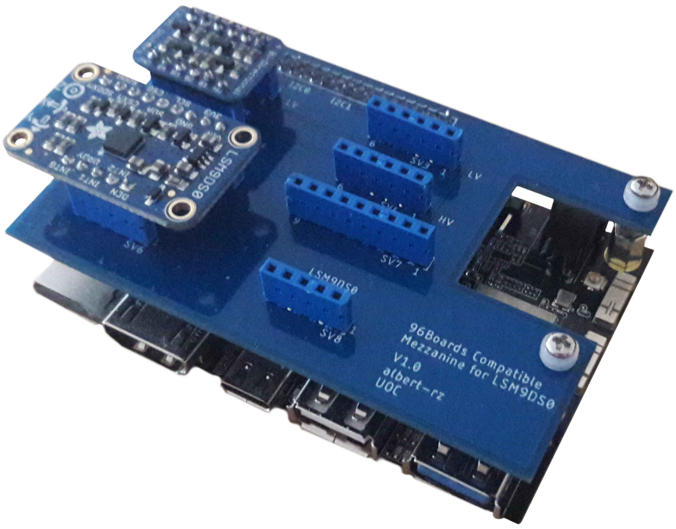


Embedded Linux Driver Development for a 9-DOF Motion Sensor

Final Degree Project

Bachelor's Degree in Computer Engineering

Albert Ruiz Alvarez

Tutor: Joaquin López Sánchez-Montañés

UNIVERSITAT OBERTA DE CATALUNYA
Barcelona, December 2018

List of Figures

List of Tables

Summary

The present document describes the steps taken to develop an embedded Linux driver for a 9-DOF motion sensor. Document is structured as follows.

Chapter 1 introduces the project, describes goals and scope, and includes the list of tasks and the schedule.

Chapter 2 is dedicated to hardware setup (Rock960 board and LSM9DS0 sensor). It explains in detail all steps to upload a Ubuntu Server 64-bit to Rock960 and configure a graphical desktop environment. It also describes how to wire LSM9DS0 to Rock960, and ends with some initial I^2C tests.

Chapter 3 is dedicated to Linux device and driver model: bus drivers, controller drivers, Device Tree... It also analyses how I^2C RK3399 platform driver works and how it is used from user space.

Chapter 4 begins with an introduction to I^2C bus: features, speed, addresses... Then the four I^2C messages that LSM9DS0 supports are described. Finally it explains the user space driver developed for LSM9DS0.

To demonstrate driver functionality, two applications were developed. One was just for unit testing and it was not included in the document. The other one is presented in chapter 5. It is a graphic application that gets data from LSM9DS0 and then updates the orientation of a 3D object.

The document ends with conclusions, bill of materials and bibliography. Some appendices were included with more details on the boards and with some code snippets.

Key words: Embedded Linux, Linux driver, Device Tree, Rock960, LSM9DS0, motion sensor, accelerometer, `libwebsockets`, `JSON-C`, `three.js`.

Contents

1 Preface

1.1 Motivation

Linux kernel was created as a hobby project by a Finnish student, Linus Torvalds, in 1991. From the beginning it was offered as open source code. Quickly it started to be used as the kernel for free software operating systems. Nowadays, thousands of developers (individuals or companies) contribute to Linux kernel and Linux based operating systems.

Among Linux kernel features, the following are particularly significant for embedded devices [1]:

- Portability and hardware support. It runs on most architectures.
- Scalability. It can run on super computers as well as on embedded devices with just 4MB of RAM.
- Security, stability and reliability. The code is reviewed by many experts.
- Modularity. It can include only what a system needs.
- Exhaustive networking support. Documentation has always been available on line. Today there are websites that help to explore and understand Linux kernel intrinsics, such as Elixir.
- Easy to program. Developers can learn from existing code.

The current stable Linux kernel (4.19.12) has more than 60.000 files, and about the 60% of them are drivers. So many drivers files? How is that possible? In the past 20 years there has been a boom of embedded devices. New microprocessors, new SoCs, new sensors, new communication protocols... Linux drivers growth has marched hand in hand with it.

The author of this document had no previous experience in Linux drivers before this project. He had already worked with embedded platforms and had developed drivers for microcontrollers and DSPs (without operating system). But Linux drivers was still a field to discover. Author's motivation was to understand how (and where) Linux drivers interacts with the hardware.

Despite all the documentation available, learning in detail the Linux device and driver model is a major challenge: many layers, many files, many little details... Furthermore, the book that is considered to be the reference book, *Linux Device Drivers*, is outdated. Last edition was in February 2005 (by that time, Linux kernel version was 2.6). And most on line examples are not much more complicated than a simple `Hello world`.

1.2 Project Description

The purpose of this project was to develop a Linux device driver for a 3D motion sensor. The driver was for an ARM-based single board computer running a Linux-based OS. Motion sensing was done with a 9 Degrees of Freedom (DoF) accelerometer/magnetometer/gyroscope with Inter-Integrated

Circuit (I^2C) interface. A demo application was developed to demonstrate driver functionality. It was a graphic application showing changes in the sensor (see figure 1.1).

For the single board computer it was selected the Rock960 board from Vamrs Limited (see figure 1.2a), with a Rockchip RK3399 Sistem on Chip (SoC). For the sensor it was chosen the LSM9DS0 breakout board from Adafruit (see figure 1.2b), which includes the LSM9DS0 iNEMO inertial module from STMicroelectronics.

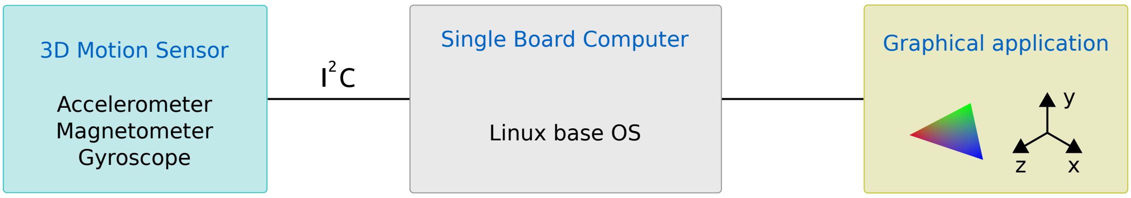
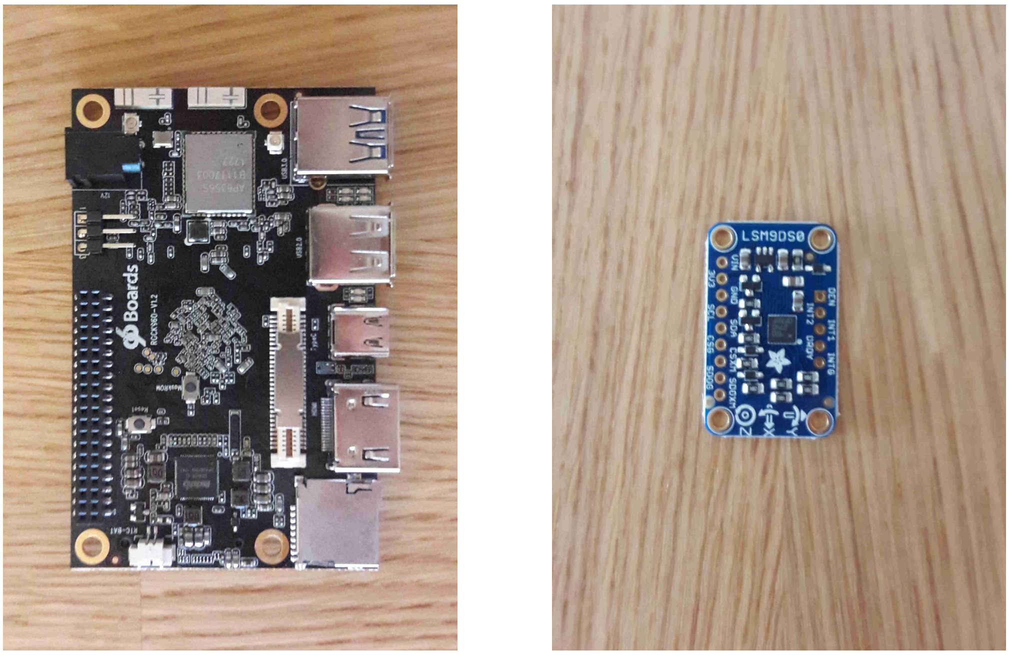


Figure 1.1: Project description



(a) **(b)**

Figure 1.2: (a) Rock960 board, (b) LSM9DS0 board.

1.3 Goals and Scope

The main goals of this project were:

- Understand Linux device and driver model.
- Learn how a driver interacts with the hardware. I^2C RK3399 platform driver was taken as reference.

- Understand the difference between kernel space and user space drivers.
- Develop a user space driver to control an I^{2}C device.
- Develop a demo application that used the driver.

Secondary goals were:

- For the demo application, since 3D motion sensors are used in autonomous machines (robots, drones, rovers...) propose a distributed architecture.
- Follow concepts learned during the bachelor's degree: design patterns, UML diagrams, OS resources...
- Follow Linux kernel C-programming guidelines.
- Do unit testing.

Rock960 is in the state of art of single board computers. Using it is an indirect goal:

- There is a small community of users.
- Not as much as other projects, such as Raspberry PI.
- Cross compile toolchain must be prepared.
- Even if the vendor has already ported the OS there are still some errors.

1.4 Project Tasks and Schedule

Table 1.1: Project task list.

A - Project setup

A1 Target setup

- Boot Rock960 from SD card
- Flash OS to Rock960 's eMMC memory and boot

A2 Host setup and first binary

- Prepare cross compiler toolchain
- Program a basic `Hello world` and run it on Rock960

A3 Testing I^{2}C interface

- Rock960 to LSM9DS0 wiring
- Test Rock960 to LSM9DS0 I^{2}C interface
- Create a custom made mezzanine for LSM9DS0

A4 First OpenGL ES 2.0 binary

- Program a basic `Hello world` and run it on Rock960

B - Linux drivers for Rock960

B1 Introduction

- Classes of devices and modules
- In-tree vs. out-of-tree drivers
- Kernel-space vs. user-space drivers
- About device drivers programming
- About Device Tree
- Analysis of Rock960 platform driver

B2 Module test

- Get Linux kernel sources
- Build the kernel
- Program a basic `Hello world` module and run it on Rock960

B3 I^{2}C subsystem

- Introduction to I^{2}C bus
- RK3399 I^{2}C interface and register map
- `i2c_dev` module analysis
- `rk3x_i2c` module analysis

C - LSM9DS0 driver development

C1 Introduction

- LSM9DS0 functionality and operation modes
- LSM9DS0 register description
- I^{2}C messages

C2 Driver development

- Functionality definition
- Structure definition
- Driver programming
- Unit tests development

C3 Testing

D - Final application

D1 Graphical engines evaluation

- Evaluate Raylib graphical engine
- Evaluate web-based graphical enignes

D2 Final application development

- Define a layered architecture
- LSM9DS0 layer development
- `libwebsockets` cross compilation
- `libjson-c` cross compilation
- Graphic application development

D3 Testing

E - Documentation

E1 Project report

E2 Video presentation

E3 Virtual defense

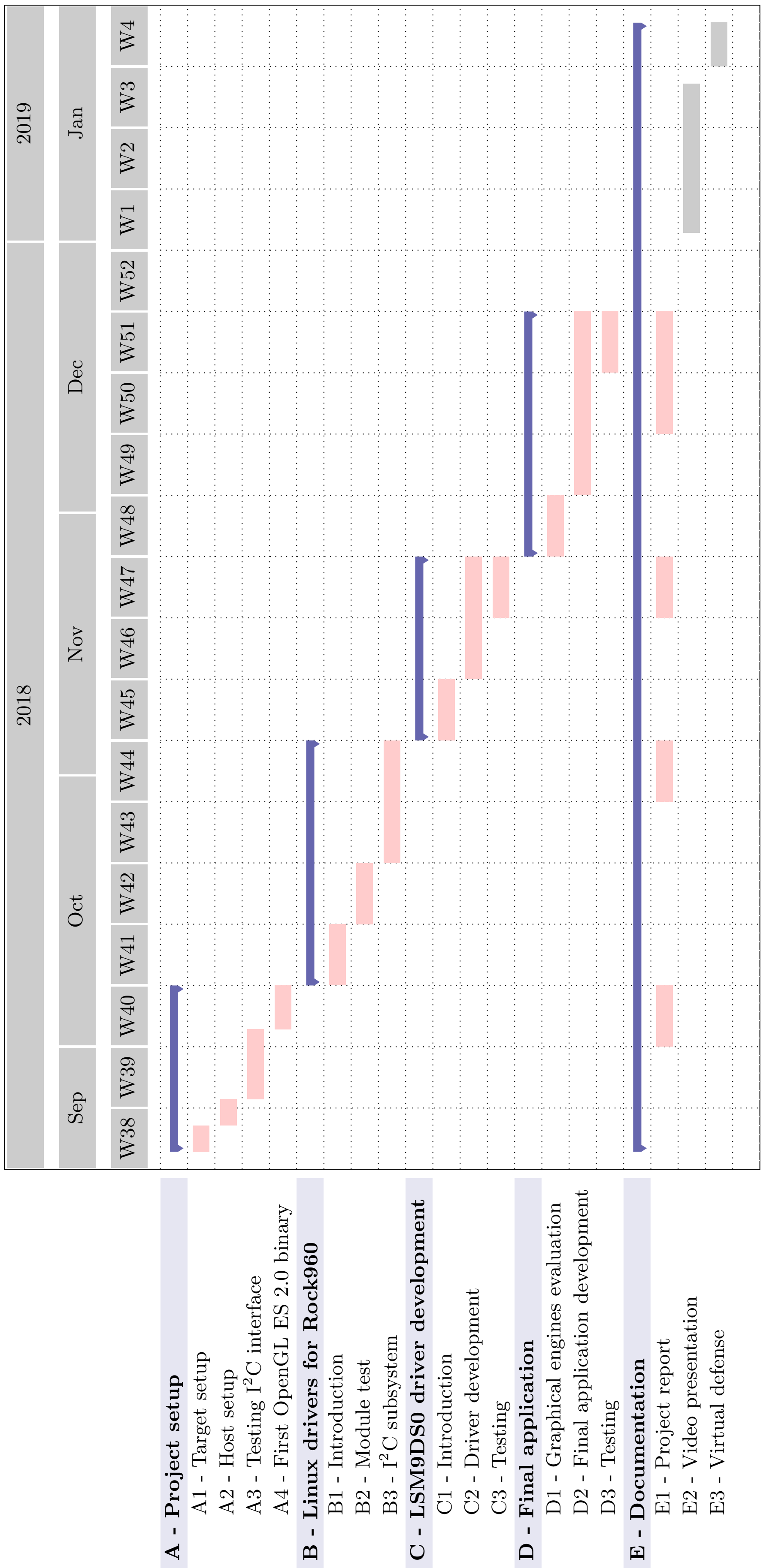


Figure 1.3: Project Gantt char.

2 Project Setup

2.1 Introduction to Linaro and 96Boards

Linaro is an open consortium founded in June 2010 by ARM processors manufacturers and communities, to tackle the four main problems of embedded Linux [2]:

- Under investment in many of the open source projects that make up a Linux platform.
- Distribution fragmentation.
- Lack of efficient SoC integration. Most SoC vendors have different approaches for power management, graphics and multimedia.
- Not enough optimization. Lots of features in latests processors may not be used.

It is a common mistake to think that Linaro is a Linux based distribution. It is not. It is an engineering organization that works on free and open source software for the ARM architecture based processors, including the Linux kernel, GCC, power management, graphics and multimedia. Linaro does not participate in the development of Debian, Ubuntu, Android... but works to make them work in ARM platforms. This may include: extend and improve GCC, drivers migration, new patches...

In February 2015 Linaro announced 96Boards initiative. 96Boards is a set of hardware specifications to make latest ARM based processors available to developers at a reasonable cost [3]. Specifications are open and independent of any specific SoC.

Today there are more than 15 boards within 96Boards community using state of the art SoC, such as: Rockchip RK3399, Qualcomm Snapdragon 410E and SnapDragon 820E, or Xilinx Zynq UltraScale+ MPSoC ZU3EG. All boards run Android and/or a Linux based distribution (mostly Debian and Ubuntu).

2.2 Getting started with Rock960

For this project 96Boards Rock960 board was chosen. Rock960 was developed by Vamrs Limited, and it is based on RK3399 SoC. RK3399 is the latest high-performance SoC developed by Chinese manufacturer Rockchip (Fuzhou Rockchip Electronics Co., Ltd.). Table 2.1 summarizes Rock960 most relevant features and figure 2.1 show its main components (for a detailed description see schematics in appendix C.1).

Table 2.1: Rock960 board features

Component	**Description**
SoC	Rockchip RK3399
CPU	ARM Cortex-A72 Dual-core (up to 1.8GHz) + ARM Cortex A53 Quad-core (upt to 1.4GHz)
GPU	ARM Mali T860MP4
RAM	4GB LPDDR3
Storage	32GB eMMC 5.1
WiFi	WLAN 802.11 ac/a/b/g/n, 2.4GHz and 5GHz, on board antenna
Bluetooth	4.2, on board antenna
USB	1 x USB 3.0 (type A), 1 x USB 2.0 (type A, host mode only) and 1 x USB 3.0 (type C OTG)
Display	1 x HDMI 2.0 type A (up to 4K x 2K at 60Hz), 1 x 4L-MIPI DSI (up to 1080p at 60Hz), 1 x DP 1.2 (type C, up to 4K x 2K at 60Hz)
Audio	HDMI output
Expansion	40 pin low speed expansion connector (+1.8V, +5V, DC power, GND, 2 x UART, 2 x I^2C, SPI, 2 x I^2S, 12 x GPIO) and 60 pin high speed expansion connector (4L-MIPI DSI, I^2C x 2, SPI (48M), USB 2.0, 2L+4LMIPI CSI)
Button	Reset button and recovery button
Debug	Rock960 exports a dedicated TTL serial console
Power source	+12V and 2A, DC, 4.0 mm Jack connector
OS support	AOSP, Debian, Ubuntu
Size	85mmx55mm

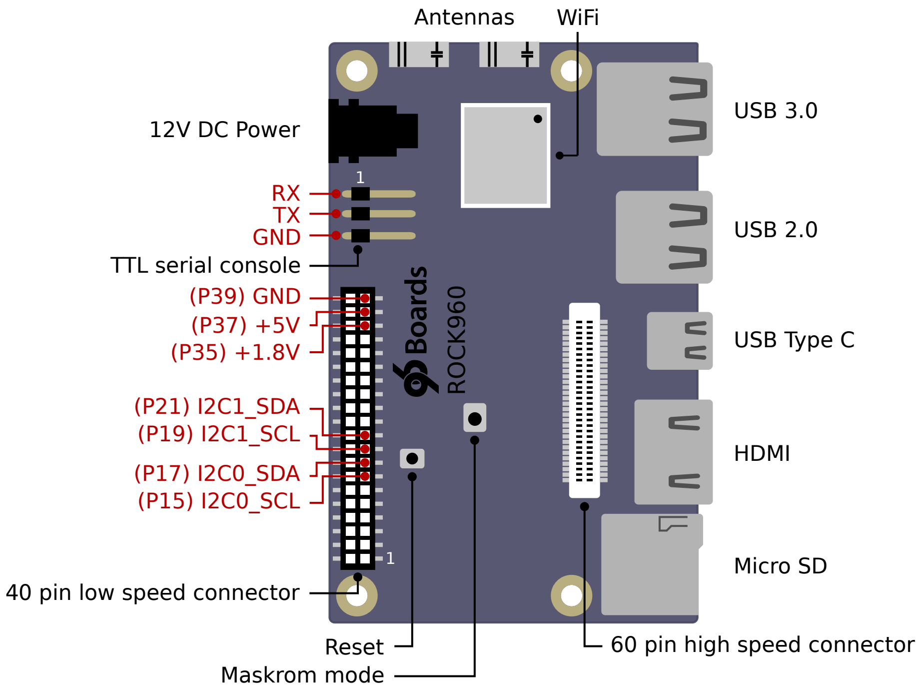


Figure 2.1: Rock960 main components.

Rock960 can run any of the following OS: Debian, Ubuntu or Android. It comes with Android pre-installed. However, OS can be switched and/or updated.

There are two ways to upload OS onto Rock960:

- From SD card - The firmware is written on an SD card. Rock960 will run the OS from the SD card.
- From eMMC memory - The firmware is downloaded to the Rock960 eMMC memory via the USB.

For this project the Ubuntu Server 64-bit image was used and it was saved in the eMMC memory. To download the image it was used `rkdeveloptool` and `Rockusb` [4], which are vendor specific tools.

Rock960 has two expansion I/O connectors:

- Low speed connector - A low profile 40 way female header for maker/community use (see the configuration of the connector in table 2.2, and for a detailed pin description see appendix B).
- High speed connector - A low profile 60 way high speed female module header for advanced maker/OEM use with high speed interfaces, including MIPI-DSI, USB and HSIC.

Table 2.2: Low speed connector pinout.

Name	Pin number	Pin number	Name
GND	1	2	GND
UART0_CTS	3	4	PWR_BTN_N
UART0_TxD	5	6	RST_BTN_N
UART0_RxD	7	8	SPI0_SCLK
UART0_RTS	9	10	SPI0_DIN
UART1_TxD	11	12	SPI0_CS
UART1_RxD	13	14	SPI0_DOUT
I2C0_SCL	15	16	PCM_FS
I2C0_SDA	17	18	PCM_CLK
I2C1_SCL	19	20	PCM_DO
I2C1_SDA	21	22	PCM_DI
GPIO-A	23	24	GPIO-B
GPIO-C	25	26	GPIO-D
GPIO-E	27	28	GPIO-F
GPIO-G	29	30	GPIO-H
GPIO-I	31	32	GPIO-J
GPIO-K	33	34	GPIO-L
$+1.8V$	35	36	SYS_DCIN
$+5V$	37	38	SYS_DCIN
GND	39	40	GND

2.3 Setting up Rock960

This section explains how to install Ubuntu Server 64-bit image, configure WLAN via serial port, add a new user add a graphical desktop

2.3.1 Ubuntu Server 64-bit Installation

Step 1: Set up `rkdeveloptool`

Some dependencies must be first installed:

```
sudo apt-get install libudev-dev libusb-1.0-0-dev dh-autoreconf
```

Then source code must be downloaded and build like:

```
git clone https://github.com/rockchip-linux/rkdeveloptool
cd rkdeveloptool
autoreconf -i
./configure
make
```

`rkdeveloptool` executable is at the current directory. Optionally, it may be copied to normal binary directory:

```
  sudo cp rkdeveloptool /usr/local/bin/
```

Step 2: Download OS image

Download the image from 96 Boards website or from Vamrs repository and extract it. The tarball includes:

- A binary that runs on Rock960 to initiate DRAM and flash (file with extensión `.bin`).
- Image with boatloader, kernel and rootfs (file with extension `.img`).
- A `README` file.

For this project image `rock960_model_ab_ubuntu_server_16.04_arm64_20181001.tar.gz` was used.

Step 3: Boot Rock960 into maskrom mode

In this mode, the RK3399 processor is waiting for commands from USB. To put the device into maskrom mode:

1. Power on Rock960.
2. Connect Rock960 to a Linux host with USB type A to type C cable.

3. Press and hold maskrom button (see figure 2.1).
4. Short press reset button.
5. Release maskrom button.

On the Linux host, `lsusb` should show the following VID/PID[1] if the board is in maskrom mode (see figure 2.2):

```
Bus 001 Device 008: ID 2207:330c
```

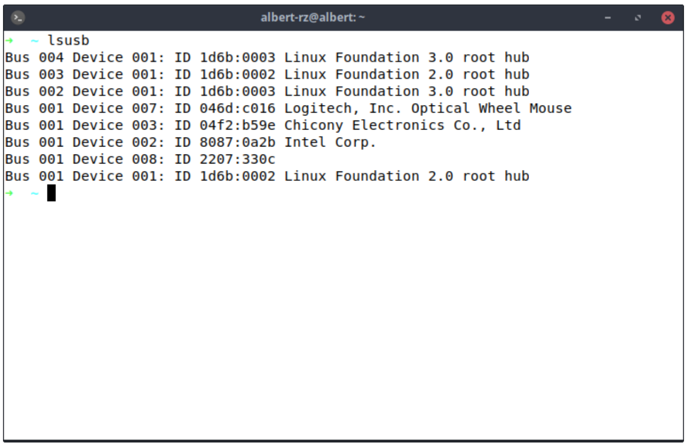


Figure 2.2: Traces showing Rock960 is in maskrom mode.

Step 4: Prepare to flash

Init DRAM with:

```
sudo rkdeveloptool db rk3399_loader_v1.12.112.bin
```

Step 5: Flash image onto eMMC

The image is written to the flash with:

```
sudo rkdeveloptool wl 0 rock960_model_ab_ubuntu_server_arm64_20181001-1845-gpt.img
```

This may take some time. Once it finishes:

```
sudo rkdeveloptool rd
```

Finally, short press the reset button again to reboot Rock960 with the new image.

[1] ID may be slightly different.

2.3.2 Serial Port configuration

The installed image did not have a graphic environment and was not configured to connect to any WLAN yet. Some of the following actions had thus to be done using the serial port. Figure 2.3 shows how to connect USB-to-TTL cable to Rock960 serial connector.

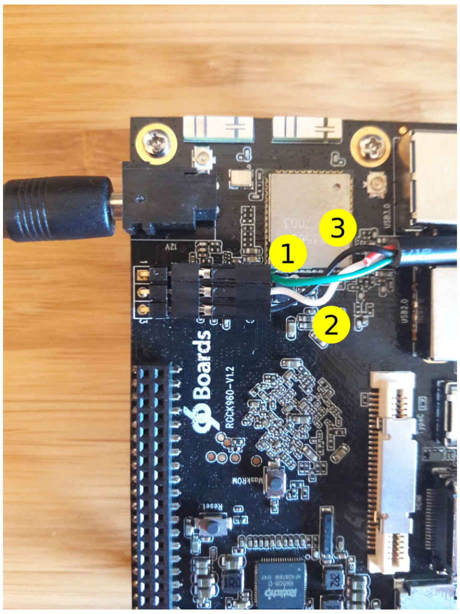


Figure 2.3: Serial cable connection - (1) Receive, (2) Transmit, (3) Ground.

In the PC side, `minicom` was used to communicate with Rock960:

```
sudo minicom -D /dev/ttyUSB0
```

Serial port was configured with:

Table 2.3: Serial port configuration.

Parameter	Value
Bit rate	1500000 bits/s
Configuration in asynchronous mode	8N1 (8 data bits, no parity bit, 1 stop bit)
Hardware flow control	No
Software flow control	No

2.3.3 Resizing `/dev/root` File System

After OS installation, it was checked the free memory with (see figure 2.4):

```
free -h
df -h
```

Existing partitions were also checked with (see figure 2.5):

```
fdisk -l
```

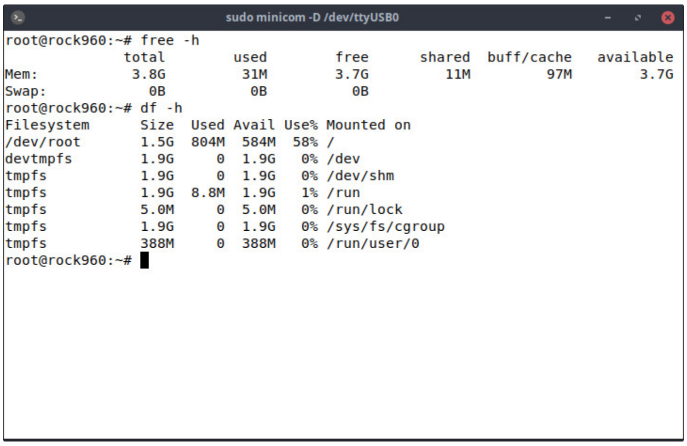


Figure 2.4: Initial file system size.

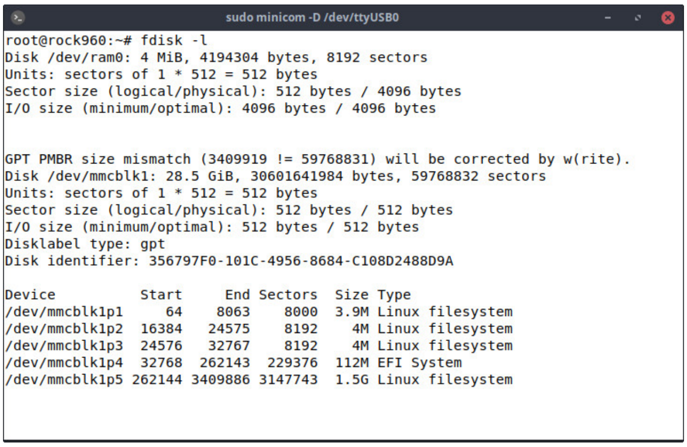


Figure 2.5: Initial partition list.

It was found out that `/dev/root` filesystem (in `/dev/mmcblk1p5`) was only $1.5GB$, while it should be about $30GB$. This meant that most of eMMC memory was unused.

To fix it, part 5 in `mmcblk1` was resized (see figure 2.6):

```
parted /dev/mmcblk1

(parted) print
(parted) resizepart 5

End? -34s
quit
```

Then the file system was extended:

```
resize2fs /dev/mmcblk1p5
```

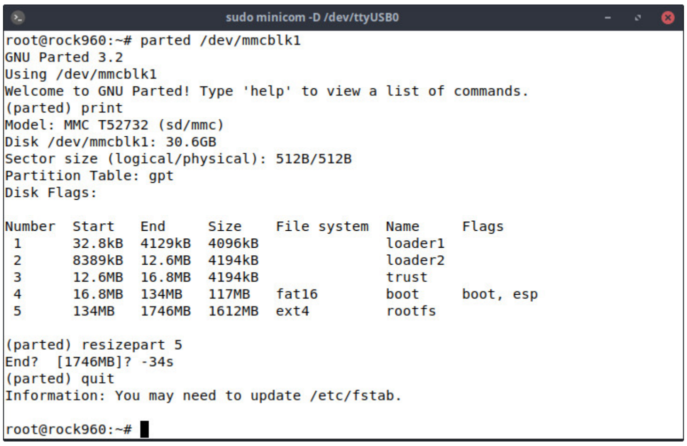


Figure 2.6: Resize `/dev/root` partition.

After that `/dev/root` size was about $28GB$ (see figure 2.7):

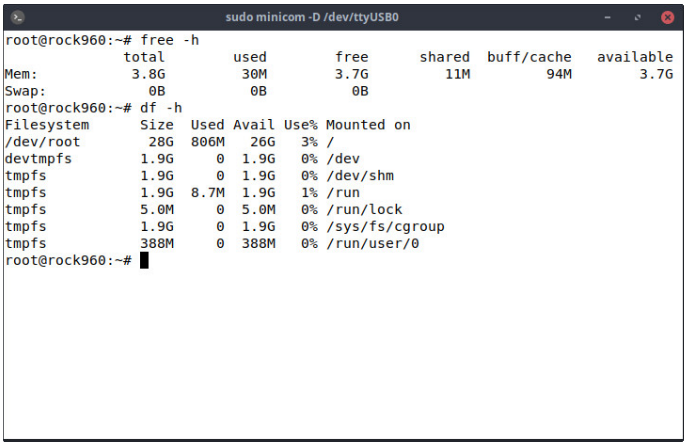


Figure 2.7: Final file system size.

2.3.4 WLAN Connection Setup

The image installed already included `NetworkManager` and `nmcli`. `NetworkManager` is a Linux daemon providing a high-level interface for the configuration of the network interfaces. `nmcli` is a command-line interface for `NetworkManager`.

WLAN connection setup was done using Rock960 serial port interface. To see all available WLANs:

```
nmcli device wifi rescan
nmcli device wifi list
```

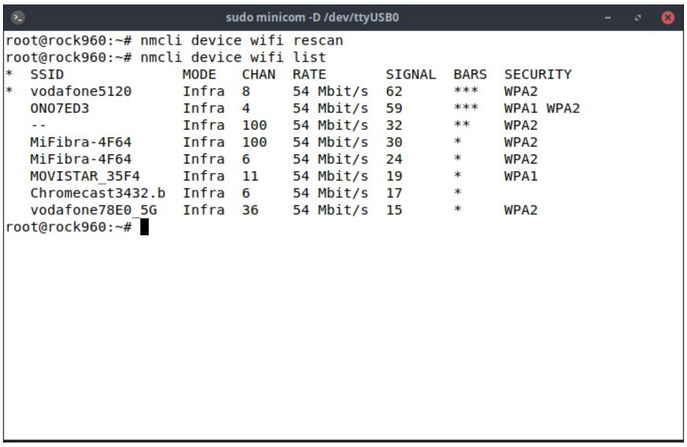


Figure 2.8: Available WLANs using `nmcli`.

WLAN was configured like:

```
nmcli con add con-name WiFi ifname wlan0 type wifi ssid vodafone5120
nmcli con modify WiFi wifi-sec.key-mgmt wpa-psk
nmcli con modify WiFi wifi-sec.psk PLAGELALONDE15!
nmcli con up WiFi
```

To see all connections:

```
nmcli con show
```

To see connection status:

```
nmcli device
```

2.3.5 Adding a New User

As explained in the `README` file in the tarball, the image already includes two pairs user/password:

- root/root
- rock/rock

A new user was added like:

```
adduser albert-rz
adduser albert-rz sudo
```

The installed Ubuntu image provided SSH access to new users (see figure 2.9).

