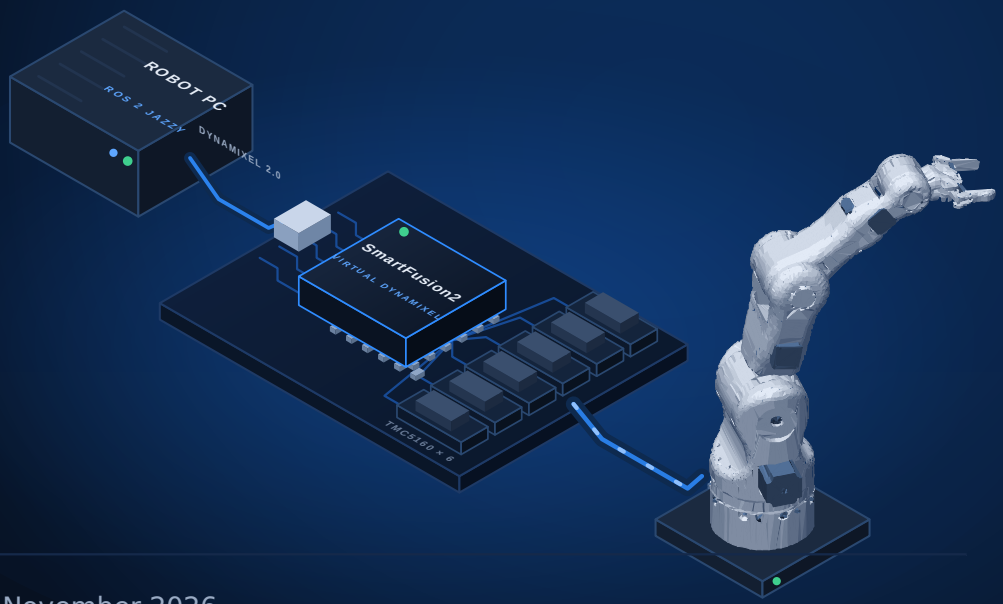
SystemFromScratch

OPEN-SOURCE ROBOTICS PROJECT

SmartManipulator-X

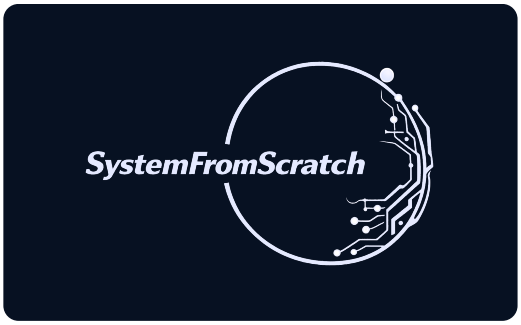
ROS 2 Robot Arm on Virtual DYNAMIXEL Servos

USER MANUAL



Version dev-c287dff · November 2026
Anton Vynohradov
systemfromscratch.com

Imprint



SmartManipulator-X User Manual
Version dev-c287dff
Edition: November 2026

Author
Anton Vynohradov
SystemFromScratch® — systemfromscratch.com
GitHub: github.com/AntonVynohradov
LinkedIn: linkedin.com/in/vynohradov
ORCID: orcid.org/0009-0005-6965-0911
E-mail: avynohradov@systemfromscratch.com

Online
Documentation: https://systemfromscratch.github.io/sfs-smartmanipulator-x-robot/
Source code: https://github.com/SystemFromScratch/sfs-smartmanipulator-x-robot

Copyright © 2026 Anton Vynohradov.

The project-owned files of SmartManipulator-X and this manual are licensed under the Apache License, Version 2.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 and in Appendix A. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SystemFromScratch® and the SystemFromScratch® logo are registered trademarks. The Apache License 2.0 grants no rights to use them; they may not be used to endorse or promote products derived from SmartManipulator-X without prior written permission.

ROBOTIS, DYNAMIXEL and OpenMANIPULATOR are trademarks of ROBOTIS Co., Ltd. SmartFusion2 and Libero are trademarks of Microchip Technology Inc. TRINAMIC and TMC5160 are trademarks of Analog Devices, Inc. ROS is a trademark of Open Source Robotics Foundation, Inc. SmartManipulator-X is not affiliated with or endorsed by these companies. All other trademarks are the property of their respective owners.

This manual is generated from the same sources as the online documentation. The information in it was checked against the version stated above; later versions may differ.

Contents

1 Overview 7
- 1.1 System components 7
- 1.2 Repositories 8
- 1.3 Documentation map 8
- 1.4 Current status 9

2 Getting started 10
- 2.1 Prerequisites 10
- 2.2 Clone 10
- 2.3 Start the Robot PC 10
- 2.4 Check it from a second host 10
- 2.5 Stop and rebuild 11
- 2.6 The same steps in VS Code 11
- 2.7 Next steps 11

3 Development 12
- 3.1 Repository layout 12
- 3.2 Robot PC image 12
- 3.3 VS Code tasks 13
- 3.4 Robot description from the CAD 13
- 3.5 Documentation 13
- 3.6 CI 14
- 3.7 Engineering rules for agents 15

4 Robot PC 16
- 4.1 Responsibility 16
- 4.2 Host and container 16
- 4.3 Simulation on one computer 16
- 4.4 Hardware shell 18
- 4.5 Network and operation 19
- 4.6 Repository ownership 19

5 BCN3D Moveo Mechanics 20
- 5.1 Mechanical source 20
- 5.2 ROS model integration 20
- 5.3 Acceptance checks 21

6 System Architecture 22
- 6.1 Project objective 22
- 6.2 System context 22
- 6.3 ARM and FPGA partition 23
- 6.4 Component ownership 23
- 6.5 Software reuse boundary 24
- 6.6 Operating assumptions 25
- 6.7 Verified state versus target 25
- 6.8 Upstream references 25

7 SmartFusion2 DYNAMIXEL Compatibility Specification 27

7.1 Scope ... 27
7.2 Sources to pin and inspect ... 27
7.3 Evidence-backed device contract ... 27
7.4 Verification record ... 28

8 Operator PC ... 29
8.1 Responsibility ... 29
8.2 Separate repository ... 29
8.3 Real robot and simulation ... 29
8.4 DDS checklist ... 30

9 Firmware: getting started ... 31
9.1 Prerequisites ... 31
9.2 Clone ... 31
9.3 Open the Dev Container ... 31
9.4 Fetch Zephyr and build ... 32
9.5 Program the board ... 32
9.6 Test the axes with the GUI (on the host) ... 32
9.7 Next steps ... 33

10 Firmware development environment ... 34
10.1 Repository layout ... 34
10.2 Pinned tool versions ... 34
10.3 Dev Container ... 35
10.4 VS Code tasks ... 35
10.5 Building ... 36
10.6 West and Zephyr ... 37
10.7 Host Python environment (.venv) ... 37
10.8 Tests ... 37
10.9 Documentation ... 38
10.10 Code style ... 38
10.11 Line endings ... 38
10.12 Docker without VS Code ... 38
10.13 Debugging ... 39
10.14 Updating tool versions ... 39
10.15 CI ... 39
10.16 Licenses and secrets ... 39
10.17 Troubleshooting ... 39

11 Firmware architecture ... 41
11.1 Firmware variants ... 41
11.2 Source layout ... 42
11.3 Layering ... 44
11.4 Start-up order ... 45
11.5 Threads ... 45
11.6 Queues ... 45
11.7 Mutexes and semaphores ... 46
11.8 Request flow ... 46
11.9 Motor detection ... 47
11.10 Virtual device summary ... 48
11.11 Mapping to the stepper axes ... 49
11.12 Configuration ... 50
11.13 Gripper servo ... 51

