

Gareth John_ACDA4013

by Gareth John

Submission date: 14-Apr-2026 10:14AM (UTC+0100)

Submission ID: 279730170

File name: 57495_Gareth_John_Gareth_John_ACDA4013_728569_178283275.docx (1.11M)

Word count: 2026

Character count: 10733



ACDA4013 EMBEDDED C

B.Eng. (Hons) in Electrical and Electronics Engineering

Gareth E John 2413461

April 17, 2026

Abstract

This report details the design and simulation of an embedded control system using the PIC16F877A microcontroller programmed in Embedded C within the CCS environment. The system simulates engine control and monitoring using digital inputs and outputs, with push buttons representing ignition, coolant, and fan controls, and outputs including LEDs, a DC motor, and an LCD for feedback. The microcontroller processes inputs to generate outputs such as status messages on the LCD and UART communication.

Proteus software was used for circuit testing, confirming the system's proper response to inputs and execution of control sequences, demonstrating practical applications of embedded systems in control and monitoring through digital I/O, serial communication, and structured programming.

Contents

1.0 Introduction 3

2.0 Background Research..... 4

 2.1 Software..... 4

 2.2 Components..... 4

3.0 Circuit Design 9

4.0 Flowchart 10

5.0 Files Structure 11

5.0 Simulation Testing and Screenshots..... 12

 5.1 Initializing..... 12

 5.2 Engine Started..... 13

 5.3 Coolant Activated..... 14

 5.4 Fan Activated 15

7.0 Conclusion..... 18

8.0 References 19

9.0 Appendices..... **Error! Bookmark not defined.**

1.0 Introduction

This report presents the design, development, and simulation of an embedded control system based on the PIC16F877A, programmed using Embedded C within the CCS compiler environment. The system demonstrates how a microcontroller can monitor digital inputs and control multiple outputs.

The project focuses on the interaction between hardware and software. Push buttons simulate conditions such as engine ignition, coolant activation, and fan operation. The microcontroller processes these inputs to control outputs, including LEDs, a DC motor, and an LCD.

The system also uses UART communication to transmit messages to a virtual terminal for monitoring. This report outlines the background research, circuit design, program flow, and simulation results to demonstrate the implementation of an embedded control system.

2.0 Background Research



This section explains each component and its purpose.

2.1 Software

Proteus

Proteus is a simulation software for building circuits before manufacturing to make sure it works correctly otherwise, it'll be very expensive to produce. This should have all the components that can be built as a product[1].

CCS (Custom Computer Services)

CCS is an embedded C compiler software for programming PIC microcontrollers. It had built-in functions that allowed control of the components included within the project[2].

2.2 Components



PIC Microcontroller

The PIC microcontroller PIC16F877A is the brain of the system that controls anything that has been programmed into the chip[3]. 5 Ports on the chip can do different tasks depending on which pin is used, from digital inputs or outputs or analogue inputs.

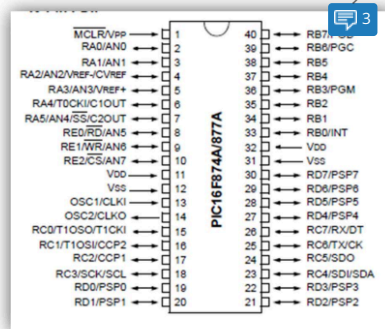


Figure 1- Shows the pinouts on the PIC[1, p3]

Clock Circuit

Figure 2 shows the design of a clock circuit that is made up of a crystal and 2 Capacitors. The quartz crystal oscillator provides a stable 20MHz clock signal for the microcontroller[5].

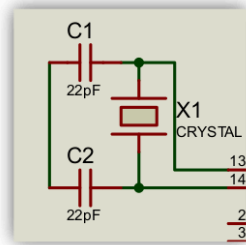


Figure 2- Shows the design of the clock circuit

Light Emitting Diode (LED)

Figure 3 below shows a different colour LED connected with a resistor. Each circuit is connected to a digital output.

