Devices

Robosoft

Board reference

v1.1.1-r1808

Notice

8devices reserves the right to change specifications without prior notice. While the information in this manual has been compiled with great care, it may not be deemed an assurance of product characteristics. 8devices shall be liable only to the degree specified in the terms of sale and delivery. The reproduction and distribution of the documentation and software supplied with this product and the use of its contents are subject to written authorization from 8devices.

Trademarks

8devices logo is a trademark of 8devices. All other registered and unregistered trademarks in this document are the sole property of their respective owners.

Table Of Contents

1. Qualcomm QCS405 SoC

1.1 Overview

Qualcomm QCS405 System-on-Chip for IoT and edge computing applications. This document describes hardware peripherals relevant for Robosoft development.

Official Documentation: Qualcomm QCS405 Product Brief

1.2 GPIO Banks

QCS405 provides 120 GPIO pins with multiple alternate functions per pin. Each GPIO can be configured via device tree for specific peripheral functions.

GPIO #	Voltage	Functions
0	1.8V	sdc1_data[7]/nand_io[7]
1	1.8V	sdc1_data[6]/nand_io[6]
2	1.8V	sdc1_data[5]/nand_io[5]
3	1.8V	sdc1_data[4]/nand_io[4]
4	1.8V	sdc1_data[3]/nand_io[3]
5	1.8V	sdc1_data[2]/nand_io[2]
6	1.8V	sdc1_data[1]/nand_io[1]
7	1.8V	sdc1_data[0]/nand_io[0]
8	1.8V	sdc1_rclk/nand_cs_n
9	1.8V	sdc1_cmd/nand_we_n
10	1.8V	sdc1_clk/nand_oe_n
11	1.8V	nand_ale
12	1.8V	nand_cle
13	1.8V	nand_rdy_nbusy
14	1.8V	hdmi_tx_cec
15	1.8V	hdmi_ddc_clk
16	1.8V	hdmi_ddc_data
17	1.8V, 3.15V	blsp_uart_tx_a[2], blsp_spi_mosi[2]
18	1.8V, 3.15V	blsp_uart_rx_a[2], blsp_spi_miso[2]
19	1.8V, 3.15V	blsp_uart_cts_n[2], aud_cdc_int2, blsp_i2c_sda_a[2], blsp_spi_cs_n[2]
20	1.8V, 3.15V	blsp_uart_rfr_n[2], aud_cdc_int1, blsp_i2c_scl_a[2], blsp_spi_clk[2]
21	1.8V	-
22	1.8V, 3.15V	blsp_uart_tx[1], blsp_spi_mosi_a[1], asdiv1
23	1.8V, 3.15V	blsp_uart_rx[1], blsp_spi_miso_a[1], asdiv2
24	1.8V, 3.15V	blsp_uart_cts_n[1], blsp_i2c_sda[1], blsp_spi_cs_n_a[1]

GPIO #	Voltage	Functions
25	1.8V, 3.15V	blsp_uart_rfr_n[1], blsp_i2c_scl[1], blsp_spi_clk_a[1]
26	1.8V	rgb_data_r[0], blsp_uart_tx[5], blsp_spi_mosi[5]
27	1.8V	rgb_data_r[1], blsp_uart_rx[5], blsp_spi_miso[5]
28	1.8V	rgb_data_r[2], blsp_uart_cts_n[5], blsp_i2c_sda[5], blsp_spi_cs_n[5], gcc_gp1_clk_b
29	1.8V	rgb_data_r[3], blsp_uart_rfr_n[5], blsp_i2c_scl[5], blsp_spi_clk[5], gcc_gp2_clk_b
30	1.8V	blsp_spi_mosi[0], blsp_uart_tx[0], gcc_gp3_clk_b
31	1.8V	blsp_spi_miso[0], blsp_uart_rx[0]
32	1.8V	blsp_spi_cs_n[0], blsp_uart_cts_n[0], blsp_i2c_sda[0]
33	1.8V	blsp_spi_clk[0], blsp_uart_rfr_n[0], blsp_i2c_scl[0]
34	1.8V	-
35	1.8V	pcie_clk_req
36	1.8V	asdiv1_mir
37	1.8V	nfc_irq, blsp_spi_mosi[4], asdiv2_mir
38	1.8V	nfc_dwl_req, blsp_spi_miso[4], audio_ts_in
39	1.8V	rgb_data_r[4], spi_lcd_dc, blsp_uart_tx_b[2], gcc_gp3_clk_a
40	1.8V	rgb_data_r[5], spi_lcd_bl_pwm, blsp_uart_rx_b[2], gp_pdm_1a
41	1.8V	rgb_data_g[0], blsp_i2c_sda_b[2], gp0_clk
42	1.8V	rgb_data_g[1], blsp_i2c_scl_b[2], gp1_clk
43	1.8V	rgb_data_g[2], pwm_led11, gp_pdm_0b
44	1.8V	rgb_data_g[3], pwm_led12, blsp_spi_cs1_n[5]
45	1.8V	rgb_data_g[4], pwm_led13, blsp_spi_cs2_n[5]
46	1.8V	rgb_data_g[5], pwm_led14, blsp_spi_cs3_n[5]
47	1.8V	rgb_data_b[0], pwm_led15, blsp_spi_mosi_b[1]
48	1.8V	rgb_data_b[1], pwm_led16, blsp_spi_miso_b[1], gp_pdm_2a

GPIO #	Voltage	Functions
49	1.8V	rgb_data_b[2], pwm_led17, blsp_spi_cs_n_b[1], gp_pdm_2b
50	1.8V	rgb_data_b[3], pwm_led18, blsp_spi_clk_b[1]
51	1.8V	rgb_data_b[4], pwm_led19, ext_mclk1_b
52	1.8V	rgb_data_b[5], pwm_led20, i2s_3_sck_b, ldo_update
53	1.8V	rgb_hsync, pwm_led21, i2s_3_ws_b
54	1.8V	rgb_vsync, i2s_3_data0_b, ldo_en
55	1.8V	rgb_de, i2s_3_data1_b, gp_pdm_0a
56	1.8V	rgb_clk, i2s_3_data2_b
57	1.8V	rgb_mdp_vsync_p, i2s_3_data3_b
58	1.8V	rgb_data_b[6], gp_pdm_1b
59	1.8V	rgb_data_b[7]
60	1.8V	-
61	1.8V	rgmii_int
62	1.8V	rgmii_wol_int
63	1.8V	rgmii_ck_tx
64	1.8V	rgmii_tx_3
65	1.8V	rgmii_tx_2
66	1.8V	rgmii_tx_1
67	1.8V	rgmii_tx_0
68	1.8V	rgmii_ctl_tx
69	1.8V	rgmii_ck_rx
70	1.8V	rgmii_rx_3
71	1.8V	rgmii_rx_2
72	1.8V	rgmii_rx_1
73	1.8V	rgmii_rx_0

GPIO #	Voltage	Functions
74	1.8V	rgmii_ctl_rx
75	1.8V	rgmii_mdio
76	1.8V	rgmii_mdc
77	1.8V	ir_in, wsa_en, hdmi_hot_plug
78	1.8V	rgb_data_g[6]
79	1.8V	rgb_data_g[7]
80	1.8V	rgb_data_r[6]
81	1.8V	rgb_data_r[7]
82	1.8V	blsp_uart_tx[3], blsp_spi_mosi[3], sd_write_protect, gp_mn
83	1.8V	blsp_uart_rx[3], blsp_spi_miso[3]
84	1.8V	blsp_uart_cts_n[3], blsp_i2c_sda[3], blsp_spi_cs_n[3], gcc_gp1_clk_a
85	1.8V	blsp_uart_rfr_n[3], blsp_i2c_scl[3], blsp_spi_clk[3], gcc_gp2_clk_a
86	1.8V	ext_mclk0, mclk_in1
87	1.8V	i2s_1_sck, dsd_clk_a
88	1.8V	i2s_1_ws, i2s_1_data0_dsd0
89	1.8V	i2s_1_data0, i2s_1_data1_dsd1
90	1.8V	i2s_1_data1, i2s_1_data2_dsd2
91	1.8V	i2s_1_data2, i2s_1_data3_dsd3
92	1.8V	i2s_1_data3, i2s_1_data4_dsd4
93	1.8V	i2s_1_data4, pwm_led22, i2s_1_data5_dsd5
94	1.8V	i2s_1_data5, pwm_led23, i2s_1_data6_mir
95	1.8V	i2s_1_data6, pwm_led1, i2s_1_data7_mir
96	1.8V	i2s_1_data7, pwm_led2
97	1.8V, 3.15V	i2s_2_sck
98	1.8V, 3.15V	i2s_2_ws

GPIO #	Voltage	Functions
99	1.8V, 3.15V	i2s_2_data0
100	1.8V, 3.15V	i2s_2_data1, pll_bist_sync
101	1.8V, 3.15V	i2s_2_data2
102	1.8V, 3.15V	i2s_2_data3
103	1.8V	ext_mclk1_a, mclk_in2
104	1.8V	i2s_3_sck_a
105	1.8V	i2s_3_ws_a
106	1.8V	i2s_3_data0_a, ebi2_lcd_rs_n, hdmi_hot_plug_mir
107	1.8V	i2s_3_data1_a, ebi2_lcd_cs_n
108	1.8V	i2s_3_data2_a, ebi2_lcd_te, pwm_led3
109	1.8V	i2s_3_data3_a, ebi2_lcd_en_n, pwm_led4
110	1.8V	i2s_4_sck, ebi2_a_d_8, dsd_clk_b, pwm_led5
111	1.8V	i2s_4_ws, i2s_4_data0_dsd0, pwm_led6
112	1.8V	i2s_4_data0, i2s_4_data1_dsd1, pwm_led7
113	1.8V	i2s_4_data1, i2s_4_data2_dsd2, pwm_led8
114	1.8V	i2s_4_data2, i2s_4_data3_dsd3, pwm_led24
115	1.8V	i2s_4_data3, i2s_4_data4_dsd4
116	1.8V	i2s_4_data5_dsd5, spkr_dac0
117	1.8V	blsp_i2c_sda[4], blsp_spi_cs_n[4], pwm_led9
118	1.8V	blsp_i2c_scl[4], blsp_spi_clk[4], pwm_led10
119	1.8V	spdifrx_opt_0

1.2.1 GPIO State Inspection

```
# List all GPIO chips
ls /sys/class/gpio/

# Show GPIO controller information
cat /sys/kernel/debug/gpio
```

```
# Output shows:
# - GPIO number
# - Direction (in/out)
# - Value (0/1)
# - Consumer (driver/process using it)
# - Flags (pull-up/down, drive strength)

# Check specific GPIO status
cat /sys/kernel/debug/pinctrl/*/pinmux-pins | grep gpio47

# View pin configuration
cat /sys/kernel/debug/pinctrl/*/pins | grep "pin 47"
```

1.3 BLSP Controllers

QCS405 provides 6 BLSP (BAM Low-Speed Peripheral) controllers supporting UART, I2C, and SPI protocols through pin multiplexing.

1.3.1 BLSP GPIO Assignments

BLSP #	GPIO #	Pad Voltage	SPI	UART	I2C
0	30	1.8 V	MOSI	TX	-
0	31	1.8 V	MISO	RX	-
0	32	1.8 V	CS_N	CTS_N	SDA
0	33	1.8 V	CLK	RFR_N	SCL
1	22	1.8 V, 3.15 V	MOSI_A	TX	-
1	23	1.8 V, 3.15 V	MISO_A	RX	-
1	24	1.8 V, 3.15 V	CS_N_A	CTS_N	SDA
1	25	1.8 V, 3.15 V	CLK_A	RFR_N	SCL
1	47	1.8 V	MOSI_B	-	-
1	48	1.8 V	MISO_B	-	-
1	49	1.8 V	CS_N_B	-	-
1	50	1.8 V	CLK_B	-	-
2	17	1.8 V, 3.15 V	MOSI	TX_A	-
2	18	1.8 V, 3.15 V	MISO	RX_A	-
2	19	1.8 V, 3.15 V	CS_N	CTS_N	SDA_A
2	20	1.8 V, 3.15 V	CLK	RFR_N	SCL_A
2	39	1.8 V	-	TX_B	-
2	40	1.8 V	-	RX_B	-
2	41	1.8 V	-	-	SDA_B
2	42	1.8 V	-	-	SCL_B
3	82	1.8 V	MOSI	TX	-
3	83	1.8 V	MISO	RX	-
3	84	1.8 V	CS_N	CTS_N	SDA
3	85	1.8 V	CLK	RFR_N	SCL
4	37	1.8 V	MOSI	-	-

BLSP #	GPIO #	Pad Voltage	SPI	UART	I2C
4	38	1.8 V	MISO	-	-
4	117	1.8 V	CS_N	-	SDA
4	118	1.8 V	CLK	-	SCL
5	26	1.8 V	MOSI	TX	-
5	27	1.8 V	MISO	RX	-
5	28	1.8 V	CS_N	CTS_N	SDA
5	29	1.8 V	CLK	RFR_N	SCL

Note: Each BLSP instance uses 4 GPIO pins configurable for different protocols via device tree. BLSP1 and BLSP2 support dual sub-configurations (A/B).

1.4 Ethernet

- **MAC:** Integrated Gigabit Ethernet MAC (10/100/1000 Mbps)
- **Bus:** RGMII, MDIO
- **PHY:** Requires external PHY chip (SoM/board-dependent)

1.5 USB

QCS405 provides two USB peripherals, both supporting dual-role controller and host/device/OTG modes.

Controller	Modes
USB 3.0	Host, Device, OTG
USB 2.0	Host, Device, OTG

1.5.1 USB Mode Control

USB mode is controlled via USB ID and VBUS signals. The QCS405 has dedicated pins USB0_HS_ID and USB1_HS_ID, but these are not used by dwc3 or PHY drivers. Instead, extcon-usb driver with GPIO-based USB ID detection provides mode switching.

USB ID Signal:

- **High:** USB device mode
- **Low:** USB host controller mode

VBUS Signal (input):

- Indicates whether USB port is externally powered
- Must be high for USB gadget mode

VBUS Control (output):

- Controls USB port power supply
- Required for host controller operation

1.5.2 Mode Control Table

Mode	ID (input)	VBUS (input)	VBUS (output)
USB device	High	High	Low
USB host	Low	Low	High

Note: GPIO controls for ID and VBUS must be synchronized when switching modes.

