ENERGY MANAGEMENT SYSTEM MODELING OF DC DATA CENTER WITH HYBRID ENERGY SOURCES USING NEURAL NETWORK

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Khalid Althomali

February 2017

COMMITTEE MEMBERSHIP

TITLE: Energy Management System Modeling of DC Data Center with Hybrid Energy Sources Using Neural Network

AUTHOR: Khalid Althomali

DATE SUBMITTED: February 2017

COMMITTEE CHAIR: Taufik, Ph.D.

Professor of Electrical Engineering

COMMITTEE MEMBER: Ahmad Nafisi, Ph.D.

Professor of Electrical Engineering

COMMITTEE MEMBER: Ali O. Shaban, Ph.D.

Professor of Electrical Engineering

ABSTRACT

Energy Management System Modeling of DC Data Center with Hybrid Energy Sources Using Neural Network

Khalid Althomali

As data centers continue to grow rapidly, engineers will face the greater challenge in finding ways to minimize the cost of powering data centers while improving their reliability. The continuing growth of renewable energy sources such as photovoltaics (PV) system presents an opportunity to reduce the long-term energy cost of data centers and to enhance reliability when used with utility AC power and energy storage. However, the inter-temporal and the intermittency nature of solar energy makes it necessary for the proper coordination and management of these energy sources.

This thesis proposes an energy management system in DC data center using a neural network to coordinate AC power, energy storage, and PV system that constitutes a reliable electrical power distribution to the data center. Software modeling of the DC data center was first developed for the proposed system followed by the construction of a lab-scale model to simulate the proposed system. Five scenarios were tested on the hardware model and the results demonstrate the effectiveness and accuracy of the neural network approach. Results further prove the feasibility in utilizing renewable energy source and energy storage in DC data centers. Analysis and performance of the proposed system will be discussed in this thesis, and future improvement for improved energy system reliability will also be presented.

Keywords: DC Data Centers, Renewable Energy Source, Neural Network, Hybrid Energy

ACKNOWLEDGMENTS

This thesis becomes a reality with the support and help of many individuals. I would like to extend my sincere thanks to all of them.

I would like to express my sincere gratitude to my advisor, Dr. Taufik for conveying his knowledge and expertise in this Master Thesis. I appreciate his constant encouragement to motivate me to do my very best. Thank you for all the times you were there when I needed you.

I would also like to thank Dr. Shaban, Dr. Nafisi, and Dr. Smilkstein for their constant mentoring throughout my Master's Program at Cal Poly. For their willingness to share their knowledge and for always having their doors open.

I am highly indebted to Mr. Anang Tjahjono, for his kind support and guidance. His vast knowledge and passion for Electrical Engineering have influenced me to continue refining my knowledge and expertise.

My Thanks and appreciations also go to my colleagues and people who have willingly helped me out with their abilities.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1: Introduction

In the late 19th century, Thomas Edison and Nikola Tesla waged the war of the currents where opposing views of the way electricity should be transmitted were battled. Edison established direct current (DC) to be the standard in the United States in the early age of electricity. However, at higher voltages, DC transmission revealed its difficulties. As a result, power plants at the time were only able to provide power to individual neighborhoods or small areas of a city. It became a challenge to provide electricity to rural areas because the power plants need to be at close proximity to prevent significant voltage drop. As the distance of the conductor increases, its resistance becomes higher which consequently results in a higher voltage drop. To solve this issue, Tesla proposed his idea of alternating the direction of current giving rise to the term alternating current (AC).

At the end, Edison's DC system proved to be inefficient and costly while Tesla's AC system became increasingly popular. The more prevalent use of AC owed greatly to Tesla's invention of poly-phase induction machine with a rotating magnetic field and an earlier invention on Transformer. With the two equipment, it was then possible to step the AC voltage produced by a three-phase generator to a much higher voltage, further allowing electrical power to be transmitted long distance while minimizing line losses.

As technology has developed in the past decades, in particular on solid-state devices and power electronics, so has the push toward the use of DC electricity again. This has also been coupled with the increasing use of renewable energy but in particular the solar energy through solar panels that inherently produce DC power. In the United States alone, the more prevalent use of solar photovoltaic installations is evidenced by the 24% increased PV installations between quarter 1 of 2015 and quarter 1 of 2016 as shown in Figure 1-1 [1]. In addition, Figure 1-2 shows that last year solar exceeded natural gas capacity in the US for the first time, and by the first quarter of 2016 solar rose to 64% of all electric generation capacities in the United States [1]. Therefore, it makes sense to see that solar PVs will present the potential of where DC electrical system may find its new home and implementations.

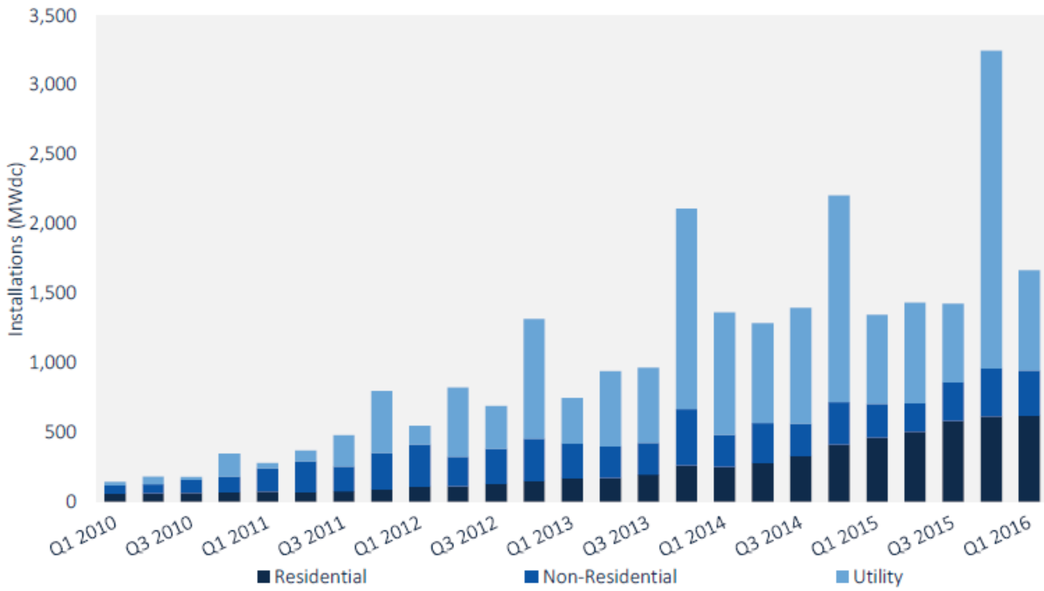

Figure 1-1 Annual U.S. Solar PV Installations

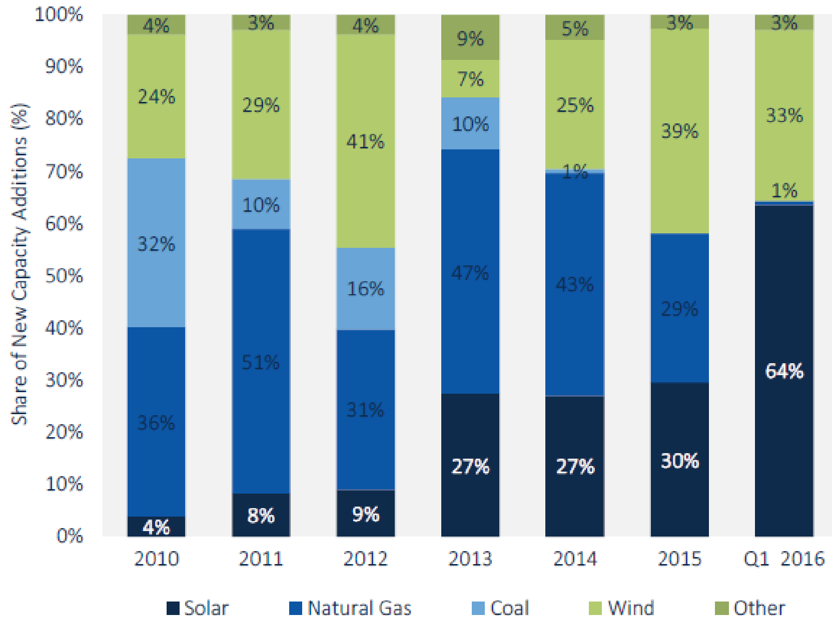

Figure 1-2 Share of New U.S. Electric Generating Capacity Additions

Another reason for the coming back of DC is that DC internally powers most electronic devices used at homes such as cell phone and tablet chargers, laptop adapters, etc. With the existing AC system, the available AC voltage must go through AC to DC conversion process that causes power loss. ABB is estimating saving from using DC instead of AC in buildings about 10% to 20% [2]. Additionally, the emerging and the fast growth in LED light bulb technology is another factor to coming back to DC. LED light bulbs phased out the traditional incandescent lamps, and it is inherently run on DC. This is estimated to save millions of kWh in the United States alone [3]. When comparing the LED light bulbs to the traditional incandescent lamps, LEDs typically use about 25% to 80% less than the incandescence lamps [4]. The benefit of LEDs includes longer operation life since they can last about 3 to 25 times longer [4]. As indicated in the United States Department of Energy website, the table below compares the incandescent with energy efficient bulbs.

Table 1-1 Comparison Between Different Light Bulbs [4]

Comparisons between Traditional Incandescents, Halogen Incandescents, CFLs, and LEDs

	60W Traditional Incandescent	43W Energy-Saving Incandescent	15W CFL		12W LED	
			60W Traditional	43W Halogen	60W Traditional	43W Halogen
Energy $ Saved (%)	–	~25%	~75%	~65%	~75%-80%	~72%
Annual Energy Cost*	$4.80	$3.50	$1.20		$1.00	
Bulb Life	1000 hours	1000 to 3000 hours	10,000 hours		25,000 hours	

One industry sector that has shown interest in taking advantage of DC electricity to improve energy efficiency is Data Center [5]. With the continued rapid growth in data centers, the demand for processing power will significantly increase. Such increase in power demand is related to the fact that there are typically tens of thousands of servers and their energy consumption is in raise. It is reported that Google alone has more than 900,000 servers [6], Meanwhile, Microsoft's Chicago data center as another example contains over 300,000 servers [7]. To further understand the scale of the problem, assuming each server consumes 200 watts per year, one million servers require 200 megawatts per year. When building data centers, usually they are priced by megawatt, and it is around $10 million per megawatt, and that will bring the cost to $2 billion. That was even before powering the data centers [8].

In 2013, the United States data centers consumed an estimated 91 billion kWh of electricity [9]. This amount of power is equivalent to an annual output power of 34 large 500-megawatt coal power plants, and it is enough to power all of the New York City households twice over a year [9]. By the year 2020, the power consumption is expected to increase to roughly 140 billion kWh annually, the equivalent output of 50 power plants with a total cost of $13 billion a year [9].

In the study of Integrated Approach to Data Center Power Management [7], it is demonstrated that 35% of the total cost of the data centers account for its energy, and the cost of this energy is competing with the cost of the computer servers. One reason for this high cost is that overall system efficiency for power usage of a typical data center is around 50% [7]. Such inefficiency, and thus the opportunity to improve the figure to reduce the energy cost, has brought back the century-old battle of AC versus DC for electrical power distribution in data centers. The AC supporters suggest that data centers maintain the AC system and improve efficiency by implementing high-efficiency UPS system. Those who believe that DC is the way to go, insist that the use of DC reduces the number of conversion stages along the power chain, and hence increases the overall efficiency.

Chapter 2: AC and DC Data Centers Power System Configuration

2.1 General Overview

Data centers consist of servers, cooling equipment, and power distribution equipment. They all come together to create a large system that consumes enormous amounts of energy which affect the system efficiency. In a typical data center, the computer load contributes to less than half the total energy consumption with the rest being lost in the UPS and cooling [10]. It has been estimated that the total energy consumption of a typical data center is at 40 TWh in 2005 in the United States, and 120 TWh worldwide [11]. Due to such tremendous amount of energy usage, the EPA in 2007 represented a report to the congress that included a recommendation to reduce the energy consumption in data centers [12]. The industry quickly responded and attempted to design more efficient servers, power supplies, and alternative ways of cooling. However, the overall complexity of the data center system poses a great challenge to find an efficient way to deliver power from the main generator to the load with less energy losses.

To help reduce energy loss, a few approaches for electrical power distribution in data centers have been used with some showing better efficiency than others. For example, data centers in North America typically use 480 V/ 600 V AC power distribution topology. This topology has been known to yield a less efficient system due to the relatively many numbers of conversions needed in order to feed a DC source (e.g. the server). This problem gets even worse when renewable energy is put into the mix. Therefore, industry and researchers have been investigating alternative ways to achieve a more efficient power distribution system in data centers. The following sections review literature that explain different topologies of power delivery architectures in data centers.

2.2 AC Powered Data Centers

As previously mentioned, in North America the traditional way to power a data center is through AC power distribution as shown in Figure 2-1 [12]. The 480 V AC connects into a UPS to produce DC voltage needed to charge energy storage device such as batteries. This power needs to be inverted back to AC voltage to feed the AC bus; hence, the process is called the double conversion mode. The AC voltage is then stepped down to 208 V AC in the power distribution unit (PDU). The next step of conversion happens in the power supply unit where the AC voltage is rectified to yield DC voltage which then goes through DC-DC converter stage to distribute the DC power to typically 12 V DC. This DC power supplies the main DC loads consisting of processors, memory, and storage.

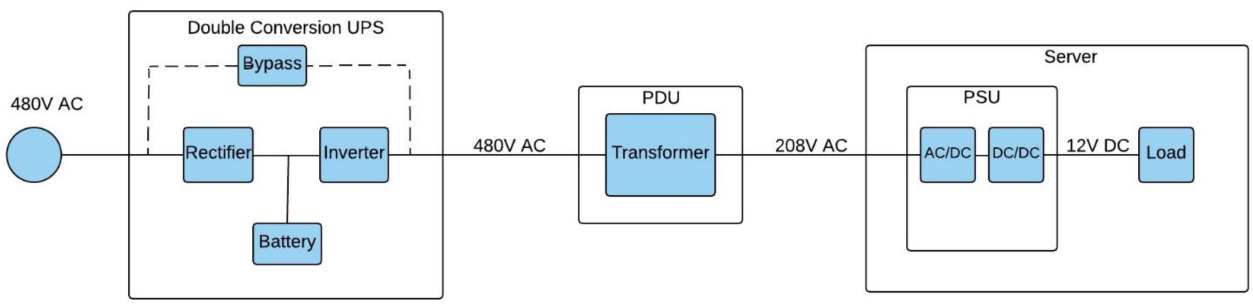

Figure 2-1. Typical 480V AC data center power system configuration

To achieve higher operating efficiency, data center manufacturers have developed some variations of the AC power distribution for data centers as illustrated below.

2.2.1 Bypass Filter (Eco-mode)

In this AC topology, manufacturers try to reduce the number of conversion in the typical 480V AC data center architecture by using a bypass filter. The bypass filter connects the AC source and the transformer as shown in Figure 2-2. This attempt eliminates the conversion step that is normally done by the inverter known as the eco-mode. This results in efficiency improvement; however, system reliability is affected. The load is no longer isolated from the power source and voltage regulation previously provided by the inverter no longer exists. To overcome these issues, a synchronous circuit may be used to ensure voltage regulation and high system reliability [12].

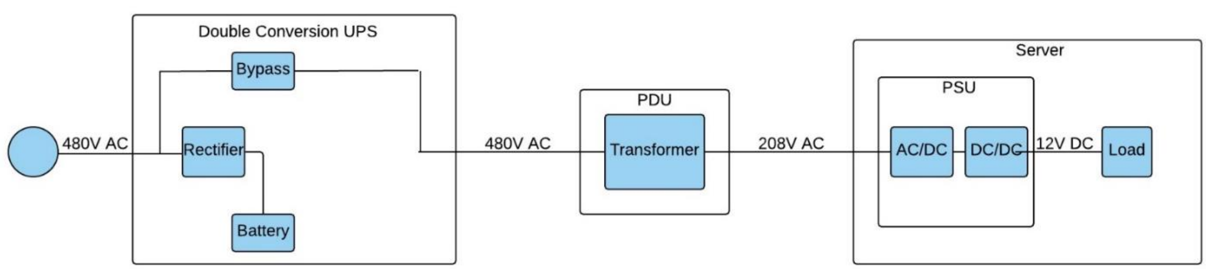

Figure 2-2. 480V AC data center power system configuration, power bypasses the inverter (ech-mode)

2.2.2 Delivering Higher Voltages to the Load

Another approach to increasing efficiency is to remove the phase voltage from the UPS (277 V AC instead of the 208 V AC coming from the transformer) and deliver a higher voltage to the load as illustrated in Figure 2-3 [12]. By doing this, the step-down conversion that is done by the power transformer in the power distribution unit is eliminated. However, this approach introduces new current harmonics in the system that will affect the efficiency gain. On top of that, the higher voltage presents a higher risk for IT workers.

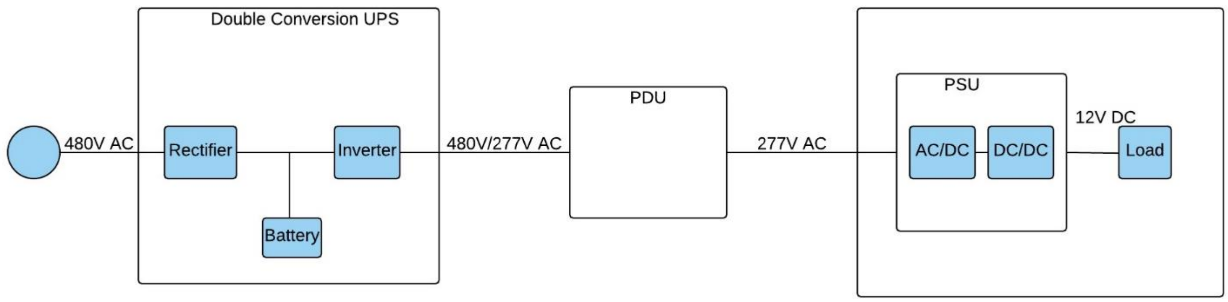

Figure 2-3. 480V/ 277V AC data center power system configuration

2.2.3 Modular Scalable AC UPS

Another method to increase the efficiency of AC data centers is a modular scalable AC UPS as depicted in Figure 2-4 [13]. The traditional double conversion AC UPS is usually built with future demand growth in mind. Data centers are typically lightly loaded and the IT loading factor is low. Therefore, the power converters will operate at lower efficiency which decreases the system's efficiency. The modular topology aims to combat this under-utilization of resources by keeping the working modules of an AC UPS at close to maximum load while retaining the flexibility to turn-on or turn-off online/offline modules as IT load/demand increases; hence, maximizing efficiency throughout the data center's life cycle [13].

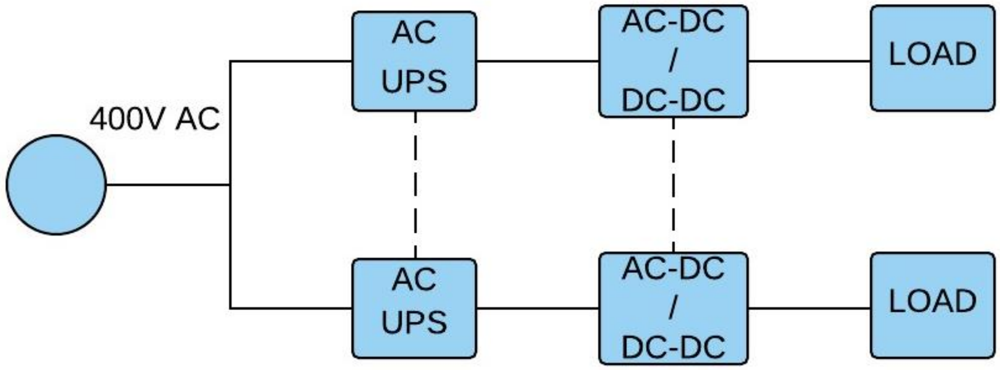

Figure 2-4. 400V AC data center power system configuration with scalable AC UPS

2.3 DC Powered Data Centers

From the system perspective, the UPS inverter that converts the DC power to AC and the PDU rectifier that converts it back to DC are only present in the architecture because electrical power is being distributed to the data center in AC form. For a DC topology, those conversion stages are duly taken out of the picture and hence may increase the overall system efficiency.

2.3.1 48V DC Data Center

One of the first DC data centers is the 48V DC data center used by telecommunication companies [14]. Here, the 480V AC is rectified to 48V DC and wired to the PDU from the DC UPS as shown in Figure 2-5. Although this method is more efficient than their AC counterparts due to the fewer conversion stages, the relatively low 48V DC voltage produces higher current in the power system and remains as a major limitation of the topology. In order to solve the problem, a much larger voltage of 400V DC was introduced. In comparison, a 400V DC topology requires cables that are 15 times smaller than a 48V topology [15] to transmit a 100kW of power.

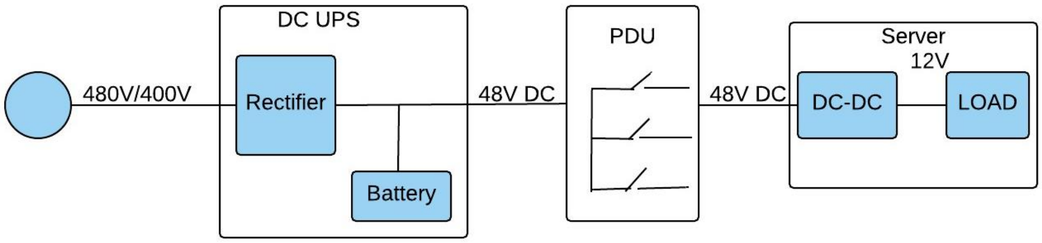

Figure 2-5. 48V DC data center power system configuration

2.3.2 400V DC Data Center

In order to further improve the efficiency of DC data centers, the DC voltage used to distribute the power is made larger at 400V DC as shown in Figure 2-6. Each of the conversion devices has been optimized in separate studies: a front-end three-phase buck-rectifier topology converting 480V AC to 400V DC and utilizing SiC (Silicon Carbide) devices in place of Si (Silicon) with 98.5% efficiency [12]. These single-stage conversion devices lead to smaller and cheaper data centers and more efficient power consumption with 28% efficiency improvement over typical AC power distribution [16].

