USER MANUAL

RK3399-Q7 System-on-Module

Hexa-Core ARM Cortex-A72/A53

featuring the Rockchip RK3399 application processor

Document revision: Release v0.2-3-g574c114
Issue date: Jun 02, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

Congratulations for acquiring our new flagship product, combining best-in-class performance with a rich set of peripherals.

Note: The latest version of this manual and related resources can always be found on our website at the following address:

https://www.theobroma-systems.com/rk3399-q7/

This is a PRELIMINARY VERSION of the RK3399-Q7 user manual. The hardware description is complete. The software chapter will be finished in the next release of this manual.

1.1 Device Overview

The RK3399 is a low power, high performance processor for computing, personal mobile internet devices and other smart device applications. Based on a big.LITTLE architecture, it integrates a dual-core Cortex-A72 and a quad-core Cortex-A53. These 64bit-capable ARMv8 processors support both the ARM Crypographic Extension (e.g. for wire-rate AES encryption) and AdvSIMD vector processing. A dual-channel memory interface sustains the memory bandwidths required by even the most demanding embedded applications.

CHAPTER

TWO

FIRST STEPS

This chapter provides instructions for getting the RK3399-Q7 EVK running after opening the box.

2.1 Required Tools

- PZ1 (Pozidriv) screwdriver

2.2 Mount Module and Heatsink

Module and heatsink must be installed at the same time because the same mouting screws hold both the heatsink and the module in place.

The heatsink has the thermal pad attached on the bottom. Peel off the protective foil.

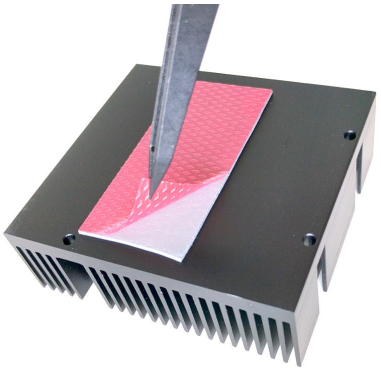


Fig. 2.1: Thermal pad protective foil

Insert the RK3399-Q7 module at a 45-degree angle into the connector in the base board. Once fully inserted, push it down until it rests on the standoffs.

Place the heatsink on the module and screw it down *very gently* using four screws of 10mm length.

Fig. 2.2: Screws used for mounting heatsink and RK3399-Q7 module.

2.3 Mount the Fan (optional)

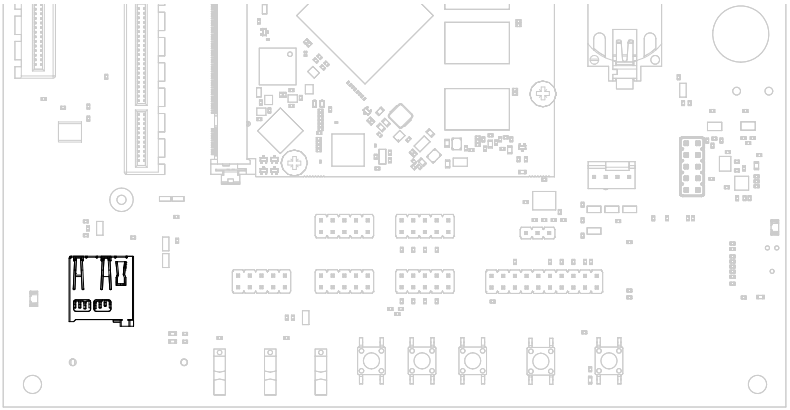
Note: The fan is only necessary in exceptionally high ambient temperatures. Under normal conditions, the RK3399-Q7 operates passively cooled.

Place the fan on the heatsink and screw it down using four 19 mm long screws.

Fig. 2.3: Screws used for mounting the fan

2.4 Insert SD Card & Power Up

Insert the enclosed SD card into the slot on the base board. The slot is marked with the SD logo.



Connect the power supply to the base board.

Press the “Power” button on the base board. The module will boot up. You will see the boot progress and get a login prompt on the RS-232 interface and on an HDMI monitor (if connected).

For further details, see the sections *3.5 Booting from SD Card*, *3.2 Power Supply* and *3.1 Evaluation Board Overview*.

CHAPTER

THREE

USING THE EVK

This chapter provides instructions for using the EVK, such as booting and how to configure and use I/O peripherals (e.g. serial console, Ethernet).

3.1 Evaluation Board Overview

An overview of the available connectors and devices on the EVK is shown below.

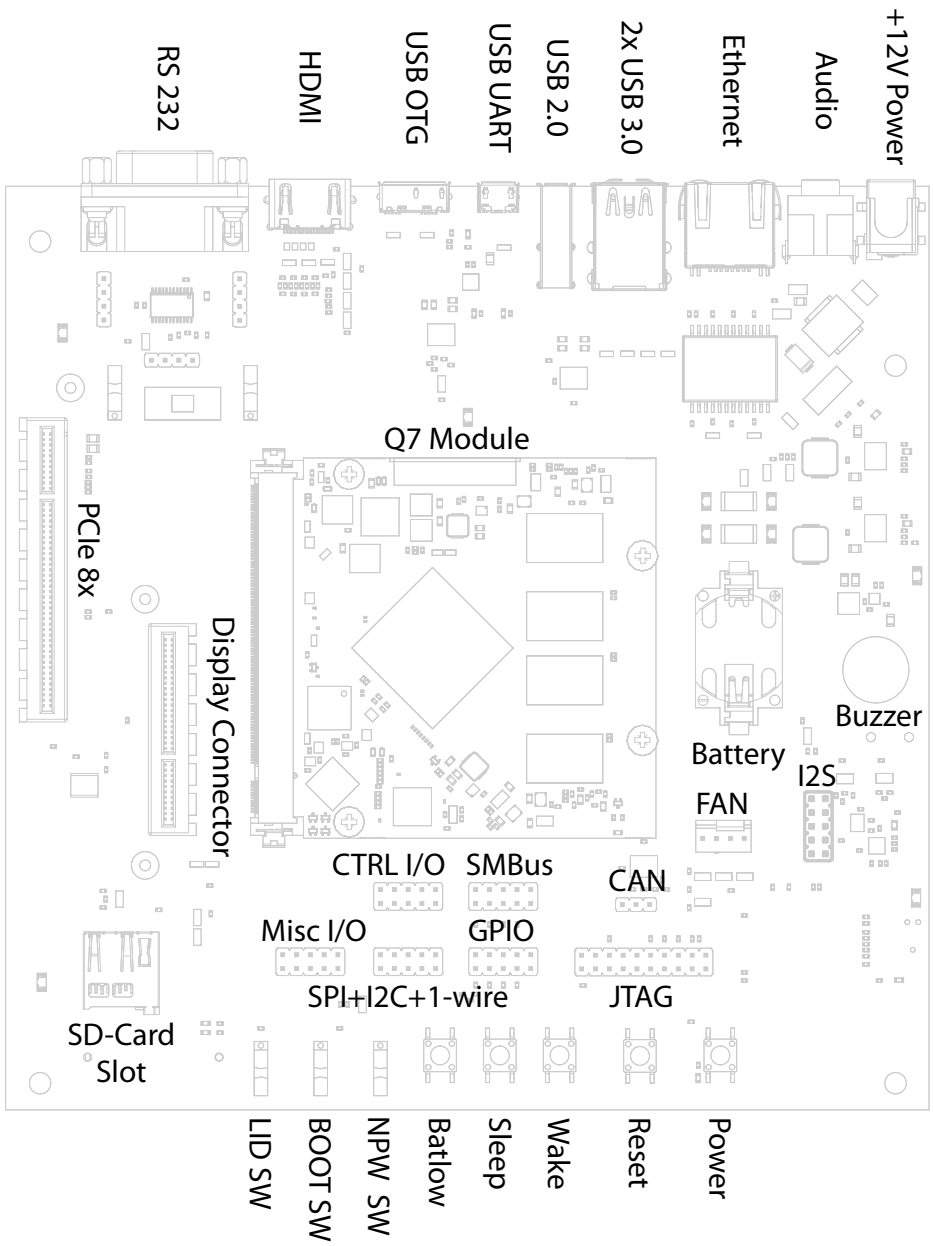


Fig. 3.1: The base board for to RK3399 Q7 module.

3.2 Power Supply

The baseboard can operate with a single 12V DC power supply.

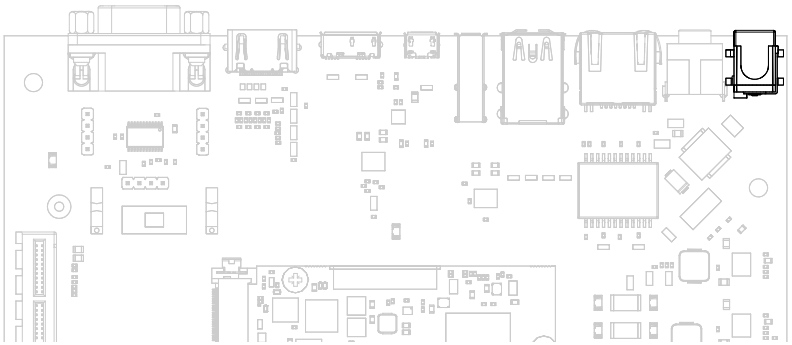


Fig. 3.2: 12V Power connector

Power can be controlled manually from the board using the "`Power`" control buttons and switches, located on the lower right side of the board (see *3.1 Evaluation Board Overview*).