1.6 Related Documentation

- TobuFi SoM - TobuFi SoM (System-on-Module) details
- Qualcomm QCS405 Product Brief
- USB On-The-Go Overview

2. 8devices TobuFi SoM (System-on-Module)

2.1 Overview

TobuFi is an 8devices System-on-Module based on Qualcomm QCS405 SoC. This document describes SoM-level hardware integration relevant for Robosoft development.

2.2 Module Components

Component	Specification	Details
SoC	Qualcomm QCS405	Quad-core ARM Cortex-A53 @ 1.4 GHz
RAM	1024 MB LPDDR3	Shared memory for CPU and GPU
Storage	8192 MB eMMC	GPT partitioned with A/B slots
GPU	Adreno 306	64bit 600 MHz
DSP	Hexagon 685	Audio/sensor processing
EEPROM	16KB AT24	I2C access
WiFi Radio #0	QCN9074 (Pine)	PCIe bus
WiFi Radio #1	WCN3980	SNOC bus

2.3 Device Tree Hierarchy

```
qcs404.dtsi
    ↓ includes
qcs405.dtsi
    ↓ includes
qcs405-8devices-tobufi.dtsi
```

TobuFi DTS: `qcs405-8devices-tobufi.dtsi` defines SoM-level hardware

2.4 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details

3. Robonode Board

3.1 Overview

Robonode is a development board for unmanned systems based on the TobuFi System-on-Module featuring Qualcomm QCS405 SoC. This document describes board-level hardware integration and peripherals.

3.2 Peripherals

- WiFi Radio #0 (HE/11ax)
- WiFi Radio #1 (VHT/11ac)
- Ethernet port (100M/1000M)
- USB Port #1 (USB 2.0 DRD/OTG)
- USB Port #2 (USB 3.0 host)
- I2C EEPROM (16KB)
- 4x BLSP UART/I2C
- 3x GPIO LEDs
- GPIO Push button
- GPIO Switch toggle
- Reboot button
- USB recovery button

3.3 Board Layout

3.3.1 Top View

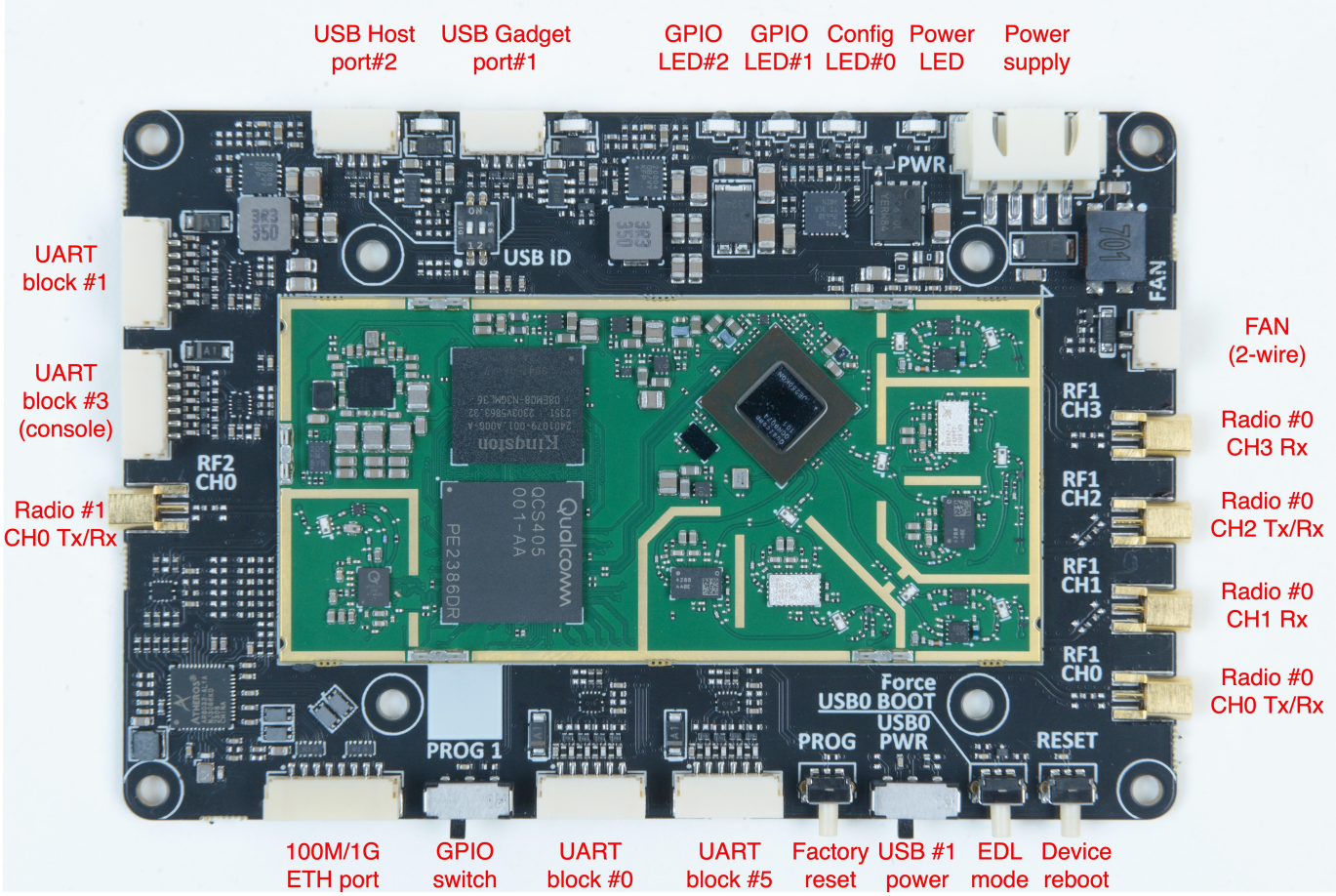


Figure 1: Robonode board top view components

3.3.2 Bottom View

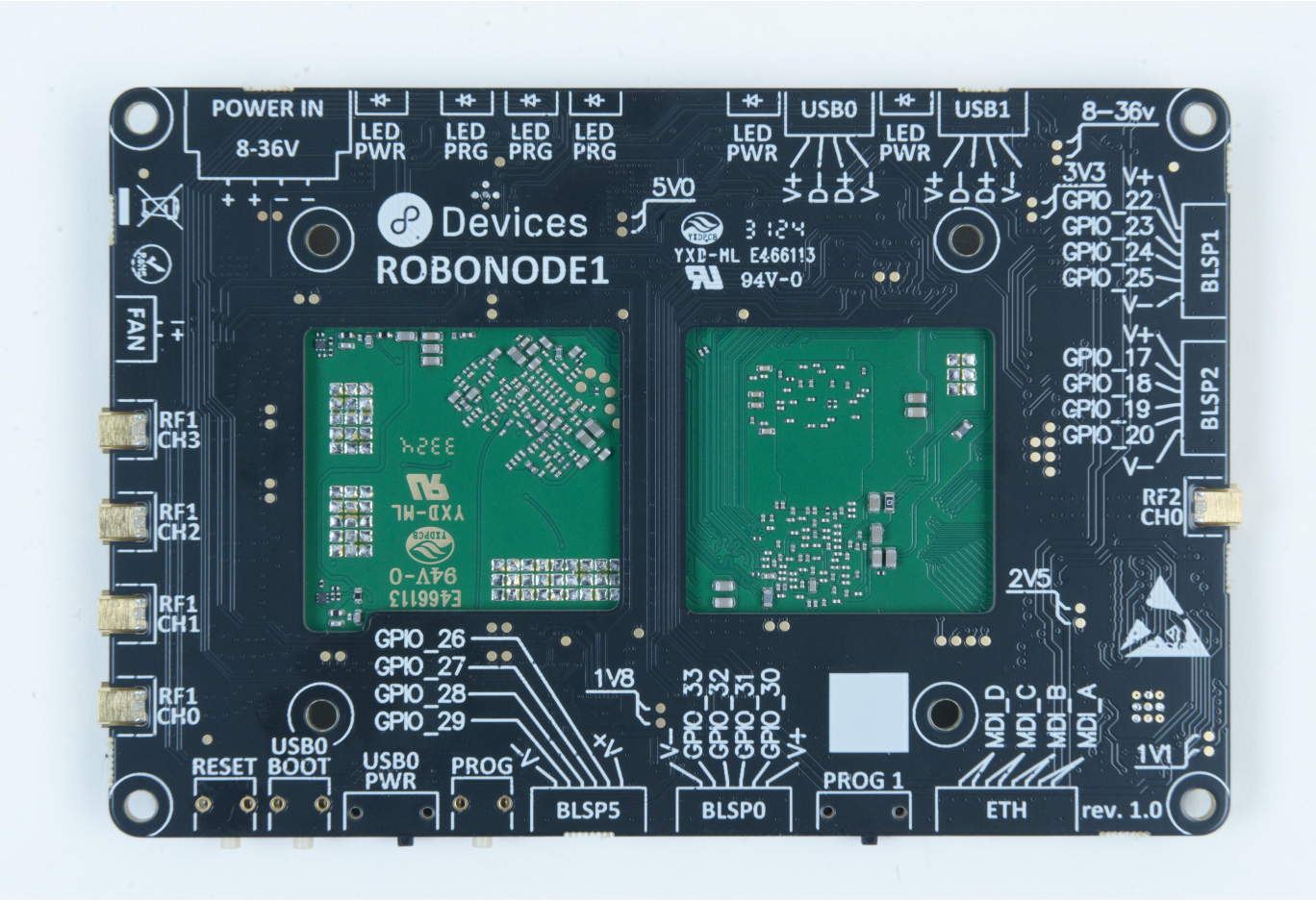


Figure 2: Robonode board bottom view components

3.3.3 Connector Identification

Top Side:

- **Ethernet Port**: RJ45 connector, labeled "ETH"
- **USB Port #1**: USB-C connector (left side)
- **USB Port #2**: USB-C connector (right side)
- **WiFi Antennas**: 4x MMCX (Radio #0), 1x MMCX (Radio #1)
- **Power Connector**: DC barrel jack (8-36V)

Bottom Side:

- **BLSP0 Header**: 4-pin header for UART/I2C (GPIOs 30-33)
- **BLSP1 Header**: 4-pin header for UART/I2C (GPIOs 22-25)
- **BLSP2 Header**: 4-pin header for UART/I2C (GPIOs 17-20), includes serial console
- **BLSP5 Header**: 4-pin header for UART/I2C (GPIOs 26-29)
- **LED0, LED1, LED2**: User-defined GPIO LEDs
- **Push Button**: User-defined GPIO button
- **Slide Switch**: User-defined GPIO switch
- **Reboot Button**: System reset button
- **Recovery Button**: EDL mode entry button

3.4 Board Interfaces

3.4.1 I2C EEPROM

Chip: AT24C128C **Capacity:** 16KB **Interface:** BLSP1 I2C (GPIO 24-25)

Non-volatile storage for board identification and production storage. Accessible through standard Linux I2C interface.

3.4.2 WiFi Radio #0 (Pine)

Parameter	Description
Radio Chip	PCIe/QCN9074
WiFi Standard	IEEE 802.11ax (WiFi 6/6E)
Channel Width	20/40/80/160 MHz
Operation Band	2GHz + 5GHz or 2GHz + 6GHz
Operation Mode	2x4 (2T4R)
Antenna Connectors	4x MMCX
Max Conducted 2GHz Tx Power (aggregate)	32 dBm
Max Conducted 5GHz Tx power (aggregate)	29 dBm
Max Conducted 6GHz Tx Power (aggregate)	29 dBm
2GHz Frequency Range	2360–3150 MHz
5GHz Frequency Range	4550–6630 MHz
6GHz Frequency Range	5325–7495 MHz

High-performance WiFi radio supporting WiFi 6E (802.11ax) with MIMO capability. Suitable for high-throughput wireless applications, mesh networking, and dual-band simultaneous operation.

3.4.3 WiFi Radio #1

Parameter	Description
Radio Chip	SNOC/WCN3980
WiFi Standard	IEEE 802.11ac (WiFi 5)
Channel Width	20/40/80 MHz
Operation Band	2GHz + 5GHz
Operation Mode	1x1 (1T1R)
Antenna Connectors	1x MMCX
Max 2GHz Tx Power	16 dBm
Max 5GHz Tx power	16 dBm
2.4 GHz Frequency Range	2412-2484 MHz
5 GHz Frequency Range	5180-5825 MHz

Integrated WiFi radio supporting dual-band operation. Suitable for management interface, station mode connectivity, or additional access point deployment.

3.4.4 Ethernet Port

Parameter	Description
Trancever Chip	RGMII/AR8033A
Speed	10/100/1000 Mbps
Duplex	Half/Full
MDI/MDI-X	Auto-crossover

Gigabit Ethernet port with RJ45 connector. Supports network connectivity for wired applications, device management, and high-bandwidth data transfer.

3.4.5 USB Port #1

Parameter	Description
Standard	USB 2.0
Speed	480 Mbps (HS)
Connector	USB-C
Mode	DRD (Host/Gadget/OTG)
Power	Switch-controlled
Identifier	usb0

Dual-role port capable of host, device and OTG modes with manual power switch control. In device mode, provides access to ADB debugging interface for development and configuration. In host mode, supports standard USB peripherals.

USB Port #1 Power Switch

Slide switch controls USB mode and power supply. When toggled on USB port is working in host controller mode, and port is powered locally, when toggled off USB port is working in device mode and it expects to be powered from the host device.

Enable USB power when desirable to use port to connect additional periphery, i.e. USB camera. Keep USB power disabled to keep port in device mode, i.e. for ADB access, USB storage or USB ethernet adapter mode.

3.4.6 USB Port #2

Parameter	Description
Standard	USB 2.0
Speed	480 Mbps (HS)
Connector	USB-C
Mode	Host only
Power	Always powered
Identifier	usb1

Host-only port for connecting USB flash drives, USB cameras, and other standard USB peripherals like mice and keyboards.

3.4.7 LED #0

GPIO: 47

Application controlled LED dedicated for system configuration state feedback. LED is fully controlled by application software therefore may be repurposed on demand, changing application source code and rebuilding software.

3.4.8 LED #1

GPIO: 48

Software controlled LED. Currently unused, dedicated for future application needs.

