USER MANUAL

RINGNECK SOM-PX30-µQ7

Power efficient **System-on-Module** with Quad-Core ARM

featuring the **Rockchip PX30** application processor

Document revision: v1.4.0
Issue date: Mar 10, 2024

Contents

1 Introduction **1**
1.1 Device Overview . . . 1

2 First Steps **2**
2.1 Insert the Module . . . 2
2.2 Mount the Fan (optional) . . . 2
2.3 Power Up . . . 2

3 Using the EVK **4**
3.1 Evaluation Board Overview . . . 4
3.2 Power Supply . . . 6
3.3 Control Buttons and Switches . . . 6
3.4 CPU Fan . . . 6
3.5 Boot Order . . . 7
3.6 USB Serial Console . . . 8
3.7 RS-232 and RS-485 . . . 9
3.8 TTL UART . . . 9
3.9 Ethernet . . . 10
3.10 SD-Card . . . 10
3.11 USB Interfaces . . . 11
3.12 Display and Camera . . . 12
3.13 RTC . . . 13
3.14 SPI and I2C . . . 13
3.15 GPIOs . . . 14
3.16 Audio . . . 16
3.17 CAN Bus . . . 17
3.18 CTRL I/O Connector . . . 17
3.19 MISC Connector . . . 18
3.20 JTAG Connector . . . 18

4 Software Overview **19**
4.1 Supported Distributions . . . 19
4.2 Compiling Linux Applications . . . 19

5 Debian image guide **20**
5.1 Prepare the host PC . . . 20
5.2 Compile the ATF . . . 20
5.3 Compile U-Boot . . . 21
5.4 Compile the Linux kernel . . . 22
5.5 Building the debos image . . . 22

6 Building a Yocto image **24**
6.1 Prerequisites . . . 24
6.2 BSP meta layer . . . 24
6.3 Extended meta layer . . . 27

7 Deploy a disk image **30**
7.1 Deploy on SD Card . . . 30
7.2 Deploy on internal eMMC . . . 30

8 Wifi **32**
8.1 Antenna . . . 32
8.2 Connecting to a Wifi network . . . 32
8.3 Flashing the wifi firmware . . . 32

9 Serial Number & MAC Address **34**
9.1 Serial Number . . . 34

9.2 MAC Address . . . 34

10 Mule Companion Controller **35**
10.1 Companion Controller 1 (STM32) . . . 35
10.2 Companion Controller 2 (ATtiny) . . . 36

11 Phosh graphical shell **38**
11.1 Usage . . . 38
11.2 Known issues . . . 39

12 Hardware Guide **40**
12.1 Q7 Implementation . . . 40
12.2 Q7 Connector Pinout . . . 41
12.3 Signal Details . . . 43
12.4 On-board Devices . . . 47
12.5 Wifi and Bluetooth module . . . 48
12.6 USB . . . 50
12.7 Using Qseven Signals as GPIO . . . 51
12.8 Electrical Specification . . . 52
12.9 Mechanical Specification . . . 53

13 Known limitations **54**

14 Contact **55**

15 Revision History **56**

1 Introduction

Congratulations for acquiring our new 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/som-product/ringneck-som-px30-uq7/

1.1 Device Overview

PX30 is a high-performance Quad-core application processor designed for personal mobile internet device and other digital multimedia applications. PX30 is a 64-bit low power processor with Quad Core ARM Cortex A35 and dual core Mali G31 GPU. These 64-bit capable ARMv8 Cortex A35 processors support both the ARM Crypographic Extension (e.g.for wire-rate AES encryption) and AdvSIMD vector processing. The ability to receive camera sensor input through a MIPI-CSI interface and to process the resulting imagestream in real-time with the powerful ARM processor cores enables vision and image-analytics applications.

2 First Steps

This chapter provides instructions for getting the RINGNECK SOM-PX30-uQ7 EVK running after opening the box.

2.1 Insert the Module

Insert the RINGNECK SOM-PX30-uQ7 module at a 30-degree angle into the connector in the base board. Once fully inserted, push it down until it rests on the standoffs and check alignment of the mounting holes.

Note: The module springs back into the 30-degree angle once released. This is expected, and alignment will be kept. The module will be secured into place.

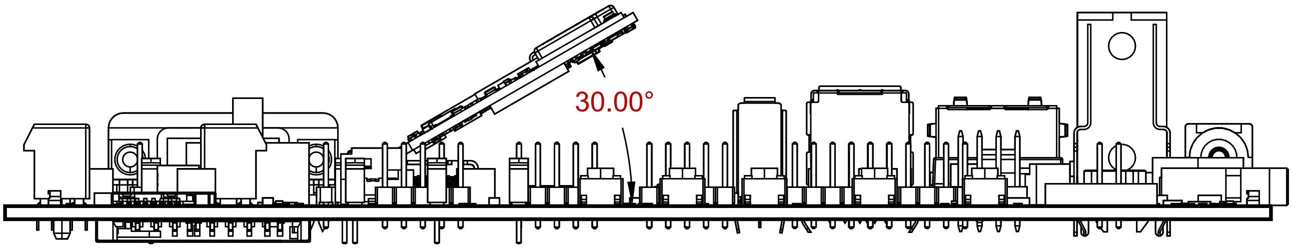

Fig. 2.1: Module mounting

2.2 Mount the Fan (optional)

The fan is only necessary in exceptionally high ambient temperatures. Under normal conditions, the PX30 operates passively cooled.

2.3 Power Up

For bootloader configuration and Linux console, the serial interface can be used. Connect either a Micro-USB or RS-232 cable to the corresponding port. Select the correct UART with UART selector slider (1). For Micro-USB, the slider has to be in the right position to route the default console (UART0) to the USB-UART bridge. For RS-232, the slider has to be in the left position and the protocol slider (2) has to be in the RS-232 position (see Fig. 2.2 *Serial console and boot configuration*).

Connect the power supply and verify the sliders are in the position `Normal Boot` (3) and `Normally Off` (4). Press the `Power Button` (5) to power the board. You will see the boot progress and later on a login prompt on the serial interface. If the display is connected, video output will follow shortly after.

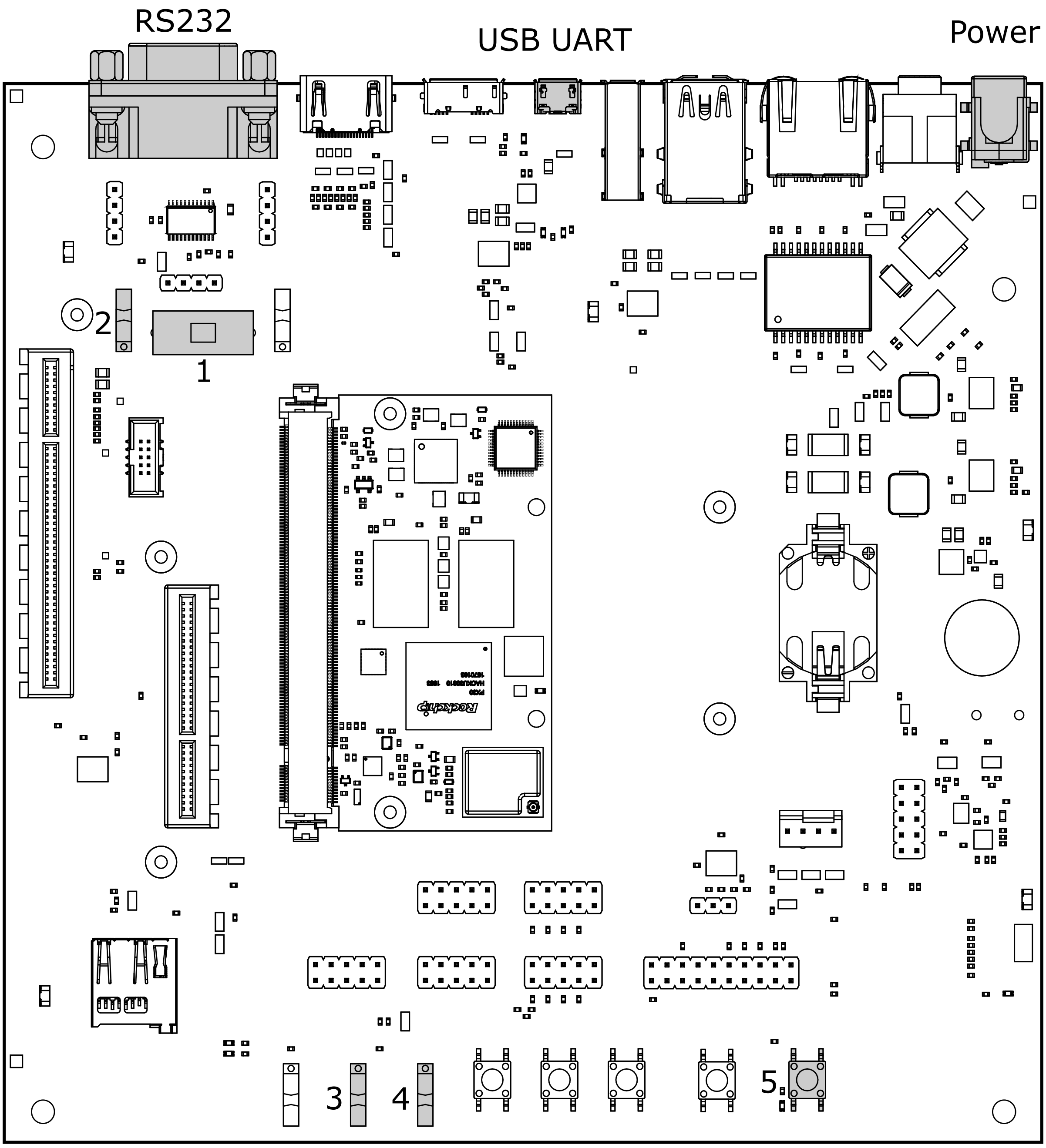

Fig. 2.2: Serial console and boot configuration

3 Using the EVK

This chapter provides instructions for using the EVK (also called Haikou), 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.

Note: The RINGNECK SOM-PX30-uQ7 does not support HDMI and PCIe x4 (they are shown with a * in the next figure).

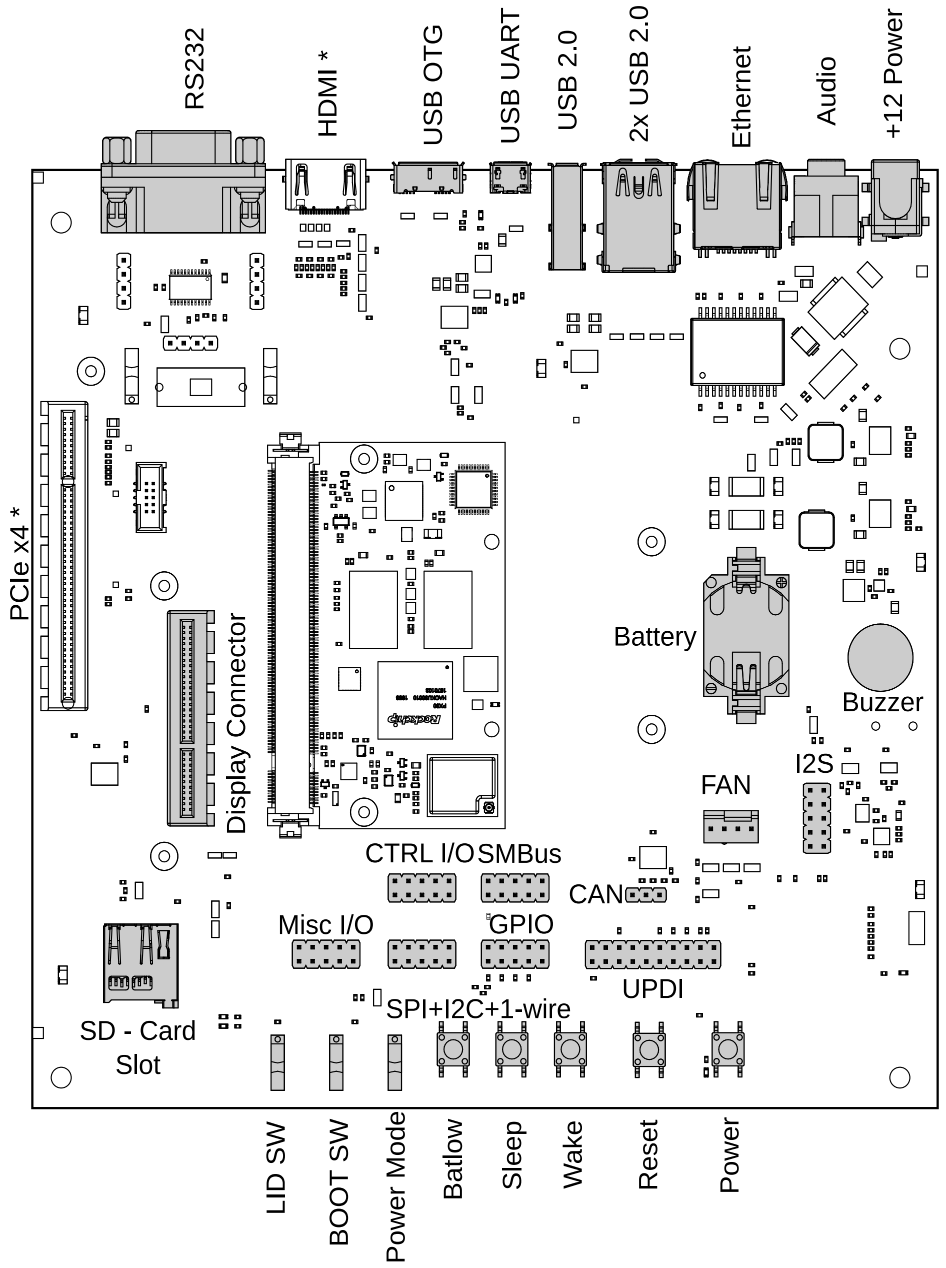

Fig. 3.1: The baseboard with RINGNECK SOM-PX30-uQ7 module
(connectors marked with a * are not supported)

3.2 Power Supply

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

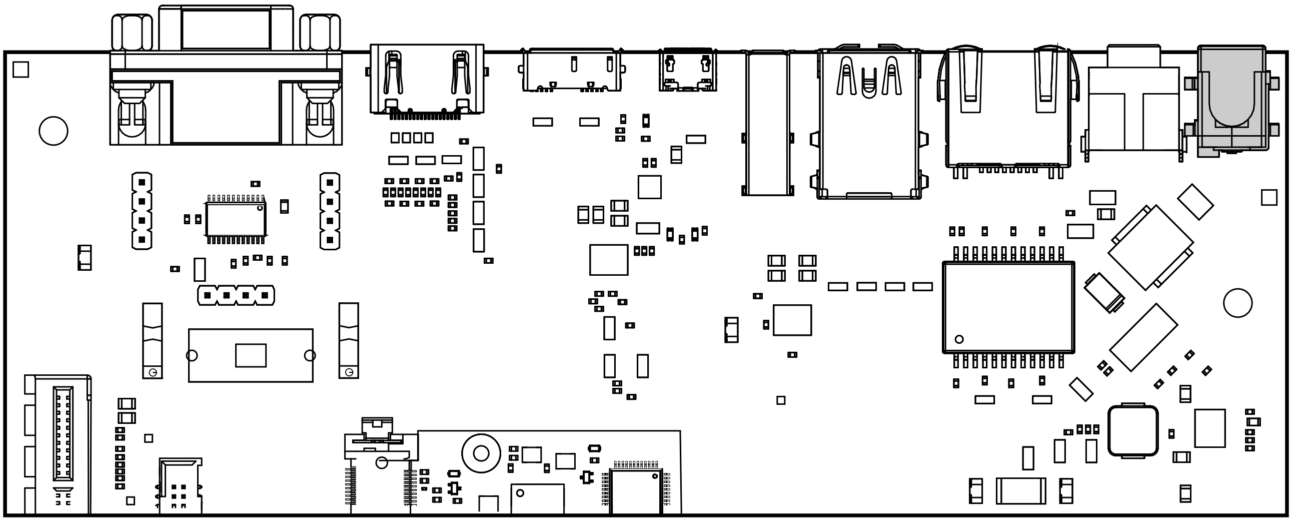

Fig. 3.2: 12V Power connector

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 Section 3.1 *Evaluation Board Overview*).