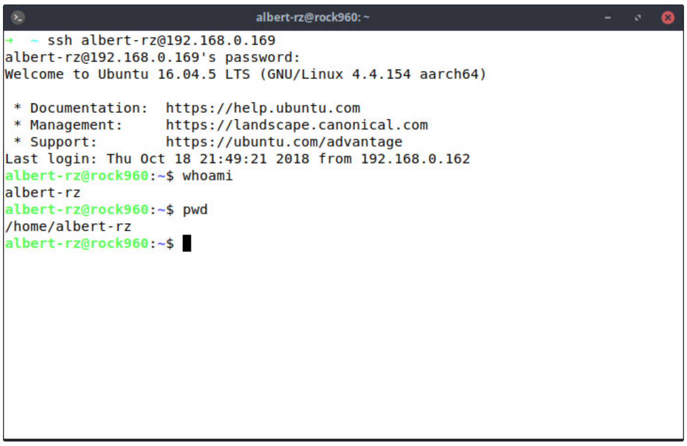


Figure 2.9: SSH connection to Rock960.

By default, the `/etc/hosts` and `/etc/hostname` were empty. As a consequence, when using `sudo` with some command lines, the following message was shown:

```
sudo: unable to resolve host
```

To fix it, a new hostname was defined: `albert-rz`. `/etc/hosts` was edited like:

```
127.0.0.1 localhost
127.0.1.1 albert-rz
```

and `/etc/hostname` was edited with:

```
albert-rz
```

2.3.6 Graphical Desktop Environment Installation

Xfce was installed with:

```
apt-get install apt-utils
apt-get install dialog
apt-get install xfce4 slim
```

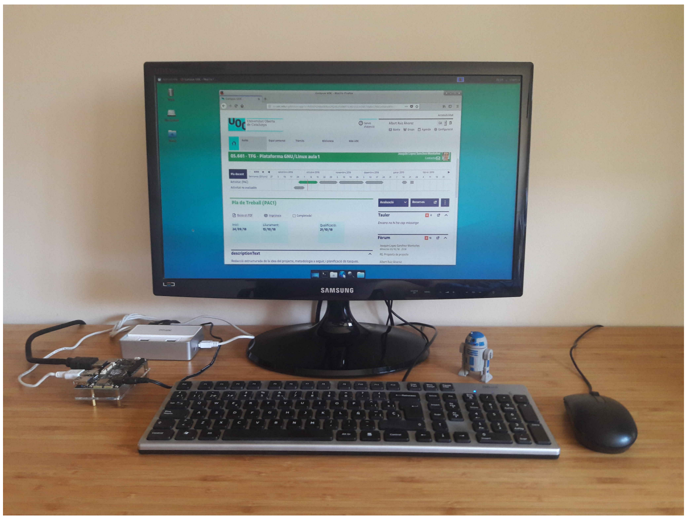


Figure 2.10: Ubuntu OS running on Rock960.

In general, before installing any desktop environment it is strongly recommended to do an update:

```
apt-get update
apt-get upgrade
```

2.4 Getting started with LSM9DS0

LSM9DS0 is a system-in-package with a digital linear acceleration sensor, a digital magnetic sensor and a digital angular rate sensor [5]. Sensor features include:

- 3 acceleration channels, 3 magnetic field channels and 3 angular rate channels.
- Configurable acceleration full scale: $\pm 2g$, $\pm 6g$, $\pm 6g$, $\pm 8g$ and $\pm 16g$.
- Configurable magnetic full scale: $\pm 2G$, $\pm 4G$, $\pm 8G$ and $\pm 12G$.
- Configurable angular rate full scale: $\pm 245 deg/s$, $\pm 500 deg/s$ and $\pm 2000 deg/s$[2].
- 16-bit data output.
- I^{2}C serial interface supporting standard mode ($100kHz$) and fast mode ($400kHz$).
- SPI serial standard interface.
- Analog supply voltage from $+2.4V$ to $+3.6V$.
- Normal mode and power-down mode (low power mode).
- Programmable interrupt generators.
- Configurable embedded FIFO.

Adafruit's LSM9DS0 breakout board features include:

- Analog supply voltage from $+3V$ to $+5V$. It includes a voltage regulator.
- I^{2}C serial interface ($+3V$ to $+5V$ logic).
- SPI serial interface ($+3V$ to $+5V$ logic).
- $10k\Omega$ pull-up resistors for both I^{2}C SDA and SCL signals.
- One input interrupt pin.
- Three output interrupt pins.
- Size: $20mm$ x $33mm$.
- Board pins separation is $2.54mm$ (compatible with most bread boards).

Figure 2.11 shows LSM9DS0 breakout board pinout and table 2.4 describes pins functionality (for a detailed description, see schematics in appendix C.2).

[2] Degrees per second. In some papers it may appear as *dps*.

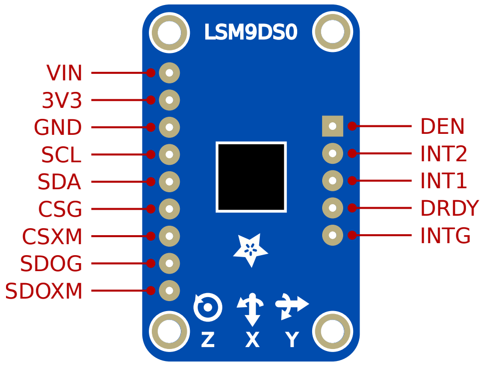


Figure 2.11: LSM9DS0 board pinout.

Table 2.4: LSM9DS0 board pinout.

Connection	Description	Voltage logic
VIN	Power input, from $+3V$ to $+5V$	-
3V3	$+3.3V$ output	-
GND	Common ground	-
SCL	I^2C clock pin or SPI clock pin	$+3V$ to $+5V$
SDA	I^2C data pin or SPI MOSI pin	$+3V$ to $+5V$
CSG	SPI chip select for gyroscope	$+3V$ to $+5V$
CSXM	SPI chip select for accelerometer and magnetometer	$+3V$ to $+5V$
SDOG	SPI MISO pin for gyroscope	$+3V$ to $+5V$
SDOXM	SPI MISO pin for accelerometer and magnetometer	$+3V$ to $+5V$
DEN	Gyroscope data enable	$+5V$
INT1	Accelerometer and magnetic sensor interrupt 1	$+3V$
INT2	Accelerometer and magnetic sensor interrupt 2	$+3V$
DRDY	Gyroscope data ready	$+3V$
INTG	Gyroscope programmable interrupt	$+3V$

After checking LSM9DS0 datasheet and board schematics (see C.2), it was concluded that some board connections could be left unconnected without any risk: INT1, INT2, INTG, DEN and DRDY.

INT1, INT2 and INTG are input/output interrupts that may be used for several purposes. For instance:

- LSM9DS0 may be configured to boot up with an interrupt signal in INT1.
- To inform about when data is ready to be read.
- To notify an overrun in the internal FIFO.
- To notify that the internal FIFO is empty.

However, all these functionalities are disabled by default and must by enabled by software. This project did not require any of them.

CSG, CSXM, SDOG and SDOXM are SPI pins. In LSM9DS0 board they are connected to $+3.3V$ with pull-up resistors. DEN is connected to DEN_G chip pin and this one is connected to $+3.3V$ with a pull-up resistor too. Because pull-up resistors guarantee enough voltage stability, these pins were left unconnected.

DRDY is an output pin. If user is careful enough and does not directly touch it, it might be left unconnected too.

2.5 Setting Up the Development Environment

2.5.1 Installing Cross Compiler for ARM Platforms

When an application is compiled (in a host machine), the compiler must be told the architecture of the machine the application will run on (the target). If the architectures of host and target are equal, then compilation is said to be native compilation. When they are different, it said to be cross compilation [6] (see figure 2.12).

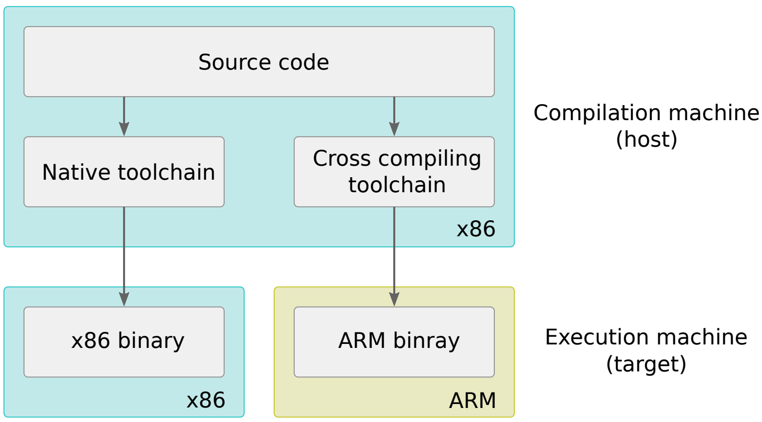


Figure 2.12: Native compilation and cross compilation (source: Bootlin).

Modern embedded platforms and single board computers (such as 96Boards, Raspberry PI, Asus Tinker board, Odroid...) have enough memory resources and computing power to compile large code projects. However, cross compilation is still a better option because:

- Host machines have faster processors, with more cores and more cache memory.
- ROM/RAM memory in host machines have faster access times.
- The number of cycles in eMMC/SD memories in target machines is significantly smaller than in host machines.

The main disadvantage of cross compiling is that a toolchain must be prepared for each platform. There are three solutions:

1. Use a vendor proprietary system development kit, such as MontaVista or Wind River.
2. Build the toolchain manually. This option may be a difficult task and may take some days: there are lots of details to learn, source files are needed, some patches may be needed...
3. Use an open source toolchain building utility, such as Crosstool-ng, Buildroot or OpenEmbedded.

For this project a minimal toolchain was built manually. In order to build the driver and the demo application, the following packages were needed:

- The GNU C++ compiler for arm64 architecture.
- `libwebsockets` library.
- `JSON-C` library.

The GNU C++ compiler for arm64 architecture was be installed with:

```
sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
sudo apt-get install build-essential autoconf libtool cmake pkg-config git python-dev
    swig3.0 libpcre3-dev nodejs-dev
```

`libwebsockets` and `JSON-C` were not used by LSM9DS0 driver, but by demo application. They were cloned from GitHub and then cross compiled as static libraries (see appendix A.2 and appendix A.3). The sole limitation was that `libwebsockets` had to be compiled without SSL support. Adding SSL would have required downloading and compiling OpenSSL too, and that was considered unnecessary for a demo application.

2.5.2 Compiling the first `Hello world`

Once the toolchain was ready, a basic `Hello world` was compiled and tested on Rock960 (see figure 2.13, for source code and `Makefile` see appendix A.1).

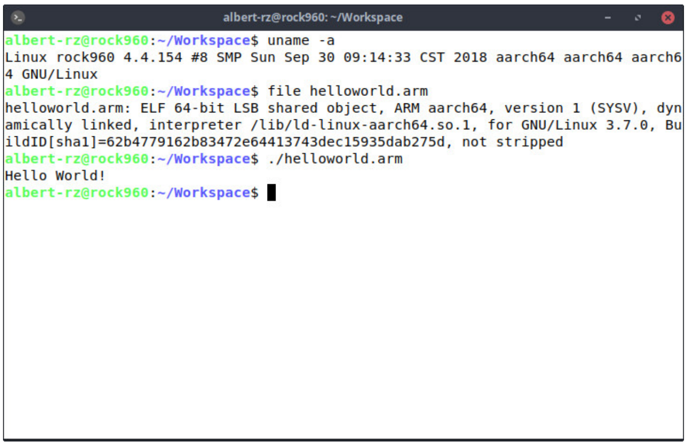


Figure 2.13: `Hello world` example running on Rock960.

2.6 Wiring Rock960 and LSM9DS0

2.6.1 Connecting I^{2}C Devices with Different Voltage Levels

I^{2}C is a synchronous serial computer bus using two bus lines: a serial data line (SDA) and a serial clock time (SCL). Both SDA and SCL are bidirectional lines connected to a positive supply via pull-up resistors (see figure 2.14). Thus, when the bus is free, SDA and SCL are HIGH.

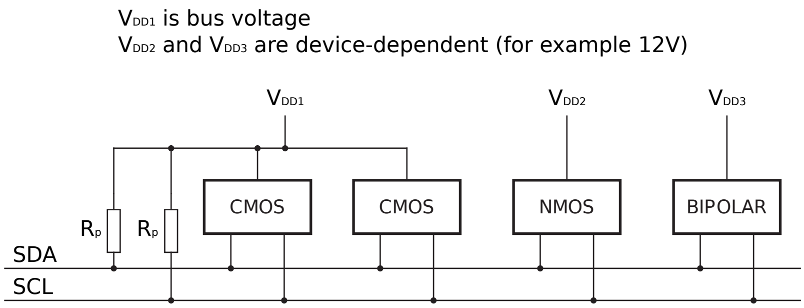


Figure 2.14: Devices with various supply voltages sharing the same bus (source: NXP Semiconductor).

Devices connected to the bus may use different technologies (CMOS, NMOS, bipolar...) and different voltage supply levels ($+1.8V$, $+3V$, $+3.3V$, $+5V$...). Because of that, logical voltage levels LOW and HIGH are not fixed by the I^{2}C standard, and depend on the bus voltage $\mathrm{V_{DD}}$($\mathrm{V_{DD1}}$ in figure 2.14):

- Logical LOW voltage level is 30% of $\mathrm{V_{DD}}$.
- Logical HIGH voltage level is 70% of $\mathrm{V_{DD}}$.

Different voltage devices may be connected to the same I^{2}C bus using bidirectional level shifters [7]. Figure 2.15 shows such configuration. As depicted, there is $+3.3V$ section and a $+5V$ section, each one with its own pull-up resistors.

I2C0 and I2C1 in Rock960 low speed connector were $+1.8V$ compatible, but LSM9DS0 I^{2}C used $+3V$ to $+5V$ logic. Thus, a voltage level shifter was necessary.

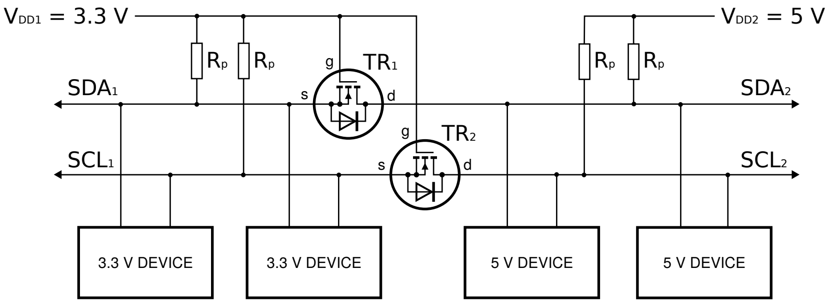
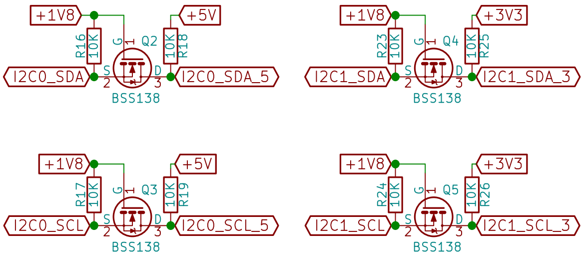


Figure 2.15: Voltage shifting for I^{2}C buses (source: NXP Semiconductor).

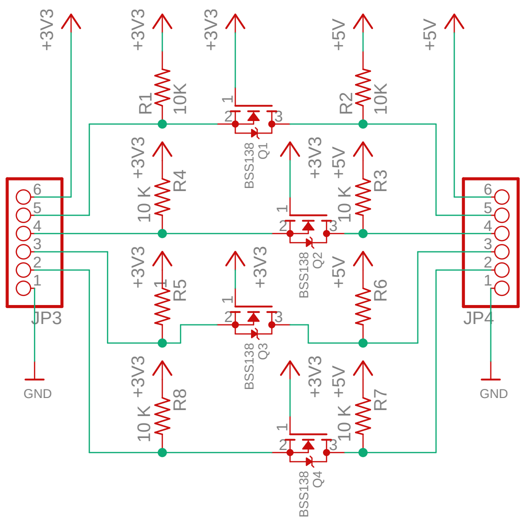
2.6.2 Voltage Level Shifter

There are some mezzanines compatible with 96Boards. Mezzanines are pluggable boards (the same idea as Arduino shields). The sensor mezzanine board [8] uses BSS138 MOSFET transistors to shift I^{2}C bus voltage from $+1.8V$ to $+5V$ (I2C0) and $+3V$ (I2C1) (see figure 2.16a).

For this project, the I^{2}C logic level converter from Adafruit was used [8]. This board also used BSS138 MOSFET transistors to shift voltages (see figure 2.16b).



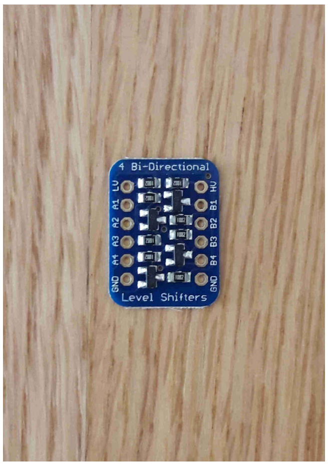
(a)



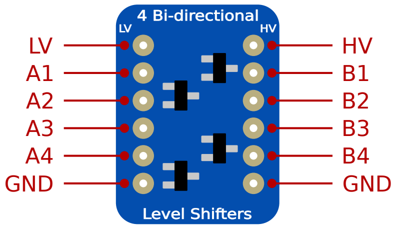
(b)

Figure 2.16: (a) I^{2}C voltage level shifters used in 96Boards sensor mezzanine (source: 96Boards), (b) Adafruit I^{2}C voltage shifter board (source: Adafruit).

Appendix C.4 includes sensor mezzanine schematics. Appendix C.3 includes Adafruit board schematics.



(a)



(b)

Figure 2.17: Adafruit I^{2}C voltage level shifter - (a) Bidirectional voltage level shifter, (b) Pinout.

2.6.3 Wiring scheme

I2C0 and I2C1 in low speed connector are respectively connected to I2C6 and I2C1 ports in RK3399 (see figures 2.18a and 2.18b).

Figure 2.19 is the wiring scheme, and figure 2.20 shows how LSM9DS0 was connected to Rock960 using the voltage level.

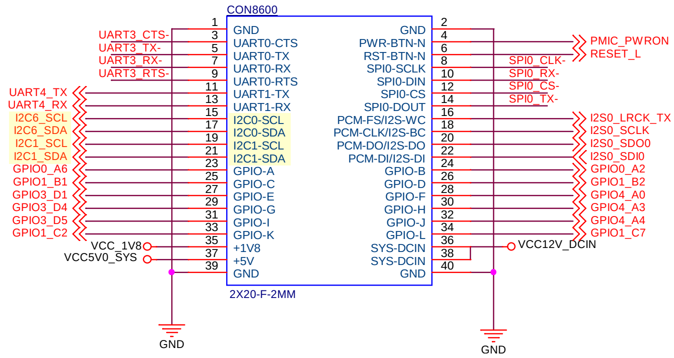
2.6.4 LSM9DS0 Custom Made Mezzanine

Setup in figure 2.20 was fine only for basic testing. However, because bread board pins did not hold wires properly, it was decided to develop and build a PCB with LSM9DS0 that would piggyback onto Rock960.

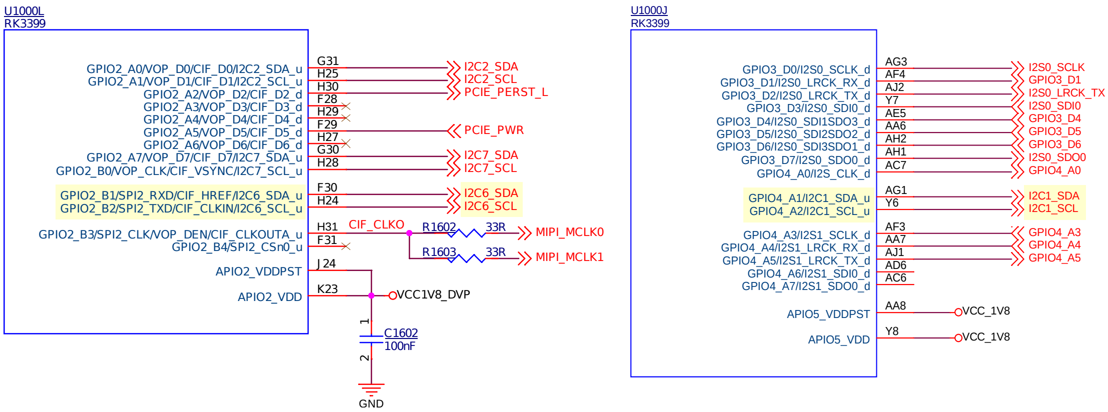
There exists a community that maintains a GitHub repository with templates to create mezzanines according to 96Boards specifications[3]. There are templates for different schematic and layout applications, such as Altium, EagleCAD, gEDA and KiCad. For this project it was used EagleCAD template.

Figure 2.21 shows the mezzanine PCB with LSM9DS0 and voltage shifter, mounted on Rock960 (for schematics and layout, see appendix C.5).

[3] https://github.com/96boards/mezzanine-community.git



(a) 40 pin low speed connector



(b) RK3399

Figure 2.18: I2C0 and I2C1 interfaces connections from low speed connector to RK3399 (source: Vamrs Limited)

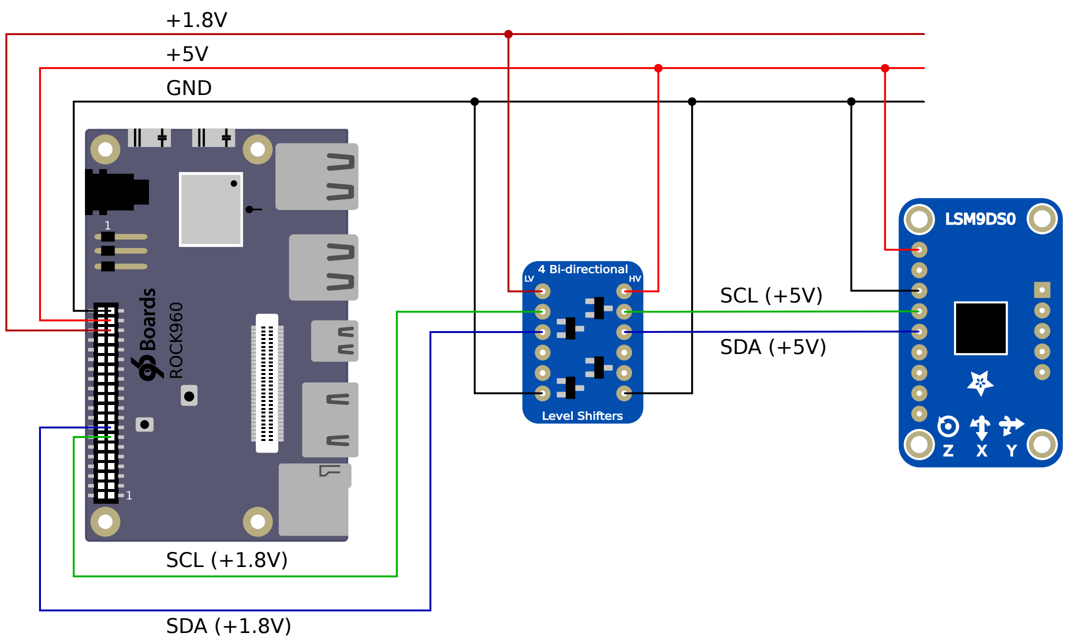


Figure 2.19: Wiring scheme.

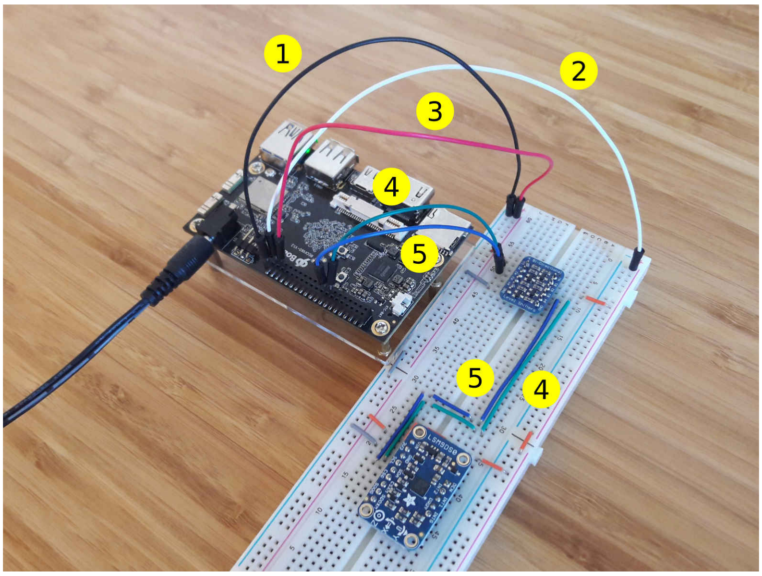


Figure 2.20: LSM9DS0 to Rock960 wiring - (1) GND, (2) +5 V, (3) +1.8 V, (4) SCL, (5) SDA.

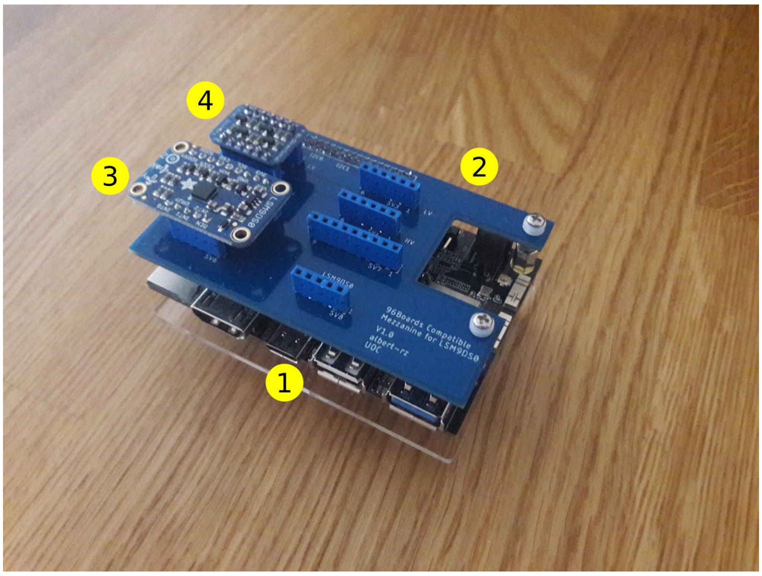


Figure 2.21: Mezzanine mounted on Rock960 : (1) Rock960, (2) Mezzanine, (3) LSM9DS0, (4) Voltage shifter.

2.7 Testing Rock960 to LSM9DS0 I^{2}C Interface

2.7.1 Linux `i2c-tools`

I^{2}C interface was tested using Linux `i2c-tools` package. This package has some programs to interact with I^{2}C devices: `i2cdetect`, `i2cget`, `i2c-stub-from-dump`, `i2cdump`, `i2cset` and `i2ctransfer`.

`i2cdetect` was used in Rock960. As depicted in figure 2.22, two slave addresses were found in I2C6 port (I2C0 in low speed connector).

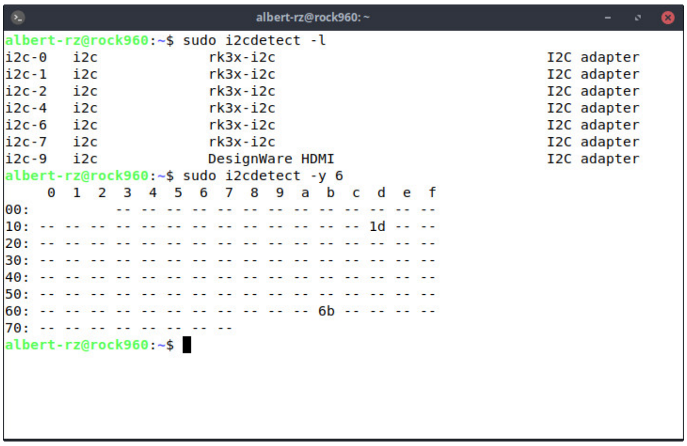


Figure 2.22: Detection of the two LSM9DS0 I^{2}C addresses connected to I2C0 port.

As it will be seen in 4.1, LSM9DS0 has two I^{2}C slaves internally:

- All registers related to accelerometer and magnetometer are accessed via 0x1D I^{2}C address
- All registers related to gyroscope are accessed via 0x6B I^{2}C address

2.7.2 Reading `WHO_AM_I_XM` and `WHO_AM_I_G` registers

For testing purposes, LSM9DS0 includes two read registers (one per I^{2}C slave address): `WHO_AM_I_XM` and `WHO_AM_I_G`. They return a fixed value, regardless of sensor configuration.

Table 2.5 includes all details of these registers. They may be read with `i2cget` (see figure 2.23).

Table 2.5: LSM9DS0 registers for testing.

Register	Register address	Slave address	Default value
`WHO_AM_I_XM`	0x0F	0x1D	01001001b (0x49)
`WHO_AM_I_G`	0x0F	0x6B	11010100b (0xD4)

Figure 2.23: Reading `WHO_AM_I_XM` and `WHO_AM_I_G` with `i2cget`.

3 Introduction to Linux Drivers

3.1 Introduction to Linux Device and Driver Model

The Linux device and driver model is a universal way of organizing devices and drivers into buses [9]. It was added to Linux kernel 2.6 to provide a single mechanism to represent devices and describe their topology in the system, providing:

- Clean code organization: device drivers are separated from controller drivers, hardware description is separated from drivers...
- Capability to determine devices in the system, view their status and power state, see the bus they are attached to and determine which driver is responsible for them.
- Capability to generate a complete tree of device structure of the system, including all buses and interconnections.
- Device classification by their type.
- Minimization of code duplication.

Linux device and driver model is split in three categories: bus drivers, bus controller drivers and device drivers. Figure 3.1 shows how they are interrelated.

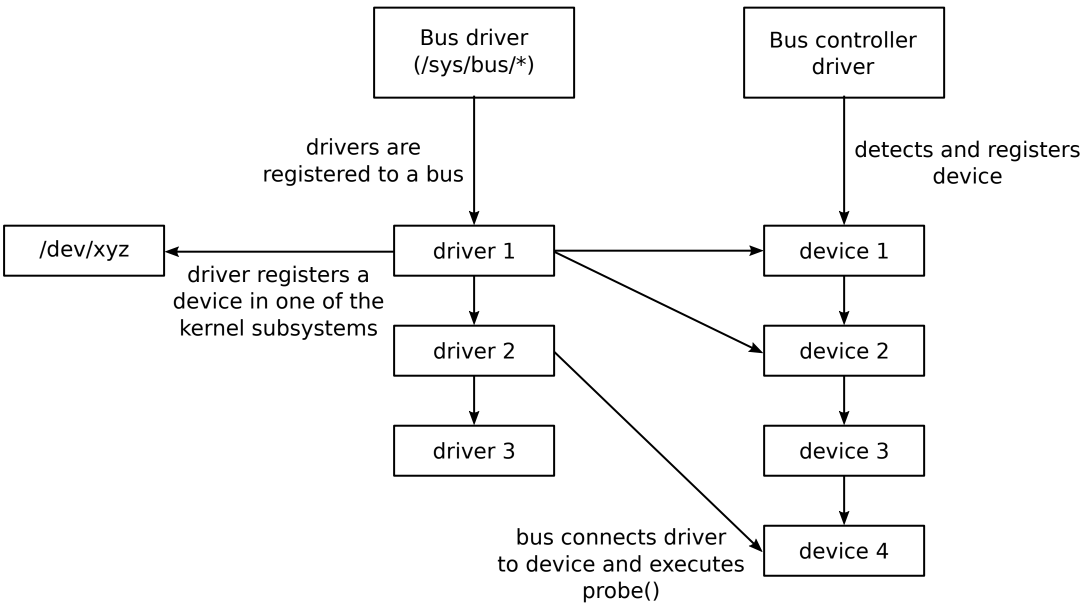


Figure 3.1: Linux device and driver model (source [9]).

3.1.1 Bus Drivers

For each bus supported by the kernel there is a generic bus driver. For the purposes of the device model, all devices are connected via a bus, even if it is an internal bus, a virtual bus or a platform bus.

The role of a bus driver includes:

- Registering buses in the system.
- Allow registration of bus controller drivers and configure their resources.
- Allow registration of device drivers.
- Match devices and drivers.

3.1.2 Bus Controller Drivers

For a specific bus type there may be many different controllers from different vendors. Each of them needs a corresponding bus controller driver.

The role of a controller driver includes:

- Register itself.
- Detect devices on the bus it is controlling and register them.

3.2 Introduction to the Device Tree

On embedded systems devices are often not connected through a bus allowing enumeration, hotplugging and providing unique identifiers for devices (for example, the devices on I^2C buses or SPI buses) [1]. Such devices, instead of being dynamically detected, must be statically described in either:

- The kernel source code (the old way)
- The Device Tree (the modern way)

The Device Tree is a data structure for describing hardware rather than hard coding every detail of a device into the kernel. The data structure is a tree of nodes and properties.

