2017

A HOST BOARD DESIGN FOR AN EXPERIMENT SYSTEM OF UNDERWATER WIRELESS SENSOR NETWORKS

He Huang
Michigan Technological University, heh@mtu.edu

A HOST BOARD DESIGN FOR AN EXPERIMENT SYSTEM OF UNDERWATER WIRELESS SENSOR NETWORKS

By

He Huang

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2017

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Electrical Engineering.

Department of Electrical and Computer Engineering

Report Advisor:	*Dr. Zhaohui Wang*
Committee Member:	*Dr. Michael C. Roggemann*
Committee Member:	*Dr. Warren F. Perger*
Department Chair:	*Dr. Daniel R. Fuhrmann*

Table of Contents

List of figures

List of tables

Acknowledgements

First, I would like to thank my advisor Dr. Zhaohui Wang for patient guidance and supporting during my master study. She always gave me useful help and advice when I was working on the research. What's more, she sets a good example of woman engineer scholar for me, so I have more courage to move on.

I would like to thank Mr. Li Wei. I am very appreciating his sincere help in this year. He is the one who encourages me to do this research, and his professional research spirit deeply influences me.

Last but not least, I would like to express my special thank for my family and friends. Without their intimate help, I could not finish my research and have a happy life in the US. Especially for my fiancé Mr. Bin Zhou, thank him for accompanying during my master study.

Abstract

Evaluation of theoretical innovation in field experiments plays an important role for research in Underwater Wireless Sensor Networks (UWSNs). During the experiments, the problems of power consumption, costs, and assembly of underwater nodes concern researchers all the time. This project develops a host board for an UWSNs field experiment system. This host board solves the above problems by deploying power management module, utilizing cheap and low power consumption chips (Raspberry Pi and MSP430), and rationalizing layout.

The main functions of the host board include battery monitoring, on-board DC converters, remotely power supply control, data ports bridge, and pluggable module for power supply and data ports. Experiments performed after the project prove that the host board could perfectly adapt to the underwater environment and also is applicable for many other field experiment systems.

1 Introduction

Underwater wireless sensor networks (UWSNs) will play an important role in future ocean explorations [1]-[3]. The applications of UWSNs spread over industry, military, and science research [4], such as coastal surveillance systems [5], mine detection [1], and environmental research [6], [7]. UWSNs could be deployed in adverse oceanographic environments to accomplish tasks, such as unmanned underwater exploration, localized and precise knowledge acquisition, and large-scale underwater monitoring [8]. Given typically long-time deployment of UWSNs, energy efficiency has been a critical consideration during the USWNs development. The practical current energy supply solution for the UWSNs nodes is always by batteries. When a node runs out of its power, researchers need to replace the dead battery by going to the site using the boat and pulling the equipment from the water, which is costly and time-consuming [1]. Beyond that, the transmission power is a trouble for UWSNs. Compared to terrestrial radio communications, the underwater acoustic communications consumes much higher energy for long-distance transmission [9]. Ways to solve these two challenges will involve reducing and monitoring power consumption.

There are a number of UWSN systems developed for different uses. SUNRISE and SUNSET are systems for monitoring underwater environments [10], [11]. SeaWeb is used by US Navy [12]. Ocean-TUNE is the system which has been developed by several universities for academic research, especially underwater communications [6], [13]-[15]. Most of these systems are open sources and use advanced technologies, which are also compatible with other kinds of systems. Besides the underwater wireless sensing networks, there also exist cabled observatories, such as the the NEPTUNE [16], the MARS [17], and the OOI [18].

In this report, a host board is designed by utilizing low power consumption MCU (Raspberry Pi Zero and MSP430) for an UWSN field experiment system, which provides data port converter function, power management function, and reliable assembly function. The rest of the report is organized as follows. Chapter 2 introduces the constraints of the system design. Chapter 3 provides an overview of Raspberry Pi. Chapter 4, 5, 6, and 7 will illustrate the detail of design, the MCU programming, and the assembly plan. Conclusions are drawn in Chapter 8.

2 UWSN Experiment System Overview

2.1 System Architecture

An UWSN for field experiments is the experimental platform for testing performances of underwater communication systems or conducting underwater sensor network experiments with various types of sensors. The entire testbed system for the USWN field consists of four surface nodes and a remote control center on the shore or a ship. As shown in Fig. 1, each surface node is held by a kayak, and is equipped with one acoustic modem, a radio frequency (RF) modem, and a host board. The housing of host board is waterproof, which can be cuboid or cylinder (Fig. 2). Among with an RF antenna, the RF modem at each node enables in-air wireless information transmission with the remote control center. The acoustic modems deployed below kayaks form an underwater acoustic network. Besides the acoustic modem and the RF modem, each node can have three more peripherals, such as GPS, sensors, and other devices. This testbed is designed for both offline and online evaluation of communication algorithms and network protocols in UWSNs.

Figure 1 Surface nodes

Figure 2 Cylinder housing

2.2 System Architecture of an UWSN Node

An UWSN node could mainly consist of a controller unit, i.e. Raspberry Pi, a RF communication system, underwater communication modems, sensors, GPS, etc. [1]. The operating system and the experiment software will be running on the controller unit. The RF communication system will enable researchers remotely to control the UWSN node via RF links above the water surface, but sometimes a UWSN node may be designed as an underwater node which doesn't have any RF communication capabilities [16], [17]. The underwater communication modems, such as acoustic or optical modems, are the key components to be tested in an experiment. Sensors, GPS, and other devices are optional in a UWSN node, and they could vary a lot in different experiments for various application scenarios.

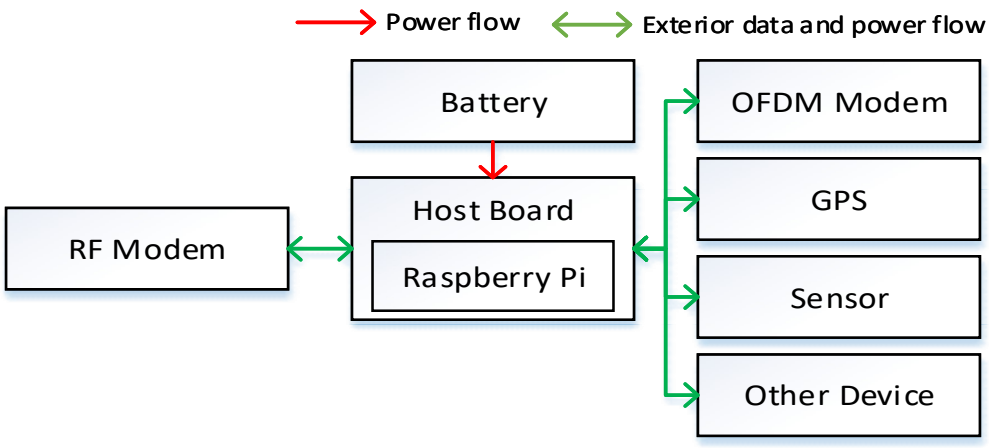

Figure 3 An UWSN node system architecture

As shown in Fig. 3, the host board is the core component of a UWSN node. All devices will be powered by a battery which is directly connected to the host board. After converting to proper voltage levels, power supply for each device will be provided by power ports on the host board. Data communication between Raspberry Pi and other devices will also be converted and bridged by the host board. The host board should be able to accommodate multiple devices with diverse types of power and data port demands.

2.3 Functionalities of the Host Board

The main objectives of designing this host board are as following:

1) Implementing power management functions for a field experiment platform, including:
    a. Converting battery voltage to 3.3V, 5V, and 12V for different devices.
    b. Remotely turning on/off the power supply of each device.
    c. Monitoring battery voltage level.
2) Providing proper data port converters between a Raspberry Pi Zero and other peripherals of a UWSN node.
3) Provide convenient connectors for reliable wiring works.

2.3.1 Power Management

Simple power management functions, such as battery level monitoring and remotely turning on/off the power supply of a specific device, could be highly demanded for an underdeveloped experimental system in a field experiment. These power management functions will save precious deploy time of an experiment in a field environment. For instance, in a sea test deployment of an UWSN system, a manually rebooting process of a deployed UWSN node could take about one hour for recovering and redeploying the mooring system of the node, let alone the voyage time of sailing to the node. Therefore, it

is necessary to program Raspberry Pi that allows researchers to remotely turn on/off the devices on the shore.

2.3.2 Data Ports

For field experiment systems, such as UWSN systems, the I/O interfaces of Raspberry Pi are not always compatible with other equipment's interfaces, i.e. most underwater sensors and acoustic modems are using RS232 or RS485 to communicate with their superior machines. Although one could always use USB adapters to convert and connect the serial connection between a Raspberry Pi and another equipment, the battery of a field experiment is limited and could be easily drained by these power-hungry USB devices.

2.3.3 Wiring Works

Reliable wiring works are always necessary for a successful field experiment. Wire connectors and device layouts need to be carefully designed. For an UWSN field experiment system, electronic devices are usually mounted in waterproof housings and hosted by a buoy or tied to a cable of a mooring system. Thus, all devices will be shaken by water current all the time, and unreliable wire connectors may cause loose connection between circuit boards and devices. The space within a waterproof housing is limited, device layouts need to take the spatial limitation into consideration, as well as the location of penetrative connectors of the housing.

Thus, a host board for Raspberry Pi Zero with properly designed connectors, which also provides data port bridges/converters and power management functions, could be a solution to accelerate the development of UWSN systems, as well as other wireless sensor network systems.

3 Host Board System Description

In this section, most of the demanding features will be discussed in detail. The suggested implementation of some functions is also advocated in corresponding subsections.

3.1 Overview of the Host Board System Architecture

Based on the aforementioned functions, besides a Raspberry Pi board, there are two types of subsystems on the host board, namely the power management module and data port converter module. The system architecture of the host board is as shown in Fig. 4.

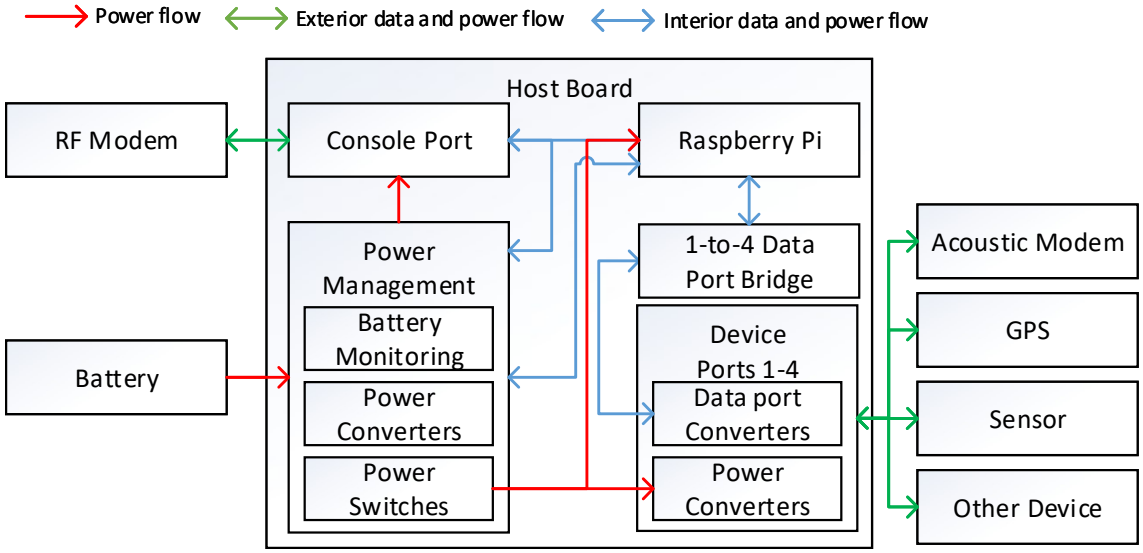

Figure 4 The host board system architecture

As the Raspberry Pi is the CPU of a UWSN node, it will receive commands from a user via its console port and operate other devices on the UWSN node through several other data ports. The console port of Raspberry Pi is usually connected to the RF modem to communicate with a remote user. Commands received by Raspberry Pi's console port will be directly executed by the operating system on the Raspberry Pi. Other devices, such as acoustic modem and GPS, etc., could be operated by experiment programs running on Raspberry Pi via other bridges and converted data ports.

The power management subsystem will be in charge of the aforementioned power management functions, including battery level monitoring, voltage converting, and control of power supply for each device by MSP430. The real-time battery level monitoring could allow users to estimate the lifetime of a specific UWSN node during an experiment. The voltage converting modules will convert the unstable battery voltage to demanding voltage levels for chips and devices within the system, i.e. 3.3V for chips, 5V for Raspberry Pi, 12V for other devices, etc.

Power management subsystem will take commands from 2 channels: the console port of the host board and the Raspberry Pi, and power supply of Raspberry Pi and other devices which connect to device ports could be turned off/on by these commands. In this case, Raspberry Pi or other devices can be rebooted when encountering a software failure. Also, some unnecessary device could be turned off to save the energy during a long-term deployment. Therefore, console port has the highest priority to make decisions.

3.2 Raspberry Pi Zero

Since students have plenty of chances to get familiar with Raspberry Pi before they start working on a research project, and abundant resources about developing Raspberry Pi based system can be found online in forum of Raspberry Pi communities, an increasing number of research groups start to choose Raspberry Pi as the microcontroller unit for developing their experimental systems. One Raspberry Pi board only costs between $5-50, which makes it relatively expendable for students to play with during their learning stage.

As a powerful single-board computer system, Raspberry Pi has achieved a great success for teaching basic computer science in schools. A Raspberry Pi system usually has an ARM core CPU, 1GB RAM, SD card storage and running a Linux core operating system namely RASPBIAN. There are multiple types of I/O interfaces available on different models of Raspberry Pi. For example, Raspberry Pi 3 has 1 Ethernet port, 4 USB ports, and 1 HDMI port, etc., which makes Raspberry Pi 3 very convenient for teaching or general uses. While, Raspberry Pi Zero only keeps minimal interfaces and the 40-Pin GPIO ports, which makes it smaller and more power efficient for embedded projects [19], [20].

3.3 Power Management Functions of the Host Board

Power management functions for a field experiment platform, including:

1) Converting battery voltage to 3.3V, 5V, and 12V for different devices.
2) Remotely turning on/off the power supply of each device.
3) Monitoring battery voltage level.

3.3.1 On-board DC Converters

The battery employed by a UWSN node may vary in different scenarios. Battery voltage could also change with the remaining energy level. The battery model used in the project is LiNiMnCo 26650 Battery Pack, who's operating voltage is from 14.8V to 16.8V [20]. Moreover, chips and devices work at different voltage levels, i.e. CMOS chips need a 3.3V power supply, and Raspberry Pi is powered by 5V TTL voltage level, while RF modem and OFDM acoustic modem are working at 12V. Thus, there should be multiple DC converters stabilizing and converting battery voltage to 3.3V, 5V, and 12V. The

corresponding circuits are listed in Appendix A1, from sheet 9 to 11. In sequence, they are 12V, 5V, and 3.3V step-down converters.

3.3.2 Remotely Power Supply Control Functions

The remote power supply control functions include remotely shutting down and rebooting a device by commands, which will help a UWSN node save energy and resolve software crash down. The remote power supply control functions could be implemented by employing an MSP430 chip to control several switch modules. MSP430 will receive commands from console port, then decide to turn on/off which device. Due to MSP430 is the only controller that need to process all the time, it is better to let it resets every period for normal operation. Since program running on MSP430 will automatically reset every 30s, a set of buffers will be needed to stabilize the control signal of switch modules, as showing in Fig.5. In addition, Raspberry Pi Zero can also control the power supply by changing the condition of MUX. Users can remotely write code on RASPBIAN, then directly turn on/off any device.

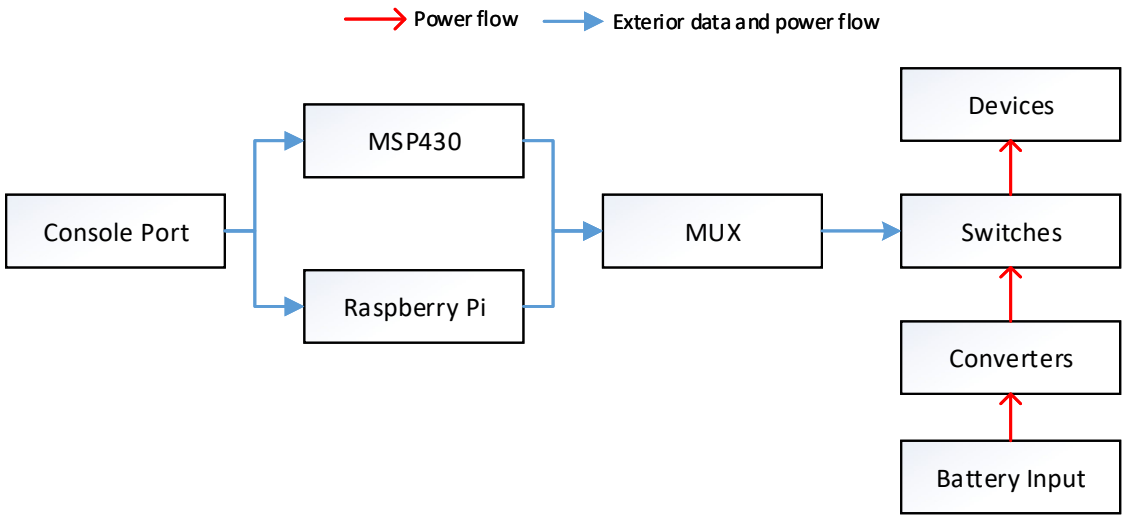

Figure 5 Remote power control subsystem

3.3.3 Battery Monitoring

Monitoring battery voltage level can help estimating the remaining lifetime of the deployed system, which will let researchers modify the experiment schedule to suit the situations they encountered. The battery monitoring function could be implemented by a voltage divider and an ADC module. Raspberry Pi could obtain the estimation of battery voltage by querying the ADC module via its SPI or I2C port. The real ADC circuit is shown in Appendix A1 sheet 4.