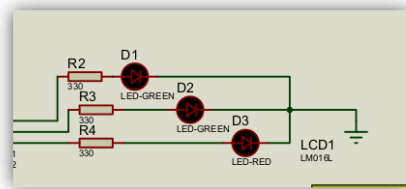


Figure 3- Shown a 3-LED circuit

Push button Circuit.

This push-button circuit with a low-resistance resistor is called a pull-down resistor[5,pp. 66-68]. There is a DC supply to the button, then when the button is pressed, the current flows through to the microcontroller. The primary role of the resistor is to force the signal to a known low (ground/0V) when the switch is open. Without the resistor, the signal can float, given the microcontroller's fault reading and assume the button was pressed.

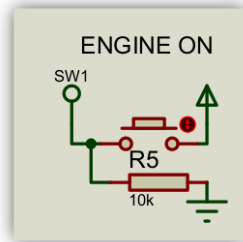


Figure 4-Shown a button circuit.

Liquid Crystal Display (LCD)

The LCD model used is a LMD16L, this device can display 2 lines of 16 characters maximum.

As shown below, the LCD only uses 4 bits from the microcontroller on D4 to D7, D0 to D3 go to ground in case of any interference from any signals[6,pp. 512].

RV1 is a potentiometer, which is important for LCD as it controls the contrast on the display. The VEE (voltage output) pin sets the contrast voltage. If the VEE connection shown in Figure 5 is adjusted closer to the VDD, the brighter the display will be, and if the VEE is adjusted closer to ground, then the darker the screen will be.

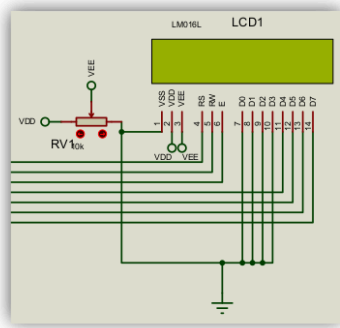


Figure 5-Shows a circuit design of an LCD

Motor

This symbol in Figure 7 is a motor that is used to activate a fan. This motor can be used clockwise or anticlockwise, depending on the signal the microcontroller sends.

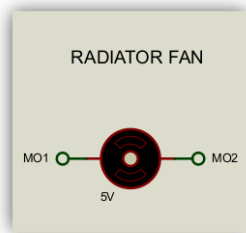


Figure 6-Shows a motor symbol

Virtual terminal using Modbus Protocol

A virtual terminal is a serial communication interface that allows Modbus protocol using UART. This is connected to the microcontroller on a 2-wire cable, 1 is for TX (transmit signal) and the other is RX (Receive signal). Some people get the wiring wrong as it needs to cross connect them on the opposite side of the cable. So once the Microcontroller sends data over TX, then on the virtual terminal it uses the RX to receive the data and vice versa.

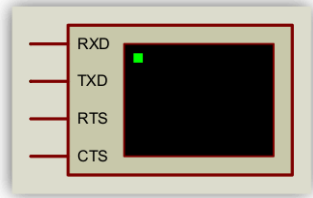


Figure 7- shows a Virtual terminal

3.0 Circuit Design

The circuit is made up of 3 digital input push switch signals, four Output signals, 3 LEDs, 1 DC motor, LCD screen, UART Modbus, all controlled by a microcontroller shown in Figure 1 above. Below is a list of digital input and output.

Digital Input

- SW1 is to imitate the engine ignition.
- SW2 is to trigger the cooling sensor.
- SW3 is to activate the radiator fan.

Digital Output

- Motor had 2 outputs (MO & MO1) to generate a clockwise movement.
- LED1 is a green light to show the engine is on.
- LED2 is a green light to show the cooling system is activated.
- LED3 is a red light to show the warning signal.
- LCD screen uses 4 outputs.

