Solar Charge Controller

Abdulla Almazrouei

Advisor: Dr. Lotfalian

April 26th, 2019

Evansville, Indiana

Acknowledgments

This project would not have been completed without the support of my project supervisor, Dr. Mohsen Lotafalian. I would also like to give special thanks to Dr. Christina Howe for her support in writing in the reports and presentation. Also, special thanks to Mr. Jeff Cron, and the rest of the EE faculty at the University of Evansville for their guidance and support throughout these two semesters.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Solar Charge Controller Definition

A solar charge controller is a voltage and current regulator that prevents a battery bank from overcharging due to solar arrays. The voltage and current coming from the solar panel is being regulated before going to the batteries by ensuring that a deep cycle battery does not overcharge during the day. Furthermore, no power gets back to the panels that will drain the battery during the night when there is no sun energy to charge up the solar panel. There are several charge regulators that have additional capabilities such as load control and lighting. However, controlling the current and voltages is their primary job. A solar charge regulator is very crucial and is needed to prevent overcharging of the batteries. Most of the 12-volt panels always supply 17 volts because if it was 12 volts, then it means it works under perfect conditions something that does not happen in all places. The extra voltage supplied by the panel caters to when the sun is low or when covered by heavy clouds so as to ensure output voltage to the batteries.

1.2. Background

The core function of the solar charge controller is the efficient transfer of power from a solar module to a battery or load. There are two different types of solar charge regulators, each with a different technology; maximum power point tracking (MPPT) and pulse width modulation (PMW). Their performance is very different from each other; for example, the MPPT solar charge controller is expensive compared to the PMW regulator. The MPPT regulator performs better than the PWM solar charge regulator. This can be seen in figure 1. The PWM charge regulator operates by making a direct connection from the solar panel to the battery, whereas the

MPPT charge regulator measures the voltage of the panel and converts it into the battery voltage [1].

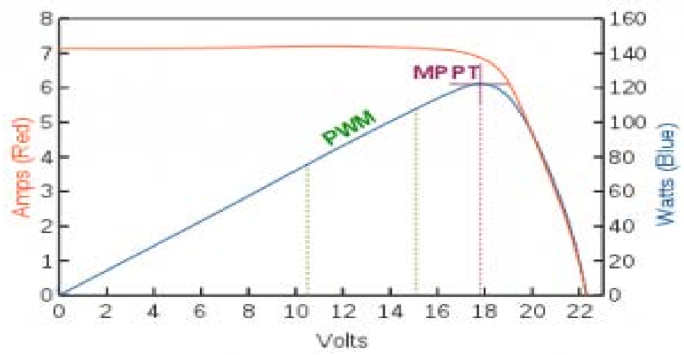

Figure 1: MPPT and PWM graph [1]

The maximum power point tracker (MPPT) is a device which converts power from DC to DC then ensures the support of the performance match between the solar panel voltage and the battery bank voltage. Therefore, the MPPT charge controller steps down high-power voltage from the solar panel to low voltage needed to charge a battery. This is illustrated in figure 2.

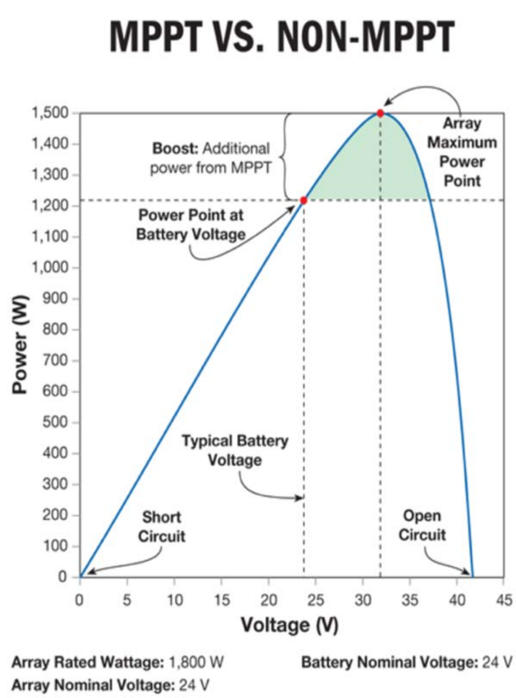

Figure 2: MPPT performance match between the solar panel and the battery [2]

2. Problem Statement

2.1. Problem Definition

There are several key features for high use of charge controllers. First, the multistage charging of batteries which saves the battery from being damaged. Second, the ability to change the power set to the batteries while considering the charge. This is significant to keep the battery healthier. Third, the reverse current protection capability prevents the solar panel from draining charge of the battery banks in the night when there is no power from the panel. In addition, it also disconnects the load when the battery is low and connects when it charges again. Finally, it displays the voltage of the battery bank and also the amount of charge from the panel. But having a solar charge controller with these features and high efficiency is very expensive to purchase, approximately $100-$200. Therefore, the goal of this project is to design a solar charge controller with a higher efficiency and a lower budget.

2.2. Client Requirements

Based on the functions of the devices being used and their technical specifications, the requirement of the project can be divided into two parts: device function and device technical specification. Each of these requirements is summarized below.

Device function:

- Based on MPPT algorithm
- LED indication for the state of charge
- LCD display for displaying voltages, current, and power
- USB charger port 5 V

- Automatic battery voltage recognition (12 V/24 V)

Device technical specification

- Battery: 55 AH
- Maximum load current: 10 A
- Open circuit voltage between 0 -12.5 V for 12 V system and 12.5 – 24.5 V for 24 V system
- Solar panel power 80 W
- Battery charge current = 5 A
- USB charge current: 0.5 A

Optional display features on LCD

- Charge time
- Battery charging percentage

3. Problem solution

3.1 Proposed Solution

3.1.1 Solar Charge Controller Block Diagram

An ADC is used for measuring analog voltages with a digital microcontroller - this is mainly used for interfacing analog sensors' output. Analog sensors convert some physical parameter (i.e. light intensity, temperature, humidity, etc.) into voltage dependency, that can be later measured using the ADC. A solar charge controller is used, which measures the voltage, current, and power. This is made with use of ADC and some external analog components. The use of the solar panel, battery, ADC, and external analog components can be represented in a solar charge block diagram as shown in figure 3.

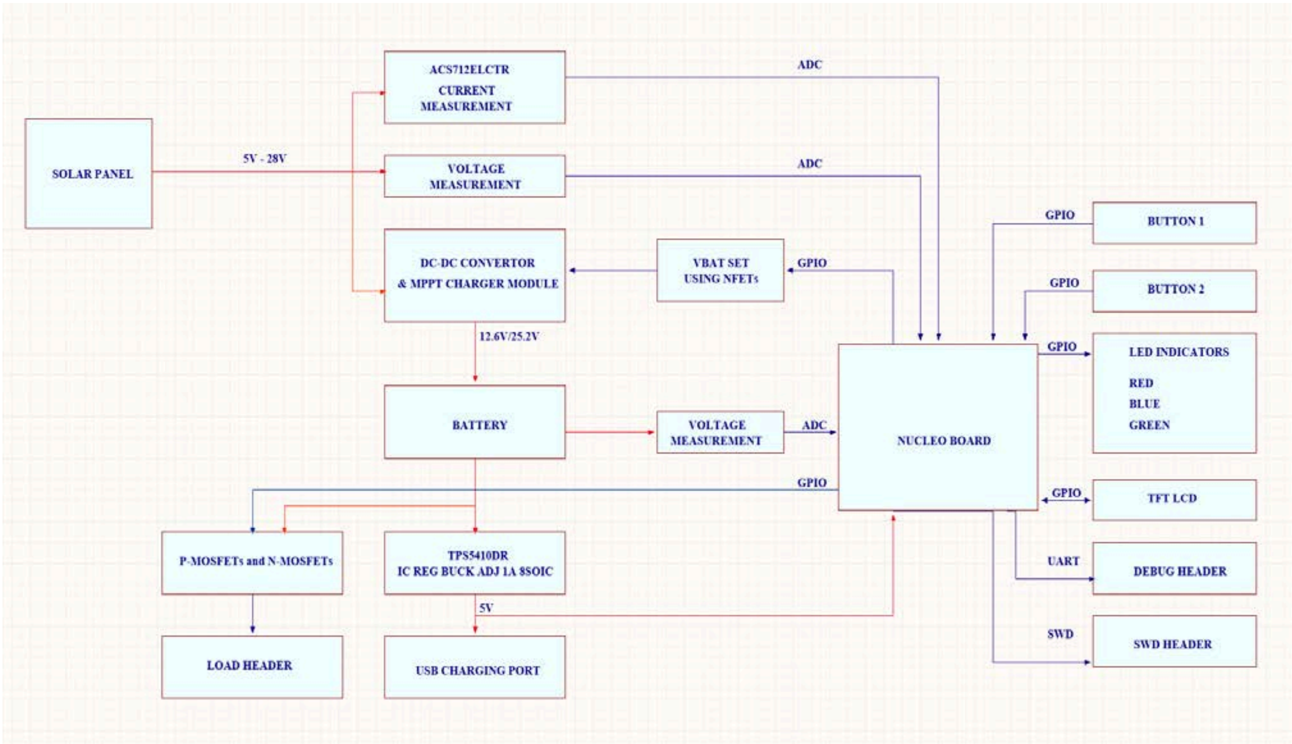

Figure 3: Solar charge block diagram solution

3.1.2 MPPT Charge Controller

Using a battery charge controller for solar power with maximum power point tracking is safer and gives a higher performance. A DC-DC buck converter is needed at the input if the solar panel produces a high voltage. The DC-DC buck converter is a step-down converter. The device regulates power input and output from a load. The converter steps down the power from a solar panel as it enters the battery or the load while at the same time stepping up power output from the load. The converter has two semiconductors, a transistor, and a diode which makes it possible to step down and step up power input and output. However, in order to avoid the risk of voltage ripples, the converter is fitted with supply side filters and load side filters which smooth out power flow. There are two DC-DC buck converters in this device. The first one is at the output of the solar panels. With high voltage solar panels, the output voltage may reach 50 V. The normal open output voltage is from 28 V – 38 V so the output of a DC-DC controller will be 24 V to match with all 12 V systems and 24 V systems. Output current needs to achieve 5 A to 8 A. Texas Instruments is a professional in this field and the most popular solution is BQ24650 [3]. Figure 4 shows a switch-mode charge controller. The second one is supply to USB port for charging. This DC-DC buck converter will regulate 12 V – 24 V to 5 V, and output current needs to achieve 2A.

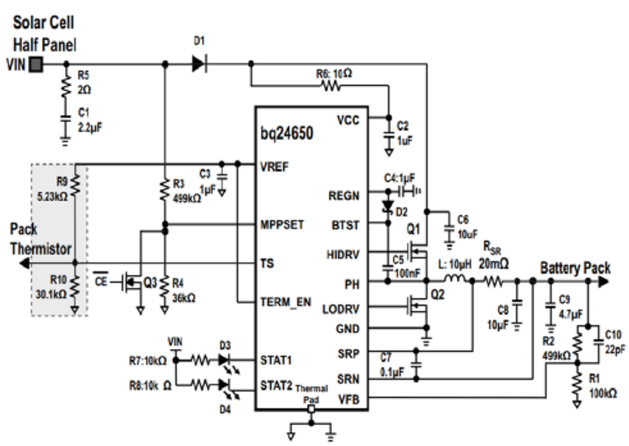

Figure 4: Switch-Mode Charger Controller (BQ24650) [3]

3.1.3 Voltage Regulator

A voltage regulator is an electronic device responsible for maintaining a steady voltage level. The device protects power machines like car engines, and laptops, among others, from fluctuating voltages by maintaining a constant flow of power entering the device. Voltage regulators may exist in various designs like the electronic or electromagnetic mechanism, negative feedback, or the feed-forward design, but the underlying principle is that they regulate DC or DC voltages from various sources. In computer devices, the regulators are used to smooth out and stabilize DC voltages in the processor and other elements. The device maintains a fixed voltage output regardless of the changes in the load or input voltage. The best choice to lower the voltage is using LDO (low-dropout regulator). LDO has fast transient response, low noise, is cheap, and will regulate from 5 V to 3.3 V. Output current of this circuit is 1 A to supply for microcontroller, LCD, LED indicators and other peripherals.