11.14 Known gaps ... 53

12 Motor tester GUI ... 54
12.1 Starting ... 54
12.2 Connecting ... 55
12.3 Motor detection and connection monitoring ... 55
12.4 Window layout ... 56
12.5 Testing one stepper axis ... 57
12.6 Testing the gripper servo ... 58
12.7 Messages you will see with the current firmware ... 60
12.8 Automated check ... 60
12.9 Internals ... 61

13 Hello FPGA Kit wiring ... 62
13.1 Concept ... 64
13.2 Host link (USB → PIC32 → SmartFusion2) ... 64
13.3 Wiring list (logic, Dupont m/f, ≤ 30 cm) ... 65
13.4 Axes and motors ... 66
13.5 Fixed wiring on each TMC5160 module ... 67
13.6 Power and servo ... 68
13.7 Kit header reference ... 68
13.8 Safety rules ... 69
13.9 Implications for the firmware ... 70

14 TMC5160 driver modules ... 71
14.1 The module in use: BIGTREETECH TMC5160T Pro ... 73
14.2 Motor connection: 17HS19-2504S ... 75
14.3 Design assumptions compared with the sources ... 76
14.4 Motor current settings ... 78
14.5 Motor profiles and axis assignment ... 78
14.6 Firmware interface review ... 79
14.7 Bring-up checklist ... 81
14.8 Reference documents ... 82

15 Gripper servo (MG996R) ... 83
15.1 Datasheet values ... 83
15.2 Connection to the Hello FPGA Kit ... 84
15.3 Signal from the firmware ... 85
15.4 Bring-up ... 86
15.5 Calibration ... 86
15.6 Safety ... 86

Author & Company ... 87
Author ... 87
SystemFromScratch® ... 88
Contact ... 88
Support and feedback ... 89
Trademarks and acknowledgements ... 89

Appendix A License ... 90
A.1 What the license allows ... 90
A.2 What it requires ... 90
A.3 What it does not give ... 90

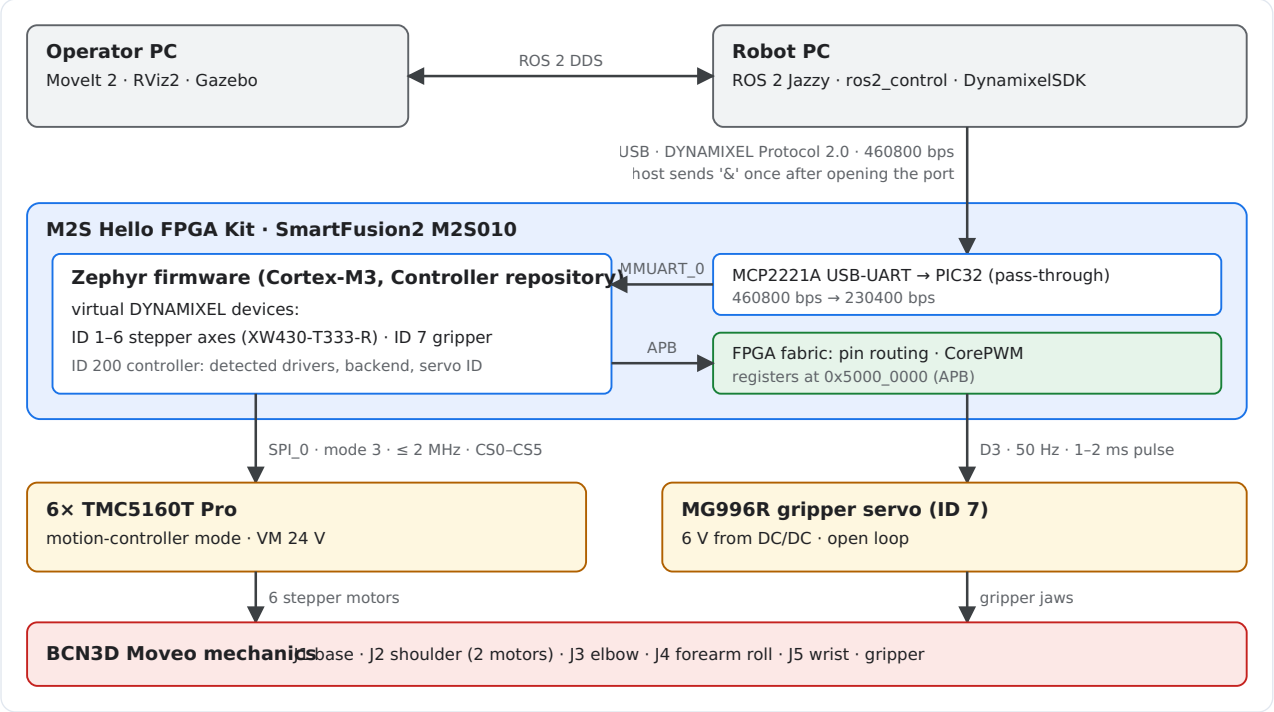
A.4 Trademarks 91
A.5 Full license text 91

CHAPTER 1

Overview

SmartManipulator-X is a robot arm project. It combines the ROS 2 control and operator-software concepts of ROBOTIS OpenMANIPULATOR-X (emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/) with the mechanical design of BCN3D Moveo (github.com/BCN3D/BCN3D-Moveo). Instead of DYNAMIXEL actuators, a Microchip SmartFusion2 controller drives the stepper motors of the Moveo joints and a hobby servo on the gripper, and presents them to the ROS 2 host as virtual DYNAMIXEL devices over Protocol 2.0. The ROBOTIS host software can then talk to the arm as if it were built from DYNAMIXEL servos.

This documentation covers the whole system: the Robot PC, the controller firmware and hardware, the shared DYNAMIXEL contract, the Operator PC, and the mechanics. It is published from the Robot PC repository.



SmartManipulator-X overview: Operator PC, Robot PC, Hello FPGA Kit with the controller firmware, stepper drivers, gripper servo and Moveo mechanics

1.1 System components

Component	Role	Platform
Operator PC	Motion planning, visualization and simulation (MoveIt 2, RViz2, Gazebo)	ROS 2, DDS client of the Robot PC
Robot PC	Robot-side ROS 2 control: `ros2_control`, the ROBOTIS hardware interface and DynamixelSDK; owns the USB link to the controller	Ubuntu 24.04, ROS 2 Jazzy in Docker

Component	Role	Platform
Controller firmware	DYNAMIXEL Protocol 2.0 device: six stepper axes (IDs 1–6), the gripper servo (ID 7) and a controller device (ID 200)	Zephyr RTOS on the SmartFusion2 Cortex-M3 (M2S Hello FPGA Kit, M2S010)
FPGA design	Pin routing of SPI and UART, CorePWM for the gripper servo	SmartFusion2 FPGA fabric (Libero)
Drives	Six BIGTREETECH TMC5160T Pro modules in motion-controller mode, MG996R gripper servo	24 V and 6 V supplies
Mechanics	Joints, links and printed parts	BCN3D Moveo

1.2 Repositories

Repository	Content
sfs-smartmanipulator-x-robot (github.com/SystemFromScratch/sfs-smartmanipulator-x-robot)	Robot PC Docker image and ROS 2 workspace, the robot description generated from the Moveo CAD, diagnostics, and this documentation
sfs-smartmanipulator-x-operator (github.com/SystemFromScratch/sfs-smartmanipulator-x-operator)	Operator PC: planning, visualization and simulation
sfs-smartmanipulator-x-skills (github.com/SystemFromScratch/sfs-smartmanipulator-x-skills)	Engineering rules for coding agents, shared by all repositories of the project
SmartManipulator-X-Controller	Zephyr firmware for the SmartFusion2 and the host-side DYNAMIXEL motor tester GUI
SmartManipulator-X-Libero	FPGA design and the hardware contract for the firmware (`docs/firmware_interface.md`)

1.3 Documentation map

Section	Pages
Guide	Getting started: run the Robot PC in simulation and check it from a second host. Development: Docker, VS Code tasks, robot description, documentation, CI.
Robot PC	Robot PC: ROS 2 control, Docker, network, USB device. BCN3D Moveo mechanics: the mechanical source and the generated robot model.
System	System architecture: component boundaries, control path, ARM/FPGA partition, verified state. DYNAMIXEL compatibility: the shared host/device contract. Operator PC: planning, simulation, DDS.
Firmware	Getting started and development environment of the controller firmware. Firmware architecture: source layout, start-up order, threads, queues, data flow, configuration. Motor tester GUI: testing every axis and the gripper over DYNAMIXEL Protocol 2.0.
Hardware	Hello FPGA Kit wiring: the kit, the six TMC5160 modules, the servo and power. TMC5160 driver modules: pinout, motor connection, firmware interface review, bring-up checklist. Gripper servo (MG996R): supply, signal, bring-up and calibration.

