600085 Embedded Systems Development

Green House Control System

Group Report

Word count: 2183

Zak Catherall, Daniel Tregoiing
December 2019

Contents

1 Introduction 2
2 Requirements 2
3 Implementation 3
3.1 Overall System 3
3.2 Hardware Configuration 4
3.3 System Logic 5
3.4 Driver Code 6
3.4.1 Buzzer Driver (buzzer_driver) 6
3.4.2 EEP Driver (eep_driver) 6
3.4.3 IO Driver (io_driver) 7
3.4.4 LCD Driver (lcd_driver) 7
3.4.5 Matrix Driver (matrix_driver) 9
3.4.6 RTC Driver (rtc_driver) 9
3.4.7 Thermometer Driver (thermometer_driver) 10
3.5 Testing 11
4 Evaluation 12
5 Appendices 13
5.1 Appendix 1: Test Report 13
5.2 Appendix 2: User Manual 16

1 Introduction

This document describes the development and implementation of the embedded, 'Green House Control System', for the Embedded Systems Development module. Code was developed in C using the MPLAB IDE with XC8 compiler and targets a PIC 16F877A microcontroller on a QL200 development board. The final application was deployed using QL-Prog.

2 Requirements

These following points are derived from the given project specification and define the required system functionality.

R1. The system shall have 2 operating modes: daytime and night time.
R2. Each operating mode shall have an upper and lower temperature threshold.
R3. The system shall have 2 output controls (for heating and cooling).
R4. Below the lower temperature threshold, the system shall turn on the heating control.
R5. Above the upper temperature threshold, the system shall turn on the cooling control.
R6. When the heating or cooling is activated, the system shall sound an indicative beep.
R7. An alarm will sound if the temperature does not to move towards "safe" temperature after the heating or cooling controls are activated.
R8. The user shall be able to disarm the alarm by pressing a button.
R9. Values set in the system will persist between uses.

The system shall display the current:

R10. Date.
R11. Weekday.
R12. Time.
R13. Temperature (to one decimal place).

The system shall allow the user to set the:

R14. Date.
R15. Time.
R16. Daytime operating period.
R17. Heating and cooling thresholds for daytime mode.
R18. Heating and cooling thresholds for night time mode.

The system shall prevent:

R19. Invalid dates from being saved (with consideration for leap years).
R20. Invalid times from being saved.
R21. The daytime mode start time from being set later than the daytime mode end time.
R22. The lower temperature threshold, for a given mode, from being set greater than the upper temperature threshold.

3 Implementation

3.1 Overall System

The system comprises several key drivers, each providing a fundamental interface to individual, electronic components (e.g. clock, LCD). The drivers are used by the general application code ('main', 'ui') to implement high-level features and functionality. Finally, a utilities library ('utils') provides common functions such as unit-conversions and delay timers.

These relationships are visualised in the figure below.

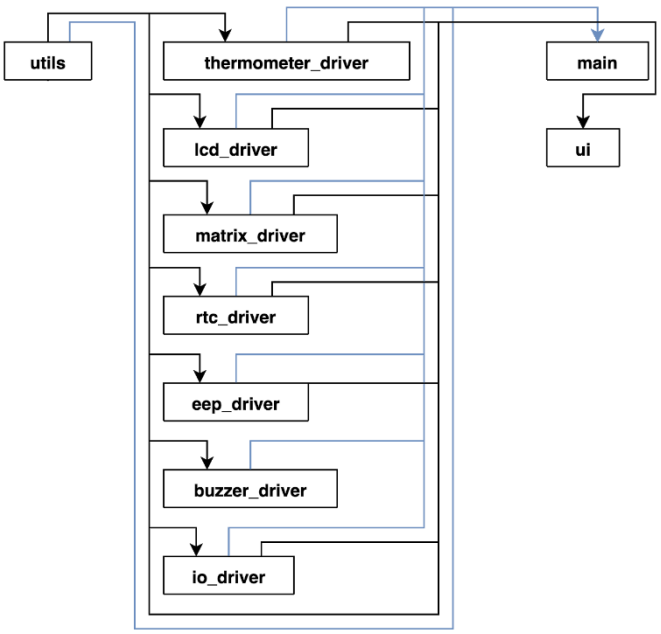

Figure 1: Drivers and application-code relationships.