3.1.4 Voltage Measurement

There are 2 voltages that need to be measured; solar panel output voltage and battery voltage. This circuit uses resistor to divide the high voltage to low voltage that matches with the ADC Range of the microcontroller unit (MCU). MCU will read the divided voltage and calculate the solar panel output voltage and battery voltage then display it on the LCD. The only adjustment needed is for input voltage range to correspond to ADC input range (which is 0-3.3V in case of STM32). The input range of 0-28V can be shrunk down to 0-3.3V using a voltage divider. The voltage divider follows equations 1 and 2 as shown in figure 5.

$$Solar\ Voltage = \frac{V_{Solar} * R_2}{R_4 + R_2} \qquad \text{Eq [1]}$$

$$Battery\ Voltage = \frac{V_{Bat} * R_1}{R_3 + R_1} \qquad \text{Eq [2]}$$

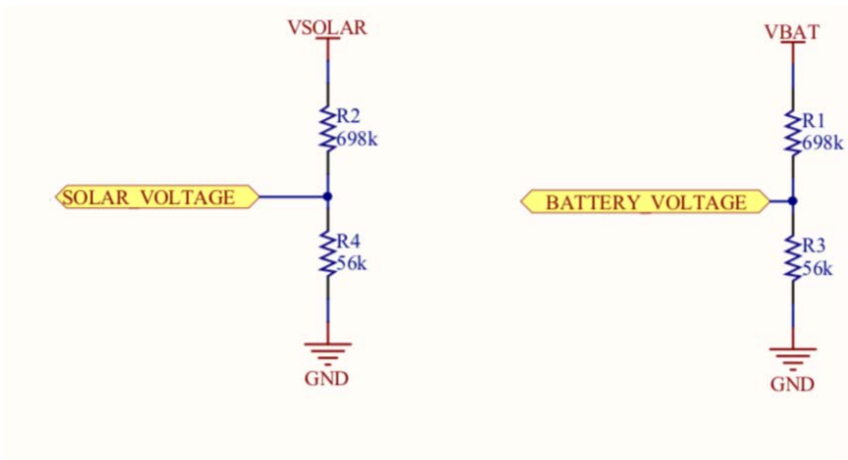

Figure 5: Voltage measurement

3.1.5 Current Measurement

Current cannot be directly measured with an ADC. Instead the ADC can be used to measure a voltage drop across a known resistor, and using Ohm's law it is possible to calculate current as

$$Current\ (I) = \frac{Voltage(V)}{Resistance\ (R)} \qquad \text{Eq [3]}$$

On the other side, if a resistor is inserted into the measured circuit it would create a voltage drop, which might alternate the circuit performance. This circuit uses a current sensor to measure the input current from the solar panel. A good solution here is ACS712 [4]; the maximum current that ACS712 can measure will be 30 A. The output of the current sensor is voltage. MCU reads this voltage and then calculates the current. Figure 6 shows a current sensor.

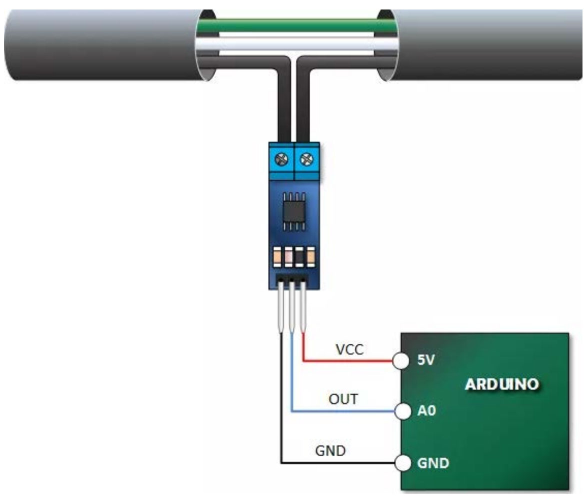

Figure 6: Current sensor (ACS712) [4]

3.1.7 DC Load Control

Load controllers are charge regulators used to prevent batteries from overcharging. Modern solar and other power devices are fitted with DC load control systems that automatically disconnect all loads when the power supply is not enough. The controllers may also be fitted in the loads or batteries to prevent overcharging or unlimited power supplies, which may damage the loads.

The DC load controller regulates the charge voltage from the solar system to the battery, as shown in figure 7. During the ON state when CTRL_LOAD logic = 1, no current flows through the third mosfit (Q_3). So, it will close, and since the first mosfit (Q_1) and the second mosfit (Q_2) are connected to Q_3 they will close too, and V_{BAT} come to output header on default state, $Q_1$and Q_2 open because G pin of $Q_1$and Q_2 is pulled up to V_{BAT}, and during the OFF state when CTRL_LOAD logic = 0, the current complete its path through Q_3, Q_2, and Q_1 so the circuit will open and V_{BAT} will come to output header on default state.

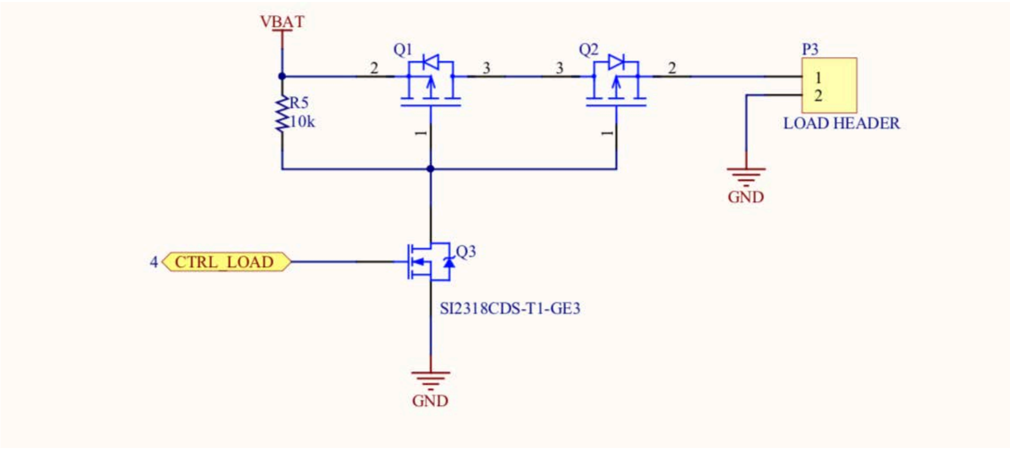

Figure7: DC load control circuit

3.1.8 Constraints

One of the constraints for this project is environmental. This was addressed by using DC-DC buck converter as mentioned previously in the report. Also, recycled materials were used on the project such as plastic. Another constraint was sustainability. The sustainability in manufacturing was increased by using a PCB circuit board. The final constraint is manufacturability. A 3D printed box and PCB was designed for manufacturing

3.1.9 Safety and Standards

IEEE standard was followed for the design of PCB [5]. For safety purposes, the project was designed to have overvoltage, overload, and lightning protection. Furthermore, the circuit is designed to have reserve power flow protection and short circuit protection.

3.2 Software design

These are the steps of the project code:

- Initialization:
  - Configure the system clock
  - Initialize all configured peripherals
  - Initialize GPIO for buttons, enable EXTI0 interrupt
- Create a screen to display the values
  - Set value
  - Create a simple style with ticker line width
- update data values
- Initialize timers for measuring microseconds and milliseconds, respectively
- Initialize ADC1 with Channels 0-2 on pins PA0, PA1 and PC1 (analog mode)
  - ADC1
  - Peripheral clock enable
  - ADC1 GPIO Configuration
  - PA0 ------> ADC1_IN0
  - PA1 ------> ADC1_IN1
  - PC1 ------> ADC1_IN2
  - PA5 ------> SPI1_SCK
  - PA6 ------> SPI1_MISO
  - PA7 ------> SPI1_MOSI
  - PA9 ------> USART1_TX
  - PA10 ------> USART1_RX
- initialize LCD on PB pins
  - Ser rotation of the screen - changes x0 and y0
  - Initialize LCD display
  - Enable LCD display
  - TURN ON DISPLAY
- Main loop:
  - Draw text on LCD to show current measurement mode
  - Check if measurement has been started
  - If yes, determine mode and perform ADC measurement:
  - Configure GPIO pin: BTN_TEST1_Pin
  - Configure GPIO for the LOAD controlee

    - Configure GPIO pins for the LEDs
    - Configure GPIO pins for the push buttons
    - Measure solar voltage in channel 0 voltage and recalculate the value.
    - Measure battery volt in channel 1 voltage and recalculate the value.
    - Measure current in channel 2 and recalculate the value.
    - Measure State charger if its 12 or 24
    - Power calculation
    - Calculate charger time
    - Display measured value or display “press start” message on LCD
    - Repeat after every 10ms
- Interrupt routine:
    - Determine which EXTI_PR flag has been set
    - Change the variables (mode or state of measurement) accordingly
    - Clear EXTI flag
- Void vChargerCtr_ADCInit(void) for ADC channel
- Void vReadChargerPara(void)
    - ADC_Value_Solar_Volt
    - ADC_Value_Solar_Curr
    - ADC_Value_Battery_Volt

3.3 Hardware design

Figure 8 shows the full schematic for the hardware design. A detailed diagram showing the connections is shown in Appendix A of this report. The top and the bottom layer of the PCB design used in this project is shown in figures 9 and 10. Sharp3D [6] software was used for the design of the 3D box. The top and the bottom view of the 3D box is shown in figure 11and 12.

