

March 11, 2023

LV firmware document Firmware version 08.00

TBD

Revision 00.01



NyQuest
Innovation Labs
Imbibe. Innovate. Inspire.

Ashwin D
Embedded Design Engineer
ashwin@nyquestindia.com

Nyquest Innovation Labs

Door No. 18/643, Keerti Mahal, Mynagappally P.O,
Kollam, Kerala, India- 690519

Phone: +91(0) 476-2847700

www.nyquestindia.com

1	Installing esp-idf	7
2	Installing visualstudio code and node.js	10
2.1	Installing vscode	10
2.2	Installing node.js	10
2.3	Creating esp32 project	11
3	esp32s2_gpio.c	12
3.1	Initialize mains sense	12
3.2	Initialize push button	12
3.3	Initialize S4 switch	12
3.4	Initialize LED	12
3.5	Initialize switches	12
3.6	Initialize I2C enable	13
3.7	Initialize buzzer	13
3.8	Buzzer control	13
3.9	Toggle buzzer	13
3.10	Mains sense	13
3.11	Battery mosfet switch	13
3.12	Solar mosfet switch	14
3.13	Set triac level	14
3.14	Set mains relay	14
3.15	Set S4 relay	14
3.16	Turn on battery mosfet	14
3.17	Turn off battery mosfet	15
3.18	Turn on solar mosfet	15
3.19	Turn off solar mosfet	15
3.20	Turn on mains relay	15
3.21	Turn off mains relay	16
3.22	Set device state	16
3.23	Transit to state 5 from state 4	16
3.24	Led blink	17
3.25	Set blink condition	17
3.26	Device initialization	17
4	esp32s2_adc.c	19
4.1	Variables used in esp32s2_adc.c	19
4.2	Initialize ADC	19
4.3	Software ADC conversion	19
4.4	Get battery voltage from PAC	19
4.5	Read digital values from sensor	20
4.6	Get analog values from sensor	20
5	esp32s2_i2c.c	23
5.1	I2C initialization	23
5.2	I2C detect slave	23

6	Header file of PAC	23
6.1	Function definitions associated with PAC	23
6.2	Macros associated with PAC	24
7	PAC1720 Current and Power Measurement Device	25
7.1	Initialization of PAC1720	25
7.2	Check PAC availability	27
7.3	Write 2bytes into the PAC	27
7.4	Write 1byte into the PAC	28
7.5	Read 2bytes from the PAC	28
7.6	Read 1byte from the PAC	29
7.7	Read data buffer from the PAC	30
8	EEPROM	31
8.1	Select I2C memory	31
8.2	Check EEPROM communication	31
8.3	Read a byte from eeprom	31
9	Write bytes into EEPROM	32
9.1	Write series of data into EEPROM	32
9.2	Read a series of data from the EEPROM	32
9.3	Write array of data into the EEPROM	33
9.4	Write a 16bit data into the EEPROM	34
9.5	Read 16bit data from the EEPROM	34
9.6	Write 32bit data into the EEPROM	34
9.7	Read 32bit data from EEPROM	34
9.8	Write float data into the EEPROM	35
9.9	Read float data from EEPROM	35
9.10	Check memory available	35
10	Data logging	36
10.1	Flush log data	36
10.2	Restore data tail	36
10.3	Save data head and tail	36
10.4	Write data into EEPROM	36
10.5	Retrive data from EEPROM	37
10.6	Updating tail of data	37
10.7	Check availability of data	37
10.8	Log data to EEPROM	38
11	esp32s2_wifi.c	39
11.1	WiFi event handler	39
11.2	Connect to station point	40
11.3	Create access point	40
11.4	Initialize WiFi module	40
11.5	Print log	40
11.6	Callback to sync time	41
11.7	Get RTC time	41
11.8	Initialize RTC module	41
11.9	Stop DHCP server	41
11.10	Start DHCP server	42
12	esp32s2_socket.c	43
12.1	TCP client task	43
12.2	TCP server task	44
13	Parsing incoming data	46
13.1	Response to utility	46
13.2	Response to utility for authentication	47
13.3	Read ADC value	47
13.4	Read EEPROM value	48
13.5	Load WiFi parameters	48
13.6	Calculate EEPROM CRC	48
13.7	Write DATA to eeprom	49
13.8	Send real values to utility	50

13.9	Process command	50
14	esp32s2_https_protocol.c	52
14.1	Variables declared in esp32s2_https_protocol.c	52
14.2	HTTPS Event Handler	52
14.3	Prepare configuration data	53
14.4	Prepare calibration data	53
14.5	Prepare system threshold data	54
14.6	Prepare useable SoC data	54
14.7	Prepare log data	55
14.8	Test connectivity to server	56
14.9	Send data to server	56
14.10	Bi-directional communication with server	56
14.11	HTTPS communication method	57
15	esp32s2_parse_data.c	59
15.1	Variables used in parse_data.c file	59
15.2	Queue for holding request to server	60
15.3	Check if data available in queue	60
15.4	Peek data in the queue	60
15.5	Flush queue	60
15.6	Parse response from server	61
15.7	Detect payload from the server	61
15.8	Detect payload type	61
15.9	Detect write request	61
15.10	Detect read request	61
15.11	Parse message sent in write request	62
15.12	Parse message sent in read request	62
16	esp32s2_task.c	63
16.1	Semaphore creation	63
16.2	Test wifi configuration task	63
16.3	Main timer task	63
16.4	On connection task	64
16.5	Overvoltage protection	64
16.6	LED blink	64
16.7	State machine task	64
16.8	data logging task	65
16.9	Time sync	66
16.10	Wait for time sync	66
16.11	Check data availability	66
16.12	Check connection status	67
16.13	Connecting to AP	67
16.14	Waiting for connection with AP	67
16.15	Server response timeout	68
17	Update tail	68
18	Disconnect from AP	68
19	Least priority operations	68
19.1	Perform OTA	69
19.2	Perform factory reset	69
19.3	Perform system reset	70
19.4	Custom watchdog task	70
19.5	Task creation	70
20	Main algorithm	72
20.1	Turtle algorithm	72
20.2	Sun up routine	72
20.3	Sundown/Solar disconnection detection	80
20.4	Sundown Routine	82
20.5	Solar energy bucketing	84
20.6	Dynamic force trip exit	86
20.7	Detection of mains fail during absorption	86

20.8	Battery SoC reset	86
20.9	Detection of equalization failure	87
20.10	Useable SoC calculation standby	88
20.11	Force trip restriction	89
20.12	Critical battery charge back	89
20.13	Solar availability detection	90
20.14	Morning and evening peak inhibition	91
20.15	Inverter overload detection during force trip	91
20.16	Inverter failure detection	93
20.17	Evening force trip exit	93
20.18	Evening battery low charging	95
20.19	Night battery low charging	95
20.20	Load energy bucketing	96
20.21	Dynamic force trip entry algorithm	96
20.22	Dynamic force trip entry time estimate	96
20.23	Battery overvoltage protection	97
21	State Change Algorithm	98
21.1	State 5 to State 7	99
21.2	State 5 to State 1	99
21.3	State 5 to State 3	99
21.4	State 1 to State 5	100
21.5	State 1 to State 4	100
21.6	State 1 to State 3	101
21.7	State 1 to State 5	101
21.8	State 3 to State 1	102
21.9	State 4 to State 5	103
21.10	State 4 to State 6	103
21.11	State 6 to State 4	103
21.12	State 6 to State 5	103
21.13	State 7 to State 5	104
21.14	State 7 to State 3	104
22	app main	105
22.1	esp32s2_main.h	105
22.2	Idle task monitor to reset controller	105
22.3	SPIFFS parition paths	105
22.4	SPIFFS Initialization	105
22.5	Read data from SPIFFS	106
22.6	Update certificate in SPIFFS	106
22.7	Read domain name from SPIFFS	107
22.8	peripheral initialization	107
22.9	app_main	108
22.10	Factory reset	108
23	Secure boot and Secure flash update	108
23.1	Generating keys for secure boot and flash encryption	108
23.2	Setting up keypurpose for ESP32S2	108
23.3	Burning the digest into the BLOCKs	109
23.4	Writing the flash encryption keys to BLOCK 7 and 8	109
23.5	Change to custom partition	109
24	Enabling secure boot and flash encryption	109
24.1	Build and sign the Bootloader	110
24.2	Building the app image	110
24.3	Flashing the controller	110

LIST OF FIGURES

1	Link to download IDF	7
2	idf_setup	7
3	Accept IDF agreement	8
4	Select the python version to use	8
5	Select the git version to use	9
6	Start ESP COMMAND LINE	9
7	Navigate to extension	10
8	C/C++ IntelliSense	10
9	Creating ESP32 APP	11
10	State transition diagram	98

LIST OF TABLES

1	Device state	98
---	------------------------	----

1 Installing esp-idf

1. Navigate to the following link <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>
2. Click on get started

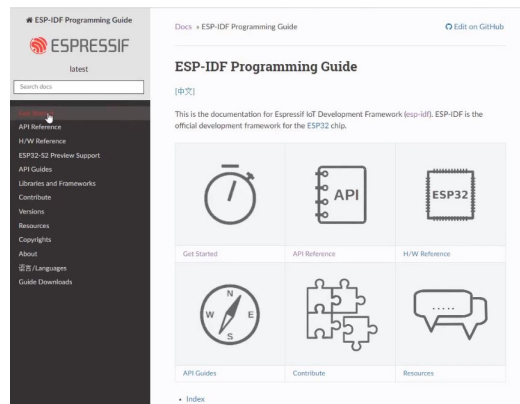


Figure 1: Link to download IDF

3. Navigate to ESP-IDF tool installer and download the esp-idf tool

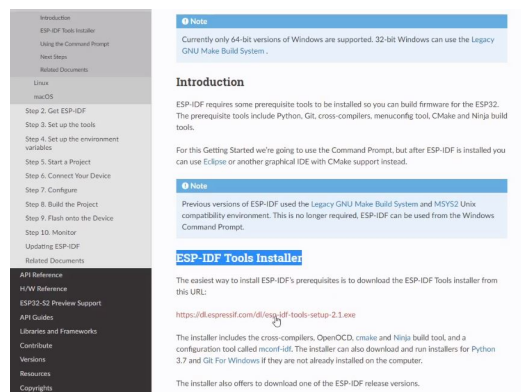


Figure 2: idf_setup

4. Run the exe file

5. Accept license agreement

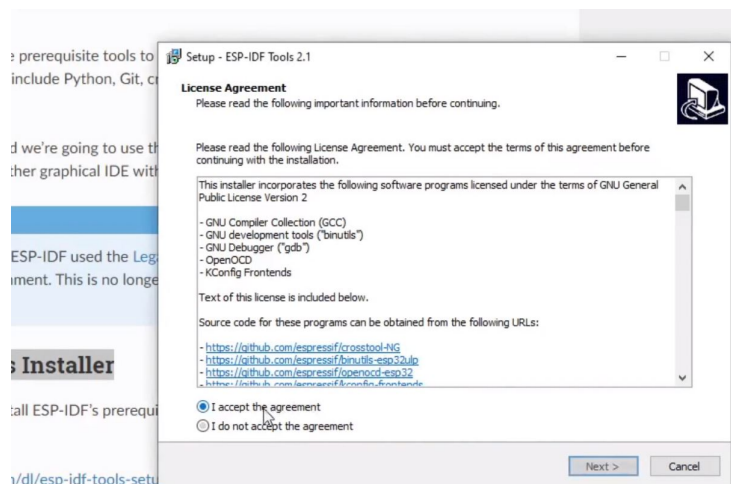


Figure 3: Accept IDF agreement

6. Select the version of python installed

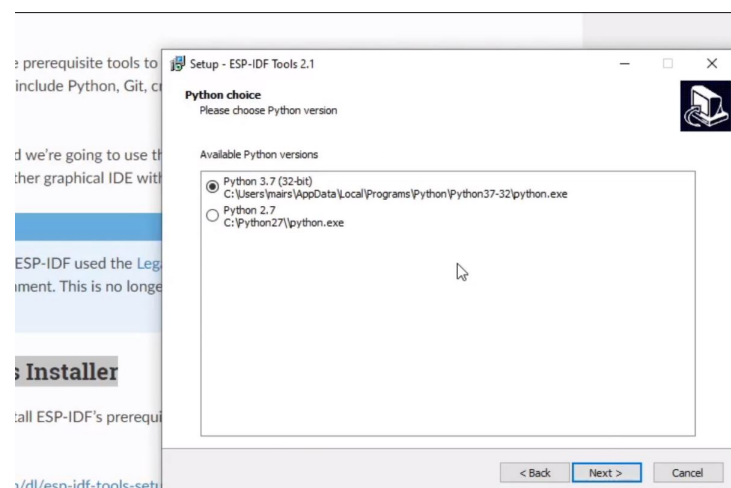


Figure 4: Select the python version to use

7. Select the version of git installed

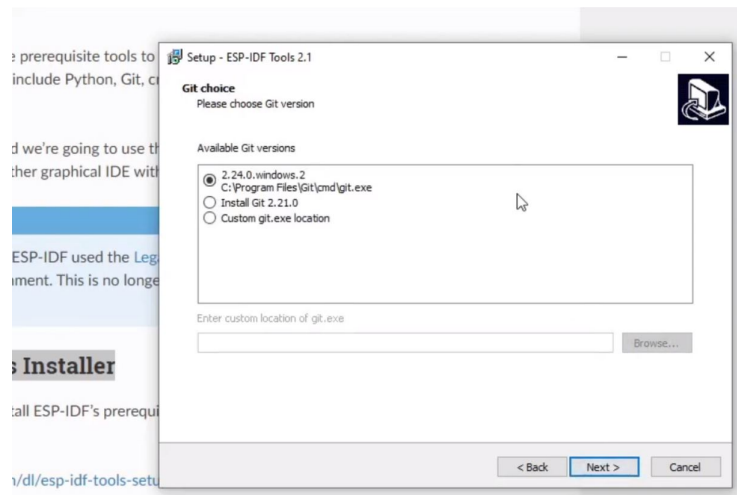
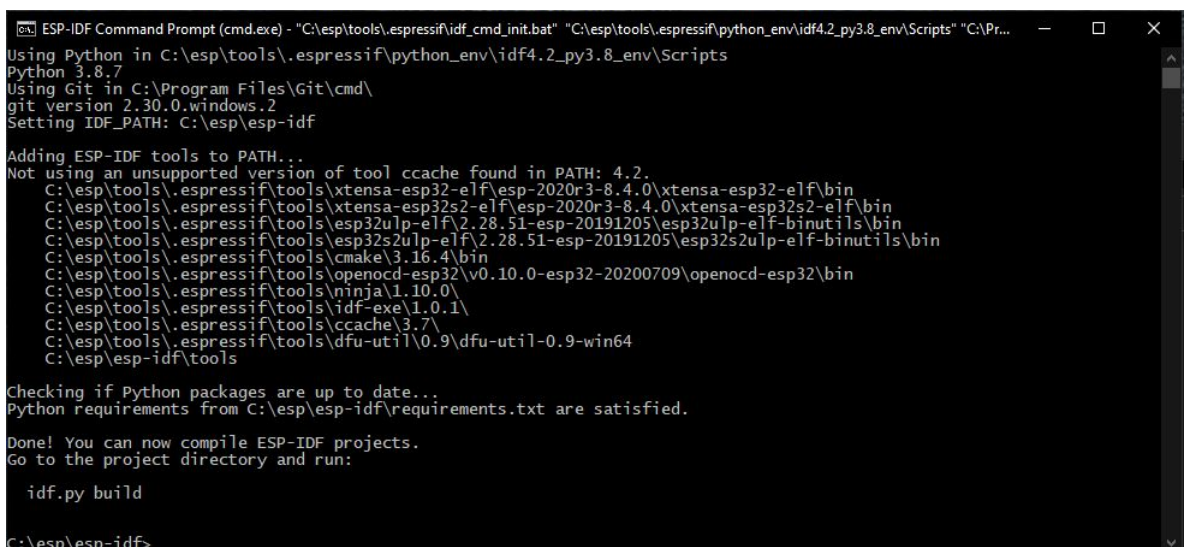


Figure 5: Select the git version to use

8. Select a release version greater than v4.1 for ESP32S2 support
9. Create a directory with name esp in C:/
10. Create a directory inside C:/esp named esp-idf
11. Create a directory inside C:/esp named tools
12. Select the directory for CHOOSE A DIRECTORY TO DOWNLOAD ESP-IDF TO as C:/esp/esp-idf
13. Select the directory for ESP TOOLS DESTINATION as C:/esp/tools
14. Click install
15. After installation ESP-IDF Command Prompt (cmd.exe) will be available
16. Click on the .exe file to ensure the success of installation



```

ESP-IDF Command Prompt (cmd.exe) - "C:\esp\tools\espressif\idf_cmd_init.bat" "C:\esp\tools\espressif\python_env\idf4.2_py3.8_env\Scripts" "C:\Pr...
Using Python in C:\esp\tools\espressif\python_env\idf4.2_py3.8_env\Scripts
Python 3.8.7
Using Git in C:\Program Files\Git\cmd\
git version 2.30.0.windows.2
Setting IDF_PATH: C:\esp\esp-idf

Adding ESP-IDF tools to PATH...
Not using an unsupported version of tool ccache found in PATH: 4.2.
C:\esp\tools\espressif\tools\xtensa-esp32-elf\esp-2020r3-8.4.0\xtensa-esp32-elf\bin
C:\esp\tools\espressif\tools\xtensa-esp32s2-elf\esp-2020r3-8.4.0\xtensa-esp32s2-elf\bin
C:\esp\tools\espressif\tools\esp32ulp-elf\2.28.51-esp-20191205\esp32ulp-elf-binutils\bin
C:\esp\tools\espressif\tools\esp32s2ulp-elf\2.28.51-esp-20191205\esp32s2ulp-elf-binutils\bin
C:\esp\tools\espressif\tools\cmake\3.16.4\bin
C:\esp\tools\espressif\tools\openocd-esp32\v0.10.0-esp32-20200709\openocd-esp32\bin
C:\esp\tools\espressif\tools\ninja\1.10.0\
C:\esp\tools\espressif\tools\idf-exe\1.0.1\
C:\esp\tools\espressif\tools\ccache\3.7\
C:\esp\tools\espressif\tools\dfu-util\0.9\dfu-util-0.9-win64
C:\esp\esp-idf\tools

Checking if Python packages are up to date...
Python requirements from C:\esp\esp-idf\requirements.txt are satisfied.

Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

idf.py build

C:\esp\esp-idf>

```

Figure 6: Start ESP COMMAND LINE

2 Installing visualstudio code and node.js

2.1 Installing vscode

1. Navigate to <https://code.visualstudio.com/download>
2. Download setup
3. Install and add to path
4. Run vscode.exe and click on extension to add IntelliSense to VSCODE

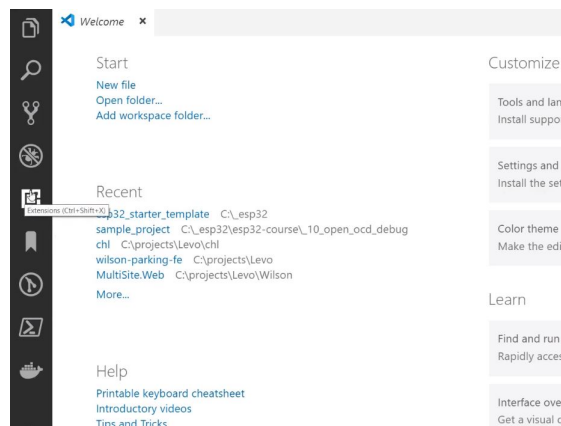


Figure 7: Navigate to extension

5. Select C/C++ IntelliSense and install it

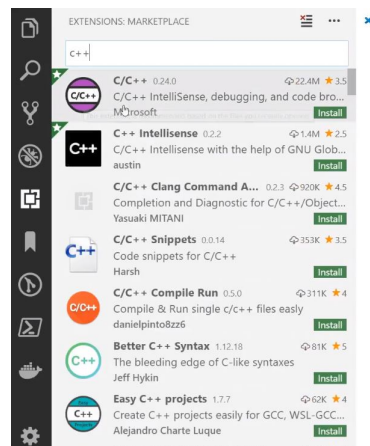


Figure 8: C/C++ IntelliSense

2.2 Installing node.js

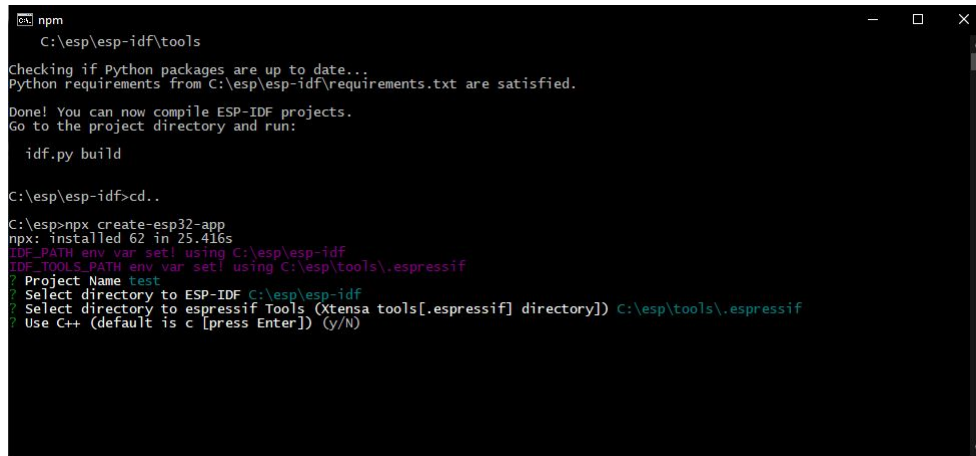
1. Navigate to nodejs.org/en/ and download the latest version of NODE.js
2. Install and add to path
3. To verify the NODE.js installation type `node -v` in CMD to check if it is successfully installed

Node:-

This helps in easy set-up of an esp32 project. Once node is installed, there is no need for installing additional packages manually. It installs all the packages necessary for the project.

2.3 Creating esp32 project

1. Open ESP-IDF Command Prompt.
2. cd to C:/esp
3. Enter npx create-esp32-app
4. Enter project name
5. Choose the directory where the ESP-IDF is available. Press TAB if the above procedures were followed.
6. Choose the directory where the tool chain is available. Press TAB if the above procedures were followed.
7. When asked to use C++ press enter since the default setting is C or choose NO
8. Click enter and the esp project with the name specified will be created
9. Navigate to the project directory and enter code . to open it in VS CODE



```
npm
C:\esp\esp-idf\tools
Checking if Python packages are up to date...
Python requirements from C:\esp\esp-idf\requirements.txt are satisfied.
Done! You can now compile ESP-IDF projects.
Go to the project directory and run:
    idf.py build

C:\esp\esp-idf>cd..

C:\esp>npx create-esp32-app
npx: installed 62 in 25.416s
IDF_PATH env var set! using C:\esp\esp-idf
IDF_TOOLS_PATH env var set! using C:\esp\tools\.espressif
? Project Name test
? Select directory to ESP-IDF C:\esp\esp-idf
? Select directory to espressif Tools (Xtensa tools[.espressif] directory) C:\esp\tools\.espressif
? Use C++ (default is c [press Enter]) (y/N)
```

Figure 9: Creating ESP32 APP

3 esp32s2_gpio.c

3.1 Initialize mains sense

The following code is used to configure the main sense pin as input GPIO pin

```

1 void init_mains_sense(void){
2     gpio_pad_select_gpio(MAINS_SENSE); /* Select the pin to configure */
3     gpio_set_direction(MAINS_SENSE, GPIO_MODE_INPUT); /* Set the mode of the pin to be
4     configured */
5     /* Enable pullup and disable pulldown */
6     gpio_pullup_en(MAINS_SENSE);
7     gpio_pullup_dis(MAINS_SENSE);
8 }

```

3.2 Initialize push button

The following code is used to configure the push button pin as input GPIO pin

```

1 void init_pushbutton(void){
2     gpio_pad_select_gpio(BUTTON); /* Select the pin to configure */
3     gpio_set_direction(BUTTON, GPIO_MODE_INPUT); /* Set the mode of the pin to be
4     configured */
5     /* Enable pullup and disable pulldown */
6     gpio_pullup_en(BUTTON);
7     /* Set interrupt type */
8     gpio_set_intr_type(BUTTON, GPIO_INTR_NEGEDGE);
9 }

```

3.3 Initialize S4 switch

The following code is used to configure the S4 switch pin as GPIO output pin

```

1 void S4_SW_init(void){
2     gpio_pad_select_gpio(S4_SW); /* Select the pin to configure */
3     gpio_set_direction(S4_SW, GPIO_MODE_OUTPUT); /* Set the mode of the pin to be
4     configured */
5     gpio_set_level(S4_SW, PIN_ON); /* Set output level */
6 }

```

3.4 Initialize LED

The following code initializes three pins as GPIO output for controlling LEDs

```

1 void init_led(void){
2     for(int i=0;i<3;i++){
3         gpio_pad_select_gpio(led_pins[i]); /* Select the pin to configure */
4         gpio_set_direction(led_pins[i], GPIO_MODE_OUTPUT); /* Set the mode of the pin to
5         be configured */
6         gpio_set_level(led_pins[i], LED_OFF); /* Set output level */
7     }
8 }

```

3.5 Initialize switches

The following code initializes four pins as GPIO output for controller solar mosfet, battery mosfet, mains relay and triac.

```

1 void init_switches(void){
2     for(int i=0;i<4;i++){
3         gpio_pad_select_gpio(switch_pins[i]); /* Select the pin to configure */
4         gpio_set_direction(switch_pins[i], GPIO_MODE_OUTPUT); /* Set the mode of the pin
5         to be configured */
6         gpio_set_level(switch_pins[i], LED_OFF); /* Set output level */
7         vTaskDelay(10/portTICK_PERIOD_MS); /* Cooling time */
8     }
9 }

```

3.6 Initialize I2C enable

The following code is used to initialize the I2C peripheral of the controller

```

1 void init_i2c_power(void){
2     gpio_pad_select_gpio(I2C_PW_EN); /* Select the pin to configure */
3     gpio_set_direction(I2C_PW_EN, GPIO_MODE_OUTPUT); /* Set the mode of the pin to be
4     configured */
5     gpio_set_level(I2C_PW_EN, PIN_ON); /* Set output level */
6 }
```

3.7 Initialize buzzer

The following code is used to initialize the GPIO pin which controls the buffer as GPIO output

```

1 void init_buzzer(void){
2     gpio_pad_select_gpio(BUZZER); /* Select the pin to configure */
3     gpio_set_direction(BUZZER, GPIO_MODE_OUTPUT); /* Set the mode of the pin to be
4     configured */
5     gpio_set_level(BUZZER, PIN_OFF); /* Set output level */
6     buzzer_state = 0;
7 }
```

3.8 Buzzer control

The following code is used to turn on and off the buzzer

```

1 void buzzer_control(uint8_t buzzer_mode){
2     buzzer_state = buzzer_mode;
3     gpio_set_level(BUZZER, buzzer_mode);
4 }
```

3.9 Toggle buzzer

The following code is used to toggle the buzzer

```

1 void toggle_buzzer(void) {
2     buzzer_state ^= 1;
3     buzzer_control(buzzer_state);
4 }
```

3.10 Mains sense

The following code returns the state of the mains sense pin

```

1 uint8_t mains_sense(void){
2     return gpio_get_level(MAINS_SENSE);
3 }
```

3.11 Battery mosfet switch

The following code is used to set the state of the battery MOSFET

```

1 /*
2  * @Info:    Sets the level of battery mosfet
3  * @param:  On or Off
4  * @Return:  None
5  */
6
7 void bat_mos_sw(uint8_t status){
8     gpio_set_level(BAT_MOS_SW, status);
9 }
```

3.12 Solar mosfet switch

The following code is used to set the state of the solar MOSFET

```

1  /*
2  * @Info:   Sets the level of solar mosfet
3  * @param:  On or Off
4  * @Return: None
5  */
6
7  void sol_mos_sw(uint8_t status){
8      gpio_set_level(SOL_MOS_SW,status);
9  }
```

3.13 Set triac level

The following code is used to set the state of the triac

```

1  /*
2  * @Info:   Sets the level of triac
3  * @param:  On or Off
4  * @Return: None
5  */
6
7  void mains_triac_sw(uint8_t status){
8      gpio_set_level(TRIAC_SW,status);
9  }
```

3.14 Set mains relay

The following code is used to set the state of the RELAY

```

1  /*
2  * @Info:   Sets the level of mains relay
3  * @param:  On or Off
4  * @Return: None
5  */
6
7  void mains_rel_sw(uint8_t status){
8      gpio_set_level(M_RELAY,status);
9  }
```

3.15 Set S4 relay

The following code is used to set the state of the S4 relay

```

1  /*
2  * @Info:   Sets the level of S4 relay
3  * @param:  On or Off
4  * @Return: None
5  */
6
7  void s4_rel_sw(uint8_t status){
8      gpio_set_level(S4_SW,status);
9  }
```

3.16 Turn on battery mosfet

The following code is used to turn on the battery mosfet

```

1  /*
2  * @Info:   Sets the level of battery mosfet to ON
3  * @param:  None
4  * @Return: None
5  */
6
7  void bat_mos_sw_on(){
8      bat_mos_sw(1);
9      switch_state |= BAT_SW;
10 }
```

3.17 Turn off battery mosfet

The following code is used to turn off the battery mosfet

```

1  /*
2  * @Info:   Sets the level of battery mosfet to OFF
3  * @param:  None
4  * @Return: None
5  */
6
7  void bat_mos_sw_off(){
8      bat_mos_sw(0);
9      switch_state &= ~BAT_SW;
10 }

```

3.18 Turn on solar mosfet

The following code is used to turn on the solar mosfet

```

1  /*
2  * @Info:   Sets the level of solar mosfet to ON
3  * @param:  None
4  * @Return: None
5  */
6
7  void sol_mos_sw_on(){
8      sol_mos_sw(1);
9      switch_state |= SOL_SW;
10 }

```

3.19 Turn off solar mosfet

The following code is used to turn off the solar mosfet

```

1  /*
2  * @Info:   Sets the level of solar mosfet to OFF
3  * @param:  None
4  * @Return: None
5  */
6
7  void sol_mos_sw_off(){
8      sol_mos_sw(0);
9      switch_state &= ~SOL_SW;
10 }

```

3.20 Turn on mains relay

The following code is used to turn on the mains relay

```

1  /*
2  * @Info:   Sets the level of mains relay to ON
3  * @param:  None
4  * @Return: None
5  */
6  void mains_rel_sw_on(){
7      if(!(switch_state & MAINS_SW)){
8          mains_triac_sw(1);
9          vTaskDelay(50);
10         mains_rel_sw(0);
11         vTaskDelay(50);
12         mains_triac_sw(0);
13         switch_state |= MAINS_SW;
14     }
15 }
16 }

