POLITECNICO DI TORINO

Department of Electronics and Telecommunication

Master of Science in Electronics Engineering

Master Thesis

Low Cost Solar Simulator

Supervisor

Prof. Leonardo Reyneri

Candidate

Sanwal Saleem

July 2015

Abstract

This project describes the development of a 1C6412 1U Light source which is a part of big project (Low Cost Solar Simulator). The main purpose of this project is to develop low cost light source which provides a controllable indoor test facility under laboratory conditions and used in electrical engineering labs to measure photoelectric effect and to characterize, compare solar cells and to design and test solar energy convertors.

This project involves merger of different technology fields like High Power Electronics, Embedded systems, Mechanical Engineering and Software Engineering. The main challenge was to create similar commercial light source but with low cost and in very compact space (100 mm x 100mm x 300mm).

First step was to design 1C601 Power Driver board capable of driving four halogen lamps (each lamp power consumption 50 watt) on a PCB of 100mm x 100 mm. The power side and Logic side are galvanically isolated including the current sense feedback. EMI filter for each channel driver is added to make the board compliant with EMC European regulations. There are two other crucial parts of the system are 1C601 Control board and 1C601 Filter board. Filter board is used to filter main input power line from the high frequency noise either generated by natural or artificial source and Control board bear all the computation power. 1C6412 Light source is equipped with 8 bit I2C (Address programmable by DIP switch) and USB 2.0 for debugging.

Last step was to develop complete command set and basic communication protocol. Command which should be able to get status of system variables and to configure the system.

This thesis is considered to be a blue print of 1C6412 Light source and contribution to the Low Cost Nano Satellites test benches. Future work regarding 1C6412 is to add code of the PID control algorithm to control the internal temperature of the 1C6412 light source, main system loop and PCB fabrications.

This Project was developed with the vision to introduce low cost space technology equipments in the high schools and colleges of Italy so that young generation should be able to think and participate in the field of space technology

ACKNOWLEDGMENTS

I would like to thank Prof. Leonardo Reyneri for his Endless support and guidance during whole project. He has been not only tremendous teacher but a true mentor for me. I would like to thank Mr.Waheed Shah from NFC Institute of Engineering and technology Multan, Pakistan for his massive support and allowing me to get hands on different Electronic instruments during my bachelors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I: Introduction

This project describes low cost 1C6412 1U Light Source. The 1C6412 1U Light Source is the key element of a low-cost solar simulator. In practice, it is a configurable light source made of four commercial MR16 halogen lamps which can generate the same solar power density which is found in Low Earth Orbits at AM0, that is, outside Earth's atmosphere.

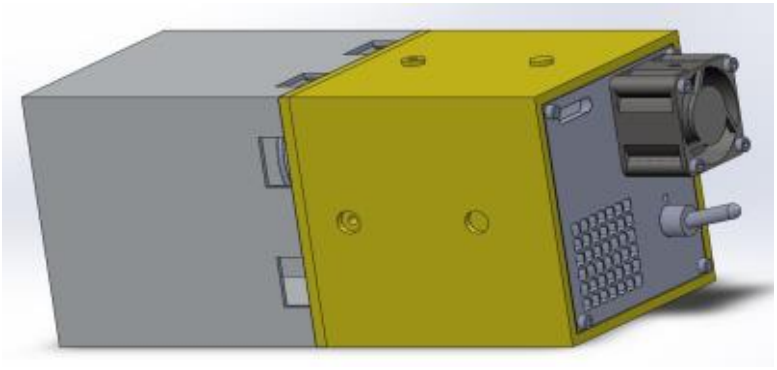


Figure 2.1: 1U Light source side view

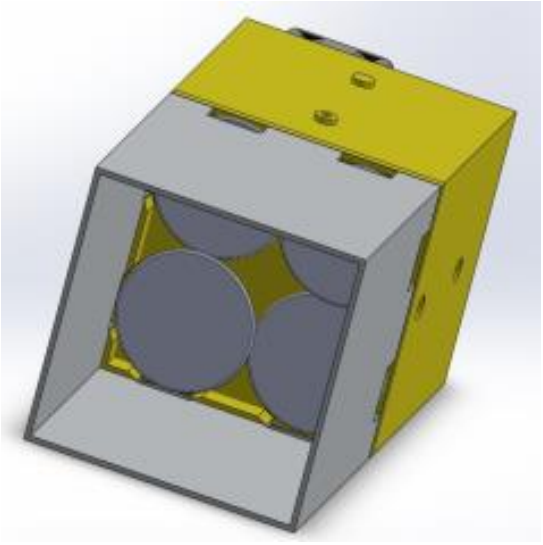


Figure 3.2: 1U Light source Front view

Since the spectrum of halogen lamps is different from the solar light spectrum, a proprietary technique compensates spectral mismatches and allows the 1C6412 1U Light Source to generate an appropriate power density to produce either:

1. the same thermal effects of the Sun spectrum on a black body (see section Illuminate Sat Thermally Black)

2. the same thermal effects of the Sun spectrum on a colored body (e.g. green or blue CubeSat) (see section Illuminate Sat Thermally Blue or Illuminate Sat Thermally Green)

3. the same electrical effects of the Sun spectrum on either single- or triple-junction GaAs solar cells (see section Illuminate Sat Electrically Single Junction Illuminate Sat Electrically Triple Junction)

4. the same electrical effects of the Sun spectrum on single-junction Si solar cells (see section Illuminate Sat Electrically Single Junction)

5. some user-defined power density (see section Illuminate Sat Variable)

The 1C6412 1U Light Source can also be controlled to generate programmable day/night illumination periods (see section Set Light Period).

The 1C6412 1U Light Source is intended to illuminate one side of a 1U CubeSat but two or more of them can be stacked to illuminate one side of a 2U CubeSat, 3U CubeSat, 6U CubeSat or even 12U CubeSat. More generally, an appropriate number of 1C6412 1U Light Source can be stacked horizontally and vertically to illuminate any surface at AM0,

provided an appropriate ventilation is provided to remove the heat produced by large arrays.

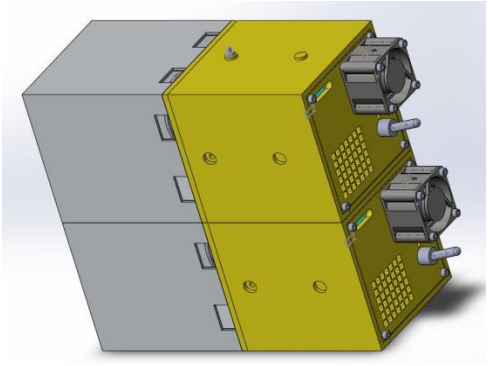


Figure 4.3: 1C6412 2U Light source

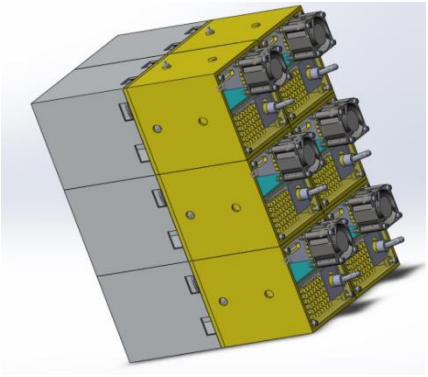


Figure 5.4: 1C6412 6U Light source

The 1C6412 1U Light Source is stand-alone equipment which only requires a 12V, 200W power supply, plus an I2C interface to configure it and to appropriately control it. The 1C6412 1U Light Source is factory calibrated at a predefined distance from the CubeSat surface. The user shall keep its CubeSat under test at appropriate distance. This distance can also be achieved by means of an optional 1C653 Stand to Light Separator plus either:

- one 1C651 1U Fixed Stand, for either 1U CubeSat or 2U CubeSat or 3U CubeSat
- one 1C651 12U Fixed Stand, for either 6U CubeSat or 12U CubeSat

Which keep the CubeSat under test at the correct distance, either perpendicularly or with any other angle of incidence multiple of 15°.

For better performance, the 1C651 1U Fixed Stand or 1C651 12U Fixed Stand can be substituted by either 1C652 1U Turn Table or 1C652 12U Turn Table, respectively, to simulate the effects of spacecraft spin.

1C6412 1U Light Source consists of three main electronic hardware boards. Following is the list of electronics boards developed for 1C6412 1U Light Source.

- 1C601 Power Driver Board
- 1C601 Filter Board
- 1C601 Control Board

1.1 1C601 Power Driver Board:

1C601 Power driver intended to convert TTL level pwm into high voltage PWM output. Power driver have 5 output channels. Each channel is galvanically isolated and takes TTL level (3.3v to 5.5v) PWM as input and also disable signal. Each channel provides current sense feedback which is also isolated by using Hall Effect technique. Each channel is capable of driving load with 9 Amps and 12v except the 5^{th} channel which is dedicated for the cooling fan to control the internal temperature. 5^{th} channel have capability of 12v and 1 Amp. The 1C601 Power Driver board should be compliant with European EMI and EMC regulations, Because of that EMI Filters are added at the output of each channel.

1.2 1C601 Filter Board:

1C601 Filter Board intended to clean up the power line from high frequency noise which is either created naturally or artificially. 1C601 Filter Board filter and divide the single Incoming power line (12v @ 20Amps) into three main power supply lines. PWR_01 is first power line output and is connected in series with over temperature cut off switch (TCS) and feeded into power driver output channel 1 and 2. Second power line (PWR_23) has also over temperature cut off switch but its feeded to the power driver output channel 2 and 3. Third line (PWR_4) is connected directly to power channel 5 for

the cooling fan, because in case of temperature control failure the fan power line should not be cut off by TCS (Temperature Cut off Switch) and cooling should be in functional.

1.2 1C601 Control Board:

1C601 Control Board provides communication interface, Computation power and connectivity with the 1C601 Power Driver Board. 1C601 Control board provides two communication interfaces with external world one is I2C and another is USB. I2C communication is main communication interface by which user can send commands to control and get values of different process variables of the system. I2C communication is used to make this project expandable, so that 6412 1U Light source can be stacked together horizontally or vertically as explained above. USB communication is used for debugging purpose only.

CHAPTER II: Specifications

First milestone of this project was to write all the specifications. Specifications are the most important part of the project as it describes the whole requirements of the project and really helpful to keep track the development of the project and cross check it with the specifications. For writing specification I used Universal modeling language (UML). UML provides very unique way to write specifications in the form of use case diagram. Specifications are described in top to down hierarchy.

Figure 2.1 shows the top level specification diagram:

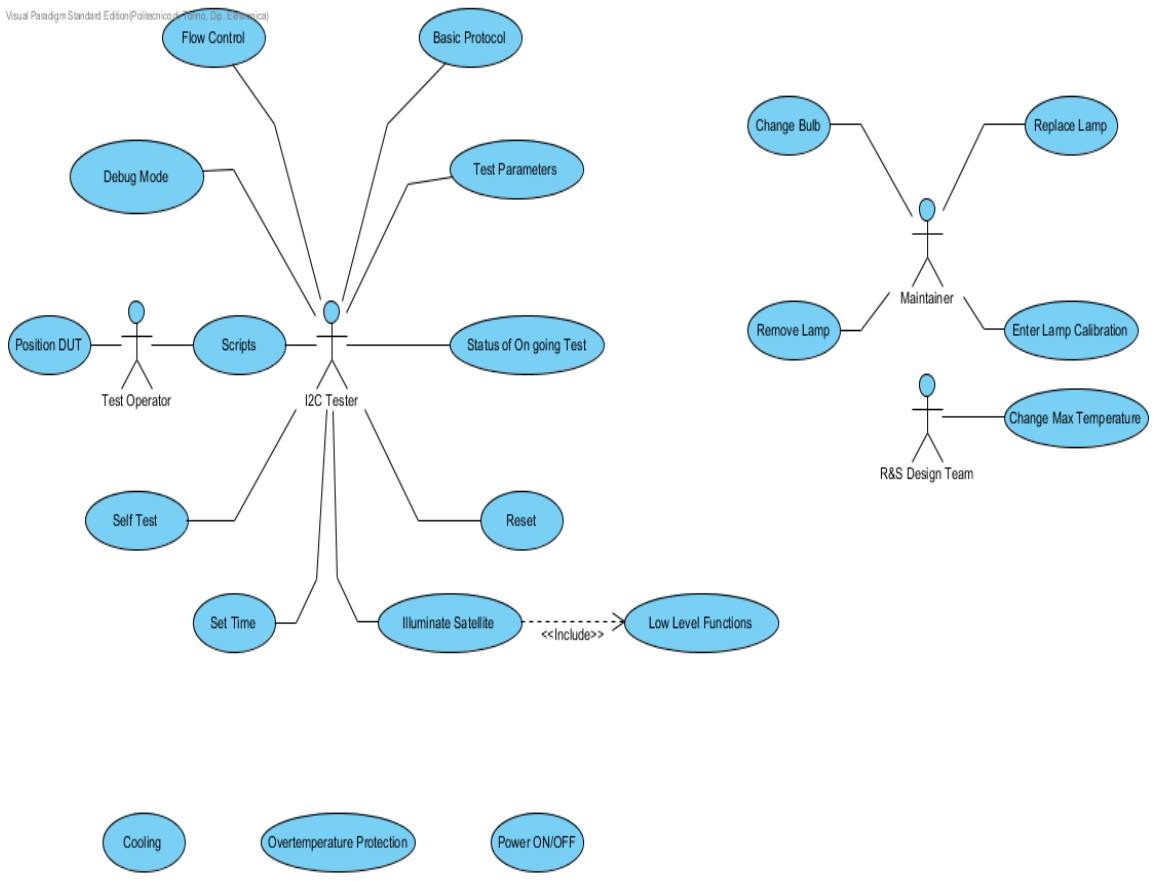


Figure 2.1: 1C6412 2U Light source Top Level Specification Diagram

2.1 I2C Tester:

The I2C Tester interacts (issues commands, reads status, etc.) with the 1C6412 1U Light Source by means of Basic Protocol via an I2C and Logic Supply Connector located on the back of the 1C6412 1U Light Source.

By means of the Basic Protocol, the I2C Tester can instruct the 1C6412 1U Light Source to perform a set of actions onto the Device under Illumination, as described later.

The I2C Tester can either be a human which uses its own I2C User Interface, or a 1C604 Main Controller capable of performing a number of complex and highly structured test onto the Device Under Illumination.

2.2 Test Operator:

Test Operator can choose type of test and also capable to position the Device under test according to its need.

2.3 Basic Protocol:

1C6412 1U Light Source used I2C for communication but communication is done in a specific format describe in chapter VII Basic Communication protocol.

2.4 Set Time:

The I2C Tester sends a command SET_TIME to the 1C6412 1U Light Source by using Write Data to set the Time in following format.

1. First Two bytes of Hours
2. Next Two Bytes of Minutes
3. Next Two Bytes of Seconds

Total 6 Bytes are required to set Time of 1C6412 1U Light Source.

2.5 Position DUT:

Position DUT is the position of device under test in front of 1C6412 1U Light Source and choose by the Test Operator according to the requirements of the Experiment or Test Operator choice.

2.6 Illuminate Satellite:

This is a group of similar commands aiming at properly illuminating the Device Under Illumination to perform the desired thermal or electric tests. When the I2C Tester sends one of the commands detailed further to the 1C6412 1U Light Source, this immediately begins to illuminate the Device under Illumination at a given intensity, which depends on the specific command issued. Illumination can either be constant or periodic.

The detailed test timing will be according to what specified in the Set Light Period section. Lamp will be turned on and off slowly to reduce bulb wear. The lamp will slowly reach correct power level within Lamp Soft Start Time.

All the commands described below require that the Device under Illumination is located at the Light Source to Satellite Distance. This position can easily be achieved by means of a 1C653 Stand to Light Separator. Following is the list of Illumination Mode:

- Illuminate Sat Variable
- Illuminate Sat Thermally Black
- Illuminate Sat Thermally Green
- Illuminate Sat Thermally Blue

- Illuminate Sat Electrically Single Junction
- Illuminate Sat Electrically Triple Junction

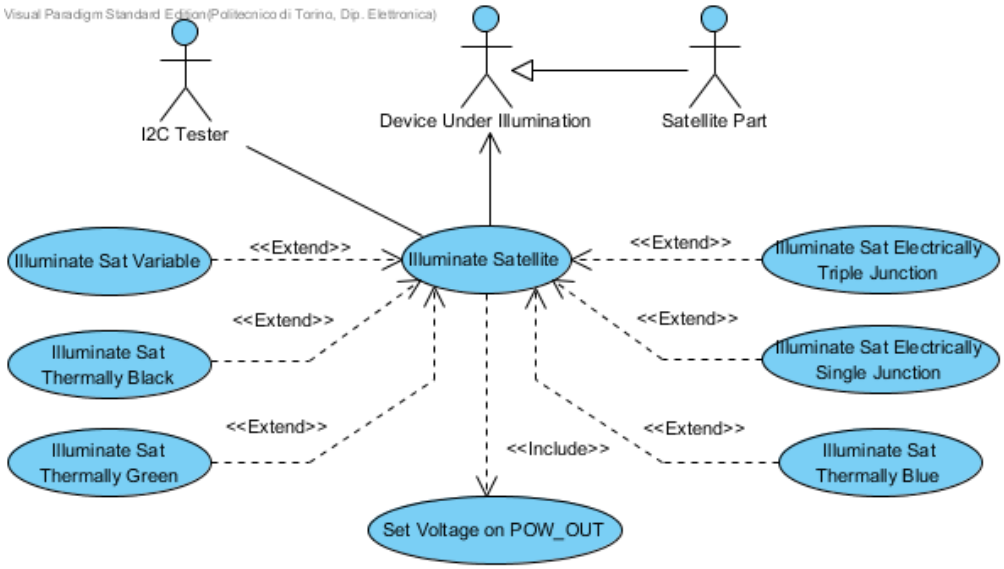


Figure 2.2: Illuminate Sat

2.6.1 Illuminate Sat Variable:

When the I2C Tester sends command ILLUMINATE_VARIABLE by using Write Data. The command ILLUMINATE_VARIABLE is such as to illuminate the Device under Illumination at an intensity defined by the I2C Tester by means of a command parameter in units of 1W/m2 thermal on black body, up to Max Thermal Intensity.

2.6.2 Illuminate Sat Thermally Black:

When the I2C Tester sends command ILLUMINATE_THERMAL_BLACK , by using Command Only , the 1C6412 1U Light Source illuminates the Device Under Illumination with such an intensity as to deposit 1366W/m2 (namely, the Solar Constant) on a black body. This command is intended for the rmal tests on dark satellites.

2.6.2 Illuminate Sat Thermally Black:

When the I2C Tester sends command ILLUMINATE_THERMAL_BLACK , by using Command Only , the 1C6412 1U Light Source illuminates the Device Under Illumination

2.6.3 Illuminate Sat Thermally Green:

When the I2C Tester sends command ILLUMINATE_THERMAL_GREEN, by using Command only, the 1C6412 1U Light Source illuminates the Device Under Illumination with such an intensity as to have the same thermal effects that would have a power density of 1366W/m2 (namely, the Solar Constant) with the proper solar spectrum on a PCB-green body. This command is intended for thermal tests on satellites with lateral walls made mostly of green PCBs, without solar panels.

2.6.4 Illuminate Sat Thermally Blue:

When the I2C Tester sends command ILLUMINATE_THERMAL_GREEN by using Command only, the 1C6412 1U Light Source illuminates to illuminate the Device under Illumination at 1366W/m2 (Solar Constant) total power on a solar-cell-blue body.

2.6.5 Illuminate Sat Electrically Single Junction:

When the I2C Tester sends command Illuminate Sat Electrically Single Junction , by using Command Only .The command Illuminate Sat Electrically Single Junction is such as to illuminate the Device Under Illumination such that the power generated by a single junction solar cell is equivalent to that generated by the same cell illuminated by sun spectrum at 1366W/m2 (Solar Constant) total power density.

2.6.6 Illuminate Sat Electrically Triple Junction:

When the I2C Tester sends command ILLUMINATE_TRIPLE_JUNCTION , by using Command Only .The command ILLUMINATE_TRIPLE_JUNCTION is such as to illuminate the Device Under Illumination such that the power generated by a triple junction solar cell is equivalent to that generated by the same cell illuminated by sun spectrum at 1366W/m2 (Solar Constant) total power density.

2.6.7 Set Voltage on POW_OUT:

Set Voltage on POW_OUT represents low level function Set_POW_OUT_Voltage (channel: int, val: unsigned short): void to set voltage on specific channel. The input arguments are channel number and value of voltage to be applied on that channel.

2.7 Flow Control:

Flow Control contains the variables of the 1C6412 1U Light Source which controls the flow of the Experiment. Flow Control contains following variables:

- Start Test
- Pause
- Resume
- Stop Test

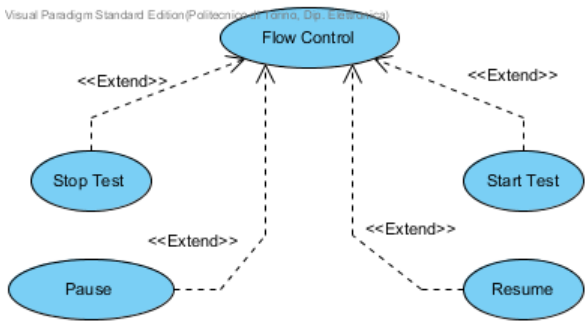


Figure 2.3: Flow Control

2.7.1 Start Test:

The I2C Tester sends a command START_TEST to the 1C6411_Simple_Sun_Simulator by using Command only to immediately start illuminating the Device under Illumination.

2.7.2 Pause:

The I2C Tester sends a command Pause to the 1C6411_Simple_Sun_Simulator by using Command only to pause the ongoing Experiment in 1C6411_Simple_Sun_Simulator.

2.7.3 Resume:

The I2C Tester sends a command Resume to the 1C6411_Simple_Sun_Simulator by using Command only to resume the paused Experiment in 1C6411_Simple_Sun_Simulator.

2.8 Stop Test:

The I2C Tester sends a command STOP_TEST to the 1C6411_Simple_Sun_Simulator by using Command only to immediately stop illuminating the Device under Illumination.

2.9 Self Test:

The Generic Tester send a command to the 1C6412 1U Light Source to start the Self Test .The output result will be store in one of the element of buffer. In Self Test Following things will be tested:

- Lamp Presence
- Lamp Remaining Life
- Power Drivers status
- Internal Temperatures
- Main high power supply presence

2.9.1 Lamp presence:

This bit represents either all lamp are presence or not.

2.9.2 Lamp Remaining Life:

This bit represents that either the lamp life increases the specified threshold or not.

1=Above the specific Threshold

0=Below the Specific Threshold (Good to Go)

2.9.3 Power Driver Status:

This bit represents either the Power drivers are working fine or not

2.9.4 Internal Temperatures:

This bit represents either the internal temperatures of 1C6412 1U Light Source is below safe level or not.

1=Above safe level (Cooling required)

0=Below safe level (Good to GO)

2.9.5 Main Power Supply Presence:

This bit represents either the Main Power Supplies of 1C6412 1U Light Source are present or not.

1=Power Present

0=No Power Presence

Following is the list of use cases for Self Test:

- Start Self Test
- Get Self Test Status

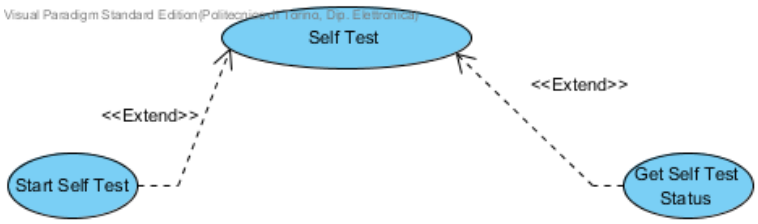


Figure 2.4: Self Test

2.9.6 Start Self Test:

This bit represents either Tester sends a command START_SELF_TEST to the 1C6411_Simple_Sun_Simulator by using Command only to start the Self Test.

2.9.7 Get Self Test Status:

The Generic Tester sends a command GET_SELF_TEST_STATUS to the 1C6411_Simple_Sun_Simulator by using Read Data to get the status of Self Test.

There are two conditions

- If return Value is 1 its means Self Test is completed and Master can Read the Outcome.

- If return Value is 0 its means Self Test is still running and Master cannot Read the Outcome.

2.10 Test Parameters:

Test Parameters are the list of parameters which are required by the test to be performed. Test Parameters contains following parameters:

- Set Light Period
- Set Experiment Period

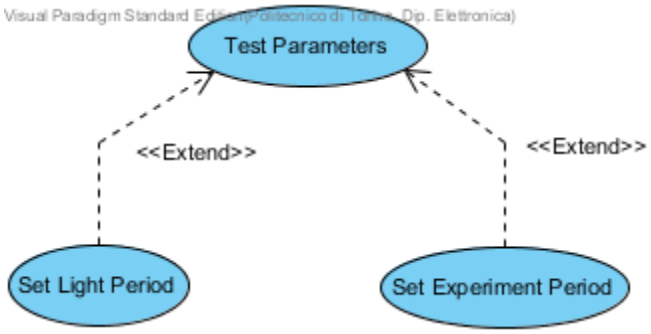


Figure 2.5: Test Parameters

2.10.1 Set Light Period:

The I2C Tester sends a command SET_LIGHT_PERIOD to the 1C6412 1U Light Source by using Write Data to set the following parameters for the next test:

1. Illumination period, in seconds (1^{st} four Bytes)
2. Illumination duty cycle, in % (3^{rd} Byte)
3. The time after Illuminate Satellite when the lamp first turns on (4^{th}, 5^{th} , 6^{th} and 7^{th} Byte)
4. Overall test duration, in seconds (8^{th} , 9^{th}, 10^{th} and 11^{th} Byte)

2.10.2 Set Experiment Time:

The I2C Tester sends a command SET_EXPERIMENT_PERIOD to the 1C6412 1U Light Source by using Write Data to set the total Experiment Period.

2.11 Status of ongoing test:

Status of ongoing Test contains list of commands use to get the status of ongoing test. These variables are used for monitoring different elements of 1C6412 1U Light Source. It Contains following Status Variables:

- Get Lamp Life
- Get Power Consumption
- Get Remaining Time of Experiment
- Get Status
- Get SW and HW Revision and serial number
- Get_orbits _elapsed
- Get Sun Exposure Time

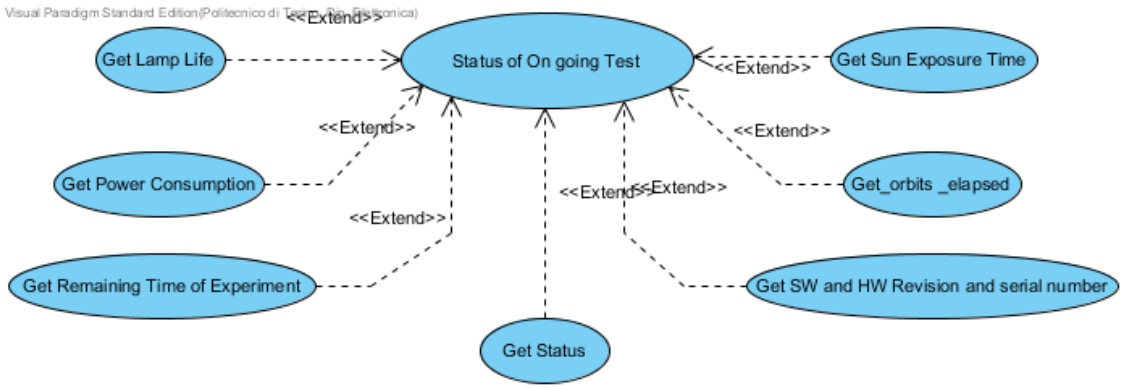


Figure 2.6: Status of ongoing test

2.11.1 Get Lamp Life:

The I2C Tester sends a command GET_LAMPS_LIFE to the 1C6411_Simple_Sun_Simulator by using Read Data to get the remaining life of all four lamps which are mounted in the 1C6412 1U Light Source.

2.11.2 Get Power Consumption:

The I2C Tester sends a command GET_POWER_CONSUMPTION to the 1C6411_Simple_Sun_Simulator by using Read Data to get the current Power consumption of 1C6412 1U Light Source.

2.11.3 Get Remaining Time of Experiment:

The I2C Tester sends a command GET_ELAPSED_EXPERIMENT_TIME to the 1C6411_Simple_Sun_Simulator by using Read Data to get the Elapsed time Experiment running on 1C6412 1U Light Source.

2.11.4 Get Status:

The I2C Tester sends a command GET_STATUS to the 1C6411_Simple_Sun_Simulator by using Read Data to get the 16 bit Status Register from 1C6412 1U Light Source.

Each bit of 16bit Status Register represents different Outcomes.

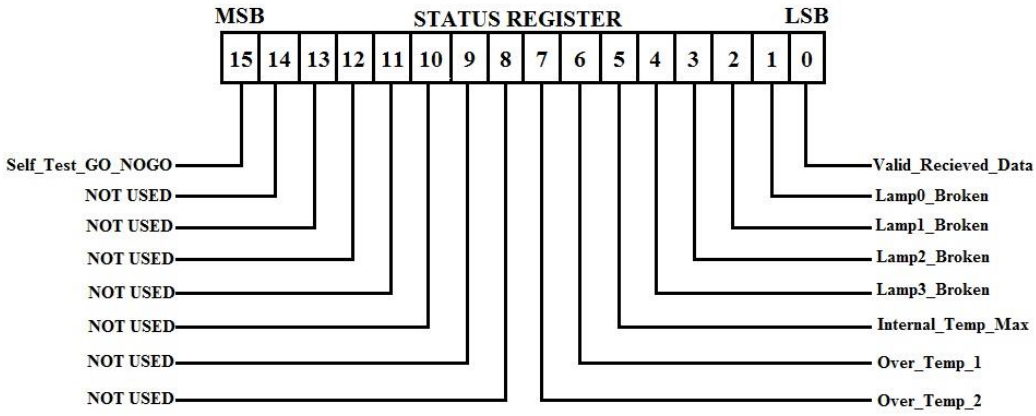


Figure 2.7: Status Register

2.11.5 Get SW and HW revision and Serial Number:

The I2C Tester sends a command GET_HW_SW_SERIAL_NUMBER to the 1C6411_Simple_Sun_Simulator by using Read Data to get the Hardware and Software Serial Number of 1C6412 1U Light Source.

First Two Bytes are of HW Serial Number and Next Two Bytes for SW Serial Number

2.11.6 Get elapsed orbits:

The I2C Tester sends a command GET_ELAPSED_ORBITS to the 1C6411_Simple_Sun_Simulator by using Read Data to get the number of elapsed orbits during the Experiment.

2.11.7 Get sun exposure time:

The I2C Tester sends a command GET_SUN_EXPOSURE_TIME to the 1C6411_Simple_Sun_Simulator by using Read Data to get the Sun Exposure time during Experiment in Minutes.

2.12 Scripts:

Scripts contain different types of scripts to execute different Experiment on one time execution of script. Each Script contains Sequence of commands and read/Write Data to perform specific task.

NOTE: NOT IN THIS VERSION

2.13 Reset:

The Generic Tester send a command RESET to the 1C6411_Simple_Sun_Simulator by using Command Only to clear flags of the 1C6411_Simple_Sun_Simulator (For example Data Error Flag etc).

2.14 Debug Mode:

The I2C Tester can turn ON and OFF debug mode by sending following commands to the 1C6411_Simple_Sun_Simulator:

1. Debug Mode OFF
2. Debug Mode ON

In debug mode the 1C6411_Simple_Sun_Simulator can be controlled and supervised by using UART debug Port.

NOTE: NOT INCLUDED IN THIS VERSION

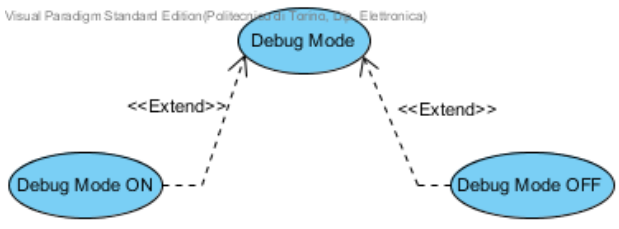


Figure 2.8: Debug Mode

2.14.1 Debug Mode ON:

When the I2C Tester sends command DEBUG_ON, by using Command only and the Debug mode will turn ON in 1C6412 1U Light Source.

2.14.2 Debug Mode OFF:

When the I2C Tester sends command DEBUG_OFF, by using Command only and the Debug mode will turn OFF in 1C6412 1U Light Source.

2.15 Cooling:

The 1C6412 1U Light Source must be cooled, to keep internal temperature as constant and uniform as possible and to avoid overheating. See also over temperature Protection.

Cooling must be based on air flow and air in- and out-let shall be on the opposite side w.r.t to light output, in order to stack as many 1C6412 1U Light Sources as desired both in horizontal and vertical directions.

2.16 Enter Lamp Calibration:

The Maintainer sends a command ENTER_LAMP_CALIBRATION to the 1C6412 1U Light Source by using Write Data to send the Lamp Calibration Data.

Lamp Calibration data consists of ten Bytes.

- Data[0]=Lamp1_Life
- Data[1]=Lamp2_Life
- Data[2]=Lamp3_Life
- Data[3]=Lamp4_Life
- Data[4]=K_Black

- Data[5]=K_Blue
- Data[6]=K_Green
- Data[7]=K_Triple_Junction
- Data[8]=K_Single_Junction
- Data[9]=Lamp1_Intensity_Relation_with_Power_Co_Efficient
- Data[10]=Lamp2_Intensity_Relation_with_Power_Co_Efficient
- Data[11]=Lamp3_Intensity_Relation_with_Power_Co_Efficient
- Data[12]=Lamp4_Intensity_Relation_with_Power_Co_Efficient

2.17 Remove Lamp:

The Maintainer Turn Off the System and wait until the lamp is cold enough. The Maintainer will open_protection (), then he/she will unplug the old lamp.

2.18 Replace Lamp:

The Maintainer plugs the new lamp, then he/she will close_protection ().

2.19 Change Lamp:

The Maintainer Turn Off the System the system, Remove Lamp (either broken or old). The Maintainer Replace Lamp with a 1C6412 pre-calibrated lamp. The Maintainer Turn On the system. The Maintainer enters its 1C6412 pre-calibrated lamp. Calibration code: t_Bulb_Calibration into the Configuration Interface using the Enter Lamp Calibration.

2.20 Over Temperature Protection:

When the temperature inside enclosure, after light bulbs, along fan air flow, overpasses:

- A first user-defined threshold, bulbs will be turned off, via SW
- A second fixed threshold (Max 1U Light Source Internal Temperature), power supply will be cut-off for the power outputs (POW_OUT_0 (), POW_OUT_1 (), POW_OUT_2 (), POW_OUT_3 ()) until temperature returns under the threshold.

2.21 Low Level Functions:

Low Level Functions contains different low level functions of 1C601 Control_Board to achieve different low level task.

Such as:

- Set Current on POW_OUT
- Set Voltage on POW_OUT
- Acquire AIN voltage
- Acquire internal temperature
- Turn off POW_OUT

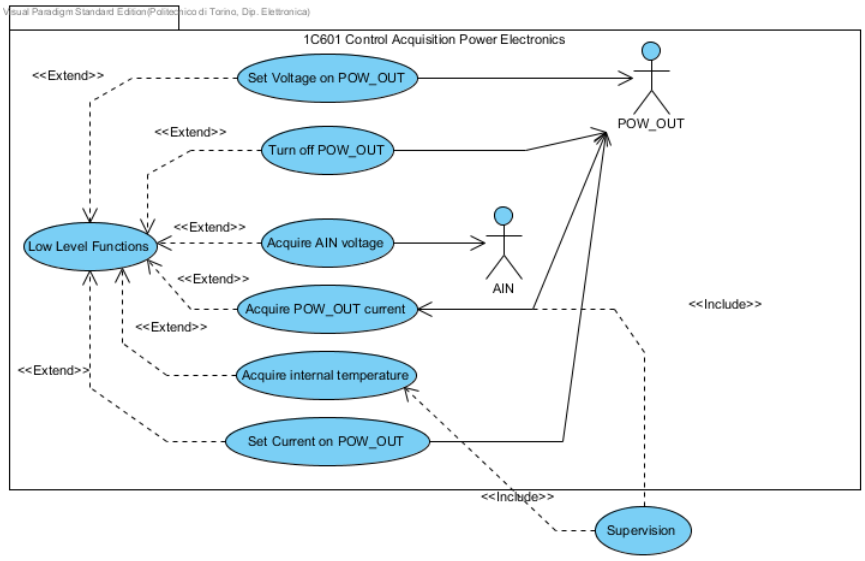


Figure 2.9: Low Level Functions

2.21.1 Set Current on POW_OUT:

Set Current on POW_OUT represents a low level function to Set_POW_OUT_Current(channel: int, val: unsigned short): void to set current on mentioned output channel. There are two input arguments one is channel number and another is unsigned short current value in mAmps which need to be set on particular channel.