Nodes contain properties and child nodes. Properties are simple name-value pairs, containing zero or more data values. As an example, the code below is a piece of Rock960 Device Tree (located in `rk3399-rock960-ab.dts` in `arch/arm64/boot/dts/rockchip/`). The example focuses on `i2c6` and `i2c1`, which correspond to I2C0 and I2C1 in low speed connector.

```
/dts-v1/;

#include <dt-bindings/pwm/pwm.h>
#include <dt-bindings/input/input.h>
#include "rk3399.dtsi"
#include "rk3399-linux.dtsi"
#include "rk3399-opp.dtsi"
```

```
/ {

  model = "ROCK960 - 96boards based on Rockchip RK3399";
  compatible = "rockchip,rock960","rockchip,rk3399";

  ...

  __symbols__ {
    dfi = "/dfi@ff630000";
    cru = "/clock-controller@ff760000";
    dmc = "/dmc";
    edp = "/edp@ff970000";
    dsi = "/dsi@ff960000";
    gic = "/interrupt-controller@fee00000";
    gpu = "/gpu@ff9a0000";
    grf = "/syscon@ff770000";
    iep = "/iep@ff670000";
    its = "/interrupt-controller@fee00000/interrupt-controller@fee20000";
    pmu = "/power-management@ff310000";
    rga = "/rga@ff680000";
    vpu = "/vpu_service@ff650000";
    i2c0 = "/i2c@ff3c0000";
    i2c1 = "/i2c@ff110000";
    i2c2 = "/i2c@ff120000";
    i2c3 = "/i2c@ff130000";
    i2c4 = "/i2c@ff3d0000";
    i2c5 = "/i2c@ff140000";
    i2c6 = "/i2c@ff150000";
    i2c7 = "/i2c@ff160000";

    ...

  i2c@ff150000 {
    reg = <0x0 0xff150000 0x0 0x1000>;
    interrupts = <0x0 0x25 0x4 0x0>;
    pinctrl-0 = <0x42>;
    compatible = "rockchip,rk3399-i2c";
    clock-names = "i2c", "pclk";
    clocks = <0x8 0x45 0x8 0x159>;
    status = "okay";
    #address-cells = <0x1>;
    phandle = <0xf9>;
    #size-cells = <0x0>;
    pinctrl-names = "default";
  };

  i2c@ff110000 {
    reg = <0x0 0xff110000 0x0 0x1000>;
    interrupts = <0x0 0x3b 0x4 0x0>;
    pinctrl-0 = <0x3a>;
    compatible = "rockchip,rk3399-i2c";
    clock-names = "i2c", "pclk";
    clocks = <0x8 0x41 0x8 0x155>;
    status = "okay";
    #address-cells = <0x1>;
    phandle = <0xf5>;
    #size-cells = <0x0>;
    pinctrl-names = "default";
  };

  ...
```

In this example:

- `@ff150000` and `@ff110000` are base addresses of `i2c6` and `i2c1`
- `reg` property describes device's reources addresses, within the space defined by its parent.
- `compatible` property allows a device to express its compatibility with a family of similar devices, potentially allowing a single device driver to match against several devices. As depicted, the driver is only compatible with RK3399.
- `status` property indicates the operational status of a device. `"okay"` means that both I2C0 and I2C1 are operational.

Device Tree source files are compiled with the Device Tree Compiler. This generates Device Tree Blob Binary files which are parsed by the kernel at boot time.

Files may include other files. The whole tree may consulted in the target machine with:

```
dtc -I fs /sys/firmware/devicetree/base/
```

3.3 RK3399 I^{2}C Platform Driver

Among the non-discoverable devices, a huge group are the devices that are directly part of RK3399: UART controllers, Ethernet controllers, I^{2}C controllers... Linux kernel includes a special bus to handle such devices: the platform bus. It works as any other bus, except that devices are enumerated statically instead of being dynamically discovered.

File `drivers/i2c/busses/i2c-rk3x.c` has the I^{2}C platform driver for RK3399. It includes register definitions:

```
#define REG_CON         0x00 /* control register */
#define REG_CLKDIV      0x04 /* clock divisor register */
#define REG_MRXADDR     0x08 /* slave address for REGISTER_TX */
#define REG_MRXRADDR    0x0c /* slave register address for REGISTER_TX */
#define REG_MTXCNT      0x10 /* number of bytes to be transmitted */
#define REG_MRXCNT      0x14 /* number of bytes to be received */
#define REG_IEN         0x18 /* interrupt enable */
#define REG_IPD         0x1c /* interrupt pending */
#define REG_FCNT        0x20 /* finished count */
```

Bus values:

```
static const struct i2c_spec_values standard_mode_spec = {
  .min_hold_start_ns = 4000,
  .min_low_ns = 4700,
  .min_high_ns = 4000,
  .min_setup_start_ns = 4700,
  .max_data_hold_ns = 3450,
  .min_data_setup_ns = 250,
  .min_setup_stop_ns = 4000,
  .min_hold_buffer_ns = 4700,
```

```
};

static const struct i2c_spec_values fast_mode_spec = {
  .min_hold_start_ns = 600,
  .min_low_ns = 1300,
  .min_high_ns = 600,
  .min_setup_start_ns = 600,
  .max_data_hold_ns = 900,
  .min_data_setup_ns = 100,
  .min_setup_stop_ns = 600,
  .min_hold_buffer_ns = 1300,
};
```

I^2C transfer related functions, such as:

```
static void rk3x_i2c_start(struct rk3x_i2c *i2c);
static void rk3x_i2c_stop(struct rk3x_i2c *i2c, int error);
static int rk3x_i2c_xfer(struct i2c_adapter *adap, struct i2c_msg *msgs, int num);
```

Code to match driver with Device Tree:

```
static const struct of_device_id rk3x_i2c_match[] = {
  {
    .compatible = "rockchip,rk3066-i2c",
    .data = (void *)&rk3066_soc_data
  },
  {
    .compatible = "rockchip,rk3188-i2c",
    .data = (void *)&rk3188_soc_data
  },
  {
    .compatible = "rockchip,rk3228-i2c",
    .data = (void *)&rk3228_soc_data
  },
  {
    .compatible = "rockchip,rk3288-i2c",
    .data = (void *)&rk3288_soc_data
  },
  {
    .compatible = "rockchip,rk3399-i2c",
    .data = (void *)&rk3399_soc_data
  },
  {},
};
MODULE_DEVICE_TABLE(of, rk3x_i2c_match);
```

Platform driver definition:

```
static struct platform_driver rk3x_i2c_driver = {
  .probe   = rk3x_i2c_probe,
  .remove  = rk3x_i2c_remove,
  .driver  = {
    .name  = "rk3x-i2c",
    .of_match_table = rk3x_i2c_match,
    .pm = &rk3x_i2c_pm_ops,
  },
```

```
};

module_platform_driver(rk3x_i2c_driver);
```

And, of course, its own `probe` and `remove` functions:

```
static int rk3x_i2c_probe(struct platform_device *pdev);
static int rk3x_i2c_remove(struct platform_device *pdev);
```

3.4 Using RK3399 I^{2}C Platform Driver from User Space

RK3399 I^{2}C functions are not directly used from user space. User space applications use `i2c-dev` driver in `drivers/i2c`, which is a character driver with the well known Linux character driver API:

```
static int i2cdev_open(struct inode *inode, struct file *file);
static ssize_t i2cdev_read(struct file *file, char __user *buf, size_t count, loff_t *offset);
static ssize_t i2cdev_write(struct file *file, const char __user *buf, size_t count,
  loff_t *offset);
static int i2cdev_release(struct inode *inode, struct file *file);
...
```

This driver is passed the platform driver to use:

```
static int i2cdev_attach_adapter(struct device *dev, void *dummy);
static int i2cdev_detach_adapter(struct device *dev, void *dummy);
```

4 LSM9DS0 Driver Development

4.1 Introduction to I^{2}C Bus

4.1.1 Basic Features

I^{2}C bus is a synchronous, half-duplex, multi-master/multi-slave, 8-bit oriented serial computer bus. Invented by Philips Semiconductor (now NXP Semiconductors), I^{2}C has become a de facto world standard in over 1000 different integrated circuits manufactured by more than 50 companies [10].

Some features of I^{2}C include:

- Only two bus lines: SDA and SCL.
- 8-bit oriented.
- Bidirectional (half-duplex), with bit rates:
  - $100kbit/s$ in standard mode
  - $400kbit/s$ in fast mode
  - $1Mbit/s$ in fast mode Plus
  - $3.4Mbit/s$ in high speed mode
- There is a special unidirectional mode, with bit rate up to $5Mbit/s$ (ultra fast mode)
- Each device connected to the bus is addressable by a unique address.
- In each data transfer between two devices, one device behaves as master and the other one as slave.
- The master initiates the transfer, generates clock signals and terminates the transfer.
- It is a true multi-master bus, including collision detection and arbitration to prevent data corruption in case two masters initiate data transfer simultaneously.
- The number of devices connected to the same bus is limited only by bus capacitance.

4.1.2 Data Transfer

All transactions start with a START and are terminated by a STOP. START is signaled with a HIGH to LOW transition on the SDA line while SCL is HIGH. Bus is considered to be busy after it. STOP is signaled with a LOW to HIGH transition on the SDA line while SCL is HIGH. Then bus is considered to be free again.

After a START condition, master and slave may put data on the SDA line. The number of data bytes that may be transmitted is unrestricted. Each data must be eight bits long, with the Most Significant Bit (MSB) first (see figure 4.1). Once the last bit is received, receiver must signal the transmitter that the byte was successfully received. This signal is called Acknowledge.

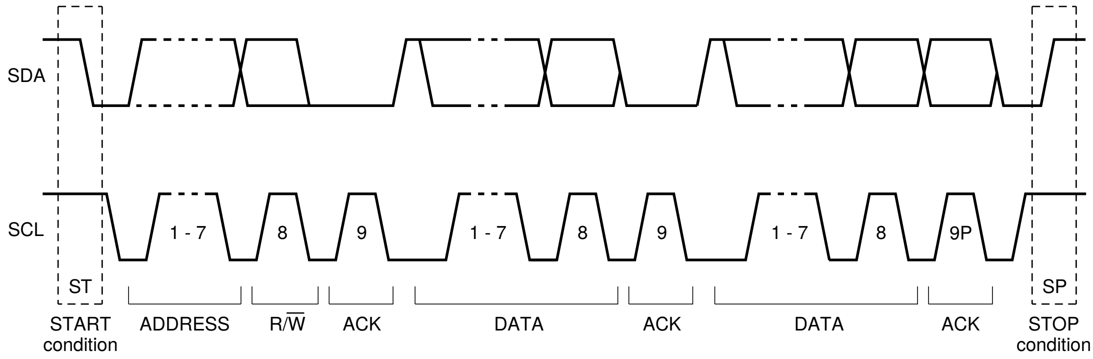


Figure 4.1: Data transfer in I^2C bus (source: NXP Semiconductors).

4.1.3 Slave Addresses

After a START condition, the first byte is sent by the master and it contains the slave address. The address is 7-bit long and it is followed by an eighth bit, called $R/\overline{W}$ bit: 0 indicates a write transmission, 1 indicates a read transmission. Some 7-bit addresses are reserved (see table 4.1).

Table 4.1: Reserved 7 bit addresses.

Address	$R/\overline{W}$	Description	Comments
0000 000	0	General call address	Used for several functions including software reset
0000 000	1	START byte	No device is acknowledge
0000 001	X	CBUS address	To enable the inter-mixing of CBUS and I^2C devices in the same system
0000 010	X	Reserved for different bus formats	To enable mixing I^2C with other protocols
0000 011	X	Reserved for future purposes	-
0000 1XX	X	High speed mode	-
1111 0XX	X	10 bit addressing	-
1111 1XX	X	To retrieve device ID	Device ID is an optional 3 byte read-only word with: • Manufacturer name • Part identification • Revision

I^2C also supports 10 bit addresses, but this feature will not be covered in this document.

4.2 LSM9DS0 I^{2}C Operation

LSM9DS0 has two I^{2}C devices:

- One for accelerometer and magnetometer registers (0x1D I^{2}C address).
- One for gyroscope registers (0x6B I^{2}C address).

Both devices behave like I^{2}C slaves. To access to LSM9DS0 registers, the following protocol must be adhered to:

- Read one single byte (described in figure 4.2).
- Read multiple byte (described in figure 4.3).
- Write one single byte (described in figure 4.4).
- Write multiple bytes (described in figure 4.5).

Master	ST	SADW		SUB		SR	SADR			$\overline{\text{MAK}}$	SP
Slave			SAK		SAK			SAK	DATA		

Figure 4.2: I^{2}C transfer when master reads one single byte.

Master	ST	SADW		SUB		SR	SADR			MAK		MAK		$\overline{\text{MAK}}$	SP
Slave			SAK		SAK			SAK	DATA		DATA		DATA		

Figure 4.3: I^{2}C transfer when master reads multiple bytes.

Master	ST	SADW		SUB		DATA		SP
Slave			SAK		SAK		SAK	

Figure 4.4: I^{2}C transfer when master writes one single byte.

Master	ST	SADW		SUB		DATA		DATA		SP
Slave			SAK		SAK		SAK		SAK	

Figure 4.5: I^{2}C transfer when master writes multiple bytes.

Table 4.2: Description of symbols used in I^2C transfers.

Symbol	Description	Num. bits
DATA	Data transmitted	8
MAK	Acknowledge signal generated by the master	1
$\overline{\text{MAK}}$	Not master acknowledge	1
SADR	Slave address + read For accelerometer and magnetometer: • Bits 7 to 1 are 0011101b (0x1D) • Bit 0 is 1 For gyroscope: • Bits 7 to 1 are 1101011b (0x6B) • Bit 0 is 1	7+1
SADW	Slave address + write For accelerometer and magnetometer: • Bits 7 to 1 are 0011101b (0x1D) • Bit 0 is 0 For gyroscope: • Bits 7 to 1 are 1101011b (0x6B) • Bit 0 is 0	7+1
SAK	Acknowledge signal generated by the slave	1
SP	Transmission stop signal	-
SR	Repeated start signal	-
ST	Transmission start signal	-
SUB	Sub-address: • The 7 LSB represent the register number to be read/written • If the MSB is 1, then register number will be automatically incremented • If the MSB is 0, then register number will not be changed	7+1

4.3 LSM9DS0 Driver

4.3.1 User Space Device Drivers

User space drivers [1] can be used if the following conditions are met:

- The kernel provides a mechanism that allows user space applications to access the hardware.
- It is not necessary to leverage an existing kernel subsystems, such as networking or file systems.
- There is no need for the kernel to act as a multiplexer for the device since only one application accesses the device.

Some of the advantages of user space drivers include:

- It is not necessary to know about kernel programming.
- Code can be used in other devices.
- Code can be killed and debugged.
- Debugging the driver will not crash kernel.
- Floating point computation can be used.
- Drivers can be updated or swapped without needing to recompile.
- Potentially higher performance, since less system calls are used.
- Drivers may be programmed in other programming languages
- Drivers may be kept proprietary.

However, user space drivers have some drawbacks, which are mostly related to interrupts:

- Interrupt handling is more difficult.
- Interrupt latency is higher.

It was considered that a device driver for LSM9DS0 would meet the conditions previously presented. For this reason, it was decided that it would be developed in user space.

4.3.2 Driver Description

LSM9DS0 driver includes necessary code to initiate, enable, configure the sensor and read measurements. It was decided that the driver would “hide” everything related to the electronics of the sensor. To that end, custom data types were defined. For instance:

```
/* Accelerometer scale*/
enum lsm9ds0_acc_scale
{
    LSM9DS0_ACC_SCALE_2,
    LSM9DS0_ACC_SCALE_4,
    LSM9DS0_ACC_SCALE_6,
    LSM9DS0_ACC_SCALE_8,
    LSM9DS0_ACC_SCALE_16,
    LSM9DS0_ACC_NUM_SCALES
};

/* Accelerometer output data rate */
enum lsm9ds0_acc_odr
{
    LSM9DS0_ACC_ODR_POWER_OFF,
    LSM9DS0_ACC_ODR_3_HZ_125,
    LSM9DS0_ACC_ODR_6_HZ_25,
    LSM9DS0_ACC_ODR_12_HZ_5,
    LSM9DS0_ACC_ODR_25_HZ,
    LSM9DS0_ACC_ODR_50_HZ,
    LSM9DS0_ACC_ODR_100_HZ,
    LSM9DS0_ACC_ODR_200_HZ,
    LSM9DS0_ACC_ODR_400_HZ,
    LSM9DS0_ACC_ODR_800_HZ,
    LSM9DS0_ACC_ODR_1600_HZ,
    LSM9DS0_ACC_NUM_RATES
};
```

Accelerometer, magnetometer and gyroscope have their data type too:

```
/* Accelerometer and magnetomer have the same I2C address and share
 * some registers.
 *
 * Thus, the following data type is for both sensors.
 */
struct lsm9ds0_xm_dev
{
    /* File descriptor*/
    int fd;

    /* I2C address */
    unsigned char i2c_addr;

    /* Related registers */
    struct lsm9ds0_xm_reg reg;
};

struct lsm9ds0_gyr_dev
{
    /* File descriptor */
    int fd;

    /* I2C address */
    unsigned char i2c_addr;

    /* Related registers */
    struct lsm9ds0_gyr_reg reg;
};
```

As depicted, `lsm9ds0_xm_dev` and `lsm9ds0_gyr_dev` include all related registers (status, configuration, data...). For instance, for the accelerometer and the magnetometer:

```
struct lsm9ds0_xm_reg
{
    /* Who am I */
    struct r_reg who_am_i_xm;
    /* Status */
    struct r_reg status_reg_a;
    struct r_reg status_reg_m;
    /* Out mag */
    struct r_reg out_x_l_m;
    struct r_reg out_x_h_m;
    struct r_reg out_y_l_m;
    struct r_reg out_y_h_m;
    struct r_reg out_z_l_m;
    struct r_reg out_z_h_m;
    /* Out acc */
    struct r_reg out_x_l_a;
    struct r_reg out_x_h_a;
    struct r_reg out_y_l_a;
    struct r_reg out_y_h_a;
    struct r_reg out_z_l_a;
    struct r_reg out_z_h_a;
    /* Control */
    struct rw_reg ctrl_reg0_xm;
    struct rw_reg ctrl_reg1_xm;
    struct rw_reg ctrl_reg2_xm;
    struct rw_reg ctrl_reg3_xm;
    struct rw_reg ctrl_reg4_xm;
    struct rw_reg ctrl_reg5_xm;
    struct rw_reg ctrl_reg6_xm;
    struct rw_reg ctrl_reg7_xm;
};
```

where:

```
/* Read register */
struct r_reg
{
    /* Register address */
    /* Initiated by the driver */
    unsigned char addr;

    /* Register value */
    unsigned char val;
};

/* Reead/Write register */
struct rw_reg
{
    /* Register address */
    /* Initiated by the driver */
    unsigned char addr;

    /* Default value */
    char def_val;

    /^Register value */
```

```
    /* Initiated by the driver */
    unsigned char val;
};
```

All this decisions were taken to guarantee the correct access to LSM9DS0. With them, it is not possible for the user to write any value in any address of the sensor. For more details of driver, see appendix A.4.

5 Demo Application

5.1 Description

A graphic application was developed to demonstrate the use and functionality of LSM9DS0 driver. This application shows a 3D object whose inclination depends on the sensor. That is to say, any tilt change on the sensor will change the tilt of 3D object.

The initial idea was to develop an embedded graphic application that would run on Rock960. That is why a graphic desktop environment was installed (see 2.3.6), and that is why OpenGL ES 2.0 was tested.

However, in the course of the project it was considered that a distributed and web-based architecture would be more interesting:

- Sensor data would be read by one application (which would run on Rock960, of course), and another application would deal with the graphical part.
- Applications would exchange information using custom JSON messages over WebSockets.
- The graphic application would run on a browser, using `Three.js`[1] as a graphical engine.

The author considered this architecture was more flexible and modular:

- Because it is distributed, applications may be updated independently. Actually, the board and/or the sensor could be replaced for new models, without affecting the graphic application
- Because it is web-based, it may run on any device with Internet connection: computers, tablets, smart phones... with Android, IOs, Ubuntu, Windows...

Author also considered this architecture to be more appropriate if sensor was installed on autonomous and mobile robots, such as UAVs, rovers or humanoid robots.

5.1.1 UML Components Diagram

Figure 5.1 is the UML components diagram of the demo application. As depicted:

- LSM9DS0 was represented with `I2C_Slave` component.
- The application responsible for reading LSM9DS0 data was represented with `WebSock_Server` and `I2C_Master`, both running on Rock960:
  - `I2C_Master` is implemented by the operating system.

[1] `Three.js` is a cross-browser JavaScript Library and API used to create and display animated 3D graphics in web browsers. It uses WebGL.

- `WebSock_Server` includes the user-space LSM9DS0 driver, which uses `I2C_Master`.

- The web application was represented with `Web_Client` and `WebSock_Client`:
  - `Web_Client` uploads web files (`.html`, `.css` and `.js` files), and instantiates `WebSock_Client`.
  - `WebSock_Client` communicates with `WebSock_Server` to get sensor data.
- Web files are served by `Web_Server`, which runs on a remote server (not in Rock960).

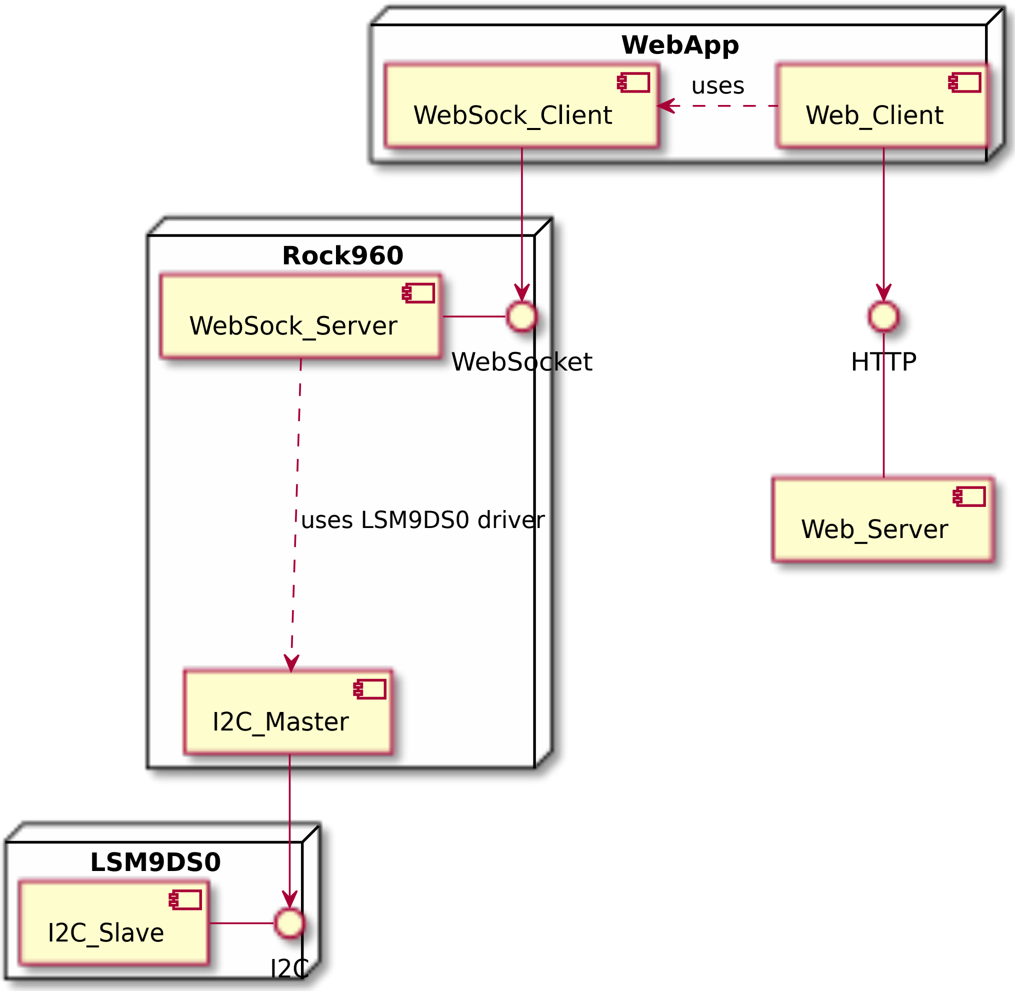


Figure 5.1: Demo application components UML diagram.

5.1.2 UML Sequence Diagram

For the demo application, four different interactions were considered:

- **Power on** - The web application sends a request to set LSM9DS0 in normal mode (see figure 5.2).
- **Power off** - The web application sends a request to set LSM9DS0 in power-down mode (see figure 5.3).
- **Read acceleration** - The web application sends a request to read LSM9DS0 acceleration data (see figure 5.4).
- **Tilt sensing** - The web application continuously reads acceleration data to calculate sensor inclination (see figure 5.5).

Because this is a demo application, some limitations were set:

- When powering LSM9DS0 on, it is `WebSock_Server` who configures sensor (scale, output data rate and low-pass filter cutoff frequency).
- Magnetometer and gyroscope data are not used. However, the LSM9DS0 driver includes all necessary code to read them (see A.4).

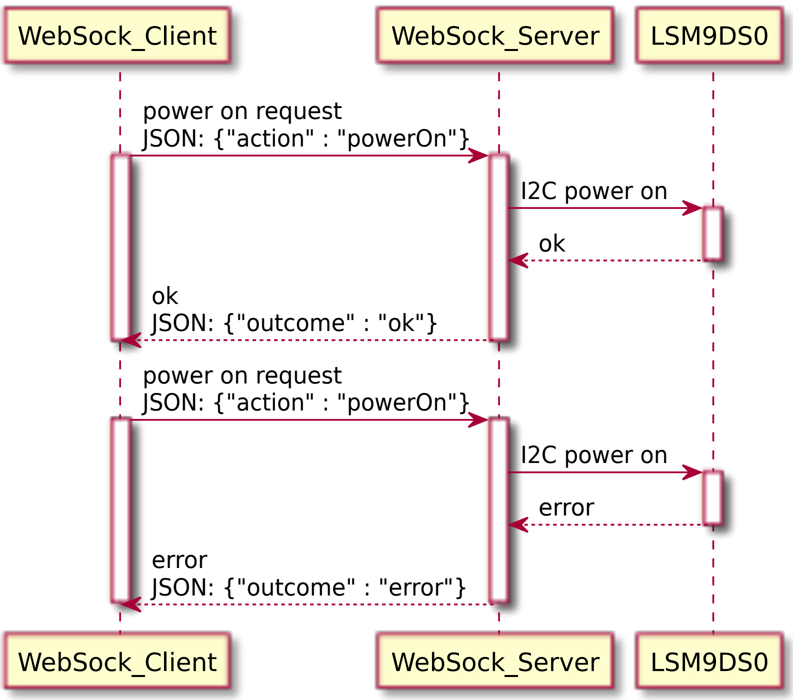


Figure 5.2: UML sequence diagram for the power on interaction.

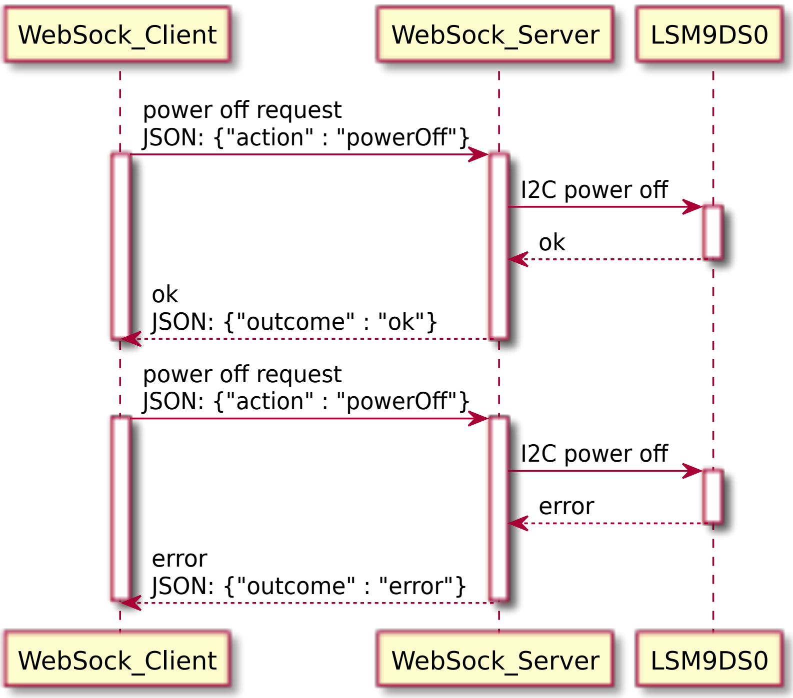


Figure 5.3: UML sequence diagram for the power off interaction.

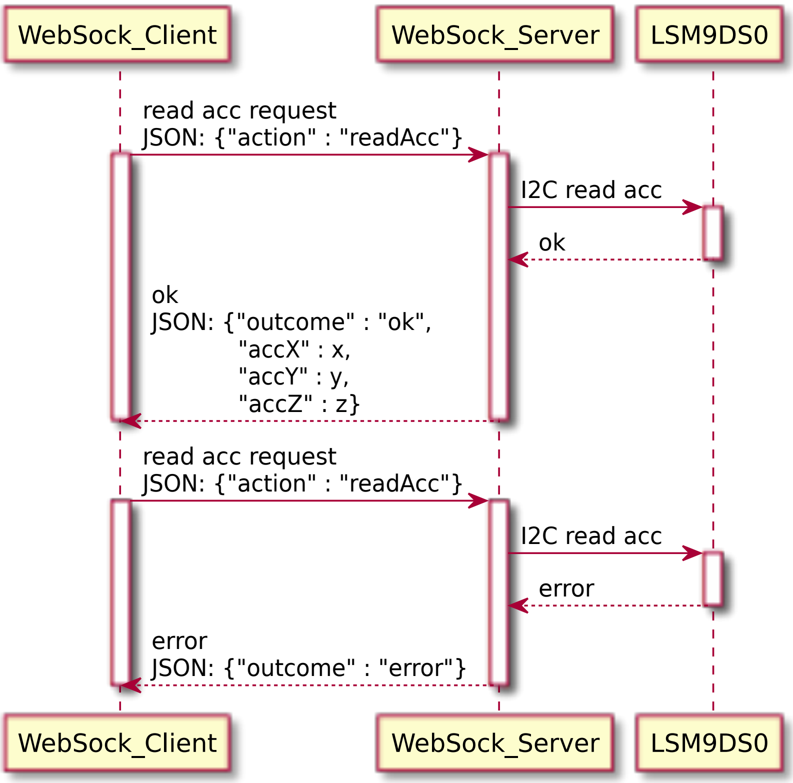


Figure 5.4: UML sequence diagram for the read acceleration interaction.

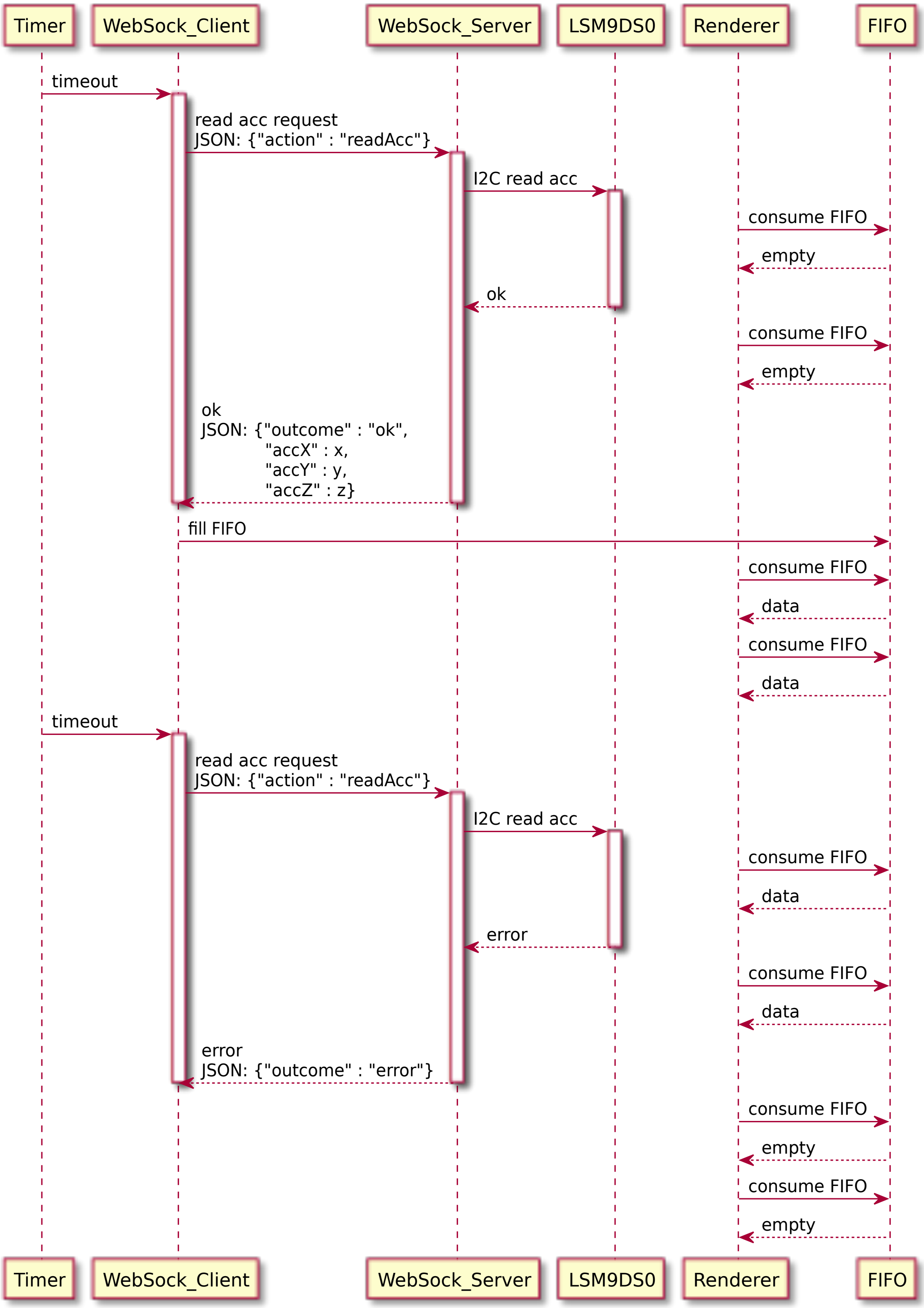
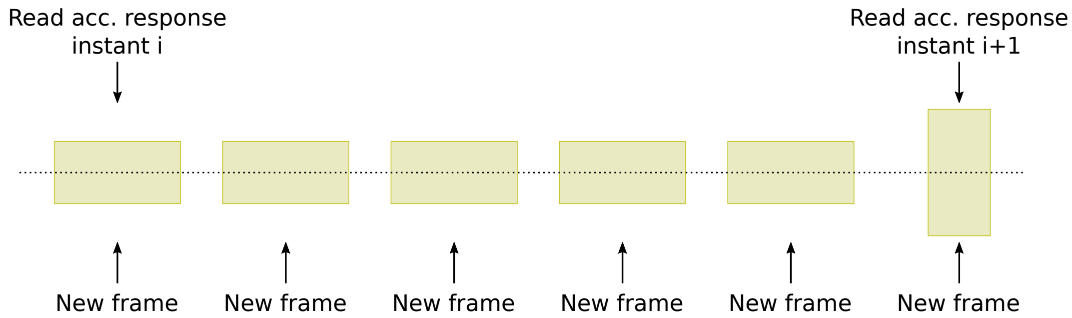


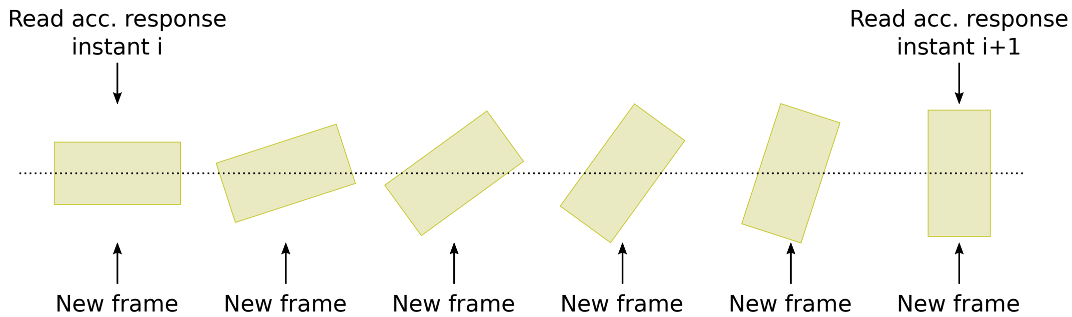
Figure 5.5: UML sequence diagram for the tilt sense interaction.

