Technical Project Report

Firmware Analysis for PIC16F877A Based Automatic Transfer Switch (ATS) Controller

PhotoNode Electronics
Project Assistant Service 2025
`Contact: +94 71 738 3208`

To:	Heshan Lasantha
Contact:	+94 78 505 6008

Project Name:	ATS Prototype with PIC16f877A
Start Date:	07 August 2025
End Date:	18 August 2025
Document Version:	1.0 (Final)

September 1, 2025

Contents

Chapter 1

Introduction

1.1 Project Overview

This document provides a comprehensive technical analysis of the firmware developed for an Automatic Transfer Switch (ATS) controller, based on the Microchip PIC16F877A microcontroller. The primary function of this ATS system is to monitor the main power supply (referred to as CEB - Ceylon Electricity Board) and automatically switch the load to a backup generator in the event of a power failure. When the main power is restored, the system safely transfers the load back to the CEB and shuts down the generator after a cooldown period.

1.2 Key Features of the Firmware

The developed C code represents a robust and reliable solution designed for continuous, unattended operation. The key features include:

- **State Machine Architecture:** The core logic is built around a finite state machine, which ensures predictable, safe, and orderly transitions between different operational modes.
- **User-Friendly Interface:** A 16x2 character LCD provides real-time status updates, system states, and countdown timers, enhancing user observability and diagnostics.
- **Configurable Timers:** All critical timing parameters (e.g., crank time, warmup, cooldown) are defined as constants, allowing for easy adjustment and tuning.
- **Audible Alarms:** An integrated buzzer provides audible alerts for critical events such as power failure and system faults.
- **Self-Healing Capability:** This is the most advanced feature of the firmware. If the system enters a critical fault state (e.g., generator failure to start or run), it continuously monitors for the return of CEB power. Upon restoration of the main supply, the system automatically recovers from the alarm state and transfers the load, ensuring power is restored to the load without manual intervention.

1.3 Purpose of this Document

The purpose of this report is to serve as a detailed technical reference for the client, Heshan Lasantha. It breaks down the code into logical sections, explaining the hardware configuration, software architecture, and the operational flow of the ATS state machine. This will aid in future maintenance, modification, and understanding of the system's behavior.

Chapter 2

Hardware and System Configuration

2.1 Microcontroller Configuration

The firmware begins by setting the configuration bits for the PIC16F877A microcontroller. These are crucial for the fundamental operation of the chip.

```
#pragma config FOSC = HS, WDTE = OFF, PWRTE = OFF, BOREN = ON, LVP = OFF,
    CPD = OFF, WRT = OFF, CP = OFF
```

Listing 2.1: Configuration Bits

- `FOSC = HS`: Configures the oscillator for High-Speed crystal mode, suitable for the 8MHz crystal defined by `_XTAL_FREQ`.
- `WDTE = OFF`: Disables the Watchdog Timer. For a final product, this might be enabled for extra safety, but is disabled here for development simplicity.
- `PWRTE = OFF`: Disables the Power-up Timer.
- `BOREN = ON`: Enables Brown-out Reset, which resets the MCU if the supply voltage drops below a certain threshold, preventing erratic behavior.
- `LVP = OFF`: Disables Low-Voltage Programming, freeing up the RB3 pin for general I/O.
- `CPD, WRT, CP = OFF`: Disables data and program code protection.

2.2 Hardware Pin Definitions

The `#define` directives map physical microcontroller pins to logical names used throughout the code, improving readability and maintainability.

```
#define RS RD2
#define EN RD3
#define D4 RD4
#define D5 RD5
#define D6 RD6
#define D7 RD7
#define BUZZER          RD1
#define GEN_ON_INPUT    RA1
```

```
#define CEB_ON_INPUT       RA0
#define GEN_RELAY          RB0
#define CEB_RELAY          RB1
#define GEN_START_RELAY    RB2
```

Listing 2.2: Hardware Pin Definitions

The pinout is summarized in the table below:

Symbolic Name	**MCU Pin**	**Function**
CEB_ON_INPUT	RA0	Senses presence of CEB mains power.
GEN_ON_INPUT	RA1	Senses if the generator is running and producing power.
GEN_RELAY	RB0	Controls the contactor/relay for generator power.
CEB_RELAY	RB1	Controls the contactor/relay for CEB power.
GEN_START_RELAY	RB2	Controls the generator's start/crank motor relay.
BUZZER	RD1	Activates the audible alarm.
RS, EN, D4-D7	RD2-RD7	Control and data lines for the 16x2 LCD in 4-bit mode.