3.4.9 LED #2

GPIO: 52

Software controlled LED, currently unused, dedicated for future application needs.

For LED control via sysfs and device tree configuration, see GPIO Configuration.

3.4.10 GPIO Push Button

GPIO: 58

Software controlled push button. Pressing and holding the button for at least 10 seconds triggers factory reset, resetting device configuration to factory defaults. This is also indicated by the configuration LED slowly blinking while the button is being held.

3.4.11 GPIO Switch Toggle

GPIO: 51

Software controlled slide switch, currently unused, dedicated for future application needs.

For button and switch programming via Linux input subsystem, see GPIO Configuration.

3.4.12 Reboot Button

Pressing and releasing the button restarts the device.

3.4.13 USB Recovery Button

Hold the button pressed when power cycling or rebooting to enter EDL (Emergency Download) mode for software recovery or initial device flashing.

3.5 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- TobuFi SoM - TobuFi SoM (System-on-Module) details
- Robonode GPIO - Board GPIO interfaces and control
- Robonode BLSP - Board UART and I2C interfaces
- Robonode Initialization - Board initialization sequence

4. Robonode Setup

This guide covers initial device setup and hands-on launch procedures for the Robonode board.

For board hardware specifications and connector locations, refer to Robonode Hardware.

4.1 Required Equipment

- Compatible DC power supply
- For console access: Serial console cable (3.3V TTL UART)
- For console access: Serial terminal software (minicom, screen, PuTTY, etc.)
- For device recovery: USB Type-C cable
- For device recovery: Linux PC with qdl tool installed or Windows PC with QFIL

4.2 Power Requirements

The Robonode board requires a DC power supply ranging from **8V to 36V** capable of supplying ~24W peak power.

Power On:

- Connect power supply to the power connector
- Device automatically starts up

Refer to Robonode Hardware for power connector location on the PCB.

4.3 Serial Console Access

To access the device via serial console:

- Connect 3.3V TTL serial console cable to the serial console connector
- Configure serial terminal with appropriate settings
- Follow System Access Guide for connection details

Refer to Robonode Hardware for serial console connector location on the PCB.

4.4 USB Recovery

4.4.1 Emergency Download (EDL) Mode

EDL (Emergency Download) mode is a low-level boot mode in the Qualcomm SoC that enables device recovery and initial programming by exposing a USB interface for flash programming.

Use cases:

- Initial factory programming of blank devices
- Recovery from bootloader corruption
- Full device reflashing
- Partition table restoration

Device appears as USB `05c6:9008` (Qualcomm HS-USB QDLoader 9008) in EDL mode.

Important: EDL recovery installs legacy software version v0.x which can subsequently be upgraded to open source software version v1.x using USB flashing (see USB Flashing section).

Warning: EDL recovery completely erases and reprograms device flash. Use correct recovery images for Robonode hardware to avoid permanent damage.

4.4.2 Entering EDL Mode

- Connect USB gadget port#1 to PC
- Press and hold "EDL mode" button
- Press and release "Device reboot" button
- Release "EDL mode" button
- Device enters EDL mode and appears as `05c6:9008`

Refer to Robonode Hardware for button locations on the PCB.

4.4.3 Obtaining Recovery Package

Recovery flash image packages are available from 8devices support. Contact 8devices support to obtain the appropriate recovery package for your device.

4.4.4 EDL Recovery Tools

EDL recovery can be performed using different tools depending on your operating system:

Linux:

- Recommended: `edl_flash.py` script (included in recovery package, Linux only)
- Alternative: QDL tool from https://github.com/linux-msm/qdl

Windows:

- QFIL (Qualcomm Flash Image Loader) from Qualcomm Product Support Tools (QPST)

4.4.5 EDL Recovery Using edl_flash.py (Linux only)

The `edl_flash.py` script provides automated EDL recovery on Linux systems.

Recovery Steps:

- Connect USB gadget port#1 to PC
- Enter EDL recovery mode (see Entering EDL Mode)
- Execute EDL flashing script from software flash image package:

```
python3 edl_flash.py
```

- Power cycle the device to boot new firmware

Multiple EDL Devices:

When multiple devices are in EDL state, specify device by serial number:

```
# Flash specific device by serial
python3 edl_flash.py <serial>

# List available EDL devices
python3 edl_flash.py -l
```

4.4.6 EDL Recovery Using QDL (Linux)

Alternative method using QDL tool:

```
# Verify device enumeration
lsusb | grep 05c6:9008

# Flash recovery images
qdl --storage ufs prog_firehose.mbn rawprogram.xml patch.xml
```

Required images: prog_firehose.mbn (programmer), rawprogram.xml (partition layout), patch.xml (patching instructions), partition images (bootloader, kernel, rootfs).

4.4.7 EDL Recovery Using QFIL (Windows)

Use QFIL from Qualcomm Product Support Tools package. Refer to QFIL documentation for flashing procedures.

4.5 USB Flashing

Fastboot provides partition-level flashing capabilities through the bootloader interface. This method is used to upgrade devices from legacy software v0.x to open source software v1.x.

4.5.1 Prerequisites

- ADB (Android Debug Bridge) and Fastboot tools
- Windows: USB driver for Android devices

4.5.2 Fastboot Flashing Using fastboot_flash.py

The `fastboot_flash.py` script automates fastboot flashing operations. The script automatically detects device state and enters fastboot mode if needed.

Flashing Procedure:

```
# Flash boot and system partitions
python3 fastboot_flash.py --boot boot-image.img --system rootfs-image.img
```

Multiple Devices:

When multiple devices are connected, specify device by serial number:

```
# Specify device by serial number
python3 fastboot_flash.py <serial> --boot boot.img --system rootfs.img

# Check available devices
fastboot devices
```

The script will:

- Automatically detect if device is in ADB mode and reboot to fastboot
- Wait for device to appear in fastboot mode (up to 120 seconds)
- Flash specified partitions
- Automatically reboot device after successful flashing

4.6 Related Documentation

- Robonode Hardware - Board hardware periphery briefing
- Robonode Initialization - Board hardware initialisation briefing
- Robonode BLSP - Board UART and I2C interfaces

5. Robonode BLSP Configuration

5.1 BLSP Endpoints

Robonode board utilizes QCS405 BLSP (BAM Low-Speed Peripheral) controllers for UART and I2C connectivity.

BLSP #	GPIOs	Function	Endpoints
0	30-33	UART	Header
0	32-33	I2C	Header
1 (A)	22-23	UART	Header
1 (A)	24-25	I2C	Header, AT24
1 (B)	47-50	GPIO	-
2 (A)	17-18	UART	Serial console
2 (A)	19-20	I2C	Header
2 (B)	39-42	GPIO	-
3	82-85	UART	WCN3980
4	37-38, 117-118	-	Not exported
5	26-29	UART	Header
5	28-29	I2C	Header

5.2 UART

5.2.1 Hardware Characteristics

Ports Specification:

Parameter	Value	Notes
Maximum Baud Rate	4 Mbps	Limited by clock divider
FIFO Depth	64 bytes	TX and RX
Data Bits	5-8	Configurable
Stop Bits	1-2	Configurable
Parity	None/Even/Odd	Configurable
Flow Control	RTS/CTS hardware	4-wire mode only

Clock Configuration:

```
// UART clock calculation
// freq = source_clock / (divider * 16)
// For 115200 baud:
// 115200 = 7372800 / (4 * 16)

#define UART_CLK_SOURCE   7372800  // 7.3728 MHz
#define BAUD_115200_DIV   4        // Divider for 115200
#define BAUD_921600_DIV   0.5      // Not integer, use closest
```

5.2.2 Hardware Mapping

UART interfaces are mapped to Linux TTY devices.

TTY Device	BLSP #	GPIOs	Endpoint	DTS Node
/dev/ttyMSM0	2 (A)	17-18	Serial console	blsp1_uart2
/dev/ttyMSM1	0	30-33	Header	blsp1_uart0
/dev/ttyMSM2	1 (A)	22-23	Header	blsp1_uart1
/dev/ttyMSM3	5	26-29	Header	blsp2_uart0
/dev/ttyMSM4	3	82-85	WCN3980 (internal)	blsp1_uart3

Note: ttyMSM0 is the primary debug serial console used by kernel for logging.

5.2.3 DTS Configuration

UART ports are configured in Robonode DTS (`qcs405-8devices-robonode.dts`). To enable or disable a port:

```
&blsp1_uart0 {
        status = "okay";     /* enable */
};

&blsp1_uart1 {
        status = "disabled"; /* disable */
};
```

5.2.4 2-Wire UART Configuration

By default, UART ports use 4-wire configuration (TX, RX, CTS, RFR). To configure 2-wire UART (TX, RX only), update the pinmux configuration in board DTS:

```
&tlmm {
        blsp1_uart0_2wire: blsp1-uart0-2wire-state {
                pins = "gpio30", "gpio31";  /* TX, RX only */
                function = "blsp_uart0";
        };
};

&blsp1_uart0 {
        pinctrl-0 = <&blsp1_uart0_2wire>;
        status = "okay";
};
```

5.2.5 Serial Port Ordering

The Linux TTY device numbering (`/dev/ttyMSMx`) is determined by device tree aliases defined in `qcs405-8devices-robonode.dts`:

```
aliases {
        serial0 = &blsp1_uart2;
        serial1 = &blsp1_uart0;
        serial2 = &blsp1_uart1;
        serial3 = &blsp2_uart0;
};
```

The alias name suffix (serial0, serial1, etc.) directly maps to the TTY device number (ttyMSM0, ttyMSM1, etc.). UART ports without aliases receive device numbers dynamically during kernel initialization.

5.2.6 Usage Example

To enable serial console on ttyMSM1 (BLSP0, GPIO 30-33), connect serial TTL cable to UART Header block #0 and execute:

```
/sbin/agetty -n -a root --noclear ttyMSM1
```

Configure UART Parameters via stty:

```
# Set baud rate to 115200, 8N1
stty -F /dev/ttyMSM1 115200 cs8 -cstopb -parenb

# Enable hardware flow control (RTS/CTS)
stty -F /dev/ttyMSM1 crtscts

# Check current configuration
stty -F /dev/ttyMSM1 -a
```

5.2.7 Debugging

Check UART Status:

```
# View UART driver statistics
cat /proc/tty/driver/msm_serial

# Check for overrun/framing errors
dmesg | grep -i "msm_serial\|uart"
```

Common UART Issues:

Symptom	Cause	Solution
Garbled output	Baud rate mismatch	Verify both sides use same baud
No output	Wrong TX/RX pins	Check pinmux configuration
Data loss	FIFO overflow	Enable flow control or reduce baud
Intermittent errors	Signal integrity	Check cable length, add termination

5.3 I2C

5.3.1 Hardware Characteristics

Supported operation modes:

Mode	Speed	Notes
Standard Mode	100 kHz	Default speed
Fast Mode	400 kHz	Common speed
Fast Mode Plus	1 MHz	Maximum supported

5.3.2 Hardware Mapping

I2C interfaces available on Robonode board. Only configured buses appear as Linux I2C devices.

I2C Device	BLSP #	GPIOs	Endpoint	DTS Node
/dev/i2c-X	0	32-33	Header	blsp1_i2c0
/dev/i2c-0	1 (A)	24-25	AT24C128C EEPROM	blsp1_i2c1
/dev/i2c-1	2 (A)	19-20	Header	blsp1_i2c2
/dev/i2c-X	5	28-29	Header	blsp2_i2c0

5.3.3 DTS Configuration

I2C buses are configured in Robonode DTS (`qcs405-8devices-robonode.dts`). To enable or disable a bus:

```
&blsp1_i2c1 {
        status = "okay";     /* enable */
};

&blsp1_i2c2 {
        status = "disabled"; /* disable */
};
```

5.3.4 Usage Examples

Detect I2C devices on bus 0:

```
i2cdetect -y 0

i2cget -y 0 0x56 0x00
```

5.3.5 Debugging

Common I2C Issues:

Symptom	Cause	Solution
No ACK	Device not present	Check address, power, connections
Bus lockup	SDA/SCL stuck low	Power cycle, check pull-ups
Data corruption	Clock stretching	Reduce bus speed
Timeout	Slow device	Increase timeout in driver

5.4 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- Robonode Hardware - Board hardware periphery briefing
- Robonode GPIO - Board GPIO interfaces and control
- Robonode Initialization - Board hardware initialisation briefing

6. Robonode GPIO Configuration

6.1 Overview

Robonode board provides general-purpose I/O interfaces for onboard devices including LEDs, buttons, fan control, and USB power management. GPIOs are accessed through the QCS405 TLMM (Top Level Mode Multiplexer) controller.

6.1.1 Allocation Summary

GPIO	Function	Direction	Default State	Pull Config	Drive Strength	Notes
15	USB2_VBUS	Output	Low	None	8 mA	USB p power contro
16	USB3_VBUS	Output	Low	None	8 mA	USB p power contro
17	UART2_TX	Output	High	None	8 mA	Serial conso
18	UART2_RX	Input	-	Pull-up	2 mA	Serial conso
19	I2C2_SCL	Bidir	High	None (ext)	2 mA	Heade
20	I2C2_SDA	Bidir	High	None (ext)	2 mA	Heade
22	UART1_TX	Output	High	None	8 mA	Heade GPIO
23	UART1_RX	Input	-	Pull-up	2 mA	Heade GPIO
24	I2C1_SCL	Bidir	High	None (ext)	2 mA	EEPRO
25	I2C1_SDA	Bidir	High	None (ext)	2 mA	EEPRO
26	UART5_TX	Output	High	None	8 mA	Heade
27	UART5_RX	Input	-	Pull-up	2 mA	Heade
28	I2C5_SCL	Bidir	High	None (ext)	2 mA	Heade functi
29	I2C5_SDA	Bidir	High	None (ext)	2 mA	Heade functi
30	UART0_TX	Output	High	None	8 mA	Heade
31	UART0_RX	Input	-	Pull-up	2 mA	Heade
32	UART0_CTS	Input	-	Pull-up	2 mA	Heade I2C0_S
33	UART0_RFR	Output	High	None	8 mA	

GPIO	Function	Direction	Default State	Pull Config	Drive Strength	Notes
						Heade I2C0_S
39	FAN_CTRL	Output	Low	Pull-down	8 mA	Coolin
47	LED0	Output	High	None	8 mA	User L
48	LED1	Output	High	None	8 mA	User L
51	SWITCH	Input	-	Pull-up	2 mA	Slide s
52	LED2	Output	High	None	8 mA	User L
58	BUTTON	Input	-	Pull-up	2 mA	Push b
82	UART3_TX	Output	High	None	8 mA	WCN3 (intern
83	UART3_RX	Input	-	Pull-up	2 mA	WCN3 (intern
84	UART3_CTS	Input	-	Pull-up	2 mA	WCN3 (intern
85	UART3_RFR	Output	High	None	8 mA	WCN3 (intern

6.2 LEDs

Three GPIO-controlled LEDs are available on Robonode board.

