ROBOTICS-DOCS

NXP Robotics SDK Documentation

Complete documentation set

Rev. 1 — 10 November 2026

Document information

Information	Content
Document identifier	ROBOTICS-DOCS
Release	0.7.0
Revision	Rev. 1
Release date	10 November 2026
Keywords	Robotics SDK, MR-NavQ95, MR-VMU-Tropic, MR-MCXN-T1, MR-ESC-AM32, i.MX 95, ROS 2, PX4, Zephyr
Abstract	This document is a single-file compilation of the NXP Robotics SDK documentation site, covering all supported reference platforms (MR-NavQ95, MR-VMU-Tropic, MR-MCXN-T1, MR-ESC-AM32), their user manuals, application notes and system-level integration guides for the release listed on the cover.

Contents

1 Robotics SDK Introduction . . . 10
1.1 Supported platforms . . . 10
1.2 Explore the docs . . . 10
2 System Level Integration Guide . . . 12
2.1 How to read this document . . . 12
2.2 1. Overview . . . 13
2.3 2. The platforms at a glance . . . 13
2.4 3. Hardware view . . . 14
2.5 4. Software layer model . . . 15
2.6 5. The communication fabric . . . 16
2.7 6. Reference architecture (drone) . . . 17
2.8 7. End-to-end data flows . . . 19
2.9 Abbreviations . . . 21
2.10 References . . . 22
3 Release Notes . . . 23
3.1 About the Robotics SDK . . . 23
3.2 How to read this document . . . 23
3.3 Releases . . . 23
4 SDK Setup and Build . . . 31
4.1 Table of contents . . . 31
4.2 Supported platforms and stacks . . . 31
4.3 Development Workflows . . . 32
4.4 Requirements . . . 32
4.5 Prerequisites and configuration . . . 33
4.6 Flow A: getting pre-built images . . . 33
4.7 Flow B: building images from source . . . 34
4.8 Command reference guide . . . 36
4.9 Warnings and environment-specific notes . . . 38
4.10 Troubleshooting / FAQ . . . 40
5 MR-NavQ95 . . . 42
6 Overview . . . 43
6.1 Functional block diagram . . . 43
6.2 Main components . . . 43
7 Getting Started . . . 45
8 Deploy and Run Quick Start . . . 46
8.1 Get the artifacts first . . . 46
8.2 What to flash . . . 46
8.3 Connect . . . 47
8.4 Flash the Cortex-A Linux image (UUU) . . . 47
8.5 Flash the Cortex-M RTOS firmware (pyOCD) . . . 48
9 User Manual . . . 49
10 Introduction . . . 50
10.1 Radio Equipment Directive compliance information . . . 50
10.2 NXP Robotics SDK and upstream open-source software . . . 51

11 Finding platform resources . . . 52

12 Getting ready . . . 53

12.1 Kit contents . . . 53
12.2 Additional hardware . . . 53
12.3 Assumptions . . . 53
12.4 Minimum system requirements . . . 53
12.5 Software . . . 53

13 Getting to know the hardware . . . 54

13.1 Board overview . . . 54
13.2 Boot Switch (SW2) . . . 56
13.3 Connector index . . . 57
13.4 Main board . . . 60
13.5 IO expansion board . . . 65
13.6 Buttons and switches . . . 71
13.7 Camera expansion board . . . 72
13.8 T1 Ethernet switch board . . . 73
13.9 Notes . . . 74
13.10 Cheat sheet . . . 74
13.11 Related Guides . . . 75
13.12 References . . . 75

14 Installing and configuring software tools . . . 76

14.1 Getting the Cortex-A Linux image . . . 76
14.2 Building the Cortex-A Linux image from source . . . 76
14.3 Getting the Cortex-M RTOS firmware . . . 77
14.4 Device-tree overlays in U-Boot . . . 78

15 Configuring the hardware . . . 82

15.1 Flashing the Cortex-A Linux image . . . 82
15.2 Flashing the Cortex-M RTOS firmware . . . 83

16 Abbreviations . . . 86

17 References . . . 87

18 Application Notes . . . 88

Hardware Architecture and Reference Design . . . 89

1. Introduction . . . 89
2. System hardware architecture . . . 89
3. Compute core . . . 90
4. Connectivity . . . 91
5. Sensing and perception . . . 91
6. Robotics I/O . . . 91
7. Expansion and acceleration . . . 92
8. Power . . . 92
9. Hardware integration and bring-up . . . 92
10. Schematic, board layout, and bill of materials . . . 92
11. Abbreviations . . . 92
12. References . . . 93

19 Camera . . . 94

19.1 Introduction . . . 94
19.2 Camera subsystem architecture . . . 94
19.3 Supported sensors . . . 95

19.4 Setup . . . 95
19.5 Usage . . . 97
19.6 Multi-camera and dual CSI-2 . . . 98
19.7 Integration with the robotics stack . . . 98
19.8 References . . . 98

20 Ethernet . . . 100
20.1 Introduction . . . 100
20.2 1. Ethernet subsystem architecture . . . 100
20.3 2. Using Ethernet via the SJA1110 interfaces . . . 102
20.4 3. VLAN membership and A55/M7 interface partitioning . . . 103
20.5 4. Known issues and limitations . . . 105
20.6 References . . . 105

21 Cellular Modem . . . 106
21.1 1. Hardware architecture . . . 106
21.2 2. Example: Quectel RM520N-EU . . . 106

22 eIQ Neutron NPU . . . 109
22.1 Prerequisites . . . 109
22.2 Obtain the eIQ Neutron SDK . . . 109
22.3 Host-Side Setup . . . 110
22.4 Convert a Model for the Neutron NPU . . . 110
22.5 Deploy and Benchmark on the Target . . . 111
22.6 Keep the Runtime and Converter in Sync . . . 111
22.7 References and Downloads . . . 112

23 ARA-240 DNPU . . . 113
23.1 Prerequisites . . . 113
23.2 Hardware Setup . . . 113
23.3 Software setup . . . 113
23.4 Run CNNs (yolov8 or yolox) . . . 114
23.5 Run LLMs . . . 114
23.6 Available Models . . . 115
23.7 Further Documentation . . . 115

24 Remote ID Transmitter . . . 116
24.1 Usage . . . 116

25 Middleware . . . 120

26 ROS2 . . . 121

27 ROS2 Setup on NavQ95 . . . 122
27.1 Overview . . . 122
27.2 Prerequisites . . . 122
27.3 Quick Install . . . 122
27.4 Installation — ROS2 Base . . . 123
27.5 Installation — ROS2 Environment . . . 125
27.6 Installation — ROS2 Packages . . . 127
27.7 Known issues . . . 129
27.8 Version History . . . 129

28 ROS2 Multinode on NavQ95 . . . 130
28.1 Overview . . . 130
28.2 Topology . . . 130
28.3 Architecture . . . 130

28.4 Enabling and Validating MCXN-T1 → NavQ95 A55 Communication . . . 131
28.5 Enabling and Validating MCXN-T1 → NavQ95 M7 Communication . . . 134
28.6 Version History . . . 134
29 Flight Stack . . . 135
30 PX4 Documentation Reference . . . 136
31 MR-VMU-Tropic . . . 137
32 Overview . . . 138
32.1 Functional block diagram . . . 138
32.2 Main components . . . 138
33 Getting Started . . . 140
34 Deploy and Run Quick Start . . . 141
34.1 Get the artifacts first . . . 141
34.2 Artifacts . . . 141
34.3 Flash with LinkServer . . . 141
34.4 Flash the PX4 application (recommended) . . . 142
34.5 Erase all flash (recovery) . . . 142
34.6 Zephyr . . . 142
35 User Manual . . . 144
36 Introduction . . . 145
36.1 NXP Robotics SDK and upstream open-source software . . . 145
37 Finding platform resources . . . 146
38 Getting ready . . . 147
38.1 Firmware images . . . 147
38.2 Tools . . . 147
39 Getting to know the hardware . . . 148
39.1 Board overview . . . 148
39.2 Connector index . . . 149
39.3 Power and debug . . . 151
39.4 Serial ports . . . 153
39.5 GPS, I2C and CAN . . . 155
39.6 Motors . . . 156
39.7 SPI and Ethernet . . . 158
39.8 Buttons and switches . . . 159
39.9 Notes . . . 159
39.10 Cheat sheet . . . 159
39.11 References . . . 159
40 Configuring the hardware . . . 160
40.1 Chip Fuse Programming . . . 160
40.2 Programming the bootloader (first time) . . . 160
40.3 Updating the PX4 application (routine) . . . 160
40.4 Recovery . . . 160
40.5 Running the flight stack . . . 160
41 Abbreviations . . . 161
42 References . . . 162
43 Application Notes . . . 163
Hardware Architecture and Reference Design . . . 164

1. Introduction . . . 164
2. System hardware architecture . . . 164
3. Processing core . . . 165
4. Power . . . 165
5. Sensing . . . 166
6. Communication interfaces . . . 169
7. Actuation (PWM) . . . 172
8. Hardware integration and bring-up . . . 172
9. Schematic, board layout, and bill of materials . . . 173
10. Abbreviations . . . 173
11. References . . . 174

44 Programming Boot Configuration Fuses . . . 175
44.1 1. Scope and prerequisites . . . 175
44.2 2. Create an i.MX RT1064 workspace . . . 175
44.3 3. Open the OTP configuration . . . 176
44.4 4. Select the boot fuse field . . . 177
44.5 5. Program the fuse . . . 178
44.6 6. Confirm completion . . . 178
44.7 7. Generate a reusable programming script . . . 179
44.8 8. Completion criteria . . . 181
44.9 9. Abbreviations . . . 181
44.10 10. References . . . 181

45 Flight Stack . . . 182

46 PX4 Documentation Reference . . . 183

47 MR-MCXN-T1 . . . 184

48 Overview . . . 185
48.1 Functional block diagram . . . 185
48.2 Main components . . . 185

49 Getting Started . . . 187

50 Deploy and Run Quick Start . . . 188
50.1 Overview . . . 188
50.2 Download or build the artifacts . . . 188
50.3 What to flash . . . 189
50.4 Required hardware and connection setup . . . 190
50.5 Flash with LinkServer . . . 191

51 User Manual . . . 192

52 Introduction . . . 193
52.1 MR-MCXN-T1 Hub . . . 193
52.2 NXP Robotics SDK and upstream open-source software . . . 193
52.3 Software enablement framework . . . 193

53 Finding platform resources . . . 195

54 Getting ready . . . 196
54.1 Kit contents . . . 196
54.2 Additional hardware . . . 196
54.3 Assumptions . . . 196
54.4 Minimum system requirements . . . 196
54.5 Software . . . 196

55 Getting to know the hardware . . . 197
55.1 Board overview . . . 197
55.2 Connector index . . . 198
55.3 Power, Ethernet and USB . . . 199
55.4 CAN . . . 200
55.5 Debug . . . 201
55.6 Board-to-board interface . . . 202
55.7 LED chain and RTC battery . . . 203
55.8 Buttons and switches . . . 204
55.9 On-board devices . . . 204
55.10 Cheat sheet . . . 205
55.11 Notes . . . 205
55.12 References . . . 206
56 Configuring the hardware . . . 207
56.1 Installing and configuring software tools . . . 207
57 Abbreviations . . . 208
58 References . . . 209
59 Application Notes . . . 210
Hardware Architecture and Reference Design . . . 211
1. Introduction . . . 211
2. System hardware architecture . . . 212
3. MR-MCXN-T1 Hub hardware . . . 213
4. MR-MCXN-T1-IO hardware . . . 221
5. MR-MCXN-T1-F9P hardware . . . 222
6. MR-MCXN-T1-OFL hardware . . . 224
7. Software context . . . 226
8. Hardware installation and bring-up . . . 226
9. Conclusion . . . 227
10. Abbreviations . . . 227
11. References . . . 229
60 Zenoh_PX4 Communication on the MR-MCXN-T1 using ZROS . . . 230
60.1 Overview . . . 230
60.2 Topology . . . 230
60.3 Prerequisites . . . 230
60.4 Default MR-MCXN-T1 node configuration and general commands to check its current state . . . 231
60.5 Scenario 1: MR-MCXN-T1 node (client) - Zenoh router (server) . . . 232
60.6 Scenario 2: MR-MCXN-T1 node (peer) - PX4 (peer) . . . 232
60.7 Additional Notes . . . 234
61 MR-ESC-AM32 . . . 235
62 Overview . . . 236
62.1 Functional block diagram . . . 236
62.2 Main components . . . 236
63 Getting Started . . . 238
64 Deploy and Run Quick Start . . . 239
64.1 Get the artifacts first . . . 239
64.2 What to flash . . . 239
64.3 Connect . . . 240
64.4 Flash with LinkServer . . . 240

64.5 Flash with Segger J-Link (alternative) . . . 240
64.6 Configure the ESC . . . 241
65 User Manual . . . 242
66 Introduction . . . 243
66.1 NXP Robotics SDK and upstream open-source software . . . 243
67 Finding platform resources . . . 244
68 Getting ready . . . 245
68.1 Kit contents . . . 245
68.2 Additional required hardware . . . 245
68.3 Assumptions . . . 245
68.4 AM32 firmware . . . 245
69 Getting to know the hardware . . . 246
69.1 Board overview . . . 246
69.2 Connector index . . . 247
69.3 Power and motor . . . 248
69.4 Control . . . 249
69.5 Expansion pads . . . 250
69.6 Test pads . . . 250
69.7 Buttons and switches . . . 251
69.8 AM32 firmware . . . 251
69.9 Cheat sheet . . . 252
69.10 Notes . . . 252
69.11 References . . . 252
70 Board components and connectors . . . 253
70.1 Kit featured components . . . 253
70.2 LEDs . . . 254
70.3 Switches . . . 254
70.4 Schematic, board layout, and bill of materials . . . 255
71 Firmware features . . . 256
71.1 Motor control . . . 256
71.2 Input protocols . . . 256
71.3 Telemetry output . . . 256
71.4 Configuration and updates . . . 256
71.5 Bootloader . . . 257
72 Installing and configuring software tools . . . 258
73 Configuring the hardware . . . 259
73.1 Setting up the ESC with the RT1176 VMU . . . 259
73.2 Working on target . . . 261
74 Abbreviations . . . 264
75 References . . . 266
76 Application Notes . . . 267
Hardware Architecture and Reference Design . . . 268
About the AM32 community . . . 268
Hardware description . . . 269
PWM input interface . . . 270
Power supply . . . 271
Back-EMF (BEMF) detection network . . . 272

Gate driver and power stage 273
Debug interface 275
Power LED and voltage sense 276
Addressable RGB LED and interface 277
Hall / encoder interface 277
ISPMODE switch 278
Schematic, board layout, and bill of materials 278
Conclusion 278
Abbreviations 278
References 279

1 Robotics SDK Introduction

Welcome to the NXP Robotics SDK documentation. This portal brings together the getting-started, setup, and reference guides for building robotics applications on the NXP Mobile Robotics platforms (MR-NavQ95, MR-VMU-Tropic, MR-MCXN-T1, MR-ESC-AM32; i.MX 95, eIQ Neutron NPU, Kinara ARA-240 DNPU, and the PX4 flight stack).

The NXP Robotics SDK provides validated software support across these platforms, spanning two software domains:

- Real-time domain – flight- and motion-control firmware (PX4 on NuttX, Zephyr RTOS, ESC firmware, System Manager, and the real-time control core).
- Compute domain (Linux / Application Processor) – the application-processor software stack (Linux BSP, U-Boot, ROS2, camera and vision pipeline, NPU inference, and connectivity).

New here? Head to Getting Started to set up and build the SDK, then pick your platform under Platforms.

1.1 Supported platforms

Platform	Based on	Role	Start here
MR-NavQ95	i.MX 95 (Cortex-A55 and Cortex-M7)	Compute and real-time vehicle computer	MR-NavQ95
MR-MCXN-T1	MCXN947 (Zephyr)	Sensor / shield node (GNSS, IO, OFLO)	MR-MCXN-T1
MR-VMU-Tropic	i.MX RT1064 (PX4 / NuttX)	Flight controller	MR-VMU-Tropic
MR-ESC-AM32	MCXA133 (AM32)	Motor-control ESC	MR-ESC-AM32

1.2 Explore the docs

Release Notes

Cumulative release-notes record for the NXP Robotics SDK, newest first.

NXP Robotics SDK — Release Notes
Getting Started

Set up and build the NXP Robotics SDK: prerequisites, download, and the `west` build/command reference.

NXP Robotics SDK
Platforms

Per-platform guides: board bring-up, connectivity and peripherals, on-device AI, and robotics middleware.

MR-NavQ95

System

Cross-platform topics that apply across the robotics hardware.

System Level Integration Guide

2 System Level Integration Guide

NXP Robotics SDK v0.7.0

The NXP Robotics SDK provides software, reference configurations, and documentation for building distributed robotics systems using NXP compute, real-time control, sensing, connectivity, and motor-control platforms.

This guide presents the system-level architecture of NXP Robotics SDK v0.7.0. It explains how the supported platforms work together, identifies the software and communication mechanisms used between them, and describes the end-to-end data paths validated for this release. Platform-specific hardware setup, software installation, flashing, and configuration procedures remain in the corresponding user manuals and application notes.

NXP Robotics SDK v0.7.0 primarily targets drone systems. The drone architecture in this guide is a reference integration baseline rather than a mandatory product architecture. Individual products may use a subset of the platforms or replace particular components according to their performance, safety, cost, size, weight, and fault-containment requirements.

Only configurations explicitly marked Supported have been enabled and validated within the stated v0.7.0 validation scope. Planned elements illustrate intended roadmap direction and are not part of the supported release.

> Note: Scope. NXP Robotics SDK v0.7.0 is primarily targeted at drones. Accordingly, the system-level architecture in this document is described from a drone perspective. The same building blocks, data fabric, and methods also apply to other robotics form factors such as AMRs and humanoids; where relevant this is noted, but the reference architecture and validation shown here are for drones.

2.1 How to read this document

Status is shown consistently throughout, in both the tables and the diagrams:

Symbol	Meaning
Yes	Supported — enabled and validated in v0.7.0
Planned	Planned — on the roadmap for a later release
—	Not applicable to this platform

In the diagrams, solid boxes are supported in v0.7.0 and dashed/dimmed boxes are planned.

2.2 1. Overview

The NXP Robotics SDK provides validated software for a family of mobile-robotics boards that together cover the full compute-to-actuation range of a robot. A typical NXP robot is built from two cooperating software domains running across one or more boards:

- Compute domain (Linux / Application Processor) — the high-level “brain”: ROS 2, the camera and vision pipeline, on-device AI inference, and connectivity. This runs on the Cortex-A cores of the MR-NavQ95 (i.MX 95), optionally accelerated by an Ara240 vision-AI NPU module.
- Real-time domain (RTOS / firmware) — the deterministic “reflexes”: flight and motion control, sensor and I/O nodes, and motor control. This runs on the Cortex-M real-time core of the MR-NavQ95, on the MR-VMU-Tropic flight controller, on the MR-MCXN-T1 sensor/I-O nodes, and on the MR-ESC-AM32 electronic speed controller.

The two domains are joined by a common, ROS 2-native communication fabric — ROS 2 over Zenoh on the Linux side, zenoh-pico on the microcontroller nodes, and RPMsg between the cores of the i.MX 95 — carried physically over 100BASE-T1 automotive Ethernet, CAN, and the standard sensor buses (MIPI-CSI, SPI, I2C, UART).

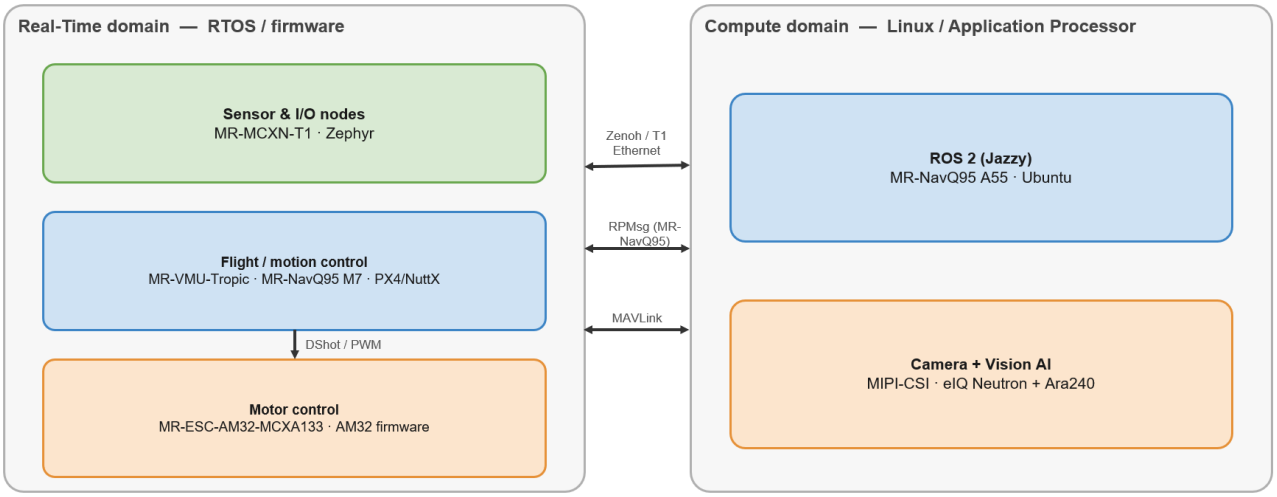


Figure 1 — The two software domains of the NXP Robotics SDK and how they connect.

2.3 2. The platforms at a glance

The SDK supports four hardware platforms. The MR-NavQ95 is the system compute host and can also run the real-time flight stack on its Cortex-M core; the other three are dedicated real-time boards. For board bring-up, flashing, and pin-level detail, follow the per-platform documentation in the SDK repository (robotics-docs).

Platform	Silicon	Role in the system	Domain(s)
MR-NavQ95	i.MX 95 (A55, M7, and Neutron NPU)	Vehicle computer — perception, AI, ROS 2, connectivity; optional on-board flight control	Compute and Real-time
MR-VMU-Tropic	Flight-controller SoC	Flight / vehicle-management controller	Real-time

Platform	Silicon	Role in the system	Domain(s)
MR-MCXN-T1	MCX N (Cortex-M)	Sensor and I/O node on 100BASE-T1 Ethernet	Real-time
MR-ESC-AM32	MCX A (Cortex-M)	Electronic speed controller (motor drive)	Real-time
Ara240 (add-on)	Vision-AI NPU (M.2)	Vision-AI accelerator for the MR-NavQ95	Compute

Note: The Ara240 is an optional companion NPU module on the MR-NavQ95 M.2 connector. The i.MX 95 on-SoC Neutron NPU provides AI inference on its own; the Ara240 adds headroom for heavier vision workloads. See the eIQ Neutron NPU and ARA-240 DNPU Application Notes.

2.4 3. Hardware view

At the hardware level, an NXP drone is organized around the MR-NavQ95 vehicle computer as the central compute hub, a flight/motion controller for deterministic control, sensor/I-O nodes distributed on automotive Ethernet, and one or more ESCs driving the motors.

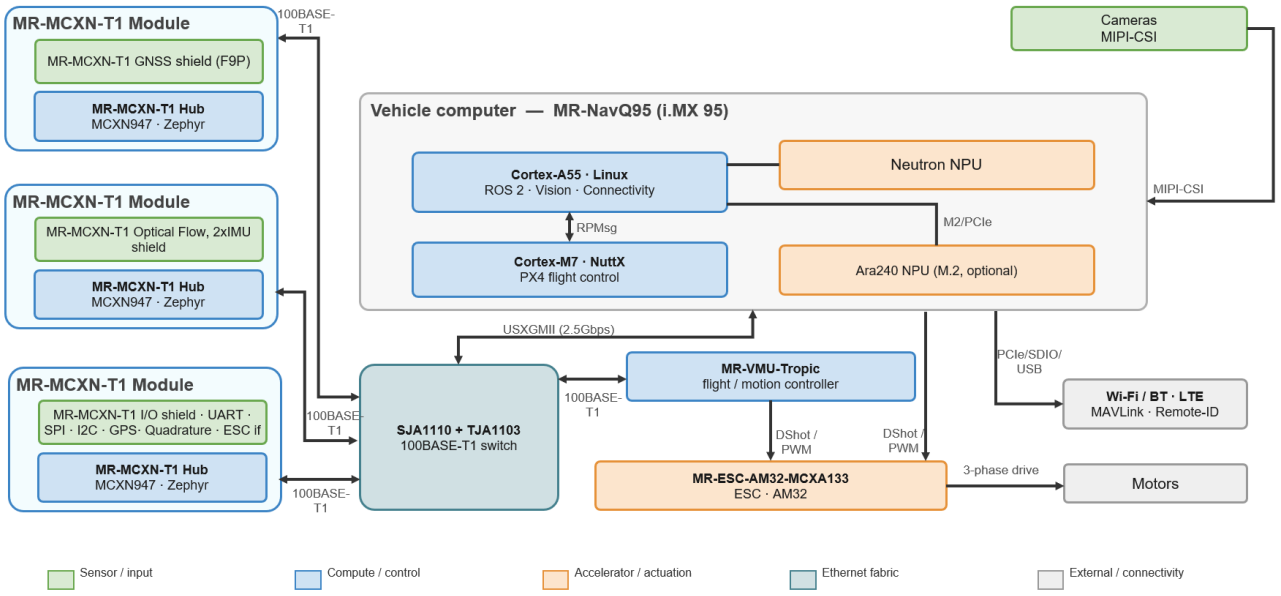


Figure 2 — System hardware topology (v0.7.0). The MR-NavQ95 hosts both the Compute (A55) and Real-time (M7) domains; sensor nodes and the flight controller share a 100BASE-T1 Ethernet fabric; ESCs are driven over DShot/PWM.

Interconnects

Interconnect	Where it is used	Notes
100BASE-T1 Ethernet (SJA1110 and TJA1103)	Between MR-NavQ95, MR-VMU-Tropic, MR-MCXN-T1 nodes	The system data backbone; carries the Zenoh fabric between nodes. See the Ethernet Application Note.
RPMsg	Between the i.MX 95 A55 and M7 on the MR-NavQ95	Inter-core messaging inside the vehicle computer.
MIPI-CSI	Cameras → MR-NavQ95	Camera capture into the vision pipeline. See the Camera Application Note.
CAN / CAN-FD	Auxiliary sensors, servos, peripherals	Field bus for distributed nodes and payload devices.
DShot / PWM	Flight/motion controller → ESC	Motor-command protocol. DShot300 and DShot600 supported in v0.7.0.
SPI / I2C / UART	On-board sensors and peripherals	Standard MCU sensor buses on the MR-MCXN-T1 shields and controllers.
Wi-Fi / Bluetooth / LTE	MR-NavQ95	Wireless connectivity for telemetry, command, and data offload.

2.5 4. Software layer model

The NXP Robotics SDK is organized as a set of layered software building blocks spanning compute, communication, real-time control, and hardware platforms. At the application level, the system is composed of two complementary software domains: Autonomy & AI, responsible for perception, decision making, and robot intelligence, and Flight & Vehicle Control, responsible for deterministic sensing, state estimation, and actuator control. These domains operate as cooperating peers, exchanging information through the underlying communication infrastructure while remaining logically independent of one another. As a result, they are represented side-by-side in the layer model rather than as higher and lower layers.

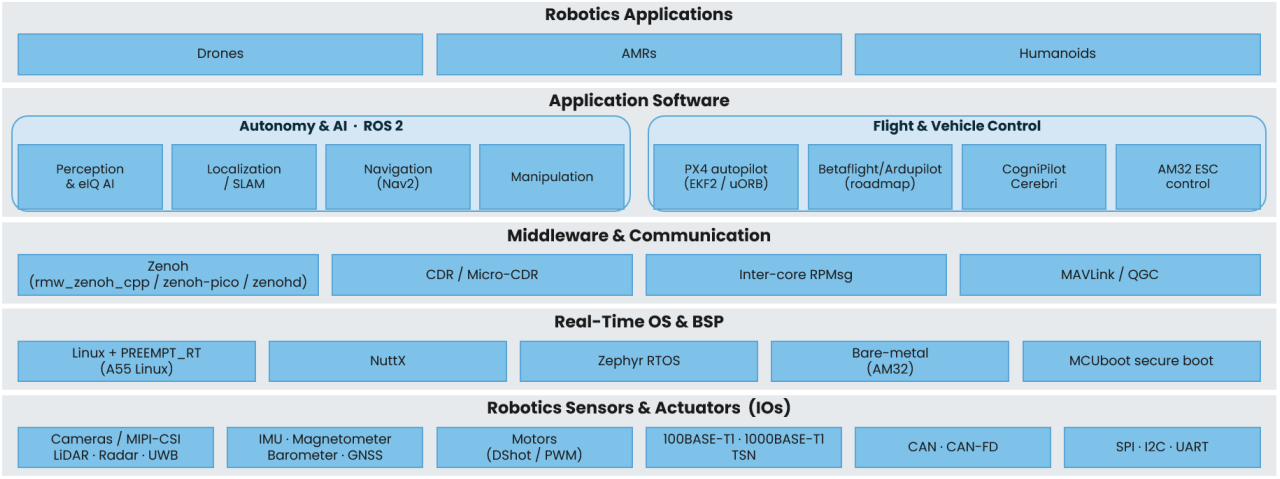


Figure 3 — The v0.7.0 software layer model. Autonomy & AI (ROS 2) and Flight & Vehicle Control (PX4) are co-equal application software that talk through the Connectivity & Data Fabric layer beneath them.

The layers, top to bottom:

- Robotics Applications — the robot mission and reference applications (v0.7.0 focus: drone).
- Application software (over the data fabric) — contains two equal groups: Autonomy & AI (ROS 2 nodes, perception / Vision AI) and Flight & Vehicle Control (PX4 autopilot with EKF2 / uORB, MAVLink).
- Connectivity & Data Fabric — Zenoh (rmw_zenoh_cpp / zenoh-pico / zenohd), CDR / Micro-CDR serialization, inter-core RPMsg, and the MAVLink router.
- Real-time OS & BSP — Linux (A55), NuttX (PX4), Zephyr RTOS, and MCUboot secure boot.
- Sensors, Actuators & Hardware — cameras (MIPI-CSI), GNSS / IMU / optical flow, 100BASE-T1 (SJA1110 / TJA1103) and CAN, motors (DShot / PWM).

> Note: Applicability. The layer model and the methods in this document are described for drones, but the same building blocks apply to other robotics form factors such as AMRs and humanoids — typically by adding autonomy packages (navigation, VSLAM) and additional sensors on the same fabric in a later release.

2.6 5. The communication fabric

The NXP Robotics SDK provides a unified data-sharing architecture across Linux and microcontroller nodes while using the communication mechanism best suited to each domain. Distributed sensor and application data are exchanged through a Zenoh-based data plane, consisting of rmw_zenoh_cpp for ROS 2 applications on the MR-NavQ95, zenohd as the local Zenoh router, and zenoh-pico on resource-constrained microcontroller nodes such as the MR-MCXN-T1. This enables supported MCU-generated data to be consumed by Linux applications as standard ROS 2 topics without requiring a user-developed gateway application. Within the heterogeneous i.MX 95 device, communication between the Cortex-A55 and Cortex-M7 cores is provided through RPMsg, while MAVLink is used for flight-controller telemetry, status, and command exchange.

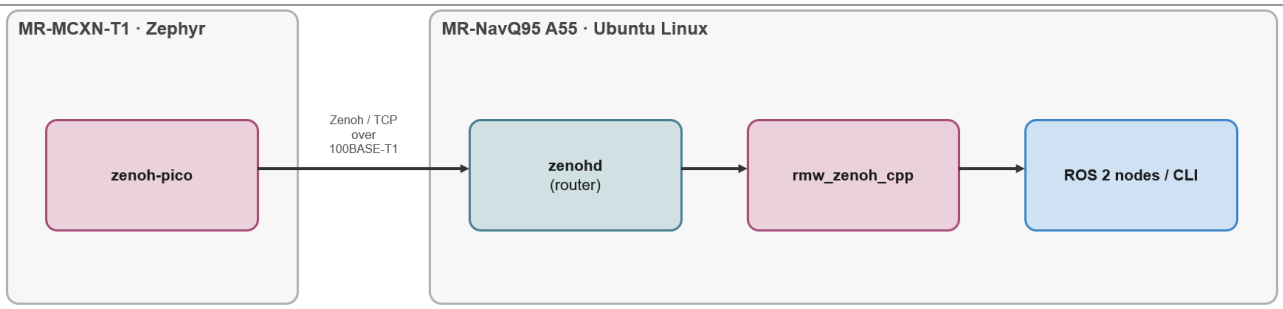


Figure 4 — The Zenoh-based ROS 2 fabric. A microcontroller node using zenoh-pico interoperates directly with ROS 2 on the MR-NavQ95 through zenohd and rmw_zenoh_cpp.

> Note: For the exact configuration and validation of the T1-node Zenoh link (px4_msgs install, static T1 IP, SJA-switch network fixes, zenoh_px4 client-mode setup, and verification with `ros2 topic echo`), see the ROS 2 Multinode Communication guide. This document does not repeat those steps.

2.7 6. Reference architecture (drone)

The following reference architecture shows how the SDK components combine for a drone — the primary target of v0.7.0. Solid boxes are supported in v0.7.0; dashed/dimmed boxes are planned for a later release.

The MR-NavQ95 hosts perception, AI, ROS 2, and connectivity on Linux (A55) while running the PX4 flight stack on the real-time core (M7); an MR-VMU-Tropic can serve as a dedicated flight controller. Sensor nodes attach over 100BASE-T1 Ethernet, and the MR-ESC-AM32 drives the motors.

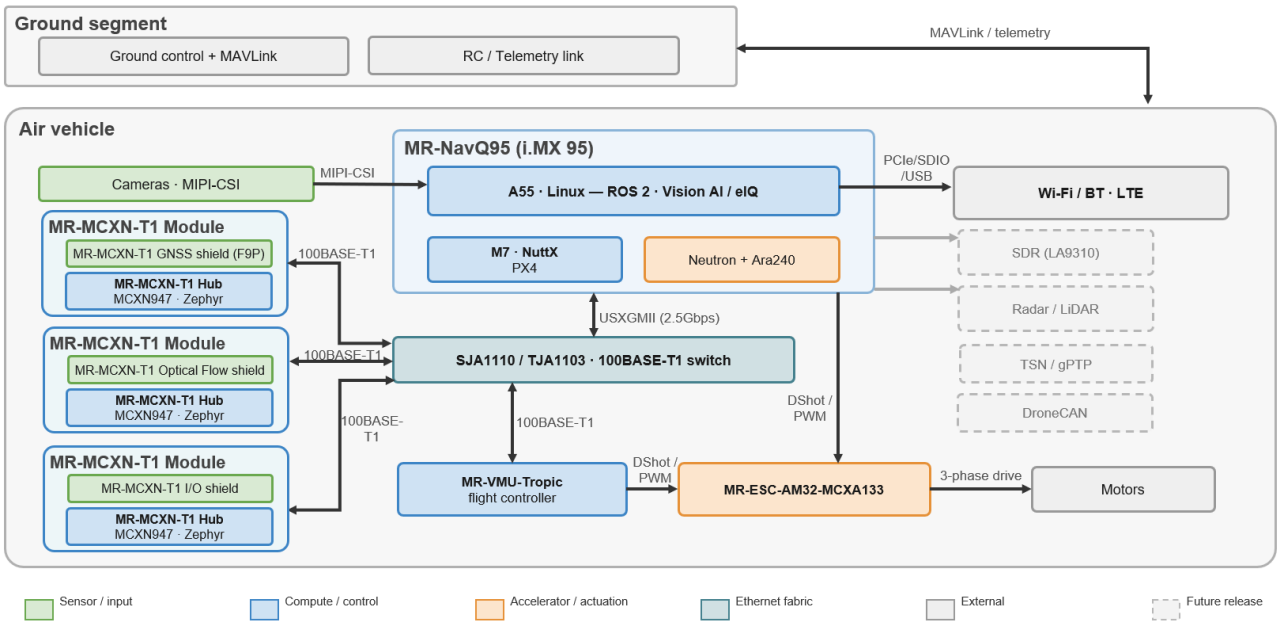


Figure 5 — Drone reference architecture. Solid = v0.7.0; dashed = planned. Radar/LiDAR, SDR, and TSN/gPTP/DroneCAN are on the roadmap.

> Note: Other form factors. An AMR or humanoid reuses the same MR-NavQ95 compute, sensor nodes, connectivity, and ROS 2 baseline shown here; it typically adds autonomy packages (navigation, VSLAM) and

> further sensors on the same fabric. These additions are planned for a later release and are not part of the v0.7.0 validated scope.

Because the MR-NavQ95 is a heterogeneous i.MX 95 device, its Cortex-M7 real-time core runs the same PX4/NuttX flight stack as the MR-VMU-Tropic. So, the MR-NavQ95 can serve as the companion computer and the flight controller in a single board. Depending on drone complexity, the architecture can therefore be built around a dedicated MR-VMU-Tropic flight controller, around the MR-NavQ95 alone (compute and flight control combined, reducing board count, size, and weight), or around both MR-VMU-Tropic and MR-NavQ95 together. Simpler or space-constrained vehicles benefit from consolidating onto the MR-NavQ95, while more complex or higher-assurance vehicles may prefer a dedicated MR-VMU-Tropic to keep the real-time flight control and the companion compute on separate boards.

Drone capability matrix (v0.7.0)

The capability matrix maps each software-layer component to the platform that hosts it and its v0.7.0 status.

Layer	Component	Platform	v0.7.0
Applications	Reference apps (Hover / Gesture), Vision AI / eIQ	MR-NavQ95 (+ Ara240)	Yes
Application SW – Autonomy & AI	ROS 2 (Jazzy), DDS, Zenoh	MR-NavQ95	Yes
Application SW – Flight & Vehicle Control	PX4	MR-NavQ95, MR-VMU-Tropic	Yes
	Betaflight	MR-NavQ95, Tropic	Planned
	ArduPilot	MR-NavQ95	Planned
Connectivity & Data Fabric	Zenoh (rmw_zenoh_cpp / zenoh-pico / zenohd), CDR / Micro-CDR, RPMsg, MAVLink router	(all)	Yes
	TSN / gPTP, DroneCAN	MR-NavQ95, MCXN-T1	Planned
Real-time OS & BSP	NuttX, Zephyr, Linux, MCUboot	MR-NavQ95, Tropic, MCXN-T1	Yes
Services & Drivers	MAVLink / Remote-ID, Camera (MIPI-CSI)	MR-NavQ95	Yes

Layer	Component	Platform	v0.7.0
	Ethernet (SJA1110 and TJA1103), Wi-Fi / BT / LTE, ESC (AM32)	MR-NavQ95, ESC-AM32	Yes
	SDR (LA9310)	MR-NavQ95, MCXN-T1	Planned
Hardware	MR-NavQ95, MR-VMU-Tropic, MR-ESC-AM32, MR-MCXN-T1, Ara240	(all)	Yes
	Radar / LiDAR, Dual Ara240	MR-NavQ95, sensors	Planned

2.8 7. End-to-end data flows

This section describes the data flows that the SDK enables in v0.7.0.

7.1 Distributed sensor data (MR-MCXN-T1)

A sensor connected to an MR-MCXN-T1 node (optical flow, GNSS, or the I/O expander) is encoded as a PX4 message and published over the Zenoh fabric. On the compute host it can be consumed as a normal ROS 2 topic by a node on the MR-NavQ95 A55, and on the flight-control side it can be consumed by a PX4 flight controller.

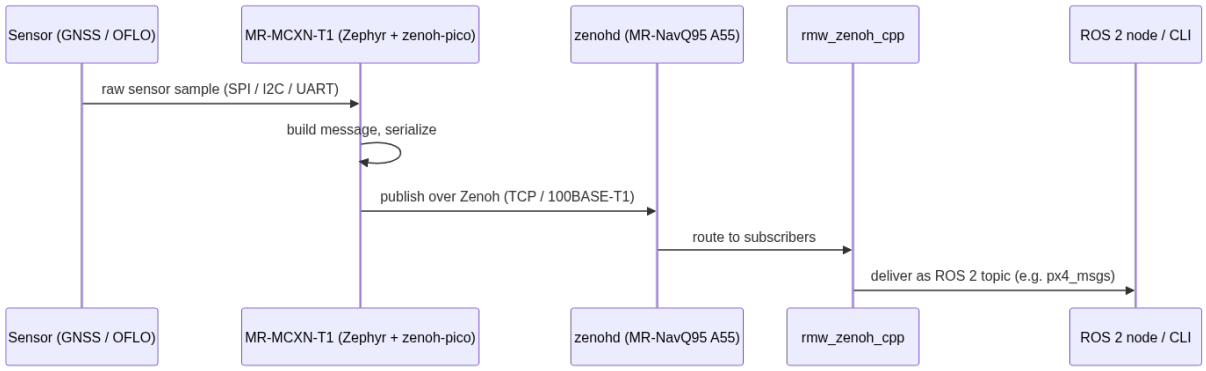


Figure 6 — Distributed sensor to ROS 2 data flow.

The following table records which of these consumers are validated as of the current release. It is the authoritative view of the supported MR-MCXN-T1 sensor data paths — the MCXN-T1 node validated setup matrix.

Setup variant	Data path	MCXN-T1 OFLO (optical-flow shield)	MCXN-T1 F9P / GNSS (F9P GNSS shield)	MCXN-T1 IO (IO expander shield)
Routed Zenoh → ROS 2 and PX4 (NavQ95)	T1 node → TCP client → Zenoh router (NavQ95 Ubuntu / A55). Router serves both consumers at once: topics visible to ROS 2 on the A55 and routed over Ethernet to PX4	✅	✅	✅
Routed Zenoh → ROS 2 only (NavQ95, no PX4)	T1 node → TCP client → Zenoh router (NavQ95 Ubuntu / A55) → ROS 2 on A55 (`ros2 topic list` / `ros2 topic echo`) — no PX4 connected	✅	✅	✅
Peer-to-Peer PX4 → MavLink → QGC (NavQ95 M7)	T1 node ↔ zenoh peer-to-peer PX4 (NavQ95 M7) → MAVLink over RPMsg → mavlink-router (A55) → Ethernet LAN → QGroundControl on laptop	✅	✅	✅
Peer-to-Peer PX4 (MR-VMU-Tropic)	T1 node ↔ PX4 on MR-VMU-Tropic, zenoh peer-to-peer over T1 Ethernet	✅	✅	✅
Peer-to-Peer PX4 (VMU-RT1176)	T1 node ↔ PX4 on VMU-RT1176, zenoh peer-to-peer over T1 Ethernet	✅	✅	✅

> Note: The configuration and verification steps for the MR-MCXN-T1 → NavQ95 A55 Zenoh link are documented in the ROS 2 Multinode Communication guide; they are not repeated here.

7.2 Camera and Vision AI

Two complementary views describe the Vision-AI data path. They serve different purposes:

- Conceptual view (Figure 7) — shows the logical stages (capture, inference, post-process) and where inference is offloaded (Neutron NPU or Ara240). Use it to understand the flow at a glance and to see the accelerator choice.
- Concrete pipeline view (Figure 8) — shows an actual GStreamer / NNStreamer pipeline on the i.MX 95, mapping each processing function to the specific hardware accelerator that executes it. It can be used as a template when building or profiling a real pipeline.

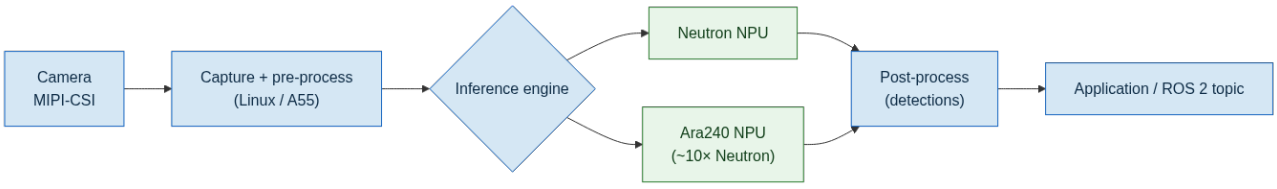


Figure 7 — Conceptual camera-to-Vision-AI data flow. Models run on the Neutron NPU or the Ara240 module. See the eIQ Neutron NPU and ARA-240 DNPU Application Notes.

Figure 8 shows an example pipeline for an object-detection model. Camera frames are captured with libcamerasrc, corrected and rotated, then split into an inference branch (downscale → tensor conversion → model inference on the NPU → tensor sink → user post-processing) and a display branch (colour conversion → overlay → display). Each block is annotated with the i.MX 95 hardware accelerator that runs it (3D GPU, DPU, or NPU); the AI model itself is generic and can be replaced with any supported detector.

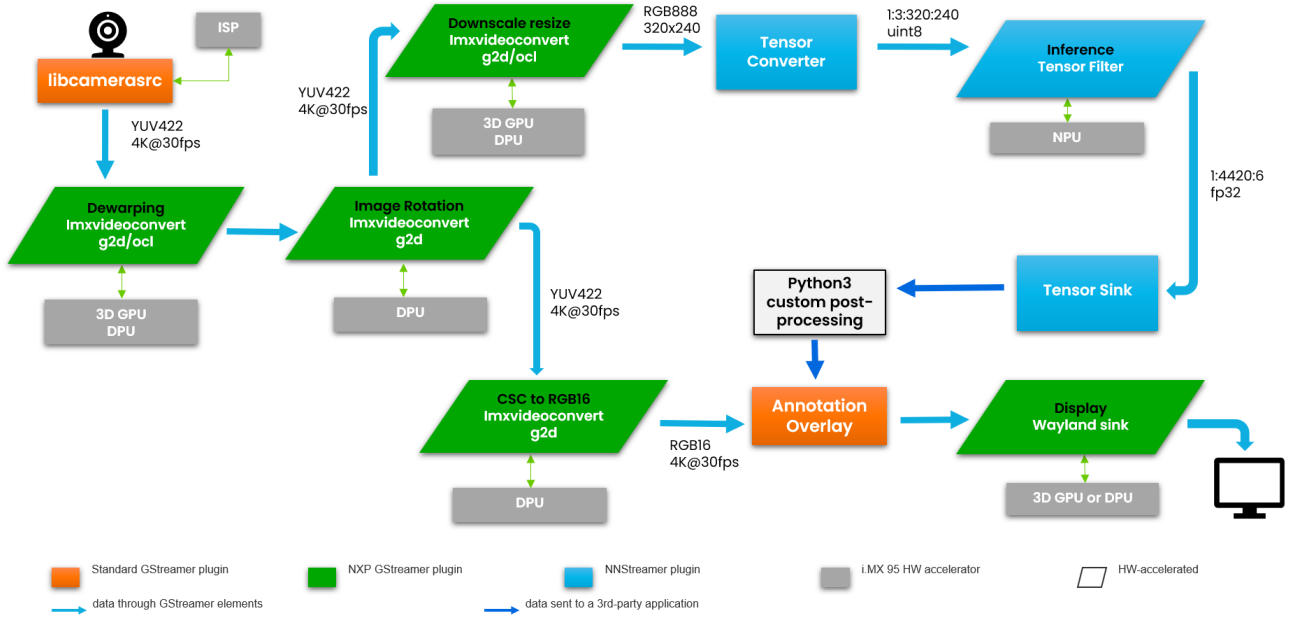


Figure 8 — Example Vision-AI pipeline on the i.MX 95 (GStreamer + NNStreamer). Processing functions are mapped to the specific hardware accelerators; the AI model name is generic.

> Note: For building and running the camera and Vision-AI pipeline (drivers, models, and the eIQ / Neutron and Ara240 runtimes), see the Camera, eIQ Neutron NPU, and ARA-240 DNPU Application Notes in the SDK documentation. This document does not duplicate that content.

2.9 Abbreviations

Acronym	Description
AMR	Autonomous mobile robot
CAN	Controller Area Network
DDS	Data Distribution Service
DNPU	Discrete neural processing unit

Acronym	Description
ESC	Electronic speed controller
MIPI-CSI	Mobile Industry Processor Interface Camera Serial Interface
NPU	Neural processing unit
RPMsg	Remote Processor Messaging
ROS 2	Robot Operating System 2
RTOS	Real-time operating system
SDK	Software development kit
TSN	Time-Sensitive Networking

2.10 References

- Modular Drone Platform product page — overview, documentation, software, tools, and ordering information
- MR-NavQ95 Hardware Architecture and Reference Design
- MR-VMU-Tropic Hardware Architecture and Reference Design
- MR-MCXN-T1 Hardware Architecture and Reference Design
- MR-ESC-AM32 Hardware Architecture and Reference Design
- Camera Application Note
- Automotive Ethernet Application Note
- eIQ Neutron NPU Application Note
- ARA-240 DNPU Application Note

3 Release Notes

This document is the cumulative release notes record for the NXP Robotics SDK. Each release is documented as a self-contained entry under Releases, with the newest release first.

3.1 About the Robotics SDK

The NXP Robotics SDK provides validated software support across NXP robotics hardware platforms, organized into two software domains:

- Real-time domain — flight and motion-control firmware (PX4 on NuttX, Zephyr RTOS, ESC firmware, System Manager, and the real-time control core)
- Compute domain (Linux/Application Processor) — the application-processor software stack (Linux BSP, U-Boot, ROS2, camera and vision pipeline, NPU inference, and connectivity)

Each release entry below lists the supported platforms, the capabilities enabled per platform, the new features introduced, and the known limitations for that specific version.

3.2 How to read this document

- Releases are listed newest-first. The latest release appears immediately below.
- Every release entry follows the same fixed structure, so features and limitations can be compared across versions.
- The capability matrix in each entry reflects the state of that release only.

3.3 Releases

v0.7.0

Overview

NXP Robotics SDK v0.7.0 delivers validated software support across four robotics hardware platforms (MR-NavQ95, MR-VMU-Tropic, MR-MCXN-T1, MR-ESC-AM32), spanning both the Real-time and Compute domains.

- Quick Start Guides are provided for each supported platform.
- Platform binaries are available for all four supported platforms.
- Test Plan and Test Report are available for this release.

Software domains

Real-time domain (PX4, Zephyr, ESC, System Manager)

Component	Description	Platforms
PX4 Autopilot (NuttX)	Flight-control stack running on NuttX	MR-NavQ95 (real-time core), MR-VMU-Tropic
Zephyr RTOS	Sensor and peripheral firmware, shield-node applications	MR-MCXN-T1 (+ shields), MR-VMU-Tropic
Cerebri (CogniPilot)	ZROS integration apps that bridge shield sensors to ROS2 over the T1 link	MR-MCXN-T1 (+ shields)
ESC firmware (AM32)	Motor-control firmware; DShot / PWM protocols	MR-ESC-AM32
System Manager	Low-level platform management and resource partitioning	MR-NavQ95

Compute domain (Linux / Application Processor)

Component	Description	Platforms
Linux BSP	Kernel 6.18.20-2.0.0, device trees, drivers	MR-NavQ95
U-Boot	Bootloader	MR-NavQ95
ROS2	ROS2 middleware, install scripts, camera / vision nodes	MR-NavQ95
Zenoh transport	Host ↔ board ROS2 connectivity over Zenoh (`rmw_zenoh`), including the ZROS → ROS2 bridge for MR-MCXN-T1 shield nodes over the T1 link	MR-NavQ95, MR-MCXN-T1
Camera / Vision	MIPI-CSI camera pipeline and Vision AI example application	MR-NavQ95
NPU Inference	ARA240 and Neutron NPU inference	MR-NavQ95
Connectivity	SJA1110 Ethernet switch, Wi-Fi / Bluetooth, LTE PCIe modem	MR-NavQ95

Software components and versions

The following table lists the versions of the main software components integrated in this release.

Domain	Component	Version	Notes
—	NXP Robotics SDK	v0.7.0	Overall release version
Compute	Linux kernel / BSP	6.18.20-2.0.0	MR-NavQ95
Compute	Host OS	Ubuntu 24.04 LTS	Application-processor user space (MR-NavQ95)
Compute	ROS2	0.11.0-1noble.20260903.093452	ROS2 middleware and nodes
Compute	U-Boot	2026.04	MR-NavQ95 bootloader
Real-time	PX4 Autopilot	1.17.0 (1cc2589cd20ca0c01d1de63d4287ceeeca52bf71 (MR-VMU-Tropic) / d09136b174db99e598095ce9255c52d203e668df (MR-NavQ95))	Flight-control stack (MR-NavQ95, MR-VMU-Tropic)
Real-time	NuttX RTOS	12.12.0	Underlying RTOS for PX4
Real-time	Zephyr RTOS	4.4.0+	MR-MCXN-T1 (+ shields), MR-VMU-Tropic
Real-time	Cerebri (CogniPilot)	ZROS integration apps	MR-MCXN-T1 shield nodes (OFLO, GNSS, IO)
Real-time	ESC firmware (AM32)	2.20	MR-ESC-AM32
Compute	eIQ Neutron SDK	3.1.2	Neutron NPU inference stack (MR-NavQ95)

Domain	Component	Version	Notes
Compute	Ara SDK	2.1.1	ARA240 NPU runtime (rt-ara-sdk, MR-NavQ95)

Supported platforms and capabilities

MR-NavQ95 (i.MX 95 vehicle computer) — Compute and Real-time

- Compute (Linux): Linux BSP 6.18.20-2.0.0; ROS2; MIPI-CSI camera pipeline and Vision AI example app.
- ROS2 connectivity: host ↔ board ROS2 communication over Zenoh (`rmw_zenoh`).
- Connectivity: SJA1110 Ethernet switch; Wi-Fi and Bluetooth; LTE PCIe modem.
- AI / NPU inference: ARA240 and Neutron NPU validated. ARA240 delivers up to approximately 10× the inference performance of Neutron across YOLOv8 / YOLOX models.
- Real-time: PX4 on NuttX running on the real-time control core.

MR-VMU-Tropic (flight controller) — Real-time

- PX4 NuttX flight stack fully validated.
- All Zephyr peripheral tests fully validated.

MR-MCXN-T1 (+ shield variants) (sensor / IO node) — Real-time (Zephyr / Cerebri) with ROS2 sensor bridging

- Three shield variants validated (GNSS, IO, and OFLO).
- Zephyr peripheral applications for board / peripheral validation.
- Cerebri (CogniPilot) integration apps (OFLO, GNSS, IO) that bridge shield sensors to a ROS2 system over Zenoh via the T1 Ethernet link (ZROS → ROS2).

MR-ESC-AM32 (electronic speed controller) — Real-time (ESC firmware)

- Boot and PWM validated.
- DShot300 and DShot600 supported.
- DShot150 and BDShot150 are not supported on MR-ESC-AM32 because the AM32 main code does not implement DShot150.

Capability matrix

Capability	MR-NavQ95	MR-VMU-Tropic	MR-MCXN-T1	MR-ESC-AM32
Linux BSP (Compute)	Yes	—	—	—

Capability	MR-NavQ95	MR-VMU-Tropic	MR-MCXN-T1	MR-ESC-AM32
ROS2	Yes	—	Sensor bridge	—
ROS2 over Zenoh (`rmw_zenoh`)	Yes	—	Yes (ZROS → ROS2)	—
Camera (MIPI-CSI)	Yes	—	—	—
NPU inference (ARA240 and Neutron)	Yes	—	—	—
Connectivity (SJA1110 / Wi-Fi+BT / LTE)	Yes	—	—	—
PX4 (NuttX)	Yes	Yes	—	—
Zephyr RTOS	—	Yes	Yes (3 shields)	—
Cerebri (CogniPilot)	—	—	Yes (3 shield apps)	—
ESC firmware (DShot / PWM)	—	—	—	Yes (DShot300/600)

Legend: Yes = enabled and validated · — = not applicable.

New features

Compute domain (Linux / Application Processor)

- ROS2 support on MR-NavQ95, including ROS2 install-script integration into the platform image and a ROS-ready image variant.
- Host ↔ board ROS2 connectivity over Zenoh (`rmw_zenoh`), including the ZROS → ROS2 bridge that carries MR-MCXN-T1 shield-sensor data to a ROS2 system over the T1 Ethernet link.
- MIPI-CSI camera pipeline enabled and validated on MR-NavQ95. Four camera sensors are supported:
  - IMX708 (official Raspberry Pi Camera Module 3) — recommended default choice.
  - IMX477 — when image quality and optics matter most (already supported on FRDM).
  - IMX219 — for lowest-cost platforms (already supported on FRDM).
  - NXP EXPI-OS08A20 — OmniVision OS08A20 8-megapixel color CMOS sensor.

- Vision AI example application with runtime model and camera selection. The Vision AI demo is validated with the NXP EXPI-OS08A20 camera.
- NPU inference on both ARA240 and Neutron, with object-detection models (YOLOv8 / YOLOX).
- Connectivity stack: SJA1110 Ethernet switch, Wi-Fi / Bluetooth, and LTE PCIe modem support.
- Platform image composition additions, including a MAVLink router and a Remote-ID transmitter.
- U-Boot board configuration for MR-NavQ95.
- Docker-based build support for the SDK.
- Improved Setup / Quick Start documentation: step-by-step flashing guides, hardware photos, boot-switch and power-cycle instructions, and software bill-of-materials in the release images.

Real-time domain (PX4, Zephyr, Cerebri, ESC)

- PX4 flight stack (NuttX) validated on MR-NavQ95 and MR-VMU-Tropic.
- Zephyr RTOS applications validated on MR-VMU-Tropic and on the three MR-MCXN-T1 shield variants (GNSS, IO, OFLO).
- Cerebri (CogniPilot) integration apps on MR-MCXN-T1 (OFLO, GNSS, IO) that bridge shield sensors to a ROS2 system over Zenoh (ZROS → ROS2) across the T1 link.
- ESC firmware (AM32) with DShot300 / DShot600 and PWM support on MR-ESC-AM32.
- PX4 command-line tooling enhancements (GPIO system command).

Cross-platform / validation

- End-to-end validation across all four supported platforms and both software domains, covering BSP, ROS2, camera, connectivity, NPU inference, PX4, Zephyr, and ESC.

Known limitations

Limitations are grouped per platform, then split into Hardware and Software gaps.

MR-NavQ95

Hardware

- The current design partitions the Ethernet ENETC interfaces between the M7 (real-time) and A55 (application) cores; achieving functional-safety compliance may require software changes, including redesign of selected components and interfaces.
- Thermal Management: System designers should account for overall thermal management. In particular, Ara240 modules require active cooling and include a separate 5V cooling fan with its own connector. When an Ara240 DNPU is running inference workloads, it generates heat that, if not adequately managed, can raise the temperature of the i.MX95 on the MR-NavQ95 board. The MPU is configured to activate thermal protection at approximately 85 °C, allowing the i.MX95 to reduce its clock frequency and prevent overheating. This expected behavior may reduce performance during sustained inference workloads when cooling is inadequate. System designers are responsible for ensuring adequate thermal management for the intended deployment. In enclosed or stationary installations with limited airflow, consider cooling fans or other active cooling to provide sufficient airflow and keep the board below its throttling threshold, thereby maintaining full inference performance.

- Ara240 cable adapter for NavQ95: The cooling fan on the current ARA2-M2-16G-GT (Ara240 16 GB M.2 Module) EVK can be powered from any suitable 5V source in the system. For convenience, a cable adapter is available from NXP and listed as an accessory on the MR-NavQ95-KIT webpage. P/N: MR-NAVQ95-FANCB

Software

- Linux: device trees do not fully adhere to Linux community standards (e.g. wrong DT node ordering, redundant / commented-out nodes left in the DTS).
- U-Boot: no dedicated board support — software currently relies on the MX95-19×19-EVK board support; users cannot build U-Boot binaries for MR-NavQ95 without Yocto.
- U-Boot: GCC 13 (Ubuntu PoC) is incompatible with GCC 15 (Yocto); resolution is to upgrade to Ubuntu 26.04 (GCC 15) in a future release.
- Networking: the SJA1110 switch uses a DSA (Distributed Switch Architecture) conduit, so TCP checksum offload is not applied across the `end1` → SJA switch → `trx*` forwarding path; plan for this when configuring the Ethernet stack.
- PX4 / M7: PyOCD flashing of the M7 image is unstable and requires multiple runs.
- PX4 / M7: flashing the PX4 image with pyOCD interferes with the SPI bus while an MR-MCXN-T1 (T1) node is connected to the SJA1110 switch — the flash halts during the erase step with a “core #0 is not halted” error. Disconnect the T1 node(s) from the SJA1110 switch before flashing the M7 PX4 image.
- PX4 / M7: M7 depends on Linux for its filesystem, leading to delayed boot.
- PX4 / M7: different software configuration (SHA commit) than the other supported PX4 platforms.
- PX4 / M7: The PX4 `reboot` command and A55/Linux reboot recovery are currently not supported — the M7 and A55 do not properly re-initialize shared resources after a reboot, so a full power-cycle of the SoC is required instead. Enablement of software-triggered reboot recovery is planned for a future release.
- System Manager: insecure / improper resource partitioning between M33 / M7 / AP-NS — e.g. LPI2C4 is accessible from M33, M7 and AP-NS simultaneously.

MR-VMU-Tropic

Hardware

- UART3 is designated at AUX instead of TELEM as it does not support hardware based flow control as per PX4 specifications.

Software

- No Ethernet driver is available in NuttX for this board revision.

MR-MCXN-T1 and Shields

Hardware

- MR-MCXN-T1-GNSS: shield unusable (shorted out) without removal of DNP resistors.
- MR-MCXN-T1-GNSS: BMM350 magnetometer Z-axis failure without a hardware rework (resistor addition).

Software

- Magnetometer and barometer sensor data are not available as ROS2 nodes, blocked by I3C bus driver, and signals being alternatively allocated to LPUART. Optical-flow and GNSS data are available to a ROS2 system via the Cerebri ZROS → ROS2 bridge over the T1 link.

MR-ESC-AM32

Hardware

- Re-configuration of the ESC (for new motors, protocols, etc.) requires reprogramming/configuring a Pixhawk 6X-RT controller as a pass-through interface.

Software

- The AM32 upstream code does not implement the DShot150 / BDShot150 protocol, so likewise these cannot be configured on MR-ESC-AM32.

Post-release patches

The patches listed below were applied on top of the v0.7.0 pinned component revisions to address issues found during post-release validation. Each patch is already present on the branch of the component repository it links to; take the top of tree of each listed repository and branch to pick up the fixes. The patches apply only to v0.7.0.

Patch	Component repository	Affected platform
Correct the NavQ95 onboard BMM350 compass orientation (fixes the ~90° heading error at every boot).	robotics-PX4-Autopilot (`robotics-0.7.0-navq95`)	MR-NavQ95
Increase the `navigator` task stack to clear the load_mon low-stack warning.	robotics-PX4-Autopilot (`robotics-0.7.0-navq95`)	MR-NavQ95
Increase the `gps` driver task stack to clear the load_mon low-stack warning.	robotics-PX4-Autopilot (`robotics-0.7.0-navq95`)	MR-NavQ95
Size the uavcan and TTY work queues for NuttX 12 to restore the stack headroom.	robotics-PX4-Autopilot (`robotics-0.7.0-navq95`)	MR-NavQ95
Applied the right RIHS hash for SensorGps PX4 message type to match the release/1.17 version of px4_msgs repo installed automatically into rootfs.	robotics-apps (`robotics-0.7.0`)	MR-MCXN-T1

4 SDK Setup and Build

The NXP Robotics SDK provides a unified software development environment for building robotics systems on supported NXP platforms. It brings together validated platform software, robotics frameworks, applications, development tooling, and pre-built artifacts into reproducible SDK releases.

The robotics-sdk repository is the SDK's integration and release layer. It defines supported platform and software-stack combinations, pins compatible revisions of the corresponding source repositories, orchestrates builds through west, and provides access to pre-built artifacts.

Firmware and application sources remain in their respective component repositories rather than being duplicated here. This allows individual projects such as PX4, Zephyr-based firmware, AM32, and robotics applications to retain their native development environments while being integrated and validated as part of a common NXP Robotics SDK release.

4.1 Table of contents

- Supported platforms and stacks
- Development Workflows
- Requirements
  - Common requirements
  - Additional requirements for the build-from-source flow
- Prerequisites and configuration
  - Host operating system
  - Corporate network: proxy and CA certificates
- Flow A: getting pre-built images
- Flow B: building images from source
- Command reference guide
- Warnings and environment-specific notes
- Troubleshooting / FAQ

4.2 Supported platforms and stacks

Platform	PX4	Cerebri	Zephyr	AM32
MR-NAVQ95	✅	❌	❌	❌
MR-VMU-TROPIC	✅	❌	✅	❌

Platform	PX4	Cerebri	Zephyr	AM32
MR-ESC-AM32-MCXA133	❌	❌	❌	✅
MR-MCXN-T1	❌	✅	✅	❌

📝 Note: The exact platform names accepted by the tooling (the value passed to `-p/--platform`) are the ones printed by `west sdk list`. Run `west sdk list -v` to also see the supported stacks, shields and examples for each platform.

4.3 Development Workflows

The NXP Robotics SDK supports two development workflows. Pre-built artifacts provide the simplest path for platform evaluation and flashing, while the build-from-source workflow supports development and customization of firmware, applications, configuration, and SDK components.

- Flow A: getting pre-built images. Download the pre-built artifacts (firmware, and the Linux WIC image for MR-NAVQ95) from the artifact store. This flow does NOT require Docker. All that is needed is Python and `west`, which can be installed in a Python virtual environment on the host. Use this flow to flash a supported platform without rebuilding anything. See Flow A: getting pre-built images.
- Flow B: building images from source. Build the firmware for a platform from the pinned upstream sources. Because the SDK spans several software stacks, each with its own toolchain and package dependencies, this flow runs inside a Docker container that bundles every dependency. Docker is only required for this build flow — it provides the cross-platform toolchains (PX4, Zephyr, AM32, etc…) and an isolated, reproducible build environment, so the host does not have to install them. See Flow B: building images from source.

📝 Note: Docker is needed only for Flow B. It is not required to download pre-built images (Flow A).

4.4 Requirements

Common requirements

Both flows need:

- `git`, to clone this repository.
- Network access to `github.com`, from which `west` resolves and clones the pinned manifest projects. Behind a firewall, make sure the proper proxy and CA certificate are configured (see Corporate network).

Flow A (getting pre-built images) additionally needs only:

- Python 3 and `pip` (to create a virtual environment and install `west` and `requests`; `requests` is used by `west sdk download`).
- Network access to the artifact store from which the pre-built binaries are downloaded.

Additional requirements for the build-from-source flow

Flow B (building images from source) builds inside a Docker container, so the host additionally needs:

- Docker (see the installation instructions below).
- At least 25 GB of free disk space on the filesystem that backs Docker's storage (`/var/lib/docker` by default). The final image is around 10 GB, but the build temporarily needs more for downloaded `.deb` packages, the Zephyr SDKs, and intermediate layers, plus additional space for the checked-out sources and build outputs. If the volume holding `/var/lib/docker` is small (a common cause of build failures), point Docker's data-root at a larger disk — see Warning: "No space left on device" during docker build.
- Network access to `github.com`, from which the Dockerfile downloads the Zephyr SDKs while building the image. Behind a firewall, make sure the proper proxy host and port are passed to the Docker commands as shown below.

4.5 Prerequisites and configuration

Host operating system

The Docker-based build flow (Flow B) is validated on Ubuntu 24.04 LTS. Use Ubuntu 24.04 as the host OS; older releases (e.g. Ubuntu 22.04) are not currently supported and may fail while building the Docker image. The download-only flow (Flow A) is not tied to a specific Ubuntu release — any host with Python 3 and `west` can download the pre-built artifacts.

Corporate network: proxy and CA certificates

> Note: This section applies only when the build is performed from a corporate network.

Building from a corporate network typically requires both a proxy and the corporate CA certificate so that TLS connections (to `github.com`, package indexes, etc.) succeed inside the Docker build.

- Proxy. The `docker build` / `docker run` examples below accept `http_proxy` / `https_proxy` values. These are placeholders of the form `http://<proxy>:<port>` — replace `<proxy>` and `<port>` with the values that apply to the local machine and site. Do not copy the examples verbatim; there is no single proxy that works for everyone. If the host does not need a proxy, omit these arguments entirely.
- CA certificate. In environments where corporate TLS inspection is enabled, downloads inside the image build can fail with certificate-verification errors. Make the corporate root CA available during the build, for example by copying the `.crt` into the build context and installing it in the image (`/usr/local/share/ca-certificates/` and running `update-ca-certificates`), or by configuring the CA on the host Docker daemon. Consult the applicable IT documentation for the appropriate certificate file and installation procedure.

4.6 Flow A: getting pre-built images

This flow downloads the pre-built artifacts from the artifact store. It does NOT require Docker — only Python and `west` are needed, installed in a Python virtual environment on the host.

First, make sure `git` and Python 3 are available. On an Ubuntu-based machine:

```
sudo apt-get update
sudo apt-get install -y git python3 python3-pip python3-venv
```

Clone the robotics-sdk repository:

```
mkdir -p ~/robotics-workdir && cd ~/robotics-workdir

git clone -b robotics-0.7.0 https://github.com/nxp-robotics/robotics-sdk.git
```

📝 Note: `-b` selects the branch or tag to use. We recommend always choosing a release branch/tag; for the ROBOTICS_0.7.0 release, use `robotics-0.7.0`.

Create and activate a Python virtual environment, then install `west` into it:

```
python3 -m venv .venv
source .venv/bin/activate

pip install west requests
```

📝 Note: `west sdk download` uses the `requests` package to fetch artifacts from the artifact store, so it must be installed alongside `west`. Without it, the download fails with `The 'requests' package is required for 'west sdk download'`.

Initialize `west` against the cloned manifest:

```
west init -l robotics-sdk/
```

Optionally, bring in the SDK's manifest projects with `west update`:

```
west update
```

📝 Note: `west update` is optional for this flow. `west sdk download` fetches pre-built artifacts from the artifact store and does not need the manifest projects to be cloned, so it works right after `west init`. Run `west update` only if the manifest projects (e.g. the documentation repository) are also wanted locally.

Download the pre-built binaries for the target platform:

```
west sdk download -p <platform>
```

For example, to download the pre-built binaries for the MR-NAVQ95 platform:

```
west sdk download -p mr-navq95
```

📝 Note: The downloaded binaries are placed under `~/robotics-workdir/bin`.

4.7 Flow B: building images from source

Given the number of software stacks supported by the SDK, managing all of their package dependencies on the build machine may be challenging. Because of this, robotics-sdk provides a Dockerfile, which can be used to build

a Docker image with all of the package dependencies included in it. Docker is required only for this build flow; it supplies the cross-platform toolchains and an isolated, reproducible build environment.

To get started, make sure Docker is installed on the build machine. For an Ubuntu-based machine, follow these instructions. To check if everything went smoothly with the installation, try to run the hello-world Docker image:

```
sudo docker run hello-world
```

Possible pitfall (hello-world)

📝 Note: To allow Docker to be managed by non-root users, follow these instructions. Throughout this guide, it is assumed that `sudo` is required.

Apart from Docker, a few system packages are required. These can be installed by running:

```
sudo apt-get update
sudo apt-get install -y git
```

To prepare the development environment, first clone the robotics-sdk repository by running:

```
mkdir -p ~/robotics-workdir && cd ~/robotics-workdir

git clone -b <branch> https://github.com/nxp-robotics/robotics-sdk.git
```

📝 Note: `branch` is the name of the branch or tag to use. We recommend always choosing a release branch/tag. For the ROBOTICS_0.7.0 release, the name of the branch would be robotics-0.7.0.

For example, to clone the robotics-0.7.0 branch:

```
mkdir -p ~/robotics-workdir && cd ~/robotics-workdir

git clone -b robotics-0.7.0 https://github.com/nxp-robotics/robotics-sdk.git
```

A tag can also be passed to the `-b` option to check out a specific release. For example, to check out the `<tag>` tag:

```
mkdir -p ~/robotics-workdir && cd ~/robotics-workdir

git clone -b <tag> https://github.com/nxp-robotics/robotics-sdk.git
```

📝 Note: Release tags are listed on the repository's releases/tags page; use the tag that matches the desired release.

Possible pitfall (detached HEAD)

The next step is to build the Docker image:

```
sudo docker build --build-arg UID=$(id -u) \
                  --build-arg GID=$(id -g) \
                  -t robotics-sdk -f robotics-sdk/Dockerfile .
```

Possible pitfall (docker build proxy)

Possible pitfall (WSL2)

Possible pitfall (disk space)

Now everything should be ready to go. To start the container, run:

```
sudo docker run -it -v ~/robotics-workdir:/home/robotics robotics-sdk
```

Possible pitfall (docker run proxy)

After starting the container, initialize `west` by running:

```
west init -l robotics-sdk/
```

Possible pitfall (git auth)

Optionally, bring in the SDK's manifest projects with `west update`:

```
west update
```

📝 Note: `west update` is optional for building. `west sdk build` sets the platform manifest and runs `west update` automatically to clone the pinned upstream sources before compiling, so the sources do not have to be fetched by hand first. Run `west update` manually only when the SDK's bootstrap manifest projects (e.g. the documentation repository, or the MCXN-T1 ROS2-node apps) are wanted up front.

If everything went well, all of the available platforms can be listed by running:

```
west sdk list
```

To build the firmware for a platform, see Build binaries from source in the command reference guide below.

4.8 Command reference guide

robotics-sdk uses west to manage all of its components (e.g. building firmware images, downloading pre-built binaries, updating git revisions, etc…). The documentation found below provides an overview of what can be achieved using `west` and robotics-sdk.

📝 Note: For all commands requiring a platform name as argument (e.g. `west sdk build -p <platform>`), one of the supported platform names listed by running `west sdk list` must be used.

📝 Note: For all build or download commands, all of the binaries will be available under `~/robotics-workdir/bin`.

1. Displaying help information

The `--help` option can be used to display additional information regarding a command or a sub-command (e.g. documentation, supported options, etc…). This option should be used as the "ultimate" source of truth since it should always be up to date.

Examples of how this option could be used:

```
# print help information about the sdk command
west sdk --help
```

```
# print help information about the build sub-command
west sdk build --help
```

1. List available platforms

To list all of the available platforms:

```
west sdk list
```

```
# prints additional information such as supported stacks, shields and examples
west sdk list -v
```

1. Download pre-built binaries

To download the pre-built binaries:

```
west sdk download -p <platform>
```

For example, to download the pre-built binaries for the MR-NAVQ95 platform:

```
west sdk download -p mr-navq95
```

1. Fetch or update a platform's sources

To set a platform's manifest and clone (or update to the pinned revision) its upstream sources without building:

```
west sdk update -p <platform>
```

For example, to fetch the sources for the MR-NAVQ95 platform:

```
west sdk update -p mr-navq95
```

📝 Note: This step is optional before a build. `west sdk build` performs the same source fetch automatically; use `west sdk update` only to bring in (or refresh) the sources without compiling.

