



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Generating a program for a specified embedded requirement in C.

Embedded C ACDA4013

Name: Luke O'Sullivan

Student Number: 2320102

Beng Electrical and Electronic Engineering
Degree Apprenticeship

Academic Year: 2025/26

Table of Contents

1. Introduction	3
2. Background Research	3
3. Circuit Design	5
4 . Flowchart/Pseudocode	9
5. Screen Dumps	10
6 . Details of Testing	18
7. Conclusion	19
References	20
Appendices	21
Appendix A – Testing Table	22

Table of Figures:

Figure 1: Proteus Circuit Design of Automotive Coolant System.

Figure 2: A flowchart designed on AutoCAD software to represent the Automotive Coolant system and assist coding.

Figure 3: Start of Simulation, representing the system/the engine starting. (Along with virtual terminal communication).

Figure 4: LCD Display confirming the engine start-up after the LED flash (Triggered by SW1 and showing virtual terminal communication).

Figure 5: Coolant LED Flashes (Triggered by SW2).

Figure 6: LCD Display and virtual terminal Confirming Coolant Check is OK.

Figure 7: Motor begins operation, LCD Display confirms this after input from SW3. (Also displaying communication to virtual terminal).

Figure 8: Alert LED Flashes after motor runs momentarily.

Figure 9: Alert is confirmed by LCD Display. (Also displaying virtual terminal communication).

Figure 10: LCD Display confirming after the Alert, the shutdown of the system has been complete. (Also displaying virtual terminal communication).

Figure 11: Initial Hex file set up.

Figure 12: Lines 1-38 of CCS Code/Program.

Figure 13: Lines 39-75 of CCS Code/Program.

Figure 14: Lines 75-110 of CCS Code/Program.

Figure 15: Lines 110-126 of CCS Code/Program.

1. Introduction

- 1.1. The following report will cover the design and programming of a microcontroller system, replicating an automotive cooling system, designed to operate vehicle control logic. The design will be implemented using a PIC16F877A microcontroller and Proteus for simulation. The program will be written using CCS C language and using various components.

2. Background Research

- 2.1. **Automotive cooling system:** The automotive cooling system being designed replicates a cooling system that is commonly used in modern vehicles. The purpose of this system is to continuously control engine temperature by maintaining a constant coolant flow and activating the radiator fan when specified temperature thresholds are reached. Modern automotive control units automate this process by monitoring sensor inputs and executing predefined responses to prevent overheating and ensure efficient operation [1]. This process normally follows the procedures for engine start, coolant level check, fan activation, and controlled shutdown. Replicating this system using embedded methods provides a simple, structured breakdown of a common, real-life system crucial to everyday safety.
- 2.2. **PIC16F877A Microcontroller:** A PIC16F877A microcontroller is an 8-bit microcontroller commonly used within embedded systems. It is well known for having versatile I/O uses and integrated peripheral modules. It provides multiple digital ports suitable for interfacing with switches, LEDs, and LCD modules, while its built-in USART enables reliable serial communication for system monitoring [2]. The PIC16F877A is ideal for educational projects due to its efficient architecture, instruction set, and simple hardware integration.
- 2.3. **Key Components:** Within system design, some of the key components used included push buttons, LED indicators/lights, LCD Module, DC motor and a UART interface.

The push buttons provide the system with inputs that can replicate vehicle sensors. When combined with resistors in the circuit, they allow the microcontroller to detect changes and perform the required operations.

LEDs provide quick and clear visual feedback from the system for each state, helping to confirm engine start-up, coolant verification, and alarm conditions.

The LCD module provides visual feedback, clearly stating the system's operations through each phase. This mode is commonly used in embedded systems to conserve I/O resources without compromising readability [3].

The circuit design uses a DC motor to replicate a vehicle radiator fan, typically driven by a transistor or driver stage to handle the current. This reflects real automotive systems, where motors require dedicated driver circuitry for reliable operation [5].

UART communication and interfaces give serial output from a virtual terminal, giving real-time detection of system messages, allowing the microcontroller to transmit serial data without requiring a shared clock signal between devices [8]. This replicates the diagnostic communication that is used in vehicle electronic control units.

- 2.4. **Embedded System Sequencing:** The system follows a structure where each button press transitions the controller through predefined operational states. This structure provides predictable behaviour and clear separation between stages. State-based design is a standard method in embedded systems for managing sequential processes and ensuring deterministic control flow [4].

3. Circuit Design

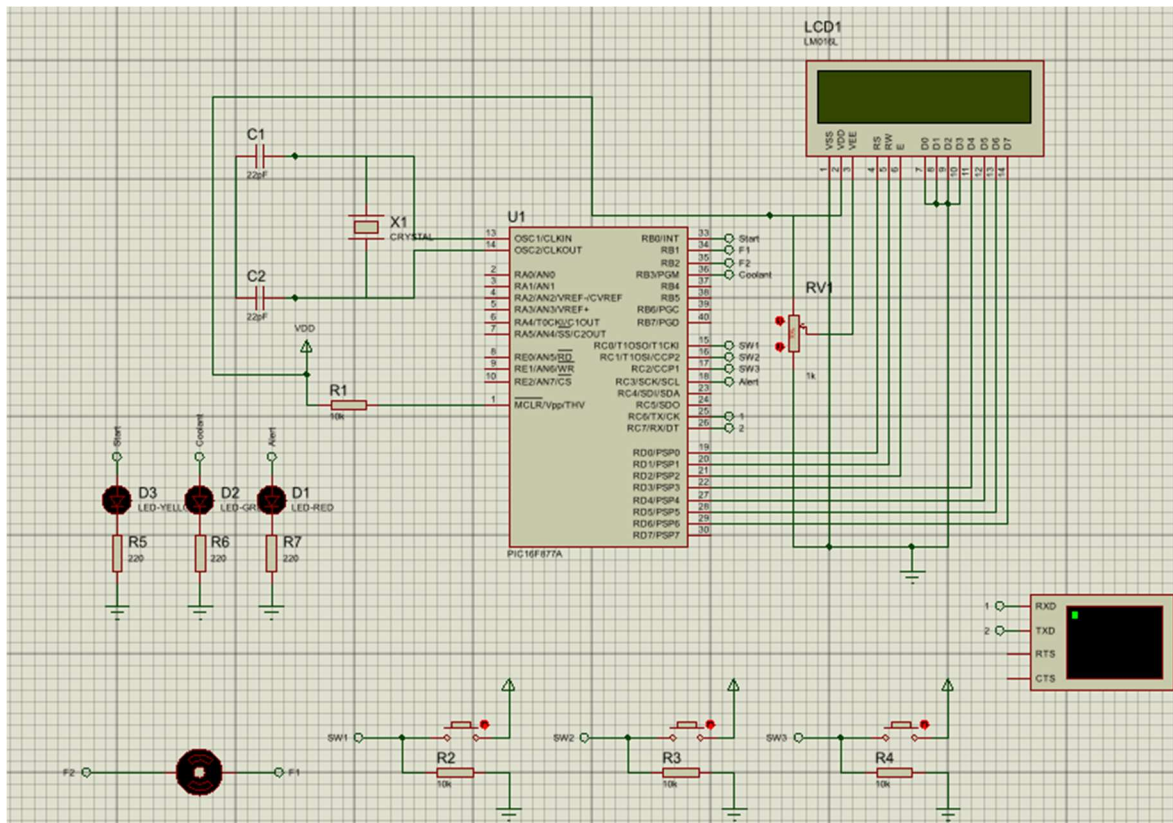
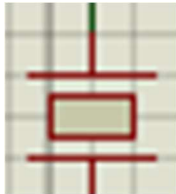
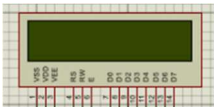
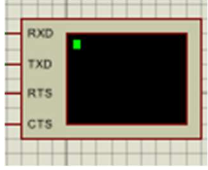
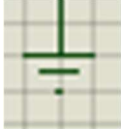