In tilt sensing interaction, `WebSock_Client` retrieves acceleration every $T_{ws} = 250ms$. This rate is greater than renderer refresh rate, which is about 60 frames per second ($T_r = 1000/60 = 16.67ms$).

In order to show continuous and smooth movements, the web client interpolates movements. The following example explains this operation.



(a)



(b)

In figure 5.6a, renderer only refreshes the image when `WebSock_Client` receives read acceleration response. As a consequence, the same image is shown for some frames. In figure 5.6b, on the other hand, tilt at time t_{i+1} is compared with tilt at time t_i. The difference is divided by a certain number of frames, which results from:

$$n = \frac{T_{ws}}{T_r} = \frac{250}{16.67} = 15 \tag{5.1}$$

Then the renderer transforms the image every frame. As a result, refresh is more smooth and visually comfortable.

As depicted in figure 5.5, `WebSock_Client` and renderer use a FIFO buffer to share data, following the producer-consumer design pattern:

- After calculation tilt changes per frame (for the next n frames), `WebSock_Server` fills in the FIFO buffer.
- For each frame, the renderer consumes a single entry in the FIFO buffer to refresh the image.

5.2 Introduction Euler Angles and Tait-Bryan Angles

According to Euler's rotation theorem, the rotation of an object with respect of a fixed coordinate system may be described using three angles [11]. For instance, in figure 5.7, axis X', Y' and Z' are defined by angles α, β and γ:

- α is the angle between X and N (N is the intersection of planes XY and $X'Y'$).
- β is the angle between Z and Z'.
- γ is the angle between X' and N.

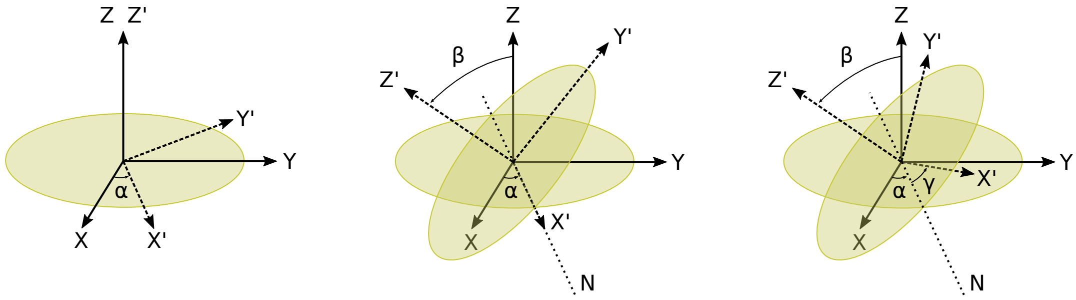


Figure 5.7: Euler angles definition.

Intrinsic rotations are elemental rotations that occur about axes of a coordinate system attached to a moving body. Euler angles may be defined by intrinsic rotations. In figure 5.7, axes X, Y and Z are fixed, and axes X', Y' and Z' are attached the circle. In this case:

- α comes after a rotation around Z'.
- β comes after a rotation around X'.
- γ comes after another rotation around Z'.

There exist twelve possible sequences of rotation axis, divide in two groups:

- **Proper Euler angles**: ZXZ, XYX, YZY, ZYZ, XZX, YXY.
- **Tait-Bryan angles**: XYZ, YZX, ZXY, XZY, ZYX, YXZ.

Any orientation may be achieved following any sequence. However, this does not mean that sequences are equal. In figure 5.8, an object is rotated 90º following different sequences. As depicted, final orientations are different.

In navigation, the most used sequence is XYZ and rotation angles have its own names (see figure 5.9):

- **Roll** - Rotation around X.
- **Pirch** - Rotation around Y.
- **Yaw** - Rotation around Z.

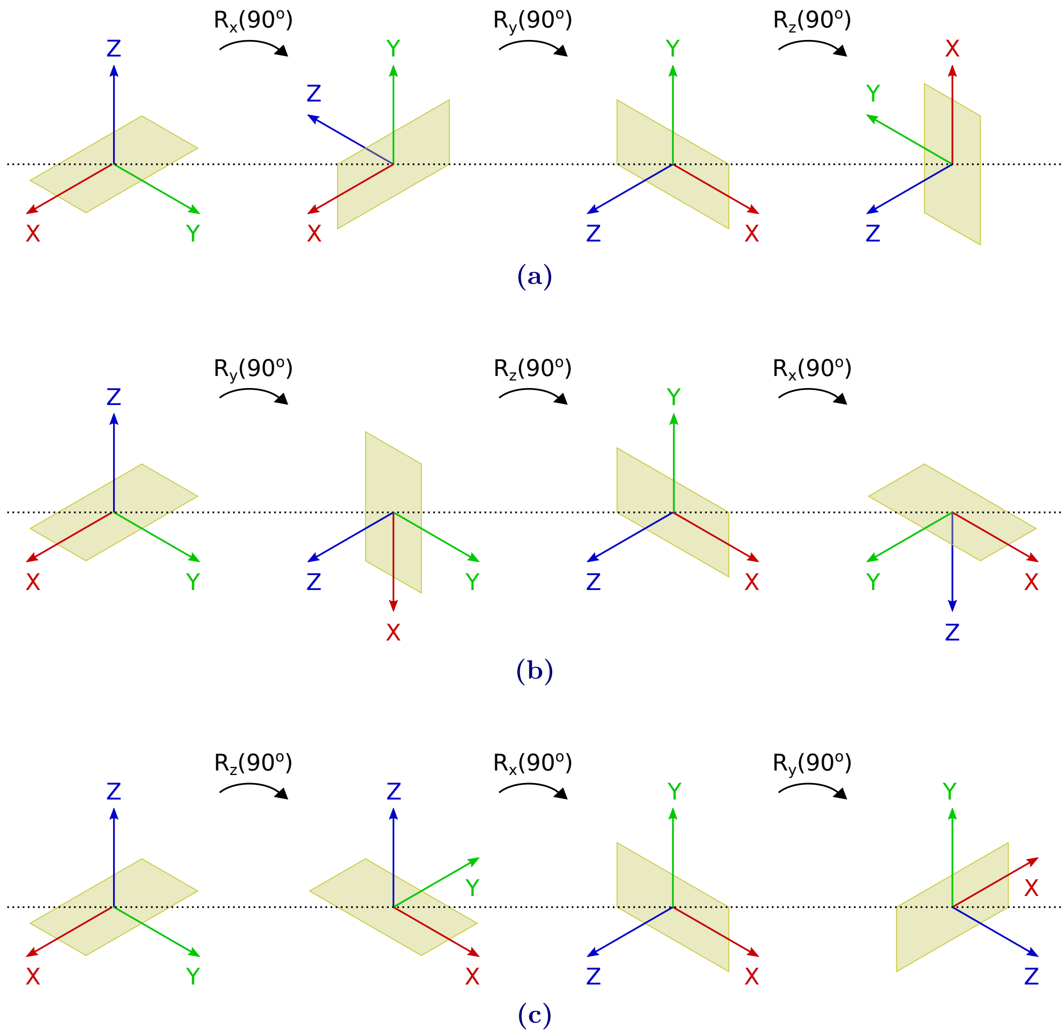


Figure 5.8: Successive rotations following different sequences - (a) XYZ, (b) YZX, (c) ZXY.

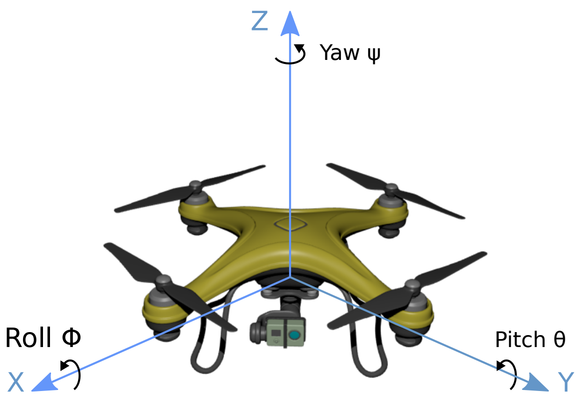


Figure 5.9: Coordinate system and rotation angles.

5.3 Pitch and Roll Estimation

As explained in [12], a three-axis accelerometer oriented in Earth's gravitational field $\vec{g}$ will have $\vec{g}_p$ output given by:

$$\vec{g}_p = \begin{pmatrix} g_{px} \\ g_{py} \\ g_{pz} \end{pmatrix} = R\vec{g} \tag{5.2}$$

where $\vec{g}$ is Earth's gravity:

$$\vec{g} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{5.3}$$

and R is a rotation matrix describing accelerometer orientation relative to Earth's coordinate system, in XYZ sequence. Equation 5.2 assumes that the accelerometer is only affected by $\vec{g}$.

R is the result of multiplying roll rotation matrix R_x, pitch rotation matrix R_y and R_z yaw rotation matrix:

$$R = R_{xyz} = R_x R_y R_z \tag{5.4}$$

where:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{pmatrix} \tag{5.5}$$

$$R_y = \begin{pmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{pmatrix} \tag{5.6}$$

$$R_z = \begin{pmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.7}$$

Expanding equation 5.4, results:

$$R_{xyz} = \begin{pmatrix} cos\psi cos\theta & sin\psi cos\theta & -sin\theta \\ cos\psi sin\theta sin\phi - sin\psi cos\phi & sin\psi sin\theta sin\phi + cos\psi cos\phi & cos\theta sin\phi \\ cos\psi sin\theta cos\phi + sin\psi sin\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\theta cos\phi \end{pmatrix} \tag{5.8}$$

Combining equations 5.2 and 5.8 results:

$$\vec{g}_p = \begin{pmatrix} g_{px} \\ g_{py} \\ g_{pz} \end{pmatrix} = \begin{pmatrix} -sin\theta \\ cos\theta sin\phi \\ cos\theta cos\phi \end{pmatrix} \tag{5.9}$$

The previous equation may be rewritten relating roll and pitch angles to the normalized acceleration:

$$\hat{g}_p = \frac{\vec{g}_p}{\|\vec{g}_p\|} = \frac{1}{\sqrt{{g_{px}}^2 + {g_{py}}^2 + {g_{pz}}^2}} \begin{pmatrix} g_{px} \\ g_{py} \\ g_{pz} \end{pmatrix} = \begin{pmatrix} \hat{g}_{px} \\ \hat{g}_{py} \\ \hat{g}_{pz} \end{pmatrix} \tag{5.10}$$

$$\begin{pmatrix} \hat{g}_{px} \\ \hat{g}_{py} \\ \hat{g}_{pz} \end{pmatrix} = \begin{pmatrix} -sin\theta \\ cos\theta sin\phi \\ cos\theta cos\phi \end{pmatrix} \tag{5.11}$$

Finally, solving this equation gives:

$$tan\phi_{xyz} = \frac{\hat{g}_{py}}{\hat{g}_{pz}} \tag{5.12}$$

$$tan\theta_{xyz} = \frac{-\hat{g}_{px}}{\sqrt{{\hat{g}_{py}}^2 + {\hat{g}_{pz}}^2}} \tag{5.13}$$

$$\begin{aligned} \phi &\in (-180^o, 180^o] \\ \theta &\in (-90^o, 90^o] \end{aligned} \tag{5.14}$$

5.4 Running Demo Application

Figure 5.10 shows the demo application. In this figure:

- (1) is the web application, running on a Chrome browser. (1a) is the 3D object whose orientation depends LSM9DS0 orientation. (1b) are static XYZ axes. (1c) are web application controls, using dat.gui library (see figure 5.11).
- (2) is the web server that serves web application.
- (3) is the WebSocket server, running on Rock960.

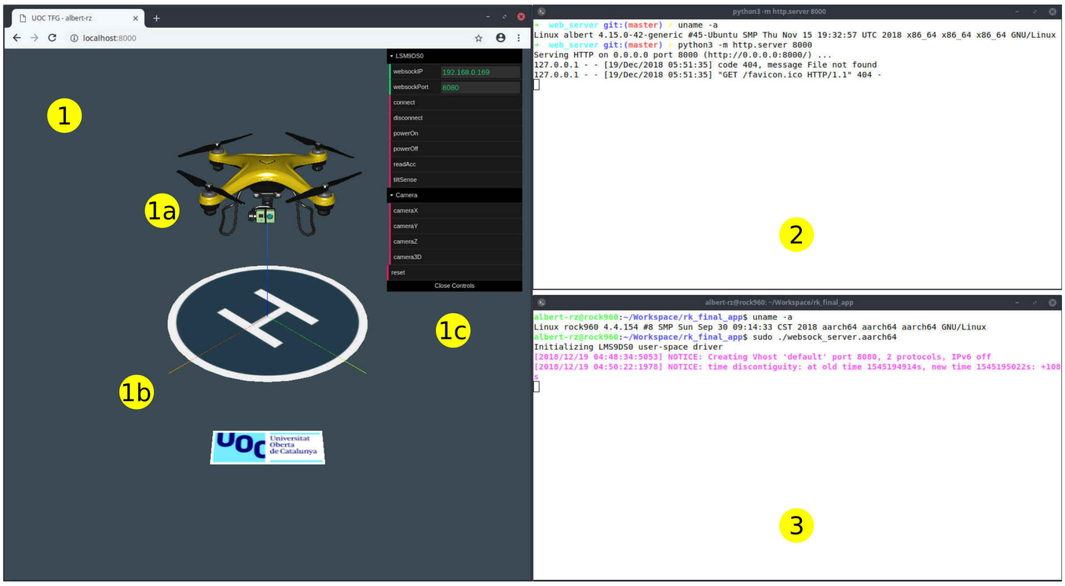


Figure 5.10: Demo application.

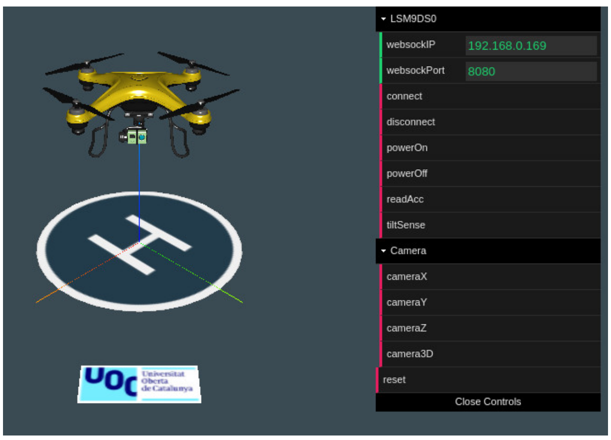
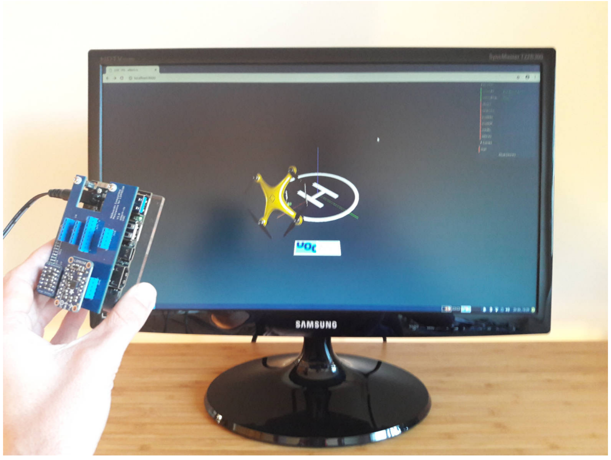
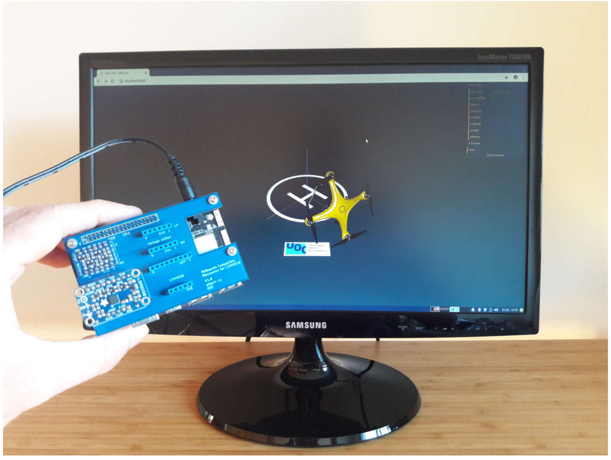


Figure 5.11: Application controls.

Figure 5.12 shows how the 3D object orientation follows LSM9DS0.



(a)



(b)

(c)

Figure 5.12: Demo application showing tilt changes - (a) Acceleration is predominant on X, (b) Acceleration is predominant on Y, (c) Acceleration is predominant on Z.

6 Conclusions

Learning Linux device and driver model is a major challenge. Many source files have to be checked to "discover" how/where drivers meet the hardware. Just for the I^2C bus, author spent hours *grepping* `.c` files, `.h` files and Device Tree files, and matching them with RK3399 reference manual.

All project goals were achieved:

- Linux device and driver model was reviewed.
- The I^2C platform driver for RK3399 was studied in detail.
- A user space driver for LSM9DS0 and a demo application were developped.

96Boards is a great initiative from Linaro. There are plenty of boards using SoCs which are in the state of the art. In the author's opinion, however, the initiative still does not guarantee the proper operation of the boards. During this project, author checked 96Boards discussion frequently and also reported two major errors:

- I2C0 port was not available (see link).
- In Ubuntu Server 64-bit image, most of eMMC memory could not be used because partition had not been well built (see link).

Adafruit LSM9DS0 is a great "hobbyist" board. It includes a voltage regulator and I^2C pull-up resistors, and it is ready to be used right out of the box. To test LSM9DS0 it was necessary to move the board and turn it up and down to see acceleration in the three channels. To avoid any wiring problem a PCB board was designed and built.

The driver covers all LSM9DS0 functionalities, except the ones that are related to interrupts. It was programmed in C and uses its own data types. That protects LSM9DS0 from developer mistakes. For instance, there is no risk that someone tries to write wrong bytes in registers that should not be used.

The demo application uses a distributed architecture. There is one application to read data from LSM9DS0 and another one (a web page) with a graphic application. Information is exchanged with JSON messages over WebSockets. To minimize visual effects due to communication delays, the graphic application interpolates LSM9DS0 data.

Finally, the author would like to say that he really enjoyed this project. Linux device and driver model was a field that he wanted to learn since long time ago. By now, author cannot consider himself an expert. Definitely not. However, he feels more confident and capable to develop device drivers. It would be very interesting to migrate the driver done to the kernel space.

7 Bill of Materials

Table 7.1: Bill of materials.

#	Reference	Manufacturer	Description	Price (€/u)	Units	Total (€)
1	ROCK960-MODEL-B	Vamrs	Development board based on RK3399 SoC	122.12	1	122.12
2	954	Adafruit	USB to TTL cable	8.74	1	8.74
3	239	Adafruit	Full size breadboard	5.23	1	5.23
4	153	Adafruit	Breadboarding wire bundle	4.35	1	4.35
5	3463	Adafruit	LSM9DS breakout board	21.92	1	21.92
6	757	Adafruit	4-channel I^2C safe bidirectional voltage level shifter - BSS138	3.47	1	3.47
7	-	Safe-PCB	Printed circuit board for LSM9DS0 custom made mezzanine	71.60	2	143.20
8	76341-312LF	Amphenol	Board-to-board connector, 2.54mm, 12 contacts	1.05	10	10.50
9	TMM-120-01-G-D	Samtec	Board-to-board connector, 2mm, 40 contacts, 2 rows	6.14	2	12.28
10	05.12.053	Ettinger	Hexagonal spacer, male-female, M2.5, 5mm	0.44	5	2.20
11	971060151	Wurth Elektronik	Hexagonal spacer, male-female, M2.5, 6mm	0.51	5	2.54
12	971070151	Wurth Elektronik	Hexagonal spacer, male-female, M2.5, 7mm	0.45	5	2.24
13	TR NWE-34815-M2	TR Fastenings	Nylown washer, M2 (pack of 100)	4.75	1	4.75
	Total					**343.56**

Prices include taxes.

Bibliography

[1] Bootlin, "Linux kernel and driver development training," December 2018. [Online]. Available: https://bootlin.com/doc/training/linux-kernel/linux-kernel-slides.pdf

[2] Linaro, "Linaro embedded Linux collaboration - An open source development consortium for ARM and the embedded community," 2010. [Online]. Available: https://elinux.org/images/0/00/ELCE-2010-Linaro.pdf

[3] 96Boards Consumer Edition Specification. [Online]. Available: https://www.96boards.org/documentation/Specifications/96Boards-CE-Specification.pdf

[4] Rockusb Wiki. [Online]. Available: http://opensource.rock-chips.com/wiki_Rockusb

[5] STMicroelectronics, *DoclD024763 Rev2 - LSM9DS0 Datasheet - iNemo inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer*, August 2013. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/LSM9DS0.pdf

[6] Bootlin, "Embedded Linux system development," December 2018. [Online]. Available: https://bootlin.com/doc/training/embedded-linux/embedded-linux-slides.pdf

[7] *AN10441 - Level shifting techniques in I2C-bus design*, NXP, June 2017. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN10441.pdf

[8] 96Boards - Sensors mezzanine description. [Online]. Available: https://www.96boards.org/product/sensors-mezzanine/

[9] A. Liberal, *Linux driver development for embedded processors*, 1st ed., Círculo Rojo, Ed., March 2017.

[10] *UM10204 - I2C-bus specificacion and user manual*, NXP, April 2014. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[11] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Euler_angles

[12] *AN3461 - Tilt sensing using a three-axis accelerometer*, NXP, March 2013. [Online]. Available: https://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf

Appendix A Code and Snippets

A.1 `Hello world!` - A Basic Example to Test Cross Compilation

A.1.1 Source Code

```
/*
 * File:  helloworld.cpp
 * Athor: Albert Ruiz
 *
 * This a simple hello world example.
 */

#include <iostream>

int main(void)
{
    std::cout << "Hello World!" << std::endl;

    return 0;
}
```

A.1.2 Makefile

```
# ********************************************************************************
# File:   Makefile
# Author: Albert Ruiz
#
# This is the Makefile for the hello_world example.
# The example may be compiled to the native architecture
# or cross-compiled to aarch64 architecture.
#
# To compile it to native architecture, just run:
# make
#
# To compile it to aarch64:
# make ARCH=arm
# ********************************************************************************

TARGET := hello_world

CC := g++

CFLAGS := -g -Wall

SRCS := $(wildcard *.cpp)
OBJS := $(SRCS:.cpp=.o)

ifeq ($(ARCH),arm)
```

```
CC := aarch64-linux-gnu-$(CC)
TARGET := $(TARGET).arm
endif

$(TARGET): $(OBJS)
  $(CC) -o $(TARGET) $(OBJS)

$(OBJS) : $(SRCS)
  $(CC) $(CFLAGS) -c $(SRCS)

run:
  ./$(TARGET)

clean:
  rm -f $(TARGET) $(TARGET).arm $(OBJS)
```

This `Hello World!` example can be compiled for the native system architecture like:

```
$ make
```

To compile it for Rock960:

```
$ make ARCH=arm
```

This will use ARM cross compiler with AArch64 instruction set.

A.2 Cross Compile `libwebsockets`

A.2.1 Getting the sources

```
git clone https://github.com/warmcat/libwebsockets.git
```

A.2.2 Cross Compilation for Rock960

```
cd libwebsockets
mkdir build.arm
cd build.arm
cmake ../ -DCMAKE_TOOLCHAIN_FILE=../contrib/cross-aarch64.cmake
    -DLWS_WITH_SSL=OFF -DLWS_WITHOUT_BUILTIN_SHA1=OFF
make
```

The file `./contrib/cross-aarch64.cmake` has:

```
#
# CMake Toolchain file for crosscompiling on ARM.
#
# This can be used when running cmake in the following way:
```

```
#  cd build/
#  cmake .. -DCMAKE_TOOLCHAIN_FILE=../cross-arm-linux-gnueabihf.cmake
#

# Target operating system name.
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR aarch64)

# Name of C compiler.
set(CMAKE_C_COMPILER "aarch64-linux-gnu-gcc")
set(CMAKE_CXX_COMPILER "aarch64-linux-gnu-g++")

#-nostdlib
SET(CMAKE_C_FLAGS "-DGCC_VER=\"\\\"$(GCC_VER)\\\"\" -DARM64=1 -D__LP64__=1 -Os -g3 -fpie
    -mstrict-align -DOPTEE_DEV_KIT=../../../optee_os/out/arm-plat-hikey/export-ta_arm64/include
    -fPIC -ffunction-sections -fdata-sections" CACHE STRING "" FORCE)

# Where to look for the target environment. (More paths can be added here)
set(CMAKE_FIND_ROOT_PATH "/projects/aist-tb/arm64-tc/")

# Adjust the default behavior of the FIND_XXX() commands:
# search programs in the host environment only.
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)

# Search headers and libraries in the target environment only.
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

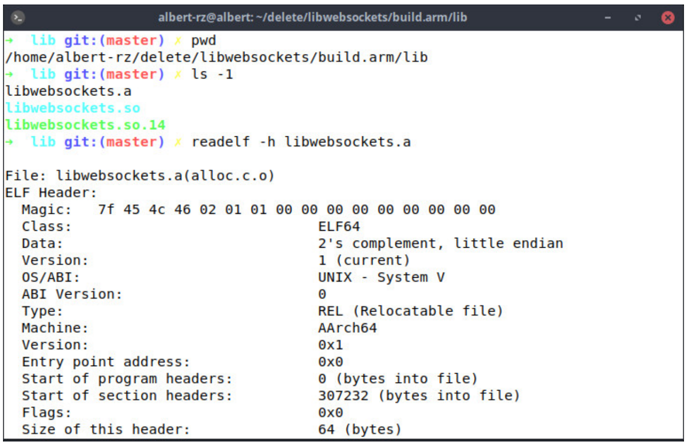


Figure A.1: `libwebsockets` cross compiled.

A.2.3 Compilation for x86_64

```
cd libwebsockets
mkdir build
cd build
cmake ../ -DLWS_WITH_SSL=OFF -DLWS_WITHOUT_BUILTIN_SHA1=OFF
make
```

To compile with SSL support:

```
cd libwebsockets
mkdir build
cd build
cmake ../
make
```

A.3 Cross Compile JSON-C

A.3.1 Getting the sources

```
git clone https://github.com/json-c/json-c.git
```

A.3.2 Cross Compilation for Rock960

```
cd json-c
mkdir contrib
cd contrib
```

Create the file `./contrib/cross-aarch64.cmake` with:

```
#
# CMake Toolchain file for crosscompiling on ARM.
#
# This can be used when running cmake in the following way:
#  cd build/
#  cmake .. -DCMAKE_TOOLCHAIN_FILE=../cross-arm-linux-gnueabihf.cmake
#

# Target operating system name.
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR aarch64)

# Name of C compiler.
set(CMAKE_C_COMPILER "aarch64-linux-gnu-gcc")
set(CMAKE_CXX_COMPILER "aarch64-linux-gnu-g++")

#-nostdlib
SET(CMAKE_C_FLAGS "-DGCC_VER=\"\\\"$(GCC_VER)\\\"\" -DARM64=1 -D__LP64__=1 -Os -g3 -fpie
    -mstrict-align -DOPTEE_DEV_KIT=../../../optee_os/out/arm-plat-hikey/export-ta_arm64/include
    -fPIC -ffunction-sections -fdata-sections" CACHE STRING "" FORCE)

# Where to look for the target environment. (More paths can be added here)
set(CMAKE_FIND_ROOT_PATH "/projects/aist-tb/arm64-tc/")

# Adjust the default behavior of the FIND_XXX() commands:
# search programs in the host environment only.
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
```

```
# Search headers and libraries in the target environment only.
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

```
cd json-c
mkdir build.arm
cd build.arm
cmake ../ -DCMAKE_TOOLCHAIN_FILE=../contrib/cross-aarch64.cmake -DBUILD_SHARED_LIBS=OFF
```

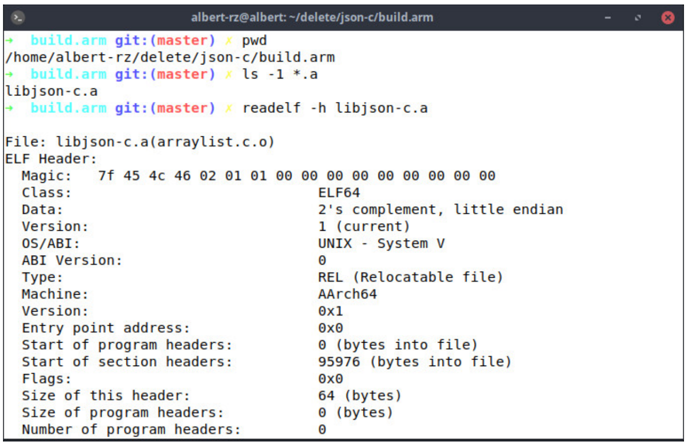


Figure A.2: `JSON-C` cross compiled.

A.3.3 Compilation for x86_64

```
cd json-c
mkdir build
cd build
cmake ../ -DBUILD_SHARED_LIBS=OFF
```

A.4 LSM9DS0 Driver

A.4.1 LSM9DS0 Register Description

Table A.1: Gyroscope status and data registers.

Register	Type	Description
WHO_AM_I_G	R	Device identification register. It always returns 0b11010100.
STATUS_REG_G	R	Status register to inform about data overrun and data ready.
OUT_X_L_G OUT_X_H_G	R	X-axis angular rate data, expressed as two's complement.
OUT_Y_L_G OUT_Y_H_G	R	Y-axis angular rate data, expressed as two's complement.
OUT_Z_L_G OUT_Z_H_G	R	Z-axis angular rate data, expressed as two's complement.

Table A.2: Gyroscope configuration registers.

Register	Type	Description
CTRL_REG1_G	R/W	Control register to: • Set gyroscope in power-down mode or in normal mode. • Configure output data rate and low-pass filter cutoff frequency. • Enable Z-axis, Y-axis and X-axis.
CTRL_REG2_G	R/W	Control register to configure high-pass filter mode and cutoff frequency.
CTRL_REG3_G	R/W	Control register to configure interrupts.
CTRL_REG4_G	R/W	Control register to: • Enable data update. • Select big/little endian. • Configure scale. • Self-test gyroscope. • Configure 4-wire or 3-wire SPI interface.
CTRL_REG5_G	R/W	Control register to: • Enable FIFO. • Enable high-pass filter. • Configure output.

Table A.3: Accelerometer and magnetometer status and data registers.

Register	Type	Description
WHO_AM_I_XM	R	Device identification register. It always returns 0b01001001.
STATUS_REG_M	R	Status register to inform about magnetic data overrun and data ready.
STATUS_REG_A	R	Status register to inform about acceleration data overrun and data ready.
OUT_X_L_A OUT_X_H_A	R	X-axis acceleration data, expressed as two's complement.
OUT_Y_L_A OUT_Y_H_A	R	Y-axis acceleration data, expressed as two's complement.
OUT_Z_L_A OUT_Z_H_A	R	Z-axis acceleration data, expressed as two's complement.
OUT_X_L_M OUT_X_H_M	R	X-axis magnetic data, expressed as two's complement.
OUT_Y_L_M OUT_Y_H_M	R	Y-axis magnetic data, expressed as two's complement.
OUT_Z_L_M OUT_Z_H_M	R	Z-axis magnetic data, expressed as two's complement.