Depending on the setting of `Power Mode` (`Normally On / Normally Off`) switch, the board will boot as soon as it receives power.

3.3 Control Buttons and Switches

The control buttons (see Fig. 3.1 *The baseboard with RINGNECK SOM-PX30-uQ7 module*) 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 uQ7 module.

Several slider switches are located on the lower left:

- `LID SW` is routed to a GPIO on the module, simulates lid open/close.
- `Power Mode` (`Normally On / Normally Off`), as described above, sets the state after power loss.
- `BOOT SW` (`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.

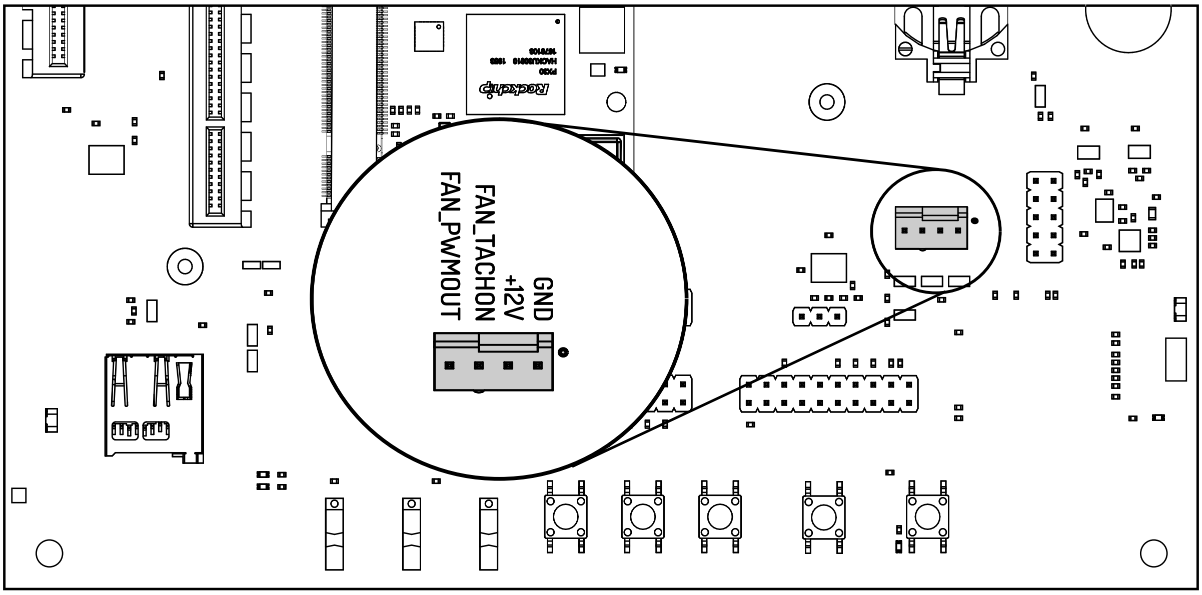

Fig. 3.3: Fan connector

Note: The fan is only necessary in high ambient temperatures. Under normal conditions, the RINGNECK SOM-PX30-uQ7 operates passively cooled.

3.5 Boot Order

The used boot order of the RINGNECK SOM-PX30-uQ7 module depends on the value of the `BIOS_DISABLE#` signal. On the Haikou baseboard, this signal can be set using a slider switch (`BOOT SW`), with the two positions labeled *Normal Boot*, and *BIOS Disable*.

As shown in the table below, the *BIOS Disable* position disables the eMMC storage device:

	Normal Boot	*BIOS Disable*
1	eMMC storage	SD card
2	SD card	USB loader
3	USB loader	

If no bootloader is found on any storage device, the RINGNECK SOM-PX30-uQ7 module will go into USB loader mode, showing up as a USB device on the USB-OTG port.

The electrical state of the `BIOS_DISABLE#` signal for both slider positions is shown below:

Slider Position	`BIOS_DISABLE#` signal
Normal Boot	Floating (on-module pull-up to 3.3V)
BIOS Disable	GND

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`:

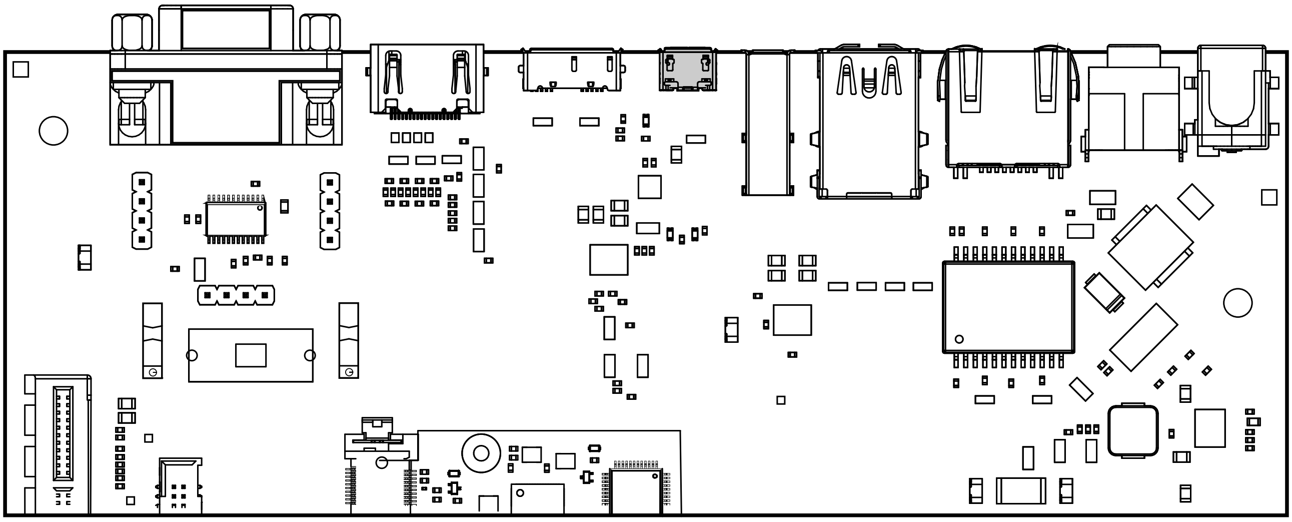

Fig. 3.4: USB UART

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

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

The RINGNECK SOM-PX30-uQ7 module 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.

Note: `UART1` is the name of the UART exposed on the Haikou baseboard. It is actually connected to the `UART5` controller on the PX30 SoC.

Incidentially, `UART0` on Haikou is connected to the `UART0` controller on the PX30 SoC.

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

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

Note: Make sure to disable software flow-control (XON/XOFF). Otherwise serial input may not be recognized.

After system boot-up, the login console appears on the terminal:

```
px30-uq7 login:
```

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

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.

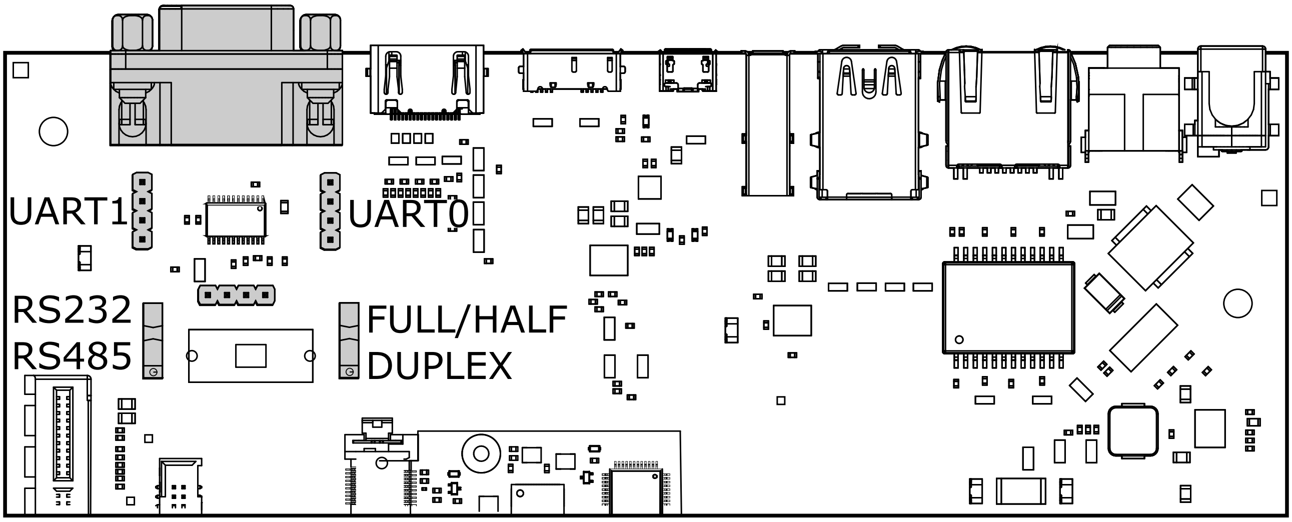

Fig. 3.5: RS-232 connector

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 RINGNECK SOM-PX30-uQ7 has built-in Fast Ethernet (100Mbit/s) routed to a standard RJ-45 jack on the evaluation board.

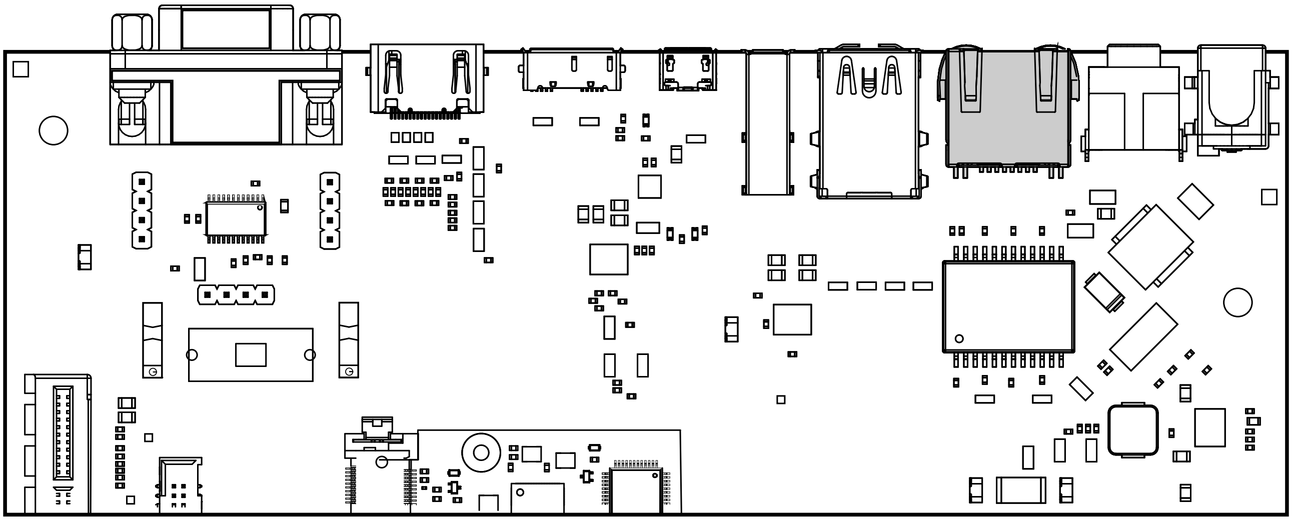

Fig. 3.6: Ethernet jack

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

3.10 SD-Card

The RINGNECK SOM-PX30-uQ7 supports UHS SD cards and maximum writing speed on the SD card is 50MB/s. The practical writing and reading speeds depend on the capabilities of the inserted SD card.

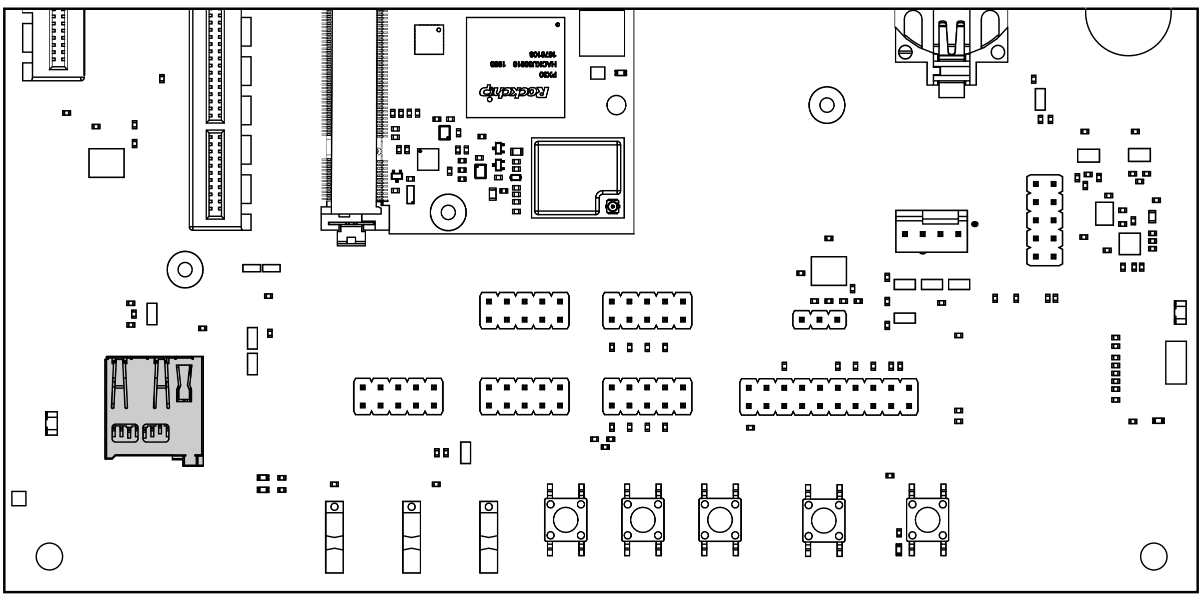

Fig. 3.7: SD card reader

3.11 USB Interfaces

The RINGNECK SOM-PX30-uQ7 provides four USB ports:

