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

1.1 Project description ......................................................... 11
1.2 Rock960 and LSM9DS0. ..................................................... 11
1.3 Project Gantt chart. ......................................................... 15

2.1 Rock960 main components. ................................................. 18
2.2 Traces showing Rock960 is in maskrom mode. ............................... 21
2.3 Serial cable connection. ................................................... 22
2.4 Initial file system size. ..................................................... 23
2.5 Initial partition list. ....................................................... 23
2.6 Resize /dev/root partition. ................................................ 24
2.7 Final file system size. ...................................................... 24
2.8 Available WLANs using nmcli. .......................................... 25
2.9 SSH connection to Rock960. .............................................. 26
2.10 Ubuntu OS running on Rock960. ......................................... 27
2.11 LSM9DS0 board pinout. .................................................. 29
2.12 Native compilation and cross compilation. ............................... 30
2.13 Hello world example running on Rock960. ................................ 32
2.14 Devices with various supply voltages sharing the same bus. .............. 32
2.15 Voltage shifting for I2C buses. ......................................... 33
2.16 I2C voltage level shifters used in 96Boards sensor mezzanine. ............ 33
2.17 I2C voltage level shifter. ............................................... 34
2.18 I2C0 and I2C1 interfaces connections from low speed connector to RK3399. 35
2.19 Wiring scheme. ........................................................... 35
2.20 LSM9DS0 to Rock960 wiring. ........................................... 36
2.21 Mezzanine mounted on Rock960. ....................................... 36
2.22 Detection of the two LSM9DS0 I2C addresses connected to I2C0 port. .... 37
2.23 Reading WHO_AM_I_XM and WHO_AM_I_G with i2cget. ................. 38

3.1 Linux device and driver model. .......................................... 39

4.1 Data transfer in I2C bus. .................................................. 46
4.2 I2C transfer when master reads one single byte. .......................... 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>I²C transfer when master reads multiple bytes.</td>
</tr>
<tr>
<td>4.4</td>
<td>I²C transfer when master writes one single byte.</td>
</tr>
<tr>
<td>4.5</td>
<td>I²C transfer when master writes multiple bytes.</td>
</tr>
<tr>
<td>5.1</td>
<td>Demo application components UML diagram.</td>
</tr>
<tr>
<td>5.2</td>
<td>UML sequence diagram for the power on interaction.</td>
</tr>
<tr>
<td>5.3</td>
<td>UML sequence diagram for the power off interaction.</td>
</tr>
<tr>
<td>5.4</td>
<td>UML sequence diagram for the read acceleration interaction.</td>
</tr>
<tr>
<td>5.5</td>
<td>UML sequence diagram for the tilt sense interaction.</td>
</tr>
<tr>
<td>5.7</td>
<td>Euler angles definition.</td>
</tr>
<tr>
<td>5.8</td>
<td>Successive rotations following different sequences.</td>
</tr>
<tr>
<td>5.9</td>
<td>Coordinate system and rotation angles.</td>
</tr>
<tr>
<td>5.10</td>
<td>Demo application.</td>
</tr>
<tr>
<td>5.11</td>
<td>Application controls.</td>
</tr>
<tr>
<td>5.12</td>
<td>Demo application showing tilt changes.</td>
</tr>
<tr>
<td>A.1</td>
<td>libwebsockets cross compiled.</td>
</tr>
<tr>
<td>A.2</td>
<td>JSON-C cross compiled.</td>
</tr>
<tr>
<td>C.1</td>
<td>Schematic - Rock960 power tree.</td>
</tr>
<tr>
<td>C.2</td>
<td>Schematic - Rock960 I²C map.</td>
</tr>
<tr>
<td>C.3</td>
<td>Schematic - Rock960 power domain main.</td>
</tr>
<tr>
<td>C.4</td>
<td>Schematic - Rock960 RK3399 power.</td>
</tr>
<tr>
<td>C.5</td>
<td>Schematic - Rock960 RK3399 PMU controller.</td>
</tr>
<tr>
<td>C.6</td>
<td>Schematic - Rock960 RK3399 DDR controller.</td>
</tr>
<tr>
<td>C.7</td>
<td>Schematic - Rock960 RK3399 flash and DMMC controller.</td>
</tr>
<tr>
<td>C.8</td>
<td>Schematic - Rock960 RK3399 USB controller.</td>
</tr>
<tr>
<td>C.9</td>
<td>Schematic - Rock960 RK3399 SARADC/Key.</td>
</tr>
<tr>
<td>C.10</td>
<td>Schematic - Rock960 RK3399 DVP interface.</td>
</tr>
<tr>
<td>C.11</td>
<td>Schematic - Rock960 RK3399 display interface.</td>
</tr>
<tr>
<td>C.12</td>
<td>Schematic - Rock960 RK3399 GPIO.</td>
</tr>
<tr>
<td>C.13</td>
<td>Schematic - Rock960 RK3399 PCIE.</td>
</tr>
<tr>
<td>C.14</td>
<td>Schematic - Rock960 DC power in.</td>
</tr>
<tr>
<td>C.15</td>
<td>Schematic - Rock960 PMIC power.</td>
</tr>
<tr>
<td>C.16</td>
<td>Schematic - Rock960 USB OTG/HOST port.</td>
</tr>
<tr>
<td>C.17</td>
<td>Schematic - Rock960 USB Type-C port.</td>
</tr>
<tr>
<td>C.18</td>
<td>Schematic - Rock960 RAM LPDDR3.</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Project task list.............................................. 12
2.1 Rock960 board features........................................ 17
2.2 Low speed connector pinout................................. 19
2.3 Serial port configuration..................................... 22
2.4 LSM9DS0 board pinout........................................ 29
2.5 LSM9DS0 registers for testing............................... 37
4.1 Reserved 7 bit addresses...................................... 46
4.2 Description of symbols used in I²C transfers................. 48
7.1 Bill of materials.................................................. 66
A.1 Gyroscope status and data registers......................... 73
A.2 Gyroscope configuration registers......................... 74
A.3 Accelerometer and magnetometer status and data registers 75
A.4 Accelerometer and magnetometer configuration registers 76
B.1 Power and reset pins description........................... 85
B.2 UART pins description......................................... 85
B.3 I²C pins description........................................... 86
B.4 SPI pins description........................................... 86
B.5 GPIO pins description.......................................... 87
B.6 PMI/I²S pins description..................................... 87
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 I²C tests.