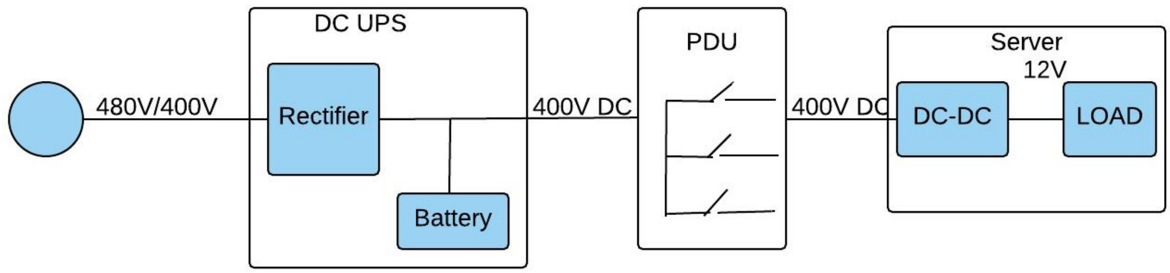

Figure 2-6. 400V DC data center power system configuration

2.3.3 Modular Scalable DC UPS

In this approach, the operation is similar to the 400V DC distribution. However, further efficiency improvement was introduced by using DC power for the scalable UPS. This approach eliminates the double conversion in AC UPS. In addition, DC UPS loss is reduced by 2% to 3% due to a single stage of conversion [12]. Figure 2-7 depicts the modified 400V DC architecture with modular scalable DC UPS.

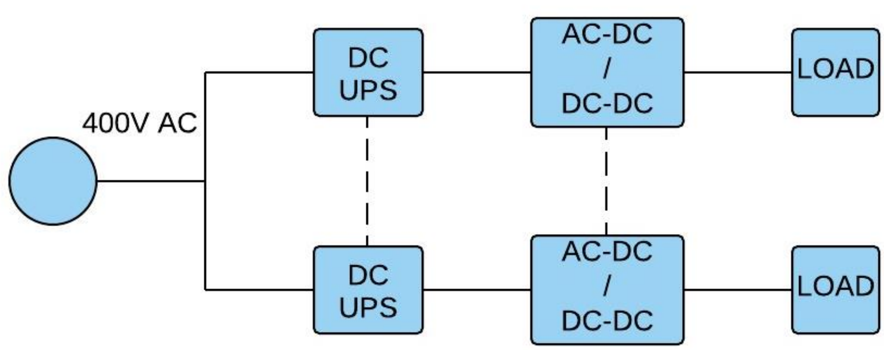

Figure 2-7. DC data center power system configuration with scalable DC UPS

2.4 Data Centers Powered by Renewable Energy

As explained above, the 400 Volt DC topology proved to be much more efficient than the traditional AC topology. As such, there has been a rise in the use of renewable resources to help power Data Centers. This trend has become popular among the largest Tech companies in the United Sates. For example, Apple has brought major changes to the functionality of their Data Centers. As stated in their 2016 Environment Responsibility Report [17], their Data Centers located in California, Nevada, North Carolina, and Oregon are 100% powered by Renewable Energy resources. These Data Centers are all powered by a mix of Solar PV, Biogas fuel cells, NC green Power, and Wind. To highlight Apple uses approximately 64% of Solar PV to power their North Carolina Datacenter and approximately 80% of Solar PV in Nevada. Ultimately, Apple's goal is to archive a 100% use of renewable energy resources to run all of their Data centers by the year 2017.

Because Solar PV energy is irregular throughout the day, it becomes a challenge to fully rely on it as the main power source. One approach to overcome this issue is to use batteries to store the energy or to maintain another backup source such as a utility grid. Further, another approach is to adapt the load and its energy demand to the generation from the renewable energy [18]. In this thesis, the approach applied is to use a grid as a backup source of the Solar PV and the use of batteries as an additional backup in the case of any power failures. The software would be developed to help control the shifting between these power sources based on the load demand.

Chapter 3: Design Requirements

System Overview The proposed topology for this thesis is influenced by the use of high voltage DC powered data center with the integration of renewable energy source as reported in references [12,16,17,18]. For the proposed topology, a PV system as the renewable energy source will supply the DC electrical power directly to the data center as the main power source instead of relying on DC power resulting from rectified AC power. However, because of the inter-temporal and the intermittency of solar energy, the design will not eliminate the use of the AC grid to achieve reliability of the data center. In addition, to enhancing power reliability even further, the proposed topology will also add an energy storage system utilizing a battery bank. The operation and coordination of the PV system, AC grid, and the battery bank depend on the load requirements and the energy availability. Since the proposed design uses a PV system as an energy source, priority assignment based on time in the day will be implemented. During the day, the solar panels get the first priority as the main energy source to supply the load. If the solar PV system does not produce enough energy, the AC grid will compensate for the energy and will be combined with the PV. If the PV system fails to generate power, the AC grid will supply the entire required energy to the load. During the nighttime, the priority goes to the AC grid to supply power to the load. For the case where the system experiences loss on both PV and AC power from the utility, the battery bank will take over and power the load. Figure 3-1 illustrates the high-level block diagram of proposed design showing its major components.

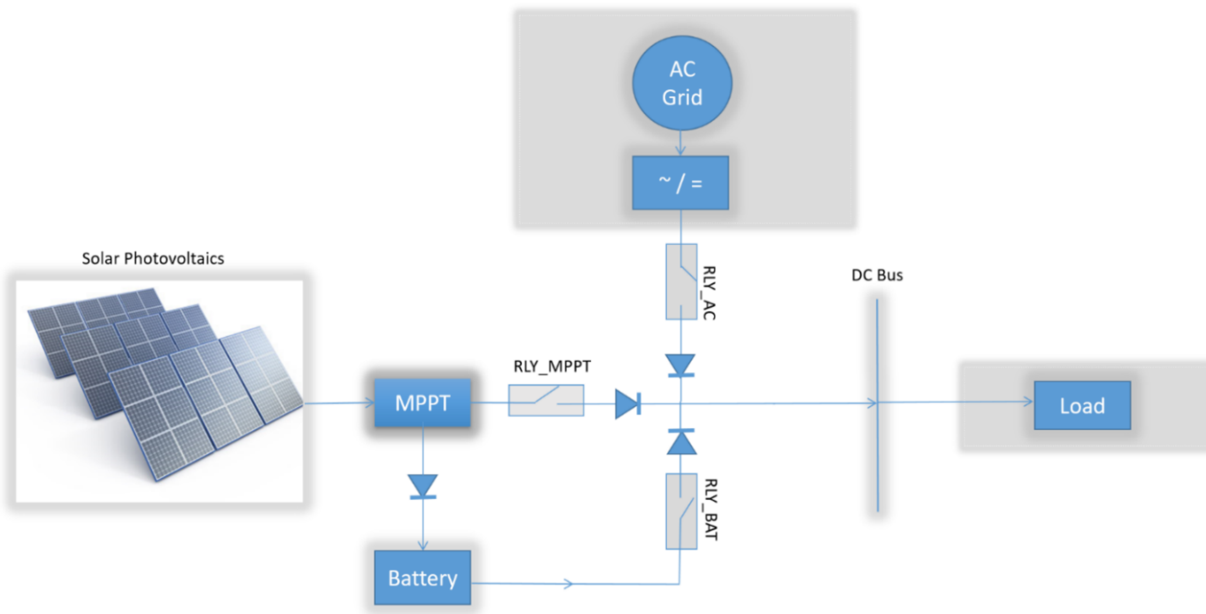

Figure 3-1 High-Level System Block Diagram

3.1 Prototype Specifications

Data centers consume a big amount of energy. As an example, Apple data center's energy consumption is estimated to be 324 million kWh in 2013 as mentioned in their Environmental Responsibility Report [17]. Therefore, the design of a PV system is based on the load requirements. Such intensive energy demand will require a large PV system especially when the sole use of PV is desired to cover the entire load demand during the day. In the proposed design, the size of the PV system size should match the peak load energy demand. For the lab-scale model of the proposed design, the system load will have a maximum power of 100 W to keep the cost of the lab setup low and to minimize risk when performing the test. Furthermore, for safety purpose, the lab setup will not be using 400 V DC as mentioned in [12,15,16]. Instead, the operating voltage of the lab setup will be 24 V DC. The lower operating DC voltage will also allow test measurements using standard lab equipment.

Lastly, to demonstrate the functionality and performance of the proposed topology, a lab-scale construction of the proposed design will be assembled and tested. The lab scale high-level block diagram will be explained in details in Chapter 5.

3.2 Switching Control and Coordination

To ensure proper operation and coordination of such mixture of energy sources, the proposed design utilized a microcontroller to control the switching between the energy sources. When developing an algorithm for the microcontroller, two combined techniques were used. The first technique utilizes Neural Network (NN), and the other technique utilizes combination using C language. NN is chosen because of its ability to respond to any unexpected inputs to the network. During training, neurons are taught to recognize different patterns and generate an output for each pattern. If an unexpected pattern is received, the NN will generate the output from the set of the patterns that has been taught. With the use of NN, all of the possible load values (load power) can be uncovered. Also, NN was able to recognize all of the possible power values generated from the PV system. These values were then used for selecting the proper switch to connect the suitable power source to supply the load.

In the hardware implementation, every power source was connected to the load via a switch that is continuously monitored by sensors. The sensors provide a Multiple Input Multiple Output (MIMO) microcontroller system that measures the amount of power coming from each source and compares them to the power demand at the load. The microcontroller then decides which power source will supply power to the load. As stated earlier and as depicted in Figure 3-2, the algorithm was developed using NN and C language. As previously shown, the block diagram is divided into two halves: the first half shows the NN modeling, and the second half shows the C code modeling. The inputs to the NN are the power generated from the PV and the power at the load. Also, the day time and the AC status are needed for the NN to run. The NN will run only if it is day-time the power from the grid is available. These two parameters are also used in the C code as illustrated in the second half of the block diagram. They are used to control the power sources during the night. The operation of the backup power source (battery bank) was controlled using “if” statement. Also, the block diagram shows that the relay connected to the PV system is controlled by the NN, while the relay connected to the battery bank is controlled by the added C code. However, the relay connected to the AC grid is controlled by the NN only in the day-time and by the C code modeling during night time.

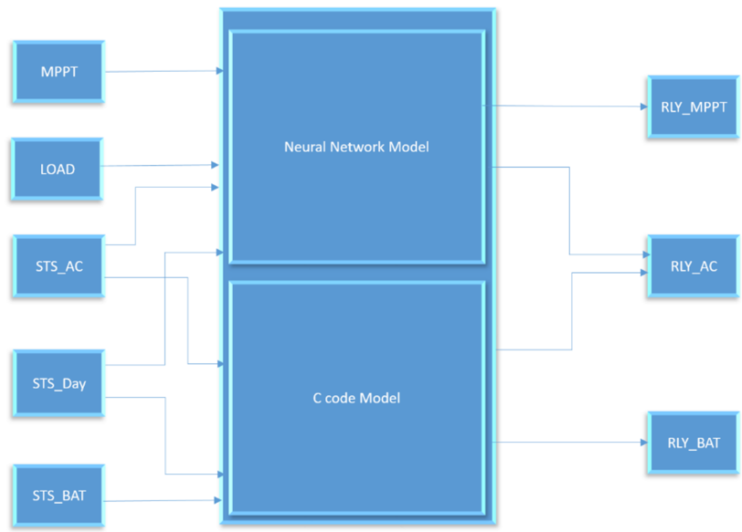

Figure 3-2 System Modeling

Chapter 4: System Design

4.1 System Flowchart

The first step to developing the C code for the proposed system was to create the flowchart as illustrated in Figure 4-1. It starts with reading five different values: the required load power, the power generated from the PV system, the AC grid availability, day/night time status, and the availability of the backup system (batteries). The power measurement of the PV system and the load are analog inputs obtained from voltage and current sensors. The time of day status is a digital input represented by either "1" for daytime or "0" for night-time. During the day and when the status is "1", the energy will be delivered to the load from the PV system if "MPPT >= Load". If the PV system is not enough to power the load, the system will combine the AC grid with the solar panels when the AC status is "1" or available. If the AC status is "0" which means unavailable, the battery bank will serve as a backup. The second possible state is the night-time that means the day status displays "0". The load gets its power either from the utility grid if the AC status is "1", or from the battery bank if the AC status is "0". Specifically, for the battery bank, the status "0" means the battery is not charged while the status "1" means the battery is charged. SW_{AC}, SW_{MPPT}, and SW_{BAT} represent switches connected to the AC grid, MPPT charge controller, and the battery bank respectively. The number "0" means the switch is not connected, and the number "1" means the switch is connected.

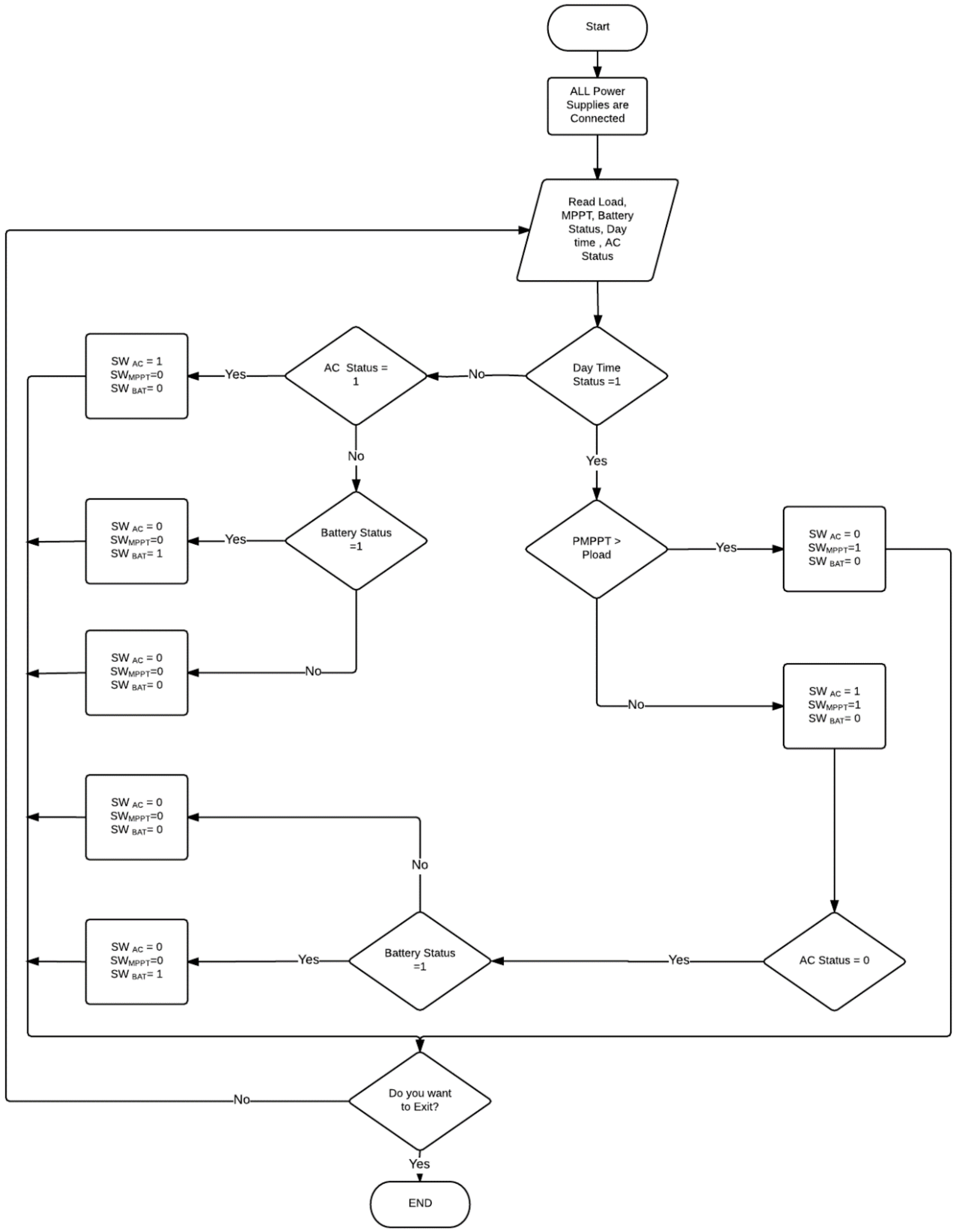

Figure 4-1 System Flowchart

To develop the firmware in C for the flowchart, different tools are utilized as illustrated in Figure 4-2. These tools are discussed separately in more details later in this chapter.

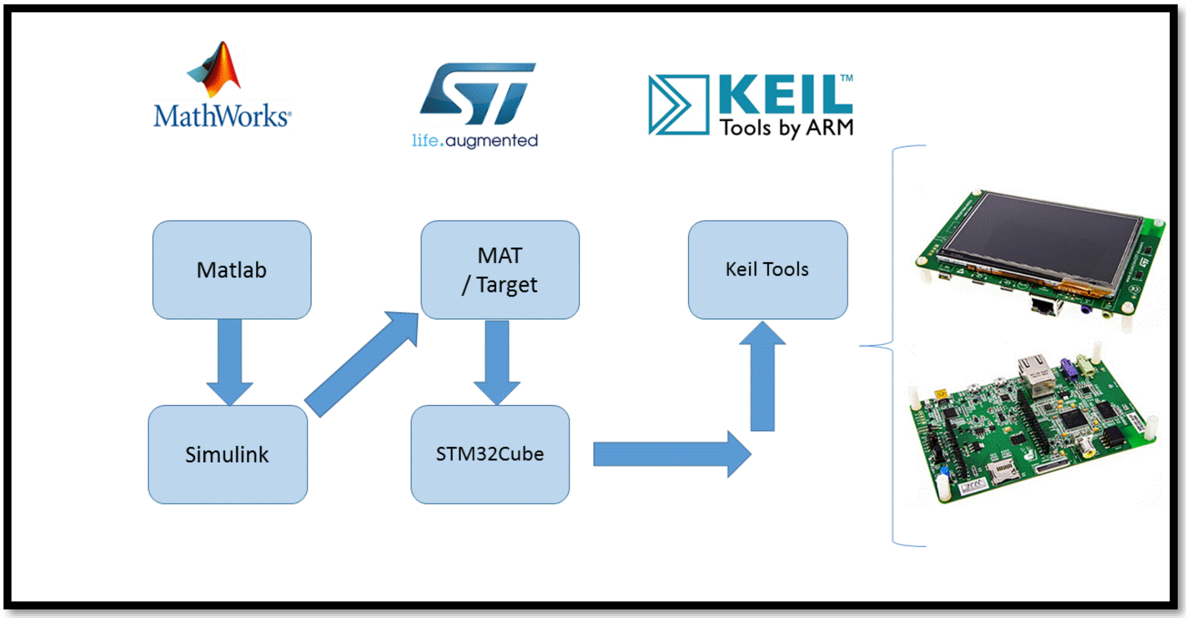

Figure 4-2 Tools to develop the FW for STM32F7

4.2 Matlab and Simulink Model for the Neural Network

A Matlab tool is used to develop an algorithm for the Neural Network (NN). The NN will function only during the day to determine which power source supplies the load, and whether or not to combine the grid with the energy coming from the solar panels. As mentioned in Chapter 3, the NN has two inputs: the load power, and the PV system power. These values are listed in the first two columns of the Table in Appendix (A). These values were chosen between 0.5W and 100W since the goal is to demonstrate a proof-of-concept lab-scale system rather than testing the concept in a real system. The third and the fourth columns of the Table in Appendix (A) are the targets representing the status of the relays connected to the PV and to the grid. Based on the load value and the PV generation, either one relay or two relays will be connected. The number “0” indicates that the relay is not connected, while “1” means that the relay is connected. Below is the NN MATLAB command line for testing the NN.

```
%----Read data----
Data = xlsread('NN data.xlsx');  %-read data from Excell
data_input = Data(1:400,1:2)';     %-data input [2] -> coloumn 1,2
data_target = Data(1:400,3:4)';    %-data target[3] -> coloumn 3,4
S=[5 2];


%----Preparing data----
[Input_,IN] = mapstd(data_input);
[Target_,TG] = mapstd(data_target);

%----Create Network----
NN = newff(minmax(Input_),S,{'tansig','purelin'});

%----Setting Param----
NN.trainParam.epochs = 1000;
NN.trainParam.min_grad = 1e-20;

%----Training Network----
NN=train(NN,Input_,Target_);

%----Check Result----
Sim_NN = sim(NN,Input_);
Rslt_NN = mapstd('reverse',Sim_NN,TG);

[r1,m1,b1] = regression(Target_,Rslt_NN)
NN.IW{1}
```

The first three lines of the code choose the input values from a predefined table. Following this step is to choose the number of layers and the number of neurons in each layer. For this design, there are two layers: a hidden layer with five neurons and an outer layer with two neurons. The mean and the standard deviation of the inputs and targets are then computed and normalized to zero mean and unity standard deviation using the “mapstd” function. Once the normalizing is done, the function “newff” follows to create a feedforward network. The transfer function in the first layer is tan-sigmoid, and the output layer is linear. The "Sim" is used to simulate the NN which takes the network inputs and target, and then return the network outputs. The network output is saved in a new variable called Rslt_NN. These values are also listed in the last two columns in the Table in Appendix A, and their values are equal to the network targets.