3.4 Data Ports of Raspberry Pi Zero

Most experiments will be driven by programs running on a Linux-based operating system RASPBIAN in the Raspberry Pi. As showing in Fig. 6, Raspberry Pi Zero employs a 1GHz 32-bit ARM1176JZF-S single-core ARMv6 processor, 512MB SDRAM which is also shared with GPU, a micro SD card as a hard drive, a micro USB port, a micro HDMI port, and a HAT-compatible 40-Pin header [19], [20].

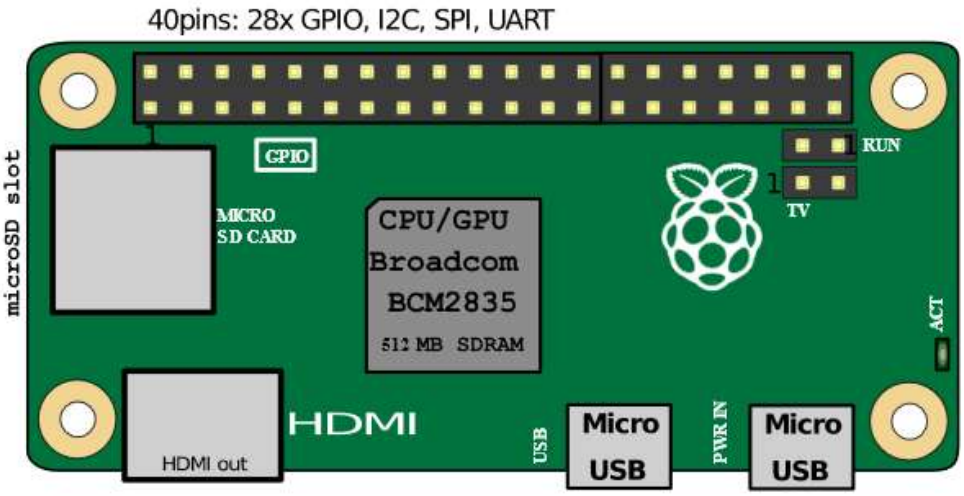

Figure 6 Raspberry Pi Zero Layout

The pinouts of the 40-Pin header are listed in Appendix B, including a UART port, a SPI and an I2C bus. The UART port will be utilized as the console port. The I2C port will be used for communication with the Raspberry Pi and ADC chip. Another 7 GPIO pins will responsible for controlling the power of all peripherals and monitor other chips.

3.4.1 UART Port

Besides the ground, pins of UART port are shown as follow:

Table 1 UART port pins of Raspberry Pi Zero

Pin Number	Function
8	TXD
10	RXD

This UART port can be configured as the console port of the Raspberry Pi. Thus, this UART port will need to be connected to RF modem. However, in order to resolve crash down of the operating system, the RF modem need to communicate with the MSP430 which controls the power supply of Raspberry Pi. The connection among RF modem, Raspberry Pi, and MSP430 is as shown in Fig. 7.

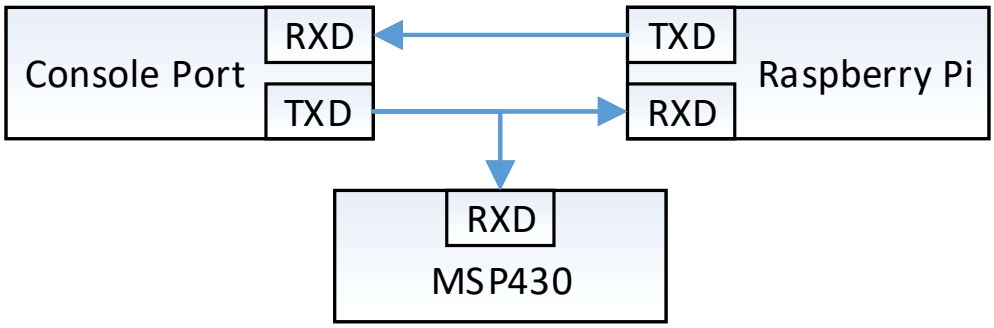

Figure 7 Connections of serial ports for RF modem, Raspberry Pi, and MSP430

To simplify the configurations on both superior machine side and the UWSN node, commonly used RF modems are usually configured as transparent serial port devices. Thus, RF modem on the UWSN node may not be able to specify which device it will talk to through its serial port. With the connection configuration shown in Fig. 7, MSP430 will overhear all messages sent from the RF modem and try to parse them every 30s, and then clear the buffer. If the messages are for Raspberry Pi, MSP430 won't be able to parse them and will just ignore them.

3.4.2 I2C Port

Besides the ground, pins of the I2C port are shown as follow:

Table 2 I2C pins of Raspberry Pi Zero

Pin Number	**Function**
3	SDA: serial data
5	SCL: serial clock

Raspberry Pi will communicate with the ADC chip via the I2C port to monitor the battery voltage level.

3.4.3 GPIO Port

Besides the ground, the GPIO pins which are used in the design are as follow:

Table 3 Typical GPIO pins of Raspberry Pi Zero

Pin Number	**Function**
29	GPIO 5
31	GPIO 6
33	GPIO 13
35	GPIO 19
36	GPIO 16
37	GPIO 26
38	GPIO 20
40	GPIO 21

For the GPIO pins shown beyond, they are responsible for power supply control of 4 peripherals. Pin 29, 31, 33, and 35 will correspondingly control the power supply switch for the 4 peripherals. Pin 36 and 37 can active the latch for changing the status of output. Pin 38 could reset chip FT4232, in case of the software crash. Pin 40 is the key to transfer the authorization of MUX. When pin 40 output is high, MUX would allow Raspberry Pi to control the power switches, otherwise, MSP430 will take charge of them.

3.4.4 USB and FT4232 Module

Since a USB device can automatically install its driver and create device port files in the operating system, a FT4232 module will be employed to branch the micro USB port on Raspberry Pi into 4 UART ports for equipment such as acoustic modem, GPS, etc.

3.5 Pluggable Modules on the Host Board

Since the types of data port, as well as the voltage of demanding power supply, of the equipment connecting to the host board may vary in different experiments, there are two types of pluggable modules need to be designed for the console port and 4 device ports of this host board. The socket connector for different data port pluggable modules or power converter modules with different output voltage levels need to be compatible with each other.

3.5.1 Pluggable Serial Port Module

With the help of FT4232H module, there will be 4 UART ports available for Raspberry Pi to communicate with other equipment. However, most field experiment equipment employ field bus ports such as RS232 or RS485. Thus, pluggable serial communication converters, such as UART-to-RS232 and UART-to-RS485, will be needed for the console port and device ports of this host board. The serial port converter module will be powered by the host board, which circuits are shown in Fig. 8 and Fig. 9. Their pins connections are corresponding, that means the pluggable serial communication converter module can be replaced easily without changing the host board circuit. The inputs and outputs of the serial port module could be:

Table 4 The Input and Outputs of Pluggable Serial Port Module

Input (UART port)	PWR (3.3V, 5V, or 12V), GND, RXD, TXD.
Output (Other serial port)	4 pins for RS232, 6 pins for RS485, etc.

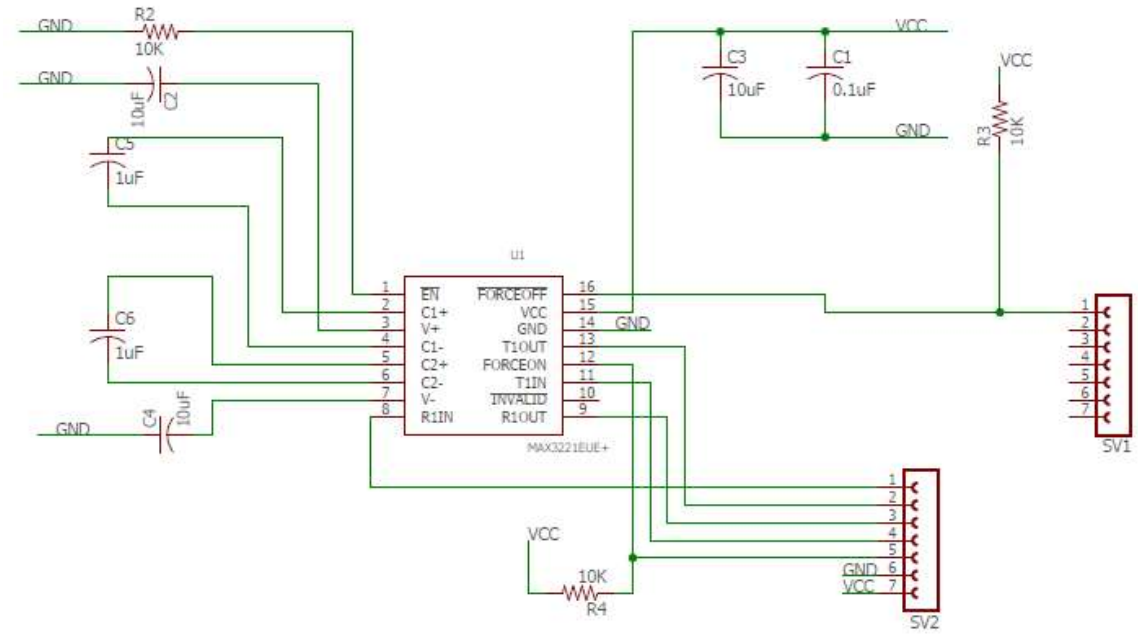

Figure 8 UART to RS232 converter circuit

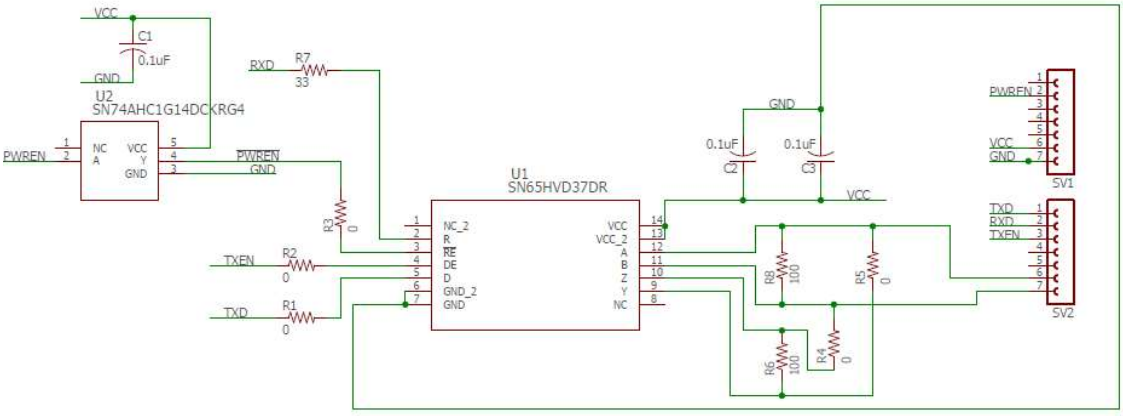

Figure 9 UART to RS485 converter circuit

3.5.2 Pluggable DC Converter Module

For simplified design and space saving, I designed a circuit that allows users to use pluggable pin hat to choose appropriate voltage level for different peripherals. In Fig. 10, all three DC converters are connected to the 4 2X3 pin headers. Each header is responsible for corresponding power supply of the peripherals. Users can choose different voltage levels by connecting the left side with the right side of pin headers using pin hats.

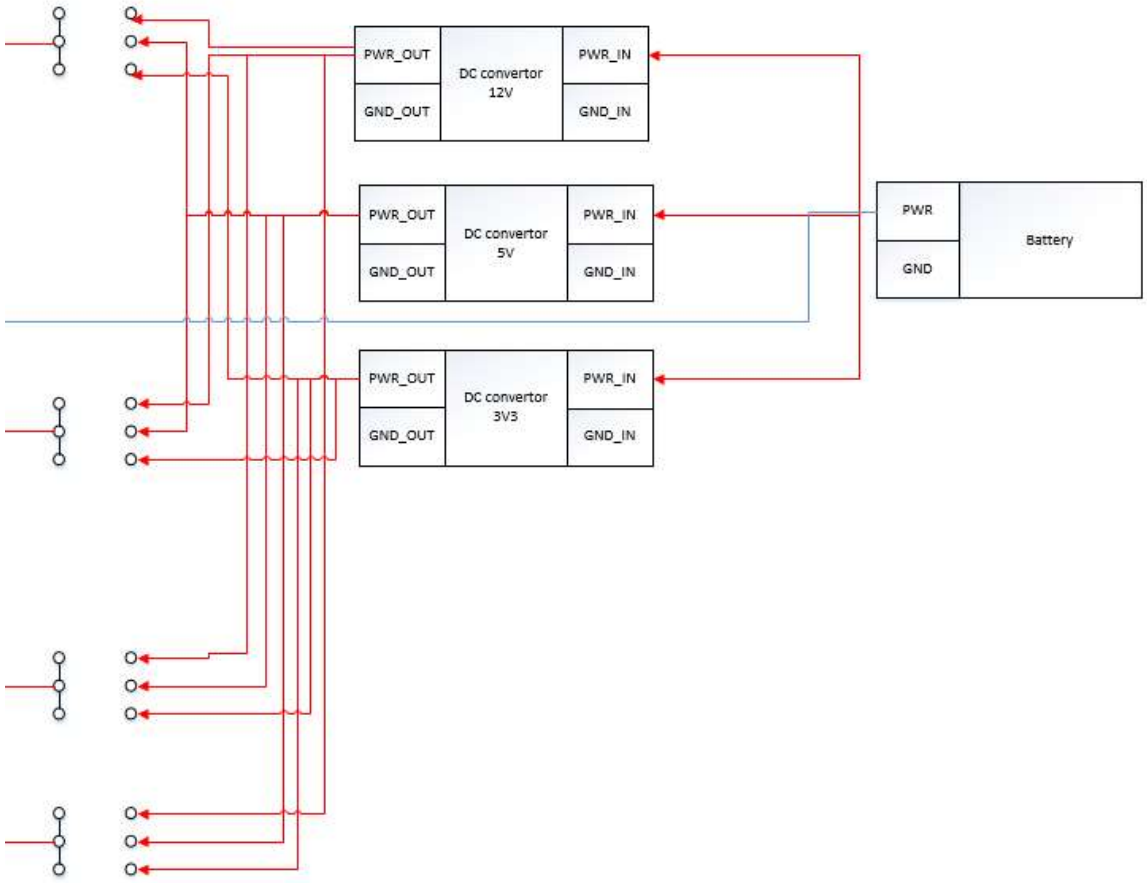

Figure 10 Principle of pluggable DC converter module

4 Overview of the Host Board Circuit Design

This chapter will list detail chips information for host board circuit design. In Appendix A, the overall circuits of host board are listed. As shown in Fig. 11, besides Raspberry Pi and MSP430, many other chips are used for distinct functions. The basic chips on the host board include transceivers, latch, buffers, switches, ADC, MUX, and voltage converters. In the next section, every major chip will be mentioned, and with their detail functions.

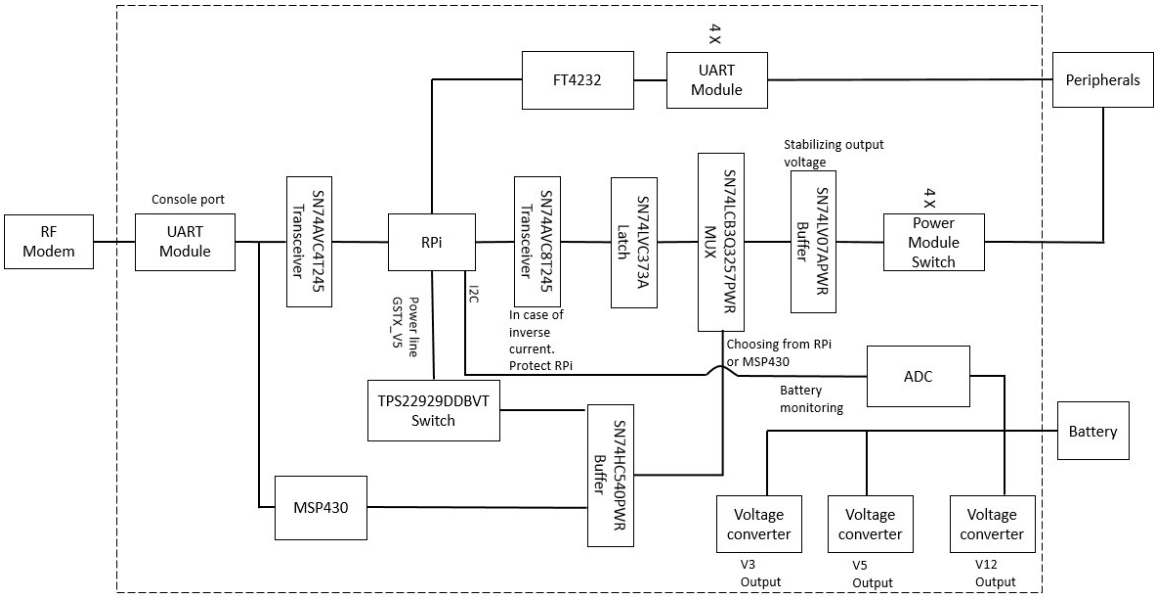

Figure 11 Raspberry Pi chip interaction map

4.1 MSP430

The MSP4302203 is a low power consumption MCU. It has several low-power modes, which rated current could be less than 1uA. This chip is the core of power management module for the host board. Unlike Raspberry Pi, as long as the experiment is going on, MSP430 will be turned on all the time. Both MSP430 and Raspberry Pi could control the power supply of four peripherals, but MSP430 takes more responsibility for keeping the system work well (Fig. 12).