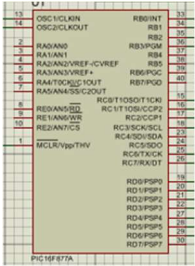
Figure 1: Proteus Circuit Design of Automotive Coolant System.

- 3.1. Figure 1 shows the Automotive cooling system created through Proteus software. The circuit is created to support and visualise the program. Providing the required input and output interfaces. Three push buttons are implemented and labelled SW1, SW2 and SW3. Each switch acts as a trigger for each phase of the program. LED indicators are then implemented to confirm and visualise the triggers' feedback, each LED paired with a current-limiting resistor. A 16x2 LCD display was used and connected to Port D of the PIC16F877A microcontroller, which acts as the main interface in the circuit and consistently presents system messages throughout the program. A crystal oscillator is used with 22 pF capacitors to ensure reliable timing during program operation. A DC Motor is operated through two control lines (MOTOR_IN1 and MOTOR_IN2). This demonstrates the fan-activation stage of the operation. Each stage of the operation is being monitored in real time through a virtual terminal for debugging and verification. The circuit has several power and ground terminals to power each component correctly and ensure smooth operation.

Table 1 : Component List

Component	Symbol	Tag	Function	Key Parameters
Resistor		R1-R7	Limits current flow and regulates voltage within the circuit.	10k Ω , 220 Ω
Capacitor		C1-C2	Stores electrical charge and smooths or filters voltage in the circuit, also used for timing or coupling, depending on the design.	22 pF
Push-Button		SW1-SW3	Provides a manual momentary input to the circuit; changes state when pressed.	Contact type (NO/NC)
LED		D1-D3	Provides visual indication of circuit status.	
Crystal		X1	Provides a highly stable and precise oscillation frequency for timing, clock generation, or microcontroller operation.	
LCD		LCD1	Provides alphanumeric visual output; displays 2 lines of 16 characters.	

Virtual Terminal		n/a	Provides a software-based interface for sending and receiving serial data between the microcontroller and a PC; used for debugging, monitoring values, and displaying text output.	
DC Motor (Fan)		n/a	Converts electrical energy into mechanical rotation; used for airflow, cooling, or mechanical movement in the circuit.	
Ground		n/a	Provides a common reference point for all circuit voltages; completes return paths for current; stabilises the circuit's electrical potential.	
Power		VDD	Provides the positive supply voltage to all active components in the circuit (e.g., microcontroller, LCD, sensors).	

PIC16F877A	 The diagram shows the pinout for the PIC16F877A microcontroller. It is a 40-pin device with pins numbered 1 to 40. The functions for each pin are as follows: 1: OSC1/CLKIN 2: OSC2/CLKOUT 3: RB0 4: RB1 5: RB2 6: RB3 7: RB4 8: RB5 9: RB6 10: RB7 11: RB8 12: RB9 13: RB10 14: RB11 15: RB12 16: RB13 17: RB14 18: RB15 19: RB16 20: RB17 21: RB18 22: RB19 23: RB20 24: RB21 25: RB22 26: RB23 27: RB24 28: RB25 29: RB26 30: RB27 31: RB28 32: RB29 33: RB30 34: RB31 35: RB32 36: RB33 37: RB34 38: RB35 39: RB36 40: RB37	PIC16F877A	Acts as the main processing unit; reads inputs, executes program instructions, controls outputs such as LCD, motor driver, and serial communication.	
------------	---	------------	--	--