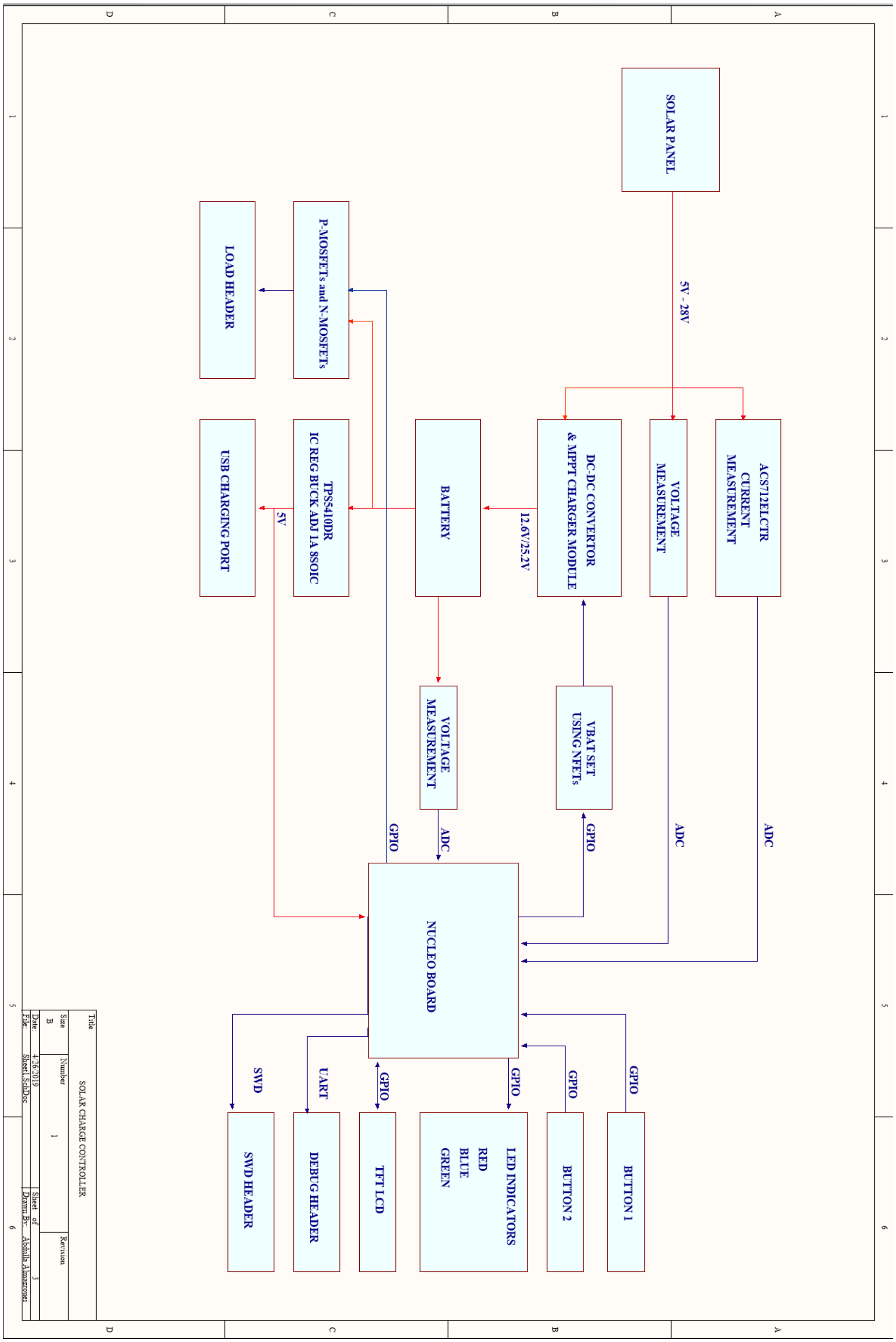

Figure 8: Full hardware schematic

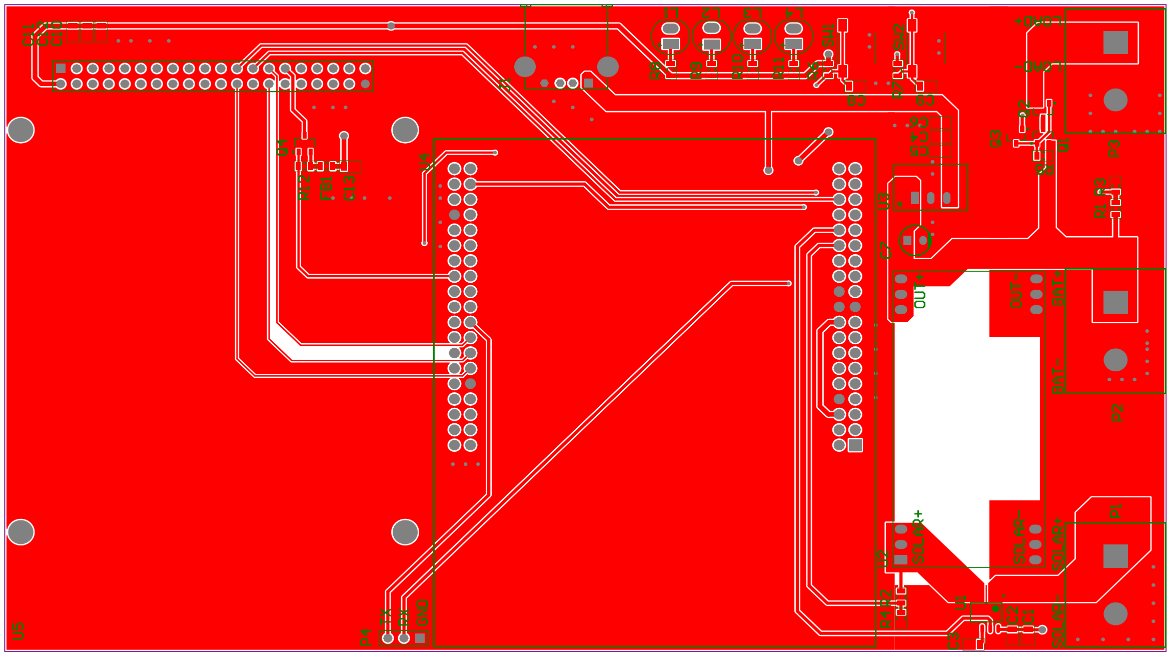

Figure 9: PCB top layer

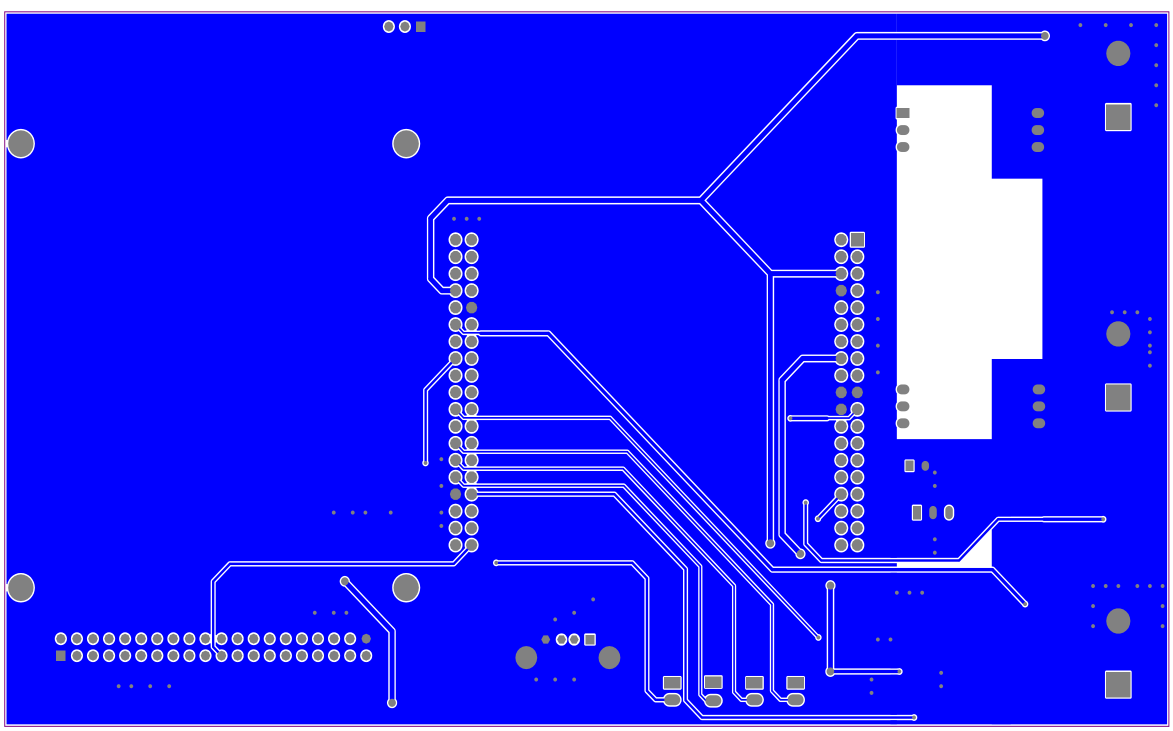

Figure 10: PCB bottom layer

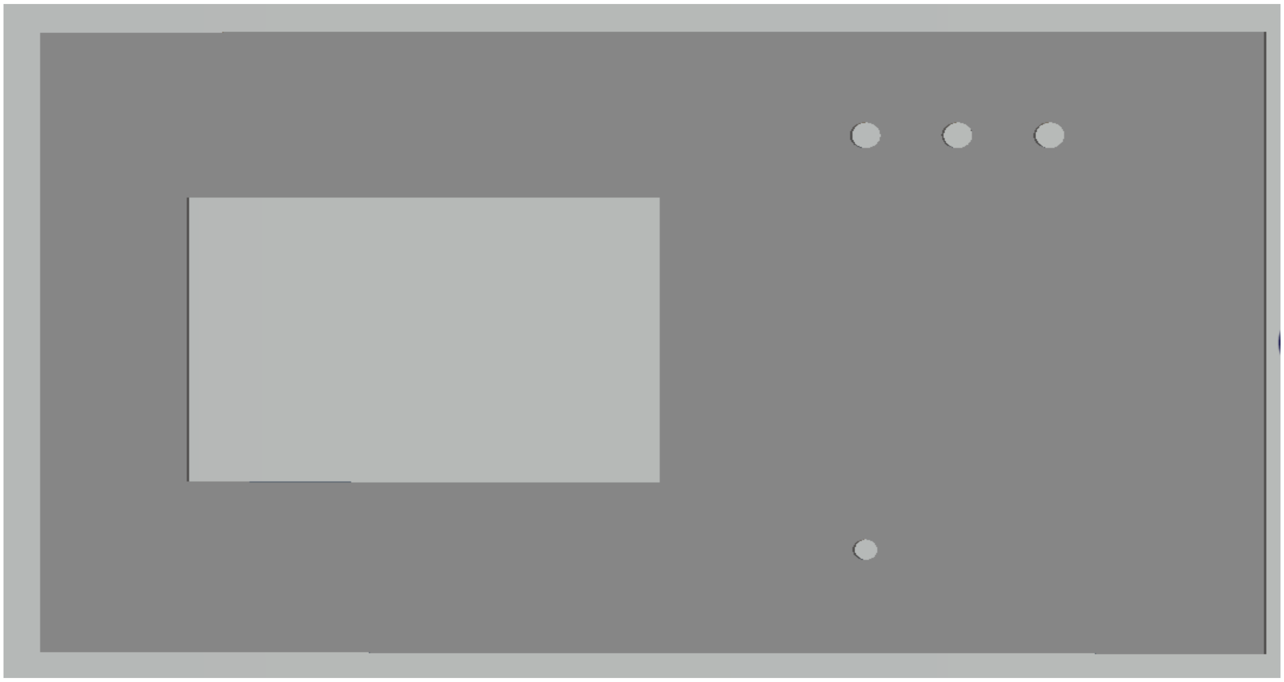

Figure 11: 3D box top part

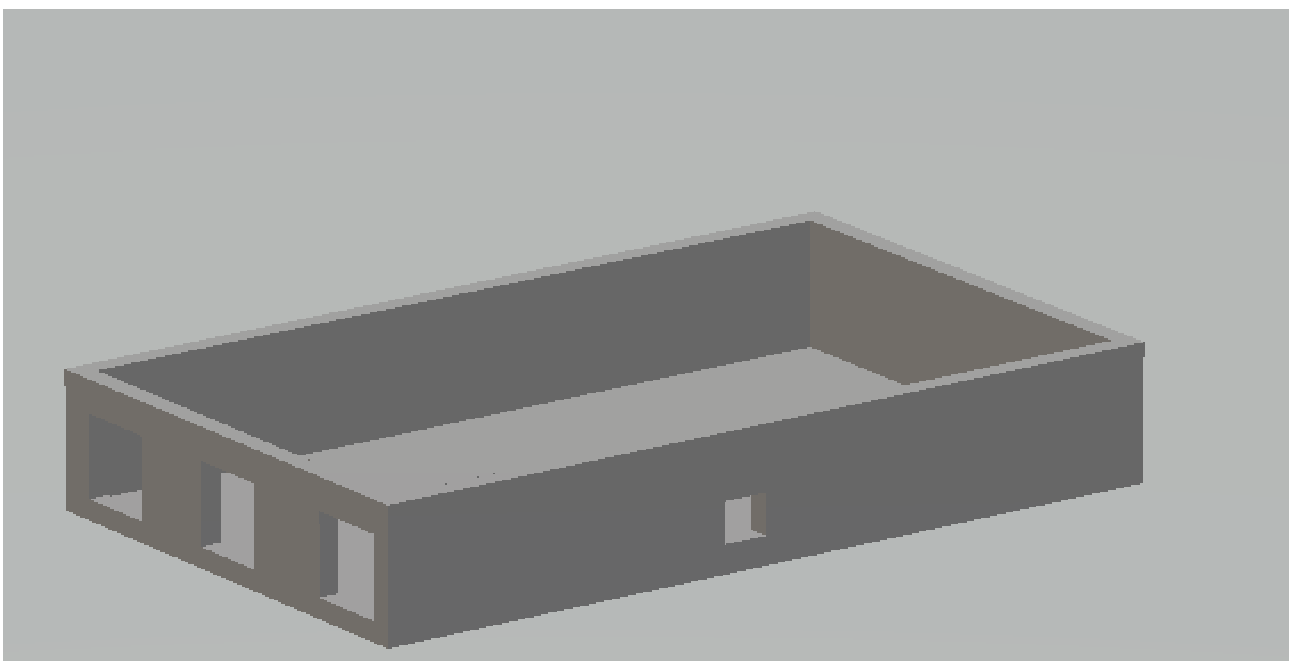

Figure 12: 3D box bottom part

3.4 Testing

Several testings are done throughout the semester to ensure that the project works. The microcontroller communication with the analog sensor is tested. Furthermore, the efficiency of the accuracy of display on the LCD is tested. Also, the microcontroller timers are tested to raise the fast connection of getting and displaying the information on the LCD. Table 1 shows the battery charging level and the corresponding voltage.

Table 1: Battery Charging Level

charge %	**12V battery**	**24V battery**
0%	11.80V	23.00V
25%	12.00V	23.50V
50%	12.20V	24.20V
75%	12.40V	24.75V
100%	12.70V	25.70V

4 Parts list and cost

4.1 Parts

Table 2 shows a list of categories used for this project with their quantities and specifications. The total number of quantities for this project is 51.

Table 2: Parts list

#	Category	Comment	Quantity
1	Capacitors	0.1µF	6
2	Capacitors	1µF	3
3	Capacitors	10pF	2
4	Capacitors	22µF	1
5	Capacitors	1000µF	1
6	BJT		1
7	USB		1
8	LED	Red	2
9	LED	Blue	1
10	LED	Green	1
11	Solar Connector		1
12	Battery Connector		1
13	Load header		1
14	MOSFET	NPN	2
15	MOSFET	PNP	1
16	Diode		1
17	Resistors	100k	2
18	Resistors	50k	2
19	Resistors	10k	4
20	Resistors	250	4
21	Switches		2
22	Sensor current		1
23	MPPT		1
24	DC-DC converter		1
25	Microcontroller	STM32	1
26	LCD	ER-TFTM032-3	1
Total			51

4.2 Costs

Table 3 shows list of categories with their designators, price per unit, and total price in USD. The total estimated budget for the project is $100, and the project ended up costing $65.

Table 3: Cost list

#	Category	Designator	price per unit in USD	total price in USD
1	Capacitors	7	0.5	1.5
2	BJT	1	0.2	0.2
3	USB	1	0.8	0.8
4	LEDs	4	0.5	2
5	Solar Connector	8	0.5	4
6	Battery Connector	1	0.5	0.5
7	Load header	1	0.5	0.5
8	MOSFET	3	0.3	0.9