Table A.4: Accelerometer and magnetometer configuration registers.

Register	Type	Description
CTRL_REG0_XM	R/W	Control register to: • Enable FIFO. • Enable high-pass filter for interrupts.
CTRL_REG1_XM	R/W	Control register to: • Set accelerometer in power-down mode or conversion mode. • Configure accelerometer output data rate. • Enable acceleration and magnetic continuous data update. • Enable acceleration Z-axis, Y-axis and X-axis.
CTRL_REG2_XM	R/W	Control register to: • Configure accelerometer low-pass filter. • Configure accelerometer scale. • Self-test accelerometer. • Configure 4-wire or 3-wire SPI interface.
CTRL_REG3_XM	R/W	Control register to configure interrupts.
CTRL_REG4_XM	R/W	Control register to configure interrupts.
CTRL_REG5_XM	R/W	Control register to: • Enable temperature sensor. • Configure magnetometer output data rate. • Configure magnetometer resolution. • Configure some interrupts.
CTRL_REG6_XM	R/W	Control register to configure magnetometer scale.
CTRL_REG7_XM	R/W	Control register to: • Set magnetometer in power-down mode or conversion mode. • Configure accelerometer high-pass filter.

A.4.2 Source Code

The full implementation of the driver is too big to be included in this document. For this reason, only the header file is shown:

```
// ********************************************************************************
// File:  lsm9ds0.h
// Author:  Albert Ruiz
//
// LSM9DS0 driver header file
// ********************************************************************************

#ifndef _LSM9DS0_H
#define _LSM9DS0_H

#define MAX_READ_REGS 9

/* Masks */
#define BIT0  0x01 << 0
#define BIT1  0x01 << 1
#define BIT2  0x01 << 2
#define BIT3  0x01 << 3
#define BIT4  0x01 << 4
#define BIT5  0x01 << 5
#define BIT6  0x01 << 6
#define BIT7  0x01 << 7
#define BIT8  0x01 << 8
#define BIT9  0x01 << 9
#define BIT10 0x01 << 10
#define BIT11 0x01 << 11
#define BIT12 0x01 << 12
#define BIT13 0x01 << 13
#define BIT14 0x01 << 14
#define BIT15 0x01 << 15

#define LSM9DS0_ACC_STATUS_XYZ_OVERRUN    (unsigned char)(0x01 << 7)
#define LSM9DS0_ACC_STATUS_X_OVERRUN    (unsigned char)(0x01 << 6)
#define LSM9DS0_ACC_STATUS_Y_OVERRUN    (unsigned char)(0x01 << 5)
#define LSM9DS0_ACC_STATUS_Z_OVERRUN    (unsigned char)(0x01 << 4)
#define LSM9DS0_ACC_STATUS_XYZ_NEW_DATA   (unsigned char)(0x01 << 3)
#define LSM9DS0_ACC_STATUS_X_NEW_DATA   (unsigned char)(0x01 << 2)
#define LSM9DS0_ACC_STATUS_Y_NEW_DATA   (unsigned char)(0x01 << 1)
#define LSM9DS0_ACC_STATUS_Z_NEW_DATA   (unsigned char)(0x01 << 0)

#define LSM9DS0_MAG_STATUS_XYZ_OVERRUN    (unsigned char)(0x01 << 7)
#define LSM9DS0_MAG_STATUS_X_OVERRUN    (unsigned char)(0x01 << 6)
#define LSM9DS0_MAG_STATUS_Y_OVERRUN    (unsigned char)(0x01 << 5)
#define LSM9DS0_MAG_STATUS_Z_OVERRUN    (unsigned char)(0x01 << 4)
#define LSM9DS0_MAG_STATUS_XYZ_NEW_DATA   (unsigned char)(0x01 << 3)
#define LSM9DS0_MAG_STATUS_X_NEW_DATA   (unsigned char)(0x01 << 2)
#define LSM9DS0_MAG_STATUS_Y_NEW_DATA   (unsigned char)(0x01 << 1)
#define LSM9DS0_MAG_STATUS_Z_NEW_DATA   (unsigned char)(0x01 << 0)

#define LSM9DS0_GYR_STATUS_XYZ_OVERRUN    (unsigned char)(0x01 << 7)
#define LSM9DS0_GYR_STATUS_X_OVERRUN    (unsigned char)(0x01 << 6)
#define LSM9DS0_GYR_STATUS_Y_OVERRUN    (unsigned char)(0x01 << 5)
#define LSM9DS0_GYR_STATUS_Z_OVERRUN    (unsigned char)(0x01 << 4)
#define LSM9DS0_GYR_STATUS_XYZ_NEW_DATA   (unsigned char)(0x01 << 3)
#define LSM9DS0_GYR_STATUS_X_NEW_DATA   (unsigned char)(0x01 << 2)
#define LSM9DS0_GYR_STATUS_Y_NEW_DATA   (unsigned char)(0x01 << 1)
```

```
#define LSM9DS0_GYR_STATUS_Z_NEW_DATA   (unsigned char)(0x01 << 0)

struct r_reg
{
  unsigned char addr;
  unsigned char val;
};

struct rw_reg
{
  unsigned char addr;
  char def_val;
  unsigned char val;
};

struct magnitude_3d
{
  double x;
  double y;
  double z;
};

struct lsm9ds0_xm_reg
{
  /* Who am I */
  struct r_reg who_am_i_xm;
  /* Status */
  struct r_reg status_reg_a;
  struct r_reg status_reg_m;
  /* Out mag */
  struct r_reg out_x_l_m;
  struct r_reg out_x_h_m;
  struct r_reg out_y_l_m;
  struct r_reg out_y_h_m;
  struct r_reg out_z_l_m;
  struct r_reg out_z_h_m;
  /* Out acc */
  struct r_reg out_x_l_a;
  struct r_reg out_x_h_a;
  struct r_reg out_y_l_a;
  struct r_reg out_y_h_a;
  struct r_reg out_z_l_a;
  struct r_reg out_z_h_a;
  /* Control */
  struct rw_reg ctrl_reg0_xm;
  struct rw_reg ctrl_reg1_xm;
  struct rw_reg ctrl_reg2_xm;
  struct rw_reg ctrl_reg3_xm;
  struct rw_reg ctrl_reg4_xm;
  struct rw_reg ctrl_reg5_xm;
  struct rw_reg ctrl_reg6_xm;
  struct rw_reg ctrl_reg7_xm;
};

struct lsm9ds0_xm_dev
{
  int fd;
  unsigned char i2c_addr;
  struct lsm9ds0_xm_reg reg;
};

enum lsm9ds0_acc_scale
```

```
{
  LSM9DS0_ACC_SCALE_2,
  LSM9DS0_ACC_SCALE_4,
  LSM9DS0_ACC_SCALE_6,
  LSM9DS0_ACC_SCALE_8,
  LSM9DS0_ACC_SCALE_16,
  LSM9DS0_ACC_NUM_SCALES
};

enum lsm9ds0_acc_odr
{
  LSM9DS0_ACC_ODR_POWER_OFF,
  LSM9DS0_ACC_ODR_3_HZ_125,
  LSM9DS0_ACC_ODR_6_HZ_25,
  LSM9DS0_ACC_ODR_12_HZ_5,
  LSM9DS0_ACC_ODR_25_HZ,
  LSM9DS0_ACC_ODR_50_HZ,
  LSM9DS0_ACC_ODR_100_HZ,
  LSM9DS0_ACC_ODR_200_HZ,
  LSM9DS0_ACC_ODR_400_HZ,
  LSM9DS0_ACC_ODR_800_HZ,
  LSM9DS0_ACC_ODR_1600_HZ,
  LSM9DS0_ACC_NUM_RATES
};

enum lsm9ds0_acc_lowpass_bandwidth
{
  LSM9DS0_ACC_LP_BW_773_HZ,
  LSM9DS0_ACC_LP_BW_194_HZ,
  LSM9DS0_ACC_LP_BW_362_HZ,
  LSM9DS0_ACC_LP_BW_50_HZ,
  LSM9DS0_ACC_NUM_LP_BW
};

enum lsm9ds0_mag_scale
{
  LSM9DS0_MAG_SCALE_2,
  LSM9DS0_MAG_SCALE_4,
  LSM9DS0_MAG_SCALE_8,
  LSM9DS0_MAG_SCALE_12,
  LSM9DS0_MAG_NUM_SCALES
};

enum lsm9ds0_mag_odr
{
  LSM9DS0_MAG_ODR_POWER_OFF,
  LSM9DS0_MAG_ODR_3_HZ_125,
  LSM9DS0_MAG_ODR_6_HZ_25,
  LSM9DS0_MAG_ODR_12_HZ_5,
  LSM9DS0_MAG_ODR_25_HZ,
  LSM9DS0_MAG_ODR_50_HZ,
  LSM9DS0_MAG_ODR_100_HZ,
  LSM9DS0_MAG_NUM_RATES
};

enum lsm9ds0_mag_res
{
  LSM9DS0_MAG_LOW_RES,
  LSM9DS0_MAG_HIGH_RES,
  LSM9DS0_MAG_NUM_RES
};
```

```
struct lsm9ds0_gyr_reg
{
  /* Who am I */
  struct r_reg who_am_i_g;
  /* Status */
  struct r_reg status_reg_g;
  /* Out gyr */
  struct r_reg out_x_l_g;
  struct r_reg out_x_h_g;
  struct r_reg out_y_l_g;
  struct r_reg out_y_h_g;
  struct r_reg out_z_l_g;
  struct r_reg out_z_h_g;
  /* Control */
  struct rw_reg ctrl_reg1_g;
  struct rw_reg ctrl_reg2_g;
  struct rw_reg ctrl_reg3_g;
  struct rw_reg ctrl_reg4_g;
  struct rw_reg ctrl_reg5_g;
};

struct lsm9ds0_gyr_dev
{
  int fd;
  unsigned char i2c_addr;
  struct lsm9ds0_gyr_reg reg;
};

enum lsm9ds0_gyr_odr{
  LSM9DS0_GYR_POWER_OFF,
  LSM9DS0_GYR_ODR_95,
  LSM9DS0_GYR_ODR_190,
  LSM9DS0_GYR_ODR_380,
  LSM9DS0_GYR_ODR_760,
  LSM9DS0_GYR_NUM_RATES
};

enum lsm9ds0_gyr_lowpass_bandwidth
{
  LSM9DS0_GYR_LP_BW_12_HZ_5,
  LSM9DS0_GYR_LP_BW_20_HZ,
  LSM9DS0_GYR_LP_BW_25_HZ,
  LSM9DS0_GYR_LP_BW_30_HZ,
  LSM9DS0_GYR_LP_BW_35_HZ,
  LSM9DS0_GYR_LP_BW_50_HZ,
  LSM9DS0_GYR_LP_BW_70_HZ,
  LSM9DS0_GYR_LP_BW_100_HZ,
  LSM9DS0_GYR_NUM_LP_BW
};

enum lsm9ds0_gyr_scale
{
  LSM9DS0_GYR_SCALE_245_DPS,
  LSM9DS0_GYR_SCALE_500_DPS,
  LSM9DS0_GYR_SCALE_2000_DPS,
  LSM9DS0_GYR_NUM_SCALES
};

int test(int number);

void print_8bit(const unsigned char value);
void print_16bit(const unsigned short value);
```

```
static int i2c_check_addr(const unsigned char addr);

static int i2c_read_byte( const int fd,
              const unsigned char i2c_addr,
              const unsigned char reg_addr,
              unsigned char *reg_val);

static int i2c_read_byte_array( const int fd,
                const unsigned char i2c_arr,
                const unsigned char start_reg_addr,
                unsigned char num_regs,
                unsigned char *reg_values);

static int i2c_write_byte(  const int fd,
              const unsigned char i2c_addr,
              const unsigned char reg_addr,
              const unsigned char reg_val);

static int i2c_read_rreg( const int fd,
              const unsigned char i2c_addr,
              struct r_reg *r_reg);

static int i2c_read_rwreg(  const int fd,
              const unsigned char i2c_addr,
              struct rw_reg *rw_reg);

static int i2c_write_rwreg( const int fd,
              const unsigned char i2c_addr,
              struct rw_reg *rw_reg,
              const unsigned char reg_val);

static int i2c_read(  const int fd,
            const unsigned char i2c_addr,
            const unsigned char reg_addr,
            unsigned int num_reg,
            unsigned char *buffer);

static int i2c_write( const int fd,
            const unsigned char i2c_addr,
            const unsigned char reg_addr,
            unsigned int num_reg,
            unsigned char *buffer);

int lsm9ds0_xm_create(  const char *device,
            const unsigned char i2c_addr,
            struct lsm9ds0_xm_dev *sensor);

int lsm9ds0_xm_delete(struct lsm9ds0_xm_dev *sensor);
int lsm9ds0_xm_init(struct lsm9ds0_xm_dev *sensor);

int lsm9ds0_xm_power_on(struct lsm9ds0_xm_dev *sensor,
            const enum lsm9ds0_acc_odr acc_odr,
            const enum lsm9ds0_acc_lowpass_bandwidth acc_bw,
            const enum lsm9ds0_mag_odr mag_odr,
            const enum lsm9ds0_mag_res mag_res);

int lsm9ds0_xm_power_off(struct lsm9ds0_xm_dev *sensor);

int lsm9ds0_xm_get_acc_status(struct lsm9ds0_xm_dev *sensor, unsigned char *status);
int lsm9ds0_xm_get_acc_scale(struct lsm9ds0_xm_dev *sensor, enum lsm9ds0_acc_scale *scale);
int lsm9ds0_xm_set_acc_scale(struct lsm9ds0_xm_dev *sensor, const enum lsm9ds0_acc_scale scale);
```

```
int lsm9ds0_xm_get_acc(struct lsm9ds0_xm_dev *sensor, struct magnitude_3d *acc);

int lsm9ds0_xm_get_mag_status(struct lsm9ds0_xm_dev *sensor, unsigned char *status);
int lsm9ds0_xm_get_mag_scale(struct lsm9ds0_xm_dev *sensor, enum lsm9ds0_mag_scale *scale);
int lsm9ds0_xm_set_mag_scale(struct lsm9ds0_xm_dev *sensor, const enum lsm9ds0_mag_scale scale);
int lsm9ds0_xm_get_mag(struct lsm9ds0_xm_dev *sensor, struct magnitude_3d *mag);

int lsm9ds0_gyr_create( const char *device,
            const unsigned char i2c_addr,
            struct lsm9ds0_gyr_dev *sensor);

int lsm9ds0_gyr_delete(struct lsm9ds0_gyr_dev *sensor);
int lsm9ds0_gyr_init(struct lsm9ds0_gyr_dev *sensor);

int lsm9ds0_gyr_power_on( struct lsm9ds0_gyr_dev *sensor,
             const enum lsm9ds0_gyr_odr gyr_odr,
             const enum lsm9ds0_gyr_lowpass_bandwidth gyr_bw);

int lsm9ds0_gyr_power_off(struct lsm9ds0_gyr_dev *sensor);

int lsm9ds0_gyr_get_status(struct lsm9ds0_gyr_dev *sensor, unsigned char *status);
int lsm9ds0_gyr_get_scale(struct lsm9ds0_gyr_dev *sensor, enum lsm9ds0_gyr_scale *scale);
int lsm9ds0_gyr_set_scale(struct lsm9ds0_gyr_dev *sensor, const enum lsm9ds0_gyr_scale scale);
int lsm9ds0_gyr_get_gyr(struct lsm9ds0_gyr_dev *sensor, struct magnitude_3d *gyr);


#endif
```

A.4.3 Makefile

```
# ********************************************************************************
# File:   Makefile
# Author: Albert Ruiz
#
# This is the Makefile to compile the user-space driver for LSM9DS0 sensor
# It also includes a test application: lsm9ds0_test
#
# The driver is compiled as a static library: liblsm9ds0.a
# To use the static library with any other application, two files are needed:
# - liblsm9ds0.a
#   - lsm9ds0.h
#
# To compile it to native architecture, just run:
# make
#
# To compile it to aarch64:
# make ARCH=aarch64
# ********************************************************************************

TARGET := lsm9ds0_test
LIB := ./lib/liblsm9ds0.a
INC := ./include

CC := gcc
AR := ar

ifeq ($(ARCH),aarch64)
  CC := aarch64-linux-gnu-$(CC)
```

```
  AR := aarch64-linux-gnu-$(AR)
  TARGET := $(TARGET).aarch64
endif

# CFLAGS := -g -Wall
CFLAGS := -I$(INC)
ARFLAGS:= rs

$(TARGET): lsm9ds0_main.o $(LIB)
  $(CC) -o $(TARGET) lsm9ds0_main.o $(LIB)

$(LIB): lsm9ds0.o
  $(AR) $(ARFLAGS) $(LIB) lsm9ds0.o

lsm9ds.o: lsm9ds0.c lsm9ds0.h
  $(CC) $(CFLAGS) -c lsm9ds0.c

clean:
  rm -f $(TARGET) $(TARGET).aarch64 $(LIB) *.o
```

A.4.4 Upload Script

The following script was created to upload binaries to Rock960 easily:

```
# ********************************************************************************
# File:   upload.sh
# Author: Albert Ruiz
#
# Upload script
# ********************************************************************************

USER_HOST=albert-rz@192.168.0.169
PWD=linus10

# Driver test binary
DRIVER_BIN=lsm9ds0_test.aarch64
DRIVER_IN_HOST=./driver/$DRIVER_BIN

# Final application binary
WEBSOCK_BIN=websock_server.aarch64
WEBSOCK_IN_HOST=./websocket_server/$WEBSOCK_BIN

# Upload driver test binary
if [ ! -f $DRIVER_IN_HOST ]
then
    echo "$DRIVER_IN_HOST not found!"
    exit -1
else
  sshpass -p $PWD scp $DRIVER_IN_HOST $USER_HOST:Workspace/rk_final_app/$DRIVER_BIN
  echo "$DRIVER_IN_HOST uploaded"
fi

# Upload final application binary
if [ ! -f $WEBSOCK_IN_HOST ]
then
    echo "$WEBSOCK_IN_HOST not found!"
    exit -1
else
```

```
  sshpass -p $PWD scp $WEBSOCK_IN_HOST $USER_HOST:Workspace/rk_final_app/$WEBSOCK_BIN
  echo $WEBSOCK_IN_HOST" uploaded"
fi
```

Appendix B 40 Pin Low Speed Connector

.

B.1 Power and Reset

Table B.1: Power and reset pins description.

Pin	Signal	Description	Voltage	Type
1, 2, 39, 40	GND	Voltage zero reference	$0V$	Power
35	$+1.8V$	$+1.8V$ power reference (max $0.1A$)	$+1.8V$	Output
37	$+5V$	+5V system power supply	$+5V$	Power
36, 38	SYS_DCIN	Input power supply	$+12V$	Power
4	PWR_BTN_N	Power on/off external request	$+1.8V$	Input
6	RST_BTN_N	Reset external request	$+1.8V$	Input

B.2 UART

Table B.2: UART pins description.

Pin	Signal	Description	Voltage	Type
3	UART0_CTS	Clear to send control	$+1.8V$	Input
5	UART0_TxD	Transmit serial data	$+1.8V$	Output
7	UART0_RxD	Receive serial data	$+1.8V$	Input
9	UART0_RTS	Request to send control	$+1.8V$	Output
11	UART1_TxD	Transmit serial data	$+1.8V$	Output
13	UART1_RxD	Receive serial data	$+1.8V$	Input

B.3 I^{2}C

Table B.3: I^{2}C pins description.

Pin	Signal	Description	Voltage	Type
15	I2C0_SCL	Serial clock	$+1.8V$	Open collector
17	I2C0_SDA	Serial data	$+1.8V$	Open Collector
19	I2C0_SCL	Serial clock	$+1.8V$	Open collector
21	I2C0_SDA	Serial data	$+1.8V$	Open Collector

B.4 SPI

Table B.4: SPI pins description.

Pin	Signal	Description	Voltage	Type
8	SPI0_SCLK	Serial clock	$+1.8V$	Output
10	SPI0_DIN	Data in	$+1.8V$	Input
12	SPI0_CS	Chip select	$+1.8V$	Output
14	SPI0_DOUT	Data out	$+1.8V$	Output

B.5 GPIO

Table B.5: GPIO pins description.

Pin	Signal	Description	Voltage	Type
23	GPIO-A	General purpose I/O	$+1.8V$	Output
24	GPIO-B	General purpose I/O	$+1.8V$	Output
25	GPIO-C	General purpose I/O	$+1.8V$	Output
26	GPIO-D	General purpose I/O	$+1.8V$	Output
27	GPIO-E	General purpose I/O	$+1.8V$	Output
28	GPIO-F	General purpose I/O	$+1.8V$	Output
29	GPIO-G	General purpose I/O	$+1.8V$	Output
30	GPIO-H	General purpose I/O	$+1.8V$	Output
31	GPIO-I	General purpose I/O	$+1.8V$	Output
32	GPIO-J	General purpose I/O	$+1.8V$	Output
33	GPIO-K	General purpose I/O	$+1.8V$	Output
34	GPIO-L	General purpose I/O	$+1.8V$	Output

B.6 PCM/I^{2}S

Table B.6: PMI/I^{2}S pins description.

Pin	Signal	Description	Voltage	Type
16	PCM_FS	PCM/I2S word clock	$+1.8V$	Output
18	PCM_CLK	PCM/I2S bit clock	$+1.8V$	Output
20	PCM_DO	PCM/I2S serial data out	$+1.8V$	Output
22	PCM_DI	PCM/I2S serial data in	$+1.8V$	Input

Appendix C Schematics

C.1 Rock960 Board

List of diagrams:

- Power tree (view)
- I^2C map (view)
- Power domain main (view)
- RK3399 power (view)
- RK3399 PMU controller (view)
- RK3399 DDR controller (view)
- RK3399 flash and DMMC controller (view)
- RK3399 USB controller (view)
- RK3399 SARADC/Key (view)
- RK3399 DVP interface (view)
- RK3399 display interface (view)
- RK3399 GPIO (view)
- RK3399 PCIE (view)
- DC power in (view)
- PMIC power (view)
- USB OTG/HOST port (view)
- USB Type-C port (view)
- RAM LPDDR3 (view)
- eMMC memory (view)
- WIFI/B (view)
- TF card (view)
- HDMI output (view)
- PCIe (view)
- Connectors (view)

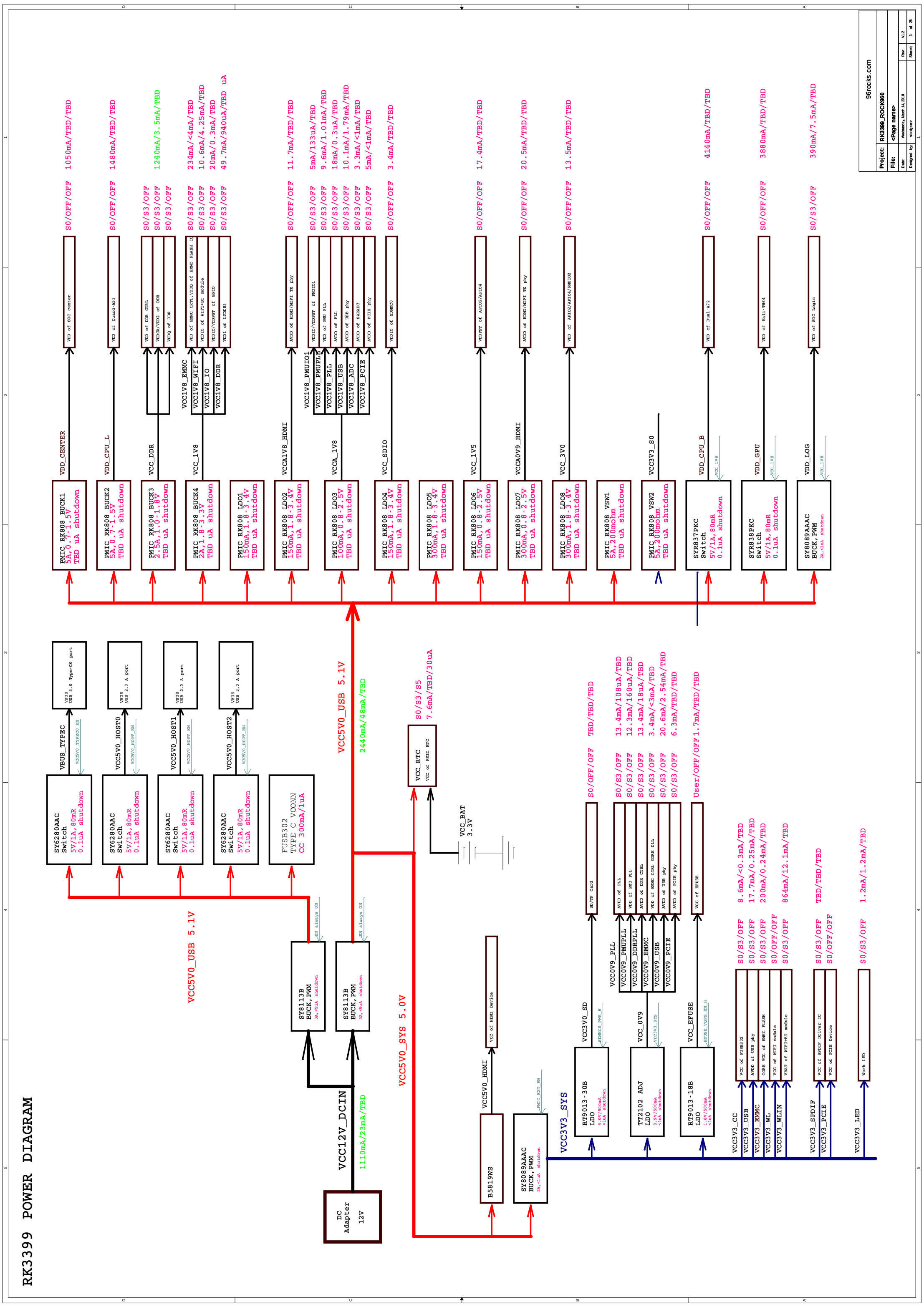


Figure C.1: Schematic - Rock960 power tree.

I2C MAP

Port	Pin name	Domain	Bus name	Pull-up voltage	Slave Device	Slave Addr (MS 7Bits)	Note	Slave Bus Capability
I2C0	GPIO1_B7/SPI3_RXD/I2C0_SDA GPIO1_C0/SPI3_TXD/I2C0_SCL	PMUIO2	I2C_SDA_PMIC I2C_SCL_PMIC	VCC_1V8	Rockchip RK808	0x1b	PMIC	100kHz,400KHz
					SYR837PKC	0x40	DC-DC BUCK	100kHz,400KHz,3.4MHz
					SYR838PKC	0x41	DC-DC BUCK	100kHz,400KHz,3.4MHz
I2C1	GPIO4_A1/I2C1_SDA GPIO4_A2/I2C1_SCL	APIO5		VCC_1V8			Low Speed CONNECTOR	
I2C2	GPIO2_A0/VOP_D0/CIF_D0/I2C2_SDA GPIO2_A1/VOP_D1/CIF_D1/I2C2_SCL	APIO2		VCC_1V8			High Speed CONNECTOR	
I2C3	GPIO4_C0/I2C3_SDA/UART2B_RX GPIO4_C1/I2C3_SCL/UART2B_TX	APIO4	I2C_SDA_HDMI I2C_SCL_HDMI	VCC_3V0				
I2C4	GPIO1_B3/I2C4_SDA GPIO1_B4/I2C4_SCL	PMUIO2	I2C_SDA_MEMS I2C_SCL_MEMS	VCC_1V8	Fairchild FUSB302B	0x44,0x46	USB-TypeC Mux	100kHz,400KHz,1MHz
I2C5	GPIO3_B2/MAC_RXER/I2C5_SDA GPIO3_B3/MAC_CLK/I2C5_SCL	APIO1	Other pin function					
I2C6	GPIO2_B1/SPI2_RXD/CIF_HREF/I2C6_SDA GPIO2_B2/SPI2_TXD/CIF_CLKIN/I2C6_SCL	APIO2		VCC_1V8			Low Speed CONNECTOR	
I2C7	GPIO2_A7/VOP_D7/CIF_D7/I2C7_SDA GPIO2_B0/VOP_CLK/CIF_VSYNC/I2C7_SCL	APIO2		VCC_1V8			High Speed CONNECTOR	

96rocks.com

Project: RK3399_ROCK960

File: <Page name>

Date: Wednesday, March 14, 2018 Rev: V1.2

Designed by: <designer> Sheet: 4 of 26

Figure C.2: Schematic - Rock960 I^{2}C map.

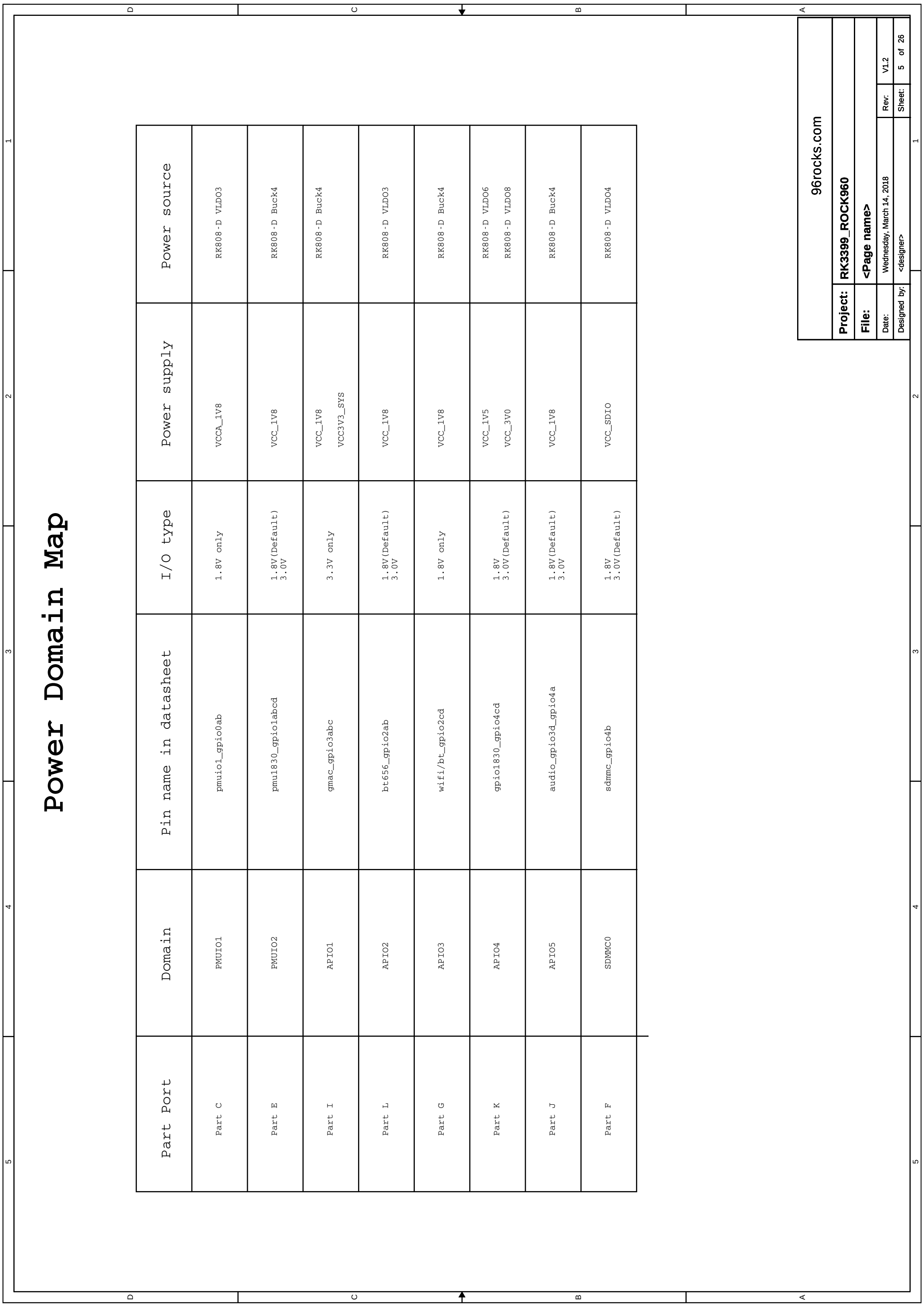


Figure C.3: Schematic - Rock960 power domain main.