2.21.2 Acquire internal temperature:

Acquire internal temperature represents low level function Get_internal_temp(): unsigned short to get internal temperature of the 1C6412 1U Light Source. There are no input arguments of this function. On calling this function it will return unsigned short digital value of the measured temperature.

2.21.3 Acquire POW_OUT current:

Acquire POW_OUT current represents low level function Get_POW_OUT_Current(channel: int): unsigned short to get the current consumption on mentioned channel. The input argument is the channel number on which current consumption to be measured and the return value is digital value of measured current consumption.

2.21.4 Acquire AIN voltage:

Acquire AIN voltage represents low level function Get_AIN_Voltage(channel: int): unsigned short to get the analog voltage on mentioned adc channel. The Input argument of this function is adc channel number and this function returns the digital value of measured voltage.

2.21.5 Turn off POW_OUT:

Turn off POW_OUT represents low level function Turn_Off_POW_OUT(channel: int): void to turn off the output voltage on particular power output channel. The input argument is the channel number which need to be turn OFF.

2.21.6 Set Voltage on POW_OUT:

Set Voltage on POW_OUT represents low level function Set_POW_OUT_Voltage (channel: int, Val: unsigned short): void to set voltage on specific channel. The input arguments are channel number and value of voltage to be applied on that channel.

2.21.7 Supervision:

The 1C6411_Simple_Sun_Simulator will autonomously verify. Over temperature of the 1C6412 pre-calibrated lamp; when in over temperature, all 1C6412 pre-calibrated lamps are turned off. Temperature will be measured every second approximately.

CHAPTER III: System Architecture

Low Cost simulator consists of four modules. For defining the system architecture of such a big and complex system UML is the best choice. System architecture of the low cost solar simulator was defined in top to down hierarchy in uml. In this thesis the task was to design 1C6412 1U light source but before designing it, it was important to lay down the rough sketch of low cost simulator so that 1C6412 1U light source should be compatible with the low cost solar simulator.

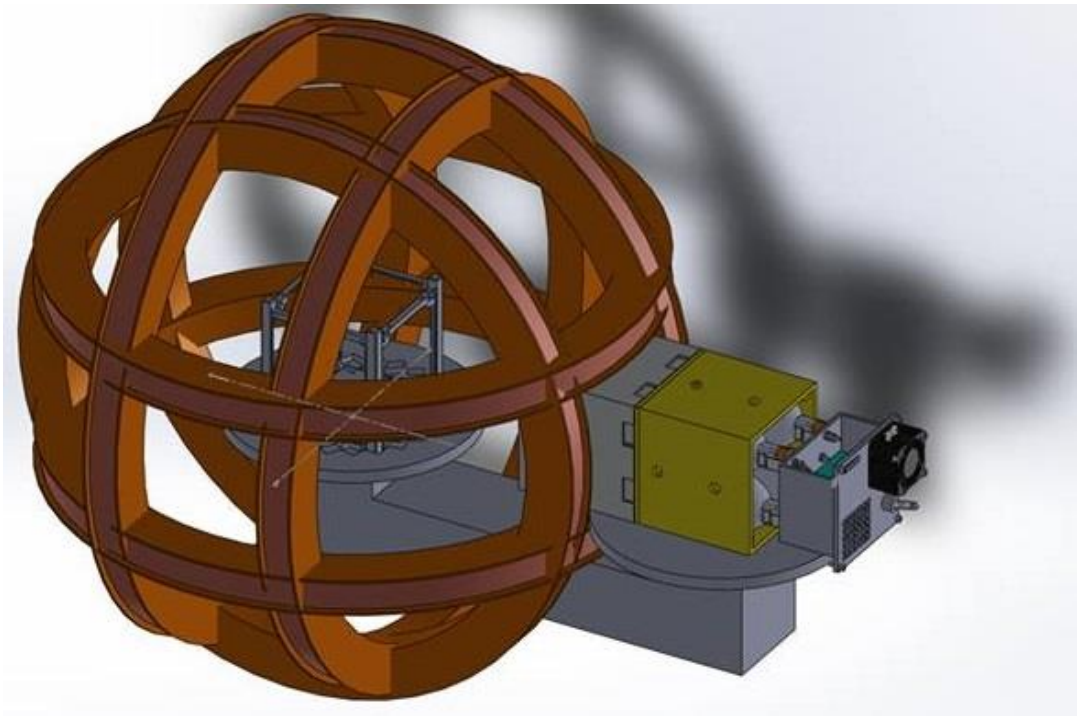


Figure 3.1: 3D Model of Low Cost Solar Simulator

Following is the list of modules required to build up solar simulator:

- 1C6412 1U Light Source
- 1C604 Main Controller
- 1C651 1U Fixed Stand
- 1C653 Stand to Light Separator

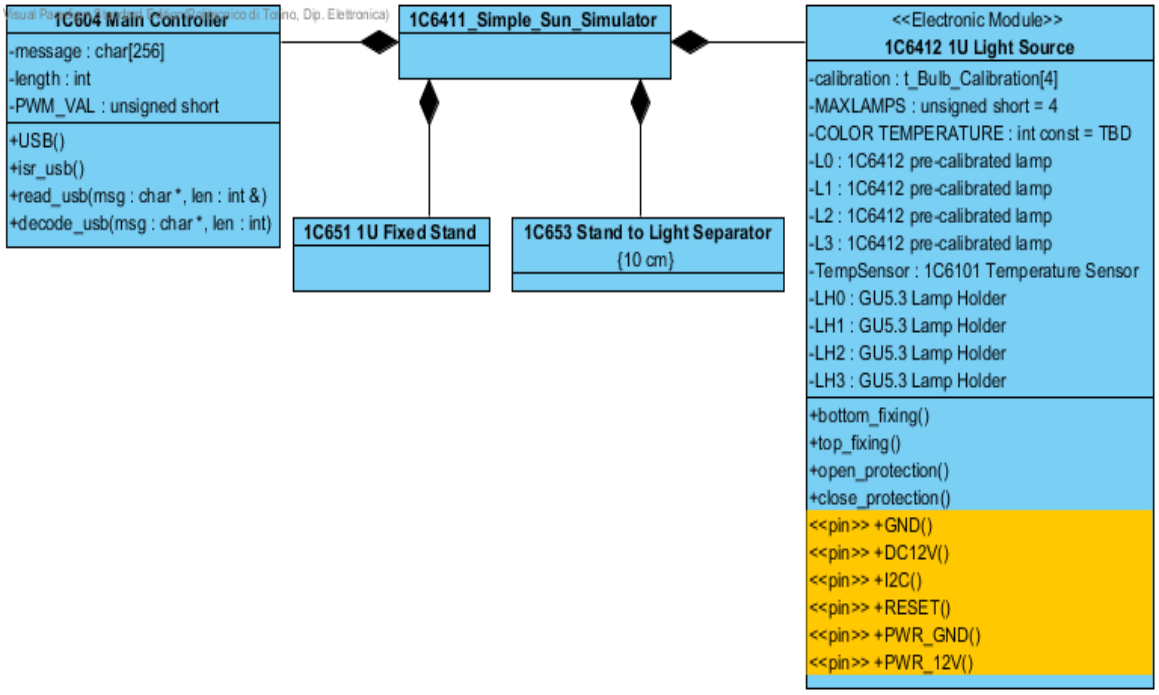


Figure 3.2: Top Level Architecture of Low Cost Solar Simulator

3.1 1C6412 1U Light Source:

The 1C6412 1U Light Source is a calibrated light source capable of illuminating one or two sides of a 1U CubeSat, at different incidence angles, with nearly uniform distribution, and it is aimed primarily at electrical and thermal testing of CubeSats.

The 1C6412 1U Light Source spectrum is not identical to and its color temperature is lower than that of the sun, yet the 1C6412 1U Light Source is factory calibrated to generate different light intensities, depending on commands, such as to produce either:

- The same thermal effects on black and colored surfaces;
- The same electrical effects on different models of solar cells.

Two or more 1C6412 1U Light Source can be stacked and packed together to illuminate one side of either 2U CubeSats or 3U CubeSats or 6U CubeSats or 12U CubeSats.

In case one bulb wears out or breaks, it can be substituted by one 1C6412 pre-calibrated lamp, preserving the proper level of calibration for all foreseen functions.

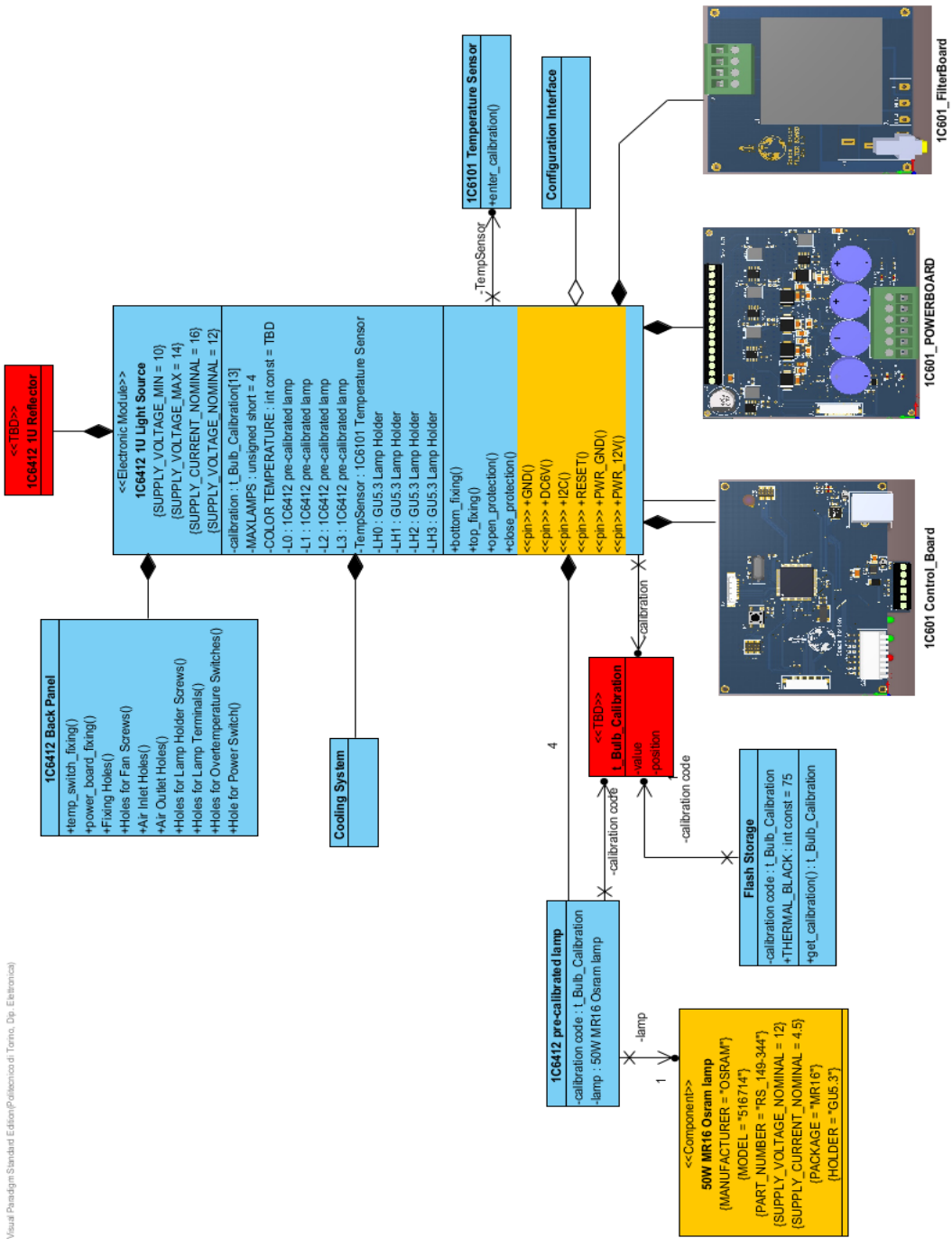


Figure 3.3: 1C6412 1U Light Source Architecture

Table 3.1Shows the List of Attributes of 1C6412 1U Light Source:

Table 3.1: Attributes of 1C6412 1U Light Source

Attributes Name	Description
Calibration	Holds 13 Byte Data for Lamp Calibration. See section 2.61
Max Lamps	Holds the value of total number of lamps which is 4.
L0	The top right bulb when seen from the Protection Glass Nut & Bolt.
L1	The top left bulb when seen from the Protection Glass Nut & Bolt.
L2	The bottom left bulb when seen from the Protection Glass Nut & Bolt.
L3	The bottom right bulb when seen from the Protection Glass Nut & Bolt.
Temp Sensor	TempSensor: 1C6101 Temperature Sensor used to measure the temperature of air inside 1C6412 1U Light Source.
LH0	The top right lamp holder, when seen from the Protection Glass Nut & Bolt. It holds lamp L0: 1C6412 pre-calibrated lamp.
LH1	The top left lamp holder, when seen from the Protection Glass Nut & Bolt. It holds lamp L1: 1C6412 pre-calibrated lamp.
LH2	The bottom left lamp holder, when seen from the Protection Glass Nut & Bolt. It holds lamp L2: 1C6412 pre-calibrated lamp.
LH3.	The bottom right lamp holder, when seen from the Protection Glass Nut & Bolt. It holds lamp L3: 1C6412 pre-calibrated lamp.

3.1.1 1C6101 Temperature Sensor:

1C6101 Temperature Sensor contains a TMP36 Temperature Sensor. Purpose of TMP36 is to measure the temperature of air intake. Measuring the intake air temperature is really important because there are high power mosfets and lamps which generates huge amount of heat and if the intake temperature is high (depends on the environment in which Light source is placed) than system should increase the speed of fan so that more air should be

intake. Which can be mounted inside 1C6412 1U Light Source or attach by adhesive tape or glue.

3.1.1.1 TMP36:

The TMP36 is low voltage, precision centigrade temperature sensor. It provides a voltage output that is linearly proportional to the Celsius (centigrade) temperature. The TMP36 do not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the −40°C to +125°C temperature range. It has shutdown pin to turn it OFF but in our application temperature plays crucial role and that's why temperature sensor never turns OFF.

TMP36 is directly connected to the Analog Input port of Control Board.

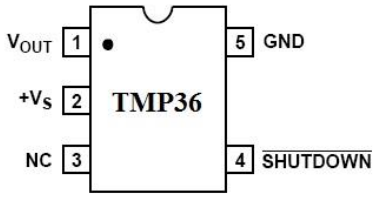


Figure 3.4: Pin Configuration of TMP36 Sensor

3.1.2 Cooling System:

The cooling system of 1C6412 1U Light Source is required to keep control over internal and components temperature, to prevent system burning and over temperature-induced effects and faults. Cooling system is the most crucial part of the system as mosfets and lamps generates large amount of heat which can cause thermal noise in electrical traces on the board and cause disturbance in the system. Another reason is overheating of the components which may become the reason of them to work abnormally or completely burn out.

Figure 3.6 shows the basic structure of cooling system.

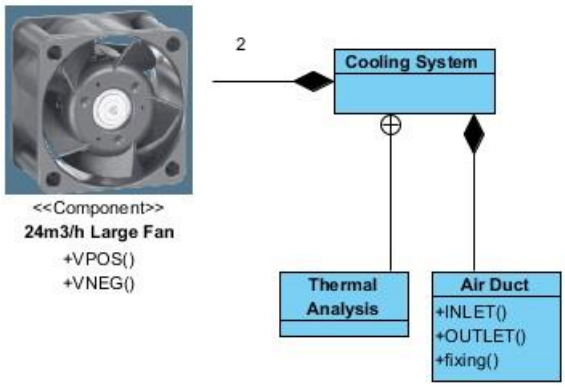


Figure 3.5: Cooling System

3.1.2.1 Thermal Analysis:

3.1.2.1.1 Air flow:

Considering a PVC Air Duct 50 mm diameter, total length 300mm with two 90 deg bends, with 24 m3/h air flow, we get a dynamic pressure of nearly 1mm H20.

With that pressure drop, from the datasheet of:

- 24m3/h Large Fan , we get an air flow in excess of 23 m3/h at 12VDC nominal voltage;
- There is no plot of flow vs. air pressure, but we can expect similar values as per 24m3/h Large Fan.

3.1.2.1.2 Heat flow:

Given:

- an air flow Q in excess of 23 m3/h (see above)
- air density ρ of 1.092 kg/m3
- air specific heat c (at nominal conditions) of *1.0 kJ/(kg K)*

The heat transfer capability of the chosen 24m3/h Large Fan is given by:

$$B * \rho * c = 7\ W/K$$

3.1.2.1.3 Temperature Increase:

Since in the worst case we have to remove a large fraction (say, 80%) of the total power of the four bulbs (0.8x4x50W = 160W), temperature increase from INLET() to OUTLET() will be

160W / (7W/K) = 23 K

By supposing INLET() air temperature of up to 50°C (considering 35°C room temperature, plus 15K increase due to the possible presence of other 1C6412 1U Light Sources around and the chance that INLET() will re circulate a fraction of OUTLET() air flow), the OUTLET() temperature might reach 75°C in the worst case.

As a consequence of this, we can place the following constraints to the design of 1C6412 1U Light Source:

- INLET and OUTLET of air flow shall be as independent as possible
- Fan shall be placed on INLET
- Control Electronics shall be placed in the coolest position
- Temperature sensor shall monitor air flow temperature
- Temperature sensor shall be off board
- PID algorithm to tightly control the Temperature

3.1.2.2 Cooling Fan:

Cooling fan is the important component of cooling system. See the section 3.1.3.1

Following is the picture of cooling fan.

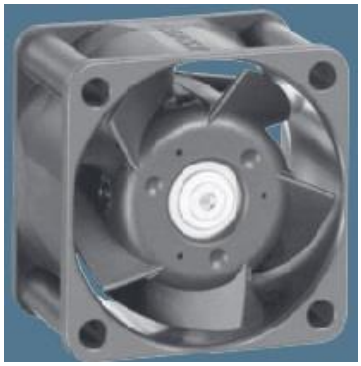


Figure 3.6: Cooling Fan

Following are the features of cooling fan:

- Very rigid compression curve for high air flow at high back pressure.
- Low operating noise level at high back pressure.
- General characteristics:
- Material: fiberglass-reinforced plastic. Impeller PA, housing PBT.
- Fully integrated electronic commutation.
- Protected against reverse polarity and locking.
- Connection via single strands AWG 26, TR 64. Bared and tin-plated.
- Air exhaust over struts. Direction of rotation counter-clockwise, seen on rotor.
- Mass: 50 g

3.1.2.3 Air Duct:

Air duct to convey air from either 24m3/h Large Fan (INLET () of Air Duct), along the four lamps (L0 : 1C6412 pre-calibrated lamp, L1 : 1C6412 pre-calibrated lamp, L2 :

1C6412 pre-calibrated lamp, L3 : 1C6412 pre-calibrated lamp), to the TempSensor : 1C6101 Temperature Sensor, up to the OUTLET() of Air Duct. Cross section of Air Duct shall possibly be in excess of 2000 mm2, equivalent to a circular duct of 50 mm diameter.

3.1.3 Electrical Specifications:

The 1C6412 1U Light source requires two separate Power input supplies, which are as follow:

- DC 6v
- PWR_12v

DC 6v @ 500mA is the logic supply and supplying voltage to all logic components on board. In 1C6412 1U Light source both logic side and power side are completely galvanically isolated that's why logic supply and high power supply have separate grounds.

PWR_12v @ 20Amps is high power rating supply used to driver lamps and fan in 1C6412 1U Light source.

3.2 1C604 Main Controller:

The 1C604 Main Controller is the master controlling element of low cost solar simulator. 1C604 Main Controller intended to control 1C6412 1U Light Source, control over motor driver for rotation of turn table and interface with external world either by USB or Ethernet. All modules in the Low Cost Solar simulator communicate via I2C communication.

Note: Designing of 1C604 Main Controller falls in future work.

3.3 1C651 1U Fixed Stand:

A 1C65 Stand whose position and orientation cannot be changed. It will be capable of hosting either 1U CubeSat or 2U CubeSat or 3U CubeSat and (consequently) either one, two or three 1C6412 1U Light Sources.

3.4 1C653 Stand to Light Separator:

A separator which keeps Satellite Part and 1C6412 1U Light Source at a distance of Light Source to Satellite Distance.

3.5 System Communication:

There are three types of communication used in this system which are as follow:

1. I2C Communication
2. SPI Communication
3. USB 2.0 Communication

3.5.1 I2C Communication:

Solar simulator communicates with external world via I2C communication. The reason for choosing I2C is to make it modular in such a way that 255 solar simulators can be attached together and connects with each other and to the main PC or main controller board via I2C. I2C is preferred over UART because UART is point to point communication and this is not feasible when required communication should be with multiple devices together.

3.5.1.1 I2C Communication details:

I²C is generically referred to as two-wire interface and is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, cell phone, or other electronic device.

Figure 3.7 shows basic schematic for one master and three salve nodes I2C communication.

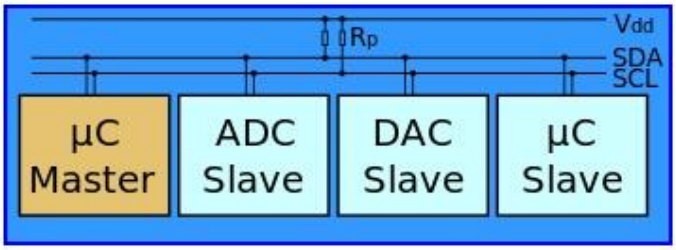


Figure 3.7: One Master and three slave nodes I2C Communication

I²C uses only two bidirectional open collector or open-drain lines. One is Serial Data Line (SDA) another is Serial Clock (SCL). Because of open drain lines pull up resistors are required. Normal supply voltage is from 3.3v to 5.5v but higher voltages are also allowed.

The I²C has a 7-bit or a 10-bit depends on the user. Usually i2C has speed of 100Kb/sec but with the new technology development I2C communication revised and now higher speeds are achievable. It can go up to 3.5Mb/sec.

The maximum number of nodes is limited by the address space, and also by the total bus capacitance of 400 pF, which restricts practical communication distances to a few meters.

3.5.1.1.1 Reference design:

In reference design I considered 7 bit address and single master multiple slaves. There are only two options for the nodes in I2C communication.

- Master node
- Slave node

Master node is the only node that issues the clock and addresses slaves and slave node is the only node that receives clock line and data line.

There are four collective functionality of master and slave nodes which are as follow:

- master transmit
- master receive
- slave transmit
- slave receive

In master transmit mode, master is sending data to slave. In master receive mode, master is receiving data from slave. In slave transmit mode, slave is sending data to the master. In slave receive; slave is receiving data from master. As I2C is two wire bidirectional communication that's why only one mode can be used at a time.

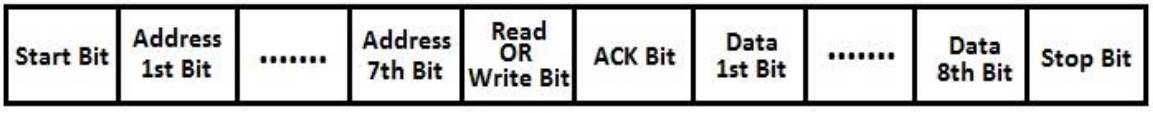


Figure 3.8: Basic communication pattern for I2C communication

All the communication is started by master because it's the only clock issuing node. The master is initially in master transmit mode and sends a start bit which is followed by the

7-bit address of the slave, which is finally followed by a single bit representing whether it wishes to write(0) to or read(1) from the slave.

If the slave exists on the bus then it will respond with an ACK bit (active low for acknowledged) for that address. The master then continues in either transmit or receive mode (according to the read/write bit it sent), and the slave continues in its complementary mode.

The address and the data bytes are sent most significant bit first. The start bit is indicated by a high-to-low transition of SDA with SCL high; the stop bit is indicated by a low-to-high transition of SDA with SCL high.

If the master wishes to write to the slave then it repeatedly sends a byte with the slave sending an ACK bit. (In this situation, the master is in master transmit mode and the slave is in slave receive mode.)

If the master wishes to read from the slave then it repeatedly receives a byte from the slave, the master sending an ACK bit after every byte but the last one. (In this situation, the master is in master receive mode and the slave is in slave transmit mode.)

The master then ends transmission with a stop bit, or it may send another START bit if it wishes to retain control of the bus for another transfer (a "combined message").

At the physical layer, both SCL & SDA lines are of open-drain design, thus, pull-up resistors are needed. Pulling the line to ground is considered a logical zero while letting the line float is a logical one. This is used as a channel access method. High speed

systems (and some others) also add a current source pull up, at least on SCL; this supports faster rise times and higher bus capacitance.

An important consequence of this is that multiple nodes may be driving the lines simultaneously. If *any* node is driving the line low, it will be low. Nodes that are trying to transmit a logical one (i.e. letting the line float high) can see this, and thereby know that another node is active at the same time.

When used on SCL, this is called "clock stretching" and gives slaves a flow control mechanism. When used on SDA, this is called arbitration and ensures there is only one transmitter at a time.

When idle, both lines are high. To start a transaction, SDA is pulled low while SCL remains high. Releasing SDA to float high again would be a stop marker, signaling the end of a bus transaction. Although legal, this is typically pointless immediately after a start, so the next step is to pull SCL low.

Except for the start and stop signals, the SDA line only changes while the clock is low; transmitting a data bit consists of pulsing the clock line high while holding the data line steady at the desired level.

While SCL is low, the transmitter (initially the master) sets SDA to the desired value and (after a small delay to let the value propagate) lets SCL float high. The master then waits for SCL to actually go high; this will be delayed by the finite rise-time of the SCL signal (the RC time constant of the pull-up resistor and the parasitic capacitance of the bus), and may be additionally delayed by a slave's clock stretching.

Once SCL is high, the master waits a minimum time (4 μs for standard speed I²C) to ensure the receiver has seen the bit, then pulls it low again. This completes transmission of one bit.

After every 8 data bits in one direction, an "acknowledge" bit is transmitted in the other direction. The transmitter and receiver switch roles for one bit and the erstwhile receiver transmits a single 0 bit (ACK) back. If the transmitter sees a 1 bit (NACK) instead, it learns that:

1. (If master transmitting to slave) The slave is unable to accept the data. No such slave, command not understood or unable to accept any more data.
2. (If slave transmitting to master) The master wishes the transfer to stop after this data byte.

3.5.1.1.2 Clock stretching using SCL:

One of the best features of the I²C protocol is clock stretching. Slave device may hold the clock line low after receiving or sending a byte, indicating that it is not yet ready to process more data. The master that is communicating with the slave may not finish the transmission of the current bit, but must wait until the clock line actually goes high. If the slave is clock stretching, the clock line will still be low. The same is true if a second, slower, master tries to drive the clock at the same time. The master must wait until it observes the clock line going high, and an additional minimum time (4 μs for standard 100 kbit/s I²C) before pulling the clock low again.

Although the master may also hold the SCL line low for as long as it desires, the term "clock stretching" is normally used only when slaves do it. Although in theory any clock pulse may be stretched, generally it is the intervals before or after the acknowledgment bit which are used. For example, if the slave is a microcontroller, its I²C interface will stretch the clock after each byte, until the software decides whether to send a positive acknowledgment or a NACK.

Clock stretching is the only time in I2C where the slave drives SCL. Many slaves do not need to clock stretch and thus treat SCL as strictly an input with no circuitry to drive it. Some masters, such as those found inside custom ASICs may not support clock stretching; often these devices will be labeled as a "two-wire interface" and not I²C.

3.5.1.1.3 Timing diagram:

Data transfer is initiated with the START bit (S) when SDA is pulled low while SCL stays high. Then, SDA sets the transferred bit while SCL is low (blue) and the data is sampled (received) when SCL rises (green). When the transfer is complete, a STOP bit (P) is sent by releasing the data line to allow it to be pulled up while SCL is constantly high. In order to avoid false marker detection, the level on SDA is changed on the negative edge and is captured on the positive edge of SCL.

Figure 3.9 shows basic timing diagram for I2C communication.

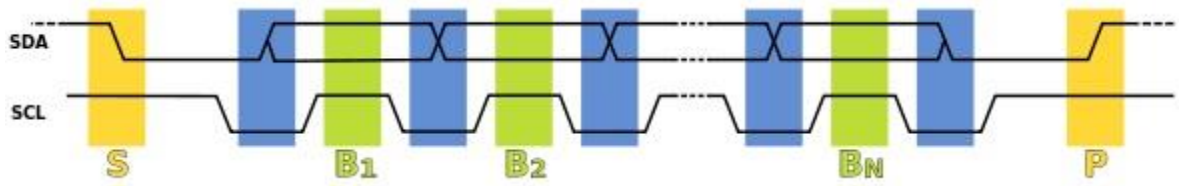


Figure 3.9: Basic timing diagram for I2C Communication

3.5.2 SPI Communication:

Control board has extra connector which is wired with SPI module of MSP430F5438. This connector is left free and can be used to attach different sensors in future.

3.5.2.1 SPI Communication Details:

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard, named by Motorola, that operates in full duplex mode. Devices communicate in master or slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select lines.

Figure 3.10 shows basic wire configuration for single master and single slave for SPI communication.

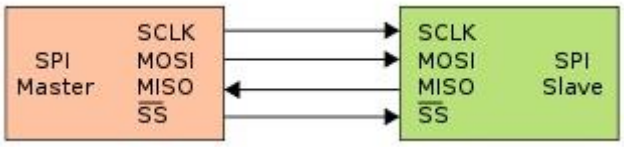


Figure 3.10: Basic Single master and single slave interface for SPI

The SPI bus has four logic signals:

- SCLK
- MOSI
- MISO
- SS

SCLK is clock output from master. MOSI is master output and slave input. This signal is generated by master. MISO is master input and slave output. SS is slave select and is also

called chip select. SS used to select slave device when there are multiple slave devices on bus.

3.5.2.1.1 SPI Operation:

The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. Some slaves require the falling edge (high to low transition) of the chip select to initiate an action such as the Maxim MAX1242 ADC, which starts conversion on said transition. With multiple slave devices, an independent SS signal is required from the master for each slave device.

Most slave devices have tri-state outputs so their MISO signal becomes high impedance (logically disconnected) when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.

3.5.2.1.2 Data transmission:

A typical hardware setup using two shift registers to form an inter-chip circular buffer. To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1–100 MHz.

Figure 3.11 shows the typical hardware setup using two shift registers to form an inter-chip circular buffer.

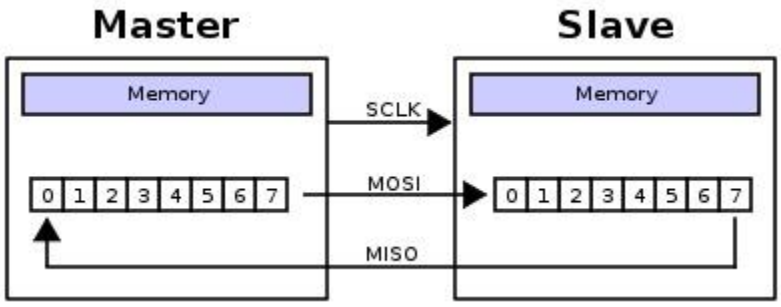


Figure 3.11: Basic hardware setup of SPI using two shift registers

The master then transmits the logic 0 for the desired chip over chip select line. A logic 0 is transmitted because the chip select line is active low, meaning its off state is a logic 1; on is asserted with a logic 0. If a waiting period is required (such as for analog-to-digital conversion), then the master must wait for at least that period of time before starting to issue clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

1. the master sends a bit on the MOSI line; the slave reads it from that same line
2. the slave sends a bit on the MISO line; the master reads it from that same line

Not all transmissions require all four of these operations to be meaningful but they do happen.

Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it,

such as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data[1] and the process repeats.

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

Transmissions often consist of 8-bit words, and a master can initiate multiple such transmissions if it wishes/needs. However, other word sizes are also common, such as 16-bit words for touchscreen controllers or audio codecs, like the TSC2101 from Texas Instruments; or 12-bit words for many digital-to-analog or analog-to-digital converters.

Every slave on the bus that hasn't been activated using its chip select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.

3.5.2.1.3 Independent slave SPI configuration

Typical SPI bus: master and three independent slaves. In the independent slave configuration, there is an independent chip select line for each slave. This is the way SPI is normally used. Since the MISO pins of the slaves are connected together, they are required to be tri-state pins.

Figure 3.12 shows the configuration for master and three independent slaves.

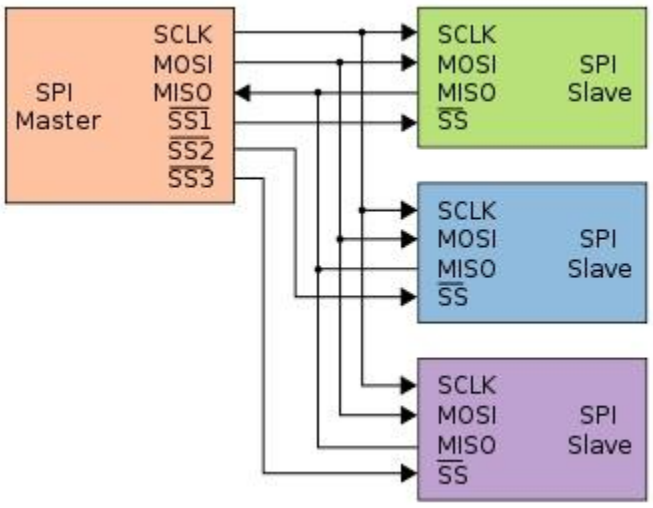


Figure 3.12: Configuration for one master and three slaves for SPI

3.5.3 USB Communication:

In this project there is a debug port which is based on USB 2.0 communication. This communication port helps user to troubleshoot the system in a quickest way.

3.5.3.1 USB Communication details:

Universal Serial Bus (USB) is a set of interface specifications for high speed wired communication between electronics systems peripherals and devices with or without PC/computer. The USB was originally developed in 1995 by many of the industry leading companies like Intel, Compaq, Microsoft, Digital, IBM, and Northern Telecom.

The major goal of USB was to define an external expansion bus to add peripherals to a PC in easy and simple manner. The new external expansion architecture, highlights,

1. PC host controller hardware and software

2. Robust connectors and cable assemblies

3. Peripheral friendly master-slave protocols

4. Expandable through multi-port hubs.

USB offers users simple connectivity. It eliminates the mix of different connectors for different devices like printers, keyboards, mice, and other peripherals. That means USB-bus allows many peripherals to be connected using a single standardized interface socket. Another main advantage is that, in USB environment, DIP-switches are not necessary for setting peripheral addresses and IRQs. It supports all kinds of data, from slow mouse inputs to digitized audio and compressed video.

USB also allows hot swapping. The "hot-swapping" means that the devices can be plugged and unplugged without rebooting the computer or turning off the device. That means, when plugged in, everything configures automatically. So the user needs not worry about terminations, terms such as IRQs and port addresses, or rebooting the computer. Once the user is finished, they can simply unplug the cable out, the host will detect its absence and automatically unload the driver. This makes the USB a plug-and-play interface between a computer and add-on devices.

The loading of the appropriate driver is done using a PID/VID (Product ID/Vendor ID) combination. The VID is supplied by the USB Implementer's forum

Fig 3.13: The USB "trident" logo

The USB has already replaced the RS232 and other old parallel communications in many applications. USB is now the most used interface to connect devices like mouse, keyboards, PDAs, game-pads and joysticks, scanners, digital cameras, printers, personal media players, and flash drives to personal computers. Generally speaking, USB is the most successful interconnect in the history of personal computing and has migrated into consumer electronics and mobile products.

USB sends data in serial mode i.e. the parallel data is serialized before sends and de-serialized after receiving.

The benefits of USB are low cost, expandability, auto-configuration, hot-plugging and outstanding performance. It also provides power to the bus, enabling many peripherals to operate without the added need for an AC power adapter.

3.5.3.1.1 Various versions USB:

As USB technology advanced the new version of USB are unveiled with time. Let us now try to understand more about the different versions of the USB.

USB1.0: Version 0.7 of the USB interface definition was released in November 1994. But USB 1.0 is the original release of USB having the capability of transferring 12 Mbps, supporting up to 127 devices. And as we know it was a combined effort of some large players on the market to define a new general device interface for computers. This USB 1.0 specification model was introduced in January1996. The data transfer rate of this version can accommodate a wide range of devices, including MPEG video devices, data gloves, and digitizers. This version of USB is known as full-speed USB.

Since October-1996, the Windows operating systems have been equipped with USB drivers or special software designed to work with specific I/O device types. USB got integrated into Windows 98 and later versions. Today, most new computers and peripheral devices are equipped with USB.

USB1.1: USB 1.1 came out in September 1998 to help rectify the adoption problems that occurred with earlier versions, mostly those relating to hubs.

USB 1.1 is also known as full-speed USB. This version is similar to the original release of USB; however, there are minor modifications for the hardware and the specifications. USB version 1.1 supported two speeds, a full speed mode of 12Mbits/s and a low speed mode of 1.5Mbits/s. The 1.5Mbits/s mode is slower and less susceptible to EMI, thus reducing the cost of ferrite beads and quality components.