Depending on the setting of "`Normally On / Normally Off`" switch the board will boot as soon as it receives power.

3.3 Control Buttons and Switches

The control buttons provide the following functionality:

- "`Power`" toggles the module power supply
- "`Reset`" triggers a module reset
- "`Batlow`", "`Sleep`" and "`Wake`" are routed to GPIOs on the Q7 module

Several slide switches are located on the lower left:

- "`LID`" is routed to a GPIO on the module, simulates lid open/close.
- "`Normally On / Normally Off`", as described above, sets the state after power loss.
- "`BIOS Disable / Normal Boot`" forces SD card boot or the normal boot order, respectively.

3.4 CPU Fan

Operation in high environmental temperatures may require a CPU fan. The fan connector is located next to the bottom right corner of the Q7 expansion area (see board overview).

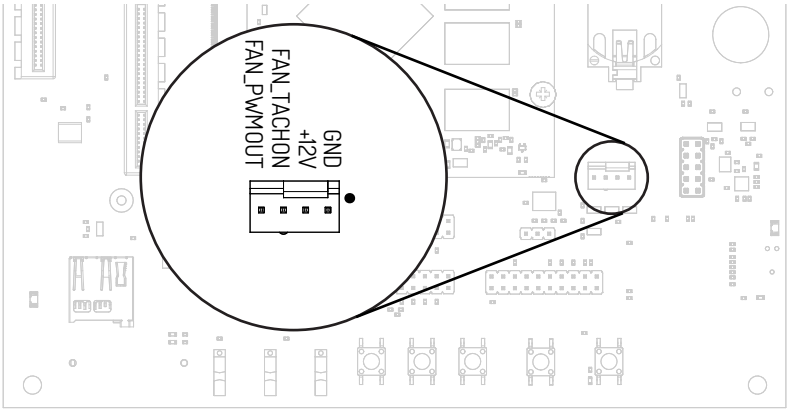


Fig. 3.3: Fan connector

Note: The fan is only necessary in high ambient temperatures. Under normal conditions, the RK3399 Q7 operates passively cooled.

3.5 Booting from SD Card

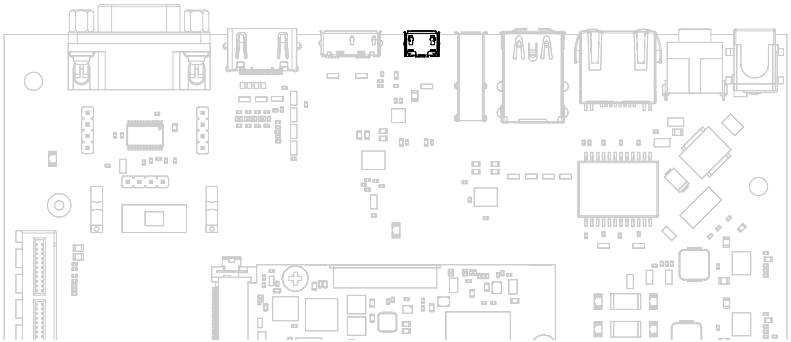
For information on preparing an SD card for the RK3399 Q7 board, have a look at the :ref: *Software Guide*. On power up, RK3399-Q7 module will normally try to boot from the internal flash. If this fails, it will attempt to boot from the SD card. If this fails as well, it will go into USB recovery mode. The standard boot order of the RK3399-Q7 is

1. SPI
2. eMMC
3. SD-Card
4. USB Recovery

The board can be forced to boot from the SD card by setting the "BIOS Disable / Normal Boot" slider to the "BIOS Disable" position. This will force the RK3399-Q7 module to boot from the SD card.

3.6 USB Serial Console

The evaluation board contains an on-board Silicon Labs CP2102N USB-serial converter. Connect the included Micro-USB cable to the Micro-USB jack labeled "USB-UART Bridge":



The serial converter does not require additional drivers on Windows and Linux.

For Mac OS, drivers are available from Silicon Labs: http://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

The Q7 modules has two external UARTs:

- UART0 is, by default, used for the serial console for interactive login.
- UART1 is unused by default and can be freely used for machine-to-machine communications or other purposes.

The switch "UART0 / UART1" cross-switches UART0 and UART1 between the "RS232 / RS485" jack and the onboard USB-serial converter:

Switch Position	RS232 / RS485 jack connected to:	USB-serial converter connected to:
UART0	UART0 (interactive console)	UART1
UART1	UART1	UART0 (interactive console)

For interactive login through the USB-serial converter, make sure the switch is on the "UART1" position

Picocom can be used to connect via the serial line (assuming the USB-serial converter is USB0):

```
picocom -b 115200 /dev/ttyUSB0
```

After system bootup, the login console appears on the terminal:

```
rk3399-q7 login:
```

You can log in as `root` with password `root` or as user `user` with password `user`.

3.7 RS-232 and RS-485

To connect via RS-232 or RS-485, connect to the "`RS232 / RS485`" jack on the base board.

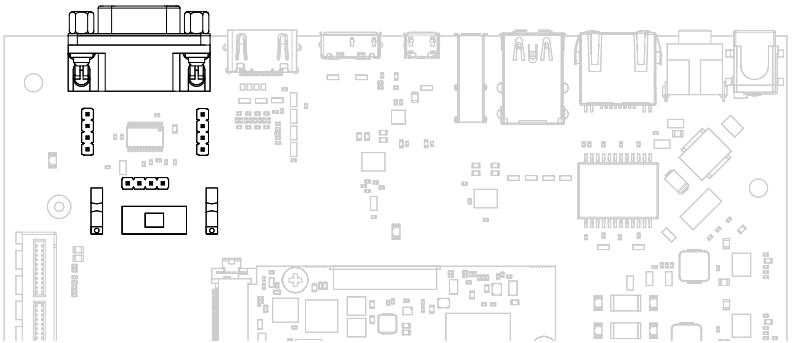


Fig. 3.4: RS-232 connector

The switch labeled "`RS-232 / RS-485`" selects between RS-232 and RS-485 mode on the jack.

In RS-485 mode, the switch labeled "`Full Duplex / Half Duplex`" selects full- or half-duplex mode, respectively. It has no effect in RS-232 mode, which is always full-duplex.

3.8 TTL UART

UART0 and UART1 are also available through the pin headers "`P12 UART0`" and "`P30 UART1`" next to the "`RS232 / RS485`" jack. The signal level is 3.3V.

3.9 Ethernet

The RK3399-Q7 has built-in Gigabit Ethernet routed to a standard jack on the evaluation board.

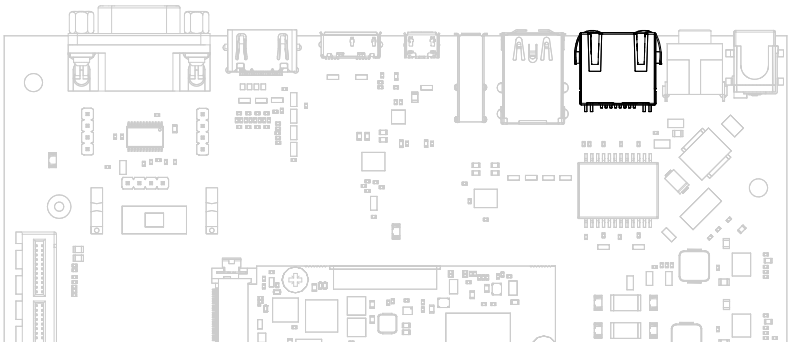


Fig. 3.5: Ethernet jack

The SD card that is shipped with the EVK is configures to automatically retrieve an IP via DHCP and provides SSH login on port 22.