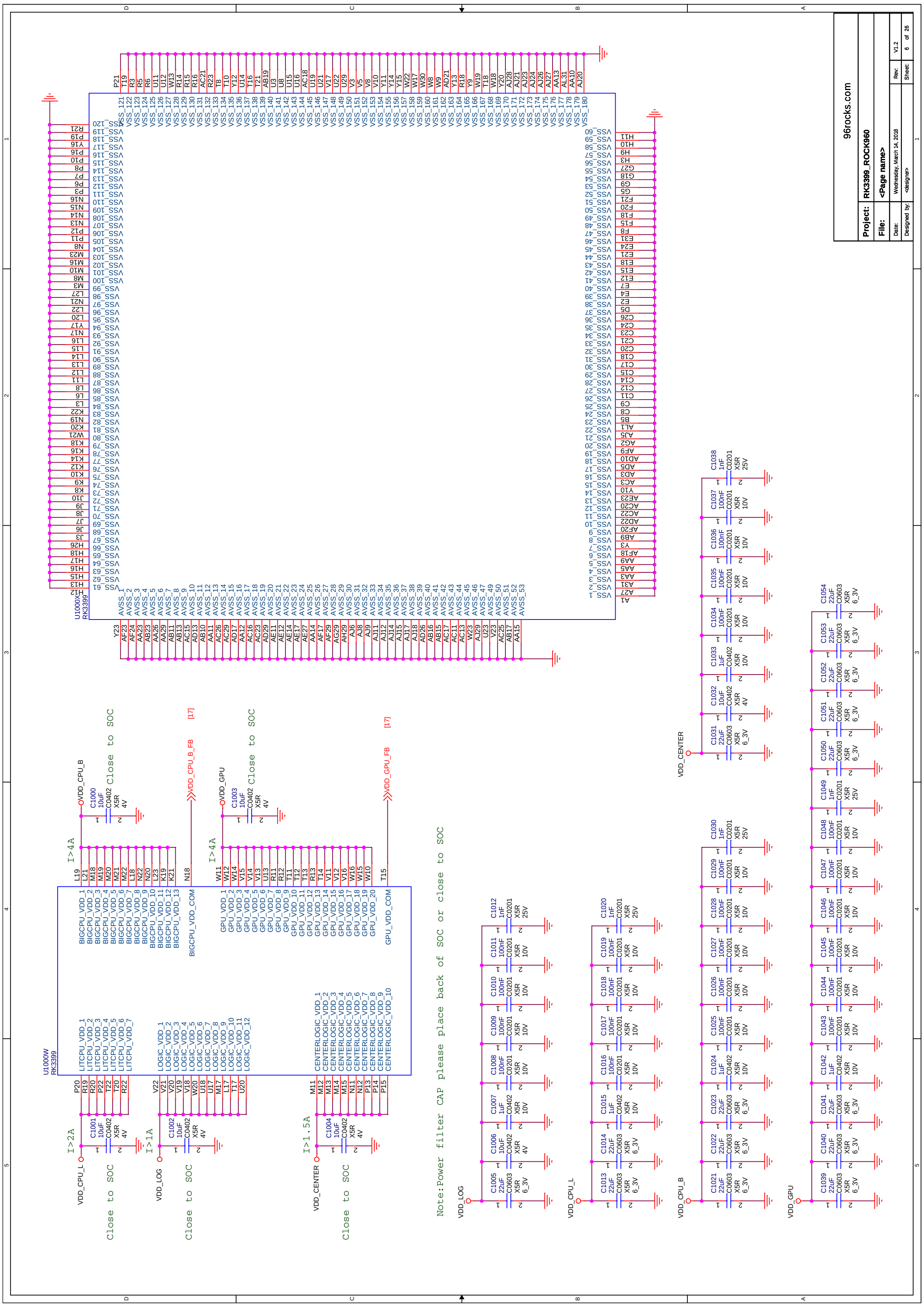


Figure C.4: Schematic - Rock960 RK3399 power.

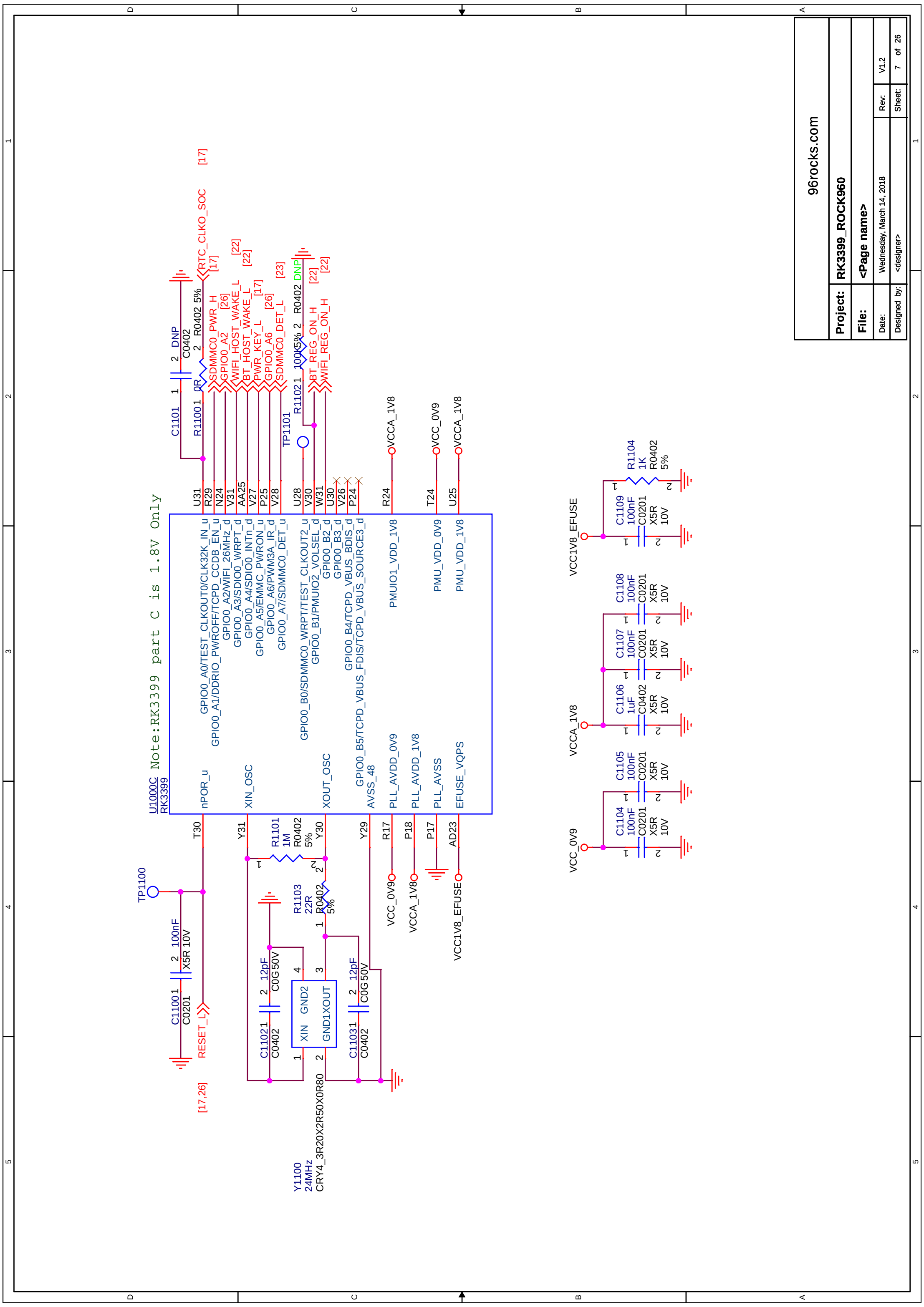


Figure C.5: Schematic - Rock960 RK3399 PMU controller.

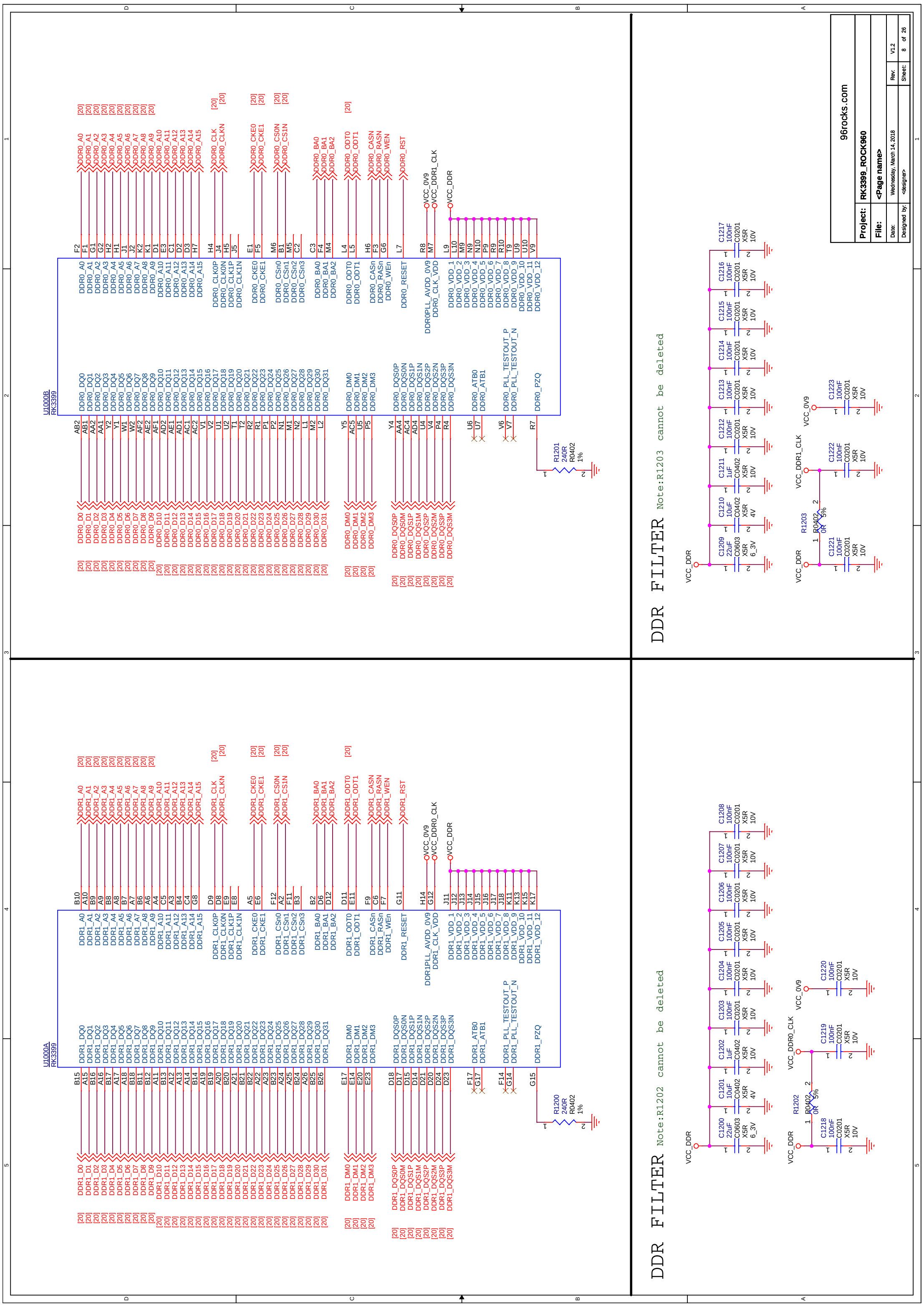


Figure C.6: Schematic - Rock960 RK3399 DDR controller.

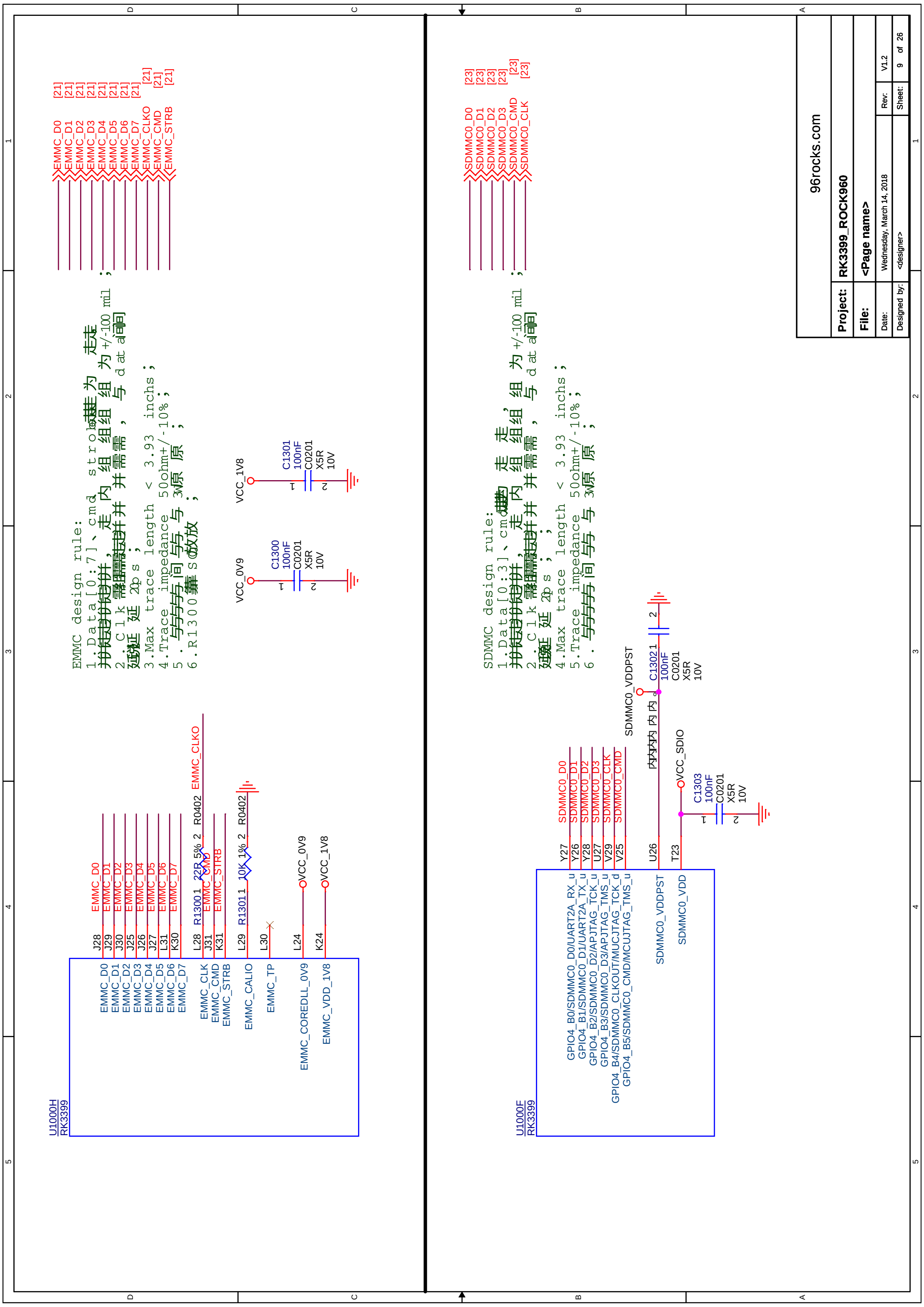


Figure C.7: Schematic - Rock960 RK3399 flash and DMMC controller.

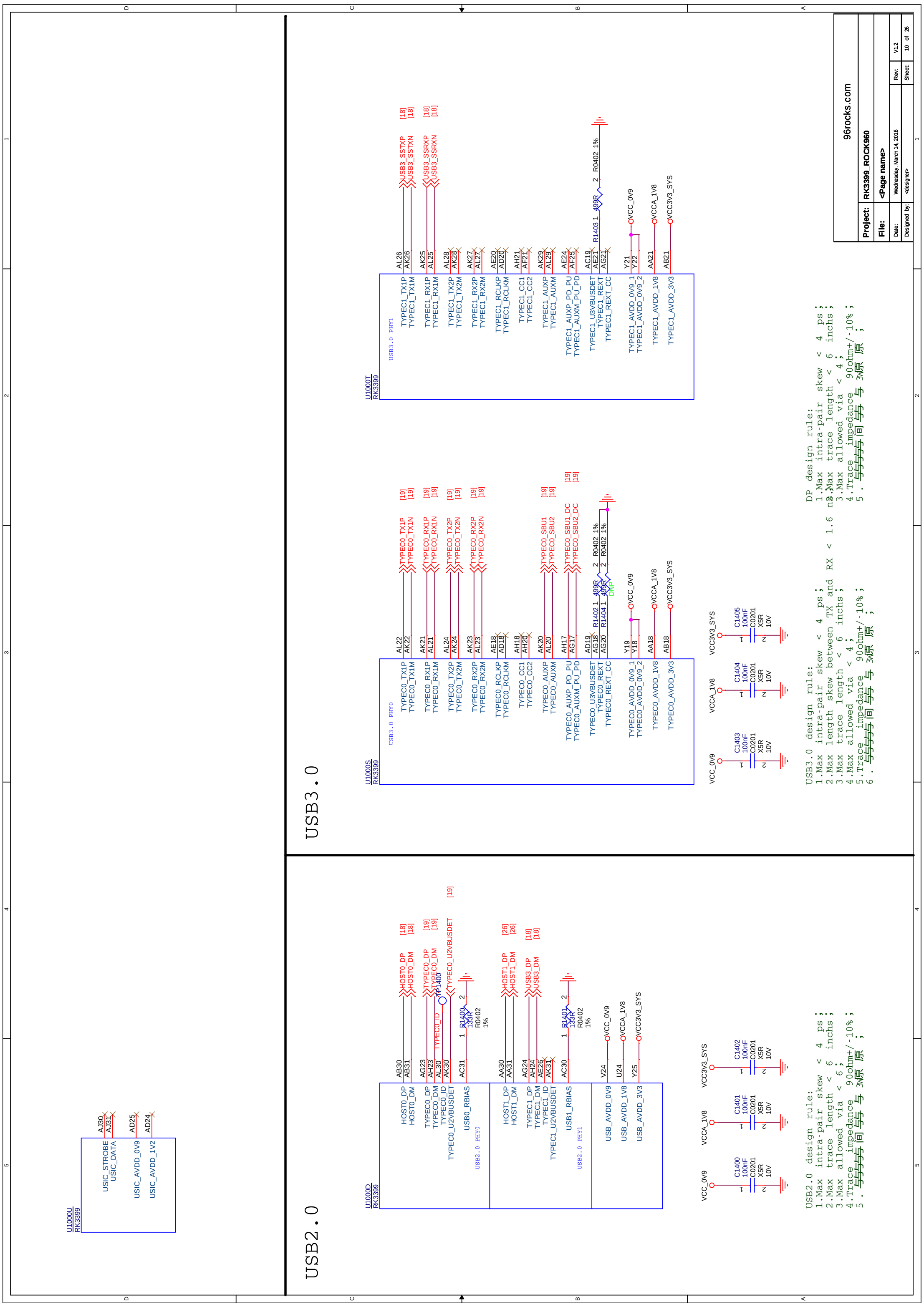


Figure C.8: Schematic - Rock960 RK3399 USB controller.

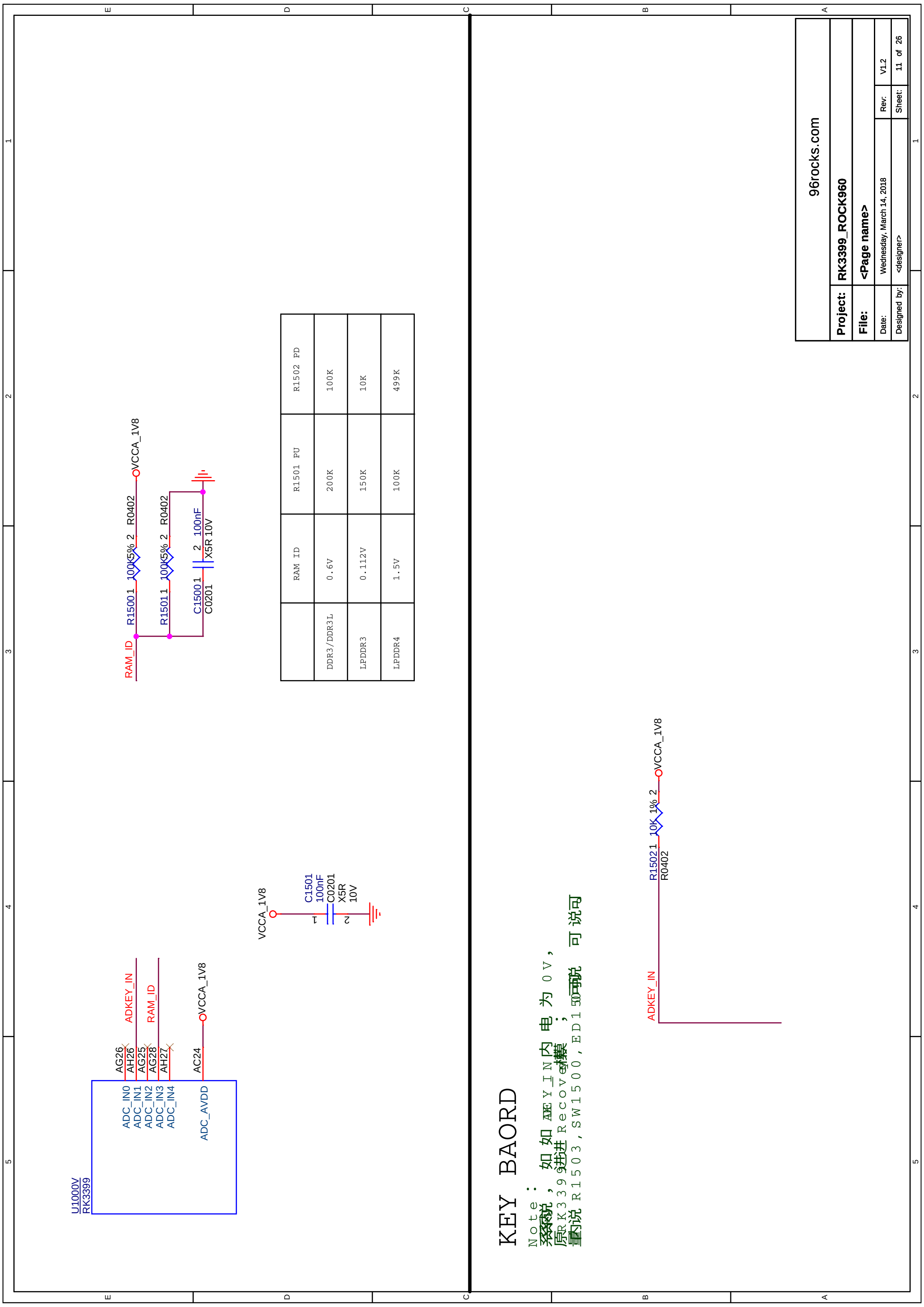


Figure C.9: Schematic - Rock960 RK3399 SARADC/Key.

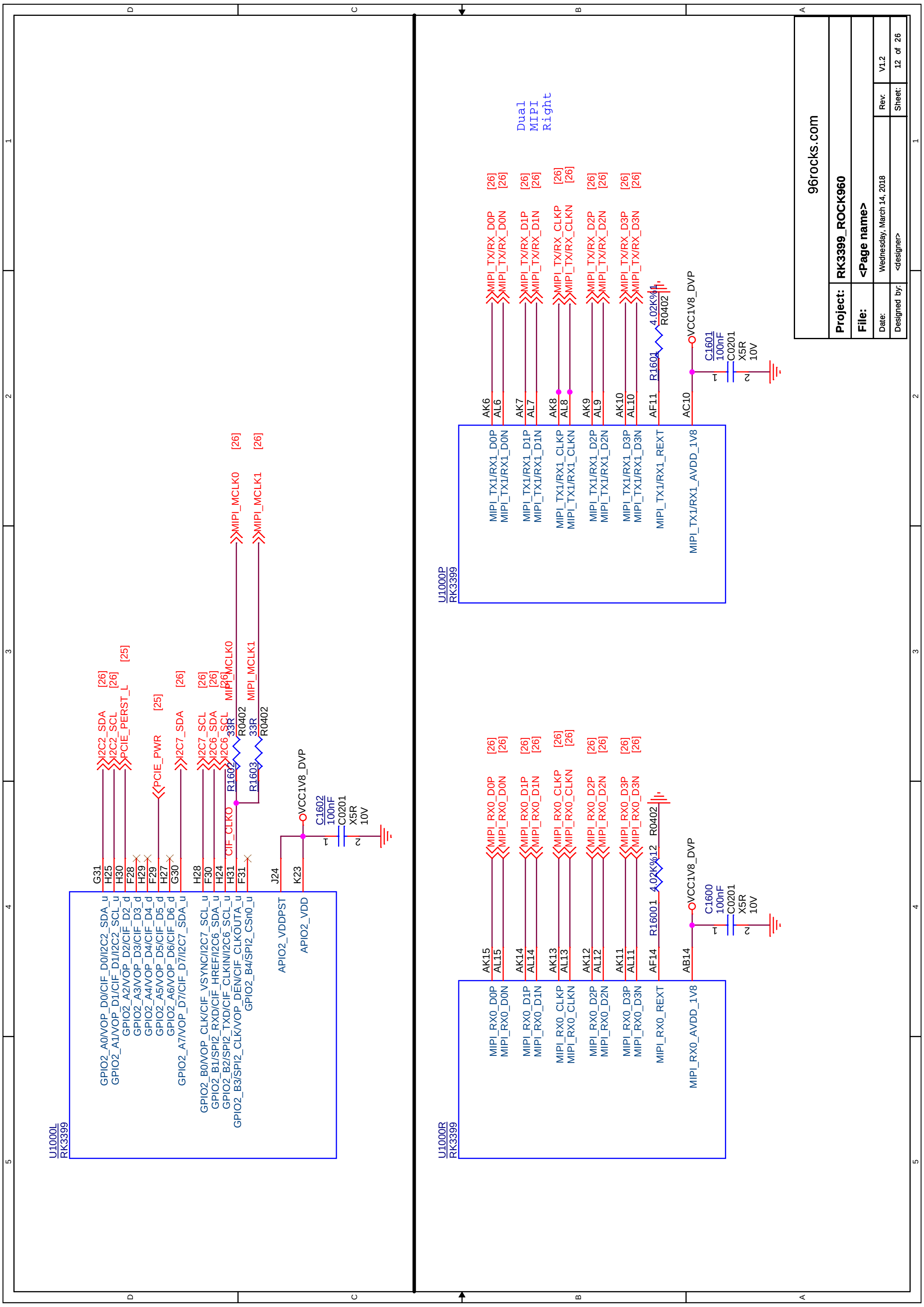


Figure C.10: Schematic - Rock960 RK3399 DVP interface.

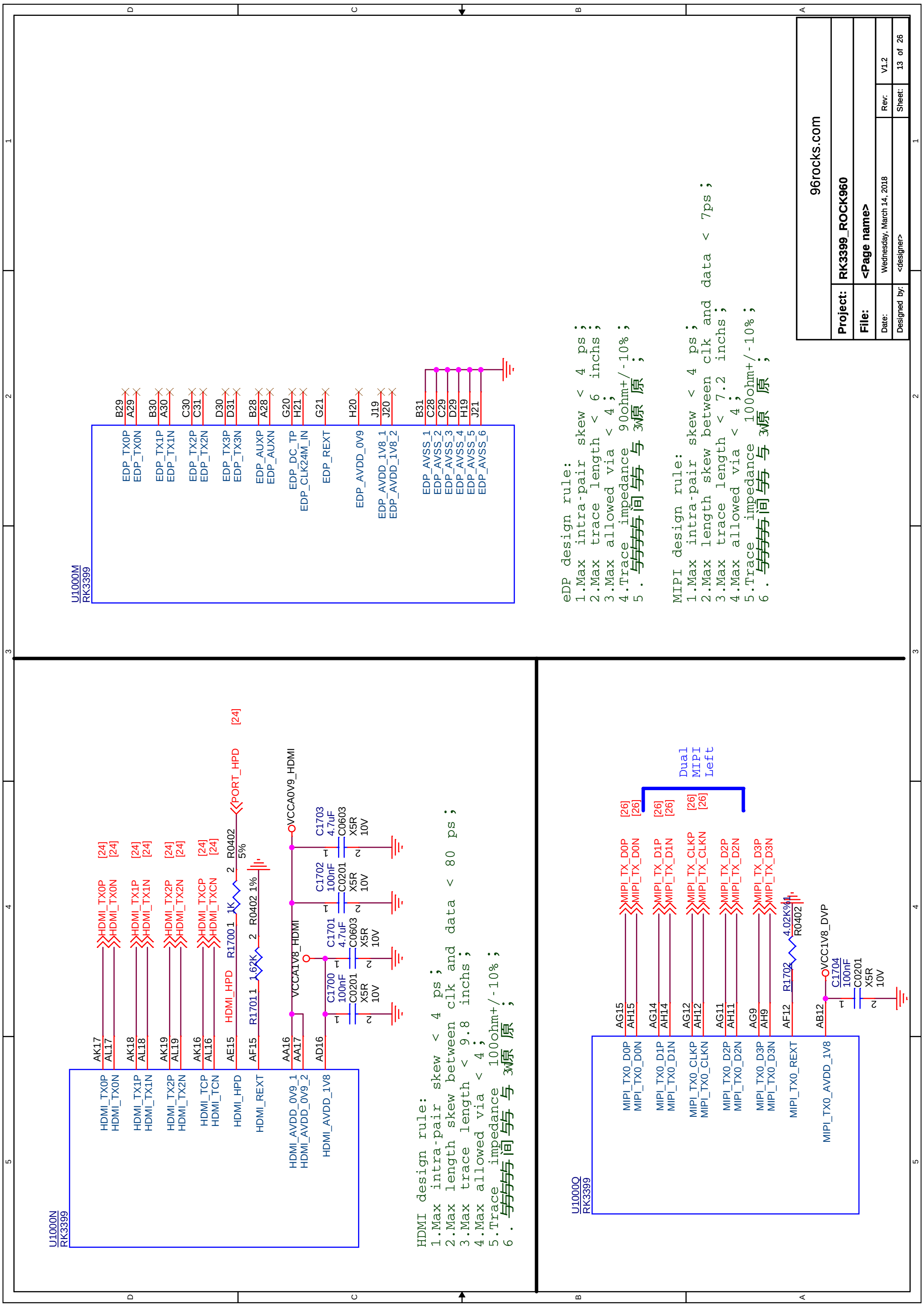


Figure C.11: Schematic - Rock960 RK3399 display interface.

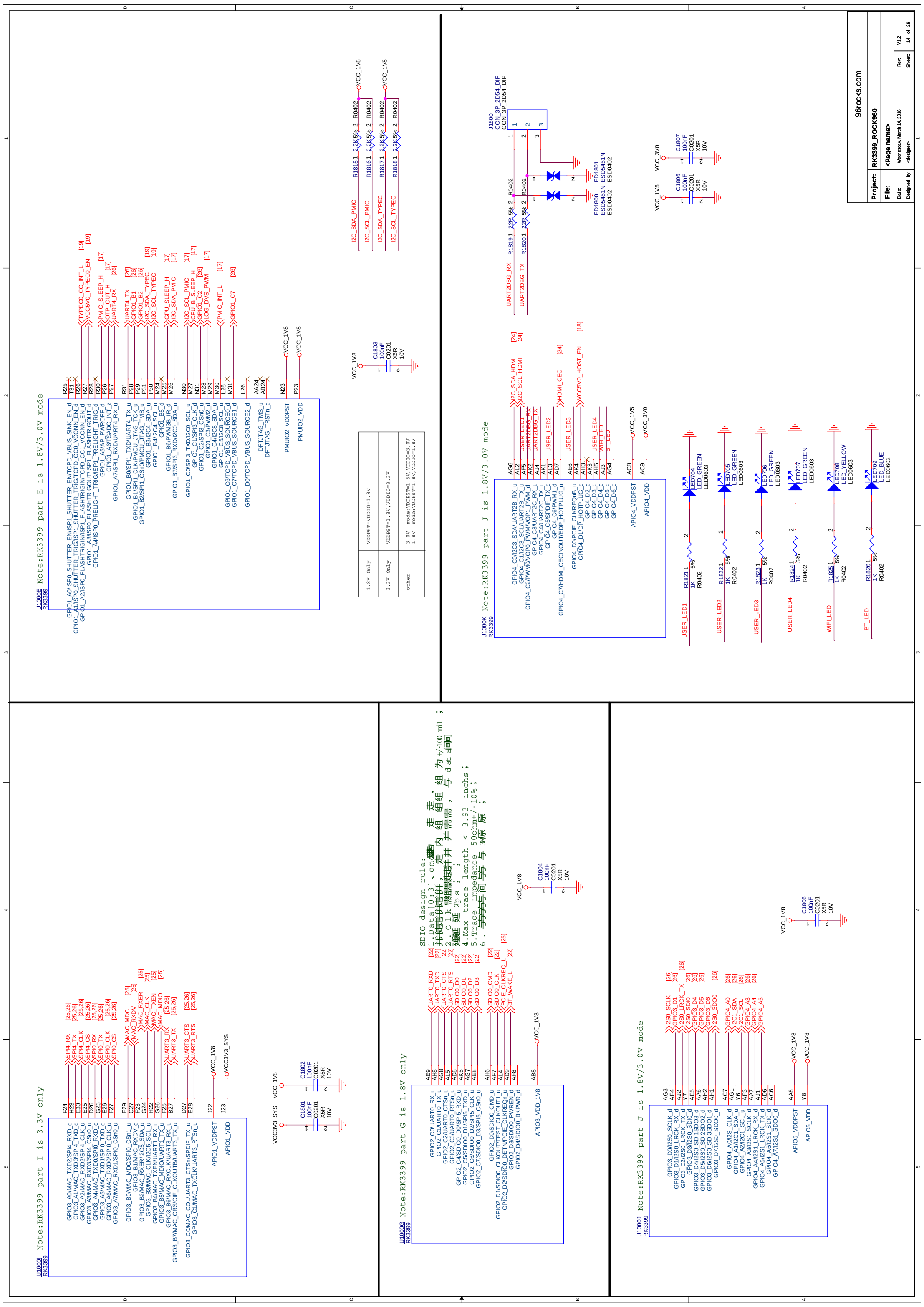


Figure C.12: Schematic - Rock960 RK3399 GPIO.