```


3.21 Turn off mains relay

The following code is used to turn off the mains relay

```

1  /*
2  * @Info:   Sets the level of mains relay to OFF
3  * @param:  None
4  * @Return: None
5  */
6  void mains_rel_sw_off(){
7      if((switch_state & MAINS_SW)){
8          mains_triac_sw(1);
9          vTaskDelay(50);
10         mains_rel_sw(1);
11         vTaskDelay(50);
12         mains_triac_sw(0);
13         switch_state &= ~MAINS_SW;
14     }
15 }

```

3.22 Set device state

The following code is used to set the state of the device

```

1  /*
2  * @Info:   Set the switch state in state change algorithm
3  * @param:  switch state; 1,3,4,5,6 or 7
4  * @return: None
5  */
6  void turtle_set_sw(uint8_t sw_stat){
7      if(sw_stat & MAINS_SW){
8          mains_rel_sw_on();
9      }
10     else{
11         mains_rel_sw_off();
12     }
13
14     if(sw_stat & BAT_SW){
15         bat_mos_sw_on();
16     }
17     else{
18         bat_mos_sw_off();
19     }
20
21     if(sw_stat & SOL_SW){
22         sol_mos_sw_on();
23     }
24     else{
25         sol_mos_sw_off();
26     }
27
28     #if iCUBE
29     if(((algo_param.dev_algo_state == ALGO_STATE_SOL_INV_DIS_7)|| (algo_param.
30         dev_algo_state == ALGO_STATE_INV_5))){
31         s4_rel_sw(1);
32     }
33     else{
34         s4_rel_sw(0);
35     }
36     #endif
37 }

```

3.23 Transit to state 5 from state 4

The following code is used to transit the device from state 3 and state 1 to state 5. Mains has to be turned off before the device is set in state 5 to protect the inverter.

```

1  /*
2  * @Info:   Transit to state 4 then to state 5;
3  *           To be used when changing from state1 or state 3 to state 5
4  * @Param:  None

```

```

5  * @Return: None
6  */
7  void switch_state_4_5(void){
8      algo_param.dev_algo_state = ALGO_STATE_BAT_FRC_TRIP_4;
9      turtle_set_sw(algo_param.dev_algo_state);
10     vTaskDelay(500);
11     algo_param.dev_algo_state = ALGO_STATE_INV_5;
12     turtle_set_sw(algo_param.dev_algo_state);
13 }

```

3.24 Led blink

The following code is used for blinking the LED

```

1  /*
2  * @Info:   Algorithm to blink led;
3  * @Param:  None; Uses global variables
4  * @Return: None
5  */
6  void led_blink(void){
7      blink_time+=1;
8      if(blink_time>blink_seconds){
9          blink_time = 0;
10     }
11     if(blink_time<=1){
12         led_control(blink_colour);
13     }
14     else{
15         led_control(led_no_color);
16     }
17 }

```

3.25 Set blink condition

The following code is used to set the condition for the led blink

```

1  /*
2  * @Info:   Set the global variables which is used in led_blink function
3  * @Param:  Led colour and led blink seconds
4  * @Return: None
5  */
6  void blink(led_color_t colour,uint16_t seconds){
7      blink_colour = colour;
8      blink_seconds = seconds;
9  }

```

3.26 Device initialization

Following code is used to initialize the algorithm

```

1  /*
2  * @Info: Initialization sequence;
3  *       1. LED blink; device turn-on buzzer sequence
4  *       2. Check memory available; Check calibration status
5  *       3. Turn on state change algorithm
6  */
7  void power_on_initialization(void){
8      uint16_t crc_check;
9      bat_mos_sw_on();
10     sol_mos_sw_off();
11     mains_rel_sw_on();
12     algo_param.dev_algo_state=ALGO_STATE_INV_5;
13     buzzer_control(BUZZER_OFF);
14     play_tone2();
15     led_control(led_no_color);
16     vTaskDelay(100/portTICK_RATE_MS);
17     led_control(led_red_color);
18     vTaskDelay(500/portTICK_RATE_MS);
19     led_control(led_green_color);
20     vTaskDelay(500/portTICK_RATE_MS);

```

```

21 led_control(led_blue_color);
22 vTaskDelay(500/portTICK_RATE_MS);
23 led_control(led_no_color);
24 is_day_flag = NOW_NIGHT_TIME;
25 select_i2c_memory(eeprom_addr);
26 if(check_memory()==0){
27     post_error(ERR_NO_MEMORY);
28 }
29 if(!error_flag){
30     ESP_LOGI("Power on initialization","Inside parameter initialization");
31     load_wifi_param();
32     init_algo();
33     #if 1
34         if(i2c_eeprom_read_int32(EЕ_CALIB_DONE_ADDR) != CALIB_CONFIG_DONE_VALUE){
35             post_error(ERR_NOT_CALIB);
36         }
37         crc_check = calc_update_eeprom_crc(EЕ_CALIB_DAT_START_ADDR, CALIB_CONST_LEN,
EE_CALIB_CRC_ADDR, 0);
38         vTaskDelay(100/portTICK_PERIOD_MS);
39         ESP_LOGI("Calibraton check","Completed");
40         if(crc_check != i2c_eeprom_read_int16(EЕ_CALIB_CRC_ADDR)){
41             post_error(ERR_NOT_CALIB);
42         }
43         ESP_LOGI("Calibraton CRC check","Completed");
44         if(i2c_eeprom_read_int32(EЕ_CONFIG_DONE_ADDR) != CALIB_CONFIG_DONE_VALUE){
45             post_error(ERR_NOT_CONFIG);
46         }
47         ESP_LOGI("Configuration check","Completed");
48         crc_check = calc_update_eeprom_crc(EЕ_CONFIG_DAT_START_ADDR, CONFIG_PARAM_LEN,
EE_CONFIG_CRC_ADDR, 0);
49         vTaskDelay(100/portTICK_PERIOD_MS);
50         if(crc_check != i2c_eeprom_read_int16(EЕ_CONFIG_CRC_ADDR)){
51             post_error(ERR_NOT_CONFIG);
52         }
53         ESP_LOGI("Configuration CRC heck","Completed");
54     #endif
55 }
56 show_led();
57 ESP_LOGI("Power on initialization","End of initialization");
58 mains_rel_sw_off();
59 vTaskDelay(2000/portTICK_RATE_MS);
60 mains_rel_sw_on();
61 vTaskDelay(500/portTICK_RATE_MS);
62 }

```

4 esp32s2_adc.c

4.1 Variables used in esp32s2_adc.c

The following are the variables used in the adc.c file

```
1 const uint8_t adc_ch_array[6] = {ADC_INV_V, ADC_SOL_V, ADC_LOAD_I, ADC_TEMP_1, ADC_V_REF
  , ADC_TEMP_2};
2 float inverter_vol=0.0,temperature1=0.0,temperature2=0.0,vol_ref=0.0;
3 uint8_t mains=0;
4 extern uint8_t i2cFlag;
```

4.2 Initialize ADC

The following code is used to initialize the ADC pins in the ESP32S2 controller.

```
1 /*
2  * @Info: The following code is used to initialize 6 ADC pins as 13bit A2B with
  attenuation of 11DB
3  * Param: None
4  * Return: None
5  */
6 void init_adc(void){
7     adc1_config_width(ADC_WIDTH_BIT_13);
8     for(int i=0;i<6;i++){
9         adc1_config_channel_atten(adc_ch_array[i], ADC_ATTEN_DB_11);
10    }
11 }
```

4.3 Software ADC conversion

The following code is used to trigger software sequential ADC conversion in ESP32S2

```
1 /*
2  * @Info: Initiate A2D conversion on 6 channels
3  * @Param: Buffer to store the converted data
4  * @Return: None
5  */
6 void get_uc_adc_all(uint8_t *adc_buff){
7     uint8_t adc_count, buff_idx;
8     uint16_t adc_val;
9     for(adc_count = 0, buff_idx = 0; adc_count < 6; adc_count++, buff_idx += 2){
10         adc_val = adc1_get_raw(adc_ch_array[adc_count]);
11         adc_buff[buff_idx] = ((adc_val >> 8) & 0xFF);
12         adc_buff[buff_idx + 1] = (adc_val & 0xFF);
13         vTaskDelay(10/portTICK_PERIOD_MS);
14     }
15 }
```

4.4 Get battery voltage from PAC

The following code is used to get the battery voltage value from the PAC

```
1 /*
2  * @Info: Poll PAC for battery voltage
3  * @Param: None
4  * @Return: Battery voltage
5  */
6 float get_bat_vol(){
7     uint8_t _adc_buff[2];
8     int8_t pac_stat;
9     uint16_t uint_pac_reg;
10    float bat_v;
11    while(i2cFlag){};
12    i2cFlag=1;
13    pac1720_busy();
14    pac_stat = pac720_read_buff(PAC1720_CH1_VSOURCE_HIGH_ADDR, _adc_buff , 2);
15    i2cFlag=0;
16    if(pac_stat != 0){
17        return 0;
```

```

18 }
19 uint_pac_reg = (((uint16_t)_adc_buff[0] << 8) | _adc_buff[1]); // CH1 source
20 uint_pac_reg = uint_pac_reg >> 5;
21 bat_v = (float)uint_pac_reg * pac_bat_v_const;
22 // ESP_LOGI(TAG,"Battery Voltage is %f",bat_v);
23 return bat_v;
24 }

```

4.5 Read digital values from sensor

The following code is used to collect digital values of sensors interfaced with the device

```

1 /*
2  * @Info: Obtain data from all sensors as digital
3  * @Param: Buffer to store the converted values
4  * @Return: Success/Failure
5  */
6 int8_t read_adc_all(uint8_t *adc_all_buff){
7     /*0 to 7 PAC; 8 to 13 ADC*/
8     int8_t pac_stat;
9     int16_t int_pac_reg;
10    pac1720_busy();
11    pac_stat = pac720_read_buff(PAC1720_CH1_VSENSE_HIGH_ADDR, adc_all_buff , 8);
12    if(pac_stat != 0){
13        return 0;
14    }
15    int_pac_reg = (((int16_t)adc_all_buff[0] << 8) | adc_all_buff[1]); // CH1 sense
16    int_pac_reg = int_pac_reg >> 4;
17    if(int_pac_reg & Bit(11)){ // convert 11 bit signed int to 16 bit signed int
18        int_pac_reg |= 0xF000;
19    }
20    adc_all_buff[0] = ((int_pac_reg >> 8) & 0xFF);
21    adc_all_buff[1] = ( int_pac_reg & 0xFF);
22    int_pac_reg = (((int16_t)adc_all_buff[2] << 8) | adc_all_buff[3]); // CH2 sense
23    int_pac_reg = int_pac_reg >> 4;
24    if(int_pac_reg & Bit(11)){ // convert 11 bit signed int to 16 bit signed int
25        int_pac_reg |= 0xF000;
26    }
27    adc_all_buff[2] = ((int_pac_reg >> 8) & 0xFF);
28    adc_all_buff[3] = ( int_pac_reg & 0xFF);
29    int_pac_reg = (((int16_t)adc_all_buff[4] << 8) | adc_all_buff[5]); // CH1 source
30    int_pac_reg = int_pac_reg >> 5;
31    adc_all_buff[4] = ((int_pac_reg >> 8) & 0xFF);
32    adc_all_buff[5] = ( int_pac_reg & 0xFF);
33    int_pac_reg = (((int16_t)adc_all_buff[6] << 8) | adc_all_buff[7]); // CH2 source
34    int_pac_reg = int_pac_reg >> 5;
35    adc_all_buff[6] = ((int_pac_reg >> 8) & 0xFF);
36    adc_all_buff[7] = ( int_pac_reg & 0xFF);
37    get_uc_adc_all(&adc_all_buff[8]);
38    return 1;
39 }

```

4.6 Get analog values from sensor

The following code is used to collect analog values from the sensor interfaced with the device

```

1 /*
2  * @Info: Obtain sensor values as analog values
3  * @Param: Moving average or not
4  * @Return: Success/Failure
5  */
6 int8_t get_sensor_values(uint8_t is_flush, uint8_t is_tx){
7     while(i2cFlag){};
8     i2cFlag=1;
9     static uint32_t sol_v_sum, load_i_sum, inv_v_sum, temp1_sum, vref_sum, temp2_sum;
10    uint16_t _sol_v, _temp1, _temp2, _load_i, _inv_v, _v_ref;
11    uint16_t moving_avg;
12    uint8_t _adc_buff[12];
13    int8_t pac_stat;
14    uint16_t uint_pac_reg;
15    int16_t int_pac_reg;

```

```

16 float float_pac_val;
17 if(test_error(ERR_NOT_CALIB)){
18     memset((uint8_t *)&algo_param, 0, SUMMARY_LEN);
19 }
20 else{
21     get_uc_adc_all(_adc_buff);
22     _inv_v = (_adc_buff[0] << 8) | _adc_buff[1];
23     _sol_v = (_adc_buff[2] << 8) | _adc_buff[3];
24     _load_i = (_adc_buff[4] << 8) | _adc_buff[5];
25     _temp1 = (_adc_buff[6] << 8) | _adc_buff[7];
26     _v_ref = (_adc_buff[8] << 8) | _adc_buff[9];
27     _temp2 = (_adc_buff[10] << 8) | _adc_buff[11];
28     if(is_flush){
29         inv_v_sum = (_inv_v << ADC_MUL_DIV_FACT);
30         sol_v_sum = (_sol_v << ADC_MUL_DIV_FACT);
31         load_i_sum = (_load_i << ADC_MUL_DIV_FACT);
32         temp1_sum = (_temp1 << ADC_MUL_DIV_FACT);
33         vref_sum = (_v_ref << ADC_MUL_DIV_FACT);
34         temp2_sum = (_temp2 << ADC_MUL_DIV_FACT);
35     }
36     else{
37         inv_v_sum += (_inv_v);
38         sol_v_sum += (_sol_v);
39         load_i_sum += (_load_i);
40         temp1_sum += (_temp1);
41         vref_sum += (_v_ref);
42         temp2_sum += (_temp2);
43     }
44     moving_avg = (inv_v_sum >> ADC_MUL_DIV_FACT); //Moving average for inverter vol
current
45     inv_v_sum -= moving_avg;
46     inverter_vol = ((float)moving_avg * uc_bat_v_const);
47
48     moving_avg = (sol_v_sum >> ADC_MUL_DIV_FACT); //Moving average for solar voltage
49     sol_v_sum -= moving_avg;
50     algo_param.cur_sol_v = (float)moving_avg * uc_sol_v_const;
51
52     moving_avg = (load_i_sum >> ADC_MUL_DIV_FACT); //Moving average for CT current
53     load_i_sum -= moving_avg;
54     algo_param.cur_ac_load_i = ((float)moving_avg / ct_load_i_const);
55
56     moving_avg = (temp1_sum >> ADC_MUL_DIV_FACT); //Moving average for temperature
57     temp1_sum -= moving_avg;
58     temperature1 = (float)moving_avg * uc_temp1_const; //(float)((float)moving_avg * ((
float)3.3/(float) 4096));
59
60     moving_avg = (vref_sum >> ADC_MUL_DIV_FACT); //Moving average for voltage
reference
61     vref_sum -= moving_avg;
62     vol_ref = (float) ((float)moving_avg*((float)3.3/(float)4096));
63
64     moving_avg = (temp2_sum >> ADC_MUL_DIV_FACT); //Moving average for temperature
65     temp2_sum -= moving_avg;
66     temperature2 = (float)moving_avg * uc_temp2_const; //(float)((float)moving_avg * ((
float)3.3/(float) 4096));
67
68     if(mains_sense()){
69         mains=1;
70     }
71     else{
72         mains=0;
73     }
74     pac1720_busy();
75     pac_stat = pac720_read_buff(PAC1720_CH1_VSENSE_HIGH_ADDR, _adc_buff , 12);
76     if(pac_stat != 0){
77         return 0;
78     }
79
80     uint_pac_reg = (((uint16_t)_adc_buff[8] << 8) | _adc_buff[9]); //CH1 power
81     pac_bat_chg_p = (float)uint_pac_reg * pac_bat_p_const;
82
83     uint_pac_reg = (((uint16_t)_adc_buff[10] << 8) | _adc_buff[11]); //CH2 power
84     pac_sol_p = (float)uint_pac_reg * pac_sol_p_const;

```

```

85
86     int_pac_reg = (((int16_t)_adc_buff[0] << 8) | _adc_buff[1]); // CH1 sense
87     int_pac_reg = int_pac_reg >> 4;
88     if(int_pac_reg & Bit(11)){ // convert 11 bit signed int to 16 bit signed int
89         int_pac_reg |= 0xF000;
90         float_pac_val = ((float)int_pac_reg * pac_bat_dis_i_const) -
pac_bat_dis_i_offset;
91         algo_param.cur_chg_i = 0.0;
92         algo_param.cur_dis_i = float_pac_val * -1;
93         if(algo_param.cur_dis_i < 0.0){
94             algo_param.cur_chg_i = float_pac_val;
95             algo_param.cur_dis_i = 0.0;
96             pac_bat_dis_p = 0.0;
97         }
98         else{
99             pac_bat_dis_p = pac_bat_chg_p;
100             pac_bat_chg_p = 0.0;
101         }
102     }
103     else{
104         float_pac_val = ((float)int_pac_reg * pac_bat_chg_i_const) -
pac_bat_chg_i_offset;
105         algo_param.cur_chg_i = float_pac_val;
106         algo_param.cur_dis_i = 0.0;
107         pac_bat_dis_p = 0.0;
108     }
109     int_pac_reg = (((int16_t)_adc_buff[2] << 8) | _adc_buff[3]); // CH2 sense
110     int_pac_reg = int_pac_reg >> 4;
111     if(int_pac_reg & Bit(11)){ // convert 11 bit signed int to 16 bit signed int
112         int_pac_reg |= 0xF000;
113     }
114     algo_param.cur_sol_i = ((float)int_pac_reg * pac_sol_i_const * 1) - pac_sol_i_offset;
115     if(algo_param.cur_sol_i < 0){
116         algo_param.cur_sol_i = 0.0;
117         pac_sol_p = 0.0;
118     }
119     uint_pac_reg = (((uint16_t)_adc_buff[4] << 8) | _adc_buff[5]); // CH1 source
120     uint_pac_reg = uint_pac_reg >> 5;
121     algo_param.cur_bat_v = (float)uint_pac_reg * pac_bat_v_const;
122
123     dc_load_i = (algo_param.cur_sol_i + algo_param.cur_dis_i) - algo_param.cur_chg_i;
124     if(dc_load_i < 0){
125         dc_load_i = 0;
126     }
127     ac_load_p = algo_param.cur_ac_load_i * AC_POWER_CONST;
128     if(pac_sol_p < pac_bat_chg_p){
129         mains_chg_p = (pac_bat_chg_p - pac_sol_p);
130         sol_bat_p = pac_sol_p;
131     }
132     else{
133         mains_chg_p = 0.0;
134         sol_bat_p = pac_bat_chg_p;
135     }
136
137     float_pac_val = (algo_param.cur_dis_i * LVD_COEFF);
138     bat_frc_trip_exit_v = (bat_frc_trip_exit_v_thr - float_pac_val);
139     bat_mains_chg_in_v = (bat_mains_chg_in_v_thr - float_pac_val);
140     sol_bat_chg_i_diff = (algo_param.cur_sol_i - algo_param.cur_chg_i);
141 }
142 i2cFlag=0;
143 return 1;
144 }

```

5 esp32s2_i2c.c

5.1 I2C initialization

The following code is used to initialize I2C peripheral of the ESP32S2 controller at 100kHz frequency

```

1 void init_i2c(void){
2     i2c_num = I2C_MASTER;
3     i2c_config_t i2c_config = {
4         .mode = I2C_MODE_MASTER,
5         .sda_io_num = SDA_GPIO,
6         .scl_io_num = SCL_GPIO,
7         .sda_pullup_en = GPIO_PULLUP_ENABLE,
8         .scl_pullup_en = GPIO_PULLUP_ENABLE,
9         .master.clk_speed = 100000};
10    i2c_param_config(I2C_MASTER, &i2c_config);
11    i2c_driver_install(I2C_MASTER, I2C_MODE_MASTER, 0, 0, 0);
12 }
```

5.2 I2C detect slave

The following code is used for detecting the slaves connected to the I2C bus

```

1 esp_err_t i2c_master_detect_slave(void){
2     i2c_cmd_handle_t cmd = i2c_cmd_link_create();
3     i2c_master_start(cmd);
4     i2c_master_write_byte(cmd, (i2c_slave_addr) | WRITE_BIT, ACK_CHECK_EN);
5     i2c_master_stop(cmd);
6     esp_err_t ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
7     i2c_cmd_link_delete(cmd);
8     return ret;
9 }
```

6 Header file of PAC

Note:- The library functions offered by microchip are used for coding. Some of the functions that are found in the codes are library functions. In order to have a proper understanding, look into the respective library functions. They are available at [Microchip\xc8\v1.34\sources\pic18\plib](#)

This chapter contains the macros and function declaration associated with PAC1720. A macro in computer science is a rule or pattern that specifies how a certain input sequence should be mapped to a replacement output sequence according to a defined procedure. A function declaration introduces an identifier that designates a function and, optionally, specifies the types of the function parameters (the prototype).

6.1 Function definitions associated with PAC

```

1 #ifndef __PAC1720_H__
2 #define __PAC1720_H__
3
4 extern int8_t pac1720_busy();
5 extern void init_pac1720();
6 extern int8_t pac1720_write_reg16(uint8_t reg_addr, uint16_t reg_val);
7 extern int8_t pac1720_write_reg8(uint8_t reg_addr, uint8_t reg_val);
8 extern int8_t pac1720_read_reg16(uint8_t reg_addr, uint16_t *reg_val);
9 extern int8_t pac1720_read_reg8(uint8_t reg_addr, uint8_t *reg_val);
10 extern int8_t pac1720_read_buff(uint16_t reg_addr, uint8_t *reg_buff, uint8_t reg_len);
11 extern void test_pac1720();
12
13 #endif /* __PAC1720_H__ */
```


6.2 Macros associated with PAC

```

1  #ifndef __PAC1720_H__
2  #define __PAC1720_H__
3
4
5  #include <base_types.h>
6  #include <xc.h>
7  #include <stdio.h>
8
9  #include <pic18_f47j53_i2c.h>
10 #include <pic18_f47j53_gpio.h>
11 #include <pic18_f47j53_clk.h>
12
13 #define PAC1720_ADDR 0x52
14
15
16
17
18 #define PAC1720_CH1_V_DENO      2047.0
19 #define PAC1720_CH2_V_DENO      2047.0
20
21 #define PAC1720_CH1_I_DENO      2047.0
22 #define PAC1720_CH2_I_DENO      2047.0
23
24 #define PAC1720_CH1_FSC      90.28    // ((IBUS * PAC1720_CH1_I_DENO) / VSENSE)
25 #define PAC1720_CH2_FSC      22.75    // ((IBUS * PAC1720_CH2_I_DENO) / VSENSE)
26
27 #define PAC1720_CH1_FSV      (40.0 - (40.0 / PAC1720_CH1_V_DENO))
28 #define PAC1720_CH2_FSV      (40.0 - (40.0 / PAC1720_CH2_V_DENO))
29 #if 1
30 #define PAC1720_CH1_V_CONST      (PAC1720_CH1_FSV / PAC1720_CH1_V_DENO)
31 #define PAC1720_CH2_V_CONST      (PAC1720_CH2_FSV / PAC1720_CH2_V_DENO)
32
33 #define PAC1720_CH1_I_CONST      (PAC1720_CH1_FSC / PAC1720_CH1_I_DENO)
34 #define PAC1720_CH2_I_CONST      (PAC1720_CH2_FSC / PAC1720_CH2_I_DENO)
35
36 #define PAC1720_CH1_P_CONST      ((PAC1720_CH1_FSV * PAC1720_CH1_FSC) / 65535)
37 #define PAC1720_CH2_P_CONST      ((PAC1720_CH2_FSV * PAC1720_CH2_FSC) / 65535)
38 #endif
39
40 #define PAC1720_CONFIG_ADDR      0x00
41 #define PAC1720_CONVERSION_RATE_ADDR      0x01
42 #define PAC1720_ONE_SHOT_ADDR      0x02
43 #define PAC1720_CH_MASK_REG_ADDR      0x03
44 #define PAC1720_HIGH_LIMIT_STATUS_ADDR      0x04
45 #define PAC1720_LOW_LIMIT_STATUS_ADDR      0x05
46
47 #define PAC1720_V_SOURCE_SAMP_CONFIG_ADDR      0x0A
48
49 #define PAC1720_CH1_VSENSE_SAMP_CONFIG_ADDR      0x0B
50 #define PAC1720_CH2_VSENSE_SAMP_CONFIG_ADDR      0x0C
51
52 #define PAC1720_CH1_VSENSE_HIGH_ADDR      0x0D
53 #define PAC1720_CH1_VSENSE_LOW_ADDR      0x0E
54 #define PAC1720_CH2_VSENSE_HIGH_ADDR      0x0F
55 #define PAC1720_CH2_VSENSE_LOW_ADDR      0x10
56
57 #define PAC1720_CH1_VSOURCE_HIGH_ADDR      0x11
58 #define PAC1720_CH1_VSOURCE_LOW_ADDR      0x12
59 #define PAC1720_CH2_VSOURCE_HIGH_ADDR      0x13
60 #define PAC1720_CH2_VSOURCE_LOW_ADDR      0x14
61
62 #define PAC1720_CH1_PWR_RAT_HIGH_ADDR      0x15
63 #define PAC1720_CH1_PWR_RAT_LOW_ADDR      0x16
64 #define PAC1720_CH2_PWR_RAT_HIGH_ADDR      0x17
65 #define PAC1720_CH2_PWR_RAT_LOW_ADDR      0x18
66
67 #define PAC1720_CH1_VSENSE_LIMIT_HIGH_ADDR      0x19
68 #define PAC1720_CH2_VSENSE_LIMIT_HIGH_ADDR      0x1A
69 #define PAC1720_CH1_VSENSE_LIMIT_LOW_ADDR      0x1B
70 #define PAC1720_CH2_VSENSE_LIMIT_LOW_ADDR      0x1C
71

```

```

72 #define PAC1720_CH1_VSOURCE_LIMIT_HIGH_ADDR 0x1D
73 #define PAC1720_CH2_VSOURCE_LIMIT_HIGH_ADDR 0x1E
74 #define PAC1720_CH1_VSOURCE_LIMIT_LOW_ADDR 0x1F
75 #define PAC1720_CH2_VSOURCE_LIMIT_LOW_ADDR 0x20
76
77 #define PAC1720_PRODUCT_ID_ADDR 0xFD
78 #define PAC1720_MANUFACTURER_ID_ADDR 0xFE
79 #define PAC1720_REVISION_ADDR 0xFF
80
81
82
83 #define PAC1720_UPDATE_1_PER_SEC (0x00)
84 #define PAC1720_UPDATE_2_PER_SEC (0x01)
85 #define PAC1720_UPDATE_4_PER_SEC (0x02)
86 #define PAC1720_UPDATE_CONT (0x03) // Default
87
88 #define PAC1720_CH2_VSOURCE_SAMP_2_5MS (0x00 << 6)
89 #define PAC1720_CH2_VSOURCE_SAMP_5MS (0x01 << 6)
90 #define PAC1720_CH2_VSOURCE_SAMP_10MS (0x02 << 6) // Default
91 #define PAC1720_CH2_VSOURCE_SAMP_20MS (0x03 << 6)
92
93 #define PAC1720_CH2_VSOURCE_AVG_DIS (0x00 << 4) //Default
94 #define PAC1720_CH2_VSOURCE_AVG_2 (0x01 << 4)
95 #define PAC1720_CH2_VSOURCE_AVG_4 (0x02 << 4)
96 #define PAC1720_CH2_VSOURCE_AVG_8 (0x03 << 4)
97
98 #define PAC1720_CH1_VSOURCE_SAMP_2_5MS (0x00 << 2)
99 #define PAC1720_CH1_VSOURCE_SAMP_5MS (0x01 << 2)
100 #define PAC1720_CH1_VSOURCE_SAMP_10MS (0x02 << 2) // Default
101 #define PAC1720_CH1_VSOURCE_SAMP_20MS (0x03 << 2)
102
103 #define PAC1720_CH1_VSOURCE_AVG_DIS (0x00) //Default
104 #define PAC1720_CH1_VSOURCE_AVG_2 (0x01)
105 #define PAC1720_CH1_VSOURCE_AVG_4 (0x02)
106 #define PAC1720_CH1_VSOURCE_AVG_8 (0x03)
107
108 #define PAC1720_VSENSE_SAMP_2_5MS (0x00 << 4)
109 #define PAC1720_VSENSE_SAMP_5MS (0x01 << 4)
110 #define PAC1720_VSENSE_SAMP_10MS (0x02 << 4)
111 #define PAC1720_VSENSE_SAMP_20MS (0x03 << 4)
112 #define PAC1720_VSENSE_SAMP_40MS (0x04 << 4)
113 #define PAC1720_VSENSE_SAMP_80MS (0x05 << 4) //Default
114 #define PAC1720_VSENSE_SAMP_160MS (0x06 << 4)
115 #define PAC1720_VSENSE_SAMP_320MS (0x07 << 4)
116
117 #define PAC1720_VSENSE_AVG_DIS (0x00 << 2) //Default
118 #define PAC1720_VSENSE_AVG_2 (0x01 << 2)
119 #define PAC1720_VSENSE_AVG_4 (0x02 << 2)
120 #define PAC1720_VSENSE_AVG_8 (0x03 << 2)
121
122 #define PAC1720_VSENSE_RANGE_10MV (0x00)
123 #define PAC1720_VSENSE_RANGE_20MV (0x01)
124 #define PAC1720_VSENSE_RANGE_40MV (0x02)
125 #define PAC1720_VSENSE_RANGE_80MV (0x04) //Default
126
127
128 #define pac1720_i2c_start i2c2_start
129 #define pac1720_i2c_write i2c2_write
130 #define pac1720_i2c_stop i2c2_stop
131 #define pac1720_i2c_read i2c2_read
132 #define pac1720_i2c_restart i2c2_restart
133 #define pac1720_i2c_send_nack i2c2_send_nack
134 #define pac1720_i2c_send_ack i2c2_send_ack
135 #endif /* __PAC1720_H__ */