LED Label	GPIO	Linux Device	Default Trigger
LED0	47	led0	default-on
LED1	48	led1	default-on
LED2	52	led2	default-on

DTS Configuration:

```
&tlmm {
        gpio_leds: gpio-leds-pins {
                pins = "gpio47", "gpio48", "gpio52";
                function = "gpio";
```

```
        };
};

/ {
        gpio-leds {
                compatible = "gpio-leds";

                led@47 {
                        label = "led0";
                        gpios = <&tlmm 47 GPIO_ACTIVE_HIGH>;
                        linux,default-trigger = "default-on";
                };
        };
};
```

Usage Example:

```
# Turn LED0 on
echo 1 > /sys/class/leds/led0/brightness

# Turn LED0 off
echo 0 > /sys/class/leds/led0/brightness

# Set LED trigger to heartbeat
echo heartbeat > /sys/class/leds/led0/trigger

# List available triggers
cat /sys/class/leds/led0/trigger
```

6.3 Buttons and Switches

Two GPIO input devices: push button and slide switch.

Device	GPIO	Linux Input	Active Level
Push Button	58	BTN_0	Active Low
Slide Switch	51	BTN_1	Active High

DTS Configuration:

```
&tlmm {
        gpio_keys: gpio-keys-pins {
                pins = "gpio51", "gpio58";
                function = "gpio";
                input-enable;
        };
```

```
};

&soc {
        gpio-keys {
                compatible = "gpio-keys";

                push@58 {
                        label = "Push Button";
                        gpios = <&tlmm 58 GPIO_ACTIVE_LOW>;
                        linux,code = <BTN_0>;
                };
        };
};
```

Usage Example:

Button and switch events are exposed through the Linux input subsystem as event devices (`/dev/input/event*`). Applications can read events using standard input APIs.

Monitor input events from command line:

```
# List all input devices
cat /proc/bus/input/devices

# Monitor events using hexdump
hexdump /dev/input/event0

# Monitor events using od
od -t x1 /dev/input/event0
```

Input events can be programmatically accessed using standard Linux input API (`linux/input.h`). Each button press/release generates an input event with type `EV_KEY` and the corresponding key code (`BTN_0` or `BTN_1`).

6.4 Fan Control

GPIO-controlled fan with thermal zone integration for automatic temperature management.

Device	GPIO	Linux Interface	Speed Levels
Fan	39	gpio_fan	0 (off), 1 (on)

DTS Configuration:

```
&tlmm {
        gpio_fan_pins: gpio-fan-pins {
                pins = "gpio39";
                function = "gpio";
        };
};

&soc {
        gpio_fan: gpio-fan {
                compatible = "gpio-fan";
                gpios = <&tlmm 39 GPIO_ACTIVE_HIGH>;
                gpio-fan,speed-map = <0 0>, <1 1>;
        };
};

&{/thermal-zones/cluster-thermal} {
        trips {
                cluster_alert0_fan: trip-point0-fan {
                        temperature = <90000>;
                        hysteresis = <10000>;
                        type = "active";
                };
        };
        cooling-maps {
                map1 {
                        trip = <&cluster_alert0_fan>;
                        cooling-device = <&gpio_fan 0 1>;
                };
        };
};
```

The fan automatically activates when CPU temperature exceeds 90°C and deactivates when temperature drops below 80°C (90°C - 10°C hysteresis).

Manual Fan Control:

```
# Find fan device
ls /sys/class/hwmon/

# Check current fan state (0=off, 1=on)
cat /sys/class/hwmon/hwmon*/fan1_input

# Manual control (requires disabling thermal management)
echo 1 > /sys/class/hwmon/hwmon*/pwm1
```

Thermal Status:

```
# Check CPU temperature (in millidegrees Celsius)
cat /sys/class/thermal/thermal_zone*/temp

# Check thermal zone type
cat /sys/class/thermal/thermal_zone*/type

# View trip points
cat /sys/class/thermal/thermal_zone*/trip_point_*_temp
```

6.5 USB VBUS Control

GPIO-controlled USB port power management.

USB Port	GPIO	Control
USB2	15	VBUS boost power control
USB3	16	VBUS boost power control

DTS Configuration:

```
&tlmm {
        usb2_vbus_boost: usb2-vbus-boost {
                pins = "gpio15";
                function = "gpio";
        };

        usb3_vbus_boost: usb3-vbus-boost {
                pins = "gpio16";
                function = "gpio";
        };
};
```

Note: USB VBUS control is typically managed by USB subsystem. Manual control via sysfs GPIO interface should be used with caution.

6.6 Headers

GPIO pins available on board header connectors.

Header	GPIOs	Active Function
BLSP0	30-33	
BLSP1	22-25	
BLSP2	17-18, 19-20	UART (serial console)
BLSP5	26-29	

For detailed UART and I2C configuration, see BLSP Configuration.

DTS Configuration:

To configure header GPIOs as general-purpose I/O, define pinmux configuration in board DTS:

```
&tlmm {
        gpio_header: gpio-header-state {
                pins = "gpio30", "gpio31";
                function = "gpio";
        };
};
```

Usage Example:

```
# Export GPIO 30
echo 30 > /sys/class/gpio/export

# Configure as output
echo out > /sys/class/gpio/gpio30/direction

# Set high
echo 1 > /sys/class/gpio/gpio30/value

# Set low
echo 0 > /sys/class/gpio/gpio30/value

# Unexport when done
echo 30 > /sys/class/gpio/unexport
```

6.7 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- Robonode Hardware - Board hardware periphery briefing

- Robonode BLSP - Board UART and I2C interfaces
- Robonode Initialization - Board hardware initialisation briefing
- Linux GPIO Sysfs Interface
- Linux LED Subsystem
- Linux Input Subsystem
- Linux Thermal Framework
- Linux Hardware Monitoring

7. Robonode Initialisation

This document describes hardware component initialization essentials required for Robonode device boot. It focuses on production data retrieval, hardware resource provisioning, and driver initialization.

7.1 Production EEPROM

The device EEPROM isprovisioned with hardware-specific identity and configuration data. This data is read during boot initialization to configure network interfaces and radio calibration.

7.1.1 EEPROM Device

Character device: /sys/bus/i2c/devices/0-0056/eeprom

```
# Detect I2C devices on bus 0
i2cdetect -y 0
#     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
# 00:          -- -- -- -- -- -- -- -- -- -- -- -- --
# 10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 50: -- -- -- -- -- -- 56 -- -- -- -- -- -- -- -- --
# 60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 70: -- -- -- -- -- -- -- --

# Read single byte from EEPROM address 0x00
i2cget -y 0 0x56 0x00
```

7.1.2 EEPROM Data Structure

- Storage size: 16384 bytes (16 KB)
- Data offset: 512 bytes
- Enabled data models: TobuFi sys-eeprom TLV, RoboSoft TLV

```
┌──────────────────────────────────────────────┐
│         EEPROM Memory Layout (16KB)          │
├──────────────────────────────────────────────┤
│ 0x0000 - 0x01FF (512 bytes)                  │
│   Reserved for bootloader/factory use        │
├──────────────────────────────────────────────┤
│ 0x0200 - 0x3FFF (15872 bytes)                │
│   TLV storage area (production data)         │
│   - Product identification                   │
│   - PCB information                          │
```

```
|   - MAC addresses                          |
|   - Radio calibration data                 |
└────────────────────────────────────────────┘
```

7.1.3 EEPROM Fields

Production EEPROM contains board metadata programmed during manufacturing:

Field	Description
`PRODUCT_ID`	Product identifier
`PCB_REVISION`	PCB hardware revision
`PCB_NAME`	PCB design name
`PCB_SN`	PCB serial number
`PCB_PROD_DATE`	Production date
`PCB_PROD_LOCATION`	Production facility
`SERIAL_NO`	Device serial number
`MAC_ADDR_eth0`	Ethernet MAC address
`MAC_ADDR_wlan1`	WiFi Radio #2 MAC address
`RADIO_CALIBRATION_DATA`	WiFi Radio #2 calibration (binary)

7.1.4 EEPROM Data Retrieval

Service: `eeprom-dump.service` (sysinit.target phase) **Utility**: `tlvs` **Output**: `/etc/board.conf`

The `eeprom-dump` service uses `tlvs` utility to read hardware EEPROM and generate `/etc/board.conf` with shell variables.

Example /etc/board.conf:

```
PRODUCT_ID=Robonode
PCB_REVISION=01
PCB_NAME=ROBONODE_V1
PCB_SN=1234567890
PCB_PROD_DATE=2025-01-15
PCB_PROD_LOCATION=Factory
SERIAL_NO=RN0987654321
MAC_ADDR_eth0=00:03:7F:12:90:8F
MAC_ADDR_wlan1=00:03:7F:12:90:91
```

7.2 Ethernet Initialization

7.2.1 Hardware

Driver: `stmmac` **Interface**: `eth0`

7.2.2 MAC Address Provisioning

`eeprom-dump.service` provisions Ethernet MAC address through kernel module parameter:

File: `/etc/modprobe.d/stmmac_mac.conf`

```
options dwmac_qcom_ethqos mac_addr=00:03:7F:12:90:8F
```

7.2.3 Initialization Sequence

- `eeprom-dump.service` reads `MAC_ADDR_eth0` from EEPROM
- Creates `/etc/modprobe.d/stmmac_mac.conf`
- `systemd-modules-load.service` loads `stmmac` driver
- Driver initializes with configured MAC
- `eth0` interface available

7.3 WiFi Radio #1 (PCIe Radio)

7.3.1 Hardware

Driver: `ath11k_pci` **Bands**: 2.4 GHz / 5 GHz **Interface**: e.g., `wlan0`

7.3.2 Firmware Requirements

Location: `/lib/firmware/ath11k/<hw_version>/`

- `board-2.bin` - Board Data File (BDF)
- `firmware-*.bin` - Firmware image

7.3.3 Board Data File Variants

BDF variant mechanism supports band-specific board data:

Parameter: `bdf_variant` **Values**: `2` (2.4 GHz), `5` (5 GHz)

Configuration:

```
options ath11k bdf_variant=5
```

Driver searches for `board-2_5.bin` or falls back to `board-2.bin`.

Band switching requires driver reload with different `bdf_variant`. The `radioconf` utility manages band configuration.

7.3.4 Initialization Sequence

- `systemd-modules-load.service` loads `ath11k_pci`
- Driver probes PCIe device
- Firmware and BDF loaded
- Radio initialized
- Wireless interface created

Initialization Timing:

Phase	Duration	Details
PCIe enumeration	~200ms	Bus scan, device detection
Firmware download	~1.5s	Load amss.bin, m3.bin
BDF loading	~300ms	Board data file
Radio calibration	~500ms	Internal calibration
Interface creation	~200ms	Network stack registration
Total	**~2.7s**	From module load to interface ready

Driver Probe Sequence:

```
// Simplified ath11k_pci driver probe flow

int ath11k_pci_probe(struct pci_dev *pdev) {
    // 1. Enable PCIe device
    pci_enable_device(pdev);
    pci_set_master(pdev);

    // 2. Map BAR regions
    pci_request_regions(pdev, "ath11k_pci");
```

```
    // 3. Enable MSI/MSI-X interrupts
    pci_enable_msi_range(pdev, 32, 32);

    // 4. Initialize QMI (Qualcomm Messaging Interface)
    ath11k_qmi_init();

    // 5. Load firmware
    request_firmware("amss.bin");
    request_firmware("m3.bin");

    // 6. Download firmware to radio
    ath11k_qmi_firmware_download();

    // 7. Load BDF
    ath11k_core_fetch_bdf();
    ath11k_qmi_bdf_download();

    // 8. Radio power-on and calibration
    ath11k_core_start();

    // 9. Register with mac80211
    ieee80211_register_hw();

    return 0;
}
```

7.4 WiFi Radio #2 (Integrated Radio)

7.4.1 Hardware

Radio: WCN3990 integrated WiFi/BT **Driver**: `ath10k_snoc` **Bands**: 2.4 GHz / 5 GHz **Interface**: `wlan1`

7.4.2 Firmware and Calibration

Firmware: `/lib/firmware/ath10k/WCN3990/hw1.0/` **Calibration**: `/lib/firmware/ath10k/cal-snoc-a000000.wifi.bin`

Radio calibration data is extracted from EEPROM `RADIO_CALIBRATION_DATA` field by `eeprom-dump.service` and written to calibration file.

7.4.3 MAC Address Provisioning

`eeprom-dump.service` provisions MAC address through kernel module parameter:

File: `/etc/modprobe.d/ath10k_mac.conf`

```
options ath10k_snoc mac_addr=00:03:7F:12:90:91
```

7.4.4 Initialization Sequence

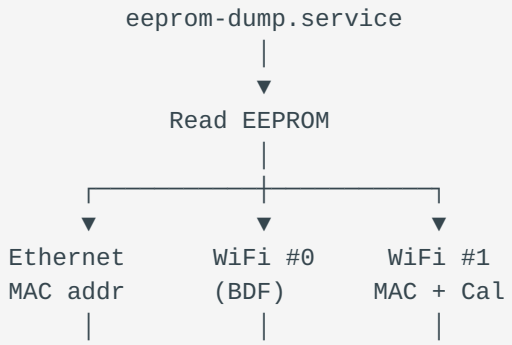
- `eeprom-dump.service` extracts:
- `MAC_ADDR_wlan1` → `/etc/modprobe.d/ath10k_mac.conf`
- `RADIO_CALIBRATION_DATA` → `/lib/firmware/ath10k/cal-snoc-a000000.wifi.bin`
- `systemd-modules-load.service` loads `ath10k_snoc`
- Driver initializes with MAC and calibration
- `wlan1` interface available

Initialization Timing:

Phase	Duration	Details
Platform device probe	~100ms	SNOC bus enumeration
MAC address setup	~50ms	Read from modprobe.conf
Calibration load	~200ms	Load cal-*.bin file
Firmware download	~800ms	Load firmware image
Radio initialization	~300ms	Hardware setup
Interface creation	~150ms	Network stack registration
Total	**~1.6s**	From module load to interface ready

7.5 Boot Sequence

Hardware initialization during **sysinit.target**:




```
EEPROM data
├── Provisions /etc/modprobe.d/stmmac_mac.conf
├── Provisions /etc/modprobe.d/ath10k_mac.conf
└── Extracts /lib/firmware/ath10k/cal-snoc-a000000.wifi.bin
```

After sysinit.target, all hardware interfaces available for ESConf configuration.