1. Build binaries from source

To build all of the available binaries from source for a given platform:

```
west sdk build -p <platform>
```

For example, to build all of the available binaries for the MR-NAVQ95 platform:

```
west sdk build -p mr-navq95
```

The aformentioned command builds all of the available binaries for the target platform (i.e. all stacks, all examples, etc…). A specific stack can be built by using the `--stack` option:

```
west sdk build -p <platform> --stack <stack>
```

For example, to build the PX4 stack for the MR-NAVQ95 platform:

```
west sdk build -p mr-navq95 --stack px4
```

1. Updating to a new release (tag)

When a new release tag is published, move the `robotics-sdk` workspace to it by fetching the tags, checking out the new tag, and running `west update`. From inside the `robotics-sdk` repository:

```
cd ~/robotics-workdir/robotics-sdk
git remote update
git checkout <new-tag>
west update
```

`west update` after the checkout is needed because the manifest (`west.yml`) may have changed between releases; it reconciles the SDK-level manifest projects (e.g. `robotics-apps`, the documentation repository, etc…) with the new tag.

📝 Note: `west update` here only brings in what is controlled by the SDK's bootstrap manifest (`robotics-apps`, docs, etc…). It does NOT fetch the upstream stack sources (PX4, Zephyr, etc…); those are fetched for the selected platform and stack when `west sdk build` is run.

Possible pitfall (detached HEAD)

4.9 Warnings and environment-specific notes

The warnings below only apply in specific situations (corporate networks, WSL2, tag checkouts). They are collected here so that the step-by-step flow above stays easy to follow — consult the relevant one only if it applies to the local setup.

Warning: hello-world on a corporate network

On a corporate network, `docker run hello-world` pulls the image from Docker Hub and will fail unless Docker can reach the registry through the configured proxy. Configure the Docker daemon proxy (see the Docker documentation) before running the validation step.

Warning: detached HEAD after a tag checkout

Doing a checkout on a tag will put the repository in a detached HEAD state. Run `git switch -c <branch>` to create a branch.

Warning: docker build proxy on a corporate network

Behind a corporate firewall, the proxies may also need to be specified for the build command, using the `--build-arg` option. Replace `<proxy>` and `<port>` with the local site's values (see Corporate network):

```
sudo docker build --build-arg UID=$(id -u) \
                  --build-arg GID=$(id -g) \
                  --build-arg http_proxy=http://<proxy>:<port> \
                  --build-arg https_proxy=http://<proxy>:<port> \
                  -t robotics-sdk -f robotics-sdk/Dockerfile .
```

Warning: WSL2 DNS resolution during docker build

When building on WSL2, the Docker container may lose DNS resolution for external hosts (e.g. `github.com`), causing the Zephyr SDK download to fail with `Temporary failure in name resolution`. Add `--network=host` to the build command so the container shares the WSL2 network stack directly (see Troubleshooting):

```
sudo docker build --network=host \
                  --build-arg UID=$(id -u) \
                  --build-arg GID=$(id -g) \
                  -t robotics-sdk -f robotics-sdk/Dockerfile .
```

Warning: “No space left on device” during docker build

A build that stops with `No space left on device` (often while unpacking an apt package such as `vim-runtime`) means the volume backing Docker’s storage (`/var/lib/docker` by default) filled up — the named package is not the cause. First reclaim space and re-run the build:

```
df -h                     # which volume (e.g. /var/lib/docker) is full?
docker system df          # space used by images / build cache
docker builder prune -f   # drop stale build cache
docker system prune -af   # remove unused images/containers/networks
```

If that volume is simply too small and another has room, move Docker’s data-root there. Stop Docker, copy the existing data across (so the current images are kept — skip the copy to start fresh), point Docker at the new path, then restart:

```
sudo systemctl stop docker
sudo rsync -aP /var/lib/docker/ /large/volume/docker/
echo '{ "data-root": "/large/volume/docker" }' | sudo tee /etc/docker/daemon.json
sudo systemctl start docker
docker info | grep "Docker Root Dir"   # verify the new path, then rm -rf the old one
```

On Docker Desktop / WSL2 / macOS / Windows, increase the VM’s virtual disk allocation instead (Docker Desktop → Settings → Resources).

Warning: docker run proxy on a corporate network

Behind a corporate firewall, the proxy-related environment variables may also need to be passed to the `run` command (substitute the local `<proxy>` / `<port>`). A full corporate-network invocation typically also uses `--network=host` (so a proxy on `127.0.0.1` is reachable), mounts the host CA bundle so TLS to external hosts is trusted, and mounts `~/.ssh` for git access:

```
sudo docker run -it -v $(pwd):/home/robotics \
                -v ~/.ssh:/home/robotics/.ssh:ro \
                -e http_proxy=http://<proxy>:<port> \
                -e https_proxy=http://<proxy>:<port> \
                -e ftp_proxy=http://<proxy>:<port> \
                --network=host \
                -v /etc/ssl/certs/ca-certificates.crt:/etc/ssl/certs/ca-certificates.crt:ro
\
                robotics-sdk
```

Warning: repeated username and password prompts inside the container

If `west init` / `west update` keep prompting for a username and password while cloning the `github.com/nxp-robotics` repositories over HTTPS, and an SSH key is authorized on the `nxp-robotics` GitHub organization, make `git` use SSH instead. Mount the SSH directory when starting the container:

```
sudo docker run -it -v ~/robotics-workdir:/home/robotics \
                -v ~/.ssh:/home/robotics/.ssh:ro \
                robotics-sdk
```

Then, inside the container, rewrite GitHub HTTPS URLs to SSH before running `west init` / `west update`:

```
git config --global url."git@github.com:".insteadOf "https://github.com/"
```

This needs outbound SSH (port 22) to `github.com`; if the network blocks it, use a GitHub personal access token over HTTPS instead (see Troubleshooting).

4.10 Troubleshooting / FAQ

Docker build fails downloading the Zephyr SDK on WSL2 (`Temporary failure in name resolution` / `unable to resolve host address 'github.com'`). On WSL2, Docker containers run inside an extra virtualization layer (Hyper-V), whose isolated virtual network can lose DNS resolution for external hosts. Add `--network=host` to the `docker build` command so the container shares the WSL2 network stack directly (see the WSL2 warning under Flow B: building images from source).

`docker build` fails with `No space left on device` (often while unpacking `vim-runtime` or another apt package). The named package is incidental — the volume backing Docker's storage (`/var/lib/docker`) ran out of room. Free space with `docker system prune -af` (and `docker builder prune -f`), and if the volume is too small, relocate Docker's `data-root` to a larger one — see the disk-space warning.

`docker run hello-world` hangs or fails on the corporate network. Docker cannot reach Docker Hub without a proxy. Configure the Docker daemon proxy per the Docker documentation, then retry.

TLS / certificate-verification errors during `docker build`. The network performs TLS inspection and the corporate CA is not trusted inside the image. Install the corporate CA certificate as described in Corporate network.

The proxy example commands do not work. The `http://<proxy>:<port>` values are placeholders — substitute the proxy host and port for the local site. If no proxy is needed at all, drop the `--build-arg http_proxy=...` / `-e http_proxy=...` arguments.

`git` / `west` keeps asking for a username and password inside the container. Over HTTPS `git` prompts for credentials on every fetch from `github.com/nxp-robotics`. If an SSH key is authorized on the `nxp-robotics`

organization, mount `~/.ssh` into the container (`-v ~/.ssh:/home/robotics/.ssh:ro`) and rewrite GitHub HTTPS URLs to SSH with `git config --global url."git@github.com:".insteadOf "https://github.com/"` before running `west init` / `west update` (see the warning under Flow B: building images from source). This needs outbound SSH (port 22); if it is blocked, use a GitHub personal access token as the password over HTTPS and cache it with `git config --global credential.helper 'cache --timeout=36000'` .

5 MR-NavQ95

The MR-NavQ95 is the i.MX 95 vehicle computer of the NXP Mobile Robotics family (Cortex-A55 compute and Cortex-M7 real-time). Bring up the board, then add connectivity, on-device AI, and robotics middleware.

- Overview
  - Functional block diagram
  - Main components
- Getting Started
  - Deploy and Run Quick Start
- User Manual
  - Introduction
  - Finding platform resources
  - Getting ready
  - Getting to know the hardware
  - Installing and configuring software tools
  - Configuring the hardware
  - Abbreviations
  - References
- Application Notes
  - Hardware Architecture and Reference Design
  - Camera
  - Ethernet
  - Cellular Modem
  - eIQ Neutron NPU
  - ARA-240 DNPU
  - Remote ID Transmitter
- Middleware
  - ROS2
- Flight Stack
  - PX4 Documentation Reference

6 Overview

The MR-NavQ95 is the vehicle-computer reference design of the NXP Mobile Robotics family. It provides a single, compact compute platform for drones and mobile robots that combines a high-performance application processor, a real-time control core, on-device AI acceleration, and automotive-grade connectivity. The board is intended as a starting point for customers building their own autonomous-vehicle products: it integrates the main NXP building blocks of a robotics compute node so that bring-up, perception, and robotics middleware can be evaluated and reproduced on validated hardware.

At its core is the NXP i.MX 95 applications processor, a heterogeneous multicore SoC built around six Arm Cortex-A55 application cores, a Cortex-M7 real-time core, and a Cortex-M33 System Manager, together with the on-chip NXP Neutron neural processing unit (NPU) for machine-learning inference. The Cortex-A55 cores run the Linux compute stack and robotics middleware, while the Cortex-M7 core handles hard real-time control, so a single device covers both the application and the real-time domains of a robot.

6.1 Functional block diagram

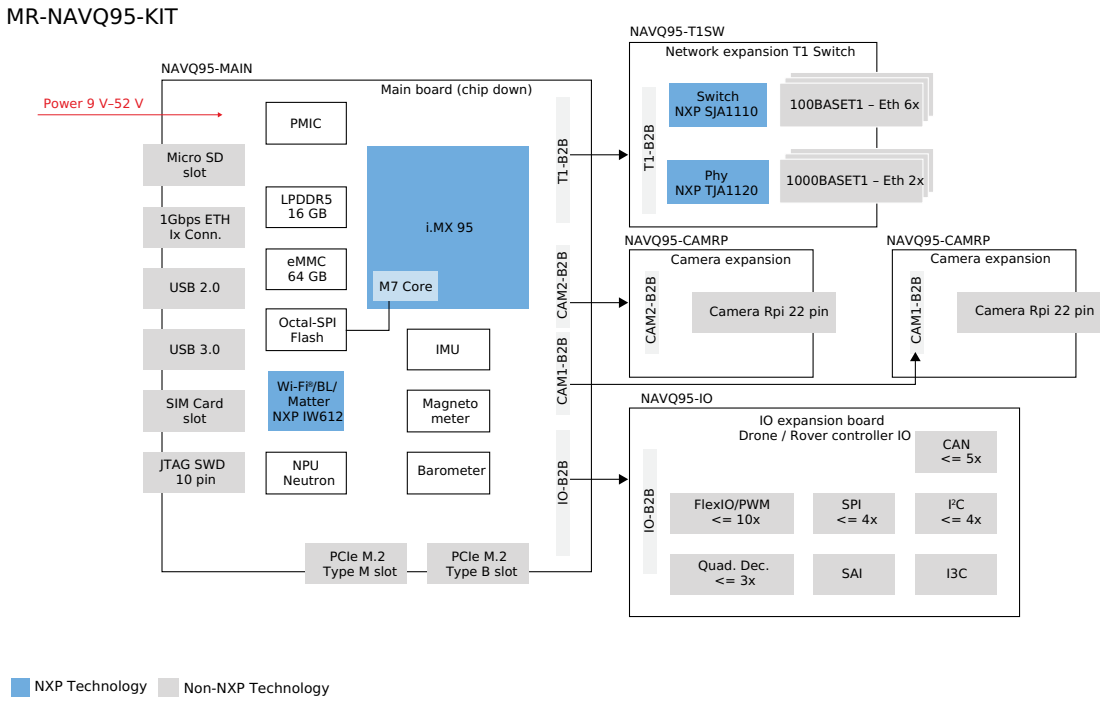


Figure 1. MR-NavQ95 functional block diagram

6.2 Main components

The reference design integrates the following main NXP components and interfaces:

- i.MX 95 applications processor (Cortex-A55 compute, Cortex-M7 real-time, Cortex-M33 System Manager, Neutron NPU)
- LPDDR memory, eMMC, octal-SPI flash, and a microSD slot
- NXP wireless connectivity for Wi-Fi, Bluetooth, and Matter
- NXP 802.15.4 radio for Thread and Zigbee

- On-board inertial, magnetometer, and barometric sensors
- Gigabit Ethernet, plus automotive 100BASE-T1 Ethernet through the NXP SJA1110 switch and TJA1120 / TJA1103 PHYs on the T1 expansion board
- Camera expansion board with MIPI CSI-2 inputs
- IO expansion board with CAN-FD, PWM, SPI, I2C, SAI, and quadrature-decoder interfaces for drone and rover control
- Two PCIe M.2 slots for storage or discrete NPU acceleration

NXP also provides the complete hardware Bill of Materials and design files, enabling customers to evaluate, reproduce, and customize the reference platform for their own products.

For overview information, documentation, software, tools, and ordering details, see the MR-NavQ95 product page on the NXP web site.

7 Getting Started

Deploy and run on the MR-NavQ95 i.MX 95 vehicle computer.

- Deploy and Run Quick Start

8 Deploy and Run Quick Start

The MR-NavQ95 is the i.MX 95 vehicle computer of the NXP Mobile Robotics family. It is built on the NXP i.MX 95 applications processor, a heterogeneous multicore SoC that combines six Arm Cortex-A55 application cores, a Cortex-M7 real-time core, and a Cortex-M33 System Manager, together with the on-chip NXP Neutron neural processing unit (NPU) for machine-learning inference. On the board this maps to two cooperating domains: a Cortex-A55 compute domain running Linux (Ubuntu) and a Cortex-M7 real-time domain running the RTOS/PX4 firmware. For a full board description and functional block diagram, see the MR-NavQ95 overview.

Unlike the single-MCU platforms, the MR-NavQ95 requires flashing two images with two different tools: the Cortex-A Linux `.wic` image with UUU, and the Cortex-M RTOS firmware with pyOCD. The Cortex-A image is flashed first.

8.1 Get the artifacts first

Every platform assumes the images are already available — either downloaded or built with the SDK:

```
# prebuilt (fastest)
west sdk download --platform mr-navq95      # lands in <west topdir>/bin/mr-navq95/

# or build from source
west sdk build    --platform mr-navq95  # lands in <west topdir>/{build,bin}/...
```

- Docker required. Downloading and building are done inside a Docker container built from the SDK's `Dockerfile` — see the NXP Robotics SDK README for how to set it up. All `west` commands are assumed to run inside that container.
- The SDK builds/downloads images but does not flash them — deployment is done with each platform's native tool.
- Where do the artifacts land? `<west topdir>` is the west workspace root — the directory containing `.west/`, created by `west init`. The `bin/` directory is created next to the `robotics-sdk/` clone, not inside it:

```
<west topdir>/
├── .west/
├── robotics-sdk/         <- the SDK clone (west init -l)
├── build/                <- west sdk build intermediates
└── bin/mr-navq95/       <- final artifacts land here
```

For MR-NavQ95 the platform is `mr-navq95`. Run `west sdk list` for the authoritative list.

```
west sdk download --platform mr-navq95                # firmware (raw + signed) +
linux .wic
# or build from source:
west sdk build    --platform mr-navq95                # PX4 (default)
west sdk build    --platform mr-navq95 --stack zephyr  # plain Zephyr
```

8.2 What to flash

MR-NavQ95 has two images that are flashed separately, in this order:

Domain	Image	Artifact (in `bin/mr-navq95/`)	Tool	Target
1. Cortex-A (Linux)	Ubuntu root filesystem	`linux/imx-image-robotics-imx95-navq.rootfs.wic.zst`	UUU	eMMC or SD card
2. Cortex-M (RTOS)	Signed M7 firmware	`realtime/nxp_imx95_default.mcuboot.bin`	pyOCD	NOR flash

> ℹ️ Use the signed `nxp_imx95_default.mcuboot.bin` image for the default TCM-based deployment. MCUboot verifies this image during startup. The `nxp_imx95_ddr.bin` variant is an alternative M7 firmware configuration that runs primarily from external DDR memory instead of TCM and is not packaged as an MCUboot-signed image.
>
> ⚠️ The Cortex-A Linux image must be flashed first; flash the Cortex-M firmware only after the board reaches the Linux login prompt.

8.3 Connect

- USB Type-C from USB1 (J13) — the Serial Download port used by UUU (Cortex-A flashing).
- USB Type-C from DEBUG USB (J10) — exposes two serial consoles (Cortex-A and Cortex-M), 115200 baud, 8N1.
- JTAG probe (NXP MCU-Link or J-Link) on the JTAG connector (J7) — used by pyOCD for Cortex-M flashing.
- 9–52 V DC power supply on the power input connector J9.

Set boot switch SW2 before powering the board (and before attaching any ARA M.2 module, which physically covers SW2). See MR-NavQ95 Board Overview for the full `SW2[1:4]` map and connector locations.

8.4 Flash the Cortex-A Linux image (UUU)

1. Download UUU (`uuu` / `uuu.exe`) from the mfgtools releases and place it next to the `.wic` image.
2. Set `SW2[1:4] = 1001` (Serial Downloader), connect USB1 (J13) + DEBUG USB (J10), apply power, and verify detection:

```
./uuu -lsusb        # expect an NXP i.MX95 device
```

3. Flash to eMMC (or SD):

```
./uuu -b emmc_all imx-image-robotics-imx95-navq.rootfs.wic.zst
# OR
./uuu -b sd_all   imx-image-robotics-imx95-navq.rootfs.wic.zst
```

   UUU prints `Done` / `Success 1` on completion.

4. Set the boot switches to the flashed media (eMMC = `SW2[1:4] 0101`, SD = `1101`), power-cycle, and watch both DEBUG-USB consoles at 115200 8N1. Default Linux login: user / user.

See Flashing the Cortex-A Linux Image for full details.

8.5 Flash the Cortex-M RTOS firmware (pyOCD)

1. Clone the NXP-Robotics pyOCD fork (the i.MX95 target is not in mainline pyOCD):

```
git clone https://github.com/NXP-Robotics/pyOCD -b pr-imx95 && cd pyOCD
```

2. To install pyOCD:

```
# Option 1: install using python virtual environments. If you choose
# this option, please make sure you always activate the virtual environment
# before trying to use pyOCD!
python3 -m venv .venv
source ./.venv/bin/activate
python3 -m pip install .

# Option 2: install using pipx
sudo apt-get install -y pipx
# you may need to open a new terminal session for this to take effect!
pipx ensurepath
pipx install .
```

3. Keep the boot switches on the media you flashed, connect the JTAG probe (J7) + DEBUG USB (J10), apply power, and **wait for the Linux login prompt**.

4. Flash the signed M7 image to NOR:

```
pyocd flash -t mimx95_cm7_mx25um --base-address 0x28020000 \
    nxp_imx95_default.mcuboot.bin -f 10m
```

> NOR writes are not fully stable in the current release — retry until pyOCD reports 0 pages programmed (confirms flash matches the binary). The clock can be dropped to `4m` after repeated failures.

5. Reboot; the M7 RTOS shell is reachable from Linux on `/dev/ttynsh`:

```
picocom /dev/ttynsh
```

See Flashing the Cortex-M RTOS Firmware for full details, including the CM33-AP fallback.

9 User Manual

Bring up the MR-NAVQ95-KIT: review the board hardware, get or build the Cortex-A Linux image and the Cortex-M RTOS firmware, flash them, and manage device-tree overlays.

- Introduction
- Finding platform resources
- Getting ready
- Getting to know the hardware
- Installing and configuring software tools
- Configuring the hardware
- Abbreviations
- References

10 Introduction

The MR-NavQ95 is the vehicle-computer reference design of the NXP Mobile Robotics family. It combines a high-performance application processor, a real-time control core, on-device AI acceleration, and automotive-grade connectivity on a single compact board, providing the compute and perception functions of a drone or mobile robot. The board is intended as a validated starting point that customers can evaluate, reproduce, and adapt for their own vehicles.

At its core is the NXP i.MX 95 applications processor, a heterogeneous multicore SoC built around Arm Cortex-A55 application cores, a Cortex-M7 real-time core, and a Cortex-M33 System Manager, together with the on-chip NXP Neutron neural processing unit (NPU). The Cortex-A55 cores run the Linux compute stack and robotics middleware, while the Cortex-M7 core handles hard real-time control.

This user manual describes how to obtain or build the Cortex-A Linux image and the Cortex-M RTOS firmware, how to flash them, and how to manage device-tree overlays. The connector pinouts and on-board devices are documented in the Hardware Overview; the hardware architecture is described in the MR-NavQ95 Hardware Architecture and Reference Design.

10.1 Radio Equipment Directive compliance information

The radiating device on the MR-NAVQ95-KIT board is the NXP LBES5PL2EL, a 2.4/5 GHz dual-band 1x1 Wi-Fi 6, Bluetooth/Bluetooth Low Energy, and 802.15.4 tri-radio solution. Table 1 provides the following information per Article 10.8 of the Radio Equipment Directive 2014/53/EU:

- Frequency bands in which the equipment operates
- The maximum RF power transmitted

Table 1. MR-NAVQ95-KIT specifications

Part number	RF technology	Frequency ranges (EU)	Maximum transmitted power
MR-NAVQ95-KIT	Wi-Fi (802.11b/g/n/ax)	2400 MHz - 2483.5 MHz	+21 dBm
MR-NAVQ95-KIT	Wi-Fi (802.11a/n/ac/ax)	5150 MHz - 5350 MHz, 5470 MHz - 5725 MHz	+21 dBm
MR-NAVQ95-KIT	Bluetooth / Bluetooth Low Energy	2400 MHz - 2483.5 MHz	+19 dBm
MR-NAVQ95-KIT	802.15.4 (Thread / Zigbee)	2400 MHz - 2483.5 MHz	+21 dBm

EUROPEAN DECLARATION OF CONFORMITY

(Simplified DoC per Article 10.9 of the Radio Equipment Directive 2014/53/EU). This apparatus, namely the MR-NAVQ95-KIT, conforms to the Radio Equipment Directive 2014/53/EU. The full EU Declaration of Conformity for this apparatus is available at European Union Declaration of Conformity for MR-NAVQ95-KIT.

10.2 NXP Robotics SDK and upstream open-source software

NXP embraces an open-source development model and actively participates in the communities behind the software used by this product. The NXP Robotics SDK provides a curated and validated set of software components, tools, and documentation that have been tested together on this hardware platform.

NXP regularly contributes improvements and fixes back to upstream repositories. Because contributions are made on a scheduled basis, the latest upstream code and the version included in the NXP Robotics SDK may not always be identical. The SDK release is recommended for users seeking a tested and supported configuration, while advanced users may choose to work directly with the upstream project and its latest features.

11 Finding platform resources

The information page for the MR-NavQ95 vehicle-computer reference design is on the NXP web site at https://www.nxp.com/design/design-center/development-boards-and-designs/ai-powered-autonomy-compute-flight-controller:MR-NAVQ95. The information page provides overview information, documentation, software and tools, and a Getting Started tab with quick-reference information, including the downloadable assets referenced in this document.

The board design files and the latest hardware reference are published in the public NXP Robotics repository at https://github.com/NXP-Robotics/MR-NAVQ95. The software, images, and build tooling are distributed through the NXP Robotics SDK; follow the NXP Robotics SDK README to set up the SDK and obtain the assets.

12 Getting ready

Working with the MR-NavQ95 requires the board, a host PC, a debug probe, and the images produced or downloaded through the NXP Robotics SDK.

12.1 Kit contents

- MR-NavQ95 vehicle-computer board

12.2 Additional hardware

- A JTAG debug probe (NXP MCU-Link or J-Link) for programming the Cortex-M RTOS firmware
- A USB Type-C cable for the serial-download port (USB1 / J13)
- A USB Type-C cable for the debug console (DEBUG USB / J10)
- A 9 V to 52 V DC power supply (3S to 12S battery) for the power input connector (J9)

12.3 Assumptions

Familiarity with embedded Linux, U-Boot, and flashing tools is helpful but not required.

12.4 Minimum system requirements

- An Ubuntu PC is assumed for all host-side steps.
- For building the Linux image from source: Ubuntu 22.04 or later, at least 300 GB of free disk space, and 16 GB of RAM (minimum recommended).

12.5 Software

The images and build tooling are provided through the NXP Robotics SDK, which runs inside a Docker container. The SDK produces the Cortex-A Linux image and the Cortex-M RTOS firmware for the board.

13 Getting to know the hardware

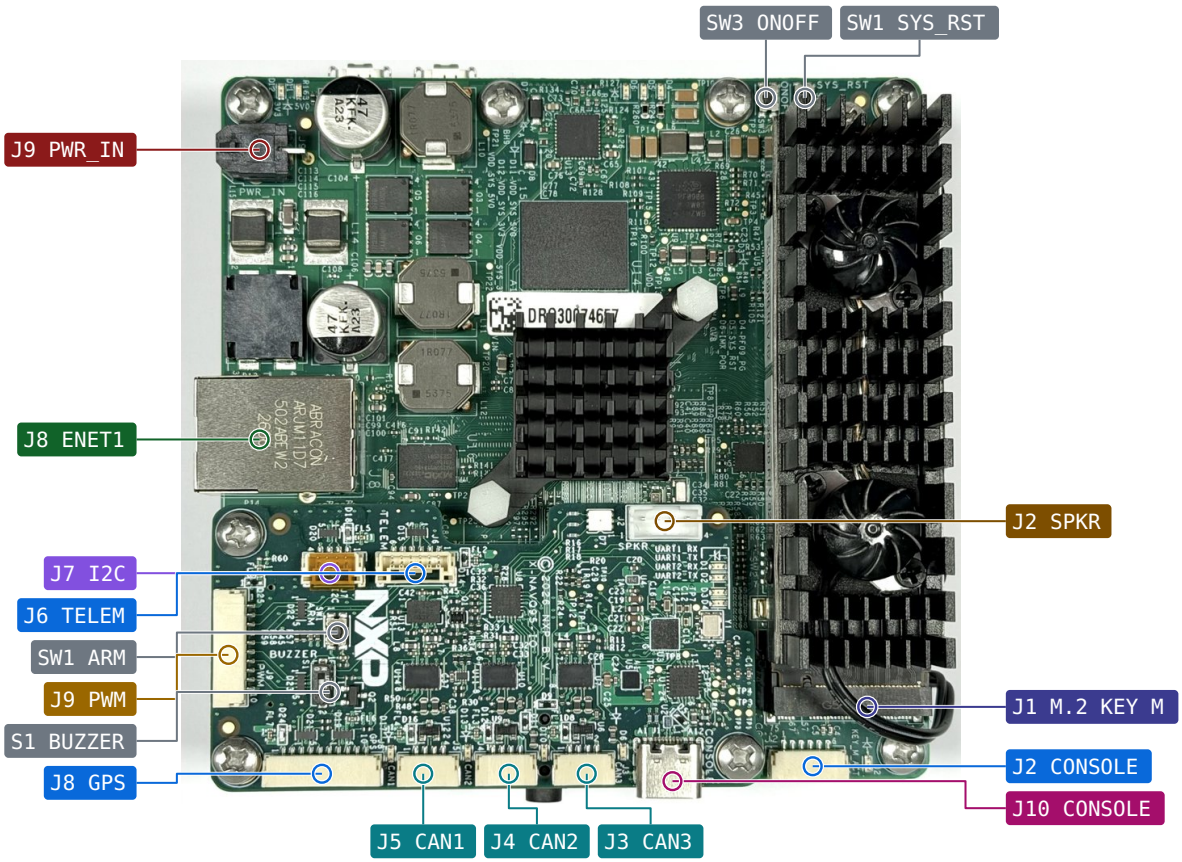
Connectors and pinouts of the NXP MR-NAVQ95 vehicle computer (main board) and its expansion boards. All data comes from the public schematics in the MR-NAVQ95 repository: main board, IO board, camera board and T1 switch board. These are the Rev. A1 production boards.

> Note: The boot-mode switch (SW2) is described under Boot Switch (SW2) below.

Each card shows three things per pin: the function, the net name in the schematic and the i.MX95 pad the signal ends up on. Parts in between, such as transceivers, level shifters and I/O expanders, are shown too. Pin 1 is the square pad.

> Tip: The pinout cards are interactive: hover over a pin to highlight it, and click a pin to copy its net/pad name to the clipboard.

13.1 Board overview

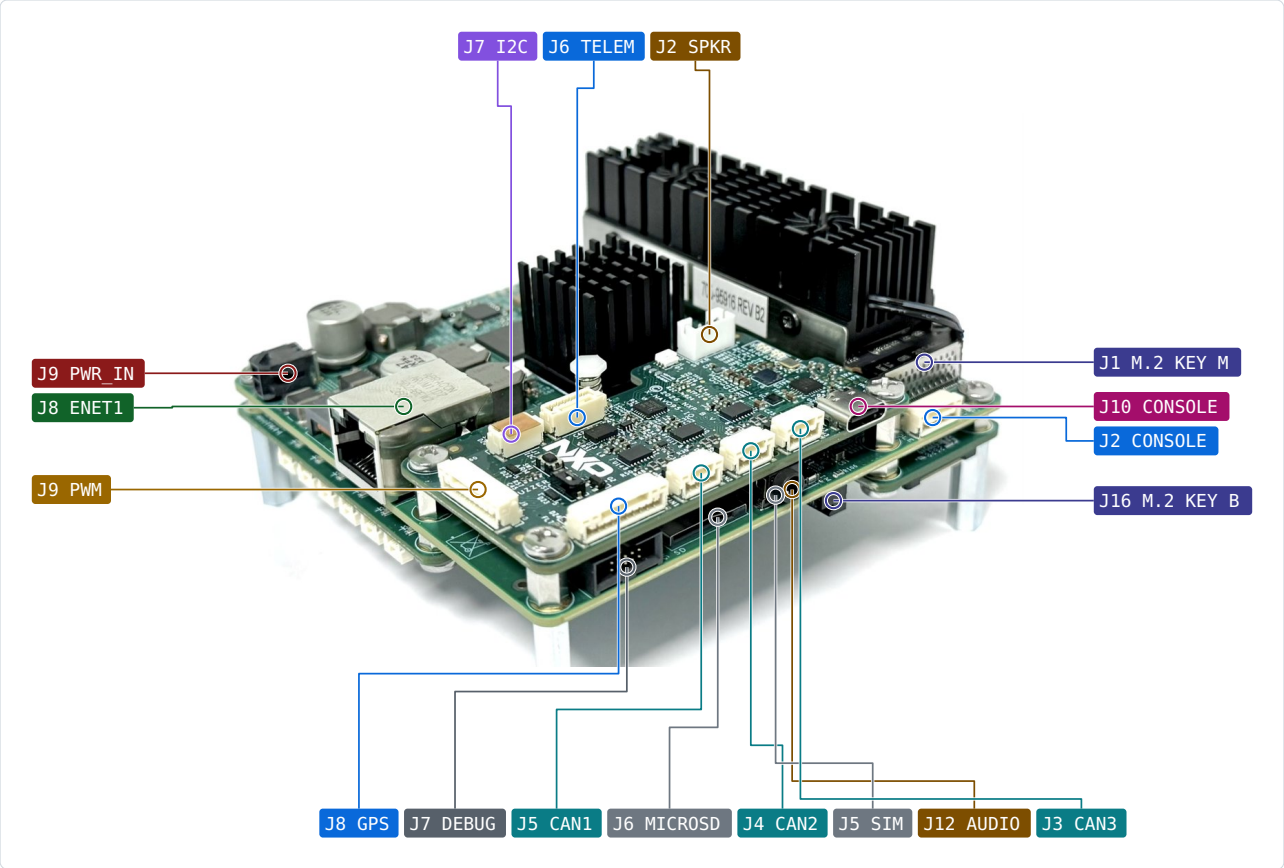


Not visible here:

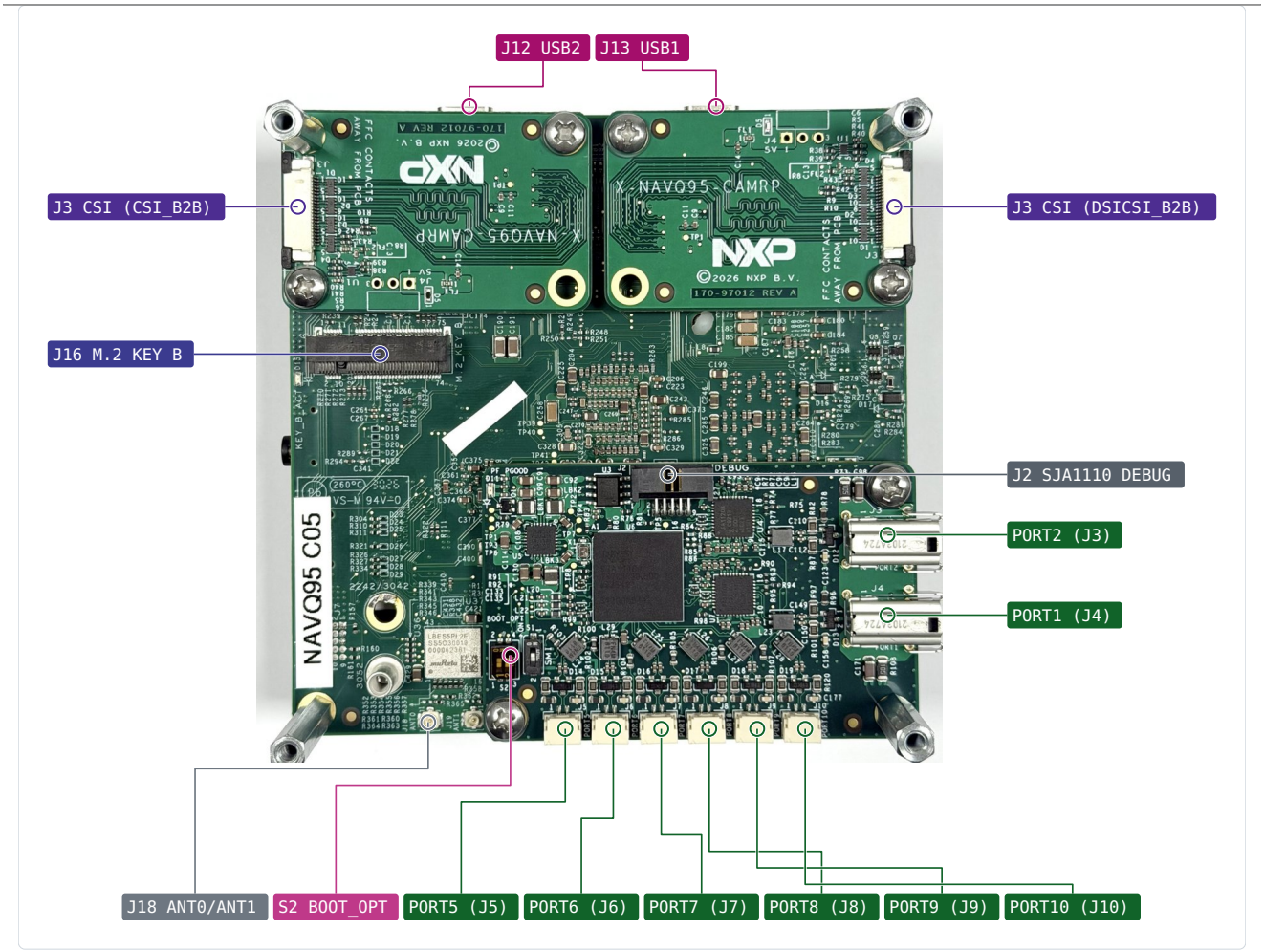
- J13 USB1 (bottom view)

- J12 USB2 (bottom view)
- J16 M.2 KEY B (bottom and front view)
- J7 DEBUG (front view)
- J6 MICROSD (front view)
- J11 RTC_BAT (not visible in the photos)
- SW2 BTMODE (top side, under the M.2 Key M module)

Front view. The JST-GH connectors of the IO board are on top. Below them, on the edge of the main board, are the debug header, the microSD slot, the SIM holder and the M.2 Key B slot:



Bottom side. In the picture above the T1 Ethernet switch board and two camera boards are fitted. The left camera board is on the CSI_B2B port, the right one on the DSICSI_B2B port; only the DSICSI_B2B port can also drive an RPi DSI display. PORT1 to PORT10 are the switch ports:



13.2 Boot Switch (SW2)

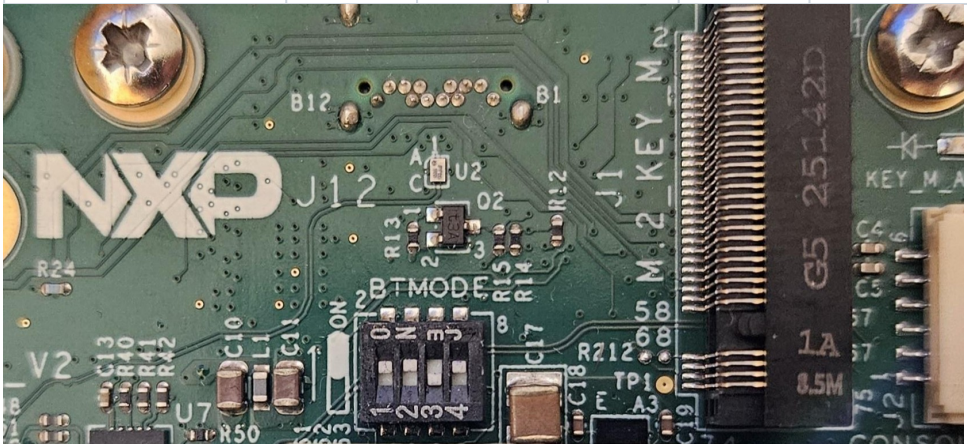
The boot mode is selected using DIP switch SW2 (BTMODE), a 4-position DIP switch on the top side of the board, located near the M.2_KEY_M PCIe connector (J1). Set the switches before powering on the board.

> Important: When an ARA module is connected to the M.2_KEY_M PCIe connector (J1), it will physically cover SW2. Always set the desired boot mode before attaching the ARA module.

The BOOT_MODE[3:0] bits map to switches SW2[4:1]:

BOOT_MODE[3:0]	SW2[1]	SW2[2]	SW2[3]	SW2[4]	Boot Device
`1001`	ON	OFF	OFF	ON	Serial Downloader via USB1 (J13)
`1010`	OFF	ON	OFF	ON	eMMC
`1011`	ON	ON	OFF	ON	SD Card

BOOT_MODE[3:0]	SW2[1]	SW2[2]	SW2[3]	SW2[4]	Boot Device
`1100`	OFF	OFF	ON	ON	Octal Flash (NOR)



SW2 boot mode switch location on the MR-NavQ95 main board, near the J1 M.2_KEY_M PCIe connector.

13.3 Connector index

Ref	Function	Connector	Details
Main J1 M.2 KEY M	M.2 Key M slot, PCIe x1 (NPU or NVMe SSD)	M.2 Socket 3 Key M, 2242 / 2280 standoffs, 3.3 V up to 3.5 A	
Main J2 CONSOLE	UART1 (Linux console) and UART2 (M7 console) header, DS-009 TELEM layout	JST-GH 1x6 (SM06B-GHS-TB)	
Main J3 B2B	Board-to-board expansion headers	Board-to-board headers	J3/J4 B2B_IO1/IO2 (2x25) for the IO board, J14/J15 B2B_CSI/DSICSI (2x20) for the camera boards, J17 B2B_ETH (2x25) for the T1 boards. Pin lists are on schematic page 25.
Main J5 SIM	SIM card holder for the Key B modem slot	SIM card holder, 6 contacts + card-detect switch, bottom side	
Main J6 MICROSD	microSD card slot (USDHC2)	microSD push-push, bottom side	USDHC2. The board can boot from it when the boot-mode switch selects SD.

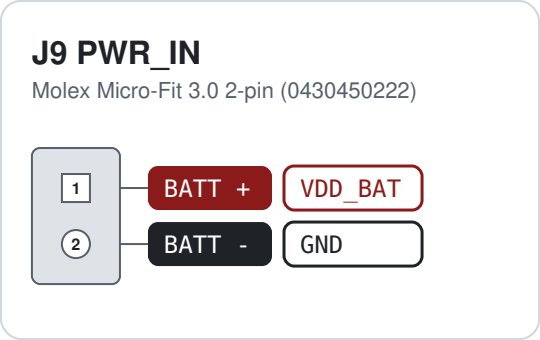
Ref	Function	Connector	Details
Main J7 DEBUG	SWD / JTAG debug header (Arm Cortex 10-pin)	2x5 1.27 mm (Samtec SHF-105-01-L-D-RA)	
Main J8 ENET1	Gigabit Ethernet with PTP	RJ45 with magnetics (Abracon ARJM11D7)	RTL8211 PHY on i.MX95 ENET1, supports IEEE 1588 PTP.
Main J9 PWR_IN	Power input 9-52 V (3S-12S battery)	Molex Micro-Fit 3.0 2-pin (0430450222)	
Main J11 RTC_BAT	RTC backup battery (rechargeable ML2020, RPi 5 compatible)	JST-SH 1.0 mm 2-pin (SM02B-SRSS-TB), top entry	
Main J12 USB2	USB 2.0 Type-C host port	USB Type-C receptacle, bottom side	Host port. 5 V VBUS through an NX20P0477 load switch with over-current protection.
Main J13 USB1	USB 3.0 Type-C, serial download / device port	USB Type-C receptacle, bottom side	Serial-download port of the i.MX95 boot ROM and the port the NavQ95 flasher uses. USB 3.0 and USB 2.0 are both connected.
Main J16 M.2 KEY B	M.2 Key B slot, PCIe x1 with SIM (cellular modem)	M.2 Socket 2 Key B / B+M, 2242 / 3042 / 3052 standoffs, bottom side, 3.3 V up to 5 A	
Main J18 ANT0/ANT1	Wi-Fi 6 / Bluetooth antennas (LBES5PL2EL), ANT0 = J18, ANT1 = J19	2x U.FL	ANT0 = J18, ANT1 = J19. LBES5PL2EL Wi-Fi 6, Bluetooth and 802.15.4.
IO J10 CONSOLE	USB-C console: CP2105 bridge, UART1 = Linux, UART2 = M7	USB Type-C receptacle (USB 2.0 device)	
IO J2 SPKR	Stereo speaker output (WM8962B class-D, 2x 1 W)	JST-PH 2.0 mm 1x4	
IO J3 CAN3	CAN FD 3 (TJA1463)	JST-GH 1x4, side entry	

Ref	Function	Connector	Details
IO J4 CAN2	CAN FD 2 (TJA1463)	JST-GH 1x4, side entry	
IO J5 CAN1	CAN FD 1 (TJA1463)	JST-GH 1x4, side entry	
IO J6 TELEM	Telemetry UART7 with flow control	JST-GH 1x6, top entry	
IO J7 I2C	I2C6 peripheral connector	JST-GH 1x4, top entry	
IO J8 GPS	GPS: UART5, I2C6, safety switch, LED, buzzer	JST-GH 1x10, side entry	
IO J9 PWM	8x PWM / timer outputs (TPM3 to TPM6), also usable as FlexIO1 pins	JST-GH 1x10, side entry	
IO J12 AUDIO	3.5 mm headset jack, CTIA wiring (WM8962B headphone out + microphone in)	3.5 mm 4-pole TRRS jack with insertion switch, board underside	
CAM J3 CSI	MIPI CSI-2 camera, 4 data lanes, RPi camera compatible	22-pin 0.5 mm FPC/FFC, contacts away from the PCB	
CAM J4 5V	5 V header for a display (not fitted)	1x3 2.54 mm right-angle header, not fitted	
T1S J2 DEBUG	Debug header of the SJA1110 (Arm Cortex-M7 inside the switch)	2x5 1.27 mm	Not the i.MX95 debug port; that is J7 on the main board.
T1S J3 PORT2	1000BASE-T1 port 2	Single-pair automotive connector	TJA1120 PHY on SJA1110 SGMII port 2.
T1S J4 PORT1	1000BASE-T1 port 1	Single-pair automotive connector	TJA1120 PHY on SJA1110 SGMII port 1.

Ref	Function	Connector	Details
T1S J5 PORT5	100BASE-T1 port 5	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 5.
T1S J6 PORT6	100BASE-T1 port 6	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 6.
T1S J7 PORT7	100BASE-T1 port 7	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 7.
T1S J8 PORT8	100BASE-T1 port 8	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 8.
T1S J9 PORT9	100BASE-T1 port 9	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 9.
T1S J10 PORT10	100BASE-T1 port 10	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 10.

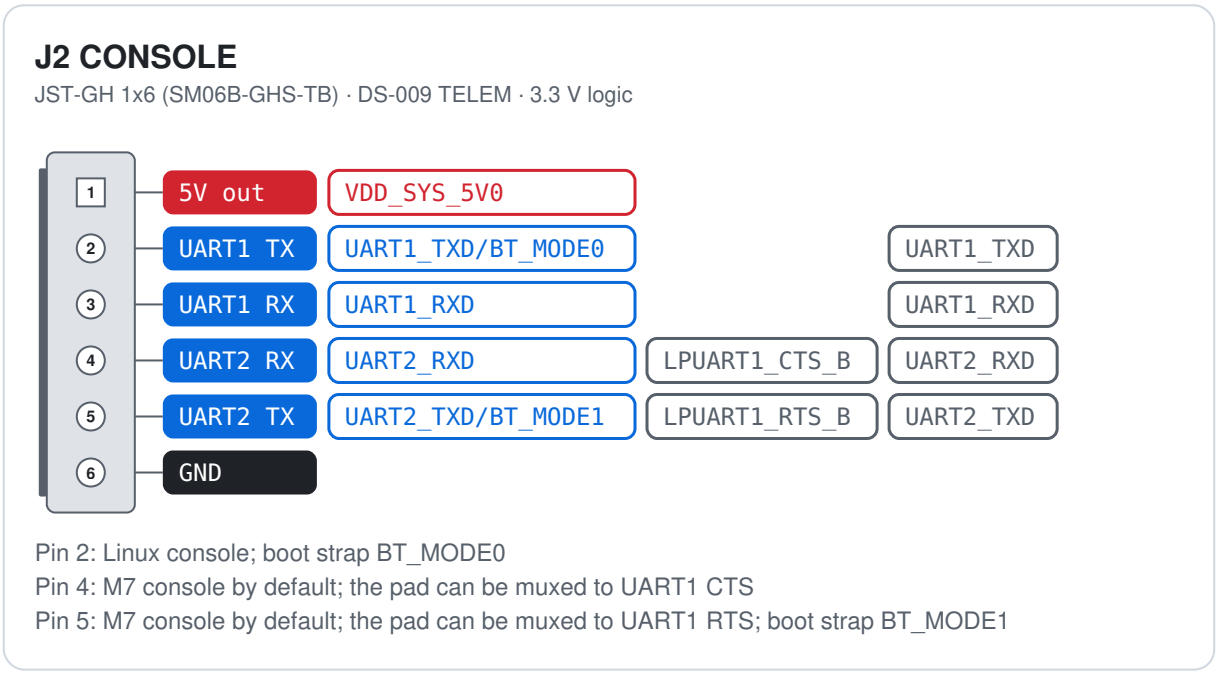
13.4 Main board

Main J9 PWR_IN



Pin 1 (BATT +) is the lower pin on the board. Input 9 V to 52 V, never more than 60 V. The board can draw 60 W between 9 V and 20 V, and 125 W between 20 V and 52 V. The input voltage is measured on ADC_IN0 (VIN / 64).

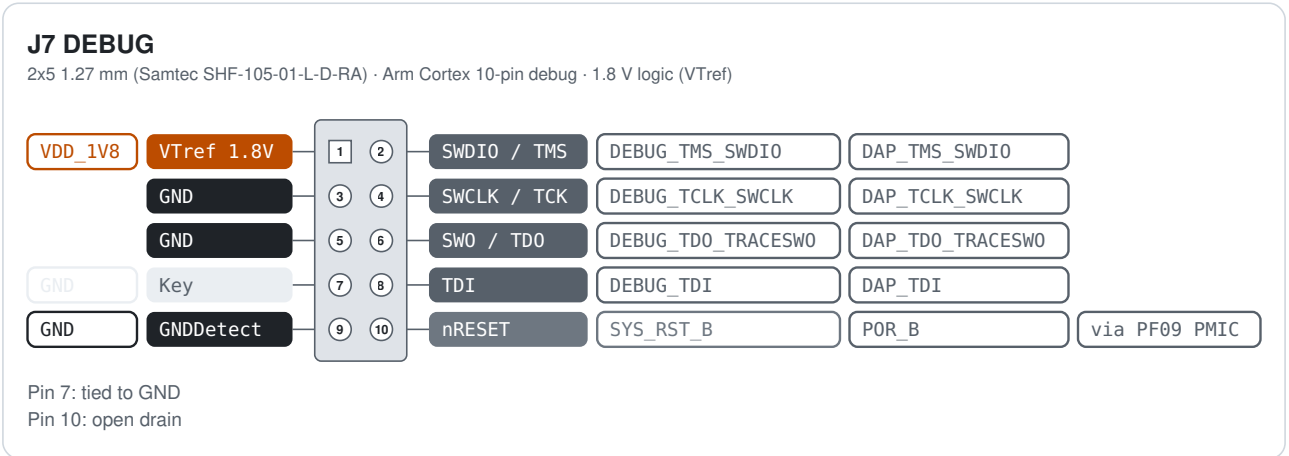
Main J2 CONSOLE



Pins 2 to 5 follow the DS-009 TELEM layout for UART1: 2 = TX, 3 = RX, 4 = CTS, 5 = RTS. By default pin 4 carries UART2 RX and pin 5 UART2 TX (the M7 console). The same pads can be muxed to UART1 CTS and RTS instead, which gives UART1 full DS-009 flow control on this header. UART1 TX and UART2 TX are also the boot-mode straps BT_MODE0 and BT_MODE1. The same two UARTs are also on the USB console port J10 of the IO board, see the figure there.

> Note: UART1 TX and UART2 TX are boot straps. Keep them idle while the board resets, or it may boot from the wrong source.

Main J7 DEBUG



Standard Arm Cortex 10-pin layout (0.05 inch). Works with J-Link, PyOCD and similar probes. nRESET goes through a diode to SYS_RST_B, so the probe can only pull it low.

> Note: VTref is 1.8 V. Use a debug probe that supports 1.8 V targets.

Main J1 M.2 KEY M



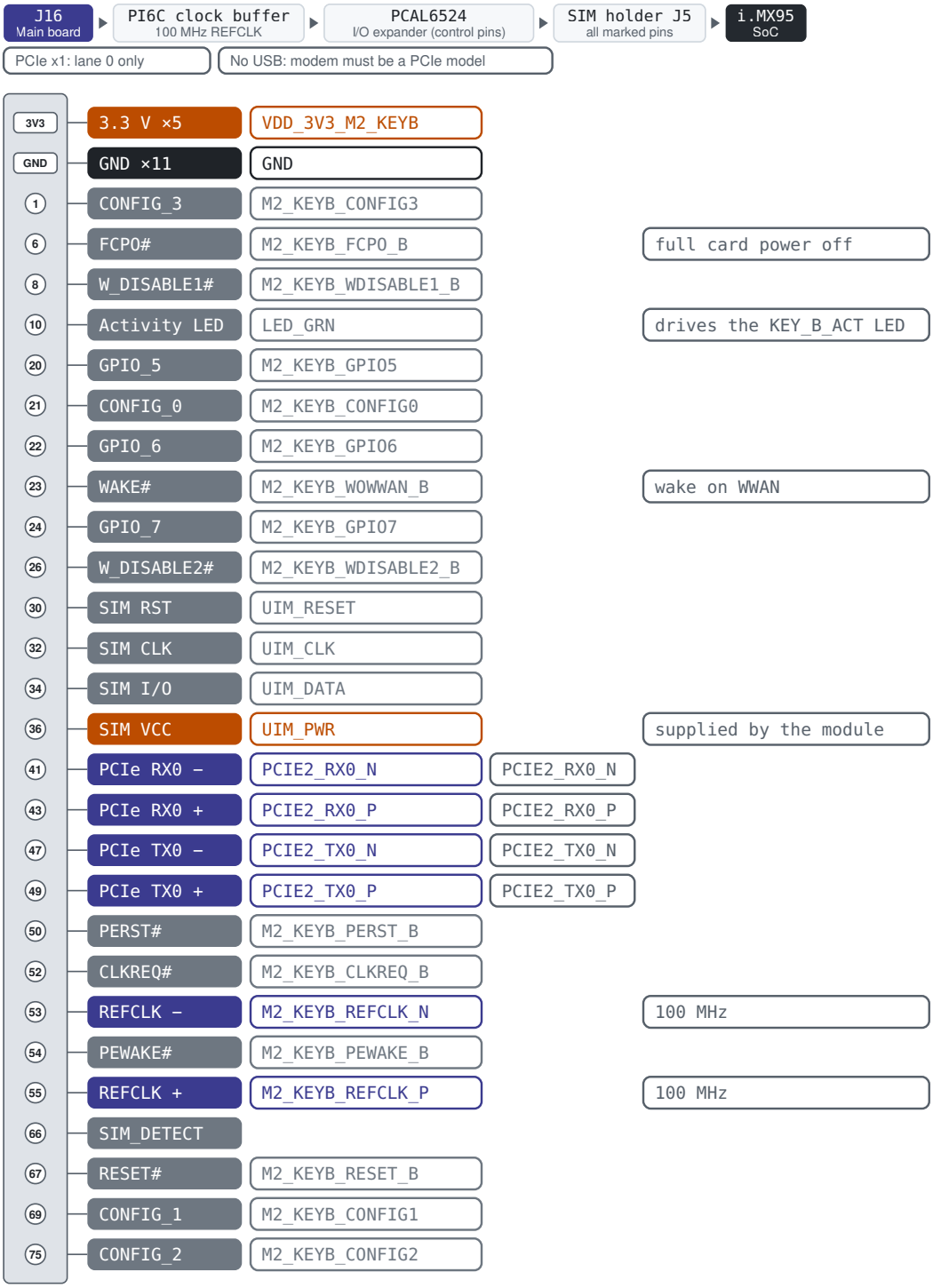
Only the pins below are connected. All other pins of the 75-pin socket are not used: PCIe lanes 1 to 3 (odd pins 5 to 37), SATA, DEVSLP, SUSCLK (pin 68, link resistor not fitted), MFG1/2 and the even pins 6 to 36. The 3.3 V comes through an NX20P5090 load switch that the I/O expander can switch off: 3.5 A continuous, 7 A peaks of 100 µs. PEDET has a pull-up on the always-on 3.3 V rail, so a card is detected even when the slot is switched off.

> Note: PCIe x1 only, no SATA. An NVMe SSD or NPU module works; a SATA M.2 drive does not.

Main J16 M.2 KEY B

J16 M.2 KEY B

M.2 Socket 2 Key B / B+M, 2242 / 3042 / 3052 standoffs, bottom side, 3.3 V up to 5 A · GPIO, RESET# and SMBus 1.8 V logic



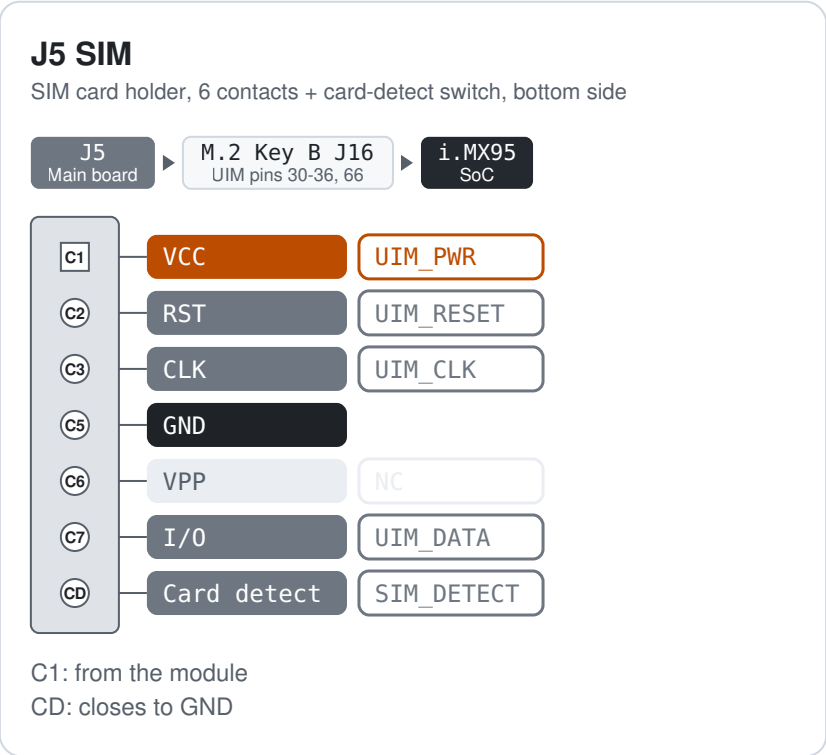
3V3: pins 2, 4, 70, 72, 74
GND: pins 3, 5, 11, 27, 33, 39, 45, 51, 57, 71, 73

Made for a PCIe cellular modem. The USB 2.0 pins (7, 9) and the USB 3.0 / lane 1 pins (29 to 37) are not connected. SMBus (40, 42, 44), SUSCLK (68) and the Bluetooth coexistence lines COEX1/2 (64, 62, to the LBES5PL2EL) are routed, but their link resistors are not fitted. ANTCTL0 to 3, COEX3, DPR, DEVSLP, GPIO_3/4/8

and MFG1/2 are not connected. The UIM pins go to the SIM holder J5. The 3.3 V comes through an NX20P5090 load switch that the I/O expander can switch off: 5 A continuous, 6 A peaks of 100 µs.

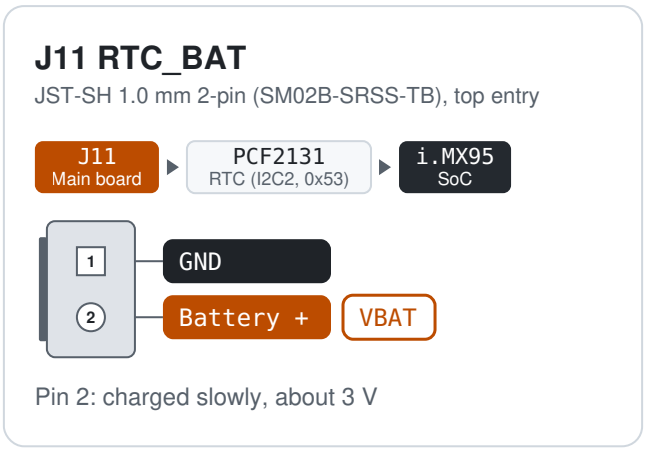
> Important: See the Using a Cellular Modem on MR-NAVQ95 application note for selection guidance and a bring-up example.

Main J5 SIM



Connected directly to the UIM pins of the Key B slot. The modem supplies the SIM voltage on UIM_PWR. The card-detect switch pulls SIM_DETECT low when a card is inserted. All contacts have ESD protection.

Main J11 RTC_BAT



Keeps the PCF2131 real-time clock running while the board has no power. The battery is charged slowly from the 3.3 V rail. This suits a rechargeable ML2020 cell such as the RPi RTC Battery.

Warning: ONLY use a rechargeable ML2020 cell only. A CR2032 or other non-rechargeable cell must not be fitted as the board supplies a charging current.

Main other connectors

Ref	Function	Connector	Notes
J13 USB1	USB 3.0 Type-C, serial download / device port	USB Type-C receptacle, bottom side	Serial-download port of the i.MX95 boot ROM and the port the NavQ95 flasher uses. USB 3.0 and USB 2.0 are both connected.
J12 USB2	USB 2.0 Type-C host port	USB Type-C receptacle, bottom side	Host port. 5 V VBUS through an NX20P0477 load switch with over-current protection.
J8 ENET1	Gigabit Ethernet with PTP	RJ45 with magnetics (Abracon ARJM11D7)	RTL8211 PHY on i.MX95 ENET1, supports IEEE 1588 PTP.
J6 MICROSD	microSD card slot (USDHC2)	microSD push-push, bottom side	USDHC2. The board can boot from it when the boot-mode switch selects SD.
J18 ANT0/ ANT1	Wi-Fi 6 / Bluetooth antennas (LBES5PL2EL), ANT0 = J18, ANT1 = J19	2x U.FL	ANT0 = J18, ANT1 = J19. LBES5PL2EL Wi-Fi 6, Bluetooth and 802.15.4.
J3 B2B	Board-to-board expansion headers	Board-to-board headers	J3/J4 B2B_IO1/IO2 (2x25) for the IO board, J14/ J15 B2B_CSI/DSICSI (2x20) for the camera boards, J17 B2B_ETH (2x25) for the T1 boards. Pin lists are on schematic page 25.

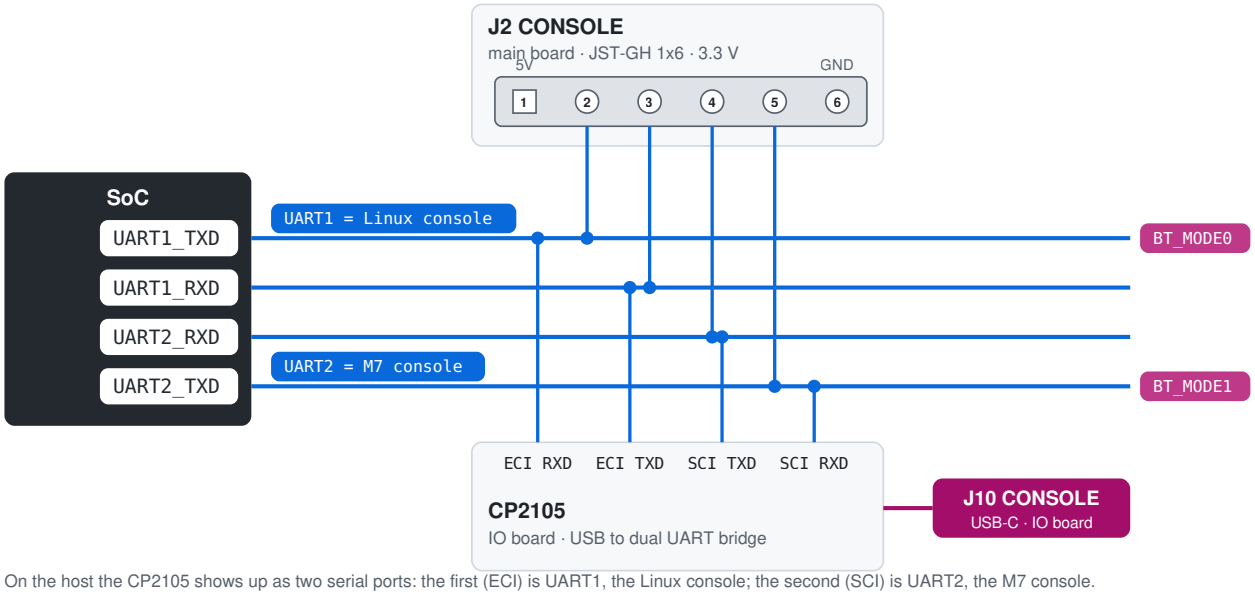
13.5 IO expansion board

IO J10 CONSOLE

A chip is used to present two serial ports on one USB-C connect at J10. On a Linux host they appear as ttyUSB0 (UART1, Linux console) and ttyUSB1 (UART2, M7 console). RX and TX LEDs for both UARTs sit next to the connector. The same two UART signals also go to the J2 header on the main board, see the figure below.

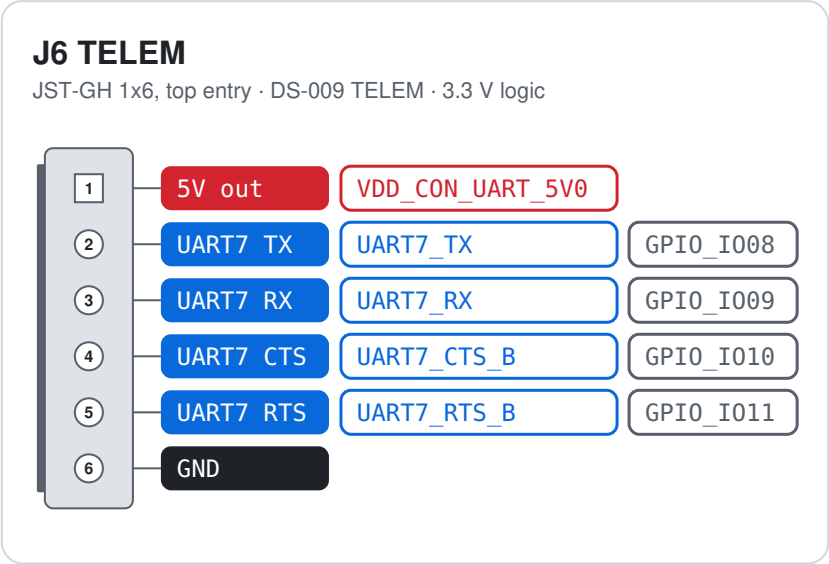
Note: Port 1 (ECI) is the Linux console, port 2 (SCI) is the M7 console. Do not let the host drive the TX lines while the board resets.

UART1 (Linux) and UART2 (M7): J2 and the USB bridge share the same SoC pins



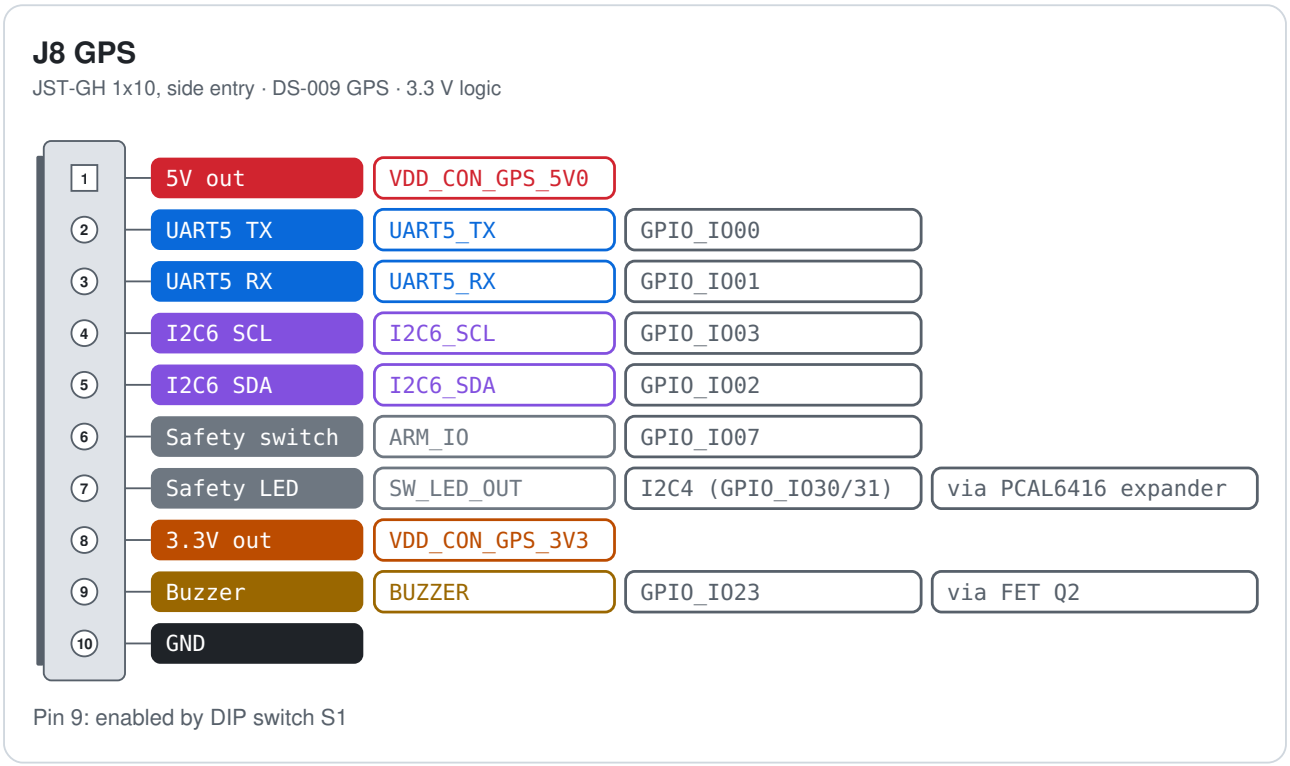
On the host the CP2105 shows up as two serial ports: the first (ECI) is UART1, the Linux console; the second (SCI) is UART2, the M7 console. Both TXD pins are boot straps: keep them idle during reset.

IO J6 TELEM



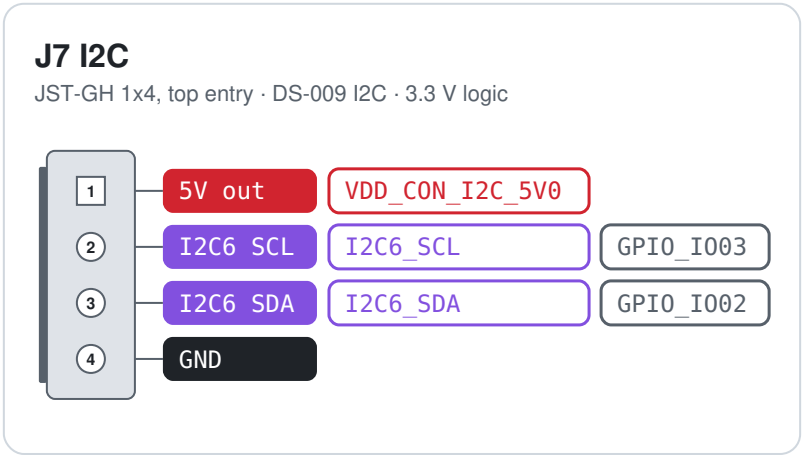
The 5 V output is filtered (VDD_CON_UART_5V0) and has ESD protection.

IO J8 GPS



Standard DS-009 GPS and compass connector. The safety switch input (ARM_IO) is also wired to the ARM button SW1 on the board, and the LED output drives the LED in the switch. The buzzer pin is switched by an N-FET (AO3400A) driven from TPM6_CH1. A buzzer on the board is connected in parallel. DIP switch S1 sits between TPM6_CH1 and the FET: with S1 OFF both buzzers are disconnected and TPM6_CH1 is free for PWM8 on the PWM header.

IO J7 I2C

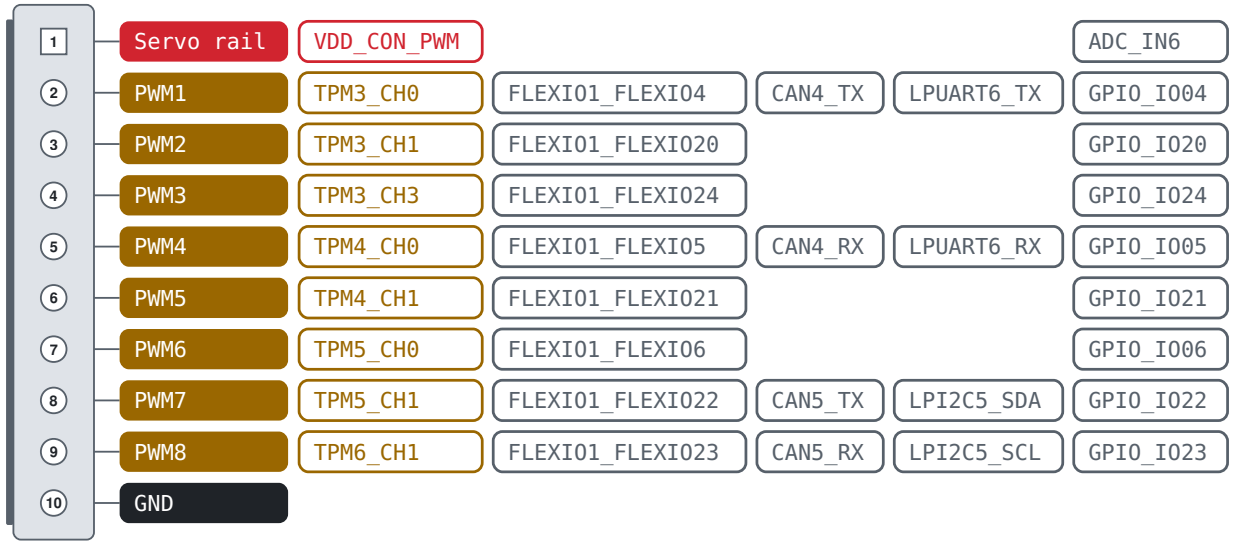


Same I2C6 bus as the GPS connector. The board has 2.2 kΩ pull-ups to 3.3 V.

IO J9 PWM

J9 PWM

JST-GH 1x10, side entry · 3.3 V logic outputs



Pin 1: measured on ADC_IN6
Pin 9: shared with the buzzer while S1 is ON

Pin 1 (VDD_CON_PWM) is the servo power rail. It is linked to the board 5 V through R60 (0 Ω, not fitted) and measured on ADC_IN6 (VDD_CON_PWM / 8, up to 12 V). The outputs are 3.3 V logic with ESD protection. PWM8 (pin 9) also drives the buzzer. Every output pad can also be muxed to FlexIO1, for example for DShot or other custom timing protocols. Pins 2 and 5 can also be CAN4 or LPUART6 (TX and RX), pins 8 and 9 CAN5 or LPI2C5 (no pull-ups on the board). CAN4 and CAN5 need an external transceiver.

> Warning: Do not put more than 5 V on the servo rail (pin 1) while R60 is fitted.

> Tip: Need all eight PWM outputs? Switch DIP switch S1 (BUZZER) off. It frees PWM8 from the buzzer.