Table 2.1: Pinout Configuration

Chapter 3

Software Architecture

The firmware is built upon a robust, event-driven architecture centered around a state machine. This design pattern is ideal for control systems as it ensures the controller is always in a well-defined state and transitions between states are handled in a controlled manner.

3.1 Main Program Flow

The program execution follows a simple but effective structure:

1. **Initialization:** The `main()` function first calls `System_Init()` to configure microcontroller peripherals (I/O ports, ADC, comparators) and the LCD. It also displays a startup splash screen.

2. **Infinite Loop:** The program then enters an infinite `while(1)` loop, which forms the heart of the application.

3. **Core Tasks:** Inside the loop, three main tasks are executed sequentially every second:

   - `ATS_StateMachine_Run()`: Executes the logic for the current state.
   - `Update_Display()`: Updates the LCD with the current system status.
   - `__delay_ms(1000)`: A one-second delay which serves as the system's main "tick" or time base.
   - `if(timer_seconds > 0) timer_seconds--`: Decrements the global countdown timer if it is active.

3.2 State Machine Design

The core of the control logic is implemented as a finite state machine. The system can only be in one of the defined states at any given time. The state is stored in the global variable `currentState`.

```c
typedef enum {
    STATE_INIT, STATE_ON_CEB, STATE_CEB_FAILED, STATE_STARTING_GEN,
    STATE_GEN_WARMING_UP, STATE_ON_GENERATOR, STATE_CEB_RETURNED,
    STATE_GEN_COOLDOWN, STATE_GEN_START_FAIL, STATE_GEN_RUNTIME_FAIL
} ATS_State;
```

Listing 3.1: ATS State Definitions (enum)

- STATE_INIT: Initial (unused) state before the main loop begins.
- STATE_ON_CEB: Normal operation. Load is powered by CEB.
- STATE_CEB_FAILED: CEB power has been lost. A short delay timer is started to avoid switching on transient faults.
- STATE_STARTING_GEN: The generator cranking cycle is in progress.
- STATE_GEN_WARMING_UP: The generator has started. A warmup period is observed before applying the load.
- STATE_ON_GENERATOR: Normal backup operation. Load is powered by the generator.
- STATE_CEB_RETURNED: CEB power has been restored. A stabilization delay is observed before transferring back.
- STATE_GEN_COOLDOWN: The load is back on CEB. The generator runs without load to cool down.
- STATE_GEN_START_FAIL: A critical alarm state. The generator failed to start within the allotted time.
- STATE_GEN_RUNTIME_FAIL: A critical alarm state. The generator stopped unexpectedly while powering the load.

3.3 Timing Constants

All time-based delays are managed using pre-defined constants and a single global timer variable, timer_seconds. This approach makes the system's timing behavior easy to configure and understand.

```
#define CEB_FAIL_DELAY         3        // seconds
#define GEN_START_CRANK_TIME     7        // seconds
#define GEN_WARMUP_TIME          10       // seconds
#define CEB_RETURN_DELAY       15       // seconds
#define GEN_COOLDOWN_TIME        30       // seconds
```

Listing 3.2: Timing Definitions

Chapter 4

Detailed Code Analysis

This chapter provides a function-by-function breakdown of the ATS controller firmware.

4.1 LCD Driver

The code includes a self-contained driver for a standard HD44780-compatible LCD operating in 4-bit mode. This reduces external dependencies.

- `Lcd_Init()`: Initializes the LCD controller, setting it to 4-bit mode, 2-line display, and turning the display on with the cursor off.
- `Lcd_Cmd()`/`Lcd_Send_Full_Byte_Cmd()`: Functions to send commands to the LCD (e.g., clear screen, set cursor position).
- `Lcd_Write_Char()`/`Lcd_Write_String()`: Functions to write character data to the LCD screen.
- `Create_Custom_Chars()`: A utility function that loads custom character patterns (for CEB, Generator, and Arrow symbols) into the LCD's CGRAM. This enhances the user interface with intuitive icons.