- 1x USB 2.0 OTG
- 3x USB 2.0 Host

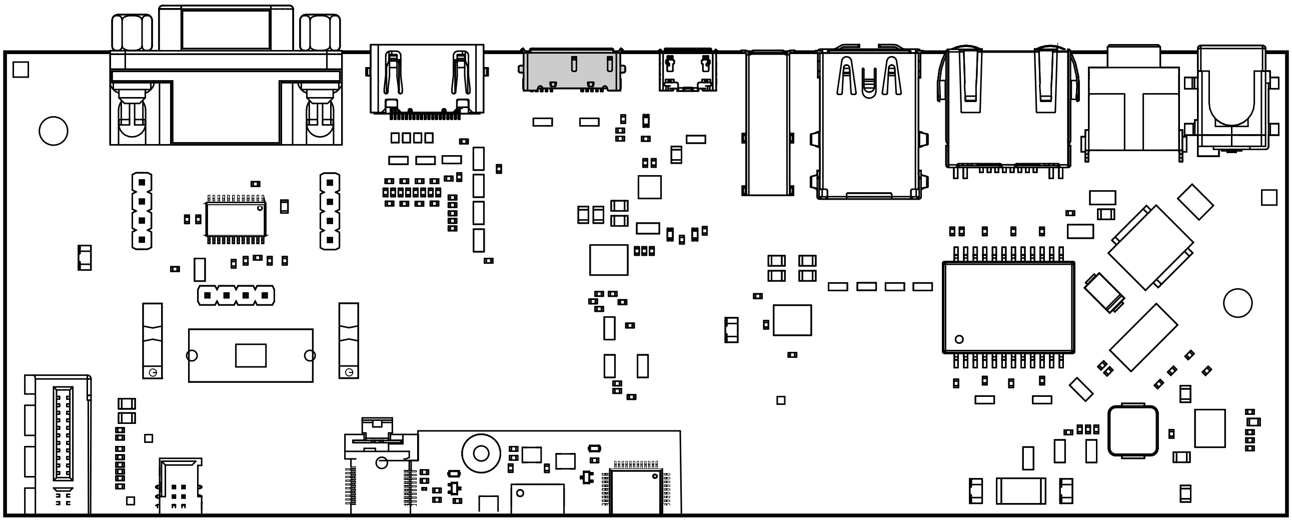

Fig. 3.8: USB 2.0 OTG port (dual-role port: can be used as a host or device interface)

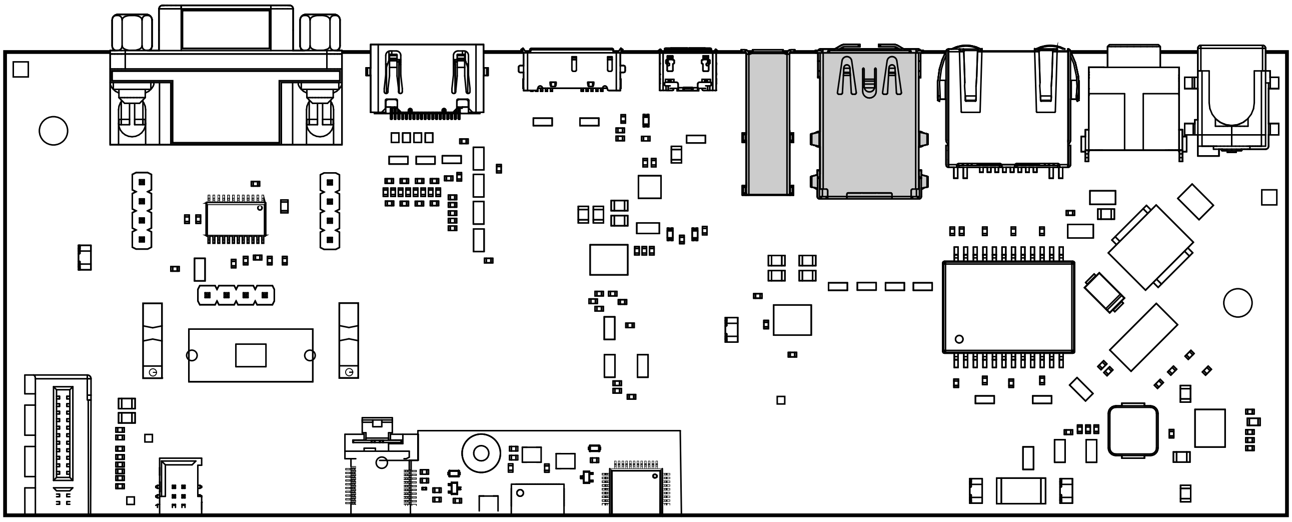

Fig. 3.9: USB 2.0 host ports

3.11.1 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:

```
dmesg -f
```

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

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

3.12 Display and Camera

The RINGNECK SOM-PX30-uQ7 supports display output on the LVDS A interface and the camera on the LVDS B interface.

For MIPI-DSI and MIPI-CSI, the Qseven LVDS pins are used. Those pins are routed to the `Video connector`. This expansion slot uses a PCIe connector as mechanical connection, which allows easy development of adapter boards for various different display types.

Qseven Port	Function	Alternate Function
LVDS A	MIPI-DSI	LVDS
LVDS B	MIPI-CSI	

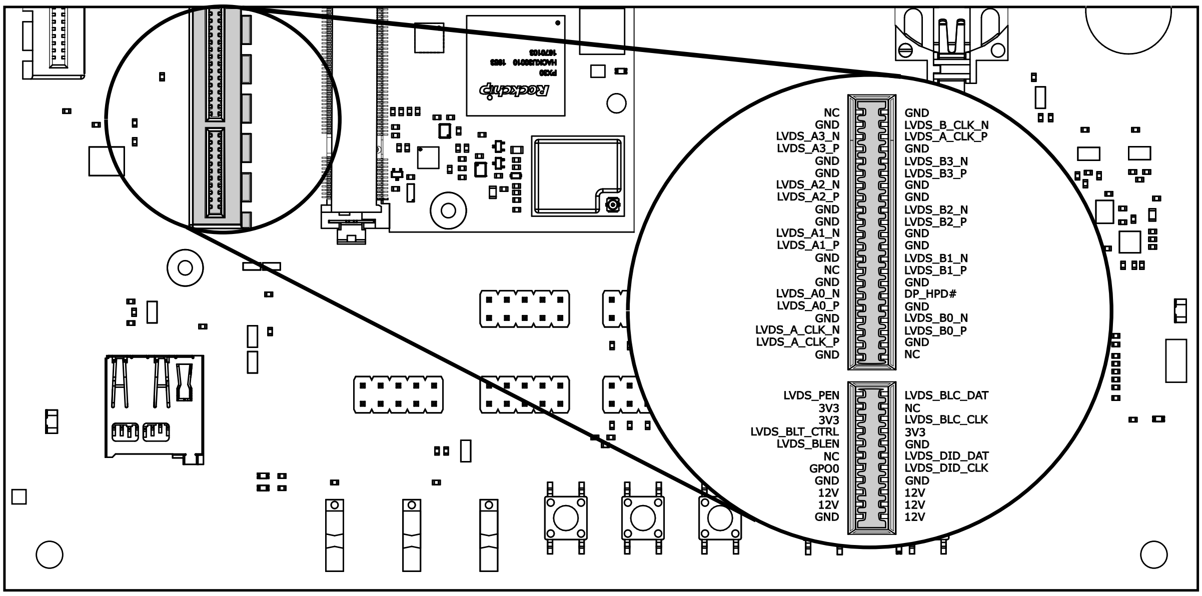

Fig. 3.10: Video connector pinout

The kernel devicetree defines the used display configuration. Example device trees for various output configurations are provided with the EVK software package.

To specify which devicetree should be loaded on boot, edit the configuration variable `FDT` in the file `/boot/extlinux/extlinux.conf`. For example to enable support for the Haikou Video Demo adapter write:

```
FDT /boot/px30-ringneck-haikou-video-demo.dtb
```

Note: For systems using FIT images (such is the case for Yocto images), the `kernel` variable should be edited instead:

```
kernel /fitImage#conf-rockchip_px30-ringneck-haikou-video-demo.dtb
```

Filename	Functions
px30-ringneck-haikou.dtb	
px30-ringneck-haikou-video-demo.dtb	Touchscreen display, camera Requires Video Demo adapter

3.13 RTC

The RINGNECK SOM-PX30-uQ7 contains a real-time clock (RTC) on-module.

Note: This functionality is implemented in the optional Mule companion controller (see Section 12.4.4 *Companion Controller 1* and Section 12.4.5 *Companion Controller 2*).

The RTC is read by the kernel on boot-up and used to set the system clock.

To check the RTC value, use `hwclock`:

```
hwclock
Thu 22 Oct 2022 01:49:20 PM CEST  -0.826662 seconds
```

The RTC will be automatically set 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 2022-10-22
date --set 04:12:33
```

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

```
hwclock -w
```

3.14 SPI and I2C

SPI and I2C interfaces are both available on the pin header labeled `SPI+I2C+1-Wire`. The RINGNECK SOM-PX30-uQ7 does not support `1-Wire`.

Additional I2C buses are available on the `SMBUS` header. Note that `SMB_DAT`, `SMB_CLK`, `SMB_ALERT#` are not supported by the RINGNECK SOM-PX30-uQ7 module (shown in thin font in Fig. 3.12).

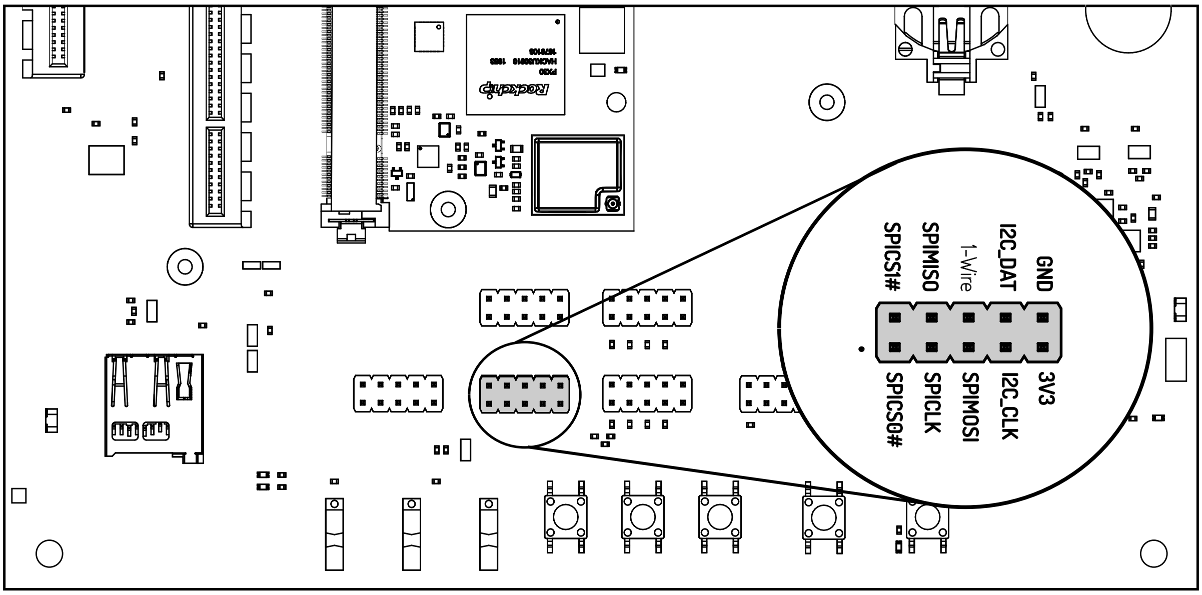

Fig. 3.11: I2C and SPI header

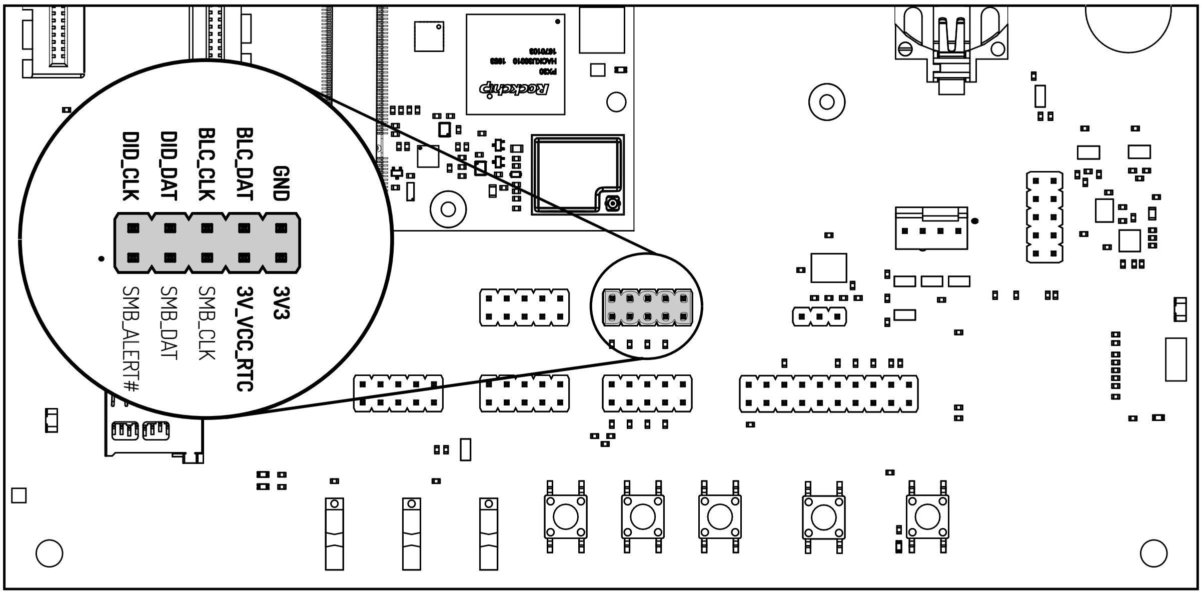

Fig. 3.12: SMBUS header

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

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

3.14.1 Linux I2C Bus Numbering

Linux identifies each I2C bus by a bus number. The table below shows the mapping between Q7 names, Linux bus number and EVK header.

Q7 signals	Linux bus #	Haikou Header(s)	Label on Haikou Header
GP2_I2C_DAT/LVDS_DID_DAT GP2_I2C_CLK/LVDS_DID_CLK	1	`SMBus & Video connector`	`DID_DAT` `DID_CLK`
GP0_I2C_DAT GP0_I2C_CLK	2	`SPI+I2C+1-Wire`	`I2C_DAT` `I2C_CLK`
eDP0_HPD#/LVDS_BLC_DAT eDP1_HPD#/LVDS_BLC_CLK	3	`SMBus & Video connector`	`BLC_DAT` `BLC_CLK`

The other I2C buses (as reported by `i2cdetect -l`) are internal to the module and not routed to the Q7 connector.

3.15 GPIOs

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

The location on the board is displayed below:

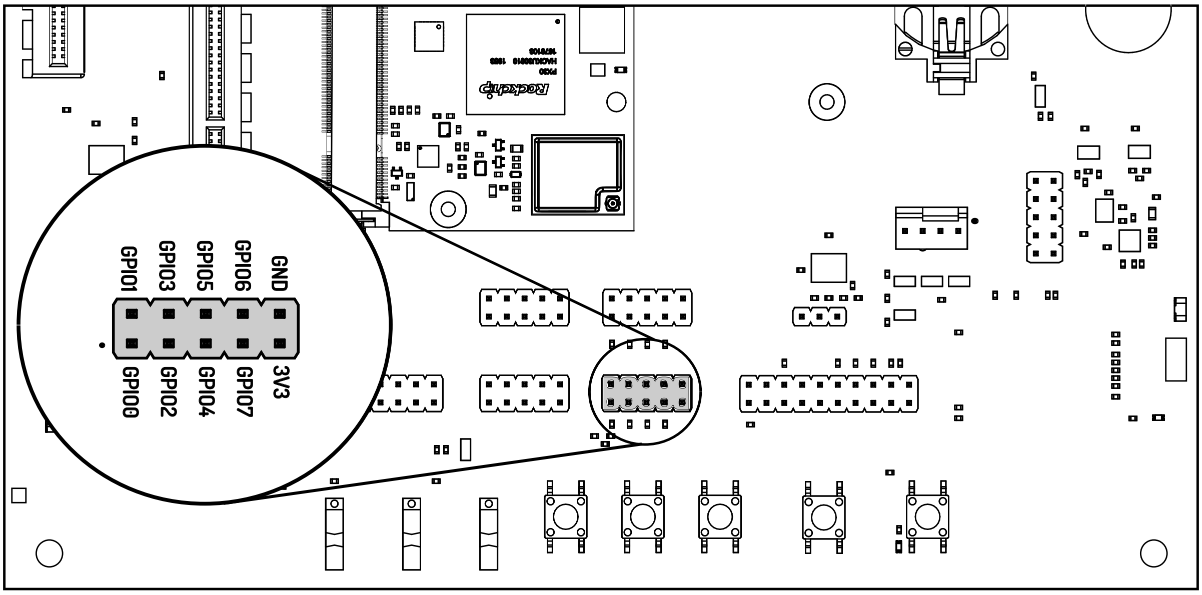

Fig. 3.13: GPIO header

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:

Q7 signal	CPU pin	Linux GPIO #
GPIO0	GPIO3_C6	118
GPIO1	GPIO3_D0	120
GPIO2	GPIO3_C7	119
GPIO3	GPIO3_D1	121
GPIO4	GPIO3_C0	112
GPIO5	GPIO3_A2	98
GPIO6	GPIO3_A1	97
GPIO7	GPIO2_B6	78

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:

```
GPIO3_C6 -> (3 * 32) + (2 * 8) + 6 = 118
```

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

```
$ echo 118 > /sys/class/gpio/export
$ cat /sys/class/gpio/gpio118/direction
in
$ cat /sys/class/gpio/gpio118/value
0
```

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

```
$ echo out > /sys/class/gpio/gpio118/direction
$ echo 1 > /sys/class/gpio/gpio118/value
```

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

3.16 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.

Note: The codec on the Haikou baseboard only supports a sample rate 48kHz . This restriction only applies to this specific codec on the Haikou baseboard.

The I2S bus on the RINGNECK SOM-PX30-uQ7 module supports a sample rate up to 192kHz.

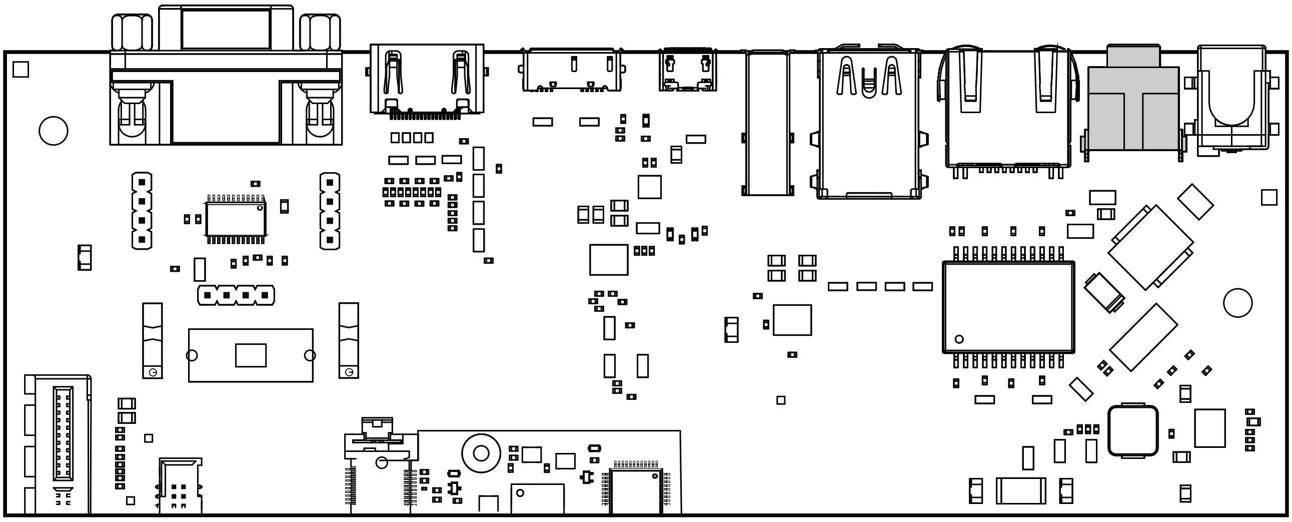

Fig. 3.14: Audio input/output port

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

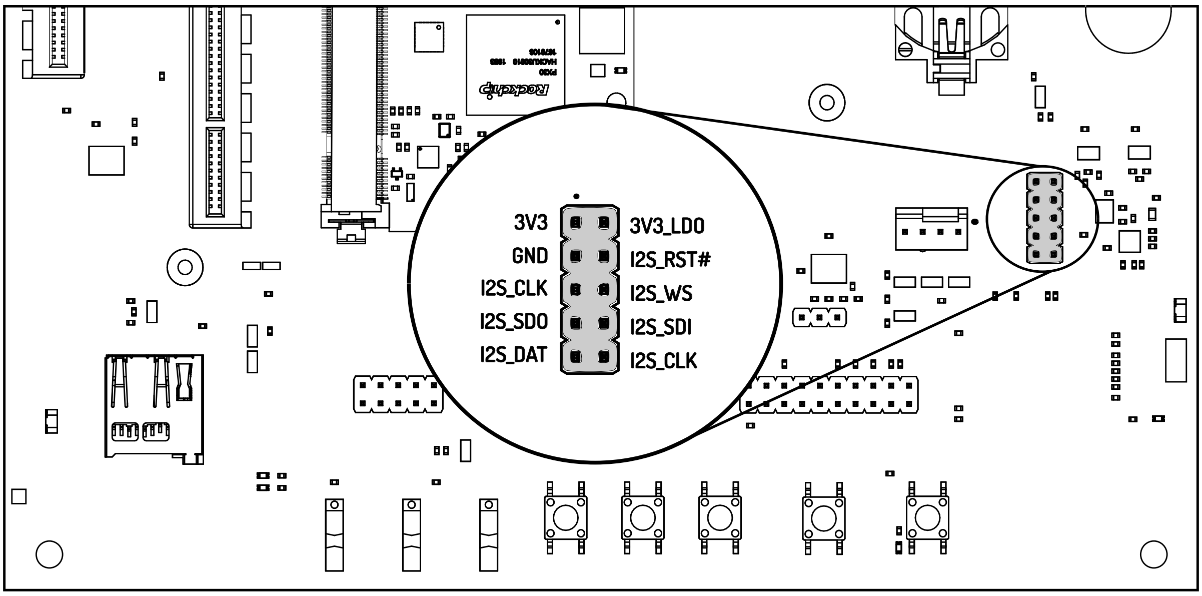

Fig. 3.15: Connecting to the audio expansion connector

3.17 CAN Bus

The board provides a CAN connector on the bottom row.

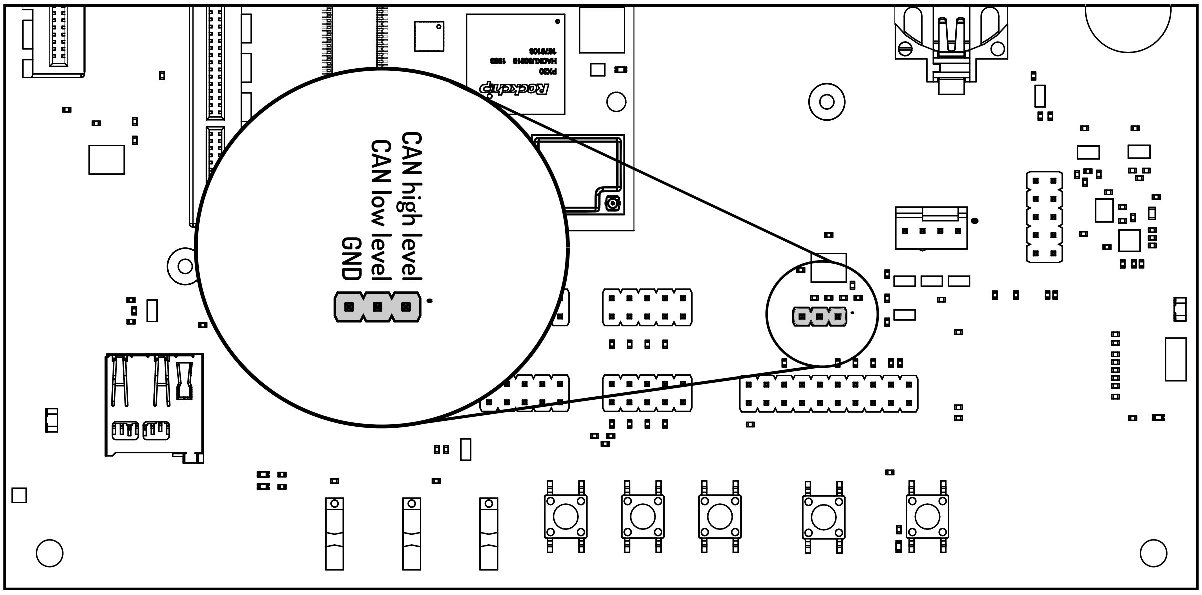

Fig. 3.16: CAN header

Note: CAN feature is only available on RINGNECK SOM-PX30-uQ7 module with an STM32, see (Section 12.4.4 *Companion Controller 1*).

3.18 CTRL I/O Connector

The board provides signals for watchdog trigger in- and output, SoM PMIC power-on input, reset and external display power enable.

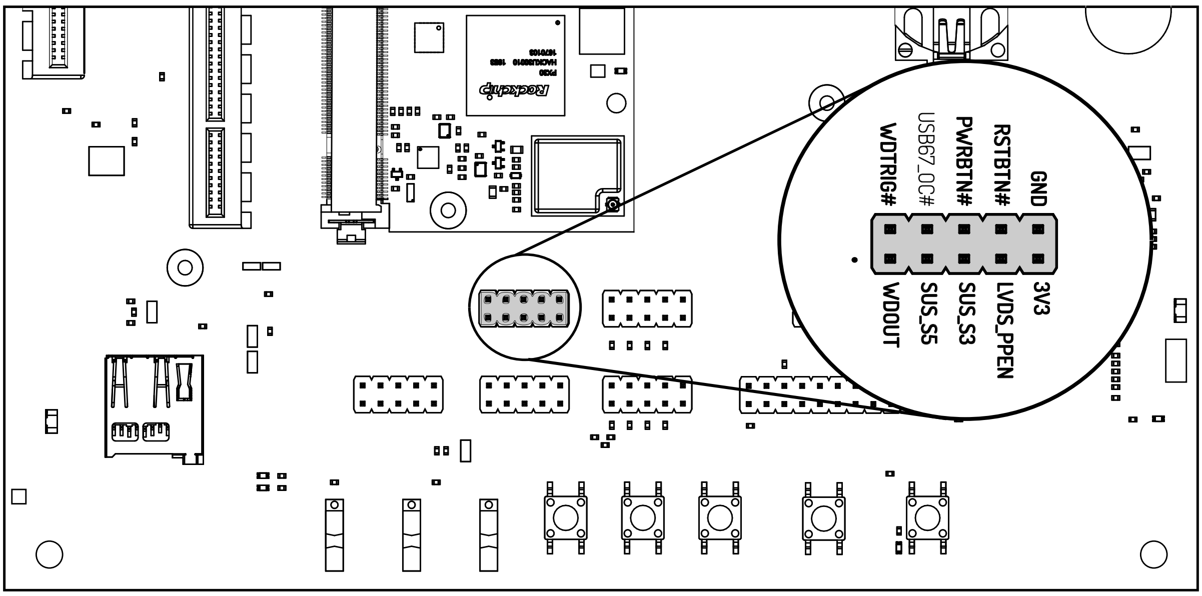

Fig. 3.17: CTRL I/O header

3.19 MISC Connector

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

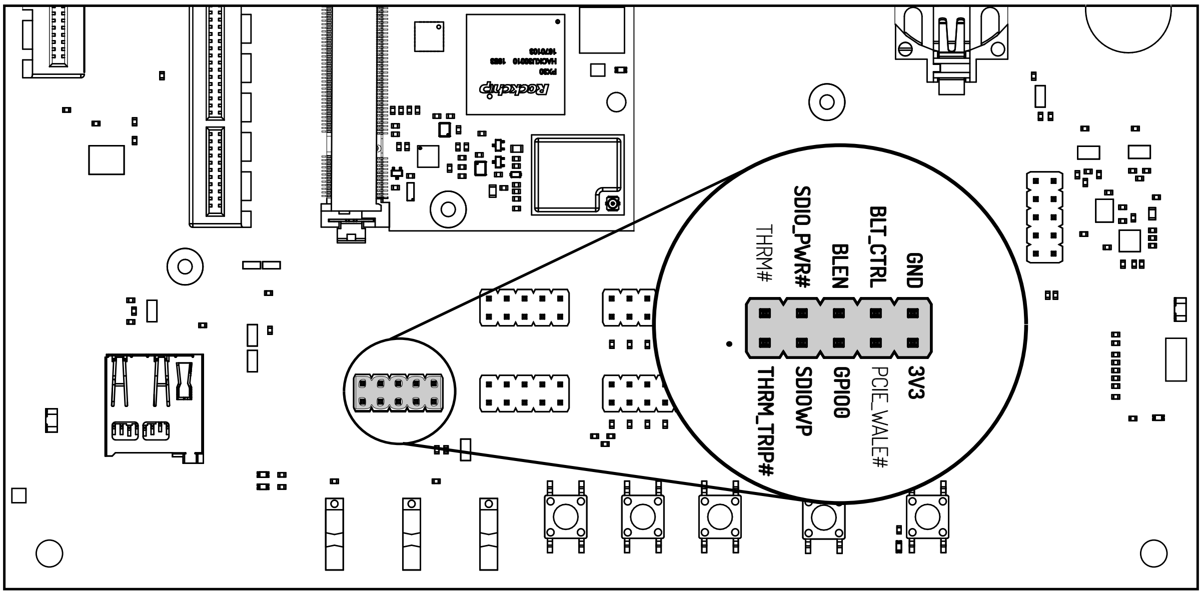

Fig. 3.18: MISC header

3.20 JTAG Connector

The board provides UPDI signals on the JTAG connector. The RINGNECK SOM-PX30-uQ7 does not support JTAG, but the ATtiny (see Section 12.4.5 *Companion Controller 2*) can be flashed over JTAG connector pins.

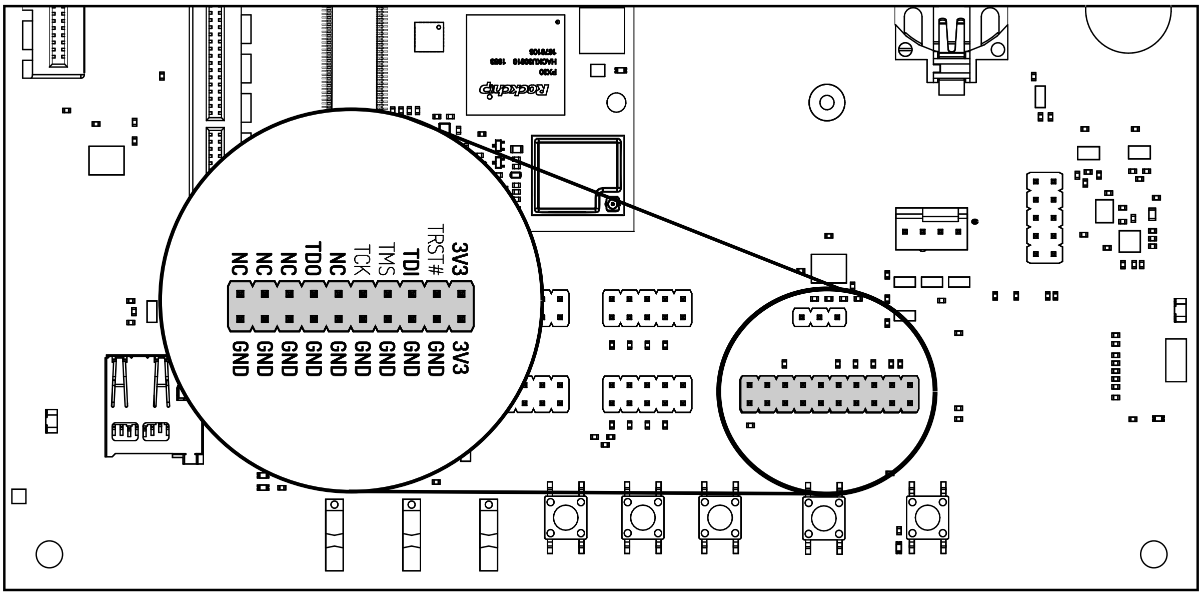

Fig. 3.19: JTAG header

JTAG header	Function
TDI	UPDI-TX
TDO	UPDI-RX

4 Software Overview

This chapter provides instructions for compiling and deploying the BSP (Board Support Package) software to the uQ7 module.

4.1 Supported Distributions

Two of the most popular embedded systems distributions are supported. The following chapters describe how to build a disk image for each of them:

- Debian: Section 5 *Debian image guide*
- Yocto: Section 6 *Building a Yocto image*

4.2 Compiling Linux Applications

The easiest option is to compile your applications directly on a module running Debian. 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 on a host PC. The compiler that was installed in Section 5.1 *Prepare the host PC* is suitable.

5 Debian image guide

As opposed to Yocto, Debian does not provide a completely integrated build experience by itself. Linux kernel and U-Boot have to be compiled manually and copied to the appropriate directory to be picked up by Debian build system.

This chapter will go through all neccessary steps, finally building a complete image using the *debos Debian image builder*. The result will be a fully-functional *Debian* system.

Alternatively, prebuilt images can be downloaded from https://downloads.theobroma-systems.com/ringneck/.

At the time of writing this document, the following Debian image variants are available for the Ringneck board:

- Debian 12 Bookworm,
- Debian 12 Bookworm with *Phosh* graphical shell.

Note: While Debian is a great tool for fast prototyping of your product, it is highly recommended to use a distribution/image tailored to your need. This can be achieved by Yocto (Section 6 *Building a Yocto image*) or Buildroot for example.

5.1 Prepare the host PC

The *debos Debian OS Builder* is only available for Debian and Debian-based distributions (like Ubuntu). This chapter assumes you use Debian or a Debian-based distribution as the host PC.

Install packages for compiling the parts and the complete image:

```
sudo apt-get -y install debos git build-essential gcc-aarch64-linux-gnu make bison bc flex \
  libssl-dev device-tree-compiler python3-dev python3-pkg-resources swig fdisk bmap-tools
```

As *debos* internally uses kvm virtualization, your user must be a member of the `kvm` group:

```
sudo adduser $(id -un) kvm
```

Log out and back for the change to take affect. Then verify that `kvm` is listed in your groups:

```
id -Gn
```

5.2 Compile the ATF

Get the source code and compile the Arm Trusted Firmware as follows:

```
# Set up cross-compilation
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-

# Download the source code
git clone https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git
cd trusted-firmware-a

# Use most recent release
LAST_ANNOTATED_V_TAG_FROM_MASTER=$(git describe --abbrev=0 --match "v?*.?*.?*")
# Find all tags from different branches that contain that last tag reachable from master.
# This will return lts tags, of which we want to take the latest available.
# If no LTS tag, take the latest non-rc tag reachable from master.
LAST_LTS_TAG=$(git tag --sort -version:refname --contains "$LAST_ANNOTATED_V_TAG_FROM_MASTER
```

(continues on next page)

(continued from previous page)

```
↪" 'lts-v?*.?*.?*' | head -1)
TAG=${LAST_LTS_TAG:-$LAST_ANNOTATED_V_TAG_FROM_MASTER}
git checkout "$TAG"

TF_LDFLAGS=""

# Fix for aarch64-linux-gnu-ld 2.39+ and TF-A prior to v2.9
# Fix available in commit 1f49db5f25cd ("feat(build): add support for new binutils versions
↪")
if "${CROSS_COMPILE}ld" --no-warn-rwx-segments -v >/dev/null 2>&1; then
        TF_LDFLAGS="-z noexecstack --no-warn-rwx-segments"
fi

# Compile
TF_LDFLAGS="$TF_LDFLAGS" make PLAT=px30 bl31

# Make the resulting file available to later steps
export BL31=$PWD/build/px30/release/bl31/bl31.elf

cd ..
```

This step should take under 1 minute total.

5.3 Compile U-Boot

Note: The variable BL31 must be already set as described in Section 5.2 *Compile the ATF* .

Get the source code and compile the U-Boot bootloader as follows:

```
# Set up cross-compilation
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-

# Download the source code
git clone https://git.theobroma-systems.com/ringneck-u-boot.git
cd ringneck-u-boot

# Compile
make ringneck-px30_defconfig
make -j$(nproc)

# Make the resulting file available to later steps
export RINGNECK_UBOOT_DIR=$PWD

cd ..
```

This step should take about 1 minute total.

5.4 Compile the Linux kernel

Get the source code and compile the Linux kernel as follows:

```
# Set up cross-compilation
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-

# Download the source code
git clone https://git.theobroma-systems.com/ringneck-linux.git
cd ringneck-linux

# Compile
make ringneck-px30_defconfig
make -j$(nproc)

# Make the resulting files available to later steps
export RINGNECK_LINUX_DIR=$PWD

cd ..
```

The time required for this step heavily depends on your internet connection and CPU power. On a quad-core 2.9GHz machine with an 1Gb/s internet connection, it takes about 20 minutes total.

5.5 Building the debos image

5.5.1 Prepare required components

Note: The variables `RINGNECK_UBOOT_DIR` and `RINGNECK_LINUX_DIR` must be already set as described in Section 5.3 *Compile U-Boot* and Section 5.4 *Compile the Linux kernel*, respectively.

Get the source code for the *debos* recipe and copy necessary components built in previous steps:

```
# Download the source code
git clone https://git.theobroma-systems.com/debos-recipes.git
cd debos-recipes

# Copy Linux & U-Boot binaries into the ``ringneck`` folder
cp $RINGNECK_LINUX_DIR/arch/arm64/boot/Image ringneck/overlay/boot/
cp $RINGNECK_LINUX_DIR/arch/arm64/boot/dts/rockchip/px30-ringneck*.dtb ringneck/overlay/boot
cp $RINGNECK_UBOOT_DIR/u-boot-rockchip.bin ringneck
```

5.5.2 Build a complete image

Different variants of Debian images are available. You can build the one of your choice or all of them. Default variant is *Debian 12 Bookworm*. Other variants can be chosen by setting the `debos_variant` environment variable when running `build.sh`.

Depending on your host PC and internet connection, this step should complete in about 5-10 minutes.

The resulting image is a file called `sdcard-ringneck-debos-VARIANT.XXX.YYY.img` and, for convenience, the symlink `sdcard-ringneck-debos-VARIANT.img` that always points to the latest version.

Debian 12 Bookworm

```
# Build the image
build_board=ringneck ./build.sh

# Or: Build the image using podman (For host PCs not using Debian)
# build_board=ringneck debos_host=podman ./build.sh
#
# Make the resulting image available to later steps
export SDCARD_IMG=$PWD/sdcard-ringneck-debos-bookworm.img
```

Note: When running inside a virtual machine that does not support nesting, you may get an error like this:

```
open /dev/kvm: no such file or directory
```

In this case, prepend `debos_host=chroot` to the `build.sh` command, resulting in:

```
debos_host=chroot build_board=ringneck ./build.sh
```

The `debos_host=chroot` mode uses `sudo` internally as it requires root permissions.

Debian 12 Bookworm with Phosh graphical shell

This image variant is targeted for the Haikou-Video-Demo. Please see the *AN60501 Haikou-Video-Demo for the Haikou baseboard* application note for more information about the Haikou-Video-Demo.

More details about the Phosh graphical shell can be found in the *Phosh graphical shell* section.

```
# Build the image
build_board=ringneck debos_variant=bookworm-phosh ./build.sh

# Or: Build the image using podman (For host PCs not using Debian)
# build_board=ringneck debos_variant=bookworm-phosh  debos_host=podman ./build.sh

# Make the resulting image available to later steps
export SDCARD_IMG=$PWD/sdcard-ringneck-debos-bookworm-phosh.img
```

Note: When running inside a virtual machine that does not support nesting, you may get an error like this:

```
open /dev/kvm: no such file or directory
```

In this case, prepend `debos_host=chroot` to the `build.sh` command, resulting in:

```
debos_host=chroot build_board=ringneck ./build.sh
```

The `debos_host=chroot` mode uses `sudo` internally as it requires root permissions.

---

6 Building a Yocto image

The Yocto Project is an open-source project that helps building Linux-based distributions, mainly for embededded products. Theobroma Systems provides a minimal BSP layer to allow building Yocto images for the company's modules. An extended layer is also provided for a less bare experience, see instructions in Section 6.3 *Extended meta layer*. Upon request, access can be given to a more featureful "demonstration" layer which provides hardware and software validation scripts as well as demo applications.

This user guide does not aim at getting the user familiar with development with the Yocto Project but rather help them setup their build environment to create a basic Yocto image that can be used on one of Theobroma Systems's modules.

The Yocto project provides an open source Linux build framework, which allows to create customized build environments for embedded systems.

Yocto consists of the following parts:

- The Yocto Project tools,
- Reference Linux distribution (Poky),
- Build system (co-maintained with OpenEmbedded),

There exists extensive documentation for the Yocto Project and BitBake.

The Yocto Project releases a new version twice a year and some versions are maintained for a longer time when marked as LTS (Long-Term Support). Such is the case of Kirkstone (4.0), supported until at least April 2024. Theobroma Systems highly recommend to use LTS versions and update to a newer version once its support has reached end-of-life, to benefit from bug fixes, security fixes, miscellaneous improvements and additional features.

6.1 Prerequisites

While the Yocto Project supports mainy different build systems, Theobroma Systems currently only tests building on Debian 11 (Bullseye).

The required packages for Debian are listed in the documentation and can be installed with the following command:

```
sudo apt-get install -y --no-install-recommends gawk wget git diffstat unzip \
 texinfo gcc build-essential chrpath socat cpio python3 python3-pip python3-venv \
 python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 \
 libegl1-mesa libsdl1.2-dev xterm python3-subunit mesa-common-dev zstd \
 liblz4-tool file
```

6.2 BSP meta layer

The Yocto Project BSP meta layer can be found at https://git.theobroma-systems.com/yocto-layers/meta-theobroma-systems-bsp.git/.

It contains the minimal configuration and recipe append files (bbappend) necessary to build a minimal working image. It is meant to be a base upon which to build and thus many tools are purposefully missing.

6.2.1 Initial setup

Clone the BSP meta layer and its dependencies from a new directory called `yocto`:

```
mkdir yocto
cd yocto
git clone https://git.theobroma-systems.com/yocto-layers/meta-theobroma-systems-bsp.git -b␣
↪kirkstone
git clone https://git.yoctoproject.org/poky -b kirkstone-4.0.10
git clone https://git.yoctoproject.org/meta-arm -b yocto-4.0.2
git clone https://git.yoctoproject.org/meta-rockchip -b kirkstone
git clone https://git.openembedded.org/meta-openembedded -b kirkstone
```

The following directory layout should be observed:

```
yocto
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
└── poky
```

Note: It is essential that the Yocto layers are checked out on a branch that supports the same release as the others, otherwise there may be some unexpected issues. With the aforementioned instructions, the layers have been checked out to a branch supporting the Yocto Project Kirkstone (4.0) release.

One can check if a branch supports a release by looking into `conf/layer.conf` and look for the `LAYERSERIES_COMPAT_*` variable. All layers should have the same one in common, here "kirkstone".

6.2.2 Initializing build environment

Once the layers have been properly cloned in their appropriate branch, the build environment needs to be initialized. This can be done by running the following command:

```
source poky/oe-init-build-env build
```

This will initialize the build environment by making the `bitbake` build tool available in the current shell and creating a `build` directory where temporary and final build artifacts will be stored.

The following directory layout should be observed:

```
yocto
├── build
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
└── poky
```

The first time the command is run, it'll create a new build directory called `build` and add the appropriate configuration files. On the later runs, if the directory still exists, the command will only configure the terminal environment and not change anything in the build directory. This makes it perfectly safe to run the command multiple times, from different terminals for example.

Note: Once the current terminal is closed or a new one is opened, this command should be re-executed to be able to interact again with the Yocto Project tools.

The Yocto Project then needs to be configured to include layers to find new recipes or configuration files, which is essential to build new pieces of software or compile for a specific hardware target system.

This can be done with the `bitbake-layers` tool:

```
bitbake-layers add-layer ../meta-arm/meta-arm-toolchain
bitbake-layers add-layer ../meta-arm/meta-arm
bitbake-layers add-layer ../meta-rockchip
bitbake-layers add-layer ../meta-openembedded/meta-oe
bitbake-layers add-layer ../meta-openembedded/meta-python
bitbake-layers add-layer ../meta-theobroma-systems-bsp
```

6.2.3 Building a minimal image

To build a bootable artifact, BitBake will be called with the specified machine and target image:

```
MACHINE="ringneck-haikou" bitbake core-image-minimal
```

Note: Technically speaking, the `MACHINE` variable could be set in `build/conf/local.conf` file once and for all. If possible, Theobroma Systems recommends passing the variable explicitly in the command directly as this makes it more visible to the user and also allows to easily build for multiple machines without modifying a file in-between.

The build process can take several hours depending on the capabilities of the build machine and the user's Internet connection.

Note: If the Bitbake process needs to be stopped for any reason, a `SIGINT` (`Ctrl` + `c`) signal can be sent **once**. Bitbake will gracefully close down upon reception of this signal. This graceful shutdown can take a lot of time depending on the tasks that are currently being executed. It is **highly** recommended to not send this signal more than once, failing to do so may hinder next Bitbake commands.

The artifacts can be found after some time in `build/tmp/deploy/images/ringneck-haikou/` directory. A flashable image is one whose extension is `.wic`, e.g. `core-image-minimal-ringneck-haikou-20221021134027.rootfs.wic`.

Make the resulting image available for later steps:

```
export SDCARD_IMG=$PWD/build/tmp/deploy/images/ringneck-haikou/core-image-minimal-ringneck-
↪haikou.wic
```

6.2.4 Building with `kas`

kas is a setup tool for Bitbake-based projects, such as the Yocto Project, which aims to replace the commands listed above for a simpler, more automated, setup and creation of images.

Theobroma Systems provides a `kas` configuration file `kas-theobroma.yml` in the BSP meta layer for convenience.

`kas` can be installed on the build machine with the following command:

```
sudo apt-get install -y --no-install-recommends kas
```

Note: It is also available as a Python package and installable with:

```
python3 -m venv venv
source venv/bin/activate
python3 -m pip install kas==4.0
```

The Section 6.2.1 *Initial setup* and Section 6.2.2 *Initializing build environment* can then be replaced by the following two commands:

```
mkdir yocto
cd yocto
git clone https://git.theobroma-systems.com/yocto-layers/meta-theobroma-systems-bsp.git -b␣
↪kirkstone
kas checkout meta-theobroma-systems-bsp/kas-theobroma.yml
```

The Section 6.2.3 *Building a minimal image* can now be replaced with:

```
KAS_MACHINE="ringneck-haikou" kas build meta-theobroma-systems-bsp/kas-theobroma.yml
```

Note: `kas` is also available in an OCI container form on GitHub container registry.

It is still recommended to install `kas` through `pip` but then use its `kas-container` wrapper script to start the container properly. E.g. to replace the last command to build an image with `kas` one can call this instead:

```
python3 -m venv venv
source venv/bin/activate
python3 -m pip install kas==4.0
KAS_IMAGE_VERSION="4.0" KAS_MACHINE="ringneck-haikou" kas-container build meta-theobroma-
↪systems-bsp/kas-theobroma.yml
```

6.3 Extended meta layer

The Yocto Project extended layer can be found at https://git.theobroma-systems.com/yocto-layers/meta-theobroma-systems-extended.git/.

In addition to the minimal features, this layer includes the network manager, and many more features will be added soon.

6.3.1 Initial setup

Clone the Extended layer and its dependencies from a new directory called `yocto`:

```
mkdir yocto
cd yocto
git clone https://git.theobroma-systems.com/yocto-layers/meta-theobroma-systems-extended.
↪git -b kirkstone
git clone https://git.theobroma-systems.com/yocto-layers/meta-theobroma-systems-bsp.git -b␣
↪kirkstone
git clone https://git.yoctoproject.org/poky -b kirkstone-4.0.10
git clone https://git.yoctoproject.org/meta-arm -b yocto-4.0.2
git clone https://git.yoctoproject.org/meta-rockchip -b kirkstone
git clone https://git.openembedded.org/meta-openembedded -b kirkstone
```

The following directory layout should be observed:

```
yocto
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
├── meta-theobroma-systems-extended
└── poky
```

Note: It is essential that the Yocto layers are checked out on a branch that supports the same release as the others, otherwise there may be some unexpected issues. With the aforementioned instructions, the layers have been checked out to a branch supporting the Yocto Project Kirkstone (4.0) release.

One can check if a branch supports a release by looking into `conf/layer.conf` and look for the `LAYERSERIES_COMPAT_*` variable. All layers should have the same one in common, here "kirkstone".

6.3.2 Initializing build environment

Once the layers have been properly cloned in their appropriate branch, the build environment needs to be initialized. This can be done by running the following command:

```
source poky/oe-init-build-env build
```

This will initialize the build environment by making the `bitbake` build tool available in the current shell and creating a `build` directory where temporary and final build artifacts will be stored.

The following directory layout should be observed:

```
yocto
├── build
├── meta-arm
├── meta-openembedded
├── meta-rockchip
├── meta-theobroma-systems-bsp
├── meta-theobroma-systems-extended
└── poky
```

The first time the command is run, it'll create a new build directory called `build` and add the appropriate configuration files. On the later runs, if the directory still exists, the command will only configure the terminal environment and not change anything in the build directory. This makes it perfectly safe to run the command multiple times, from different terminals for example.

Note: Once the current terminal is closed or a new one is opened, this command should be re-executed to be able to interact again with the Yocto Project tools.

The Yocto Project then needs to be configured to include layers to find new recipes or configuration files, which is essential to build new pieces of software or compile for a specific hardware target system.

This can be done with the `bitbake-layers` tool:

```
bitbake-layers add-layer ../meta-arm/meta-arm-toolchain
bitbake-layers add-layer ../meta-arm/meta-arm
bitbake-layers add-layer ../meta-rockchip
bitbake-layers add-layer ../meta-openembedded/meta-oe
bitbake-layers add-layer ../meta-openembedded/meta-python
bitbake-layers add-layer ../meta-openembedded/meta-networking
bitbake-layers add-layer ../meta-theobroma-systems-bsp
bitbake-layers add-layer ../meta-theobroma-systems-extended
```

6.3.3 Building an image

To build a bootable artifact, BitBake will be called with the specified machine and target image:

```
MACHINE="ringneck-haikou" bitbake theobroma-extended-image
```

Note: Technically speaking, the `MACHINE` variable could be set in `build/conf/local.conf` file once and for all. If possible, Theobroma Systems recommends passing the variable explicitly in the command directly as this makes it more visible to the user and also allows to easily build for multiple machines without modifying a file in-between.

The build process can take several hours depending on the capabilities of the build machine and the user's Internet connection.

Note: If the Bitbake process needs to be stopped for any reason, a `SIGINT` (`Ctrl` + `c`) signal can be sent **once**. Bitbake will gracefully close down upon reception of this signal. This graceful shutdown can take a lot of time depending on the tasks that are currently being executed. It is **highly** recommended to not send this signal more than once, failing to do so may hinder next Bitbake commands.

The artifacts can be found after some time in `build/tmp/deploy/images/ringneck-haikou/` directory. A flashable image is one whose extension is `.wic`, e.g. `theobroma-extended-image-ringneck-haikou-20221021134027.rootfs.wic`.

Make the resulting image available for later steps:

```
export SDCARD_IMG=$PWD/build/tmp/deploy/images/ringneck-haikou/theobroma-extended-image-
↪ringneck-haikou.wic
```

6.3.4 Building with `kas`

kas is a setup tool for Bitbake-based projects, such as the Yocto Project, which aims to replace the commands listed above for a simpler, more automated, setup and creation of images.

Theobroma Systems provides a `kas` configuration file `kas-theobroma.yml` in the BSP meta layer for convenience.

`kas` can be installed on the build machine with the following command:

```
sudo apt-get install -y --no-install-recommends kas
```

Note: It is also available as a Python package and installable with:

```
python3 -m venv venv
source venv/bin/activate
python3 -m pip install kas==4.0
```

The Section 6.3.1 *Initial setup* and Section 6.3.2 *Initializing build environment* can then be replaced by the following two commands:

```
mkdir yocto
cd yocto
git clone https://git.theobroma-systems.com/yocto-layers/meta-theobroma-systems-extended.
↪git -b kirkstone
kas checkout meta-theobroma-systems-extended/kas-theobroma.yml
```

The Section 6.3.3 *Building an image* can now be replaced with:

```
KAS_MACHINE="ringneck-haikou" kas build meta-theobroma-systems-extended/kas-theobroma.yml
```

Note: `kas` is also available in an OCI container form on GitHub container registry.

It is still recommended to install `kas` through `pip` but then use its `kas-container` wrapper script to start the container properly. E.g. to replace the last command to build an image with `kas` one can call this instead:

```
python3 -m venv venv
source venv/bin/activate
python3 -m pip install kas==4.0
KAS_IMAGE_VERSION="4.0" KAS_MACHINE="ringneck-haikou" kas-container build meta-theobroma-
↪systems-extended/kas-theobroma.yml
```

7 Deploy a disk image

This chapter describe how to write a disk image as generated in one of the previous chapters using Yocto or Debian to the module.

Note: The variable SDCARD_IMG must be already set as described in respective chapter.

Warning: Avoid having the disk image on *both* the SD Card and the internal eMMC of the module.

As the Linux kernel on the module uses PARTLABEL and PARTUUID to identify partitions to mount, it will be unpredictable whether the SD Card or the internal eMMC is used.

7.1 Deploy on SD Card

Insert an SD card into the host PC and check `dmesg -w` to find out the device name that was used.

Then, run this command, replacing `/dev/sdX` with the real device name:

```
sudo dd bs=1M conv=nocreat oflag=direct status=progress if="$SDCARD_IMG" of=/dev/sdX
```

7.2 Deploy on internal eMMC

7.2.1 Compile rkdeveloptool

To write the image directly onto the on-board eMMC, the flashing tool *rkdeveloptool* is used, and it must be compiled on the host PC:

```
# Install compile dependencies
sudo apt-get -y install git libudev-dev libusb-1.0-0-dev dh-autoreconf pkg-config build-
↪essential

# Download rkdeveloptool source code
git clone https://github.com/rockchip-linux/rkdeveloptool.git
cd rkdeveloptool

# Compile rkdeveloptool
autoreconf -i
CPPFLAGS=-Wno-format-truncation ./configure
make

# Download miniloaders used for flashing
git clone https://github.com/rockchip-linux/rkbin.git tools/rk_tools

# Build miniloader binaries
pushd tools/rk_tools/
./tools/boot_merger RKBOOT/PX30MINIALL.ini
popd

# Make the resulting files available to later steps
export RKDEVELOPTOOL_DIR=$PWD
```

This step should take about 1 minute total.

7.2.2 Enter USB flashing mode

To enter the USB flashing mode, make sure the `BOOT SW` slider (see Fig. 3.1 *The baseboard with RINGNECK SOM-PX30-uQ7 module*) is in `BIOS Disable` mode and there's no SD card inserted in the baseboard.

Then, insert a micro-USB cable into the USB-OTG port (see Fig. 3.8 *USB 2.0 OTG port (dual-role port: can be used as a host or device interface)*) on the baseboard and into a USB port of your host PC.

Then, power cycle the device by unplugging and replugging the power supply or by pressing the `Reset` button. The `lsusb` command on your host PC should return the following:

```
$ lsusb -d 2207:330d
Bus 001 Device 028: ID 2207:330d Fuzhou Rockchip Electronics Company
```

Now, put the `BOOT SW` slider back into the `Normal Boot` mode.

7.2.3 Flash the eMMC

Warning: The `BOOT SW` slider must be back in `Normal Boot` mode, otherwise the eMMC is inaccessible and stays empty. You will see rkdeveloptool making improbably quick write progress in this case.

To write the image file path stored in the variable `SDCARD_IMG` to the on-board eMMC, run:

```
cd $RKDEVELOPTOOL_DIR
sudo ./rkdeveloptool db tools/rk_tools/px30_loader_v*.bin && sleep 1
sudo ./rkdeveloptool wl 0 $SDCARD_IMG
sudo ./rkdeveloptool rd
```

This step should take about 1 minute for the Debian image.

8 Wifi

The RINGNECK SOM-PX30-uQ7 module features an on-board Wifi module. This chapter shows how to connect to an existing Wifi network and how to flash the wifi firmware, should the need arise.

8.1 Antenna

The development kit includes an antenna compatible with the Wifi module. Other antennas can be used. The connector on the antenna must be one of:

- W.FL Series connector from Hirose
- MHF III connector from I-PEX
- AMMC connector from Amphenol

8.2 Connecting to a Wifi network

You can show the available wifi networks using:

```
nmcli dev wifi
```

Connect to a network using the following command (replace the network name and password as appropriate):

```
nmcli dev wifi connect "Theobroma Example Wifi" password "hello-px30"
```

You should get a message like:

```
Device 'wlan0' successfully activated with '79ef39fc-8f49-4719-a8d9-4d6d789bb815'.
```

You should have connectivity over Wifi now. You can check the IP address you received using:

```
ip addr show dev wlan0
```

Note: By default, nmcli is not available in our Yocto core-image-minimal image. However, it is available in our Yocto theobroma-extended-image image.

8.3 Flashing the wifi firmware

You need to have `esptool.py` installed on the module.

The wifi firmware consists of three files:

- bootloader.bin
- partition-table.bin
- eagle.bin

Save all three to the `/tmp` directory on the module.

Then flash the wifi module as shown below:

```
GPIO_BOOT=1 #GPIO0_A1
GPIO_EN=72 #GPIO2_B0

echo ff380000.mmc > /sys/bus/platform/drivers/dwmmc_rockchip/unbind
echo sdio-pwrseq > /sys/bus/platform/drivers/pwrseq_simple/unbind

if [ ! -d /sys/class/gpio/gpio$GPIO_BOOT ]; then
        echo $GPIO_BOOT > /sys/class/gpio/export
fi

if [ ! -d /sys/class/gpio/gpio$GPIO_EN ]; then
        echo $GPIO_EN > /sys/class/gpio/export
fi

echo out > /sys/class/gpio/gpio$GPIO_BOOT/direction
echo out > /sys/class/gpio/gpio$GPIO_EN/direction

echo 0 > /sys/class/gpio/gpio$GPIO_BOOT/value
echo 0 > /sys/class/gpio/gpio$GPIO_EN/value
sleep 1
echo 1 > /sys/class/gpio/gpio$GPIO_EN/value
sleep 1

ESPTOOL=$(PATH=/root/.local/bin/:$PATH which esptool.py)

$ESPTOOL -p /dev/ttyS3 -b 460800 --before default_reset --after hard_reset \
  --chip esp32 write_flash --flash_mode dio --flash_size detect --flash_freq 40m \
  0x1000 /tmp/bootloader.bin \
  0x8000 /tmp/partition-table.bin \
  0x10000 /tmp/eagle.bin

sleep 1
echo 1 > /sys/class/gpio/gpio$GPIO_BOOT/value
echo 0 > /sys/class/gpio/gpio$GPIO_EN/value
sleep 1
echo 1 > /sys/class/gpio/gpio$GPIO_EN/value

echo $GPIO_BOOT > /sys/class/gpio/unexport
echo $GPIO_EN > /sys/class/gpio/unexport

echo sdio-pwrseq > /sys/bus/platform/drivers/pwrseq_simple/bind
echo ff380000.mmc > /sys/bus/platform/drivers/dwmmc_rockchip/bind
```

Note: On Debian, the `esptool` package provided by the package feed is too old. Instead, please install `esptool` software from pip:

```
apt-get -y install python3-pip
pip3 install --user esptool
```

Note: By default, esptool is not available in our Yocto core-image-minimal image.

9 Serial Number & MAC Address

9.1 Serial Number

Each RINGNECK SOM-PX30-uQ7 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 SoC *CPU ID*) and cannot be modified.

9.2 MAC Address

By default, the MAC address of each 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 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
```

10 Mule Companion Controller

Mule Companion Controller is an on-board microcontroller, that provides additional features to the CPU. Mule is available in two variants:

- Companion Controller 1 (STM32)
- Companion Controller 2 (ATtiny)

Only one variant can be available on the board.

Both variants support almost the same set of features. The only difference is CAN support.

Feature set and usage manual of both variants are described in subsections below.

10.1 Companion Controller 1 (STM32)

Controller is based on STM32 microcontroller and provides additional features to the CPU, exposed via I2C and USB. It emulates standard ICs and does not need custom drivers on Linux.

Mule STM32 controller supports the following features:

- RTC
- Temperature sensor
- Fan controller
- CAN

For hardware details, please refer to Section 12.4.4 *Companion Controller 1*.

10.1.1 Internal connections

Mule STM32 controller is connected to SoC via I2C, USB and the following pins.

Function	CPU Pin	Linux GPIO #
NRST	GPIO3_A4	100
BOOT0	GPIO3_A5	101

10.1.2 DFU mode

The USB DFU bootloader application provides access to the internal flash memory of STM32 microcontroller.