```

7 PAC1720 Current and Power Measurement Device

7.1 Initialization of PAC1720

The PAC1720 has to be initialized before using it in the device. The current sense is sampled at 80msec, collecting 8 samples for average with a resolution of 40mV. The voltage sense is sampled at 20msec,

collecting 8 samples for average.

```
1 void
2 init_pac1720()
3 {
4
5     pac1720_busy();
6     // pac1720_write_reg8(PAC1720_CH1_VSENSE_SAMP_CONFIG_ADDR, 0x5E);
7     pac1720_write_reg8(PAC1720_CH1_VSENSE_SAMP_CONFIG_ADDR, \
8         (PAC1720_VSENSE_SAMP_80MS | PAC1720_VSENSE_AVG_8 | PAC1720_VSENSE_RANGE_40MV
9         ));
10    pac1720_busy();
11    // pac1720_write_reg8(PAC1720_CH2_VSENSE_SAMP_CONFIG_ADDR, 0x5E);
12    pac1720_write_reg8(PAC1720_CH2_VSENSE_SAMP_CONFIG_ADDR, \
13        (PAC1720_VSENSE_SAMP_80MS | PAC1720_VSENSE_AVG_8 | PAC1720_VSENSE_RANGE_40MV
14        ));
15    pac1720_busy();
16    // pac1720_write_reg8(PAC1720_V_SOURCE_SAMP_CONFIG_ADDR, 0xFF);
17    pac1720_write_reg8(PAC1720_V_SOURCE_SAMP_CONFIG_ADDR, \
18        (PAC1720_CH2_VSOURCE_SAMP_20MS | PAC1720_CH2_VSOURCE_AVG_8 | \
19        PAC1720_CH1_VSOURCE_SAMP_20MS | PAC1720_CH1_VSOURCE_AVG_8));
20 }
```

7.2 Check PAC availability

The code detects the presence of PAC in the device. If PAC is detected in the first time of checking, the code exits else it checks for 5 times. If PAC is damaged or not present, 0 is returned.

```

1  /**
2  ** Check whether the pac1720 is busy
3  ** Arguments :
4  ** None
5  ** Returns :
6  ** 0 - device not busy
7  ** -1 - if there was a write collision
8  ** -2 - if negative acknowledgement received
9  **/
10 int8_t
11 pac1720_busy()
12 {
13     int8_t _pac1720_stat;
14     uint8_t num_retry = 0;
15
16     while(1)
17     {
18         soft_delay_ms(1);
19         pac1720_i2c_start();
20         _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR);
21
22         pac1720_i2c_stop();
23         num_retry++;
24
25         if((num_retry > 5) || (_pac1720_stat == 0))
26         {
27             break;
28         }
29     }
30     return (_pac1720_stat);
31 }
32

```

7.3 Write 2bytes into the PAC

The code is used to configure the registers in the PAC. These codes are used in the initialization codes of PAC. The code is used to write a 16bit register.

```

1  int8_t
2  pac1720_write_reg16(uint8_t reg_addr, uint16_t reg_val)
3  {
4      uint8_t _reg_val_byte;
5      int8_t _pac1720_stat;
6
7      _reg_val_byte = ((reg_val >> 8) & 0xFF); // MSB
8
9      pac1720_i2c_start();
10     _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR);
11
12     if(_pac1720_stat < 0)
13     {
14         pac1720_i2c_stop();
15         return (_pac1720_stat);
16     }
17
18     _pac1720_stat = pac1720_i2c_write(reg_addr);
19
20     _pac1720_stat = pac1720_i2c_write(_reg_val_byte);
21
22     _reg_val_byte = (reg_val & 0xFF); // LSB
23     _pac1720_stat = pac1720_i2c_write(_reg_val_byte);
24
25     pac1720_i2c_stop();
26
27     return (_pac1720_stat);
28 }
29

```

7.4 Write 1byte into the PAC

The code is used to configure the registers in the PAC. These codes are used in the initialization codes of PAC. The code is used to write a 8bit register.

```

1 int8_t
2 pac1720_write_reg8(uint8_t reg_addr, uint8_t reg_val)
3 {
4     int8_t _pac1720_stat;
5
6
7     pac1720_i2c_start();
8     _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR);
9
10    if(_pac1720_stat < 0)
11    {
12        pac1720_i2c_stop();
13        return (-2);
14    }
15
16    _pac1720_stat = pac1720_i2c_write(reg_addr);
17
18    if(_pac1720_stat < 0)
19    {
20        pac1720_i2c_stop();
21        return (-3);
22    }
23
24    _pac1720_stat = pac1720_i2c_write(reg_val);
25
26    if(_pac1720_stat < 0)
27    {
28        pac1720_i2c_stop();
29        return (-4);
30    }
31    pac1720_i2c_stop();
32    return (_pac1720_stat);
33 }
34

```

7.5 Read 2bytes from the PAC

The code is used to read a 16bit register from PAC.

```

1 int8_t
2 pac1720_read_reg16(uint8_t reg_addr, uint16_t *_reg_val)
3 {
4     int8_t _pac1720_stat;
5
6     uint8_t reg_msb, reg_lsb;
7
8     pac1720_i2c_start();
9     _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR);
10
11    if(_pac1720_stat < 0)
12    {
13        // led_on(RED);
14        pac1720_i2c_stop();
15        return (_pac1720_stat);
16    }
17
18
19
20    _pac1720_stat = pac1720_i2c_write(reg_addr);
21
22    pac1720_i2c_restart();
23    _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR + 1);
24
25    reg_msb = pac1720_i2c_read();
26    pac1720_i2c_send_ack();
27
28    reg_lsb = pac1720_i2c_read();
29    pac1720_i2c_send_nack();
30

```

```
30 pac1720_i2c_stop();
31
32 *_reg_val = (((uint16_t)reg_msb << 8) | reg_lsb);
33
34 return (_pac1720_stat);
35
36 }
```

7.6 Read 1byte from the PAC

The code is used to read a 8bit register from PAC.

```
1 int8_t
2 pac1720_read_reg8(uint8_t reg_addr, uint8_t *_reg_val)
3 {
4     int8_t _pac1720_stat;
5
6
7     pac1720_i2c_start();
8     _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR);
9
10    if(_pac1720_stat < 0)
11    {
12        // led_on(RED);
13        pac1720_i2c_stop();
14        return (_pac1720_stat);
15    }
16
17
18
19    _pac1720_stat = pac1720_i2c_write(reg_addr);
20
21    pac1720_i2c_restart();
22    _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR + 1);
23
24    *_reg_val = pac1720_i2c_read();
25    pac1720_i2c_send_nack();
26
27    pac1720_i2c_stop();
28
29    return (_pac1720_stat);
30 }
```

7.7 Read data buffer from the PAC

The code is used to read consecutive data from PAC.

```
1 int8_t
2 pac720_read_buff(uint16_t reg_addr, uint8_t *reg_buff, uint8_t reg_len)
3 {
4     uint8_t _reg_count;
5     int8_t _pac1720_stat;
6
7     pac1720_i2c_start();
8
9
10    _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR);
11
12    if(_pac1720_stat < 0)
13    {
14        pac1720_i2c_stop();
15        return (_pac1720_stat);
16    }
17
18    _pac1720_stat = pac1720_i2c_write(reg_addr);
19
20    pac1720_i2c_restart();
21
22    _pac1720_stat = pac1720_i2c_write(PAC1720_ADDR + 1);
23
24    for(_reg_count = 0; _reg_count < (reg_len - 1); _reg_count++)
25    {
26        reg_buff[_reg_count] = pac1720_i2c_read();
27        pac1720_i2c_send_ack();
28    }
29
30    reg_buff[_reg_count] = pac1720_i2c_read();
31
32    pac1720_i2c_send_nack();
33
34    pac1720_i2c_stop();
35
36    return (_pac1720_stat);
37 }
```

8 EEPROM

8.1 Select I2C memory

The following code is used to specify the address of the EEPROM

```

1  /*
2  * @Info:   Select the I2C memory to communicate with
3  * @Param:  I2C memory address
4  * @Return: None
5  */
6  void select_i2c_memory(uint8_t i2c_memory_addr){
7      i2c_slave_addr = i2c_memory_addr;
8  }
```

8.2 Check EEPROM communication

The following code is used to verify if the I2C line is busy

```

1  /*
2  * @Info:   Check EEPROM communication busy
3  * @Param:  None
4  * @Return: None
5  */
6
7  esp_err_t ee24lc256_busy(void){
8      select_i2c_memory(eeprom_addr);
9      return i2c_master_detect_slave();
10 }
```

8.3 Read a byte from eeprom

The following code is used to read a byte from the EEPROM

```

1  /*
2  * @Info:   Read a byte from EEPROM
3  * @Param:  Memory location to read
4  * @Return: 8bit data
5  */
6  uint8_t ee24lc256_read_byte(uint16_t memory_location){
7      select_i2c_memory(eeprom_addr);
8      uint8_t data_rd=0;
9      int ret;
10     i2c_cmd_handle_t cmd = i2c_cmd_link_create();
11     i2c_master_start(cmd);
12     i2c_master_write_byte(cmd, i2c_slave_addr | WRITE_BIT, ACK_CHECK_EN);
13     i2c_master_write_byte(cmd, memory_location>>8, ACK_CHECK_EN);
14     i2c_master_write_byte(cmd, memory_location, ACK_CHECK_EN);
15     i2c_master_stop(cmd);
16     ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
17     i2c_cmd_link_delete(cmd);
18     if (ret != ESP_OK) {
19         ESP_LOGI("I2C_Test","memory location error");
20         return 0;
21     }
22     cmd = i2c_cmd_link_create();
23     i2c_master_start(cmd);
24     i2c_master_write_byte(cmd, i2c_slave_addr | READ_BIT, ACK_CHECK_EN);
25     i2c_master_read_byte(cmd, &data_rd, NACK_VAL);
26     i2c_master_stop(cmd);
27     ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
28     i2c_cmd_link_delete(cmd);
29     if (ret != ESP_OK){
30         ESP_LOGI("I2C_Test","data read error");
31         return 0;
32     }
33     return data_rd;
34 }
```


9 Write bytes into EEPROM

The following code is used to write a byte into the EEPROM

```

1  /*
2  * @Info:   Write bytes into EEPROM
3  * @Param:  Memory location to write into and data to be written into
4  * @Return: 0 if I2C communication fails
5  */
6
7  esp_err_t ee24lc256_write_byte(uint16_t mem_addr, uint8_t mem_data){
8      select_i2c_memory(eeprom_addr);
9      uint8_t write_data_buff[3]={mem_addr>>8,mem_addr,mem_data};
10     i2c_cmd_handle_t cmd = i2c_cmd_link_create();
11     i2c_master_start(cmd);
12     i2c_master_write_byte(cmd, (i2c_slave_addr) | WRITE_BIT, ACK_CHECK_EN);
13     i2c_master_write(cmd, write_data_buff, 3, ACK_CHECK_EN);
14     i2c_master_stop(cmd);
15     esp_err_t ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
16     i2c_cmd_link_delete(cmd);
17     return ret;
18 }

```

9.1 Write series of data into EEPROM

The following code is used to write an array of bytes into the EEPROM

```

1  /*
2  * @Info:   Write series of data into EEPROM
3  * @Param:  Memory address, memory buffer of data and memory length
4  * @Return: 0 if INA communication fails
5  */
6
7  esp_err_t ee24lc256_write_page(uint16_t mem_addr, uint8_t *mem_buff, uint8_t mem_len){
8      uint8_t mem_count=0;
9      select_i2c_memory(eeprom_addr);
10     i2c_cmd_handle_t cmd = i2c_cmd_link_create();
11     i2c_master_start(cmd);
12     i2c_master_write_byte(cmd, (i2c_slave_addr) | WRITE_BIT, ACK_CHECK_EN);
13     i2c_master_write_byte(cmd, mem_addr>>8, ACK_CHECK_EN);
14     i2c_master_write_byte(cmd, mem_addr, ACK_CHECK_EN);
15     i2c_master_write(cmd, mem_buff, mem_len, ACK_CHECK_EN);
16     i2c_master_stop(cmd);
17     esp_err_t ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
18     i2c_cmd_link_delete(cmd);
19     return ret;
20 }

```

9.2 Read a series of data from the EEPROM

The following code is used to read an array of bytes from the EEPROM

```

1  /*
2  * @Info:   Read a array of data from the EEPROM
3  * @Param:  Memory address to read from, buffer to store the data, data length to be
4  *          read
5  * @Return: 0 if INA communication fails
6  */
7
8  int8_t ee24lc256_read_buff(uint16_t mem_addr, uint8_t *mem_buff, uint8_t mem_len){
9      ESP_LOGI(TAG,"Read EEPROM Buffer");
10     select_i2c_memory(eeprom_addr);
11     uint8_t data_rd=0, mem_count=0;
12     int ret;
13     i2c_cmd_handle_t cmd = i2c_cmd_link_create();
14     i2c_master_start(cmd);
15     i2c_master_write_byte(cmd, i2c_slave_addr | WRITE_BIT, ACK_CHECK_EN);
16     i2c_master_write_byte(cmd, mem_addr>>8, ACK_CHECK_EN);
17     i2c_master_write_byte(cmd, mem_addr, ACK_CHECK_EN);
18     i2c_master_stop(cmd);
19     ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
20     i2c_cmd_link_delete(cmd);

```

```

20     if (ret != ESP_OK) {
21         ESP_LOGI("I2C_Test","memory location error");
22         return 0;
23     }
24     cmd = i2c_cmd_link_create();
25     i2c_master_start(cmd);
26     i2c_master_write_byte(cmd, i2c_slave_addr | READ_BIT, ACK_CHECK_EN);
27     for(mem_count = 0; mem_count < (mem_len - 1); mem_count++){
28         i2c_master_read_byte(cmd, &mem_buff[mem_count],ACK_VAL);
29     }
30
31     i2c_master_read_byte(cmd, &mem_buff[mem_count],NACK_VAL);
32     i2c_master_stop(cmd);
33     ret = i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
34     i2c_cmd_link_delete(cmd);
35     if(ret != ESP_OK){
36         ESP_LOGI("I2C_Test","data read error");
37         return 0;
38     }
39     return ret;
40 }

```

9.3 Write array of data into the EEPROM

The following data is used to write data more than 128 bytes into EEPROM by splitting it

```

1  /*
2  * @Info:   Write a array of data into the EEPROM
3  * @Param:  Memory address to write into, buffer of data to be written, data length to
4  *          be written
5  * @Return: 0 if INA communication fails
6  */
7  esp_err_t ee24lc256_write_buff(uint16_t mem_addr, uint8_t *mem_buff, uint8_t mem_len){
8      int i=0;
9      ESP_LOGI(TAG,"Reached EEPROM buffer write function");
10     esp_err_t status = ESP_FAIL;
11     uint16_t first_page_space, first_page_size, num_writes, write_size, pages,
12     byte_written;
13     first_page_space = MEM_PAGE_LEN - (mem_addr % MEM_PAGE_LEN);
14     if(first_page_space > mem_len){
15         first_page_size = mem_len;
16         num_writes = 1;
17     }
18     else{
19         first_page_size = first_page_space;
20         num_writes = ((mem_len - first_page_size) / MEM_PAGE_LEN) + 2;
21     }
22     byte_written = 0;
23     for(pages = 0; pages < num_writes; pages++){
24         if(pages == 0){
25             write_size = first_page_size;
26         }
27         else{
28             write_size = (mem_len - byte_written) % MEM_PAGE_LEN;
29         }
30         ESP_LOGI(TAG,"Writing to address location %d",(mem_addr + byte_written));
31         ESP_LOGI(TAG,"Number of bytes to be written %d",write_size);
32         vTaskDelay(100/portTICK_PERIOD_MS);
33         status = ee24lc256_write_page((mem_addr + byte_written), &mem_buff[byte_written],
34         write_size);
35         if(status==ESP_OK){
36             byte_written += write_size;
37         }
38         else{
39             ESP_LOGI(TAG,"Error occurred while writting");
40             pages--;
41         }
42     }
43     return status;
44 }

```

9.4 Write a 16bit data into the EEPROM

The following code is used to write 2bytes of data into the EEPROM

```

1  /*
2  * @Info:   Write 16bit data into the EEPROM
3  * @Param:  Memory address to write into, data to be written
4  * @Return: None
5  */
6  void ee24lc256_write_int16(uint16_t mem_addr, uint16_t int16_dat){
7      uint8_t *dat_ptr;
8      dat_ptr = (uint8_t *)&int16_dat;
9      ee24lc256_write_buff(mem_addr, dat_ptr, 2);
10 }

```

9.5 Read 16bit data from the EEPROM

The following code is used to read 2bytes of data from the EEPROM

```

1  /*
2  * @Info:   Read 16bit data from the EEPROM
3  * @Param:  Memory address to read from
4  * @Return: 16bit value
5  */
6  uint16_t ee24lc256_read_int16(uint16_t mem_addr){
7      uint16_t int16_dat;
8      uint8_t *dat_ptr;
9      dat_ptr = (uint8_t *)&int16_dat;
10     ee24lc256_read_buff(mem_addr, dat_ptr, 2);
11     return (int16_dat);
12 }

```

9.6 Write 32bit data into the EEPROM

The following code is used to write 4bytes of data into the EEPROM

```

1  /*
2  * @Info:   Write 32bit data into the EEPROM
3  * @Param:  Memory address to write into, data to be written
4  * @Return: None
5  */
6  void ee24lc256_write_int32(uint16_t mem_addr, uint32_t int32_dat){
7      uint8_t *dat_ptr;
8
9      dat_ptr = (uint8_t *)&int32_dat;
10     ee24lc256_write_buff(mem_addr, dat_ptr, 4);
11 }

```

9.7 Read 32bit data from EEPROM

The following code is used to read 4bytes of data from the EEPROM

```

1  /*
2  * @Info:   Read 32bit data from the EEPROM
3  * @Param:  Memory address to read from
4  * @Return: 32bit value
5  */
6  uint32_t ee24lc256_read_int32(uint16_t mem_addr){
7      uint32_t int32_dat;
8      uint8_t *dat_ptr;
9      dat_ptr = (uint8_t *)&int32_dat;
10     ee24lc256_read_buff(mem_addr, dat_ptr, 4);
11     return (int32_dat);
12 }

```

9.8 Write float data into the EEPROM

The following code is used to write floating value from the EEPROM

```

1  /*
2  * @Info:   Write float data into the EEPROM
3  * @Param:  Memory address to write into, data to be written
4  * @Return: None
5  */
6  void ee24lc256_write_float(uint16_t mem_addr, float float_dat){
7      uint8_t *dat_ptr;
8      dat_ptr = (uint8_t *)&float_dat;
9      ee24lc256_write_buff(mem_addr, dat_ptr, 4);
10 }

```

9.9 Read float data from EEPROM

The follownig code is used to read floating value from the EEPROM

```

1  /*
2  * @Info:   Read flaot data from the EEPROM
3  * @Param:  Memory address to read from
4  * @Return: flaot value
5  */
6  float ee24lc256_read_float(uint16_t mem_addr){
7      float float_dat;
8      uint8_t *dat_ptr;
9      dat_ptr = (uint8_t *)&float_dat;
10     ee24lc256_read_buff(mem_addr, dat_ptr, 4);
11     return (float_dat);
12 }

```

9.10 Check memory available

Check if EEPROM is interfaced with the device

```

1  /*
2  * @Info:   Check if memory is available
3  * @Param:  None
4  * @Return: 1 if memory is available; 0 if memory is not available
5  */
6  uint8_t check_memory(){
7      vTaskDelay(100/ portTICK_RATE_MS);
8      if(ee24lc256_busy() != ESP_OK){
9          return 0;
10     }
11     else{
12         return 1;
13     }
14 }

```

10 Data logging

10.1 Flush log data

the following code is used to clear the ring buffer

```

1  /*
2  * @Info:   Flush the data in the ring buffer
3  * @Param:  None
4  * @Return: None
5  */
6  void turtle_log_flush(){
7      log_struct.log_head = LOG_START_ADDR;
8      log_struct.log_tail = log_struct.log_head;
9      log_struct.log_available = 0;
10 }
```

10.2 Restore data tail

The following code is used to reload the ring buffer values

```

1  /*
2  * @Info:   Restore the tail of EEPROM after reset
3  * @Param:  None
4  * @Return: None
5  */
6  void
7  turtle_log_idx_restore(){
8      if (i2c_eeprom_read_int32(VALID_IDX_ADDR) == VALID_IDX_VALUE) {
9          i2c_eeprom_read_buff(EE_LOG_IDX_START_ADDR, (uint8_t *) & log_struct, sizeof (
10         log_struct));
11         flush_idx = 1;
12         if (((log_struct.log_head % SUMMARY_LEN) != 0) || ((log_struct.log_tail %
13         SUMMARY_LEN) != 0)) {
14             log_struct.log_head = LOG_START_ADDR;
15             log_struct.log_tail = log_struct.log_head;
16             log_struct.log_available = 0;
17             flush_idx = 0;
18         }
19     }
20     else {
21         log_struct.log_head = LOG_START_ADDR;
22         log_struct.log_tail = log_struct.log_head;
23         log_struct.log_available = 0;
24         flush_idx = 0;
25     }
26 }
```

10.3 Save data head and tail

The following code is used to store the ring buffer head and tail values when powering off the device

```

1  /*
2  * @Info:   Save log head and tail before reset
3  * @Param:  None
4  * @Return: None
5  */
6  void turtle_log_idx_save(){
7      i2c_eeprom_write_buff(EE_LOG_IDX_START_ADDR, (uint8_t *) & log_struct, sizeof (
8      log_struct));
9      vTaskDelay(5);
10     i2c_eeprom_write_int32(VALID_IDX_ADDR, VALID_IDX_VALUE);
11 }
```

10.4 Write data into EEPROM

The following code is used to write sensor data into the EEPROM in ring buffer fashion

```

1  /*
2  * @Info:    Write data into eeprom
3  * @Param:   Data buffer
4  * @Return:  None
5  */
6  void turtle_log_put(uint8_t *log_in_buff){
7      uint16_t _log_head_offset, _log_tail_offset;
8      _log_head_offset = (log_struct.log_head + SUMMARY_LEN) % LOG_END_ADDR;
9      if(_log_head_offset == log_struct.log_tail){
10         _log_tail_offset = (log_struct.log_tail + SUMMARY_LEN) % LOG_END_ADDR;
11         log_struct.log_tail = _log_tail_offset;
12     }
13     else{
14         log_struct.log_available++;
15     }
16     ESP_LOGI(TAG,"Writing to address %d",log_struct.log_head);
17     i2c_eeprom_write_buff(log_struct.log_head, log_in_buff, SUMMARY_LEN);
18     log_struct.log_head = _log_head_offset;
19 }

```

10.5 Retrieve data from EEPROM

The following code is used to retrieve the sensor data from the EEPROM

```

1  /*
2  * @Info:    Retrieve data from EEPROM
3  * @Param:   Data and packet count
4  * @Return:  None
5  */
6  void turtle_log_get(uint8_t *log_out_buff,uint8_t pos){
7      uint16_t _log_tail_offset, record_count, record_idx = 0;
8      _log_tail_offset = log_struct.log_tail+(pos*SUMMARY_LEN);
9      for(record_count = 0; record_count < RECORDS_PER_PACKET; record_count++){
10         ESP_LOGI(TAG,"Reading address %d",_log_tail_offset);
11         i2c_eeprom_read_buff(_log_tail_offset, &log_out_buff[record_idx], SUMMARY_LEN);
12         _log_tail_offset = (_log_tail_offset + SUMMARY_LEN) % LOG_END_ADDR;
13         record_idx += SUMMARY_LEN;
14     }
15 }

```

10.6 Updating tail of data

The following code is used to update the tail after reading the data from the device

```

1  /*
2  * @Info:    Update the tail of data which has been sent to server
3  * @Param:   None
4  * @Return:  None
5  */
6  void turtle_log_update_tail(){
7      extern uint8_t packet_length;
8      uint16_t _log_tail_offset;
9      if(log_struct.log_head != log_struct.log_tail){
10         _log_tail_offset = (log_struct.log_tail + packet_length*(SUMMARY_LEN *
11             RECORDS_PER_PACKET)) % LOG_END_ADDR;
12         log_struct.log_tail = _log_tail_offset;
13         if(log_struct.log_available >= packet_length*RECORDS_PER_PACKET){
14             log_struct.log_available -= packet_length*RECORDS_PER_PACKET;
15         }
16     }
17 }

```

10.7 Check availability of data

The following code is used to check if the data is available in the EEPROM

```

1  /*
2  * @Info:    Returns the number of data packets available in the memory
3  * @Param:   None
4  * @Return:  None

```

```
5  */
6  uint16_t turtle_log_available(){
7      return (log_struct.log_available);
8  }
```

10.8 Log data to EEPROM

The following code is used to log data into the EEPROM

```
1  /*
2  * @Info:    Log the algorithm data to the EEPROM
3  * @Param:   None
4  * @Return:  None
5  */
6  void log_to_eeprom(){
7      check_time();
8      ESP_LOGI(TAG, "Inside log to eeprom function\n");
9      algo_param.date_time[0] = date_time->tm_mday;    //Set day
10     algo_param.date_time[1] = 1+(date_time->tm_mon);  //Set month
11     algo_param.date_time[2] = date_time->tm_year;    //set year
12     algo_param.date_time[3] = date_time->tm_hour;    //Set Hour
13     algo_param.date_time[4] = date_time->tm_min;     //Set minute
14     algo_param.date_time[5] = date_time->tm_sec;
15     ESP_LOGI(TAG, "Updated time\n");
16     while(i2cFlag==1);
17     i2cFlag=1;
18     turtle_log_put((uint8_t *)&algo_param);
19     i2cFlag=0;
20 }
```

11 esp32s2_wifi.c

11.1 WiFi event handler

The following code is the event handler for the WiFi

```

1  /*
2  * @Info: Handler for WiFi;
3  *       Depending on the action by WiFi the following switch case statement will be
4  *       executed
5  * @Param: system_event handler and void pointer
6  * @Return: None
7  */
8  static esp_err_t event_handler(void *ctx, system_event_t *event){
9      switch (event->event_id){
10         case SYSTEM_EVENT_STA_START:
11             esp_wifi_connect(); /* Connect to WiFi AP and PSW when wifi is started */
12             ESP_LOGI(TAG, "connecting...\n");
13             break;
14         case SYSTEM_EVENT_STA_CONNECTED:
15             /* When ESP connected to AP and PSW */
16             ESP_LOGI(TAG, "connected\n");
17             break;
18         case SYSTEM_EVENT_STA_GOT_IP:
19             /* When ESP is assigned a IP */
20             ESP_LOGI(TAG, "got ip\n");
21             extern uint8_t connection_stat_check, connection_stat;
22             /* The following code is used to test connectivity with server during mobile app
23             interaction */
24             if(connection_stat_check){
25                 connection_stat=1;
26                 xSemaphoreGive(logdataSemaphore);
27             }
28             /* Initialize RTC when connected to a AP */
29             initRTC();
30             wifi_connection_status = true;
31             /* This is to start the socket client when the device is set in calibration mode
32             */
33             xSemaphoreGive(connectionSemaphore);
34             break;
35         case SYSTEM_EVENT_STA_DISCONNECTED:
36             /* Retry connecting to hotspot in during test connectivity */
37             wifi_connection_status = false;
38             if(connection_stat_check){
39                 if(++test_connect_cnt>3){
40                     connection_stat_check=0;
41                     connection_stat=0;
42                     connection_status_obtained=1;
43                     blink(led_red_color, FULL_ON);
44                     vTaskDelay(3000/portTICK_PERIOD_MS);
45                     extern uint8_t stop_algorithm;
46                     stop_algorithm=0;
47                 }
48             }
49             else{
50                 esp_wifi_stop();
51                 connectSTA((char *)ap_ssid_buff, (char *)ap_psw_buff);
52             }
53             ESP_LOGI(TAG, "disconnected\n");
54             break;
55         case SYSTEM_EVENT_AP_STA_CONNECTED:
56             ESP_LOGI(TAG, "Station connected to ESP32 AP");
57             break;
58         case SYSTEM_EVENT_AP_STA_IP_ASSIGNED:
59             /* To start socket server when device set in configuration mode */
60             ESP_LOGI(TAG, "IP assigned to connected station");
61             xSemaphoreGive(serverSemaphore);
62             break;
63         default:
64             break;
65     }
66     return ESP_OK;
67 }

```


11.2 Connect to station point

The following code is used to connect to an AP

```

1  /*
2  * @Info:   Connecting to WiFi AP
3  * @Param:  SSID and PASSWORD to connect
4  * @Return: None
5  */
6  void connectSTA(char *ssid, char *pass){
7      ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
8      wifi_config_t wifi_config;
9      memset(&wifi_config, 0, sizeof(wifi_config));
10     strcpy((char *)wifi_config.sta.ssid, ssid);
11     strcpy((char *)wifi_config.sta.password, pass);
12     ESP_LOGI(TAG, "AP SSID is %s", (char *)wifi_config.sta.ssid);
13     ESP_LOGI(TAG, "AP PSW is %s", (char *)wifi_config.sta.password);
14     esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config);
15     ESP_ERROR_CHECK(esp_wifi_start());
16 }

```

11.3 Create access point

The following code is used to create an AP

```

1  /*
2  * @Info:   Create an AP with specified SSID and PASSWORD
3  * @Param:  None
4  * @Return: None
5  */
6  void connectAP(){
7      start_dhcp_server();
8      ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
9      wifi_config_t wifi_config = {
10         .ap = {
11 #if ICUBE
12             .ssid = "iCUBE-L0800",
13             .password = "icube1234",
14 #else
15             .ssid = "iCON-L0800",
16             .password = "icon1234",
17 #endif
18             .max_connection = 1,
19             .authmode = WIFI_AUTH_WPA_WPA2_PSK
20         }
21     };
22     esp_wifi_set_config(ESP_IF_WIFI_AP, &wifi_config);
23     ESP_ERROR_CHECK(esp_wifi_start());
24 }

```

11.4 Initialize WiFi module

The following code is used to initialize WiFi module

```

1  /*
2  * @Info:   Initialize the WiFi module of ESP
3  * @Param:  None
4  * @Return: None
5  */
6  void esp32s2_WiFi_Init(void){
7      ESP_ERROR_CHECK(nvs_flash_init());
8      tcpip_adapter_init();
9      ESP_ERROR_CHECK(esp_event_loop_init(event_handler, NULL));
10     wifi_init_config_t wifi_init_config = WIFI_INIT_CONFIG_DEFAULT();
11     ESP_ERROR_CHECK(esp_wifi_init(&wifi_init_config));
12 }

```

11.5 Print log

The following code is used to print the time from RTC in HMS fashion

```

1  /*
2  * @Info:    Print log time
3  * @Param:   Time and message
4  * @Return:  None
5  */
6  void print_time(long time, const char *message){
7      struct tm *timeinfo = localtime(&time);
8      printf("Time is  %dH:%dM:%dS \n", timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec)
9      ;
10 }