9	Diode	1	0.3	0.3
10	Resistors	12	0.1	1.2
11	Switches	2	0.5	1
12	Sensor current	1	1.7	1.7
13	MPPT	1	0.7	0.7
14	DC-DC converter	1	6.1	6.1
15	STM32	1	14	14
16	TFT LCD	1	10	10
17	PCB	1	15	15
Total		51		64.9

5 Results and concisions

The final project satisfies all the client requirements. Most client specifications are met and addressed. Moreover, the circuit design is based on MPPT algorithm with 95% efficiency. The project is built to have high efficiency and low cost. Figure 13 shows the final design of the solar charge controller. For further improvement, the buck converter can be upgraded to buck-boost converter in order to charge batteries from lower voltage sources, also the solar charge controller is designed with extra push button, LED, and four pins on the PCB for futuristic using.

Figure 13: Final design

6 References

[1] Medi, N. (2018). *MPPT charge controller advantages compare to standard PWM*. [online] MEEE. Available at: https://meee-services.com/mppt-charge-controller-advantages-compare-standard-pwm/ [Accessed 7 Sep. 2018].

[2] URJOS. (2018). *MPPT Charge Controllers: What is MPPT and its advantages? - URJOS*. [online] Available at: https://urjos.com/solar-energy/mppt-charge-controllers-what-is-mppt-and-its-advantages/ [Accessed 13 Sep. 2018].

[3] Element Community. (2018). *Evaluation Module Synchronous, Switch-Mode, Battery Charge Controller for Solar Power*. [online] Available at: https://www.element14.com/community/docs/DOC-48798/l/bq24650-evaluation-module-synchronous-switch-mode-battery-charge-controller-for-solar-power [Accessed 12 Sep. 2018].

[4] Nagar, S. (2018). *Current Sensor Module*. [online] Research Gate. Available at: https://www.researchgate.net/figure/ACS712-Current-sensor-module_fig17_281865046 [Accessed 7 Sep. 2018].

[5] Muralikrishna, S. (2018). *An overview of digital circuit design and PCB design guidelines - An EMC perspective - IEEE Conference Publication*. [online] Ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/document/5154359 [Accessed 9 Nov. 2018].

[6] Sharp3D. 3D Modelling software. www.sharp3D.com[online] [Accessed 9 Feb. 2018].