3.10 USB Interfaces

The RK3399-Q7 provides four USB ports:

- 1x USB 3.0 OTG
- 2x USB 3.0 Host
- 1x USB 2.0 Host

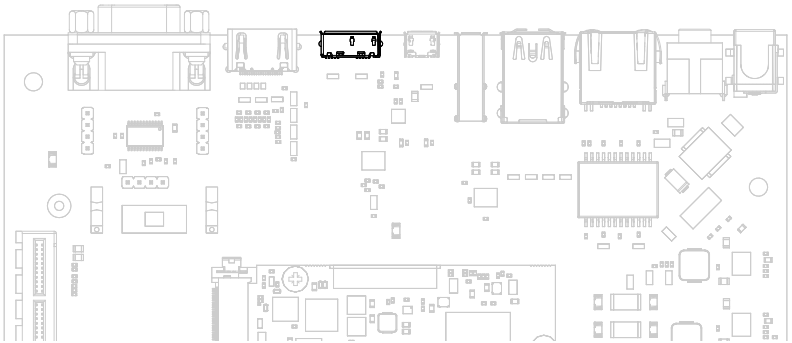


Fig. 3.6: USB 3.0 OTG port (dual-role port: can be used as a host or device interface)

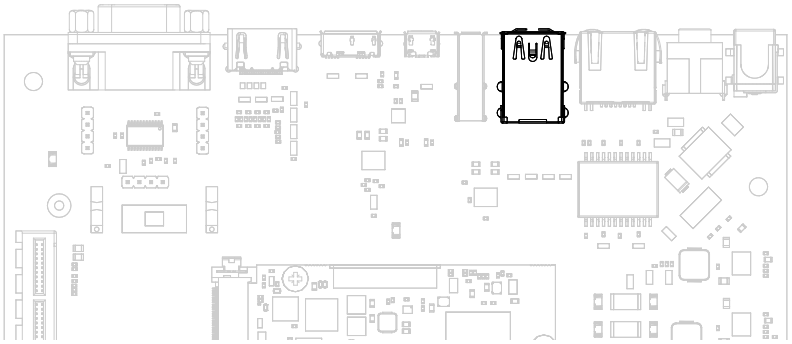


Fig. 3.7: USB 3.0 host ports

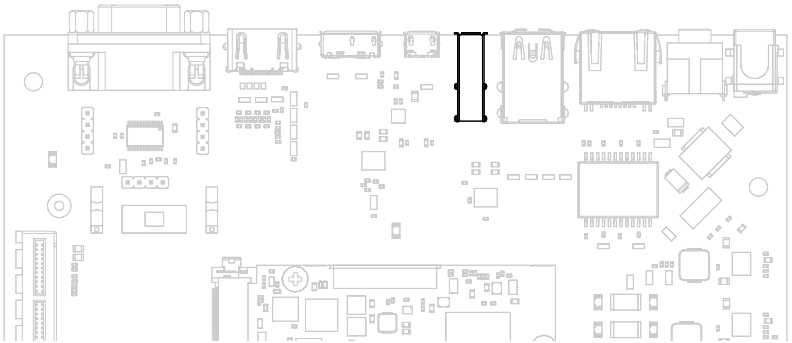


Fig. 3.8: USB 2.0 host port

3.10.1 Connecting an External USB Drive

To connect a USB drive, plug it into one of the USB ports. The system should recognize the drive immediately. Check the kernel log to find the device name:

```
journalctl -k -10
```

You will be able to mount its partitions (assuming mapping to `/dev/sdb1`):

```
mkdir /mnt/usb1
mount /dev/sdb1 /mnt/usb1
ls /mnt/usb1
```

3.11 HDMI

Before powering the board, connect a monitor to the HDMI port. The monitor will be automatically discovered and show a desktop environment once booting has finished.

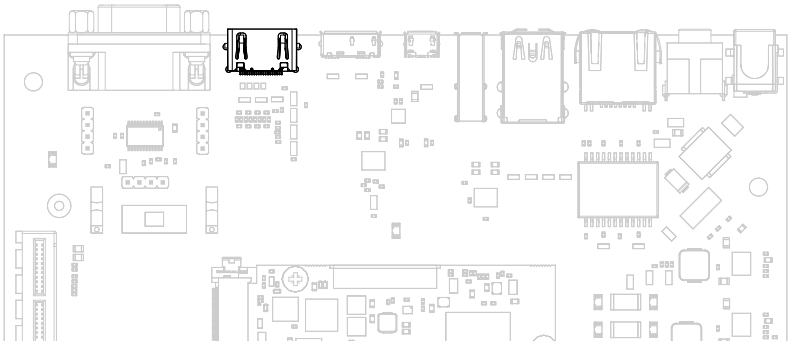


Fig. 3.9: HDMI port

3.12 RTC

the RK3399-Q7 contain a real-time clock (RTC) on-module. The RTC is read by the kernel on bootup and used to set the system clock.

To check the RTC value, use `hwclock`:

```
hwclock
Thu 30 Apr 2015 03:51:20 PM CEST  -0.826662 seconds
```

The RTC will be set automatically to the system clock on shutdown, so you can set the system clock using the date command and reboot to update the RTC:

```
date --set 2015-04-20
date --set 03:51:30
```

You can also update the RTC immediately, again with `hwclock`:

```
hwclock -v
```

You can set up an NTP client so the time will always be updated from the Internet. Install the client first:

```
apt-get install ntp
```

Feel free to change the `/etc/ntp.conf` file to use more local time sources (change servers from `pool.ntp.org` to use a server from your country, such as `at.pool.ntp.org`).

3.13 SPI, I2C and 1-Wire

The I2C (i2c-0), 1-wire-bus and SPI interfaces are both available on the connector labeled `SPI+I2C+1-Wire`.

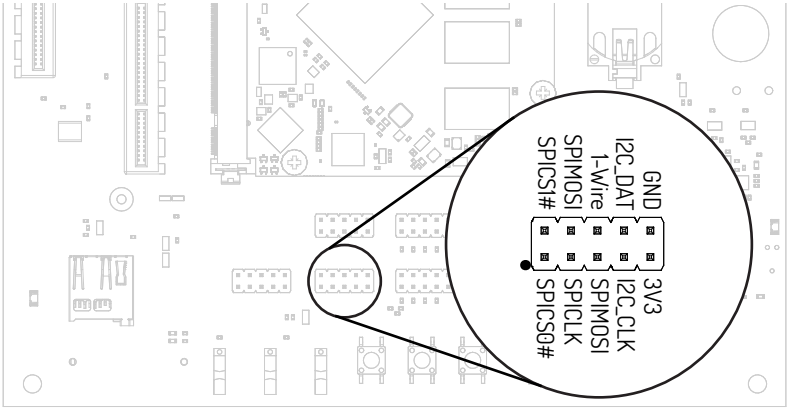


Fig. 3.10: I2C and SPI header

For I2C, there is the package i2c-tools available in Debian:

```
apt-get install i2c-tools
```

3.13.1 I2C Example - Using a Touch Keyboard

This example uses the Atmel AT42QT2160 touch keyboard (see datasheet).

Make sure the Linux kernel driver is enable via *menuconfig*:

```
make menuconfig
```

Navigate to *Device Drivers -> Input device support -> Keyboards* and check the *ATMEL AT42QT2160 Touch Sensor Chip*. You must recompile the kernel and deploy it to the SD card (see Software Guide).

3.13.2 SMBUS

The board provides communication through SMBUS. It is basically like I2C with an additional line for interrupt and is used for connecting sensors and power peripherals.

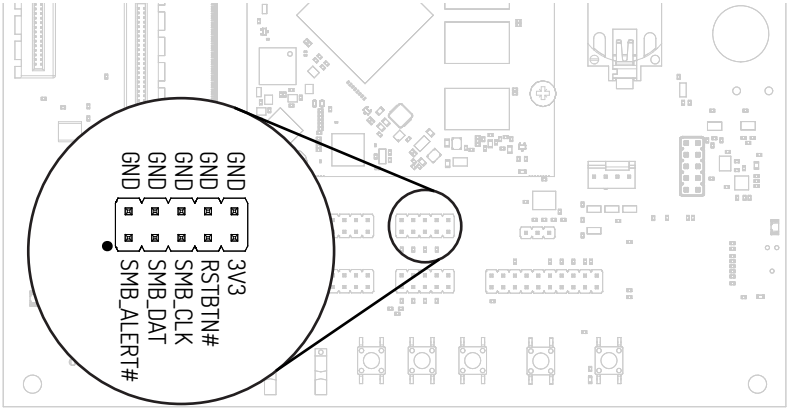


Fig. 3.11: SMBUS header

3.14 GPIOs

Eight GPIOs are provided on the pin header labled `GPIO`.

The location on the board is displayed below:

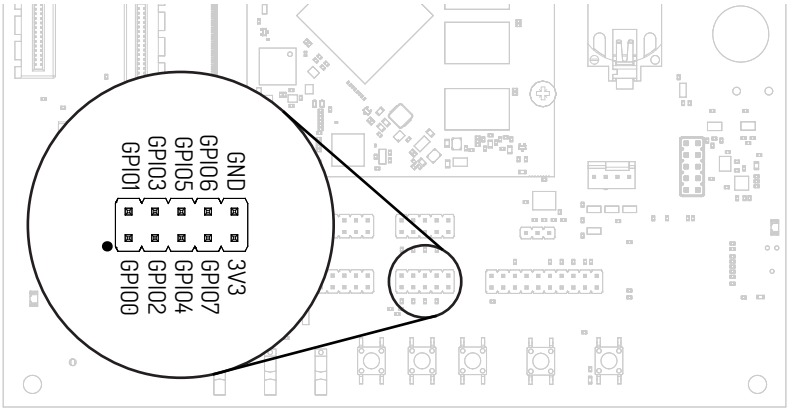


Fig. 3.12: GPIO header

The GPIO numbers printed on the board refer to numbers used in the Qseven specification. They are different than the ones used in Linux via */sys/class/gpio*.