Chapter 3 is dedicated to Linux device and driver model: bus drivers, controller drivers, Device Tree... It also analyses how I²C RK3399 platform driver works and how it is used from user space.

Chapter 4 begins with an introduction to I²C bus: features, speed, addresses... Then the four I²C 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

List of Figures 4

List of Tables 5

Summary 6

1 Preface 10
   1.1 Motivation ................................. 10
   1.2 Project Description .......................... 10
   1.3 Goals and Scope .......................... 11
   1.4 Project Tasks and Schedule ................. 12

2 Project Setup 16
   2.1 Introduction to Linaro and 96Boards ........... 16
   2.2 Getting started with Rock960 .................. 16
   2.3 Setting up Rock960 .......................... 20
      2.3.1 Ubuntu Server 64-bit Installation ........... 20
      2.3.2 Serial Port configuration .................. 22
      2.3.3 Resizing /dev/root File System ............... 22
      2.3.4 WLAN Connection Setup .................... 25
      2.3.5 Adding a New User ........................ 26
      2.3.6 Graphical Desktop Environment Installation ... 27
   2.4 Getting started with LSM9DS0 ................. 28
   2.5 Setting Up the Development Environment ......... 30
      2.5.1 Installing Cross Compiler for ARM Platforms ... 30
      2.5.2 Compiling the first Hello world .............. 31
   2.6 Wiring Rock960 and LSM9DS0 ................... 32
      2.6.1 Connecting I^2C Devices with Different Voltage Levels ... 32
      2.6.2 Voltage Level Shifter .................... 33
      2.6.3 Wiring scheme .......................... 34
      2.6.4 LSM9DS0 Custom Made Mezzanine ............. 34
   2.7 Testing Rock960 to LSM9DS0 I^2C Interface ...... 37
2.7.1 Linux i2c-tools

2.7.2 Reading WHO_AM_I_XM and WHO_AM_I_G registers

3 Introduction to Linux Drivers

3.1 Introduction to Linux Device and Driver Model

3.1.1 Bus Drivers

3.1.2 Bus Controller Drivers

3.2 Introduction to the Device Tree

3.3 RK3399 I2C Platform Driver

3.4 Using RK3399 I2C Platform Driver from User Space

4 LSM9DS0 Driver Development

4.1 Introduction to I2C Bus

4.1.1 Basic Features

4.1.2 Data Transfer

4.1.3 Slave Addresses

4.2 LSM9DS0 I2C Operation

4.3 LSM9DS0 Driver

4.3.1 User Space Device Drivers

4.3.2 Driver Description

5 Demo Application

5.1 Description

5.1.1 UML Components Diagram

5.1.2 UML Sequence Diagram

5.2 Introduction Euler Angles and Tait-Bryan Angles

5.3 Pitch and Roll Estimation

5.4 Running Demo Application

6 Conclusions

7 Bill of Materials

Bibliography
Appendix A  Code and Snippets

A.1 Hello world! - A Basic Example to Test Cross Compilation .......................... 68
   A.1.1 Source Code .................................................................................. 68
   A.1.2 Makefile ......................................................................................... 68
A.2 Cross Compile libwebsockets ................................................................. 69
   A.2.1 Getting the sources ..................................................................... 69
   A.2.2 Cross Compilation for Rock960 ................................................. 69
   A.2.3 Compilation for x86_64 ............................................................... 70
A.3 Cross Compile JSON-C ....................................................................... 71
   A.3.1 Getting the sources ..................................................................... 71
   A.3.2 Cross Compilation for Rock960 ................................................. 71
   A.3.3 Compilation for x86_64 ............................................................... 72
A.4 LSM9DS0 Driver ................................................................................. 73
   A.4.1 LSM9DS0 Register Description ............................................... 73
   A.4.2 Source Code .................................................................................. 77
   A.4.3 Makefile ......................................................................................... 82
   A.4.4 Upload Script .............................................................................. 83

Appendix B  40 Pin Low Speed Connector .................................................. 85

B.1 Power and Reset .................................................................................... 85
B.2 UART ..................................................................................................... 85
B.3 I²C ......................................................................................................... 86
B.4 SPI ......................................................................................................... 86
B.5 GPIO ...................................................................................................... 87
B.6 PCM/I²S ................................................................................................ 87

Appendix C  Schematics .............................................................................. 88

C.1 Rock960 Board ...................................................................................... 88
C.2 LSM9DS0 Board .................................................................................. 113
C.3 Voltage Level Shifter Schematic .......................................................... 114
C.4 96Boards Sensors Mezzanine .............................................................. 115
C.5 LSM9DS0 Custom Made Mezzanine .................................................... 117
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 (I²C) 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 System 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

**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. I²C 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 \textsc{i2c} 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.