The whole documentation is also available as one PDF user manual (the PDF button in the header of the website).

1.4 Current status

- **Robot PC:** ROS 2 Jazzy Docker image with the selected ROBOTIS packages and the project packages. In simulation (mock hardware) it runs `robot_state_publisher`, `ros2_control` and the arm and gripper controllers on a Docker network that stands in for the Ethernet link; a smoke test from a second container checks discovery, the controllers and a trajectory. The robot description is generated from the BCN3D Moveo CAD; its joint limits are placeholders until measured on the arm. The real-hardware `ros2_control` path is not configured yet.
- **Controller firmware:** builds in two variants, the **target** variant (TMC5160 over SPI) and the **simulation** variant (simulated axes on the real board, for protocol tests). Both implement a functional subset of a virtual XW430-T333-R (PING, READ, WRITE, REG_WRITE, ACTION, SYNC_READ, SYNC_WRITE). The simulation firmware passes the automated conformance check (`firmware_check.py`, 49/49, including the gripper servo) when it runs as `native_sim` on a PC.
- **Hardware (2026-10-11):** the host link (460800 bps PC ↔ PIC32, 230400 bps PIC32 ↔ SmartFusion2) works on the kit. On axis A0 (BTT TMC5160T Pro, 17HS19-2504S) SPI detection, the register mapping, telemetry, Current Limit and the ramp generator are verified, but **the shaft does not turn yet**: the module reads `SD_MODE=1` (STEP/DIR mode) instead of the motion-controller mode, see finding 10 in Firmware interface review (p. 80). Motor currents, ramps and the motor-to-axis assignment are described in Motor profiles and axis assignment (p. 78).
- **Gripper servo:** a virtual DYNAMIXEL device on ID 7 driven by CorePWM in the FPGA fabric. On the kit it follows Goal Position; its travel is not calibrated yet.
- **Open items for the whole arm:** verification of the upstream Jazzy sources against the device behavior (model numbers, registers), joint-to-motor mapping and gear ratios on the real arm, shoulder pairing (IDs 2 and 3), joint limits, and safety (emergency stop). The verified state versus target (p. 25) table tracks them.

CHAPTER 2

Getting started

This guide takes you from a fresh clone of the Robot PC repository to a running Robot PC in simulation, checked from a second ROS 2 host. No SmartFusion2 board and no arm are needed: `ros2_control` runs with mock hardware, where every commanded joint position becomes the joint state at once.

2.1 Prerequisites

Where	What
Simulation (any computer)	Git, Docker Engine or Docker Desktop (Windows with the WSL 2 backend works) and Docker Compose v2
Real hardware	An Ubuntu 24.04 Robot PC (amd64 or arm64) and the controller board enumerated as USB CDC, typically `/dev/ttyACM0`. Docker Desktop on Windows does not expose a Linux `/dev/ttyACM0`
Optional	VS Code, which runs every step below as a task

ROS 2 is not installed on the host: it comes with the Robot PC image (`ros:jazzy-ros-base`).

2.2 Clone

```
git clone https://github.com/SystemFromScratch/sfs-smartmanipulator-x-robot.git
cd sfs-smartmanipulator-x-robot
```

2.3 Start the Robot PC

From the repository root:

```
docker compose -f docker/docker-compose.yml up --build robot_pc
```

The first run builds the image, which takes several minutes. The Compose file creates the Docker network `smx_net` (`10.77.0.0/24`), which stands in for the Ethernet link between the Robot PC and the Operator PC, and starts the Robot PC container at `10.77.0.10`. It runs `robot_state_publisher`, `ros2_control_node` with mock hardware, `joint_state_broadcaster`, `arm_controller` and `gripper_controller`. The logs stay in the terminal.

2.4 Check it from a second host

In a second terminal, start the `probe` container on the same network and run the smoke test:

```
docker compose -f docker/docker-compose.yml run --rm probe smx_smoke_test.sh
```

The test passes when it receives `/joint_states`, finds `joint_state_broadcaster`, `arm_controller` and `gripper_controller` active, and `arm_controller` executes a short trajectory. For your own `ros2` commands, open a shell on the probe instead:

```
docker compose -f docker/docker-compose.yml run --rm probe bash
ros2 topic list
```

Any other container on `smx_net` with the same `ROS_DOMAIN_ID` (default 42) sees the Robot PC through ROS 2 DDS like a second machine on the LAN. To change the domain, the subnet or the image tag, copy `docker/env.example` to `docker/.env`.

2.5 Stop and rebuild

Stop with **Ctrl+C** in the first terminal, or remove the containers and the network:

```
docker compose -f docker/docker-compose.yml down
```

The workspace is built into the image, not mounted. After changing project sources or `ros2_robot.repos`, start with `up --build` again.

2.6 The same steps in VS Code

Ctrl+Shift+B opens the build task picker:

Task	Does
Robot PC: Start (mock hardware)	`up --build robot_pc`; the logs stay in a dedicated terminal
Robot PC: Stop	`down`: removes the containers and `smx_net`
Robot PC: Rebuild and restart	Stop, then `up --build --force-recreate robot_pc`
Robot PC: Build image	Builds the image without starting it
Robot PC: Rebuild image (no cache)	Full rebuild, pulls the base image again
Docs: Build PDF manual	Writes the PDF user manual (see Development)

The smoke test is under **Tasks: Run Test Task**, the probe shell and the documentation preview under **Tasks: Run Task**.

2.7 Next steps

- Robot PC (p. 16): the joint contract, how the Operator PC joins `smx_net`, and the hardware shell with the USB device.
- Development (p. 12): repository layout, the robot description generated from the CAD, documentation and CI.
- System architecture (p. 22): how the Robot PC, the controller firmware and the Operator PC fit together.

CHAPTER 3

Development

This page describes how the Robot PC repository is organized and built: the Docker image, the ROS 2 workspace, the robot description generated from the CAD, the documentation and the CI workflows. The controller firmware has its own development environment (p. 34).