The mapping is shown in the following table:

Qseven signal	CPU pin	Linux GPIO #
GPIO0	GPIO4_D4	156
GPIO1	GPIO4_D1	153
GPIO2	GPIO4_D0	152
GPIO3	GPIO4_D5	157
GPIO4	GPIO4_D2	154
GPIO5	GPIO4_C4	148
GPIO6	GPIO4_C3	147
GPIO7	GPIO4_D3	155

To calculate the Linux GPIO # for CPU pins that are not listed in this table, use the following formula:

```
n = (block_number * 32) + (sub_block_number * 8) + index
```

Where:

- block_number: index of the block number
- sub_block_number: the alphabetical index of the block name, minus 1
- index: the pin number within the block:

```
Example: GPIO4_D4 -> (4 * 32) + (3 * 8) + 4 = 156
```

To enable a GPIO, write the Linux GPIO # to the special *export* file:

```
echo 156 > /sys/class/gpio/export
ls /sys/class/gpio/gpio156
cat /sys/class/gpio/gpio156/direction
in
cat /sys/class/gpio/gpio156/value
0
```

To set the direction to output, write *out* in the GPIO's direction file:

```
echo out > /sys/class/gpio/gpio156/direction
echo 1 > /sys/class/gpio/gpio156/value
```

The GPIO will be set to a value of *1* (high at 3.3V).

3.15 Audio

The board provides two audio connectors for input and output. `Line-in` is on top and `Headphones` is on bottom of the audio connector.

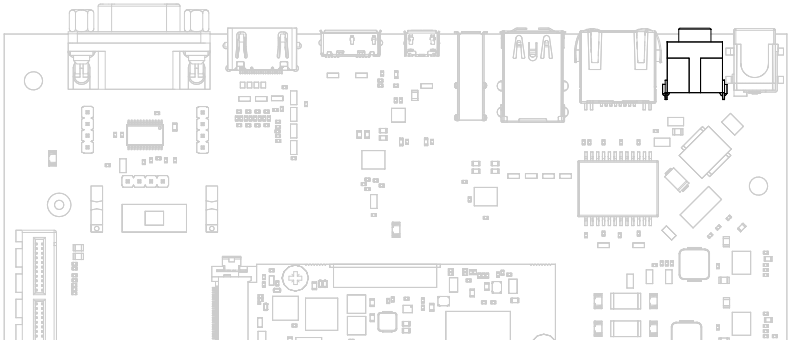


Fig. 3.13: Audio input/output port

Additionally, an expansion connector for I2S audio is available on the bottom row of the board:

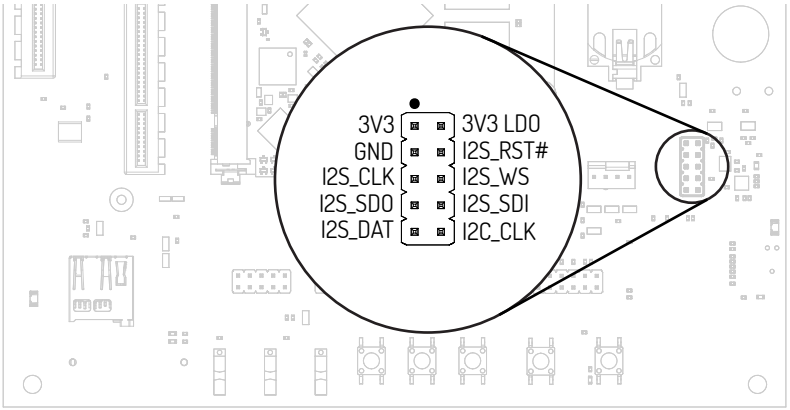


Fig. 3.14: Connecting to the audio expansion connector

3.16 CAN Bus

The board provides a CAN connector on the bottom row.

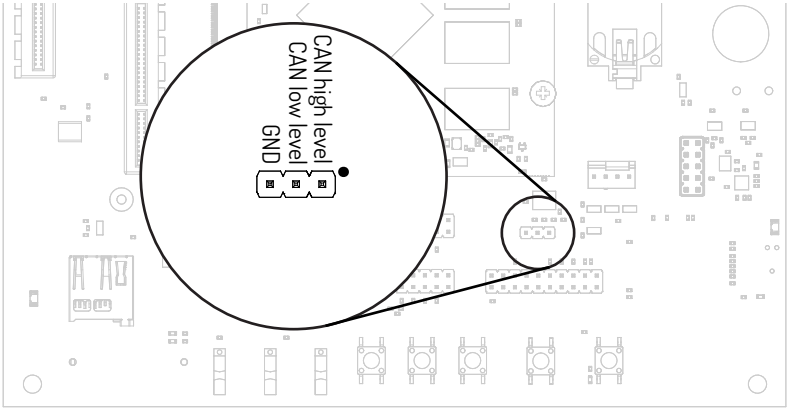


Fig. 3.15: CAN header

3.17 MISC Connector

The board provides signals for thermal overheat of external hardware and the processor, utility signals for SD and GPIO0.

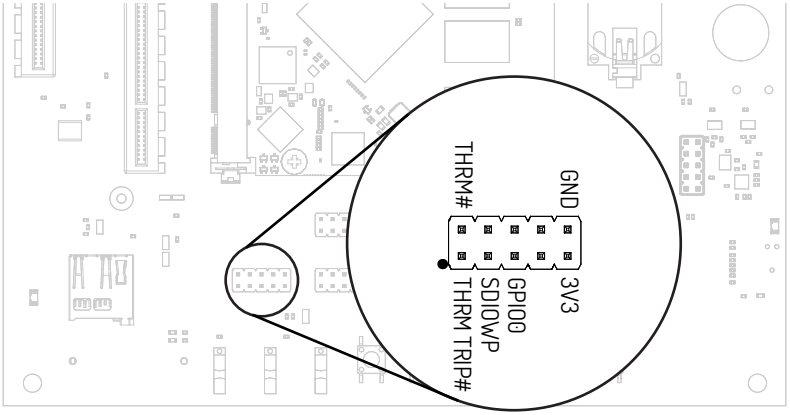


Fig. 3.16: MISC header

3.18 RF-Module

An additional RF-Module for wireless communication can be soldered on the bottom right of the baseboard.

For more info's visit: https://www.theobroma-systems.com/rf-modules-3815

CHAPTER

FOUR

SOFTWARE GUIDE

This chapter provides instructions for compiling and deploying the BSP (Board Support Package) software to the Q7 module.

4.1 Architecture Overview

The BSP consists of several parts. They run on different parts of the CPU and each play their role in the boot process. Because the CPU contains cores running different instruction sets (Cortex-M and Cortex-A), two different compilers are needed. The table below list the parts and their instruction set.

BSP Part	Instruction set
Cortex-M0 power management firmware	Cortex-M
ATF (ARM Trusted Firmware)	Cortex-A
U-Boot bootloader	Cortex-A
The Linux kernel	Cortex-A
Debian user-space	Cortex-A

The next section explains how to install suitable cross-compilers for both instruction sets.

The section “Compiling Linux Applications” provides guidance for compiling user-space applications for the RK3399.

4.2 Prerequisites

You need a recent x86_64 Linux installation to run the cross-compiler on and at least 10GB of disk space. The cross-compiler requires libc version 2.2.5. Distributions shipping this version are, among others:

- Ubuntu 16.04 “Xenial”

- Debian 8 "Jessie"

We recommend Ubuntu 16.04 "Xenial". Please install the following packages to set up the common build infrastructure:

```
sudo apt install device-tree-compiler u-boot-tools build-essential git bc debootstrap qemu-user-static
```

4.2.1 Cortex-M Compiler

The "GNU Embedded Toolchain for ARM" is suitable for compiling the Cortex-M0 power management firmware. It can be downloaded from https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads .

Direct link to the file:

https://developer.arm.com/-/media/Files/downloads/gnu-rm/6_1-2017q1/gcc-arm-none-eabi-6-2017-q1-update-linux.tar.bz2

Extract the tar.bz2 archibe to /opt:

```
sudo tar -xf gcc-arm-none-eabi-6-2017-q1-update-linux.tar.bz2 -C /opt
```

4.2.2 Cortex-A Compiler

The Linaro aarch64-linux-gnu toolchain is suitable for compiling all other parts of the BSP. It is also suitable for compiling user-space applications. You can download ready-to-use binaries from Linaro: https://releases.linaro.org/components/toolchain/binaries/6.3-2017.02/aarch64-linux-gnu/ .

Direct link to the file:

https://releases.linaro.org/components/toolchain/binaries/6.3-2017.02/aarch64-linux-gnu/gcc-linaro-6.3.1-2017.02-x86_64_aarch64-linux-gnu.tar.xz

Extract the tar.xz archive to /opt:

```
sudo tar -xf gcc-linaro-6.3.1-2017.02-x86_64_aarch64-linux-gnu.tar.xz -C /opt
```

4.3 Compile the Cortex-M0 power management firmware

The Cortex-M0 firmware runs inside a microcontroller embedded in the CPU IC. It implements power-management functionality and helpers (e.g. DRAM frequency switching support).