USB2.0: Hewlett-Packard, Intel, LSI Corporation, Microsoft, NEC, and Philips jointly led the initiative to develop a higher data transfer rate than the 1.1 specifications. The USB 2.0 specification was released in April 2000 and was standardized at the end of 2001. This standardization of the new device-specification made backward compatibility possible, meaning it is also capable of supporting USB 1.0 and 1.1 devices and cables.

Supporting three speed modes (1.5, 12 and 480 megabits per second), USB 2.0 supports low-bandwidth devices such as keyboards and mice, as well as high-bandwidth ones like high-resolution Web-cams, scanners, printers and high-capacity storage systems.

USB 2.0, also known as hi-speed USB. This hi-speed USB is capable of supporting a transfer rate of up to 480 Mbps, compared to 12 Mbps of USB 1.1.

3.5.3.1.2 USB system overview:

The USB system is made up of a host, multiple numbers of USB ports, and multiple peripheral devices connected in a tiered-star topology. To expand the number of USB ports, the USB hubs can be included in the tiers, allowing branching into a tree structure with up to five tier levels.

The tiered star topology has some benefits. Firstly power to each device can be monitored and even switched off if an overcurrent condition occurs without disrupting other USB devices. Both high, full and low speed devices can be supported, with the hub filtering out high speed and full speed transactions so lower speed devices do not receive them.

The USB is actually an addressable bus system, with a seven-bit address code. So it can support up to 127 different devices or nodes at once (the "all zeroes" code is not a valid address). However it can have only one host: the PC itself. So a PC with its peripherals connected via the USB forms a star local area network (LAN).

On the other hand any device connected to the USB can have a number of other nodes connected to it in daisy-chain fashion, so it can also form the hub for a mini-star sub-network. Similarly it is possible to have a device, which purely functions as a hub for other node devices, with no separate function of its own. This expansion via hubs is possible because the USB supports a tiered star topology. Each USB hub acts as a kind of traffic cop. for its part of the network, routing data from the host to its correct address and preventing bus contention clashes between devices trying to send data at the same time.

On a USB hub device, the single port used to connect to the host PC either directly or via another hub is known as the upstream port, while the ports used for connecting other devices to the USB are known as the downstream ports. USB hubs work transparently as

far as the host PC and its operating system are concerned. Most hubs provide either four or seven downstream ports or less if they already include a USB device of their own.

The host is the USB system's master, and as such, controls and schedules all communications activities. Peripherals, the devices controlled by USB, are slaves responding to commands from the host. USB devices are linked in series through hubs. There always exists one hub known as the root hub, which is built in to the host controller.

A physical USB device may consist of several logical sub-devices that are referred to as device functions. A single device may provide several functions, for example, a web-cam (video device function) with a built-in microphone (audio device function). In short, the USB specification recognizes two kinds of peripherals: stand-alone (single function units, like a mouse) or compound devices like video camera with separate audio processor. The logical channel connection host to peripheral-end is called pipes in USB. A USB device can have 16 pipes coming into the host controller and 16 going out of the controller.

The pipes are unidirectional. Each interface is associated with single device function and is formed by grouping endpoints.

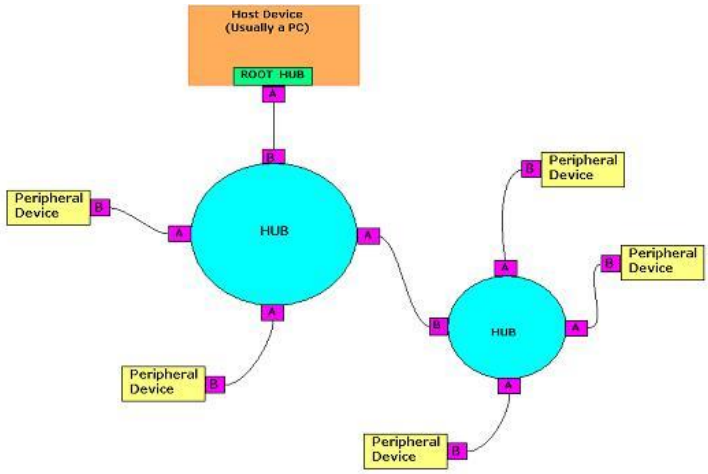


Fig 3.14: The USB "tiered star" topology

The hubs are bridges. They expand the logical and physical fan-out of the network. A hub has a single upstream connection (that going to the root hub, or the next hub closer to the root), and one to many downstream connections.

Hubs themselves are considered as USB devices, and may incorporate some amount of intelligence. We know that in USB users may connect and remove peripherals without powering the entire system down. Hubs detect these topology changes. They also source power to the USB network. The power can come from the hub itself (if it has a built-in power supply), or can be passed through from an upstream hub.

3.5.3.1.3 USB connectors & the power supply:

Connecting a USB device to a computer is very simple -- you find the USB connector on the back of your machine and plug the USB connector into it. If it is a new device, the operating system auto-detects it and asks for the driver disk. If the device has already been installed, the computer activates it and starts talking to it.

The USB standard specifies two kinds of cables and connectors. The USB cable will usually have an "A" connector on one end and a "B" on the other. That means the USB devices will have an "A" connection on it. If not, then the device has a socket on it that accepts a USB "B" connector.

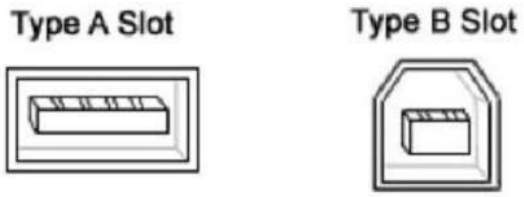


Fig 3.15: USB Type A & B Connectors

The USB standard uses "A" and "B" connectors mainly to avoid confusion:

1. "A" connectors head "upstream" toward the computer.
2. "B" connectors head "downstream" and connect to individual devices.

By using different connectors on the upstream and downstream end, it is impossible to install a cable incorrectly, because the two types are physically different.

Individual USB cables can run as long as 5 meters for 12Mbps connections and 3m for 1.5Mbps. With hubs, devices can be up to 30 meters (six cables' worth) away from the host. Here the high-speed cables for 12Mbps communication are better shielded than their less expensive 1.5Mbps counterparts. The USB 2.0 specification tells that the cable delay to be less than 5.2 ns per meter

Inside the USB cable there are two wires that supply the power to the peripherals-- +5 volts (red) and ground (brown)-- and a twisted pair (yellow and blue) of wires to carry the data. On the power wires, the computer can supply up to 500 milliamps of power at 5

volts. A peripheral that draws up to 100ma can extract all of its power from the bus wiring all of the time. If the device needs more than a half-amp, then it must have its own power supply. That means low-power devices such as mice can draw their power directly from the bus. High-power devices such as printers have their own power supplies and draw minimal power from the bus. Hubs can have their own power supplies to provide power to devices connected to the hub.

Table 3.2: USB pin connections

Pin No:	Signal	Color of the cable
1	+5V power	Red
2	- Data	White / Yellow
3	+Data	Green / Blue
4	Ground	Black/Brown

USB hosts and hubs manage power by enabling and disabling power to individual devices to electrically remove ill-behaved peripherals from the system. Further, they can instruct devices to enter the suspend state, which reduces maximum power consumption to 500 microamps (for low-power, 1.5Mbps peripherals) or 2.5ma for 12Mbps devices.

In short, the USB is a serial protocol and physical link, which transmits all data differentially on a single pair of wires. Another pair provides power to downstream peripherals.

Note that although USB cables having a Type A plug at each end are available, they should never be used to connect two PCs together, via their USB ports. This is because a USB network can only have one host, and both would try to claim that role. In any case, the cable would also short their 5V power rails together, which could cause a damaging

current to flow. USB is not designed for direct data transfer between PCs. But the "sharing hubs" technique allows multiple computers to access the same peripheral device(s) and work by switching access between PCs, either automatically or manually.

3.6 1C601 Power Driver Board:

See the Chapter IV.

3.7 1C601 Filter Board:

See the Chapter V.

3.8 1C601 Control Board:

See the Chapter VI.

CHAPTER IV: 1C601 Power Driver Board

1C601 Power Driver Board is the driver element of the lamp. It converts pwm TTL 3.3v input and give output pwm 12v. 1C601 Power driver board has five close looped outputs drive channels. Each channel is galvanically isolated and provides current sense feedback.

The power driver board has following features:

1. Logic Supply range 3.3v to 5v
2. Five independent pwm output channels
3. 3000Vrms for 1 minute Galvanic isolation between inputs and outputs
4. Current sense feedback for each pwm output channel with galvanic isolation
5. EMI and EMC compliant
6. Drive Frequency can go up to 100 KHz
7. Output PWM voltage range from 4.5v to 18v
8. Max Current rating 10 Amp for first four channels and 5 Amp for last channel

4.1 Components:

1C601 power driver Board has different components for different functionality.

Following are the list of Components:

- Gate Driver IC
- Half Bridge
- Current sense
- Main power input detection

4.1.1 Gate Driver IC:

Gate driver used in power driver board is ADuM3223 by Analog Devices. ADuM3223 is half bridge gate driver ic with independent isolated high and low side outputs. The isolation rating is 3000Vrms.Each ic provides two independent isolated channels and both channels are used to driver half bridge together.

Following are the features of ADuM3223:

1. High-side or low-side relative to input: 537 V peak
2. High-side to low-side differential: 800 V peak
3. High frequency operation: 1 MHz maximum
4. 3.3 V to 5 V CMOS input logic
5. 4.5 V to 18 V output drive
6. 54 ns maximum isolator and driver propagation delay
7. High junction temperature operation: 125°C
8. Thermal shutdown protection

Figure 4.1.1 shows the functional block diagram of ADuM3223.As power driver board should have five outputs that's why it has five ADuM3223 for each channel.

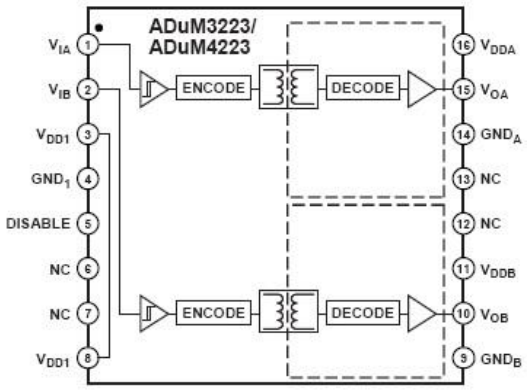


Figure 4.1: Function Block Diagram of ADuM3223

ADuM3223 is 16 pin in SOIC package. Table 3.1 shows the pin names and their description.

Table 4.1: Pin description of ADuM3223

Pin Number	Pin Name	Description
1	V_{IA}	Logic Input A
6, 7, 12, 13	NC	No Connect
2	V_{IB}	Logic Input B
3, 8	V_{DD1}	Input Supply Voltage
4	GND_1	Ground Reference for Input Logic Signals
5	DISABLE	Input Disable. Disable the isolator and refresh the circuit
9	GND_B	Ground Reference for Output B
10	V_{OB}	Output B
11	V_{DDB}	Output B Supply Voltage
14	GND_A	Ground Reference for Output A
15	V_{OA}	Output A
16	V_{DDA}	Output A Supply Voltage

4.1.1.1 ADuM3223 Schematic:

To drive the half bridge using ADuM3223 bootstrap circuit. Voltage at the gate of high side mosfet and low side mosfet is different because low side mosfet is ground referenced and high side is floating that's why bootstrap circuit is required to operate half bridge properly.

Following is the schematic of ADuM3223 with boot strap circuit:

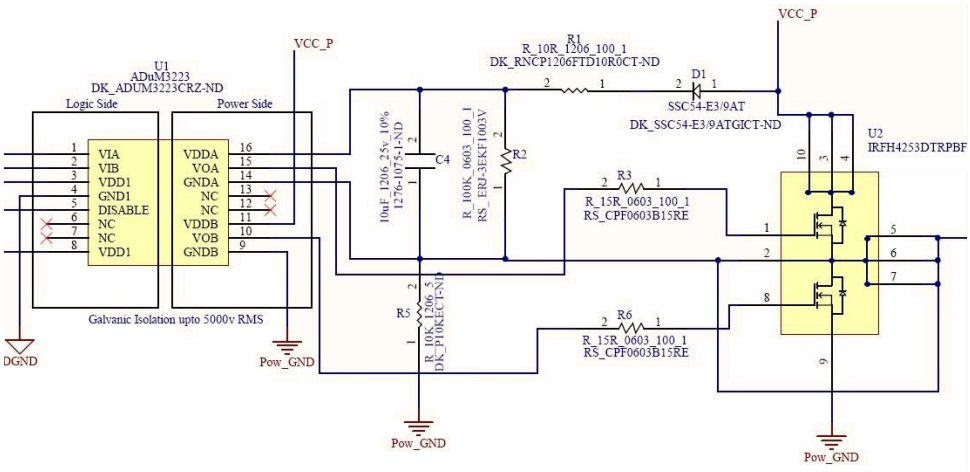


Figure 4.2: Application circuit for bootstrap

In the above Figure 4.2, the bootstrap drive circuit is implemented with the capacitor **C4**, the resistors **R1** and **R5**, and the diode **D1**. Immediately after power on, the PWM does not come instantly, and all the MOSFETs are in the high impedance state until all dc voltages are settled. During this time capacitor **C4** is charged by the dc supply through the path **R1**, **D1**, **C4**, and **R5**. The charged capacitor **C4** provides the voltage for high-side gate drive. The time constant for **C4** charging is $\tau = (\mathbf{R1} + \mathbf{R5})\,\mathbf{C4}$.

When the MOSFETs switch due to the PWM signal, the lowside mosfet is turned on, and the high-side mosfet is turned off. The GNDA of the high-side is pulled down to ground, and the capacitor C4 is charged. When high-side mosfet is turned on, and lowside mosfet is turned off, the GNDA is pulled up to the dc supply voltage. The diode **D1** is reversed biased, and the **C4** voltage forces the VDDA voltage of the ADuM3223 to approximately 24 V. The capacitor **C4**, therefore, maintains a voltage of approximately 12 V between the VDDA and GNDA terminals of the ADuM3223. In this manner, the gate drive voltage to the high-side MOSFET is always referenced to the floating source voltage of high-side MOSFET. The resistor R5 discharges during switch off and have no function during switching.

The resistor **R5** starts up the bootstrap circuit. Immediately after power-on, dc voltages are not established, and the MOSFETs are off. Under these conditions **C4** is charged through the path **R1**, **R5**, **D1**, **V**s, described by the following equation:

$$v_C(t) = (V_S - V_D)(1 - e^{-t/\tau})$$

Where $v_C(t)$ is the capacitor voltage, **V**s is the supply voltage which is 12v in this project, $\mathbf{V_D}$ is the diode voltage drop, and τ is the time constant, τ = (**R1** + **R5**) **C4**. The circuit values are **R1** = 10 Ω, **C4** = 10 μF, $\mathbf{V_D}$ = 0.5 V, and $\mathbf{V_S}$ = 12 V. From the equation, it takes one time constant (10 ms) to charge the capacitor to 67% of its final value for **R5** = 10k Ω.

If we use an inductor as the load, the current flowing through the inductor will change linearly if a constant voltage is applied. The voltage, U, is 12 V, and if we ignore the voltage drop across the MOSFETs due to the on–resistance, the following equation is true:

$$U = L\frac{di}{dt}$$

With a 50 kHz, 8% duty cycle PWM signal and a 4 μH Coilcraft power inductor (SER2014-402) as the load, the load current waveform is as shown in Figure 4.3.

Figure 4.3: Load Current as a Function of the PWM Pulses with a 4 μH Load

4.1.2 Half Bridge:

In this project instead of using two separate mosfets for each channel. I used single package containing one half bridge (IRFH4253DPbF).Following are the features of IRFH4253DPbF:

1. Control and synchronous MOSFETs in one package
2. Low charge control MOSFET (10nC typical)
3. Low RDSON synchronous MOSFET (<1.45mΩ)
4. Intrinsic Schottky Diode with Low Forward Voltage on Q2
5. Max V_{DSS} is 25v
6. Max I_D is 35 Amps

This is an excellent and small package containing half bridge with required power rating. The dimension of package is 6mm x 5mm.Figure 4.4 shows internal structure of IC and pin description.

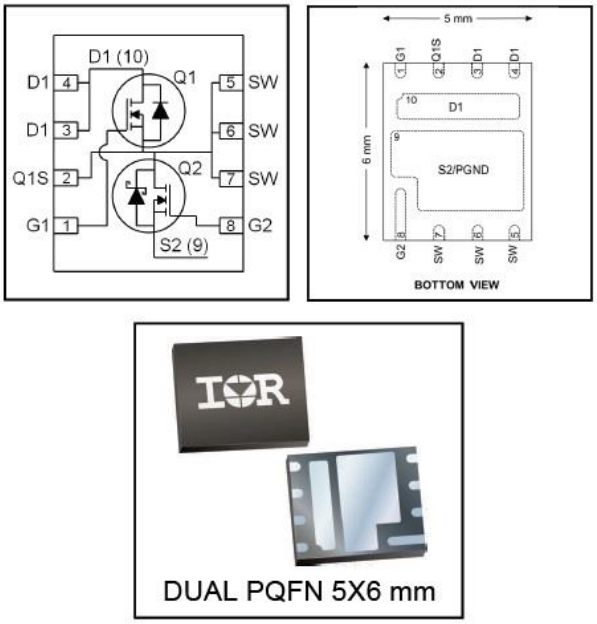


Figure 4.4 : PIN Configuration of IRFH4253DPbF

Figure 4.5 and Figure 4.6 shows typical output characteristics of Q1 and Q2 in IRFH4253DPbF.

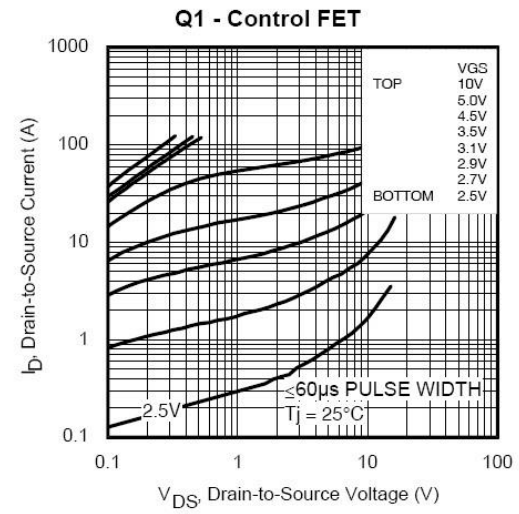


Figure 4.5: Typical Output characteristics of Q1

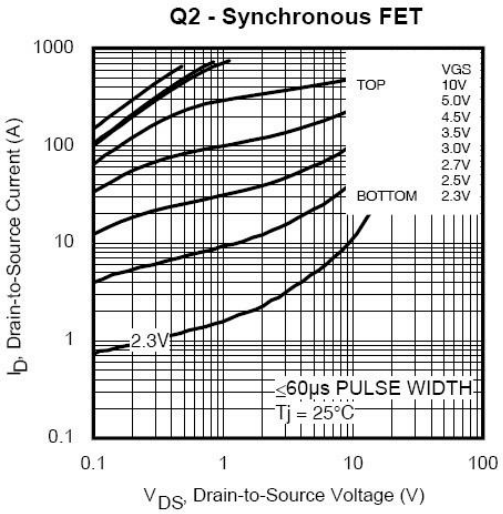


Figure 4.6: Typical output characteristics of Q2

4.1.3 Current Sense:

Basic current sense requirement for this project was to sense current with galvanic isolation and should not have high resistance. ACS711 by Allegro current sensor is used in this project. ACS711 has following features:

1. Single package solution.
2. No need of external sense resistor.
3. Very low internal conductor resistance 1.2mΩ.
4. Current measure range -25A to +25A.
5. 100 kHz Bandwidth.
6. 3.3v to 5.5v supply voltage range.
7. Built in electrostatic shield for stable analog output.
8. Output is ratiometric from supply voltage.

ACS711 works on Hall Effect principle. Hall Effect is the generation of voltage difference across electrical conductor horizontal to electric current in electric conductor and perpendicular to the electric current. Hall Effect sensors are used to measure intensity of magnetic field. The two sense leads IP- and IP+ of the sensor are place in series with the main load current path. The sensor senses the intensity of magnetic field which is proportional to the amount of current passing through its leads. In this ways both current sense and galvanic isolation achieved.

Figure 4.7 shows basic Hall Effect principle.

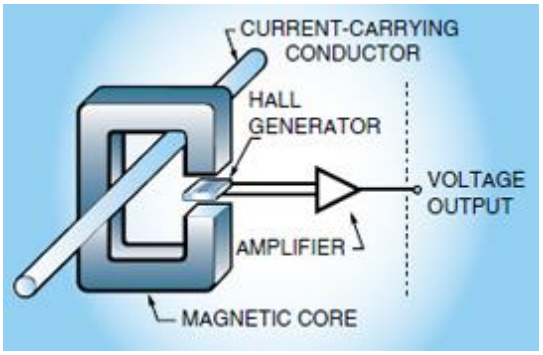


Figure 4.7: Basic Hall Effect Principle

Traditional way of measuring current is by using current sense resistor. In this method the sense resistor place in series with the load current path. As we know that according to ohm law V=IR, If resistance is known and current we can easily measure current. It is most widely and cheep method to measure current but there is one drawback, No galvanic isolation between logic side and power side which can cause serious damage to the logic circuitry and drastically reduce the performance of the circuit.

Figure 4.8 shows basic current sensing circuit by using current sense resistor.

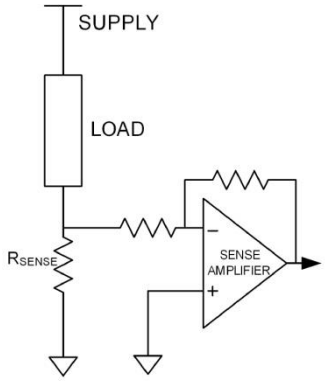


Figure 4.8: Basic Current sense using shunt resistor

As shown is above figure both power ground and analog ground are same which can be big reason of noise transfer until unless properly designed and extra components deployed which in the increase the cost.

Table 4.2 shows the pin name and their description while Figure 4.9 shows pin configuration.

Pin Name	Pin Number	Description
GND	5	Signal Ground
FAULT	6	Over Current Fault, Active low
IP-	3 and 4	Terminals for Current
IP+	1 and 2	Terminals for Current
NC	---	No Connection
VCC	8	Supply Power
VIOUT	7	Analog Output Signal

Table 4.2: Pin description of ACS711

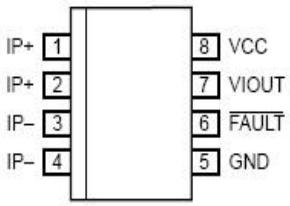


Figure 4.9:ACS711 PIN Configuration

ACS711 has linear Hall Effect sensor embedded near the copper conduction path present near the surface of die. When the current applied to the copper conduction path it will generate the magnetic field and which is sense by embedded Hall Effect sensor and

converted to proportional voltage. The output voltage of the device is directly proportional to the current flow from IP+ to IP-.

4.1.3.1 Typical Application circuit:

Typical application circuit for ACS711 is as follow:

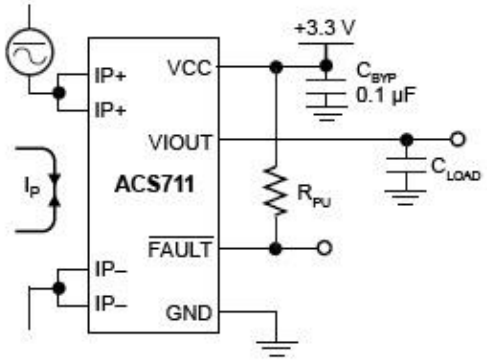


Figure 4.10: Typical Application circuit for ACS711

As shown in Figure 4.10 current flow from either **IP+** to **IP-** or **IP-** to **IP+**, the device will give proportional analog voltage through pin VIOUT. $\mathbf{R_{PU}}$ typical value mentioned in datasheet which is 1 KΩ. Output Capacitive load should not exceed from value 1nF.

The sensitivity of this device is the product of magnetic circuit sensitivity and the linear IC amplification gain. The linear amplification gain is preprogrammed in factory to optimize the sensitivity. Figure 4.11 shows the typical response time of ACS711.

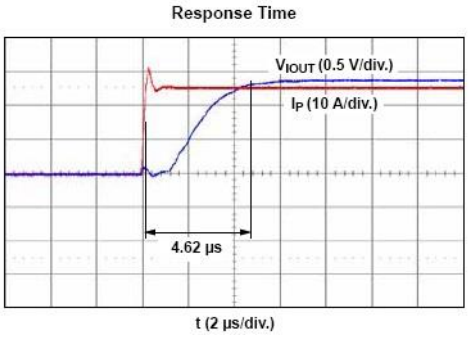


Figure 4.11: Typical response time of ACS711

4.1.3.2 ACS711 with Power Driver Schematic

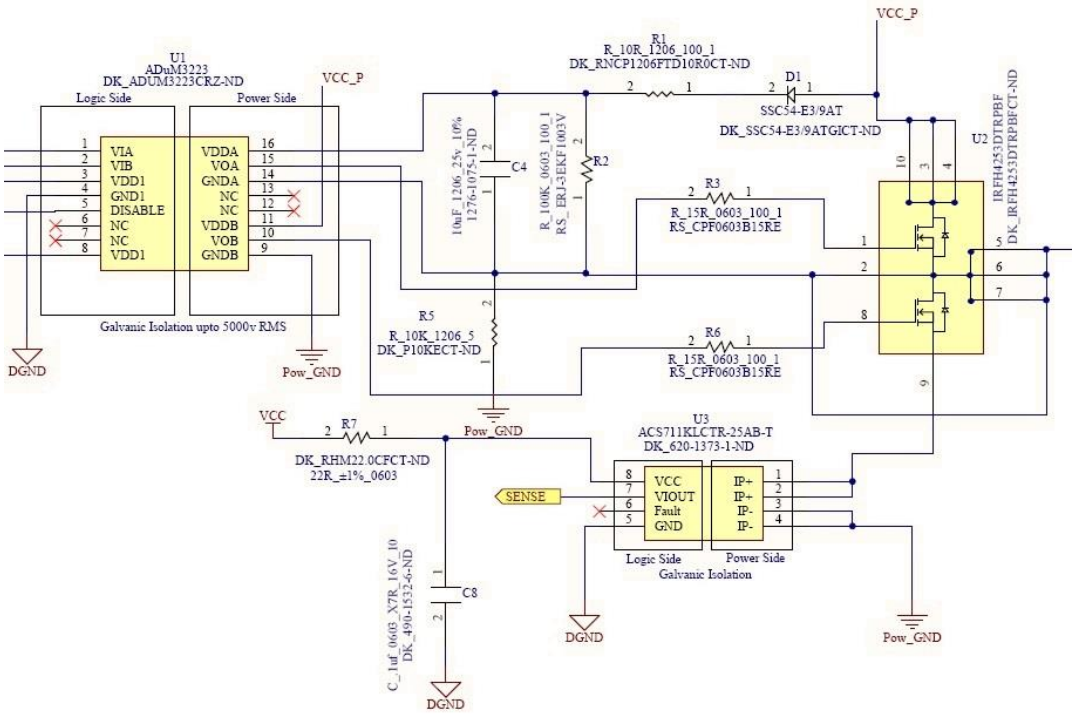


Figure 4.12: ACS711 with power driver schematic

As shown in above Figure 4.12 ACS711 is connected in series with the load current path. In this project current sensor is not used only for current sensing or over current protection. There are two other main functions of ACS711 in this project which are as follow:

1. Lamp broken detection.
2. Tight control of power by using current sense as feedback.

Broken lamp can be easily detected by putting a check in firmware, if measured current is zero than its mean lamp is broken. As we are using commercially available low cost lamps so there are many chances that one lamp behave different than the others, To counter this situation we took current consumption as feedback and match with other

lamps current consumption. This will make sure that all lamps are glowing at same intensity.

4.1.4 Main Power Supply Input Detection:

In this project the power failure detection feature is added by continuous monitoring of main power input. The power driver has complete galvanic isolation at every level. Power input detection cannot be without isolation otherwise there will no purpose of whole effort to create isolation. In power input detection section the isolation is created by using optocoupler.

Schematic of power supply input detection is shown in Figure 4.13:

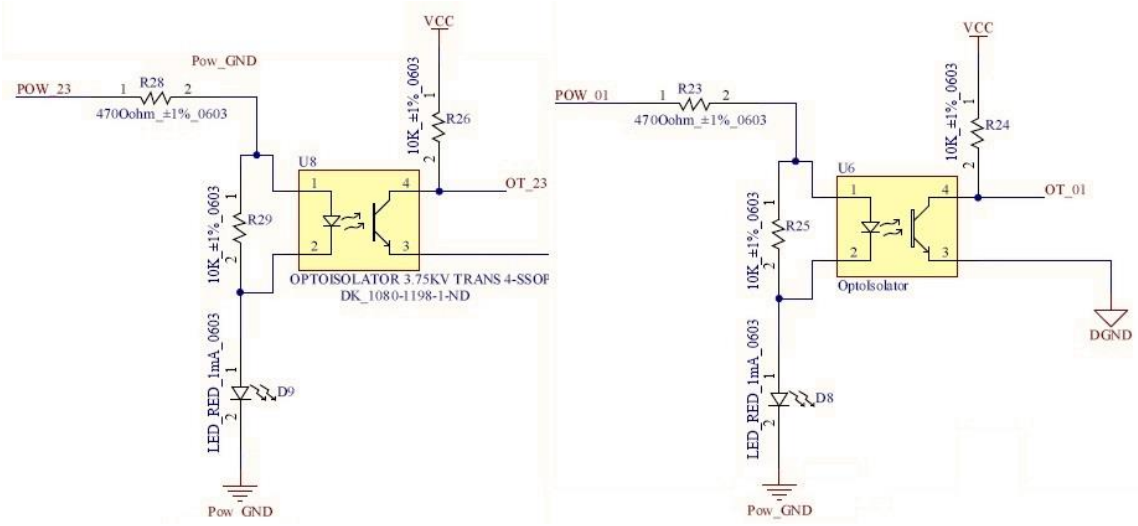


Figure 4.13: Power Supplly Input Detection Circuit

For visual power input detection LED **D8** and **D9** are added to the circuit. When power input is present LED will glow. There are two main power input lines (**POW_01** and **POW_23**). For both power lines separate detection circuit is designed. The outputs (**OT_01** and **OT_23**) of both circuits are connected to digital pins **P1.7** and **P2.0** of microcontroller.

4.2 Thermal Analysis:

1C601 Power Driver board contains four high power pwm channels. Each channel have Half bridge comprising of two NMOS. Main target of the Thermal analysis is to calculate the Temperature rise by the power element (Mosfet).

First I calculate the T_{rise} for single Half Bridge and then we can multiply with 4 because all half bridges are same. For finding Trise first calculations of power dissipation is required.

Following is the formula to calculate the power consumption:

$$PD_{Total} = PD_{Resistive} + PD_{Switching} \ldots\ldots. \text{Eq A}$$

Where

$$PD_{Resistive} = I^2 \text{ x } RDS_{ON}(HOT) \ldots\ldots. \text{Eq B}$$

As we know that:

$$RDS_{(ON)HOT} = RDS_{(ON)SPEC} [1+ \text{Temperature\_Coeffient x}(T_{j(HOT)} - T_{SPEC})] \ldots\ldots\ldots \text{Eq C}$$

1. $RDS_{(ON)HOT}$ is ON resistance with incorporation of temperature coefficient.
2. $RDS_{(ON)SPEC}$ is ON resistance mentioned in datasheet
3. T_{SPEC} is the temperature at which $R_{DS(ON)}$SPEC is specified

MOSFET $R_{DS(ON)}$ increases with temperature, exhibiting typical temperature coefficients that range from 0.35% /°C to 0.5% /°C. To calculate RDS_{ON}(HOT) we need to consider temperature co-efficient from the graph in datasheet. There are two mosfets in one half bridge Q1 and Q2. First I will calculate for Q1 then for Q2.

For Q1:

To find the Temperature Coefficient for $RDS_{ON(HOT)}$ we need to look at the following graph from datasheet.

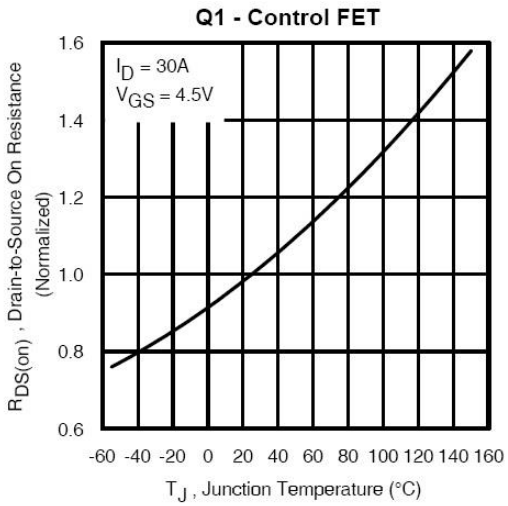


Figure 4.14: RDS_{ON} Temperature Co-efficient for Q1

From the above graph the temperature co-efficient is .5% /ºC. From datasheet $R_{DS(ON)}$SPEC is 4.60mΩ. The range of $T_{J(HOT)}$ mentioned in data sheet is from -55 to 150 ºC. I choose for worst case I select $T_{J(HOT)}$ is 100 ºC. Tspec from datasheet is 70 ºC.

By putting all these values in Eq C we get:

$RDS_{(ON)HOT}$ = 4.60mΩ [1+ .005 x(100 - 70)]

$RDS_{(ON)HOT}$ = 4.60mΩ [1+ .15]

$RDS_{(ON)HOT}$ = 4.60mΩ [1.15]

$RDS_{(ON)HOT}$ = 5.29 mΩ

Max Current for Each driver is 9Amps. By putting $RDS_{(ON)HOT}$ and I_D in Eq B

$PD_{Resistive}= 9^2$ x 5.29 mΩ

$PD_{Resistive}$= 428.9 mW

$PD_{Switching}$ is neglible so total Power dissipation by Q1 is 428.9 mW. Next step is to multiply the Total power dissipation with Thermal Resistance (Θ_{JA}). From datasheet Θ_{JA} is 34 ^{o}C/W.

$T_{rise} = PD_{Total} \times \Theta_{JA}$

$T_{rise} = 428.9mW \times 34\ ^{o}C/W$

$T_{rise(Q1)} = 14.582\ ^{o}C$

Figure 4.15 shows RDS_{ON} Temperature Co-efficient for Q2.

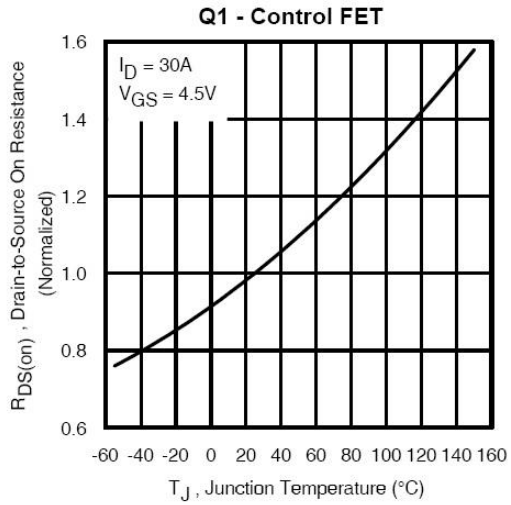


Figure 4.15: RDS_{ON} Temperature Co-efficient for Q2

For Q2 temperature co efficient is .35% and by repeating same above process (Because $R_{DS(ON)}$SPEC of Q2 is different from Q1) for Q2 the **$T_{rise(Q2)}$** is 4.4125 ^{o}C. Each pwm output channel contributes to rise in temperature with almost 14.582 ^{o}C. Total Temperature rise by the all high power output channels is 58.328 ^{o}C.

4.3 Schematics:

Schematics of the 1C601 Power Driver board were designed in Alitum PCB software. Schematics are design in Top to down hierarchy approach.