To enter DFU mode:

1. Pull BOOT0 pin high
2. Cycle reset Mule STM32 using NRST pin
3. The microcontroller will appear as a new USB device in Linux (vid:pid as 0483:df11)

To return to normal operation, BOOT0 must be pulled low again to not enter DFU mode in the next power-cycle.

10.1.3 Flashing the STM32 firmware

For convenience, `mule.sh` tool is available for controlling and flashing the STM32 microcontroller. Executing the script, SoC resets microcontroller into DFU mode and then uploads the firmware binary to internal STM32 flash memory.

The tool is available here: https://git.theobroma-systems.com/som-tools.git/tree/mule.

To flash STM32 microcontroller using `mule.sh`, please follow the steps below.

1. Install `mule.sh` dependencies according to `README.md`
2. Upload `mule.sh` tool and `mule.dfu` firmware file to a device
3. Flash controller using the following command:

```
sudo ./mule.sh --flash mule.dfu
```

Note: It is highly recommended that one reboots the main SoC interacting with the companion microcontroller after flashing to make sure device drivers are properly initialized.

10.2 Companion Controller 2 (ATtiny)

Controller is based on ATtiny microcontroller and provides additional features to the CPU, exposed via I2C. This controller is a substitute for first controller, supports the same functions except CAN. As for the first controller, it emulates standard ICs and does not need custom drivers on Linux.

Mule ATtiny controller supports the following features:

- RTC
- Temperature sensor
- Fan controller

For hardware details, please refer to Section 12.4.5 *Companion Controller 2*.

10.2.1 Internal connections

Mule ATtiny controller is connected to SoC via I2C bus and the following pins.

Function	CPU Pin	Linux GPIO #
RST	GPIO3_A4	100
BOOT	GPIO3_A5	101

10.2.2 Flashloader mode

Flashloader mode allows writing to the internal ATtiny flash memory via I2C.

To enter flashloader mode:

1. Pull BOOT pin high
2. Cycle reset ATtiny using RST pin

10.2.3 Flashing the ATtiny firmware

The ATtiny microcontroller can be flashed from SoC through the I2C interface using `i2c-flash` tools. Executing the script, SoC resets microcontroller into flashloader mode and then transfers the binary that will be committed to flash.

Tools are available here: https://git.theobroma-systems.com/som-tools.git/tree/mule-attiny.

To flash ATtiny microcontroller, please follow the steps below.

1. Setup tool dependencies according to `README.md`
2. Flash controller using the following command:

```
./i2c_flash.py -f {firmware.bin} -c 3 -g 5 -b 1 -rc 3 -rg 4
```

When using the flashing script, the bootloader version is printed to the output console. The version can also be read without flashing the device (WARNING: This resets the RTC logic) using:

```
./i2c_flash.py -c 3 -g 5 -b 1 -rc 3 -rg 4
```

without specifiying a firmware binary.

Note: It is highly recommended that one reboots the main SoC interacting with the companion microcontroller after using the flashing script to make sure device drivers are properly initialized.

---

11 Phosh graphical shell

Phosh is a graphical user interface designed for touch-based devices. It is based on the GTK widget toolkit, and derives from the GNOME Shell as a mobile-specific fork. Phosh is used as a default graphical user interface in the reference images for the Haikou-Video-Demo.

11.1 Usage

Phosh features a user interface which is similar to what is found on mobile phones today:

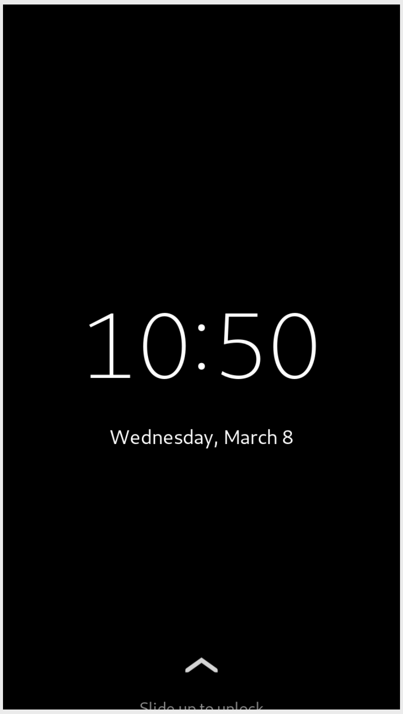

Fig. 11.1: Locked screen

Fig. 11.2: Login screen

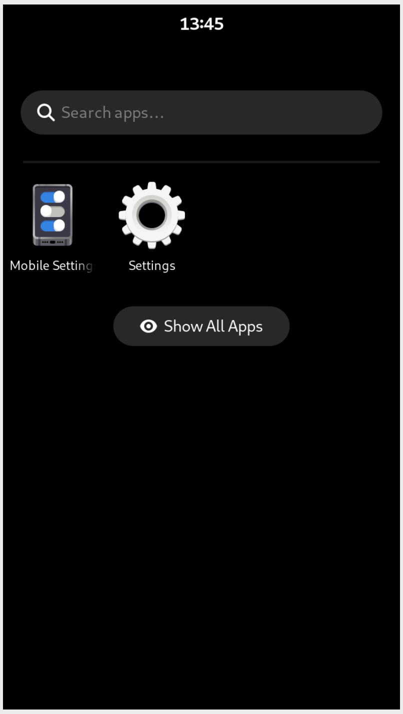

Fig. 11.3: Main screen

11.1.1 Unlocking the screen

After the boot up, the device is locked. When the device is locked, display should show a screen similar to Fig. 11.1.

To unlock the device, please follow the steps below:

1. Slide the screen from the bottom to the top, to access the login screen. The login screen looks similar to Fig. 11.2.
2. Enter the password and press the 'Unlock' button. For non-numeric password, there is a virtual keyboard available. Virtual keyboard can be opened using the bottom-left button with the keyboard icon.

Default user on Phosh image is `user`. Default password is 123123.

After unlocking the device, the *Main screen* (Fig. 11.3) should be visible on the display.

11.1.2 Waking up the device

The user can lock the device again using the *Lock Screen* button from the top bar menu. The current session will be locked, and the display will be turned off. To turn on the display again, press the `WAKE` button on the Haikou baseboard.

11.2 Known issues

1. Wrong display resolution when device is locked. *Locked screen* (Fig. 11.1) and *Login screen* (Fig. 11.2) are extended by few pixels at the bottom. This causes that button and text placed at the bottom are not displayed correctly. This issue does not occur when the device is unlocked.
2. *Settings* application (*gnome-control-center*) crashes when trying to open *Displays* tab. The last opened tab is remembered by the *Settings*, which causes crashes every time the application is opened. This makes the application unusable. To restore the application to a usable state, open another tab using the terminal:

```
gnome-control-center power
```

3. *No battery* icon is visible in the top right corner.

12 Hardware Guide

This Hardware Guide provides information about the features, connectors and signals available on the RINGNECK SOM-PX30-uQ7 module.

12.1 Q7 Implementation

Q7 has mandatory and optional features. Following table shows the feature set of the RINGNECK SOM-PX30-uQ7 module compared to the minimum ARM/RISC based and maximum configuration according to the Q7 standard.

System I/O Interface	Q7 Minimum	RINGNECK SOM-PX30-uQ7	Q7 Maximum
PCI Express lanes	0	0	4
Serial ATA channels	0	0	2
USB 2.0 ports	1	4	8
USB 3.0 ports	0	0	3
LVDS channels	0	1	2
Embedded DisplayPort	0	0	1
MIPI-CSI	0	1	2
HDMI	0	0	1
High Definition Audio / AC'97 / I2S	0	1	1
Ethernet 10/100/Gigabit	0	1x 100Mbps	1x Gigabit
UART	0	1 (+1 shared with GPIO)	1
GPIO	0	8	8
Secure Digital I/O	0	1	1
System Management Bus	0	0	1
I^2C Bus	1	3	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 RINGNECK SOM-PX30-uQ7 module is available in different variants. This document describes the maximum configuration. For details about orderable variants please refer to the order-code document.

Note: Not all interfaces are available at the same time as they might conflict with others. E.g. it is not possible to have LVDS channels and MIPI-DSI at the same time.

12.2 Q7 Connector Pinout

The following table shows the signals on the edge connector of the RINGNECK SOM-PX30-uQ7 module.

Empty cells are not connected (NC) pins.

Pin	Signal	Pin	Signal
1	GND	2	GND
3		4	
5		6	
7	GBE_LINK#	8	GBE_LINK1000#
9	GBE_MDI1-	10	GBE_MDIO0-
11	GBE_MDI1+	12	GBE_MDIO0+
13	GBE_LINK#	14	GBE_ACT#
15	GBE_CTRFF	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	
57	GND	58	GND
59	I2S_WS	60	
61	I2S_RST#	62	
63	I2S_CLK	64	
65	I2S_SDI	66	GP0_I2C_CLK
67	I2S_SDO	68	GP0_I2C_DAT
69		70	WDTRIG#
71	THRMTRIP#	72	WDOUT
73	GND	74	GND
75		76	
77		78	
79		80	
81		82	
83		84	
85	USB_OC#	86	USB_OC#
87	USB_P3-	88	USB_P2-
89	USB_P3+	90	USB_P2+
91	USB_VBUS	92	USB_ID
93	USB_P1-	94	USB_P0-
95	USB_P1+	96	USB_P0+
97	GND	98	GND
99	LVDS_A0+/DSI_D0+	100	CSI_D0+
101	LVDS_A0-/DSI_D0-	102	CSI_D0-
103	LVDS_A1+/DSI_D1+	104	CSI_D1+

continues on next page

Table 12.1 – continued from previous page

Pin	Signal	Pin	Signal
105	LVDS_A1-/DSI_D1-	106	CSI_D1-
107	LVDS_A2+/DSI_D2+	108	CSI_D2+
109	LVDS_A2-/DSI_D2-	110	CSI_D2-
111	LVDS_PPEN	112	LVDS_BLEN
113	LVDS_A3+/DSI_D3+	114	CSI_D3+
115	LVDS_A3-/DSI_D3-	116	CSI_D3-
117	GND	118	GND
119	LVDS_A_CLK+/DSI_CLK+	120	CSI_CLK+
121	LVDS_A_CLK-/DSI_CLK-	122	CSI_CLK-
123	LVDS_BLT_CTRL / GP_PWM_OUT0	124	
125	GP2_I2C_DAT / LVDS_DID_DAT	126	LVDS_BLC_DAT
127	GP2_I2C_CLK / LVDS_DID_CLK	128	LVDS_BLC_CLK
129	CAN0_TX	130	CAN0_RX
131		132	
133		134	
135	GND	136	GND
137		138	
139		140	
141	GND	142	GND
143		144	
145		146	
147	GND	148	GND
149		150	
151		152	
153		154	
155		156	
157		158	
159	GND	160	GND
161		162	
163		164	
165	GND	166	GND
167		168	
169		170	
171	UART0_TX	172	UART0_RTS#
173		174	
175		176	
177	UART0_RX	178	UART0_CTS#
179		180	
181		182	
183	GND	184	GND
185	GPIO0	186	GPIO1
187	GPIO2	188	GPIO3
189	GPIO4	190	GPIO5 / UART1_TX
191	GPIO6 / UART1_RX	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#
203	SPI_SCK	204	MFG_BIOS_DISABLE#
205		206	
207		208	UPDI_UART_TX
209	UPDI_UART_RX	210	
211		212	
213		214	
215		216	
217		218	

continues on next page

Table 12.1 – continued from previous page

Pin	Signal	Pin	Signal
219	VCC	220	VCC
221	VCC	222	VCC
223	VCC	224	VCC
225	VCC	226	VCC
227	VCC	228	VCC
229	VCC	230	VCC

12.3 Signal Details

12.3.1 Ethernet

Q7 Signal	Type	Signal Level	Description
GBE_MDI[0:1]+ GBE_MDI[0:1]-	I/O	Analog	Fast Ethernet Controller: Media Dependent Interface Differential Pairs 0,1. The MDI can operate in 100 and 10 Mbit/sec modes
GBE_ACT#	OC	3.3V	Ethernet Controller activity indicator, active low
GBE_LINK#	OC	3.3V	Ethernet Controller link indicator, active low
GBE_LINK100#	OC	3.3V	Internally connected to GBE_LINK#
GBE_CTREF	REF	Analog	Center Tap Voltage

12.3.2 USB

Q7 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_OC#	I	3.3V	Over current detect input. The baseboard can signal an USB overcurrent condition by pulling this pin low.
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

12.3.3 SDIO

Q7 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

12.3.4 I2C

Q7 Signal	Type	Signal Level	Description
Q7_I2C_CLK	O	3.3V	I2C bus clock line connected to PX30
Q7_I2C_DAT	I/O	3.3V	I2C bus data line connected to PX30
LVDS_DID_CLK /GP2_I2C_CLK	O	3.3V	I2C bus clock line connected to PX30, Secure Element, STM32, Attiny and Video connector
LVDS_DID_DAT /GP2_I2C_DAT	I/O	3.3V	I2C bus data line connected to PX30, Secure Element, STM32, Attiny and Video connector
LVDS_BLC_DAT	O	3.3V	I2C bus clock line connected to PX30, Video connector and baseboard EEPROM
LVDS_BLC_CLK	I/O	3.3V	I2C bus data line connected to PX30, Video connector and baseboard EEPROM

12.3.5 I2S

Q7 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

12.3.6 Video

The Q7 LVDS_A pins support LVDS and MIDI-DSI. LVDS and MIPI-DSI signals are electrically compatible in the sense that nothing will be damaged, but are not defined in the Qseven standard.

The MIPI-DSI specifications are:

- MIPI DSI D-PHY v1.0
- Up to four data lates
- Up to 1.0 Gbps per lane

The signal mapping is shown below:

Q7 Signal	Function 1	Function 2
LVDS_A0_P	LVDS_A0+	DSI_D0+
LVDS_A0_N	LVDS_A0-	DSI_D0-
LVDS_A1_P	LVDS_A0+	DSI_D1+
LVDS_A1_N	LVDS_A1-	DSI_D1-
LVDS_A2_P	LVDS_A2+	DSI_D2+
LVDS_A2_N	LVDS_A2-	DSI_D2-
LVDS_A3_P	LVDS_A3+	DSI_D3+
LVDS_A3_N	LVDS_A3-	DSI_D3-
LVDS_A_CLK_P	LVDS_A_CLK+	DSI_CLK+
LVDS_A_CLK_N	LVDS_A_CLK-	DSI_CLK-

The Q7 LVDS_B pins are used as MIPI-CSI. The specifications are:

- MIPI CSI D-PHY v1.0
- Up to four data lanes

- Up to 1.0 Gbps per lane

The signal mapping is shown below:

Q7 Signal	Function
LVDS_B0_P	CSI_D0+
LVDS_B0_N	CSI_D0-
LVDS_B1_P	CSI_D1+
LVDS_B1_N	CSI_D1-
LVDS_B2_P	CSI_D2+
LVDS_B2_N	CSI_D2-
LVDS_B3_P	CSI_D3+
LVDS_B3_N	CSI_D3-
LVDS_B_CLK_P	CSI_CLK+
LVDS_B_CLK_N	CSI_CLK-

12.3.7 GPIO

Q7 Signal	Type	Signal Level	Description
GPIO[0-7]	I/O	3.3V	General purpose inputs/outputs 0 to 7

12.3.8 CAN

Q7 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

12.3.9 SPI

Q7 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)

12.3.10 UART

`UART0`, as specified in the Q7 standard, is implemented including hardware flow control. This UART shows up in Linux as `/dev/ttyS0`.