Set up environment variables to make use of the Cortex-M compiler, then download the source code and compile:

export ARCH=arm64 export CROSS_COMPILE=/opt/gcc-arm-none-eabi-6-2017-q1-update/bin/arm-none-eabi- git clone https://git.theobroma-systems.com/rk3399-cortex-m0.git cd rk3399-cortex-m0 make cd ..

4.3.1 Optional: Compile the cross-compiler

As an alternative to using a ready-made compiler, the firmware repository has a mechanism to compile the Cortex-M-compiler as a part of the build process. This is called "internal toolchain".

If you want to use the internal toolchain instead you will also need the following packages:

```
sudo apt install libssl-dev autoconf gperf bison flex texinfo help2man gawk libncurses5-dev
```

Then to use the internal toolchain, specify "USE_INTERNAL_TOOLCHAIN=1" as part of your invocation to make.:

```
make USE_INTERNAL_TOOLCHAIN=1
```

4.4 Compile the ATF

Download the source code and compile using:

```
export CROSS_COMPILE=/opt/gcc-linaro-6.3.1-2017.02-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
git clone https://git.theobroma-systems.com/arm-trusted-firmware.git
cd arm-trusted-firmware
make PLAT=rk3399 bl31
cd ..
```

4.5 Compile U-Boot

U-Boot is used as the bootloader on the RK3399-Q7 module.

Download the source code using:

```
git clone https://git.theobroma-systems.com/puma-u-boot.git
```

The U-Boot build process uses the files generated in the previous steps Copy the previously generated files "`rk3399m0.bin`" from Cortex-M0 firmware and "`bl31.bin`" from the ATF to the puma-u-boot directory:

```
cp rk3399-cortex-m0/rk3399m0.bin arm-trusted-firmware/build/rk3399/debug/bl31.bin puma-u-boot
```

Then you are ready to compile U-Boot:

```
cd puma-u-boot
export ARCH=arm64
export CROSS_COMPILE=/opt/gcc-linaro-6.3.1-2017.02-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
make puma-rk3399_defconfig
make -j4
tools/mkimage -n rk3399 -T rksd -d spl/u-boot-spl.bin spl.img
make u-boot.itb
cd ..
```

The resulting bootloader consists of two files, "`spl.img`" and "`u-boot.itb`".

4.6 Compile the Boot Script

The U-Boot boot sequence is controlled by a file called "`boot.scr`". This file is generated from a plain-text file called "`boot.cmd`".

Download the repository containing the file using:

```
git clone https://git.theobroma-systems.com/som-tools.git
```

Generate "`boot.scr`" using:

```
cd som-tools
make -C boot-script/boot
cd ..
```

4.7 Compile the Linux Kernel

The kernel source code can be cloned with:

```
git clone https://git.theobroma-systems.com/puma-linux.git
```

Compile using:

```
cd puma-linux
export ARCH=arm64
export CROSS_COMPILE=/opt/gcc-linaro-6.3.1-2017.02-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
make puma-rk3399_defconfig
make -j4 rockchip/rk3399-puma.dtb Image
```

This will create the two files needed for booting with U-Boot (paths are relative to the `puma-linux` directory):

- The device tree: "`arch/arm64/boot/dts/rockchip/rk3399-puma.dtb`"
- The kernel image: "`arch/arm64/boot/Image`"

4.8 Building the root filesystem

A filesystem can be created using Debootstrap, specifying arm64 as architecture in the command line.

Supposing the target dir is called `rk3399-rootfs` and the chosen distro is xenial (recommended):

```
export targetdir=/opt/rk3399-rootfs
export distro=xenial
sudo mkdir -p $targetdir
sudo debootstrap --arch=arm64 --foreign $distro $targetdir
```

Next, copy the qemu-arm-static binary into the right place for the binfmt packages to find it and copy the resolv.conf file from the host system:

```
sudo cp /usr/bin/qemu-aarch64-static $targetdir/usr/bin/
sudo cp /etc/resolv.conf $targetdir/etc
```

This will provide a very basic arm64 rootfs in the targetdir. For the next stages, we `chroot` to the target dir and set up the environment again:

```
cd $targetdir
sudo mount -t proc chproc $targetdir/proc
sudo mount -t sysfs chsys $targetdir/sys
sudo mount -t devtmpfs chdev $targetdir/dev
sudo mount -t devpts chpts $targetdir/dev/ptsd
sudo chroot $targetdir
export LC_ALL=C
export LANG=C
```

Second stage of debootstrap inside the new root dir:

```
/debootstrap/debootstrap --second-stage
locale-gen en_US.UTF-8
update-locale LANG=en_US.UTF-8 LANGUAGE=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
```

Setting up apt sources:

```
cat << EOT > /etc/apt/sources.list
deb http://ports.ubuntu.com/ubuntu-ports xenial main
deb http://ports.ubuntu.com/ubuntu-ports xenial-security main
deb http://ports.ubuntu.com/ubuntu-ports xenial-updates main
EOT
```

We can now pull the latest apt database from the Ubuntu mirrors and install the locales package (in jessie the dialog package might be needed as well):

```
apt-get update
apt-get install locales dialog
dpkg-reconfigure locales
```

Install any additional packages inside the chroot. An ssh server and sudo are recommended:

```
apt install openssh-server sudo
```

Set the root password for logging in via console or ssh:

```
passwd
```

root login over ssh is not permitted by default (set to "`no`"), or permitted only with public key (set to "`without-password`"). To verify that, run:

```
cat /etc/ssh/sshd_config | grep "PermitRootLogin"
PermitRootLogin without-password
```

Open the file "`/etc/ssh/sshd_config`" in your editor of choice and set PermitRootLogin to "`yes`".

A better option would be to create a user besides root:

```
adduser user
```

Optionally, you can add it to the `sudo` list:

```
adduser user sudo
```

Set up a basic network configuration file with DHCP via eth0:

```
cat << EOT >> /etc/systemd/network/eth.network
[Match]
Name=eth0
[Network]
DHCP=yes
EOT
systemctl enable systemd-networkd
systemctl enable systemd-resolved
```

Set the hostname:

```
echo rk3399-q7 > /etc/hostname
```

We can now exit from the chroot and clean up the support files:

```
exit
sudo rm $targetdir/etc/resolv.conf
sudo rm $targetdir/usr/bin/qemu-aarch64-static
umount -l xenial/dev/pts >/dev/null 2>&1
umount -l xenial/dev >/dev/null 2>&1
umount -l xenial/proc >/dev/null 2>&1
umount -l xenial/sys >/dev/null 2>&1
```

We now have a root filesystem which can be deployed to the SD card.

4.9 Deploy on SD Card

4.9.1 Partition Setup

Both U-Boot and Linux will be located on the same SD card. The layout of the card after setup is as follows:

Offset	Contents	Files
0	Partition table	
32kiB	U-Boot SPL “spl.img”	spl.img
240kiB	U-Boot environment	
256kiB	U-Boot + ATF + Cortex-M0 firmware	u-boot.itb
2MiB	Partition 1 (ext4 - Linux root fs)	boot.scr, Image, rk3399-puma.dtb, defaultEnv.txt and rootfs

To setup a SD card for booting you first need to create partitions. Partitions can be created using fdisk (assuming the SD card is mapped to /dev/sd“X“ , where `X` should be replaced with your corresponding device-letter) and has no partitions (this can be checked using the `p` command):

```
sudo fdisk /dev/sdX
> p
```

This should show an empty partition table, for example:

```
Disk /dev/sdX: 3980 MB, 3980394496 bytes
123 heads, 62 sectors/track, 1019 cylinders, total 7774208 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xdbbd45c7

   Device Boot      Start         End      Blocks   Id  System
```

If there are partitions on the sdcard, they can be deleted with `d`.