```

11.6 Callback to sync time

The following code is called once time sync is successful

```

1  /*
2  * @Info:    Callback for time_sync_notification
3  * @Param:   time structure
4  * @Return:  None
5  */
6  void SyncTime_callback(struct timeval *tv){
7      rtc_sync = 1;
8      ESP_LOGI(TAG,"Time synced\n");
9      // getRTCtime();
10 }

```

11.7 Get RTC time

The following code is used to get the RTC time

```

1  /*
2  * @Info:    Get RTC time
3  * @Param:   None
4  * @Return:  None
5  */
6  void getRTCtime(){
7      time(&current_time);
8      ESP_LOGI("Time","Time is %ld",current_time);
9      print_time(current_time,"time");
10 }

```

11.8 Initialize RTC module

The following code is used to initialize the RTC

```

1  /*
2  * @Info:    Initialize the RTC module
3  * @Param:   None
4  * @Return:  None
5  */
6  void initRTC(void){
7      ESP_LOGI(TAG,"Initializing RTC");
8      sntp_set_sync_mode(SNTP_SYNC_MODE_IMMED);
9      sntp_setservername(0, "pool.ntp.org");
10     sntp_init();
11     sntp_set_time_sync_notification_cb(SyncTime_callback);
12     setenv("TZ", "UTC-5:30", 1);
13     tzset();
14 }

```

11.9 Stop DHCP server

The following code is used to stop the DHCP server used for socket communication

```

1  /*
2  * @Info:    Stop DHCP server
3  * @Param:   None
4  * @Return:  None
5  */

```

```
6 void stop_dhcp_server(){
7     esp_err_t status;
8     status = tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP);
9     ESP_LOGI(TAG, "Error value is %d", status);
10 }
```

11.10 Start DHCP server

The following code is used to start the DHCP server used for socket communication

```
1 /*
2  * @Info: Establish a DHCP server for the mobile app to connect
3  */
4
5
6 void start_dhcp_server(){
7     // stop DHCP server
8     tcpip_adapter_dhcps_stop(TCPIP_ADAPTER_IF_AP);
9     // assign a static IP to the network interface
10    tcpip_adapter_ip_info_t info;
11    memset(&info, 0, sizeof(info));
12    IP4_ADDR(&info.ip, 192, 168, 4, 1);
13    IP4_ADDR(&info.gw, 192, 168, 4, 1); //ESP acts as router, so gw addr will be its
    own addr
14    IP4_ADDR(&info.netmask, 255, 255, 255, 0);
15    ESP_ERROR_CHECK(tcpip_adapter_set_ip_info(TCPIP_ADAPTER_IF_AP, &info));
16    // start the DHCP server
17    ESP_ERROR_CHECK(tcpip_adapter_dhcps_start(TCPIP_ADAPTER_IF_AP));
18    printf("DHCP server started \n");
19 }
```

12 esp32s2_socket.c

12.1 TCP client task

The following task creates a socket client to communicate with the calibration tool

```

1  /*
2  * @Info: TCP client task; Socket allocation, binding, data receive;
3  * @Param: void task handler
4  * @Return: None
5  */
6  void tcp_client(void *pvParam){
7      config_calib_mode = CONFIG_CALIB_WIFI_CLIENT;
8      ESP_LOGI(TAG,"tcp_client task started");
9      struct sockaddr_in tcpServerAddr;
10     tcpServerAddr.sin_addr.s_addr = inet_addr(TCPServerIP);
11     tcpServerAddr.sin_family = AF_INET;
12     tcpServerAddr.sin_port = htons( PORT );
13     int r;
14     sock = socket(AF_INET, SOCK_STREAM, 0);
15     while(1){
16         if(xSemaphoreTake(connectionSemaphore, 10000 / portTICK_RATE_MS))
17         {
18             while(1){
19                 switch(socket_client_status){
20                     case idle:
21                         sock = socket(AF_INET, SOCK_STREAM, 0);
22                         if(sock < 0) {
23                             ESP_LOGE(TAG, "... Failed to allocate socket...");
24                             vTaskDelay(1000 / portTICK_PERIOD_MS);
25                         }
26                         else{
27                             ESP_LOGE(TAG, "... Allocated socket...");
28                             socket_client_status = allocated_socket;
29                         }
30                         break;
31                     case allocated_socket:
32                         if(connect(sock, (struct sockaddr *)&tcpServerAddr, sizeof(
tcpServerAddr)) != 0){
33                             socket_client_status = idle;
34                             ESP_LOGE(TAG, "... socket connect failed errno=%d ...",
errno);
35                             close(sock);
36                             vTaskDelay(5000 / portTICK_PERIOD_MS);
37                         }
38                         else{
39                             ESP_LOGI(TAG,"Connected to server");
40
41                             for(int i=0;i<30;i++){
42                                 stream_tx_buff[i]=i;
43                             }
44                             int err = send(sock, stream_tx_buff, 30, 0);
45                             if (err < 0) {
46                                 ESP_LOGE(TAG, "Error occurred during sending: errno %d",
errno);
47                             }
48                             socket_client_status = connected_socket;
49                             blink(led_magenta_color,FULL_ON);
50                         }
51                         break;
52                     case connected_socket:
53                         bzero(recv_buf, sizeof(recv_buf));
54                         r = read(sock, recv_buf, sizeof(recv_buf)-1);
55                         if(r<0){
56                             ESP_LOGI(TAG,"Socket closed");
57                             socket_client_status = allocated_socket;
58                         }
59                         else{
60                             ESP_LOGI(TAG,"Checking data");
61                             if(read_calib_config_req(&uart_cmd, stream_tx_buff, &
total_req_len)){
62                                 socket_client_status = data_received;
63                                 ESP_LOGI(TAG,"Data received");

```

```

64         extern uint8_t configuration_timeout_cnt;
65         configuration_timeout_cnt=0;
66     }
67 }
68 break;
69 case data_received:
70     process_labview_cmd();
71     break;
72 }
73 }
74 }
75 else{
76     extern uint8_t server_connect_retry_cnt;
77     extern uint8_t server_connect_retry;
78     if(server_connect_retry_cnt>15){
79         server_connect_retry_cnt=0;
80         server_connect_retry+=1;
81         esp_wifi_stop();
82         ESP_LOGI(TAG, "Retrying connecting to E24by7");
83         connectSTA("E24by7", "Energy24by7");
84     }
85     if(server_connect_retry>=3){
86         server_connect_retry=0;
87         server_connect_retry_cnt=0;
88         extern uint8_t stop_algorithm;
89         stop_algorithm=0;
90         ESP_LOGI(TAG, "...tcp_client task deleted...");
91         esp_wifi_stop();
92         vTaskDelete(NULL);
93     }
94 }
95 }
96 }
97 }

```

12.2 TCP server task

The following code creates a socket server to communicate with the mobile app and configuration tool

```

1  /*
2  * @Info: TCP server task; Socket allocation, binding, data receive;
3  * @Param: void task handler
4  * @Return: None
5  */
6  void tcp_server(void *pvParam){
7      config_calib_mode = CONFIG_CALIB_WIFI_SERVER;
8      ESP_LOGI(TAG, "tcp_server task started \n");
9      struct sockaddr_in tcpServerAddr;
10     tcpServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);
11     tcpServerAddr.sin_family = AF_INET;
12     tcpServerAddr.sin_port = htons(5000);
13     // int s, r;
14     static struct sockaddr_in remote_addr;
15     static unsigned int socklen;
16     socklen = sizeof(remote_addr);
17     socket_client_status = idle;
18     ESP_LOGI(TAG, "Reached here");
19
20     s = socket(AF_INET, SOCK_STREAM, 0);
21     if(s < 0) {
22         ESP_LOGE(TAG, "socket: %d %s", s, strerror(errno));
23     }
24
25     socket_client_status=allocated_socket;
26     blink(led_magenta_color, FULL_ON);
27     while(1){
28         if(xSemaphoreTake(serverSemaphore, 10000 / portTICK_RATE_MS)){
29             ESP_LOGI(TAG, "Entering TCP server function");
30             while(1){
31                 ESP_LOGI(TAG, "point 1");
32                 switch(socket_client_status){
33                     case idle:

```

```

34         s = socket(AF_INET, SOCK_STREAM, 0);
35         if(s < 0) {
36             ESP_LOGE(TAG, "... Failed to allocate socket errno=%d",errno
);
37             vTaskDelay(1000 / portTICK_PERIOD_MS);
38         }
39         else{
40             ESP_LOGE(TAG, "... Allocated socket...");
41             socket_client_status = allocated_socket;
42             // socket_client_status = bind_socket;
43         }
44         break;
45         case allocated_socket:
46             if(bind(s, (struct sockaddr *)&tcpServerAddr, sizeof(
tcpServerAddr)) != 0){
47                 socket_client_status = idle;
48                 ESP_LOGE(TAG, "... socket bind failed errno=%d ...", errno);
49                 close(s);
50                 vTaskDelay(4000 / portTICK_PERIOD_MS);
51                 s = socket(AF_INET, SOCK_STREAM, 0);
52             }
53             else{
54                 ESP_LOGI(TAG,"Socket bind successfull...");
55                 socket_client_status = bind_socket;
56             }
57             break;
58         case bind_socket:
59             if(listen(s, 1) != 0) {
60                 ESP_LOGE(TAG, "... socket listen failed errno=%d ", errno);
61                 close(s);
62                 vTaskDelay(4000 / portTICK_PERIOD_MS);
63                 s = socket(AF_INET, SOCK_STREAM, 0);
64             }
65             else{
66                 ESP_LOGI(TAG,"Socket listening");
67                 socket_client_status = listen_socket;
68             }
69             break;
70         case listen_socket:
71             cs=accept(s,(struct sockaddr *)&remote_addr, &socklen);
72             ESP_LOGI(TAG,"New connection request,Request data:");
73             fcntl(cs,F_SETFL,O_NONBLOCK);
74             socket_client_status = connected_socket;
75             break;
76         case connected_socket:
77             bzero(recv_buf, sizeof(recv_buf));
78             r = recv(cs, recv_buf, sizeof(recv_buf)-1,0);
79             if(r<0){
80                 // ESP_LOGI(TAG,"Socket closed");
81                 // socket_client_status = bind_socket;
82             }
83             else{
84                 ESP_LOGI(TAG,"Checking data");
85                 if(read_calib_config_req(&uart_cmd, stream_tx_buff, &
total_req_len)){
86                     socket_client_status = data_received;
87                     ESP_LOGI(TAG,"Data received");
88                     extern uint8_t configuration_timeout_cnt;
89                     configuration_timeout_cnt=0;
90                 }
91             }
92             break;
93         case data_received:
94             process_labview_cmd();
95             break;
96         case do_nothing:
97             break;
98     }
99     vTaskDelay(100/portTICK_PERIOD_MS);
100 }
101 }
102 }
103 }

```

13 Parsing incoming data

The following code is used to parse the data received from the tool

```

1  /*
2  * @Info:   Parse incoming data from the tool
3  * @Param:  None
4  * @Return: None
5  */
6
7  uint8_t read_calib_config_req(uint8_t *req_cmd, uint8_t *req_payload, uint16_t *
   req_pkt_len){
8      uint8_t _sync_word, buff_idx;
9      uint16_t _full_pkt_len, _check_sum, _calc_check_sum;
10     /* Extract the sync word from the received buffer */
11     _sync_word = recv_buf[0];
12     if(_sync_word == CONFIG_CALIB_SYNC){
13         for(buff_idx = 0; buff_idx < REQ_HEADER_LEN; buff_idx++){
14             /* Since sync word is already extracted copy the bytes starting from command
               into the rx_buffer */
15             stream_rx_buff[buff_idx] = recv_buf[buff_idx+1];
16         }
17         *req_cmd = stream_rx_buff[REQ_ID_IDX];
18         *req_pkt_len = ((stream_rx_buff[REQ_DAT_LEN_IDX + 1] << 8) | (stream_rx_buff[
19 REQ_DAT_LEN_IDX]));
20         memcpy((uint8_t *)&config_calib_ses_res, &stream_rx_buff[REQ_SES_ID_IDX],
21 SES_ID_LEN);
22         if(*req_pkt_len > MAX_PAYLOAD_LEN){
23             ESP_LOGI(TAG, "Payload greater than maximum length");
24             return 0;
25         }
26         _full_pkt_len = *req_pkt_len + REQ_HEADER_LEN + REQ_CRC_LEN;
27
28         for(buff_idx = REQ_PAYLOAD_IDX; buff_idx < _full_pkt_len; buff_idx++){
29             stream_rx_buff[buff_idx] = recv_buf[buff_idx+1];
30         }
31         _full_pkt_len -= REQ_CRC_LEN;
32         _calc_check_sum = crc_16(stream_rx_buff, (_full_pkt_len));
33         _check_sum = ((stream_rx_buff[_full_pkt_len + 1] << 8) | stream_rx_buff[
34 _full_pkt_len]);
35
36         if(*req_pkt_len > 0){
37             memcpy(req_payload, &stream_rx_buff[REQ_PAYLOAD_IDX], *req_pkt_len);
38         }
39         ESP_LOGI(TAG, "Read calib/config returned success");
40         return 1;
41     }
42     ESP_LOGI(TAG, "Sync word didn't match");
43     return 0;
44 }

```

13.1 Response to utility

The following code creates a packet structure to send data back to the tools

```

1  /*
2  * @Info:   Send response to utility
3  * @Param:  None
4  * @Return: None
5  */
6
7  void res_to_utility(uint8_t _res_type, uint8_t *_res_buff, uint16_t _res_len){
8      uint16_t _full_pkt_len, check_sum;
9      if(config_calib_mode == CONFIG_CALIB_WIFI_CLIENT){
10         ESP_LOGI(TAG, "Response to utility as a client");
11         stream_tx_buff[RES_SYNC_IDX] = CONFIG_CALIB_SYNC;
12         stream_tx_buff[REQ_SES_ID_IDX] = _res_type;
13         stream_tx_buff[RES_DAT_LEN_IDX] = (_res_len & 0xFF);
14         stream_tx_buff[RES_DAT_LEN_IDX + 1] = ((_res_len >> 8) & 0xFF);
15         memcpy((uint8_t *)&stream_tx_buff[RES_SSID_IDX], &config_calib_ses_res,
16 SES_ID_LEN);

```

```

16     if(_res_len > 0){
17         memcpy(&stream_tx_buff[RES_PAYLOAD_IDX_CLIENT], _res_buff, _res_len);
18     }
19     _full_pkt_len = RES_HEADER_LEN_CLIENT + _res_len;
20     check_sum = crc_16(stream_tx_buff, _full_pkt_len);
21     stream_tx_buff[_full_pkt_len + 1] = (check_sum >> 8);
22     stream_tx_buff[_full_pkt_len] = (check_sum & 0xFF);
23     _full_pkt_len += RES_CRC_LEN;
24     int err = send(sock, stream_tx_buff, _full_pkt_len, 0);
25     if (err < 0){
26         ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
27     }
28 }
29 else{
30     ESP_LOGI(TAG, "Response to utility as a server");
31     stream_tx_buff[RES_SYNC_IDX] = CONFIG_CALIB_SYNC;
32     stream_tx_buff[REQ_SES_ID_IDX] = _res_type;
33     stream_tx_buff[RES_DAT_LEN_IDX] = (_res_len & 0xFF);
34     stream_tx_buff[RES_DAT_LEN_IDX + 1] = ((_res_len >> 8) & 0xFF);
35     if(_res_len > 0){
36         memcpy(&stream_tx_buff[RES_PAYLOAD_IDX], _res_buff, _res_len);
37     }
38     _full_pkt_len = RES_HEADER_LEN + _res_len;
39     check_sum = crc_16(stream_tx_buff, _full_pkt_len);
40     stream_tx_buff[_full_pkt_len + 1] = (check_sum >> 8);
41     stream_tx_buff[_full_pkt_len] = (check_sum & 0xFF);
42     _full_pkt_len += RES_CRC_LEN;
43     int err = write(cs, stream_tx_buff, _full_pkt_len);
44     if (err < 0){
45         ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
46     }
47 }
48 }

```

13.2 Response to utility for authentication

The following code is used to send response to the app during authentication

```

1  /*
2  * @Info:    Send response to utility for authentication
3  * @Param:  None
4  * @Return: None
5  */
6  void
7  send_con_res_config_calib(){
8      uint8_t dev_type = 1;
9      if((i2c_eeprom_read_int32(EF_CALIB_DONE_ADDR) == CALIB_CONFIG_DONE_VALUE)){
10         stream_rx_buff[CON_RES_FLAG_IDX] = 1;
11     }
12     else{
13         stream_rx_buff[CON_RES_FLAG_IDX] = 0;
14     }
15     i2c_eeprom_read_buff(EF_DEV_SER_NO_ADDR, &stream_rx_buff[CON_RES_ID_IDX], (
16     DEV_SER_NO_LEN + DEVICE_TYPE_LEN));
17     memcpy(&stream_rx_buff[CON_RES_ID_IDX+DEV_SER_NO_LEN], &dev_type, DEVICE_TYPE_LEN);
18     memcpy(&stream_rx_buff[CON_RES_HW_VER_IDX], (uint8_t*)&turt_fm_hw_ver, HW_FM_VER_LEN
19     );
20     stream_rx_buff[CON_RES_PROT_VER_IDX] = CALIB_CONFIG_MAJ_VER;
21     stream_rx_buff[CON_RES_PROT_VER_IDX + 1] = CALIB_CONFIG_MIN_VER;
22     memcpy(&stream_rx_buff[CON_RES_SID_IDX], (uint8_t*)&config_calib_ses_id, SES_ID_LEN
23     );
24     res_to_utility(REQ_CON, stream_rx_buff, CON_RES_PKT_LEN);
25 }

```

13.3 Read ADC value

The following code is used to read sensor values in digital form

```

1  /*
2  * @info:    Read digital value from the device
3  * @Param:  None

```



```

4  * @Return: None
5  */
6  void read_send_adc_value(){
7      read_adc_all(stream_rx_buff);
8      res_to_utility(CMD_GET_ADC, stream_rx_buff, 20);
9  }

```

13.4 Read EEPROM value

The following code is used to read data from the EEPROM

```

1  /*
2  * @Info:   Read values from EEPROM
3  * @Param:  None
4  * @Return: None
5  */
6  void read_send_eeprom_val(){
7      uint16_t _ee_start_address;
8      uint8_t _ee_dat_len;
9      memcpy((uint8_t *)&_ee_start_address, &stream_tx_buff[0], 2);
10     _ee_dat_len = stream_tx_buff[2];
11     if((_ee_dat_len > 0) && (_ee_dat_len <= MAX_PAYLOAD_LEN)){
12         memcpy(&stream_rx_buff[0], (uint8_t *)&_ee_start_address, 2);
13         stream_rx_buff[2] = _ee_dat_len;
14         i2c_eeprom_read_buff(_ee_start_address, &stream_rx_buff[3], _ee_dat_len);
15         res_to_utility(CMD_READ_EEPROM, stream_rx_buff, (_ee_dat_len + 3));
16     }
17     if(_ee_start_address >= 0xFF21){
18         clear_system_flag(SF_USEABLE_SOC_RC);
19     }
20 }

```

13.5 Load WiFi parameters

The following code is used to update the ssid and password variables with the data from EEPROM

```

1  /*
2  * @Info:   Load WiFi parameters
3  * @Param:  SSID and PSW
4  * @Return: None
5  */
6  void load_wifi_param(){
7      #if 0
8          #warning "Wifi Hard coded"
9          // strcpy(ip_addr_buff, "192.168.2.24");
10         strcpy(ip_addr_buff, "mystic-swan.ddns.net");
11         server_port_address = 5000;
12         strcpy(ap_ssid_buff, "msNet");
13         strcpy(ap_psw_buff, "mystic123");
14     #else
15         i2c_eeprom_read_buff(EA_SERV_IP_ADDR, ip_addr_buff, SERV_IP_LEN);
16         server_port_address = i2c_eeprom_read_int16(EA_SERV_PORT_ADDR);
17         i2c_eeprom_read_buff(EA_AP_SSID_ADDR, ap_ssid_buff, AP_SSID_LEN);
18         i2c_eeprom_read_buff(EA_AP_PSW_ADDR, ap_psw_buff, AP_PSW_LEN);
19         ip_addr_buff[SERV_IP_LEN - 1] = '\0';
20         ap_ssid_buff[AP_SSID_LEN - 1] = '\0';
21         ap_psw_buff[AP_PSW_LEN - 1] = '\0';
22         access_point = primary;
23     #endif
24 }

```

13.6 Calculate EEPROM CRC

The following code is used to calculate CRC of the data to be written into the EEPROM

```

1  /*
2  * @Info:   Calcualte CRC for data to be stored in EEPROM
3  * @Param:  Memory starting address, Total length, memory location to store CRC
4  * @Return: CRC16
5  */

```

```

6 uint16_t calc_update_eeprom_crc(uint16_t _ee_start_address, uint8_t _ee_len, uint16_t
  _crc_addr, uint8_t is_update){
7   uint16_t _ee_crc;
8   vTaskDelay(100/portTICK_RATE_MS);
9   i2c_eeprom_read_buff(_ee_start_address, stream_rx_buff, _ee_len);
10  _ee_crc = crc_16(stream_rx_buff, _ee_len);
11  if(is_update)
12  {
13    vTaskDelay(100/portTICK_RATE_MS);
14    i2c_eeprom_write_int16(_crc_addr, _ee_crc);
15  }
16  return (_ee_crc);
17 }

```

13.7 Write DATA to eeprom

The EEPROM memory locations from 0xF000 to 0xFFFF are the location ranges accessible by the socket functions. The following code is used to write data into these areas.

```

1  /*
2  * @Info:   Write data into EEPROM
3  * @param:  None
4  * @Return: None
5  */
6  void write_const_to_eeprom(){
7    uint16_t _ee_start_address, _ee_end_address;
8    uint8_t _ee_dat_len;
9    memcpy((uint8_t *)&_ee_start_address, &stream_tx_buff[0], 2);
10   _ee_dat_len = stream_tx_buff[2];
11   _ee_end_address = (_ee_start_address + _ee_dat_len) - 1 ;
12
13   if((_ee_dat_len > 0) && (_ee_dat_len <= MAX_PAYLOAD_LEN) && (_ee_start_address >=
     EE_CONFIG_DAT_START_ADDR && _ee_end_address <= EE_SYS_THR_END_ADDR)){
14     i2c_eeprom_write_buff(_ee_start_address, &stream_tx_buff[3], _ee_dat_len);
15     ESP_LOGI(TAG, "%d written into EEPROM", _ee_dat_len);
16     vTaskDelay(20/portTICK_RATE_MS);
17     memcpy(&stream_rx_buff[0], (uint8_t *)&_ee_start_address, 2);
18     stream_rx_buff[2] = _ee_dat_len;
19     i2c_eeprom_read_buff(_ee_start_address, &stream_rx_buff[3], _ee_dat_len);
20     for(int i=0; i<_ee_dat_len; i++){
21       ESP_LOGI(TAG, "Element %d in data is %d", i, stream_rx_buff[i]);
22     }
23     ESP_LOGI(TAG, "Preparing response to utility");
24     res_to_utility(CMD_WRITE_EEPROM, stream_rx_buff, (_ee_dat_len + 3));
25     // beep(1, SHORT_BEEP);
26   }
27
28   vTaskDelay(100/portTICK_RATE_MS);
29   if(_ee_start_address >= EE_CONFIG_DAT_START_ADDR && _ee_end_address <
     EE_DC_OVER_I_THR_ADDR){
30     i2c_eeprom_write_int32(EE_CONFIG_DONE_ADDR, CALIB_CONFIG_DONE_VALUE);
31     ESP_LOGI(TAG, "Configuration done address");
32     vTaskDelay(100/portTICK_RATE_MS);
33   }
34
35   if(_ee_start_address >= EE_DC_OVER_I_THR_ADDR && _ee_end_address < EE_CONFIG_DONE_ADDR
     ){
36     i2c_eeprom_write_int32(EE_CONFIG_DONE_ADDR, CALIB_CONFIG_DONE_VALUE);
37     vTaskDelay(100/portTICK_RATE_MS);
38     clear_error(ERR_NOT_CONFIG);
39     ESP_LOGI(TAG, "Clear configuration error");
40   }
41
42   if(_ee_start_address >= EE_CONFIG_DAT_START_ADDR && _ee_end_address <
     EE_CONFIG_DONE_ADDR){
43     calc_update_eeprom_crc(EE_CONFIG_DAT_START_ADDR, CONFIG_PARAM_LEN,
     EE_CONFIG_CRC_ADDR, 1);
44     ESP_LOGI(TAG, "Calculating CRC");
45   }
46
47   if(_ee_start_address >= EE_CALIB_DAT_START_ADDR && _ee_end_address <
     EE_CALIB_DONE_ADDR){

```

```

48     i2c_eeprom_write_int32(EE_CALIB_DONE_ADDR, CALIB_CONFIG_DONE_VALUE);
49     calc_update_eeprom_crc(EE_CALIB_DAT_START_ADDR, CALIB_CONST_LEN, EE_CALIB_CRC_ADDR,
50     1);
51     clear_error(ERR_NOT_CALIB);
52     ESP_LOGI(TAG, "Calibration done address");
53 }
54
55     vTaskDelay(100/portTICK_RATE_MS);
56     load_sensor_constants();
57     load_wifi_param();
58 }

```

13.8 Send real values to utility

The following code is used to send real values to the app

```

1  /*
2  * @Info:    Send analog values to utility
3  * @Param:   None
4  * @Return:  None
5  */
6  void send_real_values(){
7      extern uint8_t mains;
8      extern float inverter_vol, temperature1, temperature2, vol_ref;
9      get_sensor_values(FLUSH_ADC_VAL, 0);
10     memcpy(&stream_rx_buff[0], (uint8_t *)&algo_param.cur_bat_v, 4);
11     memcpy(&stream_rx_buff[4], (uint8_t *)&algo_param.cur_chg_i, 4);
12     memcpy(&stream_rx_buff[8], (uint8_t *)&algo_param.cur_dis_i, 4);
13     memcpy(&stream_rx_buff[12], (uint8_t *)&algo_param.cur_sol_v, 4);
14     memcpy(&stream_rx_buff[16], (uint8_t *)&algo_param.cur_sol_i, 4);
15     memcpy(&stream_rx_buff[20], (uint8_t *)&algo_param.cur_ac_load_i, 4);
16     memcpy(&stream_rx_buff[24], (uint8_t *)&inverter_vol, 4);
17     memcpy(&stream_rx_buff[28], (uint8_t *)&temperature1, 4);
18     memcpy(&stream_rx_buff[32], (uint8_t *)&vol_ref, 4);
19     memcpy(&stream_rx_buff[36], (uint8_t *)&mains, 1);
20     memcpy(&stream_rx_buff[37], (uint8_t *)&temperature2, 4);
21     res_to_utility(CMD_GET_REAL_VAL, stream_rx_buff, 41);
22 }

```

13.9 Process command

The following code branches to the appropriate request from the APP. Depending on the type of the request the device branches the command to different function

```

1  /*
2  * @Info:    The code is used to perform the following
3  *           1. Authentication
4  *           2. Set time
5  *           3. Get ADC value
6  *           4. Get switch status
7  *           5. Set switch status
8  *           6. Test connectivity with server
9  *           7. Read from EEPROM
10 *           8. Write to EEPROM
11 *           9. Get real values
12 *           10. Buzzer sound
13 *           11. Exit calibration/configuration mode
14 * @Param:   None
15 * @Return:  None
16 */
17 void process_labview_cmd(){
18     switch(uart_cmd){
19         /* Authentication */
20         case REQ_CON:
21             ESP_LOGI("inside_process_labview_cmd", "Authentication request command");
22             send_con_res_config_calib();
23             break;
24         /* Get ADC value */
25         case CMD_GET_ADC:
26             ESP_LOGI("inside_process_labview_cmd", "Get ADC command");
27             read_send_adc_value();

```

```

28         break;
29     /* Get the status of device state */
30     case CMD_GET_SW:
31         ESP_LOGI("inside_process_labview_cmd", "Set switch command");
32         stream_rx_buff[0] = switch_state;
33         res_to_utility(CMD_GET_SW, stream_rx_buff, 1);
34         break;
35     /* Set the state of the device */
36     case CMD_SET_SW:
37         ESP_LOGI("inside_process_labview_cmd", "Get switch status command");
38         turtle_set_sw(stream_tx_buff[0]);
39         stream_rx_buff[0] = switch_state;
40         res_to_utility(CMD_SET_SW, stream_rx_buff, 1);
41         break;
42     /* Test connectivity */
43     case CMD_TEST_CONFIG:
44         ESP_LOGI("inside_process_labview_cmd", "Test configuration command");
45         res_to_utility(CMD_TEST_CONFIG, stream_rx_buff, 0);
46         load_wifi_param();
47         xSemaphoreGive(testConnectionSemaphore);
48         break;
49     /* Read EEPROM */
50     case CMD_READ_EEPROM:
51         ESP_LOGI("inside_process_labview_cmd", "EEPROM read command");
52         read_send_eeprom_val();
53         break;
54     /* Write EEPROM */
55     case CMD_WRITE_EEPROM:
56         ESP_LOGI("inside_process_labview_cmd", " EEPROM write command");
57         write_const_to_eeprom();
58         break;
59     /* Get sensor values from device */
60     case CMD_GET_REAL_VAL:
61         ESP_LOGI("inside_process_labview_cmd", "Sensor value command");
62         send_real_values();
63         break;
64     case CMD_EXIT:
65     /* Exit from task */
66         ESP_LOGI("inside_process_labview_cmd", "Exit command received");
67         res_to_utility(CMD_EXIT, stream_rx_buff, 0);
68
69         break;
70     default:
71         ESP_LOGI("inside_process_labview_cmd", "Wrong command received");
72         break;
73 }
74 socket_client_status = connected_socket;
75 }

```

14 esp32s2_https_protocol.c

14.1 Variables declared in esp32s2_https_protocol.c

The following are the variables used in the https protocol c file

```

1 extern https_request_buffer_body_t https_request_buffer_body;
2 extern uint8_t stop_lptask;
3 extern uint8_t i2cFlag;
4 extern uint8_t server_cert_pem_start[2048];
5 extern uint8_t connection_stat_check, connection_stat;
6 extern char dev_id[12];
7
8 time_t timeinfor = 0;
9 uint8_t packet_length = 10;
10 struct FetchParms msgStruct;
11 algo_param_t http_data_get;
12 char *https_receive_buffer = NULL;
13 int index_https_buffer = 0;