In short, the 'main' and 'ui' libraries achieve the following:

- **ui:** Manages navigation, rendering each screen and user-input validation.
- **main:**
  - Checks the time (determines if it is day or night).
  - Checks the temperature (determines the IO state e.g. cooling and/or alarming).
  - Navigates (checks for user input to set the UI state).
  - Renders (displays the screen for the current UI state).

3.2 Hardware Configuration

The block diagram below provides a high-level overview of the system hardware components.

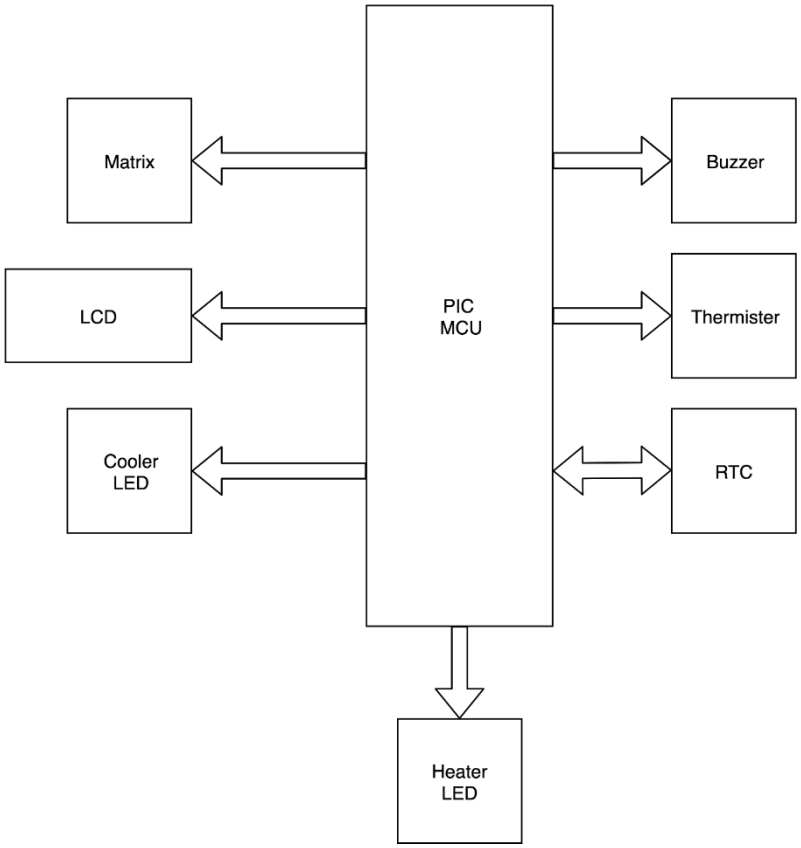

Figure 2: System block-diagram.

The pin-out diagram below shows the MCU pin mappings to the developed drivers. All drivers operate on unique pins without conflict.

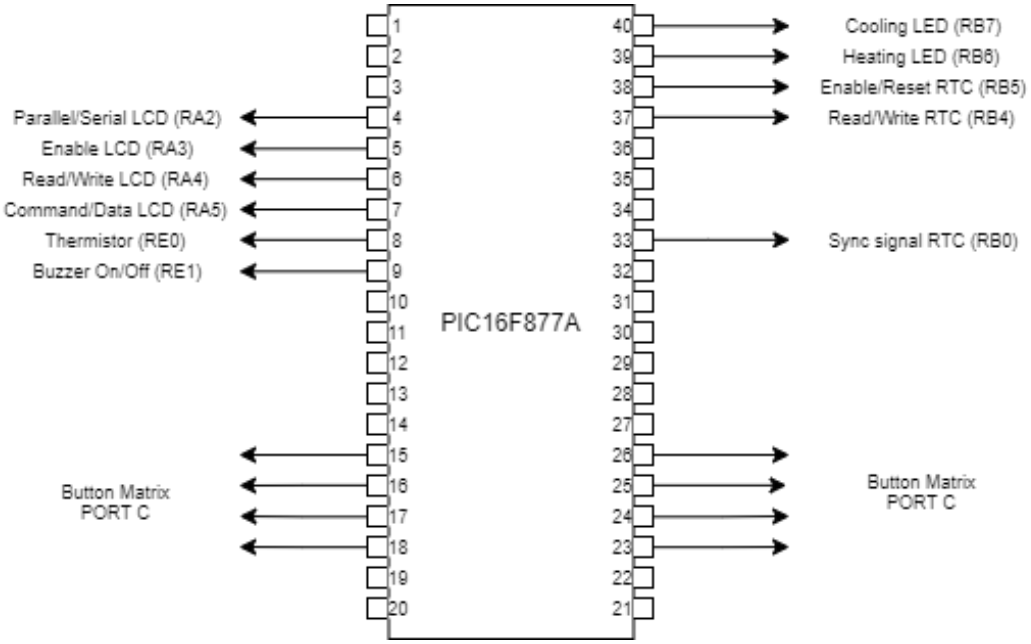

Figure 3: Pin-out diagram for the implemented system.