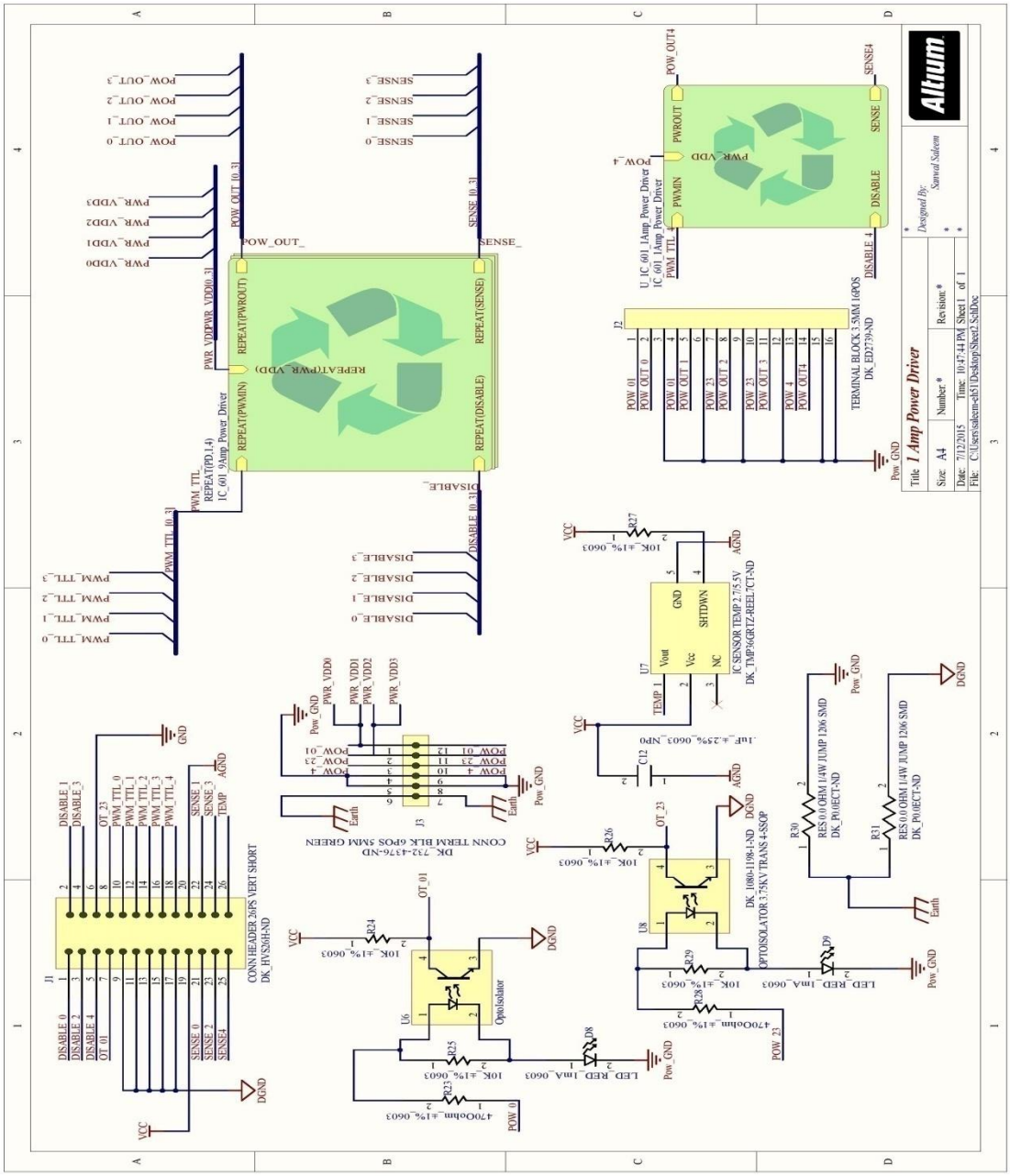


Figure 4.16: Top Schematic Sheet of 1C601 Power Driver Board

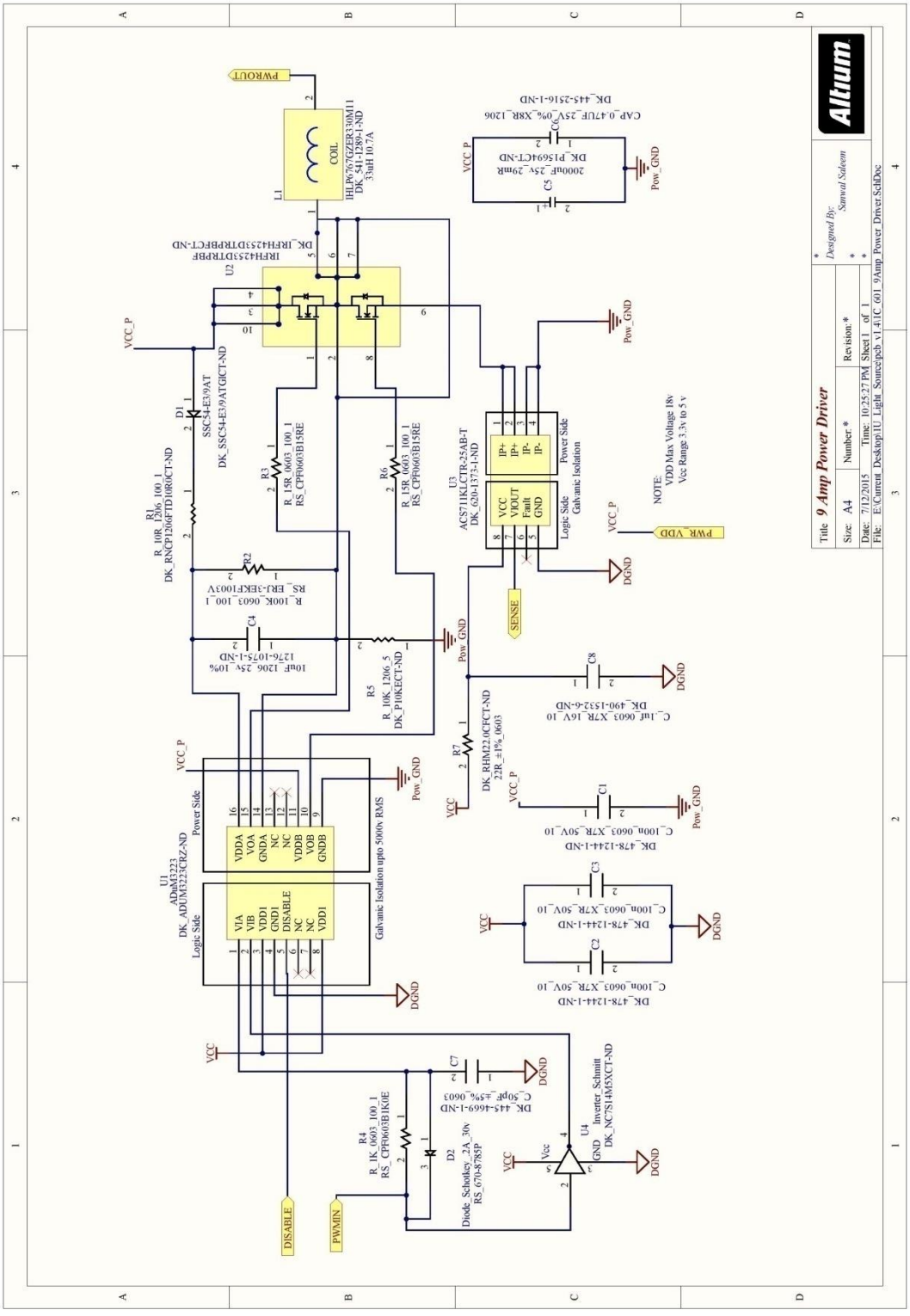


Figure 4.17: 1C601 9Amp Power Driver

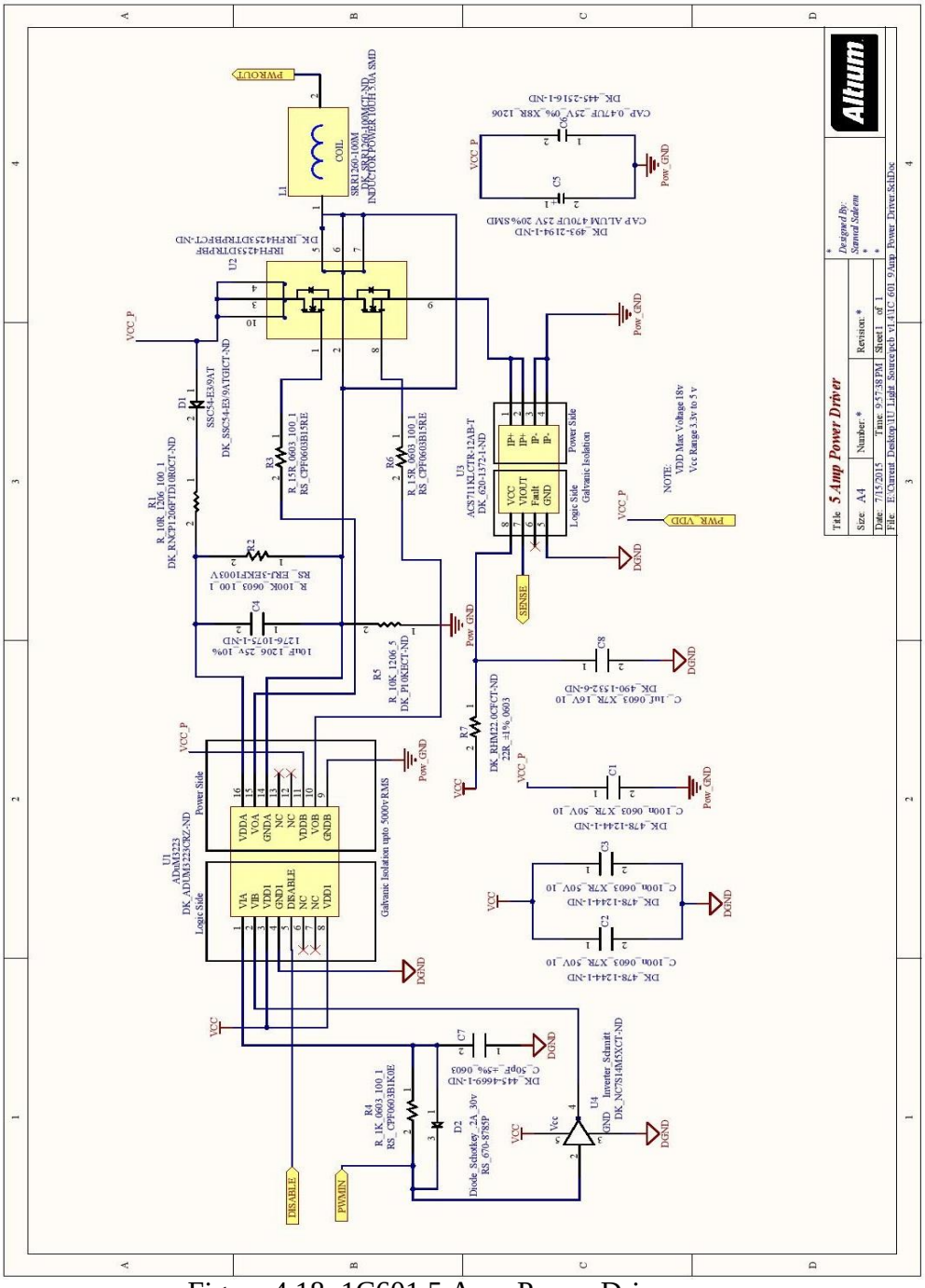


Figure 4.18: 1C601 5 Amp Power Driver

4.3 PCB Layout:

PCB layout for 1C601 Power Driver board is of four layers. Dimension of the PCB is 100mm x 100mm.