7.6 System Information Retrieval

7.6.1 Device Serial Number

The device serial number is embedded in the kernel command line during boot:

```
# Get serial number from kernel command line
grep -o 'serial_num=[^ ]*' /proc/cmdline | cut -d'=' -f2
```

7.6.2 Active Boot Slot

The currently active A/B partition slot is determined during boot:

```
# Get active slot (_a or _b)
grep -o 'SLOT_SUFFIX=_[ab]' /proc/cmdline | cut -d'=' -f2
```

This information indicates which partition set (A or B) the system booted from, used for understanding update state and determining the target slot for software updates.

7.7 Troubleshooting

7.7.1 EEPROM Read Failures

Symptom:

```
eeprom-dump.service: Main process exited, code=exited, status=1/FAILURE
```

Diagnosis:

```
# Check I2C bus availability
ls -l /sys/bus/i2c/devices/

# Expected: 0-0056 directory exists
# If missing, I2C controller or EEPROM not detected

# Check I2C controller driver
lsmod | grep i2c
# Should show: i2c_qup (Qualcomm I2C driver)

# Manual I2C scan
i2cdetect -y 0
# Should show device at address 0x56

# Attemt to read EEPROM
i2cget -y 0 0x56 0x00
# Should return data
```

7.7.2 WiFi Radio Initialization Failures

PCIe Radio (QCN9074) Firmware Load Failure:

```
# Check PCIe enumeration
lspci -v | grep -A5 "Network controller"

# Check firmware files
ls -l /lib/firmware/ath11k/QCN9074/hw1.0/
# Required: amss.bin, m3.bin, board-2.bin

# Check driver logs
dmesg | grep ath11k_pci
# Look for: "firmware load failed" or "BDF download failed"
```

Integrated Radio (WCN3980) Calibration Failure:

```
# Check calibration file
ls -l /lib/firmware/ath10k/cal-snoc-a000000.wifi.bin
# Should be 128-256 bytes

# Validate calibration data
hexdump -C /lib/firmware/ath10k/cal-snoc-a000000.wifi.bin | head
# Should NOT be all 0xFF or 0x00

# Check driver logs
dmesg | grep ath10k_snoc
# Look for: "failed to fetch calibration data"
```

7.7.3 MAC Address Provisioning Issues

Ethernet MAC Not Applied:

```
# Check modprobe configuration
cat /etc/modprobe.d/stmmac_mac.conf
# Should contain: options dwmac_qcom_ethqos mac_addr=XX:XX:XX:XX:XX:XX

# Check actual MAC address
ip link show eth0
# link/ether should match EEPROM value

# If mismatch, reload driver:
rmmod dwmac_qcom_ethqos
modprobe dwmac_qcom_ethqos
```

WiFi MAC Not Applied:

```
# Check modprobe configuration
cat /etc/modprobe.d/ath10k_mac.conf

# Check actual MAC
iw dev wlan1 info | grep addr

# Reload driver if needed
rmmod ath10k_snoc
modprobe ath10k_snoc
```

7.8 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- Robonode Hardware - Board hardware periphery briefing
- Robonode Setup - Initial board praparation

- Robonode GPIO - Board GPIO interfaces and control
- Robonode BLSP - Board UART and I2C interfaces
- TLV Storage Design - EEPROM TLV format details

8. TobuFi-DVK Board

8.1 Overview

TobuFi-DVK is a development kit for the TobuFi System-on-Module featuring Qualcomm QCS405 SoC. This document describes board-level hardware integration and peripherals. Supported board revisions: rev3, rev4, rev5.

8.2 Peripherals

- WiFi Radio #0 (HE/11ax)
- WiFi Radio #1 (VHT/11ac)
- Ethernet port (100M/1000M)
- USB Port #1 (USB 2.0 gadget)
- USB Port #2 (USB 3.0 DRD/OTG)
- I2C EEPROM (16KB)
- 4x BLSP UART/I2C
- SD card slot
- Coax audio in/out
- Optical audio out
- HDMI port
- MIPI display
- Power LED (fixed)
- Reset button (fixed)
- Boot configuration switches (DIP, rev3/rev4) / EDL button (rev5)

8.3 Board Revisions

Feature	Rev3	Rev4	Rev5
HDMI connection	Via EP92A6 bridge	Direct SoC	Direct SoC
BLSP0 UART (block A0)	Not exported	Available	Available
Power-on behavior	Button press	Button press	Auto-start
EDL mode entry	DIP switch #1	DIP switch #1	Dedicated push button