IO J5 CAN1

J5 CAN1

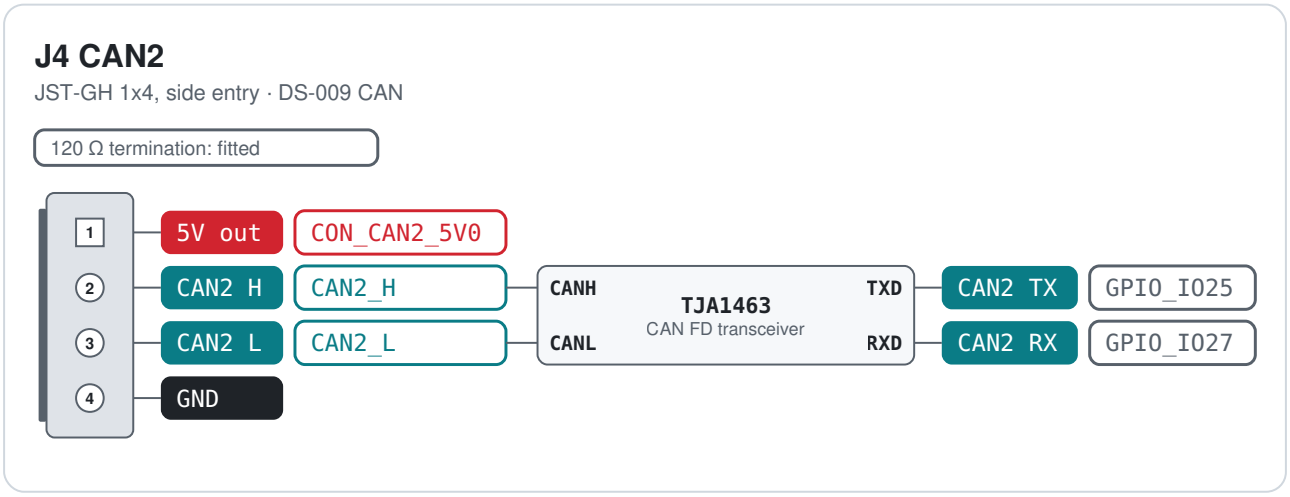
JST-GH 1x4, side entry · DS-009 CAN

120 Ω termination: fitted



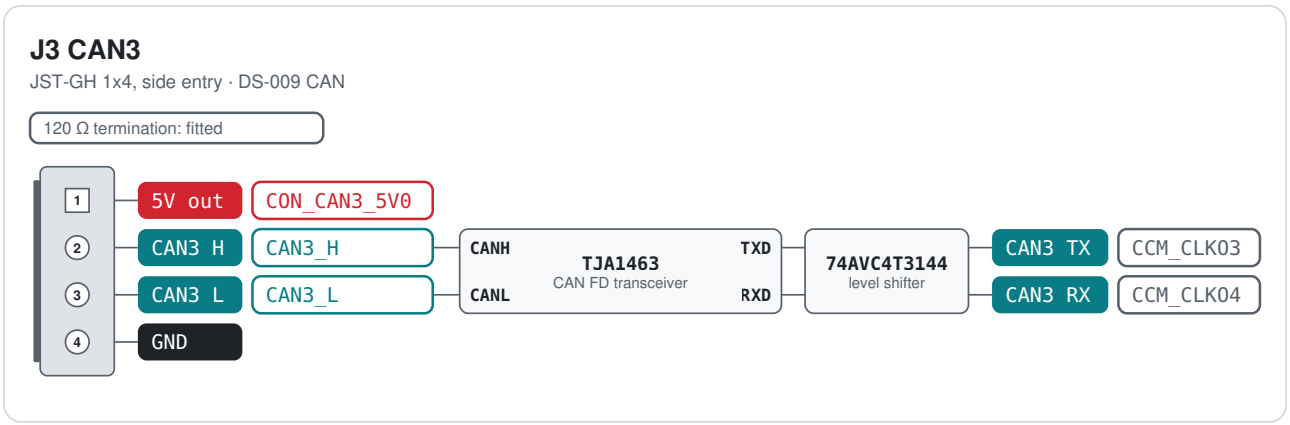
TJA1463 CAN FD transceiver with 5 V supply. The 120 Ω split termination is on the board, so each CAN port is a bus end. CANH and CANL together form the differential bus. TXD and RXD are the logic pins of the transceiver and go to the FlexCAN pads of the i.MX95. The transceiver enable and standby pins are controlled through the PCAL6524 I/O expander.

IO J4 CAN2



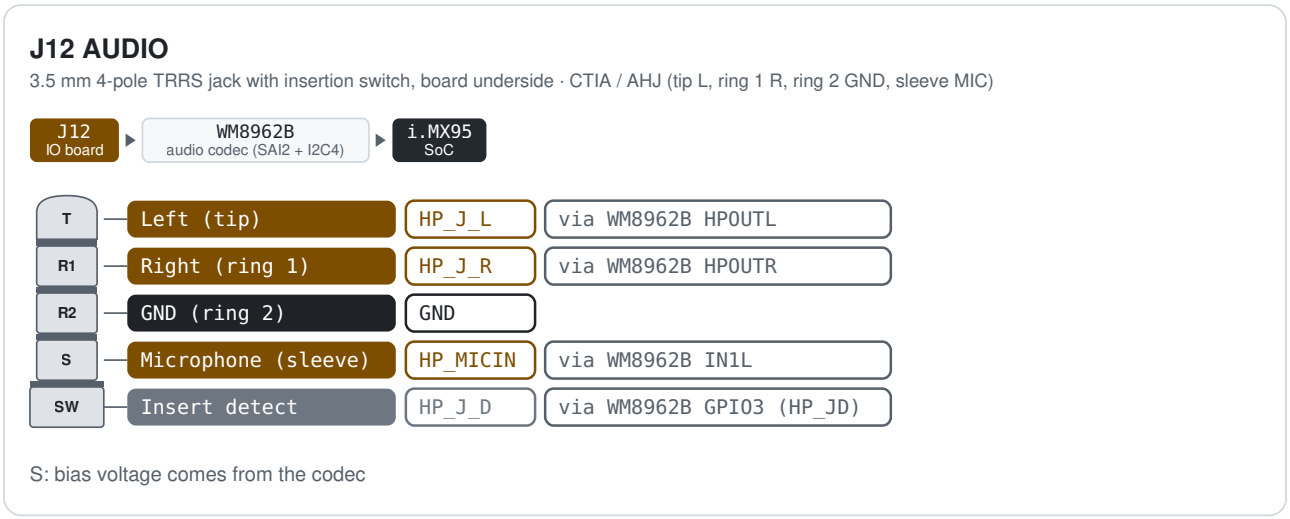
Same circuit as CAN1, with the 120 Ω termination.

IO J3 CAN3



Same circuit as CAN1, with the 120 Ω termination.

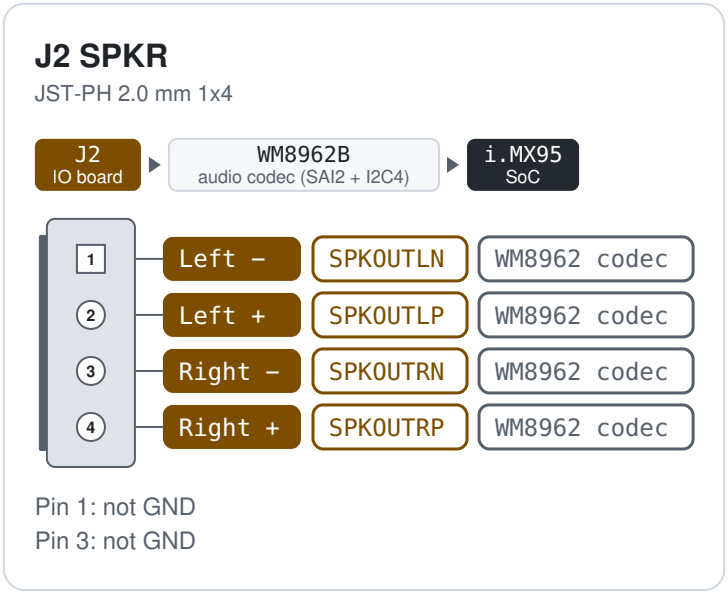
IO J12 AUDIO



Plug contacts listed from tip to sleeve. The switch in the jack drives HP_JD: high with a plug inserted, low without (100 kΩ pull-up to 1.8 V). The codec uses it to detect the plug. The headphone ground reference is the HP_FB sense line of the codec.

> Note: CTIA wiring only. OMTP headsets (microphone and ground swapped) do not work.

IO J2 SPKR



Bridge-tied class-D outputs of the WM8962B codec. The 4-pin JST-PH plug of the Adafruit 1669 Stereo Enclosed Speaker Set (2x 3 W, 4 Ω) fits directly. J25 next to it is the headset jack; the board also has two PDM microphones.

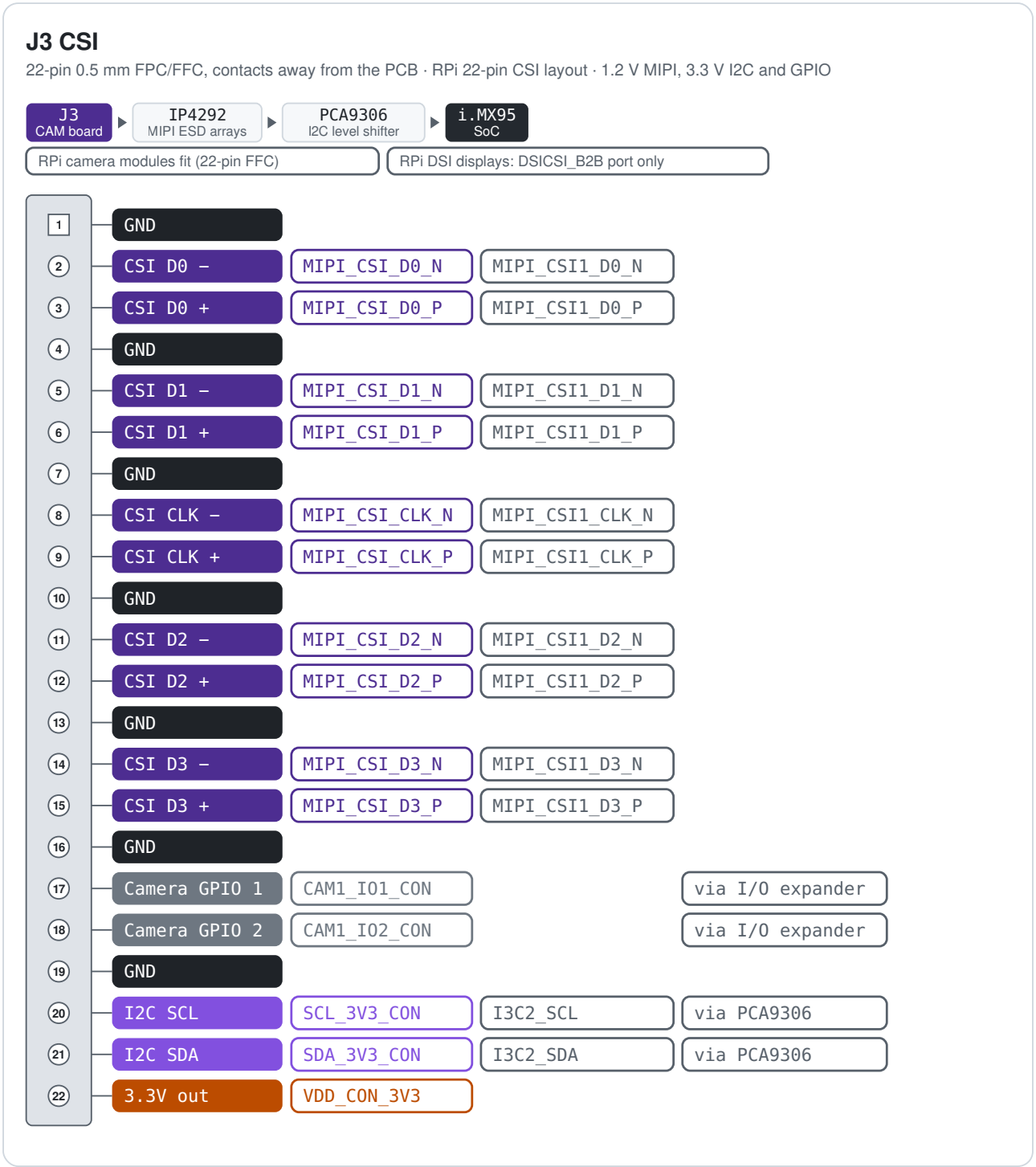
> Caution: No GND on this connector. Each speaker goes between its – and + pin. Never connect a speaker pin to ground.

13.6 Buttons and switches

Ref	Board	Function	Type	Notes
SW1 SYS_RST	Main	System reset button (short press: reset)	Tactile switch, right of the ON/OFF button, hidden under the thermal pad in the photo	Resets the whole board.
SW2 BTMODE	Main	Boot mode DIP switch	4-position DIP switch, top side under the M.2 Key M module	See Boot Switch (SW2).
SW3 ONOFF	Main	Power button (short press: on, 5 s press: off)	Tactile switch, left of the reset button	Talks to the PMIC ONOFF input.
S1 BUZZER	IO	Buzzer enable DIP switch	1-position DIP switch (CHS-01TA1), next to the ARM button	ON connects TPM6_CH1 to the buzzer FET. OFF disconnects the buzzer and the GPS buzzer pin, so PWM8 on the PWM header is a free channel.
SW1 ARM	IO	Arm / safety switch with LED	Tactile switch	Wired in parallel with the safety-switch and LED pins of the GPS connector.
S1 SMI	T1S	SMI routing switch	1-position DIP switch (CHS-01TA1)	Leave as delivered. Only used when the SJA1110 runs without the host.
S2 BOOT_OPT	T1S	SJA1110 boot option switch	2-position DIP switch	Boot option of the SJA1110; see the SJA1110 documentation before changing it.

13.7 Camera expansion board

CAM J3 CSI

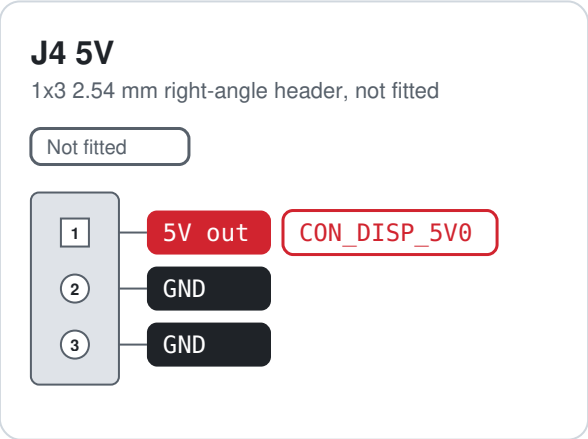


Pin-compatible with the 22-pin RPi camera connector (22-pin 0.5 mm same-side FFC). Two identical camera boards can be fitted. In the bottom view, the left one sits on CSI_B2B and goes to the i.MX95 MIPI_CSI1 port; the right one sits on DSICSI_B2B and goes to the second port, which is shared between DSI and CSI. The camera I2C is the i.MX95 I3C2 bus, shifted from 1.8 V to 3.3 V by a PCA9306. The two GPIO pins come from the I/O expander on the main board; the exact expander pins and the pad names of the second port were not traced. Whether a camera module works depends on the software image, not on the connector. J4 next to it is a 3-pin 5 V header for a display, not fitted.

Tip: RPi camera modules fit directly with a standard 22-pin FFC. For an RPi DSI display use the camera board on the DSICSI_B2B port; the CSI_B2B port cannot drive a display.

Note: RPICAM3 is one of several cameras supported. These are readily available. NXP has P/N MR-RPICAM3 which may be purchased as an accessory.

CAM J4 5V



Filtered 5 V from the system rail with ESD protection, meant for a DSI display on the DSICSI_B2B camera board. The silkscreen marks pin 1 as 5V.

13.8 T1 Ethernet switch board

The MR-NAVQ95E-T1S board is an NXP SJA1110 automotive Ethernet switch. It is connected to the main board through B2B_ETH on SGMII port 4 of the switch (the host port). Ports 1 and 2 are 1000BASE-T1 through two TJA1120 PHYs on SGMII ports 1 and 2. Ports 5 to 10 are 100BASE-T1 on the switch's own PHYs. SGMII port 3 is not used. The port numbers on the bottom view are the switch port numbers, which also match the silkscreen.

T1S ports and headers

Ref	Function	Connector	Notes
J4 PORT1	1000BASE-T1 port 1	Single-pair automotive connector	TJA1120 PHY on SJA1110 SGMII port 1.
J3 PORT2	1000BASE-T1 port 2	Single-pair automotive connector	TJA1120 PHY on SJA1110 SGMII port 2.
J5 PORT5	100BASE-T1 port 5	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 5.
J6 PORT6	100BASE-T1 port 6	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 6.

Ref	Function	Connector	Notes
J7 PORT7	100BASE-T1 port 7	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 7.
J8 PORT8	100BASE-T1 port 8	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 8.
J9 PORT9	100BASE-T1 port 9	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 9.
J10 PORT10	100BASE-T1 port 10	2-pin single-pair connector	SJA1110 internal 100BASE-T1 PHY, switch port 10.
J2 DEBUG	Debug header of the SJA1110 (Arm Cortex-M7 inside the switch)	2x5 1.27 mm	Not the i.MX95 debug port; that is J7 on the main board.

13.9 Notes

- The connectors marked DS-009 on their card (TELEM, GPS, I2C, CAN and the J2 console header) follow the Dronecode DS-009 connector standard, so standard cables fit. The other JST-GH connectors, such as the PWM header, have their own pin order.
- Input 9 V to 52 V (never more than 60 V), 3S to 12S batteries. 60 W between 9 V and 20 V, 125 W between 20 V and 52 V.
- The 5 V pin of every JST-GH connector is a filtered, ESD-protected output of the 5 V system rail (`VDD_CON_*_5V0` nets).
- The servo rail of the PWM connector (`VDD_CON_PWM`) is separated from the 5 V rail by R60, a 0 Ω link that is not fitted. This allows an external servo supply. The rail is measured on ADC_IN6 and must stay below 12 V.
- VIN, 5 V and 3.3 V are measured on ADC_IN0, ADC_IN1 and ADC_IN2.

> Warning: Never feed power into a 5 V pin of a JST-GH connector. They are outputs. Power the board through J9 PWR_IN only.

13.10 Cheat sheet

All connector cards on one double-sided A4 landscape sheet: cheat sheet PDF.

13.11 Related Guides

Cortex-A Linux image:

- Getting the Cortex-A Linux Image
- Building the Yocto Image from Source
- Flashing the Cortex-A Linux Image

Cortex-M RTOS firmware:

- Getting the Cortex-M RTOS Firmware
- Flashing the Cortex-M RTOS Firmware

Overview:

- Flashing the MR-NavQ95

13.12 References

- MR-NAVQ95 board design and documentation

14 Installing and configuring software tools

This section describes how to obtain or build the two images the MR-NavQ95 runs — the Cortex-A Linux image and the Cortex-M RTOS firmware — and how to manage the device-tree overlays applied at boot.

14.1 Getting the Cortex-A Linux image

> Host PC: An Ubuntu PC is assumed for all steps on this page.

There are two ways to obtain the Linux image for the MR-NavQ95:

Option 1: Download the Pre-built Image

Follow the NXP Robotics SDK README to set up the SDK and download the images. The Cortex-A Linux image will be located at:

```
<west_topdir>/bin/mr-navq95/linux/imx-image-robotics-imx95-navq.rootfs.wic.zst
```

Option 2: Build from Source (Yocto)

Build the image from source using the Yocto build system. See Building the Yocto Image for MR-NavQ95 for the full build instructions.

The output image will be located in:

```
<yocto_dir>/build/tmp/deploy/images/imx95-navq/imx-image-robotics-imx95-navq.rootfs.wic.zst
```

> Note: The `.wic` image bundles the M7 Cortex-M firmware alongside the Linux root filesystem. By default, the M7 core runs MCUboot loaded directly from the core's TCM. To replace the M7 firmware with a custom RTOS image, see Flashing the Cortex-M RTOS Firmware.

Once the image is available, proceed to Flashing the Cortex-A Linux Image to flash it to the board.

14.2 Building the Cortex-A Linux image from source

This section describes the steps to build the Yocto image for the MR-NavQ95 platform from source.

Prerequisites

Before starting, ensure the host machine meets the following requirements:

- OS: Ubuntu 22.04 or later (recommended)
- Disk space: At least 300 GB of free space for a full Yocto build
- RAM: 16 GB minimum recommended

1. Install Required Host Packages

Install the required host packages:

```
sudo apt-get install build-essential chrpath cpio debianutils diffstat file gawk \
  gcc git iputils-ping libacl1 libcrypt-dev locales python3 python3-git python3-jinja2 \
  python3-pexpect python3-pip python3-subunit socat texinfo unzip wget xz-utils zstd
efitools
```

2. Install the Repo Tool

Install the `repo` tool if not already present:

1. Create a `bin` folder in the home directory and download `repo`:

   ```
   mkdir ~/bin
   curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
   chmod a+x ~/bin/repo
   ```

2. Add `~/bin` to the `PATH` by appending the following line to `~/.bashrc`:

   ```
   export PATH=~/bin:$PATH
   ```

Steps

1. Set Up the Yocto Project

Create a working directory, initialize the manifest repository, and sync all required layers:

```
mkdir imx-yocto-bsp
cd imx-yocto-bsp
repo init -u https://github.com/nxp-robotics/robotics-linux-manifest.git -b robotics-0.7.0
-m imx-6.18.20-2.0.0_wrynose.xml
repo sync
```

2. Build the Image

Source the release setup script to configure the build environment, then kick off the BitBake build:

```
DISTRO=imx-desktop-xwayland MACHINE=imx95-navqdesktop source imx-setup-release.sh -b build
bitbake imx-image-robotics
```

> Note: The build may take a significant amount of time depending on the host machine's resources. The output image will be located in `build/tmp/deploy/images/imx95-navq/`.

Once the build completes, proceed to Flashing the Cortex-A Linux Image to write the image to the board.

14.3 Getting the Cortex-M RTOS firmware

There are two ways to obtain the M7 RTOS firmware for the MR-NavQ95:

Option 1: Download the Pre-built Image

Follow the NXP Robotics SDK README to set up the SDK and download the images. The Cortex-M firmware will be located at:

```
<west_topdir>/bin/navq95/realtime/
├── nxp_imx95_default.mcuboot.bin   # TCM image, signed (use this for default deployment)
└── nxp_imx95_ddr.bin               # DDR image, unsigned
```

> Use the signed `nxp_imx95_default.mcuboot.bin` for the default/TCM image — MCUboot verifies it on boot. The DDR image is intentionally unsigned.

Option 2: Build from Source

Use the NXP Robotics SDK to build the RTOS firmware from source:

```
west sdk build --platform navq95 --stack px4
```

The output firmware will be located in the `build/` directory and also copied to `bin/navq95/realtime/`.

Once the firmware is available, proceed to Flashing the Cortex-M RTOS Firmware to flash it to the board.

14.4 Device-tree overlays in U-Boot

The MR-NavQ95 board uses a single base device tree blob (`imx95-navq.dtb`)
combined with per-connector `.dtbo` overlay files that are applied at boot time by U-Boot. This design allows cameras, displays, or Ethernet adapters to be swapped on the board's B2B connectors without rebuilding the kernel.

Overview

At boot, U-Boot:

1. Loads the base DTB from the boot medium into memory at `fdt_addr_r`.
2. Resizes the in-memory DTB to leave room for overlay data.
3. Iterates over the three connector overlay variables (`csi_b2b` , `dsicsi_b2b` , `eth_b2b`).
4. For each variable that is not `none` and not empty, loads the named `.dtbo` file into `ov_addr_r` and calls `fdt apply` to merge the overlay into the in-memory DTB.
5. Boots the kernel with the fully patched device tree.

Environment Variables

The relevant variables are defined in `board/nxp/imx95_evk/imx95_evk.env` under the `CONFIG_TARGET_IMX95_NAVQ` guard.

Overlay selection variables

Variable	Default value	Connector / function
`csi_b2b`	`imx95-navq-csi-imx219.dtbo`	Port A — CSI-only (camera)

Variable	Default value	Connector / function
`dsicsi_b2b`	`imx95-navq-dsicsi-imx219.dtbo`	Port B — DSI/CSI combo
`eth_b2b`	`imx95-navq-sja1110a.dtbo`	Ethernet B2B (SJA1110A switch)

Set any variable to `none` (or leave it empty) to skip that overlay entirely.

Helper commands

Command	What it does
`apply_one`	Loads `${ovfile}` using `${get_cmd}`, then calls `fdt apply`
`apply_overlays`	Sets the FDT address, resizes the blob, then runs `apply_one` for each active connector variable
`boot_os`	Runs `apply_overlays`, then boots the kernel with `booti`
`modules_help`	Prints the list of available `.dtbo` files for each connector

---

How `apply_overlays` Works

```
apply_overlays=
    fdt addr ${fdt_addr_r};
    fdt resize 0x20000;
    echo "NavQ-B connectors: csi_b2b=${csi_b2b} dsicsi_b2b=${dsicsi_b2b}";
    if test "${csi_b2b}" != none; then if test "${csi_b2b}" != ""; then
        setenv ovfile ${csi_b2b}; run apply_one; fi; fi;
    if test "${dsicsi_b2b}" != none; then if test "${dsicsi_b2b}" != ""; then
        setenv ovfile ${dsicsi_b2b}; run apply_one; fi; fi;
    if test "${eth_b2b}" != none; then if test "${eth_b2b}" != ""; then
        setenv ovfile ${eth_b2b}; run apply_one; fi; fi;
```

For each connector, `apply_one` does:

```
apply_one=
    echo "  overlay: ${ovfile}";
    if ${get_cmd} ${ov_addr_r} ${ovfile}; then
        if fdt apply ${ov_addr_r}; then true;
        else echo "  WARN: failed to apply ${ovfile}"; fi;
    else echo "  WARN: ${ovfile} not found on boot partition"; fi;
```

- `${get_cmd}` is set by the active boot path (`mmcboot` sets it to `fatload mmc ${mmcdev}:${mmcpart}`, while `netboot` sets it to `dhcp` or `tftp`). This ensures overlay loading works for both MMC and network boot.
- `fdt apply` merges the overlay blob at `ov_addr_r` into the base DTB already set active via `fdt addr`. Phandle fixups are performed automatically by U-Boot's `fdt_overlay_apply_verbose()` (see `boot/fdt_support.c`).

Overlay application sequence

```
mmcboot
  |-- fatload mmc  ->  loadaddr  (kernel Image)
  |-- fatload mmc  ->  fdt_addr_r  (imx95-navq.dtb)
  `-- boot_os
        |-- apply_overlays
        |     |-- fdt addr ${fdt_addr_r}
        |     |-- fdt resize 0x20000
        |     |-- apply_one  (csi_b2b overlay)
        |     |     |-- fatload mmc -> ov_addr_r  (.dtbo)
        |     |     `-- fdt apply ${ov_addr_r}
        |     |-- apply_one  (dsicsi_b2b overlay)
        |     `-- apply_one  (eth_b2b overlay)
        `-- booti ${loadaddr} - ${fdt_addr_r}
```

Selecting a Different Module

From the U-Boot prompt, run `run modules_help` to see all available overlays:

```
=> run modules_help
NavQ-B module selection (one .dtbo per connector):
  csi_b2b    (port A, CSI-only): imx95-navq-csi-{imx219,imx477}.dtbo | none
  dsicsi_b2b (port B, DSI/CSI): imx95-navq-dsicsi-
{imx219,imx477,disp-7inch,disp-5inch,lt8912b}.dtbo | none
  current: csi_b2b=imx95-navq-csi-imx219.dtbo dsicsi_b2b=imx95-navq-dsicsi-imx219.dtbo
  example: setenv dsicsi_b2b imx95-navq-dsicsi-disp-7inch.dtbo; saveenv
```

To permanently switch, for example, port B to an imx477 camera:

```
=> setenv dsicsi_b2b imx95-navq-dsicsi-imx477.dtbo
=> saveenv
=> reset
```

To disable a connector overlay (e.g. no camera on port A):

```
=> setenv csi_b2b none
=> saveenv
=> reset
```

Adding a New Overlay

1. Create the DTS overlay source (`.dtso`) for the target hardware and build it to produce a `.dtbo` binary.
2. Copy the `.dtbo` file to the FAT boot partition of the eMMC or SD card (the same partition that holds `Image` and `imx95-navq.dtb`).
3. At the U-Boot prompt, point the appropriate connector variable to the new file:

   ```
   => setenv csi_b2b imx95-navq-csi-mymodule.dtbo
   => saveenv
   ```

4. Reboot — `apply_overlays` will load and apply the overlay automatically.

15 Configuring the hardware

The MR-NavQ95 has two types of images that can be flashed separately:

Domain	Description	Guide
Cortex-A Linux image	Full Linux root filesystem written to eMMC or SD card using UUU	Flashing the Cortex-A Linux Image
Cortex-M RTOS firmware	M7 RTOS firmware written to NOR flash using pyOCD	Flashing the Cortex-M RTOS Firmware

> The Cortex-A must be flashed first, prior to flashing the Cortex-M.

For boot switch settings and connector locations, see MR-NavQ95 Board Overview.

15.1 Flashing the Cortex-A Linux image

This section describes how to flash the Linux image to the MR-NavQ95 eMMC or SD card using the NXP Universal Update Utility (UUU).

For boot switch settings and connector locations, see MR-NavQ95 Board Overview.

Prerequisites

Before flashing, obtain the Linux image as described in Getting the Cortex-A Linux Image.

Steps

1. Download UUU

Download the latest UUU executable from the mfgtools GitHub releases page:

- Windows: download `uuu.exe`
- Linux: download `uuu`

For convenience, place the UUU executable in the same directory as the `.wic` image.

2. Prepare the Board

1. Connect a USB Type-C cable from USB1 (J13) on the MR-NavQ95 to the host PC. J13 is the Serial Download port used by UUU.
2. Connect a USB Type-C cable from DEBUG USB (J10) on the MR-NavQ95 to the host PC. This exposes two serial consoles, which are needed in step 4 to monitor the boot output.
3. Set the boot switches to Serial Downloader mode: `SW2[1:4] = 1001`

4. Connect a 9–52 V DC power supply to the power input connector J9.
5. Verify the board is detected by UUU:

```
./uuu -lsusb
```

An NXP i.MX95 device should be listed. If the device is not detected, check the USB cable and boot switch settings.

3. Flash the Image

Navigate to the directory containing the `.wic` image and run UUU:

```
# Flash to eMMC
./uuu -b emmc_all imx-image-robotics-imx95-navq.rootfs.wic.zst
# OR
# Flash to SD card
./uuu -b sd_all imx-image-robotics-imx95-navq.rootfs.wic.zst
```

When flashing completes successfully, UUU will print `Done` and `Success 1`.

4. Boot the Board

1. Set the boot switches to match the flashed media:
   - eMMC boot: `SW2[1:4] = 0101`
   - SD card boot: `SW2[1:4] = 1101`
2. The DEBUG USB port exposes two serial consoles — one for the Cortex-A (Linux) and one for the Cortex-M (RTOS). Open both on the host PC at 115200 baud, 8N1.
3. Power-cycle the board and monitor the boot output in both consoles.

Default Linux login: user / user

15.2 Flashing the Cortex-M RTOS firmware

This section describes how to flash the M7 RTOS firmware to NOR flash on the MR-NavQ95 using the NXP-Robotics pyOCD fork.

For boot switch settings and connector locations, see MR-NavQ95 Board Overview.

Prerequisites

- Cortex-A image: The Cortex-A Linux image must be flashed to the board before flashing the Cortex-M firmware. See Flashing the Cortex-A Linux Image.
- Host PC: Ubuntu is required to run pyOCD. Other Linux distributions may work but are not tested.
- JTAG probe: A JTAG debug probe (e.g. NXP MCU-Link or J-Link) must be connected between the host PC and the MR-NavQ95 JTAG connector before flashing.
- Firmware image: Obtain the RTOS firmware as described in Getting the Cortex-M RTOS Firmware.

Steps

1. Install the pyOCD Fork

The i.MX95 target is not available in mainline pyOCD. Install the NXP-Robotics fork on the host PC:

```
git clone https://github.com/NXP-Robotics/pyOCD -b pr-imx95 && cd pyOCD

# Option 1: install using python virtual environments. If you choose
# this option, please make sure you always activate the virtual environment
# before trying to use pyOCD!
python3 -m venv .venv
source ./.venv/bin/activate
python3 -m pip install .

# Option 2: install using pipx
sudo apt-get install -y pipx
# you may need to open a new terminal session for this to take effect!
pipx ensurepath
pipx install .
```

> Tested on Python 3.10 (Python 3.9 also works).

2. Prepare the Board

1. Set the boot switches to the boot mode corresponding to the device where the Cortex-A image was written (eMMC or SD).
2. Connect a USB Type-C cable from DEBUG USB (J10) on the MR-NavQ95 to the host PC. This exposes two serial consoles - one for Cortex-A, one for Cortex-M.
3. Connect the JTAG probe between the host PC and the MR-NavQ95 JTAG connector (J7).
4. Apply 9–52 V DC to the power input connector J9.
5. Open both serial consoles and wait for the board to reach the Linux login prompt before proceeding.

3. Flash the M7 Image to NOR

Run the following command on the host PC:

```
pyocd flash -t mimx95_cm7_mx25um --base-address 0x28020000 \
    nxp_imx95_default.mcuboot.bin -f 10m
```

Parameter	Value
Target	`mimx95_cm7_mx25um`
NOR base address	`0x28020000`

Parameter	Value
Flash clock	`10m` (10 MHz)

> Note: Writing to NOR flash is not fully stable in the current release. Retry the `pyocd flash` command until pyOCD reports it programmed 0 pages, which confirms the flash contents match the binary.

> Note: The flash clock speed can be reduced from 10 MHz down to 4 MHz (4m) if multiple (> 5) failures occur.

> Note: In the unlikely event where the flashing fails with `flash init timed out [__main__]` multiple times, flashing using the CM33 AP may be attempted by following these steps:
>
> 1. Reboot the NavQ95 platform in Serial Downloader Mode, see [Serial Downloader mode (configuring-the-hardware.md#flashing-the-cortex-a-linux-image)]
> 2. Write the image to NOR FLASH via the M33 core with `pyocd flash -t mimx95_cm33_mx25um --base-address 0x28020000 nxp_imx95_default.mcuboot.bin -f 10m`
> 3. Apply the same limitations as explained for the the CM7 direct write related to stability

4. Reboot the Board and Open the M7 Console

After reboot, the M7 RTOS shell is accessible from Linux on `/dev/ttynsh` via the serial console:

```
picocom /dev/ttynsh
```

16 Abbreviations

Acronym	Description
BSP	Board support package
DTB	Device tree blob
DTBO	Device tree blob overlay
eMMC	Embedded MultiMediaCard
NPU	Neural processing unit
RTOS	Real-time operating system
SD	Secure Digital
SDK	Software development kit
SoC	System on chip
UUU	Universal Update Utility
USB	Universal Serial Bus

17 References

- MR-NavQ95 product page — overview, documentation, software, tools, and ordering information
- i.MX 95 product page — SoC overview, documentation, and tools
- MR-NavQ95 hardware reference — connectors, pinouts, and i.MX 95 pad assignments
- MR-NavQ95 Hardware Architecture and Reference Design
- MR-NavQ95 Deploy and Run Quick Start
- MR-NAVQ95 repository — schematics, board layout, and bill of materials
- NXP Robotics SDK (robotics-sdk)

18 Application Notes

Hardware- and peripheral-level application notes for the MR-NavQ95: the hardware architecture and reference design, the MIPI-CSI camera pipeline, the automotive Ethernet stack, the cellular modem on the M.2 Key-B slot, on-device AI on the eIQ Neutron NPU and the Kinara ARA-240 DNPU, and the Remote ID transmitter.

- Hardware Architecture and Reference Design
- Camera
- Ethernet
- Cellular Modem
- eIQ Neutron NPU
- ARA-240 DNPU
- Remote ID Transmitter

Hardware Architecture and Reference Design

The MR-NavQ95 is the vehicle-computer reference design of the NXP Mobile Robotics family. It combines a high-performance application processor, a real-time control core, on-device AI acceleration, and automotive-grade connectivity on a single compact board, so that a drone or mobile robot can run its perception, autonomy, and real-time control from one validated compute platform. This application note describes the system hardware architecture, the main functional subsystems, and the hardware integration considerations for engineers designing the MR-NavQ95 into a product.

This document is hardware focused. It describes the architecture and the role of each subsystem rather than the pin-level connector detail. For connector pinouts, net names, and i.MX 95 pad assignments, see the MR-NavQ95 hardware reference. Software bring-up, image flashing, and configuration are covered in the MR-NavQ95 User Manual and the NXP Robotics SDK documentation.

1. Introduction

The MR-NavQ95 integrates the main NXP building blocks of a robotics compute node onto one board and a small set of expansion boards, so that compute, perception, connectivity, and real-time control can be evaluated and reproduced on validated hardware. It is intended as a starting point that customers adapt for their own autonomous-vehicle products.

The board is organized around the NXP i.MX 95 applications processor and three expansion boards that extend it with cameras, robotics I/O, and automotive Ethernet. A typical product would carry forward the i.MX 95 compute core and the subset of interfaces required by the target vehicle, re-laying the board to its own mechanical and electrical constraints.

2. System hardware architecture

The functional block diagram below shows how the i.MX 95 compute core connects to on-board memory, connectivity, and sensors, and to the camera, IO, and T1-switch expansion boards over board-to-board connectors.

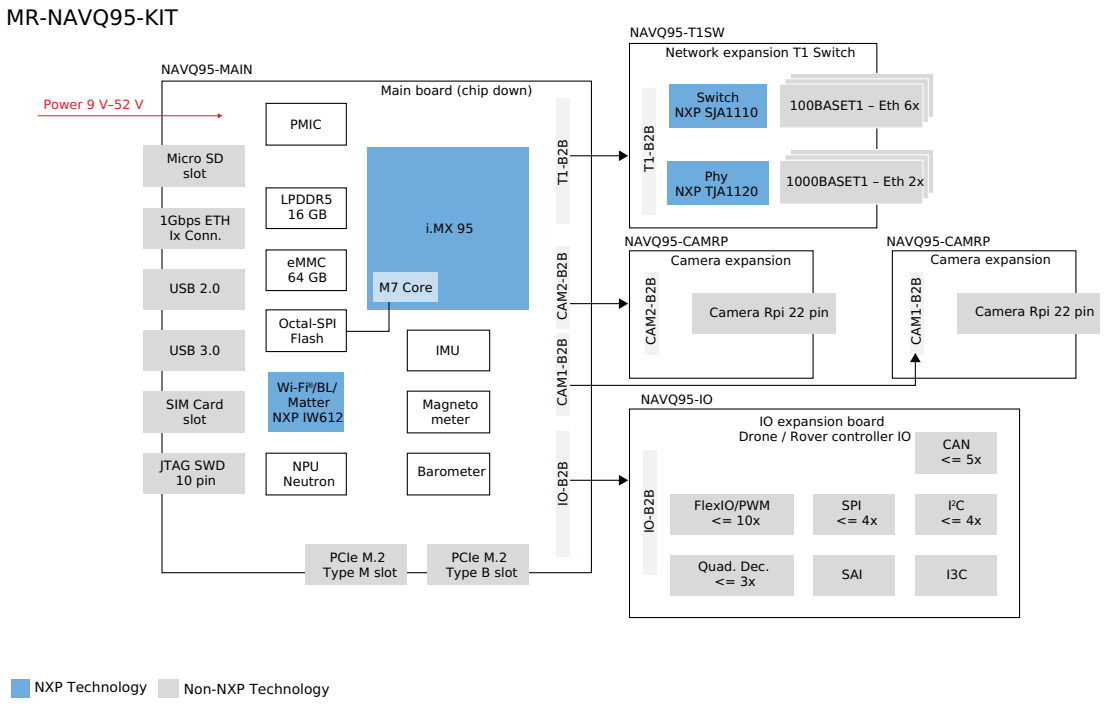


Figure 1. MR-NavQ95 functional block diagram

At the system level the design is partitioned into:

- Compute core: the i.MX 95 applications processor with its LPDDR memory, boot and mass storage, and the on-chip Neutron NPU.
- Connectivity: wireless (Wi-Fi, Bluetooth, Matter, and 802.15.4) and wired Ethernet, including the automotive 100/1000BASE-T1 subsystem on the T1-switch expansion board.
- Sensing and perception: the on-board inertial, magnetometer, and barometric sensors, and the MIPI CSI-2 camera inputs on the camera expansion board.
- Robotics I/O: the CAN-FD, PWM, serial, I2C, and audio interfaces on the IO expansion board that connect the board to a vehicle.
- Power: the wide-range input stage and the on-board rails derived from it.

Each subsystem is described in the following sections. The physical connector that carries each interface, together with its pinout, is listed in the hardware reference; it is not repeated here.

3. Compute core

The compute core is the NXP i.MX 95 applications processor, a heterogeneous multicore SoC. It combines six Arm Cortex-A55 application cores for the Linux compute stack and robotics middleware, a Cortex-M7 real-time core for hard real-time control, and a Cortex-M33 System Manager. A single device therefore covers both the application and the real-time domains of a robot, with inter-core messaging between the Cortex-A and Cortex-M domains.

The on-chip NXP Neutron neural processing unit (NPU) provides machine-learning inference for the perception and AI workloads running on the Cortex-A cores. For heavier vision workloads, the board can host a discrete NPU module on one of its M.2 slots; see Section 7, Expansion and acceleration.

The processor boots from and runs against the board's memory subsystem: LPDDR system memory, eMMC and octal-SPI flash for boot and mass storage, and a microSD slot for removable media. The boot source is selected by the boot-mode switch; the switch settings are documented in the hardware reference.

4. Connectivity

4.1 Wireless

Wireless connectivity is provided by an NXP combo radio offering Wi-Fi, Bluetooth, and Matter, together with an NXP 802.15.4 radio for Thread and Zigbee. These links carry telemetry, command, and data-offload traffic for the robot.

4.2 Automotive Ethernet

Wired networking is built around automotive Ethernet. In addition to gigabit Ethernet on the main board, the T1-switch expansion board (MR-NAVQ95E-T1S) adds a managed automotive-Ethernet switch based on the NXP SJA1110. The switch connects to the i.MX 95 over SGMII (the host port) and exposes:

- two 1000BASE-T1 ports through NXP TJA1120 PHYs, and
- six 100BASE-T1 ports on the SJA1110 internal PHYs.

This forms the time-sensitive-networking backbone that links the MR-NavQ95 to distributed sensor nodes and to the flight controller. The port assignments and connector types are listed in the hardware reference; the usage and software configuration of the Ethernet subsystem are covered in the Ethernet Application Note.

5. Sensing and perception

5.1 On-board sensors

The main board carries an on-board inertial measurement unit, a magnetometer, and a barometric pressure sensor for motion, heading, and altitude estimation. These support state estimation when the MR-NavQ95 runs the flight or motion stack on its Cortex-M7 core.

5.2 Camera interface

The camera expansion board brings the i.MX 95 MIPI CSI-2 inputs out to standard Raspberry Pi-style camera connectors, feeding the on-chip image signal processor and the vision pipeline. Camera selection, connection, and streaming are described in the Camera Application Note; the connector pinouts are in the hardware reference.

6. Robotics I/O

The IO expansion board adapts the i.MX 95 to the interfaces a vehicle needs: CAN-FD buses, PWM motor and servo outputs, UART telemetry, GPS and safety inputs, I2C, and an audio codec. Most of these connectors follow the Dronecode DS-009 connector standard, so standard cables fit. The connector set, pin order, and the DS-009 markings are documented in the hardware reference.

> Note: The 5 V pin of every JST-GH connector is a filtered, ESD-protected output of the 5 V system rail. These pins are outputs; the board must be powered through the main power input only.

7. Expansion and acceleration

The main board provides two PCIe M.2 slots. These can host mass storage or a discrete NPU module for accelerated vision and generative-AI workloads that exceed the headroom of the on-chip Neutron NPU. The expansion of the board into cameras, robotics I/O, and automotive Ethernet is handled by the three board-to-board expansion boards described above.

8. Power

The board accepts a wide input range suitable for typical robotics batteries and derives the regulated system rails required by the compute core, expansion boards, and connector outputs from it. The input range, power budget, and the ADC channels used to monitor the input and the derived rails are documented in the hardware reference. Input limits, protection behavior, and regulator ratings must be confirmed against the released schematic for the board revision in use.

> Warning: Never feed power into a 5 V output pin of a connector. Power the board through its main power input only.

9. Hardware integration and bring-up

- Confirm the main board and expansion board part numbers and revisions.
- Confirm the input supply range and polarity against the released schematic before applying power.
- Verify connector orientation and mating; the connector pinouts are in the hardware reference.
- Bring the board up using a current-limited supply, and verify the expected rails before connecting the node to the wider robotic system.
- Observe electrostatic-discharge precautions and provide mechanical support and strain relief appropriate to the installation.

Software bring-up, image flashing, and device-tree overlays are covered in the MR-NavQ95 User Manual.

10. Schematic, board layout, and bill of materials

This application note provides a system-level hardware overview and should be used together with the released design files. Electrical limits, connector pin assignments, signal direction, power-domain relationships, and assembly options must be confirmed against the released schematic and bill of materials for the applicable board revision. The schematics, board layout, and design files are published in the MR-NAVQ95 repository.

11. Abbreviations

Acronym	Description
AI	Artificial intelligence
CAN FD	Controller Area Network Flexible Data-rate

Acronym	Description
CSI	Camera Serial Interface
eMMC	Embedded MultiMediaCard
MCU	Microcontroller unit
MIPI	Mobile Industry Processor Interface
NPU	Neural processing unit
PCIe	Peripheral Component Interconnect Express
PHY	Physical-layer transceiver
ROS 2	Robot Operating System 2
SAI	Serial audio interface
SGMII	Serial Gigabit Media-Independent Interface
SoC	System on chip
SPI	Serial Peripheral Interface
TSN	Time-Sensitive Networking
UART	Universal asynchronous receiver-transmitter

12. References

- MR-NavQ95 hardware reference — connectors, pinouts, and i.MX 95 pad assignments
- MR-NavQ95 User Manual — software bring-up, flashing, and configuration
- MR-NAVQ95 repository — schematics, board layout, and bill of materials

19 Camera

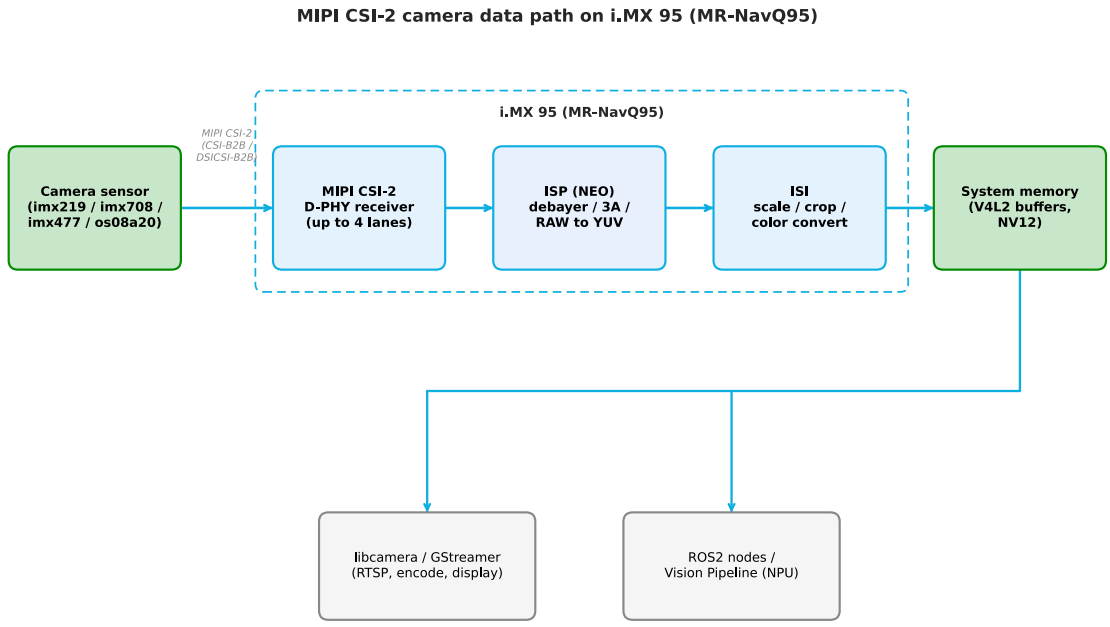
19.1 Introduction

This application note describes the MIPI CSI-2 camera subsystem of the MR-NavQ95 vehicle computer (based on the NXP i.MX 95 applications processor) and shows how to bring up, capture from, and stream a camera on the board. It covers the camera data path, the supported sensors, the physical connection, the device-tree overlay selection performed in U-Boot, and the capture / streaming software on the Linux (Cortex-A55) side.

It is intended for engineers integrating a camera into a robotics application on MR-NavQ95, whether the frames are consumed for streaming, for display, or as an input to on-device inference.

19.2 Camera subsystem architecture

On i.MX 95, image data flows from the sensor through the MIPI CSI-2 receiver, is processed by the on-chip Image Signal Processor (ISP, the “NEO” pipeline), scaled/converted by the Image Sensing Interface (ISI), and written to system memory as V4L2 buffers. From memory the frames are consumed by libcamera / GStreamer (for streaming, encode or display) or by robotics software such as a ROS2 node or the Vision Pipeline feeding the Neutron NPU.



Key i.MX 95 camera capabilities relevant to MR-NavQ95:

- 2× MIPI CSI-2 4-lane receivers — one of them is muxed with the MIPI-DSI display interface. On MR-NavQ95 these are exposed on the CSI-B2B and DSICSI-B2B connectors.
- On-chip ISP (NEO) — Bayer debayering, auto-exposure / auto-white-balance / auto-focus, and RAW→YUV conversion at up to ~500 Mpx/s.
- ISI — hardware scale, crop and color-space conversion; produces NV12 (and other) formats for downstream use.
- Up to 8 concurrent streams via CSI-2 virtual channels.

19.3 Supported sensors

Sensor	Module	Adapter	dtbo
imx219	RPi Camera Module 2	22→15-pin	`imx95-navq-(dsi)csi-imx219.dtbo`
imx708	RPi Camera Module 3	22→15-pin	`imx95-navq-(dsi)csi-imx708.dtbo`
imx477	RPi HQ Camera (lens req.)	22→15-pin	`imx95-navq-(dsi)csi-imx477.dtbo`
os08a20	EXPI-os08a20	none (22-pin direct)	`imx95-navq-(dsi)csi-os08a20.dtbo`

> Note: P/N MR-RPICAM3 is available as an accessory from NXP.com. This is also readily available online from other sources of Raspberry Pi cameras.

19.4 Setup

Connect camera cable

Note: Inserting the cable in the wrong way will damage the camera or the board. Check carefully the images below.

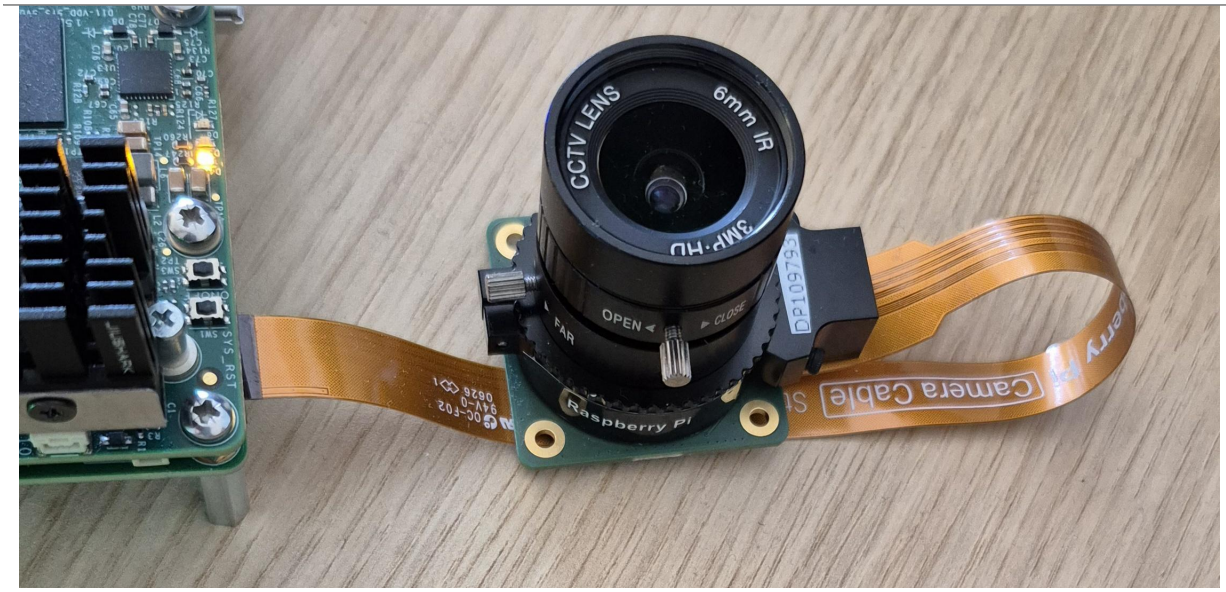
1. imx219, imx708, imx477

All cameras are using the same connector (15 pins) and needs the 22 pins to 15 pins adapter (orange cable).
All cameras are using the same cable orientation.

imx708/imx219:

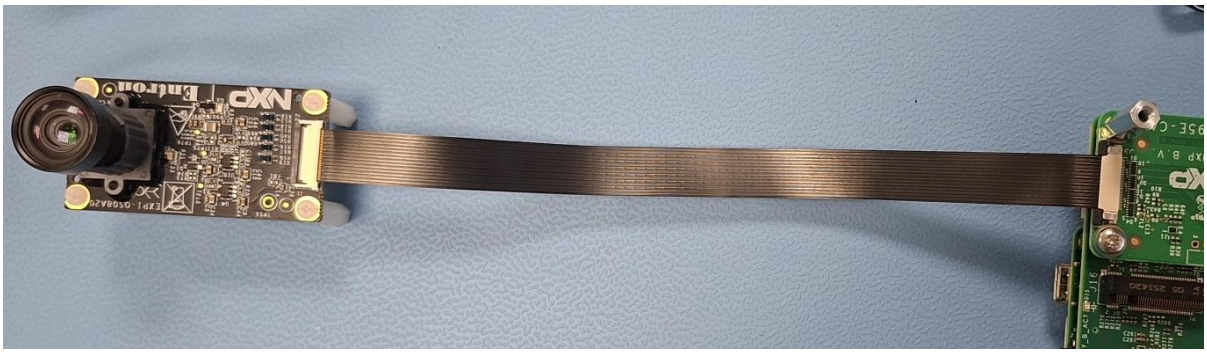


imx477 (needs lens | not built-in the module):



Cable type: same side pins.

1. EXPI-os08a20



Cable type: opposite or same side pins.

The OS08A20 uses a 22 pins connector directly, no adapter needed. Its connector has pins on top and bottom.

!!!ALWAYS make sure to match pin 22 from the camera with pin 1 on the board!!!

U-Boot configuration

The camera is enabled by loading the matching device-tree overlay (dtbo) at boot. Two environment variables select the overlay per connector: `csi_b2b` for the CSI connector and `dsicsi_b2b` for the DSICSI connector. Set the one that matches the connector the camera is plugged into; set the other to `none` if nothing is connected there.

In U-boot:

```
ls mmc 0 # list all dtbo files; use mmc 0 for eMMC and mmc 1 for SD card

setenv csi_b2b imx95-navq-csi-imx708.dtbo           # the dtbo of the connected camera; if
none is available, enter 'none' instead of a dtbo file name
setenv dsicsi_b2b imx95-navq-dsicsi-disp-7inch.dtbo # the dtbo of the connected camera; if
none is available, enter 'none' instead of a dtbo file name
saveenv
boot
```

Set the variable based on which connector is in use: DSICSI or CSI

Note: Remove the MR-NAVQ95E-CAM module to see connector name



19.5 Usage

Verify the camera is detected

After booting with the overlay applied, confirm the sensor enumerated on the Linux side before streaming. Use libcamera's `cam` tool — it lists only the camera(s), whereas `v4l2-ctl` / `media-ctl` also enumerate every other multimedia device (ISP, VPU, etc.) and are confusing to read. Export the pipeline match list first:

```
export LIBCAMERA_PIPELINES_MATCH_LIST='nxp/neo,imx8-isi,uvcvideo'
cam -l          # or: cam --list-devices
```

If the camera does not appear, re-check the cable orientation and that the correct dtbo is set for the connector in use (see Troubleshooting).

RTSP server + client

If no display is connected, use RTSP server to stream video to another device. Both devices must be in the same network (can be pinged each other).

Navq (RTSP server)

For all cameras:

```
export LIBCAMERA_PIPELINES_MATCH_LIST='nxp/neo,imx8-isi,uvcvideo'
test-launch "libcamerasrc ! video/x-raw,format=NV12 ! videoconvert ! v4l2h264enc !
h264parse ! rtph264pay name=pay0 pt=96"
```

Note: Different resolutions can be achieved by adding `,width=X,height=Y` to the video/x-raw caps, check the camera capabilities.

`LIBCAMERA_PIPELINES_MATCH_LIST` tells libcamera which pipeline handlers to try, in order: `nxp/neo` (the i.MX 95 ISP/NEO pipeline), `imx8-isi` (ISI-only path, no ISP) and `uvcvideo` (USB webcams). Encoding uses the hardware H.264 encoder via `v4l2h264enc`.

External machine (RTSP client)

1. Install GStreamer + Add-Ons

```
sudo apt install -y gstreamer1.0-tools gstreamer1.0-plugins-base gstreamer1.0-plugins-good
gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav
```

1. Connect to RTSP Stream

```
gst-launch-1.0 rtspsrc location=rtsp://<board-ip>:8554/test latency=100 ! rtph264depay !
avdec_h264 ! autovideosink
```

19.6 Multi-camera and dual CSI-2

i.MX 95 exposes two MIPI CSI-2 4-lane receivers, so two cameras can run concurrently: one on the CSI connector and one on the DSICSI connector. Set both `csi_b2b` and `dsicsi_b2b` to the matching overlays. Note the DSI/CSI mux: the DSICSI port is shared with the MIPI-DSI display, so using it for a second camera precludes a MIPI-DSI panel on that port. Virtual channels allow up to 8 logical streams over the CSI-2 links.

19.7 Integration with the robotics stack

Once frames land in system memory as V4L2 buffers they can be consumed by robotics software as well as by libcamera/GStreamer:

- ROS2 — a camera node publishes frames on a ROS2 topic for the rest of the navigation / perception graph.
- Vision Pipeline — the GStreamer + NNStreamer pipeline pre-processes frames and runs inference on the Neutron NPU (e.g. detection / segmentation) directly from the camera stream.

19.8 References

- i.MX Linux Reference Manual: https://www.nxp.com/docs/en/reference-manual/RM00293.pdf -> chapters 6.1, 6.7
- EXPI-OS08A20 camera module and board user guide (UG10244) — https://docs.nxp.com/bundle/UG10244/page/topics/module_and_board.html

- libcamera documentation — https://libcamera.org
- GStreamer documentation — https://gstreamer.freedesktop.org

20 Ethernet

20.1 Introduction

This application note describes the Ethernet networking subsystem of the MR-NavQ95 vehicle computer (based on the NXP i.MX 95 applications processor) and shows how to bring up and use its wired network interfaces on the Linux (Cortex-A55) side, including the T1 Switch (SJA1110) expansion board.

It covers the available Ethernet ports and their mapping to Linux interfaces, the T1 Switch expansion board and T1 adapter, detecting the SJA1110 switch, VLAN configuration and interface partitioning between the Cortex-A55 (Linux) and Cortex-M7 (PX4/NuttX) domains, and the known issues and limitations.

It is intended for engineers integrating MR-NavQ95 into a networked robotics application, whether the board is connected over standard Ethernet or over the automotive 100BASE-T1 / 1000BASE-T1 ports provided by the SJA1110 switch.

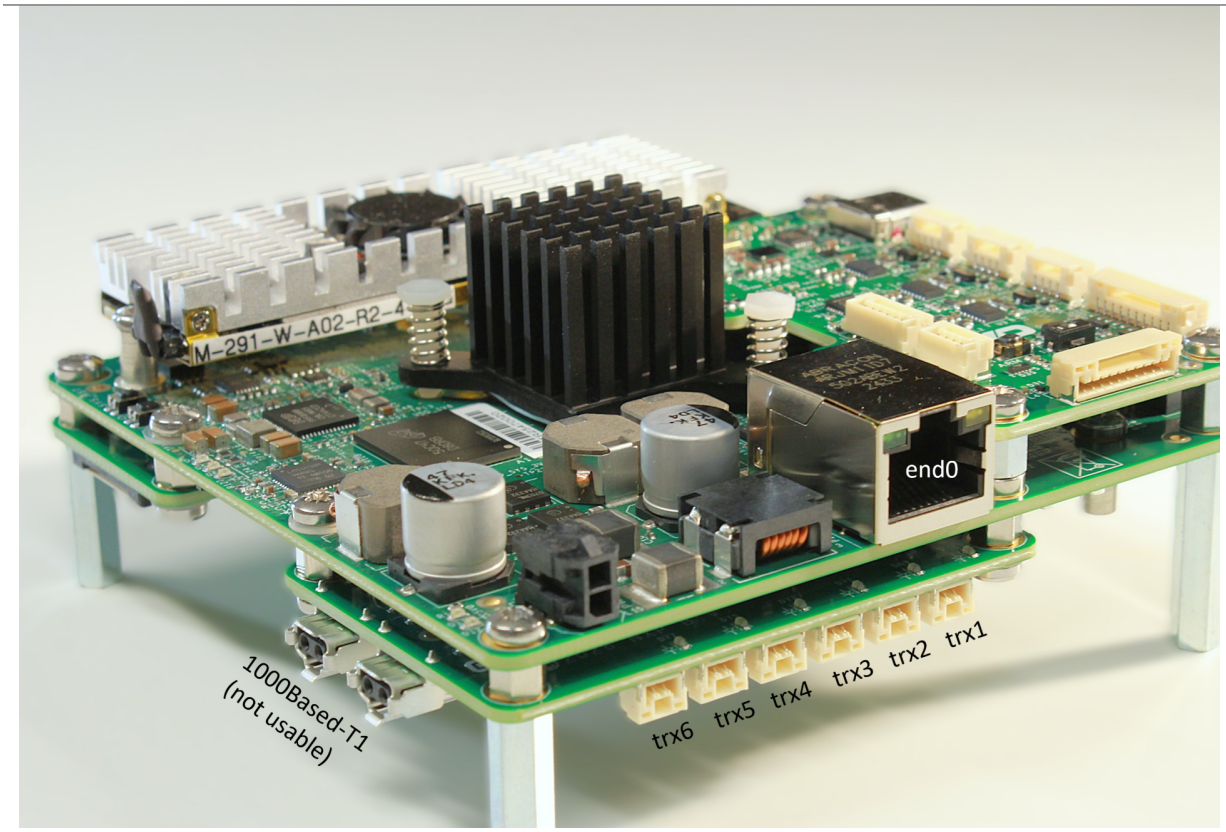
20.2 1. Ethernet subsystem architecture

The MR-NavQ95 board exposes the following Ethernet ports:

- Standard Ethernet 1 Gbps — RJ45 connector.
- XGMII-based networking expansion boards:
  - MR-NAVQ95E-T1S: T1 Switch utilizing the NXP SJA1110 — 6× 100BASE-T1 + 2× 1000BASE-T1 (1000BASE-T1 ports not currently usable).

1.1 Ports mapping to Linux interfaces

- 1 Gbps RJ45 port mapped to `end0`.
- 6× 100BASE-T1 ports mapped to `trx1` .. `trx6`.



1.2 T1 adapter

The T1 adapter is used to connect a 100BASE-T1 port to standard RJ45 equipment:

- T1 connector (JST-2 wire) — to be connected to `trx1` .. `trx6` .
- Power connector (JST-3 wire) — not used.
- Micro-USB connector — used for power.



The T1 adapter is connected to a T1 port and exposes an RJ45 connector to plug into other devices.

20.3 2. Using Ethernet via the SJA1110 interfaces

2.1 Detecting the SJA1110 switch

When the T1S (SJA1110 switch) expansion card is installed, the following output can be observed during Linux boot:

```
root@imx95-navq:/home/user# dmesg | grep -i sja
[   14.944635] sja1105 spi0.1: Probed switch chip: SJA1110A
[   15.467874] sja1105 spi0.1: configuring for fixed/2500base-x link mode
[   15.521648] sja1105 spi0.1: Link is Up - 2.5Gbps/Full - flow control off
[   15.534243] sja1105 spi0.1 trx1 (uninitialized): PHY [spi0.1-base-t1:01] driver [NXP
C45 TJA1103] (irq=POLL)
[   15.546102] sja1105 spi0.1 trx2 (uninitialized): PHY [spi0.1-base-t1:02] driver [NXP
C45 TJA1103] (irq=POLL)
[   15.559029] sja1105 spi0.1 trx3 (uninitialized): PHY [spi0.1-base-t1:03] driver [NXP
C45 TJA1103] (irq=POLL)
[   15.573131] sja1105 spi0.1 trx4 (uninitialized): PHY [spi0.1-base-t1:04] driver [NXP
C45 TJA1103] (irq=POLL)
[   15.586113] sja1105 spi0.1 trx5 (uninitialized): PHY [spi0.1-base-t1:05] driver [NXP
C45 TJA1103] (irq=POLL)
[   15.601885] sja1105 spi0.1 trx6 (uninitialized): PHY [spi0.1-base-t1:06] driver [NXP
C45 TJA1103] (irq=POLL)
```

Using a T1 adapter is required for connecting to other equipment, in the absence of a T1-compatible device.

2.2 Preventing NetworkManager from managing the T1 interfaces

NetworkManager should be disabled and/or the interfaces associated with the SJA1110 should not be managed by NetworkManager, which could cause IP settings to be overridden. This is achieved by creating `/etc/NetworkManager/conf.d/99-trx-unmanaged.conf` with the following content:

```
[device-trx-unmanaged]
match-device=interface-name:trx1;interface-name:trx2;interface-name:trx3;interface-
name:trx4;interface-name:trx5;interface-name:trx6
managed=0
```

Then restart NetworkManager:

```
sudo systemctl restart NetworkManager
```

2.3 Usage scenarios

The interfaces of the SJA1110 can be used in two different scenarios:

- No VLAN configuration (default):
  - Each `trx1` .. `trx6` can be used as a standalone interface.
  - IP addresses can be assigned individually.
  - Basic tests with ping traffic can be performed.
- VLAN configuration: each interface can belong to a different VLAN.

20.4 3. VLAN membership and A55/M7 interface partitioning

This section demonstrates VLAN membership and interface partitioning between the Cortex-A55 (Linux) and the Cortex-M7 (PX4/NuttX) domains.

3.1 Create VLANs and assign ports

The example bash script below (must be executed as `root`) creates VLANs 10 and 20, assigning `trx1`, `trx2`, `trx3` to VLAN 10 and `trx4`, `trx5`, `trx6` to VLAN 20:

```
#!/bin/bash

# 1. Create bridge with VLAN filtering
ip link add name br0 type bridge
ip link set br0 type bridge vlan_filtering 1
ip link set br0 up

# 2. Attach all ports to the bridge
for i in trx1 trx2 trx3 trx4 trx5 trx6; do
    ip link set $i master br0
    ip link set $i up
done

# 3. Remove default VLAN 1 from all ports and br0
bridge vlan del dev br0 vid 1 self

for i in trx1 trx2 trx3 trx4 trx5 trx6; do
    bridge vlan del dev $i vid 1
done

# 4. Assign trx1, trx2, trx3 to VLAN 10 (untagged access ports)
for i in trx1 trx2 trx3; do
    bridge vlan add dev $i vid 10 pvid untagged
done

# 5. Assign trx4, trx5, trx6 to VLAN 20 (untagged access ports)
for i in trx4 trx5 trx6; do
    bridge vlan add dev $i vid 20 pvid untagged
done

# 6. Add both VLANs to the bridge (CPU port)
bridge vlan add dev br0 vid 10 self
bridge vlan add dev br0 vid 20 self
```

Verify the result:

```
root@imx95-navq:/home/user# bridge vlan show
port              vlan-id
trx1              10 PVID Egress Untagged
trx2              10 PVID Egress Untagged
trx3              10 PVID Egress Untagged
trx4              20 PVID Egress Untagged
trx5              20 PVID Egress Untagged
trx6              20 PVID Egress Untagged
br0               10
                  20
```

Devices connected to ports in VLAN 10 can communicate with each other, and devices connected to ports in VLAN 20 can communicate with each other. Traffic between VLAN 10 and VLAN 20 is isolated.

3.2 Assign IP addresses and the M7 virtual interface

To perform basic traffic tests involving Linux, bridge interfaces for VLANs 10 and 20 must be created and IP addresses assigned. The script below must be executed as `root`:

```
#!/bin/bash

# Step 1 – Create VLAN interfaces on the bridge
ip link add link br0 name br0.10 type vlan id 10
ip link add link br0 name br0.20 type vlan id 20
ip link set br0.10 up
ip link set br0.20 up

# Step 2 – Add IP addresses to interfaces
ip addr add 192.168.10.1/24 dev br0.10
ip addr add 192.168.20.1/24 dev br0.20

# Step 3 – Assign M7 virtual ethernet to a VLAN (20)
ip link set dev end1 vf 0 trust on
ip link set dev end1 vf 1 trust on
ip link set end1 vf 1 vlan 20
```

3.3 Basic connectivity test

To test basic connectivity/traffic, connect devices to ports assigned to VLAN 10 and VLAN 20, assign IP addresses, and run a ping test.

Assuming the following connected devices:

- Device connected to `trx1` (VLAN 10) with IP address 192.168.10.100.
- Device connected to `trx6` (VLAN 20) with IP address 192.168.20.100.

From the Linux host, devices from both VLANs are accessible as long as the corresponding `br0.X` interfaces are UP and the configured IP settings are correct.

3.4 Using Ethernet from the M7 (PX4/NuttX)

Using Ethernet from the M7 running PX4/NuttX requires accessing the NSH console of PX4 and setting an IP address for the `eth0` interface. The default IP address set by NuttX is 10.0.0.2/24.

```
root@imx95-navq:/home/user# picocom /dev/ttynsh
nsh>
nsh> ifconfig eth0 192.168.20.200 netmask 255.255.255.0
nsh> ifconfig
...
eth0    Link encap:Ethernet HWaddr 02:04:9f:00:11:00 at RUNNING mtu 576
        inet addr:192.168.20.200 DRaddr:192.168.20.1 Mask:255.255.255.0

nsh>
nsh> ping 192.168.20.100
PING 192.168.20.100 56 bytes of data
56 bytes from 192.168.20.100: icmp_seq=0 time=2.0 ms
56 bytes from 192.168.20.100: icmp_seq=1 time=1.0 ms
...
56 bytes from 192.168.20.100: icmp_seq=9 time=2.0 ms
10 packets transmitted, 10 received, 0% packet loss, time 10010 ms
rtt min/avg/max/mdev = 1.000/1.300/2.000/0.458 ms
nsh>
```

Modifying the VLAN membership for the M7 and assigning it to VLAN 10, followed by new IP settings in NSH, allows pinging the device connected to the `trx1` interface with IP 192.168.10.100. Modify step 3 of the previous script as follows:

```
# Step 3 – Assign M7 virtual ethernet to a VLAN (10)
ip link set dev end1 vf 0 trust on
ip link set dev end1 vf 1 trust on
ip link set end1 vf 1 vlan 10
```

20.5 4. Known issues and limitations

- Traffic from M7 (PX4) to A55 (Linux) is not currently possible; however, both the M7 and the A55 can communicate (ping) an external device connected over T1 Ethernet.
- Low throughput when using a Windows PC connected to the MR-NavQ95 board. Disable the Energy Efficient Ethernet (EEE) feature for the NIC (Windows PowerShell command):

  ```
  Set-NetAdapterAdvancedProperty -Name "Ethernet" -DisplayName "Energy Efficient
  Ethernet" -DisplayValue "Off"
  ```

- The SJA1110 `end1` interface is impacted by the DSA (Distributed Switch Architecture) conduit, and TCP checksum offload is not applied across the `end1 -> SJA switch -> trx*` forwarding path. As a result, frames leave with an unfinished checksum, arrive corrupted at the external device, and eventually get dropped. Force software checksumming:

  ```
  sudo ethtool -K end1 tx off tso off gso off gro off
  ```

20.6 References

- NXP SJA1110 automotive Ethernet switch — https://www.nxp.com/products/SJA1110

21 Cellular Modem

This application note explains how to connect and use a WWAN cellular modem on the MR-NAVQ95 board's M.2 Key-B slot. It first covers the hardware architecture and general considerations that apply to any M.2 Key-B modem, then walks through a bring-up example using the Quectel RM520N-EU (5G Sub-6 module).

21.1 1. Hardware architecture

MR-NAVQ95 exposes two M.2 sockets: a Key-M slot (NPU accelerator cards) and a Key-B slot dedicated to WWAN modems. The Key-B slot is located on the back of the main board and is electrically wired as a single-lane PCIe interface (PCIe x1) off the i.MX 95.

A nano-SIM slot is available on the board for the WWAN module.

1.1 What to consider when selecting an M.2 Key-B modem

1. Interface type. The slot only wires a PCIe lane and the usual M.2 sideband control signals; it does not carry USB.

   > Attention! On MR-NAVQ95, the M.2 Key-B connector's USB signal pins are not wired to the SoC. Some WWAN modules — including the Quectel RM520N-EU used as the example below — ship from the factory configured to use their USB interface by default. Such a module will not be usable in this slot until it has been explicitly reconfigured for PCIe operation (see section 2.1 for the RM520N-EU procedure). Always check and, if needed, reconfigure the module's default interface mode before installing it on MR-NAVQ95.

2. Form factor. The socket is a multi-mount M.2 socket for Key-B (or B+M) cards — it is not limited to a single card length. Check the schematic and confirm the specific module's length/mounting-hole position is supported by the socket footprint before selecting it.

21.2 2. Example: Quectel RM520N-EU

The RM520N-EU is a 5G Sub-6 M.2 Key-B module that supports both USB and PCIe interfaces, shipping configured for USB mode by default. Once switched to PCIe mode (see section 2.1) and installed, it is supported out of the box on the NXP Robotics Linux image for MR-NAVQ95: the module is detected automatically at boot and exposed to userspace via ModemManager, without any additional configuration on the board itself.

2.1 One-time module configuration: switching to PCIe mode

Before the module can be used in the MR-NAVQ95 M.2 Key-B slot, it must be switched from its default USB mode to PCIe mode. This is a one-time configuration step, done outside of MR-NAVQ95 on a Windows or Linux PC, and requires:

- A USB-to-M.2 (Key-B) adapter, to connect the module to the PC over USB.
- The Quectel USB driver for the module, installed on the PC. Download the driver from the Quectel product page.
- A serial terminal application (e.g. PuTTY, Tera Term) to send AT commands to the module.

Step 1 — Install the Quectel driver. Install the Quectel USB driver for the RM520N-EU on the PC. Once the driver is installed, the module enumerates as a set of serial ports.

Step 2 — Connect the modem to the PC. Insert the modem into the USB-to-M.2 adapter, then insert the adapter into a USB port of the PC.

Step 3 — Open a serial session to the module's AT command port at 115200 baud, 8N1, no flow control.

Step 4 — Query the current interface mode (optional, to confirm the module is currently in USB mode):

```
AT+QCFG="data_interface"
```

Default response: `+QCFG: "data_interface",0,0` (USB mode).

Step 5 — Switch the module to PCIe mode:

```
AT+QCFG="data_interface",1,0
```

1 = network data via PCIe; 0 = diagnostic port stays on USB

Once switched, the module retains the PCIe configuration in its own non-volatile storage, so this step only needs to be done once per module, before it is installed in the MR-NAVQ95 M.2 Key-B slot.

2.2 Verifying the modem is detected

- Power off the board.
- Insert the modem into the M.2 Key-B connector (J16).
- Connect the antennas to the modem.
- Insert a nano-SIM into the SIM slot (J5).
- Power on the board.