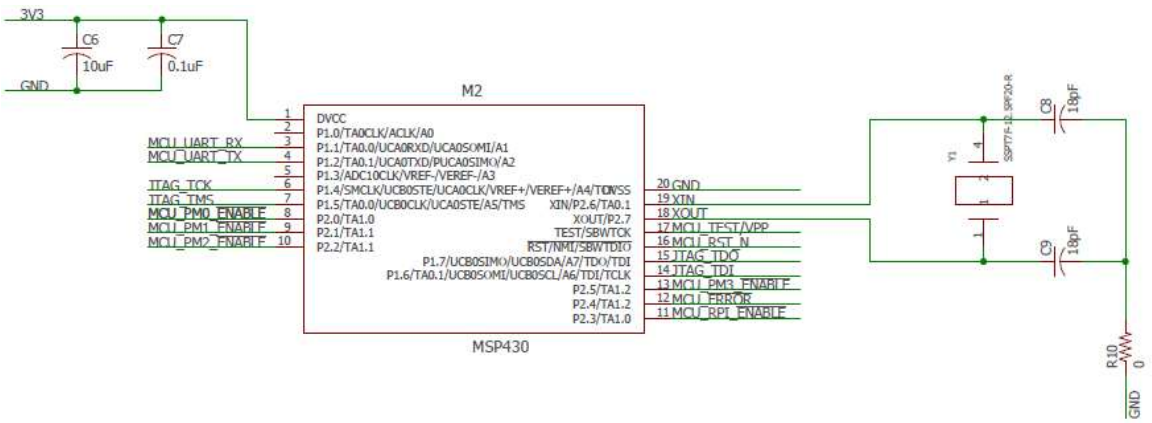

Figure 12 MSP430 circuit

4.2 Transceivers

The specifications for transceivers are SN74AVC8T245 and SN74AVC4T245. The main purpose of these transceivers is to protect Raspberry Pi, in case of inverse current burn the chip. The circuits for transceivers are shown as follow in Fig. 13 and Fig. 14.

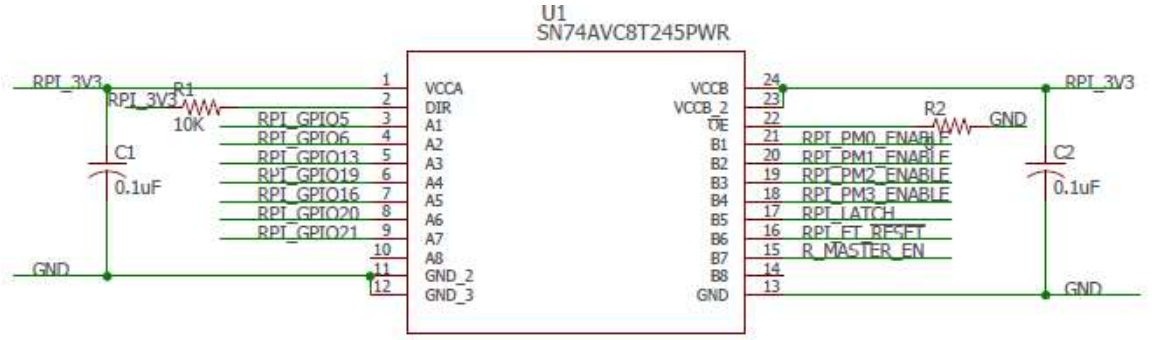

Figure 13 Transceiver circuit of SN74AVC8T245

Figure 14 Transceiver circuit of SN74AVC4T245

4.3 Latch

SN74LV373APWR is the latch used on the host board (Fig. 15). Since Raspberry Pi can be remotely reset, the output voltage level could bounce sometimes. There might be some error operation during reset, such as turn on/off any peripherals accidentally. Therefore, latch could solve this problem by changing the output states only if the latch is enabled ($\overline{OE}$ is low, LE is high).

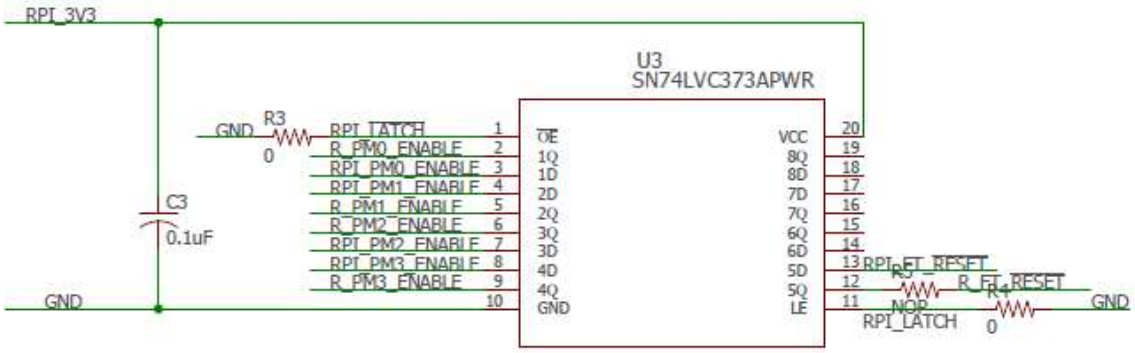

Figure 15 Latch circuit

4.4 Buffers

There are two types of buffers are used in the circuit. They are SN74LV07APWR and SN74HC540PWR. Both of them are used to stabilizing output voltage, so eliminating spark current influence. SN74HC540PWR is also an inverter because MSP430 is programmed using negative logic for eliminating ripple wave interference. The schematics for buffers are shown in Fig. 16 and Fig. 17.

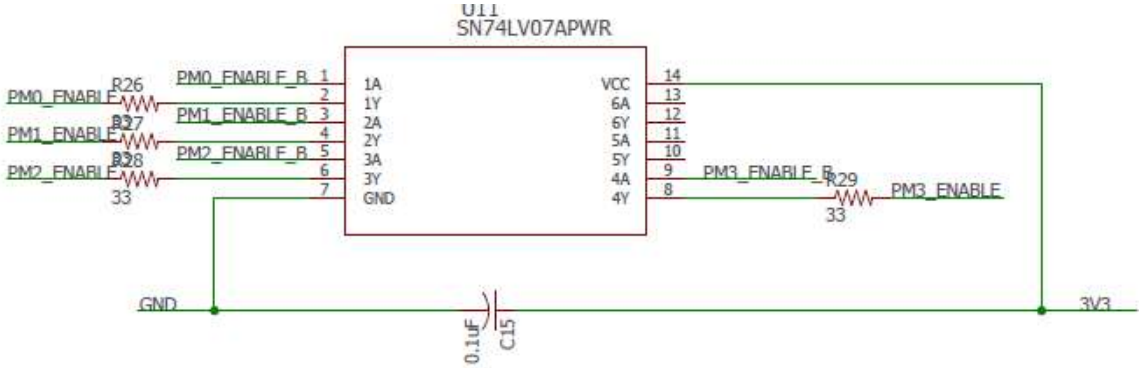

Figure 16 Buffer circuit of SN74LV07APWR

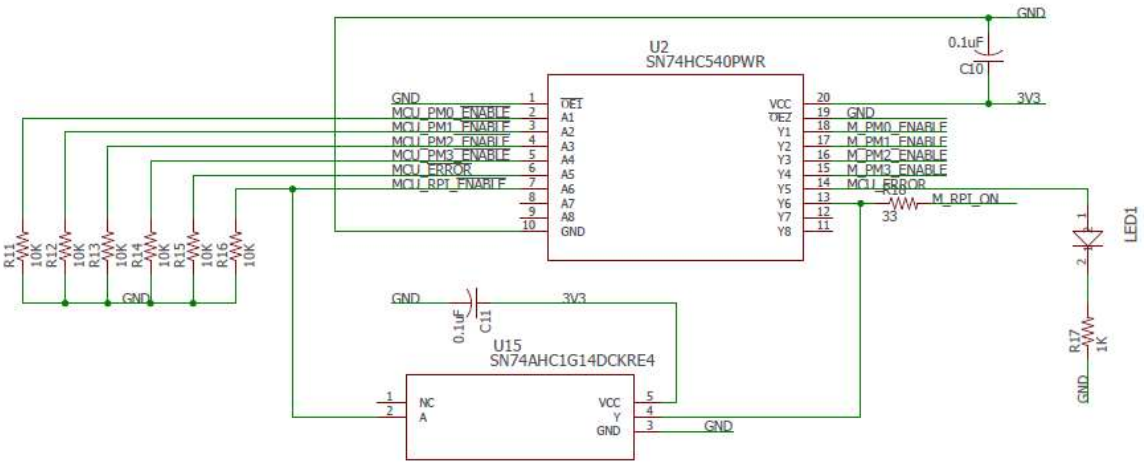

Figure 17 Buffer circuit of SN74HC540PWR

4.5 Switches

There are three kinds of switches apply in the design, which are TPS22929DDBVT, MAX5903, MIC2085. TPS22929DDBVT uses for MSP430 controls the power supply of Raspberry Pi (Fig. 18). MAX5903 and MIC2085 are for power switch modules (Fig. 19 and Fig. 20). MIC2085 can only support supply voltage from 2.3V to 16.5V. In the future, there might be some high-power devices applied, so MAX5903 is added, which can operate from 7V to 72V. Therefore, two of four power switch modules use MAX5903, and other two use MIC2085.

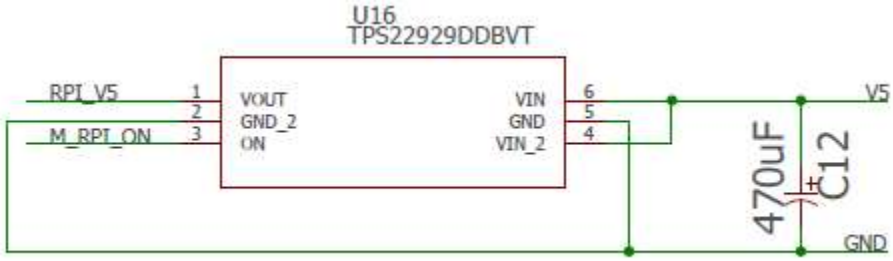

Figure 18 Switches circuit of TPS22929DDBVT

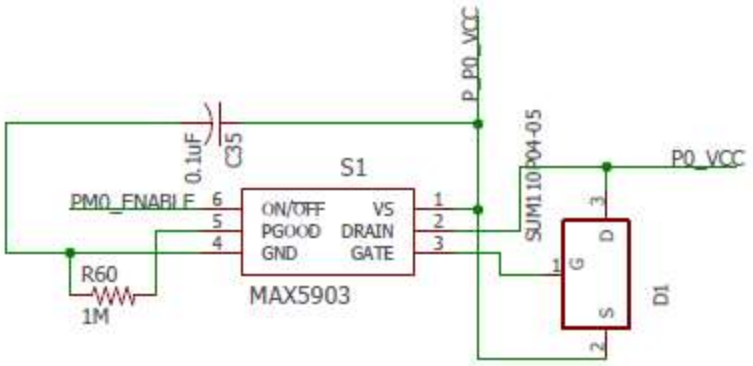

Figure 19 Switch circuit of MAX5903

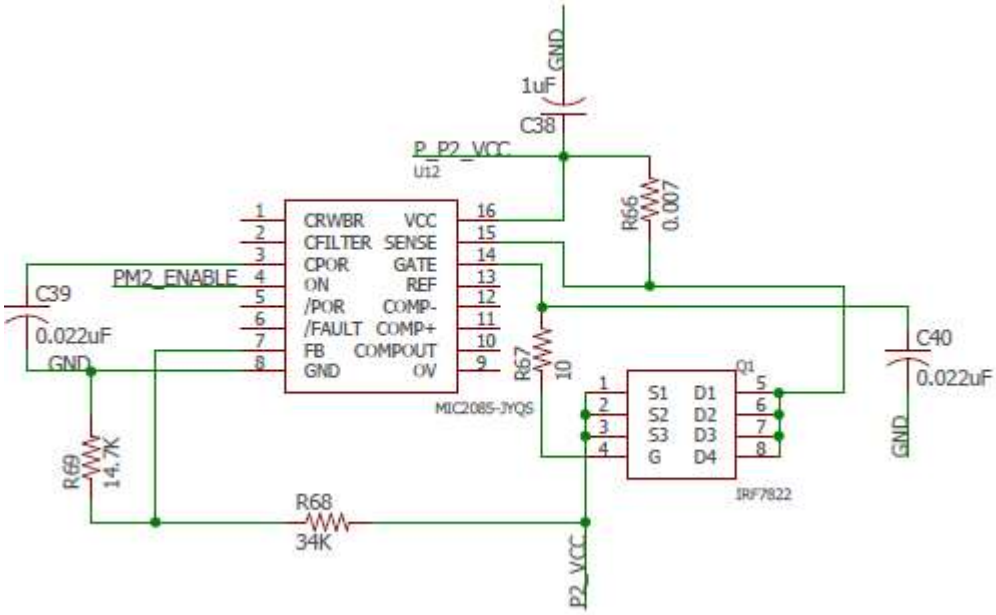

Figure 20 Switch circuit of MIC2085

4.6 ADC

ADC will be directly connected to the battery for monitoring the voltage level. ADS1015 is an ultra-small, low power consumption, and 16 bits ADC, which is well compatible with Raspberry Pi. It communicates with Raspberry Pi using I2C port. Whenever users want to acquire battery level, Raspberry Pi could just read the data from ADC. The circuit of ADC is shown in Fig. 21.

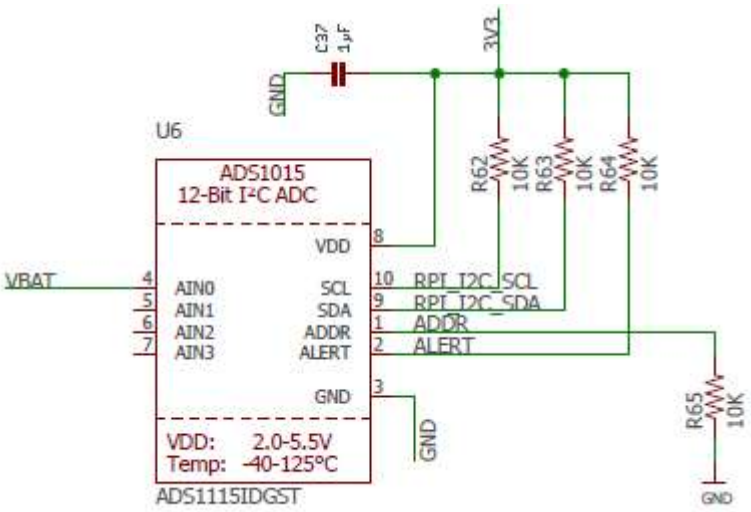

Figure 21 ADC circuit

4.7 MUX

The MUX used here is SN74CB3Q3257PWR. MUX could help to choose from Raspberry Pi or MSP430. The default inputs are from MSP430 unless the S pin is set to high, Raspberry Pi will take charge of the MUX. There are 4 LEDs to indicate which power switch is enabled (Fig. 22).

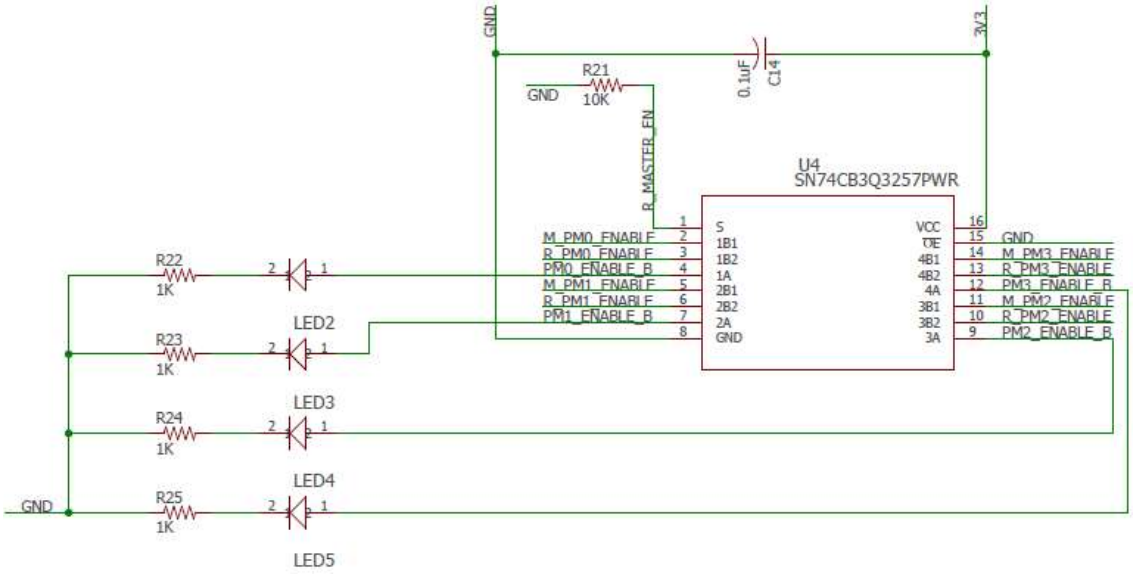

Figure 22 MUX circuit

4.8 Voltage Converters

There are three voltage converter circuits, which are for 3.3V, 5V, and 12V outputs. In Fig. 23, LT3689 is used to bring the battery voltage down to 3.3V. By changing the resistor, which in the red box in Fig. 23 to 536Ω, it will be the 5V voltage converter circuit. For Fig. 24, it utilizes JWL50 to convert battery voltage to 12V.