8.4 Board Layout

8.4.1 Top View (Rev3/Rev4)

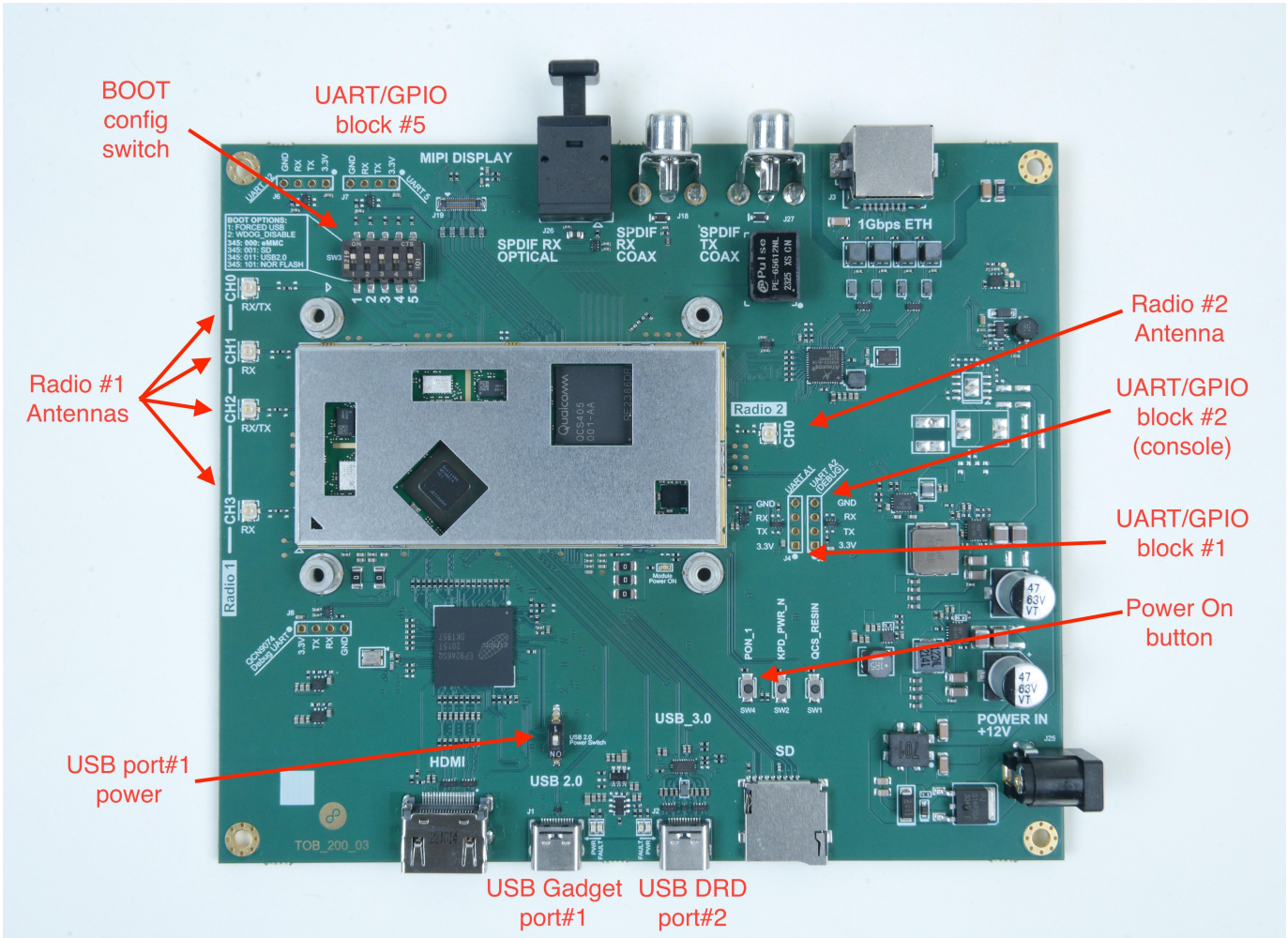


Figure 1: TobuFi-DVK rev3/rev4 board top view components

8.4.2 Bottom View (Rev3/Rev4)

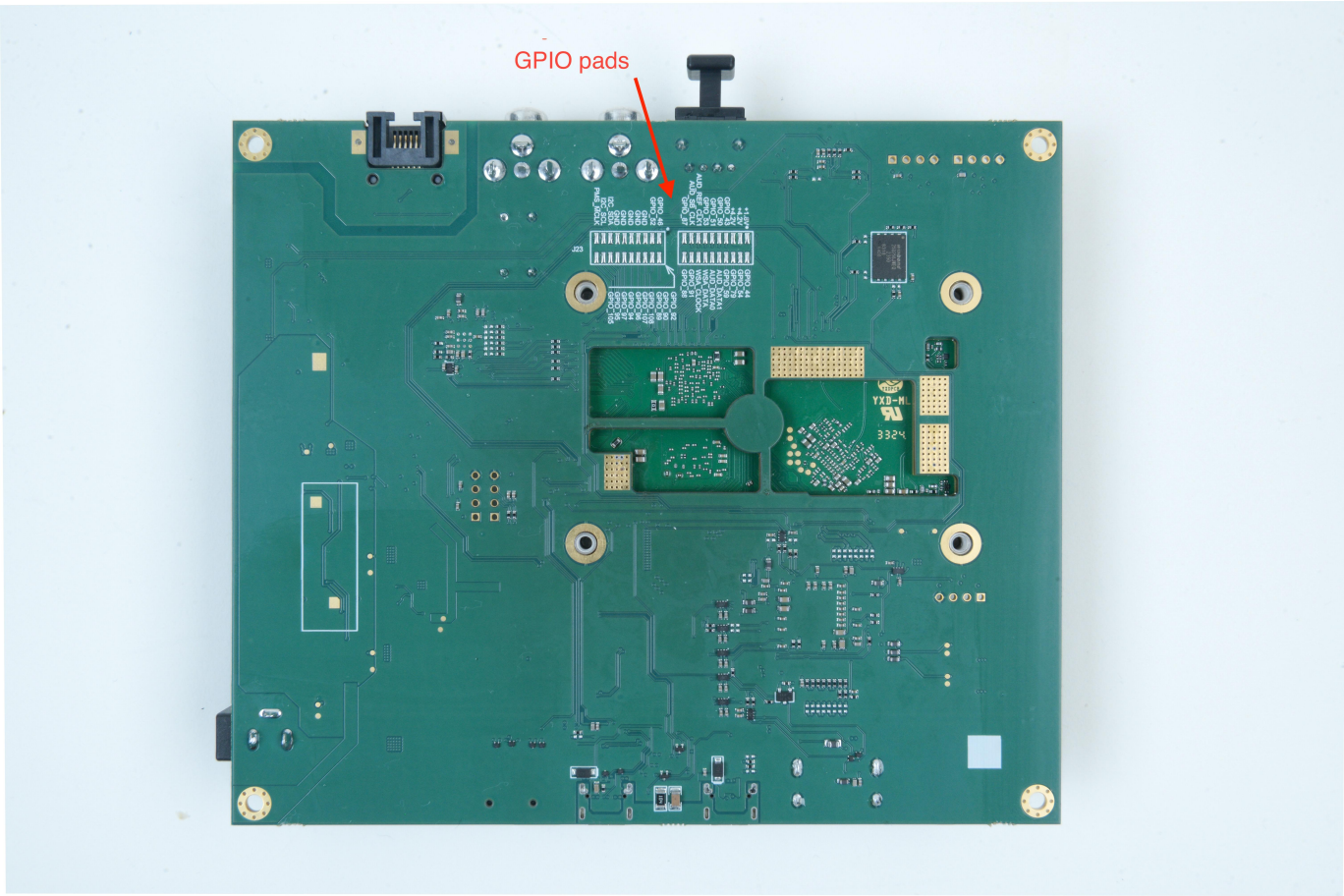


Figure 2: TobuFi-DVK rev3/rev4 board bottom view components

8.4.3 Top View (Rev5)

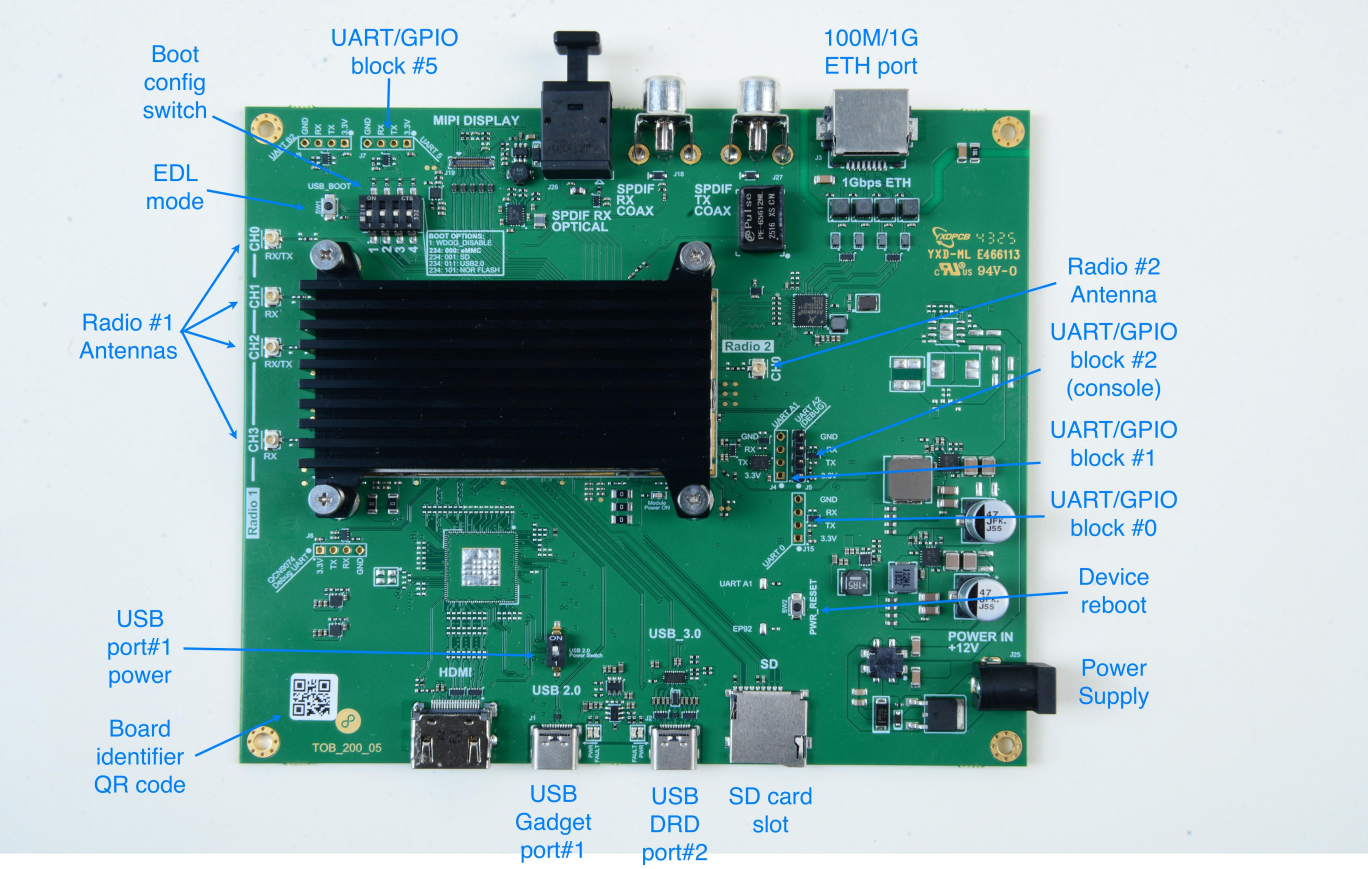


Figure 3: TobuFi-DVK rev5 board top view components

8.4.4 Bottom View (Rev5)

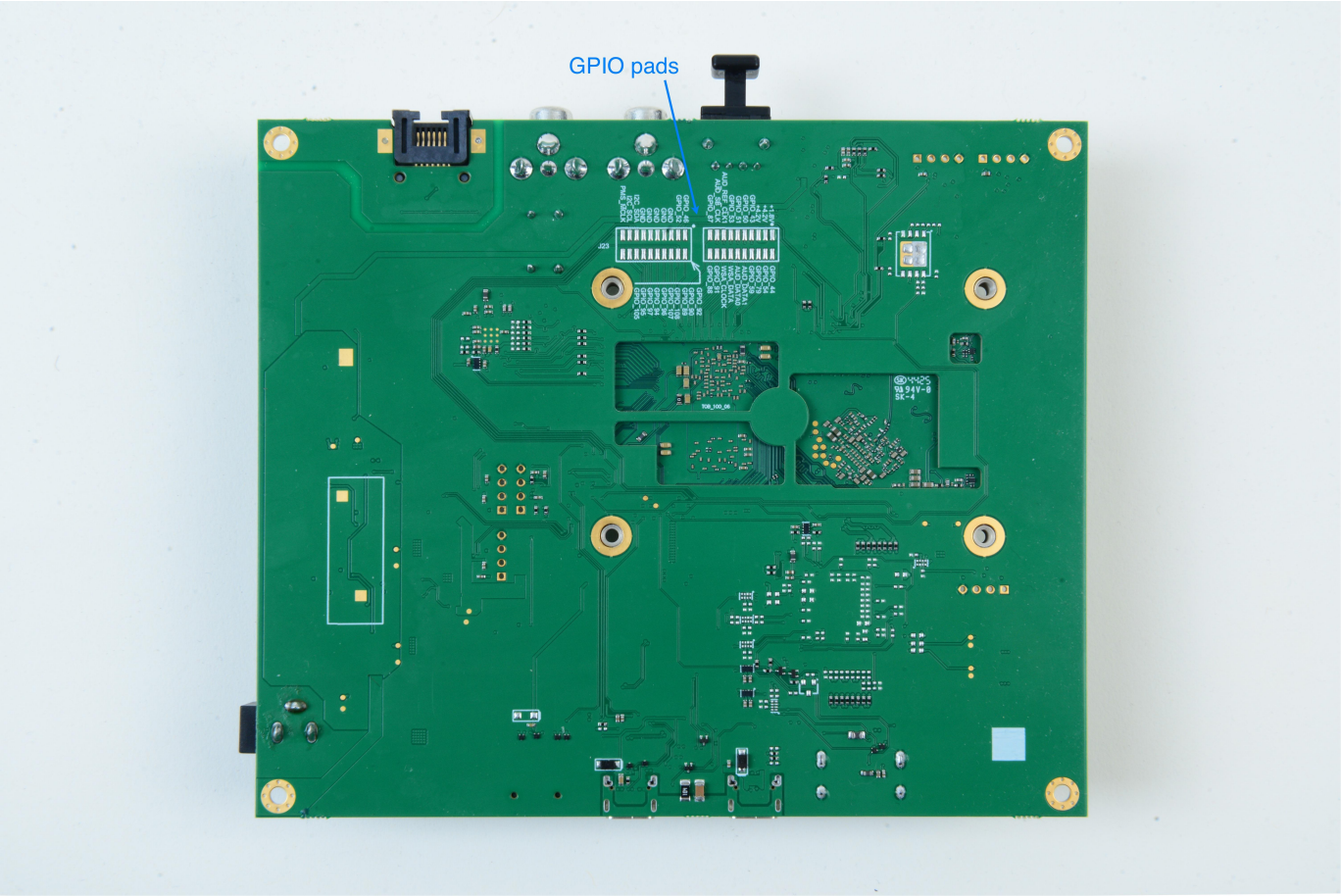


Figure 4: TobuFi-DVK rev5 board bottom view components

8.4.5 Connector Identification

Top Side:

- **Ethernet Port**: RJ45 connector
- **USB Port #1**: USB-C connector (USB 2.0 gadget)
- **USB Port #2**: USB-C connector (USB 3.0 DRD/OTG)
- **WiFi Antennas**: 4x U.FL (Radio #0), 1x U.FL (Radio #1)
- **Power Connector**: DC barrel jack (12V)
- **HDMI Port**: HDMI connector
- **SD Card Slot**: MicroSD connector
- **Audio**: Coax in/out, optical out

Bottom Side:

- **BLSP0 Header**: Pads for UART (GPIOs 30-31, rev4+ only)
- **BLSP1 Header**: Pads for UART/I2C (GPIOs 22-25)
- **BLSP2 Header**: Serial console UART (GPIOs 17-18)
- **BLSP5 Header**: Pads for UART (GPIOs 26-27)
- **Power LED**: Fixed power indicator
- **Reset Button**: System reset
- **Boot Config**: DIP switches (rev3/rev4) or EDL button (rev5)

8.5 Board Interfaces

8.5.1 I2C EEPROM

Chip: AT24C128C **Capacity:** 16KB **Interface:** BLSP1 I2C (GPIO 24-25)

Non-volatile storage for board identification and production storage. Accessible through standard Linux I2C interface.

8.5.2 WiFi Radio #0 (Pine)

Parameter	Description
Radio Chip	PCIe/QCN9074
WiFi Standard	IEEE 802.11ax (WiFi 6/6E)
Channel Width	20/40/80/160 MHz
Operation Band	2GHz + 5GHz or 2GHz + 6GHz
Operation Mode	2x4 (2T4R)
Antenna Connectors	4x U.FL
Max Conducted 2GHz Tx Power (aggregate)	32 dBm
Max Conducted 5GHz Tx power (aggregate)	29 dBm
Max Conducted 6GHz Tx Power (aggregate)	29 dBm
2GHz Frequency Range	2360–3150 MHz
5GHz Frequency Range	4550–6630 MHz
6GHz Frequency Range	5325–7495 MHz

High-performance WiFi radio supporting WiFi 6E (802.11ax) with MIMO capability. Suitable for high-throughput wireless applications, mesh networking, and dual-band simultaneous operation.

8.5.3 WiFi Radio #1

Parameter	Description
Radio Chip	SNOC/WCN3980
WiFi Standard	IEEE 802.11ac (WiFi 5)
Channel Width	20/40/80 MHz
Operation Band	2GHz + 5GHz
Operation Mode	1x1 (1T1R)
Antenna Connectors	1x U.FL
Max 2GHz Tx Power	16 dBm
Max 5GHz Tx power	16 dBm
2.4 GHz Frequency Range	2412-2484 MHz
5 GHz Frequency Range	5180-5825 MHz

Integrated WiFi radio supporting dual-band operation. Suitable for management interface, station mode connectivity, or additional access point deployment.

8.5.4 Ethernet Port

Parameter	Description
Transceiver Chip	RGMII/AR8033A
Speed	10/100/1000 Mbps
Duplex	Half/Full
MDI/MDI-X	Auto-crossover

Gigabit Ethernet port with RJ45 connector. Supports network connectivity for wired applications, device management, and high-bandwidth data transfer.

8.5.5 USB Port #1

Parameter	Description
Standard	USB 2.0
Speed	480 Mbps (HS)
Connector	USB-C
Mode	Gadget only
Identifier	usb0

Gadget port for host PC access. ADB debugging interface is available through this port.

8.5.6 USB Port #2

Parameter	Description
Standard	USB 3.0
Speed	5 Gbps (SS) / 480 Mbps (HS)
Connector	USB-C
Mode	DRD (Host/Gadget/OTG)
Switching	Automatic electrical
Identifier	usb1

USB dual-role (DRD) on-the-go (OTG) port with automatic runtime switching between host and device modes. In host mode, supports connecting USB flash drives, USB cameras, and other standard USB peripherals like mice and keyboards. In device mode, functions as a USB gadget port that can be configured with various functions in software.

8.5.7 Boot Configuration (Rev3/Rev4)

Boot configuration DIP switches allow switching the boot devices order. The following combinations are supported:

Mode	SW #1	SW #2	SW #3	SW #4	SW #5
EDL recovery	ON	-	-	-	-
eMMC → SD → USB2	OFF	-	OFF	OFF	OFF
SD → eMMC → EDL	OFF	-	ON	OFF	OFF
eMMC → EDL	OFF	-	OFF	ON	OFF
USB2	OFF	-	ON	ON	OFF
NAND → EDL	OFF	-	OFF	OFF	ON
NOR → EDL	OFF	-	ON	OFF	ON

Dashes (-) indicate that switch position is not relevant.

8.5.8 Boot Configuration (Rev5)

Rev5 uses a dedicated EDL mode push button instead of DIP switches. The EDL entry procedure is the same as Robonode: hold the EDL button pressed when power cycling or rebooting the device.

8.5.9 Power-On and Reset

Rev3/Rev4: Device requires pressing the "Power ON" button after applying power. The PON button is used to start the device on every power cycle. Button press is not required when performing a normal Linux reboot.

Rev5: Device starts automatically when power is applied. The PON button is repurposed as a hard device reset.

8.6 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- TobuFi SoM - TobuFi SoM (System-on-Module) details
- TobuFi-DVK GPIO - Board GPIO interfaces and control
- TobuFi-DVK BLSP - Board UART and I2C interfaces
- TobuFi-DVK Setup - Initial board setup and flashing
- TobuFi-DVK Initialisation - Board initialization sequence

9. TobuFi-DVK Setup

This guide covers initial device setup and hands-on launch procedures for the TobuFi-DVK board.

For board hardware specifications and connector locations, refer to TobuFi-DVK Hardware.

9.1 Required Equipment

- Compatible DC power supply (12V, 1.5A minimum)
- For console access: Serial console cable (3.3V TTL UART)
- For console access: Serial terminal software (minicom, screen, PuTTY, etc.)
- For device recovery: USB Type-C cable
- For device recovery: Linux PC with qdl tool installed or Windows PC with QFIL

9.2 Power Requirements

The TobuFi-DVK board requires a **12V DC** power supply capable of supplying at least **1.5A**.

Power On (Rev3/Rev4):

- Connect power supply to the barrel jack
- Press "Power ON" button to start the device

This procedure is needed every time the device is power cycled. However it is not required when performing a normal Linux reboot.

Power On (Rev5):

- Connect power supply to the barrel jack — device starts automatically

Refer to TobuFi-DVK Hardware for power connector location on the PCB.

9.3 Serial Console Access

To access the device via serial console:

- Connect 3.3V TTL serial console cable to the serial console connector
- Configure serial terminal with appropriate settings
- Follow System Access Guide for connection details

Refer to TobuFi-DVK Hardware for serial console connector location on the PCB.

9.4 USB Recovery

9.4.1 Emergency Download (EDL) Mode

EDL (Emergency Download) mode is a low-level boot mode in the Qualcomm SoC that enables device recovery and initial programming by exposing a USB interface for flash programming.

Use cases:

- Initial factory programming of blank devices
- Recovery from bootloader corruption
- Full device reflashing
- Partition table restoration

Device appears as USB `05c6:9008` (Qualcomm HS-USB QDLoader 9008) in EDL mode.

Important: EDL recovery installs legacy software version v0.x which can subsequently be upgraded to open source software version v1.x using USB flashing (see USB Flashing section).

Warning: EDL recovery completely erases and reprograms device flash. Use correct recovery images for TobuFi-DVK hardware to avoid permanent damage.

9.4.2 Entering EDL Mode (Rev3/Rev4)

- Connect USB gadget port#1 to PC
- Set boot config DIP switch #1 to ON
- Power cycle the device
- Press and release "Power ON" button
- Device enters EDL mode and appears as `05c6:9008`

Refer to TobuFi-DVK Hardware for button and switch locations on the PCB.