4 . Flowchart/Pseudocode

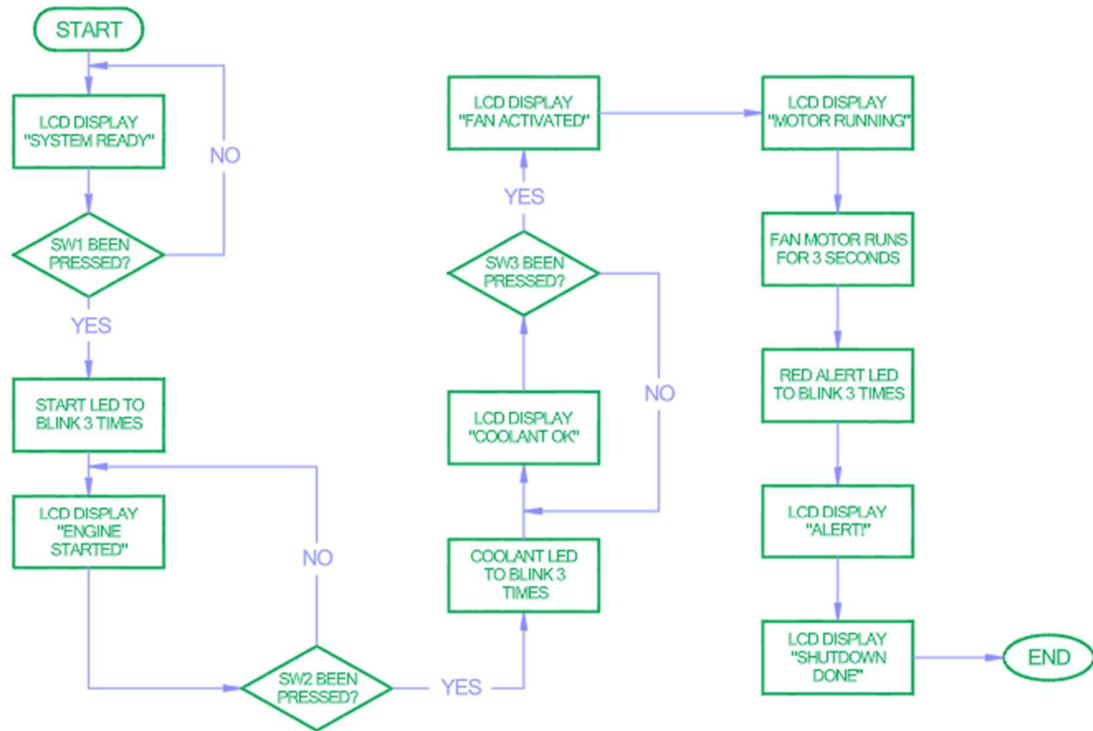


Figure 2: A flowchart designed on AutoCAD software to represent the Automotive Coolant system and assist coding.

4.1. The flowchart displayed in Figure 2 shows the step-by-step process carried out by the embedded system, showing the engine start-up, cooling and shutdown sequence. After the system starts up, the program expects three user inputs from switches SW1, SW2, and SW3, each corresponding to a separate phase of the sequence. At each phase, the system provides visual feedback through LED bulbs and an LCD display. Once the initial start-up and coolant check phases have been completed, the system will activate the cooling fan motor for a fixed duration of 3 seconds before initiating the shutdown procedure. The procedure concludes with an alert setting off and a final shutdown message. The flowchart in Figure 2 highlights the program's structure and how user inputs generate output signals, forming a complete, operational cycle. The flowchart shown in Figure 2 was produced using ISO 1028 standards to ensure consistency and accuracy of the flowchart following the relevant ISO documentation conventions [6].

5. Screen Dumps

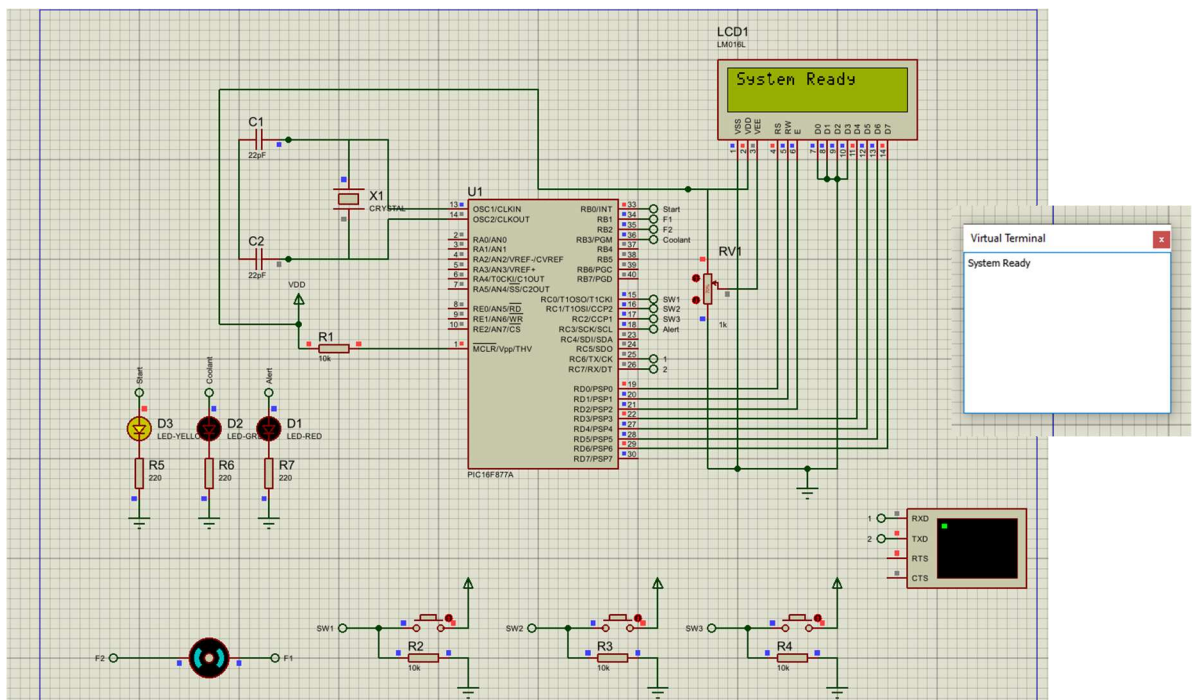


Figure 3: Start of Simulation, representing the system/the engine starting. (Along with virtual terminal communication).