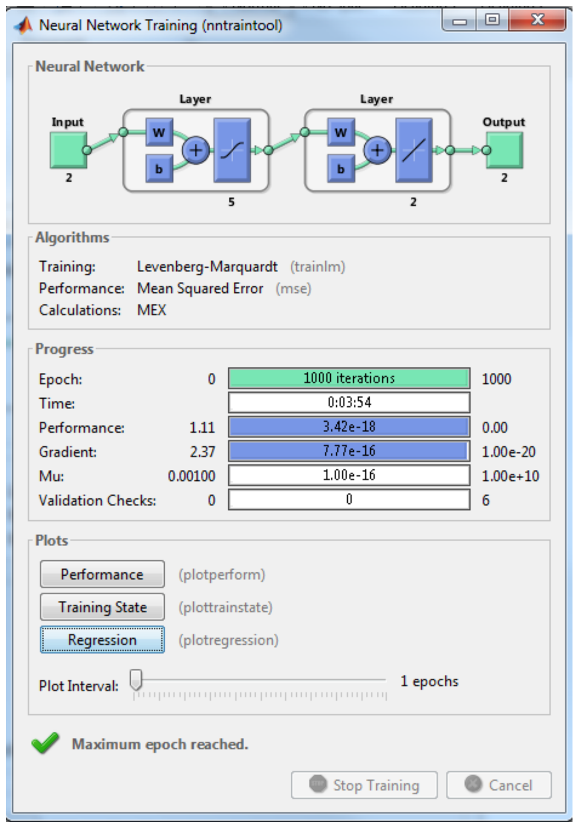

Figure 4-3 NN Training

From the MATLAB training window, three plots were captured: performance, training state, and regression. The performance plot in Figure 4-4 shows the mean square error (MSE) which is always decreasing under training. The training state in Figure 4-5 shows that we reached the bottom of the local minimum of the goal function. The regression plot in Figure 4-6 indicates that the target and the output are having a linear relationship.

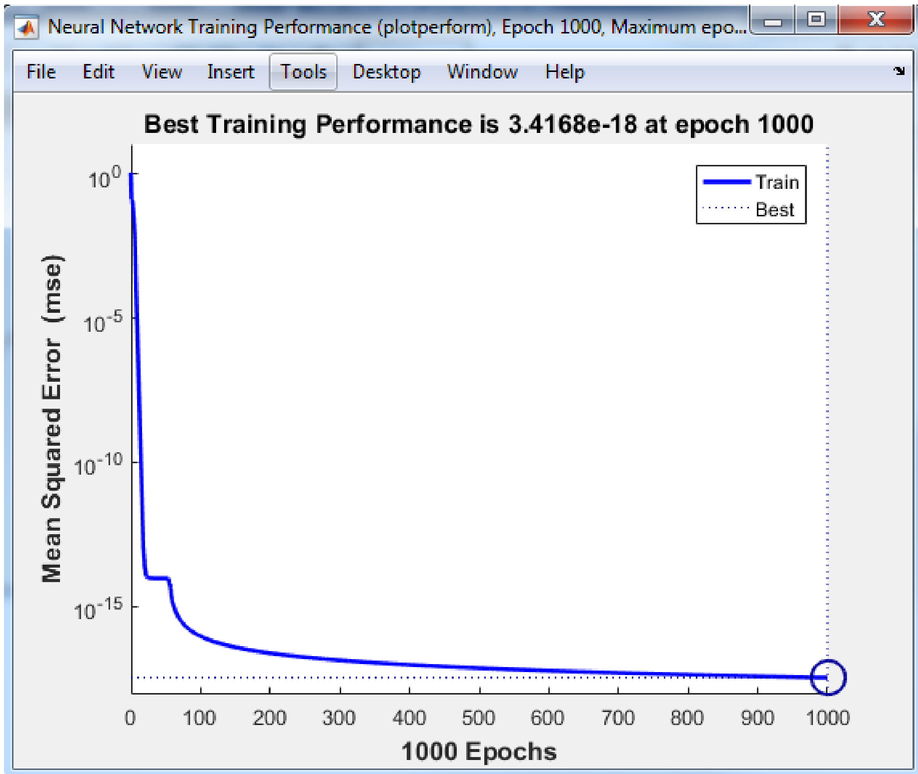

Figure 4-4 Performance plot

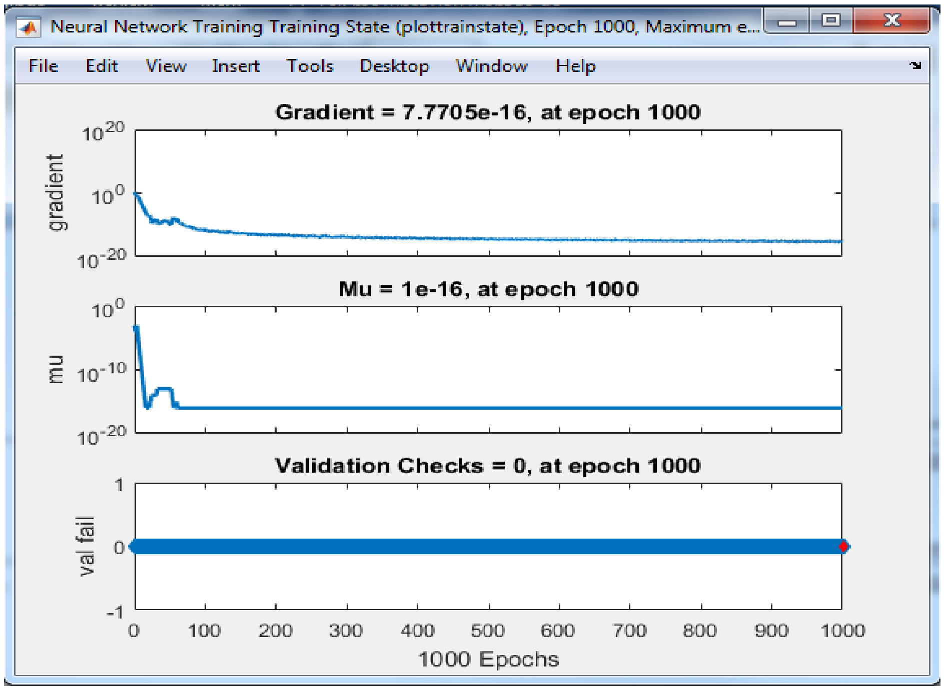

Figure 4-5 Training State

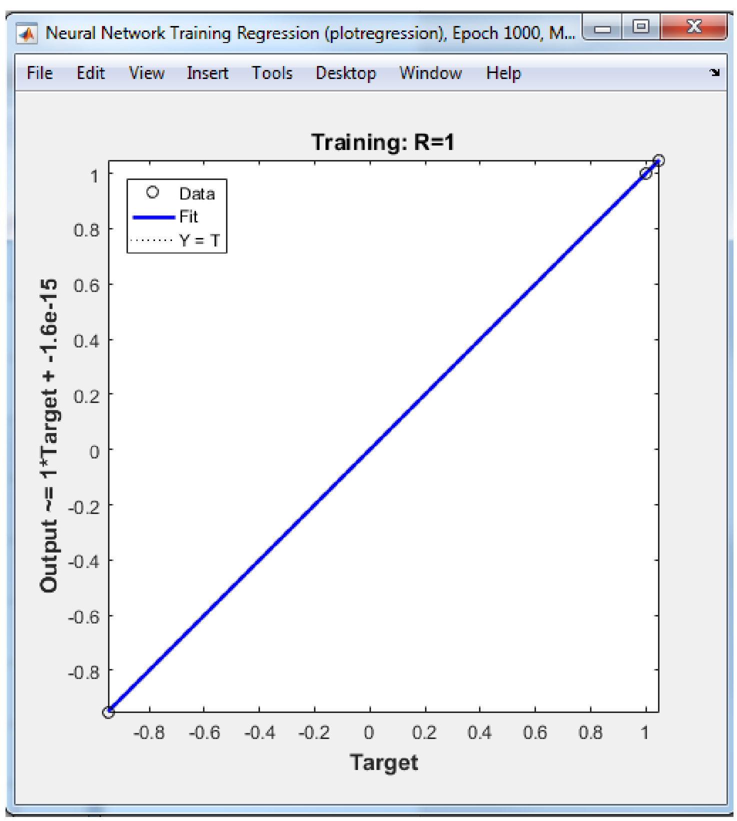

Figure 4-6 Regression Plot

4.3 Simulink

Referring to Figure 4-2, the second tool to develop our code was to build the Matlab function model in Simulink. By typing the command line “gensim(NN)” in the command window, the equivalent NN model was generated in Simulink as shown in Figure 4-7. The Custom Neural Network contains the Matlab code listed in the previous section. The function block mapstd normalizes the inputs and the targets so they both have zero mean and unity standard division. When the mapstd is used to normalize the targets, the network is trained to produce outputs with zero mean and unity standard division. These outputs are finally converted to their original units by means of the mapstd_reverse function block.

To test and verify the model, three cases are presented as depicted in Figures 4-7 through 4-9. In Figure 4-7, the relay connected to the PV system is turned ON while the other relay connected to the AC grid is turned OFF. The reason is that the PV system is generating 80 W that is equal to the power needed by the load. When the PV system is generating more power than what the load requires as shown in Figure 4-8, the relay connected to the PV system is turned ON while the other relay is turned OFF. Lastly, if the power needed by the load is 80 W and the generation from the solar panel is 75.5 W, both switches are turned ON. As expected, the 75.5 W consumed by the load comes from the PV system while the remaining 0.5 W comes from the AC grid, as presented in

Figure 4-9.

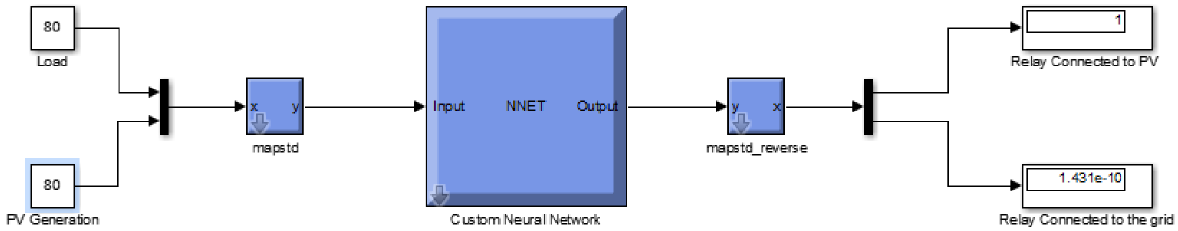

Figure 4-7 Custom NN Model in Simulink when P $_{load}$ is equal to P$_{PV}$

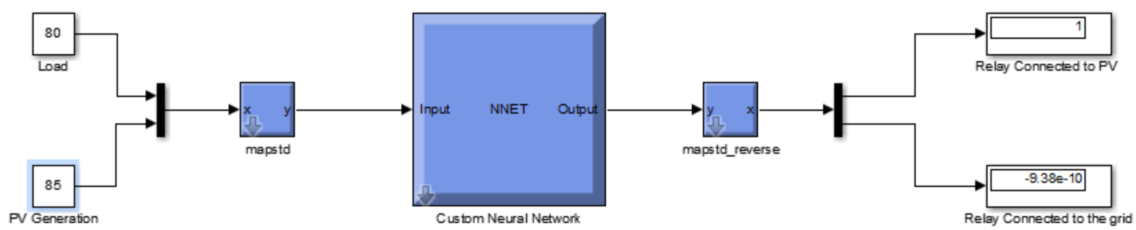

Figure 4-8 Custom NN Model in Simulink when P $_{load}$ is Less than P$_{PV}$

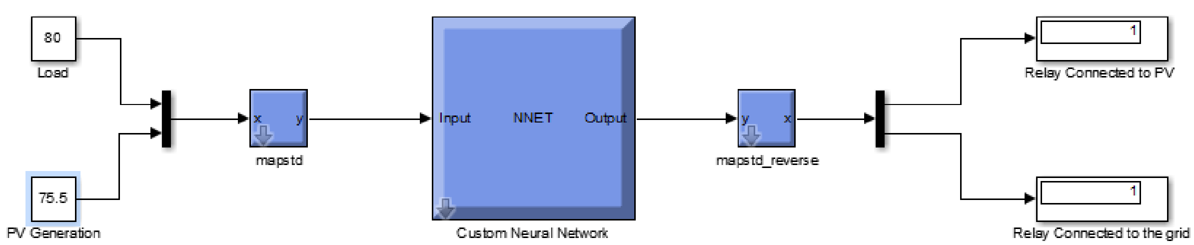

Figure 4-9 Custom NN Model in Simulink when P $_{load}$ is greater than P$_{PV}$

4.4 STM32-MAT/TARGET

After creating the MATLAB and the Simulink design files, the STM32 Embedded Target is used to deploy the design files to STM32 MCU. This step can be done before or after the configuration using STM32CubeMx. The embedded target allows the user to upload a saved configuration or create a new one. When the cubeMx file is uploaded, the STM32-MAT will generate the "C" code in Keil tool. Figure 4-10 shows the Simulink model to generate the "C" code using STM32-MAT. When using the target support package STM32 in Simulink, the equivalent STM32 model for the MCU will be created as shown in Figure 4-11.

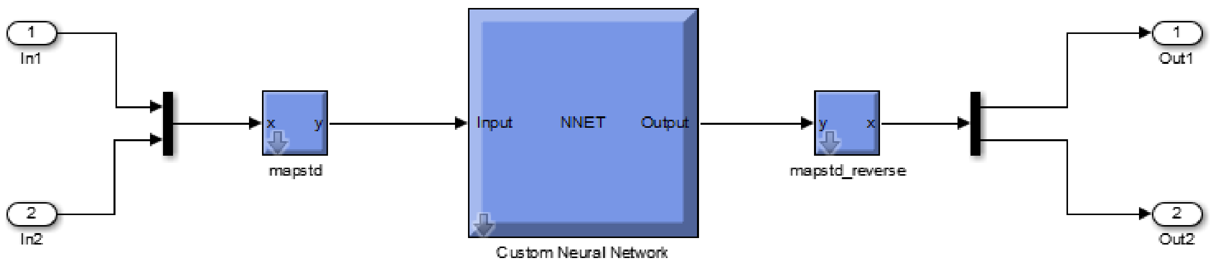

Figure 4-10 Custom NN Model for Mat/Target Showing Two Inputs and two Outputs

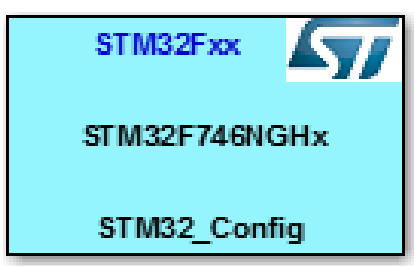

Figure 4-11 Target Support Package STM32

4.5 STM32CubeMx

The STM32CubeMX is a software tool to develop applications on STM32 Microcontrollers based on the user choice and configuration. The STM32CubeMx graphical software configuration tool helps generate the “C” code skeleton. This package includes a low-level hardware abstraction layer (HAL) that covers the Microcontroller hardware. The board STM32F746G-DISCO was first selected for the design. Following this is pin assignments from the pinout window based on the design requirements as explained later in this chapter as pictured in Figure 4-17 and Figure 4-18.

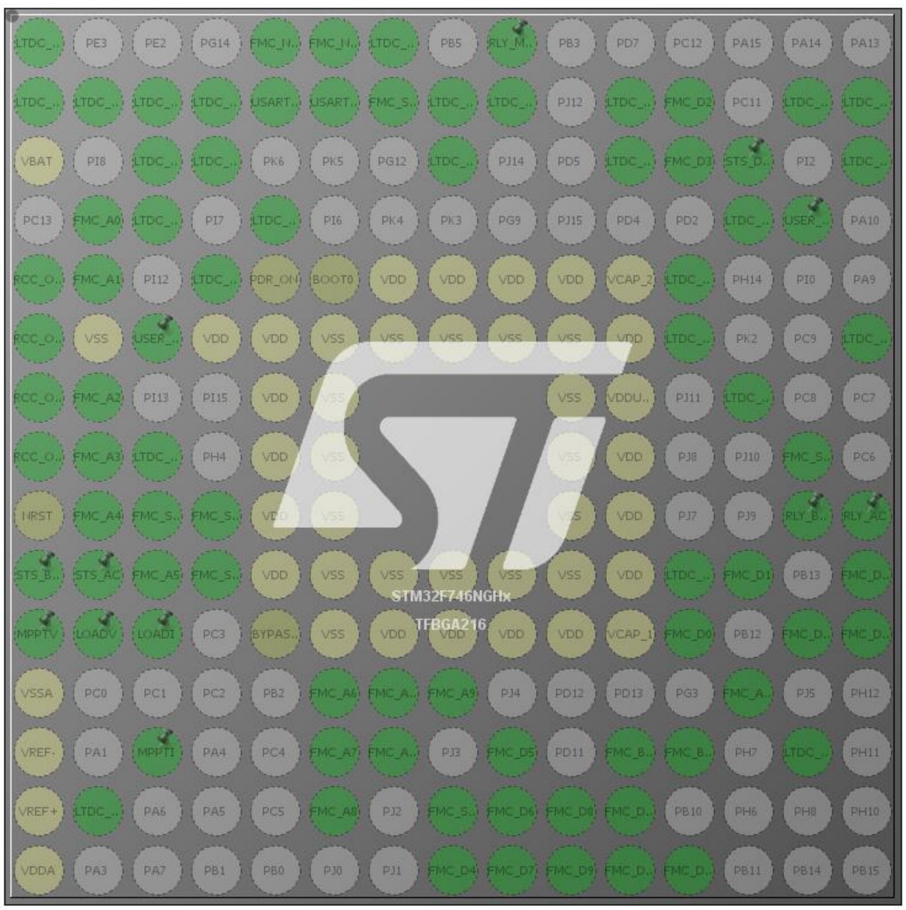

Figure 4-12 STM32CubeMx Pinout

Additionally, pin assignments were also determined according to the microcontroller pins layout in Figure 4-16 that describe the pin number and its name. In Figure 4-12 the pins in green are those assigned for building the C code. Under Pin Configuration window in Figure 4-13, there is a list of all of the assigned digital inputs/outputs. Under GPIO, the pins assigned as outputs are PB4, PG6, and PG7. These pins are for the relay connected to the PV (RLY_MPPT), the relay connected to the AC (RLY_AC), and the relay connected to the battery bank (RLY_BATT) Respectively. The input digital pin is PI3 which serves as an input pin for the day status (STS_DAY). The labels used in this configuration are discussed in more detail under the hardware design section.

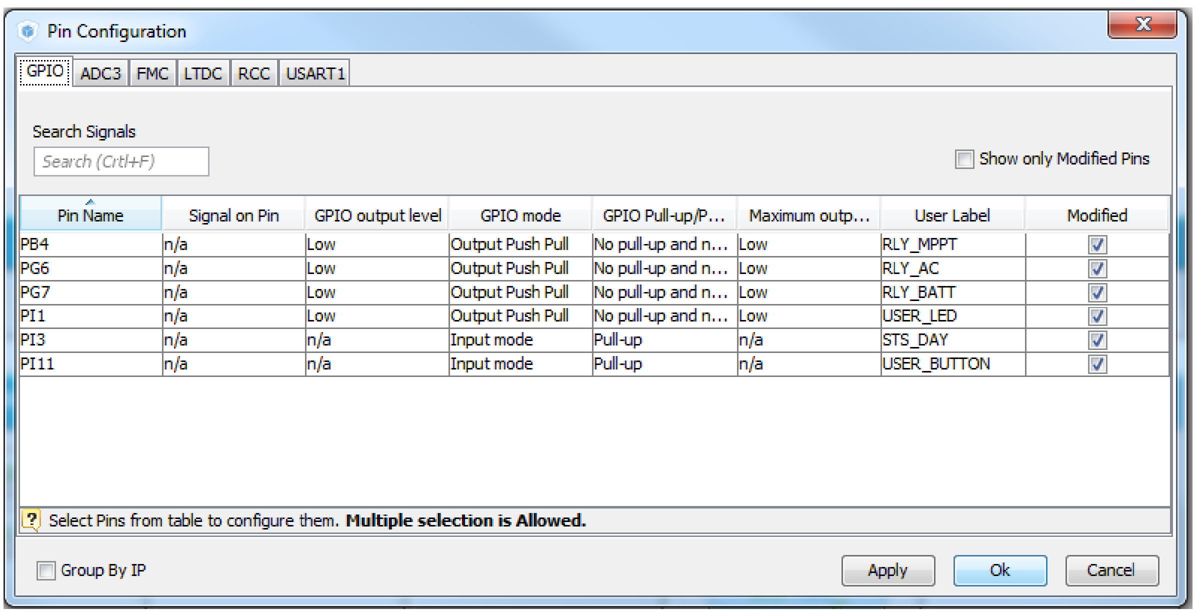

Figure 4-13 Pin Configuration GPIO

Figure 4-14 list shows all the analog pins for the Microcontroller. These pins are PA0, PF6, PF7, PF8, PF9, and PF10They are assigned to read the current coming from the

PV(MPPTI), the AC Status(STS_AC), the load current(LOADI), the load voltage (LOADV), and the PV voltage(MPPTV).

