay(100);
            output_low(ledRed);
            delay(100);
        }
    }
}

//main body of code
void main()
{
    // sets port B pins 0, 1, 2, 3(unused) to output and pins 4 - 7 to input
    set_tris_b(0xF0);
    output_b(0x00);
    //sets port D to output and initialises the LCD
    set_tris_d(0x00);
    lcd_init();

    //Prints system ready in the Virtual Terminal via the UART output
    //Then also prints system ready on the LCD
    printf("System Ready\n");
    lcd_putc("\fSystem Ready");

    //main loop of the code
    while(TRUE)
    {
    //Sets state and the state permission to 1
        if(input(sw1)){
            delay(20);//Simple debounce
            if(input(sw1)){
                //Sets 'state' to '1'
                state = 1;
            }
        }

/*
Handles 6 different bodies of code, addressable through the 'state' variable,
only one state can be active at a time, and will continue to loop indefinitely
until the value of 'state' changes
*/
        switch(state){

            case started:
                //Sets the green LED to HIGH
                output_high(ledGr);
                //Displays 'Engine starting' on the terminal and the LCD
                printf("Engine Starting\n");
                lcd_putc("\fEngine Starting");
                //Calls to the ledblink function, for state '1'
                ledblink();
                //Displays 'Warming...' on the LCD and the Terminal
                printf("Warming...\n");
                lcd_putc("\nWarming...");
                //Holds for 1s
                delay(1000);
                //Displays 'ready' on the LCD and the Terminal
                printf("Ready!\n");
                lcd_putc("\n\fReady!");
                //Sets the green LED to resume a HIGH state
                output_high(ledGr);
                //Sets the state to 2, and statepermit to 1, permitting sw2 input
```

Figure A.2 - Source Code 2/4

```
            state = 2;
            statepermit = 1;
            break;

        case coolant:
            //Only allows sw2 if statepermit is set to 1
            if(input(sw2) && statepermit == 1){
                delay(20);//Simple Debounce
                if(input(sw2)){
                //Displays 'Coolant Ok' on the Terminal and the LCD
                    printf("Coolant Ok\r\n");
                    lcd_putc("\fCoolant Ok");
                    //Sets the yellow LED to HIGH
                    output_high(ledYel);
                    delay(100);
                    //Sets state to 3, and statepermit to 2, allowing for sw3
                    state = 3;
                    statepermit = 2;
                    break;
                }
            }


        case fan:
        //Only allows sw3 if statepermit is set to 2
            if(input(sw3) && statepermit == 2){
                delay(20);//Simple Debounce
                if(input(sw3)){
                //Sets m1 high and m2 low, outputs 'Fan Activated' on the
                //Terminal and LCD
                    output_high(m1);
                    output_low(m2);
                    printf("Fan Activated\r\n");
                    lcd_putc("\fFan Activated");
                    delay(100);
                    // Outputs 'Fan Activated' on the terminal and LCD
                    printf("Motor Running \r\n");
                    lcd_putc("\nMotor Running");
                    ledblink();// Blinks the LED to show motor is running for 3s
                    //Sets m1&m2 Low
                    output_low(m1);
                    output_low(m2);
                    //Set state to 4
                    state = 4;
                    break;
                    }
            }

        case alert:
            if(state == 4){
                //Displays 'Alert!!' on the LCD and the Terminal
                printf("ALERT!!\r\n");
                lcd_putc("\fALERT!!");
                //Runs ledblink for state 4
                ledblink();
                //Sets state to 5
                state = 5;
                break;
            }

        case shutdown:
            if(state == 5){
                //Displays 'Shutting down...' on the LCD and the Terminal
                printf("Shutting down...\r\n");
                lcd_putc("\fShutting down...");
                delay(750);
                //Turns off all outputs on PortB
                output_b(0x00);
                //Displays 'Shutdown Complete' on the LCD and the Terminal
                printf("Shutdown Complete\r\n");
                lcd_putc("\fShutdown Complete");
                delay(1500);
                //Sets state to 6
                state = 6;
```

Figure A.3 - Source Code 3/4

```
                break;
            }

        case sysready:
            if(state == 6){
                //Displays 'System Ready' on the LCD and the Terminal
                printf("\f\rSystem Ready\r\n");
                lcd_putc("\fSystem Ready");
                //Returns state to 0
                state = 0;
                break;
            }