The components are used as follows:

- **Matrix:** Allows input via 16 buttons.
- **LCD:** Allows output to 4 rows of 16 characters.
- **Cooler/Heater LEDs:** Indicates the state (on/off) of individual IO pins.
- **Buzzer:** Implements the system alarm.
- **Thermistor:** Reads the temperature of the system's environment.
- **RTC (real-time clock):** Maintains the system date and time.

3.3 System Logic

The diagram below gives a simplified overview of system operation and its core features.

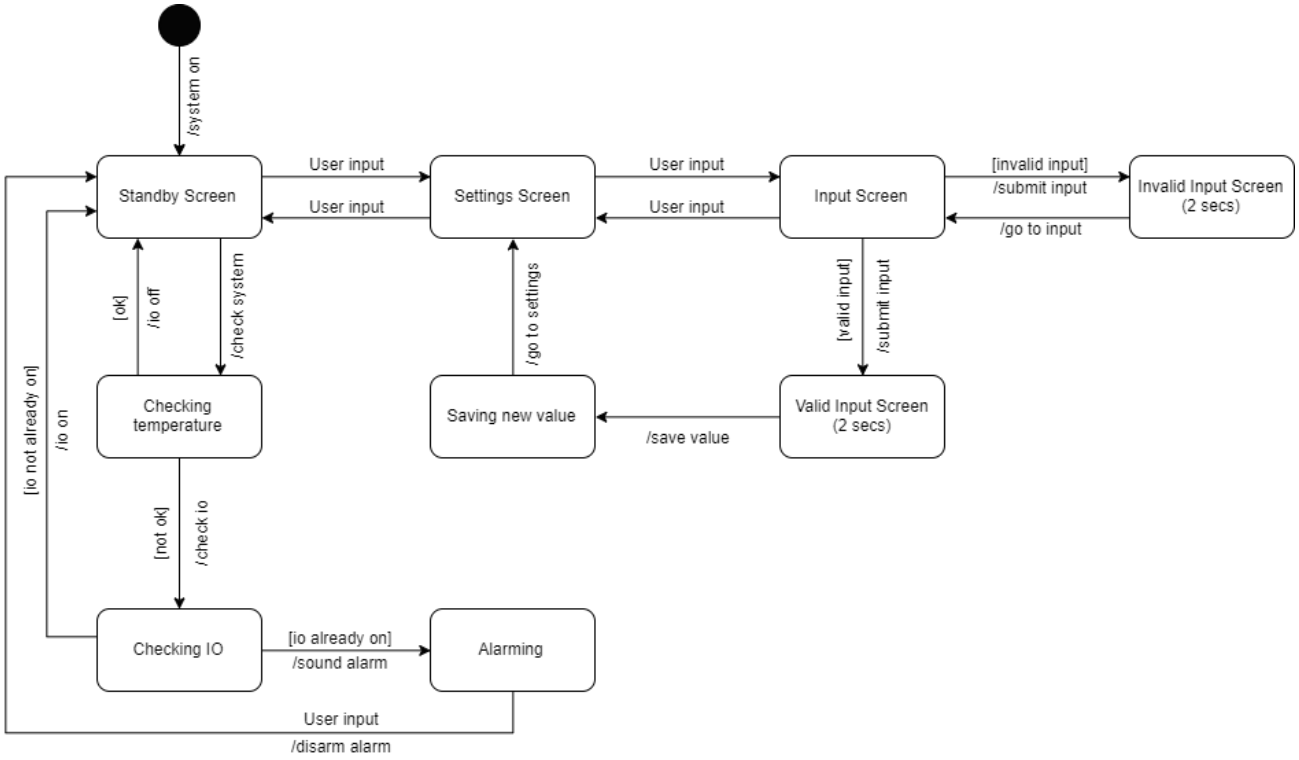

Figure 4: A high-level logic flow of the system.