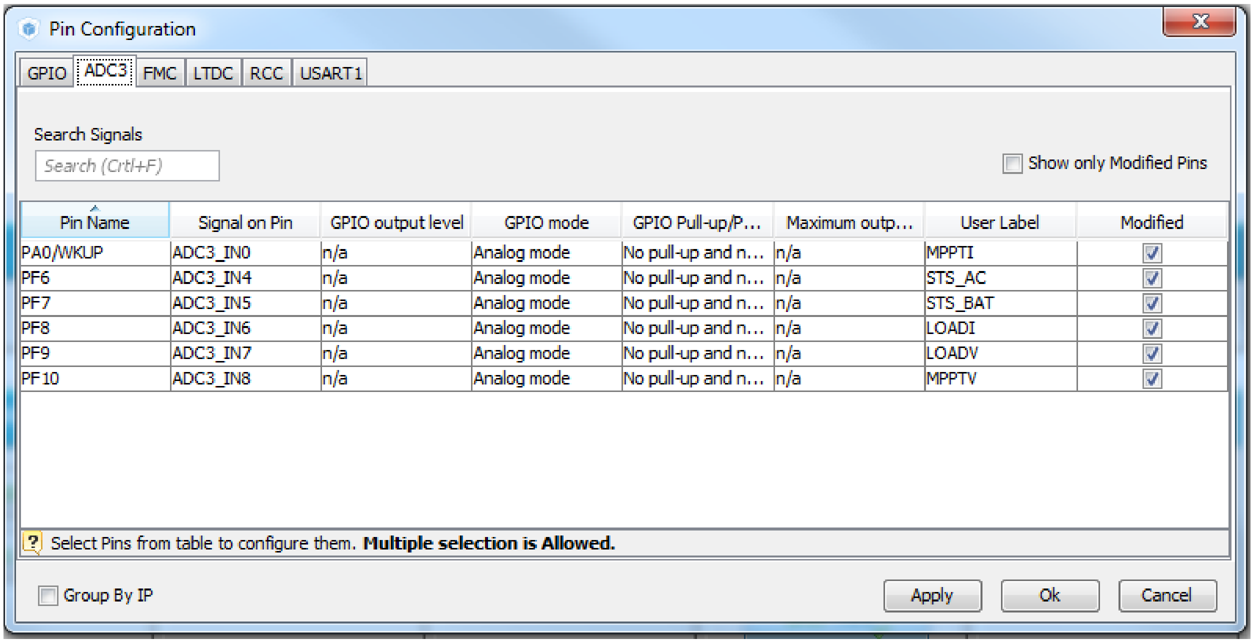

Figure 4-14 Pin Configuration ADC

4.6 KEIL Tool by ARM

The "C" code resided in STM32-MAT Embedded including all of the configurations STM32CubeMx will be opened in KEIL software developing tool. The complete "C" code is included in Appendix (B). However, some of its lines will be explained next since they pertain to this section. For the NN model illustrated in Figure 4-10 the equivalent "C" code generated is:

```
SysIntgration_U.In1=LOADP;
SysIntgration_U.In2=MPPTP;

SysIntgration_step();

my_Out1=SysIntgration_Y.Out1;
my_Out2=SysIntgration_Y.Out2;
```

Within these codes, “SysteIntgration” is the Simulink file’s name. There are two inputs and two outputs: LOADP and MPPTP are the two input quantities to the NN, and my_Out1 and my_Out2 are the output quantities. These output values will determine which relay is connected.

Next, based on the assumptions mentioned in Chapter 3, and according to the flowchart explained earlier in the Chapter the “C” code was developed.

```
    // -------------Read DAY Status                Digital Read
      STS_Day=!(HAL_GPIO_ReadPin(GPIOI, STS_DAY_Pin));


      switch (STS_Day)
      {
        case SET:
        {

          if (STS_AC==SET && MPPTP>=limit)
          {

            SysIntgration_U.In1=LOADP;
            SysIntgration_U.In2=MPPTP;

            SysIntgration_step();

            my_Out1=SysIntgration_Y.Out1;
            my_Out2=SysIntgration_Y.Out2;

            my_Out1R=round(my_Out1);
            my_Out2R=round(my_Out2);

          if (my_Out1R==0) RLY_MPPT=RESET;
          if (my_Out1R==1) RLY_MPPT=SET;
          //  DISP on LCD

          if (my_Out2R==0) RLY_AC=RESET;
          if (my_Out2R==1) RLY_AC=SET;
          // DISP on LCD
          //NN end  Plus delay......
          }
```

These codes basically summarize what is happening inside the NN. It starts with reading the day status based on the output value from the LDR. If both AC grid and PV are available, the NN will control the relays as previously explained in Chapter 3.

If for any reason and as shown in the code below the PV is not operational, then the power comes from the grid. Also, if both the PV and the grid are not supplying power then the battery bank powers the load.

```
        if (STS_AC==SET && MPPTP<=limit)
        {
          RLY_AC=SET;
          RLY_MPPT=RESET;
          RLY_BATT=RESET;
        }

        if (STS_AC==RESET && MPPTP<=0 && STS_BAT==SET)
        {
          RLY_AC=RESET;
          RLY_MPPT=RESET;
          RLY_BATT=SET;
          //Disp on LCD
        }
        if (STS_AC==RESET && MPPTP<=0 && STS_BAT==RESET)
        {
          RLY_AC=RESET;
          RLY_MPPT=RESET;
          RLY_BATT=RESET;
          //Disp on LCD
          //No Source
        }
        break;
      }
```

The equivalent “C” code for the night time is illustrated below. The codes implement the idea that if the AC is available at night time, the power will travel from the grid to the servers. Otherwise, when the AC is not available the battery will serve as a backup.

```
        case RESET:
        {
          if (STS_AC==SET)
          {
            RLY_AC=SET;
            RLY_MPPT=RESET;
            RLY_BATT=RESET;
            //Disp on LCD
          }
          if (STS_AC==RESET && STS_BAT==SET)
          {
            RLY_AC=RESET;
            RLY_MPPT=RESET;
            RLY_BATT=SET;
            //Disp on LCD
          }
          if (STS_AC==RESET && STS_BAT==RESET)
          {
            RLY_AC=RESET;
            RLY_MPPT=RESET;
            RLY_BATT=RESET;
            //Disp on LCD
            //No SOurce
          }

          break;
        }
      }
```

4.7 Hardware Design: Microcontroller

The hardware implementation requires a microcontroller unit to control the relays. STM32F746G-DISCO Board was chosen in the design as illustrated in Figure 4-8. This MCU features 12-bit ADCs, two 12-bit DACs, and a colored LCD. It also comes with powerful firmware libraries to support the hardware and comes with the STM32 comprehensive software HAL library. The HAL driver layer comes with a complete set of ready to use APIs (Application Programing Interfaces). As an example, the API that is used to read pin is “HAL_GPIO_ReadPin()”.

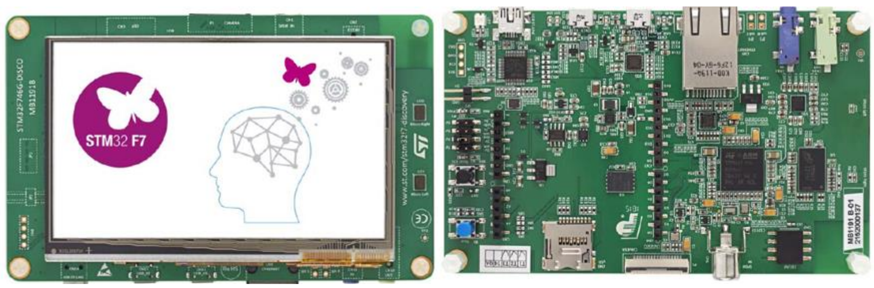

Figure 4-15 STM32F746G-DISCO Board Top and Bottom View

By looking at the STM32F7 board layout in Figure 4-16, there are six Analog pins labeled as A0, A1, A2, A3, A4, and A5. It also has 16 digital pins labeled D0 through D15. Hence, some of these pins are used for inputs and some for outputs as illustrated in Figure 4-17 and Figure 4-18.

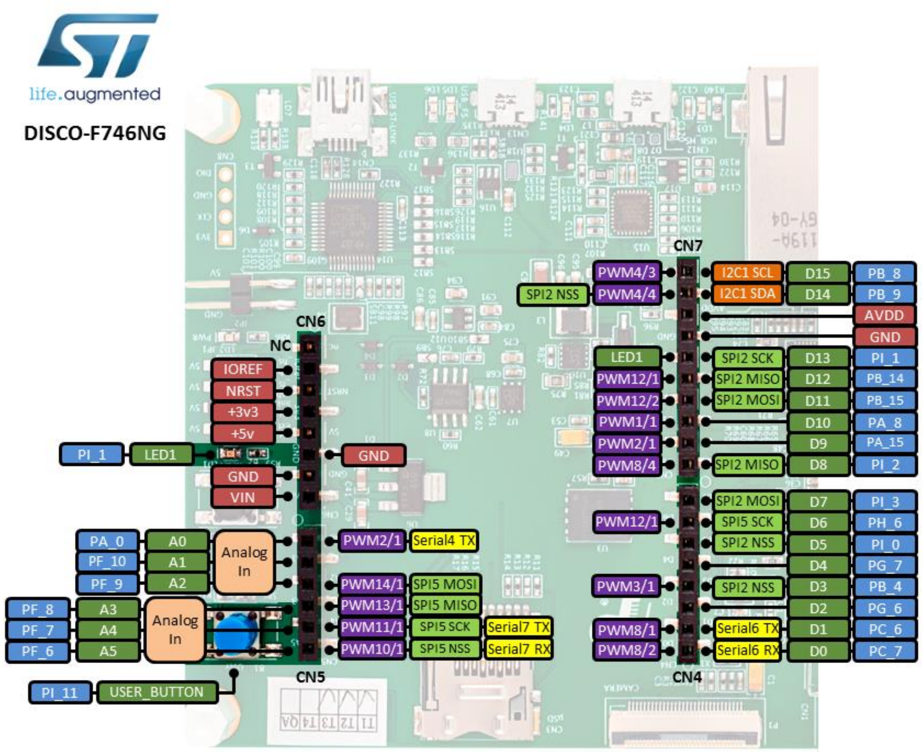

Figure 4-16 Pinout for the STM32F7

As shown in Figure 4-13, there are six analog inputs: two pins to measure the PV power, another two inputs for the load power, one input for the availability of the AC grid, and one input for the availability of a backup system. Also, there is one digital input for the day/night time. However, the three output pins are all digital, and they are for the relays.

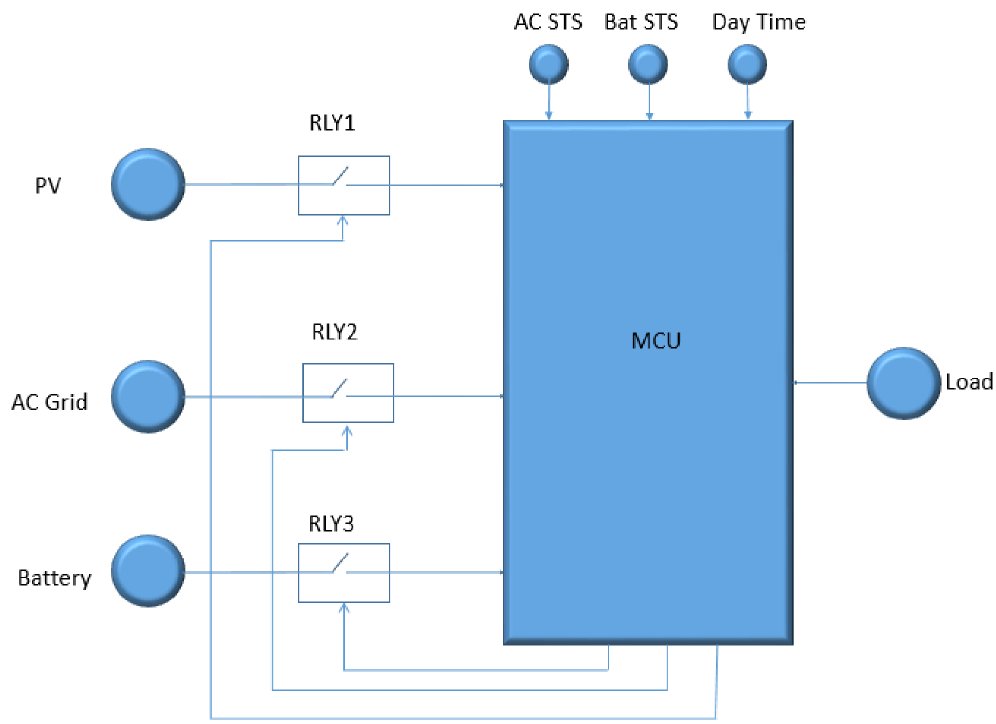

Figure 4-17 Microcontroller I/O

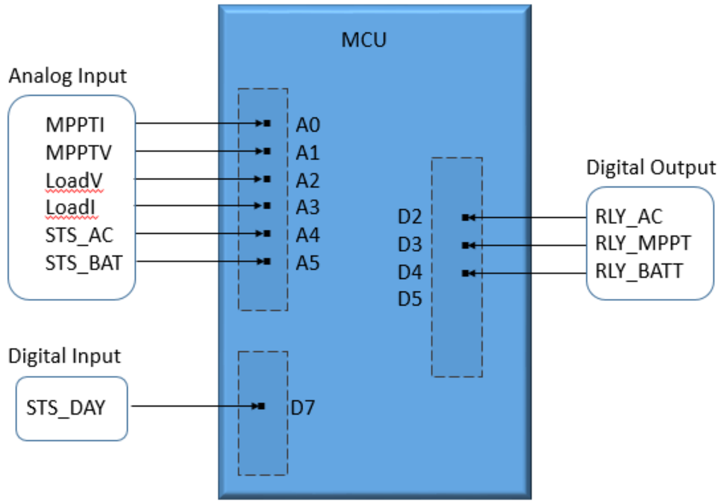

Figure 4-18 Assigned pins to the STM32F7

As previously explained, pin assignments were processed by the STM32CubeMx while the KEIL environment develops the C code. To read from the pins, the following code was utilized.

```
            // -------------Read Power
          //Power MPPT
          for(i=0;i<1000;i++)
          {

          MPPTI=(float)ADC_BUFFER[0]*3.3/4096;           //Analog Read
          MPPTI=(MPPTI-2.5)/0.066;
          MPPTV=(float)(ADC_BUFFER[1]*3.3/4096)*(7.8);           //Analog Read
          MPPTP=MPPTV*MPPTI;

          if (MPPTV>=24) STS_MPPT=SET;
          if (MPPTV<20) STS_MPPT=RESET;
//
//          //Power LOAD
          LOADV=(float) (ADC_BUFFER[2]*3.3/4096)*7.8;          //Analog Read
          LOADI=(float)ADC_BUFFER[3]*(3.3/4096);          //Analog Read
          LOADI=(LOADI-2.5)/0.066;
          LOADP=LOADV*LOADI;
          // Dsiplay to LCD

        // -------------Read AC Status
          STS_ACval=(float)ADC_BUFFER[4]*3.3/4096;          //Analog Read
          if (STS_ACval>=1)
          {
            STS_AC=SET;
          }
          if (STS_ACval<1)
          {
            STS_AC=RESET;
          }
        // -------------Read Battery Status
          STS_BATval=(float)ADC_BUFFER[5]*3.3/4095;               //Analog Read
          if (STS_BATval>=1)
          {
            STS_BAT=SET;
          }
          if (STS_BATval<1)
          {
            STS_BAT=RESET;
          }
        }
```

The library driver used in the code above to read analog input was “ADC_BUFFER[]”. These read values are from the current and voltage sensors for the PV system and the load, and from the voltage dividers for the AC and the read battery status. In addition, the HAL driver “HAL_GPIO_ReadPin()” is being used to read the digital input for the day status. Further explanation on these sensors will be presented in the next chapter.

```
// -------------Read DAY Status                Digital Read
  STS_Day=!(HAL_GPIO_ReadPin(GPIOI, STS_DAY_Pin));
```

These values were displayed in the MCU colored LCD as shown in Figure 4-19 using Board Support Package Driver (BSP). The driver provides a set of user-friendly APIs.

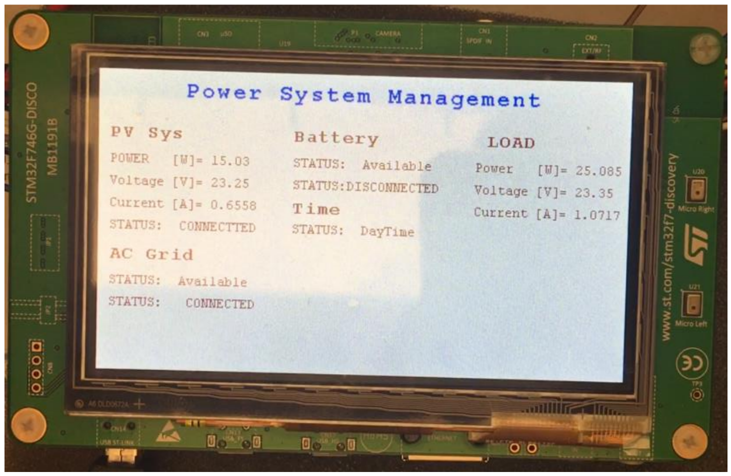

Figure 4-19 LCD Display for STM32F7

```
//      //############ PV System ########################
  for(display0=0;display0<=30;display0++)
 {
 BSP_LCD_SetFont(&Font16);          // Set font size
 BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //Set text color
 BSP_LCD_DisplayStringAt(10, 50, (uint8_t *)"PV Sys", LEFT_MODE); //set the location in the screen

 sprintf((char *)BUFFER_STR,"POWER   [W]= %3.2f",MPPTP);
 BSP_LCD_SetFont(&Font12);          //
 BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //
 BSP_LCD_DisplayStringAt(10, 75, (uint8_t *)BUFFER_STR, LEFT_MODE);

 sprintf((char *)BUFFER_STR,"Voltage [V]= %2.2f",MPPTV);
 BSP_LCD_SetFont(&Font12);          //
 BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //
 BSP_LCD_DisplayStringAt(10, 95, (uint8_t *)BUFFER_STR, LEFT_MODE);

 sprintf((char *)BUFFER_STR,"Current [A]= %2.2f",MPPTI);
 BSP_LCD_SetFont(&Font12);          //
 BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //
 BSP_LCD_DisplayStringAt(10, 115, (uint8_t *)BUFFER_STR, LEFT_MODE);

 BSP_LCD_SetFont(&Font12);          //
 BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //
 if (RLY_MPPT==RESET){
 BSP_LCD_DisplayStringAt(10, 135, (uint8_t *)"STATUS:DISCONNECTED", LEFT_MODE);
 HAL_GPIO_WritePin(RLY_MPPT_GPIO_Port, RLY_MPPT_Pin,GPIO_PIN_RESET);
 }
 if (RLY_MPPT==SET){
 BSP_LCD_DisplayStringAt(10, 135, (uint8_t *)"STATUS:  CONNECTED", LEFT_MODE);
 HAL_GPIO_WritePin(RLY_MPPT_GPIO_Port, RLY_MPPT_Pin,GPIO_PIN_SET);
 } //
```

```
////############ AC Grid ########################

  BSP_LCD_SetFont(&Font16);          //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //
  BSP_LCD_DisplayStringAt(10, 160, (uint8_t *)"AC Grid", LEFT_MODE);


  BSP_LCD_SetFont(&Font12);          //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);     //
  if (STS_AC==RESET)
  BSP_LCD_DisplayStringAt(10, 185, (uint8_t *)"STATUS:Unavailable", LEFT_MODE);
  if (STS_AC==SET)
  BSP_LCD_DisplayStringAt(10, 185, (uint8_t *)"STATUS:  Available", LEFT_MODE);

  BSP_LCD_SetFont(&Font12);          //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //
  if (RLY_AC==RESET){
  BSP_LCD_DisplayStringAt(10, 205, (uint8_t *)"STATUS:DISCONNECTED", LEFT_MODE);
  HAL_GPIO_WritePin(GPIOG, RLY_AC_Pin,GPIO_PIN_RESET);
  }
  if (RLY_AC==SET){
  BSP_LCD_DisplayStringAt(10, 205, (uint8_t *)"STATUS:   CONNECTED", LEFT_MODE);
  HAL_GPIO_WritePin(GPIOG, RLY_AC_Pin,GPIO_PIN_SET);
  }
    //
```

```
////############ Battery ########################
//
  BSP_LCD_SetFont(&Font16);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  BSP_LCD_DisplayStringAt(175, 50, (uint8_t *)"Battery", LEFT_MODE);
  BSP_LCD_SetFont(&Font12);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  if (STS_BAT==RESET)
  BSP_LCD_DisplayStringAt(175, 75, (uint8_t *)"STATUS:Unavailable", LEFT_MODE);
  if (STS_BAT==SET)
  BSP_LCD_DisplayStringAt(175, 75, (uint8_t *)"STATUS:  Available", LEFT_MODE);
  BSP_LCD_SetFont(&Font12);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);  //
  if (RLY_BATT==RESET){
  BSP_LCD_DisplayStringAt(175, 95, (uint8_t *)"STATUS:DISCONNECTED", LEFT_MODE);
  HAL_GPIO_WritePin(GPIOG, RLY_BATT_Pin,GPIO_PIN_RESET);
  }
  if (RLY_BATT==SET){
  HAL_GPIO_WritePin(GPIOG, RLY_BATT_Pin,GPIO_PIN_SET);
  BSP_LCD_DisplayStringAt(175, 95, (uint8_t *)"STATUS:   CONNECTED", LEFT_MODE);
  }
////############ Day or night ########################
  BSP_LCD_SetFont(&Font16);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  BSP_LCD_DisplayStringAt(175, 115, (uint8_t *)"Time", LEFT_MODE);
  BSP_LCD_SetFont(&Font12);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  if (STS_Day==SET)
  BSP_LCD_DisplayStringAt(175, 135, (uint8_t *)"STATUS:  DayTime", LEFT_MODE);
  if (STS_Day==RESET)
  BSP_LCD_DisplayStringAt(175, 135, (uint8_t *)"STATUS:NightTime", LEFT_MODE);

//
```

```
////############# Load #########################
  BSP_LCD_SetFont(&Font16);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  BSP_LCD_DisplayStringAt(350, 50, (uint8_t *)"LOAD", LEFT_MODE);

  sprintf((char *)BUFFER_STR,"Power   [W]= %3.2f",LOADP);
  BSP_LCD_SetFont(&Font12);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  BSP_LCD_DisplayStringAt(340, 75, (uint8_t *)BUFFER_STR, LEFT_MODE);

  sprintf((char *)BUFFER_STR,"Voltage [V]= %2.2f",LOADV);
  BSP_LCD_SetFont(&Font12);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  BSP_LCD_DisplayStringAt(340, 95, (uint8_t *)BUFFER_STR, LEFT_MODE);

  sprintf((char *)BUFFER_STR,"Current [A]= %2.2f",LOADI);
  BSP_LCD_SetFont(&Font12);         //
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);        //
  BSP_LCD_DisplayStringAt(340, 115, (uint8_t *)BUFFER_STR, LEFT_MODE);
}
  }

```

4.8 Current sensor

The design employs two current sensors ACS712ELCTR-30A-T to measure the MPPT and the load current values. Based on their datasheet and as shown in Figure 4-20 the two terminals on the left side tie to the positive terminal of the load/MPPT to measure the current. If zero current flows in the sensor, the output voltage is 2.5 V which is half of the supply voltage VCC. When an increasing current flows in the sensor, the out voltage changes as a fraction of this current creating a positive slope as shown in Figure 4-21.