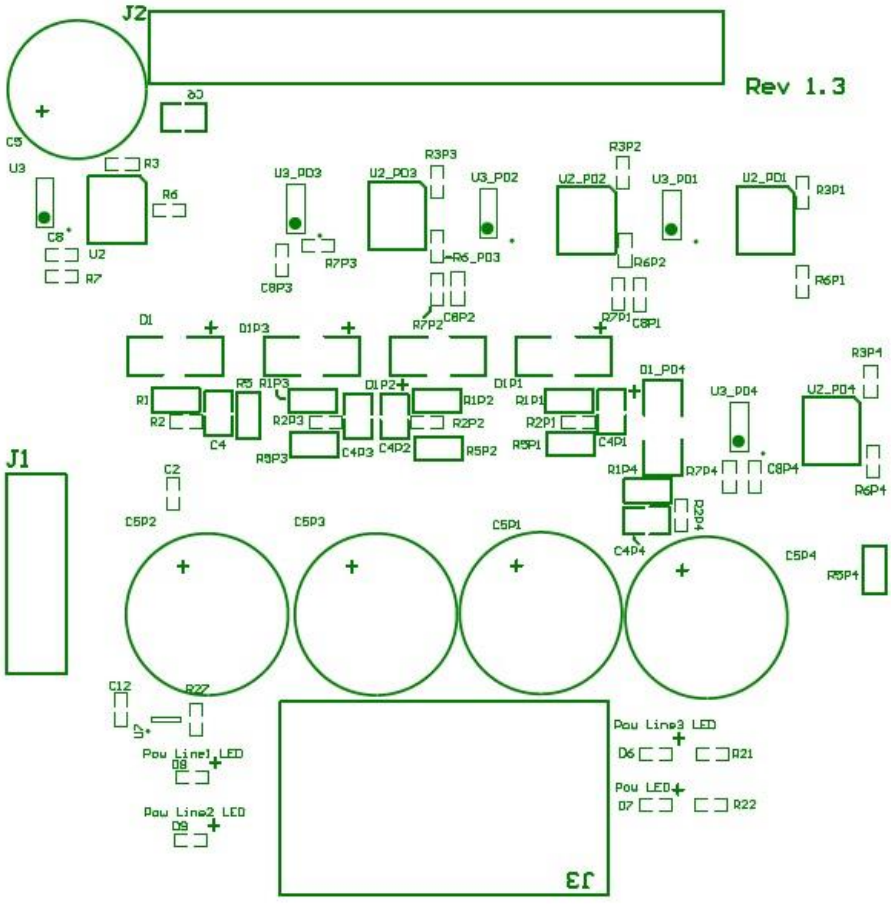


Figure 4.19: Top Over Lay of 1C601 Power Driver Board

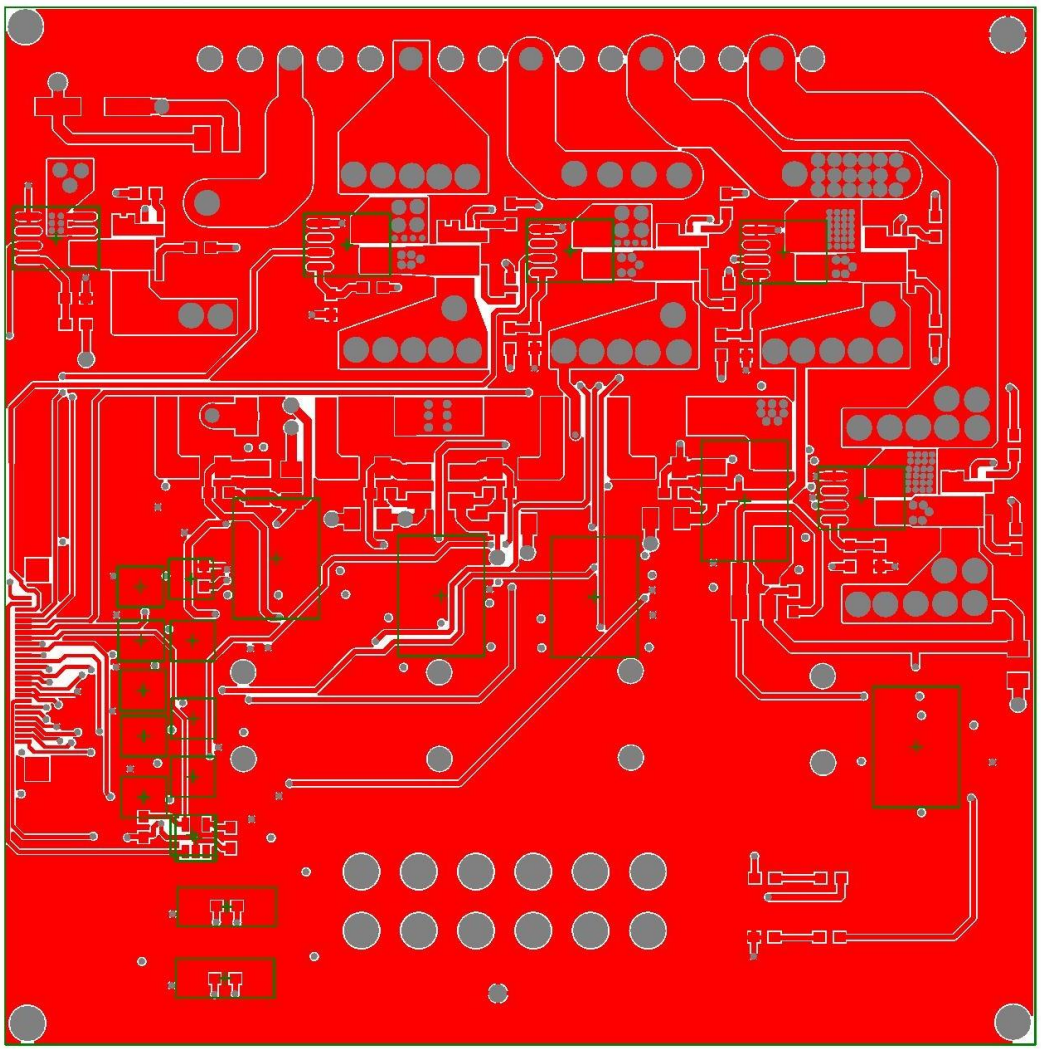


Figure 4.20: Top Layer of 1C601 Power Driver Board

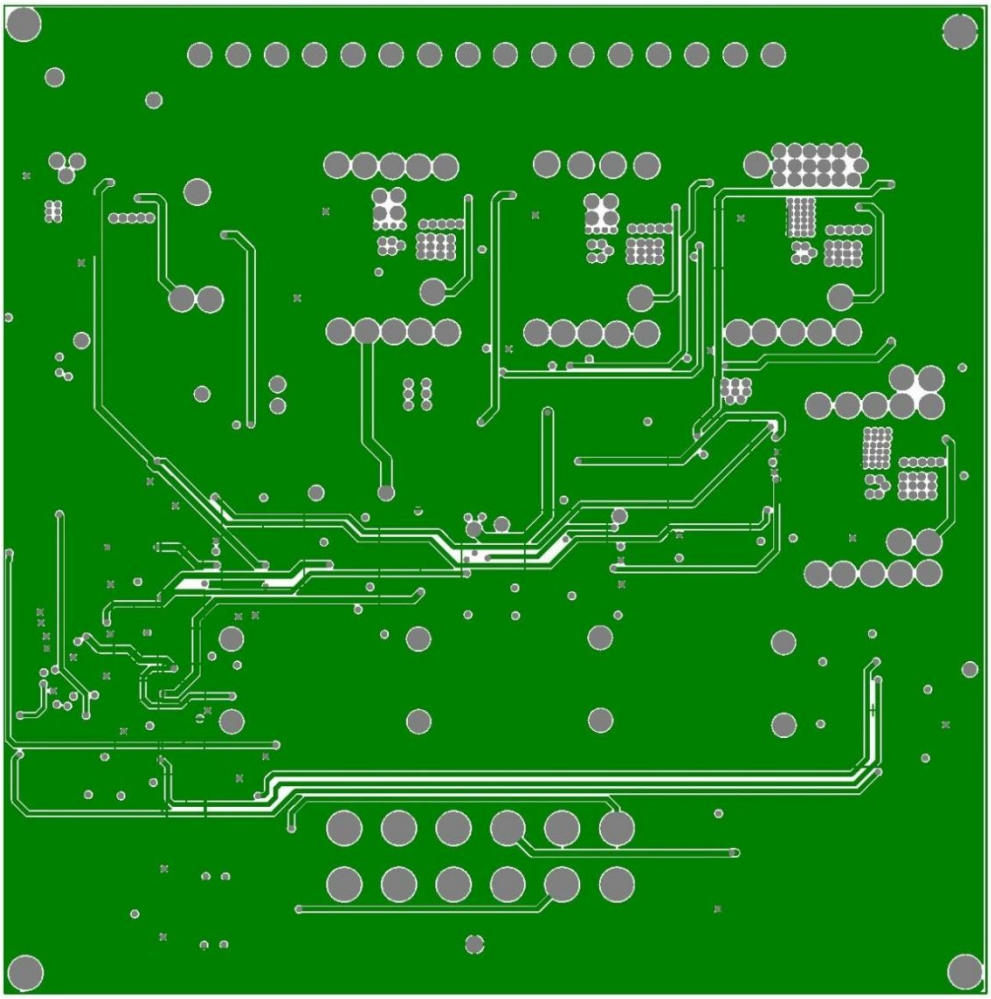


Figure 4.21: Mid Layer1 of 1C601 Power Driver Board

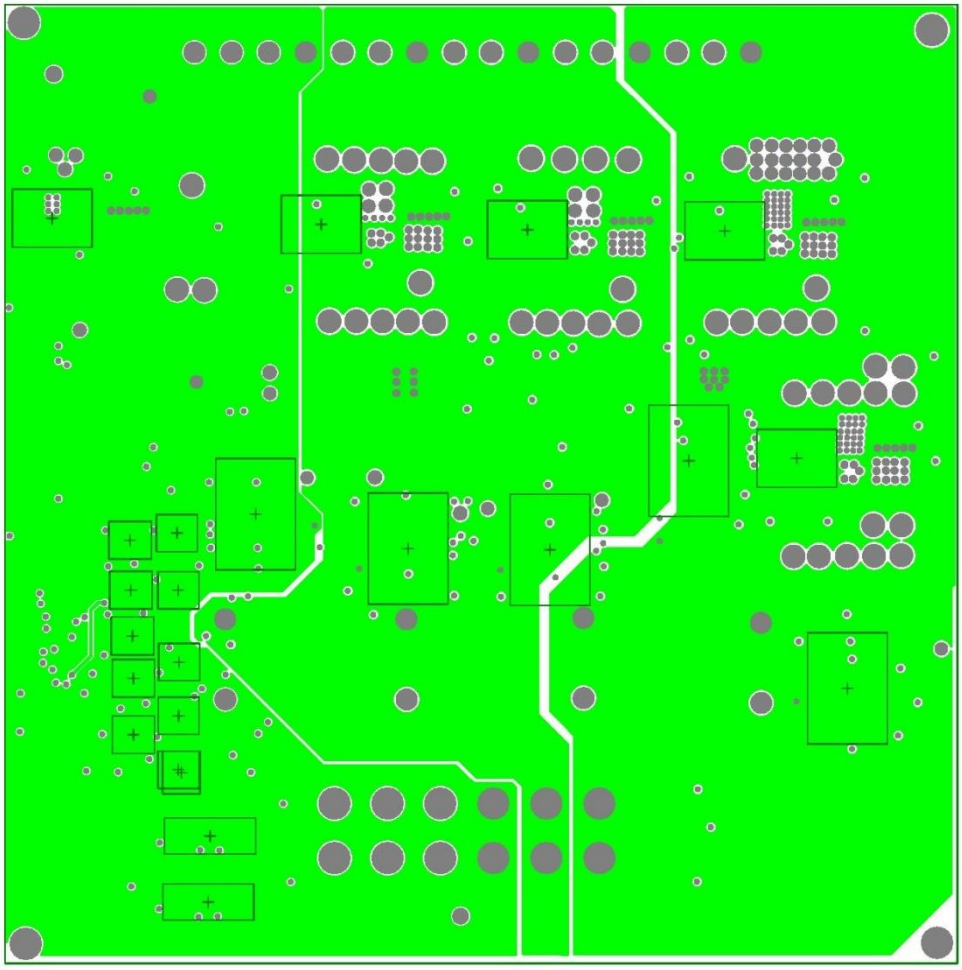


Figure 4.22: Mid Layer 2 of 1C601 Power Driver Board

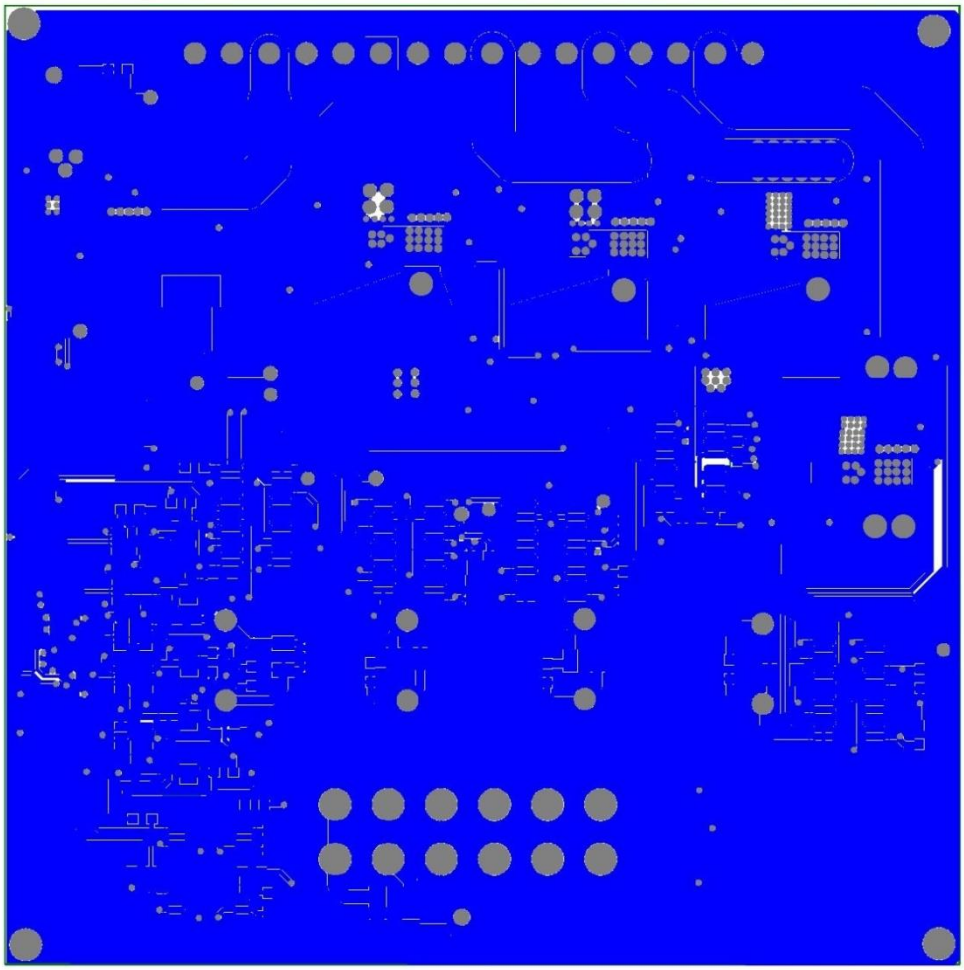


Figure 4.23: Bottom Layer of 1C601 Power Driver Board

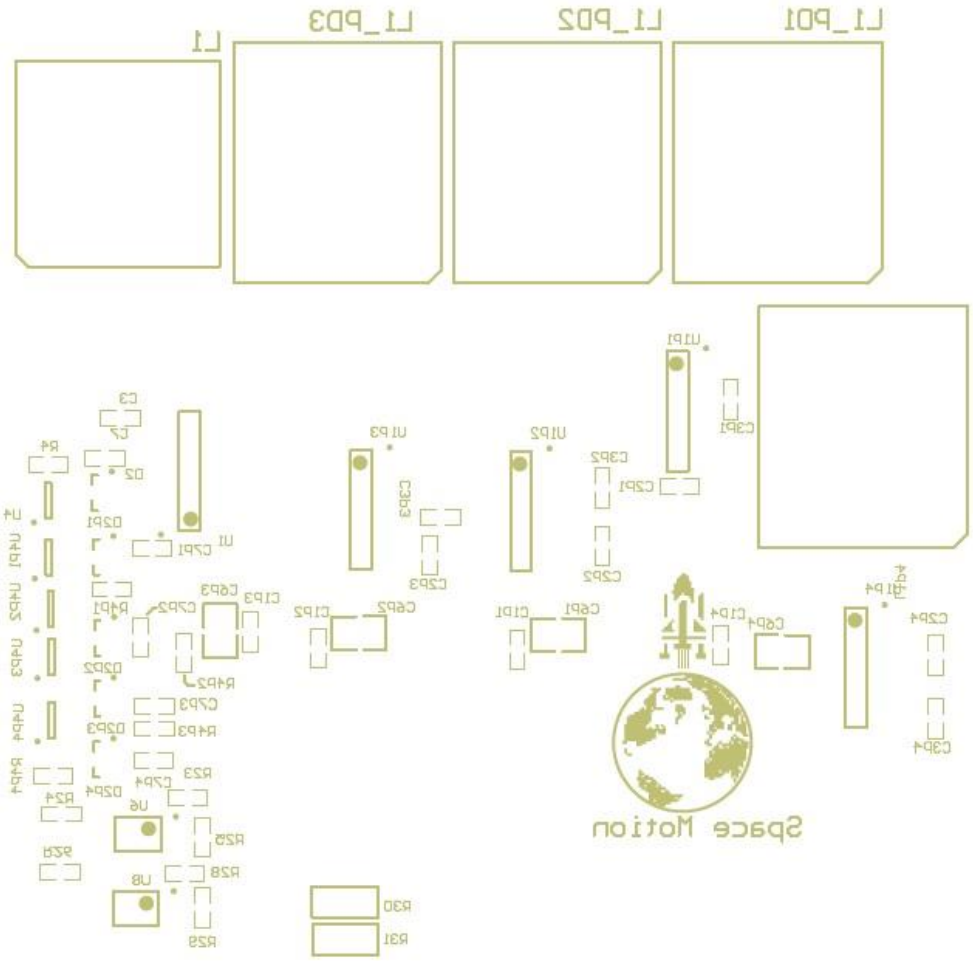


Figure 4.24: Bottom Overlay of 1C601 Power Driver Board

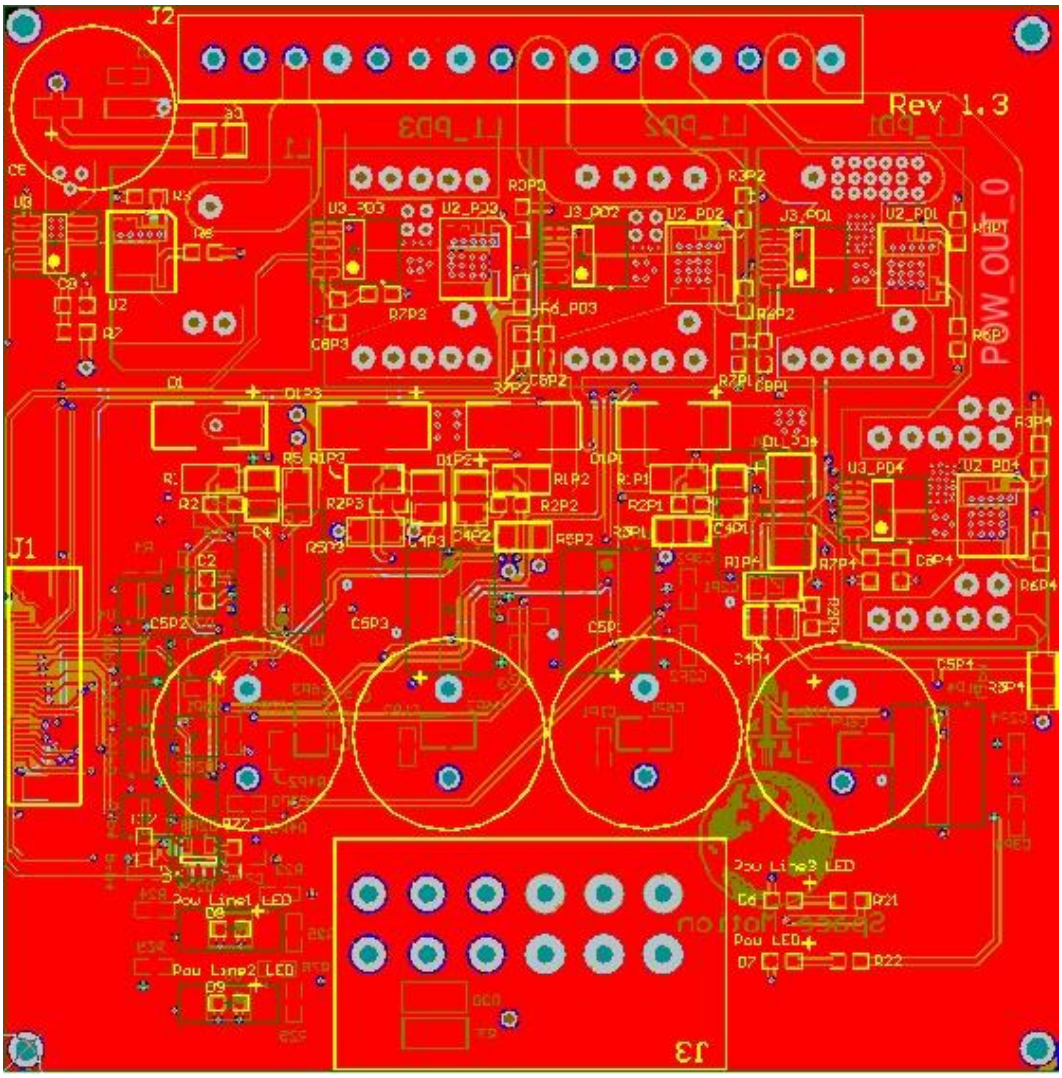


Figure 4.25: ALL PCB Layers of 1C601 Power Driver Board

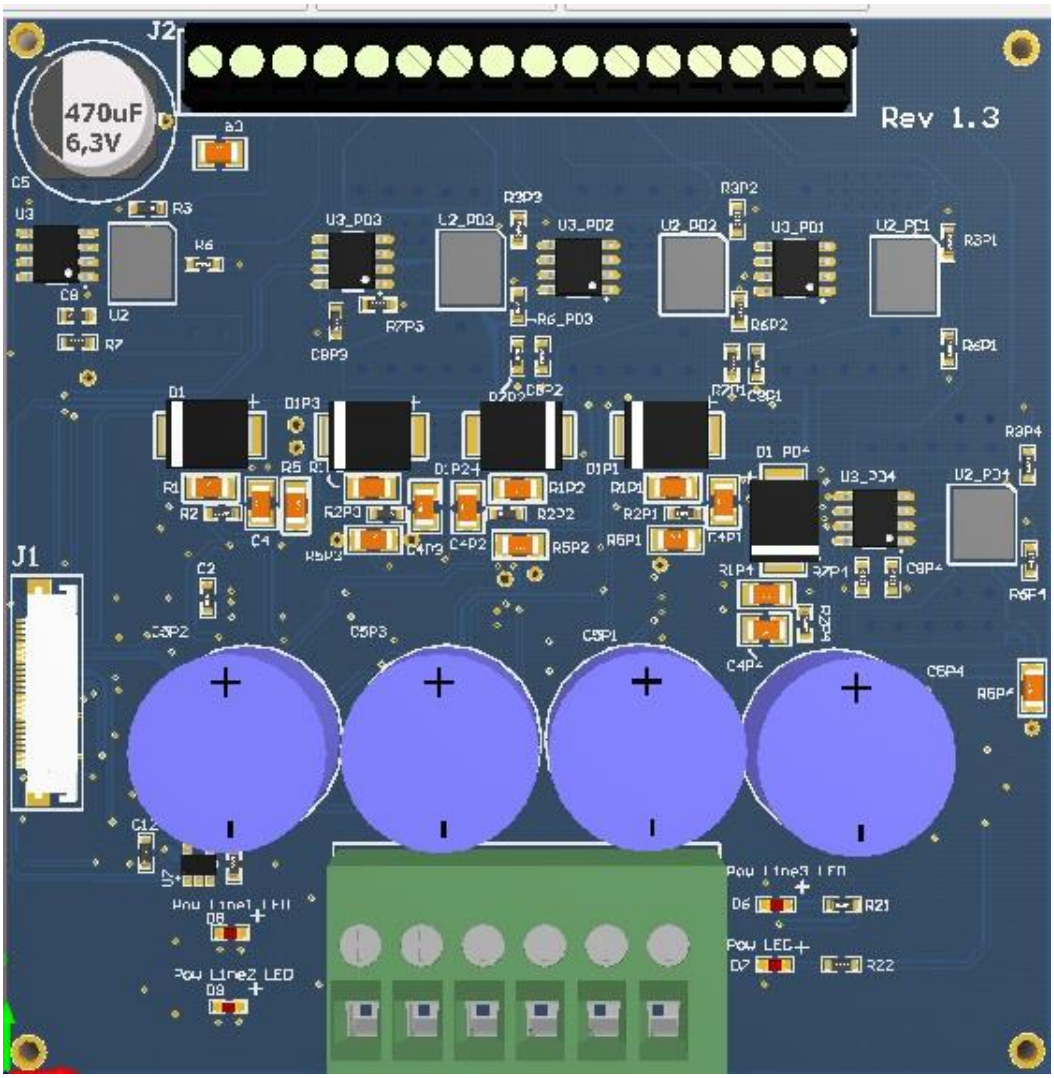


Figure 4.26: 3D Model TOP side of 1C601 Power Driver Board

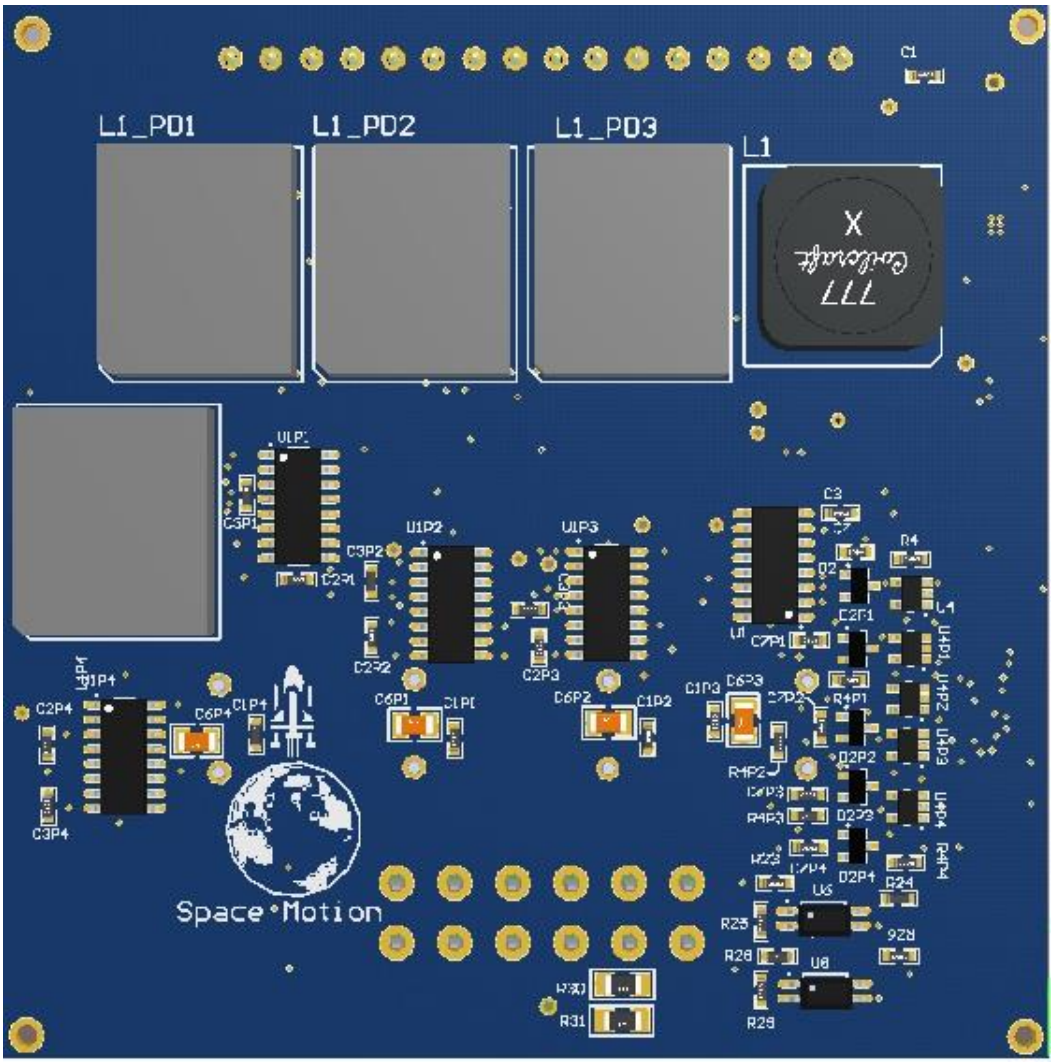


Figure 4.27: 3D Model Bottom side of 1C601 Power Driver Board

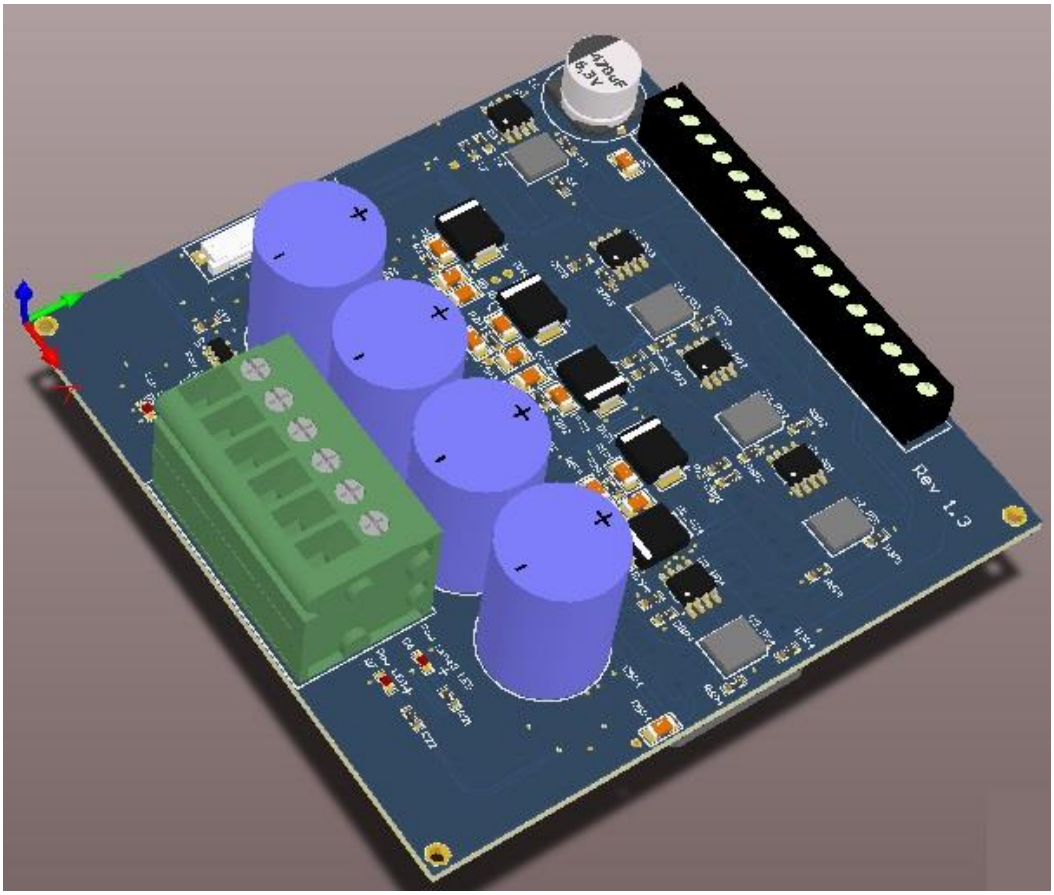


Figure 4.28: 3D Model Side view of 1C601 Power Driver Board

CHAPTER V: Filter Board

Filter board in this project is intended to filter main power input lines from Electromagnetic interference. Another name of EMI filter is RFI (Radio Frequency Interference). EMI filters are basically passive components and help to suppress the interference found on power line. Electromagnetic Interference either natural or artificial is not acceptable for electronics. This interference result in degradation of electronic devices.

Major sources of EMI generations are AC motors, microprocessors, switching power supplies and electrical power lines. EMI filters works by adding high resistance to the high frequency content. EMI consists of parallel capacitors and series inductors which results in stopping the flow of high frequency signals.

EMI filter used in this project is PAN4820 by TDK-LAMBDA. Following are the features of the filter:

1. Max Voltage rating is 76v dc
2. Max Current rating is 20 A
3. Isolation resistance is 100 MΩ
4. PCB mount

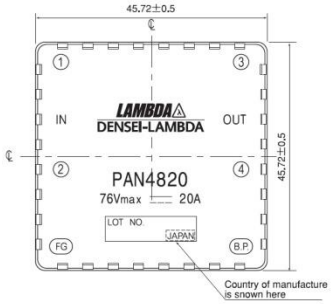


Figure 5.1: Pin Configuration of PAN4820 EMI Filter

Above Figure 5.1 shows the pin configuration of PAN4820.

Table 5.1: Pin names and description of PAN4820 EMI Filter

Pin Name	Pin Number	Description
IN1	1	Input terminal 1
IN2	2	Input terminal 2
Ground_In	FG	Pin connected to ground
Out1	BG	Output terminal 1
Out2	4	Output terminal 2
Ground_Out	5	Connected to the base plate of power module

5.1 Typical Application circuit:

Typical application circuit of the PAN4820 is as follow:

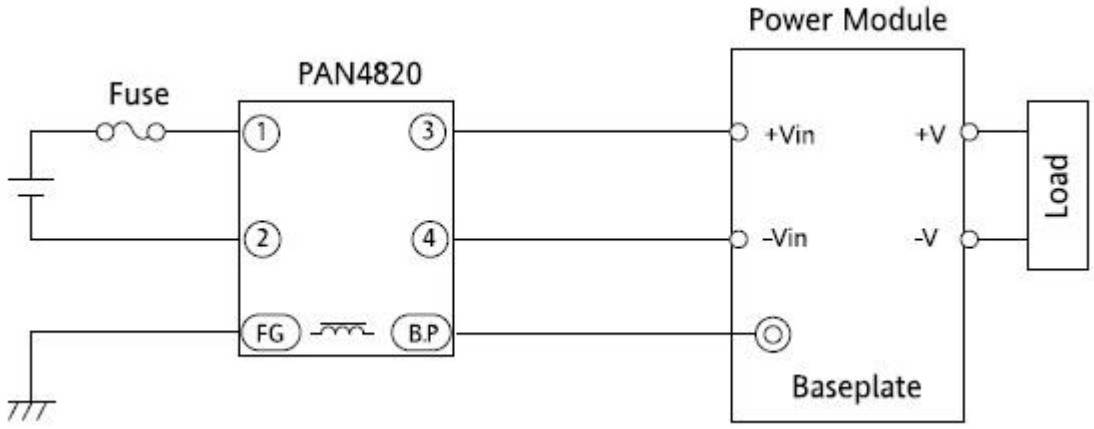


Figure 5.2: Typical application circuit of PAN4820

As clearly shown in above Figure 4.2 pin 1 and pin 2 are acting as input terminals and pin 3 and pin 4 acting as output terminals. Pin FG is connected to base plate (Earth Connection) and BP is connected to ground

Apart from advantages of high frequency component removal from the power line, there is one disadvantage of this filter, which is its max current rating 20Amps. Power driver have five channels and first four channels have 10 Amp capacity and the last one has 5 Amp. This EMI filter restricts the power distributed to the power channels 4.5 Amp. I

still used this EMI filter because we don't need more than 4.5 Amp on each channel and according to this requirement, this EMI filter is perfect.

5.2 Schematic:

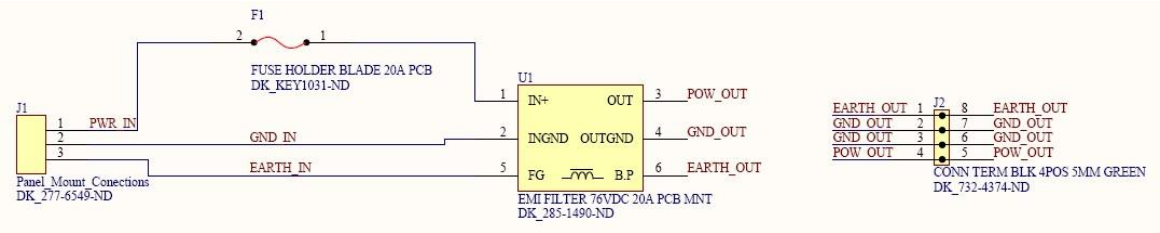


Figure 5.3: Schematic of 1C601 Filter Board

5.3 PCB Layout:

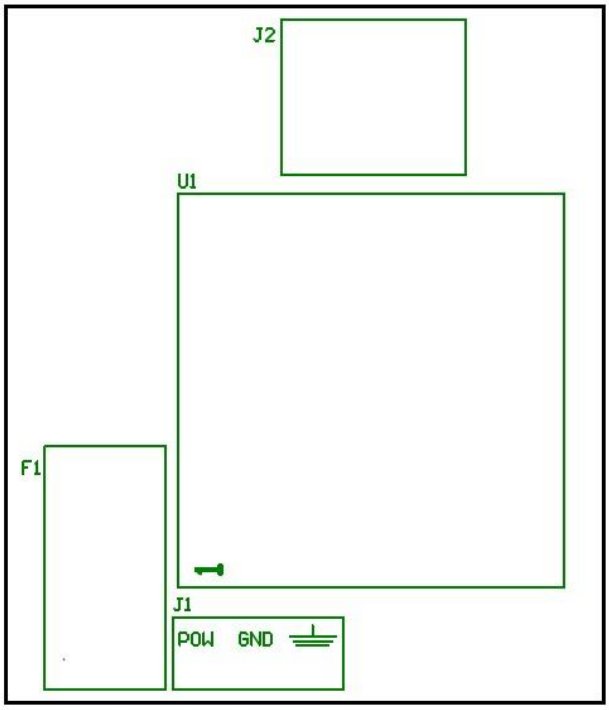


Figure 5.4: Top over Lay of 1C601 Filter Board

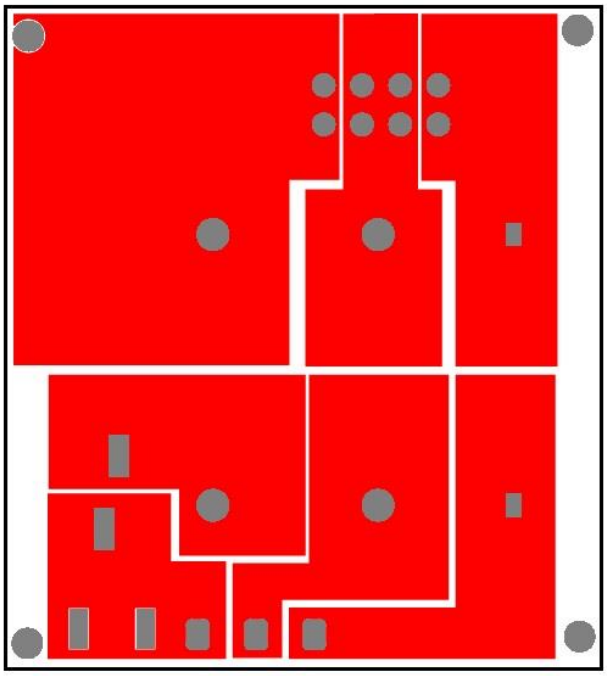


Figure 5.5: Top Layer of 1C601 Filter Board

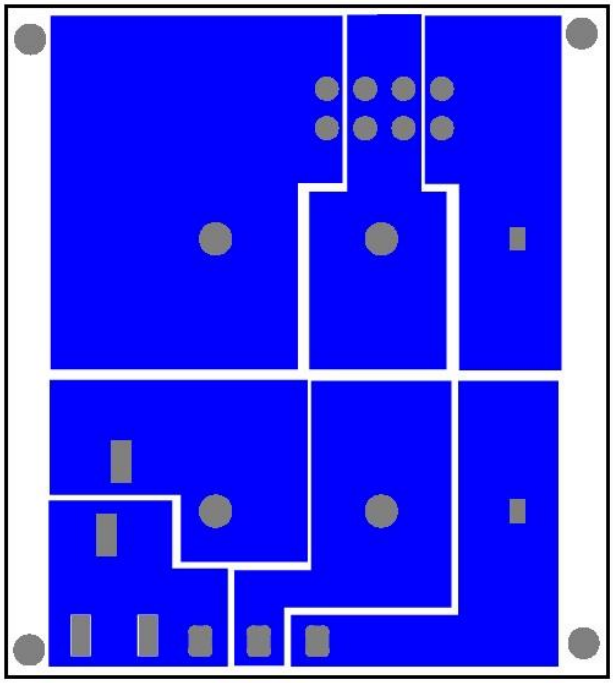


Figure 5.6: Bottom Layer of 1C601 Filter Board

Figure 5.7: 3D Model Top view of 1C601 Filter Board

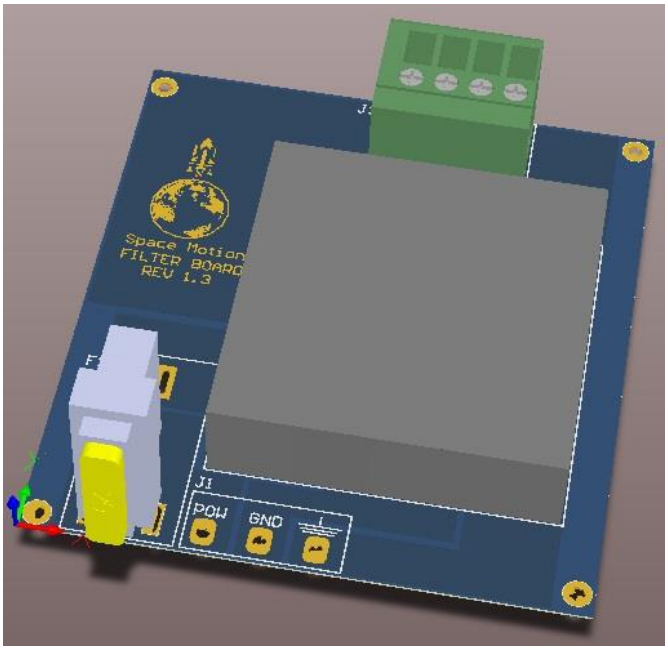


Figure 5.8: 3D Model Side view of 1C601 Filter Board

CHAPTER VI: Control Board

After Filter board next milestone was to design control board. The control board acts as a computation platform, communication platform and controlling element of power driver board.

Control board has following features:

1. MSP430F5438 microcontroller
2. USB debug port
3. DIP Switch to configure I2C address
4. Flat flex connector
5. Power convertor for logic supply
6. Analog input 8 bit port

6.1 MSP430F5438 Microcontroller:

The horse power of this project is MSP430F5438 which controls each and everything functionality of solar simulator.

MSP430F5438 has following features:

1. Low Supply Voltage Range: 3.6 V Down to 1.8 V.
2. 16-Bit RISC Architecture.
3. Fully Integrated LDO With Programmable Regulated Core Supply Voltage.
4. 16-Bit Timer TA0, Timer_A With Five Capture/Compare Registers.
5. 16-Bit Timer TA1, Timer_A With Three Capture/Compare Registers.

6. 16-Bit Timer TB0, Timer_B With Seven Capture/Compare Shadow Registers.
7. UART Communication.
8. I2C Communication.
9. 12-Bit Analog-to-Digital Converter (ADC).
10. 14 External ADC Channels, 2 ADC Internal Channels.

The MSP430F5438 microcontroller consists of different sets of peripherals targeted for various applications. The architecture is combined with large number of low power modes. The MSP430F5438 has 16 bit RISC CPU, constant generators and 16 bit registers that contribute to maximum code efficiency. It has digitally controlled oscillator (DCO) which allows the device to wake up from low-power modes (sleep modes) to active mode in about 3.5 μs.

The MSP430F5438A microcontroller has three 16-bit timers, a high-performance 12-bit analog-to-digital converter (ADC), up to four universal serial communication interfaces (USCIs), a hardware multiplier, DMA, a real-time clock module with alarm capabilities, and up to 87 I/O pins.

Folowing is the Figure 6.1 shows functional diagram of MSP430F5438:

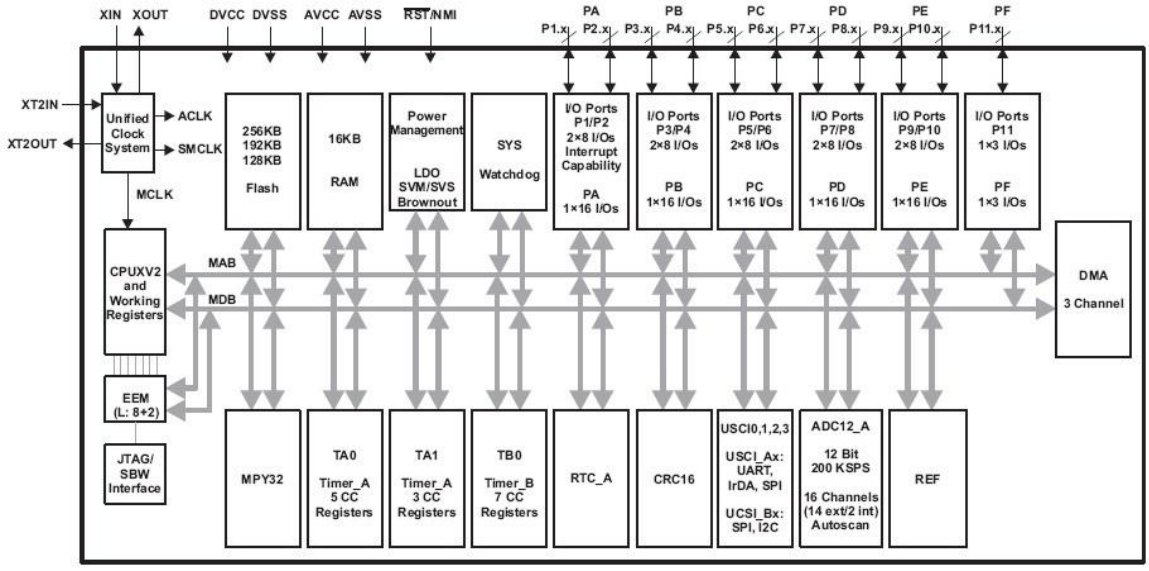


Figure 6.1: Functional diagram of MSP430F5438

MSP430F5438 is available in many PCB packages but I used 100 Pin QFN package. QFN package is easy to solder.

Figure 6.2 shows the pin names and pin configuration of MSP430F5438:

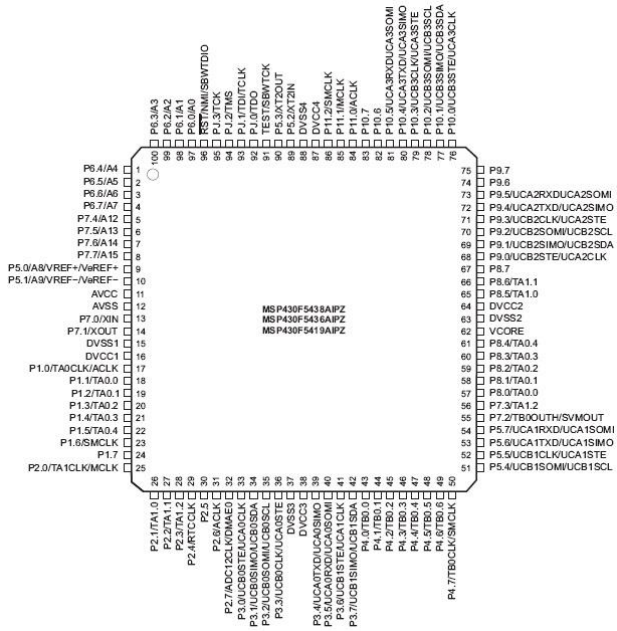


Figure 6.2: Pins Configuration of MSP430F5438

MSP430F5438 in this project provides following signals to different modules:

1. Logic level pwm to the power driver board.
2. Enable signals to the power board.
3. Take current sensor output, measure and convert into proper current value.
4. Take Analog voltage from temperature sensor on power driver board and convert it into digital temperature value
5. Continuous monitor power failure section outputs from power driver board
6. Take 7 Bit digital data from external dip switch for setting I2C address.
7. Provide logic supply to power board logic side components.
8. Status LED.

6.1.1 Typical application circuit design:

Typical application circuit design involves basic components attached to the msp430 to make it up and run. Basic circuitry includes crystal oscillators, resistors, jtag connector, Reset button and capacitors. Although msp430 provides internal oscillator (VCO) for clock but internal oscillator is very prune to environment effect (like temperature) and is not very stable. Our application requirement to have stable clock so that there will be no jitter in pwm signal.

External 4MHz crystal oscillator is attached to the pins **P5.2** and **P5.3** by nets **X1_1** and **X1_2**. Another low frequency external crystal oscillator of 32.768 KHz to the pins **P7.0** and **P7.1** by nets X1_1 and X1_2. **RST** reset pin is pull up by the 47K Pull up. **JTAG** is connected to msp430 by the pins **RST**, **TDI**, **TDO**, **TMS**, **TCK**, **TEST** and **GND** by connector **J2**. AVCC and DVCC are connected to the logic power supply with decoupling caps in parallel. AGND and DGND are analog ground and digital ground. For

making PCB EMC compliant **AGND** and **DGND** are connected to each other only at one point in whole circuit by two 0Ω resistors **R2** and **R3**.

Figure 6.3 shows typical application circuit for the MSP430F5438.

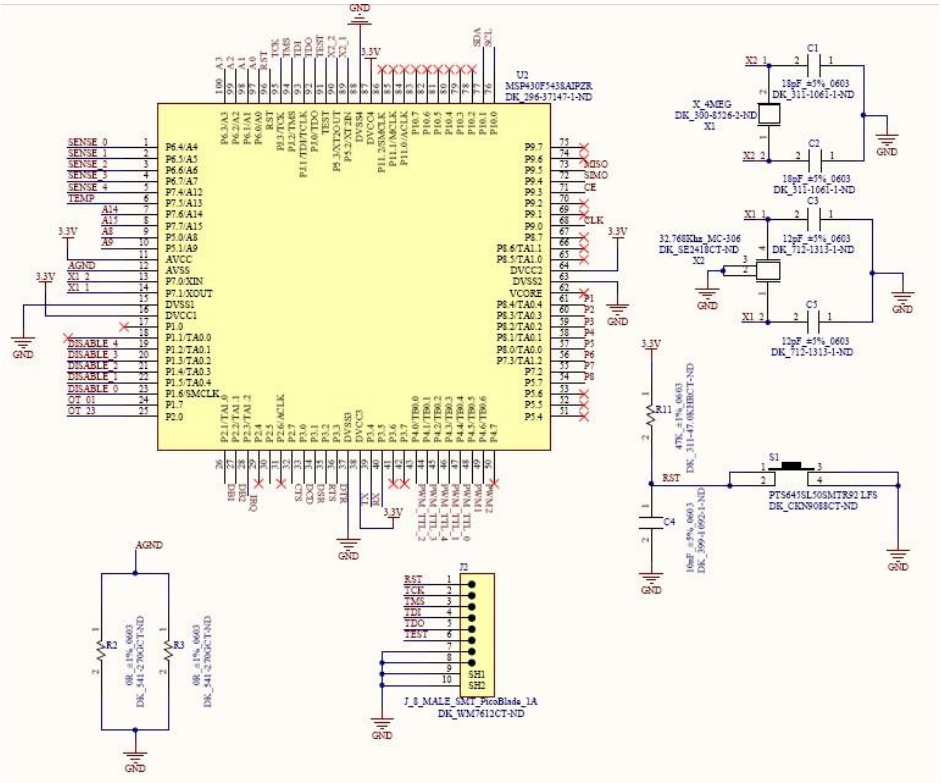


Figure 6.3: Typical Application circuit of MSP430F5438

6.2 USB Debugging:

Control board has one usb2.0 debug port. Purpose of debug port in this project is to debug the system without having core knowledge of the system. Tester can connect his laptop and check all the process variables of the system and also able to control the system.

MSP430F5438 doesn't have usb port because of that I used FTDI chip which converts the uart to usb and vice versa. Uart port could be used as debug port but it's obsolete and no more present in new PC's.

The difference between I2C and USB in this system is that, For I2C core knowledge is required to communicate with the device because it's not very straightforward to send desired commands via I2C and it requires a lot of data manipulations and for usb its very straightforward and plug & play. Another reason which makes usb debug more easy to use is user will only type the command name and data if associated with the command and system reacts to it and give back the results. USB debug port is activated by the 8th bit of DIP switch which used for setting the I2C address externally.

6.2.1 FTDI chip Interface:

FTDI chip interface is simple and straightforward. Its Converts the UART TTL level signals into usb2.0 signals. It often called usb to UART Bridge. The driver for the FTDI chip is provided by the company. FTDI chip comes with factory pre programmed (No need of programming the chip, its plug and play).

Figure 5.10 shows interface of msp430 with FTDI chip.

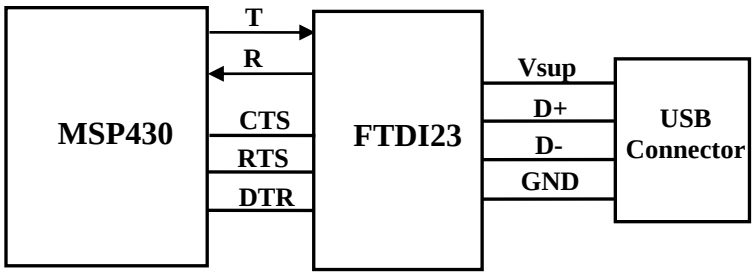


Figure 6.4: MSP430 interface with FTDI chip

6.2.2 FTDI Application circuit:

Figure 5.11 shows application circuit for FTDI used in this project.

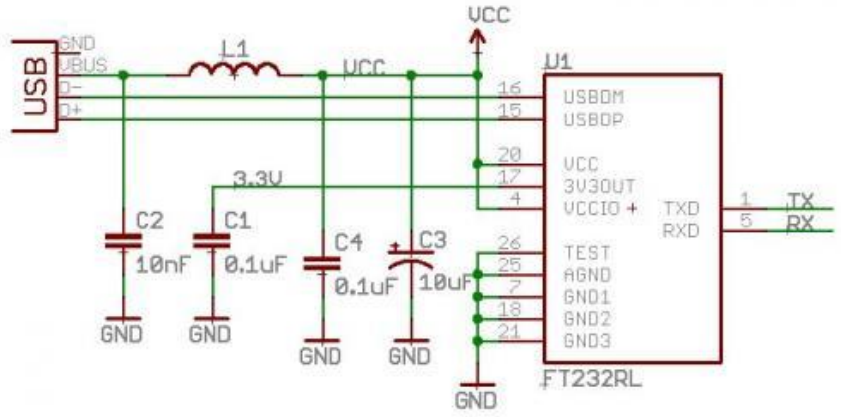


Figure 6.5: Application circuit for FTDI chip interface

6.3 DIP switch to configure I2C address:

Solar simulator is designed in such way that it achieved high modularity. For modularity the first requirement was to have communication which can support multiple slaves and for that I2C was chosen. Second requirement is the ability to set the I2C address of the system externally. Reason for setting address externally is that lets suppose user have five solar simulators and he wants to control it all together.

For simultaneous working of all five solar simulators it's required to have different I2C slave address. It's easier to set the I2C address using DIP switch externally than to set from UART or if its preprogrammed there can be problem that two simulators have same address. Seven bit I2C is used and it requires DIP switch of 7 Bit but in this project 8 bit dip switch is used. The 8th bit of DIP switch is used to turn ON and OFF debug mode.

Figure 6.6 shows basic interfacing of dip switch with MSP430. **P1** to **P8** are digital 3.3v output pins attach to the digital pins **P8.0** to **P8.4** and **P5.7**, **P7.2** and **P7.3**.

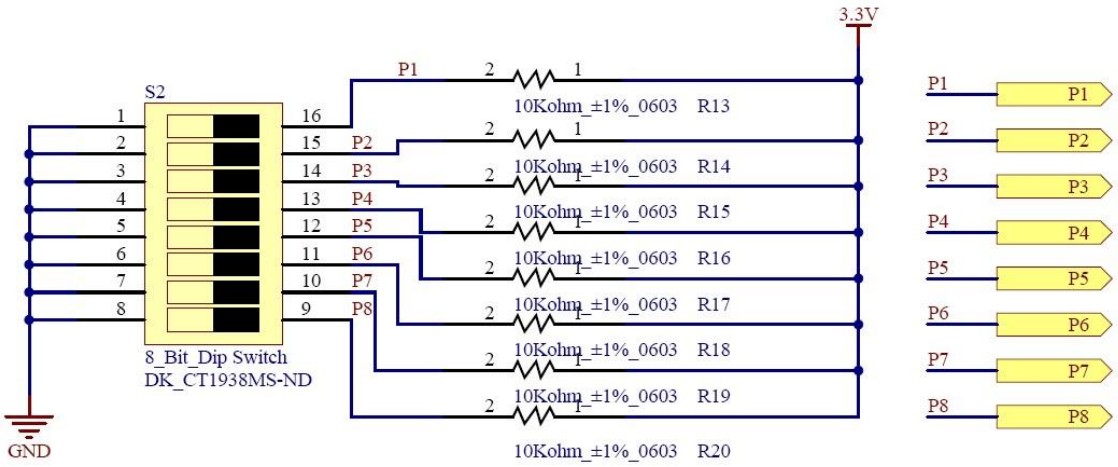


Figure 6.6: DIP switch interface with MSP430

As shown in above figure the pins of DIP switch are pulled up and P1 to P8 are its output. Output pins are connected to the digital ports of MSP430f5438. Switch debouncing of the DIP switch is done inside firmware.

6.3.1 Physical Access of DIP Switch:

DIP is mounted on the back panel of 1U Light source. User can easily access the DIP switch from the back panel.

6.4 Flat Flex Connecter:

Flat flex 26 pin connector is used to connect control board with power board. All the communication between control board and power board is done by this connector. The pitch of the connector is .5mm.

Figure 6.7 shows connections with flat flex connector.

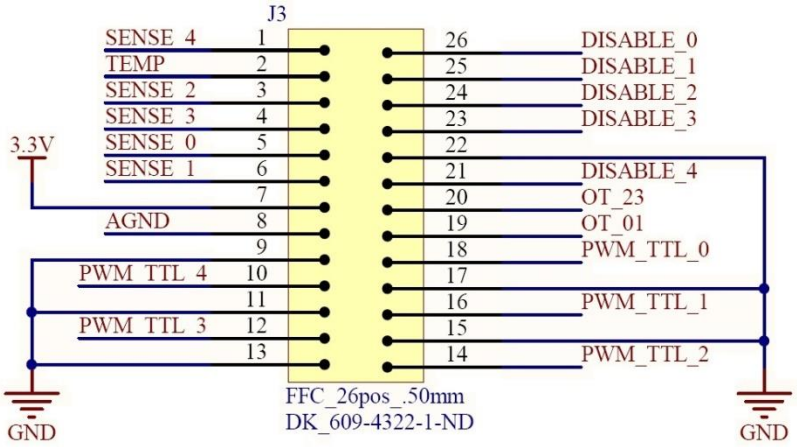


Figure 6.7: Connections with Flat Flex Connector

Figure 6.8 shows 3D model of flat flex connector.

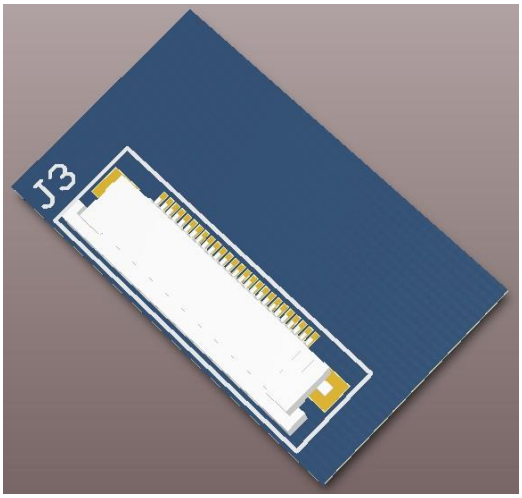


Figure 6.8: 3D Model of Flat Flex Connector

6.5 Power Convertor for Logic Supply:

There are two power supplies required for solar simulator one is main high voltage power supply and other is low voltage logic power supply. There is on board power convertor which converts 5v input supply to 3.3v and supply to all logic components on control

board and also on power board. Power convertor used in this project is TPS61200 by Texas Instruments. TPS61200.

TPS61200 has following features:

1. Automatic Transition between Boost Mode and Down Conversion Mode
2. Down Conversion Mode
3. Startup into Full Load at 0.5 V Input Voltage
4. Operating Input Voltage Range from 0.3 V to 5.5 V
5. Fixed and Adjustable Output Voltage Options from 1.8 V to 5.5 V
6. Over temperature Protection

TPS61200 is low voltage input boost convertor with down conversion mode. This convertor supports input from .3v to 5.5v. The boost convertor has fixed frequency. Output is programmable by choosing proper values of external resistors. It has shut down pin and by putting enable pin low the convertor goes into shutdown mode. In shutdown mode all internal circuitry including ULVO comparator turn OFF. The controller also uses input and output voltage feed forward. Changes of input and output voltage are monitored and immediately change the duty cycle in the modulator to achieve a fast response to those errors.

Figure 6.9 shows pin configuration of TPS61200.

Figure 6.9: PIN configuration of TPS61200

Table 6.1: Pin names and Pin description of TPS61200.

PIN Name	PIN Number	Description
EN	6	Enable input (High = enabled, Low = disabled). Do not leave floating.
Exposed Thermal Pad	---	Must be soldered to achieve appropriate power dissipation and mechanical reliability. Should be connected to PGND.
FB	10	Voltage feedback of adjustable versions, must be connected to VOUT at fixed output voltage versions
GND	9	Control / logic ground
PGND	4	Power ground
PS	8	Enable/disable Power Save mode (High = disabled, Low = enabled). Do not leave floating.
L	3	Connection for Inductor
UVLO	7	Under voltage lockout comparator input. Must be connected to VAUX if not used
V_{AUX}	1	Supply voltage for control stage
V_{in}	5	Boost converter input voltage
V_{OUT}	2	Boost converter output

6.5.1 Typical application circuit:

Figure 6.10 shows typical circuit of TPS61200.