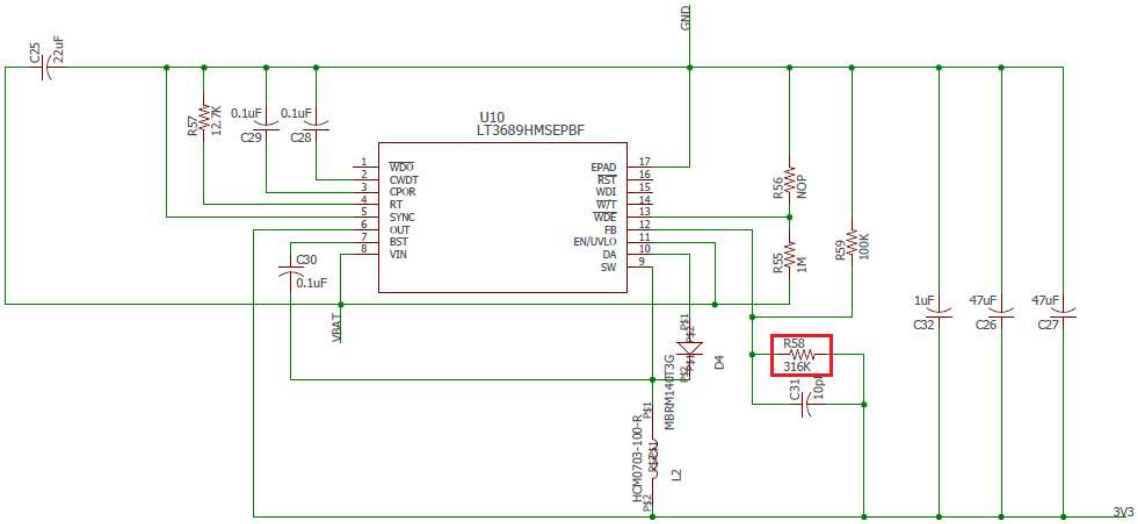

Figure 23 Voltage converter circuit for 3.3V

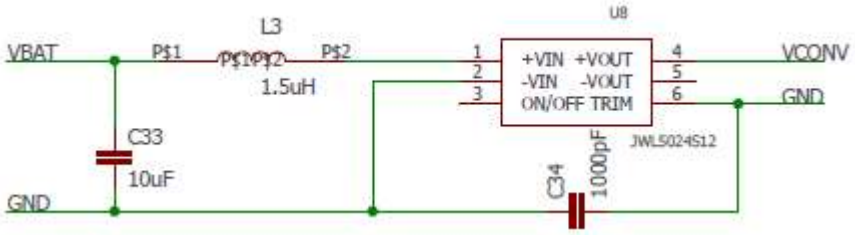

Figure 24 Voltage converter circuit for 12V

5 Raspberry Pi and MSP430 Programming

To realize the function of remotely turn on/off the peripherals, programs need to be run on both Raspberry Pi and MSP430. MUX can help Raspberry Pi and MSP430 to collaboratively control the power switches. The default control device is MSP430. It will directly take commands from RF modem, and execute directly. When the MUX Select pin turns to high, Raspberry Pi will take charge of the power control function. Furthermore, a latch is introduced to help Raspberry Pi stabilize output, which will store the previous state unless the enable pin is high.

5.1 Raspberry Pi Programming

The entire code is posted in Appendix D1, which includes a power control script and an example of how to remotely control the power supply.

The first part of power control script is shown in Fig. 25. It imports the GPIO and time package, then set the GPIO pins which are used in the previous design to be output mode. After that, all the pins are named according to their functions, such as PM0-4 represent they are the pins who control each peripherals' power module.

```
# import the GPIO and time package
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(29,GPIO.OUT)
GPIO.setup(31,GPIO.OUT)
GPIO.setup(33,GPIO.OUT)
GPIO.setup(35,GPIO.OUT)
GPIO.setup(37,GPIO.OUT)
GPIO.setup(36,GPIO.OUT)
GPIO.setup(38,GPIO.OUT)
GPIO.setup(40,GPIO.OUT)

# Signal and pin number
MASTER_EN = 40
LATCH_EN = 36
LATCH_OE = 37
PM0 = 29
PM1 = 31
PM2 = 33
PM3 = 35
FT_EN = 38
```

Figure 25 Power control script (part 1)

Next, a structure is introduced. It uses to store the states of power supply for each peripheral. True means on, while False means off. Every peripheral is initially set to on (Fig. 26).

```
# Power module status class
class PM_Stat:
    def __init__(self):
        self.PM0 = True
        self.PM1 = True
        self.PM2 = True
        self.PM3 = True
```

Figure 26 Power control script (part 2)

Then, if users want to change the status of power modules, the SetStat function could be used. At first, the LATCH_EN pin was set to be low for maintaining the previous status for this moment. Second, according to the input of users, the output pins are set. Then, the latch is enabled for adjusting the status of power modules. After 1 second, the latch will be disabled again and the power module status change is done (Fig 27).

```
# Set power modules based on status
def SetStat(pmstat):
    GPIO.output(LATCH_EN, False)
    GPIO.output(PM0, pmstat.PM0)
    GPIO.output(PM1, pmstat.PM1)
    GPIO.output(PM2, pmstat.PM2)
    GPIO.output(PM3, pmstat.PM3)
    GPIO.output(LATCH_EN, True)
    time.sleep(1)
    GPIO.output(LATCH_EN, False)
    return
```

Figure 27 Power control script (part 3)

Display function is important for users to know the current status of power modules. The display part just read the value from the structure and show each value on screen (Fig 28).

```
# Display the power module status
def ShowStat(pmstat):
    print "The current power module status is:"
    if (pmstat.PM0 == True):
        print "Peripheral 1 is ON"
    else:
        print "Peripheral 1 is OFF"
    if (pmstat.PM1 == True):
        print "Peripheral 2 is ON"
    else:
        print "Peripheral 2 is OFF"
    if (pmstat.PM2 == True):
   print "Peripheral 3 is ON"
    else:
        print "Peripheral 3 is OFF"
    if (pmstat.PM3 == True):
        print "Peripheral 4 is ON\n"
    else:
        print "Peripheral 4 is OFF\n"
    return
```

Figure 28 Power control script (part 4)

The last one is reboot function. Users only need to input the port name which they want to reboot. The program will find the port and turn the power module off and then on. The final status of all the power modules will be display at the end (Fig. 29).

```
# Reboot one peripheral
def RebootPM(port, pmstat):
    print "Rebooting:"
    if (port == PM0):
        print "Peripheral 1"
        pmstat.PM0 = False
        SetStat(pmstat)
        pmstat.PM0 = True
    elif (port == PM1):
        print "Peripheral 2"
        pmstat.PM1 = False
        SetStat(pmstat)
        pmstat.PM1 = True
    elif (port == PM2):
        print "Peripheral 3"
        pmstat.PM2 = False
        SetStat(pmstat)
        pmstat.PM2 = True
    elif (port == PM3):
        print "Peripheral 4"
        pmstat.PM3 = False
        SetStat(pmstat)
        pmstat.PM3 = True
    else:
        print "Port name error: port name should be PM0~PM3"
    SetStat(pmstat)
    ShowStat(pmstat)
    return pmstat
```

Figure 29 Power control script (part 5)

The example of turning all peripherals on shown below, in Fig. 30. It first enables the MUX and the latch, then using the PM_Stat structure and SetStat function to turn them on. After all of that, the MUX and the latch will be disabled at the end.

```
import powercontrol
GPIO.output(MASTER_EN, True)
GPIO.output(LATCH_OE, False)
stat = powercontrol.PM_Stat()
stat.PM0 = True
stat.PM1 = True
stat.PM2 = True
stat.PM3 = True
powercontrol.SetStat(stat)
powercontrol.ShowStat(stat)
GPIO.output(MASTER_EN, False)
GPIO.output(LATCH_OE, True)
```

Figure 30 An operation example of Raspberry Pi

5.2 MSP430 Programming

MSP430 programming is relatively simple than Raspberry Pi because it just receives data from RF modem and restores in its buffer. It only checks the buffer every 30 seconds to find any command need to be executed. The principle programming flow is shown in Fig. 31. The valid command that MSP430 could execute is the information that starts with "!@#$%^&*". For instance, a reset command for power module 1 is "!@#$%^&*1R". After the front 8 recognition chars, it follows two chars which contain the command information. However, RF modem will send not only the command for power modules, but also could send files for Raspberry Pi to download. Therefore, in case of any patterns in these files are similar with the 8 recognition chars, which might confuse MSP430, it is only allowed to send one command in 30 seconds. The program will check if there is any other information before or after the command. If yes, MSP430 will not execute the command. After executing the command or no valid command is found, the buffer will be cleared and wait 30 seconds for information from RF modem.

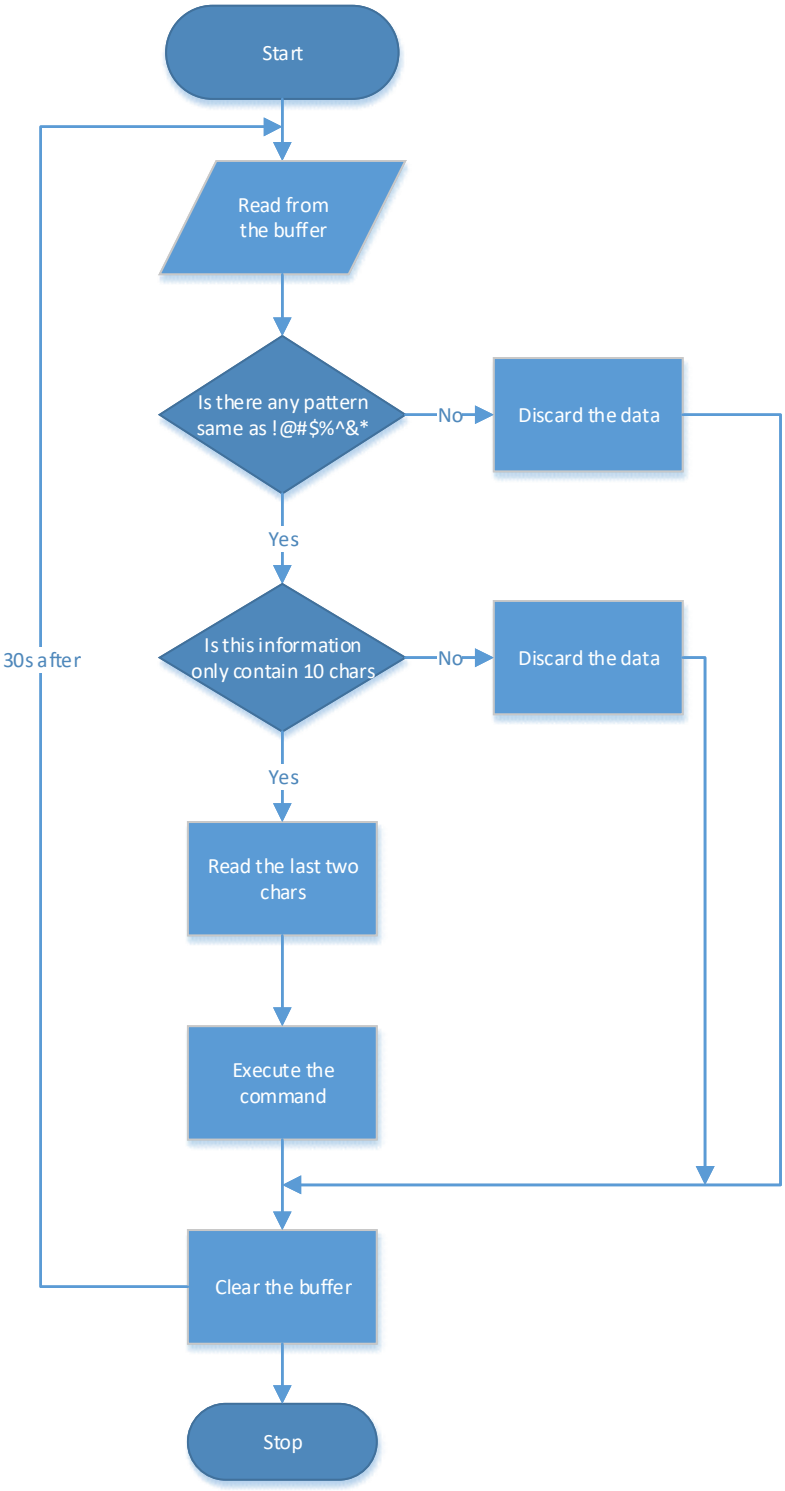

Figure 31 MSP430 programming flowchart

6 PCB Design

The PCB design follows the three principles below:

1) Space saving
2) Reasonable layout
3) Sustainable and reliable design

Since the host board is designed for underwater experiment, the space of the waterproof housing is limited. The host board should be small enough to put into the housing. Therefore, when designing the PCB layout, both sides of the board are filled with components and used 4 layers to route. For reliability, most SMD components are placed on the bottom. After many times revisions, the host board is 170*100mm^2, and the pluggable serial port module is 25*20mm^2 in size. The PCB layout is listed in Appendix C.

Considering the assembly problem, connectors should be managed well. Because the board needs to fit in a cylindrical housing with top lid, the connectors should be placed on the top side for easy assembly. The PCB layout only has sockets on one side. The places where need to plug wires are reserved with enough room.

The circumstance for the underwater experiment is severe, so every component on the host board should not be vulnerable. The PCB is well managed and the potential risks are considered. For instance, the pluggable serial port module uses 2X7 sockets, even it only needs 8 pins for transmission. Furthermore, there are many resistors, whose resistances are 0 and infinite, are added in the circuits. For sustainability, it provides more possibilities for future development. There is no need to print a new PCB for adding other functions because this kind of resistors can be moved and extend by wires for other uses.

7 The Host Board Assembly

The housings for the host board will be waterproof or submersible, so the connections should be tight and reliable when implementing. For the typical underwater experiment, the housing is in a cuboid shape, as shown in Fig. 32 and Fig. 33. Besides that, cylinder housing will be also used for some special underwater tasks that need stronger waterproof ability (Fig. 34 and Fig. 35). Thus, the host board will be connected to the penetrative connectors on the housing. For conveniences in field experiments, there will be an on-wire connector for quickly detaching the host board with housing, and a plug connector with screws for connecting wires and connector, and a compatible socket connector on the host board, which principle sketch is shown in Fig. 36.

Figure 32 Cuboid housing

Figure 33 An example of subsea housing

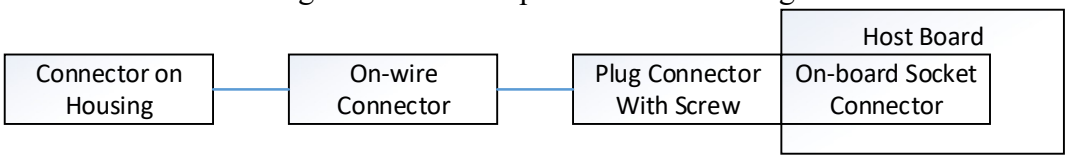

Figure 34 Connection for the host board

8 Conclusions and Future Work

So far, the design of the host board is almost finished. Before the PCB is printed, there are also some testing done for Raspberry Pi. As shown in Fig. 37, a Raspberry Pi output testing board was developed. Every pin that used for control is connected with a LED. After running the program referred before, the LEDs status could accordingly change whenever change the inputs. This test proves that the code for Raspberry Pi works well.

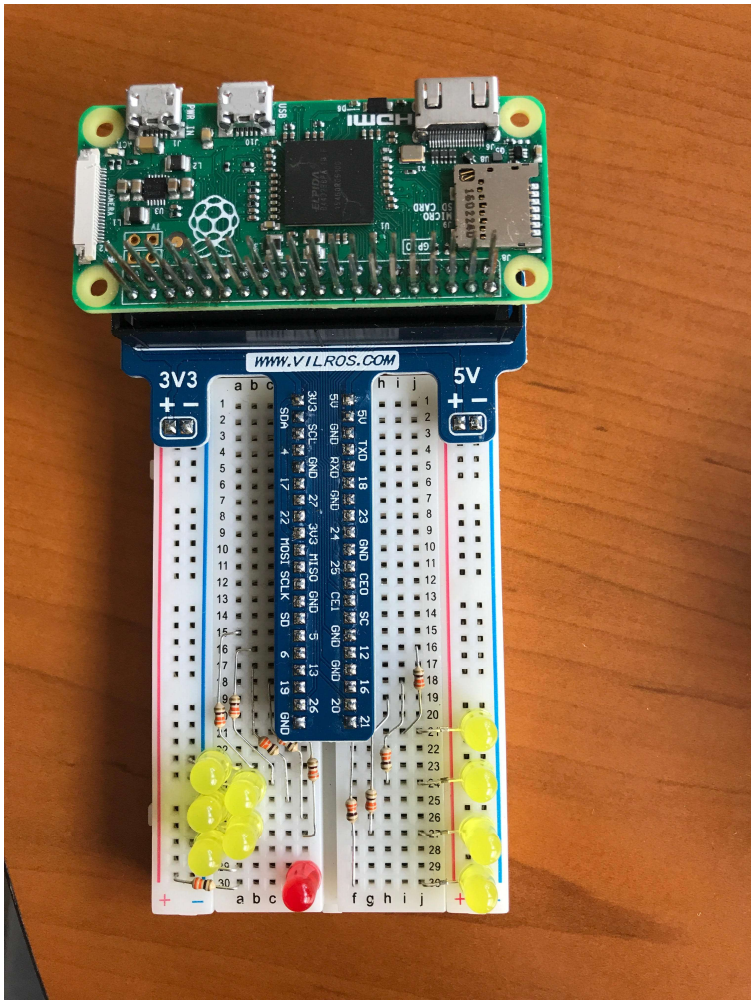

Figure 35 Raspberry Pi output testing board

Although there are a lot of parts for the design is completed, there are still many aspects can be improved. After the PCB is printed, many data collecting works need to do. Power consumption and system lifetime are the main concern for UWSN study. According to these data, there could be some optimizing methods. Besides consumption and lifetime, more noise reduction chips can be added on, for the reason that there could be many different wave interferences in the water.