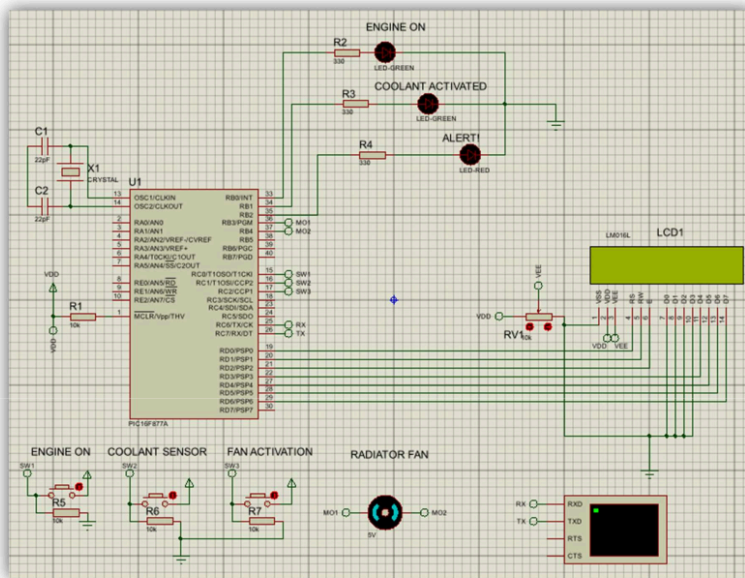


Figure 8- Circuit design

4.0 Flowchart

A flowchart is used to demonstrate in a simpler form to explain the direction and what the program does.

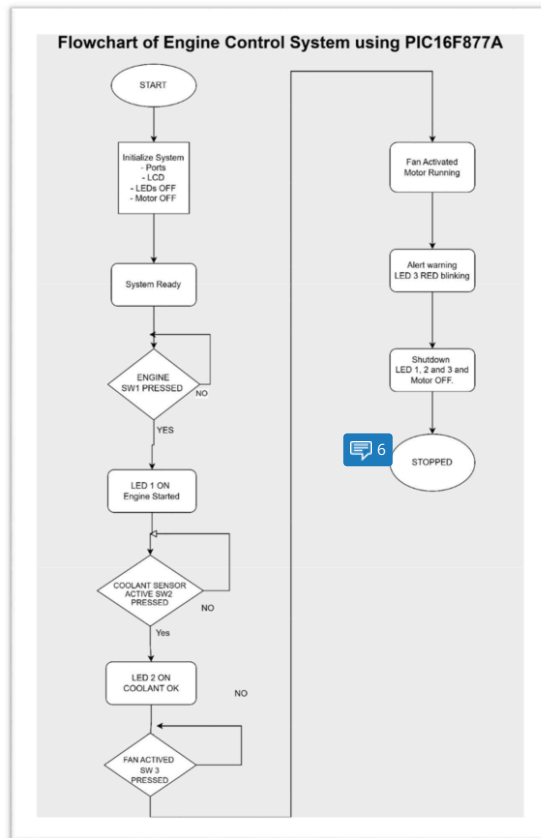


Figure 9 - Show a flow chart on how the program worked.

5.0 Files Structure

This section will explain step by step how the codes will control each light, switch, motor and message.

Before testing the code, it will need to be compiled using CCS software. It's important that all the files from the CCS.[2] and Proteus [1] They are in the same folder structure. Once the embedded C code is completed and without error, it will need to be compiled, ready to be used in Proteus. Figure 11 shows where the two main .c and .h files are added to Proteus.

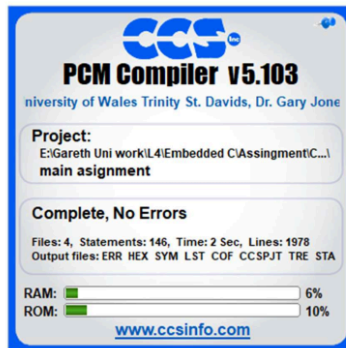


Figure 10 - PCW - CCS program[2].

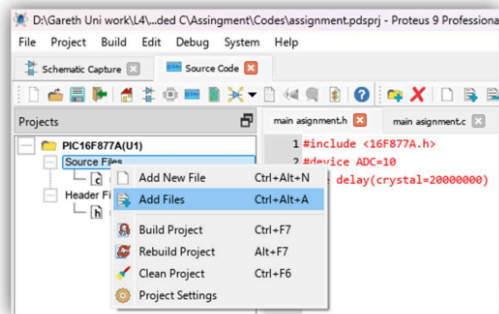


Figure 11- Adding files to the Proteus source code.