The diagram abstracts the user interface for simplicity and does not make a distinction between day and night operation, or heating and cooling (which is simply referred to as, "IO"). Exact features and functionality are provided in the operation manual (*Appendix 2*).

3.4 Driver Code

The following sections outline the API that has been developed for each hardware component.

3.4.1 Buzzer Driver (buzzer_driver)

Allows the programmer to output a variety of buzzer tones and patterns.

buzzer_Init

```
[public] void buzzer_init(void)
```

Initialises ports, pins and default values.

buzzer_Sound

```
[public] void buzzer_sound(int bT, int pT, unsigned char reps)
```

Sound the buzzer in a regular pattern of beeps and pauses.

- **Parameters**
  - bT: buzz duration as a number of system cycles.
  - pT: pause duration as a number of system cycles.
  - reps: number times to repeat the buzz/pause pattern.

3.4.2 EEP Driver (eep_driver)

Allows the programmer to store and retrieve data in the on chip EEPROM.

eep_ReadString

```
[public] char* eep_ReadString(char addr, char strN)
```

Read a series of 5 characters from the on chip EEPROM into a target string variable and returns a pointer to said variable.

- **Parameters**
  - addr: EEPROM address to begin reading from.
  - strN: id of the string variable being read into (0 or 1).
- **Returns**
  - A char pointer to one of two string variables, containing the string just read from EEPROM.

eep_WriteString

```
[public] void eep_WriteString(char addr, char str[])
```

Write up to 5 characters to a memory location, starting from the given address.

- **Parameters**
  - addr: EEPROM address to begin writing from.
  - str: string (up to 5 characters long) to be stored.

eep_ReadChar

```
[private] char eep_ReadChar(char addr)
```

Read a single character from an address in the on chip EEPROM.

- **Parameters**
  - addr: EEPROM address to read.
- **Returns**
  - The character that was read at the given EEPROM address.

eep_WriteChar

```
[private] void eep_WriteChar(char addr, char ch)
```

Write a single character to an address in the on chip EEPROM.

- **Parameters**
  - addr: EEPROM address to write to.
  - ch: the character literal or numeric equivalent to be stored.

3.4.3 IO Driver (io_driver)

Allows the programmer to toggle the output of defined pins and provides a global string (io_Status) which can be set each time a pin is toggled.

io_Init

```
[public] void io_Init(void)
```

Initialises ports, pins and default values.

io_TogglePin

```
[public] void io_TogglePin (unsigned char pinN, char status[])
```

Toggles the signal of an individual pin and updates the global status string (io_Status).

- **Parameters**
  - `pinN`: id of the pin to toggle (0 or 1).
  - `status`: string to update the global status variable with.

io_SwitchOff

```
[public] void io_SwitchOff(void)
```

Switches off all IO pins.

3.4.4 LCD Driver (lcd_driver)

Allows the programmer to interface with the 128x64 LCD.

Uses pins RA2, RA3, RA4 and RA5; and all of PORTD.

lcd_Init

`[public] void lcd_Init(void)`

Initialises ports, pins and default values.

lcd_SetCursorPos

`[public] void lcd_SetCursorPos(unsigned char lineN, unsigned char pos)`

Sets the position of the cursor. Can be used to overwrite a specific character and write to any character cell.

- **Parameters**
  - lineN: line number ranging from 0 (line 1) to 3 (line 4).
  - pos: cursor position (inset/ indentation) from the left side of the LCD. A value of 0 places the cursor at the left of the screen, whilst a value of 7 sets it to the right. **NB:** character cells are 2 characters wide.

lcd_PrintChar

`[public] void lcd_PrintChar(char character)`

Write a single character to the LCD at the current cursor position.

- **Parameters**
  - character: character literal or numeric equivalent to be output.

lcd_PrintString

`[public] void lcd_PrintString(char str[], unsigned char lineN, unsigned char pos)`

Write a series of characters to the LCD from the current cursor position.

- **Parameters**
  - str: character array. **NB:** the LCD is 16 characters wide and overflowing text is not managed.
  - lineN: line number ranging from 0 (line 1) to 3 (line 4).
  - pos: cursor position (inset/ indentation) from the left side of the LCD. A value of 0 places the cursor at the left of the screen, whilst a value of 7 sets it to the right. **NB:** character cells are 2 characters wide.

lcd_Clear

`[public] void lcd_Clear(void)`

Clear the LCD of all output.

lcd_WriteCmd

`[private] void lcd_WriteCmd(char command)`

Writes a command byte to the component.