```

14.2 HTTPS Event Handler

The following code is used to handle the https events

```

1 /*
2  * @Info:   HTTPS event handler
3  * @Param:  http client event pointer
4  * @Return: error_type
5  */
6 esp_err_t client_event_handler(esp_http_client_event_t *evt)
7 {
8     assert(evt != NULL);
9     switch (evt->event_id)
10     {
11     case HTTP_EVENT_ERROR:
12         ESP_LOGI(TAG, "HTTP_EVENT_ERROR");
13         break;
14     case HTTP_EVENT_ON_CONNECTED:
15         ESP_LOGI(TAG, "HTTP_EVENT_ON_CONNECTED");
16         break;
17     case HTTP_EVENT_HEADER_SENT:
18         ESP_LOGI(TAG, "HTTP_EVENT_HEADER_SENT");
19         break;
20     case HTTP_EVENT_ON_HEADER:
21         ESP_LOGI(TAG, "HTTP_EVENT_ON_HEADER, key=%s, value=%s", evt->header_key, evt->
            header_value);
22         break;
23     case HTTP_EVENT_ON_DATA:
24         time(&timeinfor);
25         if(stop_lptask==0){
26             ESP_LOGI("TAG","Response time is %ld",timeinfor);
27             if (https_receive_buffer == NULL){
28                 https_receive_buffer = (char *)malloc(evt->data_len);
29             }
30             else{
31                 ESP_LOGI(TAG,"Reallocating memory");
32                 https_receive_buffer = (char *)realloc(https_receive_buffer, evt->data_len +
                    index_https_buffer);
33             }
34             ESP_LOGI(TAG,"Reallocating memory done");
35             memcpy(&https_receive_buffer[index_https_buffer], evt->data, evt->data_len);
36             index_https_buffer += evt->data_len;
37         }
38         break;
39     case HTTP_EVENT_ON_FINISH:
40         if(stop_lptask==0){
41             ESP_LOGI(TAG, "HTTP_EVENT_ON_FINISH");
42             https_receive_buffer = (char *)realloc(https_receive_buffer, index_https_buffer + 1);
43             memcpy(&https_receive_buffer[index_https_buffer], "\0", 1);
44             ESP_LOGI(TAG, "%s", https_receive_buffer);
45             httpsmsgparse(https_receive_buffer,&https_request_buffer_body);
46             index_https_buffer=0;

```

```

47 free(https_receive_buffer);
48 https_receive_buffer=NULL;
49 }
50 break;
51 case HTTP_EVENT_DISCONNECTED:
52 ESP_LOGI(TAG, "HTTP_EVENT_DISCONNECTED");
53 break;
54 }
55 return ESP_OK;
56 }

```

14.3 Prepare configuration data

The following code is used to convert the configuration values into JSON packets to be sent to the server

```

1  /*
2   * @Info:   The following code prepares the configuration parameters as JSON to be sent
3   *          to the server
4   * @Param:  Buffer address to store the JSON
5   * @Return: None
6   */
7 void prepare_configuration_data(char *out){
8     extern float max_dc_load_i,max_ac_load_i,bat_max_v;
9     extern float Rated_Bat_AH,bat_frc_trip_exit_v_thr;
10    extern float sol_inst, bat_mains_chg_in_v_thr, bat_vol;
11
12    extern uint8_t battery_type, bat_age, max_frc_per_day, morn_chg_inh_flag;
13    extern uint8_t even_chg_inh_flag, bat_abs_intr, bat_sol_equ_intr;
14    extern uint8_t bat_sol_equ_dur, useable_soc_recal_dur;
15
16    sprintf(out,
17    "{"
18    "\"dev_id\": \"%s\", \"
19    \"packet\": \"
20    \"[\"
21    \"\"%0.3f\", \" /* dc overload */
22    \"\"%0.3f\", \" /* ac overload */
23    \"\"%d\", \" /* battery type */
24    \"\"%0.3f\", \" /* bat max voltage */
25    \"\"%0.3f\", \" /* bat ah capacity */
26    \"\"%d\", \" /* bat age */
27    \"\"%0.3f\", \" /* frc trip exit vol */
28    \"\"%d\", \" /* no of ft per day */
29    \"\"%0.3f\", \" /* sol capacity */
30    \"\"%0.3f\", \" /* mains charge in voltage */
31    \"\"%0.3f\", \" /* battery voltage */
32    \"\"%d\", \" /* morn chg inh flag */
33    \"\"%d\", \" /* even chg inh flag */
34    \"\"%d\", \" /* per abs int */
35    \"\"%d\", \" /* per equ int */
36    \"\"%d\", \" /* per equ dur */
37    \"\"%d\", \" /* per use soc */
38    \"]\",dev_id,max_dc_load_i,max_ac_load_i,battery_type,bat_max_v,Rated_Bat_AH,bat_age,
39    bat_frc_trip_exit_v_thr, max_frc_per_day, sol_inst, bat_mains_chg_in_v_thr,
40    bat_vol, morn_chg_inh_flag, even_chg_inh_flag, bat_abs_intr, bat_sol_equ_intr,
41    bat_sol_equ_dur, useable_soc_recal_dur);
42 }

```

14.4 Prepare calibration data

The following code is used to convert the calibration values into JSON packets to be sent to the server

```

1  /*
2   * @Info:   The following code prepares the calibration parameters as JSON to be sent to
3   *          the server
4   * @Param:  Buffer address to store the JSON
5   * @Return: None
6   */
7 void prepare_calibration_data(char *out){
8     sprintf(out,
9     "{"

```

```

9      "\"dev_id\\\": \"%s\\\", \"
10      "\"packet\\\": \"
11      \"[\"
12      \"%0.3f\\\", \"
13      \"%0.3f\\\", \"
14      \"%0.3f\\\", \"
15      \"%0.3f\\\", \"
16      \"%0.3f\\\", \"
17      \"%0.3f\\\", \"
18      \"%0.3f\\\", \"
19      \"%0.3f\\\", \"
20      \"%0.3f\\\", \"
21      \"%0.3f\\\", \"
22      \"%0.3f\\\", \"
23      \"%0.3f\\\", \"
24      \"%0.3f\\\", \"
25      \"%0.3f\\\", \"
26      \"%d\\\", \"
27      \"%0.3f\\\" \"
28      \"]}\" , dev_id, uc_bat_v_const, pac_bat_v_const, pac_bat_chg_i_const, pac_bat_dis_i_const,
29      pac_bat_chg_i_offset, pac_bat_dis_i_offset, pac_bat_p_const,
30      pac_sol_v_const, uc_sol_v_const, pac_sol_i_const, pac_sol_i_offset, pac_sol_p_const,
      uc_temp1_const, uc_temp2_const, ct_load_i_offset, ct_load_i_const );

```

14.5 Prepare system threshold data

The following code is used to convert the system threshold values into JSON packets to be sent to the server

```

1  /*
2  * @Info:   The following code prepares the system threshold parameters as JSON to be
3  *          sent to the server
4  * @Param:  Buffer address to store the JSON
5  * @Return: None
6  */
7  void prepare_systr_data(char *out){
8      sprintf(out,
9      "{
10      "\"dev_id\\\": \"%s\\\", \"
11      "\"packet\\\": \"
12      \"[\"
13      \"%d\\\", \"
14      \"%0.3f\\\", \"
15      \"%0.3f\\\", \"
16      \"%0.3f\\\", \"
17      \"]}\" , dev_id, dev_state_flag, useable_SOC1, bat_frc_trip_exit_v_thr, eve_soc_corr,
18      night_soc_corr);

```

14.6 Prepare useable SoC data

The following code is used to convert the useable SoC values into JSON packets to be sent to the server

```

1  /*
2  * @Info:   The following code prepares the useable SoC parameters as JSON to be sent to
3  *          the server
4  * @Param:  Buffer address to store the JSON
5  * @Return: None
6  */
7  void prepare_useable_soc_data(char *out){
8      sprintf(out,
9      "{
10      "\"dev_id\\\": \"%s\\\", \"
11      "\"packet\\\": \"
12      \"[\"
13      \"%02d-%02d-%02d %02d:%02d:%02d\\\", \"
14      \"%0.3f\\\", \"
15      \"%0.3f\\\" \"

```

```

15     "]]",dev_id,load.date_time[0],load.date_time[1],load.date_time[2],load.date_time[3],
16     load.date_time[4],dev_state_flag,useable_SOC1,bat_frc_trip_exit_v_thr);
    }

```

14.7 Prepare log data

The following code is used to convert the log values into JSON packets to be sent to the server

```

1  /*
2  * @Info:   The following code prepares the log data parameters as JSON to be sent to
3  *           the server
4  * @Param:  Buffer address to store the JSON
5  * @Return: None
6  */
7  void prepare_LOGDATA(char *out,uint8_t cond){
8      time(&timeinfor);
9      date_time = localtime(&timeinfor);
10     sprintf(out,
11     "{"
12     "\"token\": \"12345678\", \"
13     "\"dev_id\": \"%s\", \"
14     "\"messagetype\": \"log\", \"
15     "\"readflag\": 0, \"
16     "\"packet\": [\",dev_id
17     );
18     if(cond){
19         for(int i=0;i<packet_length;i++){
20             /*Add EEPROM read here*/
21             while(i2cFlag){
22                 }
23                 i2cFlag=1;
24                 turtle_log_get((uint8_t *)&http_data_get,i);
25                 i2cFlag=0;
26                 sprintf(out+strlen(out),
27                 "["
28                 "\"%s\", \"
29                 \"%02d-%02d-%02d %02d:%02d:%02d\", \"
30                 \"%0.2f\", \"
31                 \"%0.2f\", \"
32                 \"%0.2f\", \"
33                 \"%0.2f\", \"
34                 \"%0.04\", \"
35                 \"%0.76\", \"
36                 \"%0.2f\", \"
37                 \"%1.38\", \"
38                 \"%0.2f\", \"
39                 \"%d\", \"
40                 \"%0.2f\", \"
41                 \"%d\", \"
42                 \"%0.2f\", \"
43                 \"%0.2f\", \"
44                 \"%0.2f\", \"
45                 \"%0.2f\", \"
46                 \"%0.2f\", \"
47                 \"%d\", \"
48                 \"%d\", \"
49                 \"%d\", \"
50                 \"%d\", \"
51                 \"%d\", \"
52                 \"%0.2f\", \"
53                 \"%d\", \"
54                 \"%d\", \"
55                 \"%d\", \"
56                 \"%0.2f\"]\",dev_id,(http_data_get.date_time[2])+1900,http_data_get.
57                 date_time[1],http_data_get.date_time[0],http_data_get.date_time[3],http_data_get.
58                 date_time[4],http_data_get.date_time[5],http_data_get.cur_bat_v,
59                 http_data_get.cur_chg_i,http_data_get.cur_dis_i,http_data_get.cur_sol_i,
60                 http_data_get.cur_sol_v,http_data_get.cur_ac_load_i,http_data_get.cur_temp,
61                 http_data_get.bat_cur_cap_coul,
62                 http_data_get.dev_algo_state,http_data_get.sol_e,http_data_get.sol_bat_e,
63                 http_data_get.bat_dis_e,http_data_get.bat_ac_chg_e,http_data_get.ac_load_e,
64                 http_data_get.num_sum_samples,

```



```

59      http_data_get.log_type,http_data_get.state_change_reason,http_data_get.
      dev_stat,http_data_get.algo_stat,http_data_get.cur_inv_v,http_data_get.error,
      http_data_get.daystartflag,
60      http_data_get.resetflag,http_data_get.utility_savings
61      );
62      if(i<packet_length-1)
63      {
64          strcat(out,",");
65      }
66  }
67  }
68  sprintf(out+strlen(out),
69  "]"
70  "]"
71  );
72  ESP_LOGI("TAG"," length is %d",strlen(out));
73  ESP_LOGI(TAG,"Data to be sent %s",out);
74  }

```

14.8 Test connectivity to server

The following code is used to test connectivity with the server

```

1  /*
2  * @Info: The following function is used to connect with the server while checking
      connectivity using mobile app
3  * @Param: None
4  * @Param: None
5  */
6  void test_server_connection(void){
7      msgStruct.OnGotData = NULL;
8      msgStruct.method = Post;
9      msgStruct.headerCount = 0;
10     char buffer[0x2FFF];
11     prepare_LOGDATA(buffer,0);
12     msgStruct.body = buffer;
13     char url[2048];
14     read_spiffs(url,log_data);
15     fetch(url, &msgStruct);
16 }

```

14.9 Send data to server

The following code is used to send data to the server

```

1  /*
2  * @Info: The following function is used to log data to server
3  * @Param: None
4  * @Param: None
5  */
6  void send_DATA_to_server(void){
7      msgStruct.OnGotData = NULL;
8      msgStruct.method = Post;
9      msgStruct.headerCount = 0;
10     char buffer[0x2FFF];
11     prepare_LOGDATA(buffer,1);
12     msgStruct.body = buffer;
13     char url[2048];
14     read_spiffs(url,log_data);
15     fetch(url, &msgStruct);
16 }

```

14.10 Bi-directional communication with server

The following code is used in the bi-directional communication with the server

```

1  /*
2  * @Info: The following code is used to perform bi-directional communication with the
      server
3  * @Param: HTTPS request body

```

```

4  * @Return: None
5  */
6  void bidirectional_communication_commands(https_request_buffer_body_t *param){
7      msgStruct.OnGotData = NULL;
8      msgStruct.method = Post;
9      msgStruct.headerCount = 0;
10     char buffer[0x2FFF];
11     char url[2048];
12     ESP_LOGI(TAG,"Data is %d",https_req_buff_peek(param));
13     switch(https_req_buff_peek(param)){
14         case update_certificate:
15             read_domain(url);
16             strcat(url,"/api/getCert/");
17             msgStruct.method = Get;
18             fetch(url,&msgStruct);
19             break;
20         case get_information:
21             sprintf(buffer,"{\"dev_id\":\"%s\"}",dev_id);
22             ESP_LOGI(TAG,"Msg sent is %s",buffer);
23             msgStruct.body = buffer;
24             read_domain(url);
25             strcat(url,"/api/getReadWriteInfo/");
26             fetch(url, &msgStruct);
27             break;
28         case update_calibration:
29             prepare_calibration_data(buffer);
30             ESP_LOGI(TAG,"Msg sent is %s",buffer);
31             msgStruct.body = buffer;
32             read_domain(url);
33             strcat(url,"/api/updateCalibration/");
34             fetch(url, &msgStruct);
35             break;
36         case update_configuration:
37             prepare_configuration_data(buffer);
38             ESP_LOGI(TAG,"Msg sent is %s",buffer);
39             msgStruct.body = buffer;
40             read_domain(url);
41             strcat(url,"/api/updateConfig/");
42             fetch(url, &msgStruct);
43             break;
44         case update_system_threshold:
45             prepare_systhr_data(buffer);
46             ESP_LOGI(TAG,"Msg sent is %s",buffer);
47             msgStruct.body = buffer;
48             read_domain(url);
49             strcat(url,"/api/updateThreshold/");
50             fetch(url, &msgStruct);
51             break;
52         case send_response:
53             sprintf(buffer,"{\"dev_id\":\"%s\"}",dev_id);
54             ESP_LOGI(TAG,"Msg sent is %s",buffer);
55             msgStruct.body = buffer;
56             read_domain(url);
57             strcat(url,"/api/updateWriteStatus/");
58             fetch(url, &msgStruct);
59             break;
60         case update_useable_soc:
61             prepare_useable_soc_data(buffer);
62             ESP_LOGI(TAG,"Msg sent is %s",buffer);
63             msgStruct.body = buffer;
64             read_domain(url);
65             strcat(url,"/api/updateSOC/");
66             fetch(url, &msgStruct);
67             break;
68     }
69 }

```

14.11 HTTPS communication method

The following code is used to transfer the data to the server over HTTPS. The code manages both POST and GET request of the HTTPS client.

```

1  /*
2  * @Info: The following code is used to transfer data over HTTPS
3  * @Param:
4  *     *url - URL to hit
5  *     *fetchParams - structure containing data and headers to be transferred
6  * @Return: None
7  */
8  void fetch(char *url, struct FetchParams *fetchParams){
9      esp_http_client_config_t clientConfig = {
10         .url = (char *) url,
11         .event_handler = client_event_handler,
12         .cert_pem = (char *)server_cert_pem_start,
13         .user_data = fetchParams,
14         .password = (char *) https_password,
15         .username = (char *) dev_id,
16         .auth_type = HTTP_AUTH_TYPE_BASIC,
17     };
18
19     esp_http_client_handle_t client = esp_http_client_init(&clientConfig);
20
21     if (fetchParams->method == Post){
22         esp_http_client_set_method(client, HTTP_METHOD_POST);
23     }
24
25     for (int i = 0; i < fetchParams->headerCount; i++){
26         esp_http_client_set_header(client, fetchParams->header[i].key, fetchParams->header[i].val);
27     }
28
29     if(fetchParams->body != NULL){
30         esp_http_client_set_post_field(client, fetchParams->body, strlen(fetchParams->body));
31     }
32
33     esp_err_t err = esp_http_client_perform(client);
34     fetchParams->status = esp_http_client_get_status_code(client);
35
36     if (err == ESP_OK){
37         ESP_LOGI(TAG, "HTTP GET status = %d",
38             esp_http_client_get_status_code(client));
39         if((fetchParams->status>=200)&&(fetchParams->status<400)){
40             if(connection_stat_check){
41                 connection_status_obtained=1;
42                 connection_stat_check=0;
43                 connection_stat=2;
44                 blink(led_green_color, FULL_ON);
45                 vTaskDelay(3000/portTICK_PERIOD_MS);
46                 extern uint8_t stop_algorithm;
47                 stop_algorithm=0;
48             }
49             else{
50                 extern operations_parameters_t least_priority_operations_parameters;
51                 least_priority_operations_parameters.status=lpos_response_received;
52                 least_priority_operations_parameters.wait_time=0;
53             }
54         }
55     }
56     else{
57         ESP_LOGE(TAG, "HTTP GET request failed: %s", esp_err_to_name(err));
58         if(connection_stat_check){
59             connection_status_obtained=1;
60             connection_stat_check=0;
61             connection_stat=2;
62             blink(led_blue_color, FULL_ON);
63             vTaskDelay(3000/portTICK_PERIOD_MS);
64             extern uint8_t stop_algorithm;
65             stop_algorithm=0;
66         }
67     }
68
69     esp_http_client_cleanup(client);
70 }

```

15 esp32s2_parse_data.c

15.1 Variables used in parse_data.c file

The following array is declared to easily parse data from the server message

```

1 extern https_request_buffer_body_t https_request_buffer_body;
2
3 uint16_t write_info_address[98]={
4     EE_SERV_IP_ADDR, EE_SERV_PORT_ADDR, EE_AP_SSID_ADDR, EE_AP_PSW_ADDR,
5     EE_DC_OVER_I_THR_ADDR, EE_AC_OVER_I_THR_ADDR,
6     EE_BAT_TYPE_ADDR, EE_BAT_MAX_VOLT_ADDR,
7     EE_BAT_CAP_ADDR, EE_BAT_AGE_ADDR,
8     EE_BAT_FRC_EXIT_VOLT_ADDR, EE_FT_NUM_ADDR,
9     EE_SOL_CAP,
10    EE_BAT_MAINS_CHG_IN_VOLT_ADDR, EE_BAT_MAINS_CHG_OUT_VOLT_ADDR,
11    EE_MORN_CHG_INH_EN_ADDR, EE_EVEN_CHG_INH_EN_ADDR, EE_ABS_INTR_ADDR,
12    EE_BAT_EQU_INTR_ADDR, EE_BAT_EQU_DUR_ADDR,
13    EE_DEV_SER_NO_ADDR, EE_DEVICE_TYPE_ADDR,
14    EE_UC_BAT_V_CONST_ADDR, EE_PAC_BAT_V_CONST_ADDR,
15    EE_PAC_BAT_CHG_I_CONST_ADDR, EE_PAC_BAT_DIS_I_CONST_ADDR,
16    EE_PAC_BAT_CHG_I_OFF_ADDR, EE_PAC_BAT_DIS_I_OFF_ADDR,
17    EE_PAC_BAT_P_CONST_ADDR, EE_UC_SOL_V_CONST_ADDR,
18    EE_PAC_SOL_V_CONST_ADDR, EE_PAC_SOL_I_CONST_ADDR, EE_PAC_SOL_I_OFF_ADDR,
19    EE_PAC_SOL_P_CONST_ADDR,
20    EE_UC_TEMP1_CONST_ADDR, EE_UC_TEMP2_CONST_ADDR, EE_CT_LOAD_I_OFST_ADDR,
21    EE_CT_LOAD_I_CONST_ADDR,
22    DEV_USEABLE_SOC_MULTIPLIER, DEV_EQU_HYS,
23    DEV_ABS_MF_PER, EQ_MF_PER,
24    VFT_EXIT_HIGH, VFT_EXIT_LOW,
25    DEV_SYS_FLAG, DAILY_LOAD_THR,
26    WEEKLY_LOAD_THR, NUM_DAYS,
27    USEABLE_SOC, RES_ENE_MUL, VFT_EXT_THR,
28    EVE_BAT_SOC_COR_LOW_LIM, EVE_BAT_SOC_COR_HIGH_LIM,
29    SOL_MAX_CUR, USEABLE_SOC_STRT,
30    FT_ENTRY_STRT_DAY, FT_EXIT_STRT_DAY,
31    STATE_CHANGE_LOG_UNITS, STATE_CHANGE_LOG_UNITS_NO_WIFI,
32    SOL_E_CORR, SER_ASSERT_STATE, SER_STATE_ASSERTED_TIM_HR, SER_STATE_ASSERTED_TIM_MIN,
33    SER_STATE_ASSERTED_DUR, FT_EXIT_TIM, MORN_SOL_PRED, EVEN_SOL_PRED,
34    DAY1_MORN_LOAD, DAY1_NGT_LOAD,
35    DAY2_MORN_LOAD, DAY2_NGT_LOAD,
36    DAY3_MORN_LOAD, DAY3_NGT_LOAD,
37    DAY4_MORN_LOAD, DAY4_NGT_LOAD,
38    DAY5_MORN_LOAD, DAY5_NGT_LOAD,
39    DAY6_MORN_LOAD, DAY6_NGT_LOAD,
40    DAY7_MORN_LOAD, DAY7_NGT_LOAD,
41    COMMON_THR, DEV_SOC_VOL_TABLE,
42    DAY1_MORN_SOL, DAY1_AFT_SOL,
43    DAY2_MORN_SOL, DAY2_AFT_SOL,
44    DAY3_MORN_SOL, DAY3_AFT_SOL
45 };
46
47 uint8_t write_info_length[98]={
48     SERV_IP_LEN, SERV_PORT_LEN, AP_SSID_LEN, AP_PSW_LEN,
49     DC_OVER_I_THR_LEN, AC_OVER_I_THR_LEN,
50     EE_BAT_TYPE_LEN, BAT_MAX_VOLT_LEN,
51     EE_BAT_CAP_LEN, EE_BAT_AGE_LEN,
52     BAT_FRC_EXIT_VOLT, EE_FT_NUM_LEN,
53     EE_SOL_CAP_LEN,
54     BAT_MAINS_CHG_IN_VOLT_LEN, BAT_MAINS_CHG_OUT_VOLT_LEN,
55     MORN_CHG_INH_EN_LEN, EVEN_CHG_INH_EN_LEN, EE_ABS_INTR_LEN, BAT_EQU_INTR_LEN,
56     BAT_EQU_DUR_LEN,
57     DEV_SER_NO_LEN, DEVICE_TYPE_LEN,
58     UC_BAT_V_CONST_LEN, PAC_BAT_V_CONST_LEN,
59     PAC_BAT_CHG_I_CONST_LEN, PAC_BAT_DIS_I_CONST_LEN,
60     PAC_BAT_CHG_I_OFF_LEN, PAC_BAT_DIS_I_OFF_LEN,
61     PAC_BAT_P_CONST_LEN, UC_SOL_V_CONST_LEN,
62     PAC_SOL_V_CONST_LEN, PAC_SOL_I_CONST_LEN, PAC_SOL_I_OFF_LEN, PAC_SOL_P_CONST_LEN,
63     UC_TEMP1_CONST_LEN, UC_TEMP2_CONST_LEN, CT_LOAD_I_OFST_LEN, CT_LOAD_I_CONST_LEN,
64     DEV_USEABLE_SOC_MULTIPLIER_LEN, DEV_EQU_HYS_LEN,
65     DEV_ABS_MF_PER_LEN, EQ_MF_PER_LEN,
66     VFT_EXIT_HIGH_LEN, VFT_EXIT_LOW_LEN,
67     DEV_SYS_FLAG_LEN, DAILY_LOAD_THR_LEN,

```

```

64 WEEKLY_LOAD_THR_LEN, NUM_DAYS_LEN,
65 USEABLE_SOC_LEN, RES_ENE_MUL_LEN, VFT_EXT_THR_LEN,
66 EVE_BAT_SOC_COR_LOW_LIM_LEN, EVE_BAT_SOC_COR_HIGH_LIM_LEN,
67 SOL_MAX_CUR_LEN, USEABLE_SOC_STRT_LEN,
68 FT_ENTRY_STRT_DAY_LEN, FT_EXIT_STRT_DAY_LEN,
69 STATE_CHANGE_LOG_UNITS_LEN, STATE_CHANGE_LOG_UNITS_NO_WIFI_LEN,
70 SOL_E_CORR_LEN, SER_ASSERT_STATE_LEN, SER_STATE_ASSERTED_TIM_HR_LEN,
    SER_STATE_ASSERTED_TIM_MIN_LEN,
71 SER_STATE_ASSERTED_DUR_LEN, FT_EXIT_TIM_LEN, MORN_SOL_PRED_LEN, EVEN_SOL_PRED_LEN,
72 DAY1_MORN_LOAD_LEN, DAY1_NGT_LOAD_LEN,
73 DAY2_MORN_LOAD_LEN, DAY2_NGT_LOAD_LEN,
74 DAY3_MORN_LOAD_LEN, DAY3_NGT_LOAD_LEN,
75 DAY4_MORN_LOAD_LEN, DAY4_NGT_LOAD_LEN,
76 DAY5_MORN_LOAD_LEN, DAY5_NGT_LOAD_LEN,
77 DAY6_MORN_LOAD_LEN, DAY6_NGT_LOAD_LEN,
78 DAY7_MORN_LOAD_LEN, DAY7_NGT_LOAD_LEN,
79 COMMON_THR_LEN, DEV_SOC_VOL_TABLE_LEN,
80 DAY1_MORN_SOL_LEN, DAY1_AFT_SOL_LEN,
81 DAY2_MORN_SOL_LEN, DAY2_AFT_SOL_LEN,
82 DAY3_MORN_SOL_LEN, DAY3_AFT_SOL_LEN
83 };

```

15.2 Queue for holding request to server

The following code deals with the queue which is used to hold the request to the server from the device

```

1 void https_req_buff_write(https_request_buffer_body_t *param,uint8_t c){
2     uint8_t i;
3     i = (param->https_request_buffer_head + 1) % HTTPS_REQ_BUFFER_SIZE;
4     if (i == param->https_request_buffer_tail){
5         param->https_request_buffer_tail = (param->https_request_buffer_tail + 1) %
        HTTPS_REQ_BUFFER_SIZE;
6     }
7     param->https_request_buffer[param->https_request_buffer_head] = c;
8     param->https_request_buffer_head = i;
9 }

```

15.3 Check if data available in queue

The following code is used to check if there is any request to be serviced in the queue

```

1 uint8_t https_req_buff_available(https_request_buffer_body_t *param){
2     uint8_t buff_size = (HTTPS_REQ_BUFFER_SIZE + param->https_request_buffer_head - param
        ->https_request_buffer_tail) % HTTPS_REQ_BUFFER_SIZE;
3     ESP_LOGI(TAG,"Buffer size %d",buff_size);
4     return buff_size;
5 }

```

15.4 Peek data in the queue

The following code is used to peek into the data in the queue. The function will not update the tail of the queue.

```

1 uint8_t https_req_buff_peek(https_request_buffer_body_t *param){
2     return(param->https_request_buffer[param->https_request_buffer_tail]);
3 }

```

15.5 Flush queue

The following function is used to clear the queue values

```

1 void https_req_buff_flush(https_request_buffer_body_t *param){
2     if(param->https_request_buffer_head != param->https_request_buffer_tail){
3         param->https_request_buffer_tail = (param->https_request_buffer_tail + 1) %
        HTTPS_REQ_BUFFER_SIZE;
4     }
5 }

```

15.6 Parse response from server

The following function is used to detect the response from the server and take appropriate actions. The data received from the server can be write/read or update certificate requests.

```

1 void httpsmsgparse(char *https_receive_buffer,https_request_buffer_body_t *param){
2     if( https_req_buff_peek(param) == update_certificate ){
3         update_spiffs(https_receive_buffer,update_certificate);
4     }
5     else{
6         cJSON *payload = cJSON_Parse(https_receive_buffer);
7         ESP_LOGI(TAG,"Parsing data");
8         if(detect_payload(payload)){
9             ESP_LOGI(TAG,"Detected device");
10            if(detect_msgType(payload)){
11                ESP_LOGI(TAG,"Message type matched");
12                detect_write_msg(payload);
13                detect_read_msg(payload);
14            }
15        }
16        cJSON_Delete(payload);
17    }
18 }

```

15.7 Detect payload from the server

The following code is used to detect if the payload is meant for the device

```

1 int detect_payload(cJSON *payload){
2     cJSON *device_id = cJSON_GetObjectItem(payload, "dev_id");
3     if(device_id!=NULL){
4         extern char dev_id[12];
5         if(strcmp(device_id->valstring,"A24XA1000002")==0){return 1;}
6         else{return 0;}
7     }
8     else{return 0;}
9 }

```

15.8 Detect payload type

The following code is used to check if the request is on read/write request type

```

1 int detect_msgType(cJSON *payload){
2     cJSON *msg_type = cJSON_GetObjectItem(payload,"messagetype");
3     if(msg_type!=NULL){
4         if(strcmp(msg_type->valstring,"read write response")==0){return 1;}
5         else{return 0;}
6     }
7     else{return 0;}
8 }

```

15.9 Detect write request

The following code is used to check if the received data is for writing into the EEPROM

```

1 void detect_write_msg(cJSON *payload){
2     cJSON *writeInfo = cJSON_GetObjectItem(payload, "writeInfo");
3     if(writeInfo!=NULL){
4         parse_write_msg(writeInfo);
5     }
6 }

```

15.10 Detect read request

The following code is used to check if the command received is to read from the EEPROM

```

1 void detect_read_msg(cJSON *payload){
2     cJSON *readInfo = cJSON_GetObjectItem(payload, "readInfo");
3     if(readInfo!=NULL){
4         parse_read_msg(readInfo);
5     }
6 }