<table>
<thead>
<tr>
<th>A - Project setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 Target setup</td>
</tr>
<tr>
<td>• Boot Rock960 from SD card</td>
</tr>
<tr>
<td>• Flash OS to Rock960’s eMMC memory and boot</td>
</tr>
<tr>
<td>A2 Host setup and first binary</td>
</tr>
<tr>
<td>• Prepare cross compiler toolchain</td>
</tr>
<tr>
<td>• Program a basic \texttt{Hello world} and run it on Rock960</td>
</tr>
<tr>
<td>A3 Testing \textsc{i2c} interface</td>
</tr>
<tr>
<td>• Rock960 to LSM9DS0 wiring</td>
</tr>
<tr>
<td>• Test Rock960 to LSM9DS0 \textsc{i2c} interface</td>
</tr>
<tr>
<td>• Create a custom made mezzanine for LSM9DS0</td>
</tr>
</tbody>
</table>
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 I²C subsystem
- Introduction to I²C bus
- RK3399 I²C 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
- I²C 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 engines

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
## A - Project setup
- A1 - Target setup
- A2 - Host setup
- A3 - Testing I²C interface
- A4 - First OpenGL ES 2.0 binary

## B - Linux drivers for Rock960
- B1 - Introduction
- B2 - Module test
- B3 - I²C subsystem

## C - LSM9DS0 driver development
- C1 - Introduction
- C2 - Driver development
- C3 - Testing

## D - Final application
- D1 - Graphical engines evaluation
- D2 - Final application development
- D3 - Testing

## E - Documentation
- E1 - Project report
- E2 - Video presentation
- E3 - Virtual defense

### Figure 1.3: Project Gantt chart.
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

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC</td>
<td>Rockchip RK3399</td>
</tr>
<tr>
<td>CPU</td>
<td>ARM Cortex-A72 Dual-core (up to 1.8GHz) + ARM Cortex A53 Quad-core (up to 1.4GHz)</td>
</tr>
<tr>
<td>GPU</td>
<td>ARM Mali T860MP4</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB LPDDR3</td>
</tr>
<tr>
<td>Storage</td>
<td>32GB eMMC 5.1</td>
</tr>
<tr>
<td>WiFi</td>
<td>WLAN 802.11 ac/a/b/g/n, 2.4GHz and 5GHz, on board antenna</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>4.2, on board antenna</td>
</tr>
<tr>
<td>USB</td>
<td>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)</td>
</tr>
<tr>
<td>Display</td>
<td>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)</td>
</tr>
<tr>
<td>Audio</td>
<td>HDMI output</td>
</tr>
<tr>
<td>Expansion</td>
<td>40 pin low speed expansion connector (+1.8V, +5V, DC power, GND, 2 x UART, 2 x I2C, SPI, 2 x I2S, 12 x GPIO) and 60 pin high speed expansion connector (4L-MIPI DSI, 12I2C x 2, SPI (48M), USB 2.0, 2L+4LMII CSI)</td>
</tr>
<tr>
<td>Button</td>
<td>Reset button and recovery button</td>
</tr>
<tr>
<td>Debug</td>
<td>Rock960 exports a dedicated TTL serial console</td>
</tr>
<tr>
<td>Power source</td>
<td>+12V and 2A, DC, 4.0 mm Jack connector</td>
</tr>
<tr>
<td>OS support</td>
<td>AOSP, Debian, Ubuntu</td>
</tr>
<tr>
<td>Size</td>
<td>85mm x 55mm</td>
</tr>
</tbody>
</table>
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 `rkdevelopmenttool` 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Pin number</th>
<th>Pin number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>1</td>
<td>2</td>
<td>GND</td>
</tr>
<tr>
<td>UART0_CTS</td>
<td>3</td>
<td>4</td>
<td>PWR_BTN_N</td>
</tr>
<tr>
<td>UART0_TxD</td>
<td>5</td>
<td>6</td>
<td>RST_BTN_N</td>
</tr>
<tr>
<td>UART0_RxD</td>
<td>7</td>
<td>8</td>
<td>SPI0_SCLK</td>
</tr>
<tr>
<td>UART0_RTS</td>
<td>9</td>
<td>10</td>
<td>SPI0_DIN</td>
</tr>
<tr>
<td>UART1_TxD</td>
<td>11</td>
<td>12</td>
<td>SPI0_CS</td>
</tr>
<tr>
<td>UART1_RxD</td>
<td>13</td>
<td>14</td>
<td>SPI0_DOUT</td>
</tr>
<tr>
<td>I2C0_SCL</td>
<td>15</td>
<td>16</td>
<td>PCM_FS</td>
</tr>
<tr>
<td>I2C0_SDA</td>
<td>17</td>
<td>18</td>
<td>PCM_CLK</td>
</tr>
<tr>
<td>I2C1_SCL</td>
<td>19</td>
<td>20</td>
<td>PCM_DO</td>
</tr>
<tr>
<td>I2C1_SDA</td>
<td>21</td>
<td>22</td>
<td>PCM_DI</td>
</tr>
<tr>
<td>GPIO-A</td>
<td>23</td>
<td>24</td>
<td>GPIO-B</td>
</tr>
<tr>
<td>GPIO-C</td>
<td>25</td>
<td>26</td>
<td>GPIO-D</td>
</tr>
<tr>
<td>GPIO-E</td>
<td>27</td>
<td>28</td>
<td>GPIO-F</td>
</tr>
<tr>
<td>GPIO-G</td>
<td>29</td>
<td>30</td>
<td>GPIO-H</td>
</tr>
<tr>
<td>GPIO-I</td>
<td>31</td>
<td>32</td>
<td>GPIO-J</td>
</tr>
<tr>
<td>GPIO-K</td>
<td>33</td>
<td>34</td>
<td>GPIO-L</td>
</tr>
<tr>
<td>+1.8V</td>
<td>35</td>
<td>36</td>
<td>SYS_DCIN</td>
</tr>
<tr>
<td>+5V</td>
<td>37</td>
<td>38</td>
<td>SYS_DCIN</td>
</tr>
<tr>
<td>GND</td>
<td>39</td>
<td>40</td>
<td>GND</td>
</tr>
</tbody>
</table>
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 extension .bin).
- Image with bootloader, 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.](image)