9 References

[1] Z. Wang, L. Wei, X. Lu, Z. Peng, and J. H. Cui, "Towards power efficient deployment for underwater sensing systems," in *Proc. IEEE 12th Int. Conf. Netw. Sens. Control (ICNSC'15)*, Apr. 9–11, 2015, pp. 625–630.

[2] L. Wei, Z. Wang, J. Liu, Z. Peng, and J. H. Cui, "Power Efficient Deployment Planning for Wireless Oceanographic Systems," in *IEEE System Journal*, 2016.

[3] R. Holman, J. Stanley, and T. Ozkan-Haller, "Applying video sensor networks to nearshore environment monitoring," in *IEEE Pervasive Comput*., vol. 2, no. 4, pp. 14–21, Oct. 2003.

[4] Heidemann J, Ye W, Wills J, Syed A, Li Y., "Research challenges and applications for underwater sensor networking," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, 2006.

[5] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater acoustic sensor networks: Research challenges," in *Ad Hoc Netw*., vol. 3, no. 3, pp. 257–279, May 2005.

[6] S. Han and M. Gerla, "Ocean-TUNE UCLA Testbed for Subsurface Current Mapping," in *The 7th ACM International Conference on Underwater Networks and Systems (WUWNet)*, Los Angeles, California, USA, 2012.

[7] L. Lanbo, Z. Shengli, and C. Jun-Hong, "Prospects and problems of wireless communication for underwater sensor networks," in *Wireless Communications and Mobile Computing, Special Issue on Underwater Sensor Networks*, vol. 8, no. 8, pp. 977–994, August 2008.

[8] J.-H. Cui, J. Kong, M. Gerla, and S. Zhou, "The challenges of building mobile underwater wireless networks for aquatic applications," in *IEEE Netw*., vol. 20, no. 3, pp. 12–18, May 2006.

[9] D.B. Kilfoyle and A.B. Baggeroer, "The state of the art in underwater acoustic telemetry," in *IEEE Journal of Oceanic Engineering,* 25 (1) (2000) 4–27.

[10] C. Petrioli, "FP7 SUNRISE Project: Experimenting with the Internet of Underwater Things," in *FIRE workshop*, 2014. [Online]. Available: http://fp7-sunrise.eu/

[11] C. Petrioli and R. Petroccia, "SUNSET: Simulation, Emulation and Reallife Testing of Underwater Wireless Sensor Networks," in *Proceedings of IEEE UComms*, 2012, pp. 12–14.

[12] J. Rice, "Seaweb Acoustic Communication and Navigation Networks," in *Proceedings of the International Conference on Underwater Acoustic Measurements: Technologies and Results*, Heraklion, Crete, Greece, 2005. [Online]. Available: http://www.dtnrg.org/docs/papers/UAMeasurements2005Rice2.pdf

[13] J.-H. Cui, S. Zhou, Z. Shi, J. ODonnell, Z. Peng, S. Roy, P. Arabshahi, M. Gerla, B. Baschek, and X. Zhang, "Ocean-TUNE: A Community Ocean Testbed for Underwater Wireless Networks," in *Proceedings of The 7th ACM International Conference on Underwater Networks and Systems (WUWNet'12)*, Los Angeles, CA, USA, 2012.

[14] S. Han and M. Gerla, “Ocean-TUNE UCLA Testbed for Subsurface Current Mapping,” in *The 7th ACM International Conference on Underwater Networks and Systems (WUWNet)*, Los Angeles, California, USA, 2012.
[15] L. Wei, Z. Peng, H. Zhou, J.-H. Cui, S. Zhou, Z. Shi, and J. O’Donnell, “Long Island Sound Testbed and Experiments,” in *MTS/IEEE OCEANS’13*, San Diego, USA, 2013, pp. 1–6.
[16] C. R. Barnes, M. M. R. Best, B. D. Bornhold, S. K. Juniper, B. Pirenne, and P. Phibbs, “The NEPTUNE Project - A Cabled Ocean Observatory in the NE Pacific: Overview, Challenges and Scientific Objectives for the Installation and Operation of Stage I in Canadian Waters,” in *Symposium on Underwater Technology and Workshop on Scientific Use of Submarine Cables and Related Technologies*, Tokyo, 2007, pp. 308– 313. [Online]. Available: http://dx.doi.org/10.1109/UT.2007.370809
[17] Monterey Bay Aquarium Research Institue, “The MARS Ocean Observatory Testbed,” in *Official Website*, 2009. [Online]. Available: http://www.mbari.org/mars/
[18] OOI. (2016). *Ocean Observatories Initiative* [Online]. Available: http://oceanobservatories.org/
[19] RPi Hub. (2017). *Raspberry Pi Wiki* [Online]. Available: http://elinux.org/RPi_Hub
[20] Introducing the Raspberry Pi Zero. (2015). Adafruit Learning System [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-zero.pdf
[21] LiNiMnCo 26650 Battery Pack. (2000-2016). *AA Portable Power Corp* [Online]. Available: http://www.batteryspace.com/limnni-26650-battery-14-8v-8ah-118-4wh-8a-rate-battery-module.aspx

Appendix A. The Host Board Schematics

In this section, all schematics for the host board circuit design will be listed as follow. There are three parts, include host board circuit, UART-RS232 pluggable module, and UART-RS485 pluggable module. All of them are drawn by Eagle and under the valid license.