```

15.11 Parse message sent in write request

The following code differentiates between int, float and 8bit interger values received from the server and writes into the appropriate EEPROM location

```

1 void parse_write_msg(cJSON *writeInfo){
2     ESP_LOGI(TAG,"inside parse_write_msg");
3     https_req_buff_write(&https_request_buffer_body,send_response);
4     char temp_addr[5];
5     for(int i=0;i<sizeof(write_info_address)/sizeof(uint16_t);i++){
6         sprintf(temp_addr,"%x",write_info_address[i]);
7         cJSON *readvalue = cJSON_GetObjectItem(writeInfo,temp_addr);
8         if(readvalue!=NULL){
9             if(write_info_length[i]==4){
10                 ESP_LOGI(TAG,"Float: Writing to address %s and the value is %f",temp_addr,atof(
11                     readvalue->valuelstring));
12                 ee24lc256_write_float(write_info_address[i],atof(readvalue->valuelstring));
13                 vTaskDelay(100/portTICK_PERIOD_MS);
14             }
15             else{
16                 if(write_info_length[i]==2){
17                     ESP_LOGI(TAG,"16bit int: Writing to address %s and the value is %d",temp_addr,
18                         atoi(readvalue->valuelstring));
19                     ee24lc256_write_int16(write_info_address[i],atoi(readvalue->valuelstring));
20                     vTaskDelay(100/portTICK_PERIOD_MS);
21                 }
22                 else{
23                     ESP_LOGI(TAG,"8bit int: Writing to address %s and the value is %d",temp_addr,
24                         atoi(readvalue->valuelstring));
25                     ee24lc256_write_byte(write_info_address[i],atoi(readvalue->valuelstring));
26                     vTaskDelay(100/portTICK_PERIOD_MS);
27                 }
28             }
29         }
30     }
31     load_wifi_param();
32     load_sensor_constants();
33 }

```

15.12 Parse message sent in read request

The following code gets the read requests from the server and initiates the appropriate command

```

1 void parse_read_msg(cJSON *readInfo){
2     if(parse_response(readInfo->valueint,calibration_req)){ ESP_LOGI(TAG,"parse read:
3         calibration"); https_req_buff_write(&https_request_buffer_body,update_calibration);
4     };
5     if(parse_response(readInfo->valueint,configuration_req)){ ESP_LOGI(TAG,"parse read:
6         configuration"); https_req_buff_write(&https_request_buffer_body,
7         update_configuration); };
8     if(parse_response(readInfo->valueint,system_thr_req)){ ESP_LOGI(TAG,"parse read:
9         system threshold"); https_req_buff_write(&https_request_buffer_body,
10        update_system_threshold); };
11    if(parse_response(readInfo->valueint,useable_soc_req)){ ESP_LOGI(TAG,"parse read:
12        useable soc"); https_req_buff_write(&https_request_buffer_body,update_useable_soc);
13    };
14    if(parse_response(readInfo->valueint,factory_update)){ ESP_LOGI(TAG,"parse read:
15        firmware update"); extern uint8_t fw_update; fw_update=1; };
16    if(parse_response(readInfo->valueint,factory_reset)){ ESP_LOGI(TAG,"parse read:
17        factory reset"); extern xSemaphoreHandle factoryReset; xSemaphoreGive(factoryReset)
18    };
19 }

```

16 esp32s2_task.c

16.1 Semaphore creation

The following code is used to create binary semaphores to use in algorithm

```

1 void createSemaphores(void){
2     serverReset = xSemaphoreCreateBinary();
3     factoryReset = xSemaphoreCreateBinary();
4     otaSemaphore = xSemaphoreCreateBinary();
5     logdataSemaphore = xSemaphoreCreateBinary();
6     connectionSemaphore = xSemaphoreCreateBinary();
7     switchpressSemaphore = xSemaphoreCreateBinary();
8     switchpressConfirmedSemaphore = xSemaphoreCreateBinary();
9     serverSemaphore = xSemaphoreCreateBinary();
10    testConnectionSemaphore = xSemaphoreCreateBinary();
11    pacAlertSemaphore = xSemaphoreCreateBinary();
12    ledTaskSemaphore = xSemaphoreCreateBinary();
13    overVoltageProtectionSemaphore = xSemaphoreCreateBinary();
14    stateMachineSemaphore = xSemaphoreCreateBinary();
15    dataLoggingSemaphore = xSemaphoreCreateBinary();
16    i2cSemaphore = xSemaphoreCreateBinary();
17    task_watchdog = xEventGroupCreate();
18 }

```

16.2 Test wifi configuration task

The following code is used to test if the device can connect to the server

```

1 void test_wifi_config(void *param){
2     while(true){
3         if(xSemaphoreTake(testConnectionSemaphore, 10000 / portTICK_RATE_MS)){
4             blink(led_green_color, ONE_SECOND_BLINK);
5             deleteConfigCalibTask();
6             vTaskDelay(1000/portTICK_PERIOD_MS);
7             connection_stat_check=1;
8             connectSTA( (char *) ap_ssid_buff , (char *) ap_psw_buff );
9         }
10    }
11 }

```

16.3 Main timer task

The following code is used to synchronize the following thread.

1. ledTask
2. overvoltageTask
3. stateMachineTask
4. dataLoggingTask

```

1 void mainTimerTask( void * param){
2     static uint8_t ledTaskCount=0, overVoltageTaskCount=0, stateMachineTaskCount=0;
3     static uint16_t dataLoggingTaskCount=0;
4     TickType_t xLastWakeTime;
5     const TickType_t xFrequency = 1;
6     xLastWakeTime = xTaskGetTickCount();
7     while(1){
8         if(++ledTaskCount>10){
9             ledTaskCount=0;
10            xSemaphoreGive(ledTaskSemaphore);
11        }
12        if(++overVoltageTaskCount>30){
13            overVoltageTaskCount=0;
14            xSemaphoreGive(overVoltageProtectionSemaphore);
15        }
16        if(++stateMachineTaskCount>100){
17            stateMachineTaskCount=0;

```



```

18     xSemaphoreGive(stateMachineSemaphore);
19 }
20 if(++dataLoggingTaskCount>1000){
21     dataLoggingTaskCount=0;
22     xSemaphoreGive(dataLoggingSemaphore);
23 }
24 vTaskDelayUntil( &xLastWakeTime, xFrequency );
25 }
26 }

```

16.4 On connection task

The following code is used to send data to the server on connecting with AP

```

1 void OnConnected(void *para){
2     while(true){
3         if(xSemaphoreTake(logdataSemaphore, 10000 / portTICK_RATE_MS)){
4             if(connection_stat_check){
5                 test_server_connection();
6             }
7             else{
8                 if(https_action==1){
9                     bidirectional_communication_commands(&https_request_buffer_body);
10                }
11                else{
12                    send_DATA_to_server();
13                }
14            }
15            ESP_LOGI("inside on connected", "Done!");
16        }
17    }
18 }

```

16.5 Overvoltage protection

The following code generates ticks every 300millisecond. It is used for battery voltage monitoring.

```

1 void overvoltageProtectionTask( void * param){
2     while(1){
3         xSemaphoreTake(overVoltageProtectionSemaphore,portMAX_DELAY);
4         if(!test_error((ERR_NOT_CALIB | ERR_NOT_CONFIG))){
5             if(ledBlinkOn){
6                 if(stopBatteryOverVoltageProtectionOnCalibration==0){
7                     check_battery_voltage();
8                 }
9             }
10        }
11        xEventGroupSetBits(task_watchdog, overvoltageProtectionTaskEvent);
12    }
13 }

```

16.6 LED blink

The following code is used for blink led based on the state of the device.

```

1 void ledBlinkTask( void * param){
2     while(1){
3         xSemaphoreTake(ledTaskSemaphore,portMAX_DELAY);
4         if(ledBlinkOn){
5             led_blink();
6         }
7     }
8 }

```

16.7 State machine task

The following code generates ticks every 1second. It is used for executing the state change algorithm.

```

1 void stateMachineTask( void * param){
2     while(1){
3         xSemaphoreTake(stateMachineSemaphore,portMAX_DELAY);
4
5         if(test_common_flag(CF_RESET_DEV)){
6             clear_common_flag(CF_CLR_ALL);
7             xSemaphoreGive(serverReset);
8         }
9
10        server_connect_retry_cnt++;
11
12        ESP_LOGI(TAG,"Heap value %d",xPortGetFreeHeapSize());
13        if(stop_algorithm==0){
14            if(!test_error((ERR_NOT_CALIB | ERR_NOT_CONFIG)){
15                ESP_LOGI(TAG,"iCON LV 8.00 algorithm");
16                turtle_algorithm();
17                watchdog_timer_count=0;
18            }
19            else{
20                watchdog_timer_count=0;
21                if(test_error(ERR_NOT_CALIB)){
22                    ESP_LOGI(TAG,"Device not calibrated");
23                }
24
25                if(test_error(ERR_NOT_CONFIG)){
26                    ESP_LOGI(TAG,"Device not configured");
27                }
28            }
29        }
30        #if ICUBE==0
31            #warning "Inverter failure buzzer enabled"
32            if(test_system_flag(SF_INV_FAIL)){ toggle_buzzer(); };
33        #endif
34        xEventGroupSetBits(task_watchdog,stateMachineTaskEvent);
35    }
36 }

```

16.8 data logging task

The following code generates ticks every 10 second. It handles server communication and time sync.

```

1 void dataLoggingTask( void * param){
2     while(1){
3         xSemaphoreTake(dataLoggingSemaphore,portMAX_DELAY);
4         xEventGroupSetBits(task_watchdog,dataLoggingTaskEvent);
5
6         if(watchdog_timer_stop==0){
7             ESP_LOGI(TAG,"Checking watchdogTimer");
8             if(++watchdog_timer_count>2){
9                 ESP_LOGI(TAG,"Watchdog Timer Expired");
10                switch_state_4_5();
11                esp_restart();
12            }
13        }
14
15        static uint8_t getInfoTime=0;
16        if(++getInfoTime>30){
17            getInfoTime=0;
18            https_req_buff_write(&https_request_buffer_body,get_information);
19        }
20
21        if((stop_algorithm==0)&&(stop_lptask==0)){
22            ESP_LOGI(TAG,"10 second tick");
23            if(!test_error((ERR_NOT_CALIB | ERR_NOT_CONFIG)){
24                least_priority_operations();
25            }
26        }
27
28        if(connection_stat_check){
29            if(++connection_stat_check_cnt>6){
30                connection_stat_check=0;
31                stop_algorithm=0;

```

```

32     watchdog_timer_stop=0;
33 }
34 }
35 if(stop_algorithm==1){
36     if(++configuration_timeout_cnt>=30){
37         configuration_timeout_cnt=0;
38         stop_algorithm=0;
39         watchdog_timer_stop=0;
40         deleteConfigCalibTask();
41     }
42 }
43
44
45 }
46 }

```

16.9 Time sync

The following code checks if the time is synced and connects to AP if it isn't

```

1 void TrySyncTime(void){
2     if(rtc_sync==0){
3         ESP_LOGI(TAG,"Trying to sync time");
4         esp_wifi_stop();
5         connectSTA((char *)ap_ssid_buff,(char *)ap_psw_buff);
6         least_priority_operations_parameters.status=waiting_for_time_synced;
7     }
8 }

```

16.10 Wait for time sync

Syncing time takes time and the following function waits for the device to connect to the internet.

```

1 void TimeoutSyncTime(void){
2     if(rtc_sync==0){
3         if(++wait_time_sync>2){
4             wait_time_sync=0;
5             least_priority_operations_parameters.status=time_not_synced;
6         }
7     }
8     else{
9         wait_time_sync=0;
10        least_priority_operations_parameters.status=lpos_data_not_available;
11    }
12 }

```

16.11 Check data availability

The following code is used to check if there is any request pending to the server

```

1 void checkDataAvailable(void){
2     if(https_req_buff_available(&https_request_buffer_body)!=0){
3         ESP_LOGI(TAG,"Request is %d",https_req_buff_peek(&https_request_buffer_body));
4         https_action=1;
5         least_priority_operations_parameters.status=lpos_data_available;
6     }
7     else{
8         if(turtle_log_available()>packet_length){
9             ESP_LOGI(TAG,"Data available in EEPROM");
10            least_priority_operations_parameters.status=lpos_data_available;
11        }
12        else{
13            if(fw_update){
14                ESP_LOGI(TAG,"OTA update");
15                fw_update=0;
16                perform_ota_task=1;
17            }
18        }
19    }
20 }

```

16.12 Check connection status

The following code checks if the device is connected to AP and if not it initiates connection to access point

```

1 void connectingToAP(void){
2     if(wifi_connection_status){
3         ESP_LOGI(TAG,"Already connected to AP");
4         least_priority_operations_parameters.status=lpos_link_connected;
5     }
6     else{
7         ESP_LOGI(TAG,"Not connected to AP");
8         least_priority_operations_parameters.status=lpos_link_not_connected;
9     }
10 }

```

16.13 Connecting to AP

If the device is not connected to any access point the following code tries to connect to the AP. First it tries to connect to the primary access point if it isn't accessible then it tries to connect to secondary access point

```

1 void trySwitchingAP(void){
2     if(access_point==primary){
3         ESP_LOGI(TAG,"Connecting to primary access point");
4         connectSTA((char *)ap_ssid_buff,(char *)ap_psw_buff);
5     }
6     else{
7         ESP_LOGI(TAG,"Connecting to secondary access point");
8         connectSTA("E24by7","Energy24by7");
9     }
10     least_priority_operations_parameters.status=lpos_link_connected;
11     least_priority_operations_parameters.wait_time = 0;
12 }

```

16.14 Waiting for connection with AP

Connecting to AP might take time and hence the device waits for some time to check if it is possible to connect with the AP. IF the connection is established it transfer control to the function which handles server interactions

```

1 void waitConnectingToAP(void){
2     if(wifi_connection_status==0){
3         if(++least_priority_operations_parameters.wait_time>1){
4             esp_wifi_stop();
5             if(access_point==primary){
6                 access_point=secondary;
7             }else{
8                 access_point=primary;
9             }
10            least_priority_operations_parameters.wait_time=0;
11            least_priority_operations_parameters.status=lpos_link_not_connected;
12        }
13        else{
14            if(access_point==primary){
15                ESP_LOGI(TAG,"Wait to connect with primary access point");
16            }
17            else{
18                ESP_LOGI(TAG,"Wait to connect with secondary access point");
19            }
20        }
21    }
22    else{
23        if(access_point==primary){
24            ESP_LOGI(TAG,"Connected to primary access point");
25        }
26        else{
27            ESP_LOGI(TAG,"Connected to secondary access point");
28        }
29        if(perform_ota_task){

```

```

30     stop_lptask=1;
31     fw_update=0;
32     xSemaphoreGive(otaSemaphore);
33 }
34 else{
35     xSemaphoreGive(logdataSemaphore);
36 }
37 least_priority_operations_parameters.wait_time=0;
38 least_priority_operations_parameters.status=lpos_waiting_for_response;
39 }
40 }

```

16.15 Server response timeout

The following code is used to wait for response from the server

```

1 void waitingForServerResponse(void){
2     if(++least_priority_operations_parameters.wait_time>12){
3         least_priority_operations_parameters.status=lpos_communication_failed;
4         least_priority_operations_parameters.wait_time=0;
5         ESP_LOGI(TAG,"Data logging failed");
6     }
7     else{
8         ESP_LOGI(TAG,"Waiting for response from server");
9     }
10 }

```

17 Update tail

The following code updates the tail of queue/EEPROM ring buffer on receiving response from the server

```

1 void onResponseFromServer(void){
2     ESP_LOGI(TAG,"Received response from server");
3     least_priority_operations_parameters.wait_time=0;
4     if(https_action==1){
5         https_req_buff_flush(&https_request_buffer_body);
6         https_action=0;
7     }
8     else{
9         ESP_LOGI(TAG,"EEPROM tail updated");
10        turtle_log_update_tail();
11    }
12    least_priority_operations_parameters.status=lpos_data_not_available;
13    esp_wifi_stop();
14 }

```

18 Disconnect from AP

IF there is no respons from the server the following code disconnects from the AP

```

1 void onCommunicationFailed(void){
2     least_priority_operations_parameters.status=lpos_data_not_available;
3     esp_wifi_stop();
4 }

```

19 Least priority operations

The following code is used to handle the even for low priority operation

```

1 void least_priority_operations(void){
2     switch(least_priority_operations_parameters.status){
3     case time_not_synced:
4         TrySyncTime();
5         break;
6     case waiting_for_time_synced:
7         TimeoutSyncTime();

```

```

8         break;
9     case lpos_data_not_available:
10        checkDataAvailable();
11        break;
12     case lpos_data_available:
13        connectingToAP();
14        break;
15     case lpos_link_not_connected:
16        trySwitchingAP();
17        break;
18     case lpos_link_connected:
19        waitConnectingToAP();
20        break;
21     case lpos_waiting_for_response:
22        waitingForServerResponse();
23        break;
24     case lpos_response_received:
25        onResponseFromServer();
26        break;
27     case lpos_communication_failed:
28        onCommunicationFailed();
29        break;
30 }
31 }

```

19.1 Perform OTA

The following code is used to perform OTA of the new firmware

```

1 void perform_ota(void *params){
2     extern uint8_t https_password[SERV_IP_LEN];
3     while (true){
4         char url[2048];
5         read_spiffs(url,ota_update);
6         xSemaphoreTake(otaSemaphore, portMAX_DELAY);
7         ESP_LOGI(TAG, "Invoking OTA");
8         esp_http_client_config_t clientConfig = {
9             .url = url,
10            .event_handler = client_event_handler,
11            .cert_pem = (char *)server_cert_pem_start,
12            .username = (char *)dev_id,
13            .password = (char *) https_password,
14            .auth_type = HTTP_AUTH_TYPE_BASIC,
15        };
16        if(esp_https_ota(&clientConfig) == ESP_OK){
17            printf("OTA flash successfull");
18            printf("restarting in 5 seconds\n");
19            stop_algorithm=1;
20            vTaskDelay(pdMS_TO_TICKS(5000));
21            switch_state_4_5();
22            esp_restart();
23        }
24        stop_lptask=0;
25        fw_update=1;
26        ESP_LOGE(TAG,"Failed to update firmware");
27    }
28 }

```

19.2 Perform factory reset

The following code is used to perform factory reset of the controller. After factory reset the code available in the controller will be the one programmed in factory.

```

1 void perform_factory_reset(void *params){
2     while(1){
3         xSemaphoreTake(factoryReset, portMAX_DELAY);
4         extern void fcn_factory_reset(void);
5         fcn_factory_reset();
6     }
7 }

```

19.3 Perform system reset

The following code is used to initiate a reset from the server

```

1 void perform_system_reset(void *params){
2     while(1){
3         xSemaphoreTake(serverReset,portMAX_DELAY);
4         i2c_eeprom_write_buff(0xFE7F,(uint8_t *) &common_flag,2);
5         printf("restarting in 5 seconds\n");
6         stop_algorithm=1;
7         vTaskDelay(pdMS_TO_TICKS(5000));
8         switch_state_4_5();
9         esp_restart();
10    }
11 }
```

19.4 Custom watchdog task

The following code monitors the health of each task and resets the controller if at least one task gets stuck.

```

1 void watchdogTask(void){
2     while(1){
3
4         if(watchdog_timer_stop==0){
5             uint32_t result = xEventGroupWaitBits(task_watchdog, allEvents, pdTRUE, pdTRUE,
6             10000/portTICK_RATE_MS);
7             if((result&allEvents)==allEvents){
8                 ESP_LOGI("Inside watchdogTask","System healthy");
9             }
10            else{
11
12                ESP_LOGI(TAG,"Watchdog Timer Expired");
13                switch_state_4_5();
14
15                if(!(result&overvoltageProtectionTaskEvent)){
16                    ESP_LOGI("Inside watchdogTask","overvoltage protection task not responding");
17                }
18
19                if(!(result&stateMachineTaskEvent)){
20                    ESP_LOGI("Inside watchdogTask","state machine task not responding");
21                }
22
23                if(!(result&dataLoggingTaskEvent)){
24                    ESP_LOGI("Inside watchdogTask","data logging task not responding");
25                }
26
27                esp_restart();
28            }
29        }
30        vTaskDelay(1000/portTICK_RATE_MS);
31    }
```

19.5 Task creation

The following code is used to create the above-mentioned tasks

```

1 xTaskCreate(&buttonPressDetection_routine,"Button press",4096,NULL,5,NULL);
2 xTaskCreate(&buttonPressTask,"Button press confirmed",4096,NULL,5,NULL);
3 xTaskCreate(&dataLoggingTask,"dataLogging",4096,NULL,1,NULL);
4 xTaskCreate(&ledBlinkTask,"ledBlink",1024,NULL,3,NULL);
5 xTaskCreate(&overvoltageProtectionTask,"overvoltageProtection",4096,NULL,4,NULL);
6 xTaskCreate(&stateMachineTask,"stateMachineAlgorithm",4096,NULL,2,NULL);
7 xSemaphoreGive(i2cSemaphore);
8 xTaskCreate(&mainTimerTask,"mainTimer",1024,NULL,5,NULL);
9
10 if(test_error(ERR_NO_MEMORY)){
11     ESP_LOGI(TAG,"EEPROM not detected");
12     blink(led_red_color,LED_OFF);
13 }
```

```
14  else{
15      least_priority_operations_parameters.status=time_not_synced;
16      https_req_buff_write(&https_request_buffer_body,update_configuration);
17      watchdog_timer_stop=0;
18      xTaskCreate(&OnConnected, "data logging", 5*4096, NULL, 5, NULL);
19      xTaskCreate(&test_wifi_config,"Test configuration",4096,NULL,5,NULL);
20      xTaskCreate(perform_ota, "perform_ota", 1024 * 8, NULL, 2, NULL);
21      xTaskCreate(perform_factory_reset, "perform_fac_reset", 4096, NULL, 2, NULL);
22      xTaskCreate(perform_system_reset, "perform_system_reset", 4096 , NULL, 2, NULL);
23  }
```


20 Main algorithm

20.1 Turtle algorithm

Note:- The state change algorithm operates on the values obtained from sensors connected to the MCU. Parameters such as Solar voltage, battery voltage, battery charging current, battery discharging current, inverter voltage, AC mains current, temperature are read using ADCs and PAC device. The variables in which these values are stored are declared as a structure. The structure also holds variables into which date, time and important flags regarding the algorithm operations are stored. They are updated every 30 seconds during normal operation (Debug log). Each set of data is referred to as records. During a state change 10 records obtained every 1 second is sent to the server.(State change log) A record is sent to the server, after every one hour.(Summary log)

1. Data are sent to the server along with time stamp. The time is read from the RTC module in the controller. Hence, when the control enters the turtle algorithm the date and time from RTC module is read.
2. The state change algorithm operates on the values obtained from different sensors connected to the MCU. The sensor values are obtained.
3. A pin on the MCU is designated to sense the availability of utility. The pin is high if the utility is available and is low if there is a power failure. The pin is polled to get the status of the utility.
4. Solar energy, battery energy are calculated over a period of one hour for the summary log.
5. In order to calculate the debug log, a variable is used. It is incremented every one second and once it reaches 30, command to log data into the EEPROM is given and the variable is reset to zero.
6. After receiving a command to log data into the EEPROM as mentioned above, RTC module is checked for error. If there is no error in RTC module, a check is done to ensure whether debug log or state change log or summary log is to be logged into the eeprom. If there is an error in RTC the logging is not done.
7. Though there is an RTC available, operations like solar bucketing, load bucketing, evening force trip exit etc., are done based on sun up and sun down. Hence, two variables are used to track time based on sun up and sun down. One variable is used to hold seconds and the other holds the hour. After counting till 3600 the seconds variable resets to zero and the hour variable is incremented. The maximum value hour variable can count is 23, above which it stays in 23. The hour variable will be reset to zero during sun up.
8. A provision to halt solar charging is provided if the solar current reaches beyond 40A. 30 minutes after the solar charging is stopped, the device reconnects to check if solar current is less than 40A. During this 30 minutes the device is set in state 5.
9. The device moves into state change algorithm after calculating solar energy received, bucketing solar and load. Solar bucketing is done only for 12 hours.
10. Refer section ?? for state change algorithm

20.2 Sun up routine

1. The variable `is_day_flag` is used to indicate whether it is day time or night time. If the flag indicate night time and the device sees a solar voltage for 5 seconds, the flag is changed as day time.

```

1  /*
2  * Condition to check whether solar is available. 1) Checks if Sol_V>Bat_V 2)
  Checks the flag set for night indication.
3  * This routine is entered in the morning when solar comes in and the system is
  still in night mode
4  */
5  if(algo_param.cur_sol_v > SOL_HIGH_THR && (is_day_flag == NOW_NIGHT_TIME) && ((
  readyforsunup==0x01)|| (sunup_conf_enable==0x00)))
6  {
7  /*
```

```

8      * After detecting solar voltage, the algorithm waits for 5 seconds to ensure
      sun up.
9      */
10     if(sunup_count<4)
11     {
12         sunup_count++;
13     }
14     else
15     {
16         sunup_count=0;

```

Note:- The default value of the flag upon reset will be night. When the device is first installed and sees the solar voltage for the first time, is_solar_sensed flag is set. Only after this flag is set, state change algorithm works. Refer ?? for state change algorithm

2. Few variables like total solar received, solar time, solar seconds, solar equalization count, force trip count, need to be reset at the time of sun up for proper operation of the algorithm.
3. Load energy and solar energy calculated has to be bucketed at sun up and reset.
4. If it is the day of installation, force trip should not take place, else the force trip count is reset.
5. If the dynamic entry and exit day counts are past the specific days the variables to indicate them are enabled.

```

1      sunup_count=0;
2      /*
3      * Indicates solar panel as been connected to the device. If solar sense is 0
      it means solar panel is not connected to the device
4      * It is indicated using 1 sec yellow led blink
5      */
6      sun_down_error = 0;
7      is_sol_sensed = 1;
8      clear_system_flag(SF_SOL_DIS);
9      /*
10     * This variable holds the total solar wattage received for the day.
11     * It is updated every second.
12     */
13     Today_Recieved_Solar_Energy=0;
14     /*
15     * These two variables are used to run a clock based on sun up and sun down.
16     * The variable Solar_Time is used to track hour and the variable
      solar_time_seconds is used to track seconds
17     * The hour variable will reset everyday morning when the solar comes in for
      the first time.
18     * The seconds variable increments every one second, and when it reaches 3600
      it is reset after incrementing hour by 1.
19     * These variables are extensively used in solar bucketing, load bucketing,
      server set state, exiting FT at 4:30PM, evening battery charge back and dynamic
      entry
20     */
21     Solar_Time=0;
22     solar_time_seconds=0;
23     /*
24     * Feature to enter force trip if battery is 90% full
25     */
26     enable_early_ft_entry = 1;
27     /*
28     * Dynamic entry depends on solar energy received for the past three days and
      load energy observed for the last seven days.
29     * If the user has a low load consumption during morning hours, the device
      cycles the solar energy. This might reflect as low solar energy received.
30     * The variable is used to track the time taken for equalization and enter into
      dynamic force trip if equalization happened for more than 1 hour.
31     */
32     solar_equ_cnt=0;
33
34     /*
35     * The purpose of dynamic force trip entry is to utilize solar energy to
      optimal level possible.
36     * After a dynamic entry is completed, the battery should not be charged using
      mains. This variables blocks the charging of battery from mains after a dynamic
      entry.

```

```

37      */
38      dynamic_ft_complete=0;
39
40      /*
41      * The algorithm buckets load energy only after a proper sun up and sundown has
42      * passed.
43      * The time of installation of the device is unknown. Upon starting the solar
44      * hour and solar seconds are zero.
45      * On day 2, the solar time is reset on sun up, and sun down time is noted at
46      * sundown.
47      * Only after this correction, the algorithm buckets the load.
48      */
49      if((num_days>2)&&!(dev_state_flag&Bit(6)))
50      {
51          Load_Energy_Bucketing();
52      }
53
54      /*
55      * After bucketing the load, the variables used for bucketing are reset.
56      */
57      reset_load_power_bucket();
58      reset_solar_power_bucket();
59
60      /*
61      * This variable is used to track the number of days passed after installation.
62      * The variable will get reset once battery is removed/after a reset either from
63      * WDT or server.
64      */
65      num_days++;
66
67      /*
68      * Enable sunup confidence check
69      */
70      if(num_days>2)
71      {
72          sunup_conf_enable = 1;
73      }
74      else
75      {
76          sunup_conf_enable = 0;
77      }
78
79      readyforsunup = 0;
80
81      /*
82      * On first day of installation the device doesn't force trip. It is done by
83      * setting the force trip count allowed for the day to maximum value.
84      * During regular days the force trip count is set to zero in the morning when
85      * solar entry happens
86      */
87      if(((num_days<=1)&&!(dev_state_flag&Bit(2))))||((test_system_flag(
88      SF_INV_FAIL)))
89      {
90          frc_trip_count = max_frc_per_day;
91      }
92      else
93      {
94          frc_trip_count = 0;
95      }
96
97      /*
98      * Dynamic force trip exit feature is enabled only on the 5th day after
99      * installation of device.
100     * If a useable SoC calculation failure occurs, the feature gets delayed by a
101     * day.
102     * This feature can be blocked from the server if required.
103     */
104     if((num_days>=ft_exit_start_day)&&!(dev_state_flag&Bit(4)))
105     {
106         Dynamic_FT_Exit_Enable=1;
107     }
108
109     /*

```

```
100      * Dynamic force trip entry feature is enabled only on the 9th day after
101      installation of device.
102      * If a useable SoC calculation failure occurs, the feature gets delayed by a
103      day.
104      * This feature can be blocked from the server if required.
105      */
106      if((num_days>=ft_entry_start_day)&&!(dev_state_flag&Bit(3)))
107      {
108          Dynamic_FT_Entry_Enable=1;
109          dynamic_ft_complete=0;
110          is_time_to_dynamic_ft_entry=0;
111      }
112
113      /*
114      * Reset Dynamic Exit and Entry Flag
115      */
116      Dynamic_FT_Exit=0;
117      Dynamic_FT_Enter=0;
118      dy_exit=0;
```

6. If the previous day was absorption day, and if the mains failed for more than 3 hours, the mains failed during absorption flag is set.

```

1  /* Mains Fail Check during Absorption.
2  * On absorption days, mains fail during night time is detected.
3  * If the mains fail persisted for more than three hours SF_ABS_MF is set.
4  * The duration of mains fail can be changed as required either from server or
   using APP.
5  */
6  if(no_inh_flag&&(is_day_flag==NOW_NIGHT_TIME))
7  {
8  if((algo_param.cur_dis_i>MIN_DC_LOAD_I)&&(test_system_flag(SF_ABS_MF)==0))
9  {
10 ++mains_fail_counter;
11 if(mains_fail_counter>abs_mains_fail_tolerance)
12 {
13 post_system_flag(SF_ABS_MF);
14 }
15 }
16 }

```

7. If the absorption flag is set at sun up, the moving average of the battery charging current for the past 3 minutes is calculated. If the current is less than one-fiftieth of the Rated Ah, then the absorption is considered to be successful.

```

1  /*
2  * Case 1: If there happens a mains fail on the previous night during
   absorption, SF_ABS_MF flag is set.
3  * If SF_ABS_MF flag is set, the absorption current is checked.
4  */
5  if(test_system_flag(SF_ABS_MF))
6  {
7      for(int i=0;i<180;i++)
8      {
9          MA_absorption_current+=absorption_current[i];
10     }
11
12     MA_absorption_current=MA_absorption_current/180;
13
14     /*
15     * Case 1a: If the SF_ABS_MF flag is set the battery charging tail current is
   calculated. If it is less than 3A the absorption failed flag is set. Absorption
   current is set as 0.02 of rated battery AH.
16     * Absorption current value is a moving average value of charging current over
   a period of 3 minutes.
17     */
18     if(MA_absorption_current<(Rated_Bat_AH/50.0))
19     {
20         post_system_flag(SF_ABS_COMPLETED);
21         clear_system_flag(SF_ABS_MF);
22         clear_system_flag(SF_ABS_NEEDED);
23         if(enable_cal_useable_soc)
24         {
25             soc_reset_flag=0;
26         }
27     }

```

8. If the value is greater than one-fiftieth of the Rated Ah, then the absorption is considered to be a failure.

```

1  /*
2  * Case 1b: Else the SF_ABS_MF flag is reset and SF_ABS_COMPLETED flag is set
   to indicate the completion of absorption.
3  */
4  else
5  {
6      post_system_flag(SF_ABS_NEEDED);
7      clear_system_flag(SF_ABS_MF);
8      clear_system_flag(SF_ABS_COMPLETED);
9  }
10 /*
   * If the absorption was initiated for useable SoC calculation and if the
   absorption has failed, then useable SoC calculation is re-attempted the next day
   .