3.1 Repository layout

```
.
├── docker/
│   ├── Dockerfile                    Robot PC image (ros:jazzy-ros-base + ROBOTIS
packages + workspace)
│   ├── docker-compose.yml            robot_pc and probe services, network smx_net
│   ├── docker-compose.hw.yml         USB device pass-through (Linux hosts)
│   ├── entrypoint.sh                 sources ROS 2 and the built workspace
│   ├── env.example                   template for docker/.env
│   └── dds/fastdds_udp.xml           Fast DDS profile: UDPv4 only
├── ros2_ws/src/smartmanipulator_x/
│   ├── smartmanipulator_x_bringup/   robot.launch.py
│   ├── smartmanipulator_x_config/    controllers.yaml, hardware.yaml
│   ├── smartmanipulator_x_description/  URDF/Xacro and meshes generated from the Moveo
CAD
│   └── smartmanipulator_x_tests/     test area
├── ros2_ws/src/robotis/              checkout target of ros2_robot.repos (in the image)
├── ros2_robot.repos                  upstream sources built into the Robot PC image
├── ros2.repos                        broader workspace manifest
├── tools/
│   ├── cad/                          STEP → URDF pipeline for the robot description
│   ├── diagnostics/smx_smoke_test.sh end-to-end check over DDS
│   └── docs/                         PDF manual builder, diagrams, MkDocs hook
├── docs/, overrides/, mkdocs.yml     this documentation
├── requirements-docs.txt             documentation tools (.venv)
└── .vscode/tasks.json                Robot PC and documentation tasks
```

Generated directories (`ros2_ws/build/` , `ros2_ws/install/` , `ros2_ws/log/` , `site/` , `build/` , `docs/downloads/` , `.venv/`) are ignored.

3.2 Robot PC image

The image is based on `ros:jazzy-ros-base` . It imports the DYNAMIXEL SDK and interface packages through `ros2_robot.repos` , builds them together with the project packages, and contains ROS control, USB diagnostics and build tools. MoveIt 2, RViz2, Gazebo and the OpenMANIPULATOR GUI and bringup packages are deliberately excluded: they belong to the Operator PC.

The workspace is baked into the image, not bind-mounted, so rebuild after changing project sources or `ros2_robot.repos` :

```
docker compose -f docker/docker-compose.yml build robot_pc
docker compose -f docker/docker-compose.yml build --no-cache --pull robot_pc   # from
scratch
```

The `jazzy` refs in `ros2_robot.repos` are branch names and can move; pin immutable revisions before relying on a build for verification.

3.3 VS Code tasks

`.vscode/tasks.json` runs Docker Compose from the repository root. **Ctrl+Shift+B** opens a picker with every task of the build group; none is the default, so the picker always appears.

Task	Group	Command
Robot PC: Start (mock hardware)	build	`docker compose -f docker/docker-compose.yml up --build robot_pc`
Robot PC: Stop	build	`docker compose -f docker/docker-compose.yml down`
Robot PC: Rebuild and restart	build	Stop, then `up --build --force-recreate robot_pc`
Robot PC: Build image	build	`build robot_pc`
Robot PC: Rebuild image (no cache)	build	`build --no-cache --pull robot_pc`
Docs: Build PDF manual	build	`python tools/docs/build_manual.py` in `.venv`
Robot PC: Smoke test (probe)	test	`run --rm probe smx_smoke_test.sh`
Robot PC: Probe shell	–	`run --rm probe bash`
Docs: Live preview	–	`mkdocs serve` in `.venv`

3.4 Robot description from the CAD

`smartmanipulator_x_description` is generated from a STEP export of the BCN3D Moveo assembly by the scripts in `tools/cad/` (Python 3 with numpy, scipy and matplotlib). The STEP itself is not in the repository (about 200 MB):

```
tools/cad/run_all.sh BCN3D_Moveo_assembly.stp ../BCN3D-Moveo /tmp/smx_cad
```

The steps, the measured dimensions and the conventions of the model are in `tools/cad/README.md` and on the BCN3D Moveo mechanics (p. 20) page.

3.5 Documentation

The documentation is the Markdown in `docs/`. MkDocs Material turns it into the website, and `tools/docs/build_manual.py` turns the same pages into the printable A4 user manual (cover, imprint, contents with page numbers, numbered chapters, running headers, PDF bookmarks), rendered with WeasyPrint. Both follow the SystemFromScratch® documentation style; the landing page is configured in `mkdocs.yml → extra.landing` and the manual in `mkdocs.yml → extra.manual`.

Set up the tools once in the repository `.venv`. WeasyPrint needs the Pango library: on Windows from MSYS2 (`pacman -S mingw-w64-x86_64-pango`; the builder finds `C:\msys64\mingw64\bin` by itself), on Ubuntu `apt install libpango-1.0-0 libpangoft2-1.0-0`.

```
# Windows (PowerShell, repository root)
py -m venv .venv
.\.venv\Scripts\python.exe -m pip install -r requirements-docs.txt
```

```
# Linux (repository root)
python3 -m venv .venv
.venv/bin/python -m pip install -r requirements-docs.txt
```

Command	Result
`python -m mkdocs serve`	Live preview at http://127.0.0.1:8000
`python -m mkdocs build`	Static website in `site/` (strict: a broken link fails the build)
`python tools/docs/build_manual.py`	`build/manual/SmartManipulator-X_User_Manual_v<version>.pdf`; the version is the latest `vX.Y.Z` tag
`python tools/docs/build_manual.py --output docs/downloads/SmartManipulator-X_User_Manual.pdf`	The file the website links to (CI does this before deploying)
`python tools/docs/diagrams.py`	Regenerates the hero illustration `docs/assets/images/hero-3d.svg`; the arm is drawn from the robot description meshes and URDF by `tools/docs/hero_robot.py`, so run it again after the description changes
`python tools/docs/render_mermaid.py`	Renders `docs/**/diagrams/*.mmd` to `docs/**/images/*.svg` (needs Playwright and a network connection)

Writing rules that keep the website and the PDF correct:

- Start every page with exactly one `# Title`; chapter and section numbers are added in the PDF, so do not number headings by hand.
- Add a new page to `nav` in `mkdocs.yml`, otherwise it is missing from the site and the PDF.
- An image on a line of its own, `![Caption](images/file.svg)`, becomes a figure with its alt text as the caption (`tools/docs/hooks.py`).
- Link to other pages with relative `.md` paths; strict mode checks them, and the PDF turns them into page references.
- Diagrams are SVG files. Mermaid sources live in `diagrams/*.mmd` next to the page and are rendered with `render_mermaid.py`, because the PDF renderer runs no JavaScript.

3.6 CI

Workflow	Trigger	Does
`docker-publish.yml`	Push to `main` or a `v*` tag touching the image sources; manual	Builds the Robot PC image and publishes it to `ghcr.io/systemfromscratch/sfs-smartmanipulator-x-robot`
`documentation.yml`	Push to `main`; manual	Builds the PDF manual into `docs/downloads/`, then `mkdocs gh-deploy` publishes the website with the PDF to GitHub Pages

3.7 Engineering rules for agents

The architecture and the Robot PC, Operator PC and SmartFusion2 rules for coding agents are skills in the shared sfs-smartmanipulator-x-skills (github.com/SystemFromScratch/sfs-smartmanipulator-x-skills) repository, used by every repository of the project.

CHAPTER 4

Robot PC

4.1 Responsibility

The Robot PC owns the real-time-facing ROS 2 control path and is the only computer that connects directly to the SmartFusion2 USB CDC device:

```
ros2_control
  -> ROBOTIS dynamixel_hardware_interface
  -> DynamixelSDK
  -> USB CDC (commonly /dev/ttyACM0)
  -> SmartFusion2
  -> stepper motor control
```

The stock ROBOTIS hardware interface is the preferred starting point, but it must not be assumed compatible until the exact protocol and control-table accesses have been checked against current Jazzy source. See the compatibility specification (p. 27).

4.2 Host and container

The target Robot PC host is Ubuntu 24.04 with Docker Engine and Docker Compose. The paths below (`docker/`, `ros2_ws/`, `ros2_robot.repos`) are in this repository. The current `docker/` profile is based on `ros:jazzy-ros-base`. It imports the DYNAMIXEL SDK/interface dependencies through `ros2_robot.repos` and builds the project-owned packages; `docker-compose.hw.yml` maps `/dev/ttyACM0` into the container with the `dialout` group.

MoveIt 2, RViz2, Gazebo, and the Operator PC application are intentionally excluded from this image. The Operator PC owns those dependencies and connects over ROS 2 DDS.