The modem should be visible on the PCIe bus and reported by ModemManager:

```
user@imx95-navq:~# lspci -d 17cb:0308
0001:01:00.0 Unassigned class [ff00]: Qualcomm Technologies, Inc Device 0308

user@imx95-navq:~# mmcli -L
    /org/freedesktop/ModemManager1/Modem/0 [generic] RM520N-EU
```

Note: It may take a few seconds for the ModemManager to detect the modem.

Disable the SIM PIN (one-time per SIM):

```
sudo mmcli -i 0 --pin=<PIN> --disable-pin
```

2.3 Connecting to the network

Once the modem is detected, use ModemManager's `mmcli` to check status and bring up a data connection:

```
mmcli -m 0
mmcli -m 0 --simple-connect="apn=<your-apn>"
successfully connected the modem
```

Depending on the APN and network configuration, the resulting network interface (`wwan0`) can then be configured with the carrier-provided IP settings, or managed automatically by NetworkManager.

2.4 Propagate the settings to the network adapter (wwan0)

```
user@imx95-navq:~$ sudo nmcli connection add type gsm ifname "*" con-name quectel apn
<your-apn>
Connection 'quectel' (426eda0c-d93f-4746-aca3-8caf3481c24c) successfully added.

user@imx95-navq:~$ nmcli con show
NAME                UUID                                  TYPE      DEVICE
quectel             426eda0c-d93f-4746-aca3-8caf3481c24c  gsm
       wwan0mbim0
lo                  86eac6be-dcd5-46c2-aace-9f70549148db
loopback  lo
Wired connection 1  ce584b2c-9584-33b3-a787-a9863fa15037  ethernet  --
Wired connection 2  0694ee2b-c10a-34ff-854b-d7e2014f4e9e  ethernet  --
wired               f42fc16a-9c44-43dd-b15d-fb7243d80177  ethernet  --
```

2.5 Test the internet connection

```
user@imx95-navq:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: end0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default
qlen 1000
    link/ether 6a:8c:01:b4:6b:81 brd ff:ff:ff:ff:ff:ff
    altname enP2p0s0f0
3: end1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen
1000
    link/ether 5a:88:b5:a5:f8:c7 brd ff:ff:ff:ff:ff:ff
    altname enP2p0s16f0
4: wwan0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
qlen 1000
    link/[519]
    inet 10.131.84.97/30 brd 10.131.84.99 scope global noprefixroute wwan0
       valid_lft forever preferred_lft forever

user@imx95-navq:~$ sudo ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 time=54.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=62.1 ms
```

22 eIQ Neutron NPU

This guide provides the steps to set up the eIQ Neutron SDK and deploy a model on an NXP SoC with an integrated eIQ Neutron NPU. Unlike a discrete accelerator module, the Neutron NPU is embedded directly in the SoC (for example, i.MX 95), so there is no add-on hardware to install — the workflow centers on preparing a model on the host machine and deploying it to the target board.

The examples in this guide use the MR-NavQ95 board, which is based on the i.MX 95 SoC and its integrated Neutron NPU.

This guide covers:

- Prerequisites and host-side setup
- Obtaining the eIQ Neutron SDK (SDK download or pip install)
- Converting a model for the Neutron NPU with the `neutron-converter`
- Deploying and benchmarking the model on the target

22.1 Prerequisites

Hardware:

- A target board with an integrated Neutron NPU. This guide uses the MR-NavQ95 board (based on the i.MX 95 SoC) as the example.
- Internet access on the host machine to download and install the SDK
- 4 GB RAM or more, and a multi-core CPU on the host (recommended for reasonable conversion times)

Software:

- Host development machine running Linux or Windows
- Python 3.8 or newer, with `pip`
- NXP Yocto BSP flashed to the target board, matching the Neutron SDK version (see the compatibility tables in the Neutron SDK User Guide)

22.2 Obtain the eIQ Neutron SDK

The eIQ Neutron SDK can be obtained in one of two ways. Use whichever fits the workflow.

Option A — Download the SDK archive from nxp.com

1. Download the eIQ Neutron SDK from the NXP eIQ AI Development Environment page:

   https://www.nxp.com/design/design-center/software/eiq-ai-development-environment/eiq-toolkit-for-end-to-end-model-development-and-deployment:EIQ-TOOLKIT

2. Extract the downloaded SDK archive. The archive includes a `docs/` folder containing the full documentation set. The most relevant document for this workflow is the Neutron Converter User Guide (`docs/NeutronConverterUserGuide.md`), which details the `neutron-converter` tool, its conversion stages, and all command-line options.

> Tip: For the best experience with tables and links, read the `.md` files in a Markdown viewer such as the built-in preview in Visual Studio Code (Ctrl+Shift+V), or directly on the Git server where they render automatically.

Option B — Install via pip (alternative)

Alternatively, instead of downloading the SDK archive from nxp.com, the eIQ Neutron SDK can be installed directly from the NXP eIQ package repository with `pip` (see Host-Side Setup). This is convenient for CI or scripted host environments. Note that the documentation set shipped in the archive's `docs/` folder is only available with Option A; when installing via pip, refer to the online documentation for the same references.

22.3 Host-Side Setup

It is good practice to work inside a Python virtual environment.

Linux:

```
python3 -m venv neutron-env
source neutron-env/bin/activate
```

Windows (PowerShell):

```
python -m venv neutron-env
.\neutron-env\Scripts\Activate.ps1
```

> Windows note: If activation is blocked by the execution policy, run PowerShell as Administrator once and execute: `Set-ExecutionPolicy -Scope CurrentUser RemoteSigned`

If Option B above was chosen (or to make the converter available on the `PATH` regardless of how the SDK was obtained), install the eIQ Neutron SDK from the NXP eIQ package repository:

```
pip install eiq-neutron-sdk --index-url https://eiq.nxp.com/repository/
```

Verify the converter is available:

```
neutron-converter --help
```

> If the command is not found, ensure the virtual environment is activated and that the Python `bin` (Linux) or `Scripts` (Windows) directory is on the `PATH`.

22.4 Convert a Model for the Neutron NPU

The `neutron-converter` consumes a quantized (INT8) TFLite model and produces a custom TFLite model in which the operators supported by the NPU are grouped into `NeutronGraph` custom operators for on-NPU execution, while unsupported operators remain on the CPU.

1. Check the available targets:

```
neutron-converter --show-targets
```

2. Convert the quantized TFLite model for the target SoC. For the MR-NavQ95 board (i.MX 95), use the `imx95` target:

```
neutron-converter \
  --input my_model_quantized.tflite \
  --output my_model_converted.tflite \
  --target imx95
```

At minimum, the converter requires the target SoC and a quantized TFLite input model.

> Note: For full details on the conversion stages (optimize, extract, generate), minimum system requirements, and the complete list of command-line options, refer to the public eIQ Neutron SDK documentation.

22.5 Deploy and Benchmark on the Target

1. Copy the converted model (`my_model_converted.tflite`) to the MR-NavQ95 board.
2. Run the TensorFlow Lite `benchmark_model` application with the Neutron delegate. This utility is part of the TensorFlow package included in the NXP Yocto BSP:

```
/usr/bin/tensorflow-lite-2.19.0/examples/benchmark_model \
  --graph=my_model_converted.tflite \
  --external_delegate_path=/usr/lib/libneutron_delegate.so \
  --enable_op_profiling=true \
  --num_threads=5
```

3. In the output, operations executed via the `NeutronDelegate` node correspond to workloads offloaded to the Neutron NPU; all remaining nodes run on the CPU. A high percentage under the `NeutronDelegate` node means most of the computation is being offloaded to the NPU.

22.6 Keep the Runtime and Converter in Sync

Always keep the Neutron driver, firmware, and converter from the same Neutron SDK version. If the runtime binaries and the converted model are not aligned, a microcode version mismatch warning is printed and operation may be unreliable:

```
Microcode version mismatch! Neutron Driver is 3.1.0-0X359f358d but model converted with
3.0.1-0Xd5d7918e. Runtime (driver, firmware) should be aligned with converter
```

Refer to the SDK/BSP and SDK/MCUXpresso version compatibility tables in the Neutron SDK User Guide (`docs/NeutronSDKUserGuide.md`) to confirm the correct pairing, and to the same guide for the runtime update procedure if an upgrade is needed for the on-target driver, firmware, and delegate.

22.7 References and Downloads

Online references:

1. eIQ Neutron SDK documentation (online): https://eiq.nxp.com/learning-hub/tools/neutronSdk/index.html — full reference for the `neutron-converter` tool, conversion stages, system requirements, deployment workflow, and version compatibility tables.
2. eIQ AI Development Environment (SDK download): https://www.nxp.com/design/design-center/software/eiq-ai-development-environment/eiq-toolkit-for-end-to-end-model-development-and-deployment:EIQ-TOOLKIT

The following documents are also located in the `docs/` folder of the eIQ Neutron SDK archive (Option A):

1. Quick Start Guide: `docs/QuickStartGuide.md` — how to navigate the docs and deploy a first model.
2. Neutron SDK User Guide: `docs/NeutronSDKUserGuide.md` — end-to-end deployment workflow, runtime update procedure, and version compatibility tables.
3. Neutron Converter User Guide: `docs/NeutronConverterUserGuide.md` — full reference for the `neutron-converter` tool.

23 ARA-240 DNPU

This guide provides the steps to setup and run the ARA-240 DNPU with MR-NavQ95. It provides the steps for:

- Hardware setup
- Setting up the ARA SDK and eIQ AAF Connector, which is a REST-based server for LLM inference
- Downloading and preparing models (CNNs and LLMs)

23.1 Prerequisites

Hardware:

- MR-NavQ95 board
- ARA-240 M.2 Module
- Fan port to USB adapter
- 64 GB or greater SD card recommended
- Internet access for the board

Software:

- `imx-6.18.20-2.0.0-navq` image

23.2 Hardware Setup

1. Connect ARA-240 to top M.2 slot on MR-NavQ95.
2. The MR-NavQ95 does not have a fan port available for the ARA-240 fans. Currently, recommendation is to use a USB adapter and plug the fans in a separate USB port. Alternatively, one of the CAN ports in the CAN expansion board can be used with an adapter.

23.3 Software setup

1. Flash the resulted image to the SD card using `uuu` normally and boot.
2. After login, check that the ARA has been successfully initialized by the image by running the following command:

   ```
   sudo chip_info.sh
   ```

   Output should contain a table. Note the firmware version (raw) value.
3. If the firmware version (raw) is not `131072` or higher (like in the above screenshot where it is `65794`), run the following commands to flash the new firmware:

   ```
   sudo program_flash.sh
   reboot
   ```

4. Run the following command to complete installation of the ARA module and resize the rootfs partition to the entire size of the SD card:

```
sudo /usr/share/rt-sdk-ara240/install.sh
```

23.4 Run CNNs (yolov8 or yolox)

> Note: Internet access is required for the next step.

1. Run the following command to download the yolov8 or yolox models locally:

```
sudo fetch_models --repo-id nxp/YOLOv8
sudo fetch_models --repo-id nxp/YOLOX-Detection-Ara240
```

2. Use the built-in performance script to check the inference performance of a downloaded modelL

```
sudo run_model_perf.sh
```

3. For a more complex example, follow the readme at https://github.com/nxp-imx-support/ara2-vision-examples to install and run the vision examples.

23.5 Run LLMs

> Note: Internet access is required for the next steps.

1. Run the following command to install the eIQ AAF Connector:

```
sudo aaf-connector
```

2. Run the following command to download the Qwen2.5-7B model from NXP Huggingface. Available models are listed at https://huggingface.co/nxp

```
sudo fetch_models --repo-id nxp/Qwen2.5-7B-Instruct-Ara240
```

> Note: Internet access is no longer required from this point on.

3. Applications use the `/var/run/proxy.sock` file to communicate with the lower-level ARA-240 runtime stack. Use the following command to change permissions, allowing user-space applications to read and write to the file:

```
sudo chmod a+rw /var/run/proxy.sock
```

4. Start the eIQ AAF Connector:

```
aaf-connector &
```

5. Load the downloaded model with the following command:

```
curl -X 'POST' 'http://127.0.0.1:8000/v1/models' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{"name": "Qwen2.5-7B-Instruct"}'
```

6. Test that everything is working with the following inference request:

```
curl -H 'Content-Type: application/json' \
  -d '{ "model":"Qwen2.5-7B-Instruct","messages":[{"role":"user", "content":"Who are
you?"}]}' \
  -X POST 0.0.0.0:8000/v1/chat/completions
```

23.6 Available Models

NXP publishes a curated collection of models precompiled for the ARA-240 DNPU (CNNs such as YOLOv8/YOLOX and LLMs such as Qwen2.5) on Hugging Face. Use it to discover which models are available and how to fetch them with `fetch_models --repo-id nxp/<model>`:

- ARA240 model collection: https://huggingface.co/collections/nxp/ara240-models

23.7 Further Documentation

1. Ara Software Release Notes (RN00459, Rev 2.1): ara-software-release-notes-rn00459-2.1.pdf
2. Ara Runtime SDK documentation: https://github.com/nxp-imx/rt-sdk-ara2/tree/main#-getting-started
3. Ara Vision Examples: https://github.com/nxp-imx-support/ara2-vision-examples
4. Ara SDK for Model Compilation: https://www.nxp.com/design/design-center/software/embedded-software/ara-software-development-kit:ARA-SDK
5. eIQ AAF Connector: https://github.com/nxp-imx-support/eiq-aaf-connector

24 Remote ID Transmitter

Remote ID Transmitter is a built-in Linux application used to continuously broadcast identification and flight data. Remote ID broadcast is a standard in its own right and is required on most drones.

For more information, see the ARK Electronics RemoteIDTransmitter project.

24.1 Usage

The Remote ID Transmitter (rid-transmitter) is preinstalled on the Linux image. It must be connected to a PX4 instance. This document uses PX4 SITL (simulated); the same steps may apply to PX4 running on the Cortex-M7 core. The broadcast signal can be received using a phone with a mobile app such as Drone Scanner or with BLE sniffers.

1. PX4 SITL

In this setup, PX4 runs in simulation on a separate Linux PC (x86 architecture). The same steps may apply when installing directly on the board. The board and the PC must be discoverable to each other over the network.

a. Installation

```
sudo apt update && sudo apt install -y git
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
cd PX4-Autopilot
bash ./Tools/setup/ubuntu.sh --no-nuttx
PX4_HOME_LAT=44.4351428 PX4_HOME_LON=26.0537122 PX4_HOME_ALT=80 make px4_sitl sihsim_quadx
```

If errors are encountered: OSRF repo “NOSPLIT / no longer signed” -> switch to HTTPS & Other dependencies

```
sudo rm -f /etc/apt/sources.list.d/gazebo-stable.list
sudo apt clean
sudo rm -rf /var/lib/apt/lists/*osrfoundation* /var/lib/apt/lists/partial/*
sudo wget https://packages.osrfoundation.org/gazebo.gpg -O /usr/share/keyrings/pkgs-osrf-
archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/pkgs-osrf-
archive-keyring.gpg] https://packages.osrfoundation.org/gazebo/ubuntu-stable $(lsb_release
-cs) main" | sudo tee /etc/apt/sources.list.d/gazebo-stable.list

sudo apt install -y \
  gz-jetty libunwind-dev cppzmq-dev \
  libopencv-dev libeigen3-dev protobuf-compiler \
  libgstreamer-plugins-base1.0-dev gstreamer1.0-plugins-base \
  gstreamer1.0-plugins-good gstreamer1.0-plugins-bad \
  gstreamer1.0-plugins-ugly gstreamer1.0-libav \
  libimage-exiftool-perl libxml2-utils dmidecode pkg-config bc

sudo apt update
```

b. Running

The coordinates can be changed as needed; those below are the coordinates of the Bucharest NXP site.

```
rm -rf build/px4_sitl_default
PX4_HOME_LAT=44.4351428 PX4_HOME_LON=26.0537122 PX4_HOME_ALT=80 make px4_sitl sihsim_quadx
```

MAVLink setup, required each time PX4 starts:

```
pxh> mavlink start -x -u 14555 -r 100000 -m onboard -o 14553 -t <BOARD-IP>
pxh> mavlink stream -r 5 -s OPEN_DRONE_ID_LOCATION -u 14555
pxh> mavlink stream -r 5 -s OPEN_DRONE_ID_SYSTEM   -u 14555
pxh> commander takeoff
```

2. RID-Transmitter

The commands below must be run as the root user (the Bluetooth and apt commands require it).

a. Setup

```
mkdir -p /opt/ark/share/rid-transmitter/
nano /opt/ark/share/rid-transmitter/config.toml
```

Text:

```
bluetooth_device  = "hci0"
connection_url    = "udpin://0.0.0.0:14553"
manufacturer_code = "MFR1"
serial_number     = "950009500NXP" # any code is acceptable, provided it has the same
length
```

b. Running

```
modprobe btnxpuart
hciconfig hci0 up
rid-transmitter
```

Expected startup output:

```
UAS Serial Number: MFR1...
Waiting for MAVSDK connection: udpin://0.0.0.0:14553
Connected to autopilot
```

The broadcast uses BLE Coded PHY. If a phone is used to listen for the broadcast, it might capture only the first message, depending on its Bluetooth capabilities, as not all phones support receiving on BLE Coded PHY.

In that case, the source code must be modified. See the next section.

3. Compiling rid-transmitter to broadcast on another BLE PHY

The steps below describe how to compile the application modified to broadcast on 1M PHY, which all Bluetooth devices support.

The steps below are for compiling on the board. Compilation can also be performed with the Yocto toolchain on an external computer; this document does not cover cross compilation.

a. MAVSDK

```
cd /home/user/
git clone --recursive https://github.com/mavlink/MAVSDK.git -b v3.12.0
cd MAVSDK
cmake -Bbuild/default -H. -DCMAKE_BUILD_TYPE=Release \
      -DBUILD_MAVSDK_SERVER=OFF \
      -DCMAKE_INSTALL_PREFIX=/usr/local/MAVSDK/install
cmake --build build/default -j$(nproc)
cmake --build build/default --target install
```

b. BlueZ

```
cd /home/user/
wget https://mirrors.edge.kernel.org/pub/linux/bluetooth/bluez-5.72.tar.xz
tar xf bluez-5.72.tar.xz
mkdir -p /usr/include/bluetooth
cp bluez-5.72/lib/*.h /usr/include/bluetooth/

cd ~/bluez-5.72
echo '#define VERSION "5.72"' > config.h
gcc -c -fPIC -O2 -I. -Ilib lib/bluetooth.c lib/hci.c lib/sdp.c lib/uuid.c
ar rcs /usr/lib/libbluetooth.a bluetooth.o hci.o sdp.o uuid.o
mkdir -p /usr/local/lib/pkgconfig
nano /usr/local/lib/pkgconfig/bluez.pc
```

Text:

```
prefix=/usr
exec_prefix=${prefix}
libdir=${prefix}/lib
includedir=${prefix}/include

Name: BlueZ
Description: Bluetooth protocol stack for Linux
Version: 5.72
Libs: -L${libdir} -lbluetooth
Cflags: -I${includedir}
```

After saving the file:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH
pkg-config --modversion bluez
```

c. RemoteID Transmitter

```
cd /home/user
git clone --recursive https://github.com/ARK-Electronics/RemoteIDTransmitter.git
cd RemoteIDTransmitter
```

Modify the sources:

1. First modification:

File: `src/Bluetooth/Bluetooth.cpp`, `le_set_extended_advertising_parameters()`

The struct initialiser sets LE 1M, but two lines afterwards override it:

```
buf[20] = 0x03; // Primary_Advertising_PHY: 3 = LE Coded
buf[22] = 0x03; // Secondary_Advertising_PHY: 3 = LE Coded
```

Change both to `0x01`:

```
sed -i 's/buf\[20\] = 0x03;/buf[20] = 0x01;/; s/buf\[22\] = 0x03;/buf[22] = 0x01;/' \
  src/Bluetooth/Bluetooth.cpp
```

1. Second modification:

File: `src/Transmitter/Transmitter.cpp`, `run_state_machine()` The transmit state machine alternates the advertising mode every 200 ms:

```
uint64_t loop_rate_ms = 200;
...
_toggle_legacy = !_toggle_legacy;
if (_toggle_legacy) {
    _bluetooth->enable_legacy_advertising();
} else {
    _bluetooth->enable_le_extended_advertising();
}
```

Change `_toggle_legacy = !_toggle_legacy;` to `_toggle_legacy = true;`

Build:

```
cmake -Bbuild -H. -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_C_FLAGS="-Wno-error=stringop-truncation"
cmake --build build -j$(nproc)
```

The new rid-transmitter can now be executed (ensure the config.toml file is present):

```
setcap 'cap_net_raw,cap_net_admin+eip' build/rid-transmitter
./build/rid-transmitter --config /opt/ark/share/rid-transmitter/config.toml
```

25 Middleware

Robotics middleware on the MR-NavQ95: set up ROS2 and connect multi-node topologies across the robotics platforms.

- ROS2
  - ROS2 Setup on NavQ95
  - ROS2 Multinode on NavQ95

26 ROS2

Set up ROS2 on the MR-NavQ95 and connect multi-node topologies.

- ROS2 Setup on NavQ95
- ROS2 Multinode on NavQ95

27 ROS2 Setup on NavQ95

27.1 Overview

This guide covers the full ROS2 setup sequence on a NavQ95 platform, from base installation through environment configuration and additional packages. Setup is organized into three steps:

Step	Script	Purpose	Status
1	`install_ros2_base`	ROS2 base installation	Available
2	`install_ros2_env`	Environment setup and RMW configuration	Available
3	`install_ros2_pkgs`	NavQ95-specific ROS2 packages	Available

> Note: `imx-image-robotics` ships without ROS2 pre-installed. Installing ROS2 after deployment aligns with the upstream ROS2 installation model for Ubuntu-based systems.

27.2 Prerequisites

- `imx-image-robotics` deployed and booted on the NavQ95.
- The platform must have internet access so that apt can reach `packages.ros.org`.

27.3 Quick Install

> Note: Please execute the following sequence of steps to ensure full versions compatibility.

```
# Step 1 — ROS2 base installation (latest upstream version)
install_ros2_base

# Step 2 — Environment setup and RMW configuration (latest upstream version)
install_ros2_env -l

# Step 3 — NavQ95-specific ROS2 packages (latest upstream versions)
install_ros2_pkgs -l ros-jazzy-demo-nodes-cpp ros-jazzy-rosbag2

# Step 4 — NavQ95-specific ROS2 packages (pinned version)
install_ros2_pkgs px4_msgs
```

Installed versions:

- ROS2 base: `ros-jazzy-ros-base - latest available version`

- ROS2 environment: `ros-jazzy-rmw-zenoh-cpp - latest available version`
- ROS2 packages:
  - `ros-jazzy-demo-nodes-cpp - latest available version`
  - `ros-jazzy-rosbag2 - latest available version`
  - `px4_msgs - default fixed release/1.17`

Both `install_ros2_env` and `install_ros2_pkgs` support all of the following flags, while `install_ros2_base` only supports the `[-i] [-h]` flags. — run `<script_name> -h` for details:

```
<script_name> [-i] [-l] [-h]

  (no flags)  Non-interactive: run all steps unattended, installs fixed version(s)
  -i          Interactive: prompt [Y/n] before each step
  -l          Latest: install latest upstream packages (ignoring pinned ROS_FIXED_VERSION)
  -h          Show this help and exit

Flags can be combined, e.g.: <script_name> -i -l
```

27.4 Installation — ROS2 Base

Overview

`install_ros2_base` installs `ros-jazzy-ros-base` following the official ROS 2 Jazzy deb installation steps. It always installs the latest upstream version if ROS2 is not already installed — no version pinning or `apt-mark hold` is applied.

> Note: If `ros-jazzy-ros-base` is already installed, the script detects it automatically. If the installed version matches the latest, the install step is skipped. If it is older, the script prompts for an upgrade (auto-upgrade after 10 seconds).

It also installs `ros-dev-tools` and reports the active ROS2 distro, middleware, and installed package version.

Option A: Automated

Run on the platform:

```
install_ros2_base
```

Expected output on success:

```
ROS2 Jazzy base installer -- mode: non-interactive, version: latest upstream

=== Step 1/7: Set locale (UTF-8) ===
...
=== Step 7/7: Report installation ===
ROS_DISTRO = jazzy
middleware name    : rmw_fastrtps_cpp
ros-jazzy-ros-base installed version: <version>

SUCCESS. All steps completed.
NOTE: ROS2 is not yet active in this shell. Run: source /opt/ros/jazzy/setup.bash
```

On failure, if a shell command exits with a non-zero status (e.g. apt update fails due to no network), the script exits immediately and prints:

```
ERROR: Installation aborted at line <N>. Fix the issue above and re-run the script.
Tip: use -i for step-by-step control (prompt before each step).
```

Fix the reported issue and re-run. The script is idempotent — safe to re-run. Use `-i` to step through and skip already-completed steps:

```
install_ros2_base -i
```

Option B: Manual

Follow the official ROS 2 Jazzy installation guide: https://docs.ros.org/en/jazzy/Installation/Ubuntu-Install-Debs.html

Verification

After completing the base installation:

```
source /opt/ros/jazzy/setup.bash
printenv | grep ROS_DISTRO                          # -> ROS_DISTRO=jazzy
dpkg-query -W -f='${Version}\n' ros-jazzy-ros-base  # -> <version>
ros2 doctor --report | grep -i "middleware name"    # -> rmw_fastrtps_cpp (default)
```

Troubleshooting

No internet access — the platform requires internet access to reach `packages.ros.org`. Verify connectivity before running any script:

```
curl -I https://packages.ros.org
```

Re-running after a failure — all scripts are idempotent, safe to re-run after fixing the reported issue. Use `-i` to be prompted before each step:

```
install_ros2_base -i
# Answer [Y/n] before each step; press Enter to run, n to skip
```

27.5 Installation — ROS2 Environment

Overview

`install_ros2_env` configures the ROS2 Jazzy environment for the NavQ95 by:

1. Installing `ros-jazzy-rmw-zenoh-cpp` (Zenoh RMW middleware).
2. Generating a persistent ROS2 environment config file (`~/.ros2rc`) that sources ROS2 Jazzy and sets `RMW_IMPLEMENTATION=rmw_zenoh_cpp` .
3. Reporting the installed versions and active configuration.

Option A: Automated

Run on the platform:

```
install_ros2_env
```

The script installs the following NavQ95-validated fixed version of `ros-jazzy-rmw-zenoh-cpp` (arm64):

```
ros-jazzy-rmw-zenoh-cpp=0.2.9-1noble.20260612.051449
```

To install a specific build, override the fixed version before running:

```
export ZENOH_FIXED_VERSION=<version>
install_ros2_env
```

To install the latest upstream version instead, use the `-l` (latest) flag — it will ignore the pinned `ZENOH_FIXED_VERSION` :

```
install_ros2_env -l
```

The generated `~/.ros2rc` file activates ROS2 Jazzy and selects the Zenoh RMW:

```
# Activate ROS2 Jazzy
source /opt/ros/jazzy/setup.bash

# Use Zenoh as the RMW implementation
export RMW_IMPLEMENTATION=rmw_zenoh_cpp
```

To change the output path, override `ROS2RC_PATH` before running:

```
export ROS2RC_PATH=/custom/path/ros2rc
install_ros2_env
```

Expected output on success:

```
ROS2 Jazzy env configurator -- mode: non-interactive, version: fixed ros-jazzy-rmw-zenoh-
cpp (<version>)

=== Step 1/3: Install ros-jazzy-rmw-zenoh-cpp (fixed <version>) ===
...
ros-jazzy-rmw-zenoh-cpp set on hold.

=== Step 2/3: Generate ros2rc environment config ===
Generating /home/<user>/.ros2rc ...
ros2rc written to /home/<user>/.ros2rc

=== Step 3/3: Report installation ===
ROS_DISTRO            = jazzy
RMW_IMPLEMENTATION    = rmw_zenoh_cpp
ros-jazzy-rmw-zenoh-cpp installed version: <version>
ros2rc path           = /home/<user>/.ros2rc

SUCCESS. All steps completed.
To activate the ROS2 Zenoh environment, run:
  source /home/<user>/.ros2rc

To activate automatically on every login, add it to the shell startup script:
  echo "source /home/<user>/.ros2rc" >> ~/.bashrc
```

On failure, the script exits immediately and prints:

```
ERROR: Installation aborted at line <N>. Fix the issue above and re-run the script.
Tip: use -i for step-by-step control (prompt before each step).
```

Fix the reported issue and re-run. The script is idempotent — safe to re-run. Use `-i` to step through and skip already-completed steps:

```
install_ros2_env -i
```

Option B: Manual

Install the Zenoh RMW package manually:

```
sudo apt update
sudo apt install ros-jazzy-rmw-zenoh-cpp=<version>
sudo apt-mark hold ros-jazzy-rmw-zenoh-cpp
```

Then create `~/.ros2rc`:

```
cat > ~/.ros2rc <<'EOF'
# ros2rc -- NavQ95 ROS2 environment configuration
source /opt/ros/jazzy/setup.bash
export RMW_IMPLEMENTATION=rmw_zenoh_cpp
EOF
chmod 644 ~/.ros2rc
```

Verification

After completing the environment setup:

```
source ~/.ros2rc
printenv | grep RMW_IMPLEMENTATION                              # ->
RMW_IMPLEMENTATION=rmw_zenoh_cpp
dpkg-query -W -f='${Version}\n' ros-jazzy-rmw-zenoh-cpp        # -> <version>
```

Troubleshooting

`/opt/ros/jazzy/setup.bash` not found — the ROS2 base installation must be completed first:

```
install_ros2_base
```

Re-running after a failure — the script is idempotent, safe to re-run after fixing the reported issue. Use `-i` to be prompted before each step:

```
install_ros2_env -i
# Answer [Y/n] before each step; press Enter to run, n to skip
```

27.6 Installation — ROS2 Packages

Overview

`install_ros2_pkgs` installs a set of NavQ95-specific ROS2 packages on top of the ROS2 base and environment setup. Each package is installed at a NavQ95-validated fixed version and held to prevent unintended upgrades.

Packages installed:

```
ros-jazzy-demo-nodes-cpp      (0.33.11-1noble.20260612.180159)   # demo nodes (talker/
listener)
ros-jazzy-rosbag2             (0.26.11-1noble.20260612.202546)   # record and play back
topics
px4_msgs                      (release/1.17)                     # PX4 uORB message
definitions
```

> Prerequisite: ROS2 base must be installed first (`install_ros2_base`). Some packages may also require the ROS2 environment setup (`install_ros2_env`).

> Note: `px4_msgs` build scope. `px4_msgs` is built from source on the board. By default, only the message definitions required by the T1 usecase (`DistanceSensor.msg` and `Sensor*.msg`) are compiled, instead of the full set of 200+ messages. This reduces build time by roughly 90%, down to about 3 minutes (versus 30-60 minutes for the full set). Use the `--all-px4-msgs` flag to build every message definition instead.

Install

Run on the platform:

```
install_ros2_pkgs
```

Each package is installed at its fixed version and held via `apt-mark hold` to prevent unintended upgrades.

To install a specific build, override the fixed version before running (one variable per package, `ROS2PKG_<ID>_VERSION`):

```
export ROS2PKG_<ID>_VERSION=<version>
install_ros2_pkgs
```

To install the latest upstream versions instead, use the `-l` (latest) flag — it will ignore the fixed versions:

```
install_ros2_pkgs -l
```

To build `px4_msgs` with every message definition (instead of only the T1-required subset), use the `--all-px4-msgs` flag:

```
install_ros2_pkgs --all-px4-msgs
```

By default all packages are installed; pass package names to install only a subset. Examples:

```
install_ros2_pkgs                          # install all packages
install_ros2_pkgs ros-jazzy-rosbag2        # install only the named package
install_ros2_pkgs -i                       # prompt before each package
install_ros2_pkgs -i -l px4_msgs           # flags and packages combine
install_ros2_pkgs --all-px4-msgs           # build all px4_msgs messages
```

Run `install_ros2_pkgs -h` to see the available packages and usage.

On success, the script prints each package it installed, a version report, and any post-install usage tips, ending with a `SUCCESS` line:

```
ROS2 Jazzy package installer -- mode: <mode>, version: <version-mode>

=== Install <package-name> (<version-mode>) ===
...

=== Report installation ===
<package-name> installed version: <version>

SUCCESS. All packages completed.

=== Usage: <package-name> ===
<post-install usage tips>
```

On failure it aborts immediately and prints the failing step. Fix the reported issue and re-run — the script is idempotent. Use `-i` to be prompted before each package:

```
install_ros2_pkgs -i
```

Verification

Confirm each package is installed at the expected version:

```
dpkg-query -W -f='${Version}\n' <package-name>
```

For a functional check, follow the usage tips printed by the script at the end of the installation.

Troubleshooting

Re-running after a failure — the script is idempotent, safe to re-run after fixing the reported issue. Use `-i` to be prompted before each package:

```
install_ros2_pkgs -i
# Answer [Y/n] before each package; press Enter to run, n to skip
```

27.7 Known issues

Some of the ROS2 components containing fixed versions in the scripts are obsolete and failing to install, for full compatibility please execute the scripts as instructed in the quick install section and check which versions have been tested in the build release notes.

27.8 Version History

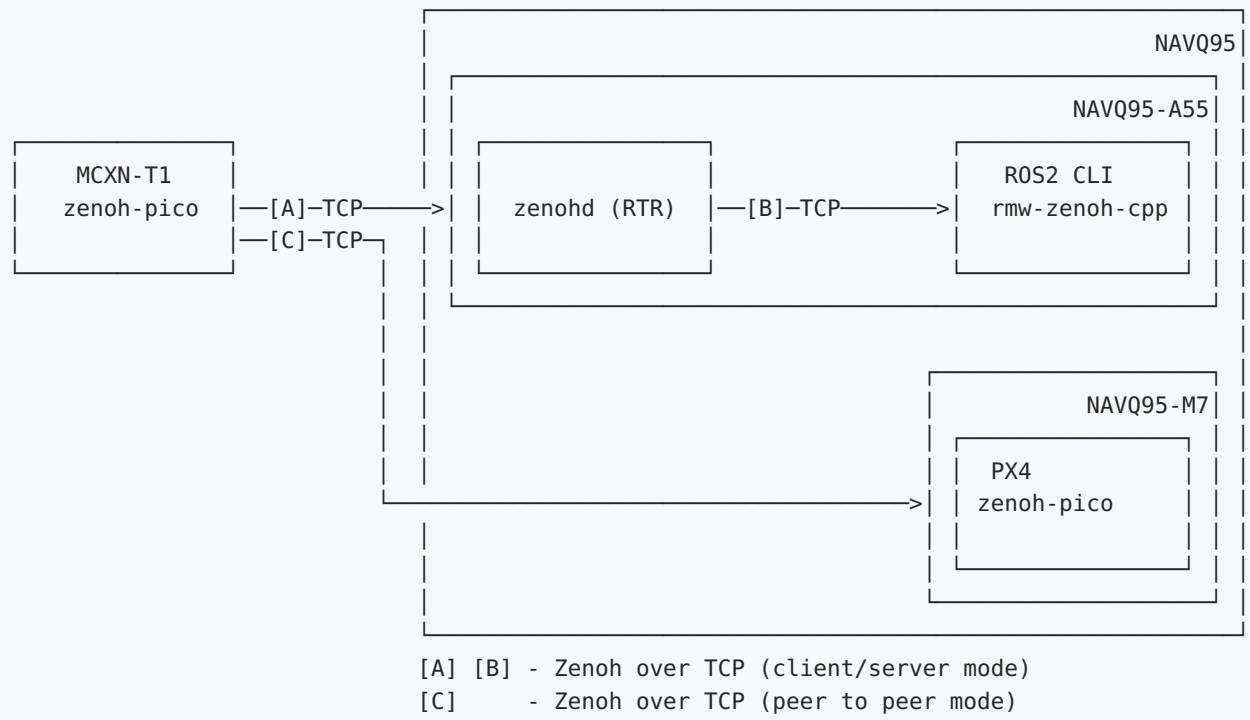
Version	Description
v7	`install_ros2_base` : remove fixed-version and `apt-mark hold` ; script installs latest version `install_ros2_env` , `install_ros2_pkgs` : add note to use `-l` flag as fixed versions are no longer available for installation
v6	`install_ros2_pkgs` : `px4_msgs` now builds only T1-required messages (`DistanceSensor` , `Sensor*`) by default. Add `--all-px4-msgs` flag to build all messages
v5	`install_ros2_base` : add early version conflict check; `apt-mark hold` only applied in fixed-version mode
v4	`install_ros2_pkgs` : NavQ95-specific ROS2 packages (available)
v3	`install_ros2_base` : add Overview section
v2	`install_ros2_env` : environment setup and Zenoh RMW configuration (available)
v1	Doc restructure (platform terminology, quick install) `install_ros2_base` : add `-l` (latest) and `-h` flags
v0	Initial release — `install_ros2_base` (base ROS2 installation)

28 ROS2 Multinode on NavQ95

28.1 Overview

This guide covers ROS2 communication between NavQ95 and other platforms in a multi-node setup. It describes the supported topologies, the required configuration, and how to verify the communication.

28.2 Topology



28.3 Architecture

ROS2 multinode communication on NavQ95 is based on Zenoh as the underlying transport. The key components are:

- zenohd — the Zenoh router. Acts as the central broker: all participants connect to it and it routes messages between them.
- rmw_zenoh_cpp — the ROS2 RMW (Robot Middleware) implementation that bridges ROS2 nodes to the Zenoh transport. ROS2 nodes are unaware of Zenoh — `rmw_zenoh_cpp` handles the translation transparently.
- zenoh-pico — a lightweight Zenoh client library targeting resource-constrained platforms. Supports multiple physical transports (TCP, serial, and others).

The topology described in this guide uses Zenoh over TCP as the transport between MCXN-T1 and NavQ95, with `zenohd` and ROS2 CLI running on the NavQ95 A55, and PX4 running on the NavQ95 M7. Interoperability is possible because `zenoh-pico` publishes messages in the same wire format that `rmw_zenoh_cpp` expects — no dedicated translation layer is required on any of the participants.

28.4 Enabling and Validating MCXN-T1 → NavQ95 A55 Communication

Prerequisites

- NavQ95 with `imx-image-robotics` image deployed and booted
- MCXN-T1 with GNSS, OFLO, or IO application flashed
- Both connected to the same network over T1 Ethernet

Configuration

Step 1 — NavQ95 A55: ROS2 setup

> Important: The complete ROS2 setup for the NavQ95 A55 is documented in ROS2 Setup.

ROS2 Jazzy and the Zenoh RMW must be installed and configured on the NavQ95 A55 before enabling multinode communication.

For this topology the one package that must be present is `px4_msgs` — the ROS2 message definitions for the PX4 topics published by MCXN-T1 (i.e. OFLO sensor), needed on the A55 side to subscribe to these topics. The full package set is not required — install just this one with:

```
install_ros2_pkgs px4_msgs
```

The branch installed must match the message definitions on the MCXN-T1. To use a different branch than the default, override it before running:

```
export ROS2PKG_PX4_MSGS_VERSION=<px4_msgs-release-branch>
install_ros2_pkgs px4_msgs
```

> Note: The `px4_msgs` build may take several minutes on the NavQ95 A55.

Step 2 — NavQ95 A55: configure T1 Ethernet interface

The T1 Ethernet link between MCXN-T1 and NavQ95 does not use DHCP — a static IP address must be assigned manually to the NavQ95 A55 interface.

```
sudo ip addr add <NAVQ95_IP>/<PREFIX> dev <eth_iface>
sudo ip link set <eth_iface> up
```

Known issues:

If MCXN-T1 is connected to one of the `trx1-trx6` interfaces via `end1` through the SJA-switch path, apply the following fixes:

- Prevent NetworkManager from managing the SJA switch interfaces (avoids it overwriting the configured IP addresses):

```
# Create 99-trx-unmanaged.conf file and fill-in the following
sudo tee /etc/NetworkManager/conf.d/99-trx-unmanaged.conf > /dev/null <<'EOF'
[device-trx-unmanaged]
match-device=interface-name:trx1;interface-name:trx2;interface-name:trx3;interface-
name:trx4;interface-name:trx5;interface-name:trx6
managed=0
EOF

# Reload NetworkManager to apply
sudo systemctl reload NetworkManager
```

- Disable TX checksum offload on `end1` to force software checksumming (TCP checksum offload is not properly applied on the SJA-switch path):

```
sudo ethtool -K end1 tx off
```

> Note: The static IP configuration and the TX checksum offload setting are lost on reboot.

Step 3 — MCXN-T1: configure zenoh-pico endpoint

The zenoh-pico client on MCXN-T1 runs in client mode and must connect to the `zenohd` router running on the NavQ95 A55. Configure the endpoint to `<NAVQ95_IP>:7447`.

For example, configure MCXN-T1 `zenoh_px4` publisher:

```
# Configure zenoh_px4 to connect to the NavQ95 A55 Zenoh router
zenoh_px4 config net client <NAVQ95_IP>:7447

# Restart zenoh_px4 to apply the configuration
zenoh_px4 restart
```

MCXN-T1 Configuration Reference

For more details on MCXN-T1 configuration and application-specific IP addresses, refer to the MCXN-T1 Zenoh guide.

Validation

Terminal 1 — NavQ95 A55: start zenohd

```
source ~/.ros2rc
ros2 run rmw_zenoh_cpp rmw_zenohd
```

Terminal 2 — MCXN-T1: start publishing (if not already running)

Start the zenoh-pico application on MCXN-T1. If using `zenoh_px4`, the endpoint was already configured in Step 3 — simply start or restart the application.

Terminal 3 — NavQ95 A55: subscribe

```
source ~/.ros2rc
source ~/ros2_ws/install/setup.bash
```

Confirm MCXN-T1 topics are visible on the NavQ95 A55 subscriber:

```
ros2 topic list
# Example output for OFLO sensor:
#   /fmu/in/distance_sensor
#   /fmu/in/sensor_optical_flow
```

Inspect topic details:

```
# Example output for OFLO sensor:
ros2 topic info /fmu/in/sensor_optical_flow
# Type: px4_msgs/msg/SensorOpticalFlow
# Publisher count: 1
# Subscription count: 0
```

Subscribe to a specific field (continuous):

```
# Example output for OFLO sensor:
ros2 topic echo /fmu/in/sensor_optical_flow px4_msgs/msg/SensorOpticalFlow --field quality
# quality: 91
# ---
# quality: 91
# ---
```

Receive a single full message:

```
# Example output for OFLO sensor:
ros2 topic echo /fmu/in/sensor_optical_flow px4_msgs/msg/SensorOpticalFlow --once
# ---
# timestamp: 4371636200
# timestamp_sample: 4371636200
# device_id: 0
# pixel_flow:
# - 0.0
# - 1.0
# delta_angle:
# - 0.0
# - 0.0
# - 0.0
# delta_angle_available: false
# distance_m: 0.0
# distance_available: false
# integration_timespan_us: 0
# quality: 91
# error_count: 0
# max_flow_rate: 0.0
# min_ground_distance: 0.0
# max_ground_distance: 0.0
# mode: 3
# ---
```

Pass criteria: `ros2 topic echo` prints decoded messages from MCXN-T1. No errors in the zenohd log.

28.5 Enabling and Validating MCXN-T1 → NavQ95 M7 Communication

In this topology MCXN-T1 communicates directly with PX4 running on the NavQ95 M7 core over a peer-to-peer Zenoh link (no router involved). The MCXN-T1 node acts as the listener and PX4 connects to it.

For the full configuration and validation steps, refer to Scenario 2 — MR-MCXN-T1 node (peer) / PX4 (peer) in the MCXN-T1 Zenoh guide.

28.6 Version History

Version	Description
v2	Add MCXN-T1 → NavQ95 M7 topology; update diagram, multi-sensor prerequisites, MCXN-T1 config reference
v1	Install `px4_msgs` via `install_ros2_pkgs`; merge px4_msgs into the ROS2 setup step
v0	Initial release — MCXN-T1 → NavQ95 A55 communication over TCP (Zenoh)

29 Flight Stack

The PX4 flight stack provides the autopilot functionality for the MR-NavQ95. It runs on the Cortex-M7 real-time core of the board and delivers the flight stack, vehicle setup, sensor calibration, parameter configuration, and communication interfaces (uORB, MAVLink, and more) that the NavQ95 robotics use cases build on.

- PX4 Documentation Reference

30 PX4 Documentation Reference

PX4 is the open-source software stack that implements the autopilot functionality running on supported NXP Robotic platforms. On the MR-NavQ95 it runs on the Cortex-M7 real-time core, while the Cortex-A55 application cores run Linux and the robotics middleware; the two domains communicate over the on-chip RPMsg/OpenAMP transport. PX4 provides the flight stack, vehicle setup, sensor calibration, parameter configuration, and communication interfaces (uORB, MAVLink, and more) that our robotics use cases build on.

For detailed, version-specific guidance, refer to the official PX4 User Guide for version 1.17, which covers hardware setup, configuration, flight modes, development, and troubleshooting:

- PX4 User Guide (v1.17)

31 MR-VMU-Tropic

The MR-VMU-Tropic is the flight-controller platform of the NXP Mobile Robotics family. Start with the quick start, then consult the PX4 flight-stack documentation.

- Overview
  - Functional block diagram
  - Main components
- Getting Started
  - Deploy and Run Quick Start
- User Manual
  - Introduction
  - Finding platform resources
  - Getting ready
  - Getting to know the hardware
  - Configuring the hardware
  - Abbreviations
  - References
- Application Notes
  - Hardware Architecture and Reference Design
  - Programming Boot Configuration Fuses
- Flight Stack
  - PX4 Documentation Reference

32 Overview

The MR-VMU-Tropic is the flight-controller (vehicle management unit) reference design of the NXP Mobile Robotics family. It is a PX4 / NuttX autopilot built around an NXP i.MX RT1064 crossover MCU, providing the real-time flight-control and I/O functions of a drone or mobile robot. The board is intended as a validated flight-controller starting point that customers can evaluate, reproduce, and adapt for their own vehicles.

The i.MX RT1064 is an NXP crossover MCU based on an Arm Cortex-M7 core with integrated flash and on-chip real-time clock. It runs the PX4 flight stack on the NuttX RTOS and drives the motor outputs, sensor interfaces, and communication links required for autonomous flight and motion control.

32.1 Functional block diagram

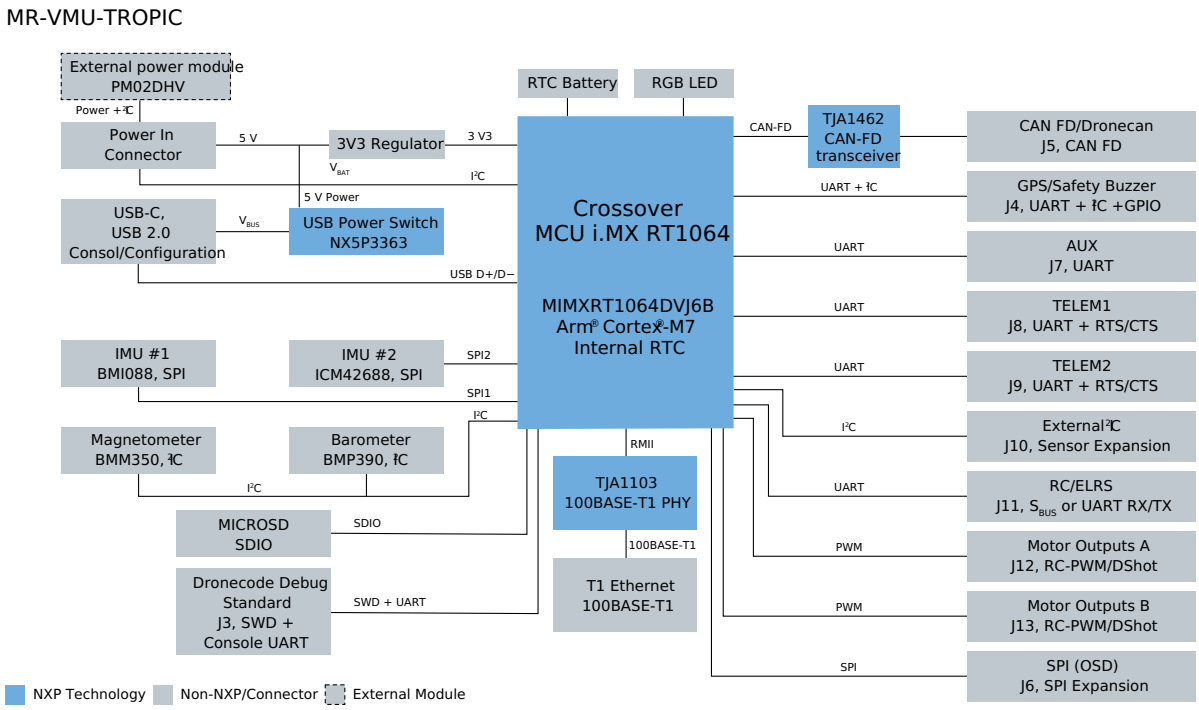


Figure 1. MR-VMU-Tropic functional block diagram

32.2 Main components

The reference design integrates the following main NXP components and interfaces:

- i.MX RT1064 crossover MCU (Arm Cortex-M7) with internal RTC
- 3.3 V regulation from a 9 V to 52 V input through the NXP PM02DHV power module
- Redundant inertial sensors, magnetometer, and a Bosch BMP390 or BMP388 barometer
- CAN-FD / DroneCAN through the NXP TJA1462 transceiver
- 100BASE-T1 automotive Ethernet through the NXP TJA1103 PHY
- UART telemetry, GPS / safety, and RC / ELRS inputs
- RC-PWM / DShot motor outputs and SPI OSD expansion

- USB-C console and SWD / JTAG debug through the Dronecode debug connector

NXP also provides the complete hardware Bill of Materials and design files, enabling customers to evaluate, reproduce, and customize the reference platform for their own products.

For overview information, documentation, software, tools, and ordering details, see the MR-VMU-Tropic product page on the NXP web site.

> Note: The barometric pressure sensor is either a Bosch BMP390 or a Bosch BMP388, depending on component availability at manufacturing time. The BMP388 is a drop-in, functionally equivalent replacement for the BMP390; the two parts differ only in minor specifications that do not affect the flight-controller use case. The software supports both sensors, so no user action is required regardless of which part is fitted.

33 Getting Started

Deploy and run on the MR-VMU-Tropic flight controller.

- Deploy and Run Quick Start

34 Deploy and Run Quick Start

MR-VMU-Tropic is a PX4/NuttX flight controller on an i.MXRT1064. Flash the bootloader and the application to external QSPI flash with LinkServer (MCU-Link). The two images go to fixed offsets in the RT1064 flash region.

34.1 Get the artifacts first

Every platform assumes the images are already available — either downloaded or built with the SDK:

```
# prebuilt (fastest)
west sdk download --platform mr-vmu-tropic      # lands in <west topdir>/bin/mr-vmu-tropic/

# or build from source
west sdk build    --platform mr-vmu-tropic  # lands in <west topdir>/{build,bin}/...
```

- Docker required. Downloading and building are done inside a Docker container built from the SDK's `Dockerfile` — see the robotics-sdk README for how to set it up. All `west` commands are assumed to run inside that container.
- The SDK builds/downloads images but does not flash them — deployment is done with each platform's native tool.
- Where do the artifacts land? `<west topdir>` is the west workspace root — the directory containing `.west/`, created by `west init`. The `bin/` directory is created next to the `robotics-sdk/` clone, not inside it:

```
<west topdir>/
├── .west/
├── robotics-sdk/         <- the SDK clone (west init -l)
├── build/                <- west sdk build intermediates
└── bin/mr-vmu-tropic/       <- final artifacts land here
```

For MR-VMU-Tropic the platform is `mr-vmu-tropic`. Run `west sdk list` for the authoritative list.

34.2 Artifacts

- `<west topdir>/bin/mr-vmu-tropic/nxp_mr-tropic_bootloader.bin` — bootloader (@ `0x70000000`)
- `<west topdir>/bin/mr-vmu-tropic/nxp_mr-tropic_default.bin` — application firmware (@ `0x70020000`)
- `<west topdir>/bin/mr-vmu-tropic/nxp_mr-tropic_default.px4` — PX4 firmware container (for QGroundControl / PX4 uploader)

34.3 Flash with LinkServer

Use this route for the initial bring-up — it programs the PX4 bootloader (needed only once, on a blank board) and, optionally, a first application image directly to QSPI. Once the bootloader is present, prefer the PX4 firmware upload route below for routine application updates.

1. Flash the bootloader (first time / blank board)

```
LinkServer flash MIMXRT1064xxxxA:EVK-MIMXRT1064 load \
    --addr 0x70000000 nxp_mr-tropic_bootloader.bin
```

2. Flash the application

```
LinkServer flash MIMXRT1064xxxxA:EVK-MIMXRT1064 load \
    --addr 0x70020000 nxp_mr-tropic_default.bin
```

3. Reset & run

   Reset the board; the bootloader chainloads the app at `0x70020000`.

34.4 Flash the PX4 application (recommended)

Once the PX4 bootloader is on the board, PX4 firmware is delivered as a `.px4` container (`nxp_mr-tropic_default.px4`) and uploaded over USB — no debug probe needed. The board's PX4 bootloader accepts the upload when the board is (re)connected; the uploader waits for it and resets automatically.

- With QGroundControl (easiest):

  1. Open QGroundControl → Vehicle Setup → Firmware.
  2. Unplug the board, then plug it back in over USB when prompted.
  3. Choose Advanced settings → Custom firmware file, select `nxp_mr-tropic_default.px4`, and let QGroundControl flash and reboot the board.

- With the PX4 command-line uploader (from a PX4-Autopilot source tree, useful for CI/scripting):

```
python3 src/drivers/bootloaders/scripts/px_uploader.py \
    --port /dev/ttyACM0 nxp_mr-tropic_default.px4
```

  Replace `/dev/ttyACM0` with the board's serial port (`COMx` on Windows). Unplug/replug the board when the uploader reports it is waiting for the bootloader.

> ℹ️ The `.px4` container carries the same signed application as `nxp_mr-tropic_default.bin`; use the `.px4` upload path for day-to-day PX4 updates and the LinkServer `.bin` path only when it is necessary to (re)install the bootloader or recover a bricked board.

34.5 Erase all flash (recovery)

```
LinkServer flash MIMXRT1064xxxxA:EVK-MIMXRT1064 erase
```

Use this to wipe the QSPI flash before a clean re-flash.

34.6 Zephyr

The MR-VMU-Tropic platform also publishes Zephyr sample images for board bring-up and peripheral checks:

```
<west topdir>/bin/mr-vmu-tropic/zephyr/<sample>/zephyr.bin
```

where `<sample>` is any one of `hello_world`, `gpio`, `bmi08x_icm45686`, `bmm350`, `bmp390`, `pwmLED`, `can`, `sdcard`, `dhcpv4_client`, `usb`, `watchdog`, `rtc`.

Flash one with the same LinkServer command, at the flash base:

```
LinkServer flash MIMXRT1064xxxxA:EVK-MIMXRT1064 load \
    --addr 0x70000000 zephyr.bin
```

35 User Manual

Get to know the MR-VMU-Tropic flight controller: review the board hardware, prepare and flash the firmware, and run the PX4 flight stack.

- Introduction
- Finding platform resources
- Getting ready
- Getting to know the hardware
- Configuring the hardware
- Abbreviations
- References

36 Introduction

The MR-VMU-Tropic is the flight-controller (vehicle management unit) reference design of the NXP Mobile Robotics family. It is a PX4 / NuttX autopilot built around an NXP i.MX RT1064 crossover MCU, providing the real-time flight-control and I/O functions of a drone or mobile robot. The board is intended as a validated flight-controller starting point that customers can evaluate, reproduce, and adapt for their own vehicles.

The i.MX RT1064 is an NXP crossover MCU based on an Arm Cortex-M7 core with integrated flash and an on-chip real-time clock. It runs the PX4 flight stack on the NuttX RTOS and drives the motor outputs, sensor interfaces, and communication links required for autonomous flight and motion control.

This user manual describes the board hardware, how to prepare the firmware images, how to flash the board, and how to run the flight stack. The connector pinouts and on-board devices are documented in the Hardware Overview; the hardware architecture is described in the MR-VMU-Tropic Hardware Architecture and Reference Design.

36.1 NXP Robotics SDK and upstream open-source software

NXP embraces an open-source development model and actively participates in the communities behind the software used by this product. The NXP Robotics SDK provides a curated and validated set of software components, tools, and documentation that have been tested together on this hardware platform.

NXP regularly contributes improvements and fixes back to upstream repositories. Because contributions are made on a scheduled basis, the latest upstream code and the version included in the NXP Robotics SDK may not always be identical. The SDK release is recommended for users seeking a tested and supported configuration, while advanced users may choose to work directly with the upstream project and its latest features.

37 Finding platform resources

The information page for the MR-VMU-Tropic flight-controller reference design is on the NXP web site at https://www.nxp.com/design/design-center/development-boards-and-designs/MR-VMU-TROPIC. The information page provides overview information, documentation, software and tools, and a Getting Started tab with quick-reference information, including the downloadable assets referenced in this document.

The board design files and the latest hardware reference are published in the public NXP Robotics repository at https://github.com/NXP-Robotics/MR-VMU-TROPIC. The software, images, and build tooling are distributed through the NXP Robotics SDK; follow the NXP Robotics SDK README to set up the SDK and obtain the assets.

38 Getting ready

Before flashing and running the MR-VMU-Tropic, prepare the firmware images and the tools used to deploy them.

38.1 Firmware images

The firmware for the MR-VMU-Tropic is provided through the NXP Robotics SDK, which builds and downloads the images inside a Docker container. The SDK produces three artifacts for the board:

- a bootloader image,
- the PX4 application firmware (a raw binary), and
- the PX4 firmware container that QGroundControl and the PX4 uploader accept.

In addition, the SDK publishes a set of Zephyr sample images for board bring-up and peripheral checks.

Obtaining or building the images with the SDK is covered in the Deploy and Run Quick Start.

38.2 Tools

- A LinkServer-compatible debug probe (MCU-Link) for the initial bootloader programming over the Dronecode debug connector.
- QGroundControl, or the PX4 command-line uploader, for routine PX4 application updates over USB.
- A USB-C cable for the console and PX4 firmware uploads.

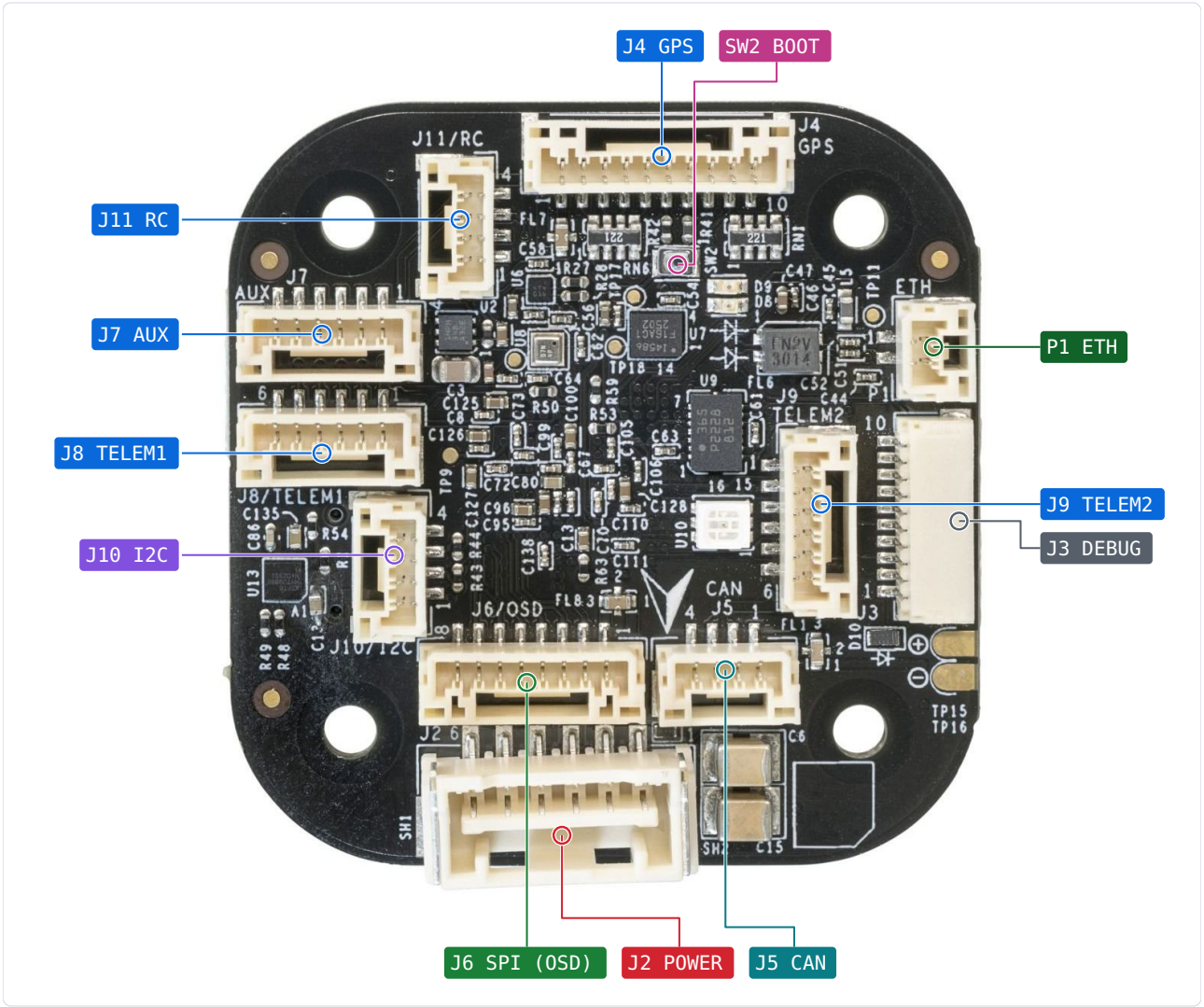
The debug connector and USB-C port locations are shown in the Hardware Overview.

39 Getting to know the hardware

Connectors, pinouts and on-board devices of the NXP MR-VMU-Tropic, an open-source Vehicle Management Unit (flight controller) built around the NXP i.MX RT1064 MCU. The board carries the Dronecode DS-009 connector set (telemetry, GPS, I2C, CAN and debug), two ESC headers, a GPS and safety-switch port, an SPI port for an on-screen display, 100BASE-T1 Ethernet, a microSD slot and a USB device port. All pin data below comes from the rev F schematic SPF-94468_F. The photos show a rev C board; the connector positions are the same.

Each card shows three things per pin: the function, the net name in the schematic and the i.MX RT1064 pad the signal ends up on. Parts in between, such as the CAN transceiver and the Ethernet PHY, are shown too. Pin 1 is the square pad.

39.1 Board overview

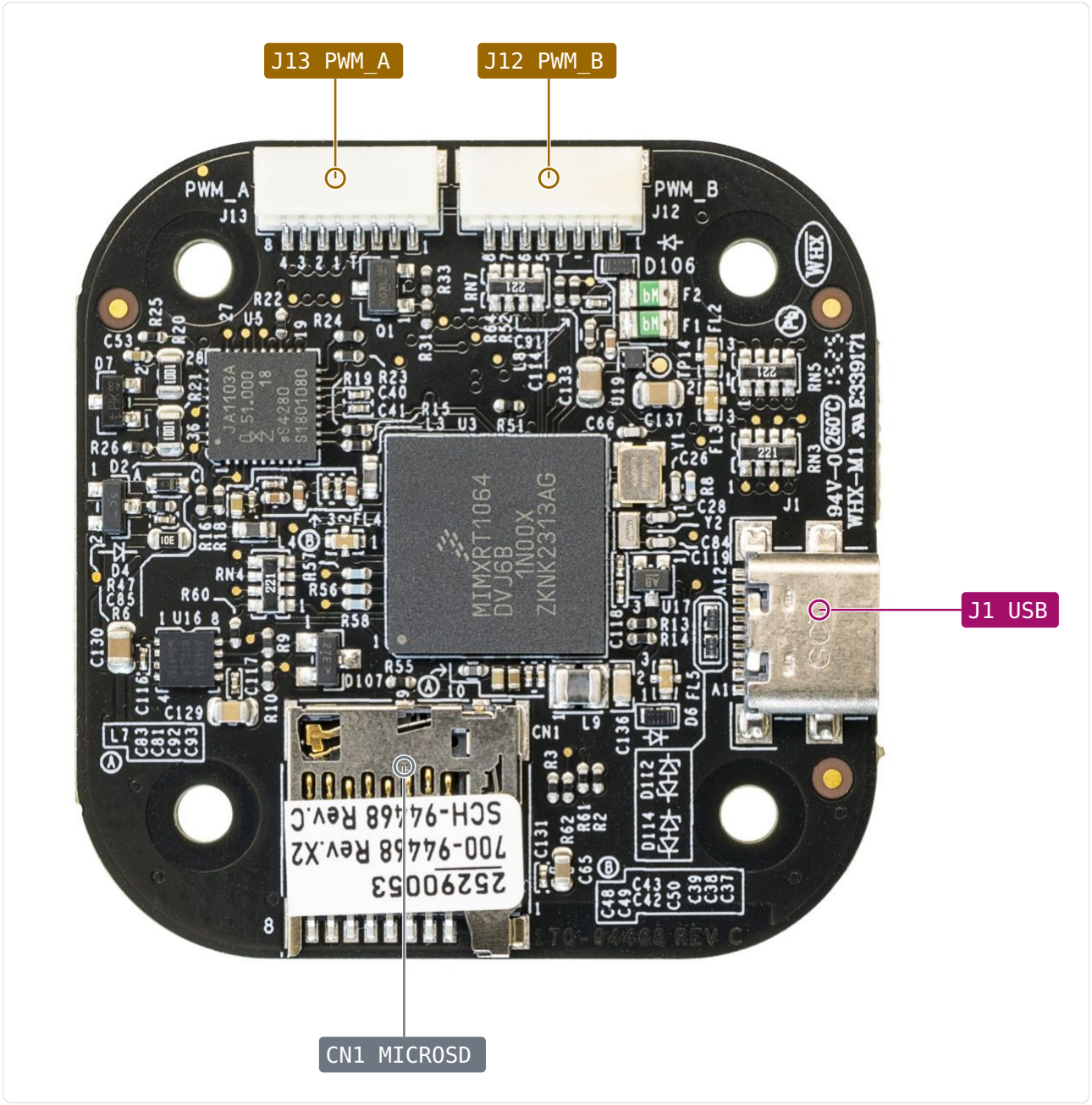


Not visible here:

- J1 USB (bottom view)
- J12 PWM_B (bottom view)
- J13 PWM_A (bottom view)

- CN1 MICROSD (bottom view)

Bottom side, with the two ESC headers, the USB-C port and the microSD slot:



39.2 Connector index

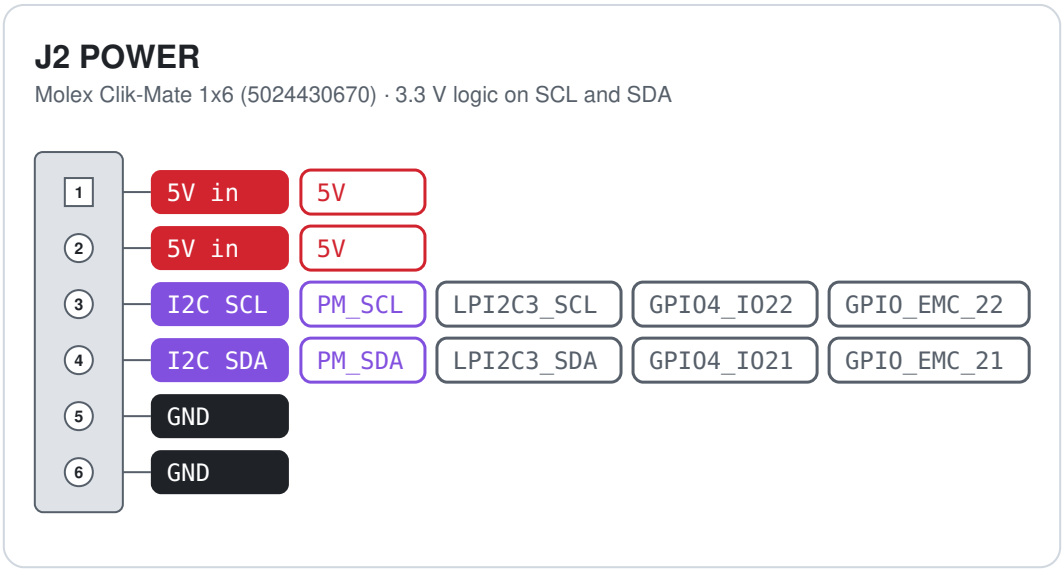
Ref	Function	Connector	Details
Main CN1 MICROSD	microSD card slot	microSD push-push, 8 pins plus card detect	Four-bit SD interface on the first SD host of the chip, used for logging. The card detect switch goes to a separate chip pin.

Ref	Function	Connector	Details
Main J1 USB	USB 2.0 Type-C device port	USB Type-C receptacle (USB4110-GF-A)	Serial download port of the i.MX RT1064 boot ROM, and the port used for flashing. The board is always a USB device. VBUS can also power the board through protection switch U13.
Main P1 ETH	100BASE-T1 single pair Ethernet	JST-GH 1x2, top entry	
Main J2 POWER	Power module input, 5 V plus battery monitor I2C	Molex Clik-Mate 1x6 (5024430670)	
Main J3 DEBUG	SWD debug and console, DS-009 debug full layout	JST-SH 1x10	
Main J4 GPS	GPS module with compass, safety switch and buzzer	JST-GH 1x10 (BM10B-GHS-TBT), top entry	
Main J5 CAN	CAN FD (TJA1462)	JST-GH 1x4, top entry	
Main J6 SPI (OSD)	External SPI port, meant for an on-screen display board	JST-GH 1x8 (BM08B-GHS-TBT), top entry	
Main J7 AUX	Spare serial port with handshake lines	JST-GH 1x6 (BM06B-GHS-TBT), top entry	
Main J8 TELEM1	Telemetry serial port 1 with flow control	JST-GH 1x6 (BM06B-GHS-TBT), top entry	
Main J9 TELEM2	Telemetry serial port 2 with flow control	JST-GH 1x6, top entry	
Main J10 I2C	External I2C bus	JST-GH 1x4, top entry	

Ref	Function	Connector	Details
Main J11 RC	Serial radio control receiver port	JST-GH 1x4, top entry	
Main J12 PWM_B	Motor outputs 5 to 8 (ESC header)	JST-SH 1x8	
Main J13 PWM_A	Motor outputs 1 to 4 (ESC header)	JST-SH 1x8	

39.3 Power and debug

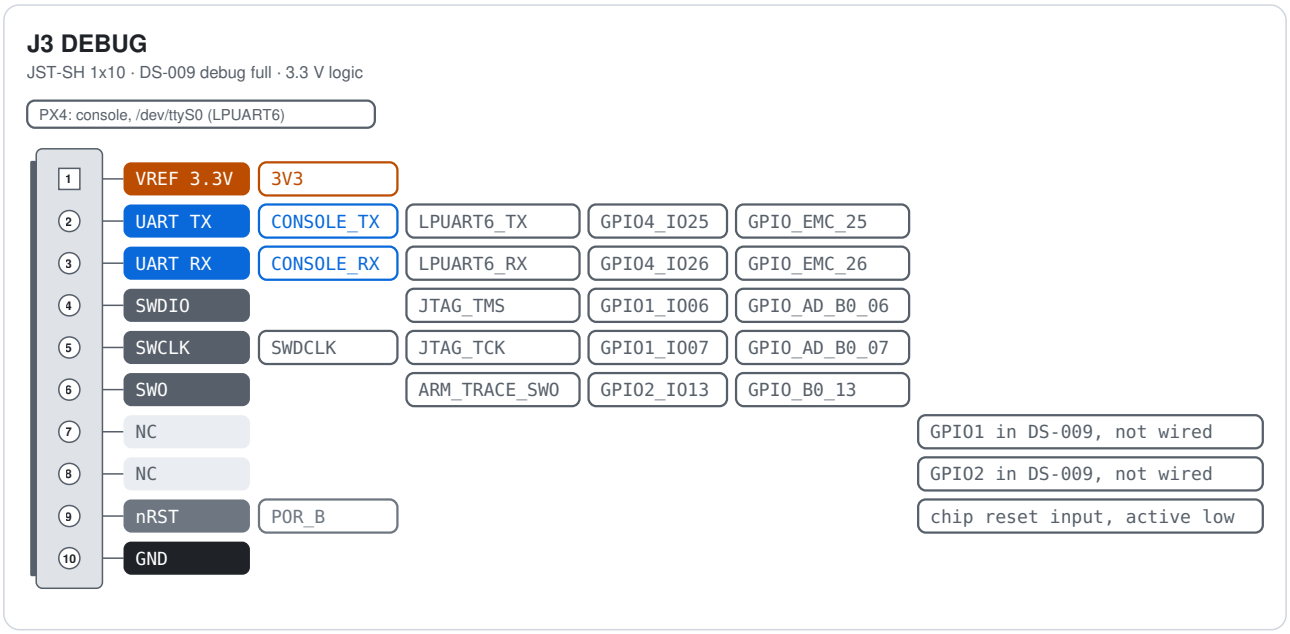
Main J2 POWER



This is the main supply of the board. The power module feeds 5 V into pins 1 and 2, and the board makes its own 3.3 V and 1.8 V from it. Pins 3 and 4 are an I2C bus that reads voltage and current from the power module.

> Warning: Feed only 5 V into pins 1 and 2. A higher voltage damages the board, because there is no regulator behind this connector.

Main J3 DEBUG

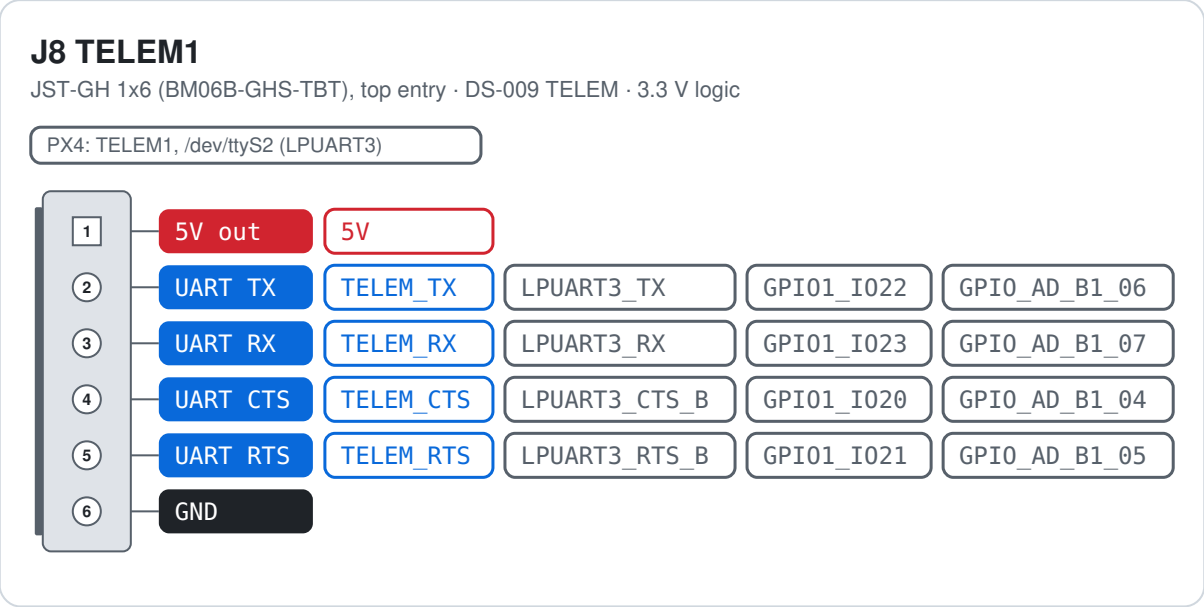


Pin 1 is a 3.3 V reference output for the debug probe. Pins 2 and 3 are the serial console. Pins 4, 5 and 6 are the SWD lines of the i.MX RT1064. Pin 9 is the reset line of the chip, which is also pulled up on the board.

> Warning: Pin 1 is an output at 3.3 V, not a supply input. Use a debug probe that works at 3.3 V and never drive this pin.

39.4 Serial ports

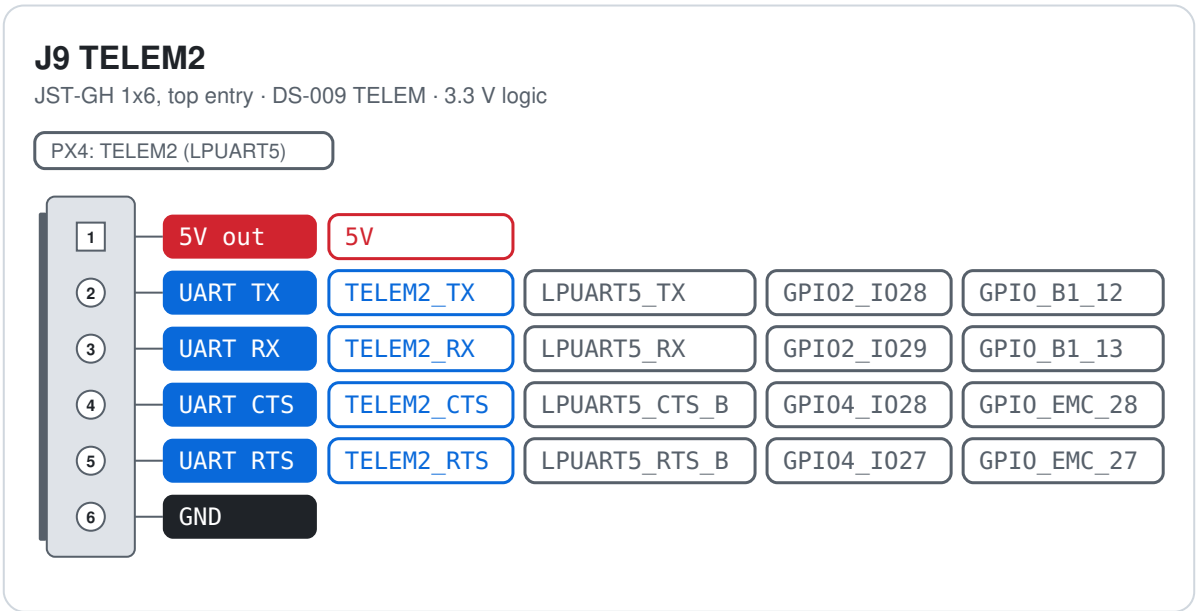
Main J8 TELEM1



Standard DS-009 telemetry layout: 5 V, TX, RX, CTS, RTS, ground. Pin 1 gives 5 V out through ferrite FL3.

> Warning: Pin 1 is a 5 V output. Never feed power into it.

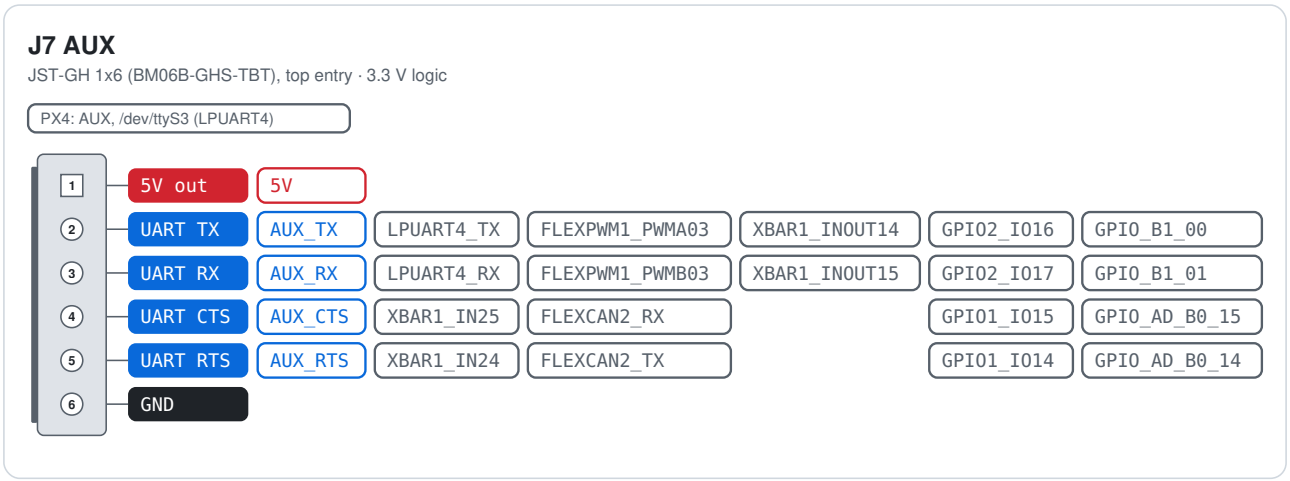
Main J9 TELEM2



Same layout and same circuit as TELEM1, on a second serial port of the chip. Pin 1 gives 5 V out.

> Warning: Pin 1 is a 5 V output. Never feed power into it.

Main J7 AUX



Pin 1 provides a 5 V output. Pins 2 and 3 are assigned to LPUART4, which is used as the AUX serial port (/dev/ttyS3 in PX4). Pins 4 and 5 are labeled CTS and RTS in the schematic; however, these pins cannot be multiplexed to the LPUART4 hardware CTS and RTS functions and therefore cannot provide hardware flow control. If RTS/CTS flow control is required, it must be implemented in software using these pins as GPIOs.

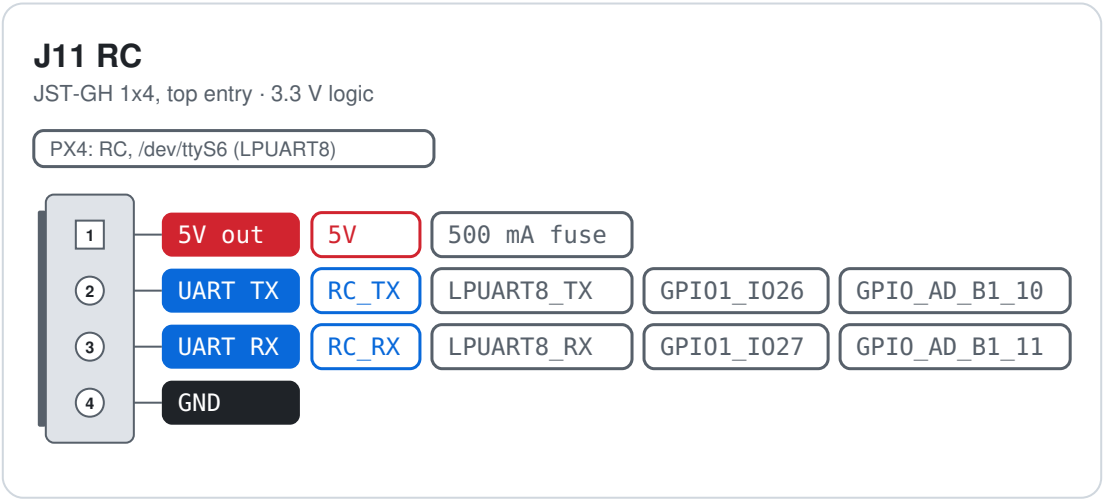
The signal pins on this connector support several alternative functions through pin multiplexing. Pins 2 and 3 can be configured as PWM outputs using FlexPWM1 module 3, channels A and B. All four signal pins can support quadrature encoder or trigger functions through XBAR1, while pins 4 and 5 can instead provide a second CAN interface using FlexCAN2 with an external CAN transceiver.

Use of these alternative functions requires appropriate software and pin-multiplexing configuration and is left to the user.

> Warning: Pin 1 is a 5 V output. Never feed power into it.

> Note: Confirm that the ESCs do in fact follow the BetaFlight connector standard, as there are many ESCs with variations on this pinout. Also note that wires are regularly soldered to pads on the ESC side.

Main J11 RC



Serial port for a digital receiver such as ELRS or CRSF. Pin 1 gives 5 V out through a resettable fuse, so a shorted receiver cannot pull down the board supply.

> Warning: Pin 1 is a 5 V output. Never feed power into it.

39.5 GPS, I2C and CAN

Main J4 GPS

J4 GPS

JST-GH 1x10 (BM10B-GHS-TBT), top entry · DS-009 GPS · 3.3 V logic

PX4: GPS, /dev/ttyS1 (LPUART2)

1	5V out	5V				
2	UART TX	GPS_TX	LPUART2_TX	GPIO1_IO18	GPIO_AD_B1_02	
3	UART RX	GPS_RX	LPUART2_RX	GPIO1_IO19	GPIO_AD_B1_03	
4	I2C SCL	EXT_SCL	LPI2C1_SCL	GPIO1_IO16	GPIO_AD_B1_00	
5	I2C SDA	EXT_SDA	LPI2C1_SDA	GPIO1_IO17	GPIO_AD_B1_01	
6	Safety switch	ARM	XBAR1_INOUT10	GPIO2_IO12	GPIO_B0_12	
7	Safety LED	GPS_LED	FLEXPWM1_PWMB03	GPIO3_IO25	GPIO_EMC_39	
8	3.3V out	3V3				
9	Buzzer	BUZZER_EXT	FLEXPWM1_PWMA00	GPIO4_IO23	GPIO_EMC_23	via FET Q1
10	GND					

Pin 1 gives 5 V to the GPS module and pin 8 gives 3.3 V. Both are outputs. Pins 2 to 5 are the GPS serial port and the shared I2C bus, the same bus as on the I2C connector J10. Pin 6 reads the safety switch and pin 7 drives its LED. Pin 9 is the buzzer, switched to ground by FET Q1, so the board only pulls this pin low.

> Warning: Pins 1 and 8 are power outputs. Never feed power into them.

Main J10 I2C

J10 I2C

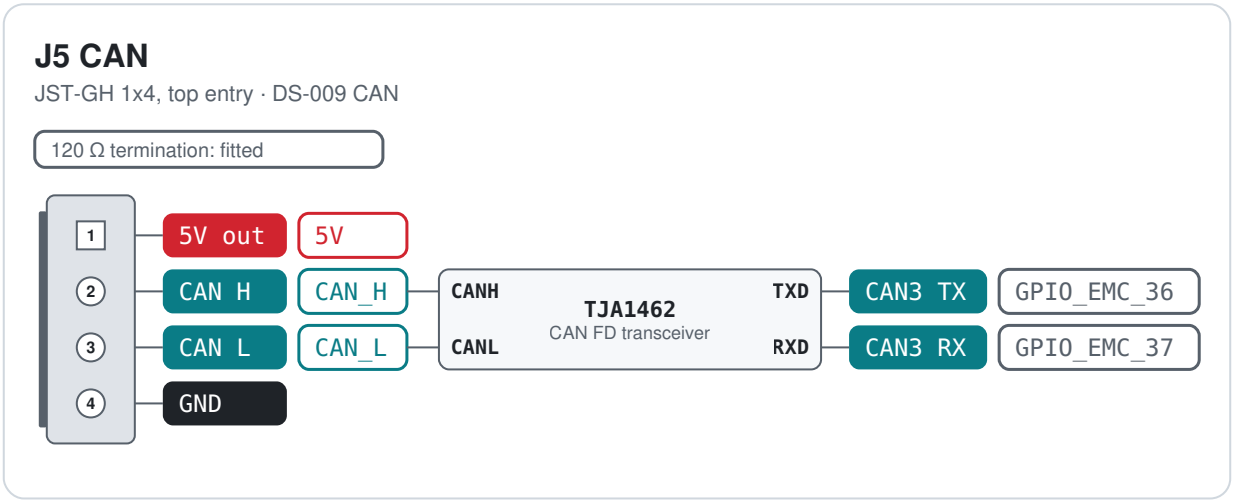
JST-GH 1x4, top entry · DS-009 I2C · 3.3 V logic

1	5V out	5V			
2	I2C SCL	EXT_SCL	LPI2C1_SCL	GPIO1_IO16	GPIO_AD_B1_00
3	I2C SDA	EXT_SDA	LPI2C1_SDA	GPIO1_IO17	GPIO_AD_B1_01
4	GND				

The same bus as pins 4 and 5 of the GPS connector, so devices here share those addresses. Pin 1 gives 5 V out.

> Warning: Pin 1 is a 5 V output. Never feed power into it.

Main J5 CAN

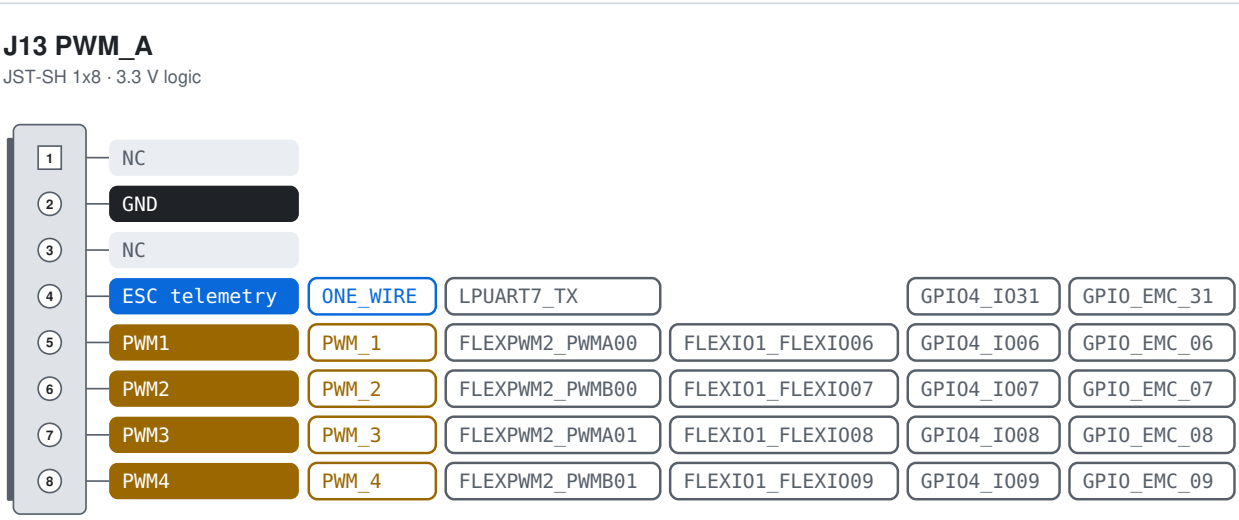


The TJA1462 transceiver runs from 5 V and talks to the chip at 3.3 V. CANH and CANL together form the differential bus, TXD and RXD are the logic lines to the i.MX RT1064. The 120 Ω termination is fitted, so this port is one end of the bus.

> Warning: Pin 1 is a 5 V output. Never feed power into it.

39.6 Motors

Main J13 PWM_A



Pin 4: LPUART7 in single-wire mode, receive only

Four motor signals for a four-in-one ESC. The pin order follows the Betaflight ESC connector standard, so a standard 8-pin ESC cable fits. The signals carry normal PWM or DShot. The same pads are also FlexIO1 pins, so DShot or other protocols can be bit-banged by FlexIO. Pin 4 is the ESC telemetry line, shared with the other ESC

header. The ESCs send their UART telemetry (voltage, current, RPM, temperature) on this one wire, and the chip receives it on LPUART7 in single-wire mode. It is receive only; the chip never transmits here. Pins 1 (battery voltage) and 3 (current) of the Betaflight standard are not connected on rev F, so the board does not read battery voltage or current from the ESC.

> Note: This header has no 5 V pin. Many ESCs have a BEC that powers the flight controller, but this board does not use it. It gets its 5 V from the power module on J2 POWER.

Main J12 PWM_B

J12 PWM_B

JST-SH 1x8 · 3.3 V logic

1	NC					
2	GND					
3	NC					
4	ESC telemetry	ONE_WIRE	LPUART7_TX		GPIO4_IO31	GPIO_EMC_31
5	PWM5	PWM_5	FLEXPWM4_PWMA01	FLEXIO1_FLEXIO02	GPIO4_IO02	GPIO_EMC_02
6	PWM6	PWM_6	FLEXPWM4_PWMB01	FLEXIO1_FLEXIO03	GPIO4_IO03	GPIO_EMC_03
7	PWM7	PWM_7	FLEXPWM4_PWMA02	FLEXIO1_FLEXIO04	GPIO4_IO04	GPIO_EMC_04
8	PWM8	PWM_8	FLEXPWM4_PWMB02	FLEXIO1_FLEXIO05	GPIO4_IO05	GPIO_EMC_05

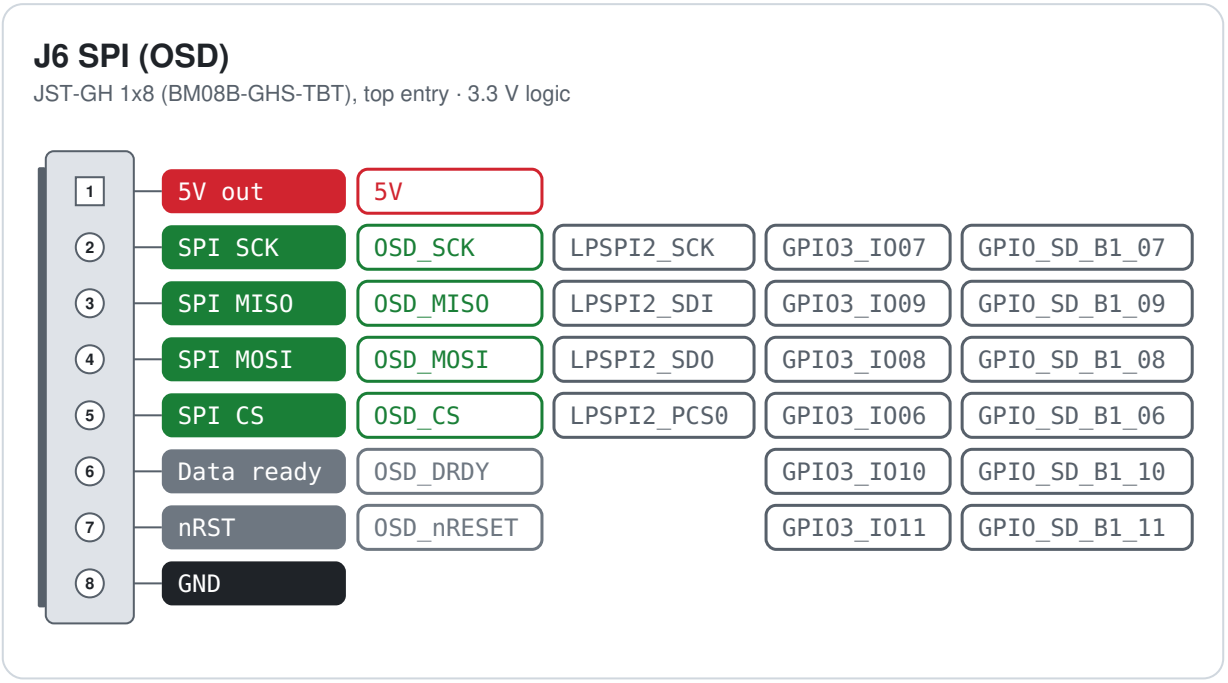
Pin 4: LPUART7 in single-wire mode, receive only

Same circuit and same Betaflight pin order as PWM_A, for motors 5 to 8. The signals carry normal PWM or DShot. The same pads are also FlexIO1 pins, so DShot or other protocols can be bit-banged by FlexIO. Pin 4 is the same shared ESC telemetry line: LPUART7 in single-wire mode, receive only. Pins 1 (battery voltage) and 3 (current) of the Betaflight standard are not connected on rev F, so the board does not read battery voltage or current from the ESC.

> Note: This header has no 5 V pin. Many ESCs have a BEC that powers the flight controller, but this board does not use it. It gets its 5 V from the power module on J2 POWER.

39.7 SPI and Ethernet

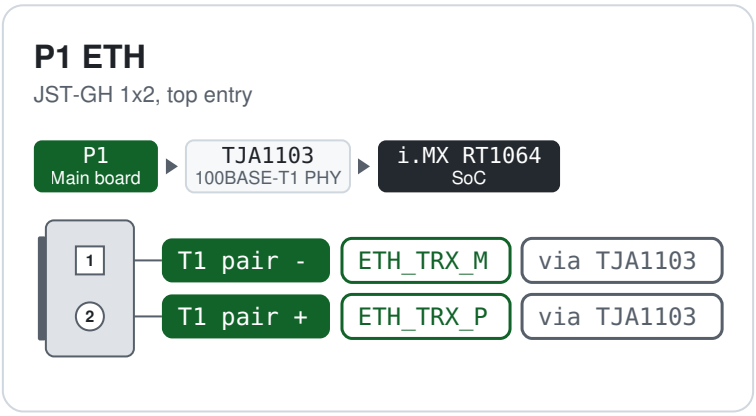
Main J6 SPI (OSD)



Pin 1 gives 5 V to the add-on board, the signal lines work at 3.3 V. Pin 6 is a data-ready input and pin 7 a reset output. This port is separate from the SPI bus of the on-board sensors.

> Warning: Pin 1 is a 5 V output. Never feed power into it.

Main P1 ETH



Two wires carry send and receive at 100 Mbit/s. The TJA1103 does the physical layer and talks to the chip over RMII, so there is no direct link between these pins and a chip pad. Use a twisted pair and keep both wires the same length. The other end needs a 100BASE-T1 device or a media converter.

Main other connectors

Ref	Function	Connector	Notes
J1 USB	USB 2.0 Type-C device port	USB Type-C receptacle (USB4110-GF-A)	Serial download port of the i.MX RT1064 boot ROM, and the port used for flashing. The board is always a USB device. VBUS can also power the board through protection switch U13.
CN1 MICROSD	microSD card slot	microSD push-push, 8 pins plus card detect	Four-bit SD interface on the first SD host of the chip, used for logging. The card detect switch goes to a separate chip pin.

39.8 Buttons and switches

Ref	Board	Function	Type	Notes
SW2 BOOT	Main	Boot mode button	Push button (NANOT 240 AS)	Holding this button while the board powers up pulls boot mode 0 high and starts the serial downloader over the USB port. The second boot mode pin is tied low on the board.

39.9 Notes

- The connectors marked DS-009 on their card (TELEM1, TELEM2, GPS, I2C, CAN and the debug port) follow the Dronecode DS-009 connector standard, so standard cables fit.
- The two ESC headers (PWM_A and PWM_B) follow the Betaflight ESC connector standard, so a four-in-one ESC with a standard 8-pin cable fits. Pins 1 (battery voltage) and 3 (current) are not connected on this board.
- The board runs from 5 V on the power module connector J2, or from USB. Every other 5 V pin is an output.

> Warning: Never feed power into the 5 V pin of a JST-GH connector. Power the board through J2 or USB only.

39.10 Cheat sheet

All connector cards on one A4 landscape sheet: cheat sheet PDF.

39.11 References

- MR-VMU-TROPIC board design and documentation

40 Configuring the hardware

The MR-VMU-Tropic is flashed in two stages: the PX4 bootloader is programmed once to the external QSPI flash with a debug probe, after which the PX4 application is updated over USB without a probe.

40.1 Chip Fuse Programming

This step is not required for EVK boards, as their fuses are already programmed.

A newly manufactured board, or any board containing a new unprogrammed processor, requires the device fuses to be programmed as part of initial bring-up. Fuse configuration and the associated programming procedure are covered in the Programming boot configuration fuses on MR-VMU-Tropic application note.

40.2 Programming the bootloader (first time)

On a blank board, use a LinkServer-compatible probe (MCU-Link) on the Dronecode debug connector to program the bootloader, and optionally a first application image, to fixed offsets in the RT1064 QSPI flash region. This route is used for the initial bring-up and for recovering a bricked board.

40.3 Updating the PX4 application (routine)

Once the bootloader is present, the PX4 firmware is delivered as a `.px4` container and uploaded over USB, with no debug probe required. The board's PX4 bootloader accepts the upload when the board is reconnected; both QGroundControl and the PX4 command-line uploader drive this automatically. This is the day-to-day update path.

40.4 Recovery

The QSPI flash can be fully erased with the debug probe before a clean re-flash.

The exact LinkServer commands, flash offsets, PX4 upload steps, and the Zephyr sample images are listed in the Deploy and Run Quick Start.

40.5 Running the flight stack

The MR-VMU-Tropic runs the PX4 flight stack on the NuttX RTOS. Once the board is flashed, PX4 is configured and operated through QGroundControl in the same way as any other PX4 vehicle management unit: airframe selection, sensor calibration, radio and flight-mode setup, and parameter tuning.

The board's communication and actuation interfaces used by the flight stack — telemetry, GPS and safety, RC / ELRS inputs, CAN-FD / DroneCAN, 100BASE-T1 Ethernet, and the ESC outputs — are described in the Hardware Overview and the MR-VMU-Tropic Hardware Architecture and Reference Design.

For the PX4 build, boot, and usage details, see the PX4 documentation reference.

41 Abbreviations

Acronym	Description
CAN FD	Controller Area Network Flexible Data-rate
DShot	Digital Shot (digital ESC control protocol)
ELRS	ExpressLRS (radio-control link)
ESC	Electronic speed controller
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
MCU	Microcontroller unit
OSD	On-screen display
PX4	PX4 autopilot flight stack
RC	Radio control
RTOS	Real-time operating system
SPI	Serial Peripheral Interface
UART	Universal asynchronous receiver-transmitter
USB	Universal Serial Bus
VMU	Vehicle management unit

42 References

- MR-VMU-Tropic product page — overview, documentation, software, tools, and ordering information
- MR-VMU-Tropic Hardware Overview — connectors, pinouts, and on-board devices
- MR-VMU-Tropic Hardware Architecture and Reference Design
- MR-VMU-Tropic Deploy and Run Quick Start
- PX4 documentation reference
- PX4 Autopilot

43 Application Notes

Hardware- and design-level application notes for the MR-VMU-Tropic: the flight controller's hardware architecture and reference design, and the boot configuration fuse programming procedure, aimed at hardware engineers integrating the board into a vehicle.

- Hardware Architecture and Reference Design
- Programming Boot Configuration Fuses

Hardware Architecture and Reference Design

The MR-VMU-Tropic is the flight-controller (vehicle management unit) reference design of the NXP Mobile Robotics family. It is a PX4 / NuttX autopilot built around an NXP i.MX RT1064 crossover MCU, providing the real-time flight-control and I/O functions of a drone or mobile robot. This application note describes the system hardware architecture, the main functional subsystems, and the hardware integration considerations for engineers designing the MR-VMU-Tropic into a product.

This document is hardware focused. It describes the architecture and the role of each subsystem rather than the pin-level connector detail. For connector pinouts, net names, and i.MX RT1064 pad assignments, see the MR-VMU-Tropic hardware reference. The flight-stack bring-up and configuration are covered in the MR-VMU-Tropic flight stack documentation and the NXP Robotics SDK documentation.

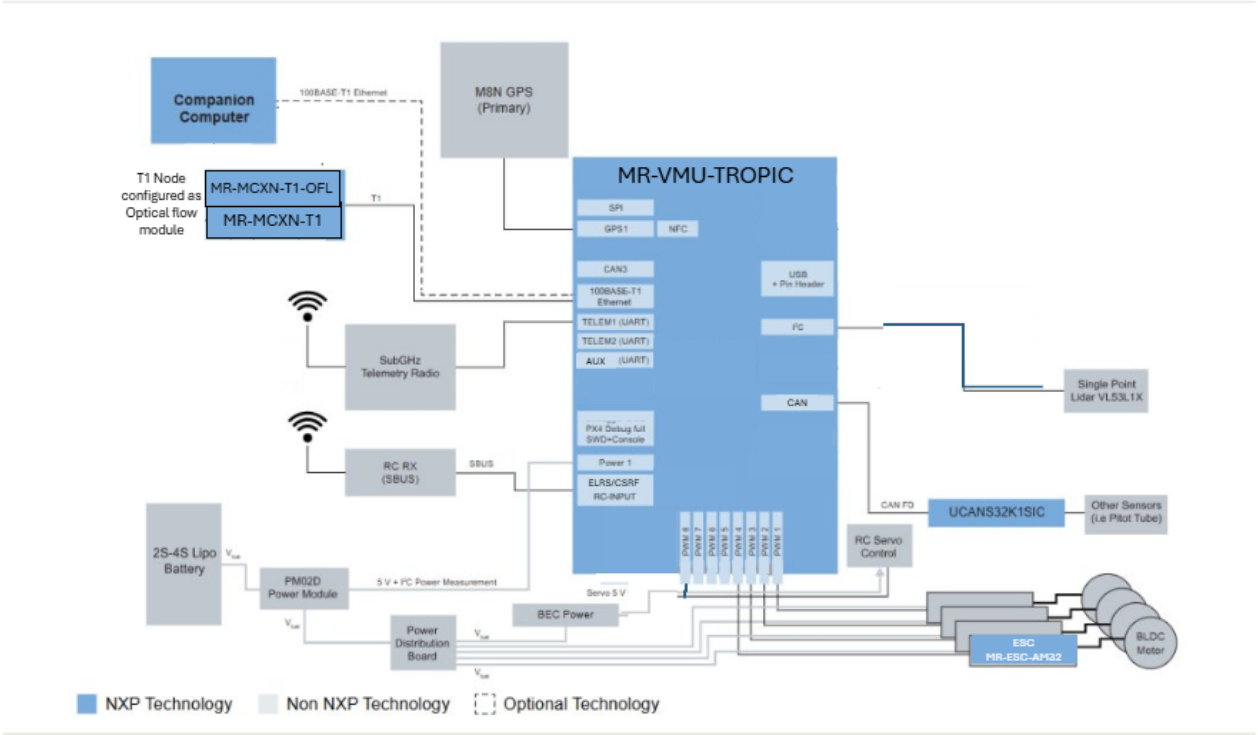


Figure 1. Typical small drone system using MR-VMU-Tropic

1. Introduction

The MR-VMU-Tropic integrates the processing, sensing, power, and I/O required for autonomous flight and motion control onto a single compact flight controller. It is intended as a validated starting point that customers evaluate, reproduce, and adapt for their own vehicles.

The board is organized around the i.MX RT1064 crossover MCU, which runs the PX4 flight stack on the NuttX RTOS and drives the motor outputs, sensor interfaces, and communication links of the vehicle. The connector set follows established drone-ecosystem standards so that standard cabling and peripherals fit.

2. System hardware architecture

The functional block diagram below shows how the i.MX RT1064 MCU connects to the sensor suite, power stage, communication links, and motor outputs.

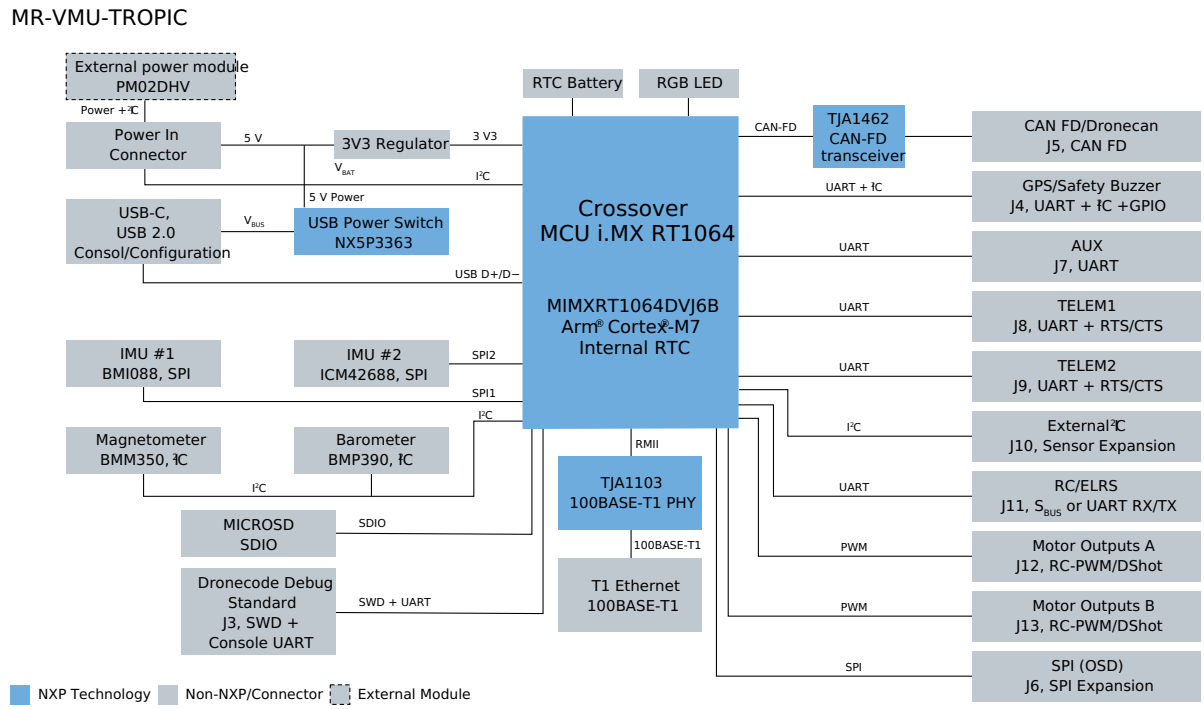


Figure 2. MR-VMU-Tropic functional block diagram

At the system level the design is partitioned into:

- Processing core: the i.MX RT1064 crossover MCU running the flight stack.
- Power: the wide-range input stage based on the NXP PM02DHV power module and the regulated rails derived from it.
- Sensing: the redundant inertial sensors, magnetometer, and barometer used for state estimation.
- Communication: CAN-FD / DroneCAN, 100BASE-T1 automotive Ethernet, UART telemetry, GPS and safety, and RC / ELRS inputs.
- Actuation: the RC-PWM / DShot motor outputs and the SPI OSD expansion.

Each subsystem is described in the following sections. The physical connector that carries each interface, together with its pinout, is listed in the hardware reference; it is not repeated here.

3. Processing core

The processing core is the NXP i.MX RT1064, a crossover MCU based on an Arm Cortex-M7 core with integrated flash and an on-chip real-time clock. It runs the PX4 flight stack on the NuttX RTOS and performs the deterministic sensor fusion, state estimation, and control-loop execution required for flight and motion control. Programming and debug use the Dronecode debug connector; a USB-C port provides the console. The connector pinouts are listed in the hardware reference.

4. Power

The board is powered from a wide input range suitable for typical robotics batteries through the NXP PM02DHV or compatible power module, which outputs a regulated 5 V and also provides battery voltage and current sensing to the flight controller over an I2C interface. The measured power covers all devices attached to and passing through the PM02DHV, including the motors.

The Tropic on-board regulators then derive the 3.3 V and other rails required by the MCU and the connected peripherals. In addition to the power-module input, the board can run from USB during bench work. An NX5P3363UKZ USB power-distribution switch allows cost-effective coexistence of 5 V power from both USB and the power input connector.

> Warning: Every 5 V connector pin other than the power input is an output. Power the board through the power-module connector or USB only; never feed power into a 5 V output pin.

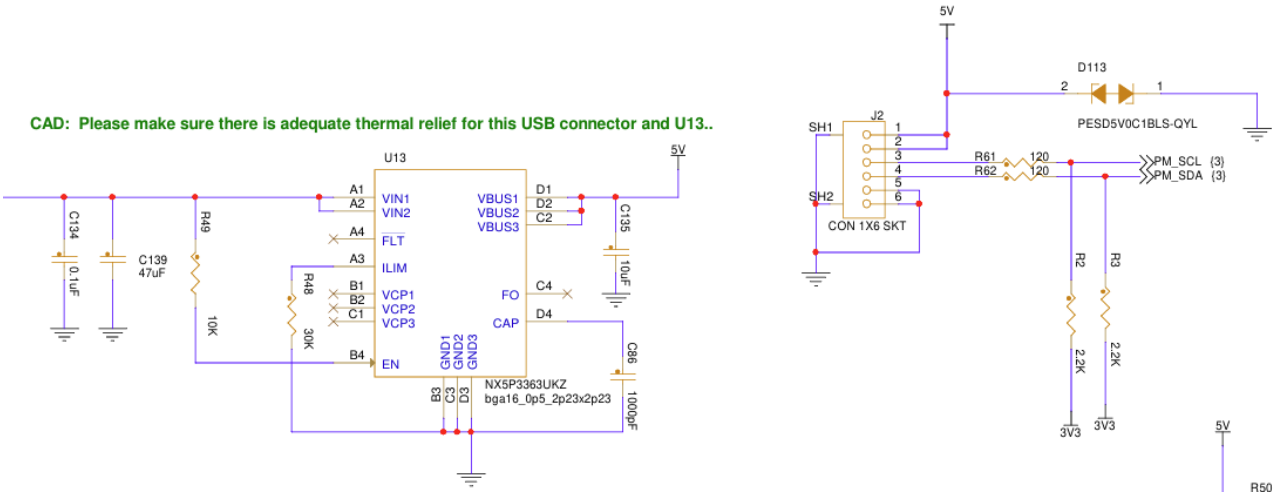


Figure 3. PX4 power input connector J2 and the U13 USB PD switch allowing dual input sources

Input limits, protection behavior, and regulator ratings must be confirmed against the released schematic for the board revision in use.

5. Sensing

The board carries a redundant set of inertial sensors, a magnetometer, and a barometric pressure sensor for attitude, heading, and altitude estimation. The redundancy supports the integrity requirements of a flight controller.

> Note: The barometric pressure sensor may be either a Bosch BMP390 or a Bosch BMP388, depending on component availability at manufacturing time. The BMP388 is a drop-in, functionally equivalent replacement, and the software supports both components. No user action is required regardless of which part is fitted.

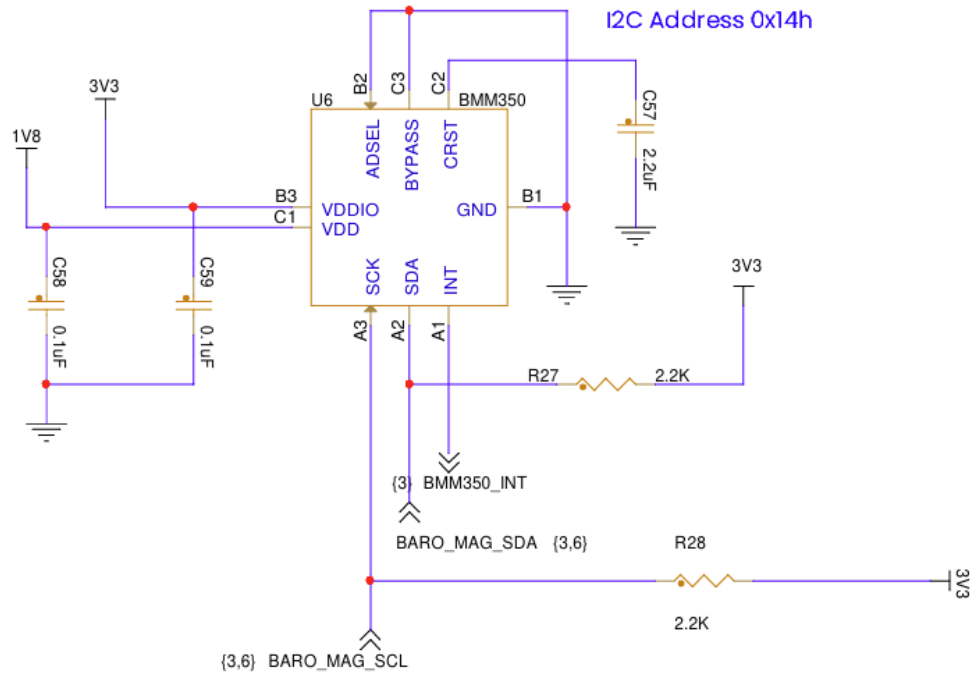


Figure 4. BMM350 magnetometer on I2C address 0x14

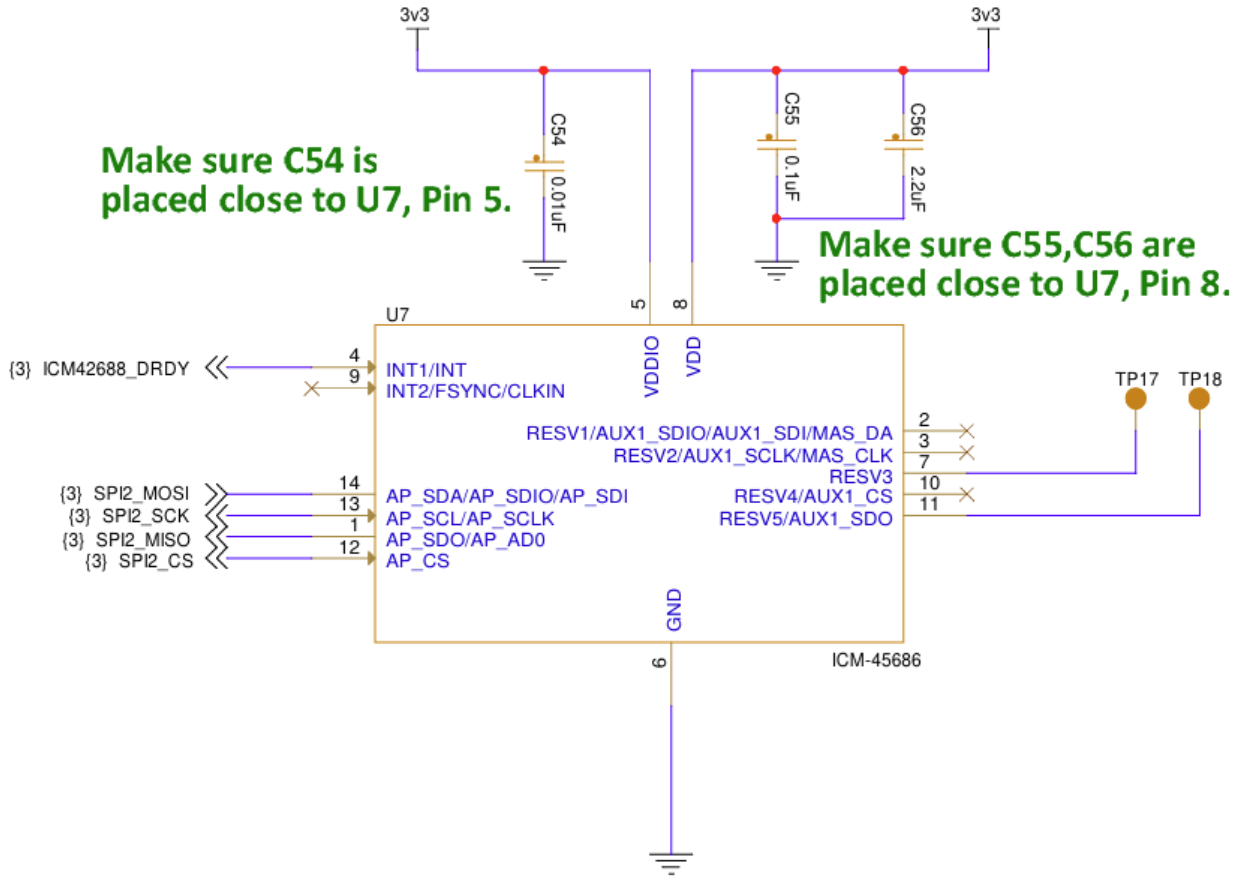


Figure 5. TDK InvenSense ICM-45686 6-axis accelerometer and gyroscope on SPI2 with DRDY interrupt

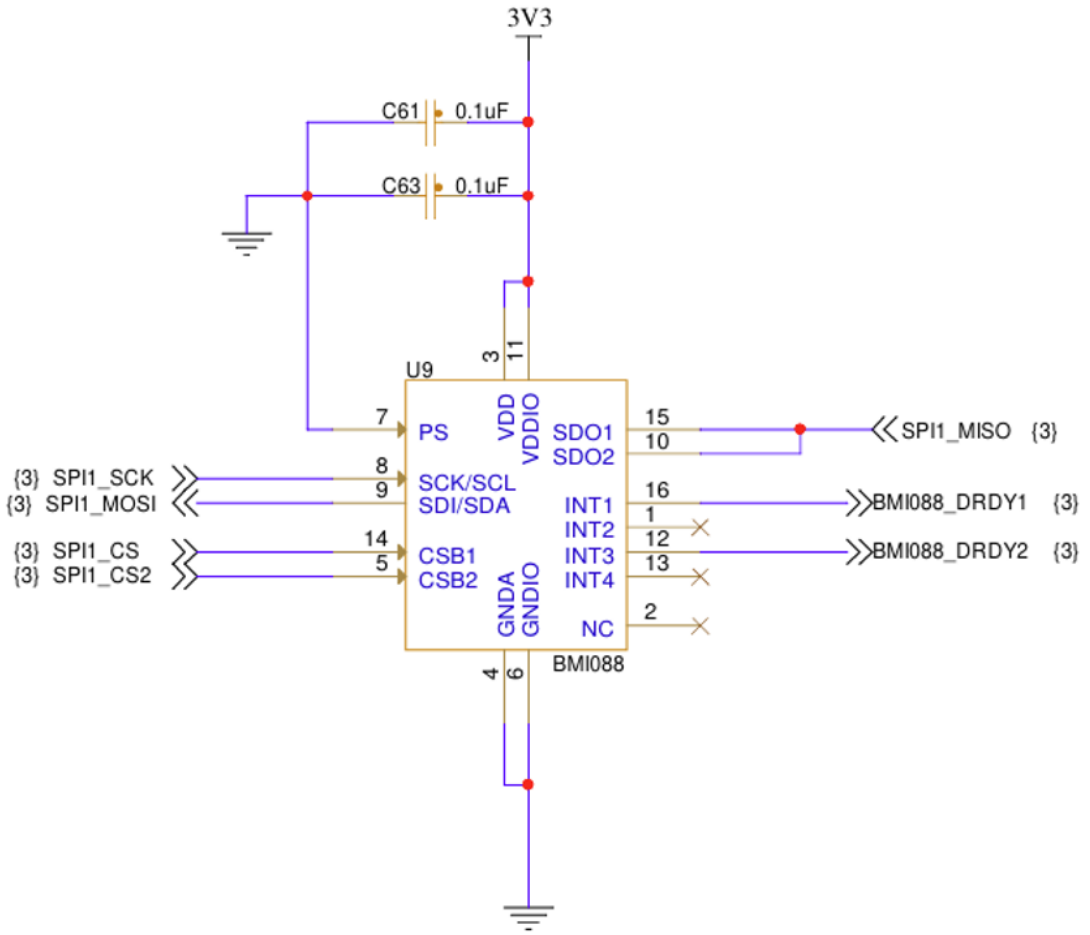


Figure 6. Bosch BMI088 6-axis accelerometer and gyroscope on SPI1 with dual DRDY interrupts

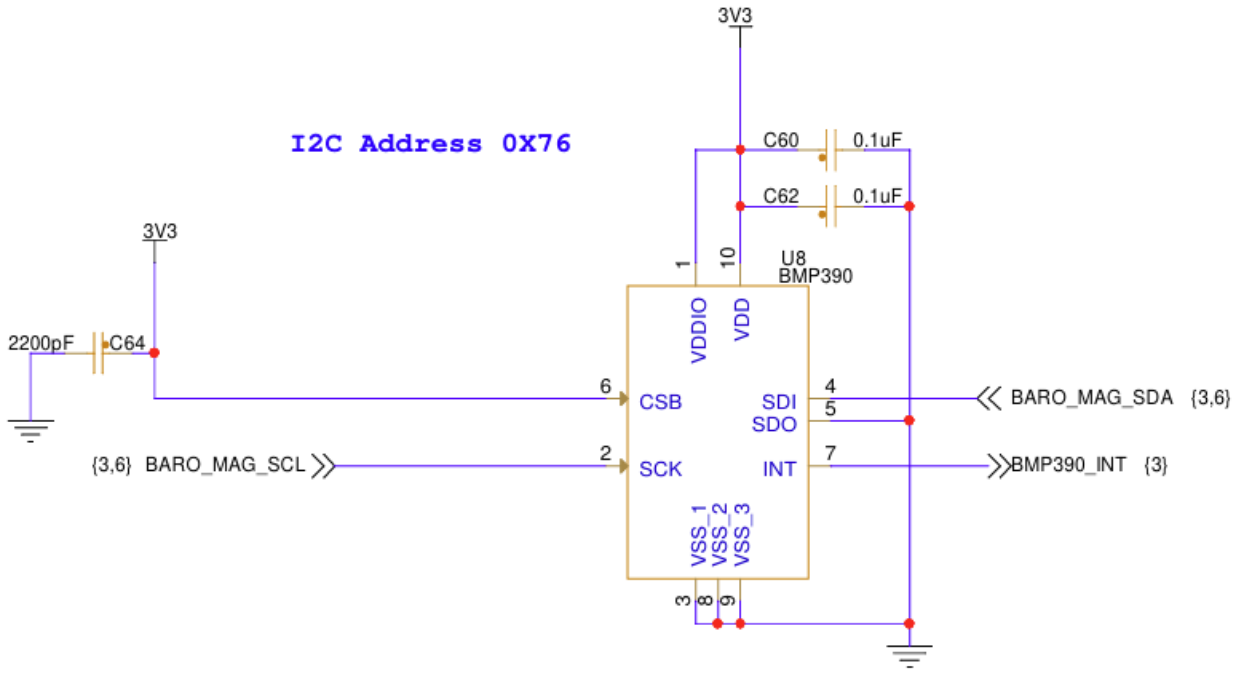


Figure 7. Bosch BMP390 barometric pressure sensor (altimeter) at I2C address 0x76

6. Communication interfaces

The MR-VMU-Tropic provides the communication links a vehicle management unit needs:

- CAN-FD / DroneCAN through an NXP TJA1462 transceiver, for distributed peripherals such as GPS, power, and smart ESCs.
- 100BASE-T1 automotive Ethernet through an NXP TJA1103 PHY, connecting the flight controller to the wider time-sensitive-networking fabric.
- Dronecode-compliant standard dual UART telemetry links, SPI, I2C, a Debug full interface, and a GPS and safety-switch port with arming, buzzer, and remote LED control.
- Bidirectional RC / ELRS port, with the pinout to be validated for specific radio-control and telemetry radios.
- An additional AUX UART following the pinout standard but without flow control.
- An on-board RGB LED for signalling status, as defined in PX4 and other flight stacks.

Where possible and defined, connectors follow the Dronecode DS-009 connector standard. This means the majority of standard drone peripherals attach without adjustment. New devices should be checked beforehand, however, since they may use a non-standard pinout. The connector set and pin order are documented in the hardware reference.

CAN-FD

The CAN-FD interface is provided using an NXP TJA1462 CAN SIC (Signal Improvement Capability) transceiver. Its active signal-improvement function suppresses reflections and ringing, making the network more tolerant of stubs and non-ideal termination while supporting larger, more flexible topologies. This includes a centrally terminated star topology, which can simplify cabling to distributed nodes such as ESCs in a drone.

Note that the Tropic does populate termination resistors by default.

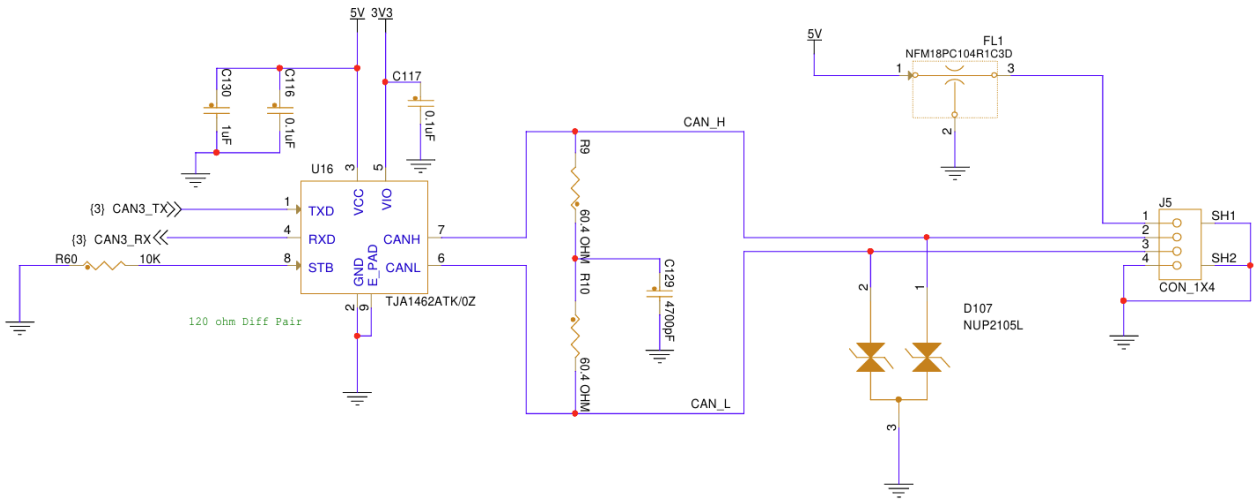


Figure 8. CAN-FD featuring a CAN-SIC signal-improvement PHY

100BASE-T1 Ethernet

The MR-VMU-Tropic provides a 100 Mbit/s 100BASE-T1 Ethernet interface using the NXP TJA1103 automotive Ethernet PHY. 100BASE-T1 carries bidirectional Ethernet traffic over a single twisted pair, providing a compact, high-speed network connection well suited to drones and other mobile robots. The TJA1103 supports 100 Mbit/s transmit and receive operation over unshielded twisted-pair cabling (TJA1103 data sheet).

Unlike conventional 100BASE-TX Ethernet, 100BASE-T1 does not require Ethernet isolation magnetics. The interface can instead use a small passive filtering, coupling, and protection network. This reduces component size and weight, making the interface particularly attractive in a drone where cable mass, connector size, and PCB area are important system considerations.

The T1 differential pair has a defined polarity, so the cabling and connectors should preserve the TRX positive and negative orientation. However, the TJA1103 provides polarity detection and correction, allowing the PHY to compensate for an accidentally reversed pair when that feature is enabled.

The board also provides status LEDs associated with the T1 interface for convenient visual indication of network state and traffic.

This section follows the standard TJA1103 reference design and datasheet guidance.

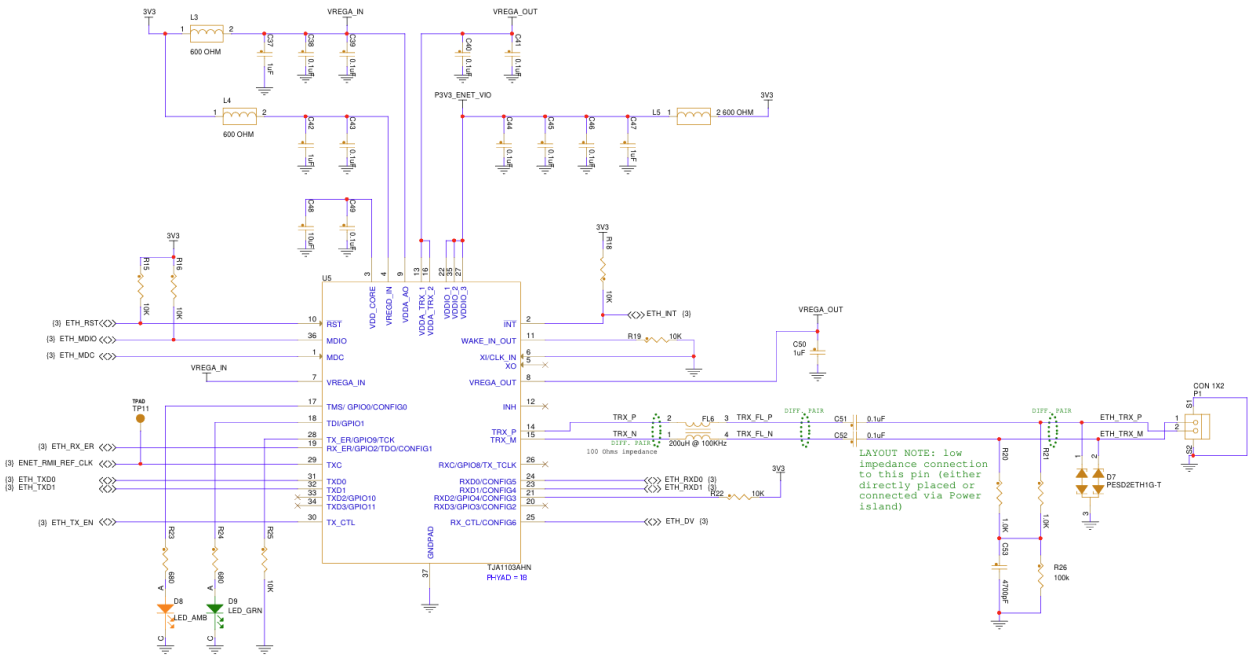


Figure 9. 100BASE-T1 two-wire Ethernet using the TJA1103

RC / ELRS port

The MR-VMU-Tropic provides a dedicated RC / ELRS interface for connecting a radio-control receiver. The interface supports both a conventional SBUS receiver input and a UART RX/TX connection suitable for ExpressLRS (ELRS) receivers using the CRSF protocol.

A simple SBUS receiver requires only power, ground, and a single signal from the receiver to the flight controller. The 5 V supply is provided through a resettable fuse to avoid the possibility that a shorted receiver could impact the rest of the board.

A typical connection is therefore:

Receiver	MR-VMU-Tropic
5 V	5 V
GND	GND
SBUS OUT	UART RX

An ELRS receiver normally uses CRSF over a UART and therefore uses both RX and TX. The receiver's TX carries RC commands to the flight controller, while its RX accepts telemetry from the flight controller for transmission back over the ELRS radio link. ExpressLRS documentation specifies this crossed UART connection explicitly.

ELRS receiver	MR-VMU-Tropic
VCC	5 V
GND	GND
TX	UART RX
RX	UART TX

This makes the ELRS connection bidirectional: RC commands travel from the operator to the flight controller, while telemetry such as vehicle and link information can travel back through the receiver to the operator.

Validate this pinout with the particular remote controller or transceiver used in the application.

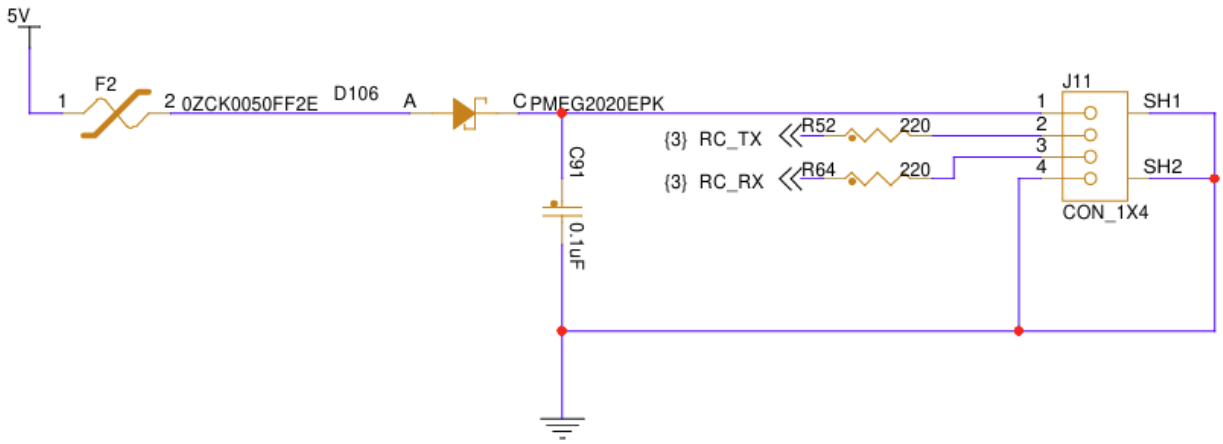


Figure 10. RC / ELRS connector and interface details

On-screen display (OSD)

The MR-VMU-Tropic provides an SPI interface on connector J6 for connection to an external OSD device. In a typical analog FPV system, this can be a MAX7456-class video-overlay device controlled by the flight controller over SPI. The OSD device is inserted into the video path between the FPV camera and video transmitter and overlays flight information such as battery status, flight mode, altitude, navigation information, and warnings onto the pilot's video. The video signal itself does not pass through the MR-VMU-Tropic.

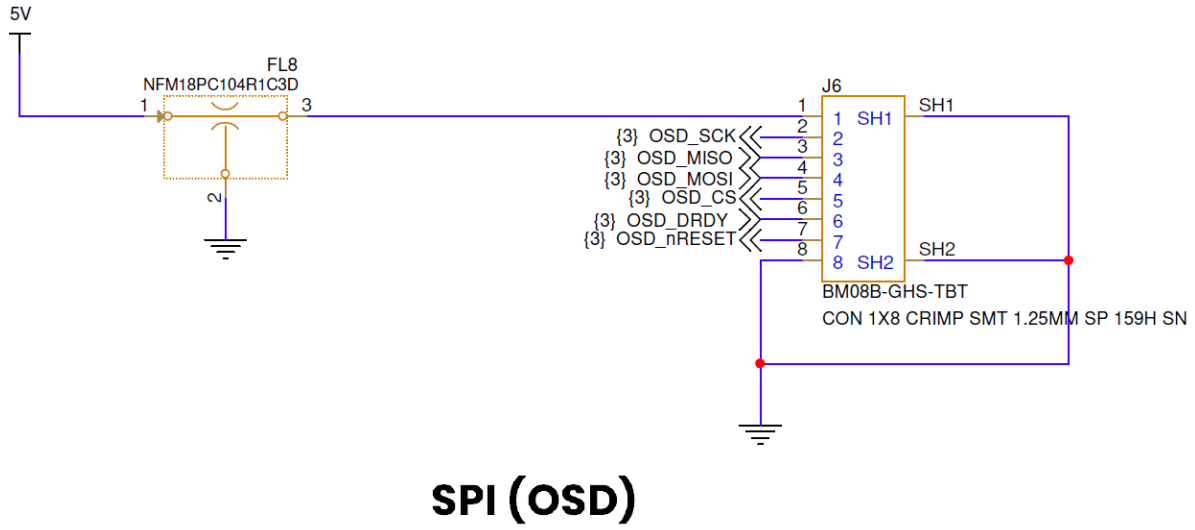


Figure 11. SPI typically used for on-screen display

7. Actuation (PWM)

Motor commands are driven out of the two ESC headers, which follow the Betaflight ESC connector standard so that a four-in-one ESC with a standard cable fits. The outputs support both RC-PWM and DShot motor-control protocols.

8. Hardware integration and bring-up

- Confirm the board part number and revision.
- Confirm the input supply range and polarity against the released schematic before applying power.
- Verify connector orientation and mating; the connector pinouts are in the hardware reference.
- Bring the board up using a current-limited supply, and verify the expected rails before connecting it to the vehicle.
- Observe electrostatic-discharge precautions and provide mechanical support and strain relief appropriate to the installation.

Flight-stack bring-up and configuration are covered in the MR-VMU-Tropic flight stack documentation.

9. Schematic, board layout, and bill of materials

This application note provides a system-level hardware overview and should be used together with the released design files. Electrical limits, connector pin assignments, signal direction, power-domain relationships, and assembly options must be confirmed against the released schematic and bill of materials for the applicable board revision.

10. Abbreviations

Acronym	Description
CAN FD	Controller Area Network Flexible Data-rate
CRSF	Crossfire (ELRS serial protocol)
DShot	Digital Shot (digital ESC control protocol)
ELRS	ExpressLRS (radio-control link)
ESC	Electronic speed controller
FMU	Flight management unit
GNSS	Global Navigation Satellite System
MCU	Microcontroller unit
OSD	On-screen display
PHY	Physical-layer transceiver
PWM	Pulse-width modulation
RC	Radio control
RTC	Real-time clock
RTOS	Real-time operating system

Acronym	Description
SBUS	Serial bus (radio-control receiver protocol)
SPI	Serial Peripheral Interface
UART	Universal asynchronous receiver-transmitter
USB	Universal Serial Bus
VMU	Vehicle management unit

11. References

- MR-VMU-Tropic hardware reference — connectors, pinouts, and i.MX RT1064 pad assignments
- MR-VMU-Tropic flight stack documentation — flight-stack bring-up and configuration
- NXP TJA1103 data sheet — 100BASE-T1 automotive Ethernet PHY
- MR-VMU-TROPIC repository — schematics, board layout, and bill of materials

44 Programming Boot Configuration Fuses

This application note describes how to configure and program the i.MX RT1064 boot-selection fuse used by the MR-VMU-Tropic design. It covers creating an MCUXpresso Secure Provisioning Tool workspace, selecting the BOOT_CFG1 fuse field, programming BT_FUSE_SEL to the required value, and generating a reusable programming script for new board builds.

> Important: MR-VMU-Tropic evaluation boards are supplied with the required fuse programming already completed. This procedure is intended for virgin board builds or boards fitted with a new, unprogrammed i.MX RT1064 device.

44.1 1. Scope and prerequisites

Use this procedure during initial bring-up of a newly manufactured MR-VMU-Tropic board or after fitting a new i.MX RT1064 processor. The procedure programs the boot configuration so that the device uses the fuse-defined boot settings expected by the platform.

> Warning: OTP fuse programming is irreversible. Confirm the selected processor, target board, and required fuse value before initiating the burn operation.

> Note: Some source screenshots were captured with an i.MX RT1176 workspace. The workflow is similar, but for MR-VMU-Tropic the MIMXRT1064 processor must be selected.

1.1 Required software and equipment

- NXP MCUXpresso Secure Provisioning Tool
- A host PC connected to the target using the supported debug and provisioning connection
- A powered MR-VMU-Tropic target containing an i.MX RT1064 device

1.2 Software download

Install the NXP MCUXpresso Secure Provisioning Tool from the MCUXpresso Secure Provisioning Tool product page.

44.2 2. Create an i.MX RT1064 workspace

Start the MCUXpresso Secure Provisioning Tool.

In the New Workspace dialog, select the i.MX RT10xx series and then select MIMXRT1064 as the processor.

Select Create to create the workspace.

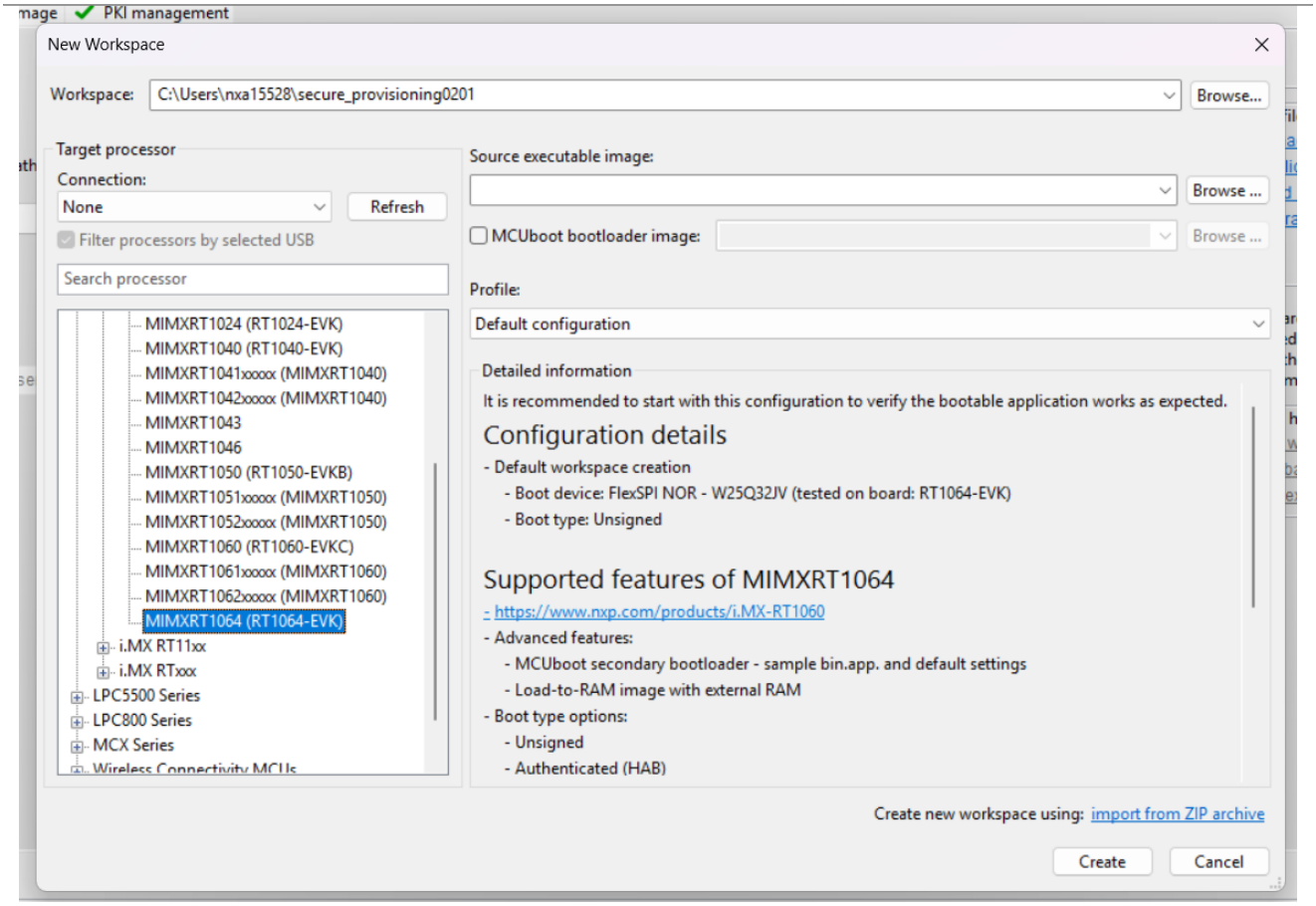


Figure 1. Select MIMXRT1064 when creating the provisioning workspace

44.3 3. Open the OTP configuration

After the workspace opens, select OTP configuration.

If the tool asks to prepare or initialize the workspace, select Yes and wait for the script to finish.

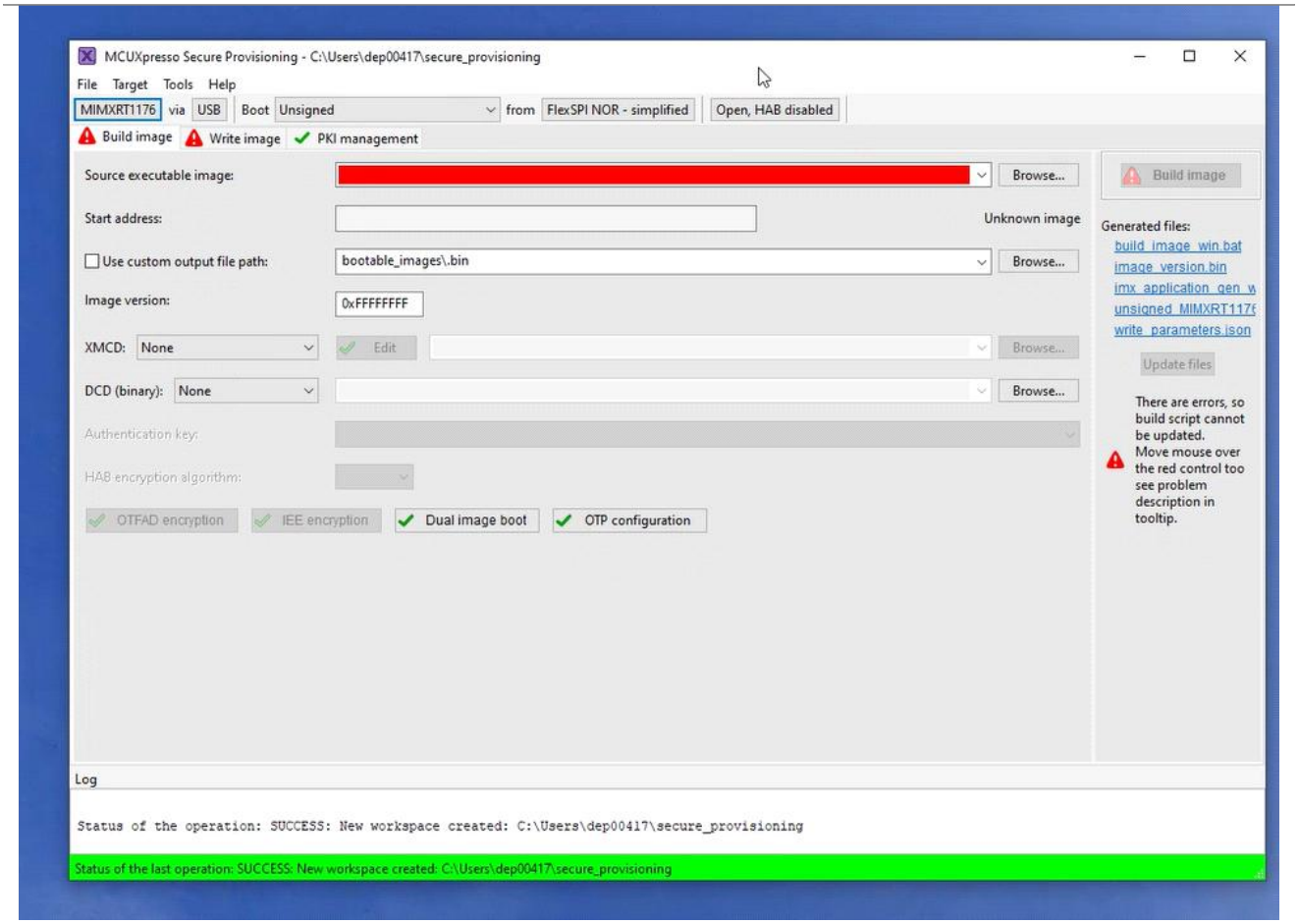


Figure 2. Open OTP configuration from the Secure Provisioning Tool workspace

44.4 4. Select the boot fuse field

In the OTP Configuration window, select BOOT_CFG1.

Locate the BT_FUSE_SEL bit-field and set its Required value to 1.

Enable Advanced mode using the control in the lower-left corner.

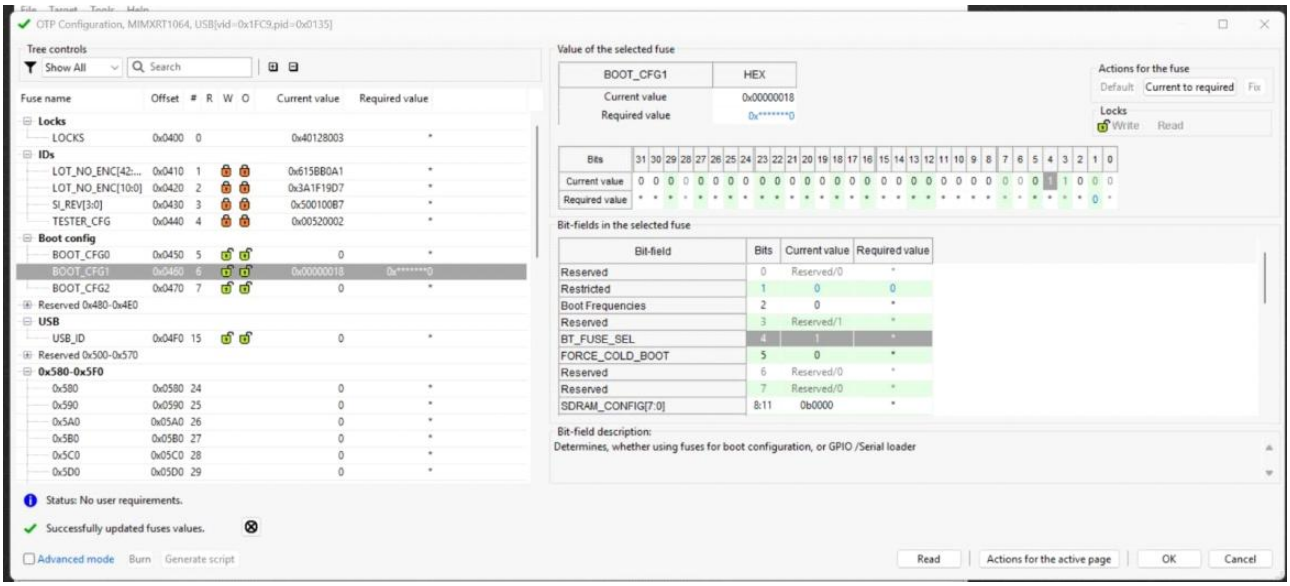


Figure 3. Select BOOT_CFG1 and set BT_FUSE_SEL to the required value of 1

> Warning: Only program the fuse value documented for this application. Do not copy other fuse values visible in screenshots.

44.5 5. Program the fuse

With the target connected and the required value reviewed, select Burn.

The tool displays an irreversible-operation confirmation.

Select Yes only after confirming the target and configuration.

Wait for the operation to complete successfully.

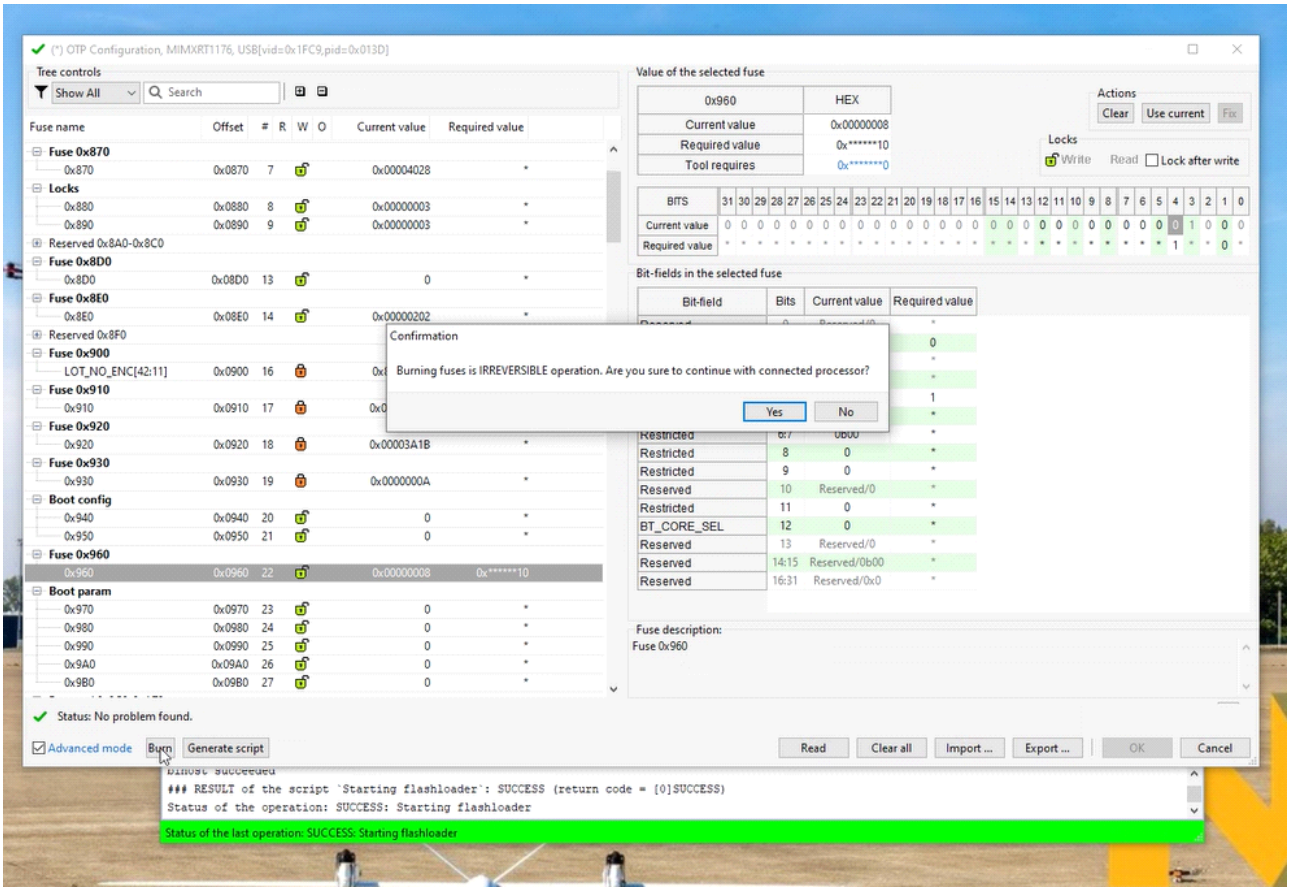


Figure 4. Confirm the irreversible fuse-programming operation

44.6 6. Confirm completion

When the tool reports that OTP configuration has finished successfully, select Close.

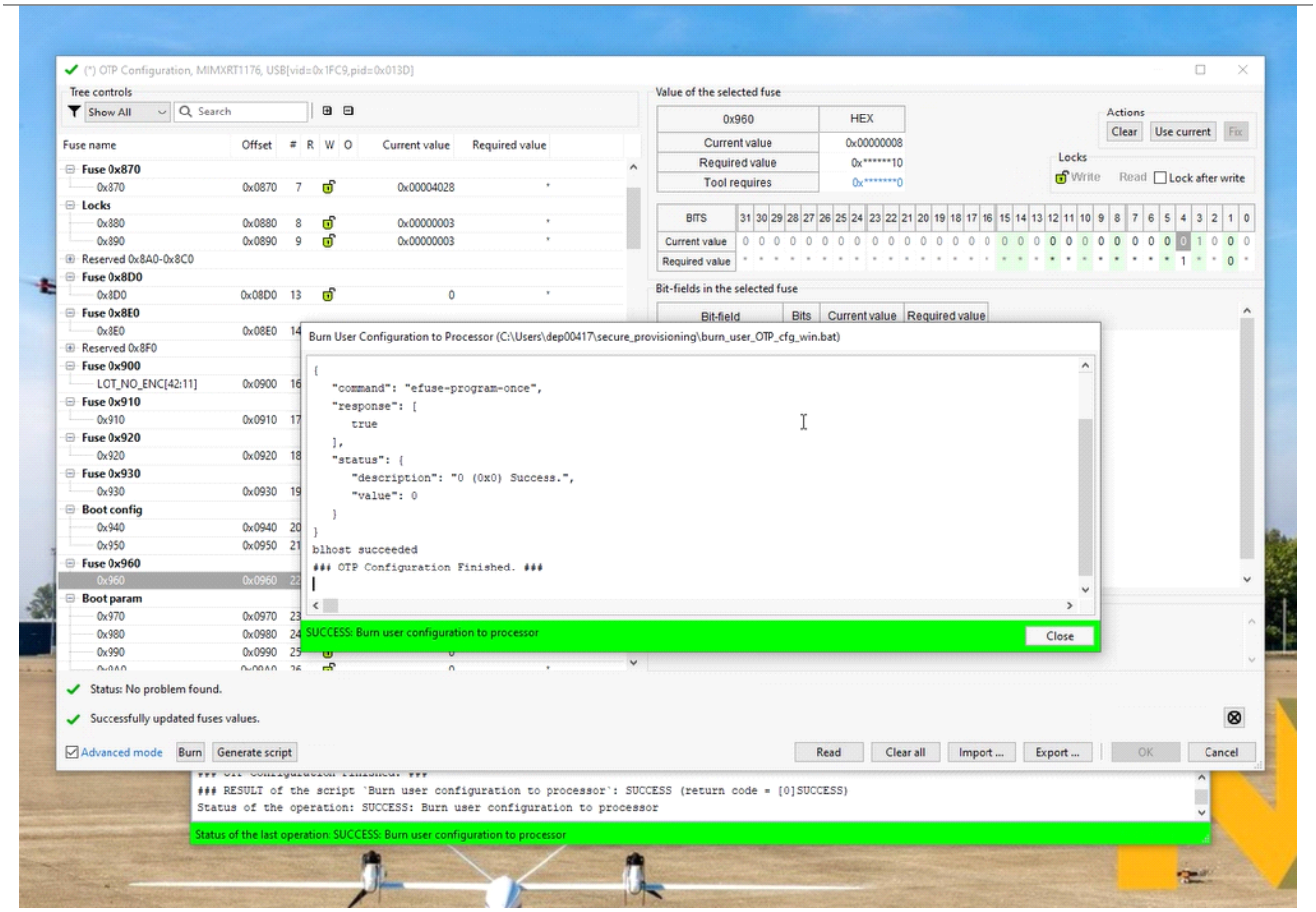


Figure 5. Successful completion of the OTP programming operation

44.7 7. Generate a reusable programming script

Select Generate script.

The generated Windows batch file is placed in the Secure Provisioning Tool workspace.

The default workspace is located under `C:\Users\<username>\secure_provisioning` and the generated file is `burn_user_OTP_cfg_win.bat`.

This script can be used to program the same required fuse configuration on subsequent new boards.

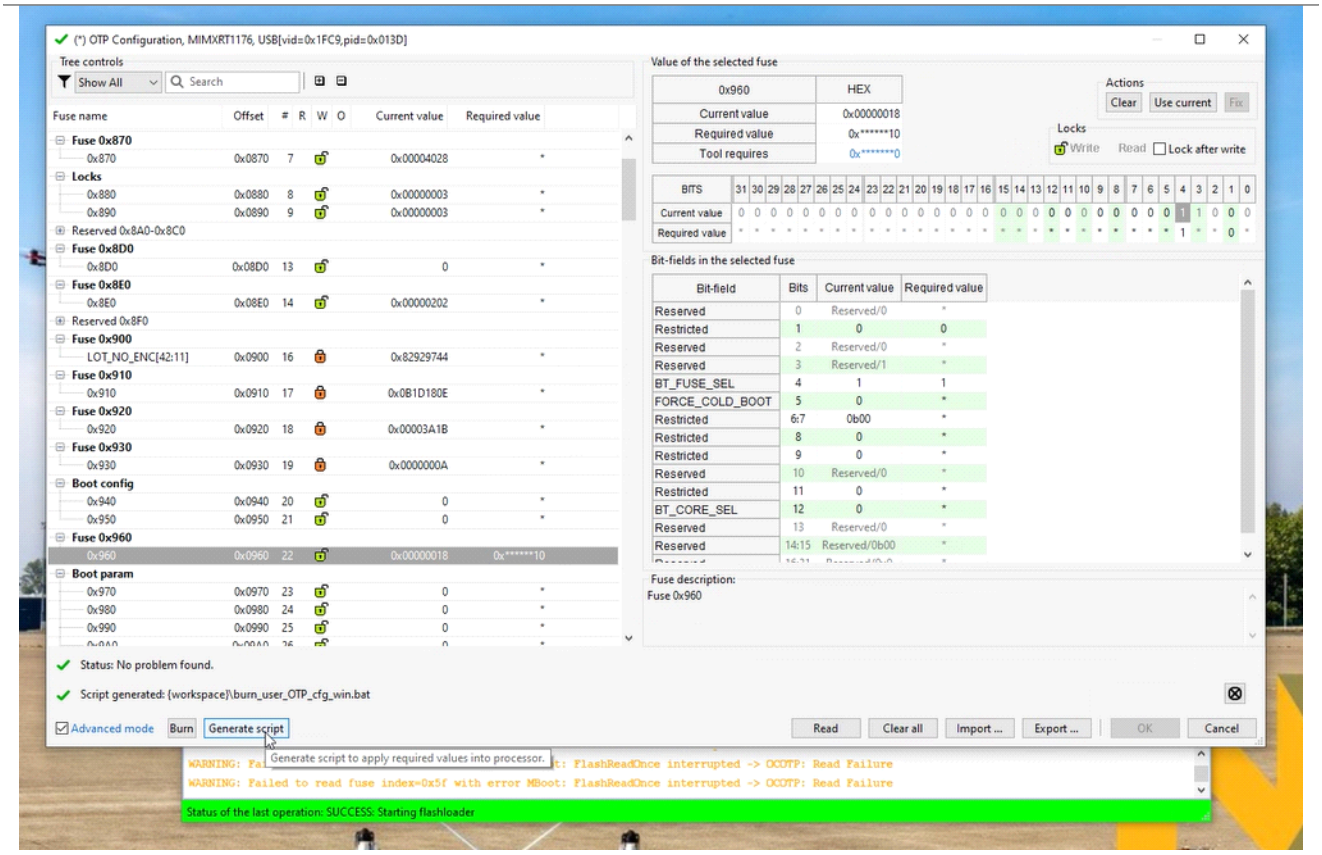


Figure 6. Generate the fuse-programming script from the OTP configuration window

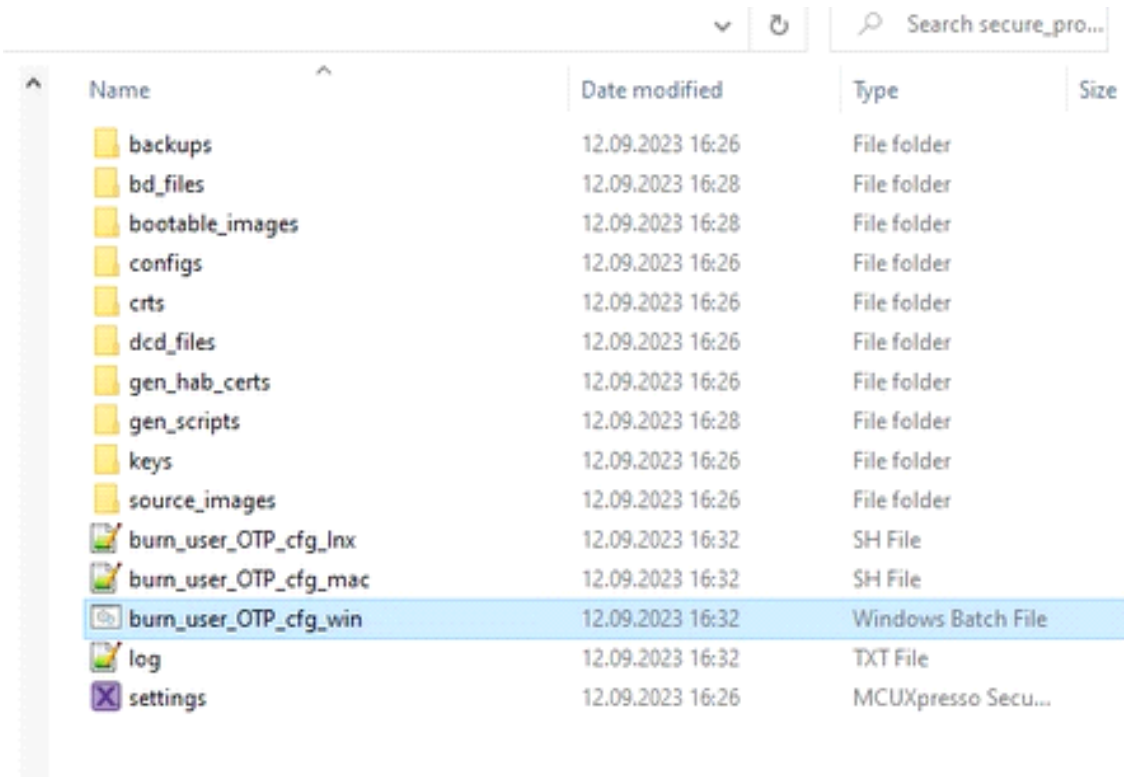


Figure 7. Example generated burn_user_OTP_cfg_win.bat file in the workspace

> Important: Before using a generated script in production, control it as a manufacturing artifact and verify that it was generated from the approved fuse configuration.

44.8 8. Completion criteria

- The operation completes without an error.
- The Secure Provisioning Tool reports successful OTP configuration.
- The generated script, if required, is retained with the controlled manufacturing files.

44.9 9. Abbreviations

Acronym	Description
BOOT_CFG	Boot configuration fuse field
BT_FUSE_SEL	Boot fuse selection bit
EVK	Evaluation kit
FMU	Flight management unit
MCU	Microcontroller unit
OTP	One-time programmable
VMU	Vehicle management unit

44.10 10. References

- MR-VMU-Tropic user manual — flashing the bootloader and application
- MR-VMU-Tropic hardware architecture and reference design — board hardware architecture
- MCUXpresso Secure Provisioning Tool product page — tool download and documentation

45 Flight Stack

The PX4 flight stack running on the MR-VMU-Tropic real-time cores.

- PX4 Documentation Reference

46 PX4 Documentation Reference

PX4 is the open-source software stack that implements the autopilot functionality running on supported NXP Robotic platforms. It runs on the real-time cores of the supported boards and provides the flight stack, vehicle setup, sensor calibration, parameter configuration, and communication interfaces (uORB, MAVLink, and more) that our robotics use cases build on.

For detailed, version-specific guidance, refer to the official PX4 User Guide for version 1.17, which covers hardware setup, configuration, flight modes, development, and troubleshooting:

- PX4 User Guide (v1.17)

47 MR-MCXN-T1

The MR-MCXN-T1 is the MCX N sensor / shield node of the NXP Mobile Robotics family (GNSS, IO, OFLO). Start with the deploy-and-run quick start.

- Overview
  - Functional block diagram
  - Main components
- Getting Started
  - Deploy and Run Quick Start
- User Manual
  - Introduction
  - Finding platform resources
  - Getting ready
  - Getting to know the hardware
  - Configuring the hardware
  - Abbreviations
  - References
- Application Notes
  - Hardware Architecture and Reference Design
  - Zenoh_PX4 Communication on the MR-MCXN-T1 using ZROS

48 Overview

The MR-MCXN-T1 Hub is the sensor and I/O node reference design of the NXP Mobile Robotics family. It is a networked node built around an NXP MCXN947 microcontroller that aggregates sensors and distributes them over automotive 100BASE-T1 Ethernet and CAN-FD, acting as a distributed sensor / I/O hub in a time-sensitive networking (TSN) robotics system. The board is intended as a validated node design that customers can evaluate, reproduce, and extend with their own shields.

The MCXN947 is an NXP Arm Cortex-M33 microcontroller with integrated analog and communication peripherals. It runs Zephyr RTOS or the Cerebri (CogniPilot) ROS 2 stack and bridges on-board sensors to the robotics network through its Ethernet and CAN-FD interfaces.

48.1 Functional block diagram

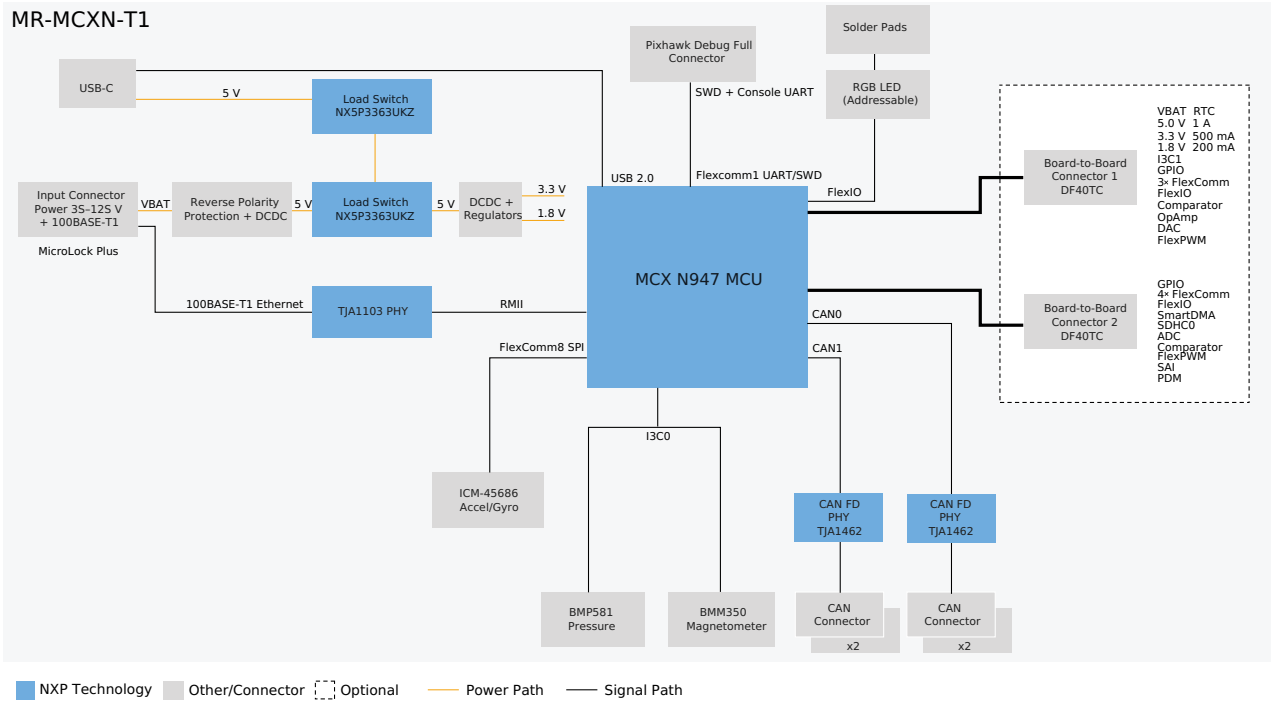


Figure 1. MR-MCXN-T1 Hub functional block diagram

48.2 Main components

The reference design integrates the following main NXP components and interfaces:

- MCXN947 microcontroller (Arm Cortex-M33)
- 3.3 V and 1.8 V regulation from a 3S to 12S input through NXP NX5P3363 load switches and a DCDC regulator
- ICM-45686 inertial sensor, BMM350 magnetometer, and BMP581 barometer
- 100BASE-T1 automotive Ethernet through the NXP TJA1103 PHY
- Two CAN-FD channels through NXP TJA1462 transceivers
- USB 2.0, addressable RGB LED, and an SWD / console debug connector

- Two DF40TC board-to-board connectors exposing FlexComm, FlexIO, ADC, comparator, DAC, SAI, PDM, and FlexPWM for shields

NXP also provides the complete hardware Bill of Materials and design files, enabling customers to evaluate, reproduce, and customize the reference platform for their own products.

For overview information, documentation, software, tools, and ordering details, see the MR-MCXN-T1 product page on the NXP web site.

49 Getting Started

Deploy and run on the MR-MCXN-T1 sensor / shield node.

- Deploy and Run Quick Start

50 Deploy and Run Quick Start

Including the MR-MCXN-T1-OFLO, MR-MCXN-T1-GNSS and MR-MCXN-T1-IO shield variants.

50.1 Overview

This guide explains how to obtain MR-MCXN-T1 firmware, either prebuilt or built from source with the NXP Robotics SDK, and how to flash it using NXP LinkServer. It applies to both Zephyr and Cerebri (CogniPilot) stacks.

Two application stacks are available, both built via `west sdk build --stack`:

Stack	`--stack`	What it is	Typical use
Zephyr	`zephyr` (default)	Stock Zephyr samples for this board	Peripheral bring-up and validation
Cerebri (CogniPilot)	`cerebri`	CogniPilot ROS 2 / zenoh sensor-node apps	Running the board as a robotics sensor node

Each build produces two images — the MCUboot bootloader and the signed application — which must be flashed to different addresses.

50.2 Download or build the artifacts

Every platform assumes the images are already available — either downloaded or built with the SDK.

- Docker required. Downloading and building are done inside a Docker container built from the SDK's `Dockerfile` — see the robotics-sdk README for how to set it up. All `west` commands are assumed to run inside that container.
- The SDK builds/downloads images but does not flash them — deployment is done with each platform's native tool.
- Where do the artifacts land? `<west topdir>` is the west workspace root — the directory containing `.west/`, created by `west init`. The `bin/` directory is created next to the `robotics-sdk/` clone, not inside it:

```
<west topdir>/
├── .west/
├── robotics-sdk/     <- the SDK clone (west init -l)
├── build/                <- west sdk build intermediates
└── bin/mr-mcxn-t1/       <- final artifacts land here
```

```
# prebuilt (fastest)
west sdk download --platform mr-mcxn-t1      # lands in <west topdir>/bin/mr-mcxn-t1/

# or build from source
west sdk build --platform mr-mcxn-t1  # lands in <west topdir>/{build,bin}/...
```

Cerebri (CogniPilot) application

Besides the stock Zephyr samples, the MR-MCXN-T1 can run Cerebri, the CogniPilot flight/robotics application layer, built with the Cerebri stack. Each Cerebri app maps to a specific shield, and the SDK injects the matching shield overlay automatically — so `--shield` is not passed:

App	Implied shield
`optical_flow`	`oflo` (optical-flow shield)
`gnss`	`f9p` (u-blox F9P GNSS shield)
`io_exp`	`io_exp` (I/O expander)

Build (or download) a Cerebri app:

```
# prebuilt
west sdk download --platform mr-mcxn-t1

# or build a specific Cerebri app from source
west sdk build --platform mr-mcxn-t1 --stack cerebri optical_flow
west sdk build --platform mr-mcxn-t1 --stack cerebri            # all Cerebri apps
```

For MR-MCXN-T1 the platform is `mr-mcxn-t1`. Run `west sdk list` for the authoritative list.

More details on how to build the examples can be found in the robotics-sdk README.

50.3 What to flash

Every MR-MCXN-T1 example is a sysbuild build producing two images. The bootloader is typically programmed once, and during normal development, only the application image needs to be updated.

Image	Build path (sysbuild domain)	Collected as	Absolute flash address
MCUboot bootloader	`build/mr-mcxn-t1/<stack>/<shield>/<example>/mcuboot/zephyr/zephyr.bin`	`mcuboot.bin`	`0x10000000` (flash base = `boot_partition`, 80 KB)
Signed application	`build/mr-mcxn-t1/<stack>/<shield>/<example>/<domain>/zephyr/zephyr.signed.bin`	`zephyr.signed.bin`	`0x10014000` (flash base + `slot0_partition` offset `0x14000`)

Collected copies also appear under `<west topdir>/bin/mr-mcxn-t1/<stack>/<shield>/<example>/` as `mcuboot.bin` and `zephyr.signed.bin`.

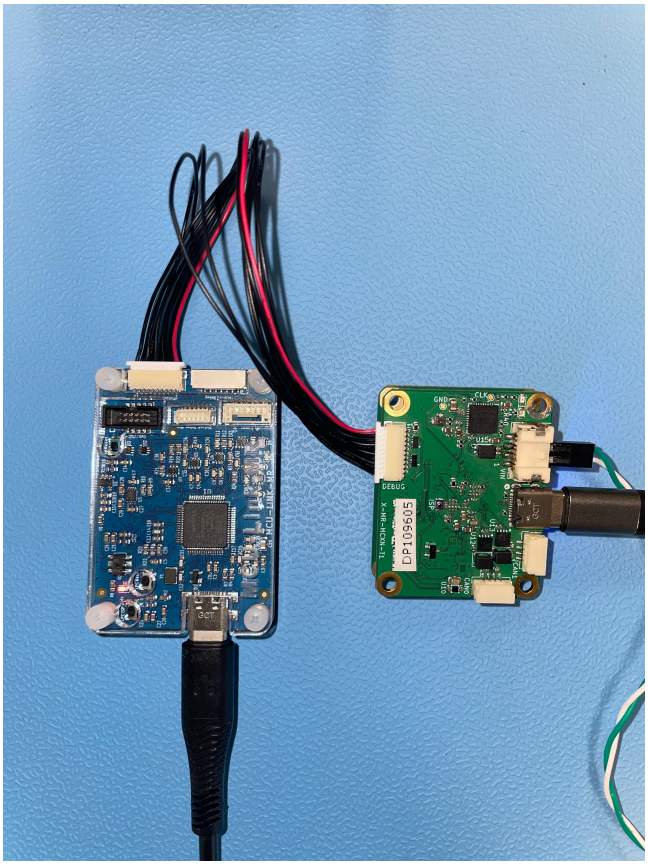
Example:

```
<west topdir>/bin/mr-mcxn-t1/cerebri/oflo/optical_flow/{mcuboot.bin, zephyr.signed.bin}
<west topdir>/bin/mr-mcxn-t1/cerebri/f9p/gnss/{mcuboot.bin, zephyr.signed.bin}
<west topdir>/bin/mr-mcxn-t1/cerebri/io_exp/io_exp/{mcuboot.bin, zephyr.signed.bin}
```

⚠️ Flash the signed application (`zephyr.signed.bin`) at the slot0 address `0x10014000` , not the raw `zephyr.bin` at flash base. MCUboot lives at flash base and chainloads the signed app in slot0. Writing an unsigned image at `0x10000000` will not boot.
ℹ️ The MCUboot bootloader is identical in layout across stacks; on a board that already has the bootloader only `zephyr.signed.bin` needs re-flashing at `0x10014000` when switching between a Zephyr sample and a Cerebri app (as long as both were signed with the same key set).

50.4 Required hardware and connection setup

- MR-MCXN-T1 base board and USB cable into the board's native USB-C connector, for power supply.
- MR-MCXN-T1 shield variant (OFLO, GNSS, or IO) mounted on the MR-MCXN-T1 board.
- MCU-Link-MR used for flash / debug and USB cable for power supply. This connection can be used for serial console output as well. It enumerates as `/dev/ttyACM0` (Linux/macOS) or `MCU-Link VCom Port (COM*)` (Windows) for the CDC-ACM console.
- 10-Pin SH Debug cable to connect the MCU-Link-MR (Pixhawk Debug Full 10 port) to the MR-MCXN-T1 base board (DEBUG port).



50.5 Flash with LinkServer

The LinkServer can be installed from here. Make sure it is installed correctly and visible by running `LinkServer -v` on a terminal.

It is possible to update the firmware of the MCU-Link-MR debugger before flashing, in order to detect the connection properly. To do so, follow the steps from the official documentation.

Open a terminal and follow these steps in order to flash the MR-MCXN-T1:

1. Flash the bootloader

   ```
   LinkServer flash MCXN947 load --addr 0x10000000 mcuboot.bin
   ```

2. Flash the signed app into slot0

   ```
   LinkServer flash MCXN947 load --addr 0x10014000 zephyr.signed.bin
   ```

   Use the absolute address (flash base + slot offset), not the bare partition offset. On Windows the LinkServer path is e.g. `C:\NXP\LinkServer_26.6.137\LinkServer`.

3. Watch the MCU-Link-MR console for serial output

   ```
   screen /dev/ttyACM0 115200      # or: minicom / tio / picocom
   ```

   MCUboot boots, validates the slot0 image, and chainloads it; sample output appears on the CDC-ACM console.

> ⚠️ Two gotchas:
> (a) J-Link's connector does not fit some MR-MCXN-T1 units — use MCU-Link + LinkServer.
> (b) `LinkServer flash` has no remote mode, so run it on the host physically wired to the board (only its gdbserver is network-facing).

51 User Manual

Get to know the MR-MCXN-T1 Hub: review the Hub hardware and the software enablement framework, and how to set up and configure the node.

- Introduction
- Finding platform resources
- Getting ready
- Getting to know the hardware
- Configuring the hardware
- Abbreviations
- References

52 Introduction

This user manual covers the MR-MCXN-T1 Hub and the following add-on boards:

- MR-MCXN-T1-IO: input/output shield with Dronecode standard connectors
- MR-MCXN-T1-F9P: GPS/GNSS shield with RTK capability
- MR-MCXN-T1-OFL: optical-flow shield with a distance sensor and optical-flow measurement

52.1 MR-MCXN-T1 Hub

The MR-MCXN-T1 node board, also referred to as the MR-MCXN-T1 Hub, is a compact, open hardware reference design for distributed sensing, connectivity, and edge processing in mobile robotics systems. It is based on the cost-effective MCXN947 microcontroller and is intended to act as an intelligent 100BASE-T1 Ethernet sensor and actuator node. Supported example shield designs are provided to demonstrate several I/O use cases.

The design demonstrates the convenience of a single-connector sensor node that carries both raw battery power and 2-wire 100BASE-T1 Ethernet to a compact edge device. This simplifies distributed sensor placement on a robot by reducing wiring, keeping local processing close to the sensor, and sending data to a flight controller, companion computer, or ROS 2 system over Ethernet.

The design intent is to provide a seamless Ethernet-connected node for roboticists using ROS 2, and it can be thought of as the physical embodiment of a ROS 2 node.

52.2 NXP Robotics SDK and upstream open-source software

NXP embraces an open-source development model and actively participates in the communities behind the software used by this product. The NXP Robotics SDK provides a curated and validated set of software components, tools, and documentation that have been tested together on this hardware platform.

NXP regularly contributes improvements and fixes back to upstream repositories. Because contributions are made on a scheduled basis, the latest upstream code and the version included in the NXP Robotics SDK may not always be identical. The SDK release is recommended for users seeking a tested and supported configuration, while advanced users may choose to work directly with the upstream project and its latest features.

52.3 Software enablement framework

The software enablement framework uses Zephyr on the MCU and separates local driver work from ROS-oriented data exchange. It provides a path for sensor and actuator data to move between the MR-MCXN-T1 node and a ROS 2-based host system over 100BASE-T1 Ethernet.

ZROS library

NXP is introducing ZROS as a new open-source library that provides the ROS-oriented topic layer on the MCU. It supports preparing, formatting, inspecting, and exchanging sensor or actuator data in a way that feels similar to working with ROS 2 topics on a Linux host. It also includes an efficient zero-copy buffer mechanism.

Data representation

ZROS uses FlatBuffers (FBS) internally for efficient MCU-side data handling. For ROS 2 interoperability, ZROS can statically pack this data into the CDR representation generated from the ROS 2 IDL definitions. Users normally

work through this default ROS 2-compatible path. Customers requiring high performance scenarios can use FlatBuffers instead.

Zenoh-pico

Zenoh-pico provides the transport and routing layer used to move data between the MCU node and the rest of the robotics system.

Developer focus

With this framework in place, the MCU developer can focus primarily on writing the sensor or actuator drivers and moving the required data to or from the specific device.

53 Finding platform resources

The information page for the MR-MCXN-T1 Hub and add-on shields is on the NXP web site at https://www.nxp.com/design/design-center/development-boards-and-designs/MR-MCXN-T1. The information page provides overview information, documentation, software and tools, and a Getting Started tab with quick-reference information, including the downloadable assets referenced in this document.

The board design files and the latest hardware reference are published in the public NXP Robotics repository at https://github.com/NXP-Robotics/mr_mcxn_t1_hub. The software, images, and build tooling are distributed through the NXP Robotics SDK; follow the NXP Robotics SDK README to set up the SDK and obtain the assets.

54 Getting ready

Because the MR-MCXN-T1 Hub is intended to demonstrate 100BASE-T1 connectivity, it must be connected to a host system that supports 100BASE-T1 Ethernet. In a typical setup, this may be an MR-NavQ95 running Linux, but other systems such as the MR-VMU-RT1176 running PX4 or Zephyr can also be used. 100BASE-T1 is the physical Ethernet interface; above that layer, it behaves like standard 100BASE-T Ethernet. For development with a PC or another system that only provides RJ45 Ethernet, an adapter such as the RDDRONE-T1ADAPT can convert between standard 100BASE-T and 2-wire 100BASE-T1.

54.1 Kit contents

- MR-MCXN-T1 Hub

54.2 Additional hardware

The following shields are designed to work with the MR-MCXN-T1 Hub:

- MR-MCXN-T1-IO: input/output shield with Dronecode standard connectors
- MR-MCXN-T1-F9P: GPS/GNSS shield with RTK capability
- MR-MCXN-T1-OFL: optical-flow shield with a distance sensor and optical-flow measurement

54.3 Assumptions

Familiarity with 100BASE-T1 Ethernet is helpful but not required.

54.4 Minimum system requirements

Working with the MR-MCXN-T1 Hub requires one of the following host setups:

- A host device or system such as the MR-NavQ95 running ROS 2 and natively supporting 100BASE-T1 Ethernet
- A host device or system running ROS 2 and using an adapter such as the RDDRONE-T1ADAPT to support 100BASE-T1 Ethernet
- A PC together with an RDDRONE-T1ADAPT, which converts between RJ45-based 100BASE-T and 2-wire 100BASE-T1

54.5 Software

Software for the MR-MCXN-T1 Hub and supported shields is provided through the NXP Robotics SDK. Follow the SDK instructions to prepare the Hub and configure the specific shield being used.

The same enablement is also expected to be visible upstream in Zephyr, including shield overlays for the example shield boards. For more background, see Software enablement framework and the Introduction.

The following software is used with the MR-MCXN-T1 Hub:

- Zephyr development environment
- A PC with ROS 2 Jazzy or later

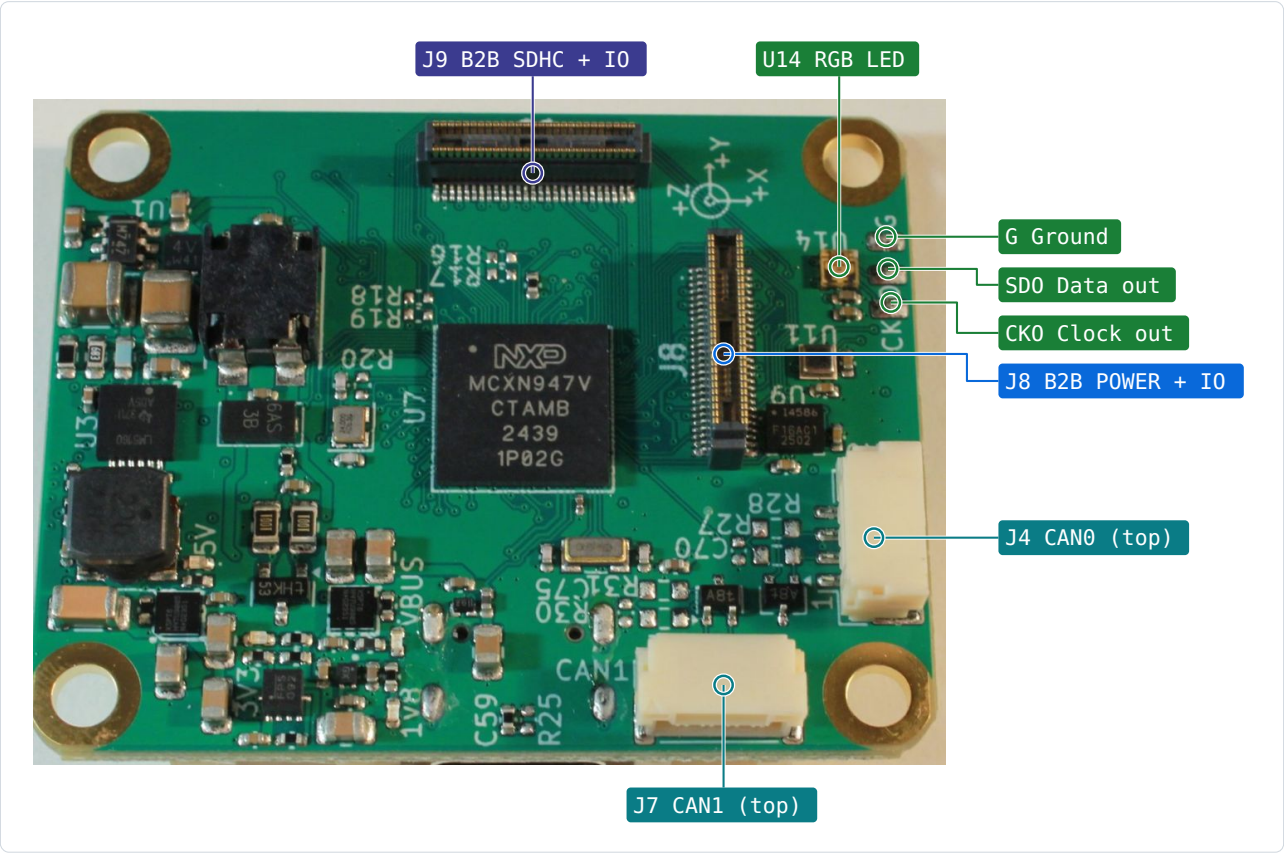
55 Getting to know the hardware

Connectors, solder pads and pinouts of the NXP MR-MCXN-T1 Hub, an MCXN947-based multi-purpose base board with a modular board-to-board add-on interface. The board carries 100BASE-T1 Ethernet, two CAN FD buses, USB, a 9-axis sensor set (IMU, magnetometer, barometer), a debug header and an addressable RGB LED. All pin data below comes from the public KiCad design files in the MR-MCXN-T1 Hub repository.

Each card shows three things per pin: the function, the net name in the schematic and the MCXN947 pad the signal ends up on. Parts in between, such as the CAN transceivers and the Ethernet PHY, are shown too. Pin 1 is the square pad.

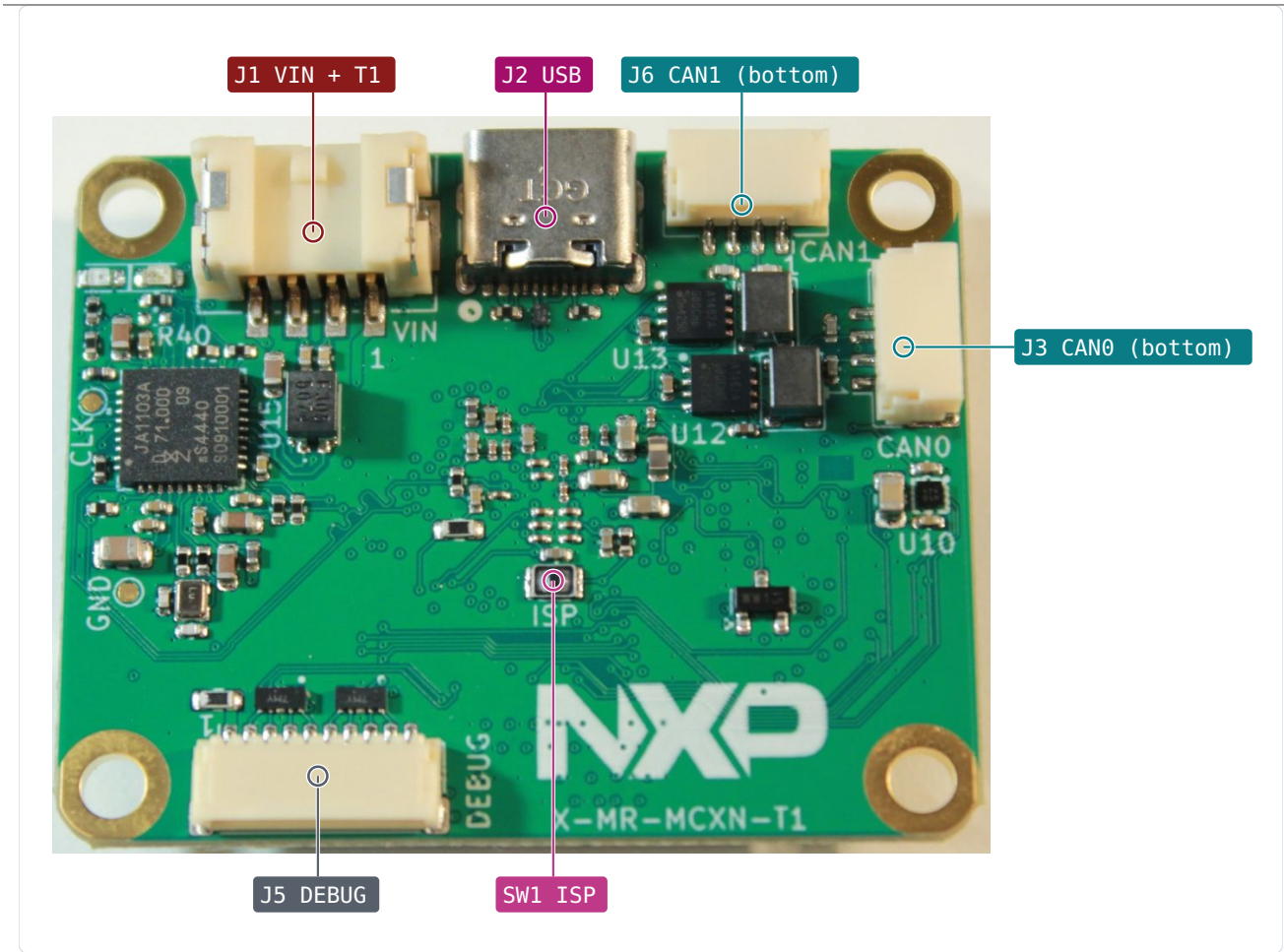
> Tip: The pinout cards are interactive: hover over a pin to highlight it, and click a pin to copy its net/pad name to the clipboard. On the board-overview figures, click a callout to jump to the matching connector card.

55.1 Board overview



Bottom side: J1 VIN + T1 · J2 USB · J5 DEBUG · J13 RTC_BAT · SW1 ISP.

Bottom side, with the power and Ethernet connector, USB, debug port and the bottom CAN connectors. The RTC battery connector J13 was added in the current revision, in the bottom right corner:



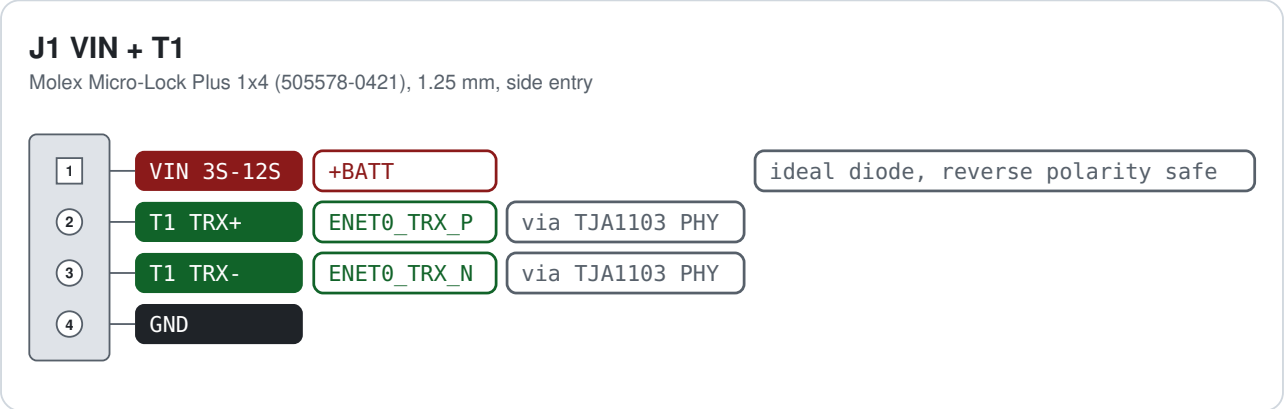
55.2 Connector index

Ref	Function	Connector
Hub J1 VIN + T1	Battery input 3S to 12S and 100BASE-T1 Ethernet on one cable	Molex Micro-Lock Plus 1x4 (505578-0421), 1.25 mm, side entry
Hub J2 USB	USB 2.0 high-speed device port	USB Type-C receptacle (USB4105-GF-A)
Hub J3 CAN0	CAN FD bus 0 (TJA1462), two connectors in parallel	JST-GH 1x4, J3 on the bottom side and J4 on the top side
Hub J5 DEBUG	SWD, JTAG and console, DS-009 debug full layout	JST-SH 1x10
Hub J6 CAN1	CAN FD bus 1 (TJA1462), two connectors in parallel	JST-GH 1x4, J6 on the bottom side and J7 on the top side

Ref	Function	Connector
Hub J8 B2B POWER + IO	Add-on board connector 1: power rails and FlexComm, I3C, analog and trace pins	Hirose DF40 50-pin header, 0.4 mm pitch, 3.0 mm stack (DF40TC(3.0)-50DP-0.4V)
Hub J9 B2B SDHC + IO	Add-on board connector 2: SDHC or FlexComm 9, SmartDMA or FlexComm 6/7, and more FlexComm pins	Hirose DF40 50-pin header, 0.4 mm pitch, 3.0 mm stack (DF40TC(3.0)-50DP-0.4V)
Hub J13 RTC_BAT	Backup battery for the RTC	JST-SH 1x2, side entry
Hub RGB LED CHAIN	Output of the on-board RGB LED, solder pads	Three pads marked CKO, SDO and G next to U14, 5 V logic

55.3 Power, Ethernet and USB

Hub J1 VIN + T1

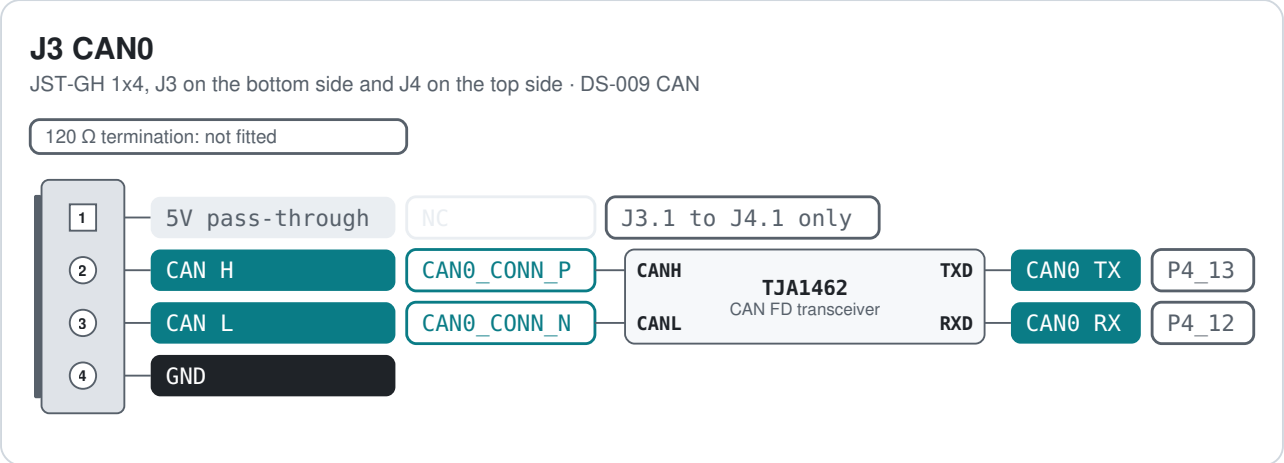


Pin 1 takes the battery, 3S to 12S (about 9 V to 50 V), 10 W budget. An ideal diode (LM74700) blocks reverse polarity, a buck converter makes 5 V from it, and 3.3 V and 1.8 V come from the 5 V rail. The 5 V rail switches on above about 8 V. Pins 2 and 3 are the 100BASE-T1 pair, one twisted pair to the TJA1103 PHY. The PHY is the link slave (R41 fitted); remove R41 to make it the master. The other end of the T1 link must be the master. Mating parts from the schematic: socket 505570-0401, crimp 505572-1100, pre-crimped wires 217500-1125 (black) and 217500-2125 (red).

> Caution: Pin 1 is the only power input besides USB. Keep the battery above 3S, the board does not start below about 8 V.

55.4 CAN

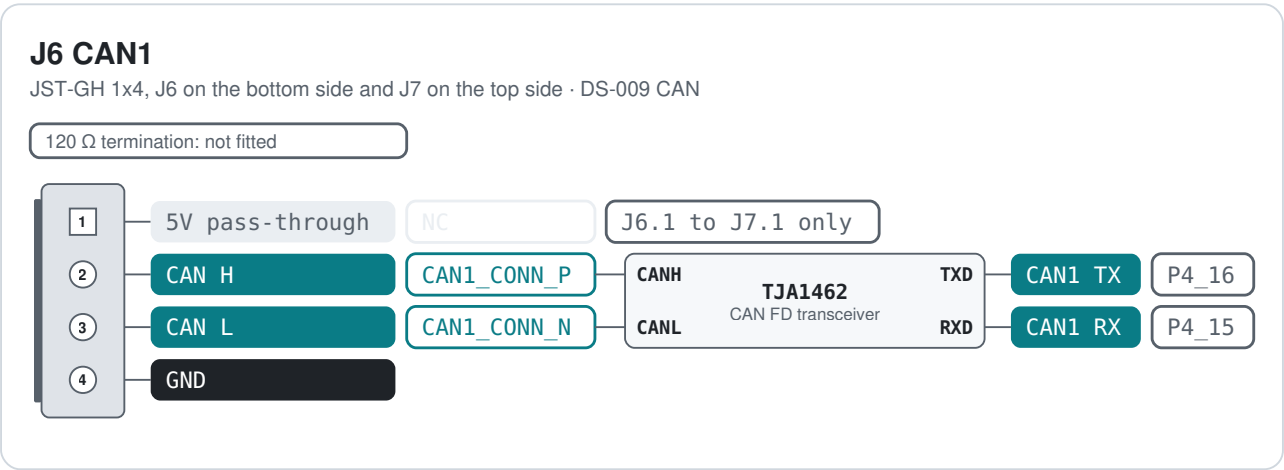
Hub J3 CAN0



J3 (bottom) and J4 (top) carry the same bus, so one cable can come in and another go out. The transceiver runs from 5 V and talks to the chip at 3.3 V. Drive P4_14 (CAN0_STB) low to enable it. The split termination (R27, R28 and C70) is not fitted, so the bus must be terminated elsewhere, or fit the parts if the hub is one end of the bus.

> Note: Pin 1 is not powered. It is only linked between J3 and J4, so a 5 V supply on one cable passes to the other.

Hub J6 CAN1



Same circuit as CAN0. J6 (bottom) and J7 (top) carry the same bus. Drive P4_17 (CAN1_STB) low to enable the transceiver. The split termination (R30, R31 and C75) is not fitted.

> Note: Pin 1 is not powered. It is only linked between J6 and J7, so a 5 V supply on one cable passes to the other.

55.5 Debug

Hub J5 DEBUG

J5 DEBUG

JST-SH 1x10 · DS-009 debug full · 3.3 V logic

1	VREF 3.3V	+3.3V				
2	UART TX	FC1_P1_UART_TXD	FC1_P1	CT0_MAT1	ADC0_B17	P0_25
3	UART RX	FC1_P0_UART_RXD	FC1_P0	CT0_MAT0	ADC0_B16	P0_24
4	SWDIO	DEBUG_TMS_SWDIO	TMS			P0_0
5	SWCLK	DEBUG_TCLK_SWCLK	TCK			P0_1
6	SWO	DEBUG_TDO_SWO	TDO			P0_2
7	TDI	DEBUG_TDI				P0_3
8	GPIO	DEBUG_GPIO	FC0_P0	CT0_MAT2		P0_4
9	nRST	RESET_b				
10	GND					

JTAG only

chip reset, active low

Pin 1 is a 3.3 V reference output for the probe. Pins 2 and 3 are FlexComm 1 as a UART console. Pins 4 to 7 are the SWD and JTAG lines of the MCXN947, pin 8 is a spare GPIO for the probe, pin 9 is the chip reset, pulled up on the board.

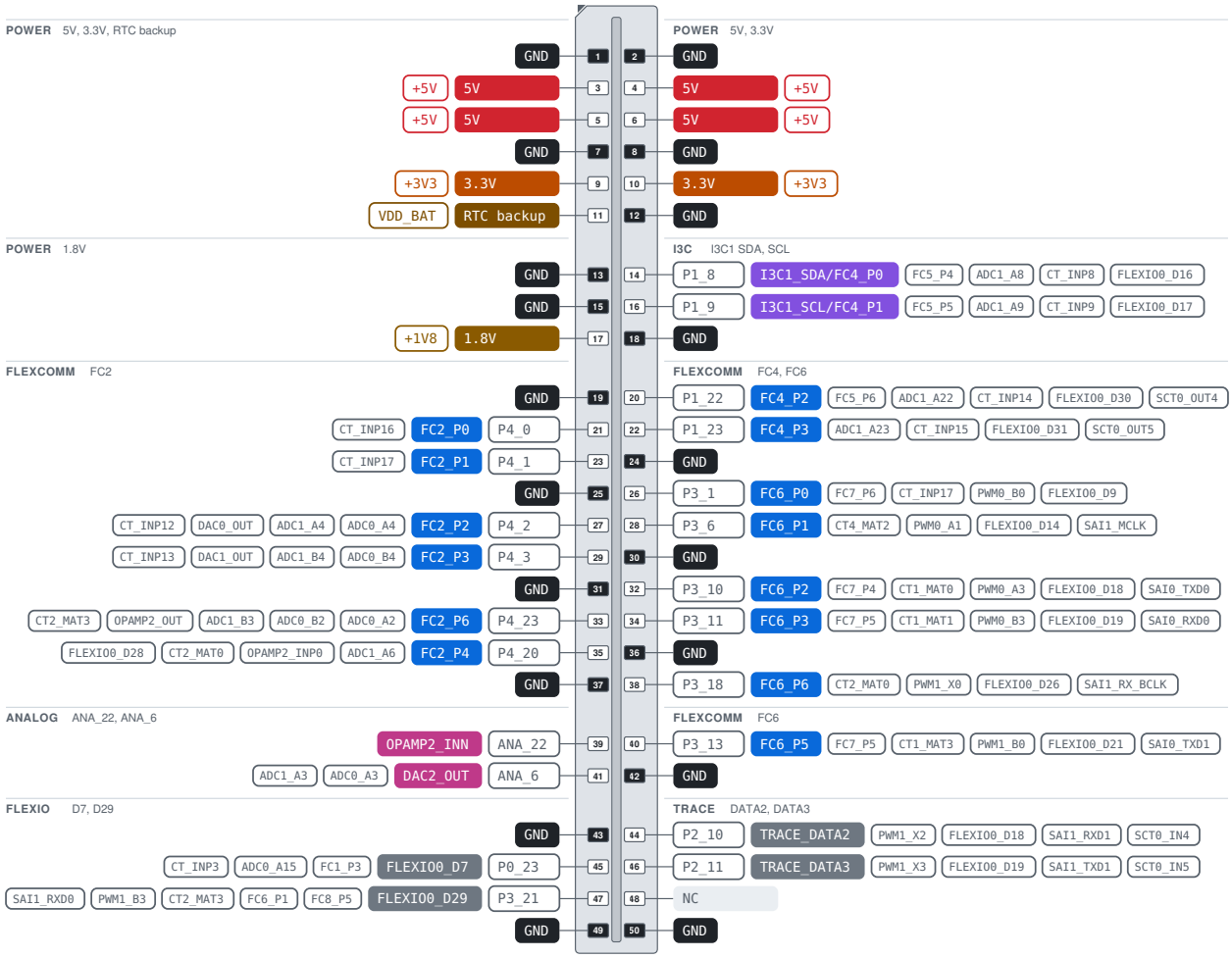
> Warning: Pin 1 is a reference output, not a supply input. Never power the board from it.

55.6 Board-to-board interface

Hub J8 B2B POWER + IO

J8 B2B POWER + IO

Hirose DF40 50-pin header, 0.4 mm pitch, 3.0 mm stack (DF40TC(3.0)-50DP-0.4V) · 3.3 V logic

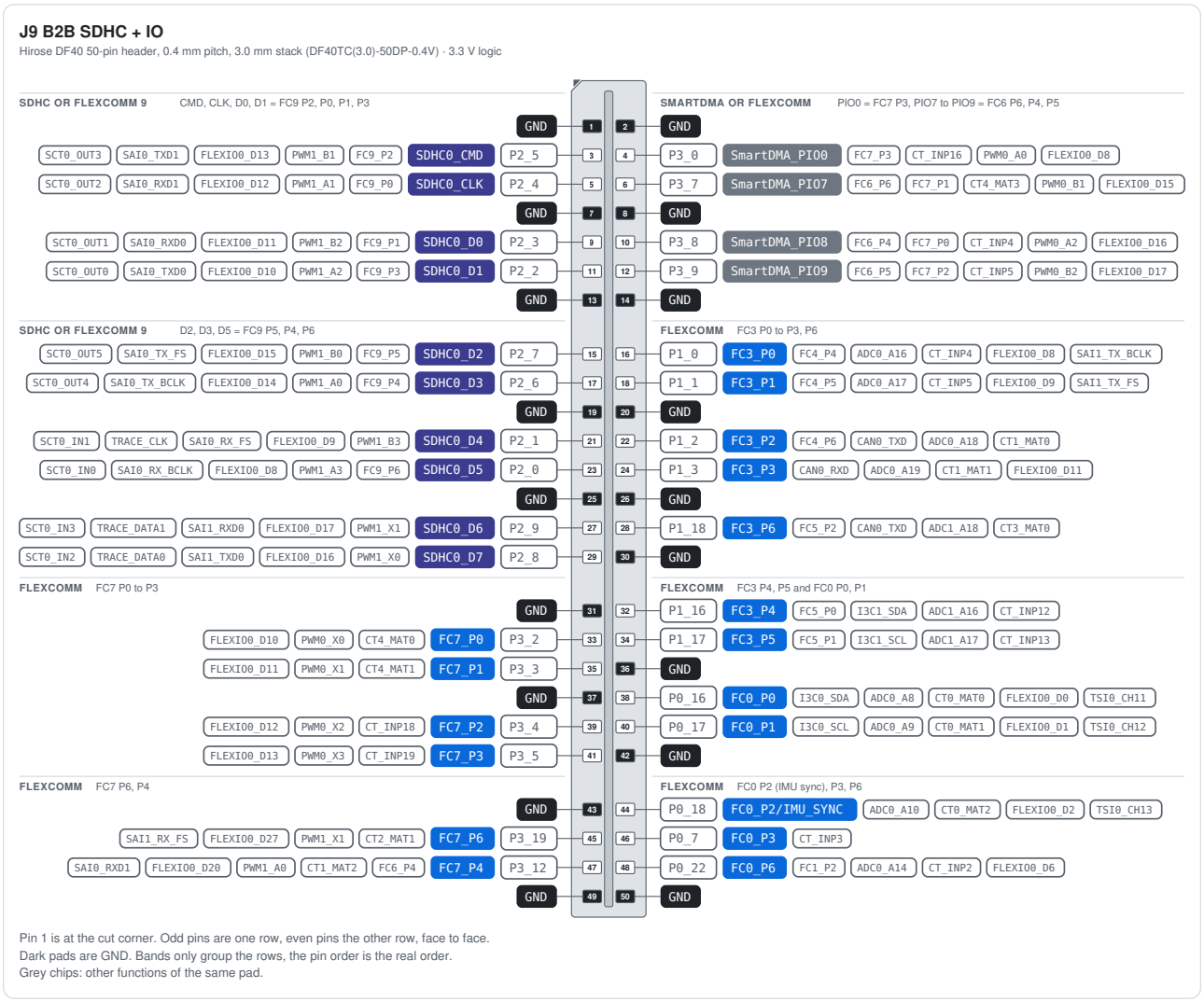


Pin 1 is at the cut corner. Odd pins are one row, even pins the other row, face to face.
Dark pads are GND. Bands only group the rows, the pin order is the real order.
Grey chips: other functions of the same pad.

Together with J9 this is the interface for the add-on boards (camera, audio, SDIO, mikroBUS, display, ADC, IO). Odd pins are one row, even pins the other, pin 1 at the cut corner. Limits from the schematic: 300 mA per pin, 5 V rail 1 A, 3.3 V rail 500 mA, 1.8 V rail 200 mA. Pin 11 is the RTC backup supply, shared with J13. The FlexComm blocks (FC2, FC4, FC6) can each be a UART, SPI or I2C. The grey chips list the other functions of each pad.

> Tip: The add-on board designs are in the spinali_mcxn_t1_shields repository.

Hub J9 B2B SDHC + IO



Second add-on connector. No power pins here, all supplies are on J8. Every signal pin has more than one job, that is the point of this interface. The SDHC0 pins carry an 8-bit SD or eMMC interface, but the same pads are FlexComm 9 (P0 to P6), so an add-on board can use them as one more UART, SPI or I2C instead. The SmartDMA pins are a parallel camera port and double as FlexComm 6 and 7 lines or PWM0 outputs. FC0, FC3 and FC7 can each be a UART, SPI or I2C. Pin 44 is also the IMU sync input. The grey chips list the other functions of each pad.

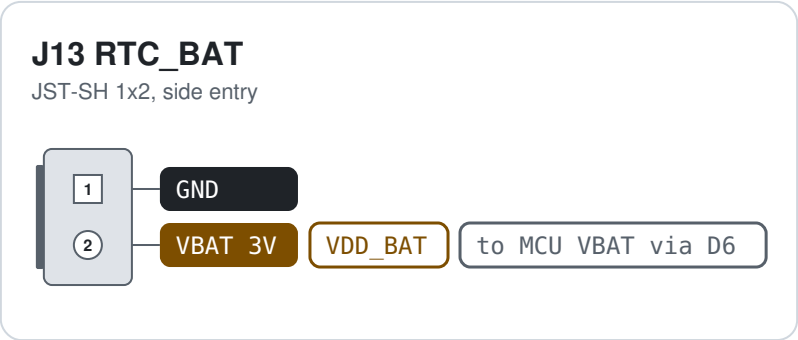
55.7 LED chain and RTC battery

Hub RGB LED CHAIN



The MCU drives one SK9822 RGB LED (U14, APA102 compatible) from FlexIO pins P0_8 (data) and P0_9 (clock). CKO and SDO are the clock and data outputs of that LED, so more LEDs of the same type can be chained from here. The LED runs from 5 V, so its outputs are 5 V.

Hub J13 RTC_BAT



Keeps the real-time clock of the MCXN947 running without main power. Fits the RPi RTC battery (ML2020 with a 2-pin JST-SH plug). The same supply is on J8 pin 11 for the add-on board. The board charges the cell from 3.3 V through D15 and a 75 Ω resistor.

> Warning: The board trickle-charges this cell. Use a rechargeable ML2020 type, never a CR2032 or other non-rechargeable cell.

Hub other connectors

Ref	Function	Connector	Notes
J2 USB	USB 2.0 high-speed device port	USB Type-C receptacle (USB4105-GF-A)	On USB1 of the MCXN947 (high speed). The board is always a USB device (CC pull-downs). VBUS powers the board through load switch U2 when no battery is connected. With SW1 held at power up the boot ROM listens here for ISP programming.

55.8 Buttons and switches

Ref	Board	Function	Type	Notes
SW1 ISP	Hub	ISP boot button	Tactile switch, bottom side	Hold it while power comes up to start the boot ROM in ISP mode (P0_6 ISPMODE_N pulled low), for programming over USB or the debug UART. Normal boot otherwise.

55.9 On-board devices

How the sensors, the PHY and the LED are wired to the MCXN947. Useful for a device tree or board file.

Device	Bus	MCU pins	Notes
ICM-45686 IMU (U9)	SPI on FlexComm 8	SCK P3_15, MOSI P3_14, MISO P3_16, CS P3_17, INT1 P3_22, FSYNC/INT2 P0_18	P0_18 is also on J9 pin 44 as IMU sync.
BMM350 magnetometer (U10)	I3C0 (or I2C)	SDA P0_20, SCL P0_21, INT P0_19	Address 0x14. Runs from 1.8 V.
BMP581 barometer (U11)	I3C0 (or I2C)	SDA P0_20, SCL P0_21, INT P0_11	Address 0x47 (SDO pulled up, R48 not fitted).
TJA1103 100BASE-T1 PHY (U15)	RMII on ENET0	TXD0 P1_6, TXD1 P1_7, TXEN P1_5, TXER P1_10, RXD0 P1_14, RXD1 P1_15, RXDV P1_13, RXER P1_12, REFCLK P1_4, MDC P1_20, MDIO P1_21, INT P1_19, RESET P1_11, WAKE P0_31	Reverse RMII: the PHY supplies the 50 MHz reference clock. PHY address 18. Link slave, because R41 is fitted; remove R41 for master. The two LEDs next to J1 are driven by the PHY GPIO0 and GPIO1.
TJA1462 CAN FD transceivers (U12, U13)	FlexCAN0, FlexCAN1	CAN0 TX P4_13, RX P4_12, STB P4_14; CAN1 TX P4_16, RX P4_15, STB P4_17	Drive STB low to enable a transceiver.
SK9822 RGB LED (U14)	FlexIO	data P0_8, clock P0_9	APA102 compatible, 5 V. Chain output on the CKO, SDO and G pads.
ISP button (SW1)	GPIO	P0_6	ISPMODE_N, boot ROM ISP mode when held at power up.

Test pads on the bottom side: CLK is the PHY reference clock (P1_4), GND is ground.

55.10 Cheat sheet

All connector cards on one A4 landscape sheet: cheat sheet PDF.

55.11 Notes

- The JST-GH and JST-SH connectors follow the Dronecode DS-009 connector standard (CAN and debug full), so standard cables fit. The CAN pin 1 is not powered on this board, and the CAN termination resistors are not fitted.
- The board runs from the battery on J1 or from USB. 5 V, 3.3 V and 1.8 V for an add-on board come from J8.

- The photos show the old revision, marked X-MR-MCXN-T1. The RTC battery connector J13 was added in the current revision, in the bottom right corner of the bottom side.

> Warning: Never feed power into the 5 V, 3.3 V or 1.8 V pins of J8, or into pin 1 of the debug port. Power the board through J1 or USB only.

55.12 References

- MR-MCXN-T1 Hub board design and documentation

56 Configuring the hardware

A typical hardware configuration incorporates the MR-MCXN-T1 Hub (with the chosen shield fitted), a power supply, and a 100BASE-T1 Ethernet host. The Hub is powered and connected over a single connector that carries both raw battery power and the 2-wire 100BASE-T1 link to the host.

- Fit the required shield (MR-MCXN-T1-IO, MR-MCXN-T1-F9P, or MR-MCXN-T1-OFL) onto the Hub.
- Connect the Hub to a 100BASE-T1 host (for example, an MR-NavQ95 running ROS 2) or to a PC through an RDDRONE-T1ADAPT adapter.
- Apply power within the supported input range.

For connector locations and pinouts, see the MR-MCXN-T1 Hub hardware reference.

56.1 Installing and configuring software tools

Software for the MR-MCXN-T1 Hub and supported shields is provided through the NXP Robotics SDK, which uses the Zephyr development environment on the MCU. Follow the SDK instructions to install the toolchain, build the firmware for the chosen shield, and flash the Hub.

For the end-to-end build and flash workflow, see the MR-MCXN-T1 Deploy and Run Quick Start.

57 Abbreviations

Acronym	Description
CAN FD	Controller Area Network with Flexible Data-rate
CDR	Common Data Representation
FBS	FlatBuffers schema
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IDL	Interface Definition Language
IMU	Inertial measurement unit
I/O	Input/output
MCU	Microcontroller
OFL	Optical flow
PHY	Physical layer transceiver
ROS 2	Robot Operating System 2
RTK	Real-Time Kinematic
USB	Universal Serial Bus

58 References

- MR-MCXN-T1 product page — overview, documentation, software, tools, and ordering information
- MR-MCXN-T1 Hub repository
- NXP i.MX and robotics documentation

59 Application Notes

Hardware- and feature-level notes for the MR-MCXN-T1 node: the hardware architecture and reference design of the Hub and the IO, F9P, and OFL add-on boards, and the zenoh transport bridge for publishing sensor samples to a router or PX4.

- Hardware Architecture and Reference Design
- Zenoh_PX4 Communication on the MR-MCXN-T1 using ZROS

Hardware Architecture and Reference Design

The MR-MCXN-T1 is a compact, open hardware reference design for distributed sensing, connectivity, and edge processing in mobile robotics systems. It is based on the cost-effective NXP MCXN947 microcontroller and is intended to act as an intelligent 100BASE-T1 Ethernet sensor and actuator node. This application note describes the system architecture, board functions, electrical interfaces, connectors, power distribution, and hardware integration considerations for hardware engineers integrating the MR-MCXN-T1 into a robotics system.

This document covers the MR-MCXN-T1 Hub and the following add-on boards:

- MR-MCXN-T1 Hub: main hub board for the MR-MCXN-T1 system
- MR-MCXN-T1-IO: input/output shield with Dronecode standard connectors
- MR-MCXN-T1-F9P: GPS/GNSS shield with RTK capability
- MR-MCXN-T1-OFL: optical-flow shield with distance sensor and optical-flow measurement

This document is hardware focused. Software is referenced only where needed to explain the intended use of a hardware function. Detailed software installation, configuration, APIs, middleware, and application development are covered in the MR-MCXN-T1 User Manual and the NXP Robotics SDK documentation.

1. Introduction

1.1 The MR-MCXN-T1 Hub

The MR-MCXN-T1 node board, also referred to as the MR-MCXN-T1 Hub, is based on the cost-effective MCXN947 microcontroller and is intended to act as an intelligent 100BASE-T1 Ethernet sensor and actuator node. Supported example shield designs demonstrate several input/output use cases.

The design demonstrates the convenience of a single-connector sensor node that carries both raw battery power and two-wire 100BASE-T1 Ethernet to a compact edge device. This approach simplifies distributed sensor placement on a robot by reducing wiring, keeping local processing close to the sensor, and sending data to a flight controller, companion computer, or ROS 2 system over Ethernet.

The design intent is to provide a seamless Ethernet-connected node for roboticists using ROS 2. In this sense, the hardware can be considered the physical embodiment of a ROS 2 node.

The key value of the design is the combination of the low-cost MCXN microcontroller, the 100BASE-T1 Ethernet interface, the wide-input-range power stage supporting typical robotics battery voltages up to 60 V, and the off-board connection to a sensor or actuator through an add-on board.

A typical product implementation would likely flatten the Hub and add-on board into the smallest practical single-board solution. The additional interfaces provided on the Hub are intended primarily for evaluation and experimentation, as they can be included with minimal additional design overhead. A standalone Ethernet-to-CAN gateway is the only identified production-oriented exception.

1.2 System design objectives

- Provide an example of offloading hard real-time ROS 2 nodes to a low-cost MCU implementation for distributed robotic sensing and actuation.
- Combine direct battery power and two-wire 100BASE-T1 Ethernet in a streamlined node connection.

- Provide an efficient, ROS 2 developer-friendly Zephyr, ZROS, and Zenoh framework for hard real-time ROS 2 nodes, enabling developers to focus on sensor or actuator drivers while reusing the underlying RTOS and transport mechanisms.
- Act as the physical embodiment of a ROS 2 node.

2. System hardware architecture

2.1 System overview

The MR-MCXN-T1 family consists of the Hub board and application-specific add-on boards. The Hub provides processing, Ethernet connectivity, power handling, and the common expansion interface. Each add-on board places a specific sensor or input/output function close to the MCXN947-based node.

2.2 Functional partitioning

- Hub board: provides the core processing, 100BASE-T1 connectivity, power distribution, debugging, and shield interface. Several additional interfaces were included for evaluation convenience because they could be implemented with minimal design overhead; however, they are generally superfluous to the intended node function and would normally be omitted from an optimized production design.
- IO board: an example actuator and general-purpose I/O shield that transforms the T1 node into an I/O interface using common robotics connectors; a production implementation would typically integrate the shield and Hub into an optimized single-board node.
- F9P board: an example GNSS sensor shield with RTK capability that transforms the T1 node into a precision-positioning node; a production implementation would typically integrate the shield and Hub into an optimized single-board node.
- OFL board: an example optical-flow and distance-sensor shield that transforms the T1 node into a local motion-sensing node; a production implementation would typically integrate the shield and Hub into an optimized single-board node.

2.3 Single-connector node concept

The node architecture reduces robot wiring by carrying raw battery power and two-wire 100BASE-T1 Ethernet to the local node on a single 4-pin connector. Power over Data Line (PoDL) was evaluated but rejected because power negotiation would add unnecessary complexity. In robotics systems, power distribution is typically managed by the system architect; a direct battery input therefore provides greater flexibility, higher current capability, and simpler circuitry. The resulting Power Adjacent to Ethernet (PAE) connection carries power and Ethernet through a rugged, latching four-pin connector. The Hub provides the local processing and interfaces required by the attached add-on board.

2.4 Hardware boundaries

Electrical limits, connector pin assignments, signal direction, power-domain relationships, and assembly options must be confirmed against the released schematic and bill of materials for the applicable board revision. This document provides a system-level hardware overview and should be used with the associated design files. The reference design demonstrates an implementation of the software framework, and its schematics may serve as a starting point; however, the hardware is expected to be adapted to the specific electrical requirements of the target application.

3. MR-MCXN-T1 Hub hardware

3.1 Hardware overview

The MR-MCXN-T1 Hub is the central board in the product family. Its MCXN947 microcontroller performs local processing and manages the interfaces used by the Hub and attached add-on board. The Hub is intended to operate as a network-connected sensor or actuator node rather than as a standalone robot controller.

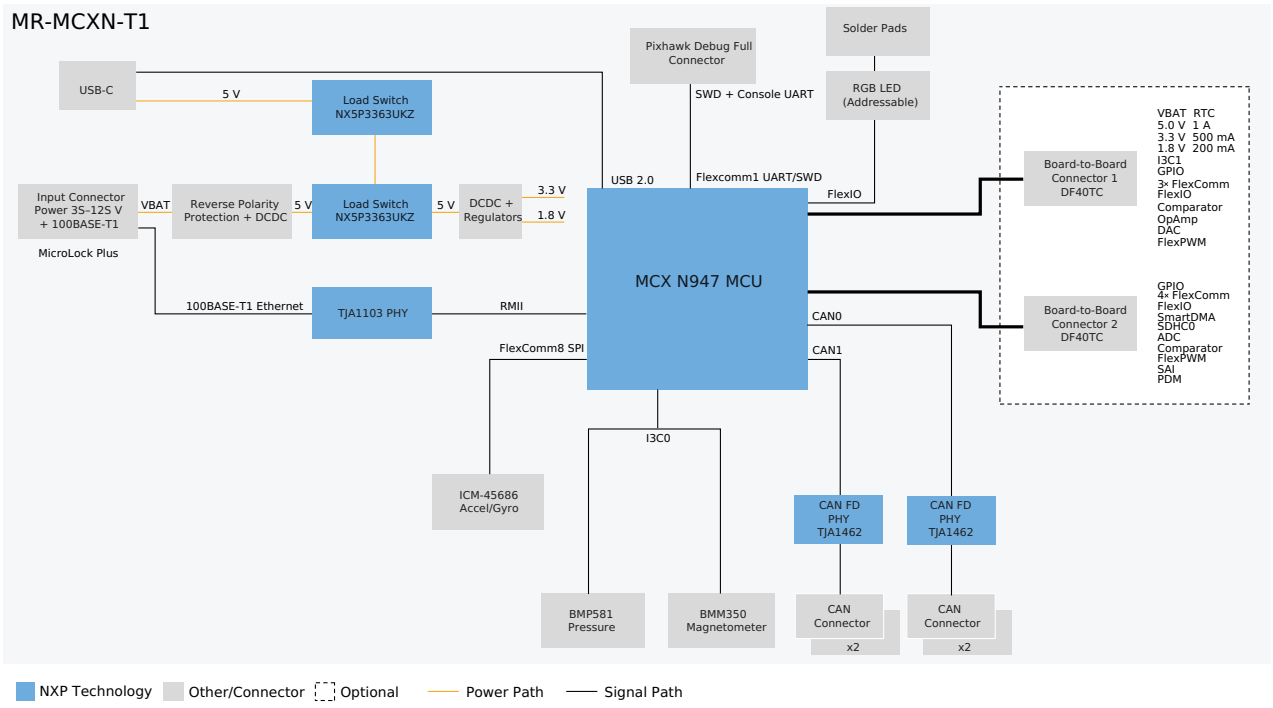


Figure 1. MR-MCXN-T1 Hub block diagram

3.2 Processing subsystem

The processing subsystem is based on the NXP MCXN947 microcontroller. The Hub also includes an ICM-45686 accelerometer and gyroscope, a BMP581 pressure sensor, and a BMM350 magnetometer for evaluation and development. The selected MCU interfaces and pin assignments depend on the applicable board revision and Zephyr hardware configuration and should be confirmed against the released schematic and overlay files.

3.3 100BASE-T1 Ethernet interface

The Hub uses an NXP TJA1103 100BASE-T1 Ethernet PHY connected to the MCXN947 through RMII. The two-wire Ethernet pair is routed through the PAE connector together with the separate battery-power conductors. The physical interface, connector implementation, termination, and protection components form part of the hardware design and should be retained when the design is adapted.

The 100BASE-T1 PHY schematic follows standard design practice. More detail can be found in the datasheet and application notes specific to the TJA1103. A common passive network configuration is used to provide filtering and ESD protection.

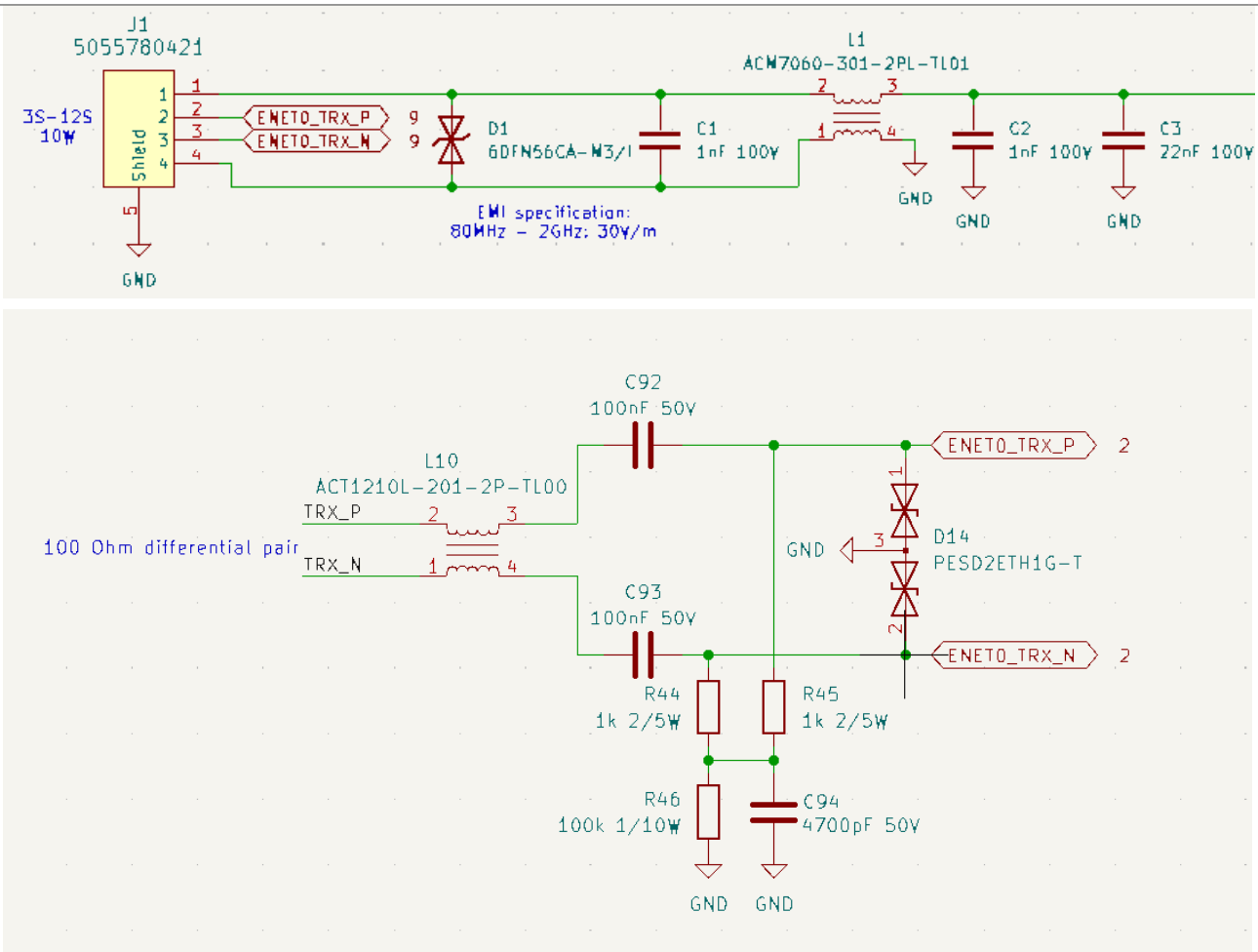


Figure 2. 100BASE-T1 simple passive filtering and ESD interface

The 100BASE-T1 Ethernet TJA1103 PHY drives two configurable status LEDs: D13 (green), typically used to indicate an established 100BASE-T1 link, and D12 (amber), typically used to indicate transmit and receive activity. The precise behavior is determined by the TJA1103 LED-output register configuration in software.

Reference	Color	Indication
D13	Green	Ethernet link status — illuminated when the 100BASE-T1 link is established.
D12	Amber	Ethernet activity — flashes when Ethernet frames are transmitted or received.

3.4 Power input and distribution

The Hub accepts a 3S-to-12S battery input, up to 60 V, through the PAE node connection and includes reverse-polarity protection. Onboard conversion provides the local rails required by the Hub and attached shield. The verified shield supplies are:

- 5 V at up to 1 A
- 3.3 V at up to 500 mA
- 1.8 V at up to 200 mA

The following table describes the power-supply status indicators.

Reference	Schematic name	Color	Function
D2	USB VBUS	Blue	Indicates that the protected 5 V VBUS supply is present. The LED turns off if the associated load switch reports a fault condition.
D3	5V-IND	Blue	Indicates that the 5 V supply rail regulated from the power connector battery input is present. The LED turns off if the associated load switch reports a fault condition.
D4	3V3-IND	Blue	Indicates that the 3.3 V supply rail is present.
D5	1V8-IND	Blue	Indicates that the 1.8 V supply rail is present.

Input limits, protection behavior, regulator ratings, available shield current, and sequencing requirements must be confirmed against the released schematic for the board revision in use.

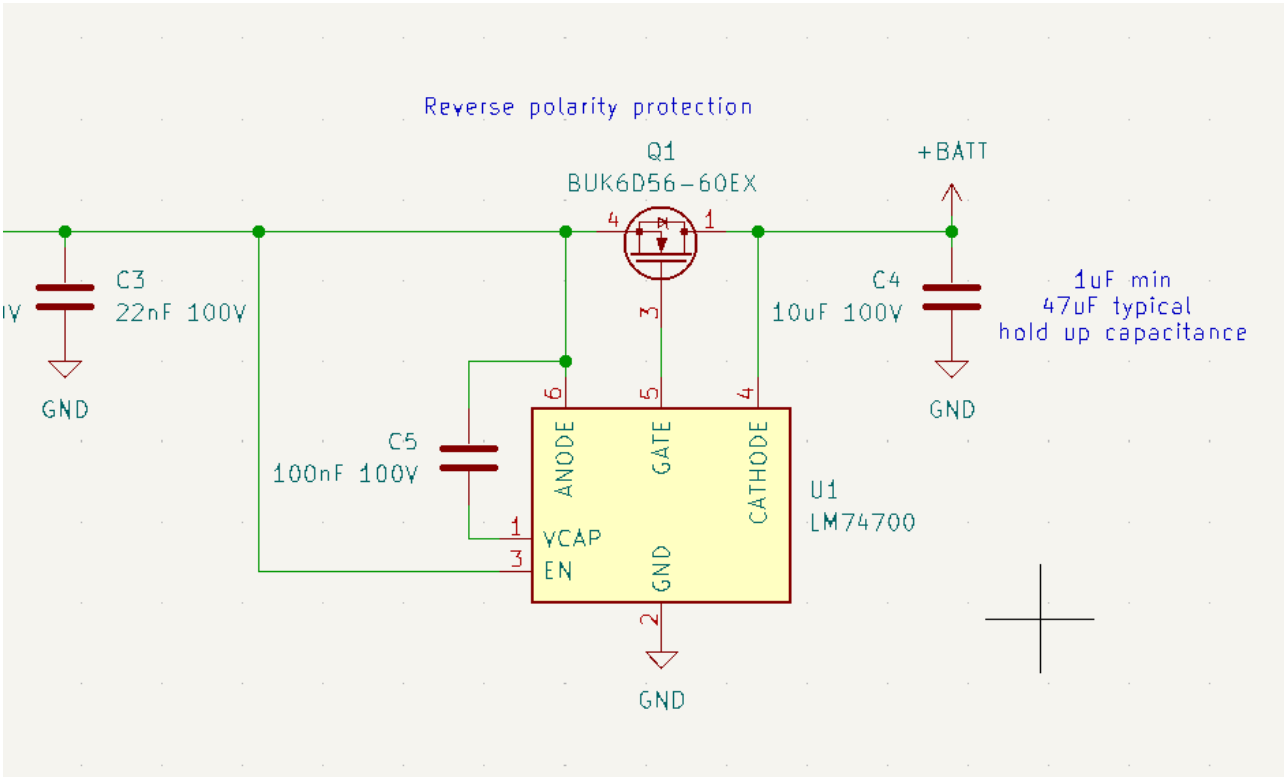


Figure 3. Primary wide-input-voltage regulator

3.5 Add-on shield board-to-board interface

A key focus of this design is to demonstrate the base 100BASE-T1 plus power functionality and then make provisions for easily prototyped additional small shield boards. This shield interface is provided through two DF40TC board-to-board connectors that supply shield power and configurable MCXN947 signals.

The interface can expose FlexComm, I3C, GPIO, SDIO/SDHC, FlexIO, SmartDMA, ADC, comparator, operational-amplifier, DAC, FlexPWM, SAI, and PDM functions. Peripheral pin multiplexing and signal mapping are not fixed solely by the hardware; they depend on the active Zephyr configuration and are defined through Zephyr shield definitions and hardware overlays.

Before attaching an add-on board, verify the applicable shield and overlay configuration together with connector orientation, pin assignment, voltage domain, current demand, and any mutually exclusive functions.

> Note: Do not interchange shields between Hubs without first confirming the active pin mapping, as configured assignments may not match the connected board.

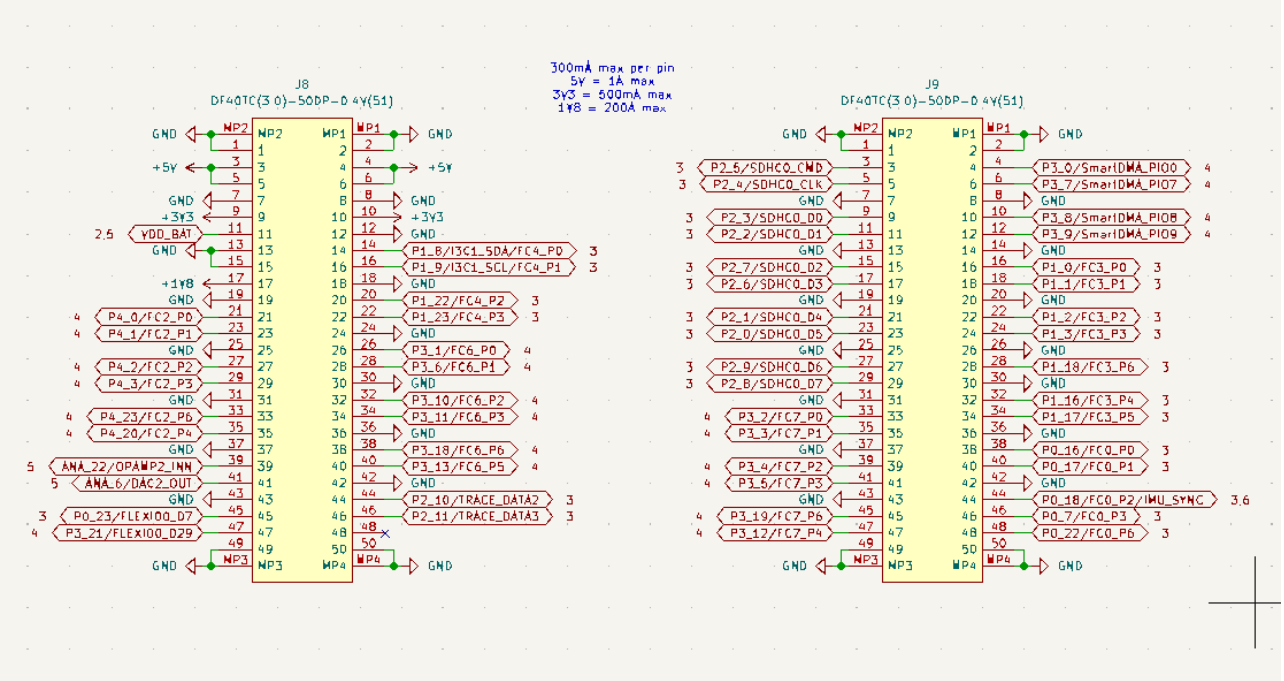


Figure 4. DF40 board-to-board interfaces

3.6 Programming and debug

The Hub provides a programming and debug connector for the MCXN947 that follows the Dronecode/Pixhawk standard Debug Full interface. Use a compatible debug probe and the cable or adapter specified for this connector and pinout.

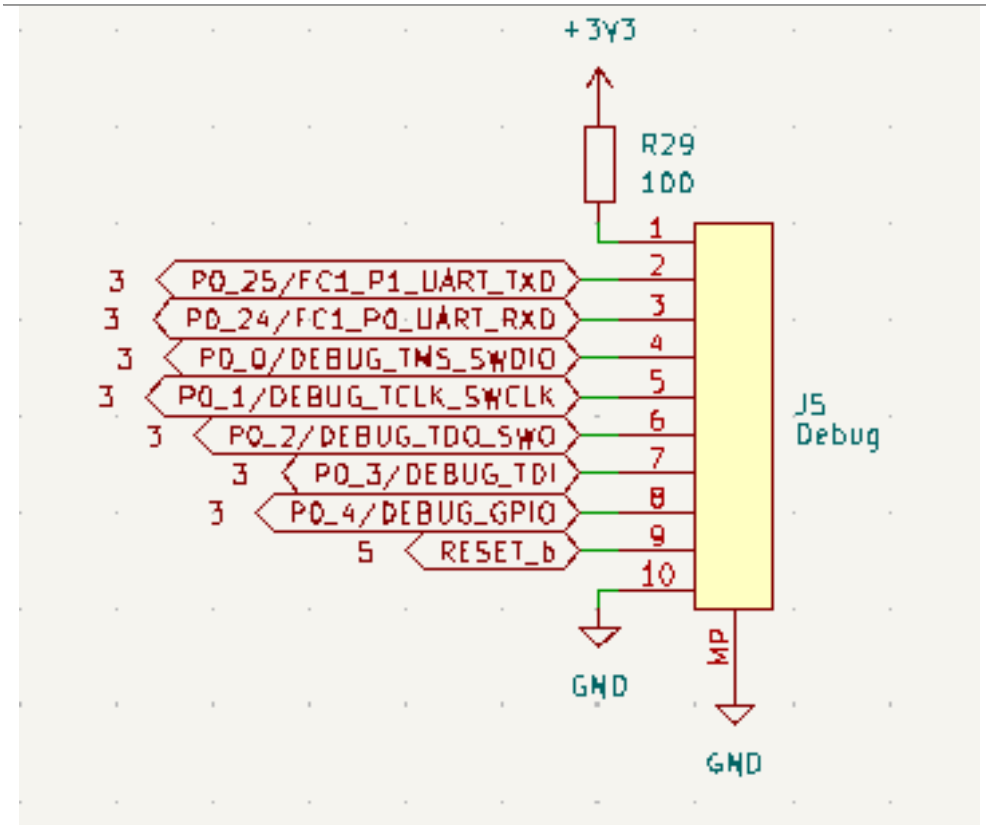


Figure 5. Standard Dronecode/Pixhawk Debug Full interface

3.7 Additional interfaces, indicators, test points, and configuration options

The T1 Hub centers on the combined 100BASE-T1 Ethernet and power interface, the MCU, the board-to-board peripheral connections, and the Zephyr-based software framework that enables operation as a ROS 2 node. The processor also provides several auxiliary interfaces that, although not central to the design, are exposed for evaluation and experimentation.

Addressable RGB status LED

U14 is an SK9822, APA102-compatible, individually addressable RGB LED that may be used to provide software-controlled visual status indication. It uses a synchronous, SPI-like serial interface with separate data and clock signals, allowing the MCU to set the LED color and brightness. Solder pads expose the LED power, ground, data, and clock connections so that additional SK9822-compatible LEDs can be daisy-chained into an external string as required.

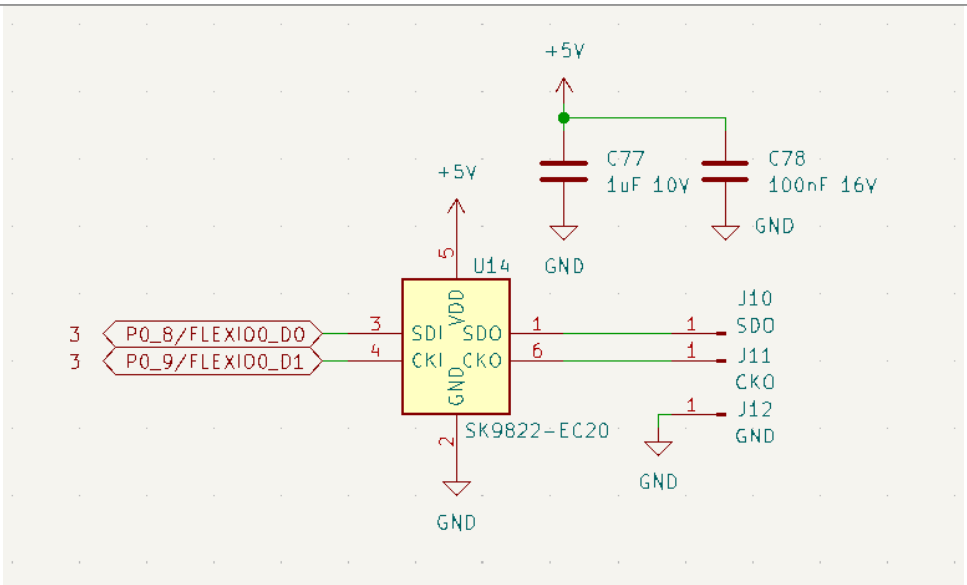


Figure 6. SK9822 APA102-compatible addressable RGB LED

USB 2.0 through a USB-C connector

This interface is available on the MCXN947 and is configured normally per the datasheet guidelines with ESD protection. Zephyr software typically configures this as a console UART.

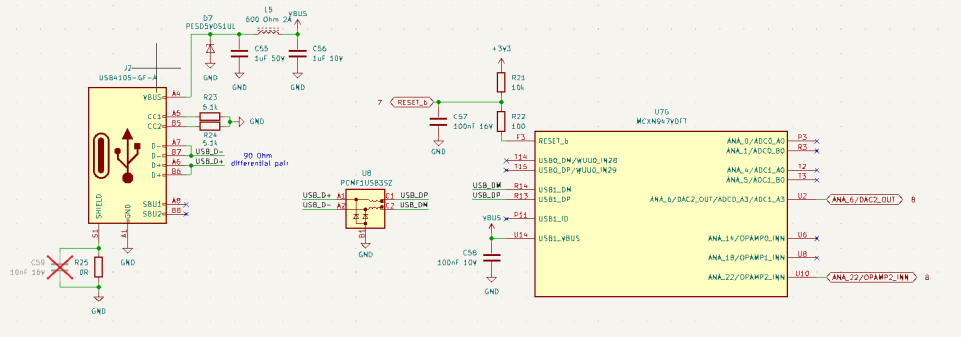


Figure 7. USB 2.0 using a USB-C connector

CAN-FD / CAN-SIC interfaces

Two independent CAN FD interfaces (CAN0, CAN1) use TJA1462 transceivers with CAN Signal Improvement Capability (CAN SIC). Each interface includes the recommended ESD protection and filtering network. Each bus provides two parallel connectors for pass-through connections (CAN0A, CAN0B and CAN1A, CAN1B). Bus-termination resistor footprints are provided but intentionally left unpopulated. A practical evaluation use case is the Hub as an Ethernet-to-CAN gateway.

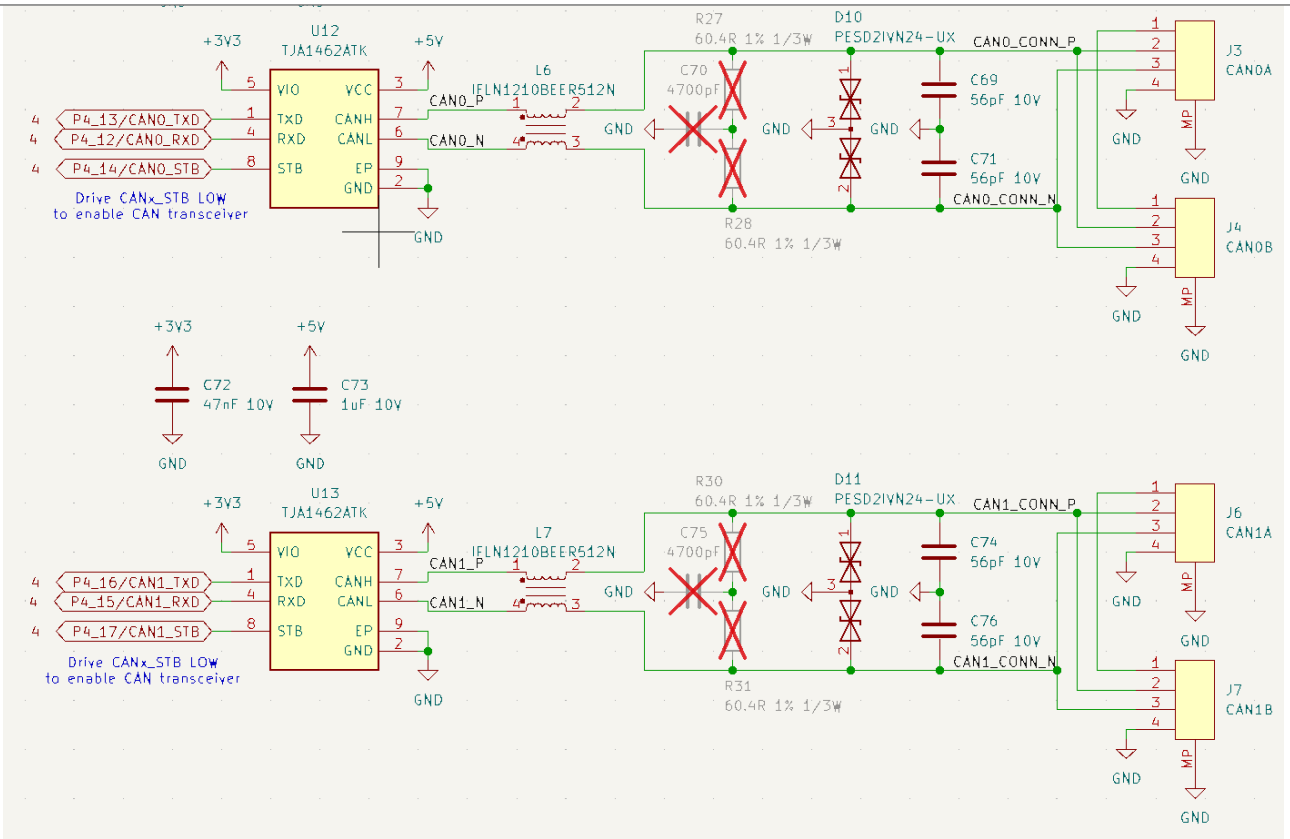


Figure 8. CAN0 and CAN1 buses with dual connectors

Onboard sensors

The Hub includes three onboard motion and environmental sensors for experimentation.

The BMM350 and BMP581 are currently disabled in software so that their shared pins can provide a dedicated UART console that captures Zephyr logs from the start of boot. To make these pins available to the sensors, the Zephyr console can instead be routed through USB CDC. I3C is the preferred interface for the BMM350 and BMP581 and is expected to be enabled once the upstream Zephyr drivers are stable. The ICM-45686 uses a separate SPI interface and is not affected by this pin-multiplexing constraint.

- ICM-45686: a six-axis inertial measurement unit combining a three-axis accelerometer and three-axis gyroscope for measuring linear acceleration and angular motion.
- BMM350: a three-axis magnetometer for measuring magnetic-field direction and supporting heading estimation.
- BMP581: a barometric pressure sensor for measuring atmospheric pressure and supporting altitude estimation.

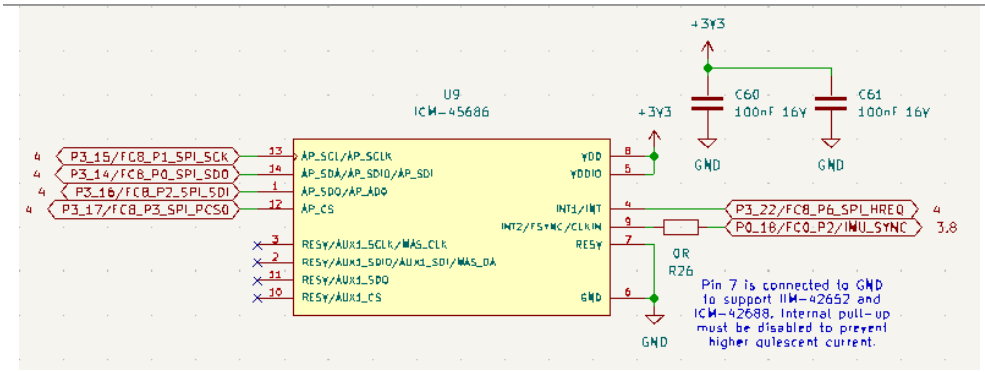


Figure 9. ICM-45686 IMU on SPI

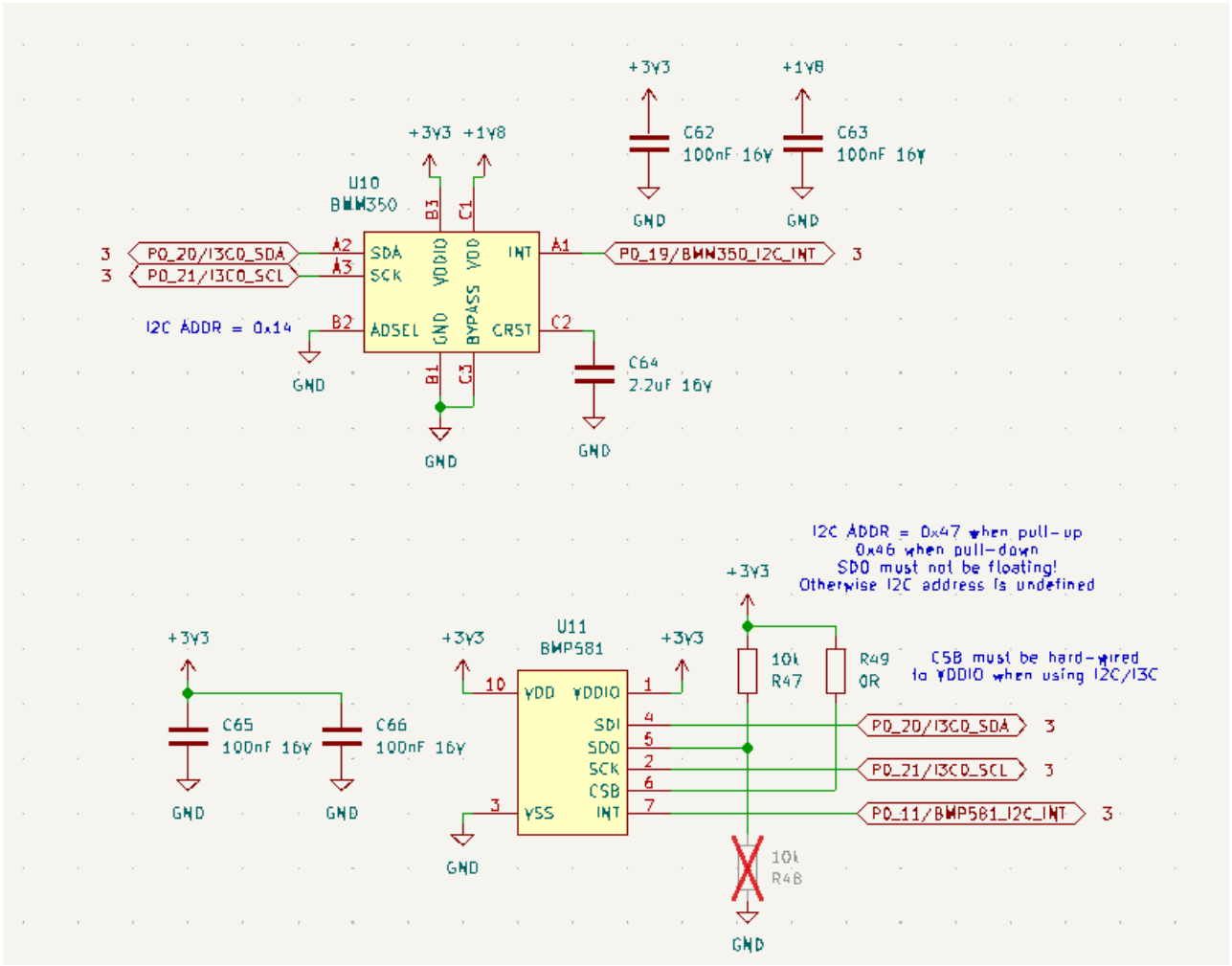


Figure 10. BMM350 and BMP581 I3C sensors

Schematic and design-file verification

The available schematic-derived information does not identify a complete, verified set of test-point reference designators; therefore, test points, straps, and assembly options must be confirmed directly from the released schematic and board drawing for the applicable revision.

4. MR-MCXN-T1-IO hardware

4.1 Overview

The MR-MCXN-T1-IO is an input/output shield for the MR-MCXN-T1 Hub. It provides access to node signals through Dronecode standard connectors for robotics integration.

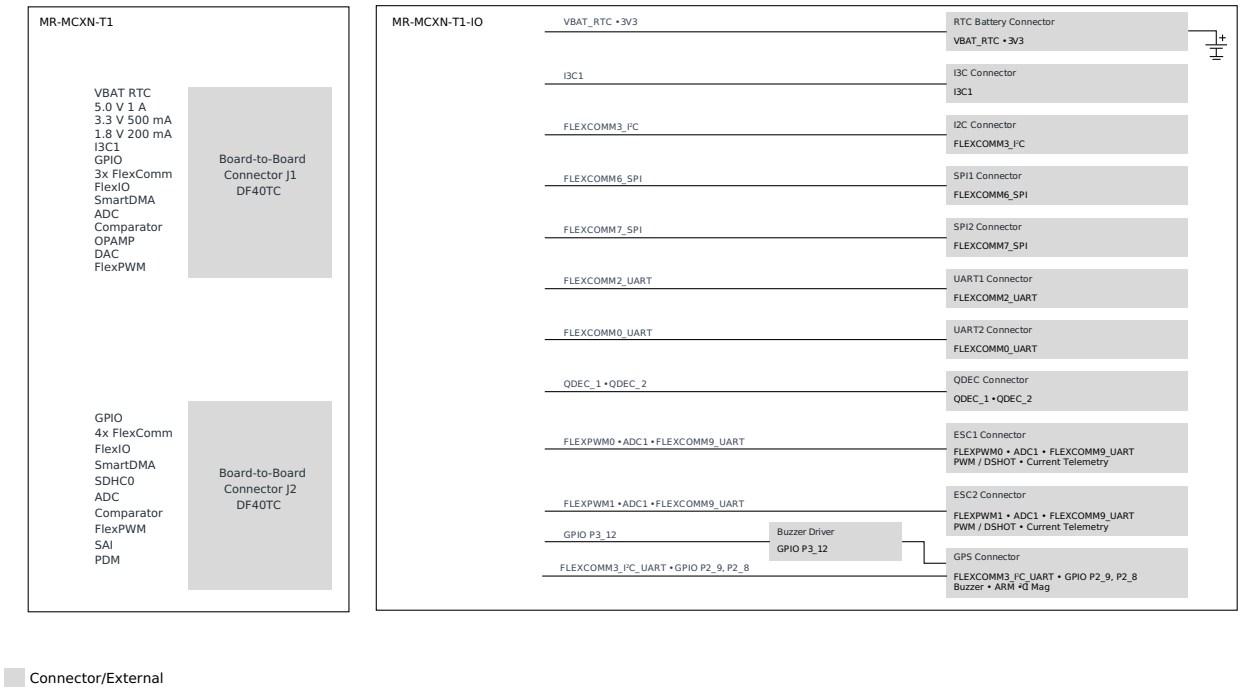


Figure 11. MR-MCXN-T1-IO shield block diagram

4.2 Connector interfaces

The connector set, pin assignments, voltage levels, and signal directions must be verified against the MR-MCXN-T1-IO schematic. External equipment must not be connected until power, ground, and logic compatibility have been confirmed.

4.3 Input/output functions

The IO shield demonstrates typical sensor and actuator connectivity through Dronecode-style interfaces. Verified functions include I3C, I²C, two SPI interfaces, two UART interfaces, a quadrature-decoder input, two ESC interfaces supporting PWM or DShot control with current telemetry, an RTC battery connection, a buzzer, and a Pixhawk-style GPS connection. Exact signal mapping, power rails, and mutually exclusive functions are defined by the schematic and the active Zephyr shield overlay.

4.4 Power and protection considerations

Confirm the current available on each connector, the source of each power rail, and any onboard protection before powering attached devices. The combined load of all connected peripherals must remain within the Hub and shield limits.

4.5 Integration checklist

- Confirm the Hub and IO board revisions.
- Verify connector orientation and pinout.

- Verify the peripheral voltage and current requirements.
- Check whether any signals are shared or mutually exclusive.
- Connect or disconnect peripherals only under conditions allowed by the hardware design.

5. MR-MCXN-T1-F9P hardware

5.1 Overview

The MR-MCXN-T1-F9P is a GPS/GNSS add-on board with RTK capability. It allows the MR-MCXN-T1 node to acquire positioning data locally and deliver it to the robotic system through the Hub Ethernet connection.

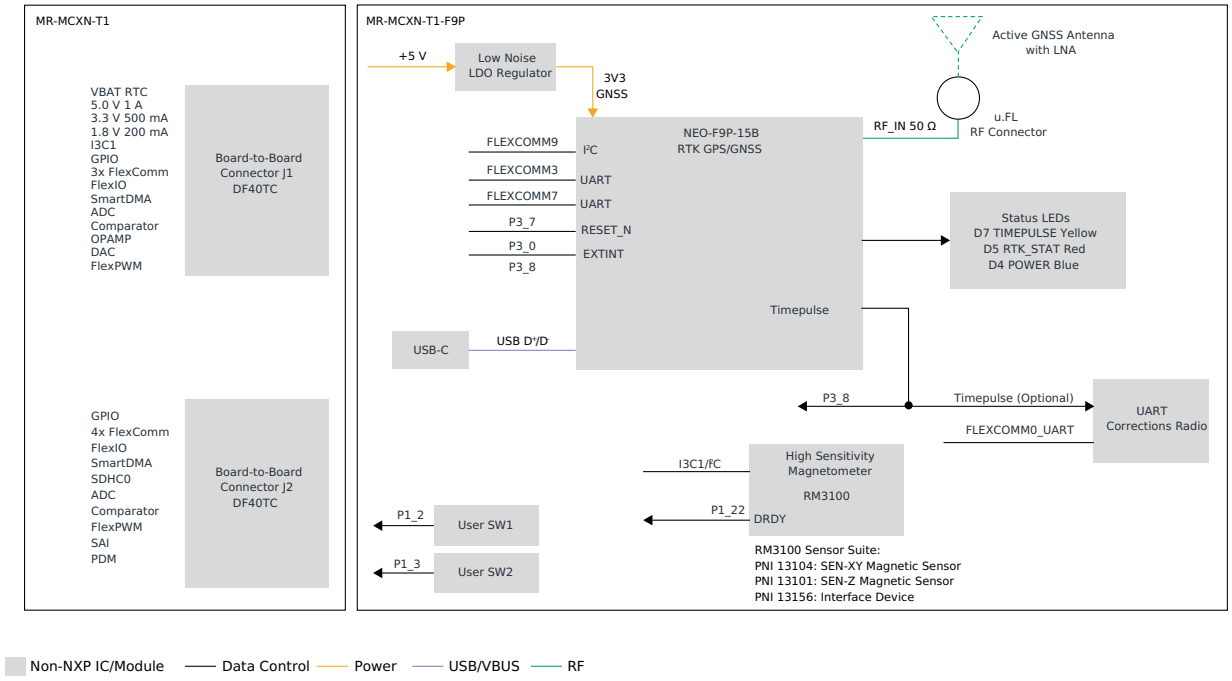


Figure 12. MR-MCXN-T1-F9P shield block diagram

5.2 GNSS receiver subsystem

The shield uses a u-blox NEO-F9P-15B GNSS receiver. It provides two UART interfaces as well as a direct USB-C interface to the receiver. Supporting hardware includes a UART connection for an external RTK correction radio and control or status signals for TIMEPULSE, RESET, EXTINT, and SAFEBOOT. A high-sensitivity RM3100 magnetometer is also included on the shield.

A dedicated low-noise LDO powers the NEO-F9P as a precaution, although it may be unnecessary in a production design.

5.3 Antenna interface

The GNSS antenna connects through a u.FL interface intended for a compatible active antenna. The preferred tested antenna is the TE Connectivity (Tyco) ANT-GNCP-C25L125100 active GNSS antenna. Other compatible antennas may be used; however, electrical compatibility and all required performance and system-level validation must be verified for the chosen antenna.

5.4 RTK correction-data path

The NEO-F9P supports correction-data delivery through UART, USB, SPI, and I²C interfaces. The MR-MCXN-T1-F9P shield routes both receiver UARTs and I²C to the Hub, provides USB-C access for direct configuration, and includes a dedicated UART connector for an external correction radio. These hardware paths allow correction data to be supplied through several possible system architectures.

The current software defaults to a multiplexed UART configuration that carries GNSS data, receiver configuration, and RTK corrections over a single connection. See the separate software documentation for supported configurations and operating modes.

5.5 Timing and communication interfaces

The following table lists the verified NEO-F9P interface connections. These connections reflect the schematic routing; the active Zephyr shield overlay determines the peripheral multiplexing and software configuration.

Function	Verified connection
NEO-F9P I²C	FLEXCOMM9
NEO-F9P UART2	FLEXCOMM3 UART
NEO-F9P UART1	FLEXCOMM7 UART
External correction-radio connector	J3, using FLEXCOMM0 UART TX/RX
Optional correction-radio timing signal	TIMEPULSE on J3
Receiver reset	RESET_N — P3_7
External interrupt	EXTINT — P3_0
Receiver timing output	TIMEPULSE — P3_8
Safe-boot control	SAFEBOOT_N — SW1 / P1_2
Additional user switch	SW2 — P1_3
Magnetometer interface	I3C1 / I²C, with DRDY
Power indicator	D4 — POWER

Function	Verified connection
RTK-status indicator	D5 — RTK_STAT
Timing indicator	D7 — TIMEPULSE

The shield status indicators are:

Reference	Name	Color	Function
D4	POWER	Blue	Indicates that board power is present.
D5	RTK_STAT	Red	Indicates the NEO-F9P RTK status output.
D6	GPS-FIX	Yellow	MCU-controlled indicator that illuminates when the software obtains a GPS fix.
D7	TIMEPULSE	Yellow	Indicates activity on the NEO-F9P TIMEPULSE output.

5.6 Installation considerations

- Mount the antenna with a suitable view of the sky.
- Keep the antenna and cable away from strong interference sources.
- Verify the antenna connection before applying power.
- Confirm that the antenna is electrically compatible with the F9P shield, including its supply-voltage and current requirements.

6. MR-MCXN-T1-OFL hardware

6.1 Overview

The MR-MCXN-T1-OFL is an optical-flow add-on board that combines optical-flow measurement with distance sensing. The board is intended to provide local motion-related measurements to the MR-MCXN-T1 Hub.

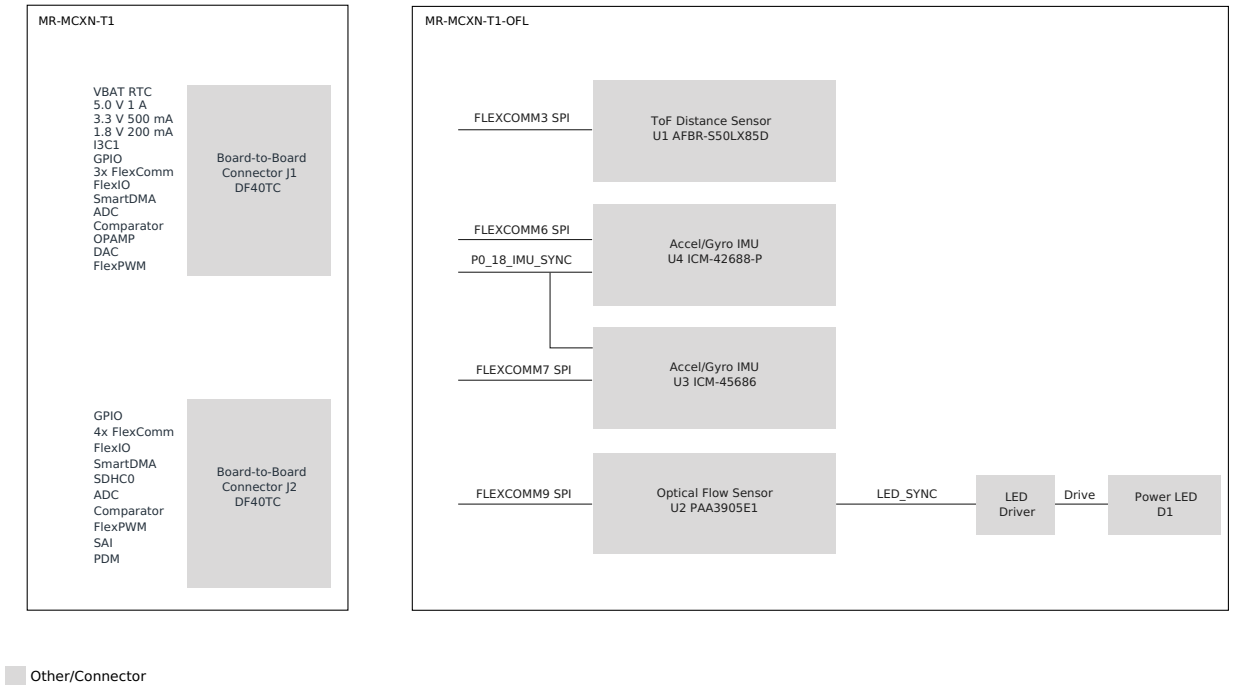


Figure 13. MR-MCXN-T1-OFL shield block diagram

6.2 Optical-flow subsystem

The optical-flow subsystem uses a PAA3905E1 sensor with a dedicated illumination LED driver and high-power LED. It communicates with the Hub through a configured FlexComm/SPI path. Optical performance depends on target texture, lighting, working distance, mounting, and motion conditions.

> Note: Alternative illumination sources may be used, including infrared LEDs such as the ams OSRAM SFH 4775S. A 940 nm LED can reduce optical interference with the AFBR distance sensor.

6.3 Distance-sensor subsystem

Distance measurement is provided by an AFBR-S50LX85D time-of-flight sensor. Its field of view, operating limits, optical clearance, interface, and power requirements must be considered when integrating the board.

6.4 Inertial measurement units

The board includes two six-axis inertial measurement units for motion sensing and evaluation:

- ICM-42688-P: a three-axis accelerometer and three-axis gyroscope that provides angular-rate and linear-acceleration measurements.
- ICM-45686: a newer three-axis accelerometer and three-axis gyroscope available as an alternative IMU for evaluation and software development.

Both devices are available to the MCXN947 through configured FlexComm/SPI paths. The active Zephyr shield overlay determines the selected interface, chip-select routing, interrupt signals, and software enablement.

6.5 Mechanical placement

Mount the board so that the optical path is unobstructed and the sensing direction matches the intended application. Avoid covers, fasteners, wiring, or structural parts within the sensor fields of view.

6.6 Environmental considerations

Surface reflectivity, texture, ambient light, contamination, vibration, and mounting height can affect measurements. Validate the installed configuration under the expected operating conditions.

6.7 Electrical integration

The shield includes ICM-42688-P and ICM-45686 inertial sensors, provisions for IMU synchronization, and configured FlexComm/SPI communication paths for the optical-flow, distance, and inertial devices. Verify shield orientation, power rails, chip-select routing, synchronization signals, reset or interrupt lines, and the active Zephyr overlay against the released schematic before operation.

7. Software context

This section provides only the software context needed to understand the intended hardware role. Software installation, build procedures, configuration, APIs, middleware, examples, and troubleshooting are outside the scope of this hardware application note; see the MR-MCXN-T1 User Manual.

The software enablement framework uses Zephyr on the MCU and separates local driver work from ROS-oriented data exchange. It provides a path for sensor and actuator data to move between the MR-MCXN-T1 node and a ROS 2-based host system over 100BASE-T1 Ethernet.

ZROS is an open-source library that provides a ROS-oriented topic layer on the MCU. It supports preparing, formatting, inspecting, and exchanging sensor or actuator data in a manner similar to ROS 2 topics on a Linux host and includes an efficient zero-copy buffer mechanism.

ZROS uses FlatBuffers internally for efficient MCU-side data handling. For ROS 2 interoperability, data can be statically packed into the CDR representation generated from ROS 2 IDL definitions. Zenoh-pico provides the transport and routing layer used to move data between the MCU node and the rest of the robotics system.

The NXP Robotics SDK provides a curated and validated set of software components, tools, and documentation tested together on this hardware platform. NXP contributes improvements and fixes to upstream projects on a scheduled basis, so the latest upstream code may differ from the version included in an SDK release.

8. Hardware installation and bring-up

8.1 Before applying power

- Confirm the Hub and add-on board part numbers and revisions.
- Inspect both boards for damage, contamination, or misplaced components.
- Verify connector orientation and mating.
- Confirm the input supply range and polarity from the applicable schematic.
- Confirm the expected current demand of the Hub and attached peripherals.
- Connect antennas or optical assemblies required by the selected add-on board.

8.2 Initial power-on checks

Use a current-limited supply where appropriate. Check for abnormal current, unexpected heating, and the expected local power rails before connecting the node to the wider robotic system.

8.3 Ethernet connection

Connect the 100BASE-T1 interface using the intended cable and mating hardware. Verify the physical link before proceeding to software-level communication checks. Although 100BASE-T1 connections are polarity-sensitive, many PHYs can correct reversed polarity automatically or under software control.

8.4 Peripheral verification

Verify the attached add-on board one functional block at a time. Hardware verification should establish correct power, connectivity, signal levels, and basic device presence before application-level testing.

8.5 Handling and integration

Use appropriate electrostatic-discharge precautions. Do not hot-plug interfaces unless the applicable board documentation explicitly permits it. Provide mechanical support and strain relief appropriate to the installation.

9. Conclusion

The MR-MCXN-T1 family provides a compact hardware approach for distributed sensing, connectivity, and edge processing in mobile robotics, and specifically acts as an efficient physical embodiment of a ROS 2 node. The MCXN947-based Hub combines local processing with two-wire 100BASE-T1 Ethernet, while the IO, F9P, and OFL add-on boards demonstrate common robotics input/output, positioning, and motion-sensing functions.

Hardware integration should always be checked against the released schematic, bill of materials, and board-revision documentation. Detailed software enablement is intentionally covered in a separate document.

10. Abbreviations

Acronym	Description
ADC	Analog-to-digital converter
CAN	Controller Area Network
CAN FD	Controller Area Network Flexible Data-rate
CAN SIC	Controller Area Network Signal Improvement Capability
CDR	Common Data Representation
DAC	Digital-to-analog converter

Acronym	Description
ESD	Electrostatic discharge
FBS	FlatBuffers schema
GNSS	Global Navigation Satellite System
GPIO	General-purpose input/output
GPS	Global Positioning System
I²C	Inter-Integrated Circuit
I3C	Improved Inter-Integrated Circuit
IDL	Interface Definition Language
IMU	Inertial measurement unit
IO	Input/output
LDO	Low-dropout regulator
LED	Light-emitting diode
MCU	Microcontroller unit
OFL	Optical flow
PAE	Power Adjacent to Ethernet
PDM	Pulse-density modulation
PHY	Physical-layer transceiver

Acronym	Description
PoDL	Power over Data Line
PWM	Pulse-width modulation
RMII	Reduced Media-Independent Interface
ROS 2	Robot Operating System 2
RTC	Real-time clock
RTK	Real-Time Kinematic
RTOS	Real-time operating system
SAI	Serial audio interface
SDHC	Secure Digital High Capacity
SDIO	Secure Digital Input Output
SDK	Software development kit
SPI	Serial Peripheral Interface
UART	Universal asynchronous receiver-transmitter
USB	Universal Serial Bus
USB CDC	USB Communications Device Class
VBUS	USB bus-supply voltage

11. References

- MR-MCXN-T1 Hub repository

60 Zenoh_PX4 Communication on the MR-MCXN-T1 using ZROS

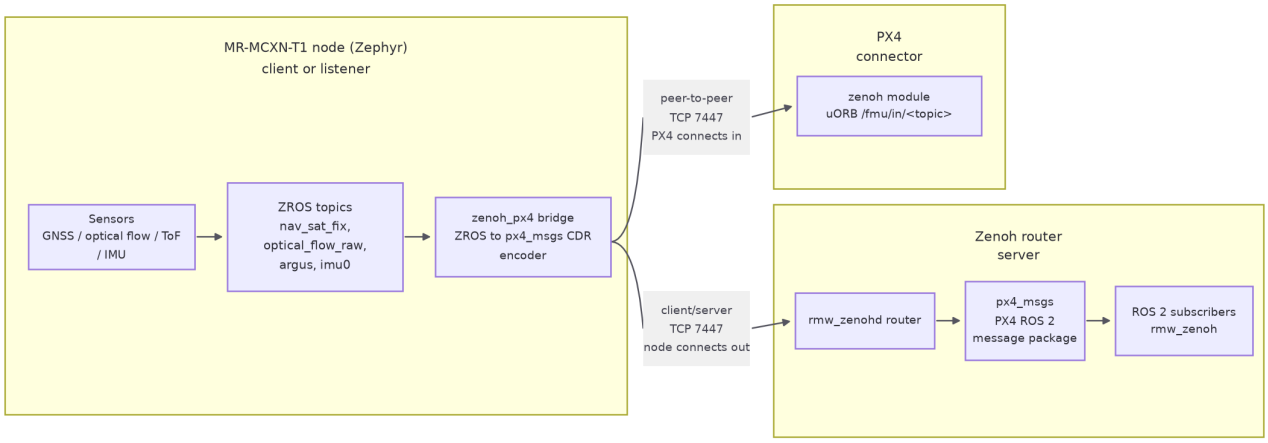
60.1 Overview

This guide describes how to configure an MR-MCXN-T1 node to publish its sensor samples as CDR-encoded `px4_msgs` over zenoh. The `zenoh_px4` bridge encodes each ZROS sensor sample as a CDR `px4_msgs` payload and publishes it on `<domain>/fmu/in/<topic>/...` over zenoh-pico. The MR-MCXN-T1 node supports two link topologies that are covered by this document (the CDR wire payload is byte-identical in both):

- client/server — the MR-MCXN-T1 node is a zenoh client that connects to an zenoh router (the server), typically on a Linux host / NavQ95 A55 core running Linux. The samples then appear as native ROS 2 topics to any `rmw_zenoh` application.
- peer-to-peer — a direct, router-free link to PX4. Here the MR-MCXN-T1 node is the listener and PX4 connects to it; the samples are fused into PX4's uORB. There is no ROS 2 graph discovery; only the CDR payload matters.

It also lists general commands for checking the current network interface, ROS 2 and the zenoh state.

60.2 Topology



60.3 Prerequisites

- Build the MR-MCXN-T1 GNSS / OFLO / IO application and flash it on MR-MCXN-T1. Follow these MR-MCXN-T1 build and flash steps.
- The end-device the MR-MCXN-T1 node is communicating with, depending on the setup:
  - Zenoh router (zenoh server) running on a Linux host / NavQ95 A55 core running Linux. If NavQ95 A55 is used, follow these ROS2 setup steps.
  - PX4 device (zenoh peer) running on RT1176, Tropic, NavQ95 M7 core.
- The MR-MCXN-T1 node's Ethernet interface must have an IP address on the same subnet as the server/peer it communicates with (rmw_zenoh, PX4 target device).

60.4 Default MR-MCXN-T1 node configuration and general commands to check its current state

Each application ships with a different static IP address:

Application	MR-MCXN-T1 node IP	Zenoh mode	Locator
`gnss`	`10.58.0.13/16`	`client`	`tcp/10.58.0.1:7447`
`optical_flow`	`10.58.0.12/16`	`peer`	`tcp/0.0.0.0:7447`
`io_exp`	`10.58.0.14/16`	`client`	`tcp/10.58.0.1:7447`

The mode and the locator are runtime parameters set from the shell; the values above are only the power-on defaults supplied by the application `prj.conf`. A `config net` change is staged only — the session is reopened on the explicit `zenoh_px4 restart`.

Network interface commands

Check the current network interface status (such as the IP address and the link state)

```
net iface
```

Confirm `oper=UP, carrier=ON` and `100 Mbits full-duplex` before anything else.

Network interface stats

```
net stats
```

Show the active network connections — look for an `ESTABLISHED` entry to port `7447`:

```
net conn
```

ZROS commands

List every ZROS topic on the MR-MCXN-T1 node

```
zros topic list
```

Show which ZROS nodes are running (`zenoh_px4` must be present)

```
zros node list
```

Inspect the ZROS topics (example: OFLO application)

```
zros topic echo optical_flow_raw
zros topic info optical_flow_raw
```

Zenoh commands

Check whether the zenoh is running

```
zenoh_px4 status
```

60.5 Scenario 1: MR-MCXN-T1 node (client) - Zenoh router (server)

In client mode the MR-MCXN-T1 node connects to a remote zenoh router (`rmw_zenohd`), so the `<ip>:<port>` given to `config net client` is the remote router address. The MR-MCXN-T1 node appears in the ROS 2 graph and PX4 (or any ROS 2 consumer) reaches it through the router.

Check the zenoh configuration.

```
zenoh_px4 config show
```

Expected `config show` output: `mode: client` , the configured locator, and `running: 1` .

If the mode is not in `client` mode, run (or adapt per the locator IP address):

```
zenoh_px4 config net client tcp/10.58.0.1:7447
```

Start the communication

```
zenoh_px4 start
```

Start the communication on the other device (the server) and confirm MR-MCXN-T1 topics are visible there.

60.6 Scenario 2: MR-MCXN-T1 node (peer) - PX4 (peer)

PX4's zenoh module glue (`ZENOH::setupSession()`) only ever inserts the zenoh `MODE` and `CONNECT` config keys; it never inserts `LISTEN` . So a stock PX4 can only be a connector in peer mode — it cannot open a listening peer transport. To make a direct peer link work without patching PX4, the roles are reversed from the usual embedded pattern: the MR-MCXN-T1 node is the LISTENER and PX4 connects to it. zenoh-pico supports this on the MR-MCXN-T1 node side via `Z_FEATURE_UNICAST_PEER` plus the `LISTEN` config key. Semantic caveat (peer mode). Unlike PX4, where `net peer <ip>:<port>` is the REMOTE connect target, here peer mode makes the MR-MCXN-T1 node the LISTENER, so the `<ip>:<port>` is this MR-MCXN-T1 node's LOCAL bind/listen address. Leave it at `tcp/0.0.0.0:7447` to accept on all interfaces, or pin it to the MR-MCXN-T1 node's own address (`tcp/10.58.0.12:7447` for optical_flow, `tcp/10.58.0.13:7447` for gnss) to bind a single interface.

Check the zenoh configuration.

```
zenoh_px4 config show
```

Expected `config show` output: `mode: peer` , locator `tcp/0.0.0.0:7447` , and `running: 1` .

If the mode is not in `peer` mode, run

```
zenoh_px4 config net peer tcp/0.0.0.0:7447
zenoh_px4 restart
```

Start the communication

```
zenoh_px4 start
```

Start the communication on the other device (the peer) and confirm MR-MCXN-T1 topics are visible there.

PX4 steps (in communication with MR-MCXN-T1 node)

The key parameters must match on both sides, otherwise the key match silently breaks and no sample is ever delivered:

- `ZENOH_DOMAIN_ID` == `CONFIG_SYNAPSE_ZENOH_PX4_DOMAIN_ID` (both default 0).
- `ZENOH_KEY_TYPE_HASH` == `CONFIG_SYNAPSE_ZENOH_PX4_RIHS_HASH` (both default `y`).

There is a known limitation on PX4: the Zenoh communication is not working with the standard publishers/subscribers default configuration. It needs to be trimmed. `sensor_optical_flow` and `distance_sensor` ship stock in `dds_topics.yaml` (no PX4 rebuild). `sensor_gps` / `sensor_mag` must be added under `subscriptions:` and PX4 rebuilt. Run these commands on PX4 NSH shell:

```
zenoh stop
cd /fs/microsd/zenoh
echo -n "" > pub.csv
echo -n "" > sub.csv

(for oflo)
zenoh config add subscriber /fmu/in/distance_sensor distance_sensor
zenoh config add subscriber /fmu/in/sensor_optical_flow sensor_optical_flow

(for gnss)
zenoh config add subscriber /fmu/in/sensor_gps sensor_gps
```

Check the current Ethernet configuration on PX4. Make sure the IP address is on the same subnet as the MR-MCXN-T1 node and change it if necessary:

```
ifconfig

ifconfig eth0 10.58.0.10 netmask 255.255.0.0
```

Configure the zenoh network peer connection with the MR-MCXN-T1 node IP address and port. Example command for communication with the MR-MCXN-T1 OFLO (optical flow) application:

```
zenoh config net peer tcp/10.58.0.12:7447#iface=eth0
```

Start the communication

```
zenoh start
```

Monitor the topics in real time

```
uorb top sensor_optical_flow distance_sensor
```

Listen the topics

```
listener distance_sensor
listener sensor_optical_flow
```

60.7 Additional Notes

Enabling UART messaging on the Holybro DroneCAN H-RTK F9P Helical (MR-MCXN-T1 IO application use case)

The MR-MCXN-T1 IO is connected, through UART1, to the Holybro DroneCAN H-RTK F9P Helical UART port in order to receive GNSS packets. The following steps are needed on the Holybro DroneCAN H-RTK F9P Helical to enable UART messaging because it is silent by default. This is a one-time receiver configuration, applied over the module's USB port — which connects directly to the u-blox ZED-F9P and enumerates as a COM port — and written to the receiver's flash so that it survives a power cycle.

1. Connect the F9P's USB port to the host and open u-center (classic, 22.07 or newer — not u-center 2, which targets M10/X20 receivers).
2. Select `Receiver -> Port ->` the F9P's COM port. The baud rate is irrelevant over USB CDC. Confirm the green data-flow indicator at the bottom right.
3. Open `View -> Generation 9 Configuration View`. This is the CFG-VALSET editor used by the F9 generation.
4. Set the configuration keys below. For each key, tick RAM, BBR and Flash in the Layers checkboxes before applying, then press Load into Configuration / Set.

Configuration key	Value	Notes
`CFG-MSGOUT-UBX_NAV_PVT_UART2`	`1`	Required. The only message the MR-MCXN-T1 node consumes
`CFG-UART2-ENABLED`	`1`	Verify; usually already set
`CFG-UART2-BAUDRATE`	`38400`	Verify; must match the MR-MCXN-T1 node's `current-speed`
`CFG-UART2INPROT-UBX`	`1`	Verify; required for the MR-MCXN-T1 node to query the receiver
`CFG-UART2OUTPROT-UBX`	`1`	Verify; required for the receiver to answer
`CFG-UART2OUTPROT-NMEA`	`0`	Optional; keeps the port free of unparsed NMEA

61 MR-ESC-AM32

The MR-ESC-AM32 is the AM32-based Electronic Speed Controller (ESC) motor-control platform of the NXP Mobile Robotics family (MCX A / AM32).

- Overview
  - Functional block diagram
  - Main components
- Getting Started
  - Deploy and Run Quick Start
- User Manual
  - Introduction
  - Finding platform resources
  - Getting ready
  - Getting to know the hardware
  - Board components and connectors
  - Firmware features
  - Installing and configuring software tools
  - Configuring the hardware
  - Abbreviations
  - References
- Application Notes
  - Hardware Architecture and Reference Design

62 Overview

The MR-ESC-AM32 is the electronic speed controller (ESC) reference design of the NXP Mobile Robotics family. It is a brushless DC (BLDC) motor controller based on the open-source AM32 firmware and implemented with an NXP MCX A microcontroller. It is designed for drones, robotic vehicles, and other applications requiring sensorless BLDC motor control under the direction of an external flight controller or motor-control system. The board is intended as a validated ESC starting point that customers can evaluate, reproduce, and customize for their own vehicles.

The controller is built around an NXP MCXA133 microcontroller, an Arm Cortex-M33 MCU running the NXP port of the AM32 firmware. It implements six-step commutation using back-EMF (BEMF) zero-cross detection and accepts RC-PWM, DShot, and bidirectional DShot control interfaces.

62.1 Functional block diagram

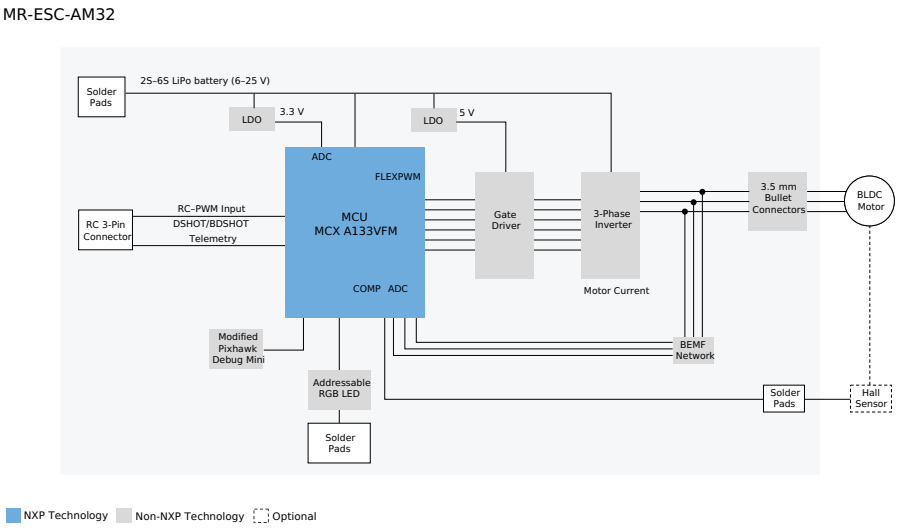


Figure 1. MR-ESC-AM32 functional block diagram

62.2 Main components

The reference design integrates the following main components and interfaces:

- NXP MCXA133 microcontroller (Arm Cortex-M33) running the AM32 firmware
- DRV8300N three-phase gate driver and a MOSFET-based power stage
- Sensorless BEMF-detection network for six-step commutation
- RC-PWM, DShot, and bidirectional DShot control interfaces
- KISS-format serial telemetry (TX) output
- Addressable RGB status LED and a modified Pixhawk debug connector
- Optional Hall-sensor / quadrature-encoder solder pads (hardware provision only)

NXP also provides the complete hardware Bill of Materials and design files, enabling customers to evaluate, reproduce, and customize the reference platform for their own products.

For overview information, documentation, software, tools, and ordering details, see the MR-ESC-AM32 product page on the NXP web site.

63 Getting Started

Deploy and run on the MR-ESC-AM32 ESC.

- Deploy and Run Quick Start

64 Deploy and Run Quick Start

MR-ESC-AM32 is an AM32 BLDC ESC running on an MCXA133 MCU. Flash the bootloader and the application with LinkServer (MCU-Link-MR) or Segger J-Link, then configure the ESC over a pass-through serial link.

64.1 Get the artifacts first

Every platform assumes the images are already available — either downloaded or built with the SDK:

```
# prebuilt (fastest)
west sdk download --platform mr-esc-am32-mcxa133      # lands in <west topdir>/bin/mr-esc-
am32-mcxa133/

# or build from source
west sdk build    --platform mr-esc-am32-mcxa133  # lands in <west topdir>/{build,bin}/...
```

- Docker required. Downloading and building are done inside a Docker container built from the SDK's `Dockerfile` — see the NXP Robotics SDK README for how to set it up. All `west` commands are assumed to run inside that container.
- The SDK builds/downloads images but does not flash them — deployment is done with the board's native tool.
- Where do the artifacts land? `<west topdir>` is the west workspace root — the directory containing `.west/`, created by `west init`. The `bin/` directory is created next to the `robotics-sdk/` clone, not inside it:

```
<west topdir>/
├── .west/
├── robotics-sdk/         <- the SDK clone (west init -l)
├── build/                <- west sdk build intermediates
└── bin/mr-esc-am32-mcxa133/       <- final artifacts land here
```

For MR-ESC-AM32 the platform is `mr-esc-am32-mcxa133`. Run `west sdk list` for the authoritative list.

64.2 What to flash

The AM32 stack produces two images. The bootloader is programmed once; during normal development only the application image needs to be updated. On a blank device both images must be flashed (bootloader + application) together.

The AM32 version is kept in the filename, so the exact version suffix depends on the release that is built or downloaded:

Image	Artifact (in `bin/mr-esc-am32-mcxa133/`)
AM32 bootloader	`AM32_A153_BOOTLOADER_PB2_V18.hex`

Image	Artifact (in `bin/mr-esc-am32-mcxa133/`)
AM32 application	`AM32_FRDM_A153_2.20.hex`

`.bin` and `.elf` variants are collected alongside each `.hex`. The `.hex` files carry their own load addresses, so no explicit flash address is needed when flashing them.

ℹ️ The version suffix (`_V18` , `_2.20` above) tracks the AM32 release; substitute the version that was actually built or downloaded in the flash commands below. Run `west sdk list` and inspect `bin/mr-esc-am32-mcxa133/` for the exact filenames present.

ℹ️ At this time the bootloader must not be trusted to load a new main-application firmware into flash; it is used to load configuration parameters into the MCXA EEPROM/flash. Always re-flash the application image directly with LinkServer or J-Link.

64.3 Connect

- MCU-Link-MR / MCU-Link-Pro (or Segger J-Link) to the ESC's SWD debug connector — used for flash.

  ⚠️ The NXP MCU-Link-MR cable differs from the J-Link cable: NXP requires pin 3 NOT connected (connectors are otherwise identical). Using the wrong cable will fail to connect.

- For ESC configuration, a flight controller flashed in pass-through mode exposes the ESC as a USB/serial port on the host PC (see the Configure the ESC section below).

64.4 Flash with LinkServer

Device name: `MCXA133` (interface SWD, speed 4000 kHz).

1. Flash the bootloader (first time / blank device)

   ```
   LinkServer flash MCXA133 load AM32_A153_BOOTLOADER_PB2_V18.hex
   ```

2. Flash the application

   ```
   LinkServer flash MCXA133 load AM32_FRDM_A153_2.20.hex
   ```

   On Windows the LinkServer path is e.g. `C:\NXP\LinkServer_<ver>\LinkServer.exe` .

3. Reset & run

   Reset the ESC; the bootloader chainloads the AM32 application.

64.5 Flash with Segger J-Link (alternative)

Run `JLink.exe -CommanderScript temp.jlink` , where `temp.jlink` contains:

```
device MCXA133
si SWD
speed 4000
connect
r
h
erase
loadfile AM32_FRDM_A153_2.20.hex
r
g
exit
```

Repeat with `loadfile AM32_A153_BOOTLOADER_PB2_V18.hex` to program the bootloader on a blank device.

64.6 Configure the ESC

The ESC needs calibration/config to match the motor and the flight controller / RC receiver.

1. Flash a flight controller in pass-through mode so the ESC enumerates as a USB/serial port on the PC.
2. Open the AM32 web configurator (or use `SerialPortConnector.exe`) and connect to that port.
3. If a config error appears after a full-erase flash, load the default configuration and save to flash, then set the relevant parameters — e.g. motor KV (motor-dependent, ~960) and input signal DShot or PWM.
4. Make sure the flight controller / RC receiver output signal matches the ESC's configured input:
    - Configure the FC to send PWM or DShot from QGroundControl, or set the signal in the TX unit for a direct RC-receiver connection.

> ⚠️ Watch connector polarity when switching between the i.MX RT1176 VMU (GND pin on the top side) and an RC RX unit (GND may be on the bottom side).

65 User Manual

Get to know the MR-ESC-AM32 ESC: review the board hardware and components, then set up, flash, and operate the board with the AM32 firmware.

- Introduction
- Finding platform resources
- Getting ready
- Getting to know the hardware
- Board components and connectors
- Firmware features
- Installing and configuring software tools
- Configuring the hardware
- Abbreviations
- References

66 Introduction

An electronic speed controller (ESC) is a (relatively) high-voltage/high-current electronic device used to control the angular speed (measured in rotations per minute, RPM) and torque of a DC electric motor.

The MR-ESC-AM32 is NXP's implementation of the open-source AM32 (Alka Motors) electronic speed controller (ESC), built with NXP devices and compatible with the original design. It is based on the MCXA133 microcontroller.

AM32 is open-source GPL-3.0 brushless motor controller (ESC) firmware for Arm Cortex-M MCUs, designed for multirotor drones, fixed-wing aircraft, and ground vehicles such as buggies, rovers, and crawlers. It is intended to be controlled by a flight controller, such as the MR-VMU-RT1176 or MR-VMU-TROPIC, rather than used as a standalone controller.

The design includes an addressable RGB LED and solder pads for adding additional LEDs. This is useful on drones where the ESC and lighting are both located at the vehicle corners, allowing the LEDs to indicate status or orientation.

66.1 NXP Robotics SDK and upstream open-source software

NXP embraces an open-source development model and actively participates in the communities behind the software used by this product. The NXP Robotics SDK provides a curated and validated set of software components, tools, and documentation that have been tested together on this hardware platform.

NXP regularly contributes improvements and fixes back to upstream repositories. Because contributions are made on a scheduled basis, the latest upstream code and the version included in the NXP Robotics SDK may not always be identical. The SDK release is recommended for users seeking a tested and supported configuration, while advanced users may choose to work directly with the upstream project and its latest features.

67 Finding platform resources

The information page for the MR-ESC-AM32 electronic speed controller is on the NXP web site at https://www.nxp.com/design/design-center/development-boards-and-designs/MR-ESC-AM32. The information page provides overview information, documentation, software and tools, and a Getting Started tab with quick-reference information, including the downloadable assets referenced in this document.

The board design files and the latest hardware reference are published in the public NXP Robotics repository at https://github.com/NXP-Robotics/MR-ESC-MCXA-AM32. The software, images, and build tooling are distributed through the NXP Robotics SDK; follow the NXP Robotics SDK README to set up the SDK and obtain the assets.

68 Getting ready

Working with the MR-ESC-AM32 board requires an AM32-supported flight controller and a Windows/Linux PC.

68.1 Kit contents

- MR-ESC-AM32 board
- Quick Start Guide

68.2 Additional required hardware

- An AM32-supported flight controller
- A small-drone BLDC motor, such as a 2212 920 KV motor

68.3 Assumptions

Familiarity with the AM32 online/offline configurator and the AM32 wiki.

68.4 AM32 firmware

The MR-ESC-AM32 board is expected to ship pre-flashed. If a firmware update is required, newer releases are available from the official AM32 downloads page.

- Bootloader: `AM32_A153_BOOTLOADER_PB2_Vxx.hex`
- Main application: `AM32_FRDM_A153_x.xx.hex`

AM32 firmware development is outside the scope of this guide. For development information, refer to the public resources available on the AM32 website and GitHub repository.

69 Getting to know the hardware

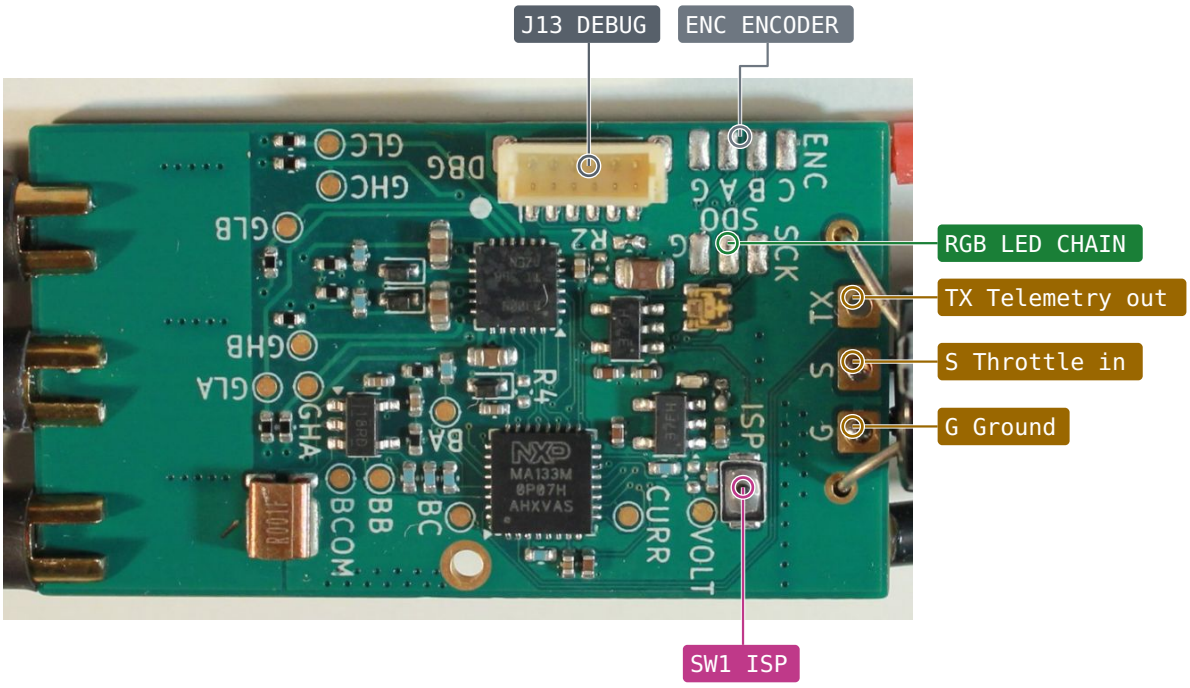
Connectors, solder pads and pinouts of the NXP MR-ESC-MCXA-AM32, a proof-of-concept drone motor ESC (Electronic Speed Controller) built on the NXP MCXA153/MCXA133 MCU and running the open-source AM32 firmware. All pin data below comes from the public KiCad design files in the MR-ESC-MCXA-AM32 repository.

Each card shows three things per pin: the function, the net name in the schematic and the MCU pad the signal ends up on. Most connections are solder pads; the only connector is the debug port.

> Tip: The pinout cards are interactive: hover over a pin to highlight it, and click a pin to copy its net/pad name to the clipboard. On the board-overview figures, click a callout to jump to the matching connector card.

69.1 Board overview

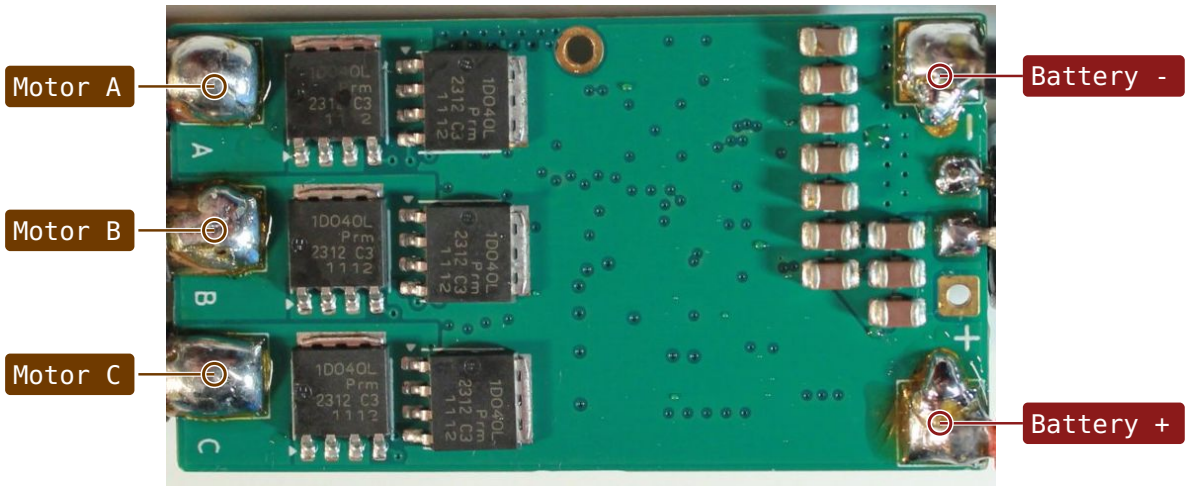
> **Danger**
>
> The board has no reverse polarity protection and no input fuse. Wrong polarity on the battery pads destroys it. Double-check battery polarity before applying power.



Not visible here:

- PWR BATTERY (bottom view)
- MOTOR PHASES (bottom view)

Bottom side, with the battery and motor pads:

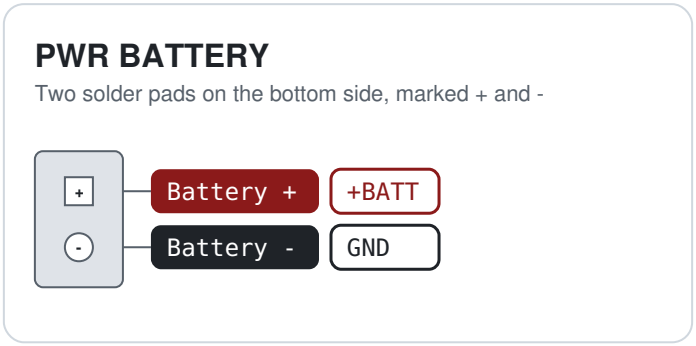


69.2 Connector index

Ref	Function	Connector
ESC ENC ENCODER	Spare MCU pins for a quadrature encoder, solder pads	Four pads marked A, B, I and G, 3.3 V logic
ESC MOTOR PHASES	Three motor phase outputs	Three solder pads on the bottom side, marked A, B and C
ESC PWR BATTERY	Battery input, 6 V to 24 V (2S to 6S LiPo)	Two solder pads on the bottom side, marked + and -
ESC RC SIGNAL	Throttle input and telemetry output, solder pads	Three wire pads marked S, G and TX, 3.3 V logic
ESC RGB LED CHAIN	Output of the on-board RGB LED chain, solder pads	Three pads marked SDO, SCK and G, 3.3 V logic
ESC J13 DEBUG	SWD debug and console (DS-009 debug mini)	JST-SH 1x6 (BM06B-SRSS-TB)

69.3 Power and motor

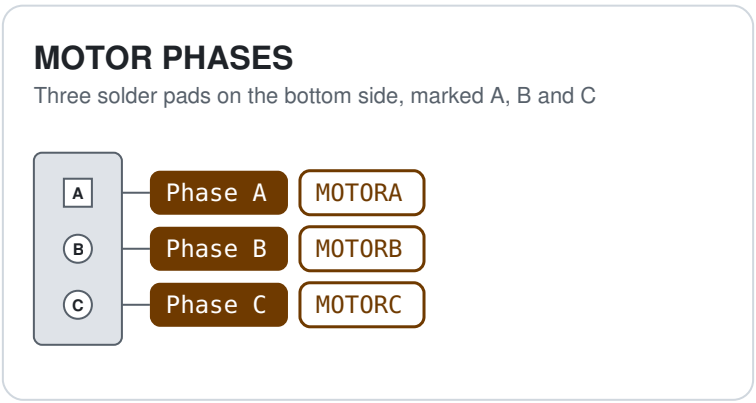
ESC PWR BATTERY



Solder the battery leads here. Input range is 6 V to 24 V, so 2S to 6S LiPo. The 5 V and 3.3 V rails come from two TPS7B92 linear regulators on this input; below about 6 V the 5 V rail drops out. All parts on the battery rail are rated 35 V or more (capacitors 35 V, regulators 40 V, FETs 80 V), which leaves margin for a full 6S pack at 25.2 V. Keep the 470 µF capacitor across the pads when using long battery leads.

> Caution: There is no reverse polarity protection. Check + and - before connecting a battery.

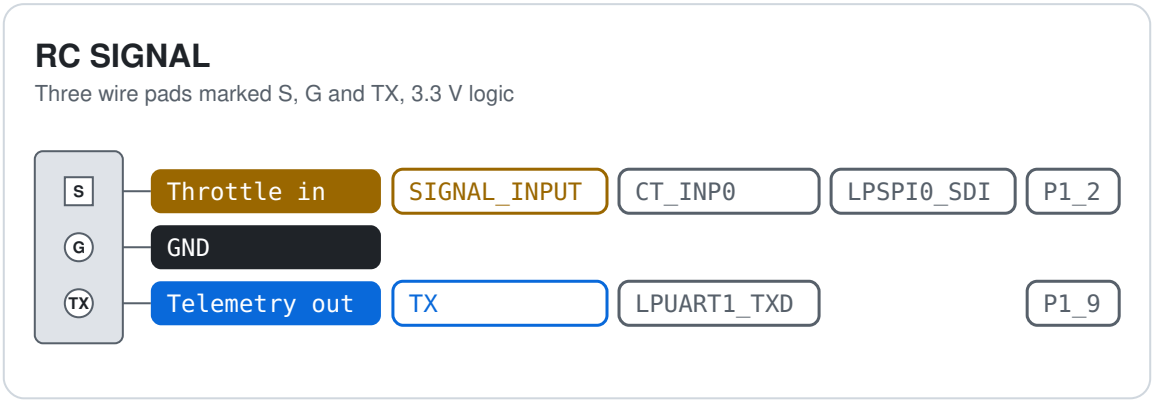
ESC MOTOR PHASES



Each phase is a half bridge of two PSMN3R3-80 FETs (80 V), driven by the gate driver U2. Swap any two phases to reverse the motor direction. The phase voltages also feed the back-EMF comparators of the MCU, which is how the ESC senses the rotor position.

69.4 Control

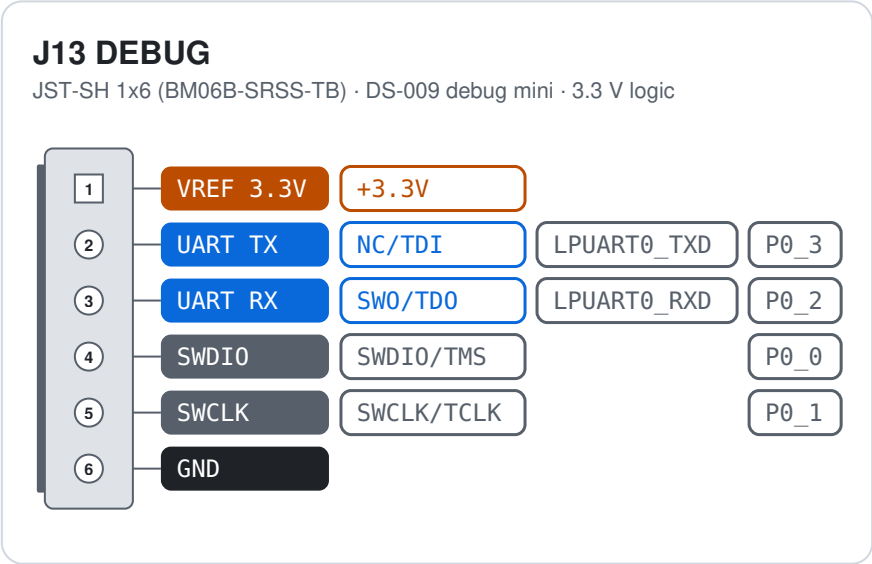
ESC RC SIGNAL



S is the throttle signal from the flight controller. The AM32 port for this MCU detects the input type by itself and supports DShot300, DShot600 (both also as bidirectional DShot) and normal servo PWM (1 ms to 2 ms, up to 400 Hz). DShot150 and other protocols are not supported. AM32 reads the signal with a CTIMER0 capture input and, for bidirectional DShot, switches the same pin to LPSPI0 to send the RPM telemetry back on the signal wire. TX is the AM32 serial telemetry output on LPUART1, 115200 baud, for a flight controller ESC telemetry input.

> Note: The signal input is 3.3 V logic. Do not connect it to a 5 V PWM source without a level shifter.

ESC J13 DEBUG

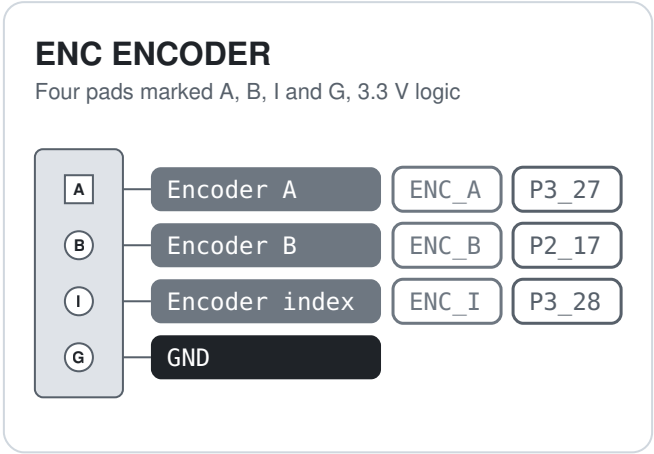


Flash and debug the MCU over SWD with any probe that supports the DS-009 debug mini cable. Pins 2 and 3 are LPUART0 on the MCU, free for a console; AM32 sends its telemetry on LPUART1 (the TX pad) instead. Pin 1 is a 3.3 V reference output for the probe.

> Warning: Pin 1 is a reference output, not a supply input. Never power the board from it.

69.5 Expansion pads

ESC ENC ENCODER



Prepared for an encoder on wheeled robots, where the ESC needs odometry. The AM32 firmware does not use them yet; it only drives A and I as debug outputs. Nothing else on the board is connected to these pins.

ESC RGB LED CHAIN



The MCU drives one APA102-type RGB LED (U5) on LPSPI1. These pads are the data and clock outputs of that LED, so more LEDs of the same type can be chained from here.

69.6 Test pads

Thirteen round pads on the top side are named on the silkscreen. They are probe points for a scope or meter, not for wires.

Pad	Net	MCU pin	What is visible there
GHA, GHB, GHC	GHA, GHB, GHC	P3_1, P3_9, P3_11 drive them	Gate of the high side FET of phase A, B, C. Output of the gate driver U2.
GLA, GLB, GLC	GLA, GLB, GLC	P3_0, P3_8, P3_10 drive them	Gate of the low side FET of phase A, B, C. Output of the gate driver U2.

Pad	Net	MCU pin	What is visible there
BA, BB, BC	BEMFA, BEMFB, BEMFC	P1_0, P1_1, P1_3	Back-EMF of phase A, B, C, scaled down by a divider. The MCU comparator uses these to find the rotor position.
BCOM	BEMF_COMMON	P2_2, P2_3	Virtual star point: the three phase dividers joined. Reference for the back-EMF comparator.
VOLT	VOLTSENSE	P2_16 (ADC0 channel 6)	Battery voltage, scaled down by a divider.
CURR	CURRENT	P2_7 (ADC0 channel 7)	Output of the INA180 current amplifier: motor current as a voltage.
GND	GND		Ground for the probe.

69.7 Buttons and switches

Ref	Board	Function	Type	Notes
SW1 ISP	ESC	ISP boot button	Tactile switch	Hold it while power comes up to start the MCU boot ROM in ISP mode (P3_29 pulled low). Normal boot otherwise.

69.8 AM32 firmware

The board runs the open AM32 firmware. The firmware fork for this MCU, the hex files and how to flash them are described in the user-manual chapters on getting ready, flashing the board and setting up the ESC.

Settings are changed with the AM32 web configurator over the signal wire, pad S. Connect it either through the flight controller (Betaflight passthrough) or with a serial link to a PC. No extra connector is needed. The input protocols the firmware accepts are listed on the RC SIGNAL card.

Useful pages on the AM32 wiki:

- ESC settings explained: what every setting does.
- Recommended settings for freestyle drones.
- Flashing a single ESC.
- PC link with an Arduino: a serial link to the signal wire without a flight controller.
- Hardware design notes: for changes to the board design.
- EEPROM format: where the settings live in flash.

> Tip: AM32 beeps through the motor at power up and after a settings change. Connect a motor before looking for the beep codes.

69.9 Cheat sheet

All connector cards on one A4 landscape sheet: cheat sheet PDF.

69.10 Notes

- The debug port follows the Dronecode DS-009 connector standard (debug mini), so a standard 6-pin debug cable fits.

69.11 References

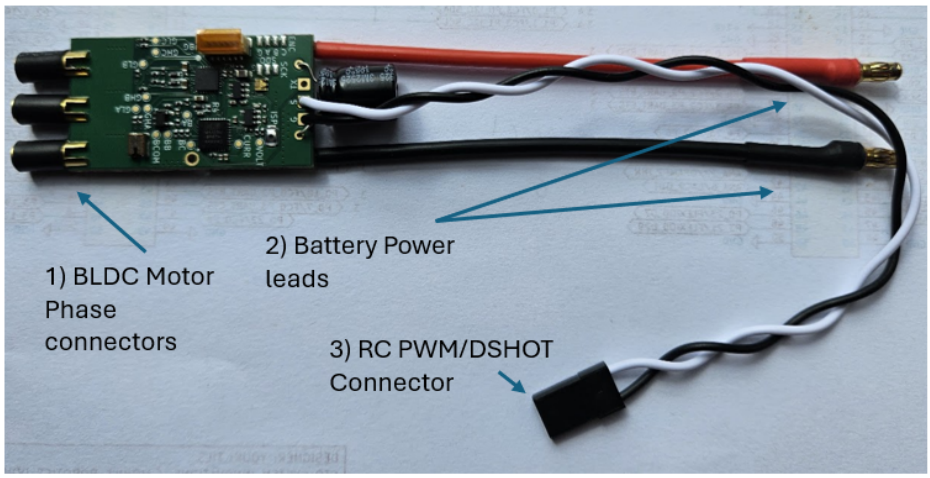
- MR-ESC-MCXA-AM32 board design and documentation

70 Board components and connectors

This chapter identifies the board connectors and featured components. For the functional-block description of the hardware (power stage, gate driver, BEMF network, and so on), see the Hardware Architecture and Reference Design application note.

70.1 Kit featured components

The following figure identifies the important connectors on the board.



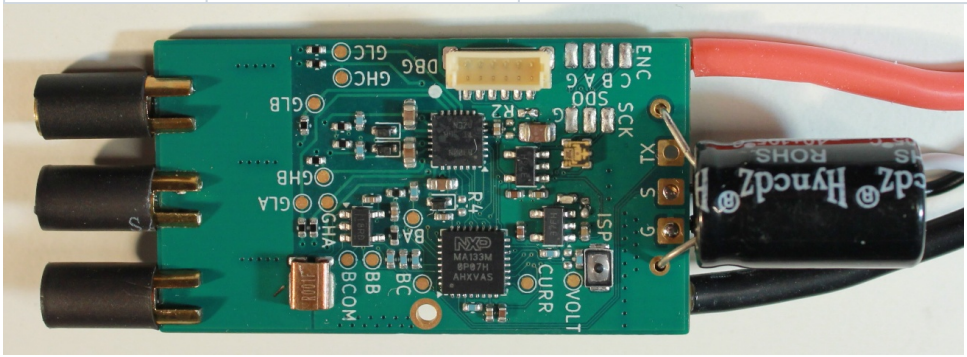
MR-ESC-AM32 board connectors

Number	Description
1	BLDC motor phase — 3.5 mm bullet female connectors
2	Battery power leads — 3.5 mm bullet male (2)
3	Dupont-style 3-pin RC-PWM/DSHOT connector

The table below provides additional detail on the board interfaces.

Part identifier	Connector type	Description
Motor phase A	3.5 mm bullet connector	Motor phase A
Motor phase B	3.5 mm bullet connector	Motor phase B

Part identifier	Connector type	Description
Motor phase C	3.5 mm bullet connector	Motor phase C
Debug	JST-GH 6-pin	SWD debug connector (modified Pixhawk Debug Mini)
VBAT+	Red wire / solder pad	Battery + (2S–6S)
VBAT-	Black wire / solder pad	Battery -
G	3-pin Dupont / black	GND
S	3-pin Dupont / white	Signal pin used for RC-PWM, DShot, BDShot, and configurator
TX	Solder pad	UART telemetry data
RGB LED	Solder pads	Addressable LED solder pads



MR-ESC-AM32 board featured components

70.2 LEDs

The board includes one onboard RGB addressable LED at U5. It can be used to indicate vehicle status or orientation, depending on how the flight-control software uses the AM32 LED-control API. Additional individually addressable RGB LEDs can be connected to the expansion pads to support linear patterns or LED arrays. U5 is an American Bright BB-2020BGR-TRB LED that uses APA102/SK9822-style programming.

70.3 Switches

The board includes one switch labeled ISP. It connects to MCXA133 GPIO `P3_29` and is not currently used by the AM32 software.

70.4 Schematic, board layout, and bill of materials

The schematic, board layout, and bill of materials for the MR-ESC-AM32 board are available on the NXP product page.

71 Firmware features

71.1 Motor control

- Sensorless 6-step (trapezoidal) commutation — BEMF zero-cross detection for rotor position.
- Sinusoidal startup mode — smooth, reliable spin-up for larger/heavier motors.
- Linear throttle response — optimized for FPV and precision flight.
- Variable PWM frequency — tunable dead-time and switching rate for different motor/ESC hardware.

The hardware includes provisions for Hall sensor feedback, but Hall sensor feedback is not currently supported in software.

71.2 Input protocols

Protocol	Notes
Servo PWM	Standard RC 1000–2000 µs
DShot 300	Digital, packet-based
DShot 600	Higher speed digital
Bi-directional DShot	Returns RPM telemetry to the flight controller via GCR encoding

71.3 Telemetry output

- KISS standard ESC telemetry — single-wire serial: temperature, voltage, current, RPM, consumption.
- Bi-directional DShot — eRPM embedded in the DShot response frame (no extra wire needed).

71.4 Configuration and updates

- Firmware upgradeable via:
  - Betaflight passthrough (no programmer needed)
  - Single-wire serial (USB-serial adapter)
  - NXP passthrough image for the RT1176 VMU
- Configurators:
  - AM32 Web Configurator (browser-based)
  - Desktop app (Windows/Linux)

- ESC Configurator (third-party, web serial)

71.5 Bootloader

- A separate bootloader is required for initial flashing (via ST-LINK / GD-LINK / AT-LINK).
- After that, all updates go over the signal wire — no programmer needed.

72 Installing and configuring software tools

1. Download the latest firmware from the AM32 downloads page:
    - Bootloader: `AM32_A153_BOOTLOADER_PB2_Vxx.hex`
    - Main application: `AM32_FRDM_A153_x.xx.hex`
2. Connect the JST 6-pin debug connector to a debugger (J-Link, MCU-Link, etc.).
3. Power up the board, either via battery or PSU (2S–6S).
4. Flash both HEX files to the MR-ESC-AM32 board.

73 Configuring the hardware

1. Connect the MR-ESC-AM32 to an AM32-supported flight controller — signal pin to the motor pin on the flight controller.
2. Connect a battery (2S–6S) or PSU to the input connections and enable power.
3. Connect the flight controller to the PC via a USB cable.
4. Open the online AM32 configurator and connect to the flight controller.
5. Adjust the settings according to the motor requirements.
6. Save the settings, disconnect the flight controller, and reboot the MR-ESC-AM32.
7. Set up the flight controller to control the MR-ESC-AM32.

> Note: Only one ESC can be configured at a time, and it must be connected to AUX1. The AM32 configurator can show up to four ESCs per vehicle, but only the ESC on AUX1 is read.

73.1 Setting up the ESC with the RT1176 VMU

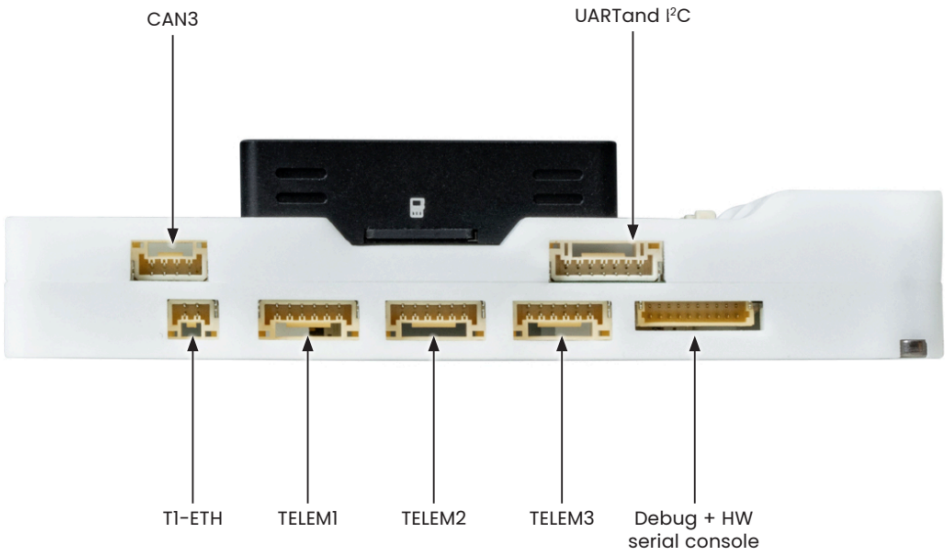
> Note: ArduPilot, Betaflight, and CogniPilot software on flight controllers can program and configure AM32 ESCs using their default passthrough programming support. This section is only needed when using a PX4-based MR-VMU-RT1176 because, at the time this document was published, PX4 did not support AM32 programming. PX4 may add AM32 programming support in the future.

> Caution: Flashing the MR-VMU-RT1176 with ESC passthrough firmware will overwrite the existing PX4 software and configuration. Before proceeding, make sure the procedure to restore the bootloader and reinstall the desired PX4 version on the flight controller is understood. Restoring PX4 is beyond the scope of this document. Review the MR-VMU-RT1176 documentation and confirm the recovery process is understood before continuing.

This section assumes using an MR-VMU-RT1176 programmed with Zephyr RTOS-based ESC passthrough software.

If the MR-VMU-RT1176 has not been programmed for ESC passthrough, complete these steps:

1. Connect the VMU to the PC using USB-C to supply power. Connect the 10-pin Pixhawk Debug Full connector to the debugger, using either MCU-Link-MR or J-Link with a Pixhawk adapter board.



MR-VMU-RT1176 connectors

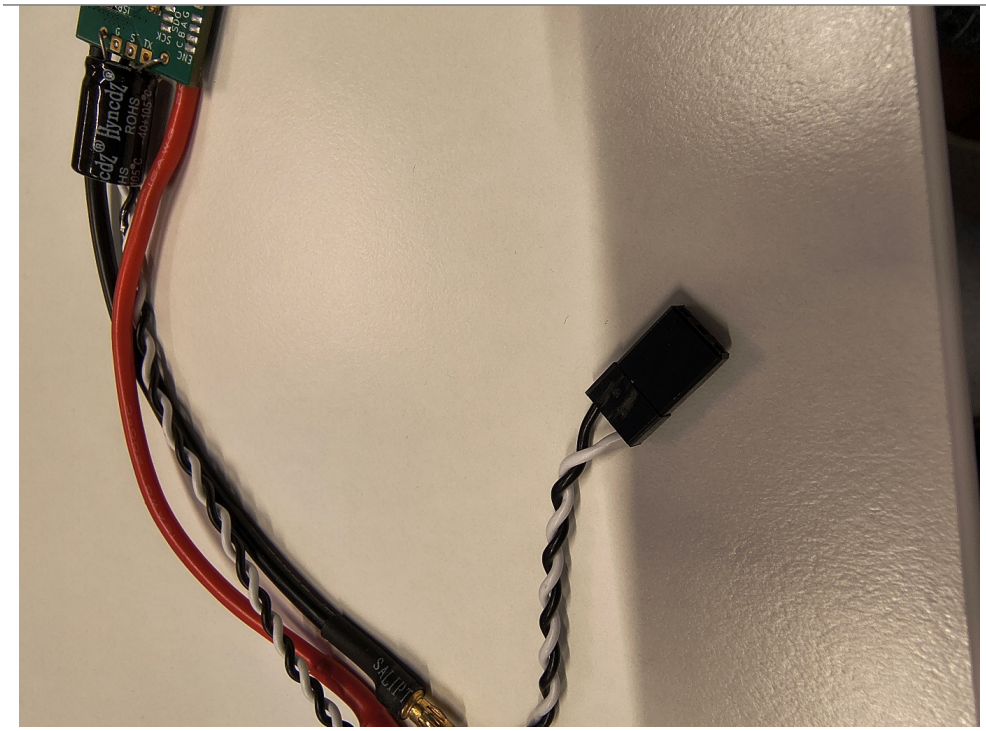
1. Flash the ESC passthrough firmware:
    - If using a J-Link debugger:
        - For Windows: execute `JLink_VMU_ESC_passthrough_flasher.bat`
        - For Linux: make `JLink_VMU_ESC_passthrough_flasher.sh` executable and run it:

```
chmod +x JLink_VMU_ESC_passthrough_flasher.sh
./JLink_VMU_ESC_passthrough_flasher.sh
```

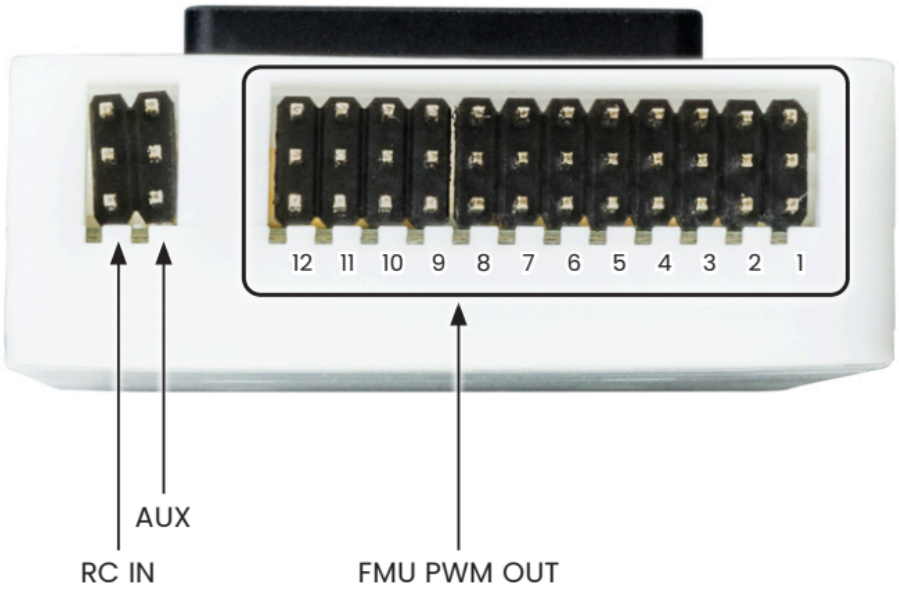
    - If using an MCU-Link-MR debugger:
        - For Windows: execute `MCU_Link_VMU_ESC_passthrough_flasher.bat`
        - For Linux: make `MCU_Link_VMU_ESC_passthrough_flasher.sh` executable and run it:

```
chmod +x MCU_Link_VMU_ESC_passthrough_flasher.sh
./MCU_Link_VMU_ESC_passthrough_flasher.sh
```

2. Follow the instructions shown when the flasher runs to flash the VMU.
3. Disconnect and reconnect the USB-C cable to power-cycle the VMU.
4. Connect the ESC PWM input connector to the VMU, ensuring that the signal and ground are correctly oriented as shown below.



RC PWM connector signal and ground



MR-VMU-RT1176 PWM out pins

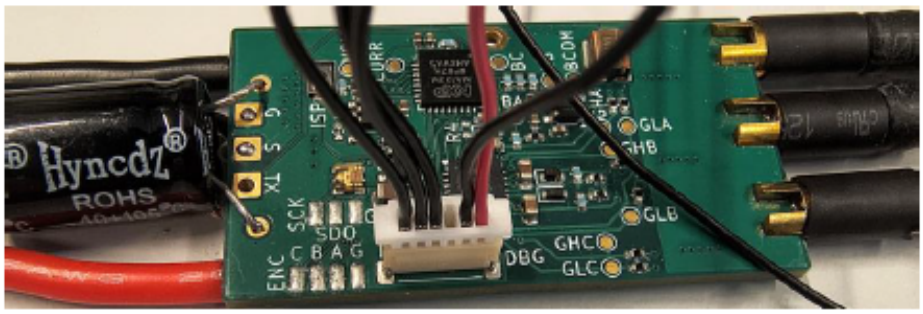
73.2 Working on target

Running the MCXA133 ESC application is a three-step process: write the application(s), configure the ESC, and connect the ESC to a flight controller or RC receiver.

1. Writing the application

Using a Segger J-Link or an NXP MCU-Link-MR(PRO) debugger, flash the main application image (for example `AM32_FRDM_A153_2.20.hex`) into the MCXA133 flash.

> Note: The NXP MCU-Link-MR hardware interface with the ESC uses a different cable from the J-Link one. Although the connectors are identical, NXP requires pin 3 to not be connected — see the image below.



MCU-Link debug connector with pin 3 removed

For J-Link, execute:

```
JLink.exe -CommanderScript temp.jlink
```

The `temp.jlink` script can be created on the fly from a batch file, for example:

```
(
echo device MCXA133
echo si SWD
echo speed 4000
echo connect
echo r
echo h
echo erase
echo loadfile C:\NXP\AM_ESC\NXP_ESC_AM32_Programming\AM32_FRDM_A153_2.20.hex
echo r
echo g
echo exit
) > temp.jlink
```

For MCU-Link-MR/PRO, execute:

```
LinkServer.exe flash MCXA133 load "C:
\NXP\AM_ESC\NXP_ESC_AM32_Programming\AM32_FRDM_A153_2.20.hex"
```

On a blank device it is mandatory to also load the AM32 bootloader together with the main AM32 firmware. The bootloader is loaded with the same operations described above.

2. Configuring the ESC

ESCs require calibration and configuration to match the hardware specifications of both the motor and the flight controller / RC receiver. Calibration of the AM32 ESC is done with the help of a flight controller flashed to run in

passthrough mode, which allows the ESC to be connected to a USB/serial port on a PC. Configuration is done via the web-based AM32 configurator or with a Windows-based serial tool.

> Note: After flashing a new image, an error may appear if the entire flash was erased during programming. To fix this, load the default configuration and save it to flash. Then set the other parameters as required — for example, the KV parameter should be set close to 960 (based on the motor characteristics), and the input signal can be either DShot or PWM depending on how the ESC is connected (directly to an RC receiver or via a flight controller).

3. Connecting the ESC to a flight controller or remote-control receiver

When connecting the ESC via a flight controller or RC receiver, ensure that the signal coming out of the controller/receiver matches the configuration of the ESC. QGroundControl can be used to configure the flight controller to send the signal to the ESC using either PWM or DShot. For a direct connection to an RC receiver, this configuration is done in the remote-control transmitter unit.

> Note: Pay attention to the polarity of the connector pins when switching between the RT1176 VMU and an RC receiver unit. The i.MX RT1176 VMU connectors have the GND pin on the top side, while the receiver may have it on the bottom side.

74 Abbreviations

Acronym	Description
ADC	Analog-to-digital converter
AM32	Alka Motors 32
BEMF	Back electromotive force
BLDC	Brushless DC
COMP	Internal analog comparator
DShot	Digital Shot (digital ESC control protocol)
EEPROM	Electrically erasable programmable read-only memory
eQDC	Quadrature-decoder peripheral
eRPM	Electrical RPM
ESC	Electronic Speed Controller
FC	Flight Controller
GPIO	General-purpose input/output
JTAG	Joint Test Action Group (debug interface)
LDO	Low-dropout linear regulator
LiPo	Lithium-polymer (battery)
MCU	Microcontroller

Acronym	Description
MOSFET	Metal-oxide-semiconductor field-effect transistor
PCB	Printed circuit board
PWM	Pulse Width Modulation
RC-PWM	Radio-control pulse-width modulation
RGB	Red-green-blue (addressable LED)
RPM	Rotations Per Minute
SWD	Serial Wire Debug
TX	Serial-telemetry transmit output
UART	Universal asynchronous receiver/transmitter
VMU	Vehicle Management Unit

75 References

- MR-ESC-AM32 product page — overview, documentation, software, tools, and ordering information
- AM32 wiki
- AM32 downloads
- AM32 firmware source (GitHub)

76 Application Notes

Hardware- and design-level application notes for the MR-ESC-AM32: the ESC hardware architecture and reference design, aimed at hardware engineers and ESC designers.

- Hardware Architecture and Reference Design

Hardware Architecture and Reference Design

The MR-ESC-AM32 is a brushless DC (BLDC) electronic speed controller (ESC) based on the open-source AM32 firmware and implemented using NXP MCX microcontrollers. It is designed for drones, robotic vehicles, and other applications requiring sensorless BLDC motor control. The controller implements six-step commutation using back-EMF (BEMF) zero-cross detection and is intended to operate under the control of an external flight controller or motor-control system. Supported control interfaces include RC-PWM, DShot, and bidirectional DShot.

The hardware consists of an NXP MCXA133 microcontroller, a DRV8300N three-phase gate driver, a MOSFET-based power stage, and the associated analog circuitry required for BEMF sensing and power regulation. In addition to the motor-control functions, the design includes support for an addressable RGB status LED and provisions for quadrature encoder feedback. The quadrature encoder interface is intended for future applications requiring rotor position or wheel-odometry feedback but is not currently implemented in the AM32 firmware.

This application note describes the hardware architecture and major functional blocks of the MR-ESC-AM32 reference design for hardware engineers and ESC designers. The AM32 firmware algorithms and configuration are described in the MR-ESC-AM32 User Manual and in the upstream AM32 project.

About the AM32 community

While the NXP SDK provides a tested version of the AM32 software on the MR-ESC-AM32, AM32 is an open-source, community-supported firmware project. Firmware, configuration tools, and documentation are available through am32.ca and wiki.am32.ca, while source code, issue tracking, and contributions are managed by the am32-firmware organization on GitHub.

The AM32 community maintains a Discord server for user support, developer discussions, hardware-target questions, and community interaction. The current invitation is provided through the Support & Community section of the AM32 website.

Hardware description

MR-ESC-AM32

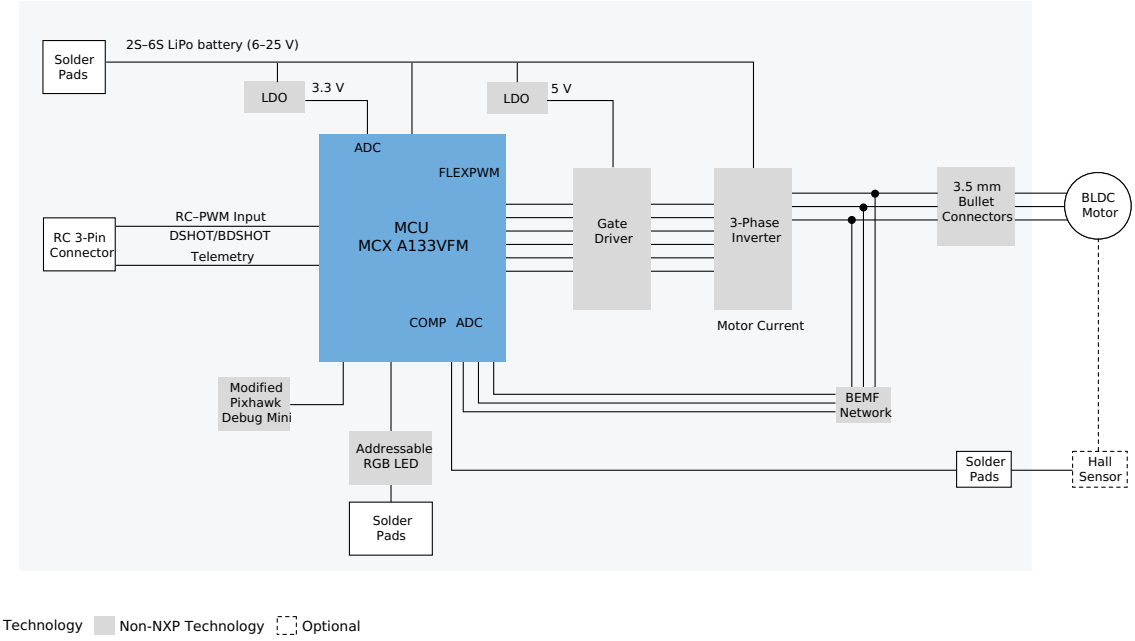


Figure 1. ESC system block diagram

The design consists of an NXP MCXA133 MCU, a DRV8300N gate driver, a three-phase MOSFET bridge, and a sensorless BEMF-detection network used by the AM32 firmware for six-step commutation. This BLDC motor-driver architecture is common and is not covered in detail here.

The board also includes:

- Modified Pixhawk Debug Mini connector for programming and debugging
- Green power LED and an addressable RGB status LED (extendable off board)
- Optional Hall-sensor / encoder solder pads (hardware provision only)
- Telemetry support through the bidirectional DShot protocol

> Note: The Hall-sensor / encoder interface is present on the PCB but is not supported by the AM32 software implementation.

PWM input interface

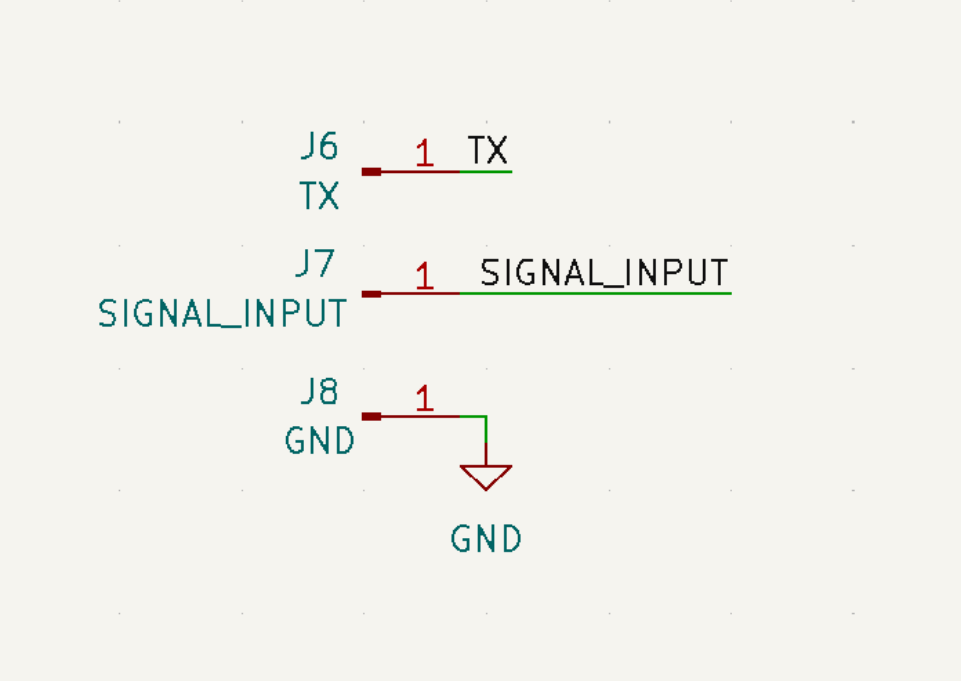


Figure 2. PWM, RC-PWM, DShot and TX interface

The MR-ESC-AM32 is intended to be controlled by a flight controller such as the MR-VMU-RT1176 or MR-VMU-TROPIC, or any other device capable of generating RC-PWM control signals, rather than operating standalone. The primary control interface is a standard 3-pin RC connector supporting RC-PWM input.

The AM32 firmware also supports DShot and bidirectional DShot telemetry modes for modern flight-controller integration.

The control signal is routed directly to the MCXA133 MCU, where timer-capture and PWM peripherals decode the throttle command. The MCU executes the AM32 sensorless commutation algorithm and generates six gate-drive control signals through its FlexPWM peripherals.

TX telemetry pin

The TX signal is supported by the NXP AM32 port as a dedicated serial-telemetry output. TX is routed from an MCXA133 UART transmit pin to a solder pad and also to pin 3 of the debug connector (J13).

AM32 uses it for KISS-format serial telemetry, reporting:

- temperature
- battery voltage
- current
- accumulated consumption
- electrical RPM

TX is optional and is not required for motor control. It is separate from bidirectional DShot telemetry, which returns eRPM over the main DShot signal wire and does not require TX. TX may be connected to a compatible flight-controller telemetry input or to a suitable 3.3 V USB-to-UART receiver for diagnostics.

Note: TX should not normally be connected to the debugger through J13. A modified debug cable must be used with pin 3 removed: a debugger or probe could load or drive the telemetry signal and interfere with operation or SWD access.

Power supply

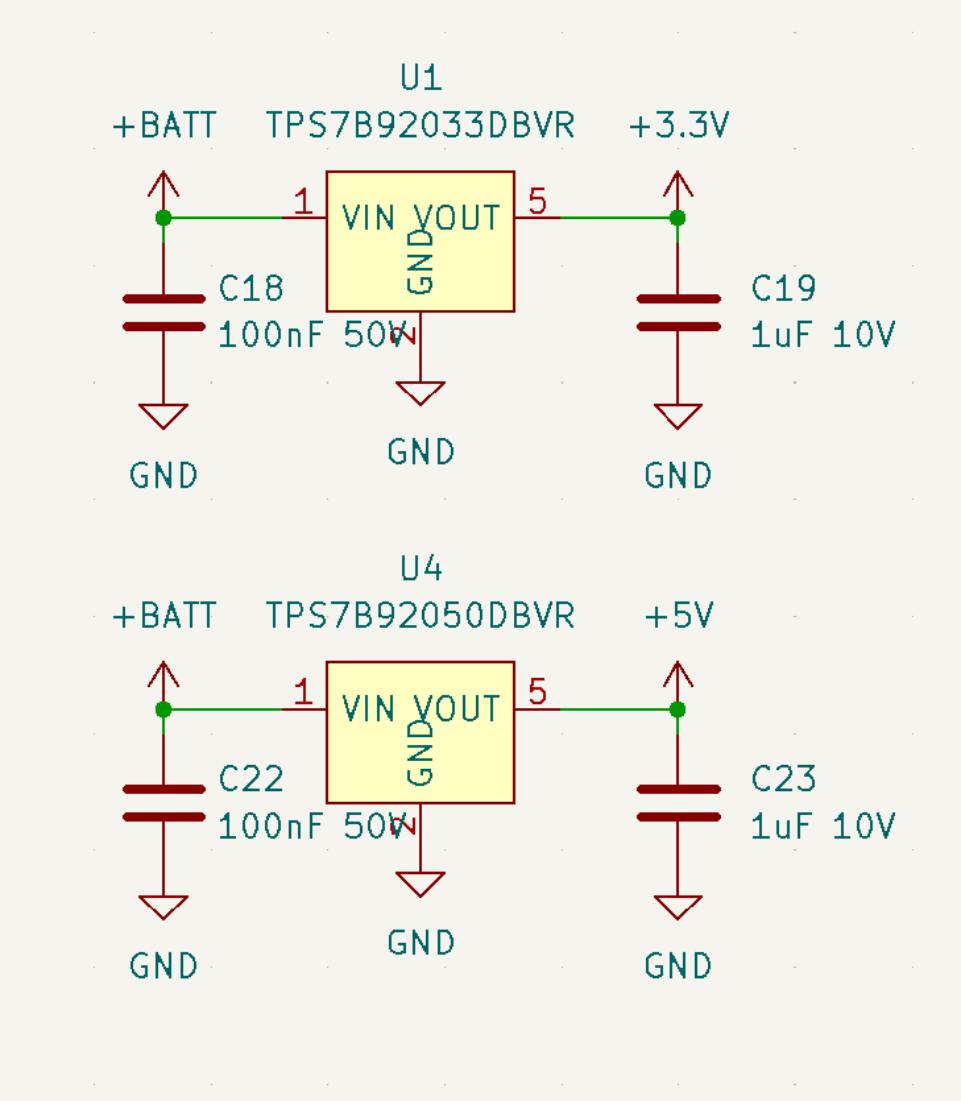


Figure 3. Power regulators

The ESC is powered directly from a 3S to 6S LiPo battery input.

- Approximately 6 V to 25 V input is possible.
- A 3S LiPo has a typical usable range of 10.5 V to 12.6 V.
- A 6S LiPo has a typical usable range of 21.0 V to 25.2 V.

Note: Although the design may operate with a 2S battery, this configuration has not been validated.

The battery voltage serves two purposes:

- It supplies the high-current three-phase inverter stage.
- It feeds the onboard linear regulators.

The board contains two regulated rails:

- 5 V power supply — an LDO generates 5 V from the battery input. This rail powers the DRV8300N gate driver and supporting analog circuitry.
- 3.3 V power supply — a second LDO generates 3.3 V for the MCXA133 MCU, the BEMF-sensing circuitry, logic interfaces, and status LEDs.

This simple dual-LDO architecture minimizes BOM cost, minimizes noise, and follows the original AM32 reference-hardware philosophy.

Back-EMF (BEMF) detection network

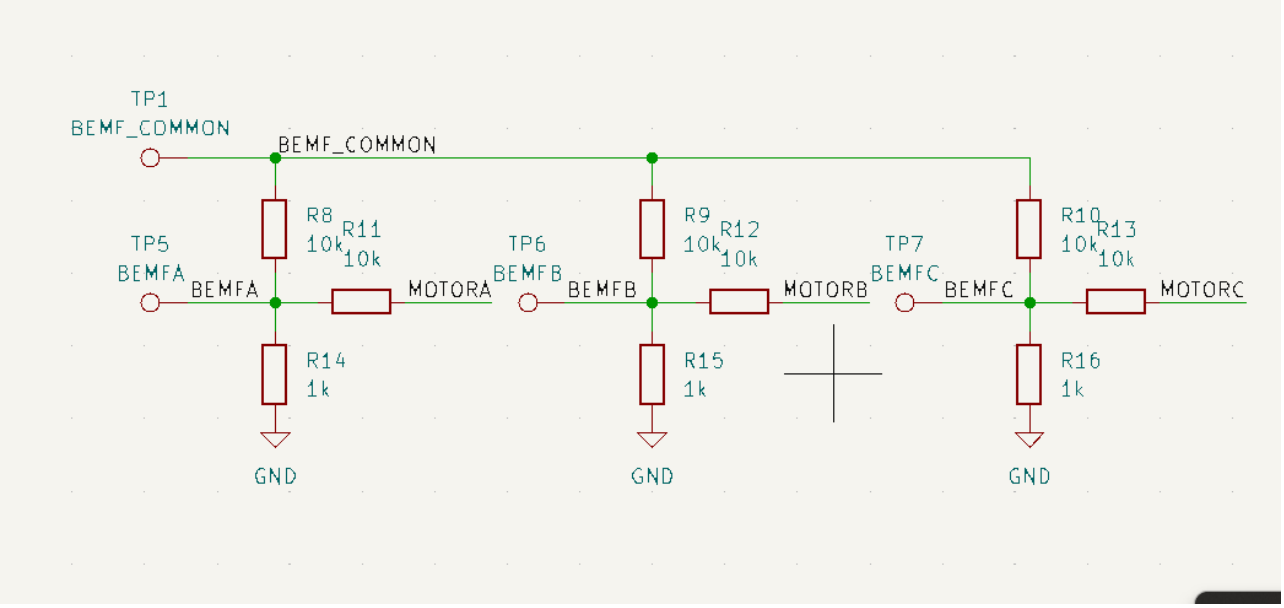


Figure 4. BEMF network

Rotor position is estimated from motor back electromotive force (BEMF) rather than from Hall sensors.

During operation:

- Two motor phases are actively driven.
- The third phase is left floating.
- The floating-phase voltage contains the motor’s back-EMF waveform.

The BEMF network scales and filters this voltage before presenting it to the MCU comparator and ADC inputs. The MCXA133 MCU uses ADC channels and an internal analog comparator (COMP) to detect the BEMF zero-crossing event.

The BEMF network typically performs:

- Voltage division of the motor-phase voltage.

- RC filtering to suppress switching noise.
- Creation of a virtual neutral reference.
- Comparator-input conditioning.

When the floating phase crosses the virtual-neutral voltage, the comparator generates a zero-cross event. The AM32 firmware then predicts the next commutation point and advances the motor to the next state. This approach eliminates the need for rotor-position sensors while maintaining low cost and compact size.

Gate driver and power stage

The power stage consists of a DRV8300N gate driver controlling a three-phase MOSFET inverter.

DRV8300N gate driver

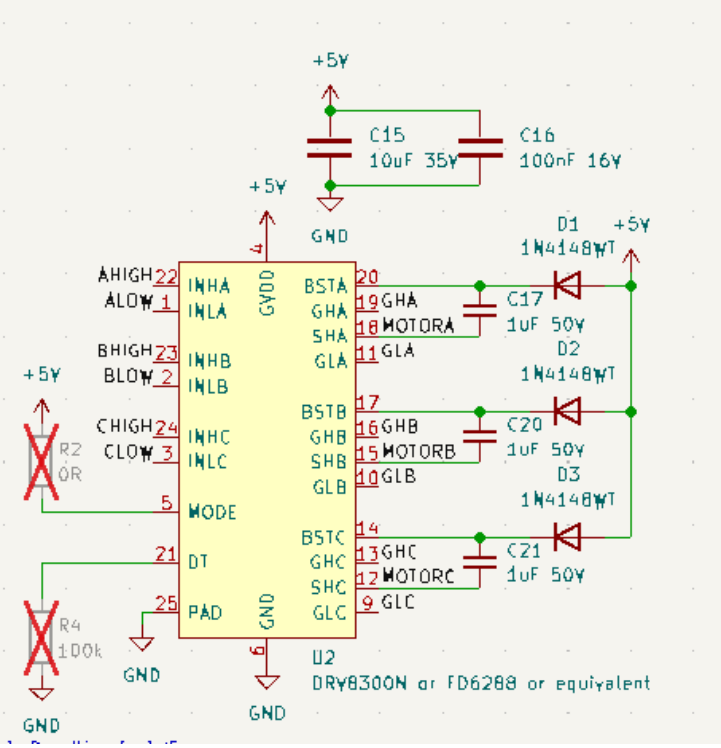


Figure 5. Gate driver

The DRV8300N performs several functions:

- Level shifting between MCU logic and MOSFET gates
- High-side gate-drive generation
- Low-side gate-drive generation
- Dead-time management
- Fast switching of large MOSFET gate capacitances

The MCU FlexPWM module generates the PWM waveforms while the DRV8300N provides the current required to rapidly charge and discharge the MOSFET gates. The mode and dead-time settings configure the device for “6 PWM” control mode, where software controls the dead time rather than the gate driver itself.

Three-phase inverter

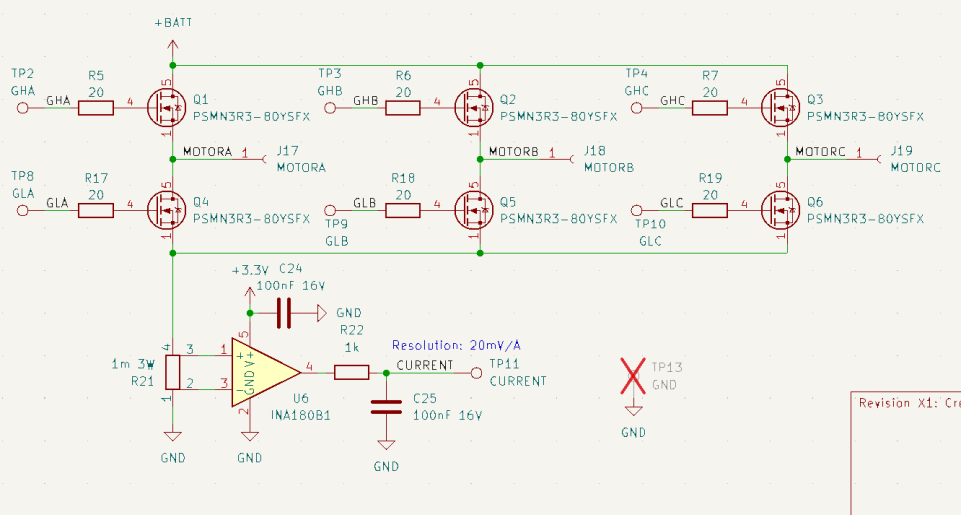


Figure 6. Power FETs and current measurement

The inverter consists of six N-channel MOSFETs arranged as:

- Phase A high-side / low-side
- Phase B high-side / low-side
- Phase C high-side / low-side

The design uses Nexperia PSMN1R4-40 MOSFETs.

> Note: This board is provided as an example implementation and does not establish a guaranteed current rating. Although the MOSFETs are rated for approximately 190 A to 240 A at the device level, the achievable ESC current is substantially lower and depends on the complete thermal design, including PCB copper area, airflow, cooling, and operating conditions. The user is responsible for implementing and validating a thermal design that supports the desired continuous and peak current ratings. Current estimates of approximately 50 A peak are anticipated but remain subject to full validation testing.

Motor-drive operation

The AM32 firmware implements:

- Sensorless six-step commutation
- Trapezoidal drive
- Variable PWM frequency (optional / configurable)
- Adjustable dead time
- Sinusoidal startup sequence (optional / configurable)

The MCU calculates the active commutation sector, drives the appropriate gate-driver inputs, and regulates motor power through PWM modulation of the active MOSFET pair. The AM32 software and algorithms are outside the scope of this document; refer to the upstream AM32 repositories and community forums for support.

Debug interface

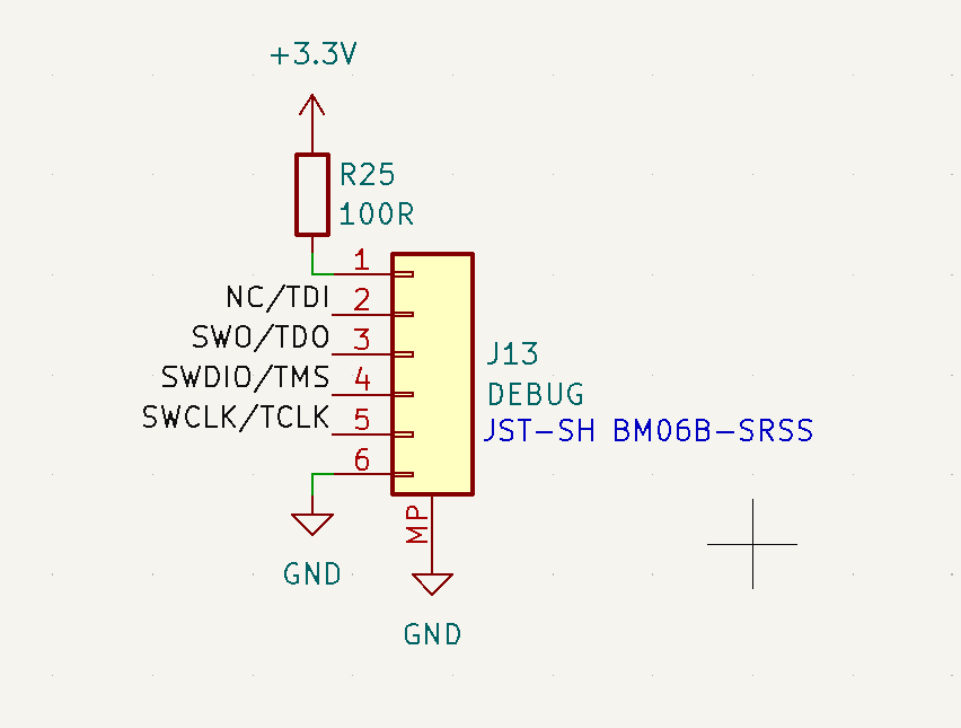


Figure 7. Modified DroneCode Debug Mini interface

J13 is a 6-pin JST-GH connector and provides the hardware programming and debug connection for the MCXA133 MCU. The connector is based on the standard Pixhawk Debug Mini interface but uses a modified pin assignment for this board. A correspondingly modified debug cable must be used: pin 3, which is assigned as a console output on the MR-ESC-AM32, should normally be left disconnected from the debugger. Connecting an unmodified debugger cable may drive or load this console signal and is not recommended.

Any SWD-capable programmer supporting MCX MCUs may be used to program this board. The MCU-Link-MR already includes the corresponding 6-pin JST-GH Dronecode Debug Mini and Standard interface connectors. Otherwise, a Dronecode / PX4 / Pixhawk JTAG-connector debug adapter may be required as documented at docs.px4.io.

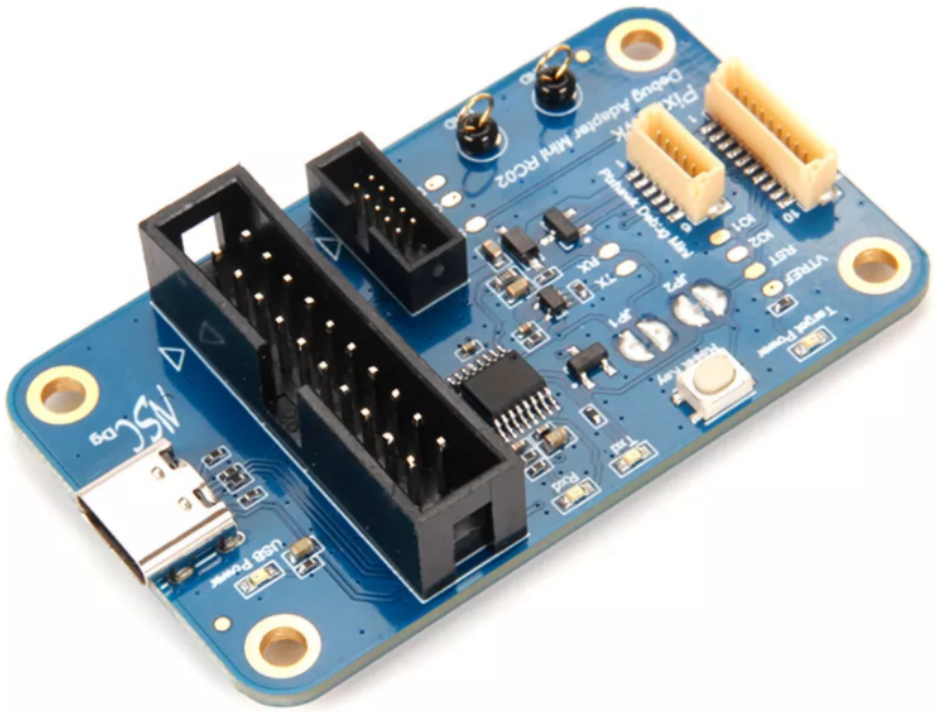


Figure 8. Example SWD (JTAG) hardware debugging interface adapter

Power LED and voltage sense

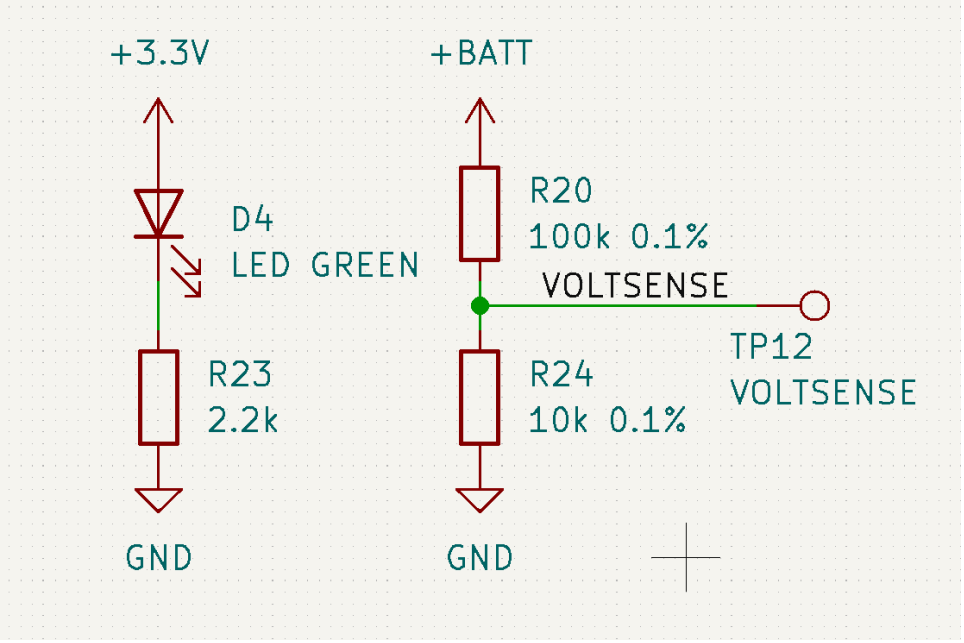


Figure 9. Power LED and voltage sensing circuit

A green LED indicates that the 3.3 V rail is present.

The VOLTSENSE circuit uses a 100 kΩ / 10 kΩ divider to scale the battery voltage by approximately 11:1 for measurement by an MCU ADC input. AM32 converts the ADC result to the estimated battery voltage and uses it for voltage telemetry and configured low-voltage protection. A dedicated test point provides the divided signal. VOLTSENSE is independent of the phase BEMF-sensing network.

Addressable RGB LED and interface

The board includes a BB-2020BGR-TRB individually addressable RGB LED using an APA102/SK9822-compatible two-wire synchronous serial interface with separate clock and data signals.

Power, ground, clock, and data are also available on interface solder pads, allowing additional compatible LEDs to be connected off board in a daisy chain.

The AM32 firmware includes software commands for controlling LED color and lighting behavior. The onboard LED and any external LEDs can be used to indicate controller or flight status, forward and reverse operation, or application-defined colors.

> Note: AM32 software does not currently control LED-strip sequences or lighting patterns. The external supply and interconnect must be reviewed and sized to accommodate the number of added LEDs and their maximum current.

The NXP port's observed default status colors are:

- Bootloader mode: RGB LED off
- Main application running, EEPROM configured, but no flight-controller signal: red
- ESC armed: green

Hall / encoder interface

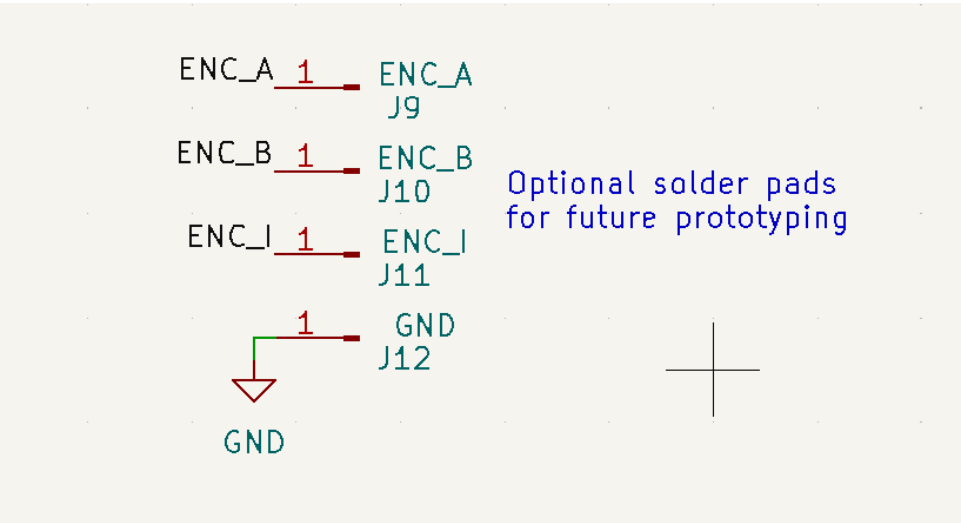


Figure 10. Hall encoder interface (not used)

The board provides solder-pad connections for an optional Hall-effect or quadrature-encoder interface. The encoder A and encoder C signals are routed directly to GPIO pins on the MCXA133 MCU that can be configured for the MCU's eQDC quadrature-decoder peripheral. A nearby ground pad provides the signal reference. This hardware provision could support rotor-position or wheel-odometry feedback in a custom motor-control application.

> Note: The interface is not implemented or supported by the current AM32 firmware; using it requires alternative firmware or additional software development.

ISPMODE switch

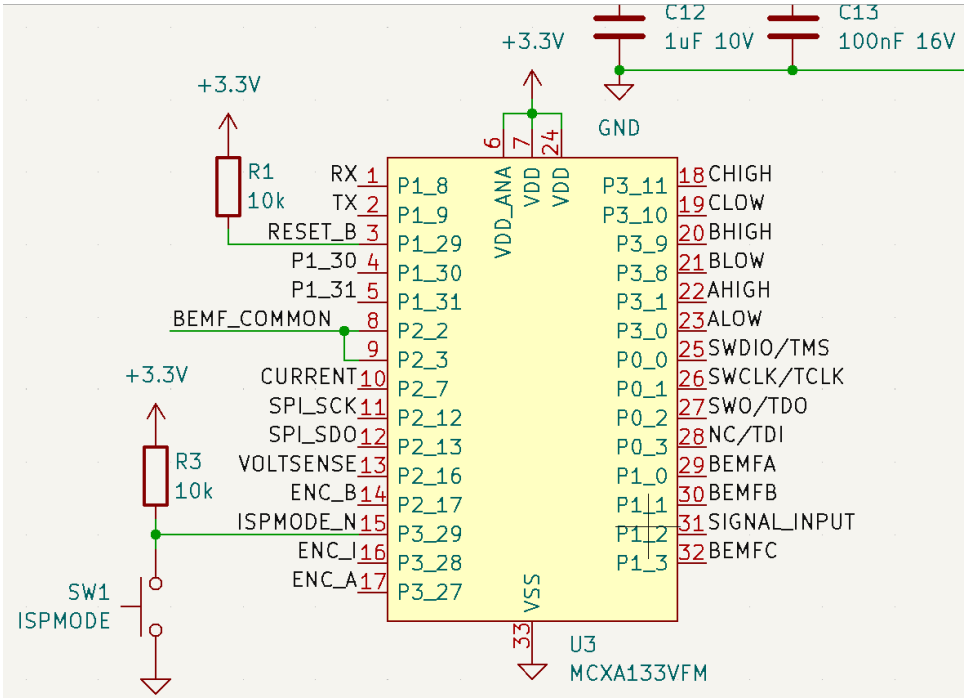


Figure 11. ISPMODE switch (not used)

The board includes an onboard switch (SW1) connected to an MCU input labeled ISPMODE. This input is not used by the current AM32 firmware and has no function during normal ESC operation. The switch may be omitted when adapting or copying the design.

Schematic, board layout, and bill of materials

The schematic, board layout, and bill of materials for the MR-ESC-AM32 board are available on the NXP product page.

Conclusion

The MR-ESC-AM32 combines a compact NXP MCXA133-based hardware platform with AM32, a widely used open-source ESC firmware for drone applications. The reference design provides a turnkey sensorless three-phase BLDC motor-control solution as implemented, including the gate driver, MOSFET power stage, BEMF sensing, and supporting interfaces. For applications with different current-handling requirements, the MOSFETs and associated power and thermal design may be resized, subject to appropriate electrical and thermal validation. The same hardware may also support additional software features or alternative motor-control firmware; however, their design, implementation, validation, and support are outside the scope of this document.

Abbreviations

Acronym	Description
ADC	Analog-to-digital converter
BEMF	Back electromotive force

Acronym	Description
BLDC	Brushless DC
DShot	Digital Shot (digital ESC control protocol)
ESC	Electronic speed controller
LDO	Low-dropout regulator
MCU	Microcontroller unit
MOSFET	Metal-oxide-semiconductor field-effect transistor
PWM	Pulse-width modulation
RC-PWM	Radio-control pulse-width modulation
RGB	Red-green-blue (addressable LED)
SWD	Serial Wire Debug
UART	Universal asynchronous receiver-transmitter
VMU	Vehicle management unit

References

- MR-ESC-AM32 User Manual — AM32 firmware setup, flashing, and operation
- AM32 firmware project (GitHub)
- AM32 wiki