```

```

11      * If already re-attempted, and it has failed again, the useable SoC
12      calculation is considered a failure.
13      */
14      if(enable_cal_useable_soc)
15      {
16          if(re_cal_useable_soc_count>1)
17          {
18              enable_cal_useable_soc=0;
19              need_eq_after_abs=0;
20              post_system_flag(SF_USEABLE_SOC_FAILED);
21              frc_trip_count = 0;
22              re_cal_useable_soc_count=0;
23              //useable SoC Calculation failed
24          }
25          else
26          {
27              ft_exit_start_day++;
28              ft_entry_start_day++;
29              post_system_flag(SF_USEABLE_SOC_CAL);
30          }
31      }
32  }
33

```

9. If the mains failed during absorption flag is not set, the absorption is considered to be a success.

```

1      /*
2      * Case 2: If the flag is not set SF_ABS_COMPLETED flag is set to indicate
3      completion of absorption.
4      */
5      else
6      {
7          if(test_system_flag(SF_ABS_NEEDED)==0)
8          {
9              if(enable_cal_useable_soc)
10             {
11                 soc_reset_flag=0;
12             }
13             post_system_flag(SF_ABS_COMPLETED);
14             clear_system_flag(SF_ABS_MF);
15             clear_system_flag(SF_ABS_NEEDED);
16         }
17     }
18

```

10. If the battery is of tubular type, periodic equalization of the battery is necessary. Before equalization, the previous night the battery has to undergo absorption process.

```

1  /* Equalization is done only if the battery is a tubular battery.
2  * Night before equalization absorption is done.
3  * Equalization is done only if absorption is complete.
4  * Periodic equalization interval can be set between 30 to 120days.
5  */
6  if(battery_type==TUBULAR)
7  {
8      bat_sol_equ_intr_cnt+=1;
9
10
11     /*
12     * It tracks the equalization day and initiates absorption the day before
13     equalization.
14     */
15     if(bat_sol_equ_intr_cnt==bat_sol_equ_intr)
16     {
17         post_system_flag(SF_ABS_NEEDED);
18     }
19
20     /*
21     * This code initiates equalization.
22     */
23     if((bat_sol_equ_intr != 0) && \
24         (bat_sol_equ_dur != 0 ) && \
25         (bat_sol_equ_intr_cnt > bat_sol_equ_intr) || test_system_flag(
26 SF_EQ_NEEDED))
27     {
28         if(test_system_flag(SF_EQ_NEEDED))
29         {
30             clear_system_flag(SF_EQ_NEEDED);
31             equalization_repeat_counter++;
32
33             /*
34             * If equalization repeat counter reaches 2 set equalization failed flag
35             and clear equalization needed flag.
36             */
37             if(equalization_repeat_counter>1)
38             {
39                 clear_system_flag(SF_EQ_NEEDED);
40                 post_system_flag(SF_EQ_FAILED);
41                 is_time_to_equ=0;
42             }
43             else
44             {
45                 is_time_to_equ=1;
46             }
47         }
48     }
49

```

11. The equalization process is carried out only if the absorption process is a success.

```

1  else
2  {
3      /*
4      * Incase absorption initiated on the previous day is not complete,
5      equalization is not performed.
6      */
7      if(test_system_flag(SF_ABS_NEEDED))
8      {
9          /*Postpone the equalization*/
10         is_time_to_equ = 0;
11         test_system_flag(SF_EQ_NEEDED);
12     }
13     else
14     {
15         is_time_to_equ = 1;
16     }
17 }

```

12. If the battery is of non-tubular type, instead of equalization, solar cycling takes place.

```

1  if(test_system_flag(SF_ABS_NEEDED))
2  {

```

```
3  /*Postpone the equalization*/  
4  is_time_to_equ = 0;  
5  test_system_flag(SF_EQ_NEEDED);  
6  }  
7  else  
8  {  
9  is_time_to_equ = 1;  
10 }  
11 }  
12 }
```

Note:- Before useable SoC calculation, the battery has to be absorbed (if non-tubular type) or has to be absorbed and equalized (if tubular type). The failure in absorption/equalization during that time will make the device re-attempt the useable SoC calculation. If the failure happens even after a re-attempt, the useable SoC calculation is deemed failure.

13. To ensure battery's health, the battery is subjected to absorption once a week.

```

1  /*The absorption feature is available for all battery types*/
2  /*Absorption can be done every 4 to 7days*/
3  if(++bat_abs_intr_cnt>bat_abs_intr)
4  {
5      bat_abs_intr_cnt=0;
6      post_system_flag(SF_ABS_NEEDED);
7  }

```

14. During useable SoC calculation, the force trip has to be prevented till the device initiates the start.

```

1  if(enable_cal_useable_soc==1){
2      frc_trip_count = max_frc_per_day;
3  }

```

20.3 Sundown/Solar disconnection detection

1. This section of the algorithm is used to detect sundown as well as a solar disconnection. The solar voltage is read via an ADC pin. A moving average method is used to read the solar voltage. Hence the ADC value falls 10 seconds after the solar switch is opened.
2. During a natural sun-down the solar voltage falls from 11V to 5V only after 3-5 minutes, whereas if there is a solar disconnection the voltage read by the ADC pin will be zero after 10 seconds. This logic is used to differentiate a natural sun down from a solar disconnection for the first two days after installation.

```

1  /*
2   * This code detects whether the solar panel is disconnected or a sundown occurs.
3   * During a solar panel disconnect, the solar voltage falls to zero in no time,
4   * where as during a sundown the voltage takes minutes to fall to zero.
5   * This logic is used in determining the condition that occurred.
6   */
7  else if((algo_param.cur_sol_v < SOL_LOW_H_THR) && (algo_param.cur_sol_v >
8      SOL_LOW_L_THR) && (is_day_flag == NOW_DAY_TIME))
9  {
10     /*
11      * If the voltage goes below a certain specified value which is provided to
12      * detect sundown, the algorithm waits for 30 seconds to ensure it.
13      */
14     if(sundown_count<30)
15     {
16         sundown_count++;
17     }
18     else
19     {
20         /*
21          * After two days from installation, after the voltage falls, check if the
22          * solar hour is greater than the previous day's sundown hour.
23          * If the solar hour is greater, then it is considered as a sundown.
24          */
25         if(num_days>2)
26         {
27             if(Solar_Time>=(Previous_Day_Sundown-1))
28             {
29                 sun_down_routine();
30             }
31             else
32             {
33                 sun_down_error=1;
34                 sol_dis_count=0;
35                 is_sol_sensed=0;
36                 reset_parameters();
37                 sundown_count=0;
38             }
39         }
40         else
41         {
42             sun_down_routine();
43             sundown_count=0;
44         }
45     }
46 }

```

```
41     }  
42 }  
43 }
```

3. After day 2 of installation, the previous day sun-down time is noted, and is used to check if it is a natural sun-down or a solar disconnection.
4. If a solar disconnection happens, all the dynamic variables are reset. The variables like useable SoC, force trip voltage exit and device status flag are stored in the EEPROM and are re-used.
5. If a natural sun-down is detected, sundown routine is executed.

```
1  /*  
2  * If the solar voltage drops less than a set voltage, and if it stays there  
3  * for 20seconds it is considered as solar disconnect.  
4  */  
5  if((algo_param.cur_sol_v < SOL_LOW_L_THR) && (is_day_flag == NOW_DAY_TIME))  
6  {  
7      if(sol_dis_count<20)  
8      {  
9          sol_dis_count++;  
10     }  
11     else  
12     {  
13         sun_down_error=1;  
14         sundown_count=0;  
15         sol_dis_count=0;  
16         is_sol_sensed=0;  
17         reset_parameters();  
18         post_system_flag(SF_SOL_DIS);  
19     }  
20 }
```

20.4 Sundown Routine

1. Once a natural sundown is detected, the day flag is set as night.

```

1 void
2 sun_down_routine()
3 {
4     /*The variable indicates the day or night time of the day*/
5     is_day_flag = NOW_NIGHT_TIME;

```

2. The solar time at which the sundown occurs is noted. The morning hour is considered to be half of the sundown time.

```

1 sundown_count=0;
2 solar_cycling=0;
3 mains_sol_chg_flag=0;
4 if(test_system_flag(SF_SMC)){
5     clear_system_flag(SF_SMC);
6 }
7 mains_chg_flag=0;
8 Sundown_Time=Solar_Time;
9 Morning_Sunhour=Sundown_Time/2;

```

3. If the number of days after installation is greater than 1, these values are considered to be the previous day's sundown time and morning sunhour time.

```

1 if(num_days>1){
2     Previous_Day_Sundown=Sundown_Time;
3     Previous_Day_Morning_Sunhour=Morning_Sunhour;
4 }

```

4. Solar energy received is bucketed. Total energy received for the day is reset.

```

1 /*Total solar received for the day is reset*/
2 Today_Recieved_Solar_Energy=0;
3 /*Solar value collected during the day is bucketed into morning and evening slots*/
4 if((num_days>=2)&&!(dev_state_flag&Bit(5))){
5     Solar_Energy_Bucketing();
6 }

```

5. Useable SoC of the battery is calculated every 6 months to determine the aging of the battery. If useable SoC calculation flag is set, the device initiates an absorption process by enabling absorption needed flag (followed by equalization if tubular battery).

```

1     /*Variables assigned to stop the force trip are reset*/
2     no_frc_trip_flag = 0;
3     frc_trip_count = 0;
4
5     /*This condition happens during the first day of installation or when a
6     useable SOC mismatch happens*/
7     /*When the SF_USEABLE_SOC_CAL flag is set. The device is pushed into
8     absorption. Equalization is done on the next day*/
9     /*Once absorption and equalization is complete the battery is evacuated to
10    estimate the useable SOC*/
11     if((++useable_soc_recal_cnt>useable_soc_recal_dur)||test_common_flag(
12     CF_RPT_INIT)))
13     {
14         post_system_flag(SF_USEABLE_SOC_CAL);
15         useable_soc_recal_cnt=0;
16
17         if(test_common_flag(CF_RPT_INIT))
18         {
19             clear_common_flag(CF_RPT_INIT);
20             WDT_init_stop=0;
21             i2c_eeeprom_write_buff(0xFE7F,(uint8_t *) &common_flag,2);
22             WDT_init_stop=1;
23         }
24     }
25
26     if(test_system_flag(SF_USEABLE_SOC_CAL)&&!(test_common_flag(CF_BYSOC)))
27     {
28         clear_system_flag(SF_USEABLE_SOC_CAL);

```

```
25         enable_cal_useable_soc=1;
26         post_system_flag(SF_ABS_NEEDED);
27
28         if(battery_type==TUBULAR)
29         {
30             need_eq_after_abs=1;
31         }
32         else
33         {
34             need_eq_after_abs=0;
35         }
36     }
```

6. If absorption flag is set by the device during sunup routine or due to useable SoC calculation, the device enters absorption.

```

1      if(test_system_flag(SF_ABS_NEEDED)|| (test_common_flag(CF_SET_ABS)) || (
2          test_common_flag(CF_SET_EQU)))
3      {
4          Dynamic_FT_Entry_Enable=0;
5          clear_system_flag(SF_ABS_MF);
6          clear_system_flag(SF_ABS_NEEDED);
7          clear_system_flag(SF_ABS_COMPLETED);
8          no_inh_flag=1; /* During Usable SoC calculation, inhibition should
9              not happen*/
10         if(test_common_flag(CF_SET_EQU))
11         {
12             need_eq_after_abs=1;
13             clear_common_flag(CF_SET_EQU);
14             WDT_init_stop=0;
15             i2c_eeeprom_write_buff(0xFE7F,(uint8_t *) &common_flag,2);
16             WDT_init_stop=1;
17         }
18         /*Since equalization doesn't happen after every absorption. This flag tells
19            us whether to do an equalization or not after absorption*/
20         if(need_eq_after_abs)
21         {
22             is_time_to_equ=1;
23         }
24         else
25         {
26             is_time_to_equ=0;
27         }
28         /*Every time the absorption is completed the battery soc has to be reset*/
29         /*The soc is reset the first time battery voltage hits bat_max_v after
30            absorption*/
31         soc_reset_flag=0;
32         if(test_common_flag(CF_SET_ABS))
33         {
34             clear_common_flag(CF_SET_ABS);
35             WDT_init_stop=0;
36             i2c_eeeprom_write_buff(0xFE7F,(uint8_t *) &common_flag,2);
37             WDT_init_stop=1;
38         }
39     }

```

7. The number of force trips for a day variable and no force trip flag are cleared, to ensure dynamic entry.If the dynamic force trip entry feature is enabled, the time for entry is calculated at sundown.

```

1  /*Once dynamic force trip entry is enabled, at sundown the time at which the entry
2     should happen is estimated*/
3  /*On the time of entry the time is again calculated and checked with this value
4     inorder to ensure that battery did not discharge between that time period*/
5  estimated_ft_entry_time=Dynamic_FT_Entry_Time_Calculate();
6  if(((algo_param.dev_algo_state == ALGO_STATE_BAT_SOL_FRC_TRIP_6) || (algo_param.
7     dev_algo_state == ALGO_STATE_BAT_FRC_TRIP_4))&&(is_time_to_cal_useable_soc==0))
8  {
9      algo_param.dev_algo_state = ALGO_STATE_INV_5;
10     turtle_set_sw(algo_param.dev_algo_state);
11 }

```

20.5 Solar energy bucketing

The morning period is calculated at sundown. Solar energy bucketed in the turtle algorithm, is segregated into morning solar energy bucket and afternoon solar energy bucket.Morning and afternoon solar energy bucket for 3 days are calculated and stored in RAM. Moving average of morning solar bucket and afternoon solar bucket is calculated for these three days.

```

1 void Solar_Energy_Bucketing(){
2     Morning_Solar_Bucket=0.0;
3     for(int i=0;i<Morning_Sunhour;i++)
4     {
5         Morning_Solar_Bucket=Morning_Solar_Bucket+solar_power_bucket[i];

```

```
6  }
7  Afternoon_Solar_Bucket=0.0;
8  for (int i=Morning_Sunhour;i<=Sundown_Time;i++)
9  {
10     Afternoon_Solar_Bucket=Afternoon_Solar_Bucket+solar_power_bucket[i];
11 }
12 Day_Morning_Solar_Bucket[0]=Day_Morning_Solar_Bucket[1];
13 Day_Morning_Solar_Bucket[1]=Day_Morning_Solar_Bucket[2];
14 Day_Morning_Solar_Bucket[2]=Morning_Solar_Bucket;
15 MA_Morning_Solar_Bucket=(Day_Morning_Solar_Bucket[0]+Day_Morning_Solar_Bucket[1]+
    Day_Morning_Solar_Bucket[2])/3;
16 Day_Afternoon_Solar_Bucket[0]=Day_Afternoon_Solar_Bucket[1];
17 Day_Afternoon_Solar_Bucket[1]=Day_Afternoon_Solar_Bucket[2];
18 Day_Afternoon_Solar_Bucket[2]=Afternoon_Solar_Bucket;
19 MA_Afternoon_Solar_Bucket=(Day_Afternoon_Solar_Bucket[0]+Day_Afternoon_Solar_Bucket
    [1]+Day_Afternoon_Solar_Bucket[2])/3;
20 reset_solar_power_bucket();
21 }
```

20.6 Dynamic force trip exit

If dynamic exit flag is enabled for the day at sun up, the device estimates the solar energy for the day considering the solar energy bucketed. When solar time becomes greater than previous day morning sun hour, the solar energy to be received for the day is estimated. When the estimated energy becomes equal to the sum of solar energy received for the day and battery's depth of discharge, the device exits out of force trip.(either force trip or solar assisted force trip)

```

1 void
2 Dynamic_FT_Exit_Algorithm()
3 {
4     //Only if Dynamic_FT_Exit_Enable is set the algorithm works
5     //For the first 4 days the dynamic algorithm doesn't work
6     //When useable SOC is to be calculated the algorithm won't work
7     if(Dynamic_FT_Exit_Enable)//Solar Bucketing Code
8     {
9         if((Previous_Day_Morning_Sunhour <= Solar_Time)&&(Correction_Factor_Enable==0))
10        {
11            Correction_Factor=((MA_Afternoon_Solar_Bucket/MA_Morning_Solar_Bucket)*
12            Today_Recieved_Solar_Energy)+Today_Recieved_Solar_Energy;
13            Correction_Factor_Enable=1;
14        }
15
16        if(Correction_Factor_Enable)
17        {
18            if((Correction_Factor - Today_Recieved_Solar_Energy - (SOC_Error*SOC_Err_Inc))
19            <=((total_bat_capacity)-Bat_SOC))
20            {
21                Dynamic_FT_Exit=1;
22                Correction_Factor_Enable=0;
23            }
24        }
25
26        if(test_common_flag(CF_FT_EXIT))
27        {
28            if(total_time>server_ft_exit_time)
29            {
30                Dynamic_FT_Exit=1;
31            }
32        }
33    }
34 }

```

20.7 Detection of mains fail during absorption

During absorption of battery, mains fail flag is set to indicate a mains fail for more than 3 hours. The code to perform the above mentioned operation is shown below.

```

1 if(no_inh_flag&&(is_day_flag==NOW_NIGHT_TIME))
2 {
3     if((algo_param.cur_dis_i>MIN_DC_LOAD_I)&&(test_system_flag(SF_ABS_MF)==0))
4     {
5         ++mains_fail_counter;
6         if(mains_fail_counter>abs_mains_fail_tolerance)
7         {
8             post_system_flag(SF_ABS_MF);
9         }
10    }
11 }

```

20.8 Battery SoC reset

On the first day of installation, the battery SoC is reset if the battery voltage reaches maximum value. After periodic absorption, the battery SoC is reset when the battery voltage reaches maximum value. If the absorption is a sub-process of useable SoC calculation and the battery is of non-tubular type, algorithm enables the calculation of useable SoC.

```

1 /*
2  * Battery SoC is reset after every absorption.

```

```

3      * After absorption the moment the battery reaches maximum voltage, the soc is reset
4      to full capacity
5      */
6      if((algo_param.cur_bat_v>bat_max_v)&&(soc_reset_flag==0))
7      {
8          soc_reset_flag=1;
9          Bat_SOC=total_bat_capacity;
10         /*
11         * If the battery is of non-tubular type, after absorption complete the device is
12         set to ready for useable SoC calculation.
13         * This is executed, if the absorption was initiated because of a useable SoC
14         calculation.
15         */
16         if(test_system_flag(SF_ABS_COMPLETED) && (battery_type!=TUBULAR))
17         {
18             if(enable_cal_useable_soc)
19             {
20                 enable_cal_useable_soc=0;
21                 is_time_to_cal_useable_soc_entry=1;
22                 previous_voltage=bat_vol*1.034;
23                 frc_trip_count = max_frc_per_day;
24                 Dynamic_FT_Exit_Enable=0; //Dynamic FT does not happen during useable SoC
25                 calculation
26             }
27         }
28     }
29     /*
30     * On the first day of installation, the battery SoC is set to zero.
31     * To correct it on the first day, when the battery voltage hits maximum for the
32     first time, SoC is reset to full capacity.
33     */
34     if((algo_param.cur_bat_v>bat_max_v)&&(num_days==1))
35     {
36         Bat_SOC=total_bat_capacity;
37     }

```

20.9 Detection of equalization failure

For tubular batteries, after 6 hours of equalization from sun up, the SoC of the battery is checked to be greater than 95%. If the SoC is greater than 95% equalization is considered to be a success, else a failure.

```

1  /*
2  * If the battery is tubular and if it is day time and equalization is being done, a
3  counter runs for 6hours.
4  * After 6hours if the battery soc has fallen below 95% the equalization process is
5  considered as success.
6  * 95% tolerance is given to accommodate any short power failure during equalization
7  * Less than 95%, it is considered a failure
8  */
9  if(is_time_to_equ&&(is_day_flag==NOW_DAY_TIME)&&(battery_type==TUBULAR))
10  {
11      if(++bat_sol_equ_dur_cnt >= bat_sol_equ_dur)
12      {
13          is_time_to_equ=0;
14          force_trip_pause_flag=0;
15          force_trip_pause_count=0;
16          bat_sol_equ_dur_cnt = 0;
17
18          /* Success*/
19          if((soc_reset_flag==1)&&(Bat_SOC>(0.95*total_bat_capacity)))
20          {
21              if(need_eq_after_abs==0)
22              {
23                  bat_sol_equ_intr_cnt = 0;
24              }
25              previous_voltage=bat_vol*1.034;
26              need_eq_after_abs=0;
27              if(enable_cal_useable_soc)
28              {
29                  enable_cal_useable_soc=0;

```



```

28         is_time_to_cal_useable_soc_entry=1;
29         frc_trip_count = max_frc_per_day;
30         Dynamic_FT_Exit_Enable=0; /*Dynamic FT does not happen during useable
SoC calculation*/
31     }
32     post_system_flag(SF_EQ_COMPLETED);
33 }

```

If the equalization was a sub-process of useable SoC calculation, and equalization fails, algorithm checks if the useable SoC can be re-attempted. If it cannot be re-attempted the useable SoC calculation is deemed failure. If it can be re-attempted, request to re-attempt useable SoC is posted.

```

1         /*Failure*/
2         else
3         {
4             if(enable_cal_useable_soc)
5             {
6                 /*
7                 * If the equalization was initiated for useable SoC calculation and if
the equalization has failed, then useable SoC calculation is re-attempted the next
day.
8                 * If already re-attempted, and it has failed again, the useable SoC
calculation is considered a failure.
9                 */
10                fail=3;
11                re_cal_useable_soc_count++;
12                if(re_cal_useable_soc_count>1)
13                {
14                    enable_cal_useable_soc=0;
15                    post_system_flag(SF_USEABLE_SOC_FAILED);
16                    need_eq_after_abs=0;
17                    re_cal_useable_soc_count=0;
18                    //useable SoC Calculation failed
19                }
20                else
21                {
22                    ft_exit_start_day++;
23                    ft_entry_start_day++;
24                    post_system_flag(SF_USEABLE_SOC_CAL);
25                }
26            }
27            else
28            {
29                post_system_flag(SF_EQ_FAILED);
30            }
31        }
32    }

```

If the equalization was a sub-process of useable SoC calculation, and equalization is a success, the algorithm enables the calculation of useable SoC.

20.10 Useable SoC calculation standby

After completion of absorption and equalization sub-processes of the useable SoC calculation, the algorithm doesn't start the useable SoC calculation instantly. It waits till next day morning 3AM to start the useable SoC calculation. This is to ensure that solar would be available, if there is a situation of mains fail happens after useable SoC calculation. During the standby if there is a mains failure, the algorithm initiates a useable SoC calculation.

```

1  /*
2  * After the completion of absorption, (and equalization if tubular battery) the device
is ready to force trip if a useable SoC calculation is needed.
3  * The force trip is initiated if a mains fail occurs or if the solar time reaches
specified time.
4  * This time can be changed using APP/Server*/
5  if(is_time_to_cal_useable_soc_entry==1)
6  {
7      if(mains_available==0)
8      {
9          force_trip_pause_flag=0;
10         frc_trip_count=0;
11         no_frc_trip_flag = 0;

```

```

12 ready_for_frc_trip = 1;
13 is_time_to_cal_useable_soc_entry=0;
14 is_time_to_cal_useable_soc=1;
15 }
16 else
17 {
18 if(Solar_Time>=useabe_SoC_start_time)
19 {
20 force_trip_pause_flag=0;
21 frc_trip_count=0;
22 no_frc_trip_flag = 0;
23 ready_for_frc_trip = 1;
24 is_time_to_cal_useable_soc_entry=0;
25 is_time_to_cal_useable_soc=1;
26 }
27 }
28 }

```

20.11 Force trip restriction

The number of force trip that can occur on a given day is preset to 3. This value can be configured based on the user. Once the number of force trip reaches this value, no more force trip for the day can happen. Dynamic force trip entry is not considered while calculating the number of force trip occurred per day. If a algorithm has exited out of a force trip, only after 30 minutes it can re-enter into a force trip. This is because, for weak batteries the voltage level rises faster though the SoC level is low.

```

1  /*
2   * If force trip count for the day reaches maximum allowable value, force trip will
3   * not happen for the reset of the day.
4   */
5   if((frc_trip_count >= max_frc_per_day))
6   {
7     no_frc_trip_flag = 1;
8   }
9
10  /*
11   * A time period of 30 minutes is assigned between two consecutive force trips
12   */
13  if(force_trip_pause_flag)
14  {
15    if(force_trip_pause_count < FRC_TRIP_PAUSE_DUR_SEC)
16    {
17      force_trip_pause_count++;
18    }
19    else
20    {
21      force_trip_pause_count = 0;
22      force_trip_pause_flag = 0;
23    }
24  }

```

20.12 Critical battery charge back

If the battery voltage goes below a predefined voltage ([mains charge in voltage](#)), the battery is charged back from both mains and solar. This predefined voltage before useable SoC calculation, will be based on the battery voltage rating. During useable SoC calculation, the [permissible force trip exit voltage](#) is also calculated. After useable SoC calculation, mains charge in voltage is set 0.2V below permissible force trip exit voltage.

```

1  /*
2   * If the battery voltages goes critical, the battery is charged using mains (and
3   * solar if available).
4   */
5   if((algo_param.cur_bat_v < bat_mains_chg_in_v) && (need_solar_mains_top_up == 0))
6   {
7     need_solar_mains_top_up=1;
8     mains_sol_chg_flag = 1;
9     post_system_flag(SF_SMC);

```

```

9     mains_topup_count = 0;
10 }
11
12 /*
13  * If mains+solar charging is allowed and if battery voltage reaches the required
14  * value the mains recharge is cut.
15  * Since voltage of old batteries reach full voltage faster a time out of 30 minutes
16  * is provided.
17  * During this 30 minutes, once the battery voltage hits full the battery will be
18  * cycled using solar.
19  */
20 if(need_solar_mains_top_up)
21 {
22     if(mains_topup_count < MIN_MAINS_TOPUP_DUR_SEC)
23     {
24         mains_topup_count++;
25     }
26     if((Bat_SOC >= night_soc_corr*total_bat_capacity) && \
27        (mains_topup_count >= MIN_MAINS_TOPUP_DUR_SEC))
28     {
29         clear_system_flag(NGT_SOC_ERR);
30         need_solar_mains_top_up = 0;
31         mains_sol_chg_flag = 0;
32         if(test_system_flag(SF_SMC))
33         {
34             clear_system_flag(SF_SMC);
35         }
36         mains_chg_flag=0;
37         mains_topup_count = 0;
38     }
39 }

```

20.13 Solar availability detection

During solar charging, solar assisted force trip and solar assisted mains charging mode, the solar mosfet is on, hence the solar voltage read from the ADC will be equal to the battery voltage. The following code detects the availability of solar during these three states. The solar detection is done by checking the solar current. If the solar current goes below 0.1A for more than 5 minutes, the solar mosfet is turned off and solar voltage is measured using ADC.

```

1  /*
2   * During state 3, state 6 and state 7 since solar switch is closed, the voltage
3   * across solar panel will be equal to battery voltage.
4   * Hence to check for presence of solar the solar current is checked.
5   * If the current is less than the minimum solar current for 3 minutes low solar
6   * flag is set.
7   * This flag is used to exit from states in which the solar switch is closed.
8   */
9
10 if((algo_param.dev_algo_state == ALGO_STATE_BAT_SOL_FRC_TRIP_6) || \
11    (algo_param.dev_algo_state == ALGO_STATE_SOL_INV_DIS_7) || (algo_param.dev_algo_state
12    == ALGO_STATE_SOL_CHG_3))
13 {
14     if(algo_param.cur_sol_i < MIN_SOL_I)
15     {
16         if(low_sol_check_count < LOW_SOL_DETECT_DUR_SEC)
17         {
18             low_sol_check_count++;
19         }
20         else
21         {
22             low_sol_flag = 1;
23             low_sol_check_count = 0;
24         }
25     }
26     else
27     {
28         low_sol_flag = 0;
29         low_sol_check_count = 0;
30     }
31 }

```

```

29 }
30 else
31 {
32     low_sol_flag = 0;
33     low_sol_check_count = 0;
34 }

```

20.14 Morning and evening peak inhibition

A provision is available to stop the charging during morning and evening peak hours. The inhibition can be enabled or disabled using the configuration tool as well as the server.

```

1  /*
2   * During peak hours, the battery is stopped from getting charged from the mains.
3   * This feature is disabled, but can be enabled using APP/Server if required.
4   * During morning peak hour and evening peak hour the battery can be inhibited.
5   * This has to be set during configuration.
6   */
7  if(!test_error(ERR_RTC))
8  {
9      if((is_time_between(ALGO_TIME_MORN_PEAK_CHG_INH_START,
10         ALGO_TIME_MORN_PEAK_CHG_INH_STOP)) && \
11         (morn_chg_inh_flag))
12      {
13          peak_inh_flag = 1;
14      }
15      else if((is_time_between(ALGO_TIME_EVEN_PEAK_CHG_INH_START,
16         ALGO_TIME_EVEN_PEAK_CHG_INH_STOP)) && \
17         (even_chg_inh_flag))
18      {
19          peak_inh_flag = 1;
20      }
21      else
22      {
23          peak_inh_flag = 0;
24      }
25  }
26  else
27  {
28      peak_inh_flag = 0;
29  }

```

20.15 Inverter overload detection during force trip

During force trip and solar assisted force trip, the customer's load is met by the inverter. If the load goes beyond, the maximum capacity of the inverter the inverter might trip causing inconvenience to the user. Hence by measuring the discharge current, the algorithm exits from force trip before the inverter trip of overload.

```

1  /*
2   * During state 4 and state 6 to prevent overload trip of inverter, the current is
3   * being checked.
4   * If current is more than the maximum allowable current for more than 5sec the
5   * overload flag is set.
6   */
7  if((algo_param.dev_algo_state == ALGO_STATE_BAT_SOL_FRC_TRIP_6) || \
8     (algo_param.dev_algo_state == ALGO_STATE_BAT_FRC_TRIP_4))
9  {
10     if(dc_load_i > max_dc_load_i)
11     {
12         if(over_load_check_count < DC_OVER_LOAD_DETECT_DUR_SEC)
13         {
14             over_load_check_count++;
15         }
16         else
17         {
18             dc_over_load_flag = 1;
19             over_load_check_count = 0;
20         }
21     }
22 }

```

```
20     else
21     {
22         dc_over_load_flag = 0;
23         over_load_check_count = 0;
24     }
```

20.16 Inverter failure detection

Inverter failure is detected during force trip. Inverter consumes a no load current when connected to battery. If during force trip, the no load current is less than the defined minimum current, the inverter is considered to have failed. The code to detect inverter failure is given below.

```

1  /*
2      * The following code checks for the failure of the inverter.
3      * An inverter connected to battery will consume a no-load current of 1.5A minimum.
4      * If the battery discharge current during FT is less than this minimum value, then
      it means the inverter has failed.
5      */
6
7      if(dc_load_i < MIN_DC_LOAD_I)
8      {
9          if(under_load_check_count < LOW_DC_LOAD_DETECT_DUR_SEC)
10         {
11             under_load_check_count++;
12         }
13         else
14         {
15             dc_under_load_flag = 1;
16             under_load_check_count = 0;
17         }
18     }
19     else
20     {
21         dc_under_load_flag = 0;
22         under_load_check_count = 0;
23     }
24 }
25 else
26 {
27     dc_over_load_flag = 0;
28     dc_under_load_flag = 0;
29     over_load_check_count = 0;
30     under_load_check_count = 0;
31 }

```

20.17 Evening force trip exit

After 4:30PM the amount of solar energy received is relatively low. Hence if the device is in force trip, the algorithm exits and moves into solar charging mode. If the battery voltage reaches maximum value, the solar energy is used to cycle the battery.