5.0 Simulation Testing and Screenshots

This section will demonstrate how each part of the CCS C code works and how it tells each component what to do.

5.1 Initializing

Figure 12 below shows a schematic design of how the software is running. Once the software is running, start the initialisation process of the code. Once that is done, then the LCD and Virtual Terminal will say 'System Ready'.

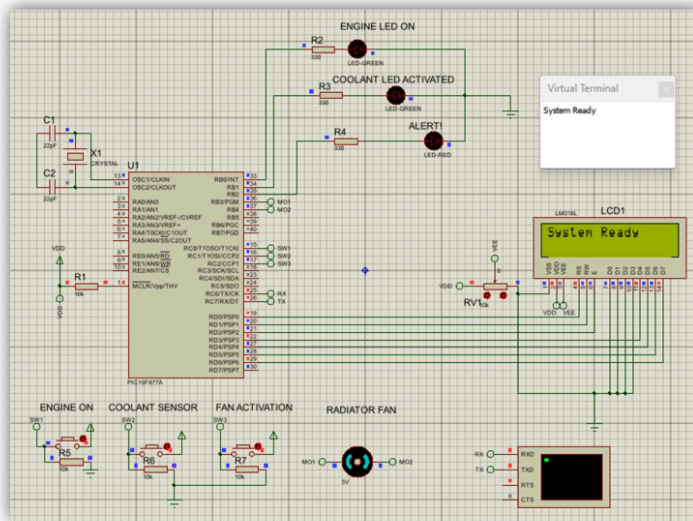


Figure 12- Shown a schematic when Proteus is running.

Figure 9 below shows the initialisation process of the code. The microcontroller configures each pin as input or output[8], as shown below. Line 35 sets RB0 to RB5 as Output. The output is to 3 LEDs and 2 for a motor. Line 38 is to set Port C as inputs, RC0 to RC2 are used for the switches. It is worth mentioning that RC6 and RC7 are used for serial connection using RS232, but why does it work as it is set as input? This is because line 5 is used to set the RS232 on RC6 and RC7.

Line 40 is to set Port D as outputs to the LCD screen. Then the controller also needs to know what state the LED and Fan motor need to be in, as shown in Figure 9, they are all set to low. Then the final two codes are to set the LCD and UART terminal to display 'System Ready'. The '\f' command clears the display, while '\r\n' moves the cursor to a new line.

```

31
32 void main(){
33
34 //PORTB
35 set_tris_b(0b11111000); // Set RB0-RB3 as Outputs
36
37 // PORT C All RC pins as inputs (SW1-SW3 use RC0-RC2)
38 set_tris_c(0b11111111);
39 // PORT D All RD pins as outputs
40 set_tris_d(0b00000000);
41 lcd_init(); // Initialize LCD once
42
43 // Initialize motor outputs LOW (motor off)
44 Output_low(MOTOR_IN1);
45 Output_low(MOTOR_IN2);
46
47 // Initialize all LED outputs LOW (LEDS off)
48 Output_low(LED_1);
49 Output_low(LED_2);
50 Output_low(LED_3);
51 lcd_puts("System Ready"); // Output Message on LCD
52 printf("System Ready\r\n"); // Output Message on UART
53 // Start engine

```

Figure 13-Shows the initialization code.

5.2 Engine Started

Figure 14 shows LED D1 turning on when the engine is activated. The switch also initiates a message 'Engine Started' displayed on the LCD.