4.2 The Main ATS State Machine: `ATS_StateMachine_Run()`

This function is the brain of the entire system. It is called once every second. Inside, a `switch` statement evaluates the `currentState` variable and executes the corresponding logic.

4.2.1 Normal Operation Cycle (CEB Fails and Returns)

1. **Case `STATE_ON_CEB`:** The system is stable on mains power. It continuously checks the `CEB_ON_INPUT`. If CEB power is lost (`!is_ceb_on`), it transitions to `STATE_CEB_FAILED`, activates a buzzer, and sets a 3-second timer.
2. **Case `STATE_CEB_FAILED`:** The system waits for the 3-second timer to elapse. This prevents a reaction to brief power flickers. If CEB returns during this time, it immediately reverts to `STATE_ON_CEB`. Otherwise, it transitions to `STATE_STARTING_GEN`.

3. **Case `STATE_STARTING_GEN`:** The `GEN_START_RELAY` is activated to crank the generator. The system monitors `GEN_ON_INPUT`. If the generator starts successfully, it deactivates the crank relay and moves to `STATE_GEN_WARMING_UP`. If the 7-second crank timer expires and the generator has not started, it transitions to the alarm state `STATE_GEN_START_FAIL`.

4. **Case `STATE_GEN_WARMING_UP`:** The generator is running but the load is not yet connected. This 10-second period allows the engine to stabilize. Once the timer expires, it moves to `STATE_ON_GENERATOR`.

5. **Case `STATE_ON_GENERATOR`:** The `GEN_RELAY` is activated, connecting the load to the generator. The system now monitors for two events:
   - The return of CEB power (`is_ceb_on`), which triggers a transition to `STATE_CEB_RETURNED`.
   - An unexpected generator shutdown (`!is_gen_on`), which triggers the alarm state `STATE_GEN_RUNTIME_FAIL`.

6. **Case `STATE_CEB_RETURNED`:** CEB power is back. A 15-second timer runs to ensure the mains supply is stable. If CEB fails again during this time, it returns to `STATE_ON_GENERATOR`. If the timer completes, it performs a ”break-before-make” transfer: disconnects the generator relay, pauses briefly, connects the CEB relay, and then transitions to `STATE_GEN_COOLDOWN`.

7. **Case `STATE_GEN_COOLDOWN`:** The load is on CEB, but the generator continues to run without load for 30 seconds to cool down properly. This extends engine life. Once the timer expires, the system returns to the stable `STATE_ON_CEB`.

4.2.2 Special Feature: The Self-Healing Logic

The most robust feature of this firmware is its ability to automatically recover from a critical generator fault. This logic is contained within the alarm states.

```
// --- SELF-HEALING LOGIC IS HERE ---
case STATE_GEN_START_FAIL:
case STATE_GEN_RUNTIME_FAIL:
    // Check for the recovery condition: Has CEB power returned?
    if (is_ceb_on) {
        // YES! Recover the system automatically.
        BUZZER = 0; // Stop the alarm.
        Buzzer_Beep(250); // Short beep to signal recovery.
        currentState = STATE_ON_CEB; // Go to the safe CEB state.
    } else {
        // NO, CEB is still off. Continue the alarm.
        GEN_RELAY = 0; CEB_RELAY = 0; GEN_START_RELAY = 0;
        BUZZER = !BUZZER; // Intermittent beep
    }
    break;
// --- END OF SELF-HEALING LOGIC ---
```

Listing 4.1: Self-Healing Logic

Operational Scenario:

1. CEB power fails.

2. The ATS attempts to start the generator, but it fails (e.g., out of fuel, mechanical fault).

3. The system enters STATE_GEN_START_FAIL. An audible alarm sounds, and the LCD displays "GEN START FAILED". The site is now completely without power.

4. **Without Self-Healing:** The system would remain in this alarm state indefinitely, requiring a manual reset. If CEB power returned, the load would remain disconnected until someone intervened.