Appendix A: Detailed Schematic

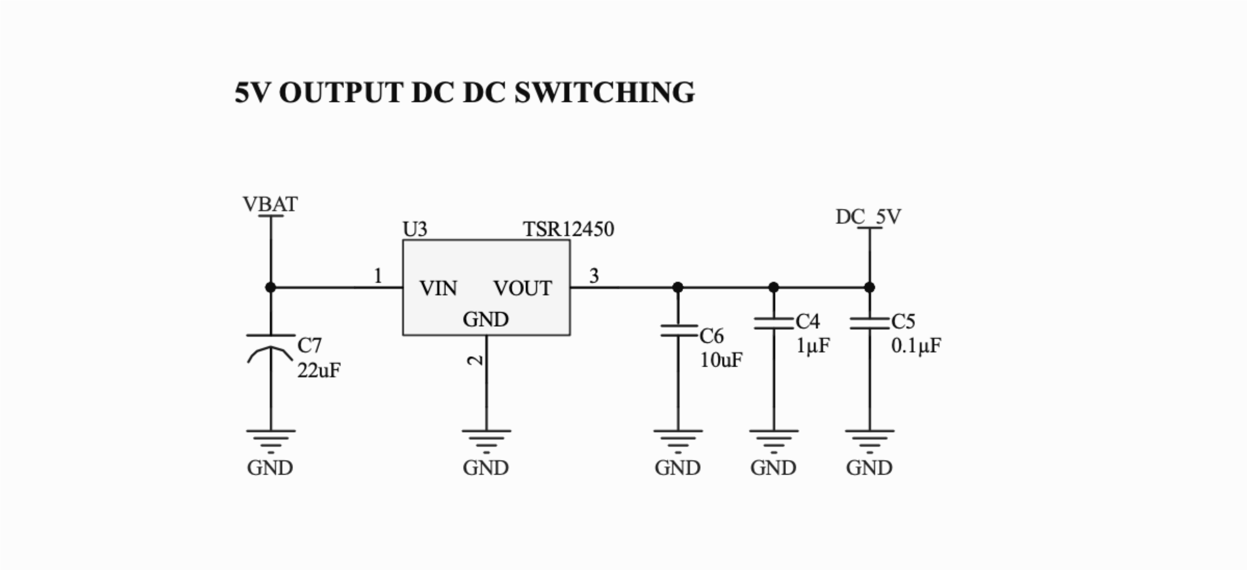

Figure 14: Detailed Schematic 1

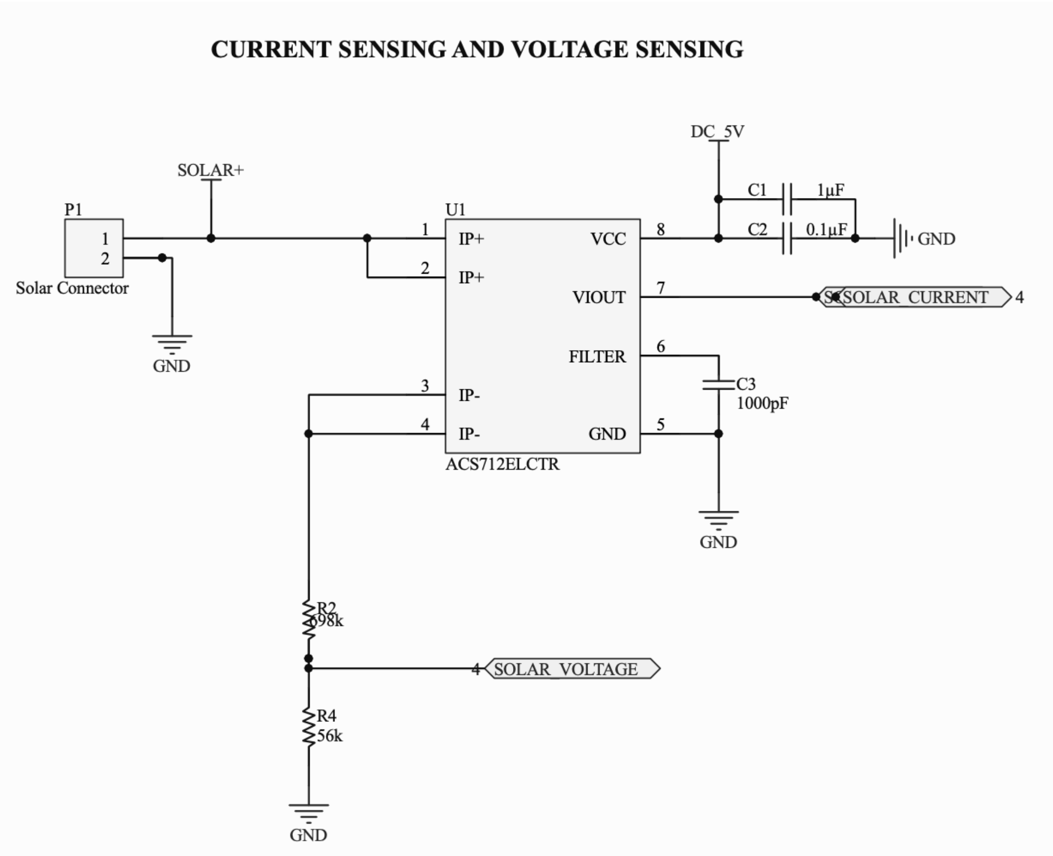

Figure 15: Detailed Schematic 2

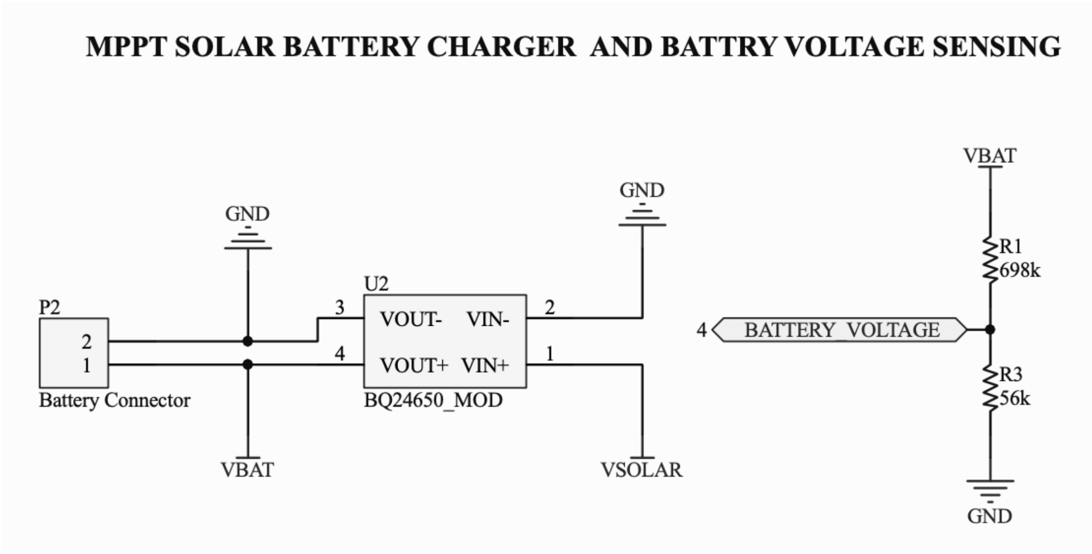

Figure 16: Detailed Schematic 3

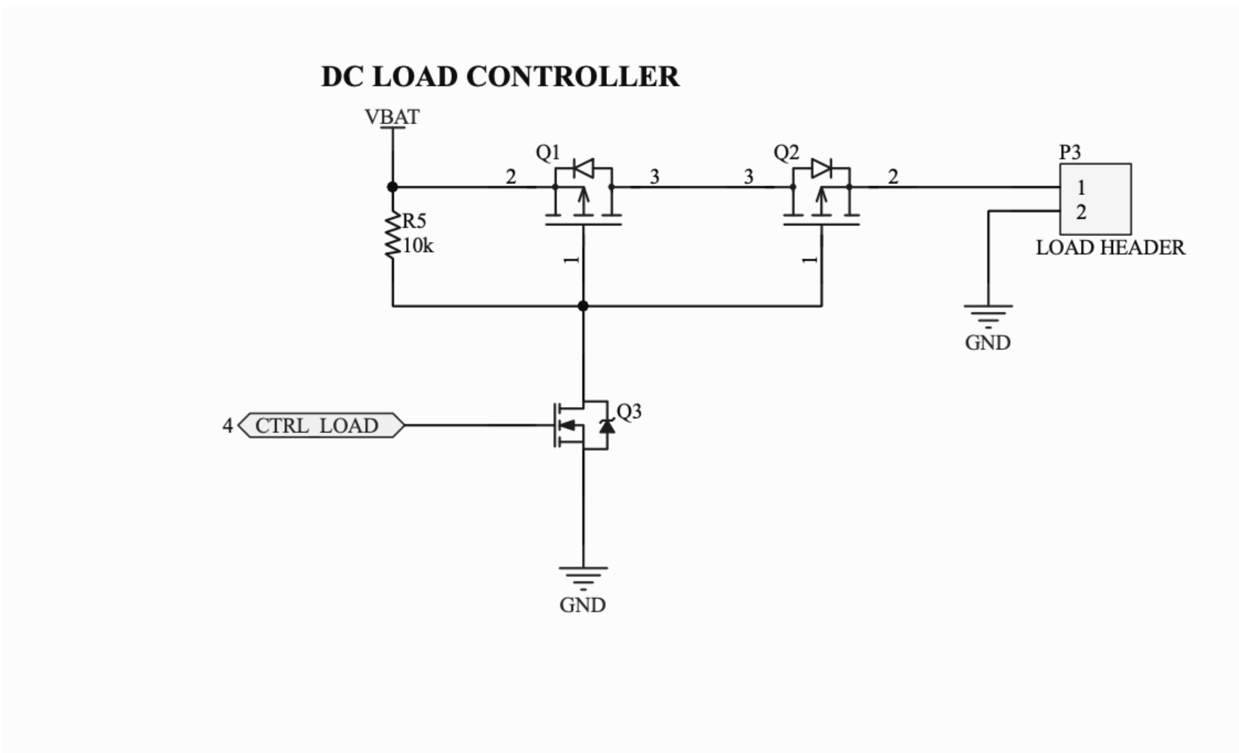

Figure 17: Detailed Schematic 4

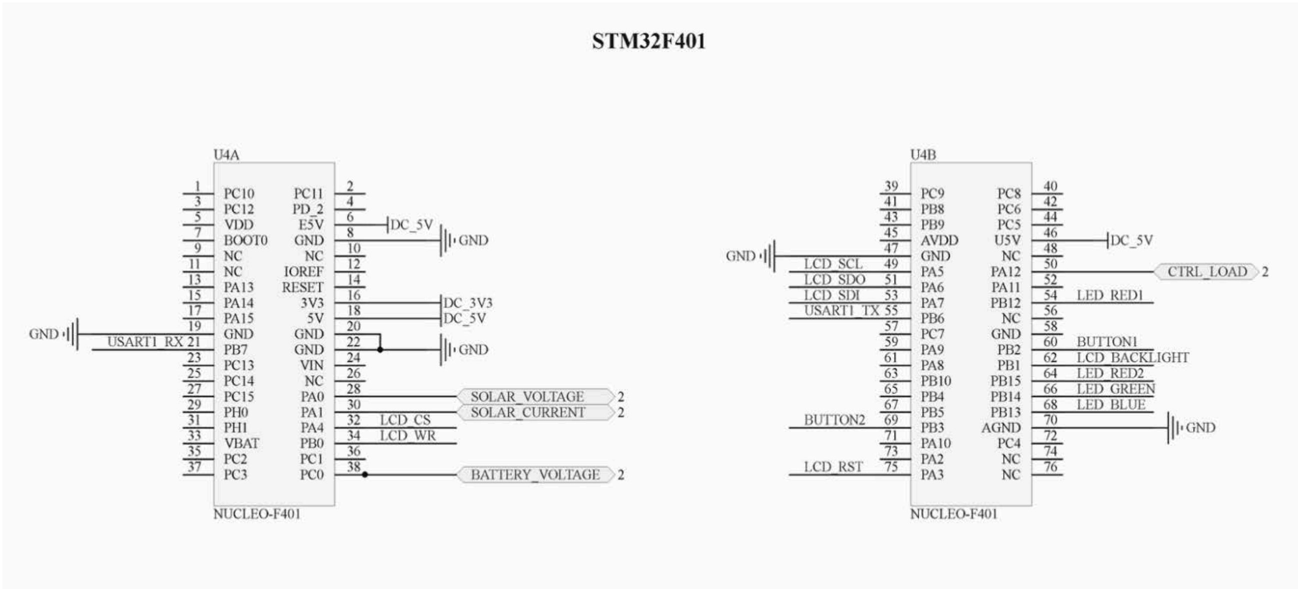

Figure 18: Detailed Schematic 5

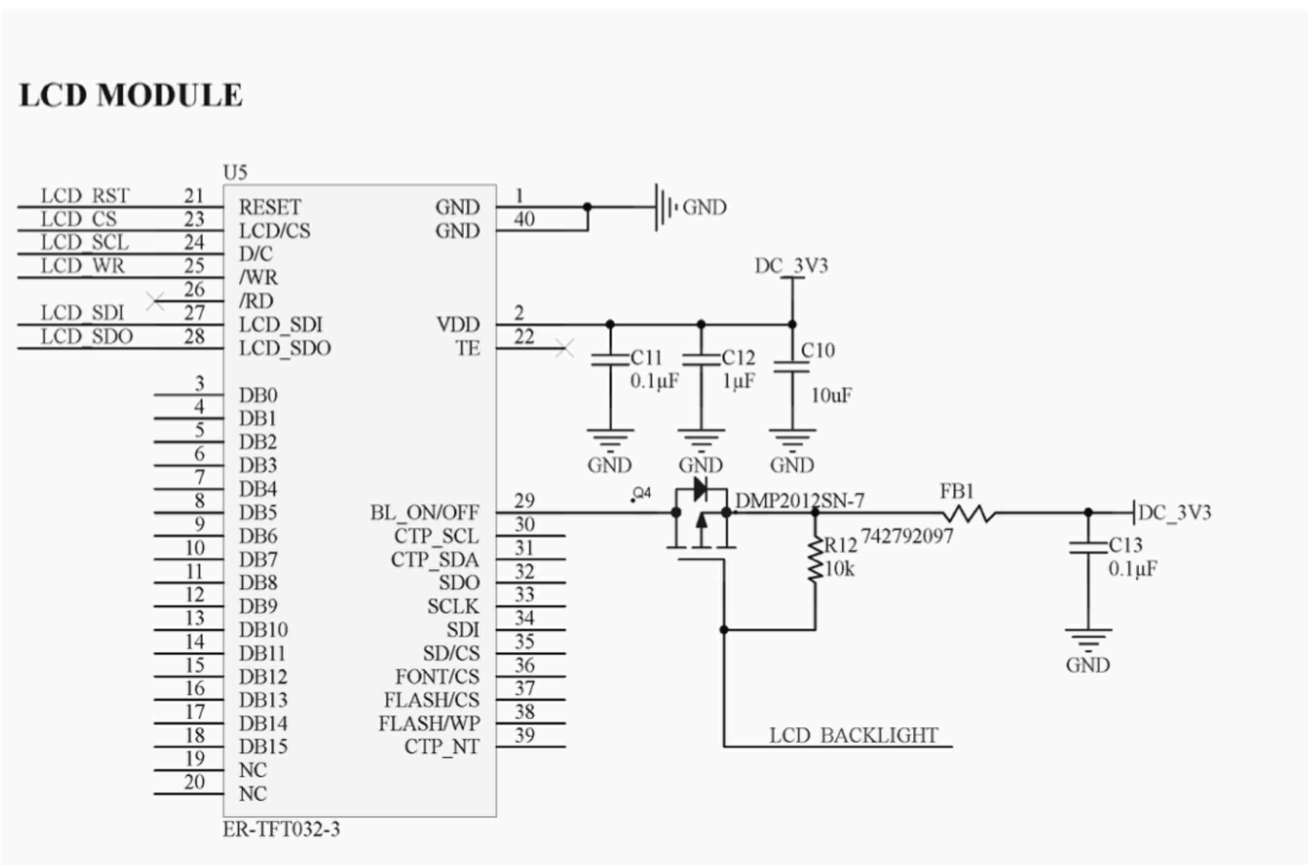

Figure 19: Detailed Schematic 6

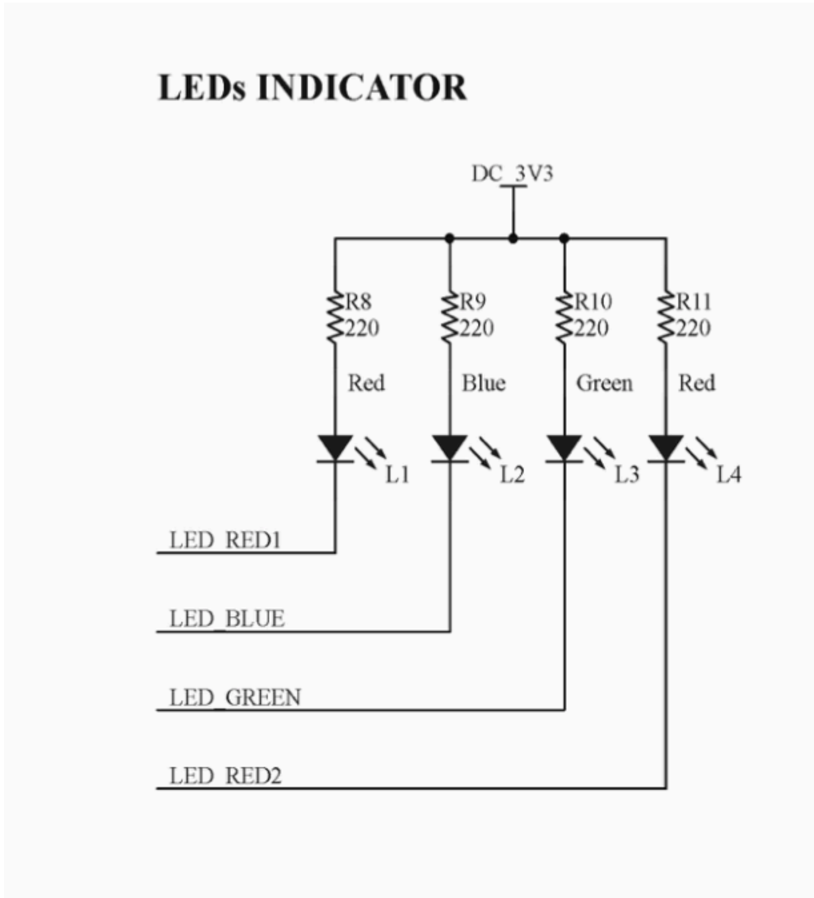

Figure 20: Detailed Schematic 7

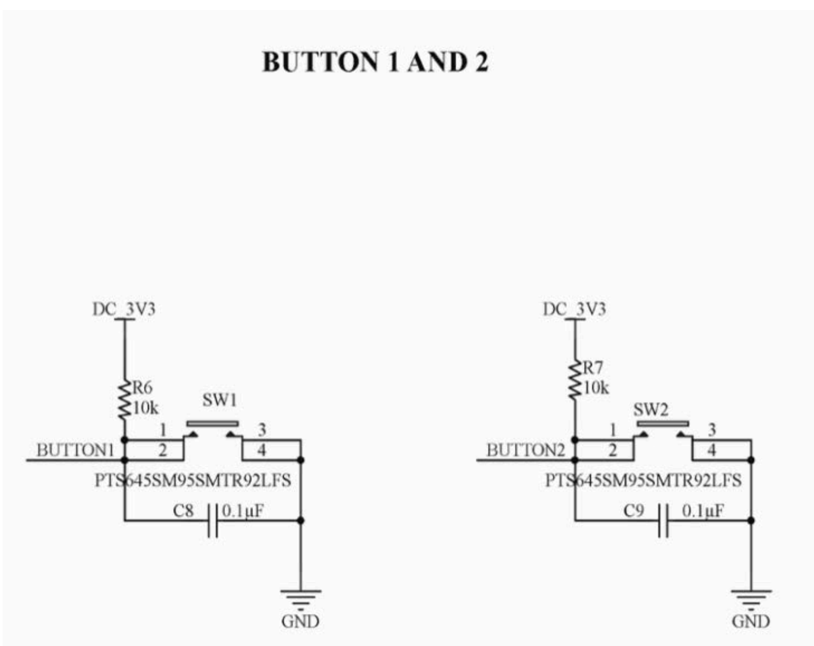

Figure 21: Detailed Schematic 8

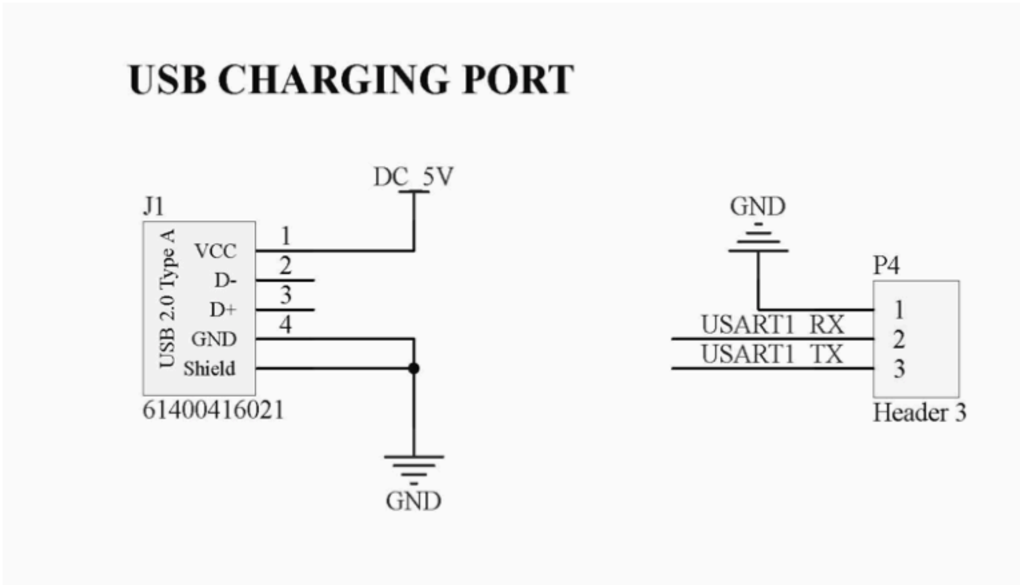

Figure 22: Detailed Schematic 9

Appendix B: Source code

```

/*******************************************************************************
 *
 *                          SOLAR CHARGE CONTROLLER
 *
 *
 ******************************************************************************/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "cmsis_os.h"
#include "lvgl/lvgl.h"
#include "ili9341Driver/ILI9341_STM32_Driver.h"
#include "ChargerCtr/ChargerCtr.h"

#define TypeValue;
#define ValueSolarVoltage;
#define ValueBatteryVoltage;
#define ValueBatteryCharge_State;
#define ValueChargeCurrent;
#define ValueChargeWatt;
#define ValueOutputState;
#define ValueOutputCurrent;

/*******************************************************************************
 *
 *                                         main
 *
 *
 ******************************************************************************/
static void vMain_LVinit(void);

int main(void)
{
    /* MCU Configuration----------------------------------------------------------*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();


    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    vMyUart_UARTInit();
    vMain_LVinit();
    vChargerCtr_ADCInit();

    /* Create the thread(s) */
    /* definition and creation of defaultTask */
    osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 256);
    defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);
    {

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
```

```
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, CTRL_LOAD_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOA, VBAT_SET_Pin, GPIO_PIN_SET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LED_RED1_Pin|LED_BLUE_Pin
                        |LED_GREEN_Pin|LED_RED2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : BTN_TEST1_Pin */
  GPIO_InitStruct.Pin = BTN_TEST1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BTN_TEST1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : VBAT_SET_Pin CTRL_LOAD_Pin */
  GPIO_InitStruct.Pin = VBAT_SET_Pin|CTRL_LOAD_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins :LED_RED1_Pin LED_BLUE_Pin
                         LED_GREEN_Pin LED_RED2_Pin */
  GPIO_InitStruct.Pin = LED_RED1_Pin|LED_BLUE_Pin
                        |LED_GREEN_Pin|LED_RED2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : BTN_1_Pin BTN_2_Pin */
  GPIO_InitStruct.Pin = BTN_1_Pin|BTN_2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  HAL_GPIO_WritePin(GPIOA, VBAT_SET_Pin, GPIO_PIN_SET);

}

static void vMain_LVinit(void)
{
    ILI9341_Init();
    ILI9341_Set_Rotation(4);
    lv_init();

    lv_disp_drv_t disp;
    lv_disp_drv_init(&disp);

    disp.disp_flush =   ILI9341_flush;
    disp.disp_fill  =   ILI9341_fill;
    disp.disp_map   =   ILI9341_map;

    lv_disp_drv_register(&disp);
    }


{
    //--------------------------- SOlar charger----------------------------------

    /*Create a screen*/
    lv_obj_t * scr = lv_obj_create(NULL, NULL);
    lv_scr_load(scr);

    static lv_style_t style_obj;
    lv_style_copy(&style_obj, &lv_style_plain);
    style_obj.body.grad_color = lv_style_plain_color.body.main_color;
    style_obj.body.grad_color = LV_COLOR_HEX(0x0000cc);
    style_obj.body.main_color = LV_COLOR_HEX(0x0000cc);
```

```
    lv_obj_t * obj1;
    obj1 = lv_obj_create(scr, NULL);
    lv_obj_set_size(obj1, LV_HOR_RES, 40);
    lv_obj_set_style(obj1, &style_obj);
    lv_obj_align(obj1, NULL, LV_ALIGN_IN_TOP_MID, 0, 0);

    static lv_style_t style_title;
    lv_style_copy(&style_title, &lv_style_plain);
    style_title.text.letter_space = 2;
    style_title.text.line_space = 1;
    style_title.text.color      =   LV_COLOR_HEX(0xffffff);

    lv_obj_t * title = lv_label_create(scr, NULL);
    lv_obj_set_style(title, &style_title);
    lv_label_set_text(title, "Solar Charge Controller");
    lv_obj_set_pos(title, lv_obj_get_width(lv_obj_get_parent(title)) / 2 - lv_obj_get_width(title) / 2 , 5);

    //o Battery type if it was 12V or 24V
    //o Solar panel voltage
    //o Battery voltage
    //o Battery charge state
    //o Charge current
    //o Charge watt
    //o Output state
    //o Output current
    //o Charge time
    static lv_style_t style_txt;
    lv_style_copy(&style_txt, &lv_style_plain);
    style_txt.text.letter_space = 2;
    style_txt.text.line_space = 1;
    style_txt.text.color      =   LV_COLOR_HEX(0x0000cc);

    static lv_style_t styleEror_txt;
    lv_style_copy(&styleEror_txt, &lv_style_plain);
    styleEror_txt.text.letter_space = 2;
    styleEror_txt.text.line_space = 1;
    styleEror_txt.text.color      =   LV_COLOR_HEX(0xcc0000);

    // For battery type value display
    lv_obj_t * BatteryType = lv_label_create(scr, NULL);
    lv_obj_set_style(BatteryType, &style_txt);
    lv_obj_set_pos(BatteryType, 0, 50);

    lv_obj_t * TypeValue = lv_label_create(scr, NULL);
    lv_obj_set_style(TypeValue, &style_txt);
    lv_obj_set_pos(TypeValue, 175, 50);

    // For Solar volt value display
    lv_obj_t * SolarVoltage = lv_label_create(scr, NULL);
    lv_obj_set_style(SolarVoltage, &style_txt);
    lv_obj_set_pos(SolarVoltage, 0 , 70);

    lv_obj_t * ValueSolarVoltage = lv_label_create(scr, NULL);
    lv_obj_set_style(ValueSolarVoltage, &style_txt);
    lv_obj_set_pos(ValueSolarVoltage, 175, 70);

    // For Battery volt value display
    lv_obj_t * BatteryVoltage = lv_label_create(scr, NULL);
    lv_obj_set_style(BatteryVoltage, &style_txt);
    lv_obj_set_pos(BatteryVoltage, 0 , 90);