9.4.3 Exiting EDL Mode (Rev3/Rev4)

- Set DIP switch #1 to OFF
- Power cycle the device
- Press and release "Power ON" button to start normal device boot

9.4.4 Entering EDL Mode (Rev5)

- Connect USB gadget port#1 to PC

- Press and hold "EDL mode" button
- Press and release "Device reboot" button
- Release "EDL mode" button
- Device enters EDL mode and appears as `05c6:9008`

Refer to TobuFi-DVK Hardware for button locations on the PCB.

9.4.5 Obtaining Recovery Package

Recovery flash image packages are available from 8devices support. Contact 8devices support to obtain the appropriate recovery package for your device.

9.4.6 EDL Recovery Tools

EDL recovery can be performed using different tools depending on your operating system:

Linux:

- Recommended: `edl_flash.py` script (included in recovery package, Linux only)
- Alternative: QDL tool from https://github.com/linux-msm/qdl

Windows:

- QFIL (Qualcomm Flash Image Loader) from Qualcomm Product Support Tools (QPST)

9.4.7 EDL Recovery Using edl_flash.py (Linux only)

The `edl_flash.py` script provides automated EDL recovery on Linux systems.

Recovery Steps:

- Connect USB gadget port#1 to PC
- Enter EDL recovery mode (see Entering EDL Mode or Rev5)
- Execute EDL flashing script from software flash image package:

```
python3 edl_flash.py
```

- Power cycle the device to boot new firmware

Multiple EDL Devices:

When multiple devices are in EDL state, specify device by serial number:

```
# Flash specific device by serial
python3 edl_flash.py <serial>

# List available EDL devices
python3 edl_flash.py -l
```

9.4.8 EDL Recovery Using QDL (Linux)

Alternative method using QDL tool:

```
# Verify device enumeration
lsusb | grep 05c6:9008

# Flash recovery images
qdl --storage ufs prog_firehose.mbn rawprogram.xml patch.xml
```

Required images: prog_firehose.mbn (programmer), rawprogram.xml (partition layout), patch.xml (patching instructions), partition images (bootloader, kernel, rootfs).

9.4.9 EDL Recovery Using QFIL (Windows)

Use QFIL from Qualcomm Product Support Tools package. Refer to QFIL documentation for flashing procedures.

9.5 USB Flashing

Fastboot provides partition-level flashing capabilities through the bootloader interface. This method is used to upgrade devices from legacy software v0.x to open source software v1.x.

9.5.1 Prerequisites

- ADB (Android Debug Bridge) and Fastboot tools
- Windows: USB driver for Android devices

9.5.2 Fastboot Flashing Using fastboot_flash.py

The `fastboot_flash.py` script automates fastboot flashing operations. The script automatically detects device state and enters fastboot mode if needed.

Flashing Procedure:

```
# Flash boot and system partitions
python3 fastboot_flash.py --boot boot-image.img --system rootfs-image.img
```

Multiple Devices:

When multiple devices are connected, specify device by serial number:

```
# Specify device by serial number
python3 fastboot_flash.py <serial> --boot boot.img --system rootfs.img

# Check available devices
fastboot devices
```

The script will:

- Automatically detect if device is in ADB mode and reboot to fastboot
- Wait for device to appear in fastboot mode (up to 120 seconds)
- Flash specified partitions
- Automatically reboot device after successful flashing

9.6 Related Documentation

- TobuFi-DVK Hardware - Board hardware periphery briefing
- TobuFi-DVK BLSP - Board UART and I2C interfaces
- TobuFi-DVK Initialisation - Board hardware initialisation briefing

10. TobuFi-DVK BLSP Configuration

10.1 BLSP Endpoints

TobuFi-DVK board utilizes QCS405 BLSP (BAM Low-Speed Peripheral) controllers for UART and I2C connectivity.

BLSP #	GPIOs	Function	Endpoints
0	30-31	UART	Pads (rev4+ only)
0	32-33	-	TP5, TP6 (test points)
1 (A)	22-23	UART	Pads
1 (A)	24-25	I2C	AT24, PI5USB30213
1 (B)	47-50	GPIO	-
2 (A)	17-18	UART	Serial console
2 (A)	19-20	-	Non-installed header
2 (B)	39-40	GPIO	Pads
2 (B)	41-42	I2C	LCD
3	82-85	UART	WCN3980
4	37-38, 117-118	-	Not exported
5	26-27	UART	Pads
5	28-29	-	Not exported

Note: BLSP0 (GPIOs 30-31) is not exported on rev3. Available as UART from rev4 onwards.

10.2 UART

10.2.1 Hardware Characteristics

Ports Specification:

Parameter	Value	Notes
Maximum Baud Rate	4 Mbps	Limited by clock divider
FIFO Depth	64 bytes	TX and RX
Data Bits	5-8	Configurable
Stop Bits	1-2	Configurable
Parity	None/Even/Odd	Configurable
Flow Control	RTS/CTS hardware	4-wire mode only

Clock Configuration:

```
// UART clock calculation
// freq = source_clock / (divider * 16)
// For 115200 baud:
// 115200 = 7372800 / (4 * 16)

#define UART_CLK_SOURCE   7372800  // 7.3728 MHz
#define BAUD_115200_DIV   4        // Divider for 115200
#define BAUD_921600_DIV   0.5      // Not integer, use closest
```

10.2.2 Hardware Mapping

UART interfaces are mapped to Linux TTY devices.

TTY Device	BLSP #	GPIOs	Endpoint	DTS Node
/dev/ttyMSM0	2 (A)	17-18	Serial console	blsp1_uart2
/dev/ttyMSM1	0	30-31	Pads (rev4+ only)	blsp1_uart0
/dev/ttyMSM2	1 (A)	22-23	Pads	blsp1_uart1
/dev/ttyMSM3	5	26-27	Pads	blsp2_uart0
/dev/ttyMSM4	3	82-85	WCN3980 (internal)	blsp1_uart3

Note: ttyMSM0 is the primary debug serial console used by kernel for logging. On rev3, BLSP0 is not exported and ttyMSM1 is not available.

10.2.3 DTS Configuration

UART ports are configured in TobuFi-DVK DTS files. The base configuration is in `qcs405-tobufi-dvk.dts` with revision-specific overrides:

- Rev3: `qcs405-tobufi-dvk-rev3.dts`
- Rev4: `qcs405-tobufi-dvk-rev4.dts`
- Rev5: `qcs405-tobufi-dvk-rev5.dts`

To enable or disable a port:

```
&blsp1_uart0 {
        status = "okay";     /* enable */
};

&blsp1_uart1 {
        status = "disabled"; /* disable */
};
```

10.2.4 2-Wire UART Configuration

By default, UART ports use 4-wire configuration (TX, RX, CTS, RFR). To configure 2-wire UART (TX, RX only), update the pinmux configuration in board DTS:

```
&tlmm {
        blsp1_uart0_2wire: blsp1-uart0-2wire-state {
                pins = "gpio30", "gpio31";  /* TX, RX only */
                function = "blsp_uart0";
        };
};

&blsp1_uart0 {
        pinctrl-0 = <&blsp1_uart0_2wire>;
        status = "okay";
};
```

10.2.5 Serial Port Ordering

The Linux TTY device numbering (`/dev/ttyMSMx`) is determined by device tree aliases defined in the board DTS:

```
aliases {
        serial0 = &blsp1_uart2;
        serial1 = &blsp1_uart0;
        serial2 = &blsp1_uart1;
```

```
        serial3 = &blsp2_uart0;
};
```

The alias name suffix (serial0, serial1, etc.) directly maps to the TTY device number (ttyMSM0, ttyMSM1, etc.). UART ports without aliases receive device numbers dynamically during kernel initialization.

10.2.6 Usage Example

To enable serial console on ttyMSM1 (BLSP0, GPIO 30-31), connect serial TTL cable to BLSP0 pads and execute:

```
/sbin/agetty -n -a root --noclear ttyMSM1
```

Configure UART Parameters via stty:

```
# Set baud rate to 115200, 8N1
stty -F /dev/ttyMSM1 115200 cs8 -cstopb -parenb

# Enable hardware flow control (RTS/CTS)
stty -F /dev/ttyMSM1 crtscts

# Check current configuration
stty -F /dev/ttyMSM1 -a
```

10.2.7 Debugging

Check UART Status:

```
# View UART driver statistics
cat /proc/tty/driver/msm_serial

# Check for overrun/framing errors
dmesg | grep -i "msm_serial\|uart"
```

Common UART Issues:

Symptom	Cause	Solution
Garbled output	Baud rate mismatch	Verify both sides use same baud
No output	Wrong TX/RX pins	Check pinmux configuration
Data loss	FIFO overflow	Enable flow control or reduce baud
Intermittent errors	Signal integrity	Check cable length, add termination

10.3 I2C

10.3.1 Hardware Characteristics

Supported operation modes:

Mode	Speed	Notes
Standard Mode	100 kHz	Default speed
Fast Mode	400 kHz	Common speed
Fast Mode Plus	1 MHz	Maximum supported

10.3.2 Hardware Mapping

I2C interfaces available on TobuFi-DVK board. Only configured buses appear as Linux I2C devices.

I2C Device	BLSP #	GPIOs	Endpoint	DTS Node
/dev/i2c-0	1 (A)	24-25	AT24C128C EEPROM, PI5USB30213	blsp1_i2c1
/dev/i2c-1	2 (B)	41-42	LCD	blsp1_i2c2

10.3.3 DTS Configuration

I2C buses are configured in TobuFi-DVK DTS files. To enable or disable a bus:

```
&blsp1_i2c1 {
        status = "okay";      /* enable */
};

&blsp1_i2c2 {
```

```
        status = "disabled"; /* disable */
};
```

10.3.4 Usage Examples

Detect I2C devices on bus 0:

```
i2cdetect -y 0

i2cget -y 0 0x56 0x00
```

10.3.5 Debugging

Common I2C Issues:

Symptom	Cause	Solution
No ACK	Device not present	Check address, power, connections
Bus lockup	SDA/SCL stuck low	Power cycle, check pull-ups
Data corruption	Clock stretching	Reduce bus speed
Timeout	Slow device	Increase timeout in driver

10.4 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- TobuFi-DVK Hardware - Board hardware periphery briefing
- TobuFi-DVK GPIO - Board GPIO interfaces and control
- TobuFi-DVK Initialisation - Board hardware initialisation briefing

11. TobuFi-DVK GPIO Configuration

11.1 Overview

TobuFi-DVK board provides general-purpose I/O interfaces for onboard devices including USB power management and header connectors. GPIOs are accessed through the QCS405 TLMM (Top Level Mode Multiplexer) controller.

11.1.1 Allocation Summary

GPIO	Function	Direction	Default State	Notes
15	USB2_VBUS	Output	Low	USB Port #1 power control
16	USB3_VBUS	Output	Low	USB Port #2 power control
17	UART2_TX	Output	High	Serial console
18	UART2_RX	Input	-	Serial console
22-23	UART1	-	-	Pads
24-25	I2C1	Bidir	High	EEPROM, PI5USB30213
26-27	UART5	-	-	Pads
30-31	UART0	-	-	Pads (rev4+ only)
32-33	-	-	-	TP5, TP6 (test points)
39-40	GPIO	-	-	Pads
41-42	I2C	Bidir	High	LCD
47-50	GPIO	-	-	-
82-85	UART3	-	-	WCN3980 (internal)

Note: BLSP0 UART (GPIOs 30-31) is not exported on rev3; available from rev4 onwards. GPIOs 19-20 are on a non-installed header. GPIOs 28-29 are not exported.

11.2 USB VBUS Control

GPIO-controlled USB port power management.

USB Port	GPIO	Control
USB Port #1	15	VBUS boost power control
USB Port #2	16	VBUS boost power control

DTS Configuration:

```
&tlmm {
        usb2_vbus_boost: usb2-vbus-boost {
                pins = "gpio15";
                function = "gpio";
        };

        usb3_vbus_boost: usb3-vbus-boost {
                pins = "gpio16";
                function = "gpio";
        };
};
```

Note: USB VBUS control is typically managed by USB subsystem. Manual control via sysfs GPIO interface should be used with caution.

11.3 Headers

GPIO pins available on board header connectors.

Header	GPIOs	Active Function
BLSP0	30-31	UART (rev4+ only)
BLSP1	22-25	UART + I2C
BLSP2	17-18	UART (serial console)
BLSP5	26-27	UART

For detailed UART and I2C configuration, see BLSP Configuration.

DTS Configuration:

To configure header GPIOs as general-purpose I/O, define pinmux configuration in board DTS:

```
&tlmm {
        gpio_header: gpio-header-state {
                pins = "gpio30", "gpio31";
                function = "gpio";
        };
};
```

Usage Example:

```
# Export GPIO 30
echo 30 > /sys/class/gpio/export

# Configure as output
echo out > /sys/class/gpio/gpio30/direction

# Set high
echo 1 > /sys/class/gpio/gpio30/value

# Set low
echo 0 > /sys/class/gpio/gpio30/value

# Unexport when done
echo 30 > /sys/class/gpio/unexport
```

11.4 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- TobuFi-DVK Hardware - Board hardware periphery briefing
- TobuFi-DVK BLSP - Board UART and I2C interfaces
- TobuFi-DVK Initialisation - Board hardware initialisation briefing
- Linux GPIO Sysfs Interface

12. TobuFi-DVK Initialisation

This document describes hardware component initialization essentials required for TobuFi-DVK device boot. It focuses on production data retrieval, hardware resource provisioning, and driver initialization.

12.1 Production EEPROM

The device EEPROM is provisioned with hardware-specific identity and configuration data. This data is read during boot initialization to configure network interfaces and radio calibration.