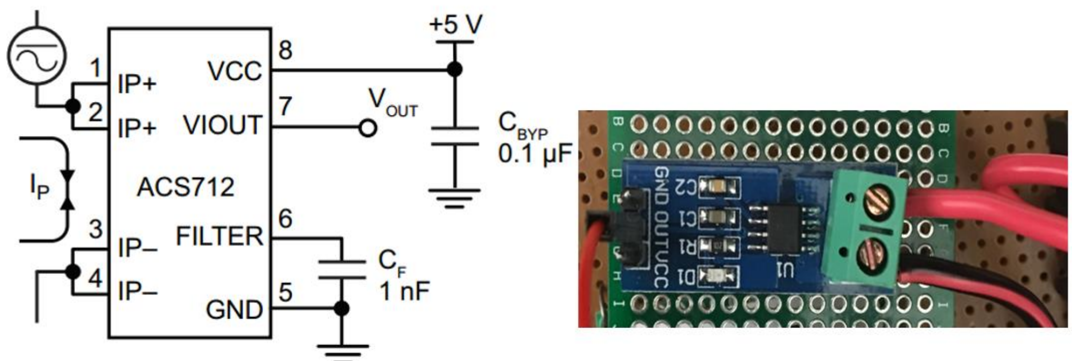

Figure 4-20 ACS712ELCTR-30A-T Current Sensor Model

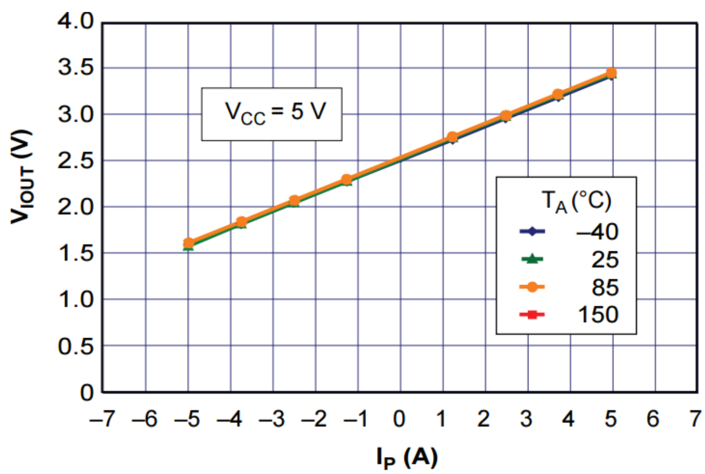

Figure 4-21 Output Voltage Vs Sensed Current (Datasheet Result)

The right side of the sensor requires three connections: 5 volt VCC, ground, and the output voltage must connect to the analog input of the microcontroller PA_0 and PF_8. For testing purposes, one terminal of the sensor connects to 40 VDC power supply while the other terminal is connected to the electronic load. The electronic load acts as a variable resistor to control the current value. The current values and their corresponding voltage value were measured at different resistance as shown in Figure 4-22. Figure 4-23 shows the resulting voltage to current ratio graph.

Figure 4-22 Current Sensor Testing

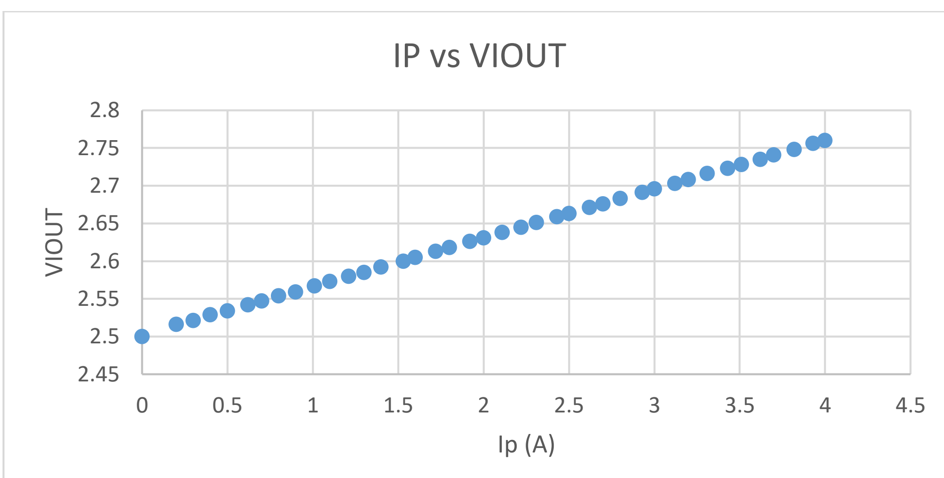

Figure 4-23 Output Voltage Vs Sensed Current (Lab Result)

Next, for the microcontroller to read the current value, each current sensor (MPPTI and LOADI) needs two lines of code. The first line is to read an analog input that converts to Ampere (A) using Equation 4-1.

$$\frac{Vrefrence}{2^N}\ X\ read\ Value$$

Equation 4—1

where N is the number of bits

The second line of code is to subtract the sensor initial value from the value read and divide it by the efficiency as in Equation 4-2.

$$\frac{(Read\ Voltage - 2.5)}{sensitivity}$$

Equation 4—2

```
MPPTI=(float)ADC_BUFFER[0]*3.3/4096;            //Analog Read
MPPTI=(MPPTI-2.5)/0.066;
```

```
LOADI=(float)ADC_BUFFER[3]*(3.3/4096);        //Analog Read
LOADI=(LOADI-2.5)/0.066;
```

4.9 Voltage Sensor

The system uses four voltage sensors to measure the MPPTV, LOADV, DAY_STS, and STS_BAT. The voltage sensors also scale down the system voltage from 24 V DC to 3.3 V DC to provide the suitable maximum analog input of the microcontroller. For the purpose of this thesis, voltage divider operates as a linear circuit that generates an output voltage Vo as a fraction of its input voltage Vin. The voltage divider accepts variable Vin

= 0 - 24 V, and maximum Vo= 3.3 V. The ratio needed to select the resistor values for the voltage divider was calculated from Equation 4-3.

$$\frac{Max\ System\ Voltage}{Max\ Vo} = \frac{24}{3.3} = 7.27$$

Equation 4—3

The resulting ratio is approximately 7:1, and an example of standard resistance values that meet this ratio are 1kΩ and 6.8 kΩ. These two resistors make the series connection as in Figure 4-24.

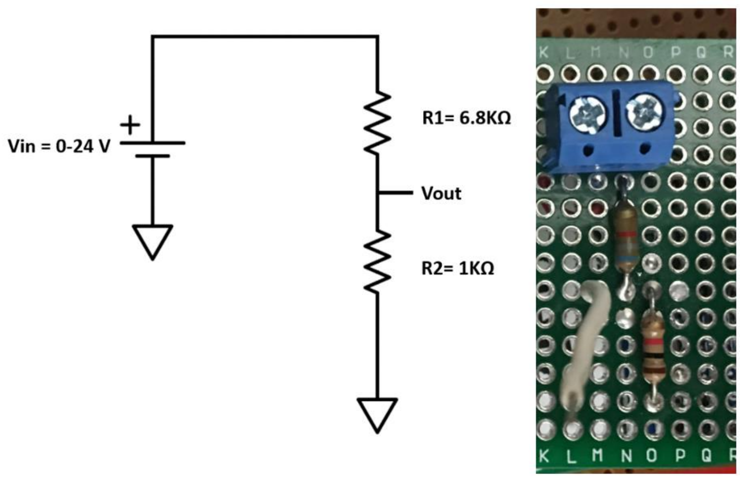

Figure 4-24 Voltage sensor Circuit

Before start using the divider, tests were conducted to the voltage sensor circuit by connecting them to a variable DC power supply. The voltage value was varied and recorded with the corresponding Vout as listed in Table 4-1. The resulting plot that shows the relationship between Vin and Vout is presented in Figure 4-25.

Table 4-1 Voltage Divider Testing Result (Vin Vs Vout)

Vin (Volt)	Vout (Volt)
0.5	0.0635
1	0.1269
1.5	0.19074
2	0.25402
2.5	0.31737
3	0.38117
3.5	0.4447
4	0.5083
4.5	0.5717
5	0.6353
5.5	0.6991
6	0.7628
6.5	0.8262
7	0.8895
7.5	0.953
8	1.0166
8.5	1.0803
9	1.1434
9.5	1.2071
10	1.2709
10.5	1.3347
11	1.3983
11.5	1.4616
12	1.5253
12.5	1.5888
13	1.6524
13.5	1.7161
14	1.7798
14.5	1.8434
15	1.9069
15.5	1.9706
16	2.0344
16.5	2.0978
17	2.1615
17.5	2.2253
18	2.2888
18.5	2.3528
19	2.4163
19.5	2.4798
20	2.5435
20.5	2.6073
21	2.671
21.5	2.7347
22	2.7985
22.5	2.8622
23	2.9259
23.5	2.9898
24	3.0536

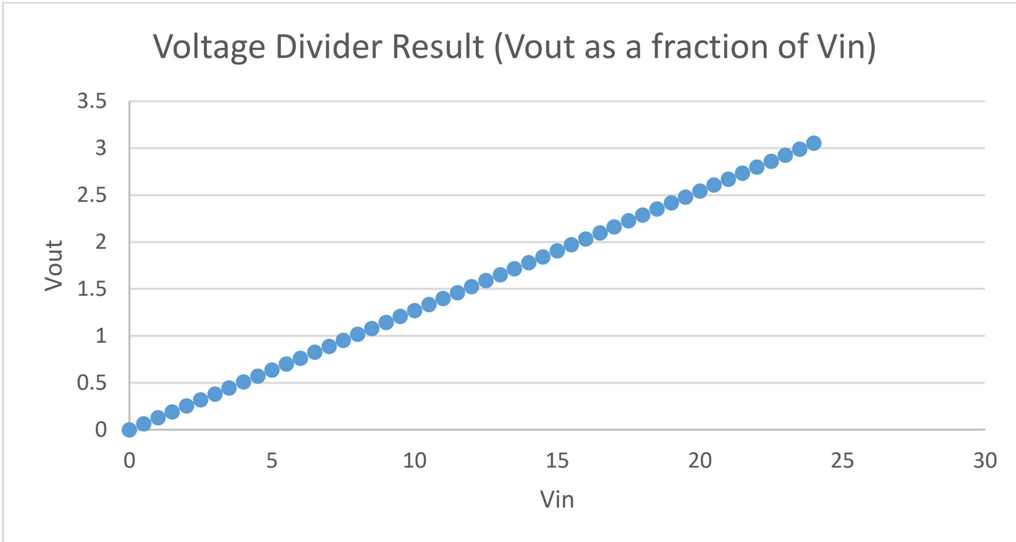

Figure 4-25 Voltage Divider Result (Vout as a fraction of Vin)

For the microcontroller to read the voltage value, only one line of code was used for the MPPTV and the LOADV divider as shown below.

```
MPPTV=(float)(ADC_BUFFER[1]*3.3/4096)*(7.8);
LOADV=(float) (ADC_BUFFER[2]*3.3/4096)*7.8;
```

Both dividers used Equation 4-4 to convert the read value to voltage

$$\frac{3.3}{2^N}\ X\ Voltage\ Divider\ Ratio\ X\ Read\ Value$$

Equation 4—4

where N: is the number of bits

The codes for the voltage dividers used in STS_AC and STS_BAT are a little bit different. If there is a voltage exist in the divider, that means the AC/Battery is available and the opposite is true if no voltage was measured, as in the code below.

```c
    // -------------Read AC Status
      STS_ACval=(float)ADC_BUFFER[4]*3.3/4096;
      if (STS_ACval>=1)
      {
        STS_AC=SET;
      }
      if (STS_ACval<1)
      {
        STS_AC=RESET;
      }
    // -------------Read Battery Status
      STS_BATval=(float)ADC_BUFFER[5]*3.3/4095;
      if (STS_BATval>=1)
      {
        STS_BAT=SET;
      }
      if (STS_BATval<1)
      {
        STS_BAT=RESET;
      }
    }
```

4.10 Light Dependent Resistor

The Light Dependent Resistor (LDR) is a device that can simulate day and night times. For this project, the LDR chosen is PGM5506 LDR. Figure 4-26 shows the voltage divider for the LDR where one leg of the LDR is connected to ground, the other leg to the digital input of the STM32F7 microcontroller and a 10.4 kΩ resistor, and the other side of the resistor is connected to 3.3 V of the Vcc.

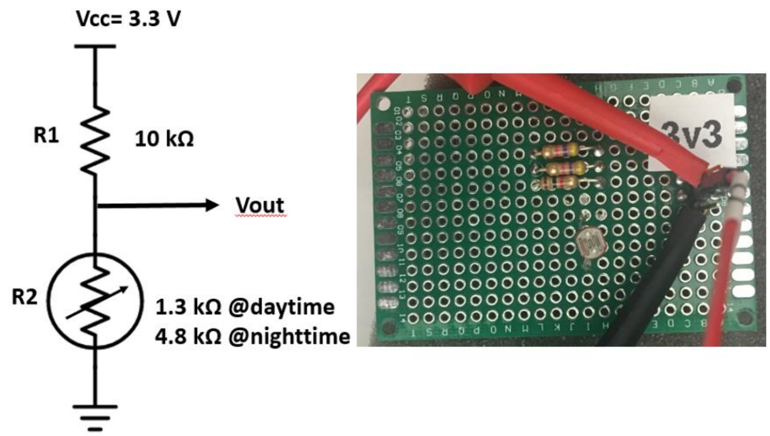

Figure 4-26 LDR Circuit Design

Before designing the circuit, the resistor across the two terminals of the LDR was measured using Fluke 116 HVAC Multimeter. The dark resistance for the LDR is 48 kΩ, and the photo resistance is 1.4 kΩ. A 10.4 kΩ in the divider is selected to maintain the output voltage values so that the voltage levels will correspond to digital 0 or digital 1. Figure 4-27 illustrates the standard CMOS voltage level [19].

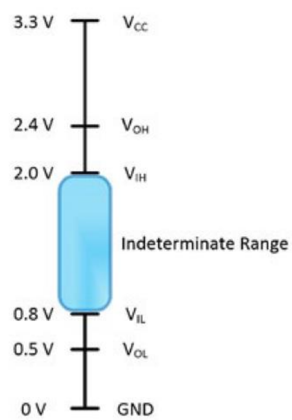

Figure 4-27 CMOS voltage level

where

VIL is the input voltage needs to be sent to the device to read logic 0

VIH is the input voltage needs to be sent to the device to read logic 1

VOH is the output high-level voltage which measures the output to generate 1

VOL is the output low-level voltage which measures the output to generate 0

For this design with LDR resistor is 48 kΩ, the output voltage is equal to:

$$\frac{48\ k\ \Omega}{48k\ \Omega\ + 10.4k\ \Omega}\ x3.3 = 2.71\ V\ (Logic\ 1)$$

Equation 4—5

And when LDR resistor is 1.4 kΩ, the output voltage is equal to:

$$\frac{1.4\ k\ \Omega}{1.4k\ \Omega\ + 10.4k\ \Omega}\ x3.3 = 0.39\ V\ (Logic\ 0)$$

Equation 4—6

The above two equations provide logic 1 for the night time and logic 0 for the day time. However, these values are the opposite to what being used in the design. Hence, these values were inverted in the C command for the microcontroller to read logic 1 during the day and logic 0 during the night. Below shows the C command line to read the value coming from the LDR:

```
    // -------------Read DAY Status                Digital Read
      STS_Day=!(HAL_GPIO_ReadPin(GPIOI, STS_DAY_Pin));
```

The status the day will then be displayed on the screen as shown in Figures 4-28 and 4-29.

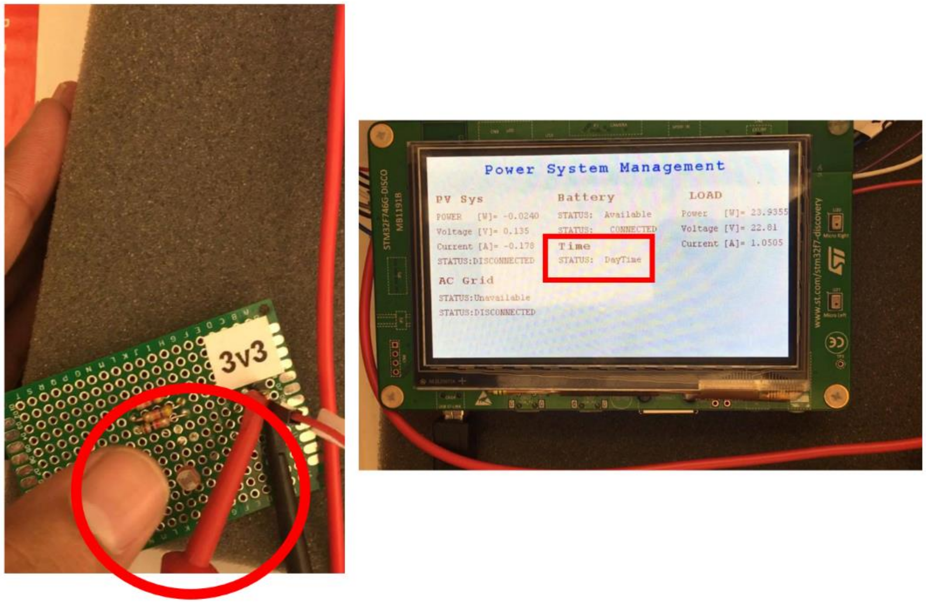

Figure 4-28 Daytime testing

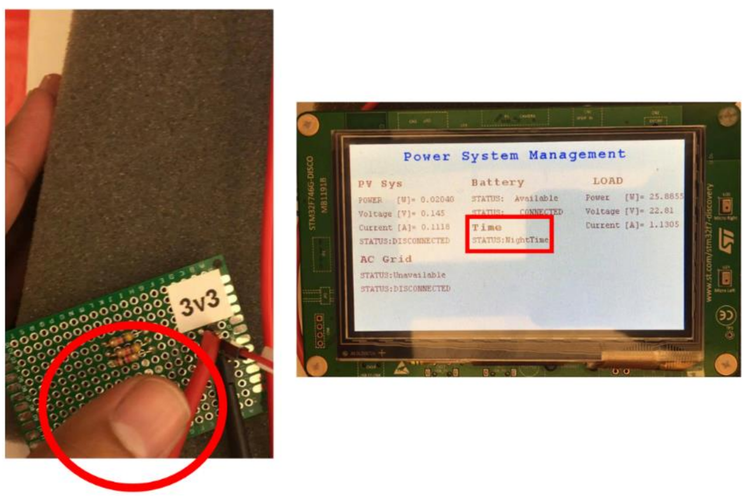

Figure 4-29 Night time Testing

4.11 Relay Board

The relay board is the DT-I/O Quad Relay Board as shown in Figure 4-30. The model requires 5 V supply voltage and can handle up to 10 A rated current. Out of the four relays, the model has only three relays are actually used which are labeled 0, 1, and 2 in the Figure. Relay 0 is connected to the MPPT charge controller which is RLY_MPPT in our C code. Relay 1 is RLY_AC in our code which connects to the AC grid, and finally Relay 2 is connected to the battery bank assigned as RLY_BAT.