Q7 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

A second UART, `UART1`, can be enabled on the `GPIO` pins. This UART shows up in Linux as `/dev/ttyS5`.

Q7 Signal	Alternate function	Type	Signal Level	Description
GPIO5	UART1_TX	O	3.3V	Serial data transmit
GPIO6	UART1_RX	I	3.3V	Serial data receive

12.3.11 Misc

Signal	Type	Signal Level	Description
WDTRIG#	I	3.3V	Watchdog trigger signal
WDOUT	O	3.3V	Watchdog event indicator
SPKR GP_PWM_OUT2	O	3.3V	PC speaker (buzzer) 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
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

12.3.12 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

12.3.13 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. Typical current: 1.4uA

12.4 On-board Devices

12.4.1 Power-Manager

The Rockchip RK809-1 is connected to the CPU via I2C:

RK809-1 Pin	Function	CPU Pin
1	SCL	I2C0_SCL_u (ball R21)
2	SDA	I2C0_SDA_u (ball M21)

12.4.2 DDR4

- Up to 4GB RAM of DDR4-1600

12.4.3 eMMC

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

Signal	CPU Pin	Linux GPIO #
RESET	GPIO1_B3	43

12.4.4 Companion Controller 1

Controller features are implemented by emulating standard ICs. Feature configuration is provided in a table below.

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

See also Section 10.1 *Companion Controller 1 (STM32)*.

12.4.5 Companion Controller 2

Controller features are implemented by emulating standard ICs. Feature configuration is provided in a table below.

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

See also Section 10.2 *Companion Controller 2 (ATtiny)*.

12.4.6 Ethernet PHY

The Texas Instruments DP83825IRMQR is connected to the CPU via RGMII and MDIO. Further connections are shown below.

PHY signal	Connected to	Linux GPIO #
RESET	CPU pin GPIO3_B0	104
MDIO	CPU pin GPIO2_A7	71
MDC	CPU pin GPIO2_B1	73
LED1	Qseven GBE_LINK1000 and GBE_LINK100 and GBE_LINK (tied together)	
LED2	Qseven GBE_ACT	

12.5 Wifi and Bluetooth module

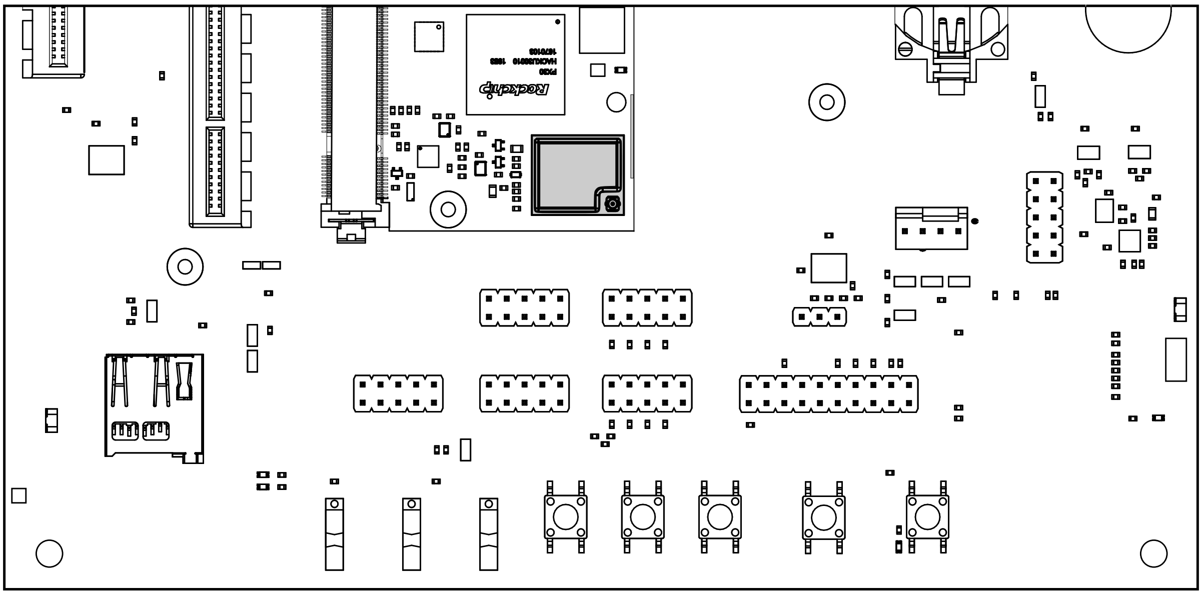

Fig. 12.1: WiFi and Bluetooth module

The WiFi and Bluetooth are part of the ESP32 PICO MINI 02U module on the RINGNECK SOM-PX30-uQ7 board. The antenna connector on the module is w.FL type. The firmware running on the ESP32 is flashed in its internal memory and unlike most wireless modules, does not require files to be present in the root filesystem. This also means that a firmware upgrade is slightly more complex since it needs to be flashed (see Section 8.3 *Flashing the wifi firmware*).

The following pins are used for boot and reset.

ESP32 signal	CPU Pin	Linux GPIO #
WiFi_RST	GPIO2_B0	72
WiFi_BOOT	GPIO0_A1	1

12.5.1 Test points RINGNECK SOM-PX30-uQ7 v1.2

Test point	Connected to
TP1	5V
TP2	VCC_3V3
TP3	VDD_LOG
TP4	VDD_ARM
TP5	VCC_DDR
TP6	VCC_3V0_1V8
TP7	VCC_1V8
TP8	VCC_1V0
TP9	VCCIO_SD
TP10	VCC_LCD
TP11	1V8_LCD
TP12	VCCA_1V8
TP13	VCC_eMMC
TP14	PMIC_INT
TP15	PMIC_SLEEP
TP16	VDC
TP17	PMIC_PWRON
TP18	I2C0_SCL
TP19	I2C0_SDA
TP20	PMIC_Xin
TP21	PMIC_Xout
TP22	PX30_Xin
TP23	PX30_Xout
TP24	MCU_UART_TX
TP25	MCU_UART_RX
TP26	Q7_LVDS_DID_CLK
TP27	Q7_LVDS_DID_DAT
TP28	Q7_LVDS_BLC_DAT
TP29	Q7_LVDS_BLC_CLK
TP30	I34 WiFi
TP31	I35 WiFi
TP32	ESP32_TXD0
TP33	ESP32_RXD0
TP34	BT_UART_TX
TP35	BT_UART_RX
TP36	BT_RESET
TP37	BT_UART_RTS_n
TP38	BT_UART_CTS_n
TP39	BT_wake_host

12.6 USB

The RINGNECK SOM-PX30-uQ7 CPU has 2 USB 2.0 controllers. A USB 2.0 hub provides two additional USB 2.0 ports for a total of four.

The routing of Qseven signals to CPU and/or hub port is shown below.

Qseven Port #	Speed	Connected to	Notes
USB_P0	USB 2.0 Hi-Speed	Hub	
USB_P1	USB 2.0 Hi-Speed	CPU	OTG Port
USB_P2	USB 2.0 Hi-Speed	Hub	
USB_P3	USB 2.0 Hi-Speed	Hub	

The `lsusb -t` command shows the USB topology in a tree view and is highly recommended. Its output is discussed below, for a RINGNECK SOM-PX30-uQ7 module without additional devices connected:

```
- Bus 03.Port 1: Dev 1, Class=root_hub, Driver=ohci-platform/1p, 12M
- Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ehci-platform/1p, 480M
  *  Port 1: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
       Port 1: Dev 3, If 0, Class=Mass Storage, Driver=usb-storage, 480M
       Port 3: Dev 4, If 0, Class=Mass Storage, Driver=usb-storage, 480M
       Port 4: Dev 6, If 0, Class=Vendor Specific Class, Driver=ucan, 12M
- Bus 01.Port 1: Dev 1, Class=root hub, Driver=dwc2 1p,  480M
```

The CAN controller is connected to `Port 4` on the hub.

The USB hub can be held in reset, if required. This disables all USB ports connected to the hub. The reset signal routing is shown below:

Hub signal	CPU Pin	Linux GPIO #
USBHUB_RESETn	GPIO0_A5	5

12.7 Using Qseven Signals as GPIO

Most Qseven signals can be reused as a general purpose I/O pin. The following table shows the mapping and the possible direction as seen from the baseboard.

Qseven Pin	Signal	CPU Pin	Linux GPIO #	Direction
16	SUS_S5#	GPIO3_A0	96	Bidirectional
17	WAKE#	GPIO1_B6	46	Input
18	SUS_S3#	GPIO3_A3	99	Bidirectional
19	GPO0	GPIO2_B3	75	Bidirectional
21	SLP_BTN#	GPIO1_B7	47	Input
22	LID_BTN#	GPIO3_A6	102	Bidirectional
27	BATLOW#	GPIO3_A7	103	Bidirectional
42	SDIO_CLK#	GPIO1_D6	62	Bidirectional
43	SDIO_CD#	GPIO0_A3	3	Bidirectional
44	SDIO_LED	GPIO3_B3	107	Birectional
45	SDIO_CMD	GPIO1_D7	63	Bidirectional
46	SDIO_WP	GPIO3_B5	109	Bidirectional
47	SDIO_PWR#	GPIO3_D3	123	Bidirectional
48	SDIO_DAT1	GPIO1_D3	59	Bidirectional
49	SDIO_DAT0	GPIO1_D2	58	Bidirectional
50	SDIO_DAT3	GPIO1_D5	61	Bidirectional
51	SDIO_DAT2	GPIO1_D4	60	Bidirectional
59	I2S_WS	GPIO3_C2	114	Bidrectional
63	I2S_CLK	GPIO3_C3	115	Bidirectional
65	I2S_SDI	GPIO3_C5	117	Bidirectional
66	GP0_I2C_CLK	GPIO2_B7	79	Bidirectional
67	I2S_SDO	GPIO3_C4	116	Bidirectional
68	GP0_I2C_DAT	GPIO2_C0	80	Bidirectional
71	THRMTRIP#	GPIO3_D2	122	Bidirectional
111	LVDS_PPEN	GPIO0_A2	2	Bidirectional
112	LVDS_BLEN	GPIO0_A0	0	Bidirectional
123	LVDS_BLT_CTRL / GP_PWM_OUT0	GPIO0_B7	15	Bidirectional
125	GP2_I2C_DAT / LVDS_DID_DAT	GPIO0_C3	19	Bidirectional
127	GP2_I2C_CLK / LVDS_DID_CLK	GPIO0_C2	18	Bidirectional
171	UART0_TX	GPIO0_B2	10	Bidirectional
172	UART0_RTS#	GPIO0_B5	13	Bidirectional
177	UART0_RX	GPIO0_B3	11	Bidirectional
178	UART0_CTS#	GPIO0_B4	12	Bidirectional
185	GPIO0	GPIO3_C6	118	Bidirectional
186	GPIO1	GPIO3_D0	120	Bidirectional
187	GPIO2	GPIO3_C7	119	Bidirectional
188	GPIO3	GPIO3_D1	121	Bidirectional
189	GPIO4	GPIO3_C0	112	Bidirectional
190	GPIO5	GPIO3_A2	98	Bidirectional
191	GPIO6	GPIO3_A1	97	Bidirectional
192	GPIO7	GPIO2_B6	78	Bidirectional
199	SPI_MOSI	GPIO3_B4	108	Bidirectional
200	SPI_CS0#	GPIO3_B1	105	Bidirectional
201	SPI_MISO	GPIO3_B6	110	Bidirectional
202	SPI_CS1#	GPIO3_B2	106	Bidirectional
203	SPI_SCK	GPIO3_B7	111	Bidirectional

12.8 Electrical Specification

12.8.1 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

12.9 Mechanical Specification

12.9.1 Module Dimensions

The mechanical dimensions of the module are shown below.

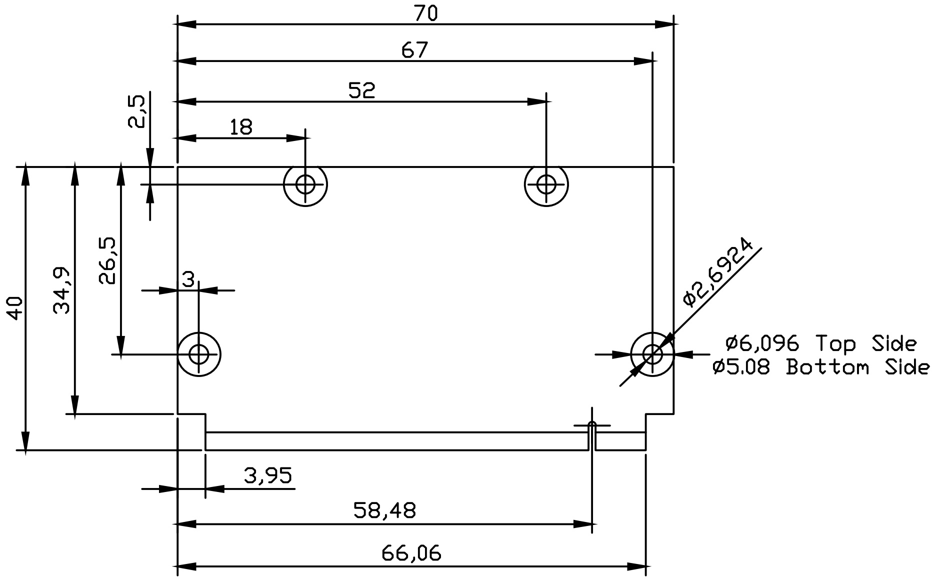

Fig. 12.2: Module dimensions (all values in mm)

12.9.2 Baseboard Dimensions

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

13 Known limitations

1. The Rockchip PX30 watchdog should not be used if booting from SD card otherwise the RINGNECK SOM-PX30-uQ7 module will not boot unless power-cycled manually.

14 Contact

Theobroma Systems Design und Consulting GmbH
Seestadtstraße 27
1220 Vienna
Austria

Inquiries: sales@theobroma-systems.com
Technical Support: support@theobroma-systems.com

15 Revision History

Date	Revision	Major changes
Nov 1, 2022	v0.0.1	First internal release
Dec 21, 2022	v1.0.0	Internal review ; public release
Dec 21, 2022	v1.0.1	Fix one missing image in html output
Mar 29, 2023	v1.1.0	Add Phosh graphical shell
Apr 27, 2023	v1.2.0	Add meta-extended demo image build instructions Fix incorrect Haikou header(s) for LVDS_DID* and GP0_I2C_* signals CAN only supported with STM32
Aug 23, 2023	v1.3.0	Fix Yocto directory tree layout Add companion controller 1 and 2 (STM & ATtiny) flashing instructions Fix companion controller 1 (STM) flashmode entering instructions Move companion controller 1 and 2 flashing instructions to separate section
Dec 18, 2023	v1.3.1	recalled version
Feb 15, 2024	v1.3.2	Rename files for consistency between products Replace dd flashing instructions with bmaptool Update pip instructions for Bookworm Add yocto directory creation Update kas container instructions Rephrase layer version requirement Bump yocto layer and kas versions
Mar 08, 2024	v1.4.0	Add known SoC watchdog limitation note Update build instruction to match git repositories changes Add note on how to build debos on non Debian systems