12.1.1 EEPROM Device

Character device: /sys/bus/i2c/devices/0-0056/eeprom

```
# Detect I2C devices on bus 0
i2cdetect -y 0
#     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
# 00:          -- -- -- -- -- -- -- -- -- -- -- -- --
# 10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 50: -- -- -- -- -- -- 56 -- -- -- -- -- -- -- -- --
# 60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
# 70: -- -- -- -- -- -- -- --

# Read single byte from EEPROM address 0x00
i2cget -y 0 0x56 0x00
```

12.1.2 EEPROM Data Structure

- Storage size: 16384 bytes (16 KB)
- Data offset: 512 bytes
- Enabled data models: TobuFi sys-eeprom TLV, RoboSoft TLV

```
┌──────────────────────────────────────────────┐
│         EEPROM Memory Layout (16KB)          │
├──────────────────────────────────────────────┤
│ 0x0000 - 0x01FF (512 bytes)                  │
│   Reserved for bootloader/factory use        │
├──────────────────────────────────────────────┤
│ 0x0200 - 0x3FFF (15872 bytes)                │
│   TLV storage area (production data)         │
│   - Product identification                   │
│   - PCB information                          │
```

```
|   - MAC addresses                          |
|   - Radio calibration data                 |
└────────────────────────────────────────────┘
```

12.1.3 EEPROM Fields

Production EEPROM contains board metadata programmed during manufacturing:

Field	Description
`PRODUCT_ID`	Product identifier (TobuFi-DVK)
`PCB_REVISION`	PCB hardware revision
`PCB_NAME`	PCB design name (TOB_200)
`PCB_SN`	PCB serial number
`PCB_PROD_DATE`	Production date
`PCB_PROD_LOCATION`	Production facility
`SERIAL_NO`	Device serial number
`MAC_ADDR_eth0`	Ethernet MAC address
`MAC_ADDR_wlan1`	WiFi Radio #1 MAC address
`RADIO_CALIBRATION_DATA`	WiFi Radio #1 calibration (binary)

12.1.4 EEPROM Data Retrieval

Service: `eeprom-dump.service` (sysinit.target phase) **Utility**: `tlvs` **Output**: `/etc/board.conf`

The `eeprom-dump` service uses `tlvs` utility to read hardware EEPROM and generate `/etc/board.conf` with shell variables.

Example /etc/board.conf:

```
PRODUCT_ID=TobuFi-DVK
PCB_REVISION=01
PCB_NAME=TOB_200
PCB_SN=1234567890
PCB_PROD_DATE=2025-01-01
PCB_PROD_LOCATION=Factory
SERIAL_NO=0987654321
MAC_ADDR_eth0=00:03:7F:12:90:8F
MAC_ADDR_wlan1=00:03:7F:12:90:91
```

12.2 Ethernet Initialization

12.2.1 Hardware

Driver: `stmmac` **Interface**: `eth0`

12.2.2 MAC Address Provisioning

`eeprom-dump.service` provisions Ethernet MAC address through kernel module parameter:

File: `/etc/modprobe.d/stmmac_mac.conf`

```
options dwmac_qcom_ethqos mac_addr=00:03:7F:12:90:8F
```

12.2.3 Initialization Sequence

- `eeprom-dump.service` reads `MAC_ADDR_eth0` from EEPROM
- Creates `/etc/modprobe.d/stmmac_mac.conf`
- `systemd-modules-load.service` loads `stmmac` driver
- Driver initializes with configured MAC
- `eth0` interface available

12.3 WiFi Radio #0 (PCIe Radio)

12.3.1 Hardware

Driver: `ath11k_pci` **Bands**: 2.4 GHz / 5 GHz **Interface**: e.g., `wlan0`

12.3.2 Firmware Requirements

Location: `/lib/firmware/ath11k/<hw_version>/`

- `board-2.bin` - Board Data File (BDF)
- `firmware-*.bin` - Firmware image

12.3.3 Board Data File Variants

BDF variant mechanism supports band-specific board data:

Parameter: `bdf_variant` **Values**: `2` (2.4 GHz), `5` (5 GHz)

Configuration:

```
options ath11k bdf_variant=5
```

Driver searches for `board-2_5.bin` or falls back to `board-2.bin`.

Band switching requires driver reload with different `bdf_variant`. The `radioconf` utility manages band configuration.

12.3.4 Initialization Sequence

- `systemd-modules-load.service` loads `ath11k_pci`
- Driver probes PCIe device
- Firmware and BDF loaded
- Radio initialized
- Wireless interface created

Initialization Timing:

Phase	Duration	Details
PCIe enumeration	~200ms	Bus scan, device detection
Firmware download	~1.5s	Load amss.bin, m3.bin
BDF loading	~300ms	Board data file
Radio calibration	~500ms	Internal calibration
Interface creation	~200ms	Network stack registration
Total	**~2.7s**	From module load to interface ready

Driver Probe Sequence:

```
// Simplified ath11k_pci driver probe flow

int ath11k_pci_probe(struct pci_dev *pdev) {
    // 1. Enable PCIe device
    pci_enable_device(pdev);
    pci_set_master(pdev);

    // 2. Map BAR regions
    pci_request_regions(pdev, "ath11k_pci");
```

```
    // 3. Enable MSI/MSI-X interrupts
    pci_enable_msi_range(pdev, 32, 32);

    // 4. Initialize QMI (Qualcomm Messaging Interface)
    ath11k_qmi_init();

    // 5. Load firmware
    request_firmware("amss.bin");
    request_firmware("m3.bin");

    // 6. Download firmware to radio
    ath11k_qmi_firmware_download();

    // 7. Load BDF
    ath11k_core_fetch_bdf();
    ath11k_qmi_bdf_download();

    // 8. Radio power-on and calibration
    ath11k_core_start();

    // 9. Register with mac80211
    ieee80211_register_hw();

    return 0;
}
```

12.3.5 MAC Address

WiFi Radio #0 MAC address and calibration data are pre-programmed during production into the radio's internal EEPROM storage.

12.4 WiFi Radio #1 (Integrated Radio)

12.4.1 Hardware

Radio: WCN3990 integrated WiFi/BT **Driver**: `ath10k_snoc` **Bands**: 2.4 GHz / 5 GHz **Interface**: `wlan1`

12.4.2 Firmware and Calibration

Firmware: `/lib/firmware/ath10k/WCN3990/hw1.0/` **Calibration**: `/lib/firmware/ath10k/cal-snoc-a000000.wifi.bin`

Radio calibration data is extracted from EEPROM `RADIO_CALIBRATION_DATA` field by `eeprom-dump.service` and written to calibration file.

12.4.3 MAC Address Provisioning

`eeprom-dump.service` provisions MAC address through kernel module parameter:

File: `/etc/modprobe.d/ath10k_mac.conf`

```
options ath10k_snoc mac_addr=00:03:7F:12:90:91
```

12.4.4 Initialization Sequence

- `eeprom-dump.service` extracts:
- `MAC_ADDR_wlan1` → `/etc/modprobe.d/ath10k_mac.conf`
- `RADIO_CALIBRATION_DATA` → `/lib/firmware/ath10k/cal-snoc-a000000.wifi.bin`
- `systemd-modules-load.service` loads `ath10k_snoc`
- Driver initializes with MAC and calibration
- `wlan1` interface available

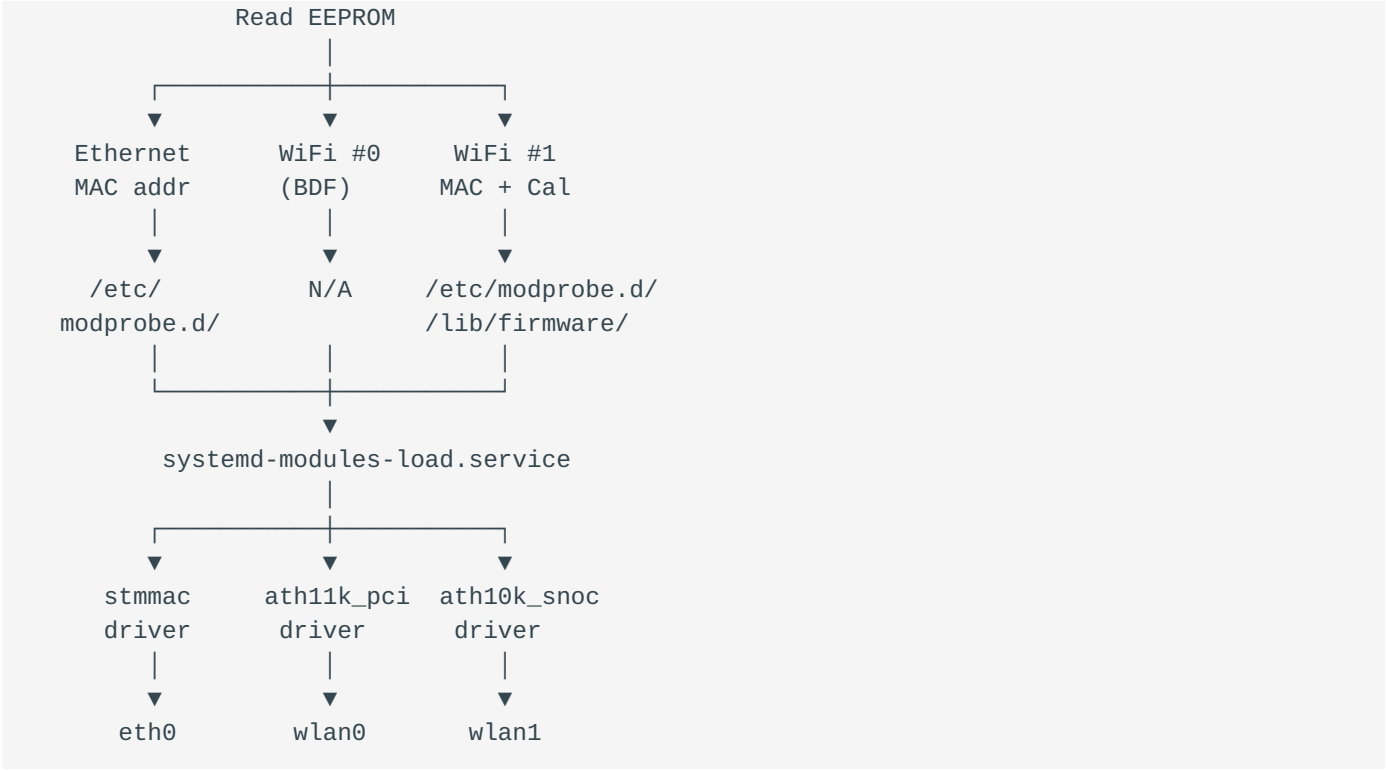
Initialization Timing:

Phase	Duration	Details
Platform device probe	~100ms	SNOC bus enumeration
MAC address setup	~50ms	Read from modprobe.conf
Calibration load	~200ms	Load cal-*.bin file
Firmware download	~800ms	Load firmware image
Radio initialization	~300ms	Hardware setup
Interface creation	~150ms	Network stack registration
Total	**~1.6s**	From module load to interface ready

12.5 Boot Sequence

Hardware initialization during **sysinit.target**:

```
          eeprom-dump.service
                  |
                  ▼
```




```
EEPROM data
├── Provisions /etc/modprobe.d/stmmac_mac.conf
├── Provisions /etc/modprobe.d/ath10k_mac.conf
└── Extracts /lib/firmware/ath10k/cal-snoc-a000000.wifi.bin
```

After sysinit.target, all hardware interfaces available for ESConf configuration.

12.6 System Information Retrieval

12.6.1 Device Serial Number

The device serial number is embedded in the kernel command line during boot:

```
# Get serial number from kernel command line
grep -o 'serial_num=[^ ]*' /proc/cmdline | cut -d'=' -f2
```

12.6.2 Active Boot Slot

The currently active A/B partition slot is determined during boot:

```
# Get active slot (_a or _b)
grep -o 'SLOT_SUFFIX=_[ab]' /proc/cmdline | cut -d'=' -f2
```

This information indicates which partition set (A or B) the system booted from, used for understanding update state and determining the target slot for software updates.

12.7 Troubleshooting

12.7.1 EEPROM Read Failures

Symptom:

```
eeprom-dump.service: Main process exited, code=exited, status=1/FAILURE
```

Diagnosis:

```
# Check I2C bus availability
ls -l /sys/bus/i2c/devices/

# Expected: 0-0056 directory exists
# If missing, I2C controller or EEPROM not detected

# Check I2C controller driver
lsmod | grep i2c
# Should show: i2c_qup (Qualcomm I2C driver)

# Manual I2C scan
i2cdetect -y 0
# Should show device at address 0x56

# Attempt to read EEPROM
i2cget -y 0 0x56 0x00
# Should return data
```

12.7.2 WiFi Radio Initialization Failures

PCIe Radio (QCN9074) Firmware Load Failure:

```
# Check PCIe enumeration
lspci -v | grep -A5 "Network controller"

# Check firmware files
ls -l /lib/firmware/ath11k/QCN9074/hw1.0/
# Required: amss.bin, m3.bin, board-2.bin

# Check driver logs
dmesg | grep ath11k_pci
# Look for: "firmware load failed" or "BDF download failed"
```

Integrated Radio (WCN3980) Calibration Failure:

```
# Check calibration file
ls -l /lib/firmware/ath10k/cal-snoc-a000000.wifi.bin
# Should be 128-256 bytes

# Validate calibration data
hexdump -C /lib/firmware/ath10k/cal-snoc-a000000.wifi.bin | head
# Should NOT be all 0xFF or 0x00

# Check driver logs
dmesg | grep ath10k_snoc
# Look for: "failed to fetch calibration data"
```

12.7.3 MAC Address Provisioning Issues

Ethernet MAC Not Applied:

```
# Check modprobe configuration
cat /etc/modprobe.d/stmmac_mac.conf
# Should contain: options dwmac_qcom_ethqos mac_addr=XX:XX:XX:XX:XX:XX

# Check actual MAC address
ip link show eth0
# link/ether should match EEPROM value

# If mismatch, reload driver:
rmmod dwmac_qcom_ethqos
modprobe dwmac_qcom_ethqos
```

WiFi MAC Not Applied:

```
# Check modprobe configuration
cat /etc/modprobe.d/ath10k_mac.conf

# Check actual MAC
iw dev wlan1 info | grep addr

# Reload driver if needed
rmmod ath10k_snoc
modprobe ath10k_snoc
```

12.8 Related Documentation

- QCS405 SoC - QCS405 SoC (System-on-Chip) details
- TobuFi SoM - TobuFi SoM (System-on-Module) details
- TobuFi-DVK Hardware - Board hardware periphery briefing
- TobuFi-DVK Setup - Initial board setup and flashing
- TobuFi-DVK BLSP - Board UART and I2C interfaces
- TLV Storage Design - EEPROM TLV format details