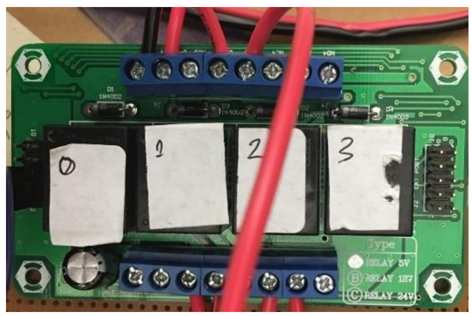

Figure 4-30 DT-I/O Quad Relay Board

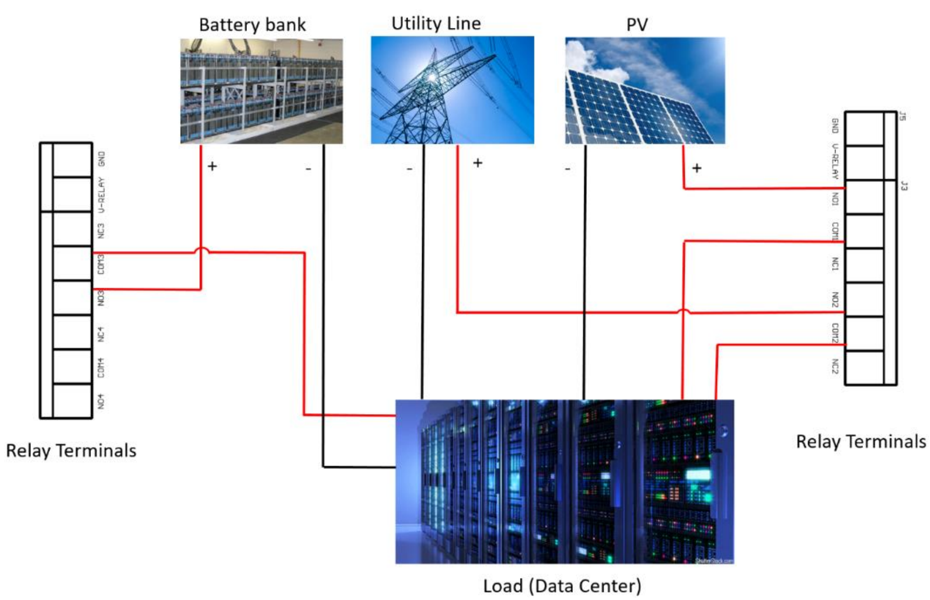

Figure 4-31 High Level Block Diagram showing Power Sources Connections to the Relay Board

From Figure 4-31, the NO1 (Normally Open) terminal is connected to the positive line that is coming from the MPPT, and the COM1 (Common) is connected to the load. Also, NO2 terminal is connected to the phase line coming from the AC grid, and the COM2 is connected to the load. The last two relay terminals NO3 and COM3 are connected to the battery bank.

To connect the relay board to the STM32F7 Discovery kit, the communication pins (J1 IN Port), are labeled as IN1, IN2, and IN3. IN1 is the control input signal connected to a digital input D3 in the microcontroller. IN2 and IN3 are also control signals connected to D2 and D4 respectively. For the MCU to send the command to the relay to be turned ON of OFF, the GPIO HAL function is utilized from the HAL driver as illustrated below:

```
HAL_GPIO_WritePin(RLY_MPPT_GPIO_Port, RLY_MPPT_Pin,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOG, RLY_AC_Pin,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOG, RLY_BATT_Pin,GPIO_PIN_SET);
```

Chapter 5: Hardware Results

A lab-scale construction of the system previously explained in Figure 3-1 was conducted as illustrated in Figure 5-1. Each power source of the system was simulated using a DC power supply (RIGOL DP832 Triple Output Power Supply). Since each power supply has three outputs 30V/3A, 30V/3A, and 5V/3A, only two of the higher voltage outputs are being used from the first power supply, while the third source comes from one output of the second power supply to meet the design requirements of 24V/4.2A. The two outputs of the first power supply simulate the power coming from the solar MPPT charge controller and the rectified AC grid. The second power supply functions to simulate the power coming from the battery. A diode connects to the positive side of each power supply to prevent the current flowing back to the outputs of the other power supplies when one power supply is conducting. These diodes are marked as D1, D2, D3 in Figure 5-1(NTE5812) and rated at 6A. For D1, the cathode of the diode connects to a voltage divider (V1) and current sensor (I1) to measure the power coming from the PV. These two sensors are connected to the MCU to control the relay (RLY_MPPT) based on their values. The operation of the relays follows the assumptions as described in Chapter 3 and Chapter 4. Moreover, the cathode of D2 connects to a voltage divider (V2) to simulate the power coming from the grid. The voltage divider informs the MCU whether or not the AC grid is available. As explained in Chapter 4, if a voltage exists at the output of the divider, then the AC grid is available to provide power to the load. This connection operates based on the assumption made in Chapter 4 via a relay (RLY_AC). The last line is the line connected to D3 to simulate the battery. To determine the status of the battery, the line connects to a voltage divider (V3). If the

MCU detects any voltage at the output of the divider, then the battery is available. The power coming from the battery is controlled by the relay connected to D3 (RLY_BAT).

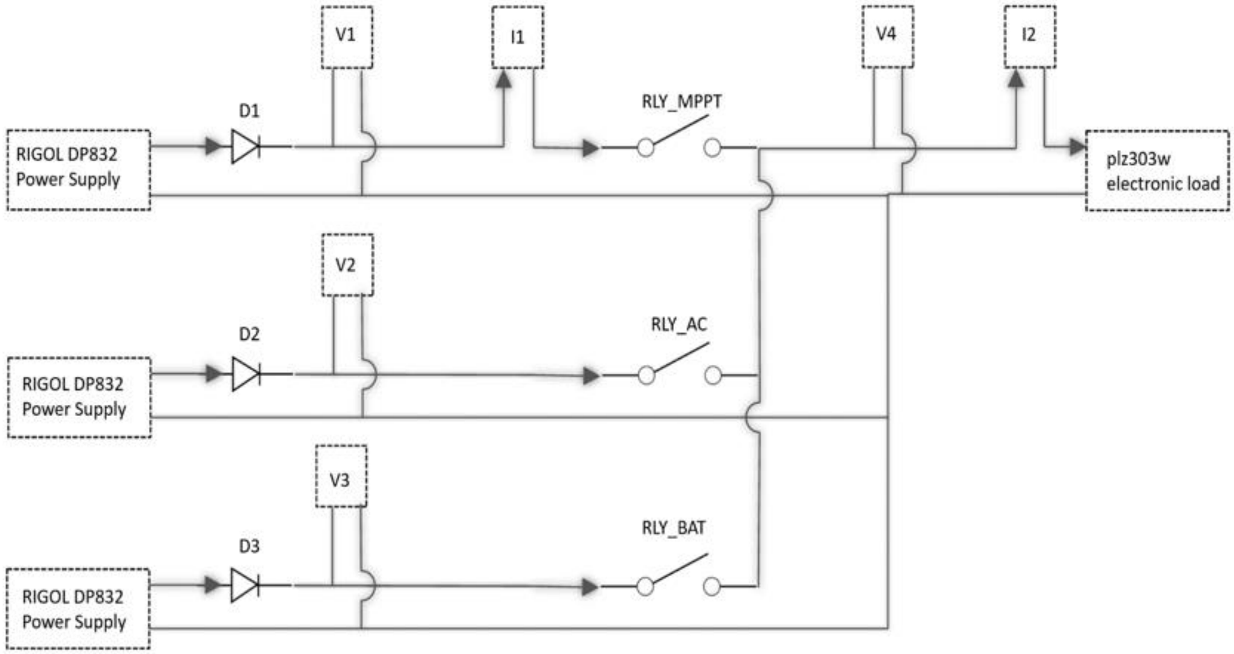

Figure 5-1 Lab-Scale wiring diagram

Again, the STM32F7 Microcontroller controls all relays by following the assumptions made in Chapter 3. These relays connect to an electronic load to simulate the load (PLZ303W Electronic Load). This load-current ties to a voltage divider (V4) and a current sensor (I2) to measure the power at the load. Based on the two values measured via these two sensors the microcontroller controls the operation of the system.

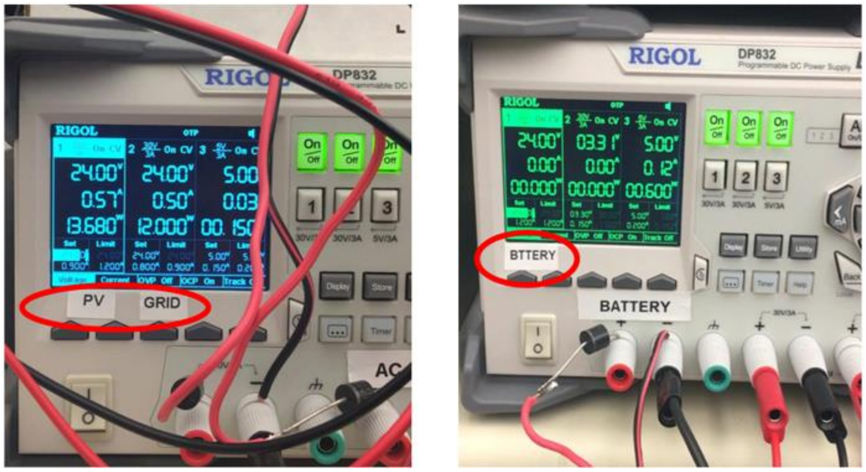

Figure 5-2 RIGOL DP832 triple output power supply used for test setup

Figure 5-3 PLZ303W electronic load to simulate the load

In order to analyze the results and to demonstrate the functionality and performance of the proposed system, the lab setup illustrated in Figure 5-4 was assembled. Additionally, several cases were tested whose results were captured from the LCD of the STM32F7 microcontroller. These cases are as follows:

- Case #1: getting the microcontroller to combine the power from the PV and the AC grid
- Case #2: powering the load from the PV system only

- Case #3: connecting the battery to the load when the grid and the PV are not available
- Case #4: supplying the load from the grid at night
- Case #5: using the batteries as a backup at night.

Figure 5-4 Lab setup

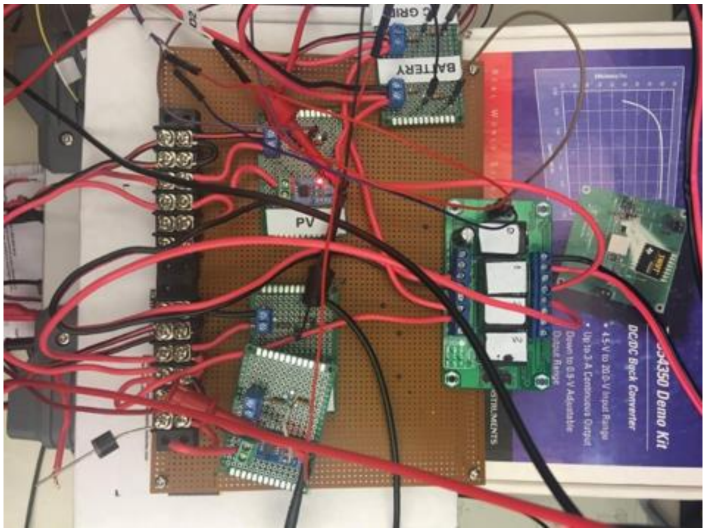

Figure 5-5 Lab setup zoomed in

Case #1

When simulating the first case, the LDR was exposed to light, so the MCU is automatically set as “day-time” as shown in Figure 5-6. In this case, the load power was adjusted from the electronic load to 24 W and the PV system power was adjusted from the DC power supply to 14 W. Since the generation from the PV is less than the power required by the load, the MCU will automatically turn on the two relays connected to the AC grid and the MPPT charge controller. Therefore, the AC grid will compensate the remaining 10 W needed to power the load. Figure 5-6 displays the STM32F7 microcontroller screen indicating that the PV system and the AC grid are both connected. Further, we can see that the battery is available and can be used as a backup if needed.

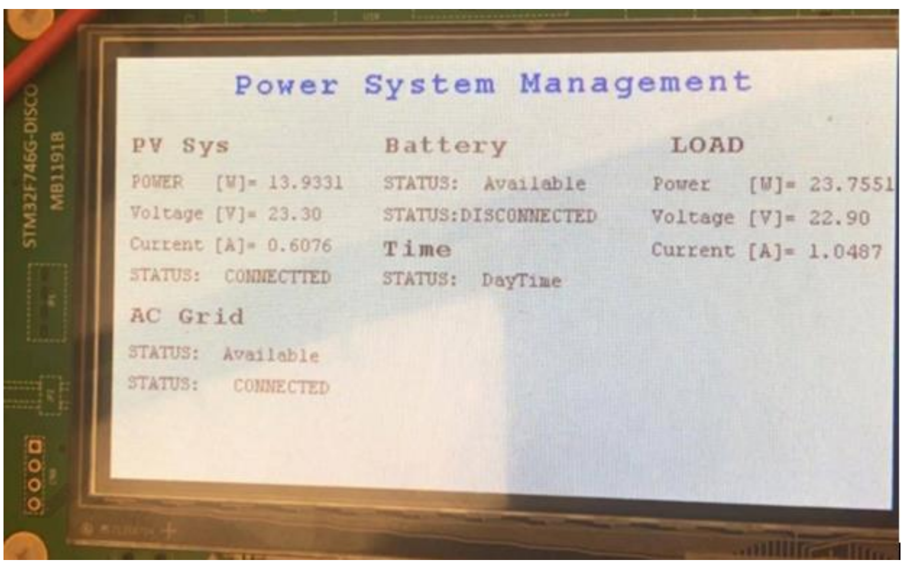

Figure 5-6 Case 1: AC Grid and PV are combined

Case #2

In this case, the microcontroller remained set at “day-time”. The load was adjusted from the electronic load to 18W and the PV system was adjusted from the DC power supply to 24W. Since the load only needs 18W of power, and the PV system was sufficient to supply the needed power to the load; therefore, the MCU automatically turned off the relay connected to the grid and the relay from the PV charge controller remained connected. As you can see in Figure 5-7, the AC grid is available but disconnected, while the PV system is the only source of power connected. As mentioned above the battery will still serve as a backup power source if needed.

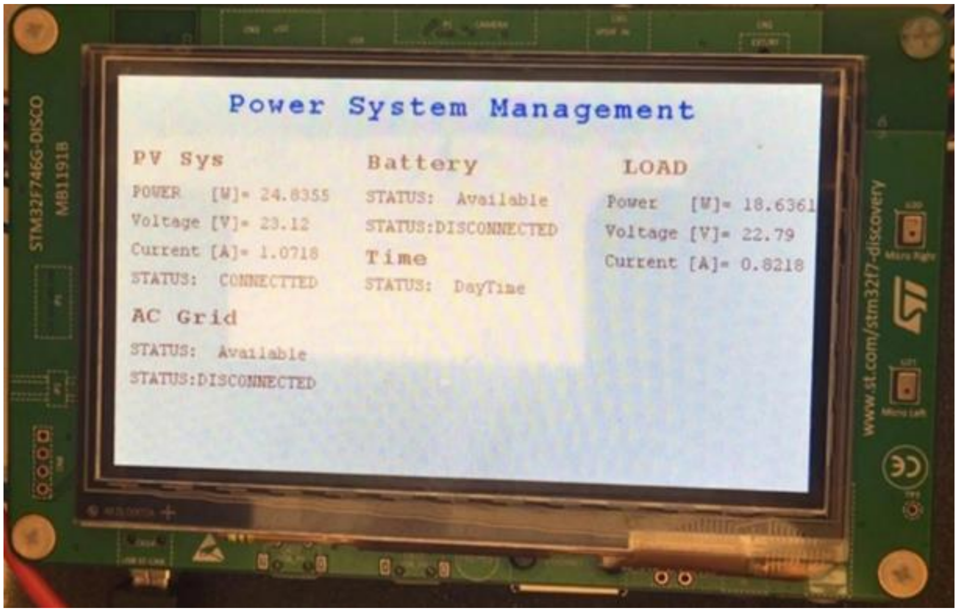

Figure 5-7 Case 2: PV is Supplying the load

Case #3

With the microcontroller still functioning at day-time, the PV and the AC system were both set to 0W. Because the two sensors at the PV line did not send any readings to the microcontroller, the relay connected to the PV was automatically turned off. Further, the voltage sensor that was placed in the positive line coming from the AC grid did not measure any voltage across its terminals, the MCU also automatically disconnected that relay. As shown in Figure 5-8, the PV system and the AC grid are not available and disconnected. The battery is the only source of power that is available and connected to the system.

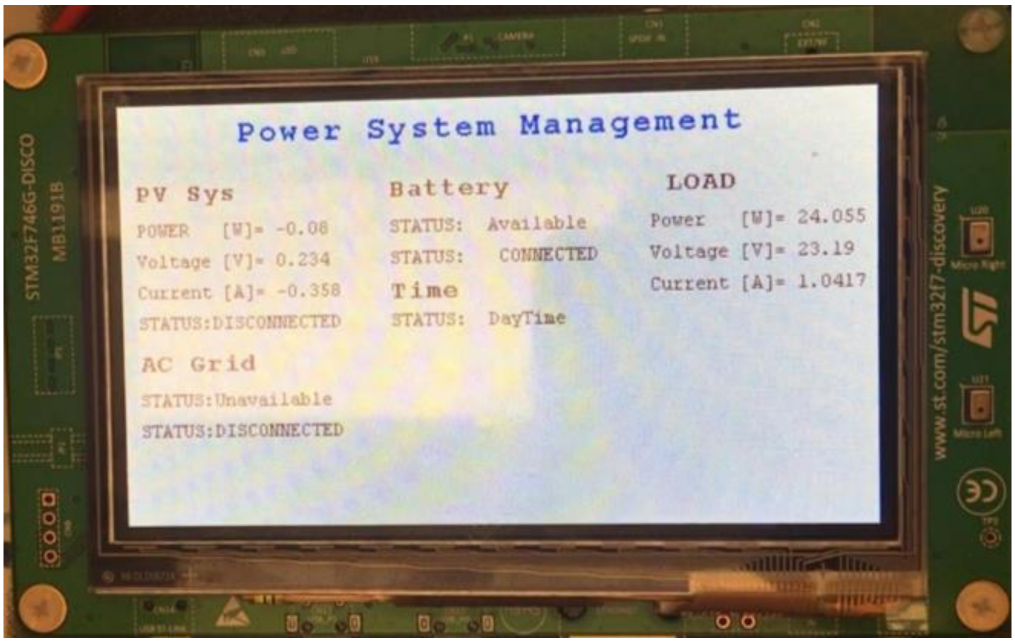

Figure 5-8 Case 3: Battery is connected as backup during the day

Case #4

In conducting this case, the LDR sensor was covered to notify the MCU that it is “night time”. During this setting, the MCU will read 0 power coming from the PV; therefore the PV is automatically disconnected from the network. The MCU will fully rely on the AC grid power source. The relay connected to the grid will be turned on to deliver the power to the load. As shown in Figure 5-9, the PV system is disconnected and the AC grid is available and connected to the system. Again the battery will remain as a backup power source if needed.

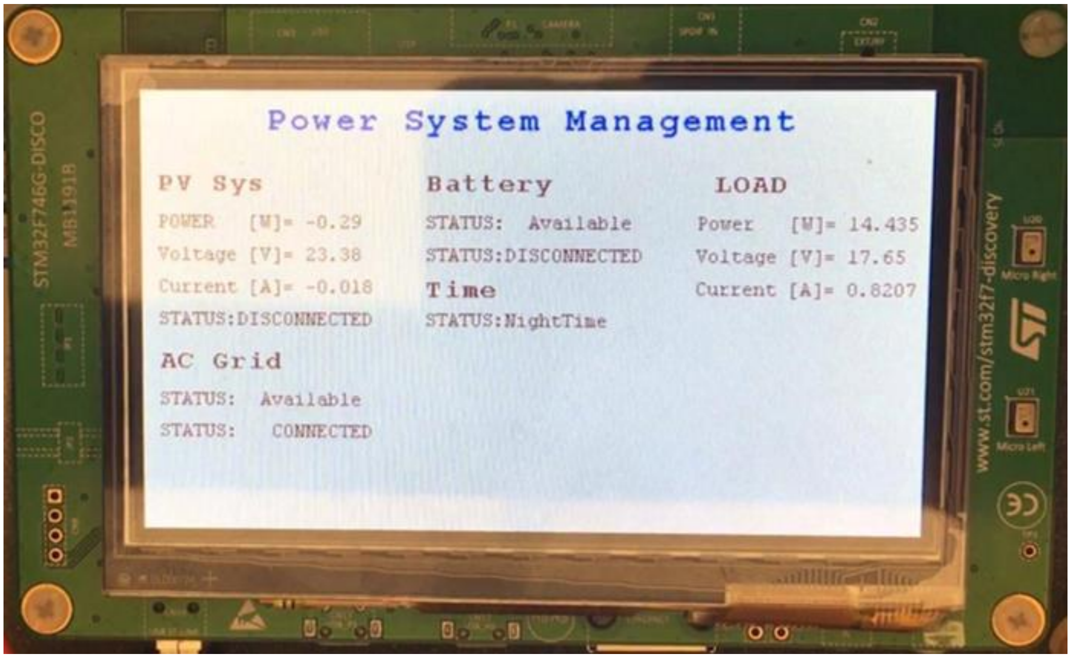

Figure 5-9 Case 4: AC Grid is supplying load

Case #5

The purpose of this case was to test the backup power supply during night time. The power supply that was assumed to be the rectifier AC grid was turned off, leaving the backup battery as the only power source available to feed the load. Figure 5-10 displays both AC and PV as not available and not connected. the backup battery as a source that is available and connected to the system.