    lv_obj_t * ValueBatteryVoltage = lv_label_create(scr, NULL);
    lv_obj_set_style(ValueBatteryVoltage, &style_txt);
    lv_obj_set_pos(ValueBatteryVoltage, 175, 90);

    // For State charger
    lv_obj_t * BatteryCharge_State = lv_label_create(scr, NULL);
    lv_obj_set_style(BatteryCharge_State, &style_txt);
    lv_obj_set_pos(BatteryCharge_State, 0 , 110);
```

```

    lv_obj_t * ValueBatteryCharge_State = lv_label_create(scr, NULL);
    lv_obj_set_style(ValueBatteryCharge_State, &style_txt);
    lv_obj_set_pos(ValueBatteryCharge_State, 175, 110);

    // For Char Current
    lv_obj_t * ChargeCurrent = lv_label_create(scr, NULL);
    lv_obj_set_style(ChargeCurrent, &style_txt);
    lv_obj_set_pos(ChargeCurrent, 0 , 130);

    lv_obj_t * ValueChargeCurrent = lv_label_create(scr, NULL);
    lv_obj_set_style(ValueChargeCurrent, &style_txt);
    lv_obj_set_pos(ValueChargeCurrent, 175, 130);

    // For Power
    lv_obj_t * ChargeWatt = lv_label_create(scr, NULL);
    lv_obj_set_style(ChargeWatt, &style_txt);
    lv_obj_set_pos(ChargeWatt, 0 , 150);

    lv_obj_t * ValueChargeWatt = lv_label_create(scr, NULL);
    lv_obj_set_style(ValueChargeWatt, &style_txt);
    lv_obj_set_pos(ValueChargeWatt, 175, 150);

    // For on off 12.5 or 24.6 V
    lv_obj_t * OutputState = lv_label_create(scr, NULL);
    lv_obj_set_style(OutputState, &style_txt);
    lv_obj_set_pos(OutputState, 0 , 170);

    lv_obj_t * ValueOutputState = lv_label_create(scr, NULL);
    lv_obj_set_style(ValueOutputState, &style_txt);
    lv_obj_set_pos(ValueOutputState, 175, 170);

    // For Out out of solar
    lv_obj_t * OutputCurrent = lv_label_create(scr, NULL);
    lv_obj_set_style(OutputCurrent, &style_txt);
    lv_obj_set_pos(OutputCurrent, 0 , 190);

    lv_obj_t * ValueOutputCurrent= lv_label_create(scr, NULL);
    lv_obj_set_style(ValueOutputCurrent, &style_txt);
    lv_obj_set_pos(ValueOutputCurrent, 175, 190);
    // For Charge time
    lv_obj_t * ChargeTime = lv_label_create(scr, NULL);
    lv_obj_set_style(ChargeTime, &style_txt);
    lv_obj_set_pos(ChargeTime, 0 , 210);

    lv_obj_t * ValueChargeTime = lv_label_create(scr, NULL);
    lv_obj_set_style(ValueChargeTime, &style_txt);
    lv_obj_set_pos(ValueChargeTime, 125, 255);

    lv_label_set_text(BatteryType,      "Battery Type");
    lv_label_set_text(SolarVoltage,     "Solar Voltage");
    lv_label_set_text(BatteryVoltage,     "Battery Voltage");
    lv_label_set_text(BatteryCharge_State,  "Battery Charge");
    lv_label_set_text(ChargeCurrent,    "Charge Current");
    lv_label_set_text(ChargeWatt,       "Charge Watt");
    lv_label_set_text(OutputState,      "Output State");
    lv_label_set_text(OutputCurrent,    "Output Current");
    lv_label_set_text(ChargeTime,       "Charge Time");
    // Set value
//    char buf[6];
//    sprintf(buf, "%2.2f", ADC_Value_t.fSolar_Volt);
//    lv_label_set_text(TypeValue,"24V");
//    lv_label_set_text(ValueSolarVoltage, buf);
//    lv_label_set_text(ValueBatteryVoltage,"24.05V");
//    lv_label_set_text(ValueBatteryCharge_State,"24V");
//    lv_label_set_text(ValueChargeCurrent,"4.5mA");
//    lv_label_set_text(ValueChargeWatt,"10W");
//    lv_label_set_text(ValueOutputState,"24V");
//    lv_label_set_text(ValueOutputCurrent,"24V");
    lv_label_set_text(ValueChargeTime,"0:00:00s");
```

```
    /*Create a simple style with ticker line width*/
    static lv_style_t style_lmeter1;
    lv_style_copy(&style_lmeter1, &lv_style_pretty_color);
    style_lmeter1.line.width = 2;
    style_lmeter1.line.color = LV_COLOR_SILVER;
    style_lmeter1.body.main_color = LV_COLOR_HEX(0x91bfed);         /*Light blue*/
    style_lmeter1.body.grad_color = LV_COLOR_HEX(0x04386c);         /*Dark blue*/

    /*Create the first line meter */
    lv_obj_t * lmeter;
    lmeter = lv_lmeter_create(lv_scr_act(), NULL);
    lv_lmeter_set_range(lmeter, 0, 100);                  /*Set the range*/
    lv_lmeter_set_value(lmeter, 0);                      /*Set the current value*/
    lv_lmeter_set_style(lmeter, &style_lmeter1);          /*Apply the new style*/
    lv_obj_set_size(lmeter, 60, 60);
    lv_obj_align(lmeter, NULL, LV_ALIGN_IN_BOTTOM_LEFT, 20, -20);

    /*Add a label to show the current value*/
    lv_obj_t * label;
    label = lv_label_create(lmeter, NULL);
    lv_label_set_text(label, "0%");
    lv_label_set_style(label, &lv_style_pretty);
    lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

//------------------------------ END ------------------------------------------
    int iCount = 0;
    char bufTypeValue[6];
    char bufValueSolarVoltage[6];
    char bufValueBatteryVoltage[6];
    char bufValueBatteryCharge_State[6];
    char bufValueChargeCurrent[6];
    char bufValueChargeWatt[6];
    char bufValueOutputState[6];
    char bufValueOutputCurrent[6];

    lv_label_set_text(TypeValue,"0V");
    lv_label_set_text(ValueSolarVoltage, "0V");
    lv_label_set_text(ValueBatteryVoltage,"0V");
    lv_label_set_text(ValueBatteryCharge_State,"12.6V");
    lv_label_set_text(ValueChargeCurrent,"0A");
    lv_label_set_text(ValueChargeWatt,"0W");
    lv_label_set_text(ValueOutputState,"OFF");
    lv_label_set_text(ValueOutputCurrent,"0A");
    /* Infinite loop */
    for(;;)
    {
        lv_task_handler();
        lv_tick_inc(1);
        osDelay(1);
        iCount++;
        vMain_BTNSTT();
        if(iCount > 1000)
        {
            iCount = 0;
            vReadChargerPara();

            // update data
            if(SolarChargerData_t.eTypePin == eUnknown)
            {
                lv_obj_set_style(TypeValue, &styleEror_txt);
            }
            else lv_obj_set_style(TypeValue, &style_txt);

            sprintf(bufTypeValue, "%s", SolarChargerData_t.cTypePin);
            lv_label_set_text(TypeValue, bufTypeValue);

            sprintf(bufValueSolarVoltage, "%2.2fV", ADC_Value_t.fSolar_Volt);
            lv_label_set_text(ValueSolarVoltage, bufValueSolarVoltage);

            sprintf(bufValueBatteryVoltage, "%2.2f", ADC_Value_t.fBattery_Volt);
```

```
            lv_label_set_text(ValueBatteryVoltage, bufValueBatteryVoltage);

            sprintf(bufValueBatteryCharge_State, "%2.2f",  SolarChargerData_t.fVoltCharge);