The required partition can be created with the command `n`, then accepting the defaults, accept for the first sector use 4096:

```
> n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): <ENTER>
Partition number (1-4, default 1): <ENTER>
First sector (2048-7774207, default 2048): 4096
Last sector, +sectors or +size{K,M,G} (...): <ENTER>
> w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

This will create a primary partition at offset 2MiB.

Now, to format the partition as ext4:

```
sudo /sbin/mkfs.ext4 -E lazy_itable_init=0 /dev/sdX1
```

The option "`lazy_itable_init=0`" speeds up the first boot because it initializes the inode tables in advance.

The SD card is now ready to have the U-Boot bootloader and Linux deployed.

4.9.2 Deploy U-Boot

The U-Boot images `spl.img` and `u-boot.itb` are written to the SD card. Assuming the SD card is mapped to /dev/sdX:

```
sudo dd if=puma-u-boot/spl.img of=/dev/sdX bs=1k seek=32
sudo dd if=puma-u-boot/u-boot.itb of=/dev/sdX bs=1k seek=256
```

4.9.3 Deploy the Linux Kernel and the Root Filesystem

Mount the SD card partition and copy the rootfs (assuming that the rootfs is located at "`/opt/rk3399-rootfs`" and the sd card at "`/dev/sdX1`"):

```
sudo mkdir -p /mnt/sdcard
sudo mount /dev/sdX1 /mnt/sdcard
sudo cp -av /opt/rk3399-rootfs/* /mnt/sdcard
```

Copy kernel image, device tree and boot script into the boot directory:

```
sudo cp -r som-tools/boot-script/boot/{boot.scr,puma_rk3399} /mnt/sdcard/boot
sudo cp rockchip_linux/arch/arm64/boot/dts/rockchip/rk3399-puma.dtb /mnt/sdcard/boot/puma_rk3399
sudo cp rockchip_linux/arch/arm64/boot/Image /mnt/sdcard/boot/puma_rk3399
```

Finally, unmount the SD card:

```
sudo umount /mnt/sdcard
```

The SD card is ready for booting.

4.9.4 U-Boot Customization

The boot script */boot/boot.scr* handles the boot sequence. Unless you want to customize the sequence, no further action is required.

If you want to step through the sequence manually or customize it, you can execute the following commands on the U-Boot prompt:

```
setenv bootargs console=tty0 console=ttyS2,115200 root=/dev/mmcblk1p1 rw rootwait
ext4load mmc 0:1 0x40008000 boot/uImage
ext4load mmc 0:1 0x40000000 boot/rk3399-puma.dtb
bootm 0x40008000 - 0x40000000
```

Optionally, these commands can be compiled together in a single command and saved so it is performed on every subsequent boot:

```
setenv boot_sd "ext4load mmc 0:1 0x40008000 boot/uImage
ext4load mmc 0:1 0x40000000 boot/rk3399-puma.dtb
bootm 0x40008000 - 0x40000000"
setenv bootcmd "run boot_sd"
saveenv
```

To reset the U-Boot settings to default, execute:

```
env default -f -a
saveenv
```

4.10 Deploy on NOR-flash

To have a reliable boot sequence even if eMMC and/or SD-card fail, U-Boot can be written in to the onboard NOR-flash. The file ”spl.img” for this has to be built differently than before.

Compile using:

```
cd puma-u-boot
export CROSS_COMPILE=/opt/gcc-linaro-6.3.1-2017.02-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
sudo make puma-rk3399_defconfig
sudo make -j4
sudo tools/mkimage -n rk3399 -T rkspi -d spl/u-boot-spl.bin spl.img
```

U-boot:

```
sf probe
load mmc 1 $kernel_addr_r root/spl.img
sf erase 0 +$filesize
sf write $kernel_addr_r 0 $filesize
```

```
load mmc 1 $kernel_addr_r root/u-boot.itb
sf erase 0x40000 +$filesize
sf write $kernel_addr_r 0x40000 $filesize
```

4.11 Deploy on On-Board eMMC storage

As the eMMC storage is only accessible from the module itself, you must first boot the RK3399-Q7 from SD card.

Partition and format the eMMC storage as described in *4.9.1 Partition Setup*, but using the device `/dev/mmcblk1`.

Mount the eMMC partition and copy the contents of the SD card to the eMMC storage. The copy process will take about 30 seconds:

```
sudo mkdir -p /mnt/emmc
sudo mount /dev/mmcblk1p1 /mnt/emmc
sudo cp -ax / /mnt/emmc
sudo umount /mnt/emmc
```

The final step is copying the bootloader to the eMMC:

```
dd if=spl.img of=/dev/mmcblk1 bs=1k seek=32
dd if=u-boot.itb of=/dev/mmcblk1 bs=1k seek=256
```

Shut down the board (`poweroff` command) and remove the SD card. Make sure the boot selector switch is set to "`Normal Boot`". The next boot will run U-Boot off the internal eMMC storage.

4.12 Compiling Linux Applications

The easiest option is to compile your applications directly on a running RK3399-Q7 module. Just install the gcc package and related utilities and you are good to go:

```
sudo apt-get install build-essential
```

The second option is to cross-compile your applications. The Cortex-A compiler that was installed earlier is suitable to compile applications for the RK3399-Q7.

4.13 Serial Number

Each RK3399-Q7 module has a unique serial number that can be read by software.

In U-Boot, the serial number is contained in the environment variable `serial#`. You can print it using the command:

```
printenv serial#
```

Under Linux, it is represented by a simple text file in `/sys`:

```
cat /sys/firmware/devicetree/base/serial-number
```

The serial number is fixed in hardware (derived from the eMMC *Product Serial Number*) and cannot be modified.

4.14 MAC Address

By default, the MAC address of each RK3399-Q7 module is a random value derived from the serial number. The properties of this default MAC address are:

- It is a *Locally Administered Address*: The U/L bit of the MAC address is set to 1
- It is not guaranteed to be globally unique
- The address is fixed for each RK3399-Q7 module. It stays constant across reboots as it is deterministically derived from the serial number

To set your own *Universally Administered Address*, you overwrite the U-Boot environment variable `ethaddr`. On the U-Boot prompt, with *XX:XX:XX:XX:XX:XX* replaced by your MAC address:

```
setenv ethaddr XX:XX:XX:XX:XX:XX
saveenv
```

The MAC address can be queried from the U-Boot prompt using:

```
printenv ethaddr
```

To reset the MAC address to the default value, run:

```
env delete ethaddr
saveenv
```

CHAPTER

FIVE

HARDWARE GUIDE

This Hardware Guide provides information about the features, connectors and signals available on the RK3399-Q7 module.

5.1 Qseven Implementation

Qseven has mandatory and optional features. Following table shows the feature set of the RK3399-Q7 module compared to the minimum ARM/RISC based and maximum configuration according to the Q7 standard.

System I/O Interface	Q7 Minimum	RK3399-Q7	Q7 Maximum
PCI Express lanes	0	4	4
Serial ATA channels	0	0	2
USB 2.0 ports	1	1	8
USB 3.0 ports	0	3	3
LVDS channels	0	0	2
Embedded DisplayPort	0	1	1
MIPI_CSI	0	2	2
HDMI	0	1	1
High Definition Audio / AC‘97 / I2S	0	1	1
Ethernet 10/100/Gigabit	0	1x Gigabit	1x Gigabit
UART	0	1	1
GPIO	0	8	8
Secure Digital I/O	0	1	1
System Management Bus	0	1	1
I^2C Bus	1	4	4
SPI Bus	0	1	1
CAN Bus	0	1	1
Watchdog Trigger	1	1	1
Power Button	1	1	1
Power Good	1	1	1
Reset Button	1	1	1
LID Button	0	1	1
Sleep Button	0	1	1
Suspend to RAM (S3 mode)	0	1	1
Wake	0	1	1
Battery low alarm	0	1	1
Thermal control	0	1	1
FAN control	0	1	1

Note: The RK3399-Q7 module is available in different variants. This document describes the maximum configuration. For details about orderable variants please refer to the order-code document.

5.2 Q7 Connector Pinout

The following table shows the signals on the edge connector of the RK3399-Q7 module.

Empty cells are not connected (NC) pins.

Pin	Signal	Pin	Signal
1	GND	2	GND
3	GBE_MDI3-	4	GBE_MDI2-
5	GBE_MDI3+	6	GBE_MDI2+
7	GBE_LINK100#	8	GBE_LINK1000#
9	GBE_MDI1-	10	GBE_MDIO0-
11	GBE_MDI1+	12	GBE_MDIO0+
13	GBE_LINK#	14	GBE_ACT#
15		16	SUS_S5#
17	WAKE#	18	SUS_S3#
19	GP0	20	PWRBTN#
21	SLP_BTN#	22	LID_BTN#
23	GND	24	GND
25	GND	26	PWGIN
27	BATLOW#	28	RSTBTN#
29		30	
31		32	
33			34 GND
35		36	
37		38	
39	GND	40	GND
41	BIOS_DISABLE# / BOOT_ALT#	42	SDIO_CLK#
43	SDIO_CD#	44	SDIO_LED
45	SDIO_CMD	46	SDIO_WP
47	SDIO_PWR#	48	SDIO_DAT1
49	SDIO_DAT0	50	SDIO_DAT3
51	SDIO_DAT2	52	
53		54	
55		56	USB_OTG_PEN
57	GND	58	GND
59	I2S_WS	60	SMB_CLK / GP1_I2C_CLK
61	I2S_RST#	62	SMB_DAT / GP1_I2C_DAT

Continued on next page

Table 5.1 – continued from previous page

Pin	Signal	Pin	Signal
63	I2S_CLK	64	SMB_ALERT#
65	I2S_SDI	66	GP0_I2C_CLK
67	I2S_SDO	68	GP0_I2C_DAT
69	THRM#	70	WDTRIG#
71	THRMTRIP#	72	WDOUT
73	GND	74	GND
75	USB_SSTX0-	76	USB_SSRX0-
77	USB_SSTX0+	78	USB_SSRX0+
79		80	
81	USB_SSTX2-	82	USB_SSRX2-
83	USB_SSTX2+	84	USB_SSRX2+
85	USB_2_3_OC#	86	USB_0_1_OC#
87	USB_P3-	88	USB_P2-
89	USB_P3+	90	USB_P2+
91	USB_CC	92	USB_ID
93	USB_P1-	94	USB_P0-
95	USB_P1+	96	USB_P0+
97	GND	98	GND
99	LVDS_A0+	100	LVDS_B0+
101	LVDS_A0-	102	LVDS_B0-
103	LVDS_A1+	104	LVDS_B1+
105	LVDS_A1-	106	LVDS_B1-
107	LVDS_A2+	108	LVDS_B2+
109	LVDS_A2-	110	LVDS_B2-
111	LVDS_PPEN	112	LVDS_BLEN
113	LVDS_A3+	114	LVDS_B3+
115	LVDS_A3-	116	LVDS_B3-
117	GND	118	GND
119	LVDS_A_CLK+	120	LVDS_B_CLK+
121	LVDS_A_CLK-	122	LVDS_B_CLK-
123	LVDS_BLT_CTRL / GP_PWM_OUT0	124	GP_1-Wire_Bus
125	GP2_I2C_DAT / LVDS_DID_DAT	126	LVDS_BLC_DAT / eDP0_HPD#
127	GP2_I2C_CLK / LVDS_DID_CLK	128	LVDS_BLC_CLK
129	CAN0_TX	130	CAN0_RX
131	TMDS_CLK+	132	USB_SSTX1-

Continued on next page

Table 5.1 – continued from previous page

Pin	Signal	Pin	Signal
133	TMDS_CLK-	134	USB_SSTX1+
135	GND	136	GND
137	TMDS_LANE1+	138	
139	TMDS_LANE1-	140	
141	GND	142	GND
143	TMDS_LANE0+	144	USB_SSRX1-
145	TMDS_LANE0-	146	USB_SSRX1+
147	GND	148	GND
149	TMDS_LANE2+	150	HDMI_CTRL_DAT
151	TMDS_LANE2-	152	HDMI_CTRL_CLK
153	DP_HDMI_HPD#	154	
155	PCIE_CLK_REF+	156	PCIE_WAKE#
157	PCIE_CLK_REF-	158	PCIE_RST#
159	GND	160	GND
161	PCIE3_TX+	162	PCIE3_RX+
163	PCIE3_TX-	164	PCIE3_RX-
165	GND	166	GND
167	PCIE2_TX+	168	PCIE2_RX+
169	PCIE2_TX-	170	PCIE2_RX-
171	UART0_TX	172	UART0_RTS#
173	PCIE1_TX+	174	PCIE1_RX+
175	PCIE1_TX-	176	PCIE1_RX-
177	UART0_RX	178	UART0_CTS#
179	PCIE0_TX+	180	PCIE0_RX+
181	PCIE0_TX-	182	PCIE0_RX-
183	GND	184	GND
185	GPIO0	186	GPIO1
187	GPIO2	188	GPIO3
189	GPIO4	190	GPIO5
191	GPIO6	192	GPIO7
193	VCC_BAT	194	SPKR / GP_PWM_OUT2
195	FAN_TACHOIN / GP_TIMER_IN	196	FAN_PWMOUT / GP_PWM_OUT1
197	GND	198	GND
199	SPI_MOSI	200	SPI_CS0#
201	SPI_MISO	202	SPI_CS1#

Continued on next page

Table 5.1 – continued from previous page

Pin	Signal	Pin	Signal
203	SPI_SCK	204	
205		206	
207		208	
209		210	
211		212	
213		214	
215		216	
217		218	
219	VCC	220	VCC
221	VCC	222	VCC
223	VCC	224	VCC
225	VCC	226	VCC
227	VCC	228	VCC
229	VCC	230	VCC

5.3 Signal Details

5.3.1 Ethernet

Signal	Type	Signal Level	Description
GBE_MDI[0:3]+ GBE_MDI[0:3]-	I/O	Analog	Gigabit Ethernet Controller: Media Dependent Interface Differential Pairs 0,1,2,3. The MDI can operate in 1000, 100 and 10 Mbit/sec modes
GBE_ACT#	OC	3.3V	Gigabit Ethernet Controller activity indicator, active low
GBE_LINK#	OC	3.3V	Gigabit Ethernet Controller link indicator, active low
GBE_LINK100#	OC	3.3V	Internally connected to GBE_LINK#
GBE_LINK1000#	OC	3.3V	Internally connected to GBE_LINK#
GBE_CTREF	REF	Analog	Center Tap Voltage

5.3.2 USB

Signal	Type	Signal Level	Description
USB_P[0:2]+ USB_P[0:2]-	I/O	USB	High speed universal Serial Bus Port 0, 1, 2 differential pairs
USB_SSTX[0:2]+ USB_SSTX[0:2]-	I/O	USB	Super speed universal Serial Bus Port 0, 1, 2 transmit differential pairs
USB_SSRX[0:2]+ USB_SSRX[0:2]-	I/O	USB	Super speed universal Serial Bus Port 0, 1, 2 receive differential pairs
USB_0_1_OC#	I	3.3V	Over current detect input 1. This pin is used to monitor the USB power over current of the USB Ports 0 and 1
USB_2_3_OC#	I	3.3V	Over current detect input 1. This pin is used to monitor the USB power over current of the USB Ports 2 and 3
USB_ID	I	3.3V	Configures the mode of the USB Port 1. If the signal is `active high` the Port will be configured as USB Client
USB_VBUS	I	5.0V	USB VBUS pin, 5V tolerant
USB_OTG_PEN	O	3.3V	USB Power enable for OTG port USB 1

5.3.3 SDIO

Signal	Type	Signal Level	Description
SDIO_CD#	I	3.3V	SDIO Card Detect. This signal indicates when a SDIO/MMC card is present
SDIO_CLK	O	3.3V	SDIO Clock
SDIO_CMD	I/O	3.3V	SDIO Command/Response
SDIO_LED	O	3.3V	SDIO LED. Used to drive an external LED to indicate transfers on the bus
SDIO_WP	I	3.3V	SDIO Write Protect
SDIO_PWR#	O	3.3V	SDIO Power Enable. This signal is used to enable the power being supplied to a SD/MMC card device
SDIO_DAT0-4	I/O	3.3V	SDIO Data lines

5.3.4 I2C

Signal	Type	Signal Level	Description
Q7_I2C_CLK	O	3.3V	I2C bus clock line connected to RK3399
Q7_I2C_DAT	I/O	3.3V	I2C bus data line connected to RK3399
LVDS_DID_CLK /GP2_I2C_CLK	O	3.3V	I2C bus clock line connected to RK3399
LVDS_DID_DAT /GP2_I2C_DAT	I/O	3.3V	I2C bus data line connected to RK3399
SMB_CLK GP1_I2C_CLK	O	3.3V	Clock line of System Management Bus. Alternate function I2C Bus clock line
SMB_DAT GP1_I2C_DAT	I/O	3.3V	Data line of System Management Bus. Alternate function I2C Bus data line
LVDS_BLC_DAT	O	3.3V	I2C bus clock line connected to RK3399, Kerkey and baseboard EEPROM