5. **With Self-Healing:** While in the alarm state, the code *continues to check for the return of CEB power* once per second.

6. As soon as is_ceb_on becomes true, the system immediately:

   - Silences the continuous alarm.
   - Emits a short beep to indicate recovery.
   - Transitions directly to STATE_ON_CEB.
   - In the next cycle, the STATE_ON_CEB logic will engage the CEB_RELAY, restoring power to the load.

This intelligent recovery mechanism ensures that the load is never left powerless if a viable power source (CEB) becomes available, even if the backup system has failed. It transforms a critical failure into a temporary outage, significantly enhancing system reliability.

4.3 Display Update Function: `Update_Display()`

This function is responsible for all output to the 16x2 LCD. It is optimized to prevent screen flickering by only redrawing the entire screen when the system's state changes. It maintains a previousDisplayState variable for this comparison. For states with active timers, it efficiently updates only the timer digits on the screen each second, leaving the rest of the text intact.

Chapter 5

Conclusion and Recommendations

5.1 Conclusion

The firmware developed for the PIC16F877A ATS controller is a complete, robust, and reliable solution. Its state machine architecture ensures predictable and safe operation through all phases of power transfer. The user interface provides clear and concise status information, and the configurable timing parameters allow for easy adaptation to different generator models and site requirements.

The standout feature, the ”self-healing” logic, elevates this design from a standard ATS controller to a highly resilient power management system. By enabling automatic recovery from generator faults upon the return of mains power, it guarantees maximum uptime for the connected load and minimizes the need for manual intervention. The project successfully meets all the specified requirements and provides a solid foundation for a commercial-grade product.

5.2 Future Recommendations

While the current firmware is fully functional, the following enhancements could be considered for future versions:

- **Engine Parameter Monitoring:** Integrate sensors to monitor generator fuel level, oil pressure, and engine temperature. The system could then provide pre-emptive warnings or refuse to start the generator under unsafe conditions.
- **Multiple Crank Attempts:** Modify the start logic to attempt cranking the generator 2-3 times with short pauses in between, which can overcome minor starting issues.
- **Data Logging:** Implement logging of events (power failures, generator starts, faults) to an external EEPROM. This data would be invaluable for diagnostics and maintenance history.
- **Remote Communication:** Add a serial (RS232/RS485) or wireless (GSM/Wi-Fi) module to enable remote monitoring of the ATS status and receive SMS alerts for critical events.