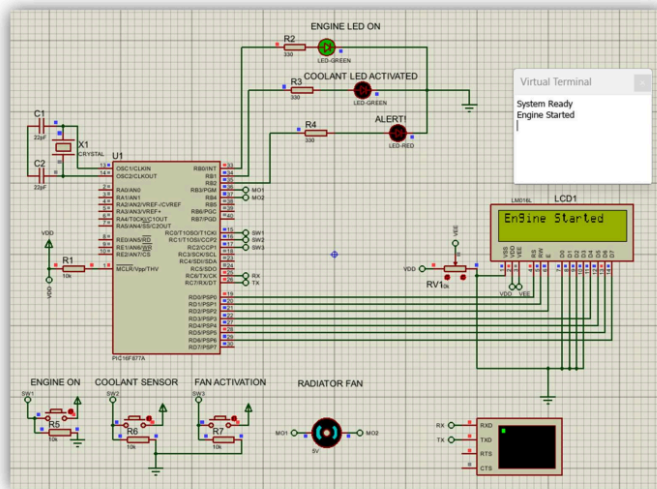


Figure 14- Shows the Schematic design after the Engine is turned on.

```

53 // Start engine
54 While (input(SW1));
55 output_high(LED_1);           // Green LED 1 ON
56
57 Delay_ms(50);                // Debounce timer
58 lcd_putc("\nEngine Started"); // Output Engine Started Message on LCD
59 Printf("Engine Started\r\n"); // Output Engine Started Message on UART
60

```

Figure 15- Shows the Code how the process worked from SW1

5.3 Coolant Activated

Figure 17 shows a live schematic design after the coolant sensor 'Switch' is activated. This then turned on LED D2 to show that it's activated by using the high code on line 62 in Figure 18. Then a message is sent to the LCD and terminal stating Coolant OK.

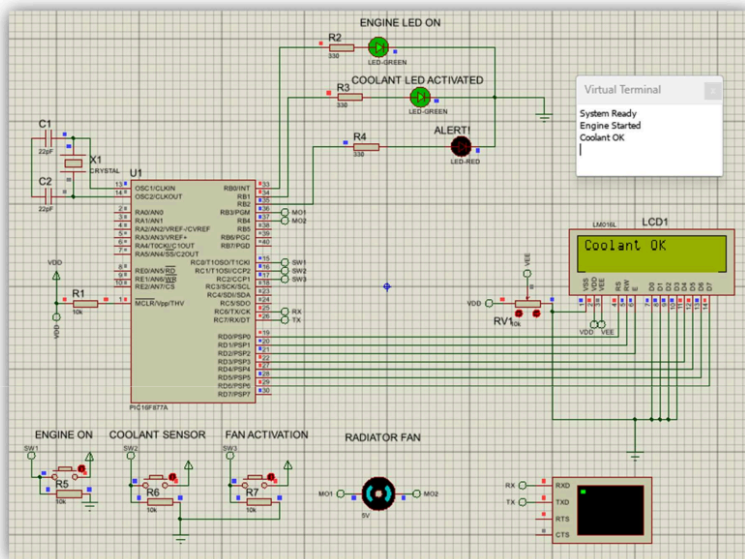


Figure 16- shows the schematic design when the coolant sensor is activated.

```

60
61 while (input(SW2));           // Coolant Sensor Trigger
62 output_high(LED_2);          // Green LED 2 ON
63 lcd_putc("\fcoolant OK");    // Output Coolant OK Message on LCD
64 printf("Coolant OK\r\n");    // Output Coolant OK Message on UART
65

```

Figure 17-Shows the code to process how the Sensor turned on the LED

5.4 Fan Activated

When switching the Fan activation, this will activate the Fan (DC Motor) outputting from PIN RB3, which then goes to MO1, and RB4 goes to MO2, as shown in Figure 18. Line 71 on Figure 21 shows that MO1 is high, making the Motor turn clockwise because MO2 is Low. If it were the other way around, this would make the motor turn anticlockwise, and the LCD and UART would display Fan Activated and Motor Running.

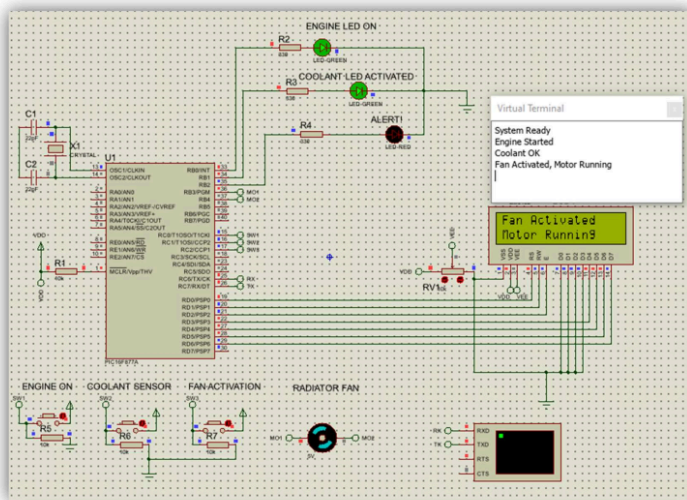


Figure 18- Shows a design after the Fan Switch is activated.