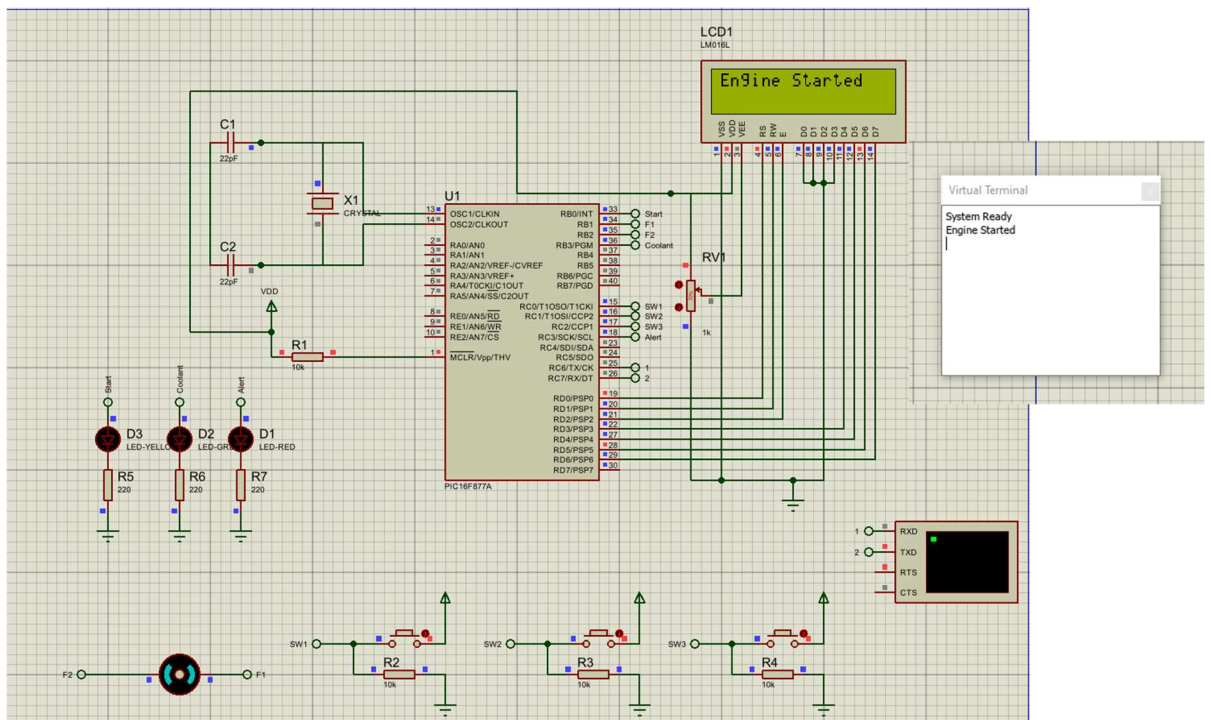


Figure 4: LCD Display confirming the engine start up after the LED flash (Triggered by SW1 and showing virtual terminal communication).

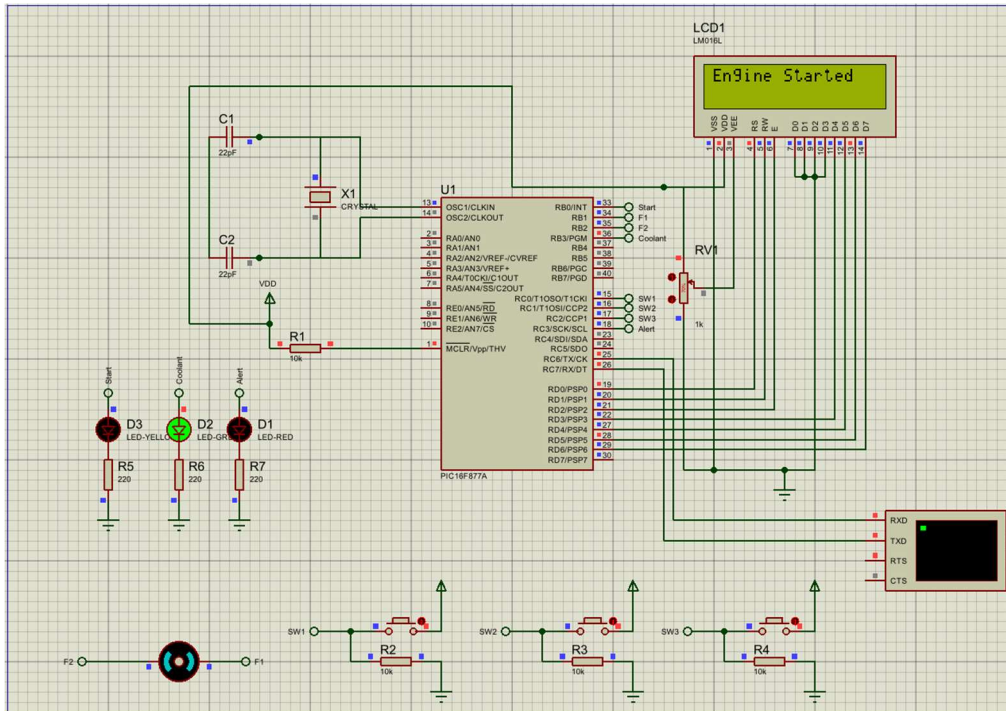


Figure 5: Coolant LED Flashes (Triggered by SW2)