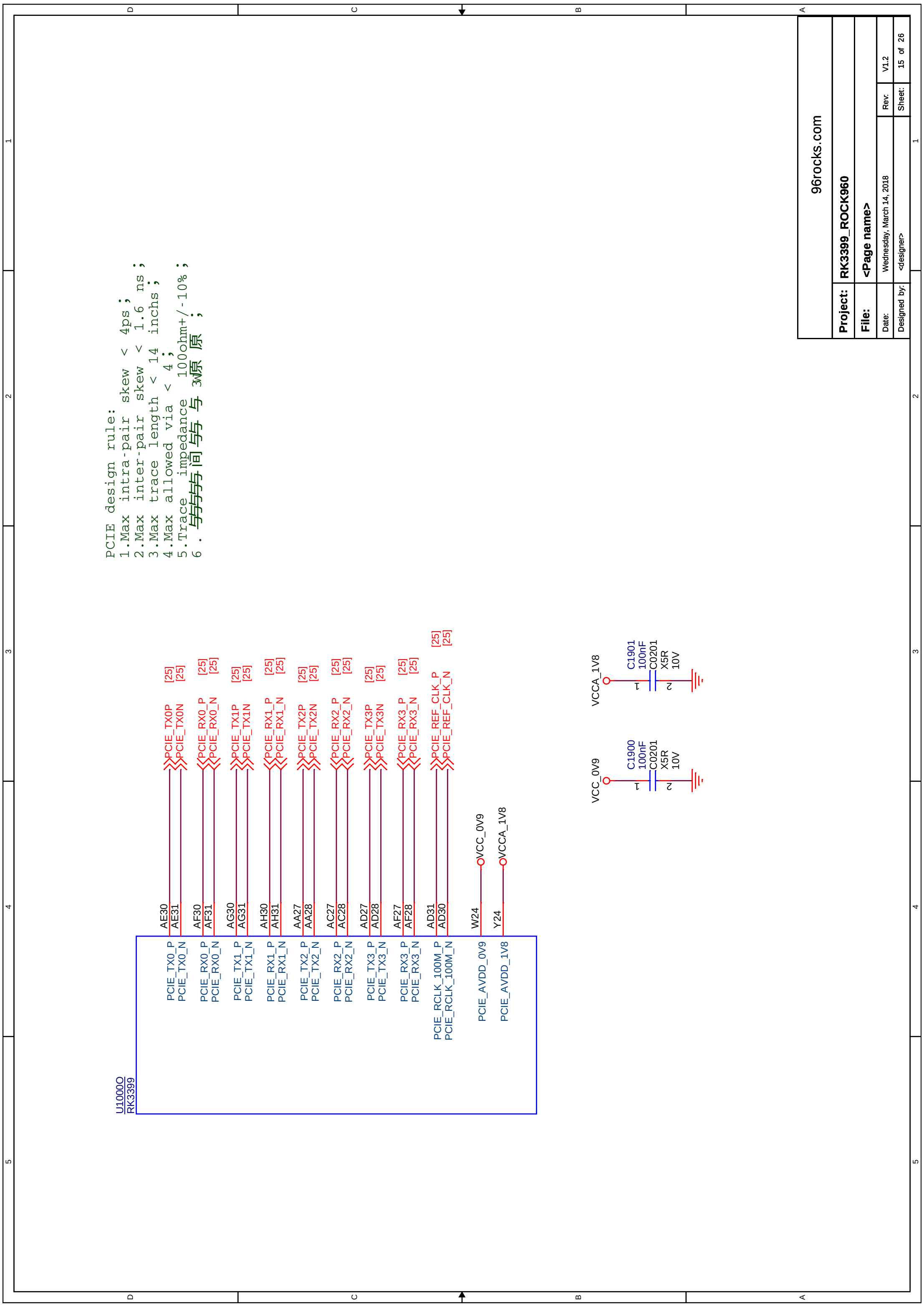


Figure C.13: Schematic - Rock960 RK3399 PCIE.

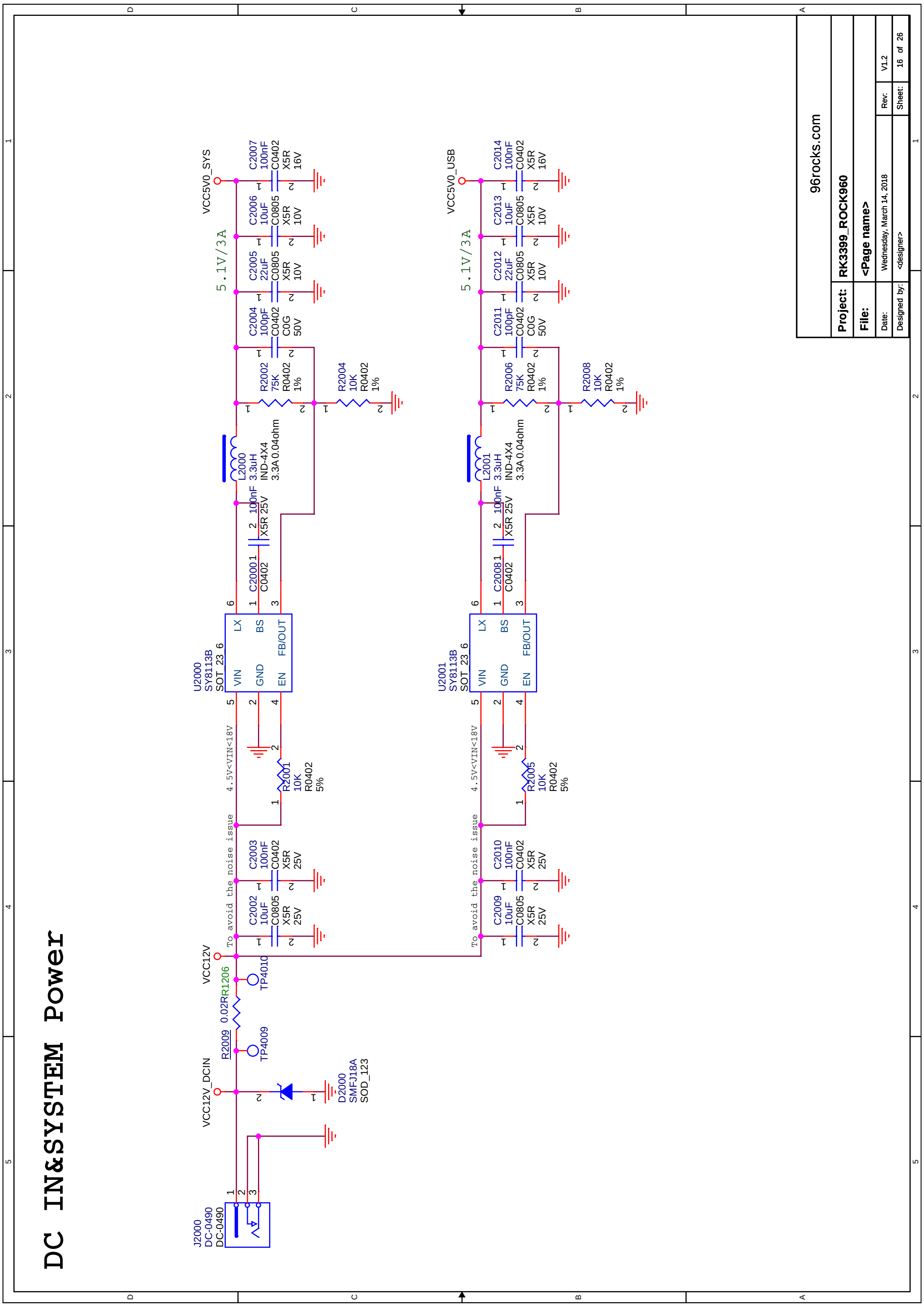


Figure C.14: Schematic - Rock960 DC power in.

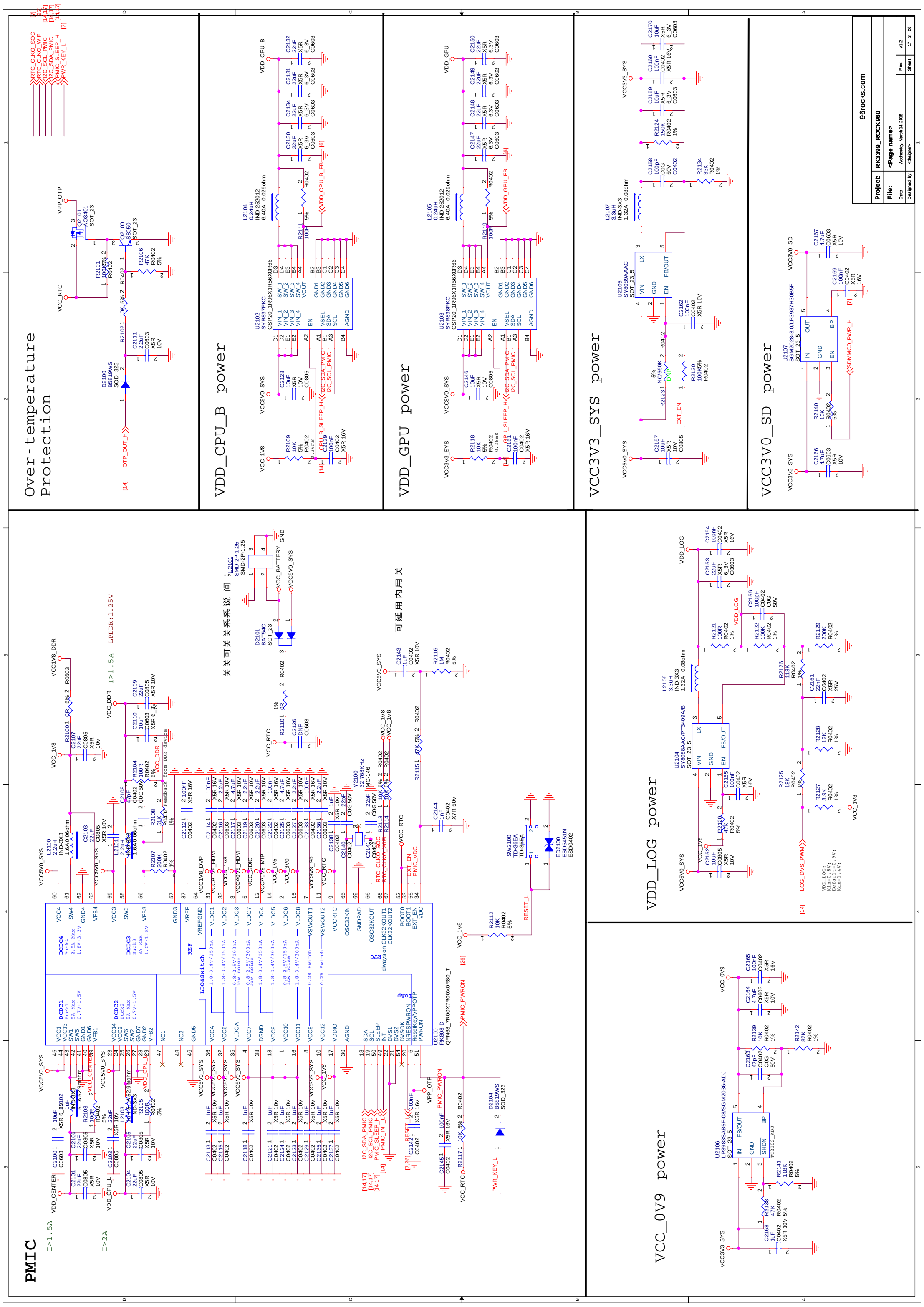


Figure C.15: Schematic - Rock960 PMIC power.

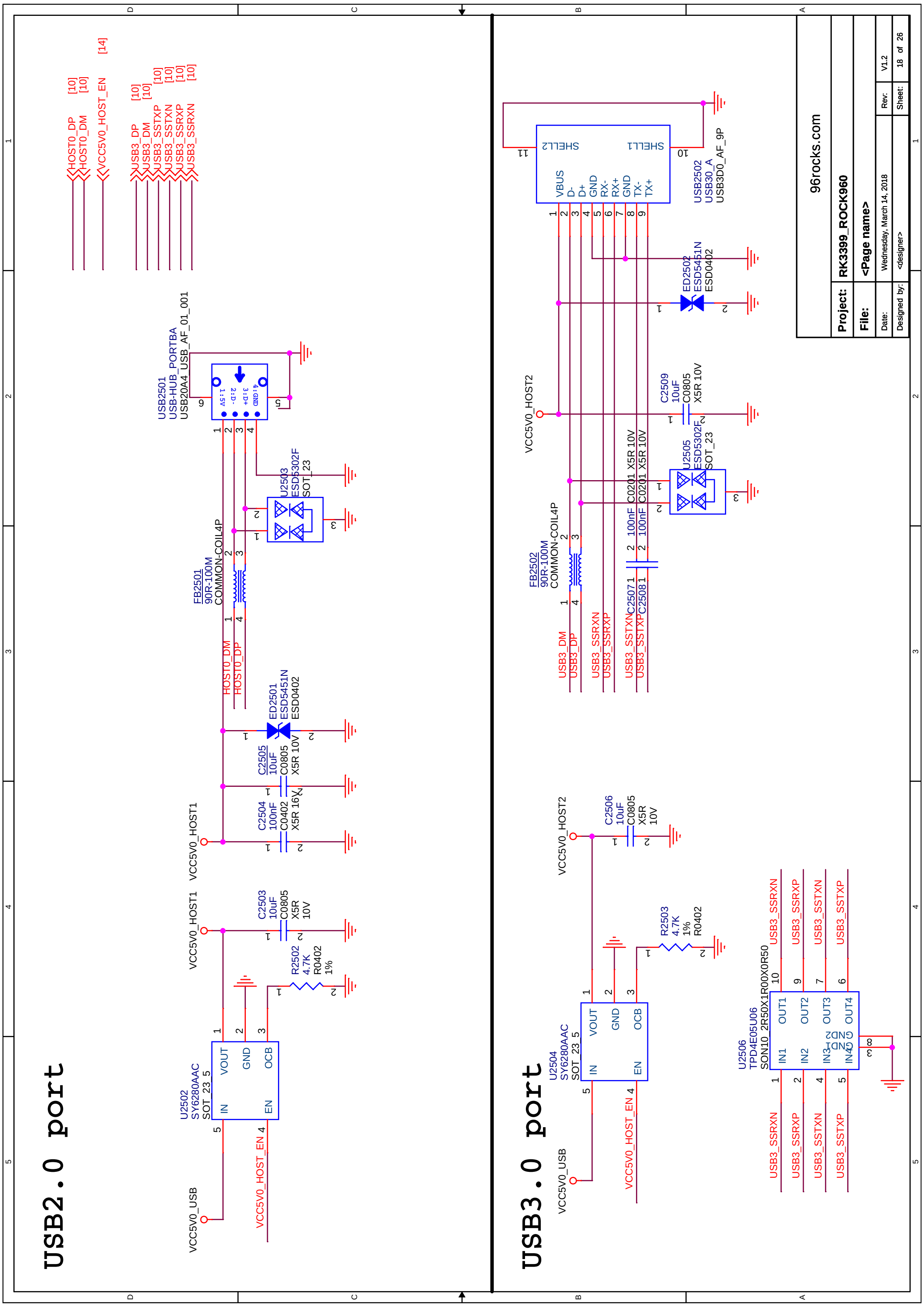


Figure C.16: Schematic - Rock960 USB OTG/HOST port.

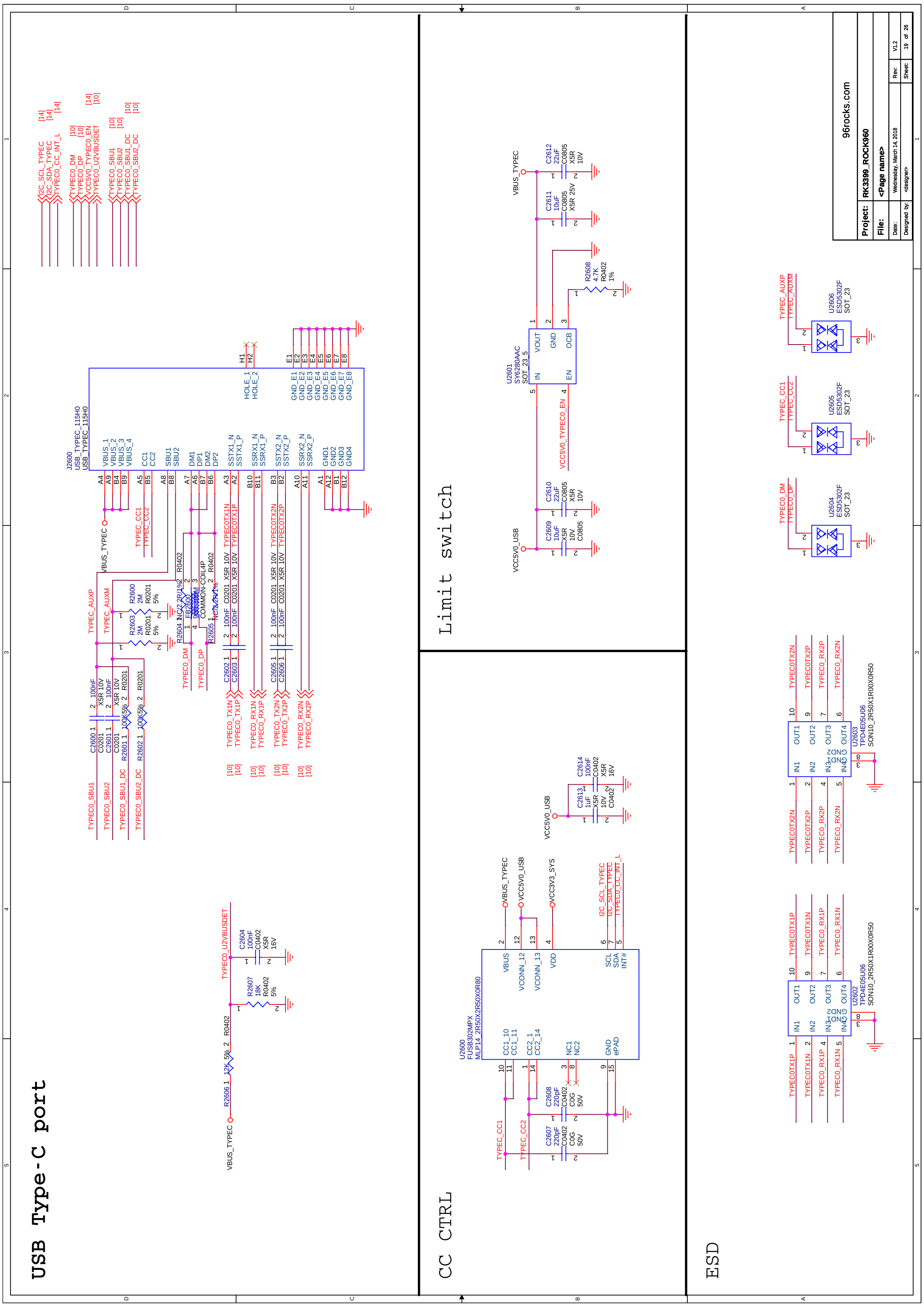


Figure C.17: Schematic - Rock960 USB Type-C port.

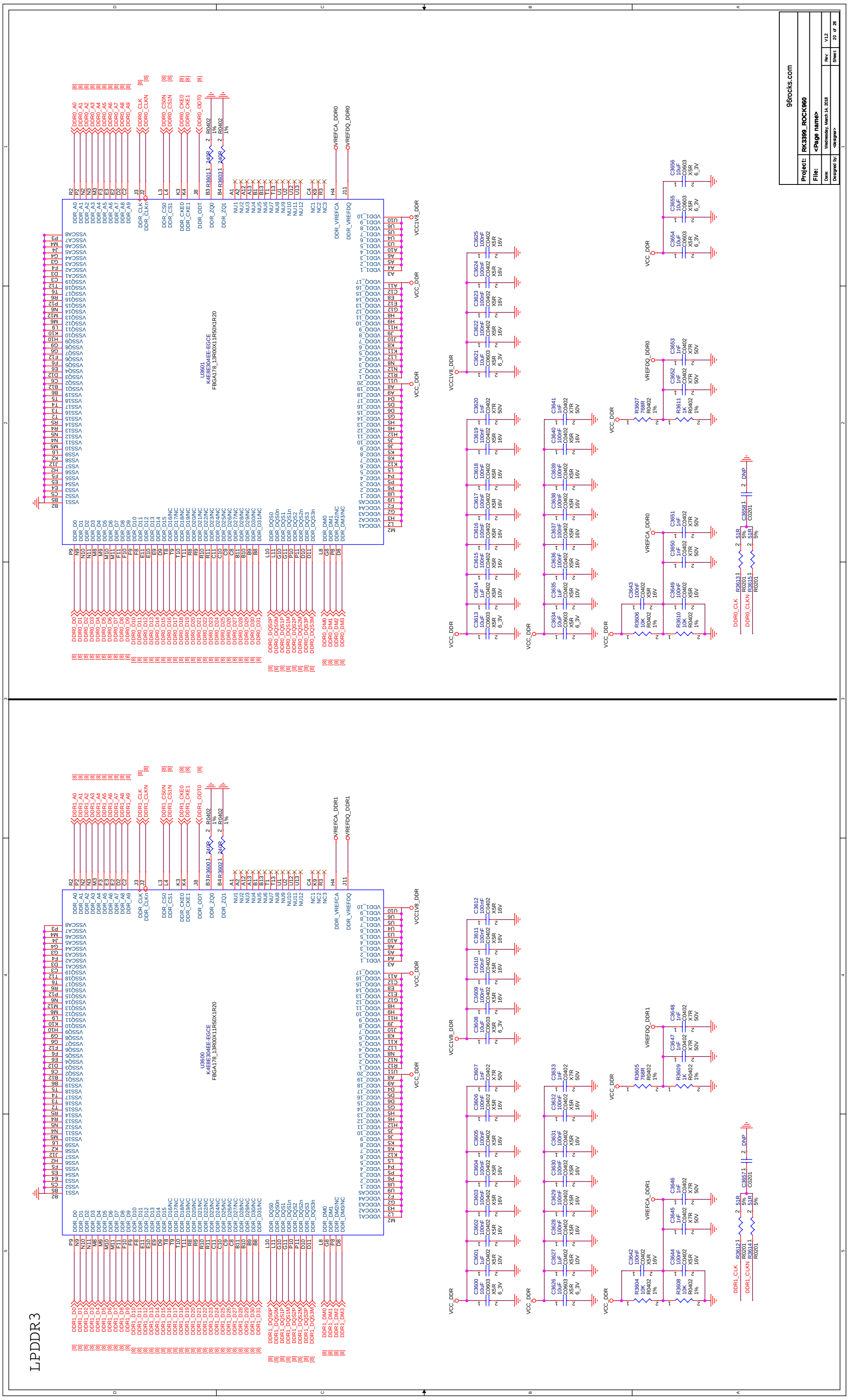


Figure C.18: Schematic - Rock960 RAM LPDDR3.

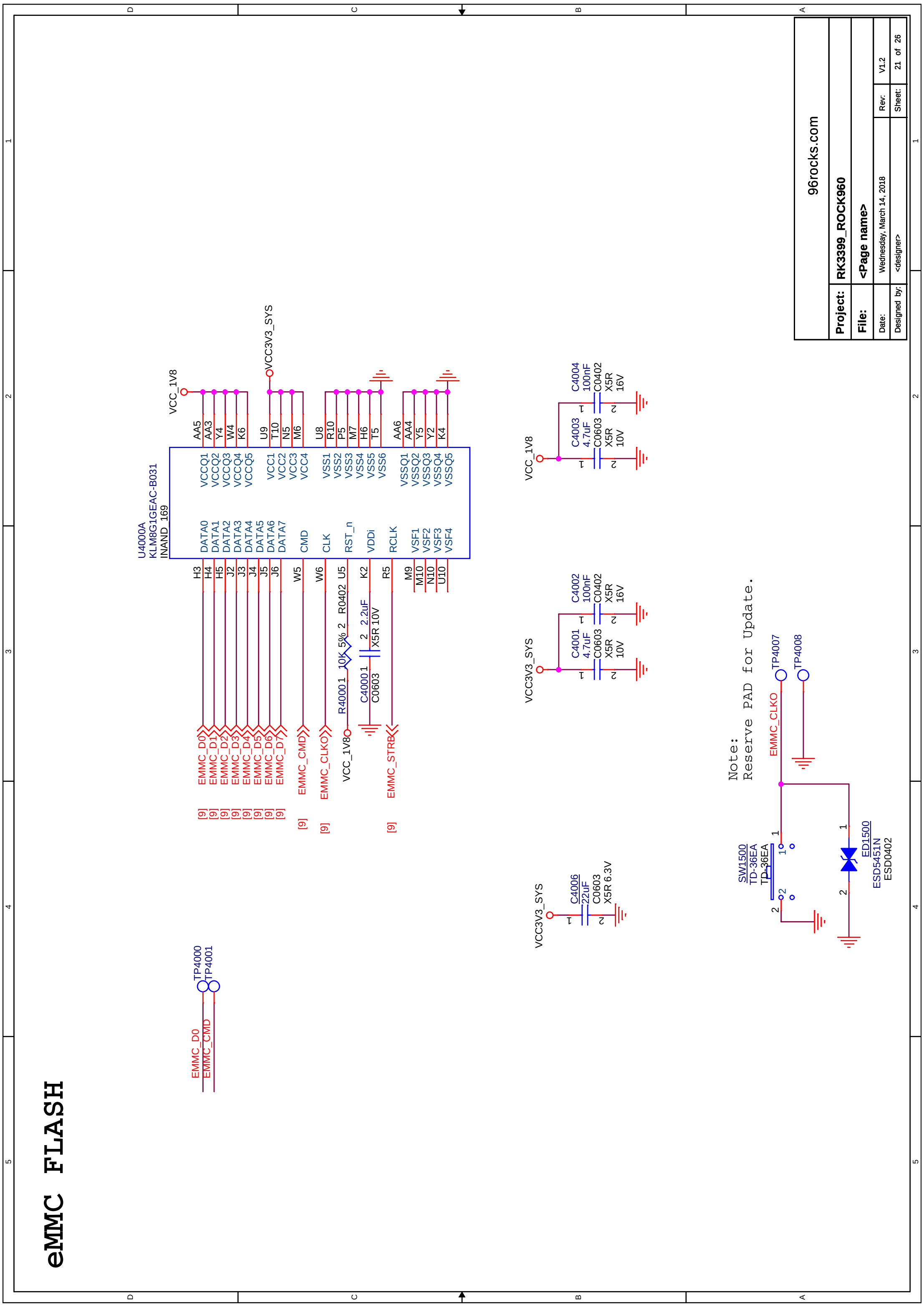


Figure C.19: Schematic - Rock960 eMMC memory.

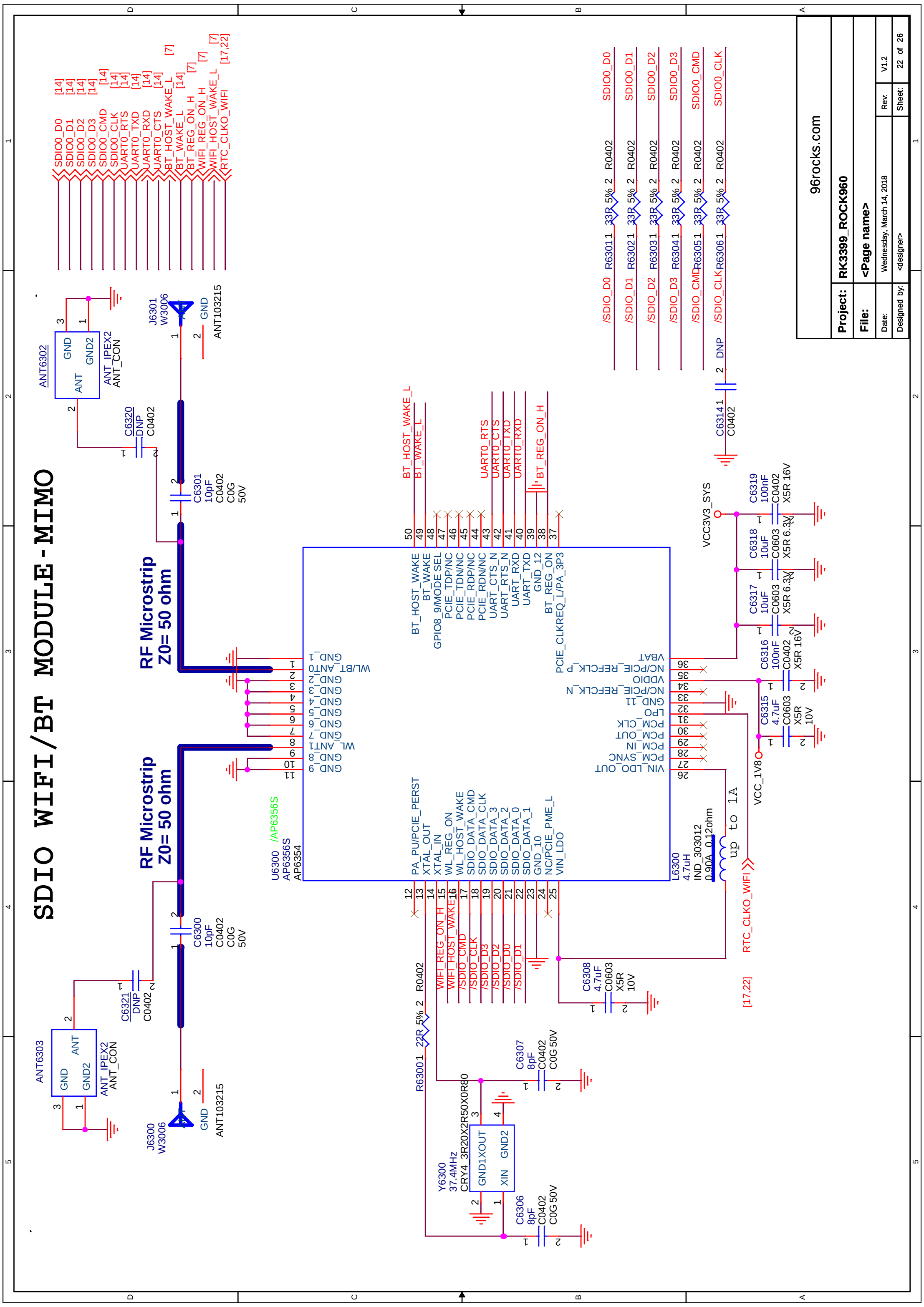


Figure C.20: Schematic - Rock960 WIFI/B.

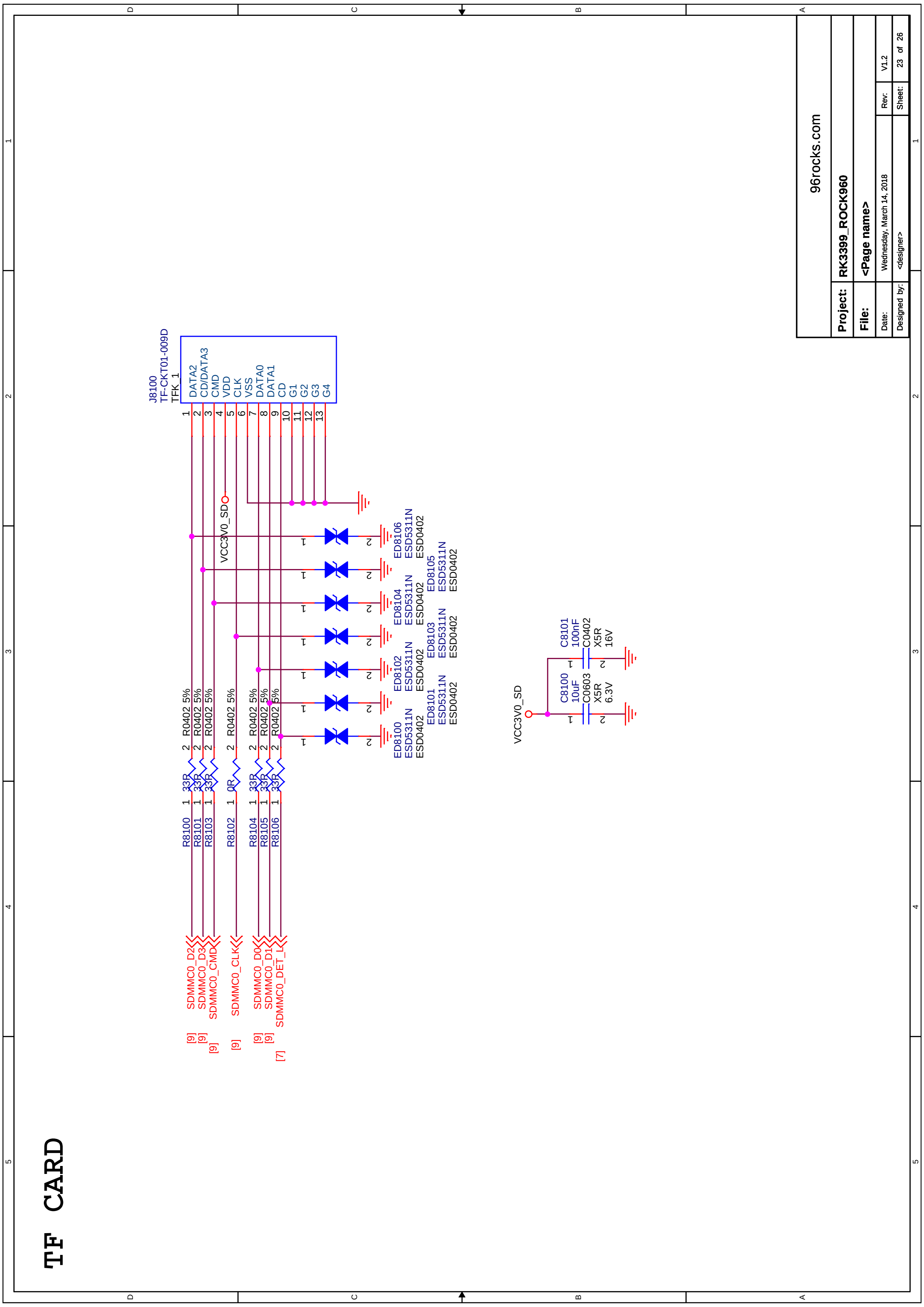


Figure C.21: Schematic - Rock960 TF card.