            lv_label_set_text(ValueBatteryCharge_State, bufValueBatteryCharge_State);

            sprintf(bufValueChargeCurrent, "%2.2fA", SolarChargerData_t.fCurrentCharge);
            lv_label_set_text(ValueChargeCurrent, bufValueChargeCurrent);

            sprintf(bufValueChargeWatt, "%2.1fW", SolarChargerData_t.fPower);
            lv_label_set_text(ValueChargeWatt, bufValueChargeWatt);

            if(cSttBtn == 0)
            {
        HAL_GPIO_WritePin(CTRL_LOAD_GPIO_Port, CTRL_LOAD_Pin, GPIO_PIN_RESET);
        lv_obj_set_style(ValueOutputState, &styleEror_txt);
        sprintf(bufValueOutputState, "%s", "OFF");
            }
            else
            {
                HAL_GPIO_WritePin(CTRL_LOAD_GPIO_Port, CTRL_LOAD_Pin, GPIO_PIN_SET);
                lv_obj_set_style(ValueOutputState, &style_txt);
                sprintf(bufValueOutputState, "%s", "ON");
            }

            lv_label_set_text(ValueOutputState, bufValueOutputState);

            sprintf(bufValueOutputCurrent, "%2.2fA", ADC_Value_t.fSolar_Curr);
            lv_label_set_text(ValueOutputCurrent, bufValueOutputCurrent);

  HAL_RCC_SYSCFG_CLK_ENABLE();
  HAL_RCC_PWR_CLK_ENABLE();

  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
}

void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(hadc->Instance==ADC1)
  {


  /*  ADC1 */
    /* Peripheral clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    /**ADC1 GPIO Configuration
    PA0-WKUP  ------> ADC1_IN0
    PA1       ------> ADC1_IN1
    PC0       ------> ADC1_IN2
    */
    GPIO_InitStruct.Pin = SOLAR_VOLT_Pin|SOLAR_CURR_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = BATT_VOLT_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  }

}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* hadc)
{

  if(hadc->Instance==ADC1)
  {
```

```
    __HAL_RCC_ADC1_CLK_DISABLE();

    /**ADC1 GPIO Configuration
    PA0-WKUP   ------> ADC1_IN0
    PA1        ------> ADC1_IN1
    PC0        ------> ADC1_IN2
    */
    HAL_GPIO_DeInit(GPIOA, SOLAR_VOLT_Pin|SOLAR_CURR_Pin);
    HAL_GPIO_DeInit(GPIOC, BATT_VOLT_Pin);
  }

}

void HAL_SPI_MspInit(SPI_HandleTypeDef* hspi)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(hspi->Instance==SPI1)
  {

    /* Peripheral clock enable */
    __HAL_RCC_SPI1_CLK_ENABLE();

    /**SPI1 GPIO Configuration
    PA5     ------> SPI1_SCK
    PA6     ------> SPI1_MISO
    PA7     ------> SPI1_MOSI
    */
    GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* SPI1 interrupt Init */
    HAL_NVIC_SetPriority(SPI1_IRQn, 5, 0);
    HAL_NVIC_EnableIRQ(SPI1_IRQn);

  }

}

void HAL_SPI_MspDeInit(SPI_HandleTypeDef* hspi)
{

  if(hspi->Instance==SPI1)
  {

    /* Peripheral clock disable */
    __HAL_RCC_SPI1_CLK_DISABLE();

    /**SPI1 GPIO Configuration
    PA5     ------> SPI1_SCK
    PA6     ------> SPI1_MISO
    PA7     ------> SPI1_MOSI
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7);

    /*interrupt  */
    HAL_NVIC_DisableIRQ(SPI1_IRQn);
  }

}

void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(huart->Instance==USART1)
```

```
  {
    /* Peripheral clock enable */
    __HAL_RCC_USART1_CLK_ENABLE();

    /**USART1 GPIO Configuration
    PA9     ------> USART1_TX
    PA10     ------> USART1_RX
    */
    GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART1;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  }

}

void HAL_UART_MspDeInit(UART_HandleTypeDef* huart)
{
  if(huart->Instance==USART1)
  {
    /* Peripheral clock disable */
    __HAL_RCC_USART1_CLK_DISABLE();

    /**USART1 GPIO Configuration
    PA9     ------> USART1_TX
    PA10     ------> USART1_RX
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9|GPIO_PIN_10);

  /* USER CODE END USART1_MspDeInit 1 */
  }

/* Includes ------------------------------------------------------------------*/
#include "ILI9341_STM32_Driver.h"
#include "stm32f4xx_hal_spi.h"
#include "stm32f4xx_hal_gpio.h"


/*
* Static Function
*/
static void ILI9341_SPI_Init(void);
static void ILI9341_SPI_Send(unsigned char SPI_Data);
static void ILI9341_Write_Command(uint8_t Command);
static void ILI9341_Write_Data(uint8_t Data);
static void ILI9341_Set_Address(uint16_t X1, uint16_t Y1, uint16_t X2, uint16_t Y2);
static void ILI9341_Reset(void);
static void ILI9341_Enable(void);
static void ILI9341_Draw_Pixel(uint16_t X,uint16_t Y,uint16_t Colour);

/* Global Variables ------------------------------------------------------------*/
volatile uint16_t LCD_HEIGHT = ILI9341_SCREEN_HEIGHT;
volatile uint16_t LCD_WIDTH  = ILI9341_SCREEN_WIDTH;

}
static void ILI9341_SPI_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    /* GPIO Ports Clock Enable */
      __HAL_RCC_GPIOA_CLK_ENABLE();
      __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, LCD_CS_PIN|LCD_RST_PIN, GPIO_PIN_RESET);//CS OFF

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, LCD_DC_PIN|LCD_BL_PIN, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, LCD_BL_PIN, GPIO_PIN_SET);
```

```
    /*Configure GPIO pins : LCD_RST_Pin LCD_CS_Pin */
    GPIO_InitStruct.Pin = LCD_CS_PIN|LCD_CS_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /*Configure GPIO pins : LCD_WR_Pin LCD_BL_Pin LED_RED1_Pin LED_BLUE_Pin
                           LED_GREEN_Pin LED_RED2_Pin */
    GPIO_InitStruct.Pin = LCD_DC_PIN|LCD_BL_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);


    MX_SPI1_Init();
}

/*Send data (char) to LCD*/
static void ILI9341_SPI_Send(unsigned char SPI_Data)
{
    //HAL_SPI_Transmit_DMA(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size)
    HAL_SPI_Transmit(HSPI_INSTANCE, &SPI_Data, 1, 1);
}

/* Send command (char) to LCD */
static void ILI9341_Write_Command(uint8_t Command)
{
    HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LCD_DC_PORT, LCD_DC_PIN, GPIO_PIN_RESET);
    ILI9341_SPI_Send(Command);
    HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_SET);
}

/* Send Data (char) to LCD */
static void ILI9341_Write_Data(uint8_t Data)
{
    HAL_GPIO_WritePin(LCD_DC_PORT, LCD_DC_PIN, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_RESET);
    ILI9341_SPI_Send(Data);
    HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_SET);
}

/* Set Address - Location block - to draw into */