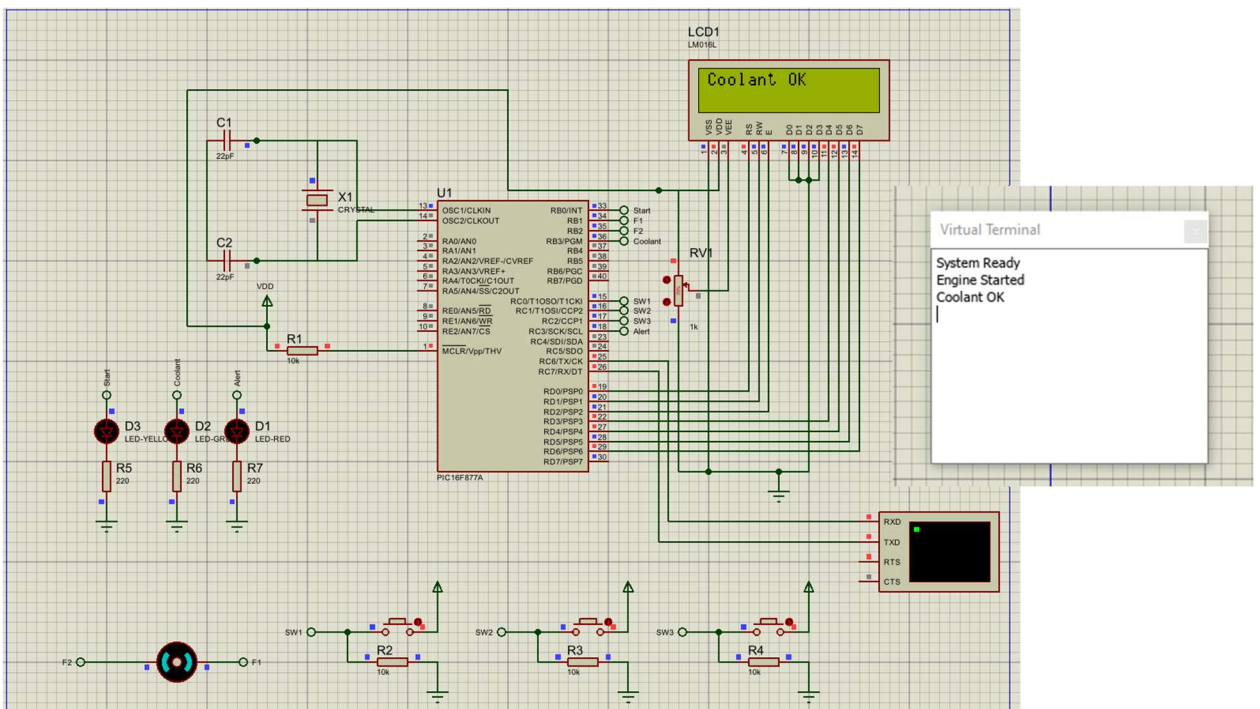


Figure 6: LCD Display and virtual terminal Confirming Coolant Check is OK.

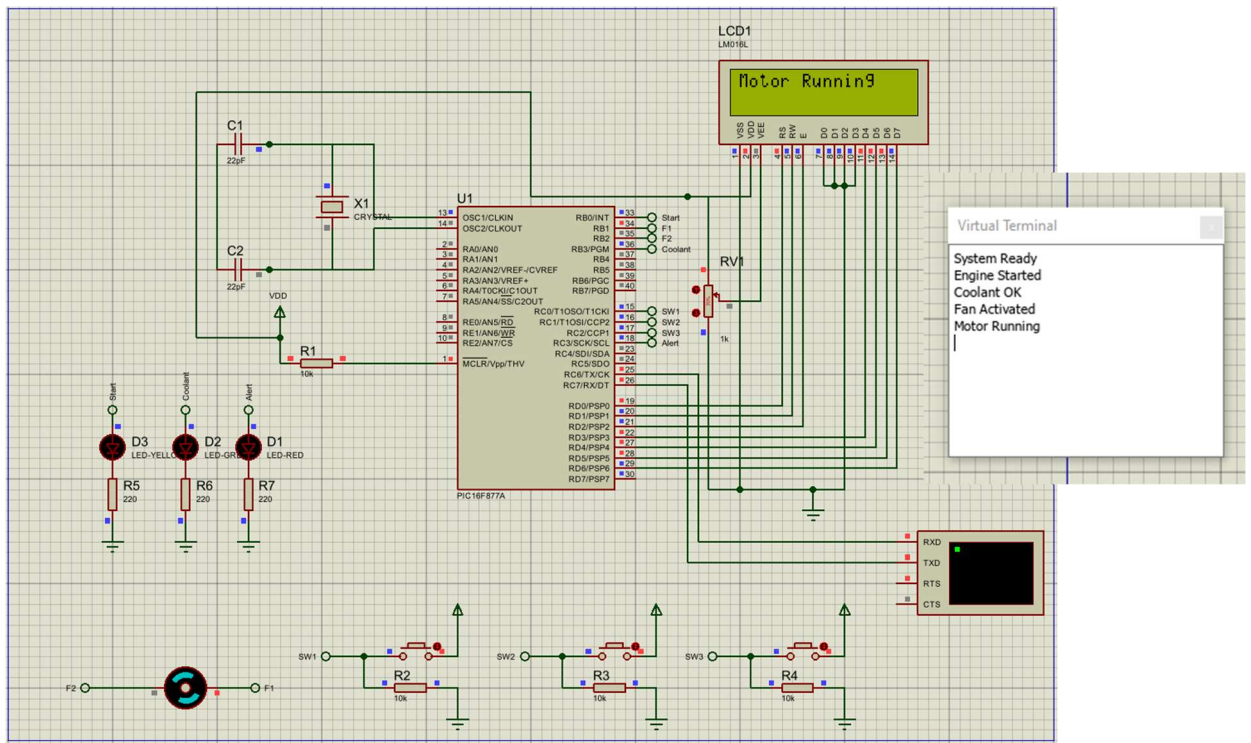


Figure 7: Motor begins operation, LCD Display confirms this after input from SW3. (Also displaying communication to virtual terminal).

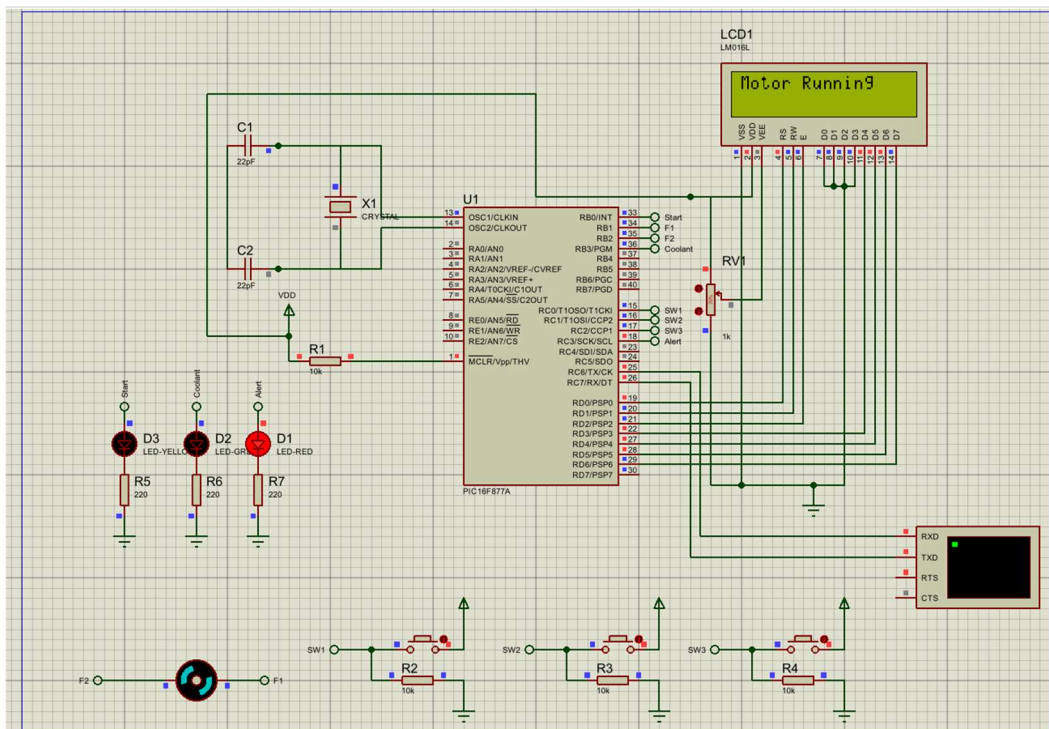


Figure 8: Alert LED Flashes after motor runs momentarily.