After the Fan Switch is activated, the motor will be running for 3 seconds until an Alert! Is displaying on the LCD and UART. The red LED in Figure 19 will blink 3 times with a delay of 0.1 seconds, which is driven by the CCS loop[9,pp. 13] code in Figure 21, lines 80 to 86. This improves code efficiency and readability.

Once the Blinking stopped, it then goes to a shutdown procedure by turning the motor off on a low code Figure 21, lines 90 and 91, also turning all LEDs off, showing low on lines 92 and 93. A final message on the UART and LCD stating SHUTDOWN DONE.

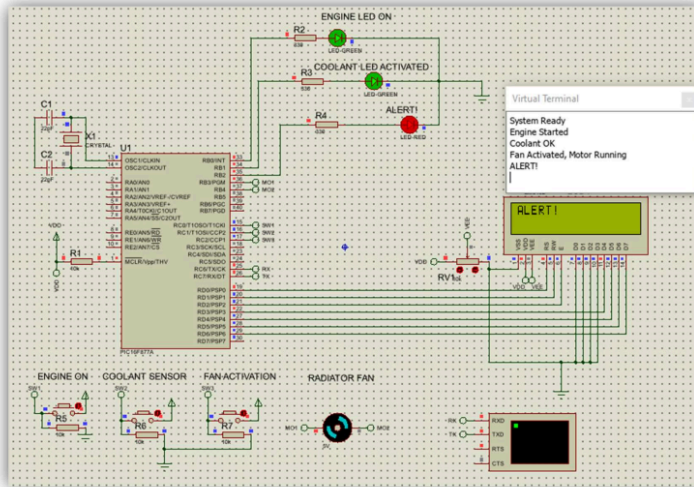


Figure 19- Shows when the Warning is activated.