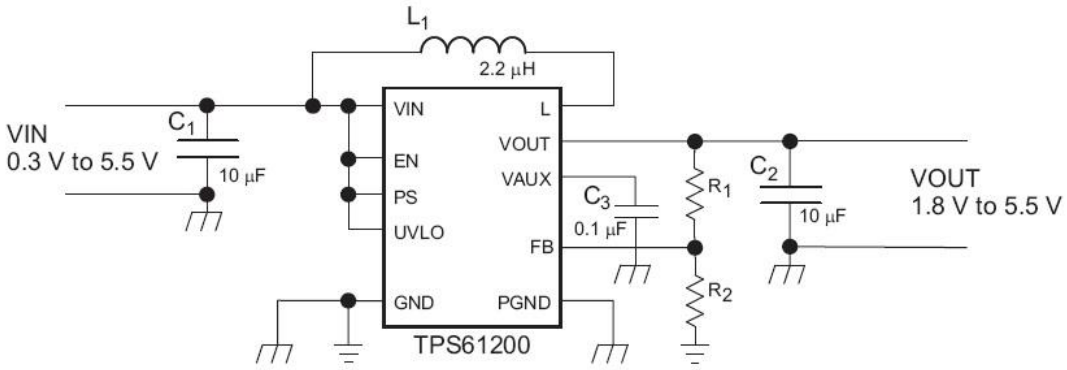


Figure 6.10: Application circuit of TPS61200

For properly configuring the fixed output voltage, the **FB** pin is used to sense the output voltage. FB pin should be connected to **V_{out}** so that output voltage can be measure. The resistor divider must be connected between **VOUT**, **FB** and **GND**. When the output

voltage is regulated properly, the typical value of the voltage at the FB pin is 500 mV. **R1** and **R2** control the output voltage.

6.5.2 Power Convertor Schematic

Figure 6.11 shows power convert schematic in Solar Simulator.

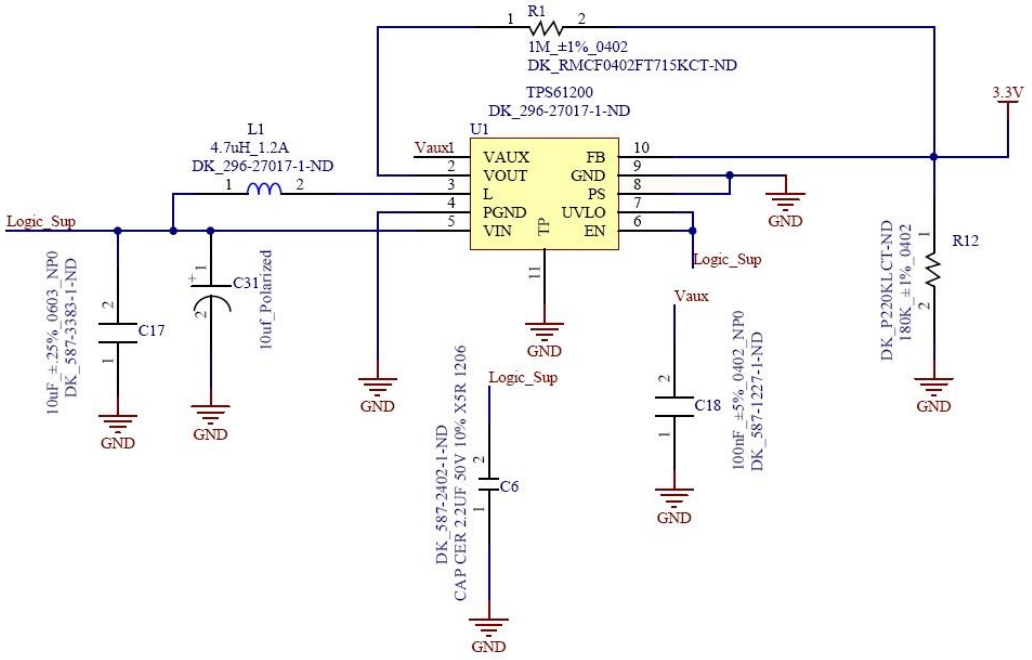


Figure 6.11: Power Convertor Schematic

The typical current into the **FB** pin is 0.01 µA, and the voltage across the resistor between **FB** and GND, **R15** is typically 500 mV. Based on those two values, the recommended value for **R15** should be lower than 500 kΩ, in order to set the divider current at 1 µA or higher. It is recommended to keep the value for this resistor in the range of 200 kΩ. The value of the resistor connected between VOUT and FB, **R1**, depending on the needed output voltage (VOUT), can be calculated by following equation:

$$R1 = R15 \text{ x} \left(\frac{V\text{out}}{\text{Vfb}} - 1 \right)$$

By putting V_{FB} 500mv **R15** 180k and **V_{out}** 3.3v (Desired output voltage) in above equation we get the value of **R1** 1Mohm. Capacitor C8 is connected to Vaux. This capacitor is used to maintain and filter the control supply voltage, which is chosen from the highest of VIN, VOUT, and L. It is charged during startup and before the main output VOUT is turned on. To ensure stable operation, using at least 0.1µF is recommended.

6.6 Analog Input 8 Bit port:

Control board has an extra connector which provides interface to ADC of MSP430F5438. Eight Analog input channels are available. This connector can be used to add more temperature sensors or other analog sensors.

Figure 6.12 shows the Analog Input Connector.

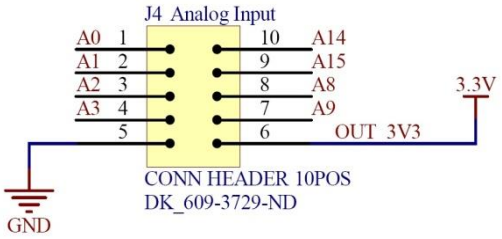


Figure 6.12: Analog Input Port

Figure 6.13 shows 3D model of Analog Input Connector.

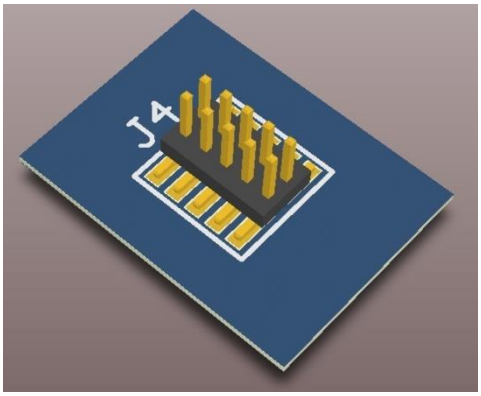


Figure 6.13: 3D model of analog input connector

6.7 Schematics:

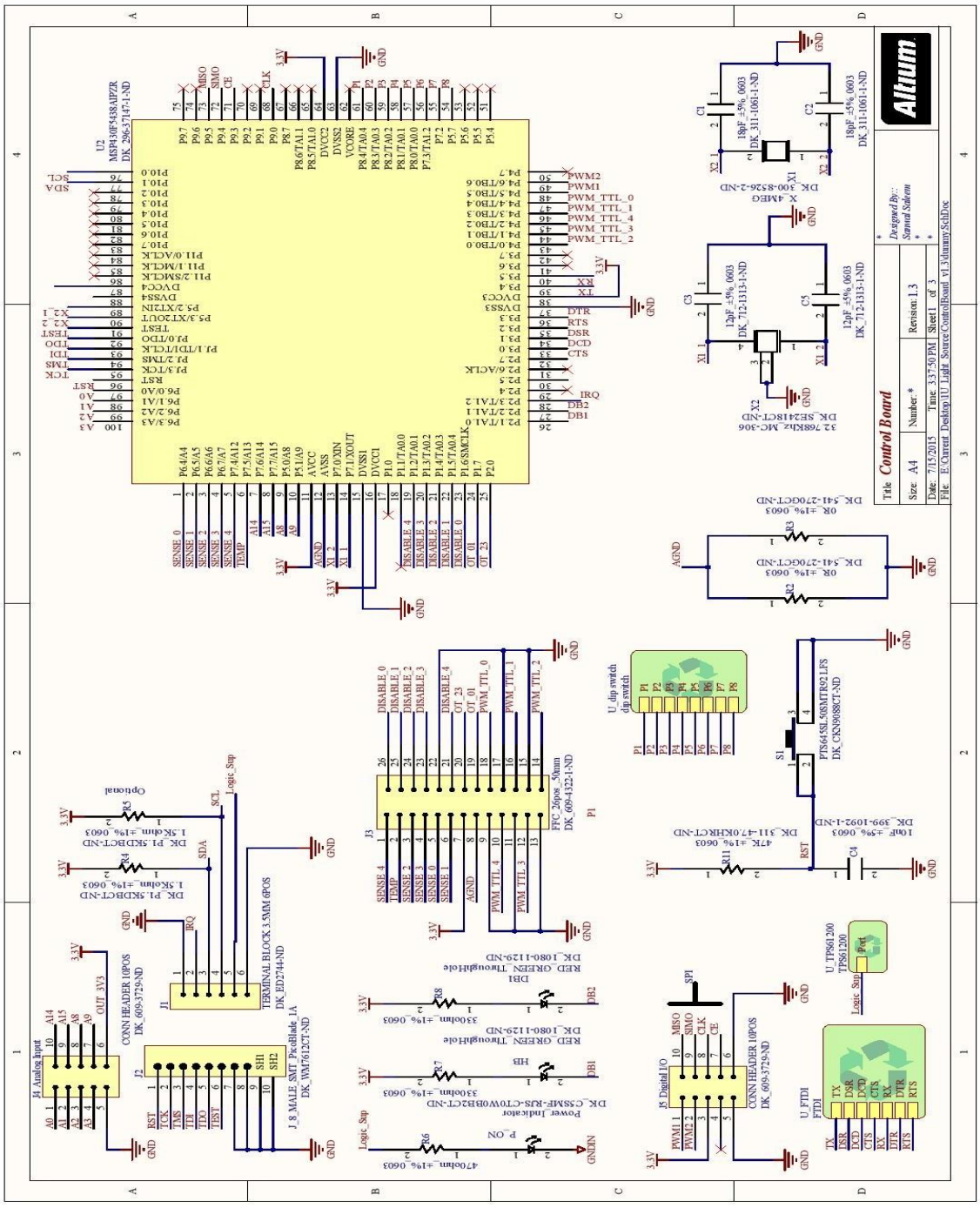


Figure 6.14: TOP Level 1C601 Control Board Schematic

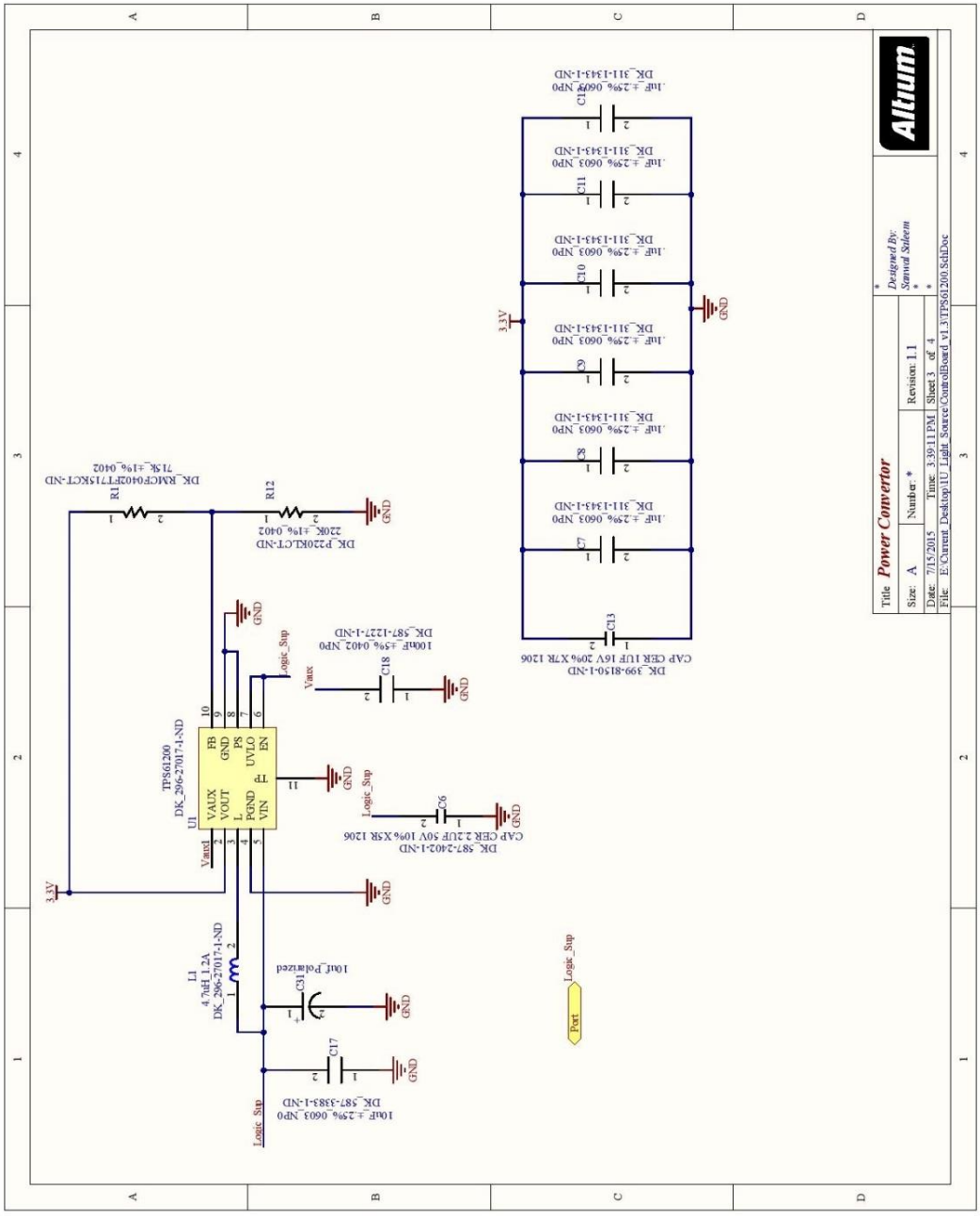


Figure 6.15: Schematic of Power Convertor TPS61200

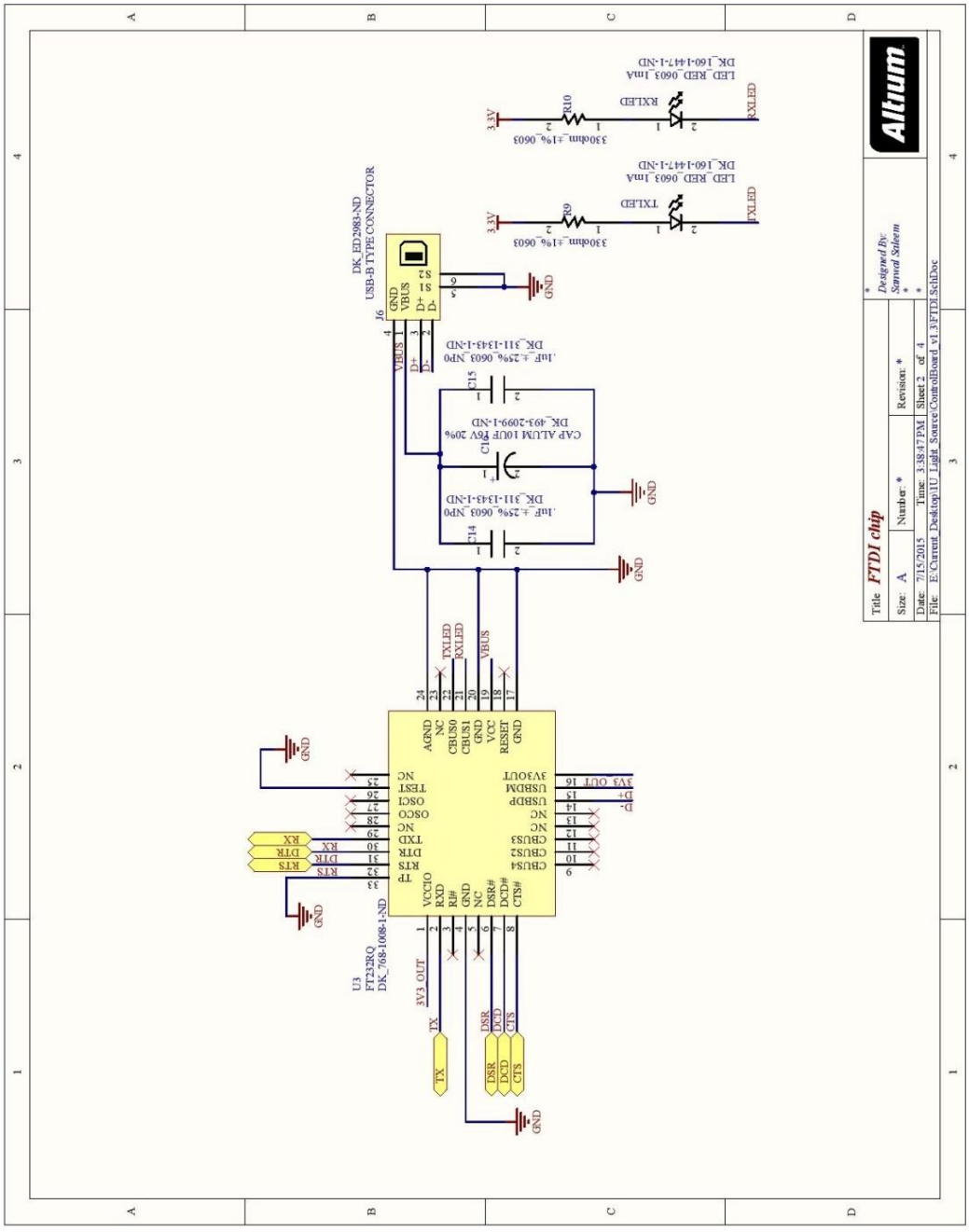


Figure 6.16: Schematic of FTDI Chip

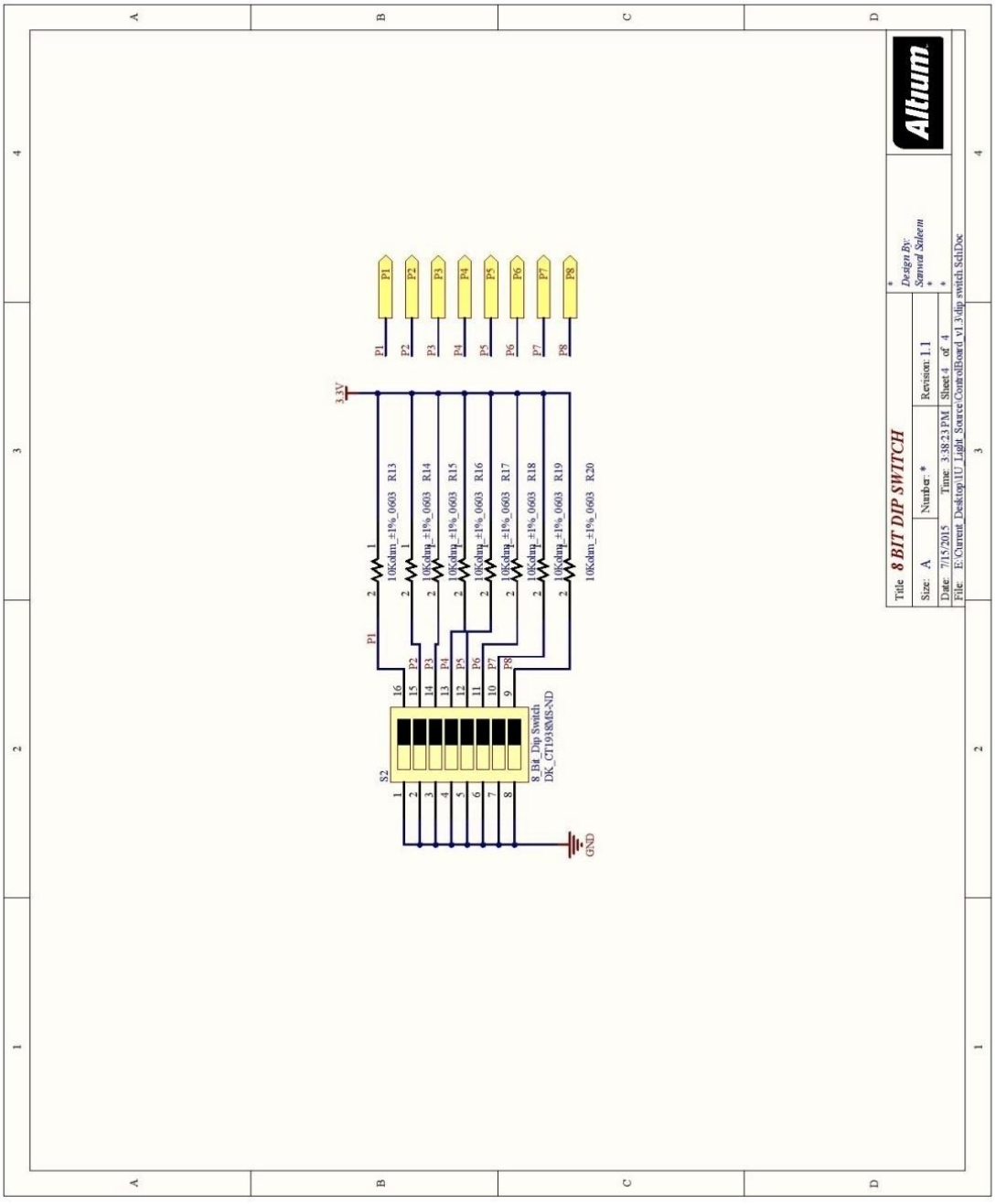


Figure 6.17: Schematic of 8 BIT DIP Switch

6.8 PCB Layout:

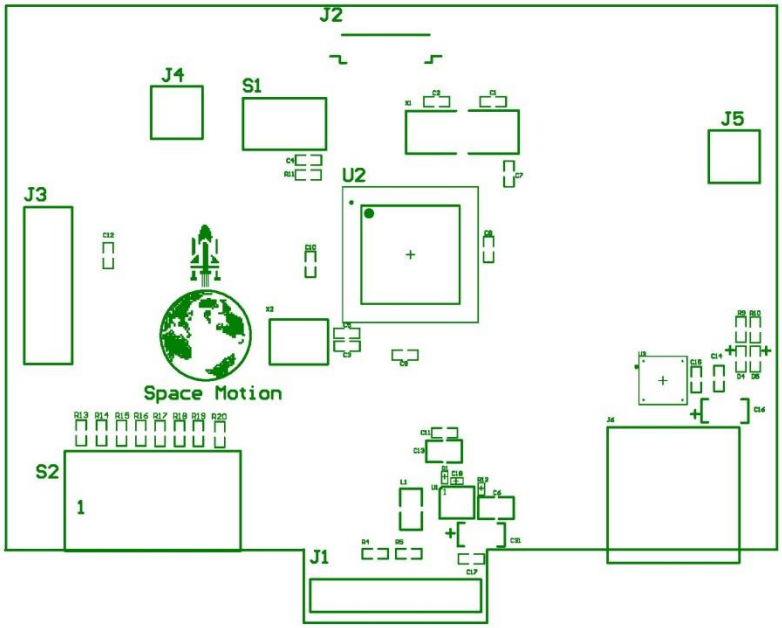


Figure 6.18: Top over Lay of 1C601 Control Board

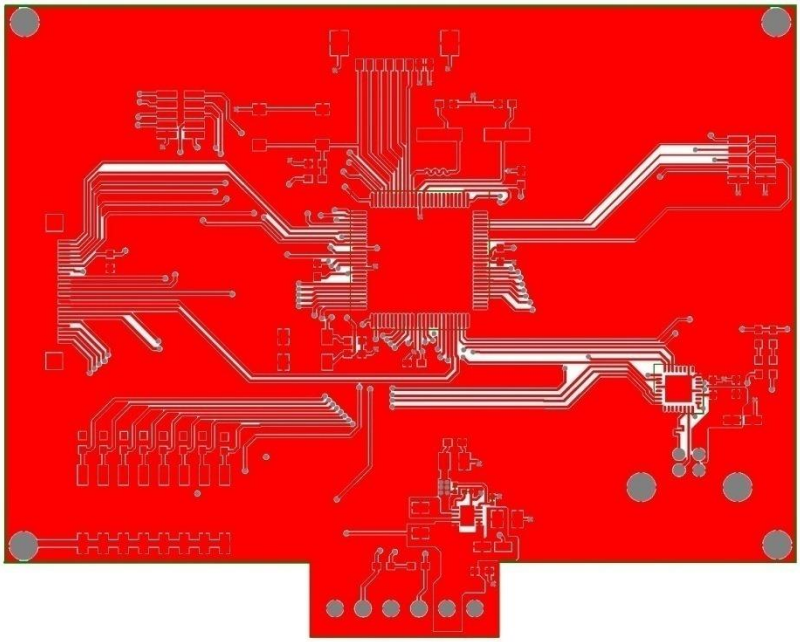


Figure 6.19: Top Layer of 1C601 Control Board

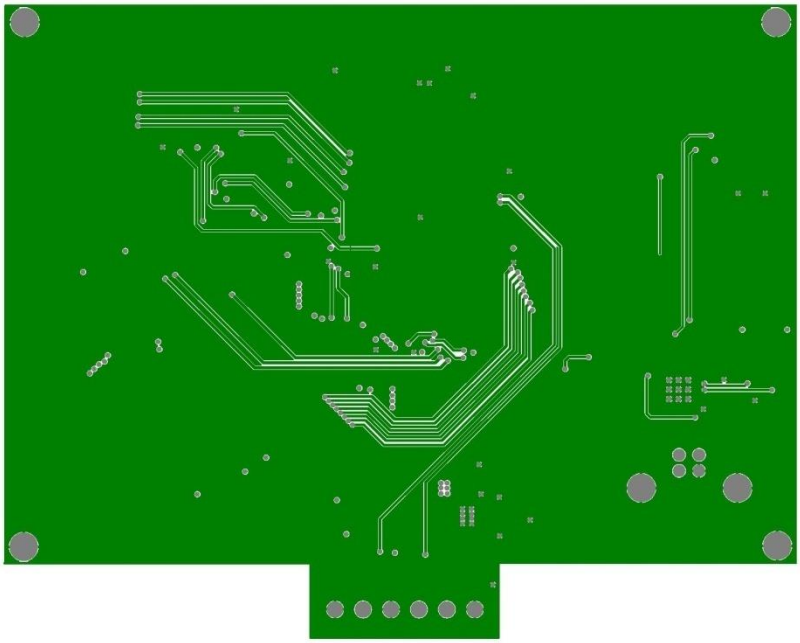


Figure 6.20: Mid1 Layer of 1C601 Control Board

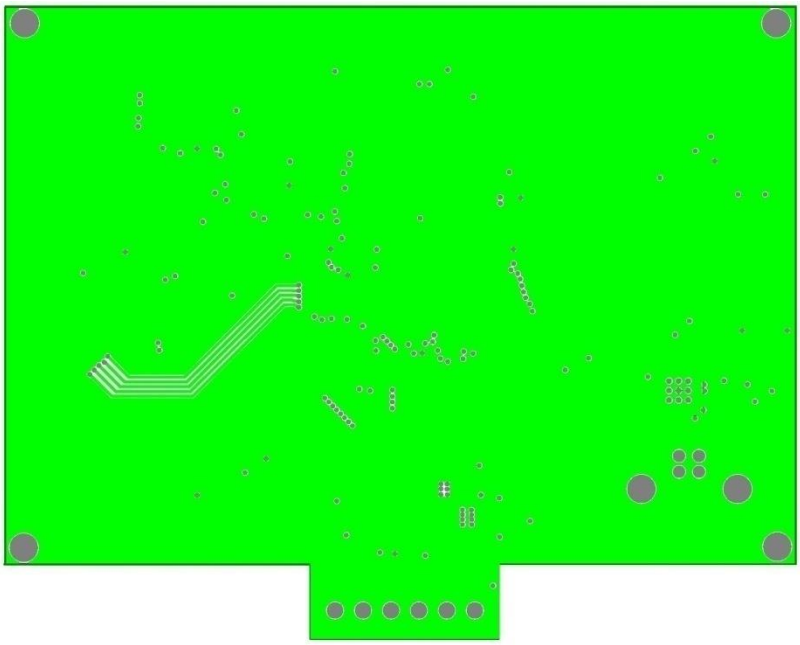


Figure 6.21: Mid2 Layer of 1C601 Control Board

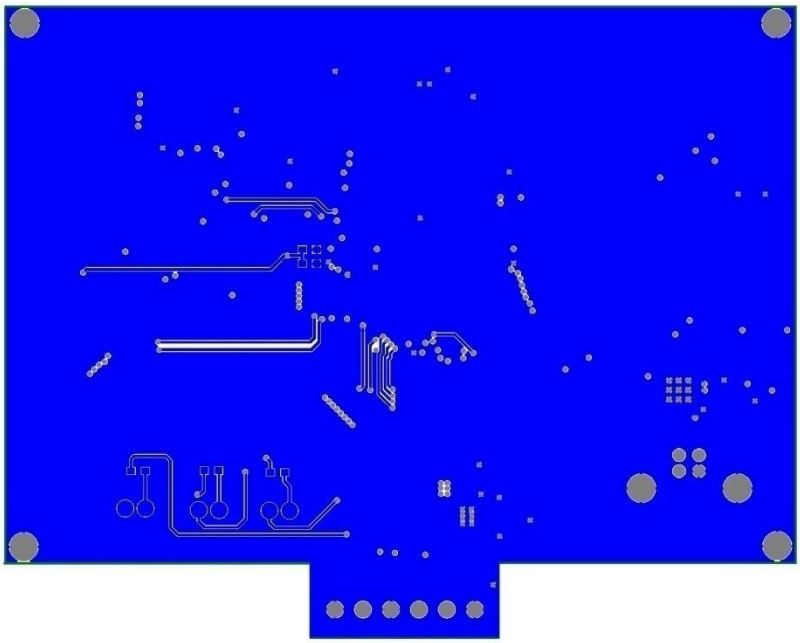


Figure 6.22: Bottom Layer of 1C601 Control Board

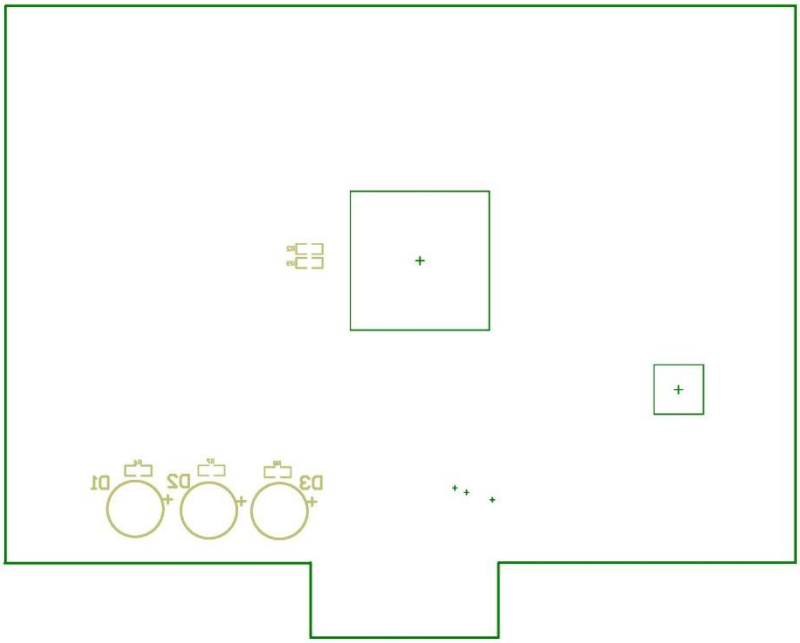


Figure 6.23: Bottom Overlay of 1C601 Control Board

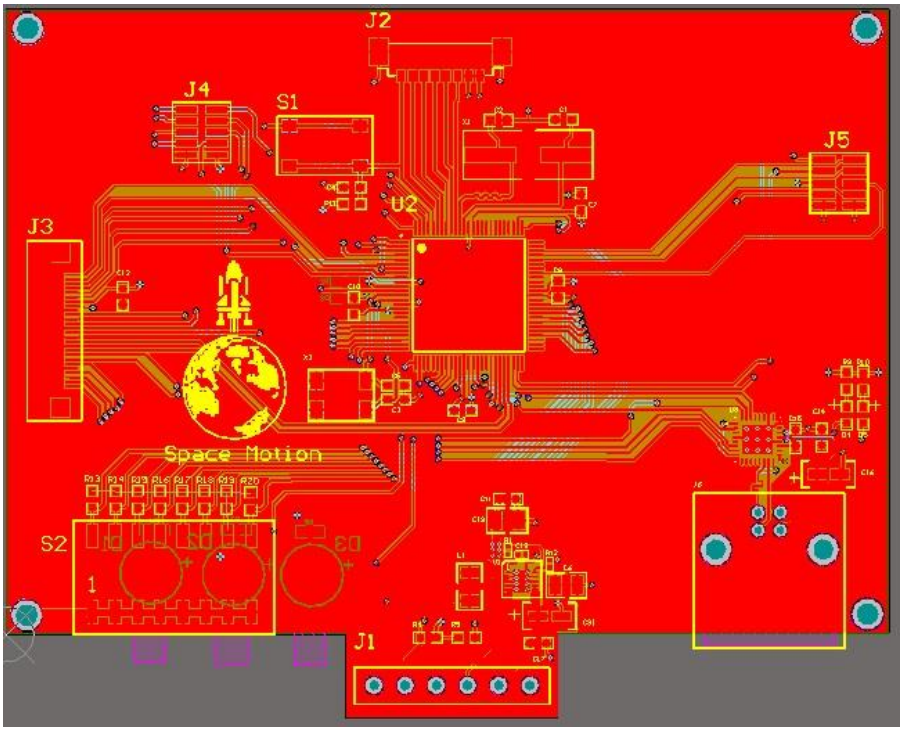


Figure 6.24: All PCB Layers 1C601 Control Board

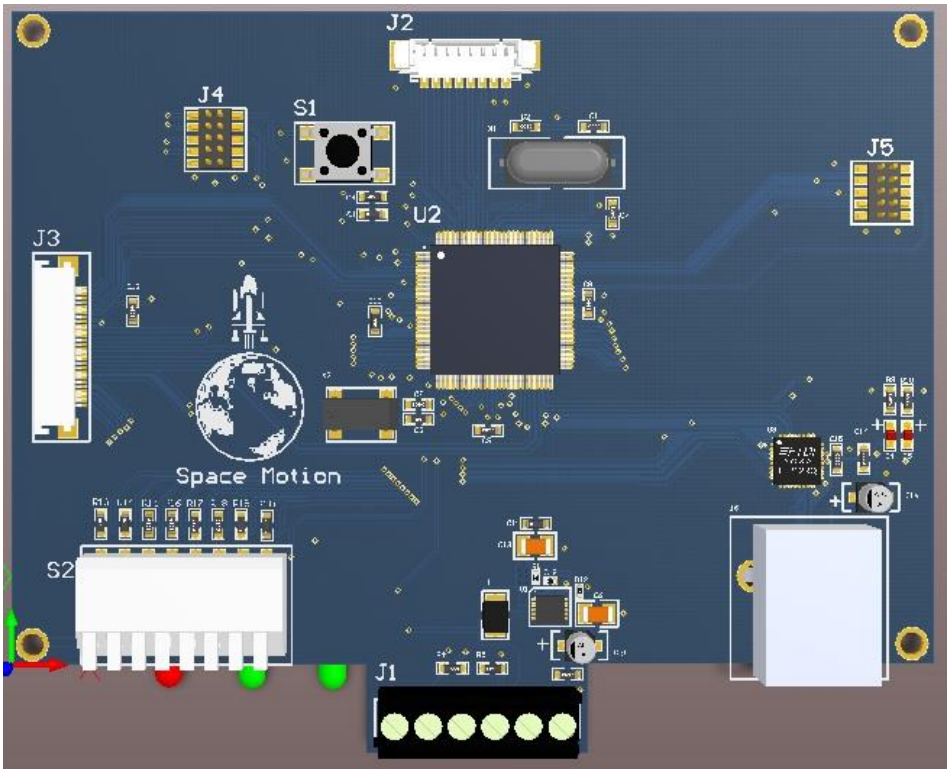


Figure 6.25: TOP view 3D Model of 1C601 Control Board

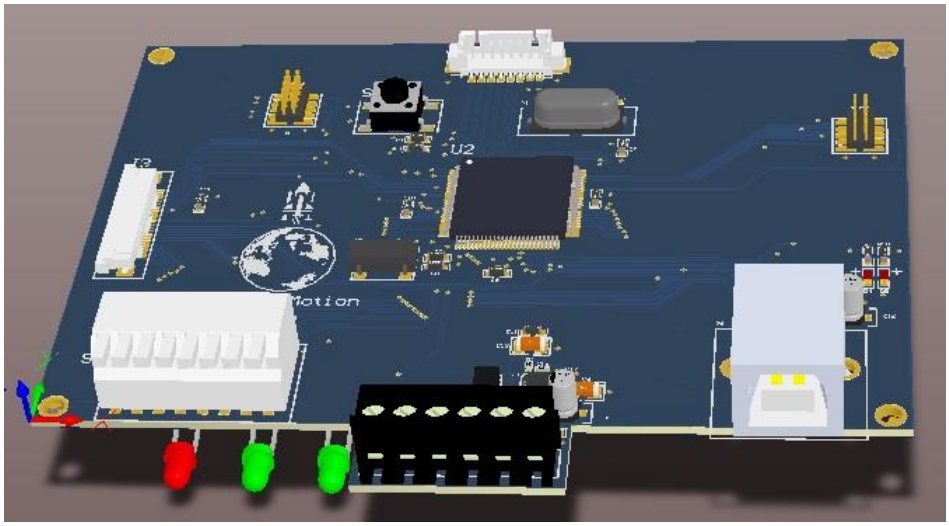


Figure 6.26: SIDE view 3D Model of 1C601 Control Board

CHAPTER VII: Basic Communication Protocol

This section describes the half-duplex Basic Communication Protocol for communication between one and only one Master and slave one or more.