Appendix A

Full Source Code

```c
/*
 * File:   ATS_Controller_Self_Healing.c
 * Author: Sahan (UI Enhanced by AI)
 *
 * Description: The complete and final ATS controller for PIC16F877A.
 * This is the most robust version with a "self-healing" capability.
 * If the system is in an alarm state (due to generator failure), it will
 * automatically recover and transfer to CEB power if it becomes available.
 */

#define _XTAL_FREQ 8000000

#include <xc.h>
#include <stdio.h>

//
    ============================================================================

// CONFIGURATION BITS
//
    ============================================================================

#pragma config FOSC = HS, WDTE = OFF, PWRTE = OFF, BOREN = ON, LVP = OFF,
    CPD = OFF, WRT = OFF, CP = OFF

//
    ============================================================================

// HARDWARE PIN DEFINITIONS
//
    ============================================================================

#define RS RD2
#define EN RD3
#define D4 RD4
#define D5 RD5
#define D6 RD6
#define D7 RD7
#define BUZZER           RD1
#define GEN_ON_INPUT     RA1
#define CEB_ON_INPUT     RA0
#define GEN_RELAY        RB0
```

```
#define CEB_RELAY         RB1
#define GEN_START_RELAY   RB2

// ============================================================================
// LCD DRIVER (Integrated)
// ============================================================================
void Lcd_Port(char a);
void Lcd_Cmd(char a);
void Lcd_Clear();
void Lcd_Set_Cursor(char a, char b);
void Lcd_Init();
void Lcd_Write_Char(char a);
void Lcd_Write_String(char *a);
void Lcd_Send_Full_Byte_Cmd(char cmd);

void Lcd_Port(char a) {
  if(a & 1) D4=1; else D4=0; if(a & 2) D5=1; else D5=0;
  if(a & 4) D6=1; else D6=0; if(a & 8) D7=1; else D7=0;
}
void Lcd_Cmd(char a) { RS=0; Lcd_Port(a); EN=1; __delay_ms(4); EN=0; }
void Lcd_Send_Full_Byte_Cmd(char cmd) { Lcd_Cmd(cmd >> 4); Lcd_Cmd(cmd & 0x0F); }
void Lcd_Clear() { Lcd_Send_Full_Byte_Cmd(0x01); __delay_ms(2); }
void Lcd_Set_Cursor(char a, char b) {
  char temp = (a == 1) ? (0x80 + b - 1) : (0xC0 + b - 1);
    Lcd_Send_Full_Byte_Cmd(temp);
}
void Lcd_Init() {
    Lcd_Port(0x00); __delay_ms(20); Lcd_Cmd(0x03); __delay_ms(5); Lcd_Cmd(0x03);
    __delay_ms(11); Lcd_Cmd(0x03); Lcd_Cmd(0x02); Lcd_Send_Full_Byte_Cmd(0x28);
    Lcd_Send_Full_Byte_Cmd(0x0C); Lcd_Send_Full_Byte_Cmd(0x06); Lcd_Clear();
}
void Lcd_Write_Char(char a) {
   RS=1; Lcd_Port(a >> 4); EN=1; __delay_us(40); EN=0;
   Lcd_Port(a & 0x0F); EN=1; __delay_us(40); EN=0;
}
void Lcd_Write_String(char *a) { for(int i=0; a[i]!='\0'; i++) Lcd_Write_Char(a[i]); }

// ============================================================================
// ATS APPLICATION LOGIC
// ============================================================================
#define CEB_FAIL_DELAY        3
#define GEN_START_CRANK_TIME    7
#define GEN_WARMUP_TIME         10
#define CEB_RETURN_DELAY      15
#define GEN_COOLDOWN_TIME       30
```

```c

typedef enum {
    STATE_INIT, STATE_ON_CEB, STATE_CEB_FAILED, STATE_STARTING_GEN,
    STATE_GEN_WARMING_UP, STATE_ON_GENERATOR, STATE_CEB_RETURNED,
    STATE_GEN_COOLDOWN, STATE_GEN_START_FAIL, STATE_GEN_RUNTIME_FAIL
} ATS_State;

ATS_State currentState = STATE_INIT;
ATS_State previousDisplayState = -1;
unsigned int timer_seconds = 0;

const unsigned char ceb_char[8]   = {0x0E,0x1F,0x1F,0x0E,0x04,0x0E,0x11,0x0E};
const unsigned char gen_char[8]   = {0x00,0x1F,0x11,0x11,0x1F,0x1B,0x1B,0x00};
const unsigned char arrow_char[8] = {0x00,0x04,0x06,0x1F,0x06,0x04,0x00,0x00};
#define CEB_CHAR_CODE   0
#define GEN_CHAR_CODE   1
#define ARROW_CHAR_CODE 2

void System_Init(void);
void Create_Custom_Chars(void);
void Buzzer_Beep(unsigned int ms);
void Update_Display(void);
void ATS_StateMachine_Run(void);