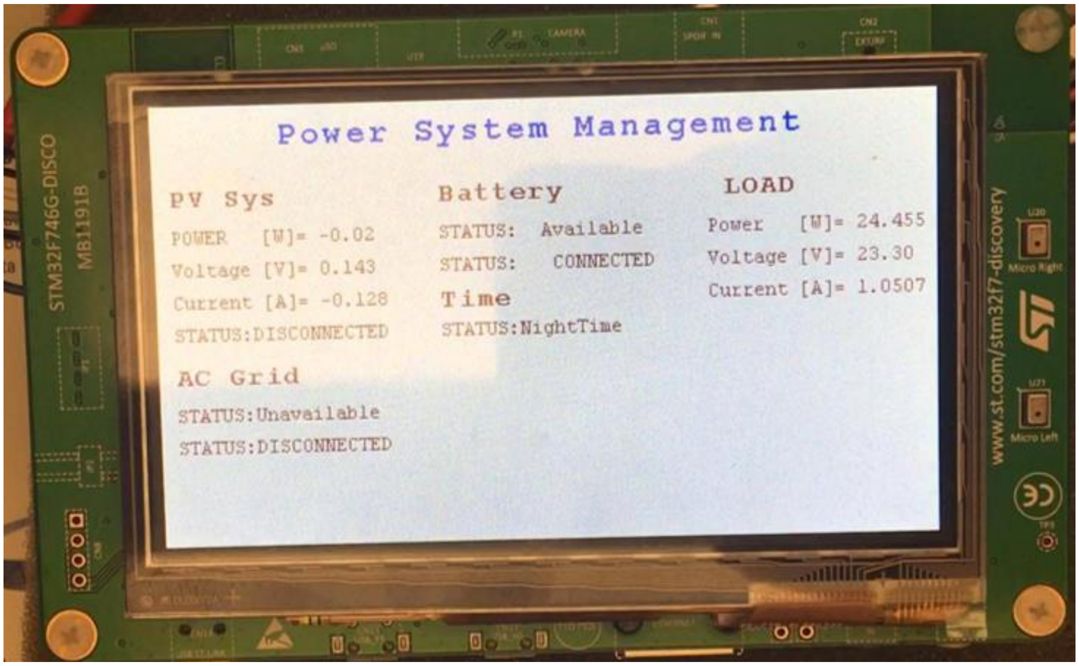

Figure 5-10 Case 5: Battery is connected as backup at night

Chapter 6: Conclusion and Future Work

This thesis presents the hardware and software for modeling DC data center using hybrid power sources that combines the utility grid with a renewable energy source (PV system). The proposed design topology was first simulated before implementing the design in the lab. The results from the simulation and the neural network model of the system as elaborated in Chapter 4 demonstrate the accuracy in the operation of such mix of energy sources and hence proved the possibility of combining different power sources. However, the assumptions implemented when developing the software could be an improvement in future work. Voltage and current sensors could be employed to the line coming from the utility grid for monitoring the power. Then the measured power value could be used as an input to the NN. In addition, the battery state of charge could be measured and used as an input when developing the NN model. These additional requirements consequently will demand the use of a different microcontroller with more analog to digital ports.

Furthermore, the hardware design was constructed using two DC power supplies to characterize each power source. The goal was to demonstrate a proof-of-concept lab-scale system rather than testing the concept in a real system. When interfacing the microcontroller with the sensors and the relay board, the result demonstrated accuracy with the software. Also, the NN model worked very well as looked-for when developing the model. However, further work could be done to achieve a much closer lab-scale design to the real system. Instead of using DC power supplies to simulate the energy sources, a solar panel, the utility grid, and lead-acid batteries could be used. Additionally, the assumption that the battery bank could only be charged from the PV system may be improved by allowing the utility line to charge the battery, in case PV power is not enough or not available.

REFERENCES

[1] “Solar Market Insight Report 2016 Q2,” *SEIA*. [Online]. Available: http://www.seia.org/research-resources/solar-market-insight-report-2016-q2. [Accessed: 06-Feb-2017].

[2] “Tesla vs Edison: the war of currents.” [Online]. Available: http://www.abb.com/cawp/seitp202/c646c16ae1512f8ec1257934004fa545.aspx. [Accessed: 06-Feb-2017].

[3] “City is replacing 70% of street lights with LED lighting — expects to cut energy use by 50% - Electronic Products.” [Online]. Available: http://www.electronicproducts.com/Optoelectronics/LEDs/City_is_replacing_70_of_street_lights_with_LED_lighting_expects_to_cut_energy_use_by_50.aspx. [Accessed: 06-Feb-2017].

[4] “How Energy-Efficient Light Bulbs Compare with Traditional Incandescents | Department of Energy.” [Online]. Available: https://energy.gov/energysaver/how-energy-efficient-light-bulbs-compare-traditional-incandescents. [Accessed: 06-Feb-2017].

[5] X. Wang, M. Li, and Z. Wang, “Research and application of green power system for new data centers,” in *Telecommunications Energy Conference (INTELEC), 2015 IEEE International*, 2015, pp. 1–6.

[6] S. Gan, G. Chen, and X. Li, “Energy and Power Reduction of Cloud Data Centers.,” *Metallurgical & Mining Industry*, no. 10, 2015.

[7] L. Ganesh, H. Weatherspoon, T. Marian, and K. Birman, “Integrated Approach to Data Center Power Management,” *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1086–1096, Jun. 2013.

[8] “Microsoft now has one million servers - less than Google, but more than Amazon, says Ballmer - ExtremeTech.” [Online]. Available: https://www.extremetech.com/extreme/161772-microsoft-now-has-one-million-servers-less-than-google-but-more-than-amazon-says-ballmer. [Accessed: 06-Feb-2017].

[9] “scaling up energy efficiency across the Data Center Industry: evaluating Key Drivers and Barriers,” no. ISSUE PAPER, p. 35, Aug. 2014. *

[10] F. Xu, B. Guo, L. M. Tolbert, F. Wang, and B. J. Blalock, "An All-SiC Three-Phase Buck Rectifier for High-Efficiency Data Center Power Supplies," *IEEE Transactions on Industry Applications*, vol. 49, no. 6, pp. 2662–2673, Nov. 2013.

[11] A. Pratt, P. Kumar, and T. V. Aldridge, "Evaluation of 400V DC distribution in telco and data centers to improve energy efficiency," in *INTELEC 07 - 29th International Telecommunications Energy Conference*, 2007, pp. 32–39.

[12] M. Murrill and B. J. Sonnenberg, "Evaluating the opportunity for dc power in the data center," *Emerson Network Power White Paper*, 2011.

[13] S. Mondal and E. Keisling, "Efficient data center design using novel modular DC UPS, server power supply with DC voltage and modular CDU cooling," in *Power Electronics, Drives and Energy Systems (PEDES), 2012 IEEE International Conference on*, 2012, pp. 1–6.

[14] A. Pratt, P. Kumar, and T. V. Aldridge, "Evaluation of 400V DC distribution in telco and data centers to improve energy efficiency," in *Telecommunications Energy Conference, 2007. INTELEC 2007. 29th International*, 2007, pp. 32–39.

[15] R. Simanjorang *et al.*, "High-efficiency high-power dc-dc converter for energy and space saving of power-supply system in a data center," in *Applied Power Electronics Conference and Exposition (APEC), 2011 Twenty-Sixth Annual IEEE*, 2011, pp. 600–605.

[16] D. P.Symanski, "Why Not Operate Data Centers & Telecom Central Offices at 400 VDC???," p. 20, Oct. 2009.

[17] "Environmental Responsibility Report." Apple, 2016.

[18] Í. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Designing and managing data centers powered by renewable energy," *IEEE Micro*, vol. 34, no. 3, pp. 8–16, 2014.

[19] "Digital States, Voltage Levels, and Logic Families - National Instruments." [Online]. Available: http://www.ni.com/white-paper/3292/en/. [Accessed: 07-Feb-2017].

Appendix A: Neural Network Model Table

Input to the NN		Target Values		Output (NN Result)	
Load in W	PV in W	PV On or Off	AC On or Off	PV On or Off	AC On or Off
100	100	1	0	1	0
100	95	1	1	1	1
100	90	1	1	1	1
100	85	1	1	1	1
100	80	1	1	1	1
100	75	1	1	1	1
100	70	1	1	1	1
100	65	1	1	1	1
100	60	1	1	1	1
100	55	1	1	1	1
100	50	1	1	1	1
100	45	1	1	1	1
100	40	1	1	1	1
100	35	1	1	1	1
100	30	1	1	1	1
100	25	1	1	1	1
100	20	1	1	1	1
100	15	1	1	1	1
100	10	1	1	1	1
100	0.5	1	1	1	1
95	100	1	0	1	(0)

95	95	1	0	1	0
95	90	1	1	1	1
95	85	1	1	1	1
95	80	1	1	1	1
95	75	1	1	1	1
95	70	1	1	1	1
95	65	1	1	1	1
95	60	1	1	1	1
95	55	1	1	1	1
95	50	1	1	1	1
95	45	1	1	1	1
95	40	1	1	1	1
95	35	1	1	1	1
95	30	1	1	1	1
95	25	1	1	1	1
95	20	1	1	1	1
95	15	1	1	1	1
95	10	1	1	1	1
95	0.5	1	1	1	1
90	100	1	0	1	0
90	95	1	0	1	(0)
90	90	1	0	1	0
90	85	1	1	1	1
90	80	1	1	1	1
90	75	1	1	1	1

90	70	1	1	1	1
90	65	1	1	1	1
90	60	1	1	1	1
90	55	1	1	1	1
90	50	1	1	1	1
90	45	1	1	1	1
90	40	1	1	1	1
90	35	1	1	1	1
90	30	1	1	1	1
90	25	1	1	1	1
90	20	1	1	1	1
90	15	1	1	1	1
90	10	1	1	1	1
90	0.5	1	1	1	1
85	100	1	0	1	0
85	95	1	0	1	0
85	90	1	0	1	(0)
85	85	1	0	1	0
85	80	1	1	1	1
85	75	1	1	1	1
85	70	1	1	1	1
85	65	1	1	1	1
85	60	1	1	1	1
85	55	1	1	1	1
85	50	1	1	1	1

85	45	1	1	1	1
85	40	1	1	1	1
85	35	1	1	1	1
85	30	1	1	1	1
85	25	1	1	1	1
85	20	1	1	1	1
85	15	1	1	1	1
85	10	1	1	1	1
85	0.5	1	1	1	1
80	100	1	0	1	(0)
80	95	1	0	1	0
80	90	1	0	1	0
80	85	1	0	1	(0)
80	80	1	0	1	0
80	75	1	1	1	1
80	70	1	1	1	1
80	65	1	1	1	1
80	60	1	1	1	1
80	55	1	1	1	1
80	50	1	1	1	1
80	45	1	1	1	1
80	40	1	1	1	1
80	35	1	1	1	1
80	30	1	1	1	1
80	25	1	1	1	1

80	20	1	1	1	1
80	15	1	1	1	1
80	10	1	1	1	1
80	0.5	1	1	1	1
75	100	1	0	1	(0)
75	95	1	0	1	(0)
75	90	1	0	1	0
75	85	1	0	1	0
75	80	1	0	1	(0)
75	75	1	0	1	0
75	70	1	1	1	1
75	65	1	1	1	1
75	60	1	1	1	1
75	55	1	1	1	1
75	50	1	1	1	1
75	45	1	1	1	1
75	40	1	1	1	1
75	35	1	1	1	1
75	30	1	1	1	1
75	25	1	1	1	1
75	20	1	1	1	1
75	15	1	1	1	1
75	10	1	1	1	1
75	0.5	1	1	1	1
70	100	1	0	1	(0)

70	95	1	0	1	(0)
70	90	1	0	1	(0)
70	85	1	0	1	0
70	80	1	0	1	0
70	75	1	0	1	(0)
70	70	1	0	1	0
70	65	1	1	1	1
70	60	1	1	1	1
70	55	1	1	1	1
70	50	1	1	1	1
70	45	1	1	1	1
70	40	1	1	1	1
70	35	1	1	1	1
70	30	1	1	1	1
70	25	1	1	1	1
70	20	1	1	1	1
70	15	1	1	1	1
70	10	1	1	1	1
70	0.5	1	1	1	1
65	100	1	0	1	(0)
65	95	1	0	1	(0)
65	90	1	0	1	(0)
65	85	1	0	1	(0)
65	80	1	0	1	0
65	75	1	0	1	0

65	70	1	0	1	(0)
65	65	1	0	1	0
65	60	1	1	1	1
65	55	1	1	1	1
65	50	1	1	1	1
65	45	1	1	1	1
65	40	1	1	1	1
65	35	1	1	1	1
65	30	1	1	1	1
65	25	1	1	1	1
65	20	1	1	1	1
65	15	1	1	1	1
65	10	1	1	1	1
65	0.5	1	1	1	1
60	100	1	0	1	(0)
60	95	1	0	1	(0)
60	90	1	0	1	(0)
60	85	1	0	1	(0)
60	80	1	0	1	(0)
60	75	1	0	1	0
60	70	1	0	1	0
60	65	1	0	1	(0)
60	60	1	0	1	0
60	55	1	1	1	1
60	50	1	1	1	1

60	45	1	1	1	1
60	40	1	1	1	1
60	35	1	1	1	1
60	30	1	1	1	1
60	25	1	1	1	1
60	20	1	1	1	1
60	15	1	1	1	1
60	10	1	1	1	1
60	0.5	1	1	1	1
55	100	1	0	1	(0)
55	95	1	0	1	(0)
55	90	1	0	1	(0)
55	85	1	0	1	(0)
55	80	1	0	1	(0)
55	75	1	0	1	(0)
55	70	1	0	1	0
55	65	1	0	1	0
55	60	1	0	1	(0)
55	55	1	0	1	0
55	50	1	1	1	1
55	45	1	1	1	1
55	40	1	1	1	1
55	35	1	1	1	1
55	30	1	1	1	1
55	25	1	1	1	1

55	20	1	1	1	1
55	15	1	1	1	1
55	10	1	1	1	1
55	0.5	1	1	1	1
50	100	1	0	1	(0)
50	95	1	0	1	(0)
50	90	1	0	1	(0)
50	85	1	0	1	(0)
50	80	1	0	1	(0)
50	75	1	0	1	(0)
50	70	1	0	1	(0)
50	65	1	0	1	0
50	60	1	0	1	0
50	55	1	0	1	(0)
50	50	1	0	1	0
50	45	1	1	1	1
50	40	1	1	1	1
50	35	1	1	1	1
50	30	1	1	1	1
50	25	1	1	1	1
50	20	1	1	1	1
50	15	1	1	1	1
50	10	1	1	1	1
50	0.5	1	1	1	1
45	100	1	0	1	(0)

45	95	1	0	1	(0)
45	90	1	0	1	(0)
45	85	1	0	1	(0)
45	80	1	0	1	(0)
45	75	1	0	1	(0)
45	70	1	0	1	(0)
45	65	1	0	1	(0)
45	60	1	0	1	0
45	55	1	0	1	0
45	50	1	0	1	(0)
45	45	1	0	1	0
45	40	1	1	1	1
45	35	1	1	1	1
45	30	1	1	1	1
45	25	1	1	1	1
45	20	1	1	1	1
45	15	1	1	1	1
45	10	1	1	1	1
45	0.5	1	1	1	1
40	100	1	0	1	0
40	95	1	0	1	(0)
40	90	1	0	1	(0)
40	85	1	0	1	(0)
40	80	1	0	1	(0)
40	75	1	0	1	(0)

40	70	1	0	1	(0)
40	65	1	0	1	(0)
40	60	1	0	1	(0)
40	55	1	0	1	0
40	50	1	0	1	0
40	45	1	0	1	(0)
40	40	1	0	1	0
40	35	1	1	1	1
40	30	1	1	1	1
40	25	1	1	1	1
40	20	1	1	1	1
40	15	1	1	1	1
40	10	1	1	1	1
40	0.5	1	1	1	1
35	100	1	0	1	0
35	95	1	0	1	0
35	90	1	0	1	(0)
35	85	1	0	1	(0)
35	80	1	0	1	(0)
35	75	1	0	1	(0)
35	70	1	0	1	(0)
35	65	1	0	1	(0)
35	60	1	0	1	(0)
35	55	1	0	1	(0)
35	50	1	0	1	0

35	45	1	0	1	0
35	40	1	0	1	(0)
35	35	1	0	1	0
35	30	1	1	1	1
35	25	1	1	1	1
35	20	1	1	1	1
35	15	1	1	1	1
35	10	1	1	1	1
35	0.5	1	1	1	1
30	100	1	0	1	0
30	95	1	0	1	0
30	90	1	0	1	0
30	85	1	0	1	(0)
30	80	1	0	1	(0)
30	75	1	0	1	(0)
30	70	1	0	1	(0)
30	65	1	0	1	(0)
30	60	1	0	1	(0)
30	55	1	0	1	(0)
30	50	1	0	1	(0)
30	45	1	0	1	0
30	40	1	0	1	0
30	35	1	0	1	(0)
30	30	1	0	1	0
30	25	1	1	1	1

30	20	1	1	1	1
30	15	1	1	1	1
30	10	1	1	1	1
30	0.5	1	1	1	1
25	100	1	0	1	0
25	95	1	0	1	0
25	90	1	0	1	0
25	85	1	0	1	0
25	80	1	0	1	(0)
25	75	1	0	1	(0)
25	70	1	0	1	(0)
25	65	1	0	1	(0)
25	60	1	0	1	(0)
25	55	1	0	1	(0)
25	50	1	0	1	(0)
25	45	1	0	1	(0)
25	40	1	0	1	0
25	35	1	0	1	0
25	30	1	0	1	(0)
25	25	1	0	1	0
25	20	1	1	1	1
25	15	1	1	1	1
25	10	1	1	1	1
25	0.5	1	1	1	1
20	100	1	0	1	0

20	95	1	0	1	0
20	90	1	0	1	0
20	85	1	0	1	0
20	80	1	0	1	0
20	75	1	0	1	(0)
20	70	1	0	1	(0)
20	65	1	0	1	(0)
20	60	1	0	1	(0)
20	55	1	0	1	(0)
20	50	1	0	1	(0)
20	45	1	0	1	(0)
20	40	1	0	1	(0)
20	35	1	0	1	0
20	30	1	0	1	0
20	25	1	0	1	(0)
20	20	1	0	1	0
20	15	1	1	1	1
20	10	1	1	1	1
20	0.5	1	1	1	1
15	100	1	0	1	0
15	95	1	0	1	0
15	90	1	0	1	0
15	85	1	0	1	0
15	80	1	0	1	0
15	75	1	0	1	0

15	70	1	0	1	(0)
15	65	1	0	1	(0)
15	60	1	0	1	(0)
15	55	1	0	1	(0)
15	50	1	0	1	(0)
15	45	1	0	1	(0)
15	40	1	0	1	(0)
15	35	1	0	1	(0)
15	30	1	0	1	0
15	25	1	0	1	0
15	20	1	0	1	(0)
15	15	1	0	1	0
15	10	1	1	1	1
15	0.5	1	1	1	1
10	100	1	0	1	0
10	95	1	0	1	0
10	90	1	0	1	0
10	85	1	0	1	0
10	80	1	0	1	0
10	75	1	0	1	0
10	70	1	0	1	0
10	65	1	0	1	(0)
10	60	1	0	1	(0)
10	55	1	0	1	(0)
10	50	1	0	1	(0)

10	45	1	0	1	(0)
10	40	1	0	1	(0)
10	35	1	0	1	(0)
10	30	1	0	1	(0)
10	25	1	0	1	0
10	20	1	0	1	0
10	15	1	0	1	(0)
10	10	1	0	1	0
10	0.5	1	1	1	1
0.5	100	1	0	1	0
0.5	95	1	0	1	0
0.5	90	1	0	1	0
0.5	85	1	0	1	0
0.5	80	1	0	1	0
0.5	75	1	0	1	0
0.5	70	1	0	1	0
0.5	65	1	0	1	0
0.5	60	1	0	1	0
0.5	55	1	0	1	(0)
0.5	50	1	0	1	(0)
0.5	45	1	0	1	(0)
0.5	40	1	0	1	(0)
0.5	35	1	0	1	(0)
0.5	30	1	0	1	(0)
0.5	25	1	0	1	(0)