        }
    }

}
```

Figure A.3 - Source Code 4/4

APPENDIX B - Flow Diagram

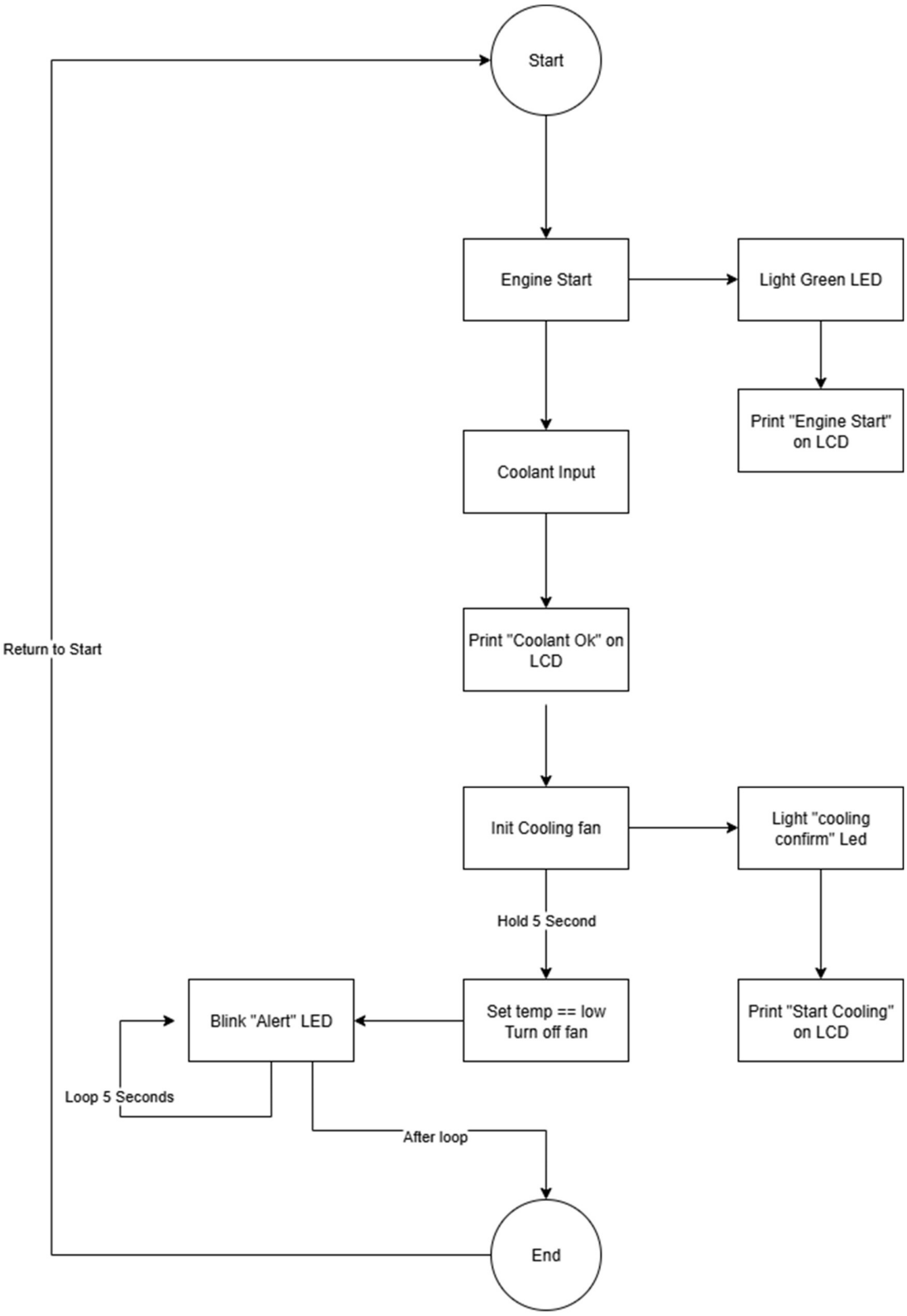

Figure B.1 – Flow Diagram

APPENDIX C – Circuit Diagram

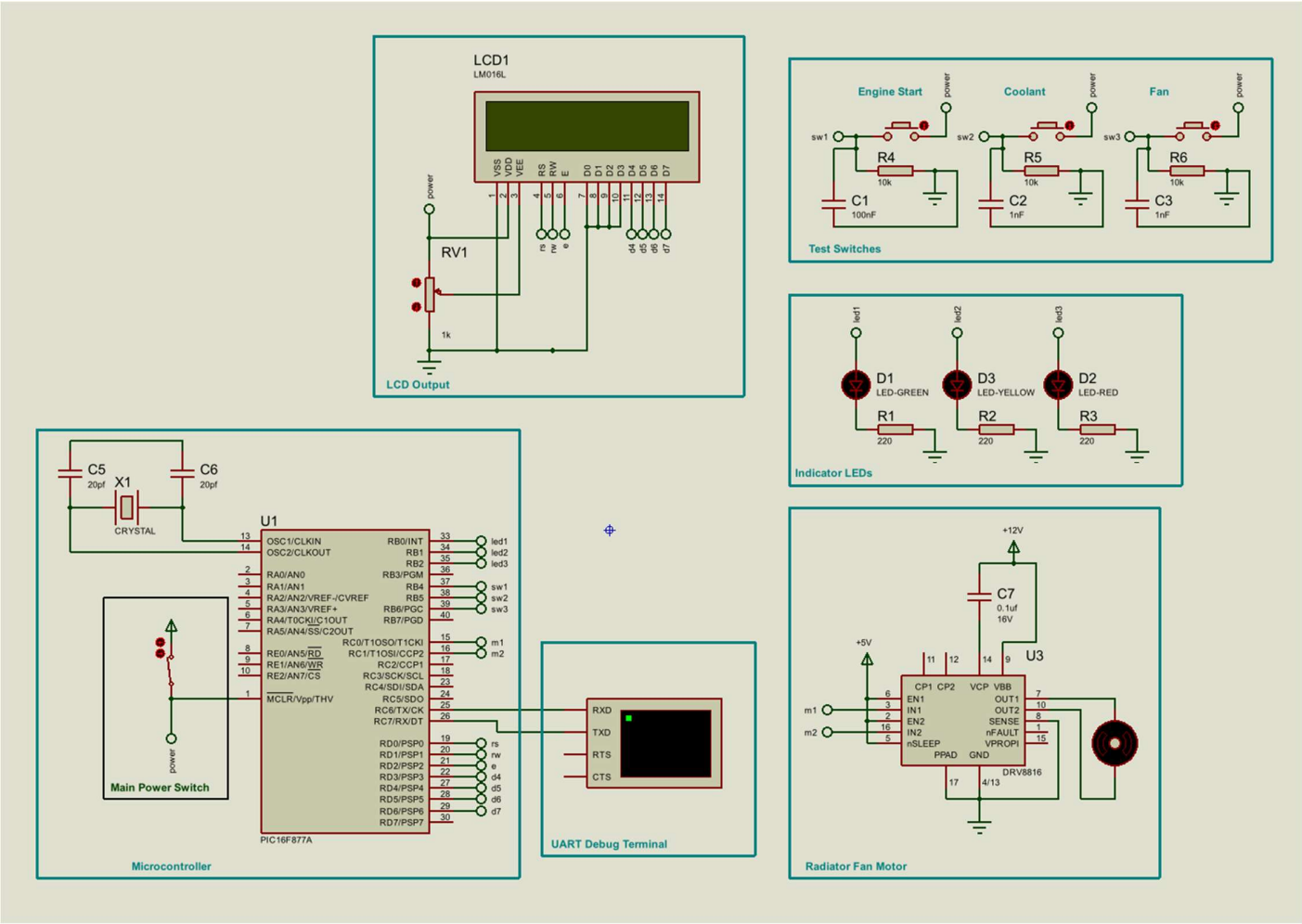

Figure C.1 - Circuit Diagram

APPENDIX D – State Table

State ID	State Name	Entry Actions / Outputs	Stay Condition	Transition Trigger	Next State
0	Idle (waiting)	No specific outputs enforced (system shows “System Ready” from initialisation and later reset state).	Remains idle while **SW1 not pressed**.	**SW1 pressed** (debounced)	1
1	Started	Green LED ON UART: “Engine Starting”, “Warming…”, “Ready!”. LCD shows corresponding messages, calls LED blink routine for green. Sets statepermit = 1.	Runs its sequence then forces transition.	End of “Engine Starting/Warming/Ready” sequence	2
2	Coolant	Wait state for coolant confirmation; when valid, LCD/UART: “Coolant Ok”; Yellow LED ON; sets statepermit = 2.	Remains in state 2 while **SW2 not pressed** or statepermit != 1.	**SW2 pressed** AND statepermit == 1 (debounced)	3