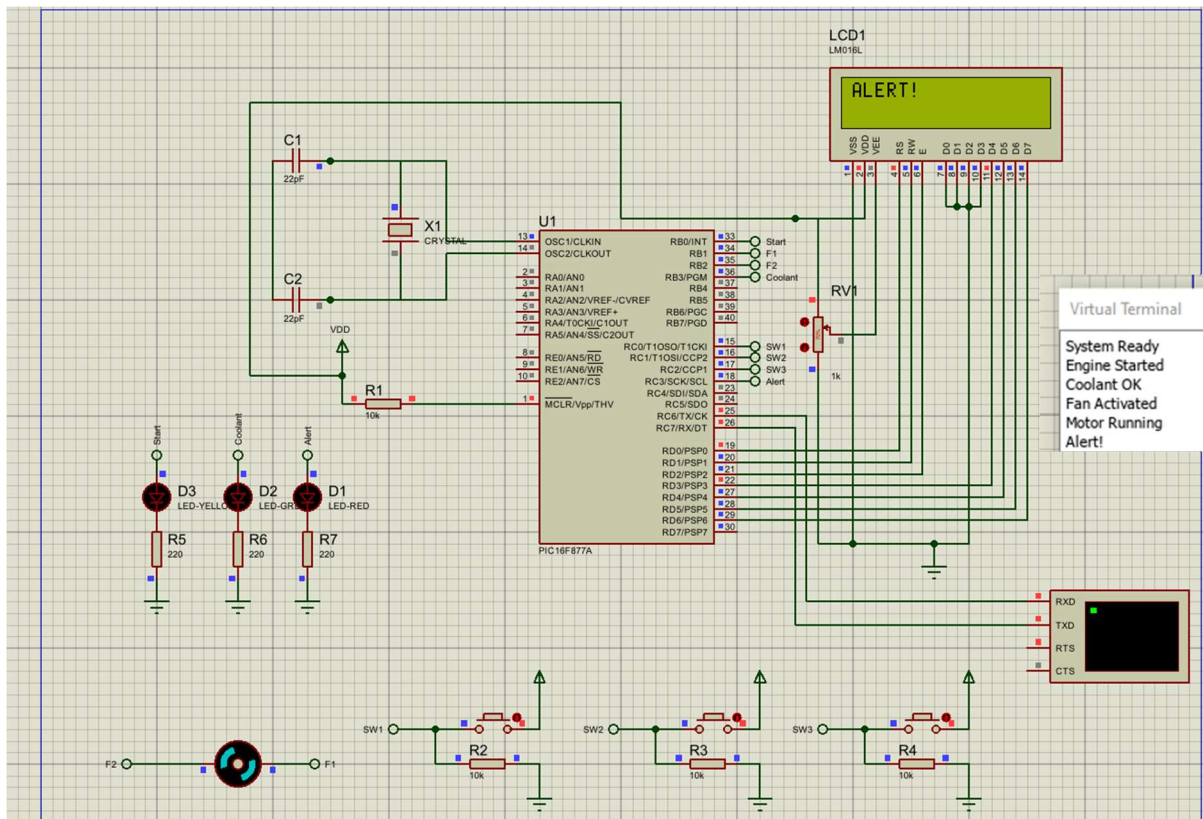


Figure 9: Alert is confirmed by LCD Display. (Also displaying virtual terminal communication).