0.5	20	1	0	1	(0)
0.5	15	1	0	1	0
0.5	10	1	0	1	0
0.5	0.5	1	0	1	0

Appendix B: STM32F7 C Code in Keil Software

C:\Users\Khalid\Desktop\Thesis\RealSystem\Src\main.c

```
/**
  ******************************************************************************
  * File Name          : main.c
  * Description        : Main program body
  ******************************************************************************
  *
  * COPYRIGHT(c) 2016 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include "stm32f7xx_hal.h"

/* USER CODE BEGIN Includes */
#include "stdbool.h"


#include "stm32746g_discovery_lcd.h"
#include <string.h>
#define RGB565_BYTE_PER_PIXEL     2      //
#define ARBG8888_BYTE_PER_PIXEL   4      //

#define LCD_FRAME_BUFF1       SDRAM_DEVICE_ADDR
#define LCD_FRAME_BUFF2       ((uint32_t)(LCD_FRAME_BUFF1 + (RK043FN48H_WIDTH * RK043FN48H_HEIGHT * ARBG8888_BYTE_PER_PIXEL)))

#define LTDC_LAYER_1       ((uint32_t)1) /* Layer 1 */
#define LTDC_LAYER_2       ((uint32_t)2) /* Layer 2 */



/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc3;
DMA_HandleTypeDef hdma_adc3;

DMA2D_HandleTypeDef hdma2d;

LTDC_HandleTypeDef hltdc;

UART_HandleTypeDef huart1;

SDRAM_HandleTypeDef hsdram1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
#define STR_BUFFER_SIZE 128
char BUFFER_STR[STR_BUFFER_SIZE];
uint16_t ADC_BUFFER[6];
```

```c
uint32_t nilaiX,nilaiY;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC3_Init(void);
static void MX_DMA2D_Init(void);
static void MX_FMC_Init(void);
static void MX_LTDC_Init(void);
static void MX_USART1_UART_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */
#include <stdio.h>
#include "SysIntgration.h"             /* Model's header file */
#include "rtwtypes.h"                  /* MathWorks types */
float my_Inp1,my_Inp2;
float my_Out1,my_Out2,my_Out1R,my_Out2R;
double SW_MPPT, SW_AC;
double MPPTP, LOADP;
double LOADV, LOADI, MPPTV, MPPTI;
bool RLY_MPPT, RLY_BATT, RLY_AC, STS_Day, STS_AC, STS_BAT;
double SW_MPPT, SW_AC;
double P_MPPT, P_LOAD, limit=0.5;
double  STS_ACval,STS_BATval;
bool RLY_MPPT, RLY_BATT, RLY_AC, STS_Day, STS_AC, STS_BATT,STS_MPPT;





/* Real-time model */
extern RT_MODEL_SysIntgration *const SysIntgration_M;

/* Set which subrates need to run this base step (base rate always runs).*/
/* Defined in SysIntgration.c file */
extern void SysIntgration_SetEventsForThisBaseStep(boolean_T*);

/* Flags for taskOverrun */
static boolean_T OverrunFlags[1];

/* Number of auto reload timer rotation computed */
static uint32_t autoReloadTimerLoopVal_S = 1;

/* Remaining number of auto reload timer rotation to do */
static uint32_t remainAutoReloadTimerLoopVal_S = 1;

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */
  /* Data initialization */
  int_T i;
  uint8_t display0;

  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/

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

  /* Configure the system clock */
```

```c
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC3_Init();
  MX_DMA2D_Init();
  MX_FMC_Init();
  MX_LTDC_Init();
  MX_USART1_UART_Init();

  /* USER CODE BEGIN 2 */
  /* Systick configuration and enable SysTickHandler interrupt */
  if (SysTick_Config((uint32_t)(SystemCoreClock * 0.001))) {
    autoReloadTimerLoopVal_S = 1;
    do {
      autoReloadTimerLoopVal_S++;
    } while ((uint32_t)(SystemCoreClock * 0.001)/autoReloadTimerLoopVal_S >
             SysTick_LOAD_RELOAD_Msk);

    SysTick_Config((uint32_t)(SystemCoreClock * 0.001)/autoReloadTimerLoopVal_S);
  }

  remainAutoReloadTimerLoopVal_S = autoReloadTimerLoopVal_S;


  BSP_LCD_Init(); //Initialize the LCD
  BSP_LCD_LayerDefaultInit(LTDC_LAYER_1, LCD_FRAME_BUFF1); //apply the layer configuration
  BSP_LCD_SelectLayer(LTDC_LAYER_1);    //Select the LCD layer to be used
  BSP_LCD_SetBackColor(LCD_COLOR_WHITE);  //Set the background color
  BSP_LCD_Clear(LCD_COLOR_WHITE); // Clrear the whole LCD
  BSP_LCD_DisplayOn(); // Enable the LCD display

  BSP_LCD_SetFont(&Font20); //Set the font size
  BSP_LCD_SetTextColor(LCD_COLOR_BLUE);               //Select the color of the text
  BSP_LCD_DisplayStringAt(0, 10, (uint8_t *)"Power System Management", CENTER_MODE );  //Display string line


  HAL_DMA_Init(&hdma_adc3);         // ADC
  HAL_ADC_Start_DMA(&hadc3, (uint32_t*)ADC_BUFFER, 6);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  for (i=0;i<1;i++) {
    OverrunFlags[i] = 0;
  }

  /* Model initialization call */
  SysIntgration_initialize();                     //My NN

  /* Infinite loop */
  /* Real time from systickHandler */

      RLY_MPPT=SET;
      HAL_Delay(1000);

  while (1) {
//    if (remainAutoReloadTimerLoopVal_S == 0) {
//      remainAutoReloadTimerLoopVal_S = autoReloadTimerLoopVal_S;

//      /* Check base rate for overrun */
//      if (OverrunFlags[0]) {
//        rtmSetErrorStatus(SysIntgration_M, "Overrun");
//      }

//      OverrunFlags[0] = true;

//      /* Step the model for base rate */
//      SysIntgration_step();
```

```
//      /* Get model outputs here */

//      /* Indicate task for base rate complete */
//      OverrunFlags[0] = false;
//    }

          // -------------Read Power
        //Power MPPT
        for(i=0;i<1000;i++)
        {

        MPPTI=(float)ADC_BUFFER[0]*3.3/4096;            //Analog Read
        MPPTI=(MPPTI-2.5)/0.066;
        MPPTV=(float)(ADC_BUFFER[1]*3.3/4096)*(7.8);            //Analog Read
        MPPTP=MPPTV*MPPTI;

        if (MPPTV>=24) STS_MPPT=SET;
        if (MPPTV<20) STS_MPPT=RESET;
//
//        //Power LOAD
        LOADV=(float) (ADC_BUFFER[2]*3.3/4096)*7.8;          //Analog Read
        LOADI=(float)ADC_BUFFER[3]*(3.3/4096);         //Analog Read
        LOADI=(LOADI-2.5)/0.066;
        LOADP=LOADV*LOADI;
        // Dsiplay to LCD

      // -------------Read AC Status
        STS_ACval=(float)ADC_BUFFER[4]*3.3/4096;         //Analog Read
        if (STS_ACval>=1)
        {
          STS_AC=SET;
        }
        if (STS_ACval<1)
        {
          STS_AC=RESET;
        }
      // -------------Read Battery Status
        STS_BATval=(float)ADC_BUFFER[5]*3.3/4095;              //Analog Read
        if (STS_BATval>=1)
        {
          STS_BAT=SET;
        }
        if (STS_BATval<1)
        {
          STS_BAT=RESET;
        }
      }
      // -------------Read DAY Status                Digital Read
        STS_Day=!(HAL_GPIO_ReadPin(GPIOI, STS_DAY_Pin));


        switch (STS_Day)
        {
          case SET:
          {

            if (STS_AC==SET && MPPTP>=limit)
            {

              SysIntgration_U.In1=LOADP;
              SysIntgration_U.In2=MPPTP;

              SysIntgration_step();

              my_Out1=SysIntgration_Y.Out1;
              my_Out2=SysIntgration_Y.Out2;

              my_Out1R=round(my_Out1);
              my_Out2R=round(my_Out2);

            if (my_Out1R==0) RLY_MPPT=RESET;
```

```
            if (my_Out1R==1) RLY_MPPT=SET;
            //  DISP on LCD

            if (my_Out2R==0) RLY_AC=RESET;
            if (my_Out2R==1) RLY_AC=SET;
            // DISP on LCD
            //NN end  Plus delay......
            }


            if (STS_AC==SET && MPPTP<=limit)
            {
              RLY_AC=SET;
              RLY_MPPT=RESET;
              RLY_BATT=RESET;
            }

            if (STS_AC==RESET && MPPTP<=0 && STS_BAT==SET)
            {
              RLY_AC=RESET;
              RLY_MPPT=RESET;
              RLY_BATT=SET;
              //Disp on LCD
            }
            if (STS_AC==RESET && MPPTP<=0 && STS_BAT==RESET)
            {
              RLY_AC=RESET;
              RLY_MPPT=RESET;
              RLY_BATT=RESET;
              //Disp on LCD
              //No Source
            }
            break;
          }

          case RESET:
          {
            if (STS_AC==SET)
            {
              RLY_AC=SET;
              RLY_MPPT=RESET;
              RLY_BATT=RESET;
              //Disp on LCD
            }
            if (STS_AC==RESET && STS_BAT==SET)
            {
              RLY_AC=RESET;
              RLY_MPPT=RESET;
              RLY_BATT=SET;
              //Disp on LCD
            }
            if (STS_AC==RESET && STS_BAT==RESET)
            {
              RLY_AC=RESET;
              RLY_MPPT=RESET;
              RLY_BATT=RESET;
              //Disp on LCD
              //No SOurce
            }

            break;
          }
        }

//      //############ PV System ########################
   for(display0=0;display0<=30;display0++)
  {
  BSP_LCD_SetFont(&Font16);          // Set font size
  BSP_LCD_SetTextColor(LCD_COLOR_BROWN);         //Set text color
  BSP_LCD_DisplayStringAt(10, 50, (uint8_t *)"PV Sys", LEFT_MODE); //set the location in the screen

  sprintf((char *)BUFFER_STR,"POWER   [W]= %3.2f",MPPTP);
```

```c
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(10, 75, (uint8_t *)BUFFER_STR, LEFT_MODE);

    sprintf((char *)BUFFER_STR,"Voltage [V]= %2.2f",MPPTV);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(10, 95, (uint8_t *)BUFFER_STR, LEFT_MODE);

    sprintf((char *)BUFFER_STR,"Current [A]= %2.2f",MPPTI);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(10, 115, (uint8_t *)BUFFER_STR, LEFT_MODE);

    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    if (RLY_MPPT==RESET){
    BSP_LCD_DisplayStringAt(10, 135, (uint8_t *)"STATUS:DISCONNECTED", LEFT_MODE);
    HAL_GPIO_WritePin(RLY_MPPT_GPIO_Port, RLY_MPPT_Pin,GPIO_PIN_RESET);
    }
    if (RLY_MPPT==SET){
    BSP_LCD_DisplayStringAt(10, 135, (uint8_t *)"STATUS:  CONNECTTED", LEFT_MODE);
    HAL_GPIO_WritePin(RLY_MPPT_GPIO_Port, RLY_MPPT_Pin,GPIO_PIN_SET);
    } //
  ////############ AC Grid ########################

    BSP_LCD_SetFont(&Font16);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(10, 160, (uint8_t *)"AC Grid", LEFT_MODE);


    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);     //
    if (STS_AC==RESET)
    BSP_LCD_DisplayStringAt(10, 185, (uint8_t *)"STATUS:Unavailable", LEFT_MODE);
    if (STS_AC==SET)
    BSP_LCD_DisplayStringAt(10, 185, (uint8_t *)"STATUS:  Available", LEFT_MODE);

    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    if (RLY_AC==RESET){
    BSP_LCD_DisplayStringAt(10, 205, (uint8_t *)"STATUS:DISCONNECTED", LEFT_MODE);
    HAL_GPIO_WritePin(GPIOG, RLY_AC_Pin,GPIO_PIN_RESET);
    }
    if (RLY_AC==SET){
    BSP_LCD_DisplayStringAt(10, 205, (uint8_t *)"STATUS:   CONNECTED", LEFT_MODE);
    HAL_GPIO_WritePin(GPIOG, RLY_AC_Pin,GPIO_PIN_SET);
    }
      //
  ////############ Battery ########################
  //
    BSP_LCD_SetFont(&Font16);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(175, 50, (uint8_t *)"Battery", LEFT_MODE);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    if (STS_BAT==RESET)
    BSP_LCD_DisplayStringAt(175, 75, (uint8_t *)"STATUS:Unavailable", LEFT_MODE);
    if (STS_BAT==SET)
    BSP_LCD_DisplayStringAt(175, 75, (uint8_t *)"STATUS:  Available", LEFT_MODE);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);  //
    if (RLY_BATT==RESET){
    BSP_LCD_DisplayStringAt(175, 95, (uint8_t *)"STATUS:DISCONNECTED", LEFT_MODE);
    HAL_GPIO_WritePin(GPIOG, RLY_BATT_Pin,GPIO_PIN_RESET);
    }
    if (RLY_BATT==SET){
    HAL_GPIO_WritePin(GPIOG, RLY_BATT_Pin,GPIO_PIN_SET);
    BSP_LCD_DisplayStringAt(175, 95, (uint8_t *)"STATUS:   CONNECTED", LEFT_MODE);
    }
  ////############ Day or night ########################
    BSP_LCD_SetFont(&Font16);          //
```

```c
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(175, 115, (uint8_t *)"Time", LEFT_MODE);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    if (STS_Day==SET)
    BSP_LCD_DisplayStringAt(175, 135, (uint8_t *)"STATUS:  DayTime", LEFT_MODE);
    if (STS_Day==RESET)
    BSP_LCD_DisplayStringAt(175, 135, (uint8_t *)"STATUS:NightTime", LEFT_MODE);

  //
  ////############ Load #########################
    BSP_LCD_SetFont(&Font16);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(350, 50, (uint8_t *)"LOAD", LEFT_MODE);

    sprintf((char *)BUFFER_STR,"Power   [W]= %3.2f",LOADP);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(340, 75, (uint8_t *)BUFFER_STR, LEFT_MODE);

    sprintf((char *)BUFFER_STR,"Voltage [V]= %2.2f",LOADV);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(340, 95, (uint8_t *)BUFFER_STR, LEFT_MODE);

    sprintf((char *)BUFFER_STR,"Current [A]= %2.2f",LOADI);
    BSP_LCD_SetFont(&Font12);          //
    BSP_LCD_SetTextColor(LCD_COLOR_BROWN);          //
    BSP_LCD_DisplayStringAt(340, 115, (uint8_t *)BUFFER_STR, LEFT_MODE);
  }
    }


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    /* USER CODE END 3 */

  }

  /** System Clock Configuration
  */
  void SystemClock_Config(void)
  {

    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

    __HAL_RCC_PWR_CLK_ENABLE();

    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = 16;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 10;
    RCC_OscInitStruct.PLL.PLLN = 210;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 2;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
      Error_Handler();
    }

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
```

```c
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_LTDC|RCC_PERIPHCLK_USART1;
  PeriphClkInitStruct.PLLSAI.PLLSAIN = 192;
  PeriphClkInitStruct.PLLSAI.PLLSAIR = 2;
  PeriphClkInitStruct.PLLSAI.PLLSAIQ = 2;
  PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC_PLLSAIP_DIV2;
  PeriphClkInitStruct.PLLSAIDivQ = 1;
  PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_2;
  PeriphClkInitStruct.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);


  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC3 init function */
static void MX_ADC3_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
    */
  hadc3.Instance = ADC3;
  hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc3.Init.Resolution = ADC_RESOLUTION_12B;
  hadc3.Init.ScanConvMode = ENABLE;
  hadc3.Init.ContinuousConvMode = ENABLE;
  hadc3.Init.DiscontinuousConvMode = DISABLE;
  hadc3.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc3.Init.NbrOfConversion = 6;
  hadc3.Init.DMAContinuousRequests = ENABLE;
  hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc3) != HAL_OK)
  {
    Error_Handler();
  }

    /**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_144CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = 2;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

```c

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_7;
  sConfig.Rank = 3;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_6;
  sConfig.Rank = 4;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_5;
  sConfig.Rank = 5;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = 6;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* DMA2D init function */
static void MX_DMA2D_Init(void)
{

  hdma2d.Instance = DMA2D;
  hdma2d.Init.Mode = DMA2D_M2M;
  hdma2d.Init.ColorMode = DMA2D_OUTPUT_ARGB8888;
  hdma2d.Init.OutputOffset = 0;
  hdma2d.LayerCfg[1].InputOffset = 0;
  hdma2d.LayerCfg[1].InputColorMode = DMA2D_INPUT_ARGB8888;
  hdma2d.LayerCfg[1].AlphaMode = DMA2D_NO_MODIF_ALPHA;
  hdma2d.LayerCfg[1].InputAlpha = 0;
  if (HAL_DMA2D_Init(&hdma2d) != HAL_OK)
  {
    Error_Handler();
  }

  if (HAL_DMA2D_ConfigLayer(&hdma2d, 1) != HAL_OK)
  {
    Error_Handler();
  }

}

/* LTDC init function */
static void MX_LTDC_Init(void)
{

  LTDC_LayerCfgTypeDef pLayerCfg;
```

```
  LTDC_LayerCfgTypeDef pLayerCfg1;

  hltdc.Instance = LTDC;
  hltdc.Init.HSPolarity = LTDC_HSPOLARITY_AL;
  hltdc.Init.VSPolarity = LTDC_VSPOLARITY_AL;
  hltdc.Init.DEPolarity = LTDC_DEPOLARITY_AL;
  hltdc.Init.PCPolarity = LTDC_PCPOLARITY_IPC;
  hltdc.Init.HorizontalSync = 7;
  hltdc.Init.VerticalSync = 3;
  hltdc.Init.AccumulatedHBP = 14;
  hltdc.Init.AccumulatedVBP = 5;
  hltdc.Init.AccumulatedActiveW = 654;
  hltdc.Init.AccumulatedActiveH = 485;
  hltdc.Init.TotalWidth = 660;
  hltdc.Init.TotalHeigh = 487;
  hltdc.Init.Backcolor.Blue = 0;
  hltdc.Init.Backcolor.Green = 0;
  hltdc.Init.Backcolor.Red = 0;
  if (HAL_LTDC_Init(&hltdc) != HAL_OK)
  {
    Error_Handler();
  }

  pLayerCfg.WindowX0 = 0;
  pLayerCfg.WindowX1 = 0;
  pLayerCfg.WindowY0 = 0;
  pLayerCfg.WindowY1 = 0;
  pLayerCfg.PixelFormat = LTDC_PIXEL_FORMAT_ARGB8888;
  pLayerCfg.Alpha = 0;
  pLayerCfg.Alpha0 = 0;
  pLayerCfg.BlendingFactor1 = LTDC_BLENDING_FACTOR1_CA;
  pLayerCfg.BlendingFactor2 = LTDC_BLENDING_FACTOR2_CA;
  pLayerCfg.FBStartAdress = 0;
  pLayerCfg.ImageWidth = 0;
  pLayerCfg.ImageHeight = 0;
  pLayerCfg.Backcolor.Blue = 0;
  pLayerCfg.Backcolor.Green = 0;
  pLayerCfg.Backcolor.Red = 0;
  if (HAL_LTDC_ConfigLayer(&hltdc, &pLayerCfg, 0) != HAL_OK)
  {
    Error_Handler();
  }

  pLayerCfg1.WindowX0 = 0;
  pLayerCfg1.WindowX1 = 0;
  pLayerCfg1.WindowY0 = 0;
  pLayerCfg1.WindowY1 = 0;
  pLayerCfg1.PixelFormat = LTDC_PIXEL_FORMAT_ARGB8888;
  pLayerCfg1.Alpha = 0;
  pLayerCfg1.Alpha0 = 0;
  pLayerCfg1.BlendingFactor1 = LTDC_BLENDING_FACTOR1_CA;
  pLayerCfg1.BlendingFactor2 = LTDC_BLENDING_FACTOR2_CA;
  pLayerCfg1.FBStartAdress = 0;
  pLayerCfg1.ImageWidth = 0;
  pLayerCfg1.ImageHeight = 0;
  pLayerCfg1.Backcolor.Blue = 0;
  pLayerCfg1.Backcolor.Green = 0;
  pLayerCfg1.Backcolor.Red = 0;
  if (HAL_LTDC_ConfigLayer(&hltdc, &pLayerCfg1, 1) != HAL_OK)
  {
    Error_Handler();
  }

}

/* USART1 init function */
static void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_7B;
```

```c
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA2_Stream0_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}
/* FMC initialization function */
static void MX_FMC_Init(void)
{
  FMC_SDRAM_TimingTypeDef SdramTiming;

  /** Perform the SDRAM1 memory initialization sequence
  */
  hsdram1.Instance = FMC_SDRAM_DEVICE;
  /* hsdram1.Init */
  hsdram1.Init.SDBank = FMC_SDRAM_BANK1;
  hsdram1.Init.ColumnBitsNumber = FMC_SDRAM_COLUMN_BITS_NUM_8;
  hsdram1.Init.RowBitsNumber = FMC_SDRAM_ROW_BITS_NUM_11;
  hsdram1.Init.MemoryDataWidth = FMC_SDRAM_MEM_BUS_WIDTH_16;
  hsdram1.Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
  hsdram1.Init.CASLatency = FMC_SDRAM_CAS_LATENCY_1;
  hsdram1.Init.WriteProtection = FMC_SDRAM_WRITE_PROTECTION_DISABLE;
  hsdram1.Init.SDClockPeriod = FMC_SDRAM_CLOCK_DISABLE;
  hsdram1.Init.ReadBurst = FMC_SDRAM_RBURST_DISABLE;
  hsdram1.Init.ReadPipeDelay = FMC_SDRAM_RPIPE_DELAY_0;
  /* SdramTiming */
  SdramTiming.LoadToActiveDelay = 16;
  SdramTiming.ExitSelfRefreshDelay = 16;
  SdramTiming.SelfRefreshTime = 16;
  SdramTiming.RowCycleDelay = 16;
  SdramTiming.WriteRecoveryTime = 16;
  SdramTiming.RPDelay = 16;
  SdramTiming.RCDDelay = 16;

  if (HAL_SDRAM_Init(&hsdram1, &SdramTiming) != HAL_OK)
  {
    Error_Handler();
  }

}

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

```c

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_GPIOJ_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOI_CLK_ENABLE();
  __HAL_RCC_GPIOK_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(RLY_MPPT_GPIO_Port, RLY_MPPT_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOG, RLY_BATT_Pin|RLY_AC_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : RLY_MPPT_Pin */
  GPIO_InitStruct.Pin = RLY_MPPT_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(RLY_MPPT_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : STS_DAY_Pin USER_BUTTON_Pin */
  GPIO_InitStruct.Pin = STS_DAY_Pin|USER_BUTTON_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

  /*Configure GPIO pin : USER_LED_Pin */
  GPIO_InitStruct.Pin = USER_LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(USER_LED_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : RLY_BATT_Pin RLY_AC_Pin */
  GPIO_InitStruct.Pin = RLY_BATT_Pin|RLY_AC_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
/*****************************************************
   SysTick_Handler callback function
   This handler is called every tick and schedules tasks
 *****************************************************/
void HAL_SYSTICK_Callback(void)
{
  /* For TIME OUT processing */
  HAL_IncTick();

  if (remainAutoReloadTimerLoopVal_S) {
    remainAutoReloadTimerLoopVal_S--;
  }
}

/* USER CODE END 4 */

/**
```

```c
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```