- **Parameters**
  - command: some command byte from the LCD component datasheet.

3.4.5 Matrix Driver (matrix_driver)

Allows the programmer to capture user input via the 4x4 button matrix.

matrix_Init

```
[public] void matrix_Init(void)
```

Initialises ports, pins and default values.

matrix_GetInput

```
[public] char matrix_GetInput(void)
```

Scans each row of the matrix circuit for the location of a button press and returns the character mapped to the location of the press.

- **Returns**
  - A character mapped to the button pressed.
  - An underscore ('_') if no button was pressed.

matrix_Scan

```
[private] unsigned char matrix_Scan(unsigned char row)
```

Scans a given row of the matrix and stores the location of the button press in the private, global variable: "result".

- **Returns**
  - 1 if a button press was detected.
  - 0 if no button press was detected.

3.4.6 RTC Driver (rtc_driver)

Allows the programmer to interface with the system clock in 24hr mode.

rtc_Init

```
[public] void rtc_Init(void)
```

Initialises ports, pins and default values.

rtc_SetTimeComponent

```
[public] void rtc_SetTimeComponent(char addr, char val)
```

Sets a specific component of time (e.g. the year) to the given value.

- **Parameters**
  - addr: memory address of the target time component (see DS1302 datasheet).
  - val: value to set.

rtc_Update

```
[public] void rtc_Update(void)
```

Reads the RTC values into the private, global variable: “rtc_Vals”, using burst mode.

rtc_GetString

```
[public] char* rtc_GetString(char isDate)
```

Reads the values from the private global, “rtc_Vals”, and stores them as a string in the private global, “rtc_StrVals”, in the format (“YY-MM-DD” or “HH:MM:SS”).

- **Parameters**
  - isDate**:** whether to read date values (1) or time values (0).
- **Returns**
  - A pointer to the private global: “rtc_StrVals”.

rtc_SetDay

```
[private] void rtc_SetDay(void)
```

Calculates and sets the weekday component in the rtc register.

rtc_WriteByte

```
[private] void rtc_WriteByte(char addr)
```

Writes a byte to the clock.

- **Parameters**
  - addr: clock command (e.g. burst mode) or address (e.g. year).

rtc_ReadByte

```
[private] char rtc_ReadByte(void)
```

Reads a byte from the clocks (previously set) active register.

- **Returns**
  - A BCD (binary coded decimal) representation of the time component.

3.4.7 Thermometer Driver (thermometer_driver)

Allows the programmer to read the temperature in degrees Celsius from the thermistor, using 1-pin mode.

therm_GetTemp

```
[public] char* therm_GetTemp(void)
```

Gets the temperature from the thermistor to 1 decimal place and returns it as a formated string.

- **Returns**
  - The current temperature, to one decimal place, formatted as a string (“00.0C”).

therm_Reset

```
[private] void therm_Reset(void)
```

Resets the component and waits for a response.

therm_WriteByte

```
[private] void therm_WriteByte (unsigned char val)
```

Writes a command byte to the component.

- **Parameters**
  - val: some command byte from the thermistor component datasheet.

therm_ReadTemp

```
[private] int therm_ReadTemp (void)
```

Reads the integer and decimal components of the temperature into the private globals, “TLV”, and, “THV”, combines them, and returns them as a single integer.

- **Returns**
  - The temperature as an integer, where the leftmost nibble is the integer value and the rightmost is the decimal component.

therm_ReadByte

```
[private] unsigned char therm_ReadByte(void)
```

Reads a temperature component from the thermistor, depending on the timing.

- **Returns**
  - The integer or decimal component of the current temperature, depending on the timing.

3.5 Testing

Test reports for individual drivers (unit tests) are included in the associated individual reports. A test report for overall system operation (integration tests) is included in appendix 1 of this document.

The tests were designed to be run sequentially, and they validate the delivery of the proposed user requirements.

Testing did reveal some errors, such as:

- Leap-year validation - the formula was incorrectly implemented leap years weren’t being identified.
- Night mode - some brackets were misplaced, causing the system to determine it was always night time.

These errors and any others have since been corrected, and there are no known bugs at this time.

4 Evaluation

The project was an opportunity to revisit many fundamentals of computer science and programming. Bit shifting, hexadecimal, and knowledge of data types were key to its success.