A.1 On-board Circuit Design

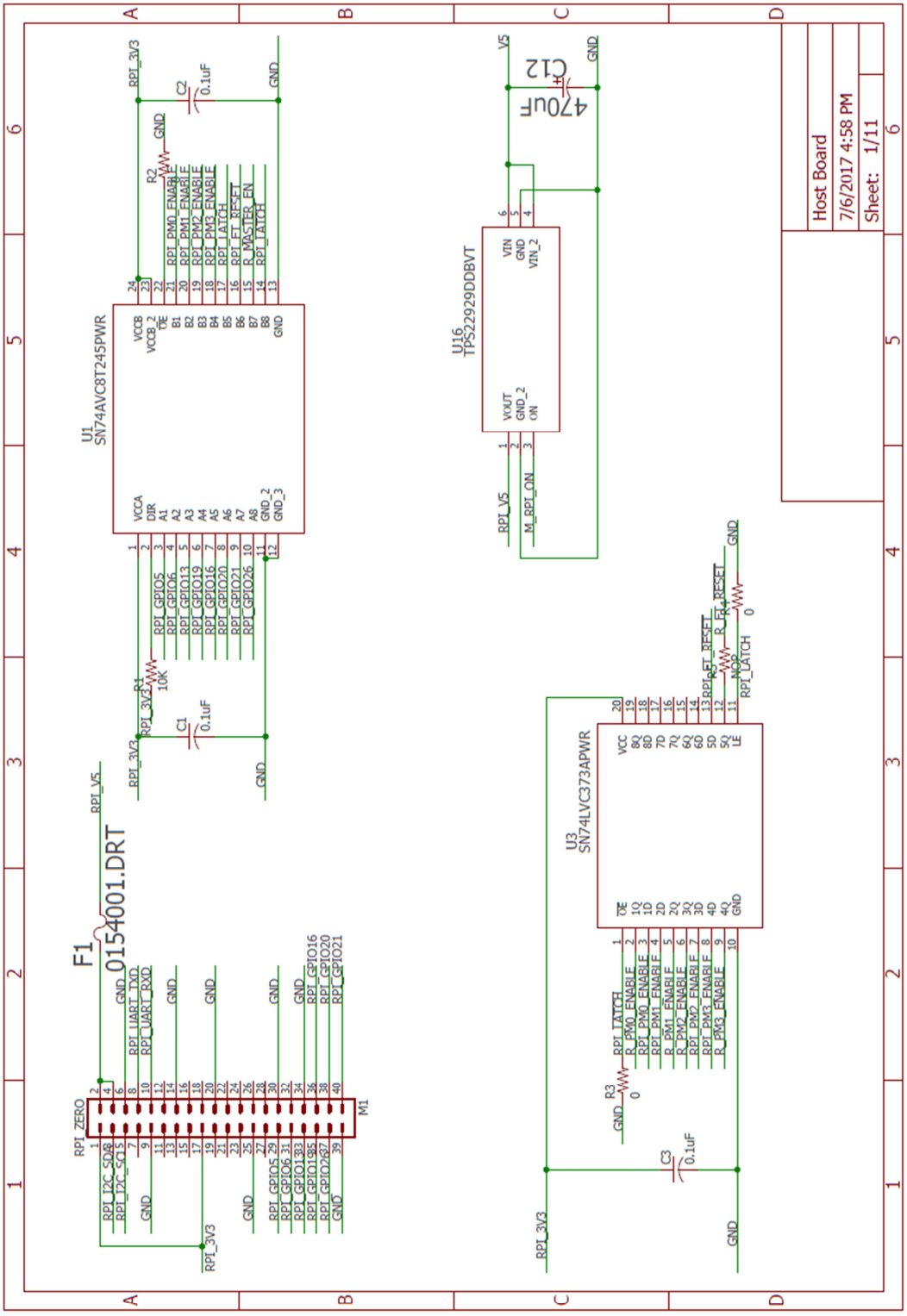

Figure 36 Host board schematic 1/11

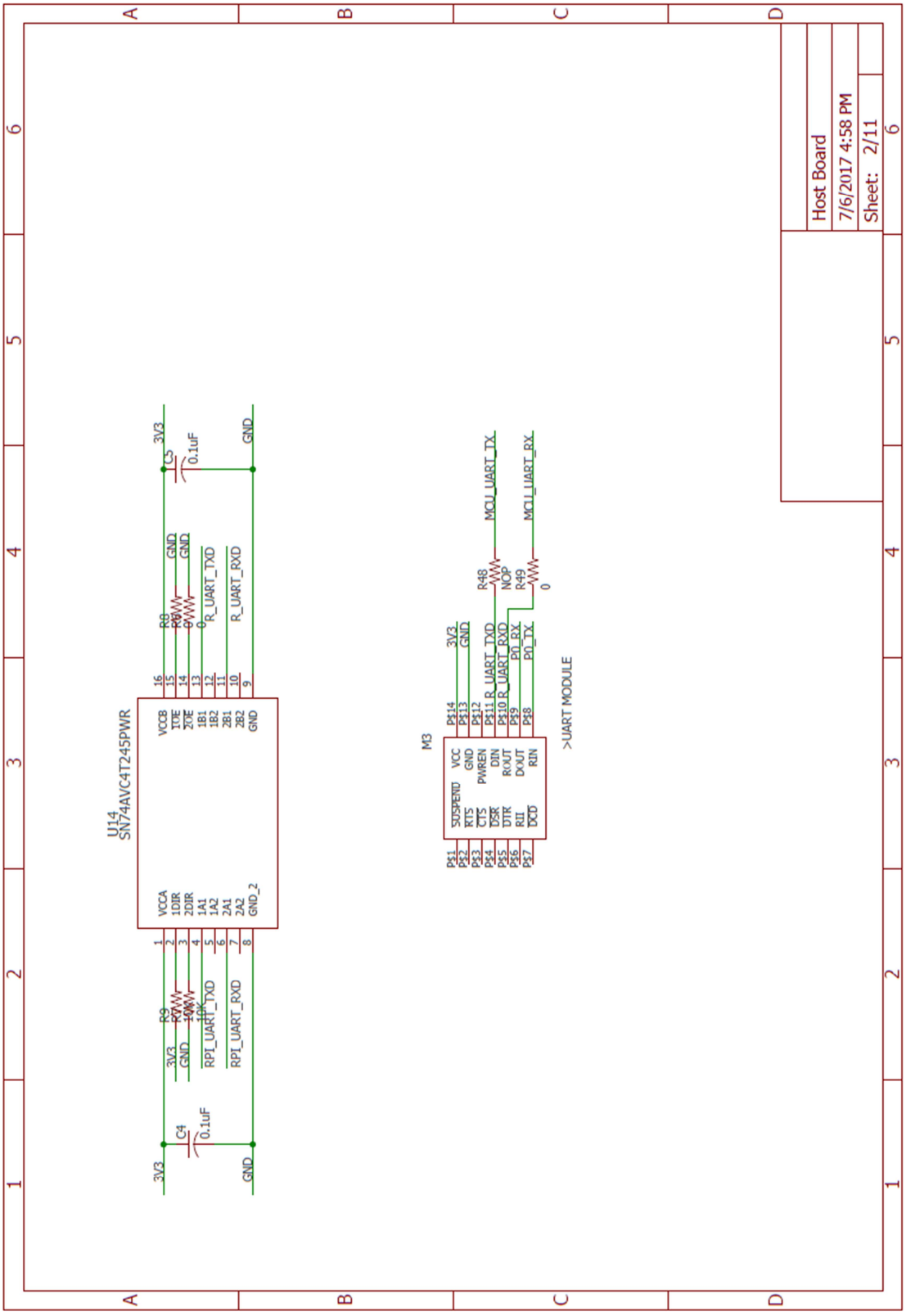

Figure 37 Host board schematic 2/11

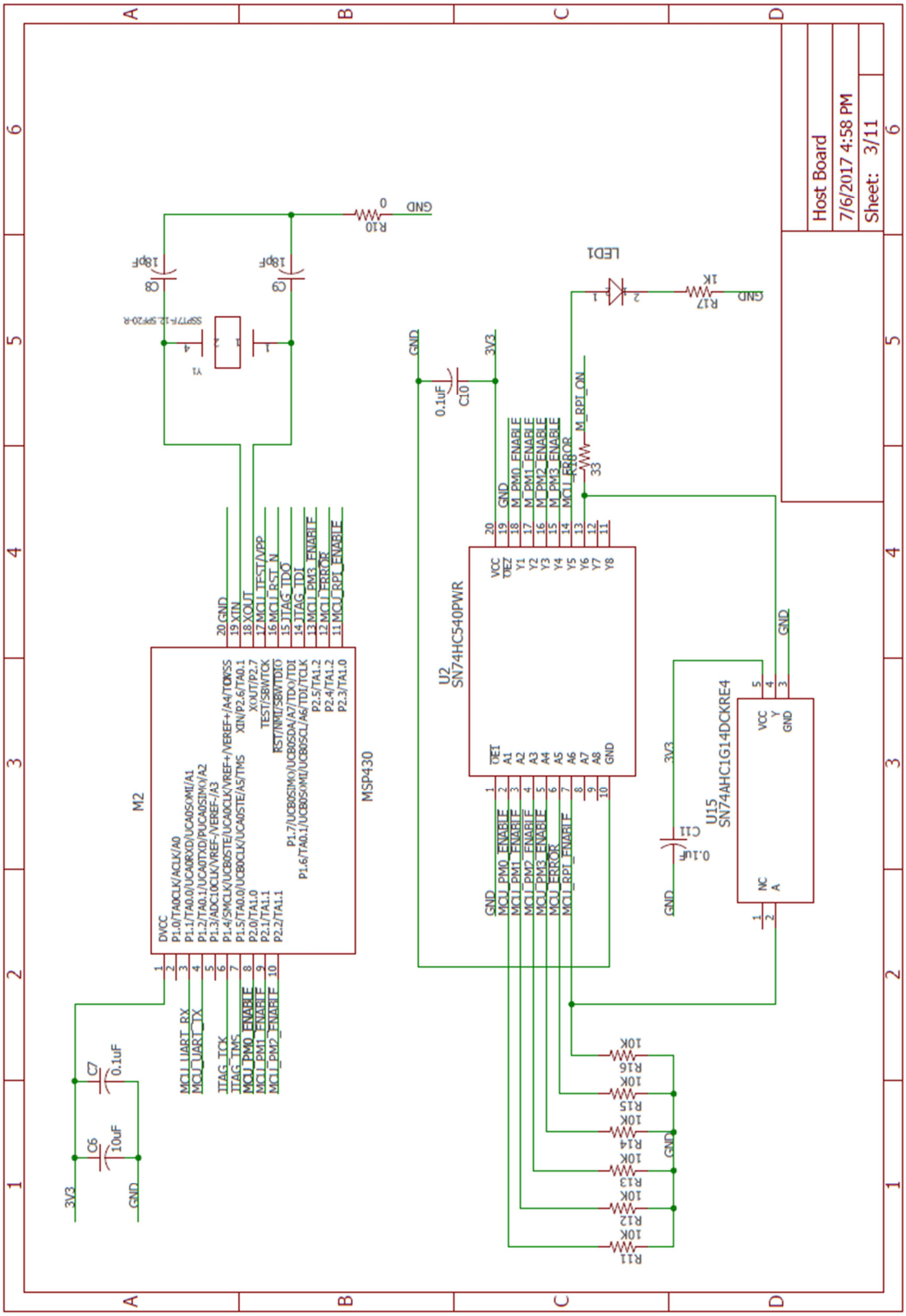

Figure 38 Host board schematic 3/11

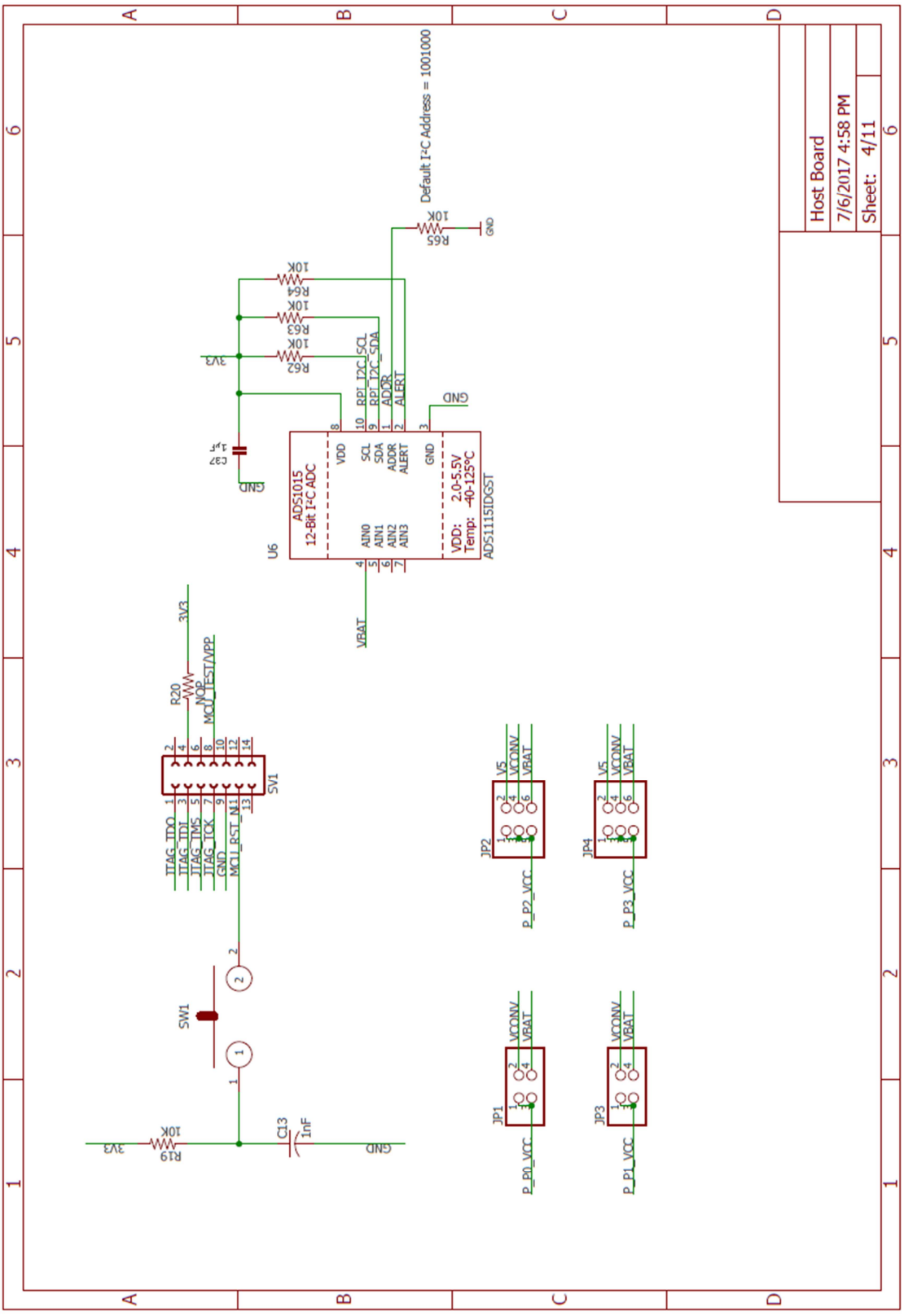

Figure 39 Host board schematic 4/11

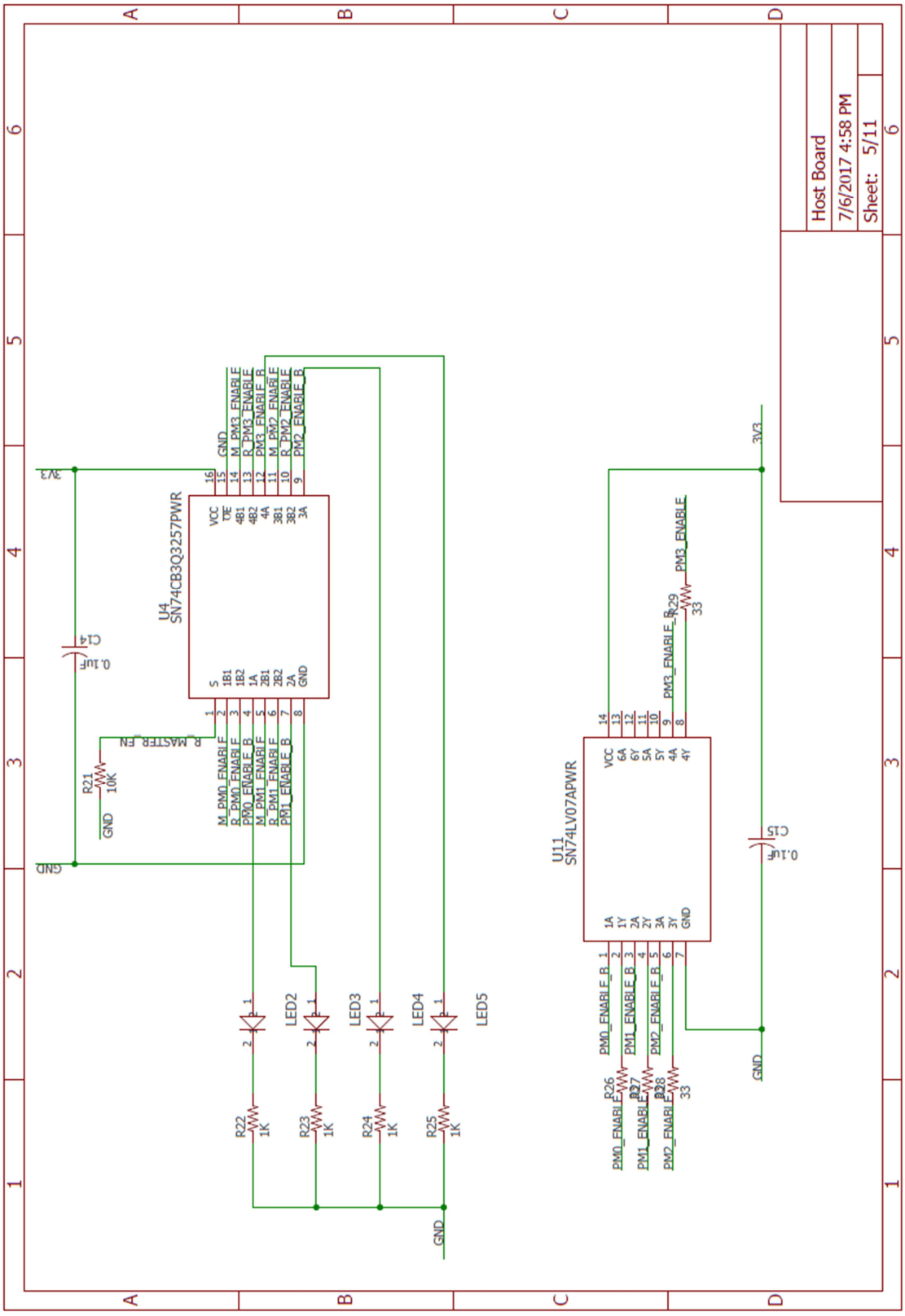

Figure 40 Host board schematic 5/11

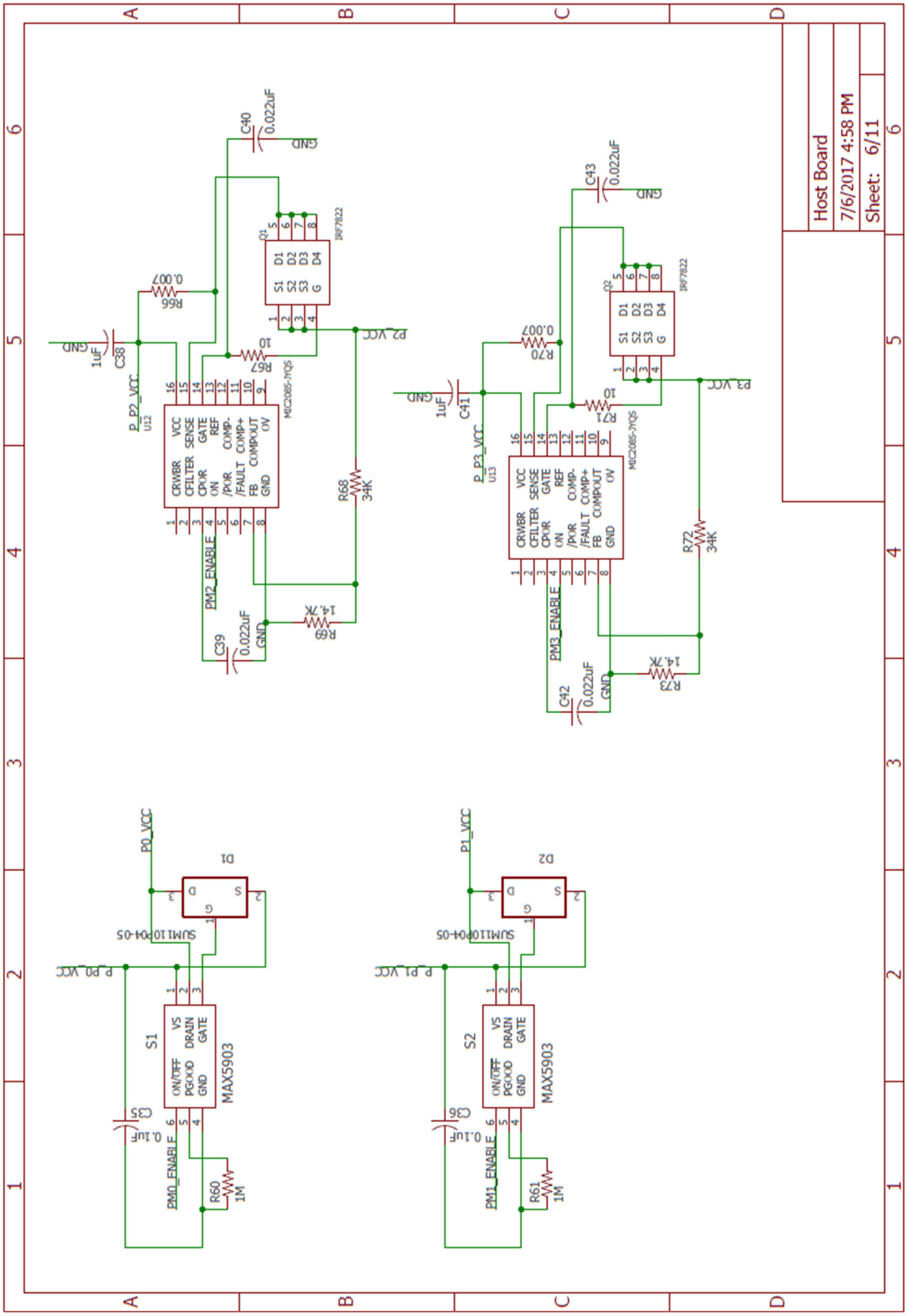

Figure 41 Host board schematic 6/11

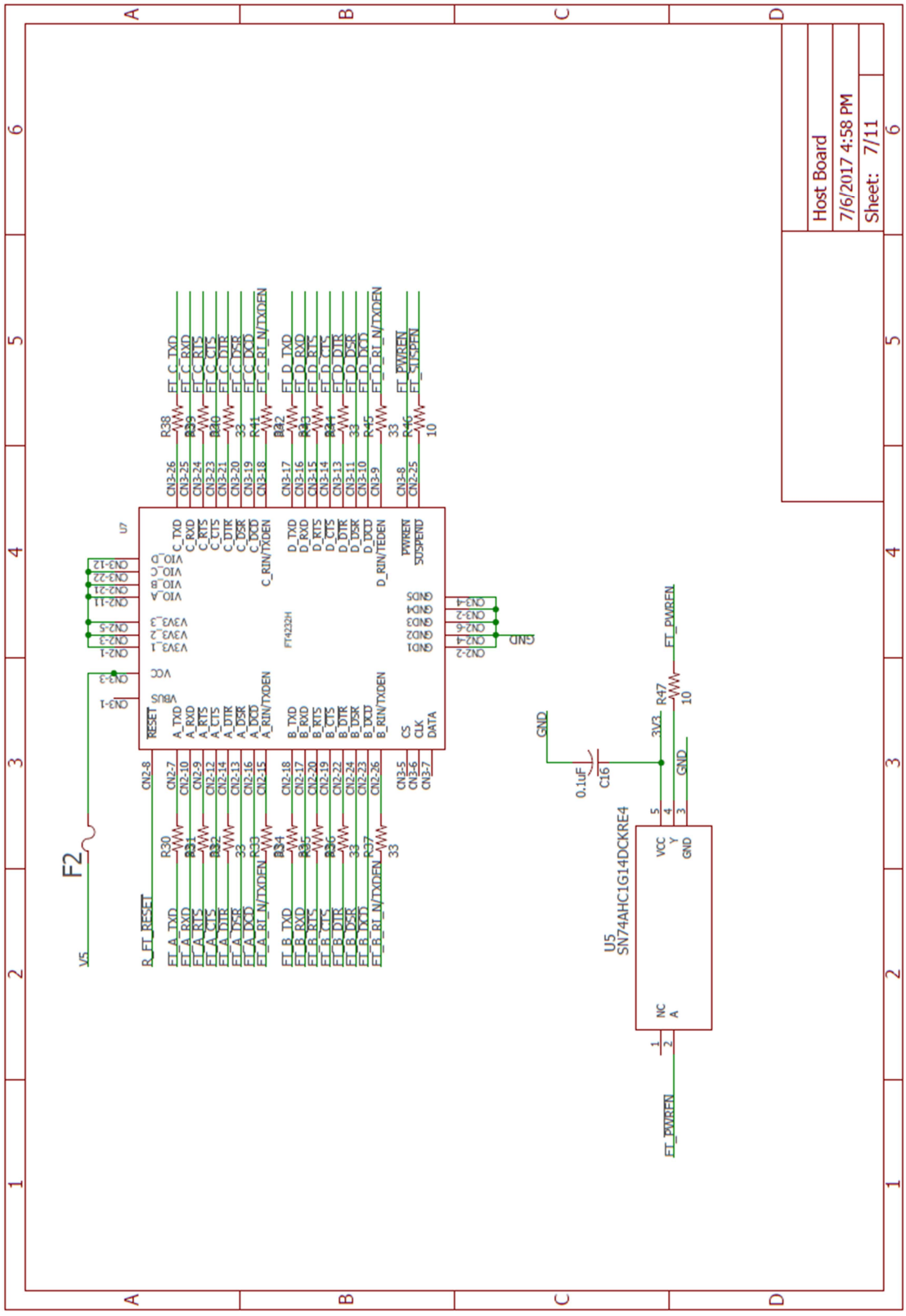

Figure 42 Host board schematic 7/11

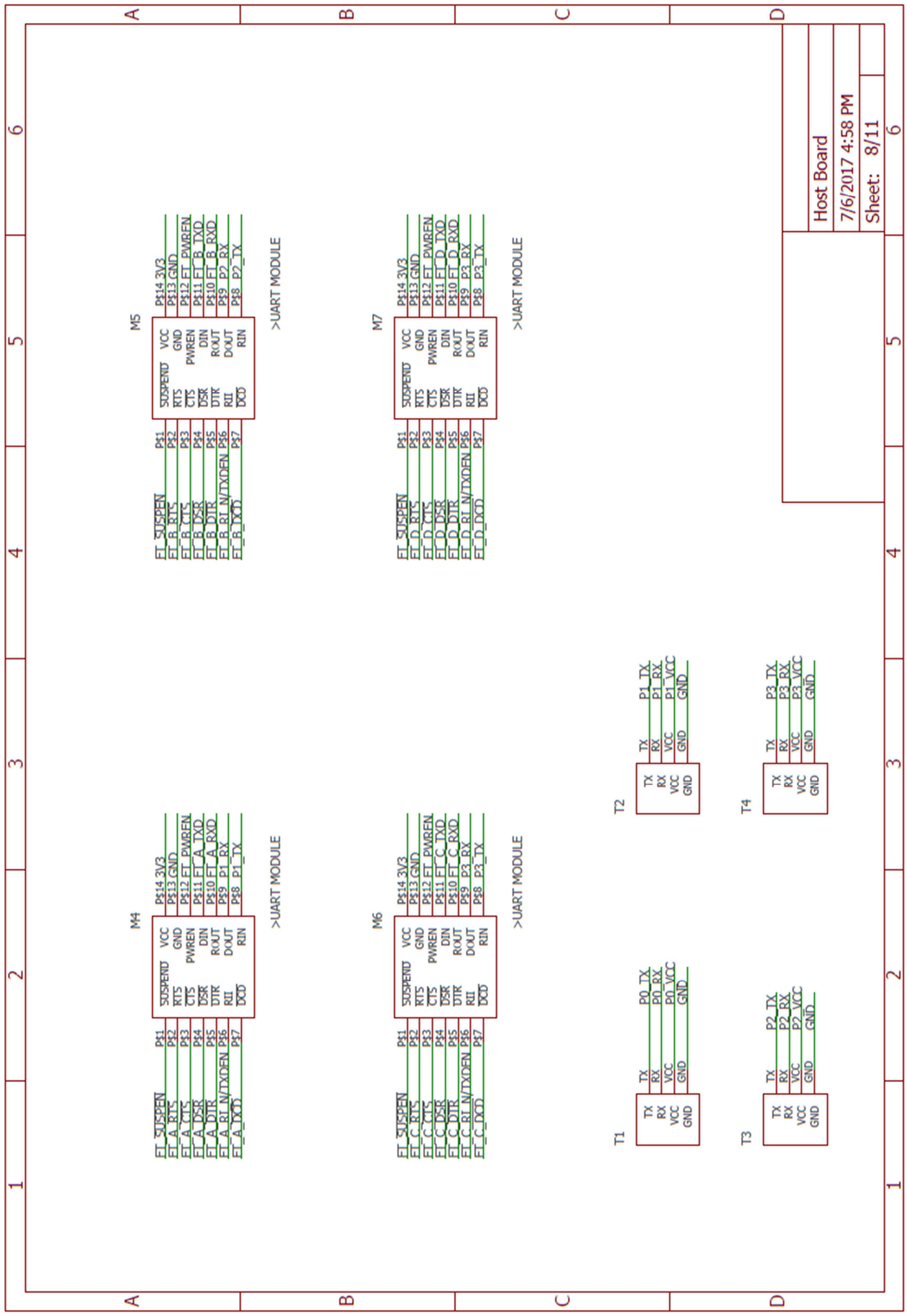

Figure 43 Host board schematic 8/11

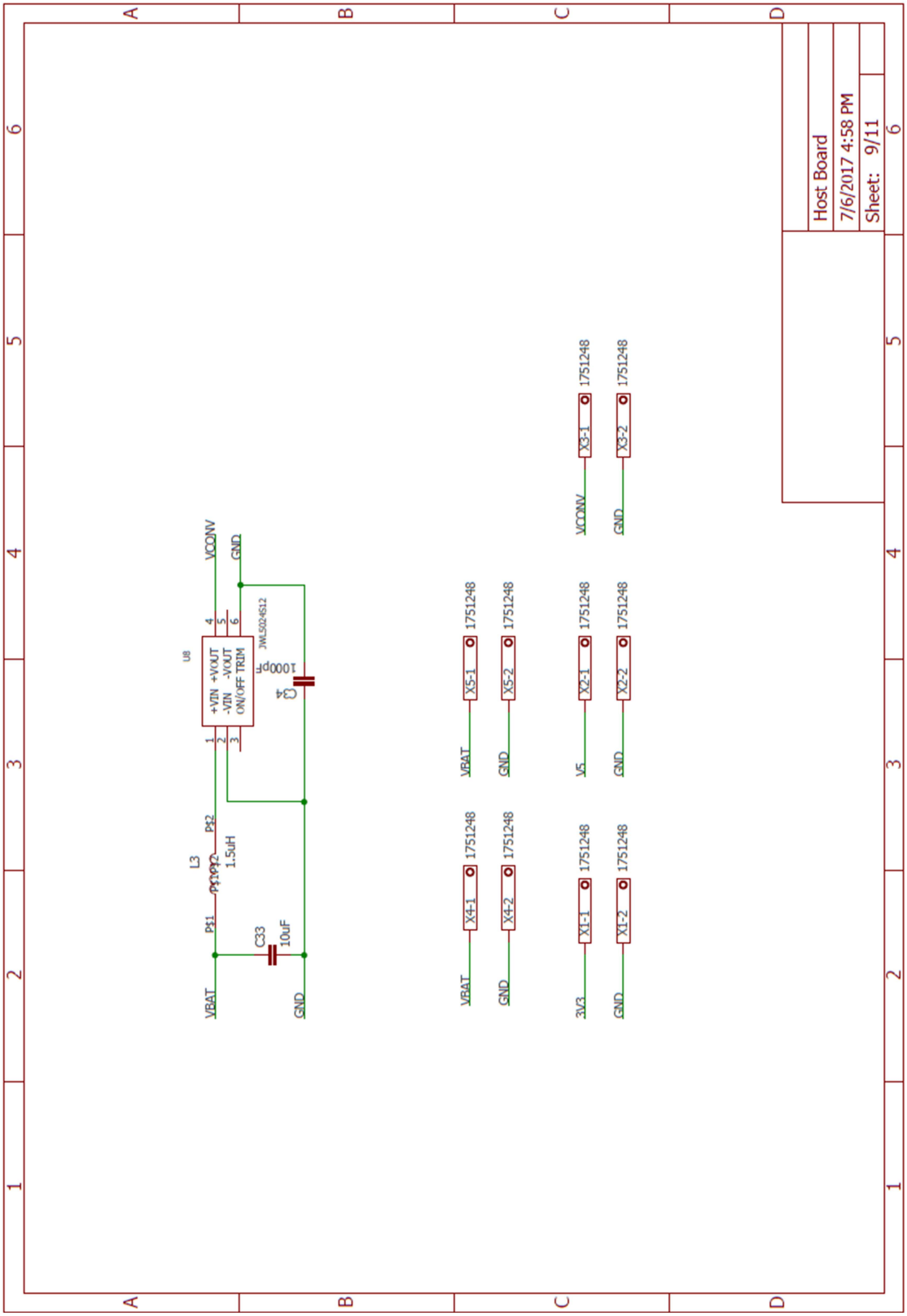

Figure 44 Host board schematic 9/11

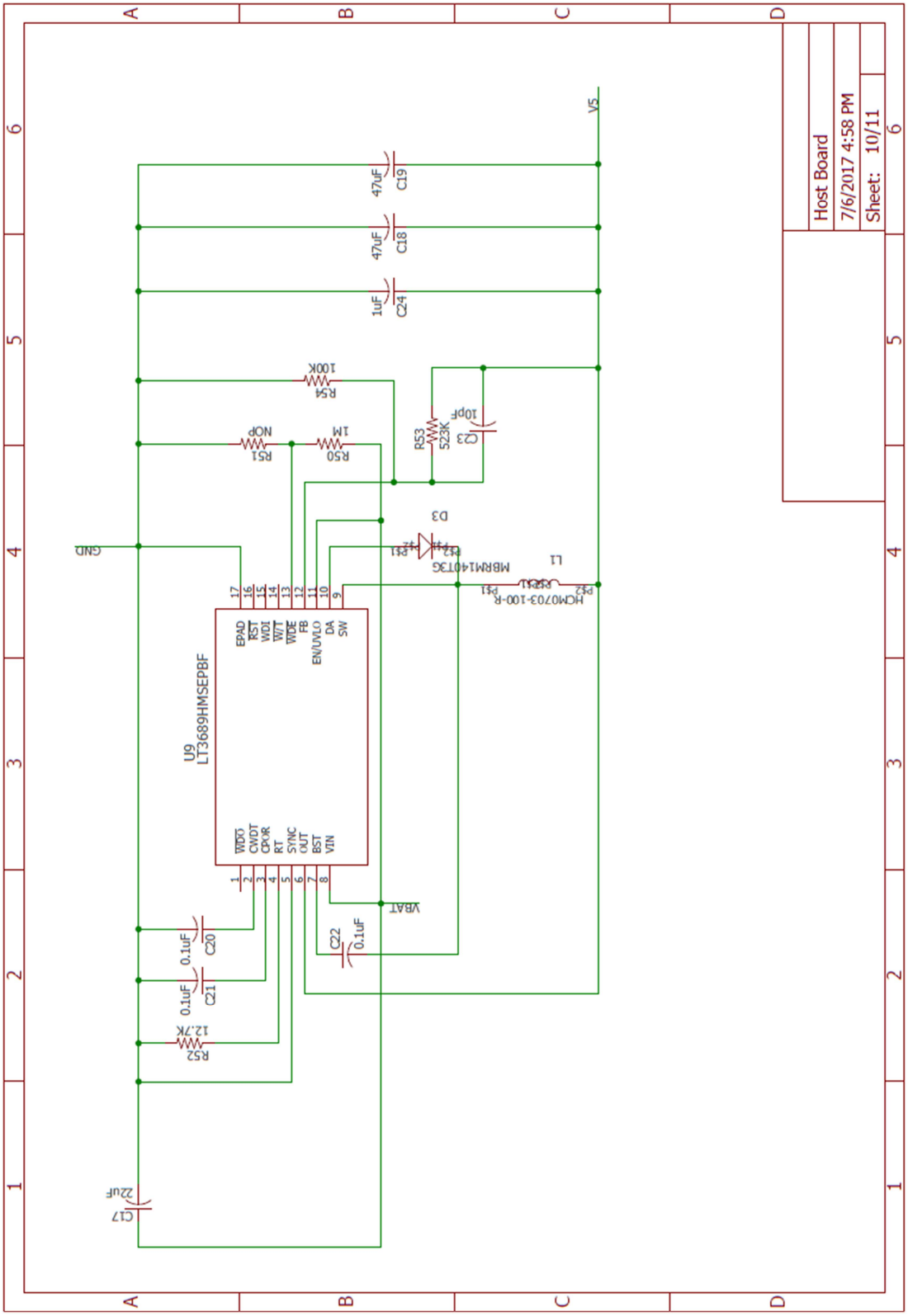

Figure 45 Host board schematic 10/11

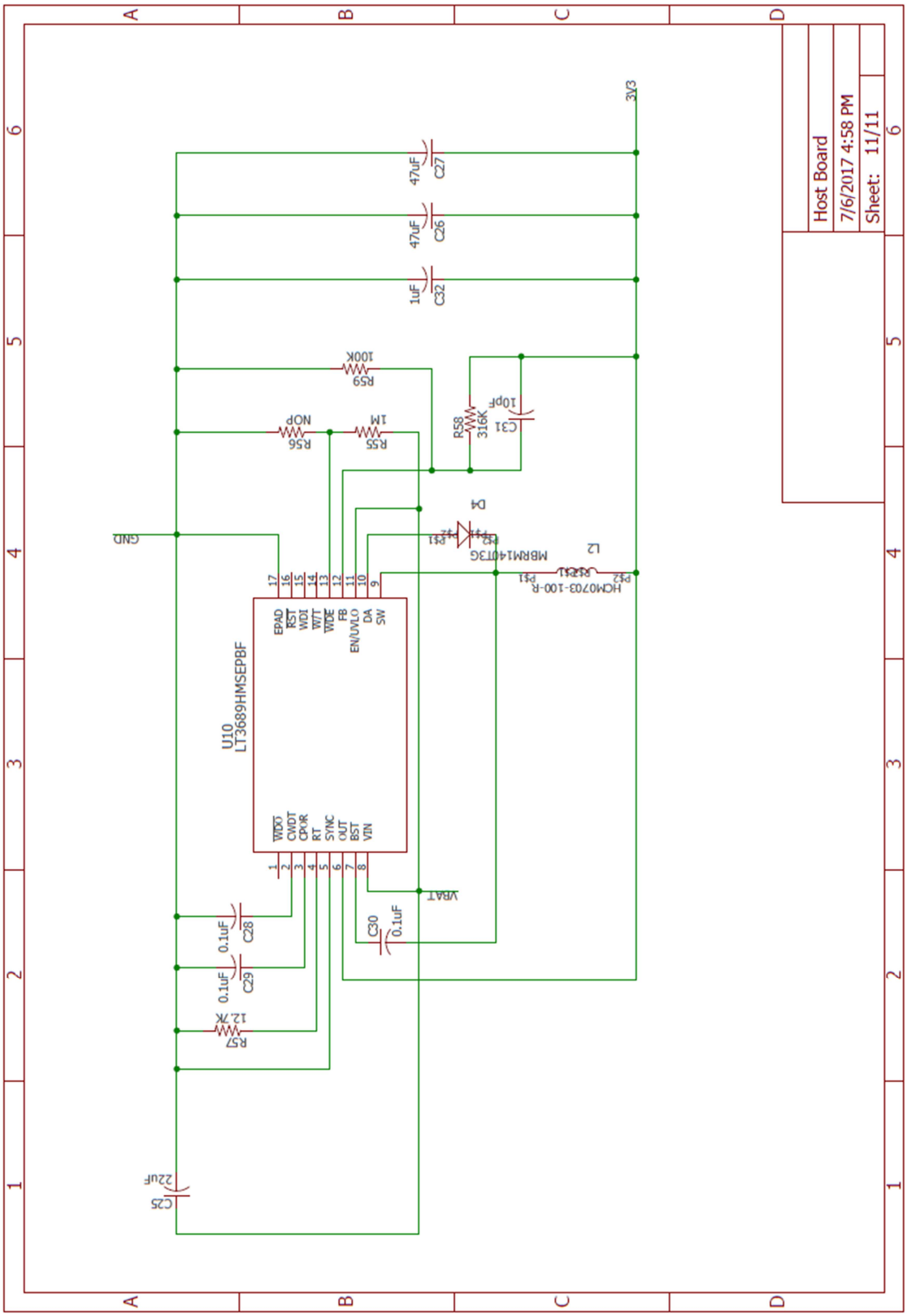

Figure 46 Host board schematic 11/11

A.2 Pluggable Serial Port Module Design

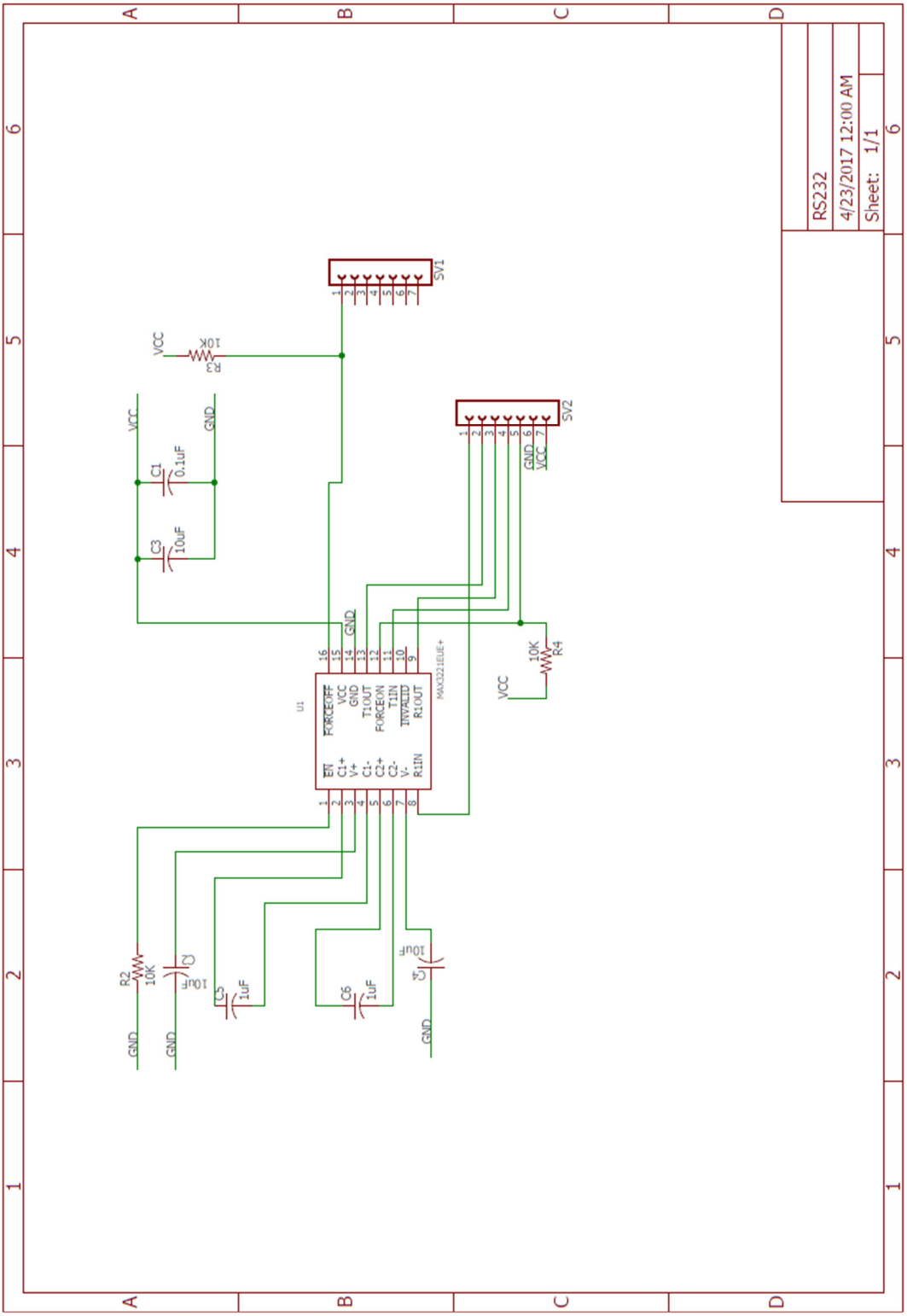

Figure 47 UART-RS232 schematic

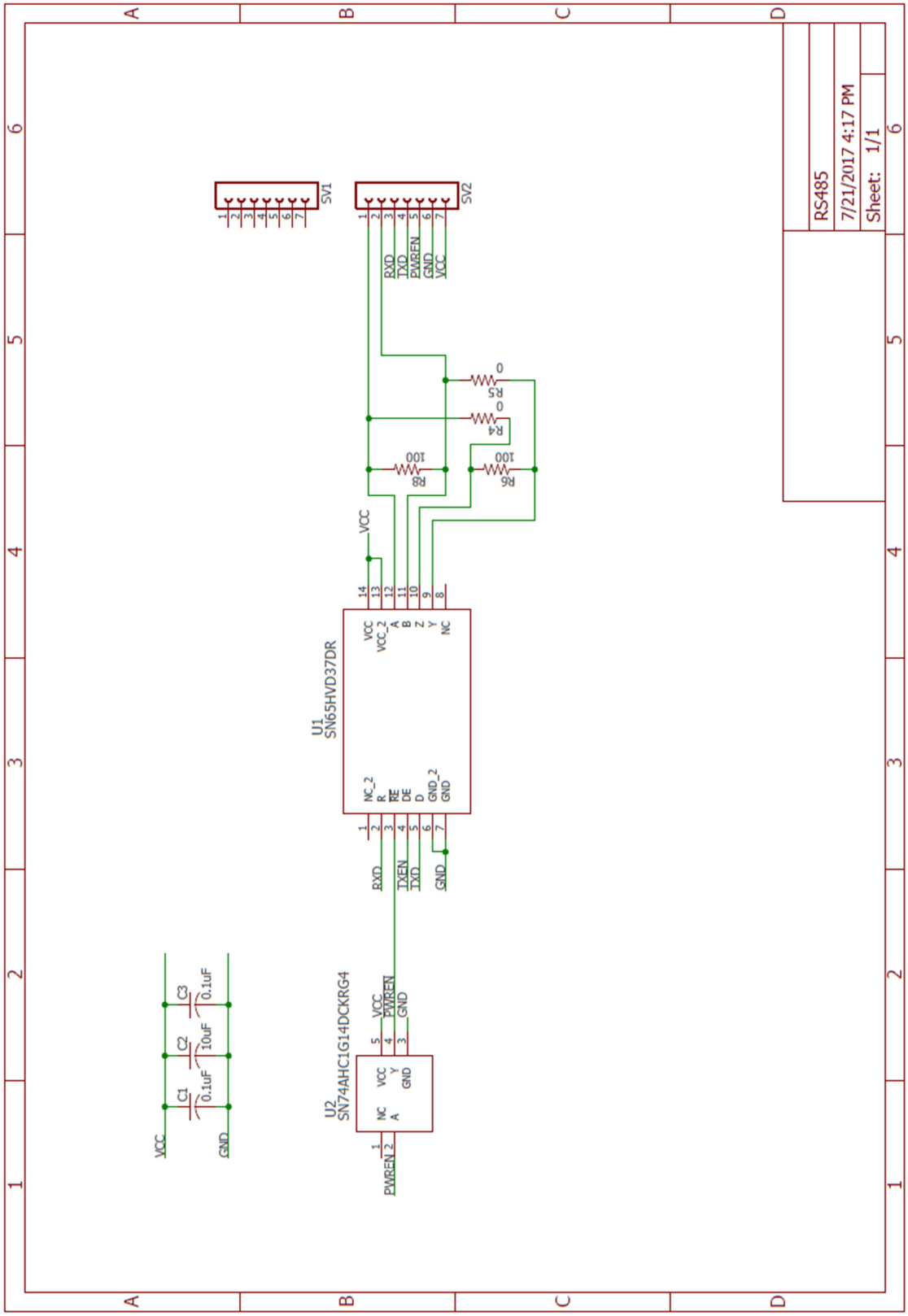

Figure 48 UART-RS485 schematic

Appendix B. Raspberry Pi Zero Pin Map

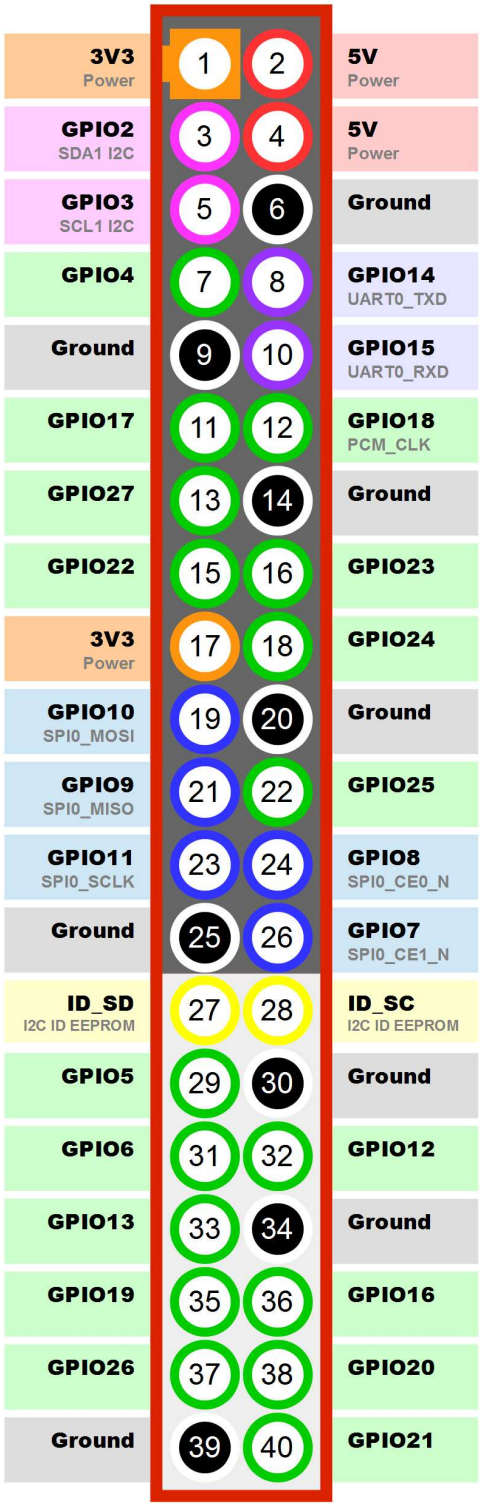

Figure 49 Raspberry Pi Zero pin map

Appendix C. The Host Board PCB Layouts

This section includes all PCB layouts which generate by the schematics from Appendix A. All the routings are created on Eagle software.

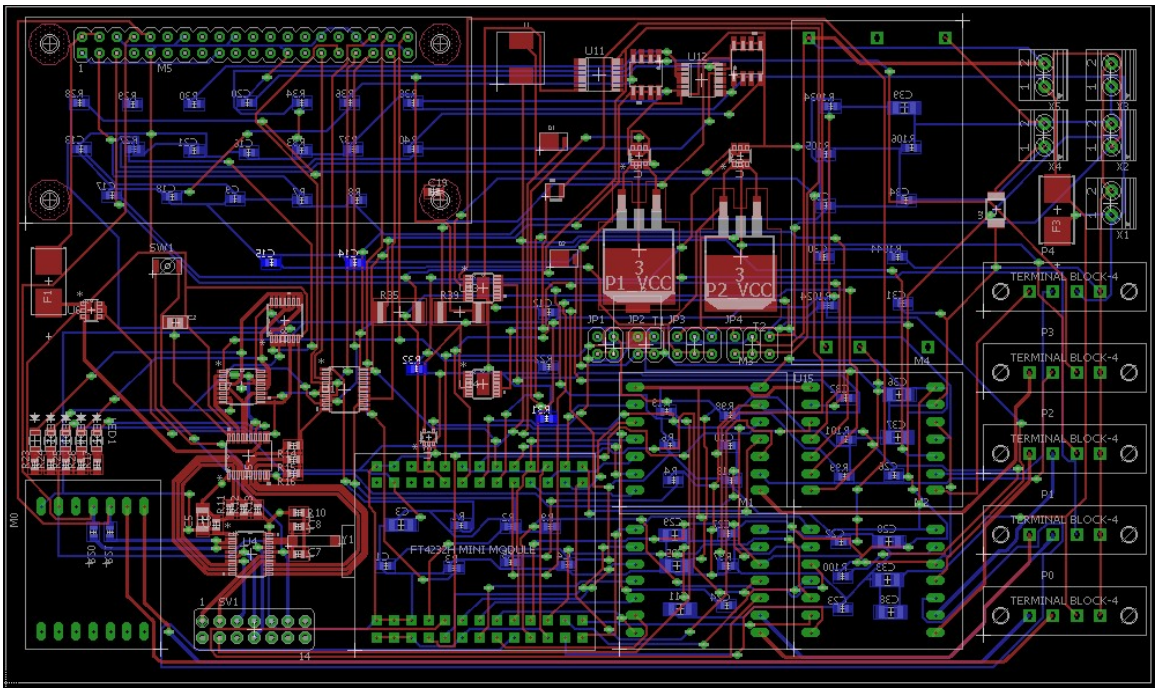

Figure 50 The host board layout

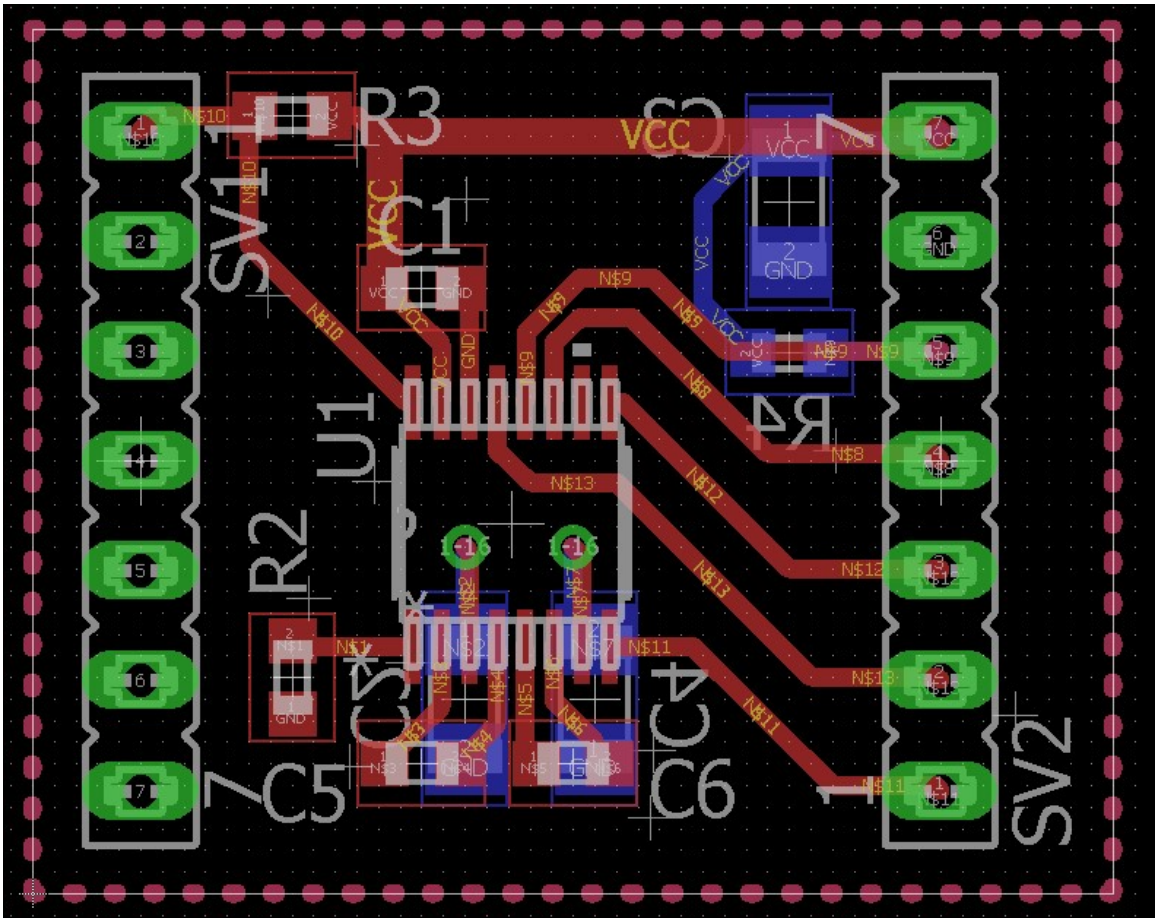