The Basic Communication Protocol supports

the following actions (see the corresponding descriptions):

- Write Data - when a Master wants to transfer up to 256B of data to a Slave;
- Read Data - when a Master wants to read up to 256B of data from a Slave;
- Command Only - when a Master wants to deliver a data-less command to a Slave.

Most data transfers contain, from the Master to the Slave:

- an appropriate START Indicator; the nature of the START Indicator depends on the actual protocol chosen
- an 8-bit Slave address to address a specific Slave
- a 16-bits command
- an 8-bit data length field (only for Write Data, Write Read Data and Broadcast Write Data);
- data (1B to 256B; only for Write Data)
- a 16-bit CRC check. CRC algorithm is a CRC-16 of all bytes (including command/ID and length fields)

- An appropriate STOP Indicator; the nature of the STOP Indicator depends on the actual protocol chosen.

From the Slave to the Master:

- an 8-bit Slave ID to identify the Slave type
- an 8-bit data length field (only for Read Data)
- data (1B to 256B; only for Read Data)
- a 16-bit CRC check. CRC algorithm is a CRC-16 of all bytes (including command/ID and length fields).

Figure 7.1 shows complete message format write data from Master to Slave.

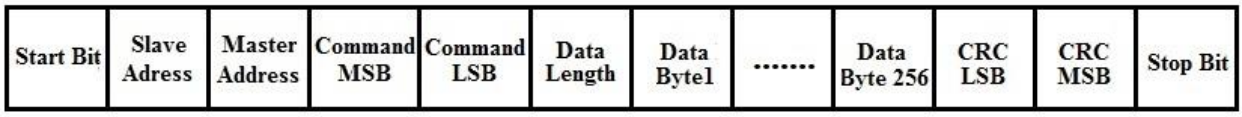


Figure 7.1: Message format for write data from Master to Slave

The actual 1st byte of the message starts from Master Address because start bit, slave address are handled at low level of micro controller I2C module. After slave address reception actual data will be counted.

7.1 Start Indicator:

All data transfers are initiated by the Master only when the bus is free, that is, all previous communications are terminated or after an appropriate timeout. Communication starts with a START Indicator, It sends START flag (Master -> Slave), that is, by lowering SDA line when SCK is high (invalid data bit).

7.2 Stop Indicator:

Communication is terminated by the Master with a stop operation, which can either be, It sends STOP flag (Master -> Slave), that is, by rising SDA line when SCK is high (invalid data bit).

7.3 Slave Address:

Master sends 7 bit address with 8^{th} bit for read and writes.

7.4 Master Address:

Master sends its own address to the slave. Only useful when multi master mode is used.

7.5 Command MSB and LSB:

Command size in this protocol is 16 bit. Master Divide the 16 Bit command and sends command MSB first than LSB of Command. When Slave receives the MSB and LSB of the command, it recombines both bytes and stores it as a command receives from master.

7.6 Data Length:

This protocol allows from 1 to 256 bytes of data in one message. 4^{th} field of the message is reserved for data length. Master writes the data length of the data packet it wishes to send. After receiving data length slave knows the size of next incoming data packet. This technique avoids data loss and increase code efficiency to handle the data.

7.7 CRC Check:

The CRC module produces a signature for a given sequence of data values. The signature is generated through a feedback path from data bits 0, 4, 11, and 15 (see Figure 7.2).

The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial:

$$f(x) = x16 + x12 + x5 + 1 \quad (10)$$

Identical input data sequences result in identical signatures when the CRC is initialized with a fixed seed value, whereas different sequences of input data, in general, result in different signatures. Initial seed must be CRC_SEED: ushort (0xFFFF) for all Basic

Communication Protocols. Once all data have been processed through the CRC check, the value stored inside the register is added at the end of data for error protection.

As above mention 16 Bit CRC engine is used. Master calculates the CRC of complete message including: Master address, Command, Data Length, Data Packet and 16 Bit CRC. When slave receives 16 Bit CRC it will join them and calculate CRC of received message and compare both received and calculated. Slave will take action on commands only if only both calculated and received CRC are matches with each other.

7.8 Data Operation Modes:

First four most significant bits of Command is used to detect which data operation is required. Following are data operation modes correspond to first four bits of Command:

1. 0000= No Operation
2. 0001= Write Operation (Slave to Master)
3. 0010= Read Operation (Master to Slave)
4. 0011= Command Only

7.8.1 Command NOP Operation:

When first four bits of the received command are zero it means no operation. Slave respond nothing.

7.8.2 Write Operation:

When the value of first four bits of command is 1 it represents write operation. As soon as these bits receives slave immediately identify the data operation mode and keep itself ready to write data to the Master.

7.8.3 Read Operation:

When the value of first four bits of command is 2 it represents read operation. As soon as these bits receives slave immediately identify the data operation mode and keep itself ready to read data from the Master.

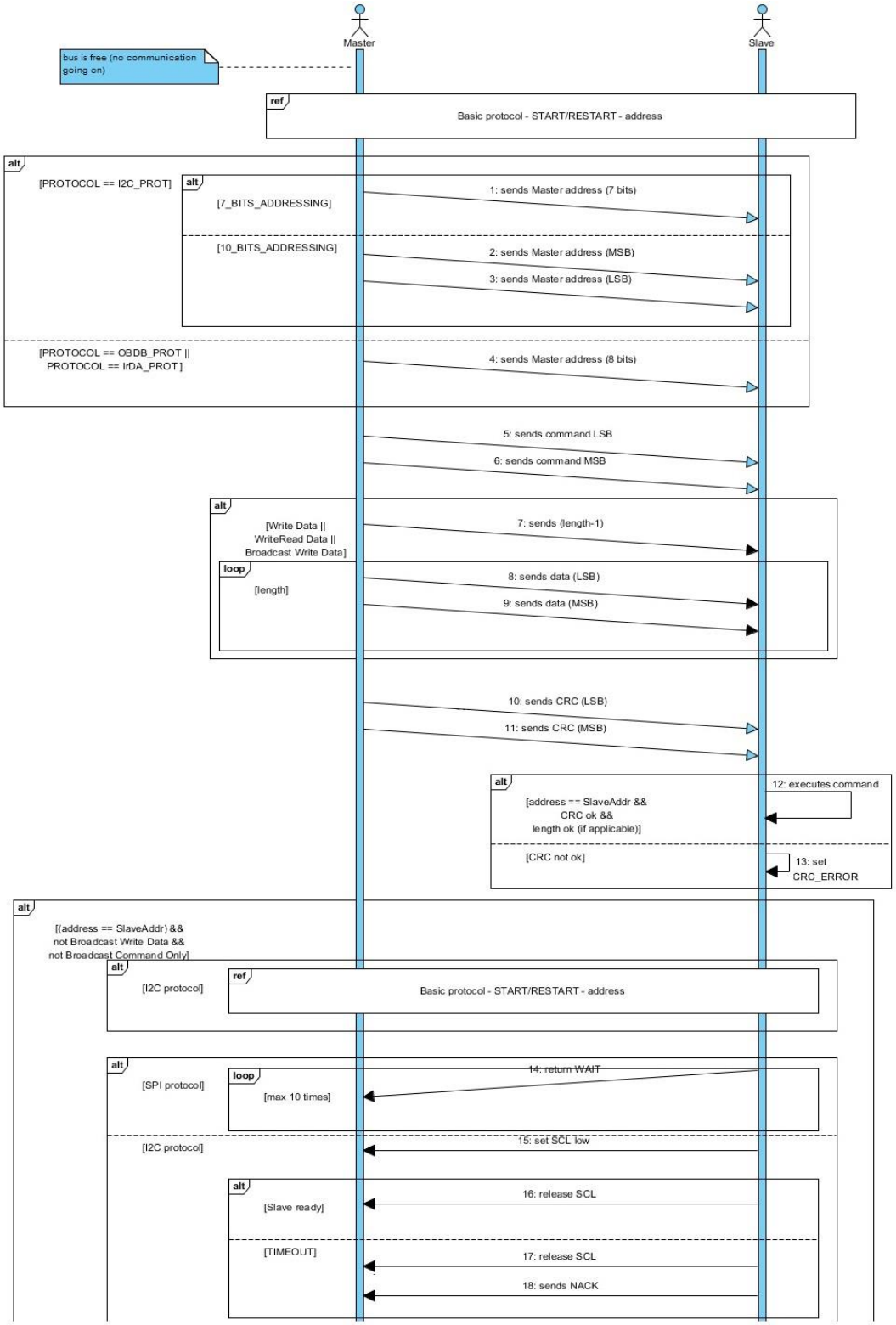
7.8.4 Command Only:

When the value of first four bits of command is 3 it represents Command Only operation. As soon as these bits receives slave immediately identify the data operation mode and keep itself ready to read Data length which will be zero and as CRC. Command mode only associated with only those commands which require no data.

7.9 Error Handling:

If calculated and received CRC does not match slave will add one to the error counter. After 15 attempts of wrong data when error counter hits 15, slave will turn on Buzzer.

7.10 Sequence Diagram:



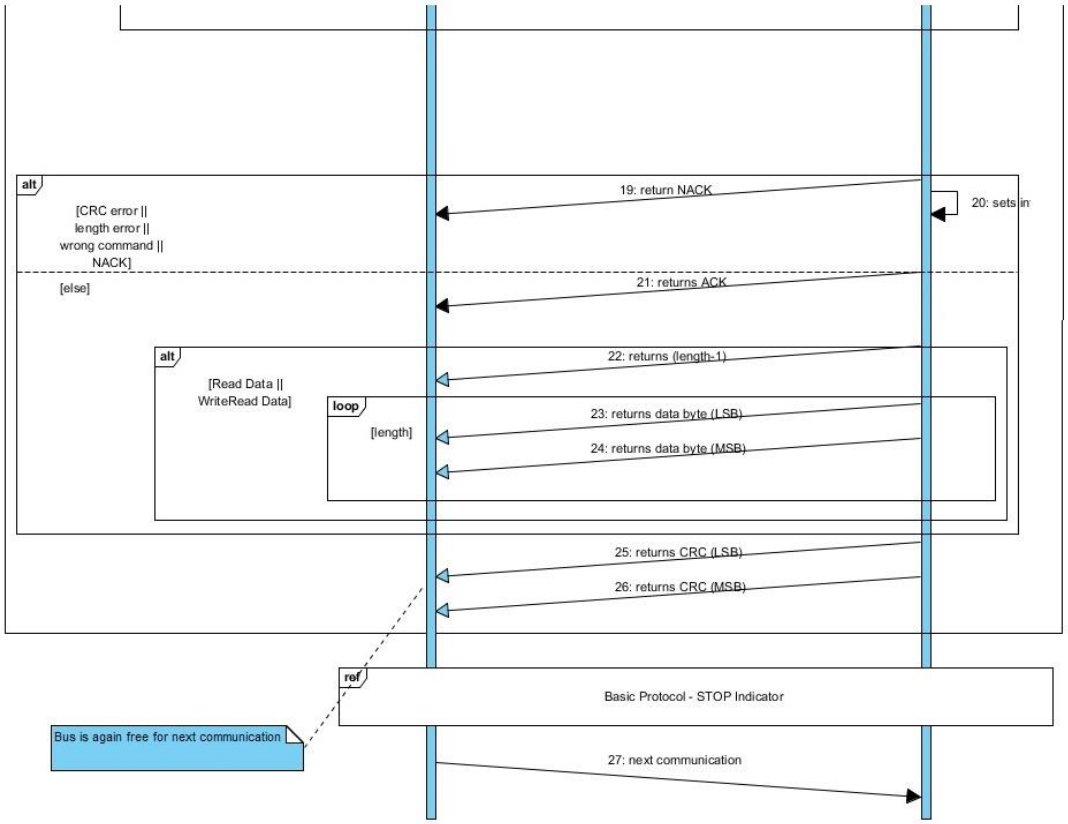


Figure 7.2 Sequence Diagram of Basic Communication protocol

CHAPTER VIII: Command Set

I developed special 16 Bit command set for the solar simulator. This command set contain all the require commands to configure, run, monitor and trouble shoot the solar simulator. Solar simulator is programmed in such a way that whenever certain command from command set is sent to solar simulator it will respond to that command. All commands are sending by following the basic communication protocol which explained above. With each command 16 bit hex value is assigned. First four bits are used for data operation while the remaining 12 bits are for the system command set. Max number of commands is 4096.

Table 8.1 shows the command list and their values:

COMMAND NAME	COMMAND VALUE
ILLUMINATE_THERMAL_BLACK	0x30B1
ILLUMINATE_THERMAL_GREEN	0x30B3
ILLUMINATE_THERMAL_BLUE	0x30B2
ILLUMINATE_SINGLE_JUNCTION	0x30B5
ILLUMINATE_TRIPLE_JUNCTION	0x30B4
ILLUMINATE_VARIABLE	0x20CF
SET_LIGHT_PERIOD	0x20EE
START_TEST	0x30BB
PAUSE	0x30BD
RESUME	0x30C2
STOP_TEST	0x30BC
START_SELF_TEST	0x30B6
GET_SELF_TEST_STATUS	0x10B7
GET_STATUS	0x10B8
RESET	0x30BE
DEBUG_ON	0x30B9
DEBUG_OFF	0x30BA
SET_EXPERIMENT_PERIOD	0x20BF
GET_SUN_EXPOSURE_TIME	0x10C2
GET_ELAPSED_ORBITS	0x10C3
GET_ELAPSED_EXPERIMENT_TIME	0x10C4
GET_POWER_CONSUMPTION	0x10C5
GET_HW_SW_SERIAL_NUMBER	0x10C6

ACQUIRE_ANALOG_VOLTAGE_CH1	0x10C8
ACQUIRE_ANALOG_VOLTAGE_CH2	0x10C9
ACQUIRE_ANALOG_VOLTAGE_CH3	0x10CA
ACQUIRE_INTERNAL_TEMPERATURE	0x10CB
ACQUIRE_CURRENT_ALL_CHANNELS	0x10CC
TURN_OFF_POWER	0x30CD
ENTER_LAMP_CALIBRATION	0x20CE
SET_VOLTAGE_ON_POW_OUT	0x20D2
GET_LAMPS_LIFE	0x10D3
SET_Current_ON_POW_OUT	0x20D4
SET_TIME	0x20D5

Commands starting with 1 are the commands which request data from the slave by the Master, with 2 are the commands which gives data to the slave by the master and commands start with 3 request no data.

8.1 Commands Description:

This section explains the description of commands.

8.1.1 ILLUMINATE_THERMAL_BLACK:

ILLUMINATE_THERMAL_BLACK 16 bit command illuminates the Sat Thermally Black.

8.1.2 ILLUMINATE_THERMAL_GREEN:

ILLUMINATE_THERMAL_GREEN 16 bit command illuminates the Sat Thermally Green

8.1.2 ILLUMINATE_THERMAL_BLUE:

ILLUMINATE_THERMAL_BLUE 16 bit command illuminates the Sat Thermally Blue.

8.1.2 ILLUMINATE_SINGLE_JUNCTION:

ILLUMINATE_SINGLE_JUNCTION 16 bit command illuminates the Sat Electrically Single Junction.

8.1.2 ILLUMINATE_TRIPLE_JUNCTION:

ILLUMINATE_TRIPLE_JUNCTION 16 bit command illuminates the Sat Electrically Triple Junction.

8.1.2 ILLUMINATE_VARIABLE

ILLUMINATE_VARIABLE 16 bit command Illuminates the Sat with user defined custom intensity. When Master sends this command to Slave, Master should also send Custom Intensity Level in first two bytes of Slave Receive Buffer Right after the command.

8.1.2 SET_LIGHT_PERIOD:

SET_LIGHT_PERIOD 16 bit command sets the Light period. When Master sends this command to Slave Master should also send following parameters immediately to the Slave:

1. Write Illumination Period in seconds into first four bytes of receive Buffer of Slave.

2. Write Duty Cycle in % in 5th Byte of Receive Buffer of Slave

3. Writes Overall test duration in Seconds from 6^{th} Byte and of 9^{th} Byte Receive Buffer of Slave.

8.1.2 START_TEST:

START_TEST 16 bit command starts the Experiment.

8.1.2 PAUSE:

PAUSE 16 bit command to pause the running experiment.

8.1.2 RESUME:

RESUME 16 bit command to resume the pause Experiment.

8.1.2 STOP_TEST:

STOP_TEST 16 bit command stops the Experiment.

8.1.2 START_SELF_TEST:

START_SELF_TEST 16 bit command starts the Self Test.

8.1.2 GET_SELF_TEST_STATUS:

GET_SELF_TEST_STATUS 16 bit command to get the status of running Self Test and Slave will write the status in first element of transmit buffer of Slave and master can access it by reading it. The result is either self test is running or its completed.

If result is 1 than its means self test is completed and for 0 it's meaning self test is still running. Result of self test is stored in status register and user can access the result by reading status register content.

8.1.2 GET_STATUS:

GET_STATUS 16 bit command to get the 16 bit Status Register and Slave will write the status register in first two elements of transmit buffer of Slave and master can access it by reading it.

8.1.2 RESET:

RESET 16 bit command resets the system flags such as error Flag

8.1.2 DEBUG_ON:

DEBUG_ON 16 bit command to turn ON Debug mode.

8.1.2 DEBUG_OFF:

DEBUG_OFF 16 bit command to turn OFF Debug mode.

8.1.2 SET_EXPERIMENT_PERIOD:

SET_EXPERIMENT_PERIOD16 bit command to Set the Experiment Period. After Sending this command Master will write the Experiment Time in Minutes to the first four elements of Receive Buffer of Slave.

8.1.2 GET_SUN_EXPOSURE_TIME:

GET_SUN_EXPOSURE_TIME 16 bit command to get how much is the exposure time of sun. When Slave receive this command , it will write the Sun exposure time in its first four elements of transmit buffer and Master can access the exposure time by reading first four elements of Slave Transmit buffer.

8.1.2 GET_ELAPSED_ORBITS:

GET_ELAPSED_ORBITS 16 bit command to get how many orbits are elapsed When Slave receive this command it will write the number of elapsed orbits in its first four elements of transmit buffer and Master can access the orbits number by reading first four elements of Slave Transmit buffer.

8.1.2 GET_ELAPSED_EXPERIMENT_TIME:

GET_ELAPSED_EXPERIMENT_TIME 16 bit command to get Elapsed Time of Experiment. When Slave receive this command it will write the Elapsed time of Experiment in its first four elements of transmit buffer and Master can access the Elapsed time of Experiment by reading first four elements of Slave Transmit buffer.

8.1.2 GET_POWER_CONSUMPTION:

GET_POWER_CONSUMPTION 16 bit command to get the power consumption of 1C6412 1U Light Source. When Slave receive this command it will write the power consumption in its first four elements of transmit buffer and Master can access the power consumption of 1C6412 1U Light Source by reading first two elements of Slave Transmit buffer.

8.1.2 GET_HW_SW_SERIAL_NUMBER:

GET_HW_SW_SERIAL_NUMBER 16 bit command to get the Hardware and Software serial number of 1C6412 1U Light Source. When Slave receive this command it will write the Hardware and Software serial number in its first four elements of transmit

buffer and Master can access the Hardware and Software serial number by reading first four elements of Slave Transmit buffer.

8.1.2 ACQUIRE_ANALOG_VOLTAGE_CH1:

ACQUIRE_ANALOG_VOLTAGE_CH1 16 bit command to get the analog voltage on channel 1.When Slave receive this command it will write the analog voltage on channel 1 in its first two elements of transmit buffer and Master can access the analog voltage on channel 1 by reading first two elements of Slave Transmit buffer.

8.1.2 ACQUIRE_ANALOG_VOLTAGE_CH2:

ACQUIRE_ANALOG_VOLTAGE_CH2 16 bit command to get the analog voltage on channel 2.When Slave receive this command it will write the analog voltage on channel 2 in its first two elements of transmit buffer and Master can access the analog voltage on channel 2 by reading first two elements of Slave Transmit buffer

8.1.2 ACQUIRE_ANALOG_VOLTAGE_CH3:

ACQUIRE_ANALOG_VOLTAGE_CH3 16 bit command to get the analog voltage on channel 3.When Slave receive this command it will write the analog voltage on channel 3 in its first two elements of transmit buffer and Master can access the analog voltage on channel 3 by reading first two elements of Slave Transmit buffer.

8.1.2 ACQUIRE_INTERNAL_TEMPERATURE:

ACQUIRE_INTERNAL_TEMPERATURE 16 bit command to get the internal temperature of 1C6412 1U Light Source. When Slave receive this command it will write

the internal temperature in its first element of transmit buffer and Master can access the internal temperature by reading first element of Slave Transmit buffer.

8.1.2 ACQUIRE_CURRENT_ALL_CHANNELS:

ACQUIRE_CURRENT_ALL_CHANNELS 16 bit command to get current consumption on each power drive channel. When Slave receive this command it will write the current consumption of channel 1 to channel 5 its first five elements of transmit buffer and Master can access the current consumption's by reading first five elements of Slave Transmit buffer.

8.1.2 TURN_OFF_POWER:

TURN_OFF_POWER 16 bit command to turn off all power driver channels.

8.1.2 ENTER_LAMP_CALIBRATION:

ENTER_LAMP_CALIBRATION 16 bit command to send lamp calibration data to the 1C6412 1U Light Source. After sending this command Master will write the lamp calibration data to the first thirteen elements of Receive Buffer of Slave.

8.1.2 SET_VOLTAGE_ON_POW_OUT:

SET_VOLTAGE_ON_POW_OUT 16 bit command to Set the voltage on specific power out channel. After Sending this command Master will write the Power Out channel number in first element and voltage in second element and third Byte in the Receive Buffer of Slave.

8.1.2 GET_LAMPS_LIFE:

GET_LAMPS_LIFE 16 bit command to get the remaining life of all four lamps mounted in 1C6412 1U Light Source. When Slave receive this command it will write

the remaining lamps life in its first four elements of transmit buffer and Master can access the remaining lamps life by reading first four elements of Slave Transmit buffer.

8.1.2 SET_Current_ON_POW_OUT:

SET_Current_ON_POW_OUT 16 bit command to set the Current on specific power out channel. After Sending this command Master will write the Power Out channel number in first element and Current in second and third element in the Receive Buffer of Slave.

8.1.2 TURN_OFF_POWER:

SET_TIME 16 bit command to set the time of 1C6412 1U Light Source. After Sending this command Master will write the Time in HR: MIN: SEC Format. Hours, Minutes and Seconds each requires two bytes. So Master writes time in first six element of a Receive Buffer of Slave.

CHAPTER IX: Potential Applications

Table 9.1: Potential Applications

S.No	Applications	Description
1	Photovoltaic Cell performance Testing	Solar simulators are widely using to performance of Photovoltaic cell from small scale to massive scale. There is huge market for solar cell testing. As the time is passing the solar cells are being used widely from cell phone charger to home power. 1C6412 1U Light source due to compact size and extreme electronics makes it versatile. These can be stacked together horizontally and vertically for larger applications. Upto 1024 1C6412 1U Light source can be stacked together.
2	Materials testing	Sunlight can have adverse affects on materials and components, oftentimes initiating and accelerating the degradation process as it interacts with temperature, moisture and other environmental effects. In addition, it is critical to understand the effects of heat created by sunlight with respect to operational performance, thermal management, noise and dimensional stability. A new product should be tested under solar environmental conditions representative of those locations in which it will exist - anywhere ranging from the heat of the outback in Australia to the frigid climate of arctic areas. 1C6412 1U Light source is well designed in both power and size. Having this flexibility, it can be integrated into various types of environmental test chambers whether they be small or walk-in, chambers used in component/ small product testing or drive-in chambers for complete vehicle testing, even up to extra large systems for trains, trucks and aircrafts.

3	Cosmetic testing	Photo safety testing is suggested for any chemicals that absorb light in the range from 290-700nm for any substance that may be applied topically or can reach the skin and/or eyes by systemic exposure (oral or intravenous). There are four basic results or endpoints that photo-safety testing address. These are: • **Photo-toxicity** sometimes referred to in literature as photo-irritation is described as an acute light-induced skin response to a photo-reactive chemical. • **Photo-allergy** is described as an immune reaction to a chemical initiated by the formation of photo-products. This can be a byproduct of exposure to an antigen. • **Photo-genotoxicity** is described as a genotoxic response after exposure to a chemical which is photo-activated by UV or VIS light. • **Photo-carcinogenicity** is described as the potential for a chemical to promote skin tumor formation in combination with exposure to UV light. • 1C6412 1U Light source is well suited to test cosmetic to check above mentioned results.
4	Solar thermal collector Testing	A solar thermal collector collects heat by absorbing sunlight. A collector is a device for capturing solar radiation. Solar Thermal Collector is huge market from small level to massive for power Generation.
5	Lightning	1C6412 1U Light source can be used for lightning application. There are 4 lamps in Light Source. By putting Three colors Glass in front of lamps we will have capability of generating almost any color with high intensity.

CHAPTER X: Future Work & Conclusion

The work discussed in this thesis allows to understand the development of 1C601 1U Light source which is major part of low cost solar simulator. Although this light source is design specifically to test nano satellites but can also be used for other large scale applications due to its modularity.

This thesis describes the development of complete electronics and communication protocol of 1C601 1U light source. Electronics of 1C6412 1U Light source includes power driver board, Filter board and control board.

Main problem was to design the power driver board which should be EMC and EMI Compliant and also have four galvanically closed loop isolated 120 watt pwm output channels in a very small PCB and one low power 1Amp pwm output channel.

After completion of 1C601 power driver board next milestone was to design 1C601 Filter board and 1C601 Control board. Alll Schematics were carefully designed by considering power and thermal constraints with respect to 1C6412 1U Light source.

Last milestone was to design complete command set and communication protocol able to handle all communications regarding commands, large data exchange with 16Bit CRC check.

Till now electronics and communication protocol is completely developed for 1C6412 1U Light source. Communication protocol is tested and working efficiently.

Finally, future lines for this project will be related on the development of system firmware to handle all system level tasks, PCB fabrications and their testing and mounting in 1C6412 1U Light source.

REFERENCES

[1] MSP430F5438 Datasheet
http://www.ti.com/lit/ds/symlink/msp430f5438.pdf

[2] FTDI232 Datasheet
http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf

[3] TPS61200 Datasheet
http://www.ti.com/lit/ds/symlink/tps61200.pdf

[4] Pressure Drop in circular Ventilation Channel
http://www.claredot.net/en/sec-Aerolics/pressure-drop-circular-air-pipe.php

[5] Air Specific Heat Capacity
http://www.engineeringtoolbox.com/air-specific-heat-capacity-d_705.html

[6] Mosfet Heat Dissipation Calculation
http://electronicdesign.com/boards/calculate-dissipation-mosfets-high-power-supplies

[7] EMI Filters wiki
https://en.wikipedia.org/wiki/Electromagnetic_interference

[8] I2C Communication wiki
https://en.wikipedia.org/wiki/I%C2%B2C

[8] USB Communication wiki
https://en.wikipedia.org/wiki/USB

[9] USB Communication wiki
https://en.wikipedia.org/wiki/USB

[10] ADuM3223 Datasheet
http://www.analog.com/media/en/ /datasheets/ADuM3223_4223.pdf

[11] ACS711 Datasheet
http://www.allegromicro.com/~/Media/Files/Datasheets/ACS711-Datasheet.ashx

[13] Yafae Yuan, " Research of Solar Simulator irradiance non-uniformity Measurement," In Electronics Measurement & Instruments (ICEMI), Vol.3 , Chengdu: IEEE, 2011, pp. 307-310

[13] Magden, E.S.; Han Chen; Downs, C.; Vandervelde, T.E. "3+1 multijunction testing And operations platform for improved PV and TPV efficiencies", Innovative Technlogies for an Efficient and Reliable Electricity Supply (CITRES), 2010 IEEE Conference on, On page(s): 116 - 120

Appendix A

Slave.h:

```
//*****************************************************************************//
//                                                                             //
//                          LOW COST SOLAR SIMULATOR                           //
//                                Version 1.1                                  //
//                             Author:Sanwal Saleem                            //
//                         Supervisor:Prof. Leonardo Reyneri                   //
//                                                                             //
//*****************************************************************************//

#pragma diag_suppress=Pa050
#pragma diag_suppress=Pe382
#ifndef __Test_h__
#define __Test_h__

#include "platform.h"
#if MSP430FRxxxx
#include "MSP_430FR6989_IPZ.h"
#else
#include "MSP_430F5438A.h"
#endif

#include "CPU_DESCRIPTOR_DEFAULT.h"
#include "IOdriver.h"
#include "PWM_A0.h"

/*********************************Command Set*****************************//
enum t_command
    {
        ILLUMINATE_THERMAL_BLACK = 0x30B1,
        ILLUMINATE_THERMAL_GREEN=0x30B3,
        ILLUMINATE_THERMAL_BLUE=0x30B2,
        ILLUMINATE_SINGLE_JUNCTION=0x30B5,
        ILLUMINATE_TRIPLE_JUNCTION=0x30B4,
        ILLUMINATE_VARIABLE=0x20CF,
        SET_LIGHT_PERIOD=0x20EE,
        START_TEST=0x30BB,
        PAUSE=0x30BD,
        RESUME=0x30C2,
        STOP_TEST=0x30BC,
        START_SELF_TEST=0x30B6,
        GET_SELF_TEST_STATUS=0x10B7,
        GET_STATUS=0x10B8,
        RESET=0x30BE,
        DEBUG_ON=0x30B9,
        DEBUG_OFF=0x30BA,
        SET_EXPERIMENT_PERIOD=0x20BF,
        GET_SUN_EXPOSURE_TIME=0x10C2,
        GET_ELAPSED_ORBITS=0x10C3,
        GET_ELAPSED_EXPERIMENT_TIME=0x10C4,
        GET_POWER_CONSUMPTION=0x10C5,
        GET_HW_SW_SERIAL_NUMBER=0x10C6,
        ACQUIRE_ANALOG_VOLTAGE_CH1=0x10C8,
        ACQUIRE_ANALOG_VOLTAGE_CH2=0x10C9,
        ACQUIRE_ANALOG_VOLTAGE_CH3=0x10CA,
        ACQUIRE_INTERNAL_TEMPERATURE=0x10CB,
```

```
        ACQUIRE_CURRENT_ALL_CHANNELS=0x10CC,
        TURN_OFF_POWER=0x30CD,
        ENTER_LAMP_CALIBRATION=0x20CE,
        SET_VOLTAGE_ON_POW_OUT=0x20D2,
        GET_LAMPS_LIFE=0x10D3,
        SET_Current_ON_POW_OUT=0x20D4,
        SET_TIME=0x20D5
    };

/*****************************System Variabls*************************/
static int data[260];
unsigned long static Illumination_Period=0,Test_Duration=0;
unsigned long static Sun_Exposure_Time=25678,Elapsed_Orbits,Elapsed_Time;
unsigned long static HW_SW_SERIAL_NUMB,Experiment_Time=0;
unsigned int  static cnt=0,RX_Counter=4,TX_Cnt=0,count=0;
unsigned char static Command_LSB,Command_MSB,Data_Length,TX_Length;
unsigned char static Command_Data_Op, CRC_Cnt=0,TX_Data_Length=0,RX_Length;
unsigned char static CRC_MSB,Rx_Data[260],DUMMY,Bulb_Callibration[13];
unsigned char static Illumination_Duty_Cycle=0,Channel_Number_v;
unsigned char static TX_DATA[256],CRC_TX_MSB,CRC_TX_LSB,Internal_Temp;
unsigned char static Lamp1_Life,Lamp2_Life,Lamp3_Life,Lamp4_Life;
unsigned char static Self_test_status=1,Channel_Number_i,CRC_LSB;
unsigned int static results[4],adc_data=54545545;
short static CRC,CALCULATED_CRC,Voltage_mv=0,Current_mA=0,CRC_TX;
short static voltage_ch1_mv,voltage_ch2_mv,voltage_ch3_mv,STATUS_REG=0xABCD;
short static Current_ch1_mA, Current_ch2_mA, Current_ch3_mA, Current_ch4_mA;
short static Current_ch5_mA;
unsigned short static Status=0x0000,duty_cycle=0,Length=0,Custom_Intensity=0;
unsigned short static Hours=0,Minutes=0,Seconds=0,Current_system_Power_mW;
unsigned short static AN_Val_1=0;
unsigned short static Command;
bool static Command_Flag=0,Command_MSB_Flag=0,Data_Length_Reception=0;
bool static Write_Flag=0,Read_Flag=0,Command_Only=0,COM_NOP=0,CRC_Flag=0;
bool static Start_Flag=0,Stop_Flag=0,Resume_flag=0,Pause_flag=0;
bool static Debug_OFF_flag=0,Start_Self_test_flag=0,Reset=0,Debug_ON_flag=0;
bool static Power_Tranfer=0,TX_Ready,Start_self_test_flag=0;

/*************************CRC 16 Bit Cal Function**********************/
short Calc_CRC_C(unsigned char *Buffer, unsigned short Len);


namespace MSP_430
{
    template <long PORT, byte BIT, bool INVERT> class IOdriver;
    class Test;
}

namespace MSP_430
{
    class Test
    {
          static int cnt;
        private:
        #if MSP430FRxxxx
static MSP_430::MSP_430FR6989_IPZ<MSP_430::CPU_DESCRIPTOR_DEFAULT> proc;
#else
static MSP_430::MSP_430F5438A<MSP_430::CPU_DESCRIPTOR_DEFAULT> proc;
#endif

        private:

        public: void test();
```

```
/*******************************I2C Interrupt*****************************/
        public:
#pragma vector = USCI_B0_VECTOR
        static  __interrupt void uartB0_isr() {


          if (proc.uartB0.isI2Cstart())  // Start Condition detection
          {

          }
          if (proc.uartB0.isRXready())     // I2C reception Ready
          {
            TX_Cnt=0;
            P1OUT ^= BIT0; // Toggling data on every Byte Reception
            if(count==0)   //Store First recieved byte in buffer
            {
               Rx_Data[0]=  proc.uartB0.readData();
               TX_Data_Length= Rx_Data[0];
            }
            if(count==1)  //Store Command MSByte in buffer
            {
               Command_MSB=proc.uartB0.readData();
               Rx_Data[1]=Command_MSB;
            }
            if(count==2) //Store Command LSByte in buffer
            {
               Command_LSB=proc.uartB0.readData();
               Rx_Data[2]=Command_LSB;
               Command_Flag=1;  //Command Reception Flag
            }
            if(count==3)  //Store Data Length in buffer
            {
               Data_Length=proc.uartB0.readData();
               Rx_Data[3]=Data_Length;
               Data_Length_Reception=1;// Data Reception Flag
            }
            if(count==(4+Data_Length+1)) // CRC MSByte Store
            {
               CRC_MSB=proc.uartB0.readData();
            }
            else if(count==(4+Data_Length+2)) // CRC LSByte Store
            {
               CRC_LSB=proc.uartB0.readData();
               CRC_Flag=1;     // CRC reception Flag
            }
            else   // Storing remaining Byte untill it reaches Data
            {  // Data Length Limit
               Rx_Data[count]=proc.uartB0.readData();
            }
            count++;   // Incrementing Recieved Bytes Counter
          }
          if (proc.uartB0.isTXready())  // Transmit Ready
          {
            proc.uartB0.writeData(TX_DATA[TX_Cnt++]); // Sending Data
          }
          if (proc.uartB0.isI2Cstop())             //Stop Condition
          {
            count=0;       // Reseting the Recieved Byte Counter
          }

        }};}#endif
```