Initial learning included referencing the datasheet more often. Time was spent, for example, trying to activate pin RA7, which wasn't implemented, and for a time the LCD component flickered because the parallel flag wasn't set.

An issue occurred where *tested* input-validation code was failing. Eventually the **warning** related to stack depth was found in the build-output. By sacrificing the preferred practice of one method doing one job (so that the complexity of the application could be maintained) the depth was reduced to the allowed 7 and the input validation began working.

Nearing completion, the program was failing to build due to the program size. A variety of modifications learnt and applied, such as converting integer literals under 255 to unsigned chars and limiting functions that return values (using 'tables' instead). At the end of the project, it was discovered that the xc8 compiler also provides different optimisation configurations, which could free up to ~15% program space.

Work was split evenly and both team-members developed a thorough understanding of the concepts used by overcoming challenges individually and collaboratively. Individual implementations were reviewed and refactored by both members to further personal understanding and improve the application, with consideration for best practices (e.g. private variables).

The deliverables and operation of the group could have been improved by taking the time to produce a project plan at the start of the project, including discussion and agreement on the tools and conventions to use. This caused some repetition of work at critical moments, which could have otherwise been avoided.

Overall, the team could have worked more effectively together but still operated to a level that enabled and encouraged each other's learning to produce a complete, quality application.

5 Appendices

5.1 Appendix 1: Test Report

The table below servers as both the test script and test report for the integration tests conducted on the final system. Individual components (drivers) were tested separately; the reports for which are found in the individual reports.

In summary, all tests passed after making some small corrections to bugs that were revealed by the tests (e.g. leap year calculation not working) and there are no known bugs.

ID	Test Description	Steps	Expected	Pass/Fail
1	Settings Screen (page 2) loads on first use.	Reprogram the device.	Settings screen 2 displays.	PASS
2	User cannot leave the settings screen until the daytime and both thresholds have been set.	1. Press the close button. 2. Set the daytime and press close. 3. Set the day threshold and press close. 4. Set the night threshold and press close.	1. Nothing happens. 2. Nothing happens. 3. Nothing happens. 4. Standby screen displays.	PASS
3	Displays current date.	1. View the standby screen.	A date is displayed on the first line of the LCD in the format: "Da: *YY-MM-DD Day*"	PASS
4	Weekday is correct.	View the standby screen.	Cross-check the day of the week for the current system date with a calendar – the day of the week should be the same (e.g. "Fri").	PASS
5	Displays current time.	View the standby screen.	1. A time is displayed on the second line of the LCD in the format: "Ti: *HH:MM:SS*" 2. The seconds component updates at least once every second.	PASS
6	Displays current temperature.	1. View the standby screen. 2. Hold the thermistor to raise the temperature.	1. A temperature is displayed on the third line of the LCD in the format: "Te: *00.0C"* 2. The displayed temperature increases.	PASS
7	Setting daytime.	Navigate to the set daytime screen and input the following values: 1. 0600 0600 2. 0600 2400 3. 0690 1900 4. 0600 1900	1. Invalid 2. Invalid 3. Invalid 4. Success	PASS

8	Setting date.	Navigate to the set date screen and input the following values: 1. 19-00-01 2. 19-01-00 3. 19-01-32 4. 19-01-31 5. 19-02-29 6. 19-04-31 7. 20-02-29 8. Return to the standby screen.	1. Invalid 2. Invalid 3. Invalid 4. Success 5. Invalid 6. Invalid 7. Success 8. The first line of the LCD reads: "Da: 20-02-29 Sat"	PASS
9	Setting time.	Navigate to the set time screen and input the following values: 1. 24:00:00 2. 23:60:00 3. 23:59:60 4. 23:59:00 5. Return to the standby screen.	1. Invalid 2. Invalid 3. Invalid 4. Success 5. The second line of the LCD reads: "Ti: 23:59:*01*"	PASS
10	Setting thresholds (daytime).	Navigate to the set thresholds (daytime) screen and input the following values: 1. 10.0C 05.0C 2. 05.0C 05.1C 3. 05.0C 05.2C	1. Invalid 2. Invalid 3. Success	PASS
11	Setting thresholds (night).	Navigate to the set thresholds (night time) screen and input the following values: 1. 30.0C 20.0C 2. 30.0C 30.1C 3. 30.0C 30.2C	1. Invalid 2. Invalid 3. Success	PASS
12	Heating activates.	Return to the standby screen.	The system beeps, the status changes to, "HEATING", and the led turns on.	PASS
13	Alarm sounds.	Hold the thermistor.	Screen reads: "Hold to disarm!" and alarm begins sounding.	PASS
14	Alarm disarms.	Whilst the alarm is sounding, press any button.	Alarm stops sounding and standby screen is displayed.	PASS
15	Alarm doesn't sound if IO works.	Hold the thermometer (increase the temperature).	Alarm doesn't sound whilst the temperature is increasing at least 0.1C per 3 seconds.	PASS
16	Cooling activates.	1. Navigate to the set time screen and set the current time to be	The system beeps, the status changes to, "COOLING", and the led turns on.	PASS