Figure 51 The UART-RS232 layout

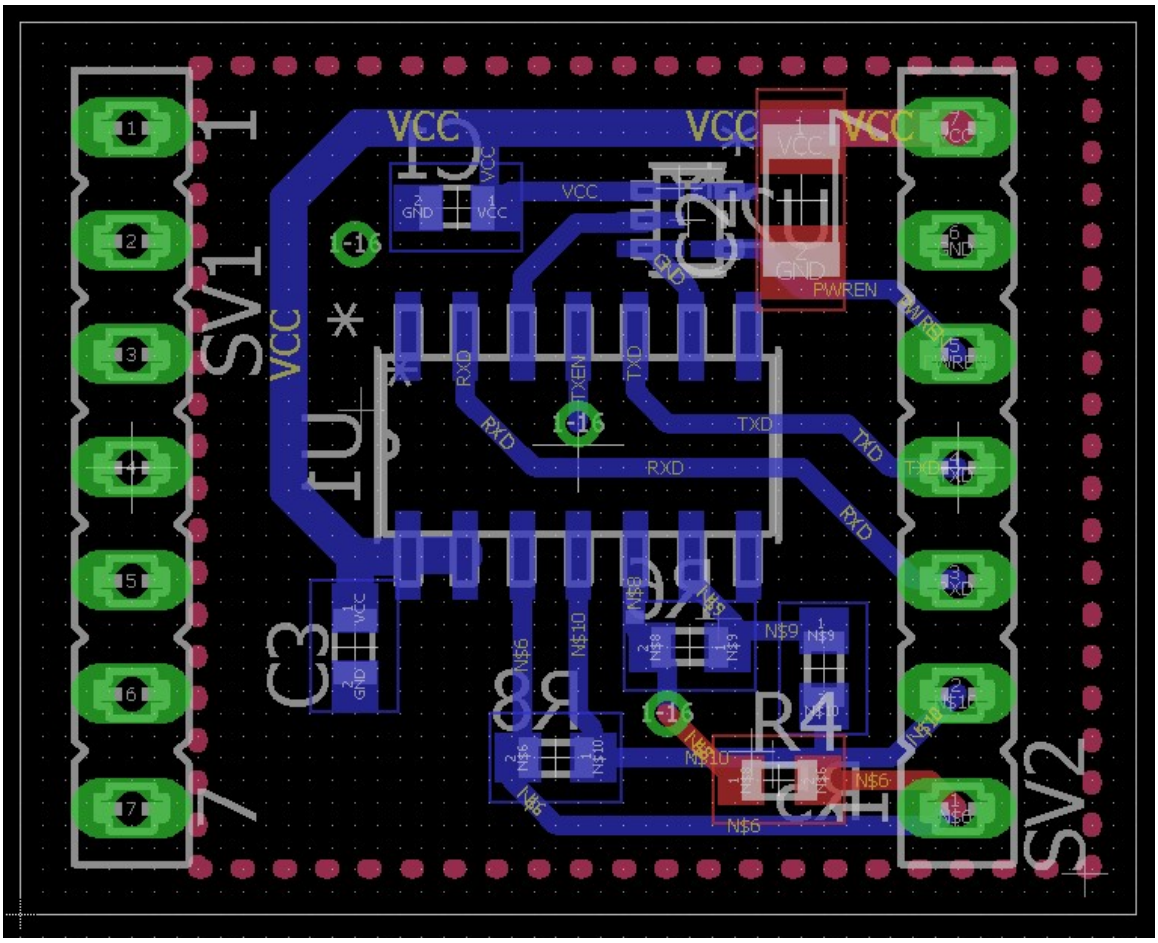

Figure 52 The UART-RS485 layout

Appendix D. Raspberry Pi Code and Example

In this section, it includes the code written for Raspberry Pi using RASPBIAN, which contains a power control script and an example of how to remotely operate Raspberry Pi to turn on all peripherals.

D.1 Raspberry Pi code (powercontrol.py)

```
# import the GPIO and time package

import RPi.GPIO as GPIO

import time



GPIO.setmode(GPIO.BOARD)

GPIO.setup(29,GPIO.OUT)

GPIO.setup(31,GPIO.OUT)

GPIO.setup(33,GPIO.OUT)

GPIO.setup(35,GPIO.OUT)

GPIO.setup(37,GPIO.OUT)

# GPIO.setup(32,GPIO.OUT)

GPIO.setup(36,GPIO.OUT)

GPIO.setup(38,GPIO.OUT)

GPIO.setup(40,GPIO.OUT)



# Signal and pin number

MASTER_EN = 40

LATCH_EN = 36
```

```
LATCH_OE = 37
PM0 = 29
PM1 = 31
PM2 = 33
PM3 = 35
FT_EN = 38

# Power module status class
class PM_Stat:
    def __init__(self):
        self.PM0 = True
        self.PM1 = True
        self.PM2 = True
        self.PM3 = True

# Set power modules based on status
def SetStat(pmstat):
    GPIO.output(LATCH_EN, False)
    GPIO.output(PM0, pmstat.PM0)
    GPIO.output(PM1, pmstat.PM1)
    GPIO.output(PM2, pmstat.PM2)
    GPIO.output(PM3, pmstat.PM3)
    GPIO.output(LATCH_EN, True)
    time.sleep(1)
```

```
    GPIO.output(LATCH_EN, False)
    return


# Display the power module status
def ShowStat(pmstat):
    print "The current power module status is:"
    if (pmstat.PM0 == True):