Appendix B

Main.cpp:

```
//*********************************************************************//
//                                                                     //
//                        LOW COST SOLAR SIMULATOR                     //
//                              Version 1.1                            //
//                          Author:Sanwal Saleem                       //
//                     Supervisor:Prof. Leonardo Reyneri               //
//                                                                     //
//*********************************************************************//
#include "Test.h"

#include "CPU_DESCRIPTOR_DEFAULT.h"
#include "IOdriver.h"
#include "platform.h"
unsigned long delay;
unsigned long value45;
unsigned long CurrentPosition=0xABCDEBFE;
unsigned char *qs = (unsigned char*)&CurrentPosition;
unsigned char arr[4];
int x=0,y=4;
bool Data_valid=0;

void MSP_430::Test::test()
{
   // Stop watchdog timer to prevent time out reset
   proc.wdt.disable();
   __enable_interrupt();  // Enabling Interrupts
   proc.init();           // Initialzing Hal
   P1dir |= BIT0;         // Setting Direction of the Digital Pins
   P1out |= BIT0;
   proc.uartB0.init();    // Initializing I2C
   proc.uartB0.enable(MSP_430::I2C_SLAVE_MODE, 100000);
   proc.uartB0.set_I2C_address(0x20,0x10); //Setting I2C Address
   proc.uartB0.enableInterrupts(true,true,true,true,false,false);

   while (1)
   {

      if(Command_Flag==1)// Merging Command MSB and Command LSB
      {
          Command= ( Command_MSB << 8 ) | Command_LSB;
          Command_Flag=0;   // Reseting Command reception Flag
      }
      if(CRC_Flag==1)     //Commbining CRC MSB and LSB and
      {                   //check either its matched or not
          CALCULATED_CRC= Calc_CRC_C(Rx_Data,Data_Length+4);
          CRC= ( CRC_MSB << 8 ) | CRC_LSB;
          if(CRC==CALCULATED_CRC)
          {
              Data_valid=1; //On CRC match Data Valid Flag is 1
          }
          else           //On CRC Un match Data Valid Flag is 0
          {
              Data_valid=0;
          }
          CRC_Flag=0;
```

```
    RX_Counter=0;
    CRC_Cnt=0;
}
if(Data_valid==1)  // On Valid Data Reception, Command
{                  // Encoding and actions on it
    switch (Command)
    {
        case ILLUMINATE_THERMAL_BLACK:
             duty_cycle=1024;
        break;
        case ILLUMINATE_THERMAL_GREEN:
        break;
        case ILLUMINATE_THERMAL_BLUE:
        break;
        case ILLUMINATE_SINGLE_JUNCTION:
        break;
        case ILLUMINATE_TRIPLE_JUNCTION:
        break;
        case ILLUMINATE_VARIABLE:  // merging two bytes
             Custom_Intensity= ( Rx_Data[4] << 8 )
             | Rx_Data[5];
        break;
        case SET_LIGHT_PERIOD:
             // Merging four recieved bytes of Ilumination
             // Period and storing in long variable
             Illumination_Period = long(Rx_Data[7]) << 24 |
             long(Rx_Data[6]) << 16 | long(Rx_Data[5]) << 8
             | long(Rx_Data[4]);
             Illumination_Duty_Cycle=Rx_Data[8];
             // Merging four recieved bytes of Test
             // duration and storing in long variable
             Test_Duration = long(Rx_Data[12]) << 24 |
             long(Rx_Data[11]) << 16 | long(Rx_Data[10]) << 8
             | long(Rx_Data[9]);
        break;
        case START_TEST:
             Start_Flag=1; // Setting Start Test Flag
             Stop_Flag=0;
             Power_Tranfer=1;
        break;
        case PAUSE:
             Resume_flag=0; // Setting Pause Test Flag
             Pause_flag=1;
             Power_Tranfer=0;
        break;
        case RESUME:
             Resume_flag=1; // Setting Resume Test Flag
             Pause_flag=0;
             Power_Tranfer=1;
        break;
        case STOP_TEST:
             Start_Flag=0;
             Stop_Flag=1;  // Setting Stop Test Flag
             Power_Tranfer=0;
        break;
        case SET_EXPERIMENT_PERIOD:
             // Merging four recieved bytes of Experiment
             // Period and storing in long variable
             Experiment_Time = long(Rx_Data[7]) << 24 |
             long(Rx_Data[6]) << 16 | long(Rx_Data[5]) << 8
             | long(Rx_Data[4]);
        break;
```

```
case GET_SELF_TEST_STATUS:
     Self_test_status=0;
     TX_Length=1;
     TX_DATA[0]=TX_Length;//Data_Length
     TX_DATA[1]=Self_test_status;
     CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
     CRC_TX_MSB = CRC_TX >> 8;
     CRC_TX_LSB = CRC_TX & 0x00ff;
     TX_DATA[TX_Length+1]=CRC_TX_MSB;
     TX_DATA[TX_Length+2]=CRC_TX_LSB;
break;
case ENTER_LAMP_CALIBRATION:
     for(x=0;x<RX_Length;x++,y++)
     {
         Bulb_Callibration[x]=Rx_Data[y];
     }
     x=0;
     y=4;
break;
case DEBUG_ON:
     Debug_ON_flag=1;
     Debug_OFF_flag=0;
break;
case DEBUG_OFF:
     Debug_ON_flag=0;
     Debug_OFF_flag=1;
break;
case TURN_OFF_POWER:
     Power_Tranfer=0;
break;
case SET_VOLTAGE_ON_POW_OUT:
     Channel_Number_v=Rx_Data[4];
     Voltage_mv= ( Rx_Data[5] << 8 ) | Rx_Data[6];
     //Calling HAL
     //Define Specific Function
break;
case SET_Current_ON_POW_OUT:
     Channel_Number_i=Rx_Data[4];
     Current_mA= ( Rx_Data[5] << 8 ) | Rx_Data[6];
     //Calling HAL
     //Define Specific Function
break;
case SET_TIME:
     Hours= ( Rx_Data[4] << 8 ) | Rx_Data[5];
     Minutes=( Rx_Data[6] << 8 ) | Rx_Data[7];
     Seconds= ( Rx_Data[8] << 8 ) | Rx_Data[9];
     //Calling HAL
     //Define Specific Function
break;
case GET_SUN_EXPOSURE_TIME:
     Sun_Exposure_Time=0xABCDEFDA;
     unsigned char *qs = (unsigned char*)&
     Sun_Exposure_Time;
     TX_Length=4;
     TX_DATA[0]=TX_Length;//Data_Length
     TX_DATA[1]=qs[0];
     TX_DATA[2]=qs[1];
     TX_DATA[3]=qs[2];
     TX_DATA[4]=qs[3];
     CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
     CRC_TX_MSB = CRC_TX >> 8;
     CRC_TX_LSB = CRC_TX & 0x00ff;
```

```
     TX_DATA[TX_Length+1]=CRC_TX_MSB;
     TX_DATA[TX_Length+2]=CRC_TX_LSB;
break;
case GET_ELAPSED_ORBITS:
     Elapsed_Orbits=867876;
     unsigned char *ts = (unsigned char*)&
     Elapsed_Orbits;
     TX_Length=4;
     TX_DATA[0]=TX_Length;//Data_Length
     TX_DATA[1]=ts[0];
     TX_DATA[2]=ts[1];
     TX_DATA[3]=ts[2];
     TX_DATA[4]=ts[3];
     CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
     CRC_TX_MSB = CRC_TX >> 8;
     CRC_TX_LSB = CRC_TX & 0x00ff;
     TX_DATA[TX_Length+1]=CRC_TX_MSB;
     TX_DATA[TX_Length+2]=CRC_TX_LSB;
break;
case GET_ELAPSED_EXPERIMENT_TIME:
     Elapsed_Time=12345;
     unsigned char *ms = (unsigned char*)&
     Elapsed_Time;
     TX_Length=4;
     TX_DATA[0]=TX_Length;//Data_Length
     TX_DATA[1]=ms[0];
     TX_DATA[2]=ms[1];
     TX_DATA[3]=ms[2];
     TX_DATA[4]=ms[3];
     CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
     CRC_TX_MSB = CRC_TX >> 8;
     CRC_TX_LSB = CRC_TX & 0x00ff;
     TX_DATA[TX_Length+1]=CRC_TX_MSB;
     TX_DATA[TX_Length+2]=CRC_TX_LSB;
     break;
     case GET_POWER_CONSUMPTION:
          Current_system_Power_mW=0xABCD;
          TX_Length=2;
          TX_DATA[0]=TX_Length;//Data_Length
          TX_DATA[1]=Current_system_Power_mW & 0xFF;
          TX_DATA[2]=Current_system_Power_mW >> 8;
          CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
          CRC_TX_MSB = CRC_TX >> 8;
          CRC_TX_LSB = CRC_TX & 0x00ff;
          TX_DATA[TX_Length+1]=CRC_TX_MSB;
          TX_DATA[TX_Length+2]=CRC_TX_LSB;
     break;
     case GET_HW_SW_SERIAL_NUMBER :
          HW_SW_SERIAL_NUMB=9568226;
          unsigned char *bs = (unsigned char*)&
          HW_SW_SERIAL_NUMB;
          TX_Length=4;
          TX_DATA[0]=TX_Length;//Data_Length
          TX_DATA[1]=bs[0];
          TX_DATA[2]=bs[1];
          TX_DATA[3]=bs[2];
          TX_DATA[4]=bs[3];
          CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
          CRC_TX_MSB = CRC_TX >> 8;
          CRC_TX_LSB = CRC_TX & 0x00ff;
          TX_DATA[TX_Length+1]=CRC_TX_MSB;
          TX_DATA[TX_Length+2]=CRC_TX_LSB;
```

```
break;
case ACQUIRE_ANALOG_VOLTAGE_CH1:
     voltage_ch1_mv=120;
     TX_Length=2;
     TX_DATA[0]=TX_Length;//Data_Length
     TX_DATA[1]=voltage_ch1_mv & 0xFF;
     TX_DATA[2]=voltage_ch1_mv >> 8;
     CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
     CRC_TX_MSB = CRC_TX >> 8;
     CRC_TX_LSB = CRC_TX & 0x00ff;
     TX_DATA[TX_Length+1]=CRC_TX_MSB;
     TX_DATA[TX_Length+2]=CRC_TX_LSB;
break;
case ACQUIRE_ANALOG_VOLTAGE_CH2:
     voltage_ch2_mv=240;
     TX_Length=2;
     TX_DATA[0]=TX_Length;//Data_Length
     TX_DATA[1]=voltage_ch2_mv & 0xFF;
     TX_DATA[2]=voltage_ch2_mv >> 8;
     CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
     CRC_TX_MSB = CRC_TX >> 8;
     CRC_TX_LSB = CRC_TX & 0x00ff;
     TX_DATA[TX_Length+1]=CRC_TX_MSB;
     TX_DATA[TX_Length+2]=CRC_TX_LSB;
     break;
     case ACQUIRE_ANALOG_VOLTAGE_CH3:
          voltage_ch3_mv=480;
          TX_Length=2;
          TX_DATA[0]=TX_Length;//Data_Length
          TX_DATA[1]=voltage_ch3_mv & 0xFF;
          TX_DATA[2]=voltage_ch3_mv >> 8;
          CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
          CRC_TX_MSB = CRC_TX >> 8;
          CRC_TX_LSB = CRC_TX & 0x00ff;
          TX_DATA[TX_Length+1]=CRC_TX_MSB;
          TX_DATA[TX_Length+2]=CRC_TX_LSB;
     break;
     case ACQUIRE_INTERNAL_TEMPERATURE:
          Internal_Temp=25;
          TX_Length=1;
          TX_DATA[0]=TX_Length;//Data_Length
          TX_DATA[1]=Internal_Temp;
          CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
          CRC_TX_MSB = CRC_TX >> 8;
          CRC_TX_LSB = CRC_TX & 0x00ff;
          TX_DATA[TX_Length+1]=CRC_TX_MSB;
          TX_DATA[TX_Length+2]=CRC_TX_LSB;
     break;
     case RESET:
          Debug_ON_flag=0;
          Debug_OFF_flag=0;
          Resume_flag=0;
          Pause_flag=0;
          Start_Flag=0;
          Stop_Flag=0;
     break;
     case ACQUIRE_CURRENT_ALL_CHANNELS:
          Current_ch1_mA=20;
          Current_ch2_mA=40;
          Current_ch3_mA=60;
          Current_ch4_mA=80;
          Current_ch5_mA=100;
```

```
      TX_Length=10;
      TX_DATA[0]=TX_Length;//Data_Length
      TX_DATA[1]=Current_ch1_mA & 0xFF;
      TX_DATA[2]=Current_ch1_mA >> 8;
      TX_DATA[3]=Current_ch2_mA & 0xFF;
      TX_DATA[4]=Current_ch2_mA >> 8;
      TX_DATA[5]=Current_ch3_mA & 0xFF;
      TX_DATA[6]=Current_ch3_mA >> 8;
      TX_DATA[7]=Current_ch4_mA & 0xFF;
      TX_DATA[8]=Current_ch4_mA >> 8;
      TX_DATA[9]=Current_ch5_mA & 0xFF;
      TX_DATA[10]=Current_ch5_mA >> 8;
      CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
      CRC_TX_MSB = CRC_TX >> 8;
      CRC_TX_LSB = CRC_TX & 0x00ff;
      TX_DATA[TX_Length+1]=CRC_TX_MSB;
      TX_DATA[TX_Length+2]=CRC_TX_LSB;
break;
case GET_LAMPS_LIFE:
      Lamp1_Life=12;
      Lamp2_Life=24;
      Lamp3_Life=36;
      Lamp4_Life=48;
      TX_Length=4;
      TX_DATA[0]=TX_Length;//Data_Length
      TX_DATA[1]=Lamp1_Life;
      TX_DATA[2]=Lamp2_Life;
      TX_DATA[3]=Lamp3_Life;
      TX_DATA[4]=Lamp4_Life;
      CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
      CRC_TX_MSB = CRC_TX >> 8;
      CRC_TX_LSB = CRC_TX & 0x00ff;
      TX_DATA[TX_Length+1]=CRC_TX_MSB;
      TX_DATA[TX_Length+2]=CRC_TX_LSB;
break;
case START_SELF_TEST:
      Start_self_test_flag=1;
break;
case GET_STATUS:
      STATUS_REG=0xABCD;
      TX_Length=2;
      TX_DATA[0]=TX_Length;//Data_Length
      TX_DATA[1]=STATUS_REG & 0xFF;
      TX_DATA[2]=STATUS_REG >> 8;
      CRC_TX= Calc_CRC_C(TX_DATA,TX_Length+1);
      CRC_TX_MSB = CRC_TX >> 8;
      CRC_TX_LSB = CRC_TX & 0x00ff;
      TX_DATA[TX_Length+1]=CRC_TX_MSB;
      TX_DATA[TX_Length+2]=CRC_TX_LSB;
break;
default:
// WRONG COMMAND
break;
}
Data_valid=0;
Command=0;
count=0;
RX_Counter=4;
RX_Length=0;
Length=0;
CRC_Cnt=0;
}
```

```
                }

}
//**********************CRC Calculation Function*****************//
short Calc_CRC_C(unsigned char *Buffer,unsigned short Len)
{
   short x;
   short crc = 0xFFFF;

   while(Len--)
   {
      x = ((crc >> (1*8)) & 0xff) ^ *Buffer++;

      x ^= x>>4;

      crc = (crc << 8) ^ (x << 12) ^ (x <<5) ^ x;
   }
   return crc;
}
```

Appendix C

Master.cpp:

Arduino Code for testing 1U Light Source.

```
//*********************************************************************//
//                                                                     //
//                          LOW COST SOLAR SIMULATOR                   //
//                                Version 1.1                          //
//                             Author:Sanwal Saleem                    //
//                         Supervisor:Prof. Leonardo Reyneri           //
//                                                                     //
//*********************************************************************//
#include <Wire.h>


/*****************************System Variabls*************************//
int i=0,j=0,cnt,Loop_Length;
int led = 7;
unsigned char Command_LSB,Command_MSB,Data_Length=15;
unsigned char CRC_LSB,CRC_MSB,RX_LENGTH=0,Internal_Temp;
unsigned char Lamp1_Life,Lamp2_Life,Lamp3_Life,Lamp4_Life;
unsigned short command=0x2BCD,CRC=0xEACD,Intensity=1300;
unsigned short voltage_mv=0,hours,minutes,seconds,REC_CRC;
unsigned short CAL_CRC,  Power_System_mw,Analog_Voltage_ch1;
unsigned short Analog_Voltage_ch2;
unsigned short Analog_Voltage_ch3,Current_ch1,Current_ch2;
unsigned short Current_ch3,Current_ch4,Current_ch5;
unsigned short STATUS_REGISTER,Current_mA;
unsigned char data[260],Command_Data_Op=0,Self_test_status=0;
short Calc_CRC_C(unsigned char *Buffer, unsigned short Len);
unsigned long CurrentPosition=122354,Experiment_Time=122354;
unsigned long Sun_expose_time1,Elapsed_Orbits,Elapsed_Time;
unsigned long HW_SW_SERIAL_NUM,illumination_period=12;
unsigned long duty_cycle=25,overall_time=22;
unsigned char *p = (unsigned char*)&illumination_period;
unsigned char *q = (unsigned char*)&overall_time;
unsigned char RX_DATA[256],RX_CNT=0;
unsigned char data_length;
unsigned char arr[4];
/*********************************Command Set***************************//
enum t_command
    {
        ILLUMINATE_THERMAL_BLACK = 0x30B1,
        ILLUMINATE_THERMAL_GREEN=0x30B3,
        ILLUMINATE_THERMAL_BLUE=0x30B2,
        ILLUMINATE_SINGLE_JUNCTION=0x30B5,
        ILLUMINATE_TRIPLE_JUNCTION=0x30B4,
        ILLUMINATE_VARIABLE=0x20CF,
        SET_LIGHT_PERIOD=0x20EE,
        START_TEST=0x30BB,
        PAUSE=0x30BD,
        RESUME=0x30C2,
        STOP_TEST=0x30BC,
        START_SELF_TEST=0x30B6,
        GET_SELF_TEST_STATUS=0x10B7,
        GET_STATUS=0x10B8,
        RESET=0x30BE,
```

```
        DEBUG_ON=0x30B9,
        DEBUG_OFF=0x30BA,
        SET_EXPERIMENT_PERIOD=0x20BF,
        GET_SUN_EXPOSURE_TIME=0x10C2,
        GET_ELAPSED_ORBITS=0x10C3,
        GET_ELAPSED_EXPERIMENT_TIME=0x10C4,
        GET_POWER_CONSUMPTION=0x10C5,
        GET_HW_SW_SERIAL_NUMBER=0x10C6,
        ACQUIRE_ANALOG_VOLTAGE_CH1=0x10C8,
        ACQUIRE_ANALOG_VOLTAGE_CH2=0x10C9,
        ACQUIRE_ANALOG_VOLTAGE_CH3=0x10CA,
        ACQUIRE_INTERNAL_TEMPERATURE=0x10CB,
        ACQUIRE_CURRENT_ALL_CHANNELS=0x10CC,
        TURN_OFF_POWER=0x30CD,
        ENTER_LAMP_CALIBRATION=0x20CE,
        SET_VOLTAGE_ON_POW_OUT=0x20D2,
        GET_LAMPS_LIFE=0x10D3,
        SET_Current_ON_POW_OUT=0x20D4,
        SET_TIME=0x20D5
    };


void setup()
{
    command=SET_TIME;
    Command_MSB=highByte(command);
    Command_LSB=lowByte(command);
    Command_Data_Op = (char) ((Command_MSB >> 4)
    & (char) 0x0F);
    if(Command_Data_Op==3) Data_Length=0;
    Loop_Length=Data_Length+4;
    data[0]=40;
    data[1]=Command_MSB;
    data[2]=Command_LSB;
    data[3]=Data_Length;
    j=0;
    for(i=4;i<(Loop_Length);i++,j++)
    {
        data[i]=j;
    }
    j=0;


    //SET TIME
    hours=2;
    minutes=15;
    seconds=35;
    data[4]=highByte(hours);
    data[5]=lowByte(hours);
    data[6]=highByte(minutes);
    data[7]=lowByte(minutes);
    data[8]=highByte(seconds);
    data[9]=lowByte(seconds);*/

    CRC=Calc_CRC_C(data,Loop_Length);

    CRC_MSB=highByte(CRC);
    CRC_LSB=lowByte(CRC);
    pinMode(7, OUTPUT);
    Wire.begin(0x20);
    Serial.begin(9600);
```

```
Serial.begin(9600);
Serial.println( CRC,HEX);
j=0;

Wire.beginTransmission(0x20);

for(i=0;i<=(Loop_Length+2);i++,j++)
{
     digitalWrite(led, HIGH);
     delay(1);
     digitalWrite(led, LOW);
     delay(1);
     if(i<=Loop_Length)   Wire.write(data[i]);
     if(i==Loop_Length+1) Wire.write(CRC_MSB);
     if(i==Loop_Length+2) Wire.write(CRC_LSB);
     Serial.println( i,DEC);
}
Wire.endTransmission();
delay(10000);


command=START_TEST;
Command_MSB=highByte(command);
Command_LSB=lowByte(command);
Command_Data_Op = (char) ((Command_MSB >> 4)
& (char) 0x0F);
if(Command_Data_Op==3) Data_Length=0;
Loop_Length=Data_Length+4;
data[0]=40;
data[1]=Command_MSB;
data[2]=Command_LSB;
data[3]=Data_Length;
j=0;
for(i=4;i<(Loop_Length);i++,j++)
{
    data[i]=j;
}
j=0;
CRC=Calc_CRC_C(data,Loop_Length);
CRC_MSB=highByte(CRC);
CRC_LSB=lowByte(CRC);
pinMode(7, OUTPUT);
j=0;
Wire.beginTransmission(0x20);
for(i=0;i<=(Loop_Length+2);i++,j++)
{
      digitalWrite(led, HIGH);
      delay(1);
      digitalWrite(led, LOW);
      delay(1);
      if(i<=Loop_Length)Wire.write(data[i]);
      if(i==Loop_Length+1) Wire.write(CRC_MSB);
      if(i==Loop_Length+2) Wire.write(CRC_LSB);
      Serial.println( i,DEC);
}
Wire.endTransmission();
delay(10000);

command=STOP_TEST;
Command_MSB=highByte(command);
Command_LSB=lowByte(command);
Command_Data_Op = (char)
```

```
((Command_MSB >> 4) & (char) 0x0F);
if(Command_Data_Op==3) Data_Length=0;
Loop_Length=Data_Length+4;
data[0]=40;
data[1]=Command_MSB;
data[2]=Command_LSB;
data[3]=Data_Length;
j=0;
for(i=4;i<(Loop_Length);i++,j++)
{
    data[i]=j;
}
j=0;
CRC=Calc_CRC_C(data,Loop_Length);
CRC_MSB=highByte(CRC);
CRC_LSB=lowByte(CRC);
pinMode(7, OUTPUT);
j=0;
Wire.beginTransmission(0x20);
for(i=0;i<=(Loop_Length+2);i++,j++)
{
    digitalWrite(led, HIGH);
    delay(1);
    digitalWrite(led, LOW);
    delay(1);
    if(i<=Loop_Length)Wire.write(data[i]);
    if(i==Loop_Length+1) Wire.write(CRC_MSB);
    if(i==Loop_Length+2) Wire.write(CRC_LSB);
    Serial.println( i,DEC);
}
Wire.endTransmission();


delay(10);
if(command==GET_SUN_EXPOSURE_TIME)
{
    Wire.requestFrom(0x20,7);
    while(Wire.available())
    {
          RX_DATA[RX_CNT] = Wire.read();
          Serial.println(RX_DATA[RX_CNT],HEX);
          RX_CNT++;
    }
}
RX_CNT=0;
if(command==GET_ELAPSED_ORBITS)
{
    Wire.requestFrom(0x20,7);
while(Wire.available())
{
    RX_DATA[RX_CNT] = Wire.read();
    Serial.println(RX_DATA[RX_CNT],HEX);
    RX_CNT++;
}
Elapsed_Orbits =  long(RX_DATA[1])
|( long(RX_DATA[2])<<8)|( long(RX_DATA[3])<<16)
|( long(RX_DATA[4])<<24);
Serial.print("Elapsed Orbits=");
Serial.println(Elapsed_Orbits,DEC);
RX_CNT=0;
}
```

```
  if(command==GET_ELAPSED_EXPERIMENT_TIME )
  {
    Wire.requestFrom(0x20,7);
    while(Wire.available())
    {
        RX_DATA[RX_CNT] = Wire.read();
        Serial.println(RX_DATA[RX_CNT],HEX);
        RX_CNT++;
    }
    RX_CNT=0;
    RX_LENGTH=RX_DATA[0];
    REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 )
    | RX_DATA[RX_LENGTH+2];
    Serial.print("Recieved CRC=");
    Serial.println(REC_CRC,HEX);
    CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
    Serial.print("Calculated CRC=");
    Serial.println(CAL_CRC,HEX);
    if(REC_CRC==CAL_CRC)
    {
       Elapsed_Time =  long(RX_DATA[1])|
       ( long(RX_DATA[2])<<8)|( long(RX_DATA[3])<<16)
       |( long(RX_DATA[4])<<24);
       Serial.print("Elapsed Timme=");
       Serial.println(Elapsed_Time,DEC);
    }
   else
    {
       Serial.print("CRC NOT MATCHED");
    }

}

if(command==GET_POWER_CONSUMPTION )
{
    Wire.requestFrom(0x20,5);
    while(Wire.available())
    {
        RX_DATA[RX_CNT] = Wire.read();
        Serial.println(RX_DATA[RX_CNT],HEX);
        RX_CNT++;
    }
    RX_CNT=0;
    RX_LENGTH=RX_DATA[0];
    REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 ) | RX_DATA[RX_LENGTH+2];
    Serial.print("Recieved CRC=");
    Serial.println(REC_CRC,HEX);
    CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
    Serial.print("Calculated CRC=");
    Serial.println(CAL_CRC,HEX);
    if(REC_CRC==CAL_CRC)
    {
        Power_System_mw= (  RX_DATA[2] << 8 )
        |  RX_DATA[1];
        Serial.print("System Power Consumption=");
        Serial.println(Power_System_mw,HEX);
    }
   else
    {
       Serial.print("CRC NOT MATCHED");
    }
```

```
}
 if(command==GET_HW_SW_SERIAL_NUMBER )
{
    Wire.requestFrom(0x20,7);
    while(Wire.available())
    {
        RX_DATA[RX_CNT] = Wire.read();
        Serial.println(RX_DATA[RX_CNT],HEX);
        RX_CNT++;
    }
    RX_CNT=0;
    RX_LENGTH=RX_DATA[0];
    REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 )
    | RX_DATA[RX_LENGTH+2];
    Serial.print("Recieved CRC=");
    Serial.println(REC_CRC,HEX);
    CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
    Serial.print("Calculated CRC=");
    Serial.println(CAL_CRC,HEX);
    if(REC_CRC==CAL_CRC)
    {
       HW_SW_SERIAL_NUM =  long(RX_DATA[1])
       |( long(RX_DATA[2])<<8)|( long(RX_DATA[3])<<16)
       |( long(RX_DATA[4])<<24);
       Serial.print("HW_SW_SERIAL_NUMBER=");
       Serial.println(HW_SW_SERIAL_NUM,DEC);
    }
   else
    {
       Serial.print("CRC NOT MATCHED");
    }

}

  if(command==ACQUIRE_ANALOG_VOLTAGE_CH1)
{
    Wire.requestFrom(0x20,5);
    while(Wire.available())
    {
        RX_DATA[RX_CNT] = Wire.read();
        Serial.println(RX_DATA[RX_CNT],HEX);
        RX_CNT++;
    }
    RX_CNT=0;
    RX_LENGTH=RX_DATA[0];
    REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 )
    | RX_DATA[RX_LENGTH+2];
    Serial.print("Recieved CRC=");
    Serial.println(REC_CRC,HEX);
    CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
    Serial.print("Calculated CRC=");
    Serial.println(CAL_CRC,HEX);
    if(REC_CRC==CAL_CRC)
    {
       Analog_Voltage_ch1= (  RX_DATA[2] << 8 )
       |  RX_DATA[1];
       Serial.print("Voltage on CH1(mv)=");
       Serial.println(Analog_Voltage_ch1,DEC);
    }
    else
    {
       Serial.print("CRC NOT MATCHED");
```

```
    }

}
 if(command==ACQUIRE_ANALOG_VOLTAGE_CH2)
{
    Wire.requestFrom(0x20,5);
    while(Wire.available())
    {
        RX_DATA[RX_CNT] = Wire.read();
        Serial.println(RX_DATA[RX_CNT],HEX);
        RX_CNT++;
    }
    RX_CNT=0;
    RX_LENGTH=RX_DATA[0];
    REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 )
    | RX_DATA[RX_LENGTH+2];
    Serial.print("Recieved CRC=");
    Serial.println(REC_CRC,HEX);
    CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
    Serial.print("Calculated CRC=");
    Serial.println(CAL_CRC,HEX);
    if(REC_CRC==CAL_CRC)
    {
       Analog_Voltage_ch2= (  RX_DATA[2] << 8 )
       |  RX_DATA[1];
       Serial.print("Voltage on CH2(mv)=");
       Serial.println(Analog_Voltage_ch2,DEC);
    }
   else
    {
       Serial.print("CRC NOT MATCHED");
    }

}

 if(command==ACQUIRE_ANALOG_VOLTAGE_CH3)
{
    Wire.requestFrom(0x20,5);
    while(Wire.available())
    {
        RX_DATA[RX_CNT] = Wire.read();
        Serial.println(RX_DATA[RX_CNT],HEX);
        RX_CNT++;
    }
    RX_CNT=0;
    RX_LENGTH=RX_DATA[0];
    REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 )
    | RX_DATA[RX_LENGTH+2];
    Serial.print("Recieved CRC=");
    Serial.println(REC_CRC,HEX);
    CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
    Serial.print("Calculated CRC=");
    Serial.println(CAL_CRC,HEX);
    if(REC_CRC==CAL_CRC)
    {
        Analog_Voltage_ch3= (  RX_DATA[2] << 8 )
        |  RX_DATA[1];
        Serial.print("Voltage on CH3(mv)=");
        Serial.println(Analog_Voltage_ch3,DEC);
    }
   else
    {
```

```
        Serial.print("CRC NOT MATCHED");
      }

 }
if(command==ACQUIRE_INTERNAL_TEMPERATURE)
 {
     Wire.requestFrom(0x20,4);
     while(Wire.available())
     {
         RX_DATA[RX_CNT] = Wire.read();
         Serial.println(RX_DATA[RX_CNT],HEX);
         RX_CNT++;
     }
     RX_CNT=0;
     RX_LENGTH=RX_DATA[0];
     REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 )
     | RX_DATA[RX_LENGTH+2];
     Serial.print("Recieved CRC=");
     Serial.println(REC_CRC,HEX);
     CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
     Serial.print("Calculated CRC=");
     Serial.println(CAL_CRC,HEX);
     if(REC_CRC==CAL_CRC)
     {
         Internal_Temp=RX_DATA[1];
        Serial.print("Internal Temperature=");
        Serial.println(Internal_Temp,DEC);
     }
    else
     {
        Serial.print("CRC NOT MATCHED");
     }

 }

   if(command==ACQUIRE_CURRENT_ALL_CHANNELS )
 {   delay(500);
     Wire.requestFrom(0x20,13);
     while(Wire.available())
     {
         RX_DATA[RX_CNT] = Wire.read();
         Serial.println(RX_DATA[RX_CNT],HEX);
         RX_CNT++;
     }
     RX_CNT=0;
     RX_LENGTH=RX_DATA[0];
     REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 )
     | RX_DATA[RX_LENGTH+2];
     Serial.print("Recieved CRC=");
     Serial.println(REC_CRC,HEX);
     CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
     Serial.print("Calculated CRC=");
     Serial.println(CAL_CRC,HEX);
     if(REC_CRC==CAL_CRC)
     {
        Current_ch1= (  RX_DATA[2] << 8 ) |  RX_DATA[1];
        Current_ch2= (  RX_DATA[4] << 8 ) |  RX_DATA[3];
        Current_ch3= (  RX_DATA[6] << 8 ) |  RX_DATA[5];
        Current_ch4= (  RX_DATA[8] << 8 ) |  RX_DATA[7];
        Current_ch5= (  RX_DATA[10] << 8 ) |  RX_DATA[9];
        Serial.print("Current on Ch1=");
        Serial.println(Current_ch1,DEC);
```

```
            Serial.print("Current on Ch2=");
            Serial.println(Current_ch2,DEC);
            Serial.print("Current on Ch3=");
            Serial.println(Current_ch3,DEC);
            Serial.print("Current on Ch4=");
            Serial.println(Current_ch4,DEC);
            Serial.print("Current on Ch5=");
            Serial.println(Current_ch5,DEC);
        }
       else
        {
            Serial.print("CRC NOT MATCHED");
        }

    }
   if(command==GET_LAMPS_LIFE)
    {
        Wire.requestFrom(0x20,7);
        while(Wire.available())
        {
            RX_DATA[RX_CNT] = Wire.read();
            Serial.println(RX_DATA[RX_CNT],HEX);
            RX_CNT++;
        }
        RX_CNT=0;
        RX_LENGTH=RX_DATA[0];
        REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 ) | RX_DATA[RX_LENGTH+2];
        Serial.print("Recieved CRC=");
        Serial.println(REC_CRC,HEX);
        CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
        Serial.print("Calculated CRC=");
        Serial.println(CAL_CRC,HEX);
        if(REC_CRC==CAL_CRC)
        {
            Lamp1_Life=RX_DATA[1];
            Lamp2_Life=RX_DATA[2];
            Lamp3_Life=RX_DATA[3];
            Lamp4_Life=RX_DATA[4];
           Serial.print("Lamp 1 Remaining Life(Years)=");
           Serial.println(Lamp1_Life,DEC);
           Serial.print("Lamp 2 Remaining Life(Years)=");
           Serial.println(Lamp2_Life,DEC);
           Serial.print("Lamp 3 Remaining Life(Years)=");
           Serial.println(Lamp3_Life,DEC);
           Serial.print("Lamp 4 Remaining Life(Years)=");
           Serial.println(Lamp4_Life,DEC);
        }
       else
        {
           Serial.print("CRC NOT MATCHED");
        }

    }
  if(command==GET_STATUS)
    {
        Wire.requestFrom(0x20,5);
        while(Wire.available())
        {
            RX_DATA[RX_CNT] = Wire.read();
            Serial.println(RX_DATA[RX_CNT],HEX);
            RX_CNT++;
        }
```

```
        RX_CNT=0;
        RX_LENGTH=RX_DATA[0];
        REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 ) |
         RX_DATA[RX_LENGTH+2];
        Serial.print("Recieved CRC=");
        Serial.println(REC_CRC,HEX);
        CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
        Serial.print("Calculated CRC=");
        Serial.println(CAL_CRC,HEX);
        if(REC_CRC==CAL_CRC)
        {
            STATUS_REGISTER= (  RX_DATA[2] << 8 ) |  RX_DATA[1];
           Serial.print("STATUS_REGISTER=");
           Serial.println(STATUS_REGISTER,HEX);
        }
       else
        {
           Serial.print("CRC NOT MATCHED");
        }

   }
   if(command==GET_SELF_TEST_STATUS)
   {
        Wire.requestFrom(0x20,4);
        while(Wire.available())
        {
            RX_DATA[RX_CNT] = Wire.read();
            Serial.println(RX_DATA[RX_CNT],HEX);
            RX_CNT++;
        }
        RX_CNT=0;
        RX_LENGTH=RX_DATA[0];
        REC_CRC=( RX_DATA[RX_LENGTH+1] << 8 ) | RX_DATA[RX_LENGTH+2];
        Serial.print("Recieved CRC=");
        Serial.println(REC_CRC,HEX);
        CAL_CRC=Calc_CRC_C(RX_DATA,RX_LENGTH+1);
        Serial.print("Calculated CRC=");
        Serial.println(CAL_CRC,HEX);
        if(REC_CRC==CAL_CRC)
        {
            Self_test_status=RX_DATA[1];
            if(Self_test_status==1)
            {
            Serial.println("Self Test Completed");
            }
            if(Self_test_status==0)
            {
            Serial.println("Self Test Running....");
            }


        }
       else
        {
           Serial.print("CRC NOT MATCHED");
        }

   }


}
```

```
void loop()
{

Serial.println("************************************************************"
);
delay(5000);


}

short Calc_CRC_C(unsigned char *Buffer, unsigned short Len)
{
   short x;
   short crc = 0xFFFF;

   while(Len--)
   {
      x = ((crc >> (1*8)) & 0xff) ^ *Buffer++;

      x ^= x>>4;

      crc = (crc << 8) ^ (x << 12) ^ (x <<5) ^ x;
   }
   return crc;
}
```