**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 w1 0 rock960_model_abUbuntu_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.

![Serial port connection diagram](image)

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

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

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

Serial port was configured with:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit rate</td>
<td>1500000 bits/s</td>
</tr>
<tr>
<td>Configuration in asynchronous mode</td>
<td>8N1 (8 data bits, no parity bit, 1 stop bit)</td>
</tr>
<tr>
<td>Hardware flow control</td>
<td>No</td>
</tr>
<tr>
<td>Software flow control</td>
<td>No</td>
</tr>
</tbody>
</table>

**Table 2.3:** Serial port configuration.

2.3.3 Resizing /dev/root File System

After OS installation, it was checked the free memory with (see figure 2.4):
Existing partitions were also checked with (see figure 2.5):

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

![Figure 2.4: Initial file system size.](image)

![Figure 2.5: Initial partition list.](image)

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):

```bash
parted /dev/mmcblk1
(parted) print
(parted) resizepart 5
End? -34s
quit
```

Then the file system was extended:

```bash
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
```

![Available WLANs using nmcli](image)

**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).

![SSH connection to Rock960.](image)

**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:

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

![Ubuntu OS running on Rock960.](image)

**Figure 2.10:** Ubuntu OS running on Rock960.

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

```bash
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: $\pm 245\, \text{deg/s}$, $\pm 500\, \text{deg/s}$ and $\pm 2000\, \text{deg/s}$.
- 16-bit data output.
- I²C 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.
- I²C serial interface (+3V to +5V logic).
- SPI serial interface (+3V to +5V logic).
- 10kΩ pull-up resistors for both I²C 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.

<table>
<thead>
<tr>
<th>Connection</th>
<th>Description</th>
<th>Voltage logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIN</td>
<td>Power input, from +3V to +5V</td>
<td>-</td>
</tr>
<tr>
<td>3V3</td>
<td>+3.3V output</td>
<td>-</td>
</tr>
<tr>
<td>GND</td>
<td>Common ground</td>
<td>-</td>
</tr>
<tr>
<td>SCL</td>
<td>I²C clock pin or SPI clock pin</td>
<td>+3V to +5V</td>
</tr>
<tr>
<td>SDA</td>
<td>I²C data pin or SPI MOSI pin</td>
<td>+3V to +5V</td>
</tr>
<tr>
<td>CSG</td>
<td>SPI chip select for gyroscope</td>
<td>+3V to +5V</td>
</tr>
<tr>
<td>CSXM</td>
<td>SPI chip select for accelerometer and magnetometer</td>
<td>+3V to +5V</td>
</tr>
<tr>
<td>SDOG</td>
<td>SPI MISO pin for gyroscope</td>
<td>+3V to +5V</td>
</tr>
<tr>
<td>SDOXM</td>
<td>SPI MISO pin for accelerometer and magnetometer</td>
<td>+3V to +5V</td>
</tr>
<tr>
<td>DEN</td>
<td>Gyroscope data enable</td>
<td>+5V</td>
</tr>
<tr>
<td>INT1</td>
<td>Accelerometer and magnetic sensor interrupt 1</td>
<td>+3V</td>
</tr>
<tr>
<td>INT2</td>
<td>Accelerometer and magnetic sensor interrupt 2</td>
<td>+3V</td>
</tr>
<tr>
<td>DRDY</td>
<td>Gyroscope data ready</td>
<td>+3V</td>
</tr>
<tr>
<td>INTG</td>
<td>Gyroscope programmable interrupt</td>
<td>+3V</td>
</tr>
</tbody>
</table>
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).

![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).
2.6 Wiring Rock960 and LSM9DS0

2.6.1 Connecting I²C Devices with Different Voltage Levels

I²C 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.

\[ V_{DD1} \text{ is bus voltage} \]
\[ V_{DD2} \text{ and } V_{DD3} \text{ are device-dependent (for example 12V)} \]

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

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 I²C standard, and depend on the bus voltage \( V_{DD} \) (\( V_{DD1} \) in figure 2.14):

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

Different voltage devices may be connected to the same I²C 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 I²C used +3V to +5V logic. Thus, a voltage level shifter was necessary.
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^2C bus voltage from +1.8V to +5V (I^2C0) and +3V (I^2C1) (see figure 2.16a).

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

![Diagram of I^2C voltage level shifter](image)

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

Appendix C.4 includes sensor mezzanine schematics. Appendix C.3 includes Adafruit board schematics.
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
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 I²C Interface

2.7.1 Linux i2c-tools

I²C interface was tested using Linux i2c-tools package. This package has some programs to interact with I²C 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).

![i2cdetect output showing two slave addresses](image)

Figure 2.22: Detection of the two LSM9DS0 I²C addresses connected to I2C0 port.

As it will be seen in 4.1, LSM9DS0 has two I²C slaves internally:

- All registers related to accelerometer and magnetometer are accessed via 0x1D I²C address
- All registers related to gyroscope are accessed via 0x6B I²C 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 I²C 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.

<table>
<thead>
<tr>
<th>Register</th>
<th>Register address</th>
<th>Slave address</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHO_AM_I_XM</td>
<td>0x0F</td>
<td>0x1D</td>
<td>01001001b (0x49)</td>
</tr>
<tr>
<td>WHO_AM_I_G</td>
<td>0x0F</td>
<td>0x6B</td>
<td>11010100b (0xD4)</td>
</tr>
</tbody>
</table>
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]).](image-url)
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 I²C 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
#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 = <0x08 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 0x25 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 resources 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 \(^2\)C Platform Driver

Among the non-discoverable devices, a huge group are the devices that are directly part of RK3399: UART controllers, Ethernet controllers, \(^2\)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 \(^2\)C platform driver for RK3399. It includes register definitions:

```c
#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:

```c
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,
};

I2C 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,
    },
};
And, of course, its own probe and remove functions:

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

### 3.4 Using RK3399 I²C Platform Driver from User Space

RK3399 I²C 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:

```c
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:

```c
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 I²C Bus

4.1.1 Basic Features

I²C bus is a synchronous, half-duplex, multi-master/multi-slave, 8-bit oriented serial computer bus. Invented by Philips Semiconductor (now NXP Semiconductors), I²C has become a de facto world standard in over 1000 different integrated circuits manufactured by more than 50 companies [10]. Some features of I²C 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.
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/W bit: 0 indicates a write transmission, 1 indicates a read transmission. Some 7-bit addresses are reserved (see table 4.1).

<table>
<thead>
<tr>
<th>Address</th>
<th>R/W</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 000</td>
<td>0</td>
<td>General call address</td>
<td>Used for several functions including software reset</td>
</tr>
<tr>
<td>0000 000</td>
<td>1</td>
<td>START byte</td>
<td>No device is acknowledge</td>
</tr>
<tr>
<td>0000 001</td>
<td>X</td>
<td>CBUS address</td>
<td>To enable the inter-mixing of CBUS and I²C devices in the same system</td>
</tr>
<tr>
<td>0000 010</td>
<td>X</td>
<td>Reserved for different bus formats</td>
<td>To enable mixing I²C with other protocols</td>
</tr>
<tr>
<td>0000 011</td>
<td>X</td>
<td>Reserved for future purposes</td>
<td>-</td>
</tr>
<tr>
<td>0000 1XX</td>
<td>X</td>
<td>High speed mode</td>
<td>-</td>
</tr>
<tr>
<td>1111 0XX</td>
<td>X</td>
<td>10 bit addressing</td>
<td>-</td>
</tr>
<tr>
<td>1111 1XX</td>
<td>X</td>
<td>To retrieve device ID</td>
<td>Device ID is an optional 3 byte read-only word with:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Manufacturer name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Part identification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revision</td>
</tr>
</tbody>
</table>

I²C also supports 10 bit addresses, but this feature will not be covered in this document.
4.2 LSM9DS0 I²C Operation

LSM9DS0 has two I²C devices:

- One for accelerometer and magnetometer registers (0x1D I²C address).
- One for gyroscope registers (0x6B I²C address).

Both devices behave like I²C 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).

**Figure 4.2:** I²C transfer when master reads one single byte.

**Figure 4.3:** I²C transfer when master reads multiple bytes.

**Figure 4.4:** I²C transfer when master writes one single byte.

**Figure 4.5:** I²C transfer when master writes multiple bytes.
### Table 4.2: Description of symbols used in I²C transfers.

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

```c
/* Accelerometer and magnetometer 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:

```c
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:

```c
/* Read register */
struct r_reg
{
    /* Register address */
    /* Initiated by the driver */
    unsigned char addr;
    /* Register value */
    unsigned char val;
};
/* Read/Write register */
struct rw_reg
{
    /* Register address */
    /* Initiated by the driver */
    unsigned char addr;
    /* Default value */
    char def_val;
    /* Register value */
```
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).

![UML Diagram](image.png)

**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).

![UML sequence diagram for the power on interaction.](image)
Figure 5.3: UML sequence diagram for the power off interaction.

Figure 5.4: UML sequence diagram for the read acceleration 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.

![Diagram](image)

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:

$$n = \frac{T_{wa}}{T_r} = \frac{250}{16.67} = 15$$  \hspace{1cm} (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 $\alpha$, $\beta$ and $\gamma$: 

![Diagram](image)
• $\alpha$ is the angle between $X$ and $N$ ($N$ is the intersection of planes $XY$ and $X'Y'$).

• $\beta$ is the angle between $Z$ and $Z'$.

• $\gamma$ 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:

• $\alpha$ comes after a rotation around $Z'$.

• $\beta$ comes after a rotation around $X'$.

• $\gamma$ 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 90° 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:

$$
R = R_{xyz} = R_xR_yR_z \tag{5.4}
$$

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:
\[
R_{xyz} = \begin{pmatrix}
\cos\psi \cos\theta & \sin\psi \cos\theta & -\sin\theta \\
\cos\psi \sin\theta \sin\phi - \sin\psi \cos\phi & \sin\psi \sin\theta \sin\phi + \cos\psi \cos\phi & \cos\theta \sin\phi \\
\cos\psi \sin\theta \cos\phi + \sin\psi \sin\phi & \sin\psi \sin\theta \cos\phi - \cos\psi \sin\phi & \cos\theta \cos\phi
\end{pmatrix}
\] (5.8)

Combining equations 5.2 and 5.8 results:

\[
\vec{g}_p = \begin{pmatrix}
g_{px} \\
g_{py} \\
g_{pz}
\end{pmatrix} = \begin{pmatrix}
-g_{py} \\
g_{px} \\
c_{\theta} \sin\phi
\end{pmatrix}
\] (5.9)

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

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

\[
\begin{pmatrix}
\hat{g}_{px} \\
\hat{g}_{py} \\
\hat{g}_{pz}
\end{pmatrix} = \begin{pmatrix}
-g_{py} \\
g_{px} \\
c_{\theta} \sin\phi
\end{pmatrix}
\] (5.11)

Finally, solving this equation gives:

\[
\tan \phi_{xyz} = \frac{\hat{g}_{py}}{\hat{g}_{pz}}
\] (5.12)

\[
\tan \theta_{xyz} = \frac{-\hat{g}_{px}}{\sqrt{\hat{g}_{py}^2 + \hat{g}_{pz}^2}}
\] (5.13)

\[
\phi \in (-180^\circ, 180^\circ]
\]

\[
\theta \in (-90^\circ, 90^\circ]
\] (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.

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 I²C 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 I²C platform driver for RK3399 was studied in detail.
- A user space driver for LSM9DS0 and a demo application were developed.

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 I²C 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.

<table>
<thead>
<tr>
<th>#</th>
<th>Reference</th>
<th>Manufacturer</th>
<th>Description</th>
<th>Price (€/u)</th>
<th>Units</th>
<th>Total (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ROCK960-MODEL-B</td>
<td>Vamrs</td>
<td>Development board based on RK3399 SoC</td>
<td>122.12</td>
<td>1</td>
<td>122.12</td>
</tr>
<tr>
<td>2</td>
<td>954</td>
<td>Adafruit</td>
<td>USB to TTL cable</td>
<td>8.74</td>
<td>1</td>
<td>8.74</td>
</tr>
<tr>
<td>3</td>
<td>239</td>
<td>Adafruit</td>
<td>Full size breadboard</td>
<td>5.23</td>
<td>1</td>
<td>5.23</td>
</tr>
<tr>
<td>4</td>
<td>153</td>
<td>Adafruit</td>
<td>Breadboarding wire bundle</td>
<td>4.35</td>
<td>1</td>
<td>4.35</td>
</tr>
<tr>
<td>5</td>
<td>3463</td>
<td>Adafruit</td>
<td>LSM9DS breakout board</td>
<td>21.92</td>
<td>1</td>
<td>21.92</td>
</tr>
<tr>
<td>6</td>
<td>757</td>
<td>Adafruit</td>
<td>4-channel I²C safe bidirectional voltage level shifter - BSS138</td>
<td>3.47</td>
<td>1</td>
<td>3.47</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>Safe-PCB</td>
<td>Printed circuit board for LSM9DS0 custom made mezzanine</td>
<td>71.60</td>
<td>2</td>
<td>143.20</td>
</tr>
<tr>
<td>8</td>
<td>76341-312LF</td>
<td>Amphenol</td>
<td>Board-to-board connector, 2.54mm, 12 contacts</td>
<td>1.05</td>
<td>10</td>
<td>10.50</td>
</tr>
<tr>
<td>9</td>
<td>TMM-120-01-G-D</td>
<td>Samtec</td>
<td>Board-to-board connector, 2mm, 40 contacts, 2 rows</td>
<td>6.14</td>
<td>2</td>
<td>12.28</td>
</tr>
<tr>
<td>10</td>
<td>05.12.053</td>
<td>Ettinger</td>
<td>Hexagonal spacer, male-female, M2.5, 5mm</td>
<td>0.44</td>
<td>5</td>
<td>2.20</td>
</tr>
<tr>
<td>11</td>
<td>971060151</td>
<td>Wurth Elektronik</td>
<td>Hexagonal spacer, male-female, M2.5, 6mm</td>
<td>0.51</td>
<td>5</td>
<td>2.54</td>
</tr>
<tr>
<td>12</td>
<td>971070151</td>
<td>Wurth Elektronik</td>
<td>Hexagonal spacer, male-female, M2.5, 7mm</td>
<td>0.45</td>
<td>5</td>
<td>2.24</td>
</tr>
<tr>
<td>13</td>
<td>TR NWE-34815-M2</td>
<td>TR Fastenings</td>
<td>Nylown washer, M2 (pack of 100)</td>
<td>4.75</td>
<td>1</td>
<td>4.75</td>
</tr>
</tbody>
</table>

**Total** 343.56

Prices include taxes.
Bibliography


Appendix A  Code and Snippets

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

A.1.1  Source Code