Note:-The 4:30PM mentioned is not calculated from the RTC, whereas it is calculated relative to the previous day sunset time.

```

1  /*
2      * Dynamic force trip exit feature is provided to ensure that the battery is fully
      charged at sundown.
3      * Forecasting the previous solar pattern and the present battery SoC the decision
      to exit FT is done.
4      */
5
6      if(((algo_param.dev_algo_state == ALGO_STATE_BAT_SOL_FRC_TRIP_6) || \
7      (algo_param.dev_algo_state == ALGO_STATE_BAT_FRC_TRIP_4))&&(
      is_time_to_cal_useable_soc==0)&&(is_time_to_dynamic_ft_entry==0)&&(is_day_flag ==
      NOW_DAY_TIME)&&(sss_start==0))
8      {
9          if(total_time>(((uint32_t) Previous_Day_Sundown *(uint32_t) 3600)-(uint32_t)
      5400))
10         {
11             Dynamic_FT_Exit_Enable=0;
12             Dynamic_FT_Exit=0x00;
13             no_frc_trip_flag = 1;
14             state_change_reason=25; //Evening FT Exit
15             eve_dy_exit=1;
16             ready_for_frc_trip=0;
17             frc_trip_count=max_frc_per_day;
18
19             if(is_time_to_dynamic_ft_entry)

```

```
20     {
21         if(is_day_flag==NOW_NIGHT_TIME)
22         {
23             dynamic_ft_complete=1;
24         }
25         is_time_to_dynamic_ft_entry=0;
26     }
27 }
28 }
```

20.18 Evening battery low charging

After exiting from force trip at 4:30PM, if the battery is less than 90% of the total battery capacity (Rated battery voltage x Rated battery AH), the battery is charged back from both mains and solar.

```

1      if(((total_time>((uint32_t) Previous_Day_Sundown *(uint32_t) 3600)-(uint32_t)
2      5400))&&(eve_soc_error==0)&&(is_day_flag == NOW_DAY_TIME)&&(
3      is_time_to_cal_useable_soc==0)&&(dynamic_ft_complete==0))
4      {
5          /*This is to ensure battery is greater than 90% of total capacity during sun
6      down*/
7          if(Bat_SOC<(eve_soc_corr*total_bat_capacity))
8          {
9              mains_sol_chg_flag = 1;
10             post_system_flag(SF_SMC);
11             eve_soc_error=1;
12             post_system_flag(EVE_SOC_ERR);
13         }
14         else
15         {
16             eve_soc_error=0;
17         }
18     }

```

20.19 Night battery low charging

There are various reasons a battery can get low during night time. The algorithm might be performing a useable SoC calculation or a dynamic force trip entry or it might have exited from a dynamic entry. During the above mentioned conditions the battery must not be charged back from the mains. If the battery goes low during a regular operation due to a power failure, the battery is charged back at night

```

1  /*
2  * At sundown if the battery SoC falls below 90% of total capacity charge back to 90%.
3  * The battery will not be charged back if the battery was discharged during dynamic
4  entry.
5  */
6  if(((is_day_flag == NOW_NIGHT_TIME)&&(Bat_SOC<(night_soc_corr*total_bat_capacity))&&(
7  is_time_to_dynamic_ft_entry==0)&&(no_inh_flag==0)&&(num_days!=0)&&(
8  dynamic_ft_complete==0)&&(is_time_to_cal_useable_soc==0)&&(is_sol_sensed!=0))
9  {
10     need_solar_mains_top_up=1;
11     post_system_flag(NGT_SOC_ERR);
12 }

```


20.20 Load energy bucketing

The user's load energy, is bucketed over a period of 24 hours. Bucketing is based on solar time and not the RTC time. The day bucket is pushed at sun up. Sun up to sun down is considered as the morning load hours, sun down + 5 hours is considered as the evening load hours, sun down + 5 hours to next sun up is considered as night load hours. The bucketed energy is separated into morning, evening and night loads. Morning and night loads for the past seven days are stored in RAM for forecasting purpose.

```

1 void Load_Energy_Bucketing()
2 {
3     Day_Load_Bucket=0.0;
4     for (int i = 0; i < Sundown_Time; i++)
5     {
6         Day_Load_Bucket=Day_Load_Bucket+load_power_bucket[i];
7     }
8
9     Night_Load_Bucket=0.0;
10    for (int i = Sundown_Time+6; i <= 23; i++)
11    {
12        Night_Load_Bucket=Night_Load_Bucket+load_power_bucket[i];
13    }
14
15    Day_Morning_Load_Bucket[0]=Day_Morning_Load_Bucket[1];
16    Day_Morning_Load_Bucket[1]=Day_Morning_Load_Bucket[2];
17    Day_Morning_Load_Bucket[2]=Day_Morning_Load_Bucket[3];
18    Day_Morning_Load_Bucket[3]=Day_Morning_Load_Bucket[4];
19    Day_Morning_Load_Bucket[4]=Day_Morning_Load_Bucket[5];
20    Day_Morning_Load_Bucket[5]=Day_Morning_Load_Bucket[6];
21    Day_Morning_Load_Bucket[6]=Day_Load_Bucket;
22
23    Day_Night_Load_Bucket[0]=Day_Night_Load_Bucket[1];
24    Day_Night_Load_Bucket[1]=Day_Night_Load_Bucket[2];
25    Day_Night_Load_Bucket[2]=Day_Night_Load_Bucket[3];
26    Day_Night_Load_Bucket[3]=Day_Night_Load_Bucket[4];
27    Day_Night_Load_Bucket[4]=Day_Night_Load_Bucket[5];
28    Day_Night_Load_Bucket[5]=Day_Night_Load_Bucket[6];
29    Day_Night_Load_Bucket[6]=Night_Load_Bucket;
30
31    reset_load_power_bucket();
32 }

```

20.21 Dynamic force trip entry algorithm

Once the estimated time becomes equal to the actual device time, the device enters into dynamic force trip.

If there happens a power failure, after the estimation, the device estimates the entry time again. If the previous entry time and calculated entry time difference is below 10 minutes then the device enters into force trip else the device delays the dynamic force trip entry.

20.22 Dynamic force trip entry time estimate

Dynamic force trip entry time is estimated when the dynamic force trip entry enable is set. The load energy bucketed for the past seven days is used to estimate the load of the user. The load prediction can be done in two ways. Either one can be selected depending on the dev_state_flag.

If dev_state_flag is zero then the previous week's same day and yesterday will be considered for load prediction. If the dev_state_flag is one then the previous week's same day and the average of the other days will be considered for prediction. This flag is configured to 1 during configuration and can be changed from the server.

The depth of discharge of the battery is estimated. A correction to this is offered to account for any temperature influence on the AH calculation.

The next day's Solar is predicted based on the past three day's Solar energy bucketed.

Conditions:-

1. If the Solar energy to be received tomorrow is greater than morning load predicted the device is ready to enter into dynamic FT. The time at which it has to enter is to be estimated.

2. If the Solar predicted subtracted by morning load predicted and DoD is greater than the useable SoC, then an FT constant is calculated based on useable SoC and night load predicted.
3. If the Solar predicted subtracted by morning load predicted and DoD is less than the useable SoC, we check if it is greater than the night load predicted, if it is greater we set the FT constant to be 1.
4. If the Solar predicted subtracted by morning load predicted and DoD is less than the predicted night load, We find a constant based on the remaining energy and set it as the FT constant.

Depending on the FT constant calculated as mentioned above, the dynamic force trip enters anytime between 12:00PM to 6:00AM. Even in some conditions if the FT constant turns out to be a value which allows to force trip before 12:00PM, the device force trips only at 12:00PM.

Exceptions:-

1. If some users have low load during morning sun hours, the energy bucketed will be low. This is because the device goes into Solar cycling, to make sure the battery doesn't get over charged. In these cases, if the equalization was for more than one hour and battery SoC is greater than 95% the device enters into dynamic force trip by estimating a FT constant based on useable SoC and the predicted night load.
2. If the Solar energy is not sufficient to meet the predicted morning load, the dynamic force trip doesn't happen at all for the day.

20.23 Battery overvoltage protection

The battery voltage is checked if it is greater than the allowable value in every 300 millisecond. If the battery voltage reaches the set limit, the solar switch is opened. A counter in the loop calculates the time taken for equalization.

21 State Change Algorithm

The device operates in six different modes based on the sensor values. They are as given in the table 1.

Sl.No	State	Operation	Battery	Load
1	State 1	Inhibition	Not charged	Mains
2	State 3	Solar charging	Solar	Mains
3	State 4	Force trip	Not charged	Battery
4	State 5	Mains/Inverter mode	Mains	Mains
5	State 6	Solar assisted force trip	Solar	Battery
6	State 7	Mains+solar charging	Mains+Solar	Mains

Table 1: Device state

The following section describes the state transition conditions for iCON LV. Refer 10 for state transition diagram. The following conditions are to be met for state transition.

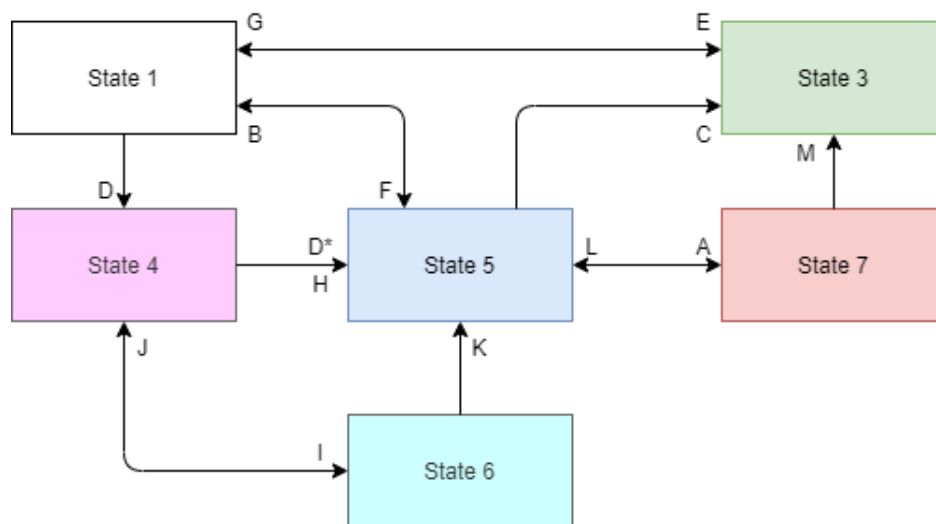


Figure 10: State transition diagram

21.1 State 5 to State 7

Transition Code:- A

Conditions:-

1. Solar voltage available **and**
2. Battery voltage less than solar connectivity voltage **and**
3. Charging of battery from mains and solar necessary

OR

1. mains charging necessary **and**
2. Solar available **and**
3. Battery voltage less than solar connectivity voltage

21.2 State 5 to State 1

Transition Code:- B

Conditions:-

1. Mains available **and**
2. Mains charging not necessary

21.3 State 5 to State 3

Transition Code:- C

Conditions:-

1. Mains not available **and**
2. Solar available **and**
3. Battery voltage less than solar connectivity voltage **and**
4. Mains charging not necessary

21.4 State 1 to State 5

Transition Code:- D*

Conditions:-

1. Mains charging necessary **or**
2. Solar and mains charging necessary

NOTE:- * indicates that to transition from state 1 to state 5, the device goes through 4. This is to ensure, that the inverter doesn't get damaged while turning the battery switch ON

21.5 State 1 to State 4

Transition Code:- D

Conditions:-

1. Night **and**
2. AC load less than maximum AC load **and**
3. Dynamic entry flag set

OR

1. Battery full **and**
2. Solar available **OR** early FT entry **and**
3. Device ready for force trip **and**
4. AC load less than maximum AC load

21.6 State 1 to State 3

Transition Code:- E

Conditions:-

1. Mains not available.

NOTE:- From v7.0 solar assisted power failure state is removed. Hence when power fails, the device transits to state 3 and the battery switch is turned on by the current interlock

OR

1. Solar available and
2. Battery voltage less than solar connectivity voltage

21.7 State 1 to State 5

Transition Code:- F

Conditions:-

1. Solar unavailable and
2. Mains unavailable

21.8 State 3 to State 1

Transition Code:- G

Conditions:-

1. Battery voltage greater than solar connectivity voltage **or**
2. Solar current low for 5 minutes **or**
3. Charging of battery from mains and solar necessary **or**
4. Early FT entry

21.9 State 4 to State 5

Transition Code:- H

Conditions:-

1. Battery depth of discharge is less than useable SoC **or**
2. Battery voltage less than force trip exit voltage **or**
3. Battery voltage less than useable SoC exit voltage during useable SoC calculation **or**
4. DC overload **or**
5. Inverter failure **or**
6. User button press force trip exit **or**
7. Dynamic force trip exit **or**
8. Evening force trip exit

21.10 State 4 to State 6

Transition Code:- I

Conditions:-

1. Battery voltage less than solar connectivity voltage **and**
2. Solar available **and**
3. Not a useable SoC calculation force trip

21.11 State 6 to State 4

Transition Code:- J

1. Battery voltage greater than battery maximum voltage **or**
2. Solar current low for 5 minutes

21.12 State 6 to State 5

Transition Code:- K

1. Battery voltage less than force trip exit voltage **or**
2. DC overload **or**
3. Inverter failure **or**
4. Battery depth of discharge greater than useable SoC **or**
5. Dynamic exit **or**
6. Evening force trip exit **or**
7. User button press force trip exit

21.13 State 7 to State 5

Transition Code:- L

Conditions:-

1. Battery voltage greater than maximum battery voltage **or**
2. Solar current low for 5 minutes

21.14 State 7 to State 3

Transition code:- M

Conditions:-

1. Charging current greater than maximum allowable charging current

NOTE:- During state 7 if there is a mains failure, the device stays in state 7. As soon as the mains is available, the device charges the battery with both Solar and mains

22 app main

22.1 esp32s2_main.h

```

1 #ifndef _esp32s2_main_h
2 #define _esp32s2_main_h
3
4 #include "stdio.h"
5 #include "stdlib.h"
6 #include "stdint.h"
7
8 #define ICUBE 1 // Set 1 for iCUBE code. For iCON code make this 0 %
9
10 typedef enum{
11     no_press = 0,
12     single_press = 1,
13     double_press = 2,
14     long_press = 3,
15 }switchpress_value;
16
17 void init_spiffs(void);
18 void read_domain(char *ptr);
19 void read_spiffs( char *ptr, int type );
20 void update_spiffs( char *param, int type );
21 void fcn_factory_reset(void);
22
23 #endif

```

22.2 Idle task monitor to reset controller

```

1 /*
2  * Reset the chip if any thread gets stuck. WDT is enabled using idf.py menuconfig
3  * Component config -> Common ESP related
4  */
5 extern void esp_task_wdt_isr_user_handler(void){
6     esp_restart();
7 }

```

22.3 SPIFFS partition paths

```

1 /*
2  * Following are the files stored in the SPIFFS partition
3  * Each file corresponds to a particular URL
4  * .cer file corresponds to the certificate used to connect with HTTPS
5  */
6 char urls[10][32] = {
7     "/spiffs/getReadWrite.txt",
8     "/spiffs/updateCalib.txt",
9     "/spiffs/updateConfig.txt",
10    "/spiffs/updateThreshold.txt",
11    "/spiffs/updateStatus.txt",
12    "/spiffs/updateSoC.txt",
13    "/spiffs/updateLog.txt",
14    "/spiffs/getCert.txt",
15    "/spiffs/google.cer",
16    "/spiffs/updateOTA.txt"
17 };

```

22.4 SPIFFS Initialization

```

1 /*
2  @Info:
3  The following code is used to initialize SPIFFS.
4  This enables the FLASH to be partitioned into a storage space
5  In our application the SPIFFS is used to store server certificate
6  Incase the server certificate expires it is downloaded over HTTP and is stored in flash
7  Since the certificate is a public key HTTPS transfer is not required

```

```

8
9 @Param:
10 None
11
12 @Return:
13 None
14 */
15 void init_spiffs(void){
16     esp_vfs_spiffs_conf_t conf = {
17         .base_path = "/spiffs", .partition_label = NULL,
18         .max_files = 5, .format_if_mount_failed = true
19     };
20     esp_err_t ret = esp_vfs_spiffs_register(&conf);
21     if (ret!=ESP_OK){
22         if (ret == ESP_FAIL){ ESP_LOGE(TAG, "Failed to mount or format filesystem"); }
23         else if (ret == ESP_ERR_NOT_FOUND){ ESP_LOGE(TAG, "Failed to find SPIFFS partition"); }
24         else{ ESP_LOGE(TAG, "Failed to initialize SPIFFS (%s)", esp_err_to_name(ret)); }
25     }
26     return;
27 }
28 size_t total = 0, used = 0;
29 ret = esp_spiffs_info(NULL, &total, &used);
30 if (ret != ESP_OK){ESP_LOGE(TAG, "Failed to get SPIFFS partition information (%s)",
31     esp_err_to_name(ret));}
32 else{ ESP_LOGI(TAG, "Partition size: total: %d, used: %d", total, used); }
33 }

```

22.5 Read data from SPIFFS

```

1 /*
2 @Info:
3 The following code is used to retrieve URLs and certificate from the SPIFF partition
4 These data are used for HTTPS application (data logging and OTA)
5
6 @Param:
7 *ptr - Address of character array to store the data from SPIFFS is passed to it
8 type - Type of data is being read
9
10 @Return:
11 None
12 */
13 void read_spiffs(char *ptr, int type){
14     bzero(ptr,2048);
15     ESP_LOGI(TAG, "reading from %s",urls[type]);
16     FILE* f = fopen(urls[type], "r");
17     fseek(f, 0, SEEK_END);
18     long fsize = ftell(f);
19     fseek(f, 0, SEEK_SET); /* same as rewind(f); */
20     fread(ptr, 1, fsize, f);
21     fclose(f);
22     ptr[fsize] = 0;
23     ESP_LOGI(TAG, "Data read is \n %s",ptr);
24 }

```

22.6 Update certificate in SPIFFS

```

1 /*
2 @Info:
3 The following function removes the existing certificate file and replaces it with the
4 new file received over HTTP
5 This is done when the already existing certificate expires
6 After replacing the certificate read_certificate function is called to load the
7 certificate
8
9 @Param:
10 None
11
12 @Return: None
13 */
14 void update_certificate_in_spiffs(char *param){

```

```

14 ESP_LOGI(TAG, "updating google.cer");
15 remove("/spiffs/google.cer");
16 FILE* f = fopen("/spiffs/google1.cer", "w");
17 if(f==NULL){ESP_LOGE(TAG, "Failed to open google.cer");return;}
18 fprintf(f,"%s",param);
19 fclose(f);
20 rename("/spiffs/google1.cer","/spiffs/google.cer");
21 read_certificate();
22 }

```

22.7 Read domain name from SPIFFS

The following code is used to read the domain name of the API stored in the SPIFFS.

```

1 void read_domain(char *ptr){
2     bzero(ptr,2048);
3     ESP_LOGI(TAG, "reading from domain.txt");
4     FILE* f = fopen("/spiffs/domain.txt", "r");
5     fseek(f, 0, SEEK_END);
6     long fsize = ftell(f);
7     fseek(f, 0, SEEK_SET); /* same as rewind(f); */
8     fread(ptr, 1, fsize, f);
9     fclose(f);
10    ptr[fsize] = 0;
11    ESP_LOGI(TAG, "Data read is \n %s",ptr);
12 }

```

22.8 peripheral initialization

```

1  /*
2  @Info:
3  All peripherals initialization and task initialization functions are group under the
4  following function
5  This function is called inside the app_main.
6  Any new initializations in the future must be added to this function.
7
8  @Param:
9  None
10
11 @Return:
12 None
13 */
14 void peripheral_initializaton(void){
15     #if ICUBE
16     #warning "Device configured as iCUBE"
17     #else
18     #warning "Device configured as iCON"
19     #endif
20     init_buzzer();
21     init_led();
22     init_i2c();
23     init_switches();
24     #if ICUBE
25     S4_SW_init();
26     #endif
27     init_pushbutton();
28     init_adc();
29     init_mains_sense();
30     initializeBatteryOvervoltageSense();
31     init_pac1952();
32     check_time();
33     init_spiffs();
34     read_spiffs((char *)server_cert_pem_start,read_certificate);
35     power_on_initialization();
36     esp32s2_WiFi_Init();
37     createSemaphores();
38     createTasks();
39 }

```

22.9 app_main

```

1  /*
2  @Info:
3  This function is equivalent to main() in a baremetal program
4  All peripherals and tasks are created inside this function
5  The code placed inside app_main should not contain a infinite loop
6  Incase there is an infinite loop the WDT will clear the controller after 12 seconds
7  @Param:
8  None
9  @Return:
10 None
11 */
12 void app_main(void){
13     peripheral_tasks_initializaton();
14 }

```

22.10 Factory reset

The following code boots the device from the factory partition if the codes in OTA partitions turns bad

```

1 void fcn_factory_reset(void){
2     esp_partition_iterator_t pi = esp_partition_find(ESP_PARTITION_TYPE_APP,
3     ESP_PARTITION_SUBTYPE_APP_FACTORY, "factory");
4     if( pi != NULL ){
5         const esp_partition_t * factory = esp_partition_get ( pi );
6         esp_partition_iterator_release ( pi );
7         if(esp_ota_set_boot_partition(factory)==ESP_OK) esp_restart () ;
8     }
9 }

```

23 Secure boot and Secure flash update

Note:-

1. Please follow the instructions given below carefully.
2. The chances of the controller getting bricked is high in case of faulty configurations.

23.1 Generating keys for secure boot and flash encryption

1. Use openssl to generate the key for secure boot. ESP32S2 can accept upto 3 keys. If git is already installed the openssl will be available in the machine. Add the *C : \ProgramFiles\Git\usr\bin* to path variable to access openssl from cmd.
2. Use the command `openssl genrsa -out secure_boot_signing_key_x.pem 3072` to generate the keys. Here x denotes the number of keys.
3. Use the following commands to generate bin for flash encryption
 - (a) `python path\espsecure.py generate_flash_encryption_key flash_encryption_key1.bin`
 - (b) `python path\espsecure.py generate_flash_encryption_key flash_encryption_key2.bin`

23.2 Setting up keypurpose for ESP32S2

The following code will link the keypurpose the required key. After executing the following command the command requests for a confirmation. After entering BURN (all caps) the respective fuse will be written into the device. The commands link BLOCK 4, BLOCK 5, BLOCK 6 to SECURE BOOT and BLOCK 7, BLOCK 8 to FLASH ENCRYPTION

1. `python path\espefuse.py -p COM burn_efuse KEY_PURPOSE_0 9`

2. `python path\espefuse.py -p COM burn_efuse KEY_PURPOSE_1 10`
3. `python path\espefuse.py -p COM burn_efuse KEY_PURPOSE_2 11`
4. `python path\espefuse.py -p COM burn_efuse KEY_PURPOSE_3 2`
5. `python path\espefuse.py -p COM burn_efuse KEY_PURPOSE_4 3`

23.3 Burning the digest into the BLOCKs

In the previous section, each BLOCKs 4,5,6,7 and 8 were assigned a specific purpose. In this section explains how to write the key values into the BLOCKs.

Writing the secure boot keys to BLOCKs 4, 5 and 6

The following code is used to write the keys into the BLOCKs. After each command the cmd expects BURN to be entered to burn the keys.

1. `python path\espefuse.py -p COM burn_key_digest BLOCK4 path\secure_boot_signing_key_1.pem SECURE_BOOT_DIGEST0`
2. `python path\espefuse.py -p COM burn_key_digest BLOCK5 path\secure_boot_signing_key_2.pem SECURE_BOOT_DIGEST1`
3. `python path\espefuse.py -p COM burn_key_digest BLOCK6 path\secure_boot_signing_key_3.pem SECURE_BOOT_DIGEST2`

23.4 Writing the flash encryption keys to BLOCK 7 and 8

The following code is used to write the keys into the BLOCKs 7 and 8. Enter BURN after each command.

1. `python path\espefuse.py --port COM burn_key BLOCK7 flash_encryption_key1.bin XTS_AES_256_KEY_1`
2. `python path\espefuse.py --port COM burn_key BLOCK8 flash_encryption_key2.bin XTS_AES_256_KEY_2`

23.5 Change to custom partition

Before enabling the secure boot and flash encryption the partition table has to be edited. This is because after enabling secure boot and flash encryption the size of bootloader will increase and it will overlap with the partition table address. If one doesn't want to change the partition table then the log verbosity has to be compensated which will increase debugging complexity. The following is the easiest way to update the partition table.

The command `python path_gen_esp32part.py build\partition_table\partition-table.bin custom_partition_table.csv` will generate a custom_partition_table.csv file inside the project. Remove all the offset values of every sector from the custom_partition_table.csv file. This way the partition table offset in the menuconfig will automatically set the other partition's starting address. This offset can be changed from the menuconfig.

24 Enabling secure boot and flash encryption

The process carried out till this point doesn't enable secure boot or flash encryption. It only writes the respective fuses which will be used by the secure boot and flash encryption. The secure boot and flash encryption is enabled via menuconfig.

1. Enter the command `idf.py menuconfig`. It opens the menuconfig dialog box.
2. Select `Security_features` settings in the menuconfig dialog box
3. Check the `Enable hardware Secure Boot in bootloader`
4. Uncheck `Sign binaries during build`
5. Check `Enable flash encryption on boot`

6. Select [size of key](#) as 512-bit
7. Select [Release](#) usage mode
8. Check [permanently disable ROM download mode](#)
9. Select Partition table, choose custom partition and add the custom_partition_table.csv which was created in the previous section.
10. Change the offset from 0x8000 to 0xB000
11. Save and close the dialog box

24.1 Build and sign the Bootloader

The command `idf.py bootloader` to build the bootloader. Once the bootloader is successfully build it has to be signed using three keys generated.

The command `python.exe path\espsecure.py sign_data -k path\private_key_1.pem path\private_key_2.pem path\private_key_3.pem -v 2 --append_signatures -o signed_bootloader.bin build\bootloader\bootloader.bin`

The signed_bootloader.bin must be flashed into the controller instead of bootloader.bin

24.2 Building the app image

The command `idf.py build` builds the app. Once the app is complete. The app_image.bin and app_data_initial.bin has to be signed before flashing the chip

Assuming ota_00_05.bin is the image binary file and ota_data_initial.bin is the app_data_initial.bin the following command must be used to sign the binary files.

1. `python.exe path\espsecure.py sign_data -k path\private_key_1.pem -v 2 build\ota_00_05.bin`
2. `python.exe path\espsecure.py sign_data -k path\private_key_1.pem -v 2 build\ota_data_initial.bin`

Don't sign the binary will all three signatures. It must be signed only with one signature. If signature 1 is compromised, sign the app with signature 2 and transfer it over OTA. The esp32s2 chip will automatically revoke the first signature and use the second signature henceforth.

24.3 Flashing the controller

The following command is used to flash the controller.

`python path\esptool.py -p COM -b 115200 --before default_reset --after no_reset --chip esp32s2 --no-stub write_flash --flash_mode dio --flash_size keep --flash_freq 80m 0x1000 signed_bootloader.bin 0xb000 build\partition_table\partition-table.bin 0x10000 build\ota_data_initial.bin 0x20000 build\ota_00_05.bin`

After flashing, on the next reboot the secure boot and flash encryption will be enabled. The firmware can be flashed only via OTA.

It sufficient to update only the app.bin to perform OTA update. Before updating app.bin on the server build and sign the image using command mentioned in section 24.2 The current esp-idf version (4.2) supports only plaintext flashing over OTA.