// ============================================================================
// MAIN PROGRAM
// ============================================================================
void main(void) {
    System_Init();
    Lcd_Clear(); Lcd_Set_Cursor(1, 2); Lcd_Write_String("ATS Controller");
    Lcd_Set_Cursor(2, 4); Lcd_Write_String("Starting...");
    Buzzer_Beep(200); __delay_ms(2000);
    currentState = STATE_ON_CEB;
    while (1) {
        ATS_StateMachine_Run();
        Update_Display();
        __delay_ms(1000);
        if(timer_seconds > 0) timer_seconds--;
    }
}

void System_Init(void) {
    ADCON1 = 0x07; CMCON = 0x07;
    TRISA = 0xFF; TRISB = 0x00; TRISD = 0x00;
    GEN_RELAY = 0; GEN_START_RELAY = 0; CEB_RELAY = 0; BUZZER = 0;
    Lcd_Init(); Create_Custom_Chars();
}

void Create_Custom_Chars(void) {
    Lcd_Send_Full_Byte_Cmd(0x40);
    for (int i = 0; i < 8; i++) Lcd_Write_Char(ceb_char[i]);
```

```
    for (int i = 0; i < 8; i++) Lcd_Write_Char(gen_char[i]);
    for (int i = 0; i < 8; i++) Lcd_Write_Char(arrow_char[i]);
}

void Buzzer_Beep(unsigned int ms) {
    BUZZER = 1; for (unsigned int i = 0; i < ms; i++) __delay_ms(1); BUZZER = 0;
}

// ==============================================================================
// MAIN ATS STATE MACHINE (with Auto-Recovery)
// ==============================================================================
void ATS_StateMachine_Run(void) {
    unsigned char is_ceb_on = CEB_ON_INPUT;
    unsigned char is_gen_on = GEN_ON_INPUT;

    switch (currentState) {
        case STATE_ON_CEB:
            CEB_RELAY = 1; GEN_RELAY = 0; GEN_START_RELAY = 0;
            if (!is_ceb_on) {
                currentState = STATE_CEB_FAILED; timer_seconds = CEB_FAIL_DELAY; Buzzer_Beep(500);
            }
            break;
        case STATE_CEB_FAILED:
            CEB_RELAY = 0;
            if (timer_seconds == 0) {
                currentState = STATE_STARTING_GEN; timer_seconds = GEN_START_CRANK_TIME;
            }
            if (is_ceb_on) { currentState = STATE_ON_CEB; }
            break;
        case STATE_STARTING_GEN:
            GEN_START_RELAY = 1;
            if (is_gen_on) {
                GEN_START_RELAY = 0;
                currentState = STATE_GEN_WARMING_UP; timer_seconds = GEN_WARMUP_TIME;
            } else if (timer_seconds == 0) {
                GEN_START_RELAY = 0;
                if (!is_gen_on) { currentState = STATE_GEN_START_FAIL; }
                else { currentState = STATE_GEN_WARMING_UP; timer_seconds = GEN_WARMUP_TIME; }
            }
            break;
        case STATE_GEN_WARMING_UP:
            if (timer_seconds == 0) { currentState = STATE_ON_GENERATOR; Buzzer_Beep(100); }
            break;
        case STATE_ON_GENERATOR:
            GEN_RELAY = 1; CEB_RELAY = 0;
            if (!is_gen_on) {
                GEN_RELAY = 0; Buzzer_Beep(1000);
                currentState = STATE_GEN_RUNTIME_FAIL;
```

```
            } else if (is_ceb_on) {
                currentState = STATE_CEB_RETURNED; timer_seconds =
    CEB_RETURN_DELAY; Buzzer_Beep(150);
            }
            break;
        case STATE_CEB_RETURNED:
             if (timer_seconds == 0) {
                GEN_RELAY = 0; __delay_ms(500); CEB_RELAY = 1;
                currentState = STATE_GEN_COOLDOWN; timer_seconds =
    GEN_COOLDOWN_TIME;
            }
             if (!is_ceb_on) { currentState = STATE_ON_GENERATOR; }
            break;
        case STATE_GEN_COOLDOWN:
            if (!is_ceb_on) {
                Buzzer_Beep(300); CEB_RELAY = 0; __delay_ms(500); GEN_RELAY
     = 1;
                currentState = STATE_ON_GENERATOR;
            }
            else if (timer_seconds == 0) { currentState = STATE_ON_CEB; }
            break;

        // --- SELF-HEALING LOGIC IS HERE ---
        case STATE_GEN_START_FAIL:
        case STATE_GEN_RUNTIME_FAIL:
            // Check for the recovery condition: Has CEB power returned?
            if (is_ceb_on) {
                // YES! Recover the system automatically.
                BUZZER = 0; // Stop the alarm.
                Buzzer_Beep(250); // Short beep to signal recovery.
                currentState = STATE_ON_CEB; // Go to the safe CEB state.
            } else {
                // NO, CEB is still off. Continue the alarm.
                GEN_RELAY = 0; CEB_RELAY = 0; GEN_START_RELAY = 0;
                BUZZER = !BUZZER; // Intermittent beep
            }
            break;
        // --- END OF SELF-HEALING LOGIC ---

        default:
            currentState = STATE_ON_CEB;
            break;
    }
}

//
    ==============================================================================
// LCD UPDATE FUNCTION (No changes needed)
//
    ==============================================================================
void Update_Display(void) {
    char buffer[17];
    if (currentState == previousDisplayState && currentState !=
    STATE_GEN_START_FAIL && currentState != STATE_GEN_RUNTIME_FAIL) {
        if(currentState == STATE_CEB_FAILED || currentState ==
    STATE_STARTING_GEN ||
```

```
            currentState == STATE_GEN_WARMING_UP || currentState == STATE_CEB_RETURNED ||
            currentState == STATE_GEN_COOLDOWN)
        {
            Lcd_Set_Cursor(2, 14); sprintf(buffer, "%02uS", timer_seconds); Lcd_Write_String(buffer);
        }
        return;
    }
    previousDisplayState = currentState;
    if (currentState != STATE_GEN_START_FAIL && currentState != STATE_GEN_RUNTIME_FAIL) Lcd_Clear();

    switch (currentState) {
        case STATE_ON_CEB:
            Lcd_Set_Cursor(1,1); Lcd_Write_Char(CEB_CHAR_CODE); Lcd_Write_String(" CEB");
            Lcd_Set_Cursor(1,10); Lcd_Write_Char(ARROW_CHAR_CODE); Lcd_Write_String(" LOAD");
            Lcd_Set_Cursor(2,1); Lcd_Write_String("Gen Status: IDLE");
            break;
        case STATE_CEB_FAILED:
            Lcd_Set_Cursor(1,1); Lcd_Write_String("!!  CEB FAIL  !!");
            Lcd_Set_Cursor(2,1); Lcd_Write_String("Starting Gen in ");
            break;
        case STATE_STARTING_GEN:
            Lcd_Set_Cursor(1,1); Lcd_Write_String("Source: <NONE>");
            Lcd_Set_Cursor(2,1); Lcd_Write_String("Cranking Gen... ");
            break;
        case STATE_GEN_WARMING_UP:
            Lcd_Set_Cursor(1,1); Lcd_Write_Char(GEN_CHAR_CODE); Lcd_Write_String(" Gen Running");
            Lcd_Set_Cursor(2,1); Lcd_Write_String("Warming up...   ");
            break;
        case STATE_ON_GENERATOR:
            Lcd_Set_Cursor(1,1); Lcd_Write_Char(GEN_CHAR_CODE); Lcd_Write_String(" GENERATOR");
            Lcd_Set_Cursor(1,10); Lcd_Write_Char(ARROW_CHAR_CODE); Lcd_Write_String(" LOAD");
            Lcd_Set_Cursor(2,1); Lcd_Write_String("CEB Status: OFF");
            break;
        case STATE_CEB_RETURNED:
            Lcd_Set_Cursor(1,1); Lcd_Write_Char(CEB_CHAR_CODE); Lcd_Write_String(" CEB Stable");
            Lcd_Set_Cursor(2,1); Lcd_Write_String("Transfer in...  ");
            break;
        case STATE_GEN_COOLDOWN:
            Lcd_Set_Cursor(1,1); Lcd_Write_String("Source: CEB");
            Lcd_Set_Cursor(2,1); Lcd_Write_String("Gen Cooldown... ");
            break;
        case STATE_GEN_START_FAIL:
            Lcd_Clear(); Lcd_Set_Cursor(1, 1); Lcd_Write_String("!!  CRITICAL  !!");
            Lcd_Set_Cursor(2, 1); Lcd_Write_String("GEN START FAILED");
            break;
        case STATE_GEN_RUNTIME_FAIL:
            Lcd_Clear(); Lcd_Set_Cursor(1, 1); Lcd_Write_String("!!  CRITICAL  !!");
```

```
            Lcd_Set_Cursor(2, 1); Lcd_Write_String("GEN RUN FAILURE");
            break;
    }
}
```

Listing A.1: ATS_Controller_Self_Healing.c