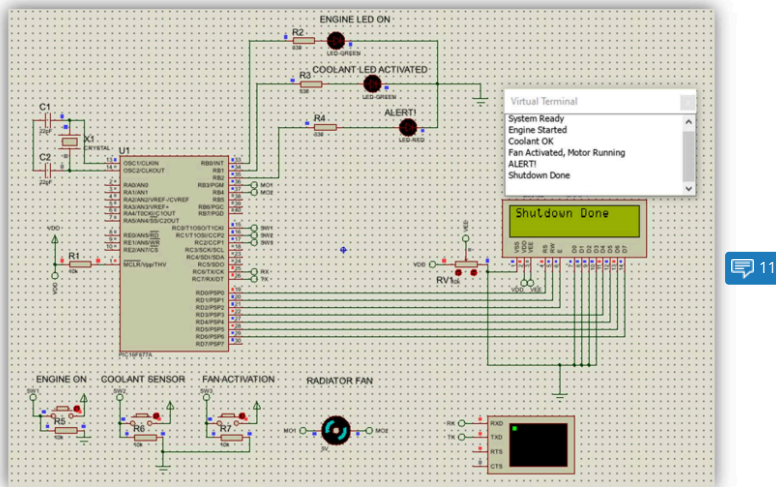


Figure 20- Shows when the shutdown procedure happened.

```

66 // Fan activated
67 while (!input(SW3));
68 lcd_puts("\xFFan Activated\nMotor Running"); // Output Fan Activated on 1st Line and Motor Running on 2nd Line Message on LCD
69 printf("Fan Activated, Motor Running\n"); // Output Fan Activated Message on UART Output Motor Running on UART
70 output_high(MOTOR_IN1); // Fan running Clockwise
71 output_low(MOTOR_IN2);
72
73 delay_ms(3000); // 3 seconds delay
74 lcd_puts("\fALERT!"); // Output ALERT! Message on LCD
75 printf("\fALERT!\n"); // Output ALERT! Message on UART
76
77 // RED LED 3 blink 3 times with delay on and off 100ms
78 for(int i = 0; i < 3; i++) {
79     output_high(LED_3);
80     delay_ms(100);
81     output_low(LED_3);
82     delay_ms(100);
83 }
84
85 delay_ms(1000);
86 output_low(MOTOR_IN1); // Fan Stop
87 output_low(MOTOR_IN2);
88 output_low(LED_2); // Output RED LED 2 OFF
89 output_low(LED_3); // Output RED LED 2 OFF
90 lcd_puts("\fShutdown Done"); // Output Shutdown Done Message on LCD
91 printf("\fShutdown Done\n"); // Output Shutdown Done Message on UART
92
93 while (TRUE) {
94 }
95 }

```

Figure 21- Shows the CCS code for the Fan and Shutdown.

7.0 Conclusion 13

This project demonstrated how a PIC16F877A microcontroller can be programmed using Embedded C within the CCS environment to control and monitor a system. The integration of digital inputs, outputs, and communication interfaces enabled the simulation of an engine cooling and warning application.

Push-button inputs were used to trigger system states such as engine start, coolant activation, and fan operation. These inputs controlled outputs including LEDs, a DC motor, and an LCD, providing clear system feedback. UART communication also enabled real-time monitoring through a virtual terminal.

Simulation in Proteus Design Suite confirmed that the system operated as intended. The use of structured programming techniques improved code efficiency and readability. Overall, the project demonstrates key embedded system concepts, including input/output control, timing, and serial communication [10].

8.0 References

- [1] 'Circuit Simulation Software with SPICE'. Accessed: Apr. 04, 2026. [Online]. Available: <https://www.labcenter.com/simulation/>
- [2] J. W. Valvano, *Embedded Microcomputer Systems: Real Time Interfacing*. Cengage Learning, 2011.
- [3] 'PCB Design and Circuit Simulator Software - Proteus'. Accessed: Apr. 03, 2026. [Online]. Available: <https://www.labcenter.com/>
- [4] 'PCW IDE Compiler for Microchip PIC10/12/16 Devices - CCS, Inc.'. Accessed: Apr. 03, 2026. [Online]. Available: https://www.ccsinfo.com/product_info.php?products_id=PCW_full
- [5] 'PIC16F87XA Data Sheet', p. 234.
- [6] F. Vahid and T. Givargis, 'Platform tuning for embedded systems design', *Computer*, vol. 34, no. 3, pp. 112–114, Mar. 2001, doi: 10.1109/MC.2001.910903.
- [7] 'Programming 8-bit PIC Microcontrollers in C : With Interactive Hardware Simulation - EBSCO'. Accessed: Apr. 04, 2026. [Online]. Available: <https://research-ebSCO-com.uwtsd.idm.oclc.org/c/onvpsy/ebook-viewer/pdf/qzZbkww6er?location=https%25253A%25252F%25252Fresearch-ebSCO-com.uwtsd.idm.oclc.org%25252Fc%25252Fonvpsy%25252Fsearch%25252Fdetails%25252FqzZbkww6er%25253Fdb%25253Dnlebk>
- [8] 'Programming 8-bit PIC Microcontrollers in C : With Interactive Hardware Simulation - EBSCO'. Accessed: Apr. 04, 2026. [Online]. Available: https://research-ebSCO-com.uwtsd.idm.oclc.org/c/onvpsy/ebook-viewer/pdf/qzZbkww6er/page/pp_10?location=https%25253A%25252F%25252Fresearch-ebSCO-com.uwtsd.idm.oclc.org%25252Fc%25252Fonvpsy%25252Fsearch%25252Fdetails%25252FqzZbkww6er%25253Fdb%25253Dnlebk
- [9] P. Horowitz and W. Hill, *The art of electronics*, Third edition, 21st printing with corrections. Cambridge, New York: Cambridge University Press, 2024.
- [10] D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, 'UNIX time-sharing system: The C programming language', *The Bell System Technical Journal*, vol. 57, no. 6, pp. 1991–2019, Jul. 1978, doi: 10.1002/j.1538-7305.1978.tb02140.x.