LVDS_BLC_CLK	I/O	3.3V	I2C bus data line connected to RK3399, Kerkey and baseboard EEPROM

5.3.5 I2S

Signal	Type	Signal Level	Description
I2S_RST#	O	3.3V	I2S Codec Reset
I2S_WS	O	3.3V	I2S Word Select
I2S_CLK	O	3.3V	I2S Serial Data Clock
I2S_SDO	O	3.3V	I2S Serial Data Output
I2S_SDI	I	3.3V	I2S Serial Data Input

5.3.6 HDMI

Signal	Type	Signal Level	Description
TMDS_CLK+ TMDS_CLK-	O	TMDS	TMDS differential pair clock lines
TMDS_LANE[0:2]+ TMDS_LANE[0:2]-	O	TMDS	TMDS differential pair lanes 0, 1, 2
HDMI_CTRL_CLK	O	3.3V	DDC based control signal (clock) for HDMI device
HDMI_CTRL_DAT	I/O	3.3V	DDC based control signal (data) for HDMI device
HDMI_HPD#	I	3.3V	Hot plug detection signal

5.3.7 GPIO

Signal	Type	Signal Level	Description
GPIO[0-7]	I/O	3.3V	General purpose inputs/outputs 0 to 7

5.3.8 CAN

Signal	Type	Signal Level	Description
CAN0_TX	O	3.3V	CAN (Controller Area Network) TX output for CAN Bus channel 0
CAN0_RX	I	3.3V	CAN (Controller Area Network) RX input for CAN Bus channel 0

5.3.9 SPI

Signal	Type	Signal Level	Description
SPI_MOSI	O	3.3V	Master serial output/Slave serial input signal
SPI_MISO	I	3.3V	Master serial input/Slave serial output signal
SPI_SCK	O	3.3V	SPI clock output
SPI_CS0#	O	3.3V	SPI chip select 0 output
SPI_CS1#	O	3.3V	SPI chip select 1 output (used when two devices are connected)

5.3.10 UART

Signal	Type	Signal Level	Description
UART0_TX	O	3.3V	Serial data transmit
UART0_RX	I	3.3V	Serial data receive
UART0_CTS#	I	3.3V	Handshake signal: ready to send data
UART0_RTS#	O	3.3V	Handshake signal: ready to receive data

5.3.11 Misc

Signal	Type	Signal Level	Description
WDTRIG#	I	3.3V	Watchdog trigger signal
WDOUT	O	3.3V	Watchdog event indicator
SMB_CLK GP1_I2C_CLK	O	3.3V	Clock line of System Management Bus. Alternate function I2C Bus clock line
SMB_DAT GP1_I2C_DAT	I/O	3.3V	Data line of System Management Bus. Alternate function I2C Bus data line
SMB_ALERT#	I	3.3V	System Management Bus Alert input
SPKR GP_PWM_OUT2	O	3.3V	Audio enunciator output. Alternate function general purpose PWM output
BIOS_DISABLE# /BOOT_ALT#	I	3.3V	Disables the onboard bootloader and uses the one the SD card instead. If no bootloader is available on the SD card it falls back to USB recovery mode
GP_1-Wire_Bus	I/O	3.3V	General Purpose 1-Wire bus interface
THRM#	I	3.3V	Thermal Alarm active low signal generated by the external hardware to indicate an over temperature situation. This signal can be used to initiate thermal throttling
THRMTRIP#	O	3.3V	Thermal Trip indicates an overheating condition of the processor. If 'THRMTRIP#' goes active the system immediately transitions to the S5 State (Soft Off)
FAN_PWMOUT /GP_PWM_OUT1	O	3.3V	PWM output for fan speed control. Alternate function general purpose PWM output. Function based on microcontroller firmware
FAN_TACHOIN /GP_TIMER_IN	I	3.3V	Fan tachometer input. Alternate function general purpose timer input. Function based on microcontroller firmware

5.3.12 Power Management

Signal	Type	Signal Level	Description
RSTBTN#	I	3.3V	Reset button input. An active low signal resets the module
BATLOW#	I	3.3V	Battery low input
WAKE#	I	3.3V	External system wake event. An active low signal wakes the module from a sleep state
SUS_S3#	O	3.3V	Indicated that the system is in suspend to ram (S3)
SUS_S5#	O	3.3V	Indicated that the system is in soft-off state (S5)
SLP_BTN#	I	3.3V	Sleep button. Signals the system with an falling edge to transition into sleep or wake from a sleep state
LID_BTN#	I	3.3V	LID button. Low active signal to detect a LID switch to transition into sleep or wake from a sleep state

5.3.13 Power

Signal	Nominal Input	Description
VCC	5V	Main supply for the module
VCC_RTC	3V	Backup supply for the RTC. If not used it can be left unconnected

5.4 On-board Devices

5.4.1 Power-Manager

The Rockchip RK808 is connected to the CPU via RSB and an interrupt line:

RK808 Pin	Function	CPU Pin
19	SCL	I2C0_SCL_u (ball N30)
18	SDA	I2C0_SDA_u (ball M26)
49	IRQ	GPIO1_C6 (ball L25)

5.4.2 DDR3

- 4GB RAM of DDR3-1600 (2 independent channels, each 32-bit wide)

5.4.3 eMMC

- eMMC connected through the 8-bit wide SDIO interface EMMC_D on the CPU.

Signal	CPU Pin	Linux GPIO #
RESET	GPIO0_A5	5

5.4.4 NOR Flash

- 32 MiB serial NOR flash
- Connected to the CPU via SPI1:

Signal	CPU Pin
CLK	GPIO_B1 (ball P28)
MOSI	GPIO_B0 (ball R31)
MISO	GPI1_A7 (ball P27)
CS	GPIO_B2 (ball P29)

5.4.5 Companion Controller

The on-board microcontroller provides additional features to the CPU, exposed via I2C and USB. It emulates standard ICs and does not need custom drivers in Linux.

Feature	CPU Connection	Emulated IC	Qseven Pins
RTC	I2C	ISL1208	none
Temperature sensor and fan controller	I2C	AMC6821	FAN_TACHOIN, FAN_PWMOUT
CAN	USB	UCAN	CAN0_TX, CAN0_RX

The STM32-microcontroller can be flashed from the CPU by taking it into DFU mode (USB recovery). Pull BOOT0 low and cycle reset (GPIOs listed below). The microcontroller will appear as a new USB device in Linux.

Function	CPU Pin	Linux GPIO #
NRST	GPIO1_D0	56
BOOT0	GPIO2_B4	76

5.4.6 USB

The Genesys Logic, Inc. GL3523 provides two additional USB 3.0 super-speed ports, and two additional USB 2.0 high-speed ports. The routing of Qseven signals to CPU and/or hub port is shown below.

Qseven Port #	Speed	Connected to	Hub Port #	Notes
USB_P0	USB 2.0 Hi-Speed	HUB0	1	
USB_P1	USB 2.0 Hi-Speed	CPU		USB 2.0 OTG Port
USB_P2	USB 2.0 Hi-Speed	HUB0	2	
USB_P3	USB 2.0 Hi-Speed	Hub0	3	
USB_SS(R/T)X0	USB 3.0 Hi-Speed	Hub0	1	
USB_SS(R/T)X1	USB 3.0 Hi-Speed	CPU		
USB_SS(R/T)X2	USB 3.0 Hi-Speed	Hub0	2	

The routing of the reset signal is shown below.

Hub signal	CPU Pin	Linux GPIO #
USBHUB_RESETn	GPIO4_A3	128

5.4.7 Ethernet PHY

The Micrel KSZ9031RNX is connected to the CPU via RGMII and MDIO. Further connections are shown below.

PHY signal	Connected to	Linux GPIO #
RESET	CPU pin GPIO3_C0	112
MDIO	CPU pin GPIO3_B5	107
MDC	CPU pin GPIO3_B0	102
LED1	Qseven GBE_LINK1000 and GBE_LINK100 and GBE_LINK (tied together)	
LED2	Qseven GBE_ACT	

5.4.8 Test points

Test point	Connected to
TP1	STM32 USART2 TX
TP2	STM32 USART2 RX
TP3	GND

5.5 Electrical Specification

5.5.1 Power Supply

The power supply requirements are listed in the table below and are identical to the Qseven specification.

Rail	Description	Nominal voltage	Tolerance
VCC	Main power supply	5V	4.75 ... 5.25V
VCC_RTC	Backup battery	3V	2.4 ... 3.3V

5.6 Mechanical Specification

5.6.1 Module Dimensions

The mechanical dimensions of the module are shown below.

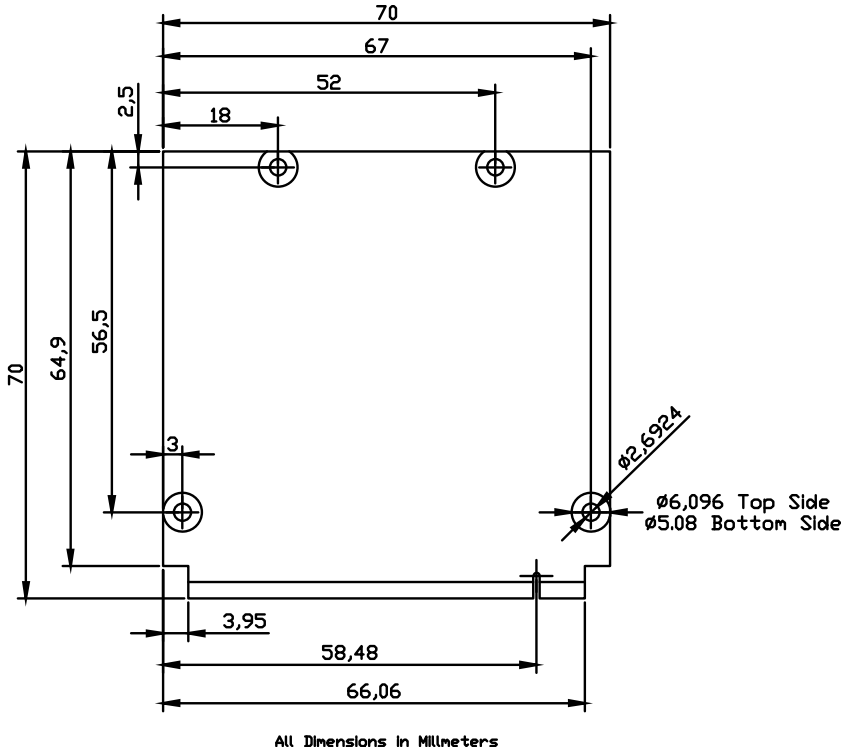


Fig. 5.1: Module dimensions (all values in mm)

5.6.2 Baseboard Dimensions

The mechanical dimensions of the baseboard are conform with the form factor for Micro-ATX and it can be mounted in a standard Micro-ATX PC Case.

CHAPTER

SIX

REVISION HISTORY

Date	Revision	Changes
Jun 2, 2017	v0.2	Preliminary public release
May 30, 2017	v0.1	First internal release