		within of the current daytime range. 2. Return to the standby screen.		
17	System values persist.	• Make a note of all system values (date, time, daytime, threshold (d), threshold (n) • Switch off the system. • Switch the system on. • Check all the values.	The date and time incremented according to the time the device was switched off and all other set values persist.	PASS

Table 5: Integration tests, test script.

5.3 Appendix 2: User Manual

Contents

1 System Overview 16
2 First time use 16
3 User Interface 17
3.1 Inputs (Button Mappings) 17
3.2 Screens 17
4 Settings 19
4.1 Setting the Date 19
4.2 Setting the Time 19
4.3 Setting the Daytime Operating Period 20
4.4 Setting Temperature Thresholds 20
5 Passive Operations 20
5.1 Day and Night Modes 20
5.2 Heating and Cooling 20
5.3 Alarm 21

1 System Overview

The greenhouse control system monitors time and temperature to manage temperature deviations, by providing power to external cooling and heating systems and alerting the user of failure of these systems.

2 First time use

When starting the device for the first-time, the user is presented with the settings screen where they must input values for daytime and thresholds before normal operation resumes. This ensures that the system is configured before use.

3 User Interface

3.1 Inputs (Button Mappings)

The following is a diagram of the button mappings in the 4x4 matrix.

7 *Input the number 7*	**8** *Input the number 8*	**9** *Input the number 9*	**x** *Cancel/back*
4 *Input the number 4*	**5** *Input the number 5*	**6** *Input the number 6*	**<** *Up/previous*
1 *Input the number 1 /select menu item 1*	**2** *Input the number 2 /select menu item 2*	**3** *Input the number 3 /select menu item 3*	**>** *Down/next*
0 *Input the number 0*	**.** *Input a decimal point*	**b** *Backspace*	**s** *Enter/select*

Figure 6: Device inputs and button map.

3.2 Screens

The following are representations of the different screens displayed by the device, with explanations for screen items and diagrams of available inputs (highlighted in green).

Name	Screen	Description	Inputs
Standby	Da: YY-MM-DD Day Ti: HH:MM:SS Te: ##.#C St: <status text>	Date (year: month: day) Time (hours: minutes: seconds) Temperature (degrees Celsius) Status (OK/HEATING/COOLING)	7 8 9 x 4 5 6 < 1 2 3 **>** 0 . b s

Settings (page 1)	Settings 1. Date 2. Time …	Screen Title Access the *Set Date* screen. Access the *Set Time* screen. Next-page indicator	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Settings (page 2)	… 1. Daytime 2. Threshold (d) 3. Threshold (n)	Previous-page indicator Access the *Set Daytime* screen. Access the *Set Thresholds (daytime)* screen. Access the *Set Thresholds (night)* screen.	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Set Date	Date Cur: YY-MM-DD New: …	Screen Title Current value Input area Confirm-input indicator	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Set Time	Time Cur: HH:MM:SS New: …	Screen Title Current value Input area Confirm-input indicator	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Set Daytime	Daytime Cur: HH:MM HH:MM New: …	Screen Title Current values (start & end) Input are Confirm-input indicator	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Set Thresholds (daytime)	Threshold (d) Cur: ##.#C ##.#C New: …	Screen Title Current values (lower & upper) Input area Confirm-input indicator	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Set Thresholds (night time)	Threshold (n) Cur: ##.#C ##.#C New: …	Screen Title Current values (lower & upper) Input area Confirm-input indicator	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Alarm	Hold to disarm!	Message	7 8 9 x 4 5 6 < 1 2 3 > 0 . b s
Valid Input	Success!	Message	None

Invalid Input	Invalid!	Message	None

Figure 2: The different views of the system (UI).