4.3 Simulation on one computer

Until the hardware path is verified, the Robot PC runs with `mock_components/GenericSystem`: the commanded joint position becomes the state at once, so the whole ROS side (DDS, controllers, MoveIt on the Operator PC) can be developed without the SmartFusion2. Everything runs in containers on one computer; a Docker network replaces the Ethernet cable:

```
one computer (Docker)
┌──────────────────────────────── smx_net 10.77.0.0/24 ────────────────────────────────┐
│                                                                                      │
│ robot_pc 10.77.0.10          probe (dynamic IP)      Operator PC container           │
│ robot_state_publisher        ros2 CLI                MoveIt 2, RViz2                 │
│ ros2_control_node (mock)     smx_smoke_test.sh       (Operator PC repo)              │
│ joint_state_broadcaster                                                              │
│ arm_controller                                                                       │
│ gripper_controller                                                                   │
└──────────────────────── ROS 2 DDS, UDPv4, ROS_DOMAIN_ID=42 ──────────────────────────┘
```

From the repository root (the getting started (p. 10) guide walks through it):

```
docker compose -f docker/docker-compose.yml up --build robot_pc                # Robot PC
docker compose -f docker/docker-compose.yml run --rm probe smx_smoke_test.sh    # second
host
```

File	Role
`docker/docker-compose.yml`	`robot_pc` service (mock hardware by default), `probe` service (profile `tools`), network `smx_net`
`docker/docker-compose.hw.yml`	Override that passes the controller's USB device (Linux hosts)
`docker/env.example`	Template for `docker/.env`: `ROS_DOMAIN_ID`, subnet, Robot PC address, image tag, serial device
`docker/dds/fastdds_udp.xml`	Fast DDS profile: UDPv4 only, no shared memory between containers
`smartmanipulator_x_bringup/launch/robot.launch.py`	`robot_state_publisher`, `ros2_control_node`, controller spawner
`smartmanipulator_x_config/config/controllers.yaml`	`joint_state_broadcaster`, `arm_controller` (J1–J5), `gripper_controller`, 100 Hz
`smartmanipulator_x_description/`	Moveo model generated from the BCN3D CAD (`tools/cad/`) and its `ros2_control` block
`tools/diagnostics/smx_smoke_test.sh`	Discovery, controller state, and trajectory check from another host

Joint contract

The Operator PC, the simulation, and the real hardware must use the same joint names. These are defined in `smartmanipulator_x_description`:

Joint	Moveo axis	DYNAMIXEL ID (SmartFusion2)	Controller
`j1_base`	J1 base	1	`arm_controller`
`j2_shoulder`	J2 shoulder	2 and 3 (two mirrored motors)	`arm_controller`
`j3_elbow`	J3 elbow	4	`arm_controller`
`j4_forearm_roll`	J4 forearm roll	5	`arm_controller`
`j5_wrist`	J5 wrist	6	`arm_controller`
`gripper`	gripper servo	7	`gripper_controller`

Lengths, joint axes, and meshes come from the BCN3D Moveo CAD (BCN3D Moveo mechanics (p. 20)); joint limits are placeholders until measured on the arm. The gripper also has mimic joints (`gripper_idler_gear`, `gripper_pivot_*`, `gripper_finger_*`) that follow `gripper`; they are not in `ros2_control`. On the real robot, `dynamixel_hardware_interface` can map the two shoulder motors onto one joint with its `transmission_to_joint_matrix` and `joint_to_transmission_matrix` parameters; this is still to be configured and verified.

Operator PC on the same network

The Operator PC container joins the network by name. In its Compose file:

```
services:
  operator_pc:
    environment:
      ROS_DOMAIN_ID: "42"
      RMW_IMPLEMENTATION: rmw_fastrtps_cpp
      FASTDDS_DEFAULT_PROFILES_FILE: /etc/smx/fastdds_udp.xml  # copy of docker/dds/
fastdds_udp.xml
    networks: [smx_net]
networks:
  smx_net:
    external: true
```

Start the Robot PC first; it creates `smx_net`. On Windows, run the Operator PC container from a WSL 2 distribution so RViz2 can use WSLg (mount `/tmp/.X11-unix` and `/mnt/wslg`, pass `DISPLAY`, `WAYLAND_DISPLAY`, and `XDG_RUNTIME_DIR`). ROS 2 installed natively on Windows does not discover containers on `smx_net`: Docker Desktop does not route multicast between the host and its container networks.

Moving to another Robot PC

The image has no host-specific state. On a Raspberry Pi CM4 or a Jetson (both arm64, Ubuntu 24.04), build it there or pull a multi-architecture image, and replace `smx_net` by the real Ethernet: drop the `networks` entries and use `network_mode: host`. `ROS_DOMAIN_ID` and the Fast DDS profile stay the same.

4.4 Hardware shell

Build and start an interactive Robot PC shell with the USB device from the repository root:

```
docker compose -f docker/docker-compose.yml -f docker/docker-compose.hw.yml run --rm
robot_pc bash
```

Before starting, connect the board and confirm the device path on the Linux host. The kit appears through its MCP2221A USB-UART; the host must open it at 460800 bps and send `&` once before the first DYNAMIXEL packet (see Host link (p. 64)):

```
lsusb
ls -l /dev/ttyACM0
```

If the board enumerates under another `/dev/ttyACM*` path, set `SMX_SERIAL_DEVICE` in `docker/.env` (template: `docker/env.example`). Ensure the host user and container have appropriate device permissions. Windows does not expose a Linux `/dev/ttyACM0` device directly to this container configuration.

Inside the container, inspect the available packages with:

```
ros2 pkg list | grep -E '^(dynamixel_sdk|dynamixel_interfaces|
dynamixel_hardware_interface|smartmanipulator_x_)'
```

`robot.launch.py` refuses `use_mock_hardware:=false` until the real-hardware `ros2_control` configuration is verified. No physical bringup or SmartFusion2 hardware operation is documented as validated.

4.5 Network and operation

The Robot PC communicates with the Operator PC using ROS 2 DDS over the configured network. Configure the same `ROS_DOMAIN_ID` on both computers and validate DDS discovery, selected network interfaces, multicast/unicast policy, and firewall behavior on the actual network. Do not use SSH as the ROS transport.

Do not enable motor motion until the actuator mapping, motor-driver configuration, limits, homing behavior, and an appropriate stop/safety procedure have been reviewed on the assembled robot. Test incrementally: upstream ROS without hardware, USB/device communication, one virtual actuator, one physical axis, then the complete mechanism.

4.6 Repository ownership

- Robot PC Docker and bringup: `docker/` and `ros2_ws/src/smartmanipulator_x/smartmanipulator_x_bringup/`.
- Project controller and hardware configuration: `ros2_ws/src/smartmanipulator_x/smartmanipulator_x_config/`.
- Canonical robot description location: `ros2_ws/src/smartmanipulator_x/smartmanipulator_x_description/`.
- Upstream dependencies: separate source checkouts selected by `ros2_robot.repos`; pin immutable revisions for reproducible builds before production use.

CHAPTER 5

BCN3D Moveo Mechanics

5.1 Mechanical source

The target arm mechanics are based on the BCN3D Moveo project (github.com/BCN3D/BCN3D-Moveo), not on the OpenMANIPULATOR-X geometry. The upstream repository provides CAD files, STL files, assembly/user documentation, a bill of materials, and original firmware. Its README describes an educational, additively manufactured arm controlled by Arduino-based electronics.

This project replaces/adapts the control electronics; it does not claim that the original Moveo firmware or electronics are part of the SmartFusion2 control path. Refer to the upstream repository and its license for the authoritative mechanical files, assembly details, and attribution requirements.