        print "Peripheral 1 is ON"
    else:
        print "Peripheral 1 is OFF"
    if (pmstat.PM1 == True):
        print "Peripheral 2 is ON"
    else:
        print "Peripheral 2 is OFF"
    if (pmstat.PM2 == True):
            print "Peripheral 3 is ON"
    else:
        print "Peripheral 3 is OFF"
    if (pmstat.PM3 == True):
        print "Peripheral 4 is ON\n"
    else:
        print "Peripheral 4 is OFF\n"
    return
```

```
# Reboot one peripheral
def RebootPM(port, pmstat):
    print "Rebooting:"
    if (port == PM0):
        print "Peripheral 1"
        pmstat.PM0 = False
        SetStat(pmstat)
        pmstat.PM0 = True
    elif (port == PM1):
        print "Peripheral 2"
        pmstat.PM1 = False
        SetStat(pmstat)
        pmstat.PM1 = True
    elif (port == PM2):
        print "Peripheral 3"
        pmstat.PM2 = False
        SetStat(pmstat)
        pmstat.PM2 = True
    elif (port == PM3):
        print "Peripheral 4"
        pmstat.PM3 = False
        SetStat(pmstat)
        pmstat.PM3 = True
    else:
```

```
    print "Port name error: port name should be PM0~PM3"
SetStat(pmstat)
ShowStat(pmstat)
return pmstat
```

D.2 An Operation Example of Raspberry Pi

```
import powercontrol

GPIO.output(MASTER_EN, True)
GPIO.output(LATCH_OE, False)

stat = powercontrol.PM_Stat()

stat.PM0 = True
stat.PM1 = True
stat.PM2 = True
stat.PM3 = True

powercontrol.SetStat(stat)
powercontrol.ShowStat(stat)

GPIO.output(MASTER_EN, False)
GPIO.output(LATCH_OE, True)
```