4 Settings

For all settings there is an individual input screen which accepts a limited number of numeric inputs. All input screens exhibit the following behaviours:

- The expected number of inputs must be provided i.e. a single digit value of '1' must be provided as '01', if the expected number inputs is 2.
- Previous inputs can be deleted by pressing the backspace button.
- The user can press the cancel button to return to the settings screen without saving changes.
- When the expected number of inputs have been provided, "…", appears at the bottom of the screen to indicate that the user can press the select button to submit.
- If the inputs are **invalid**, the 'Invalid Input' screen will display for a short time and the user will be returned to the input screen.
- If the inputs are **valid**, the 'Valid Input' screen will display for a short time, the time will be set, and the user will be returned to the settings screen.

4.1 Setting the Date

NB: *the weekday is automatically calculated.*

Navigation:

Standby → Settings (page 1) → Set Date

Purpose:

The system date is used for displaying the current date on the standby screen.

Input:

- The screen accepts 6 numeric input characters in the format: YYMMDD (year 00-99, month 01-12, date 01-31).
- A separation character ('-') is inserted automatically after the YY and MM inputs for readability.
- Dates must be valid.

4.2 Setting the Time

NB: *the system clock operates in 24hr mode.*

Navigation:

Standby → Settings (page 1) → Set Time

Purpose:

The system time is used for displaying the current time on the standby screen and deciding which operation mode (daytime or night time) is active.

Input:

- The screen accepts 6 numeric input characters in the format: HHMMSS (hours 00-23, minutes 00-59, seconds 00-59).
- Separation characters (':') are inserted automatically after the HH and MM inputs for readability.

4.3 Setting the Daytime Operating Period

Navigation:

Standby → Settings (page 1) → Settings (page 2) → Set Daytime

Purpose:

Within the bounds of the configured time, the system operates in daytime mode. Outside of those bounds, the system operates in night time mode; each mode has individually configurable temperature thresholds.

Input:

- The screen accepts 6 numeric input characters in the format: HHMHHM (hours 00-23, minutes 0-5), where the first HHM represents the daytime start and the second represents the end (when night time begins).
- Start and end times can only be configured to the nearest 10 minutes and a zero is automatically inserted after a 'M' input.
- Separation characters (':') are inserted automatically after the HH inputs for readability.
- A space is automatically inserted between the start and end times.

4.4 Setting Temperature Thresholds

Navigation:

For daytime operation:

Standby → Settings (page 1) → Settings (page 2) → Set Thresholds (daytime)

For night time operation:

Standby → Settings (page 1) → Settings (page 2) → Set Thresholds (night time)

Purpose:

The temperature thresholds are used by the system to determine if the heating or cooling systems should be active.

Input:

- The screen accepts 6 numeric input characters 0-9, where the first 3 represent the lower threshold (when heating should be activated) and the second 3 represent the upper threshold (when cooling should be activated).
- Values are to 1 decimal place i.e. the thresholds are in the format '00.0'.
- Decimal points are inserted automatically.
- A 'C' is inserted after the every 3rd input to indicate the units of measurement (Celsius).
- A space is automatically inserted between the lower and upper thresholds.
- A lower threshold must be less than its corresponding upper threshold by at least 0.2C.

5 Passive Operations

The following describe the fundamental functions of the Green House Control System.

5.1 Day and Night Modes

The system automatically switches between day and night operation based on the current system time and configured "Daytime" setting.

5.2 Heating and Cooling

Whenever the system temperature moves outside of the user-specified range for the current operating period, the heating or cooling system will be activated accordingly.

When either system is activated, the system emits an indicative beep, and the LCD status on the standby screen changes to, “COOLING”, or, “HEATING”. At normal temperatures this status will display as, “OK”, plus a reference to the current mode (day or night).

5.3 Alarm

NB: The alarm will not sound whilst the user is not on the standby screen, as avoid disrupting user input whilst configuring the system.

The alarm sounding indicates that the heating or cooling systems are not operating sufficiently to correct the system temperature.

The alarm will sound if the temperature falls below the configured lower temperature threshold of the current operating mode (day or night) and continues to fall after the heating system has been activated. Likewise, the alarm will also sound the temperature continues to rise after the cooling system is enabled.

Whilst the alarm is sounding, the screen will display, “Hold to disarm!”, indicating that the user can hold any button to deactivate the alarm.

The alarm will remain disarmed for approximately 5 seconds before sounding again (if the temperature is not improving).