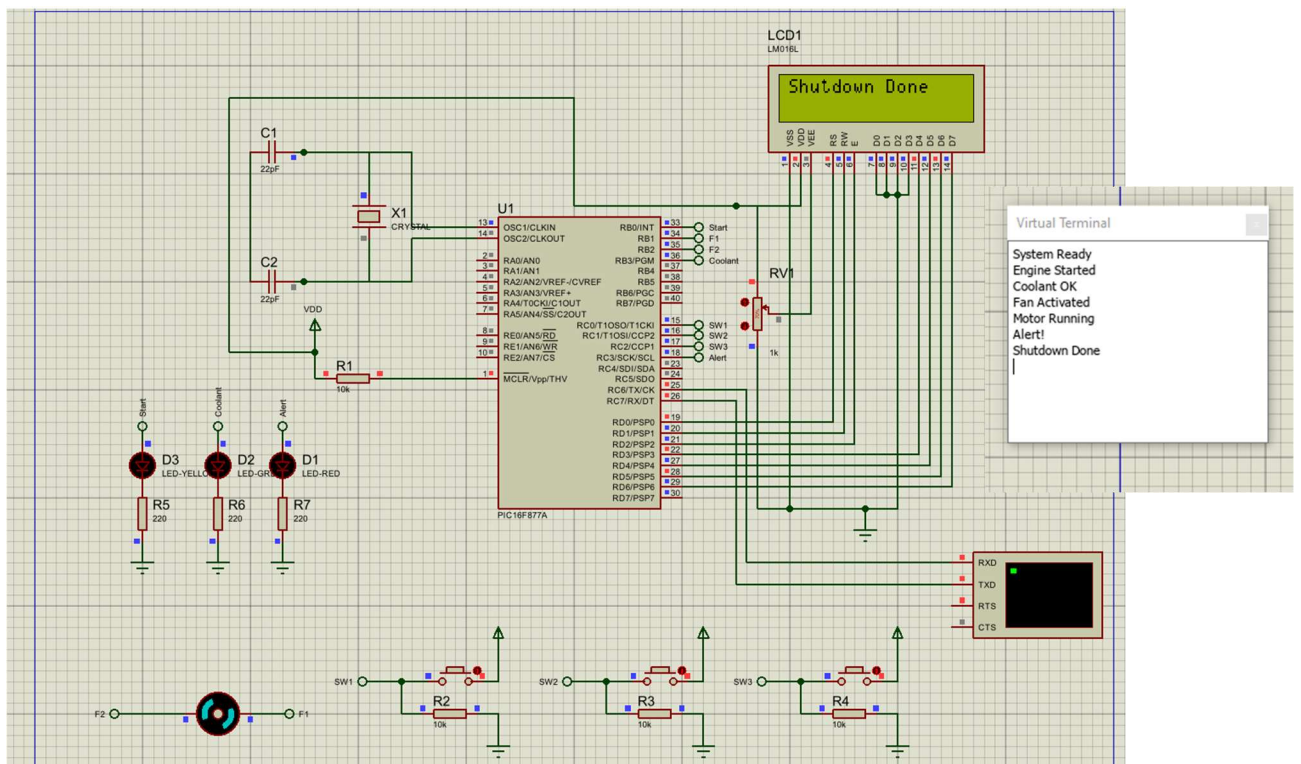


Figure 10: LCD Display confirming after the Alert, the shutdown of the system has been complete. (Also displaying virtual terminal communication).

```

1 #include <16F877A.h>
2 #fuses NOWDT, HS, NOPROTECT, NOLVP
3 #use delay(crystal=20000000)
4 #use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, bits=8, parity=N, stop=1)

```

Figure 11: Initial Hex file set up.

```

1 #include <mainlcd.h>
2
3
4
5 void main()
6
7 {
8
9     set_tris_d(0x00); // RD0-RD6 outputs to LCD
10    lcd_init(); // Initialise LCD once
11    // RB0-RB2 as outputs (blue LED + motor driver)
12    set_tris_b(0b11111000); // RB0, RB1, RB2 = outputs
13    // RC3 must be an output for the alert LED
14    // set_tris_c(0b00000111); // RC3 = 0 (output), others inputs
15
16
17    // Ensure all outputs start LOW
18    output_low(PIN_B0); // Blue LED OFF
19    output_low(MOTOR_IN1);
20    output_low(MOTOR_IN2);
21    output_low(ALERT_LED); // Red LED OFF
22
23    // System Ready
24    lcd_init();
25    lcd_putc("\fSystem Ready");
26    Printf("System Ready\r\n");
27    delay_ms(1000);
28
29    // Wait for switches
30    while(!input(SW1));
31    delay_ms(50);
32
33
34
35
36    for(int i = 0; i < 3; i++)
37    {
38

```

Figure 12: Lines 1-38 of CCS Code/Program.

5.1. Lines 1-38 of the program establish the initial software process flow by configuring the microcontroller's inputs and outputs. Specifically, PORT D and PORT B pins are designated as outputs, permitting software-driven control of the LCD, LEDs, and motor required for later program stages. The software process begins by initializing the LCD to a defined operating state, which ensures it is ready to show messages as the program advances. After the LCD setup, all output pins are explicitly driven to a low logic level. This step initiates the system in a controlled and predictable state and prevents accidental activation of the outputs. Establishing this orderly sequence at program start is a fundamental aspect of reliable embedded software process flow, securing operational safety and consistent functioning throughout the subsequent stages.

The LCD then displays a “System Ready” message and the virtual terminal, which provides immediate feedback. The program will then enter a waiting phase for SW1 input, ensuring that no processes or operations are carried out without deliberate input. A simple delay is also included to avoid switch bounce(s).

```

39
40     output_high(START_LED);
41     delay_ms(100);
42     output_low(START_LED);
43     delay_ms(100);
44
45 }
46
47 // Engine Started
48 lcd_putc("\fEngine Started");
49 Printf("Engine Started\r\n");
50 delay_ms(1000);
51 while(!input(SW2));
52     delay_ms(50);
53
54 for(int i = 0; i < 3; i++)
55 {
56
57     output_high(COOLANT_LED);
58     delay_ms(100);
59     output_low(COOLANT_LED);
60     delay_ms(100);
61
62 }
63
64 // Coolant OK
65 lcd_putc("\fCoolant OK");
66 Printf("Coolant OK\r\n");
67 delay_ms(1000);
68 while(!input(SW3));
69     delay_ms(50);
70
71 // Fan Activated
72 lcd_putc("\fFan Activated");
73 Printf("Fan Activated\r\n");
74 delay_ms(1000);
75

```

Figure 13: Lines 39-75 of CCS Code/Program.

5.2. Firstly, this section of code demonstrates the visual confirmation of system operations by pulsing the Start LED three times using high/low output logic. Once this sequence finishes, both the LCD and the virtual terminal respond with the message “Engine Started”, confirming successful operation and readiness for the next stage. The program again waits for a deliberate user input from SW2, ensuring a controlled procedure.

Another three-pulse sequence for visual verification is then added for the Coolant verification LED. This stage is processed once SW2 has been pressed. Once this operation is complete, the LCD and virtual terminal show “Coolant OK”, another waiting phase is activated until SW3 is pressed, which activates the motor/fan, with the LCD and virtual terminal displaying “Fan Activated”, demonstrating the next phase of operation.

```

75
76 // Motor Running
77 lcd_putc("\fMotor Running");
78 Printf("Motor Running\r\n");
79 delay_ms(1000);
80
81
82 // Start fan operation cycle
83 output_high(MOTOR_IN1);
84 output_low(MOTOR_IN2);
85
86 // Run fan for 3 seconds
87 delay_ms(3000);
88
89 // Stop fan
90 output_low(MOTOR_IN1);
91 output_low(MOTOR_IN2);
92
93 // Blink RED alert LED 3 times with 100 ms interval
94 for(int i = 0; i < 3; i++)
95 {
96
97     output_high(ALERT_LED);
98     delay_ms(100);
99     output_low(ALERT_LED);
100     delay_ms(100);
101
102 }
103
104 {
105
106     // LCD Show (ALERT!)
107     lcd_putc("\fALERT!");
108     Printf("Alert!\r\n");
109     delay_ms(1000);
110

```

Figure 14: Lines 75-110 of CCS Code/Program

5.3. This section of code demonstrates the motor running by providing clear feedback that reports “Motor Running” on both the LCD and the virtual terminal. The motor is driven by setting MOTOR_IN1 high and MOTOR_IN2 low, causing it to rotate for three seconds.