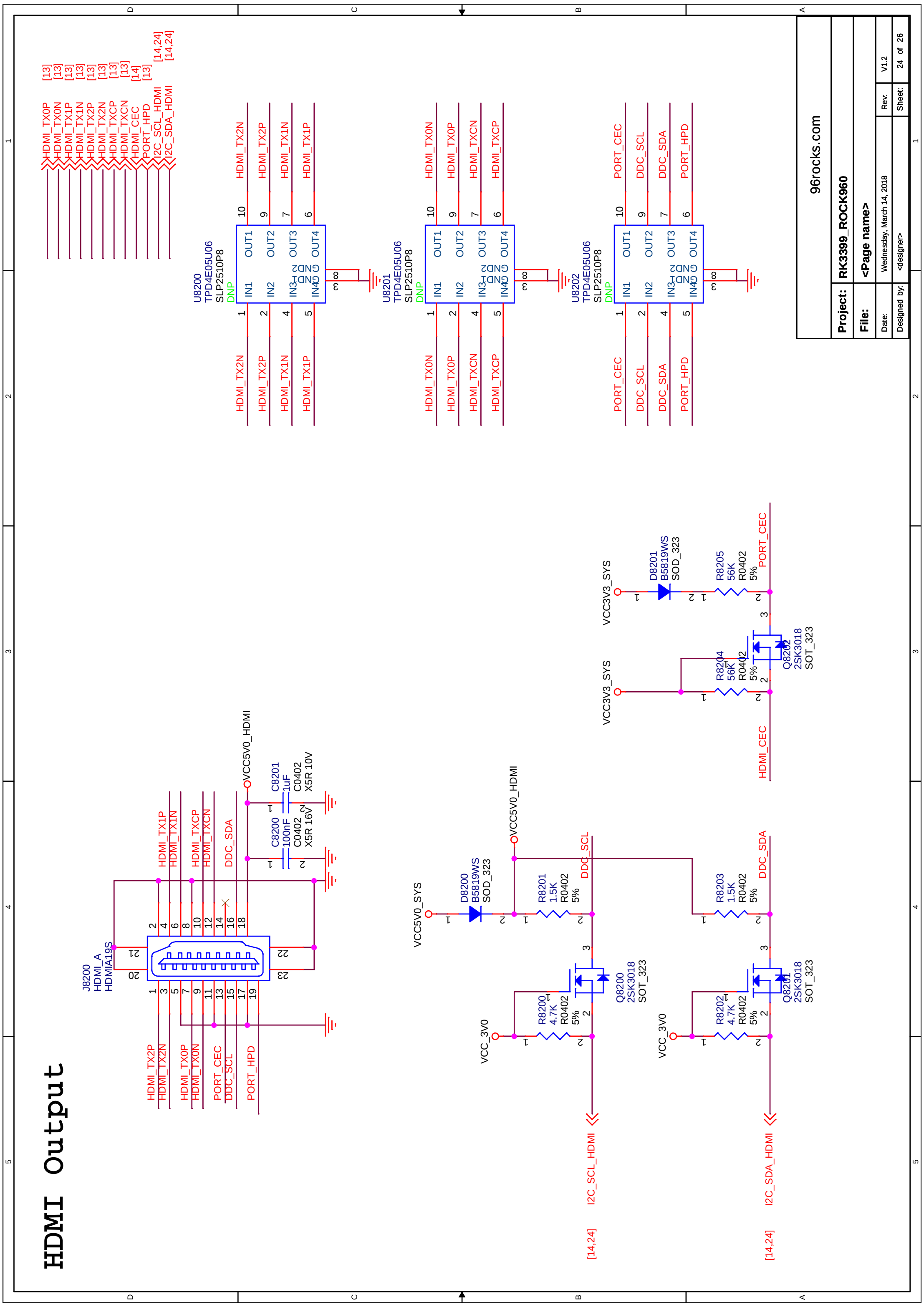


Figure C.22: Schematic - Rock960 HDMI output.

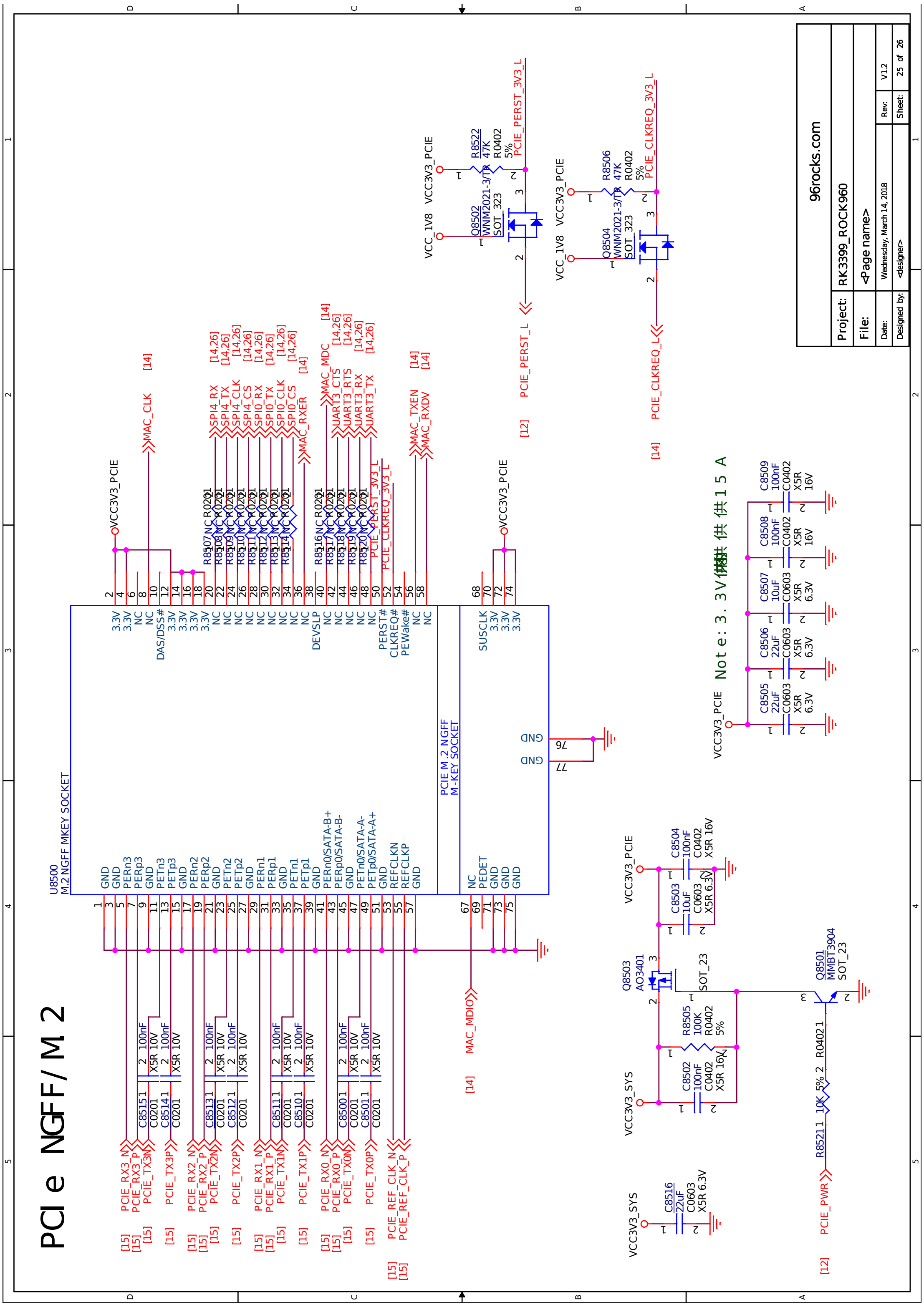


Figure C.23: Schematic - Rock960 PCIe.

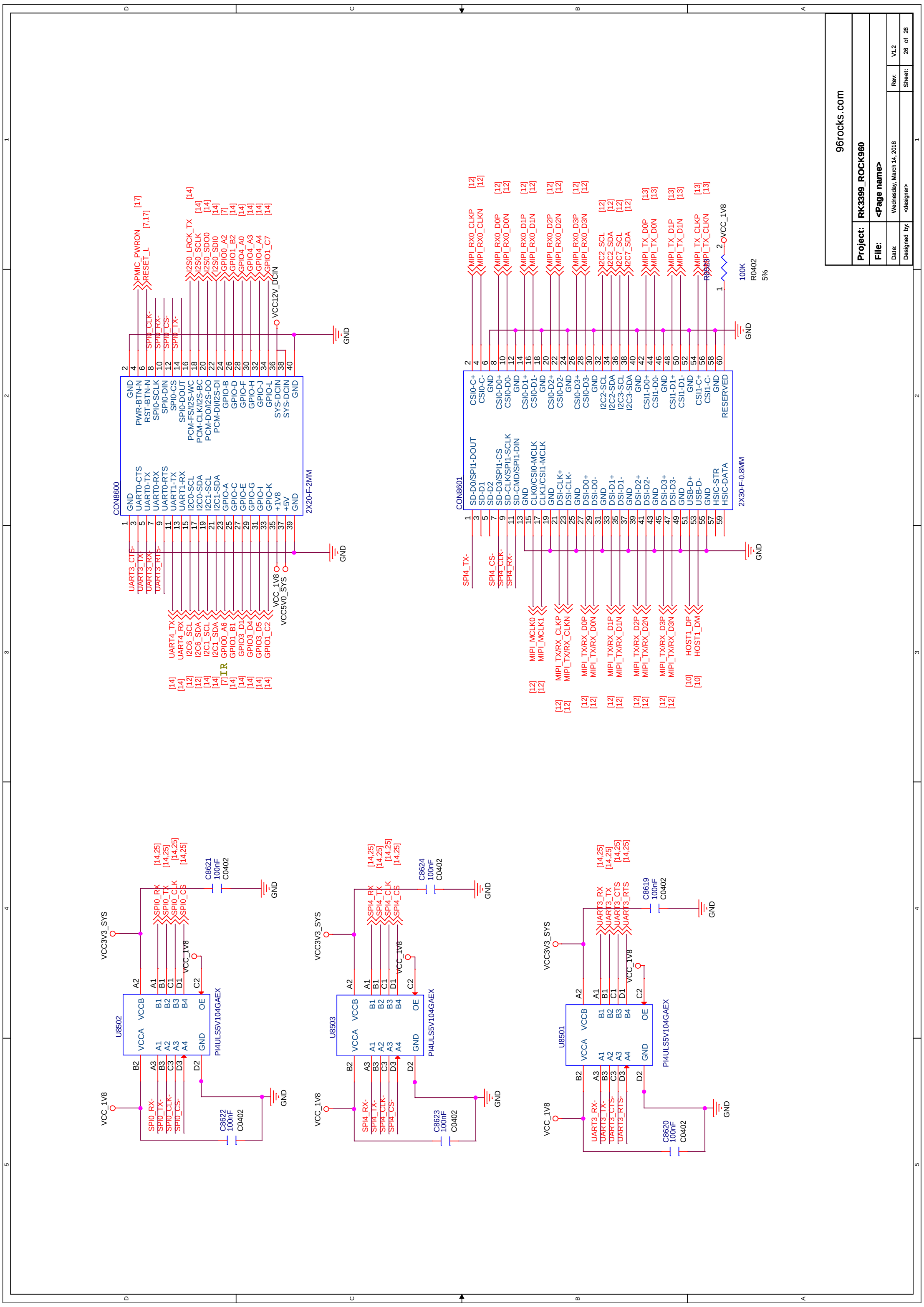


Figure C.24: Schematic - Rock960 connectors.

Universitat Oberta de Catalunya

C.2 LSM9DS0 Board

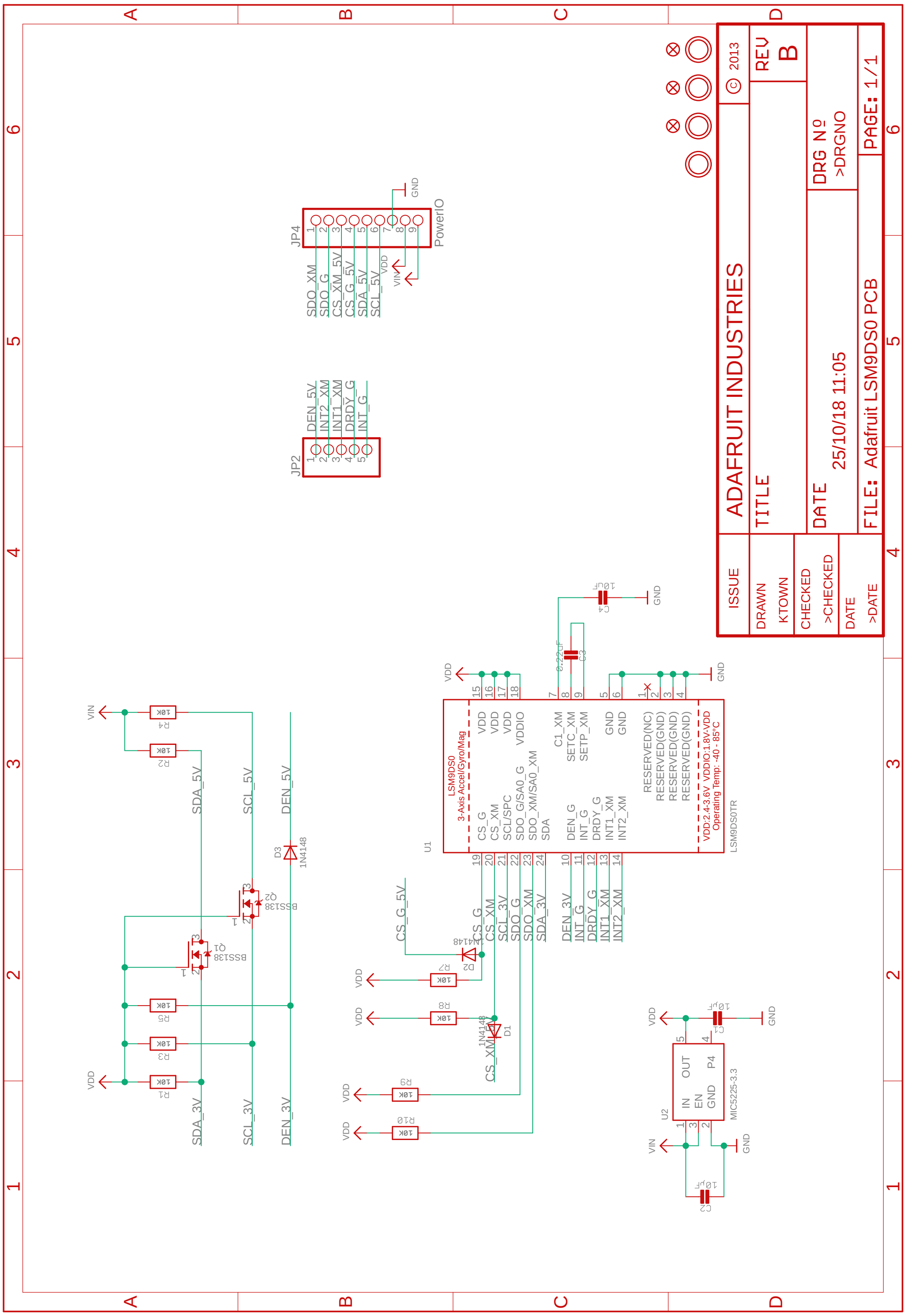


Figure C.25: Schematic - LSM9DS0 board.

C.3 Voltage Level Shifter Schematic

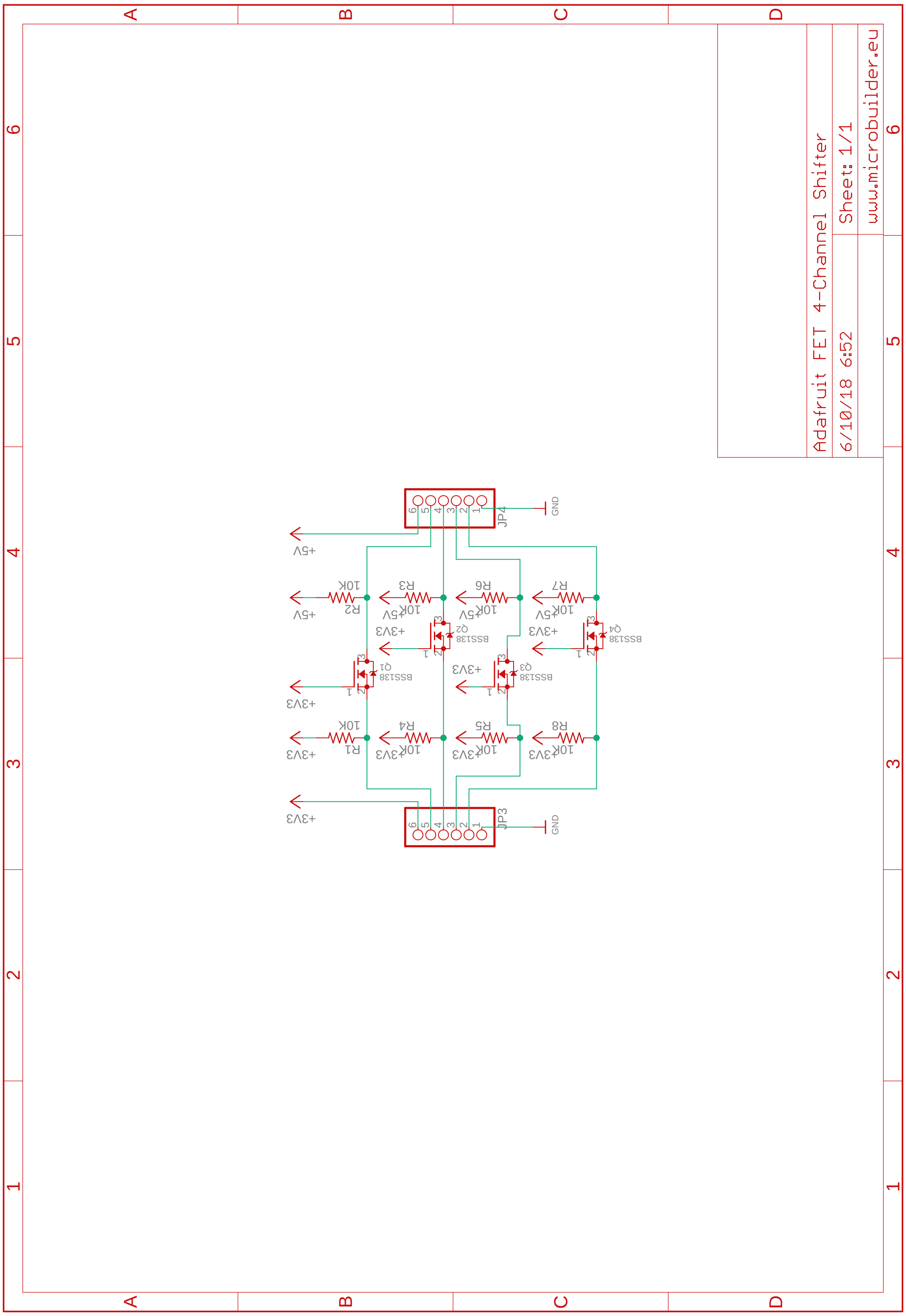


Figure C.26: Schematic - Voltage level shifter.

C.4 96Boards Sensors Mezzanine

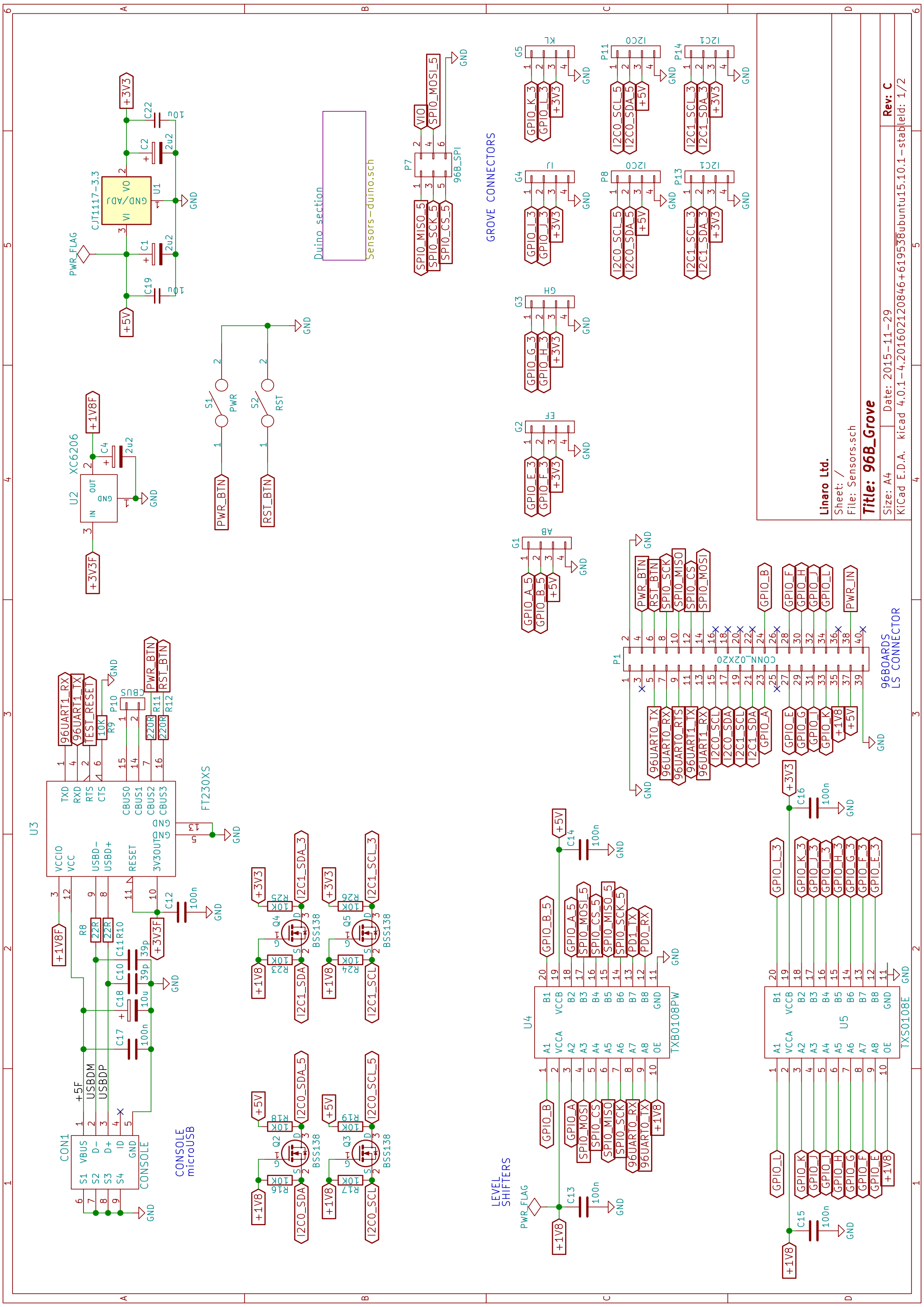


Figure C.27: Schematic - 96Boards sensor mezzanine.

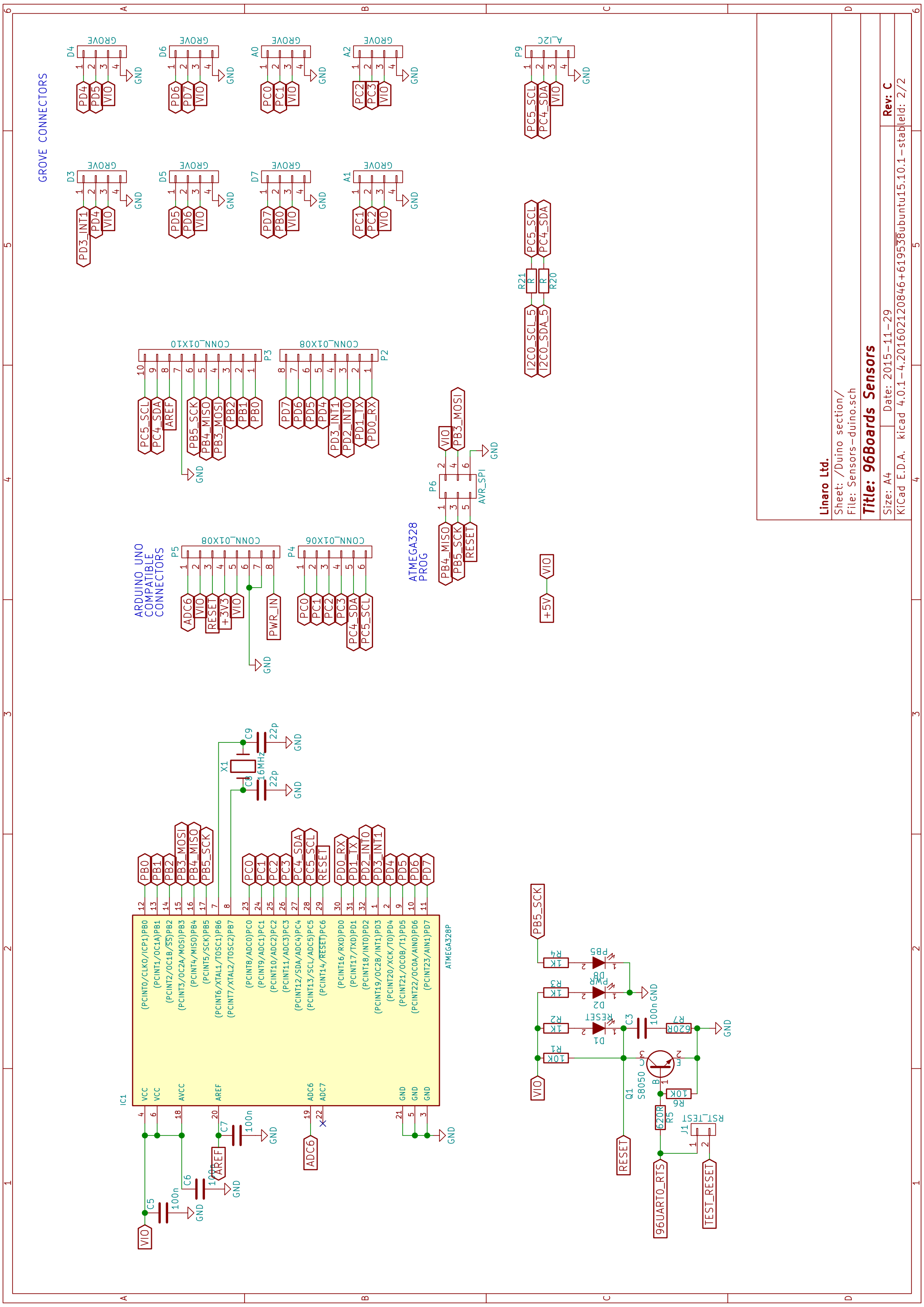


Figure C.28: Schematic - 96Boards sensor mezzanine.

C.5 LSM9DS0 Custom Made Mezzanine

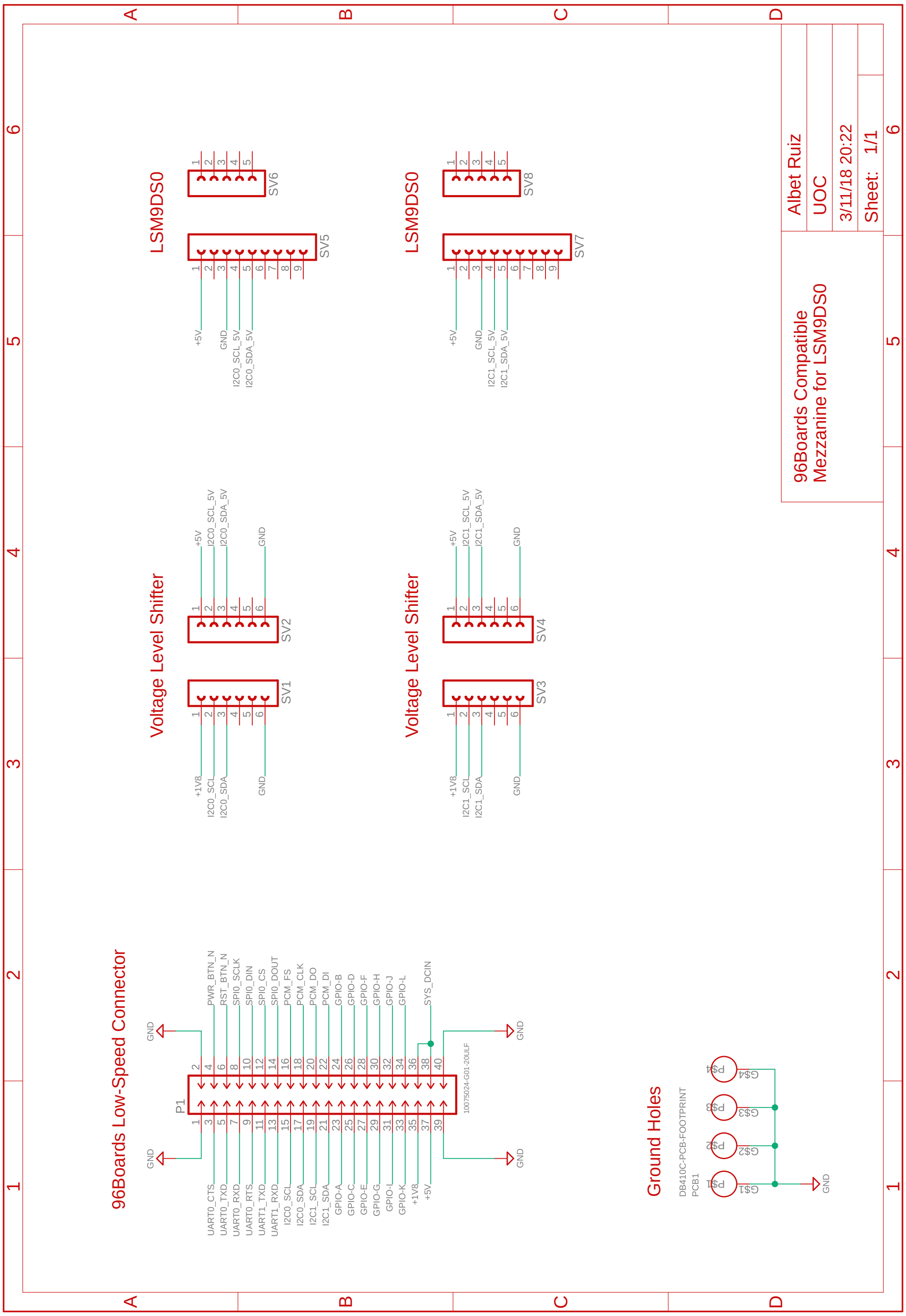
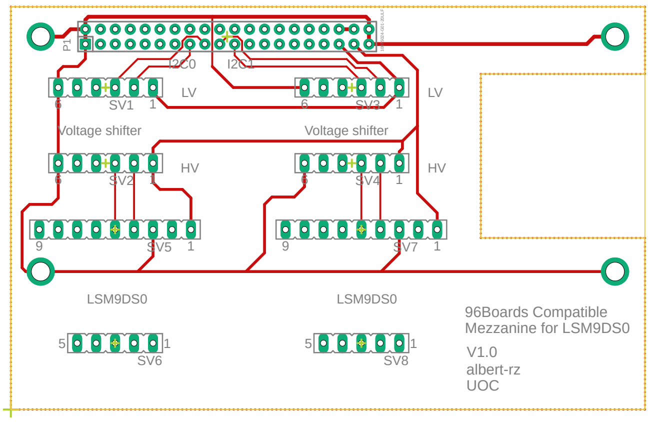
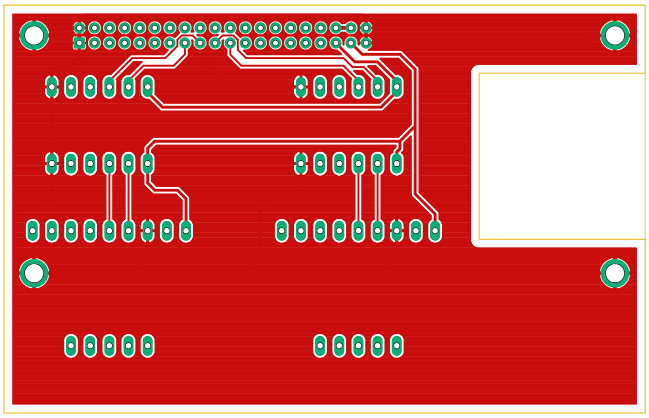


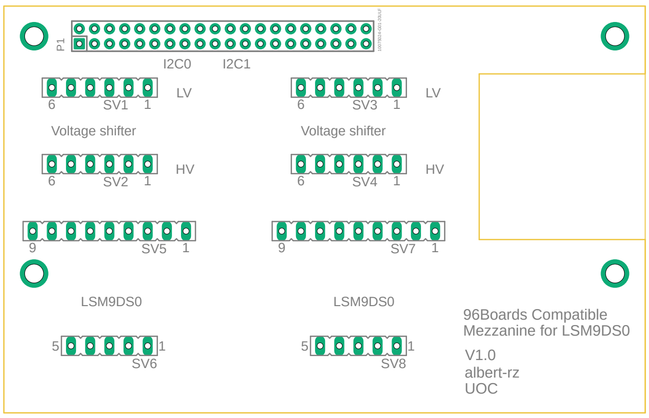
Figure C.29: Schematic - LSM9DS0 Custom made mezzanine.



(a)



(b)



(c)

Figure C.30: Custom made mezzanine layout - (a) Top layer, (b) Top layer with ground plane, (c) Top silkscreen layer.