5.2 ROS model integration

The intended home for the project-owned ROS model is:

```
ros2_ws/src/smartmanipulator_x/smartmanipulator_x_description/
```

The model in that package is generated from the BCN3D Moveo CAD: a STEP export of `BCN3D Moveo assembly.SLDASM` (upstream `CAD files/`, MIT license; the license text is in `meshes/LICENSE-BCN3D-Moveo`). The scripts in `tools/cad/` read the STEP, assign each part to a link, take the joint axes from the axles in the CAD, and write `urdf/smartmanipulator_x.urdf.xacro`, the decimated visual meshes, and convex-hull collision meshes. See `tools/cad/README.md` for the steps; `meshes/geometry.json` records the derived values.

Quantity	From the CAD
Base (table plate top) to J2 axis	231.64 mm
J2 to J3	221.12 mm
J3 to J5, along the J4 roll axis	223.00 mm (J4 joint frame at 96.09 mm, where the wrist part begins)
Gripper	MG996R servo gear and idler gear with 40 mm arms, pivot arms of the same length: a parallelogram; 80 mm finger gap when open

Conventions of the model:

- Zero pose: arm straight up, gripper fully open. The SmartFusion2 start pose (arm placed by hand before power-up) must match it, or the zero offsets must be set in the controller.
- Positive J2, J3 and J5 bend the arm forward (+x, away from the electronics box of the original Moveo); positive J1 and J4 turn counter-clockwise seen from the tip of their axis.
- `gripper` is the servo gear angle, 0 (open) to 1.45 rad (closed). The idler gear, pivot arms and fingers are mimic joints.

- The original electronics box is available with the xacro argument `electronics_box:=true`; it is off by default because this build uses other electronics.

Still to verify on the assembled arm: joint limits (the URDF values are placeholders), the direction of each motor against the conventions above, and any printed part changed for the NEMA 23 shoulder or the geared elbow motor.

When integrating the model:

- Use BCN3D Moveo geometry and assembly as the mechanical source of truth.
- Record the source revision and any project-specific changes to CAD/STL-derived geometry.
- Establish the base frame, link frames, joint axes, zero positions, and units from the source design and assembled robot.
- Verify joint ranges and collision geometry against the physical arm; do not inherit OpenMANIPULATOR-X limits or joint names.
- Define and document each Moveo joint-to-stepper mapping, transmission, microstepping, homing/reference method, and position scaling only after the hardware is selected and measured.
- Keep MoveIt planning groups, `ros2_control` joints, Gazebo joints, and physical motor mapping consistent.
- Preserve upstream license and attribution notices for imported or adapted assets. Do not assume that the project repository's license changes third-party asset licensing.

The electronics are selected in the controller project: six BTT TMC5160T Pro drivers and the stepper motors per joint listed in Axes and motors (p. 66) (J1 base, J2 shoulder with two motors, J3 elbow, J4 forearm roll, J5 wrist), plus an MG996R gripper servo; the joint-to-motor assignment is in `include/motor/motor_axes.h` of the controller firmware. Joint naming in the ROS model, dimensions, limits, gear ratios and the sensing approach remain to be verified for the selected mechanical build. They are deliberately not inferred here from the project name.

5.3 Acceptance checks

Before a model is treated as the canonical robot description, validate that it parses in URDF/Xacro tooling, has a connected link tree and correct joint types/axes, and produces a useful RViz visualization. Compare each joint range and frame placement with the assembled Moveo. Then check that the same joint contract can be loaded by both the physical `ros2_control` configuration and the selected Gazebo integration.

CHAPTER 6

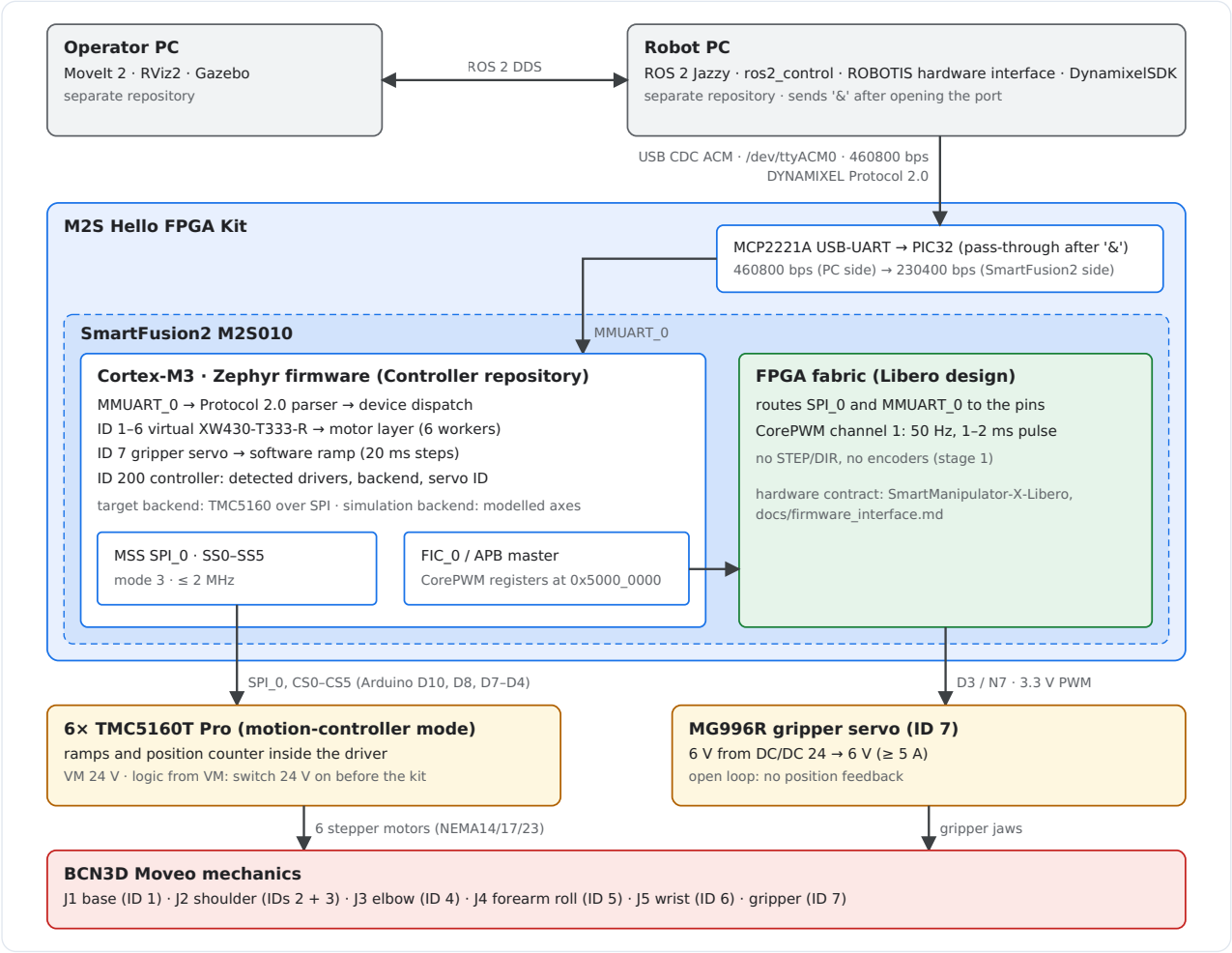
System Architecture

6.1 Project objective

The target is to use the ROS 2 control and operator-software concepts of ROBOTIS OpenMANIPULATOR-X (docs.robotis.com/docs/systems/openmanipulator_x/overview/) with the mechanical design of BCN3D Moveo (github.com/BCN3D/BCN3D-Moveo). A Microchip SmartFusion2 (M2S Hello FPGA Kit) replaces the DYNAMIXEL actuators: it drives stepper motors on the Moveo mechanism and a hobby servo on the gripper, and presents them to the host as virtual DYNAMIXEL devices.