static void ILI9341_Set_Address(uint16_t X1, uint16_t Y1, uint16_t X2, uint16_t Y2)
{
    ILI9341_Write_Command(0x2A);
    ILI9341_Write_Data(X1>>8);
    ILI9341_Write_Data(X1);
    ILI9341_Write_Data(X2>>8);
    ILI9341_Write_Data(X2);

    ILI9341_Write_Command(0x2B);
    ILI9341_Write_Data(Y1>>8);
    ILI9341_Write_Data(Y1);
    ILI9341_Write_Data(Y2>>8);
    ILI9341_Write_Data(Y2);

    ILI9341_Write_Command(0x2C);
}

/*HARDWARE RESET*/
static void ILI9341_Reset(void)
{
    HAL_GPIO_WritePin(LCD_RST_PORT, LCD_RST_PIN, GPIO_PIN_RESET);
    HAL_Delay(200);
    HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_RESET);
    HAL_Delay(200);
    HAL_GPIO_WritePin(LCD_RST_PORT, LCD_RST_PIN, GPIO_PIN_SET);
```

```
}

/*Ser rotation of the screen - changes x0 and y0*/
void ILI9341_Set_Rotation(uint8_t Rotation)
{
    uint8_t screen_rotation = Rotation;

    ILI9341_Write_Command(0x36);
    HAL_Delay(1);

    switch(screen_rotation)
  {
    case SCREEN_VERTICAL_1:
      ILI9341_Write_Data(0x40|0x08);
      LCD_WIDTH = 240;
      LCD_HEIGHT = 320;
      break;
    case SCREEN_HORIZONTAL_1:
      ILI9341_Write_Data(0x20|0x08);
      LCD_WIDTH  = 320;
      LCD_HEIGHT = 240;
      break;
    case SCREEN_VERTICAL_2:
      ILI9341_Write_Data(0x80|0x08);
      LCD_WIDTH  = 240;
      LCD_HEIGHT = 320;
      break;
    case SCREEN_HORIZONTAL_2:
      ILI9341_Write_Data(0x40|0x80|0x20|0x08);
      LCD_WIDTH  = 320;
      LCD_HEIGHT = 240;
      break;
    default:
      //EXIT IF SCREEN ROTATION NOT VALID!
      break;
  }
}

/*Enable LCD display*/
static void ILI9341_Enable(void)
{
    HAL_GPIO_WritePin(LCD_RST_PORT, LCD_RST_PIN, GPIO_PIN_SET);
}

/*Initialize LCD display*/
void ILI9341_Init(void)
{
    ILI9341_Enable();
    ILI9341_SPI_Init();
    ILI9341_Reset();

    //TURN ON DISPLAY
    ILI9341_Write_Command(0x29);

    }
/*******************************************************************************
 *                    void vChargerCtr_ADCInit(void)
 *                           Init ADC channel
 *
 *
*******************************************************************************/
void vChargerCtr_ADCInit(void)
{
  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
conversion)
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
```

```
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ENABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 3;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_480CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 2;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_2;
  sConfig.Rank = 3;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
}
/*******************************************************************************
 * Function    : void vReadChargerPara(void)
 * Description  : ADC_Value_t.fSolar_Volt
                  ADC_Value_t.fSolar_Curr
                  ADC_Value_t.fBattery_Volt

*******************************************************************************/

void vReadChargerPara(void)
{
    //
    HAL_ADC_Start(&hadc1);

    HAL_ADC_PollForConversion(&hadc1,100);
    ADC_Value_t.fSolar_Volt = HAL_ADC_GetValue(&hadc1);

    HAL_ADC_PollForConversion(&hadc1,100);
    ADC_Value_t.fSolar_Curr = HAL_ADC_GetValue(&hadc1);

    HAL_ADC_PollForConversion(&hadc1,100);
    ADC_Value_t.fBattery_Volt = HAL_ADC_GetValue(&hadc1);

    HAL_ADC_Stop(&hadc1); // stop the adc
```

```
    HAL_Delay(200);

    ADC_Value_t.fSolar_Volt = (( ADC_Value_t.fSolar_Volt * (3.3/4096.0) ) * (56+698)) / 56;

    ADC_Value_t.fSolar_Curr = ( ADC_Value_t.fSolar_Curr * (3300/4096.0) - 2500) / 185;
    if (ADC_Value_t.fSolar_Curr < -5)
        ADC_Value_t.fSolar_Curr = -5;

    ADC_Value_t.fBattery_Volt = ((ADC_Value_t.fBattery_Volt * (3.3/4096.0) ) * (56+689)) / 56;

    // Find type of pin 12V or 24V
    if(ADC_Value_t.fBattery_Volt >= 24)
    {
        SolarChargerData_t.eTypePin = eAcid24V;
        sprintf(SolarChargerData_t.cTypePin, "%s", "24V");
    }
    else
    {
        if( (ADC_Value_t.fBattery_Volt >=12) && (ADC_Value_t.fBattery_Volt <=15))
        {
            SolarChargerData_t.eTypePin = eAcid12V;
            sprintf(SolarChargerData_t.cTypePin, "%s", "12V");
        }
        else
        {
            SolarChargerData_t.eTypePin = eUnknown;
            sprintf(SolarChargerData_t.cTypePin, "%s", "None");
            printf("--- ChargerCtr: Typepin Unkown \r\n");
        }
    }

    switch (SolarChargerData_t.eTypePin)
  {
    case eAcid12V:
      SolarChargerData_t.fVoltCharge = 12.6; //deffault in hardware
      SolarChargerData_t.fCurrentCharge = 5;//deffault in hardware
      HAL_GPIO_WritePin(GPIOA, VBAT_SET_Pin, GPIO_PIN_SET);
      break;
    case eAcid24V:
      SolarChargerData_t.fVoltCharge = 25.2;//deffault in hardware
      HAL_GPIO_WritePin(GPIOA, VBAT_SET_Pin, GPIO_PIN_RESET);
      SolarChargerData_t.fCurrentCharge = 5;//deffault in hardware
      break;
    default:
      SolarChargerData_t.fVoltCharge = 0;
      SolarChargerData_t.fCurrentCharge = 0;
      break;
  }

    // Charger Pin cal Power
    SolarChargerData_t.fPower = SolarChargerData_t.fVoltCharge * SolarChargerData_t.fCurrentCharge;

    printf("--- ChargerCtr: Solar_Volt     is %2.2f V\r\n", ADC_Value_t.fSolar_Volt);
    printf("--- ChargerCtr: Solar_Curr     is %2.2f A\r\n", ADC_Value_t.fSolar_Curr);
    printf("--- ChargerCtr: Battery_Volt   is %2.2f V\r\n", ADC_Value_t.fBattery_Volt);


    printf("--- ChargerCtr: Type of pin    is %d \r\n", SolarChargerData_t.eTypePin);
    printf("--- ChargerCtr: Volt Charge    is %2.2f V\r\n", SolarChargerData_t.fVoltCharge);
    printf("--- ChargerCtr: Current Charge is %2.2f A\r\n", SolarChargerData_t.fCurrentCharge);
    printf("--- ChargerCtr: Power          is %2.2f W\r\n", SolarChargerData_t.fPower);
}

/*********************************************************************************
 *              void vCalTimeCharge(void)
 *              Calculate charger time
 *
 *
 ********************************************************************************/
void vCalTimeCharge(void)
{
    int iPowerof12V = 12*55*60; // 12V 55Ah
    SolarChargerData_t.iTimeChar = iPowerof12V / SolarChargerData_t.fPower;
}


    ILI9341_Set_Rotation(SCREEN_VERTICAL_1);
}
```