After the run, the motor's operation is stopped by using a low input to stop the fan, ensuring a safe shutdown. Another three-pulse alert is sequenced for the ALERT phase of the program, which enables the red LED and provides a message from the LCD and virtual terminal displaying “ALERT!”.

```

110  delay_ms(1000);
111
112  // LCD Show (Shutdown Done)
113  lcd_putc("\fShutdown Done");
114  Printf("Shutdown Done\r\n");
115  delay_ms(1000);
116
117  }
118
119  // End of operation Cycle
120  while(TRUE) { }
121
122
123  }
124
125
126

```

Figure 15: Lines 110-126 of CCS Code/Program.

5.4. The final section of code demonstrates the Shutdown and the system's end. This is visually confirmed by the LCD and virtual terminal when they display “Shutdown Done”, indicating that all operations have been completed and programmed successfully. The system then enters an infinite loop, preventing any further action, whether deliberate or not.

6 . Details of Testing

6.1. The program was tested using stages, with component-specific tests to verify the operation of each block before full integration [9]. Individual outputs such as the LEDs, DC motor, LCD, etc., were tested to verify that the program produced the expected visual response. Each switch input was validated to ensure that the user action was appropriately detected and progressed the program through each phase. Each delay was checked and adjusted to ensure a clear, aesthetic program, using a trial-and-error process. Finally, the program was tested from start to finish, observing the complete cycle to ensure that the compatibility of each phase and behaviour was consistent and that no unexpected events occurred.

6.2. During testing, two major issues arose that affected the system's development and operation. Firstly, after the initial setup of the circuit, the Proteus software became unresponsive due to a software or system fault, requiring technical support to update and restore it to a working state. Once this issue was resolved, testing and development of the program resumed. Next, an issue emerged with the virtual terminal: the relevant files had not been added and configured correctly for it to operate. Meaning, in the first instance, there was no output from the virtual terminal as intended. Given these issues, a more careful approach would be taken when using software, ensuring all relevant updates have been completed and completing a setup checklist or validation process to confirm all relevant paths, files, and settings have been added accordingly. (Please refer to **Appendix A** for Testing Table).

7. Conclusion

7.1. The embedded system was successfully constructed using the PIC16F877A microcontroller, with each stage functioning as intended after testing and resolving issues that had been obstructing operation (as shown in Appendix A). Overall, the system met functional requirements and demonstrated effective state-based control, consistent with embedded design practices [7].

Future improvements should include implementing quality checklists during setup to ensure thorough software operation and correct file configuration. Additionally, the system could be enhanced by integrating temperature sensors for more precise temperature-driven controls and by upgrading the virtual terminal communication to more accurately simulate a modern vehicle's warning symbols. Adopting these recommendations would improve both the reliability and the representational fidelity of the embedded system.

References

- [1] R. Bosch GmbH, *Automotive Handbook*, 10th ed. Wiley, 2018.
<https://www.scribd.com/document/820482861/Bosch-10>.
- [2] *Microchip Technology Inc.*, “PIC16F877A Data Sheet,” 2023. “PIC16F87XA Devices Included in this Data Sheet: High-Performance RISC CPU.” Available:
<https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/39582C.pdf>
- [3] *Hitachi Ltd.*, “HD44780U LCD Controller/Driver Datasheet,” 2014.
<https://cdn.sparkfun.com/assets/9/5/f/7/b/HD44780.pdf>
- [4] *J. W. Valvano*, *Embedded Systems: Real-Time Interfacing*, 2017.
<https://www.scribd.com/document/1014101948/Embedded-Systems-Real-Time-Operating-Systems-for-Arm-Cortex-M-Microcontrollers-4th-Edition-Valvano-ebook-one-tap-reading>.
- [5] *Texas Instruments*, “DC Motor Driver Fundamentals,” *Application Report*, 2016.
- [6] *ISO 1028:1973 – Flowchart Symbols and Their Usage in Information Processing Documentation*, International Organization for Standardization, Geneva, Switzerland, 1973.
- [7] *M. Barr and A. Massa*, *Programming Embedded Systems*. 2006.
<https://www.oreilly.com/library/view/programming-embedded-systems/0596009836/>
- [8] *T. Finnerty and H. Strasser*, *Embedded Systems Architecture*, 2nd ed. Packt Publishing, 2023. [Online]. Available: <https://www.oreilly.com/library/view/embedded-systems-architecture/9781788832502/>
- [9] *F. Vahid and T. Givargis*, *Embedded System Design: A Unified Hardware/Software Introduction*. New York, NY, USA: Wiley, 2001.
https://books.google.com/books/about/Embedded_System_Design.html?id=0ZW6QgAACAAJ.

Appendices

Appendix A – Testing Table

Appendix A – Testing Table

Test Number	Test Description	Expected Outcome	Actual Outcome	Result	Resolution
1	Proteus and Hardware operation & stability.	Smooth-running software and PC system.	Unresponsive Proteus software.	Fail	Technical support to restore software operation.
2	Check LCD and system initialisation, “System Ready” Message.	LCD to display correct text after power up.	Display as expected.	Pass	-
3	Check SW1 input detection.	No operation until button is pressed and LED starts.	Correctly detected and LED starts flashing.	Pass	-
4	Start LED Pulse check.	LED pulses 3 times with appropriate intervals.	Intervals and pulses correct.	Pass	-
5	Check the LCD for the “Engine Started” message and the detection of SW2 input.	LCD to display correct text and wait for input.	Correct display and behaviour.	Pass	-
6	Coolant LED pulse check	LED pulses 3 times with appropriate intervals.	Intervals and pulses correct.	Pass	-
7	Check the LCD for the “Coolant OK” message and SW3 input detection.	LCD to display correct text and wait for input.	Correct display and behaviour.	Pass	-

8	Check the start of the motor/fan operation	Motor runs for 3s then stops.	Operated correctly and for the correct duration.	Pass	-
9	Check the Alert LED sequence and LCD for "Alert!" message.	LCD to display the correct message and LED to pulse 3 times with appropriate intervals	LCD displayed correctly. Intervals and pulses correct.	Pass	-
10	Check the "Shutdown Done" message and the final loop.	LCD to display the correct message and system to enter idle state.	LCD displayed correctly, and idle state achieved	Pass	-
11	Check Virtual terminal output.	Each serial message to appear correctly.	No messages or output at all.	Fail	UART hex files not configured correctly. Files added and resolved.
12	Full Cycle test.	System progresses through all stages smoothly	Completed successfully.	Pass	Success due to resolving issues early on.