These sources have different roles. OpenMANIPULATOR-X is the reference for the ROS/DYNAMIXEL software stack; BCN3D Moveo is the mechanical reference. The OpenMANIPULATOR-X robot geometry, joints, actuator assumptions, and hardware configuration must not be copied as if they described Moveo.

6.2 System context



SmartManipulator-X system: Operator PC, Robot PC, Hello FPGA Kit, TMC5160 axes and gripper servo

The SmartFusion2 is a motor controller implementing device behavior; it is not merely a USB-to-DYNAMIXEL converter. Details of each link:

Link	Implementation	Reference
Robot PC ↔ kit	USB Mini-B → MCP2221A (CDC ACM) → PIC32 → MMUART_0. PC side 460800 bps, firmware side 230400 bps; the PIC32 forwards only after the host has sent `&`	Host link
Firmware ↔ stepper drivers	MSS SPI_0, routed through the fabric to the Arduino header, six chip selects; TMC5160 in motion-controller mode (no STEP/DIR)	Hello FPGA Kit wiring, TMC5160 driver modules
Firmware ↔ gripper servo	CorePWM channel 1 in the fabric, APB registers at `0x5000_0000`, output on D3 (N7)	Gripper servo (MG996R)
Power	24 V PSU for the TMC5160 modules (VM), DC/DC 24 → 6 V for the servo, kit from USB; one GND star point at the PSU minus	Power and servo

6.3 ARM and FPGA partition

As built (Libero stage 2, firmware on `main`):

Function	Where	Status
Host link, Protocol 2.0, Control Tables, unit conversion	Cortex-M3 (Zephyr)	Implemented; host link verified on the kit
Axis control: ramps, position counter, current	TMC5160 (motion-controller mode), configured over SPI by the Cortex-M3	Implemented; detection, registers and ramp generator verified on one module (A0); the shaft does not turn yet, the module is in STEP/DIR mode (`SD_MODE=1`)
Gripper servo ramp (20 ms steps)	Cortex-M3 (`k_timer`)	Implemented; works on the kit
Gripper servo pulse generation	FPGA fabric (CorePWM)	In the Libero design; the servo follows Goal Position on the kit (travel not calibrated)
Pin routing of SPI_0 and MMUART_0	FPGA fabric	Verified on the kit
STEP/DIR generation, encoders, high-rate loops	–	Not used: the drivers generate the ramps; no encoders in stage 1

6.4 Component ownership

Component	Responsibility	Repository boundary
SmartFusion2 firmware	DYNAMIXEL Protocol 2.0 device behavior, Control Tables, TMC5160 configuration and commands, gripper servo	Controller repository (`SmartManipulator-X-Controller`: `apps/SmartManipulator_Controller`, plus the host motor tester `tools/dxl_test_gui`)

Component	Responsibility	Repository boundary
SmartFusion2 FPGA design	Pin routing, CorePWM, clocks; the hardware contract for the firmware	Separate Libero repository (`SmartManipulator-X-Libero`, `docs/firmware_interface.md`)
Robot PC	ROS 2 Jazzy, `ros2_control`, verified ROBOTIS hardware path, USB access, robot bringup, canonical robot description	Robot PC repository (`sfs-smartmanipulator-x-robot`, which also publishes this documentation)
Operator PC	MoveIt 2, RViz2, Gazebo, operator workflows, ROS 2 DDS client	Operator PC repository (`sfs-smartmanipulator-x-operator`)
BCN3D Moveo	Mechanical links, joints, printed parts, assembly and mechanical limits	Upstream mechanical source; the project model (`smartmanipulator_x_description`) is generated from its CAD in the Robot PC repository
ROBOTIS packages	Reusable SDK, hardware interface, messages, and upstream OpenMANIPULATOR software	Separate upstream checkouts managed by the Robot PC manifests

6.5 Software reuse boundary

The physical control path is:

```
MoveIt 2
  -> ros2_control
  -> ROBOTIS dynamixel_hardware_interface
  -> DynamixelSDK
  -> USB (MCP2221A, /dev/ttyACM0) -> PIC32 -> MMUART_0
  -> SmartFusion2 virtual DYNAMIXEL devices (IDs 1-6 axes, ID 7 gripper)
  -> TMC5160 over SPI -> stepper motors
  -> CorePWM -> gripper servo
```

Reuse the existing ROBOTIS hardware interface if source analysis confirms that its model configuration, protocol operations, control-table accesses, command/state interfaces, and timing requirements can be supported. Do not add a project-specific `smartfusion2_hardware_interface` pre-emptively. Implement only the device behavior needed by the verified host stack, not a presumed complete XM430 clone. Two host requirements come from the kit, not from DYNAMIXEL: the host must send `&` once after opening the port, and a `0x00` after any packet that ends with `0x25` (see Host link (p. 64)).

The devices report project model numbers where they differ from a real DYNAMIXEL: the axes report XW430-T333-R (1270), the gripper servo `0x5356`, the controller `0x5358`. Whether the ROBOTIS stack accepts these must be checked before it is configured (see the compatibility specification (p. 27)).

Reusing the OpenMANIPULATOR-X ROS application does not make its robot model or joint/controller configuration valid for Moveo. MoveIt planning groups, joint names, limits, kinematics, transmission assumptions, ros2_control interfaces, and controller configuration must agree with a verified Moveo description and the physical stepper setup.

The Robot PC image is intentionally separate from the Operator PC runtime. MoveIt, RViz2, and Gazebo do not belong in the Robot PC image. In simulation, route `ros2_control` to the selected Gazebo integration; on the real robot, route it to the DYNAMIXEL/SmartFusion2 path. Keep the robot model and joint semantics consistent across both modes.

6.6 Operating assumptions

- The Robot PC target is Ubuntu 24.04 with ROS 2 Jazzy; the Robot PC profile uses Docker.
- The Operator PC runs ROS 2 Jazzy and reaches the Robot PC using DDS over the configured network.
- The kit enumerates through its MCP2221A as a USB CDC ACM device, commonly `/dev/ttyACM0`. Only the Robot PC owns this connection; the Operator PC does not require direct USB access.
- DDS domain ID, interface selection, discovery, and firewall policy must be configured consistently across hosts. SSH is for administration, not the ROS data path.
- Positioning is open loop: no encoders, no limit switches, no homing. The arm is placed in the start pose by hand before power-up; the servo position is unknown until the first command.
- Power-up order: 24 V (TMC5160 logic runs from VM) and the servo DC/DC before the kit.

6.7 Verified state versus target

Area	Current state	Target or open verification
SmartFusion2 firmware	Implemented in the controller repository: 6 axes, gripper servo, controller device. Passes `firmware_check.py` 49/49 on `native_sim`. On the kit: host link, TMC5160 detection, register mapping and ramp generator on A0, and the gripper servo work	Motion of the axes (A0 module in STEP/DIR mode, `SD_MODE=1`); servo travel calibration; gear ratios, joint limits
FPGA design	Libero stage 2: pin routing and CorePWM; the CorePWM output drives the servo on the kit	Pulse widths not measured with an oscilloscope yet
Robot description	Generated from the BCN3D Moveo CAD (`tools/cad/`): link lengths, joint axes, visual and collision meshes, gripper linkage	Measure joint limits on the arm; verify motor directions against the model conventions
Robot PC	ROS 2 Jazzy image; runs with mock hardware on `smx_net`, smoke test passes from a second container; real-hardware `ros2_control` path not configured	Verify upstream Jazzy sources against the device behavior (model numbers, registers, `&` and `0x25` handling)
Operator PC	Separate repository	Consume the canonical Moveo description and validate DDS
Mechanical/electrical integration	Wiring Rev. D (draft): six TMC5160T Pro, MG996R gripper servo, 24 V and 6 V supplies	Joint-to-motor mapping on the real arm, shoulder pairing (IDs 2 and 3), limits, safety (emergency stop)