```cpp
/*
 * File: helloworld.cpp
 * Author: 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)
```
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 -function-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:

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

### A.3 Cross Compile JSON-C

#### A.3.1 Getting the sources

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

#### A.3.2 Cross Compilation for Rock960

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

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

  ```bash
  # 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++")

  # 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.

<table>
<thead>
<tr>
<th>Register</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHO_AM_I_G</td>
<td>R</td>
<td>Device identification register. It always returns 0b11010100.</td>
</tr>
<tr>
<td>STATUS_REG_G</td>
<td>R</td>
<td>Status register to inform about data overrun and data ready.</td>
</tr>
<tr>
<td>OUT_X_L_G</td>
<td>R</td>
<td>X-axis angular rate data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_X_H_G</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>OUT_Y_L_G</td>
<td>R</td>
<td>Y-axis angular rate data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_Y_H_G</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>OUT_Z_L_G</td>
<td>R</td>
<td>Z-axis angular rate data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_Z_H_G</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>
Table A.2: Gyroscope configuration registers.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHO_AM1_XM</td>
<td>R</td>
<td>Device identification register. It always returns 0b01001001.</td>
</tr>
<tr>
<td>STATUS_REG_M</td>
<td>R</td>
<td>Status register to inform about magnetic data overrun and data ready.</td>
</tr>
<tr>
<td>STATUS_REG_A</td>
<td>R</td>
<td>Status register to inform about acceleration data overrun and data ready.</td>
</tr>
<tr>
<td>OUT_X_L_A</td>
<td>R</td>
<td>X-axis acceleration data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_X_H_A</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>OUT_Y_L_A</td>
<td>R</td>
<td>Y-axis acceleration data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_Y_H_A</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>OUT_Z_L_A</td>
<td>R</td>
<td>Z-axis acceleration data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_Z_H_A</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>OUT_X_L_M</td>
<td>R</td>
<td>X-axis magnetic data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_X_H_M</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>OUT_Y_L_M</td>
<td>R</td>
<td>Y-axis magnetic data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_Y_H_M</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>OUT_Z_L_M</td>
<td>R</td>
<td>Z-axis magnetic data, expressed as two’s complement.</td>
</tr>
<tr>
<td>OUT_Z_H_M</td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>
Table A.4: Accelerometer and magnetometer configuration registers.

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

```c
// ********************************************************************************
// 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)
#endif
```
```c
#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);

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 ARCHAARCH=aarch64
# ********************************************************************************

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

CC := gcc
AR := ar
ifeq ($(ARCH),aarch64)
CC := aarch64-linux-gnu-$(CC)
A.4.4 Upload Script

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

```bash
# 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=websocket_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"i

# 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.

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

B.2 UART

Table B.2: UART pins description.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
<th>Voltage</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>UART0_CTS</td>
<td>Clear to send control</td>
<td>+1.8V</td>
<td>Input</td>
</tr>
<tr>
<td>5</td>
<td>UART0_TxD</td>
<td>Transmit serial data</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>7</td>
<td>UART0_RxD</td>
<td>Receive serial data</td>
<td>+1.8V</td>
<td>Input</td>
</tr>
<tr>
<td>9</td>
<td>UART0_RTS</td>
<td>Request to send control</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>11</td>
<td>UART1_TxD</td>
<td>Transmit serial data</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>13</td>
<td>UART1_RxD</td>
<td>Receive serial data</td>
<td>+1.8V</td>
<td>Input</td>
</tr>
</tbody>
</table>
B.3 \textit{I}^2\textit{C}

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

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
<th>Voltage</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>I2C0_SCL</td>
<td>Serial clock</td>
<td>+1.8V</td>
<td>Open collector</td>
</tr>
<tr>
<td>17</td>
<td>I2C0_SDA</td>
<td>Serial data</td>
<td>+1.8V</td>
<td>Open Collector</td>
</tr>
<tr>
<td>19</td>
<td>I2C0_SCL</td>
<td>Serial clock</td>
<td>+1.8V</td>
<td>Open collector</td>
</tr>
<tr>
<td>21</td>
<td>I2C0_SDA</td>
<td>Serial data</td>
<td>+1.8V</td>
<td>Open Collector</td>
</tr>
</tbody>
</table>

B.4 SPI

\textbf{Table B.4:} SPI pins description.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
<th>Voltage</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>SPI0_SCLK</td>
<td>Serial clock</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>10</td>
<td>SPI0_DIN</td>
<td>Data in</td>
<td>+1.8V</td>
<td>Input</td>
</tr>
<tr>
<td>12</td>
<td>SPI0_CS</td>
<td>Chip select</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>14</td>
<td>SPI0_DOUT</td>
<td>Data out</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
</tbody>
</table>
B.5 GPIO

Table B.5: GPIO pins description.