3	Fan	Wait state for fan activation; on valid input: Motor ON (M1 high), UART/LCD: “Fan Activated”, “Motor Running”; yellow LED blink routine runs (3s); Motor OFF; then forces transition.	Remains in state 3 while **SW3 not pressed** or statepermit != 2.	**SW3 pressed** AND statepermit == 2 (debounced)	4
4	Alert	UART/LCD: “ALERT!!”; Red LED blink routine	Runs its sequence then	End of alert blink sequence	5

		runs (1s); then forces transition.	forces transition.		
5	Shutdown	UART/LCD: “Shutting down…”, then “Shutdown Complete”; Port B outputs cleared (LEDs off); then forces transition.	Runs its shutdown sequence then forces transition.	End of shutdown sequence	6
6	SysReady (Reset/Ready)	UART/LCD: “System Ready”; resets state = 0.	Immediately returns to idle.	After printing readiness	0

Table D.1 – A state stable outlining the state progression of the circuit.

APPENDIX E – Test Table

Test Case	**Input Action**	**Expected Result**	**Observed Result**
TC1	Press SW1	Engine start sequence	As expected
TC2	Press SW2 before SW1	No response	As expected
TC3	Press SW3 after SW2	Fan activates	As expected
TC4	Press SW1 again after SW2 and SW1	No Response	As expected
TC5	No input	System remained in a ready state	As expected

Table E.1 – A test table showing different tests, their expected results and the observed outcome