72/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1 | Engagement with Literature(10)

It is also necessary to include more information about the cooling content in the introduction or background review, explaining the general concepts, where it is commonly used, and the latest versions or cooling technologies currently adopted in industries.



Comment 2 | Engagement with Literature(10)

The task should be critically analysed in the literature review by examining how similar implementations have been carried out by others and comparing them with your own work. This analysis should include a critical comparison of journal articles and other reference books to evaluate the differences, similarities, and effectiveness of the approaches used.



Comment 3 | Cognitive & Intellectual (25)

The choice of the PIC series needs to be justified with relevant discussion of its importance in terms of power efficiency, performance, and architecture, which is appropriate for the application.

PAGE 6

PAGE 7

PAGE 8

PAGE 9



Comment 4 | Knowledge & Understanding(20)

It is good to see the detailed information about the LCD, and similarly, details of the UART such as the baud rate and other parameters could also be included.

PAGE 10



Comment 5 | Knowledge & Understanding(20)

The timing justification used in the code needs to be discussed and explained in greater detail.

PAGE 11



Comment 6 | Cognitive & Intellectual (25)

The pseudocode is included, which is appreciated, though additional explanation would further improve clarity.

PAGE 12



Comment 7 | Knowledge & Understanding(20)

CCS C program constructs such as header files, pin definitions are seen it could be better if the main structure, control statements, I/O configuration, and any functions used is explained on how the program is structured.

PAGE 13



Comment 8 | Transferable Skills (15)

A sufficient amount of testing evidence is presented in the report, which is appreciable.

PAGE 14



Comment 9 | Transferable Skills (15)

The code is readable, and a sufficient explanation of the code linked with the I/O operations of the PIC microcontroller is provided.

PAGE 15

PAGE 16



Comment 10 | Knowledge & Understanding(20)

It is observed that the sequencing has been explained; however, more detailed points and explanations need to be included.

PAGE 17

PAGE 18



Comment 11 | Practical Skills (30)

Practical Observations

The system is designed as expected and operates as intended. However, it was observed that the motor continues to run even after the alert is triggered three times, which differs slightly from the expected behaviour and can be considered a minor issue. Aside from this, the system shuts down in a proper sequence and functions effectively. No additional innovation is evident in the design, and incorporating some enhancements could further improve it.



Comment 12 | Transferable Skills (15)

The structure and clarity are good; however, there are many ways to improve the work further by providing a more detailed explanation of the project and the selection of the microcontroller in the background review.

PAGE 19



Comment 13 | Cognitive & Intellectual (25)

It is also essential to include the limitations of the design and discuss how the design could be further improved in the future.

PAGE 20

GRADING FORM: EMBEDDED C

GARETH JOHN

72.05

ENGAGEMENT WITH LITERATURE(10)

Research supports embedded automotive cooling logic and PIC-based simulation. Uses scholarly/technical sources and correct referencing.



Some relevant sources but weak synthesis; referencing inconsistent.

5.25

KNOWLEDGE & UNDERSTANDING(20)

Demonstrates knowledge of sequencing logic, timers/delays, I/O interfacing, LCD 4-bit mode, UART monitoring and real-world cooling workflow.



Good understanding with brief explanations

14.5

COGNITIVE & INTELLECTUAL (25)

Evaluates design approach and makes justified decisions using logical reasoning. Demonstrates structured thinking and reflective judgement.



Some reasoning shown; limited justification and weak evaluation of alternatives/limitations.

15

PRACTICAL SKILLS (30)

Implements and tests a PIC16F877A Proteus simulation meeting : switch sequencing, LED states, fan motor control, alert blink, LCD and UART messaging.



Mostly functional system; Core requirements met; good integration and testing evidence.

24

TRANSFERABLE SKILLS (15)

Communicates work clearly and professionally through a structured report and well-documented code. Submits required files and evidence reliably.



Highly professional report and submission package; clear structure, strong communication, excellent code style and documentation.

13.3