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

B.6 PCM/I²S

Table B.6: PMI/I²S pins description.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
<th>Voltage</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>PCM_FS</td>
<td>PCM/I²S word clock</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>18</td>
<td>PCM_CLK</td>
<td>PCM/I²S bit clock</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>20</td>
<td>PCM_DO</td>
<td>PCM/I²S serial data out</td>
<td>+1.8V</td>
<td>Output</td>
</tr>
<tr>
<td>22</td>
<td>PCM_DI</td>
<td>PCM/I²S serial data in</td>
<td>+1.8V</td>
<td>Input</td>
</tr>
</tbody>
</table>
Appendix C  Schematics

C.1  Rock960 Board

List of diagrams:

- Power tree (view)
- I²C 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.
<table>
<thead>
<tr>
<th>Port</th>
<th>Pin name</th>
<th>Domain</th>
<th>Bus name</th>
<th>Pull-up voltage</th>
<th>Slave Device</th>
<th>Slave Addr (MS 7Bits)</th>
<th>Note</th>
<th>Slave Bus Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C0</td>
<td>GPIO1_B7/12_800/I2C0_SDA</td>
<td>SPI3</td>
<td>I2C0_SDA,GPIO1_B7</td>
<td>VCC_1V8</td>
<td>Rockchip ROCK96</td>
<td>0x00</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz</td>
</tr>
<tr>
<td></td>
<td>GPIO1_C0/SPI3_TXD/I2C0_SCL</td>
<td>SPI3</td>
<td>I2C0_SCL,GPIO1_C0</td>
<td>VCC_1V8</td>
<td>SYR837PKC</td>
<td>0x01</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz, 1MHz</td>
</tr>
<tr>
<td></td>
<td>GPIO3_B3/I2C4_SDA</td>
<td>SPI3</td>
<td>I2C4_SDA,GPIO3_B3</td>
<td>VCC_1V8</td>
<td>PMUIO2</td>
<td>0x02</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz, 1MHz</td>
</tr>
<tr>
<td></td>
<td>GPIO3_B4/I2C4_SCL</td>
<td>SPI3</td>
<td>I2C4_SCL,GPIO3_B4</td>
<td>VCC_1V8</td>
<td>PMUIO2</td>
<td>0x03</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz, 1MHz</td>
</tr>
<tr>
<td></td>
<td>GPIO3_B5/SPI2_RXD/CIF_HREF/I2C6_SDA</td>
<td>SPI2</td>
<td>I2C6_SDA,GPIO3_B5</td>
<td>VCC_1V8</td>
<td>AP101</td>
<td>0x04</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz, 1MHz</td>
</tr>
<tr>
<td></td>
<td>GPIO3_B6/SPI2_TXD/CIF_CLKIN/I2C6_SCL</td>
<td>SPI2</td>
<td>I2C6_SCL,GPIO3_B6</td>
<td>VCC_1V8</td>
<td>AP101</td>
<td>0x05</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz, 1MHz</td>
</tr>
<tr>
<td></td>
<td>GPIO3_B7/VOP_D7/CIF_D7/I2C7_SDA</td>
<td>SPI2</td>
<td>I2C7_SDA,GPIO3_B7</td>
<td>VCC_1V8</td>
<td>AP101</td>
<td>0x06</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz, 1MHz</td>
</tr>
<tr>
<td></td>
<td>GPIO3_B0/VOP_CLK/CIF_VSYNC/I2C7_SCL</td>
<td>SPI2</td>
<td>I2C7_SCL,GPIO3_B0</td>
<td>VCC_1V8</td>
<td>AP101</td>
<td>0x07</td>
<td>DC-DC BUCK</td>
<td>100kHz, 400kHz, 1MHz</td>
</tr>
</tbody>
</table>

**Figure C.2:** Schematic - Rock960 I2C 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.
HDMI design rule:
1. Max intra-pair skew < 4 ps;
2. Max length skew between clk and data < 80 ps;
3. Max trace length < 9.8 inches;
4. Max allowed via < 4;
5. Trace impedance 100Ω ± 10%;

eDP design rule:
1. Max intra-pair skew < 4 ps;
2. Max trace length < 6 inches;
3. Max allowed via < 4;
4. Trace impedance 90Ω ± 10%;
5. Trace impedance 100Ω ± 10%;

MIPI design rule:
1. Max intra-pair skew < 4 ps;
2. Max length skew between clk and data < 7ps;
3. Max trace length < 7.2 inches;
4. Max allowed via < 4;
5. Trace impedance 100Ω ± 10%;
Figure C.12: Schematic - Rock960 RK3399 GPIO.
PCIE design rule:
1. Max intra-pair skew < 4ps;
2. Max inter-pair skew < 1.6 ns;
3. Max trace length < 14 inchs;
4. Max allowed via < 4;
5. Trace impedance 100ohm ± 10%.

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.
C.2  LSM9DS0 Board

Figure C.25: Schematic - LSM9DS0 board.
C.3 Voltage Level Shifter Schematic

![Schematic - Voltage level shifter.](image)

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