6.8 Upstream references

- ROBOTIS OpenMANIPULATOR-X overview (docs.robotis.com/docs/systems/openmanipulator_x/overview/)
- BCN3D Moveo source repository (github.com/BCN3D/BCN3D-Moveo)

- ROBOTIS OpenMANIPULATOR ROS 2 source (github.com/ROBOTIS-GIT/open_manipulator)
- DynamixelSDK source (github.com/ROBOTIS-GIT/DynamixelSDK)
- DYNAMIXEL hardware interface source (github.com/ROBOTIS-GIT/dynamixel_hardware_interface)

CHAPTER 7

SmartFusion2 DYNAMIXEL Compatibility Specification

Status: unverified. This is the shared contract for the Robot PC and the separate SmartFusion2 firmware. No register values, actuator identity, or drop-in compatibility claim is currently established. Keep unknowns explicit; do not infer XM430 behavior from the OpenMANIPULATOR-X product description.

7.1 Scope

The target firmware is a virtual DYNAMIXEL device over USB CDC that controls stepper motors on the BCN3D Moveo mechanism. The preferred ROS path reuses ROBOTIS `dynamixel_hardware_interface` and DynamixelSDK if source analysis confirms compatibility. The SmartFusion2 is not intended to transparently forward USB packets to physical DYNAMIXEL actuators.

The ROBOTIS OpenMANIPULATOR-X source is a software-stack reference, not evidence that its stock actuator model, joint names, motion ranges, or register set fit the Moveo build.

7.2 Sources to pin and inspect

For every finding, record the immutable commit or release, file path, and relevant symbol/line or documented section. Branch names such as `jazzy` can move and are not sufficient as reproducible evidence.

- DynamixelSDK (github.com/ROBOTIS-GIT/DynamixelSDK)
- dynamixel_hardware_interface (github.com/ROBOTIS-GIT/dynamixel_hardware_interface)
- dynamixel_interfaces (github.com/ROBOTIS-GIT/dynamixel_interfaces)
- OpenMANIPULATOR ROS 2 (github.com/ROBOTIS-GIT/open_manipulator)
- Selected DYNAMIXEL model documentation and the actual SmartFusion2 USB/firmware implementation.

Inspect the host implementation rather than guessing from a model control-table PDF: identify the actual startup reads/writes, register widths and signedness, command/state interfaces, operating modes, packet instructions, status/error handling, retries, timeout behavior, and update loop. Confirm what behavior is meaningful for a stepper-driven Moveo joint.

7.3 Evidence-backed device contract

Replace `Unverified` only when source or hardware evidence is recorded. If a requirement does not apply to the virtual stepper device, document why and how the host path handles it.

Requirement	Verified value	Source revision and evidence
Host package versions/commits	Unverified	Unverified
Host-requested actuator model and protocol	Unverified	Unverified
Device ID and discovery behavior	Unverified	Unverified
USB CDC framing, baud/line settings, and response timing	Unverified	Unverified
Protocol instructions actually used	Unverified	Unverified
Control Table addresses, widths, signedness, and access rules	Unverified	Unverified
Initialization and configuration sequence	Unverified	Unverified
Command interfaces and supported operating modes	Unverified	Unverified
State interfaces and measurement source	Unverified	Unverified
Torque/enable and fault behavior	Unverified	Unverified
Update rate, latency, retries, and timeouts	Unverified	Unverified
Moveo joint-to-device/motor mapping	Unverified	Unverified
Unmodified stock hardware-interface compatibility	Unverified	Unverified

7.4 Verification record

For each test, record hardware and firmware revisions, host OS/ROS distribution, exact source commits, configuration, command, expected result, and observed result. Distinguish protocol-level simulation from a physical stepper-axis test.

Minimum staged evidence:

1. Confirm the board enumerates as USB CDC and the Robot PC can open the device.
2. Verify Protocol 2.0 packet framing and status/error responses against the selected DynamixelSDK transport.
3. Test required device discovery and the precise read/write transactions observed in the hardware-interface source.
4. Verify initialization, state reporting, command updates, and timeout/failure behavior for one virtual actuator.
5. Validate one real stepper axis with its selected driver, scaling, limits, reference method, and stop behavior.

Do not claim full XM430 emulation or production readiness based only on successful packet exchange. A device model identifier, register behavior, mechanical mapping, and physical motion all require separate evidence.

CHAPTER 8

Operator PC

8.1 Responsibility

The Operator PC owns motion planning, operator visualization, and simulation workflows:

- MoveIt 2 planning and execution clients.
- RViz2 visualization and operator-facing configuration.
- Gazebo simulation and simulation-specific `ros2_control` integration.
- ROS 2 DDS communication with the Robot PC over the configured network.

It does not own the SmartFusion2 USB connection, motor hardware, or physical `ros2_control` hardware interface. Do not pass `/dev/ttyACM0` through to the Operator PC.

8.2 Separate repository

Operator PC implementation is maintained in the separate sfs-smartmanipulator-x-operator (github.com/SystemFromScratch/sfs-smartmanipulator-x-operator) repository; this documentation contains its architecture guidance only. Keep Operator-specific launch files, MoveIt configuration, RViz setup, Gazebo assets, and runtime/container configuration in that repository. The Robot PC Docker image must remain free of Operator PC GUI and simulation dependencies.

Consume the canonical robot description from `smartmanipulator_x_description` (in the Robot PC repository) or a future shared-description repository. The package is generated from the BCN3D Moveo CAD (see BCN3D Moveo mechanics (p. 20)); its joint limits are placeholders until measured on the arm, so planning limits must be revisited then. Do not duplicate an independent URDF/Xacro or substitute the OpenMANIPULATOR-X model for Moveo.

8.3 Real robot and simulation

The intended MoveIt application should work with either the physical Robot PC or simulation. The robot model, joint names, limits, planning groups, and controller-facing joint semantics must agree in both modes:

```
Real robot: MoveIt -> DDS -> Robot PC ros2_control -> verified DYNAMIXEL path ->
SmartFusion2
Simulation: MoveIt -> Gazebo ros2_control integration -> simulated Moveo joints
```

Use the physical path only after SmartFusion2 compatibility and robot bringup are verified. Gazebo integration and MoveIt configuration belong to the Operator PC repository.

8.4 DDS checklist

For two-host operation, configure matching ROS 2 domain IDs and verify that both hosts use reachable network interfaces. Check DDS discovery, routing/multicast support, and firewall rules in the deployment network. SSH can be used for administration, but it is not a substitute for DDS connectivity. Simulation-only runs can be local to the Operator PC and do not require physical hardware.

CHAPTER 9

Firmware: getting started

This guide takes you from a fresh clone of the controller firmware repository (SmartManipulator-X-Controller, separate from the Robot PC repository) to a built firmware image and a running motor tester. The development environment (p. 34) page has details and troubleshooting.

9.1 Prerequisites

Where	What
Host (Windows or Linux)	Git, Docker Desktop / Docker Engine with BuildKit, VS Code with the **Dev Containers** extension
Host, for the GUI	Python 3.9+ with Tkinter (the python.org installer on Windows; `python3-tk python3-venv` on Ubuntu)
Host, for the FPGA design	Microchip Libero SoC with its license, outside Docker
Hardware	M2S Hello FPGA Kit, USB Mini-B cable, and for motion: six TMC5160 modules, motors, and a 24 V supply (wiring)

The firmware toolchain (Zephyr SDK, west, CMake, Ninja, ccache) comes from the repository `Dockerfile`. Do not install it on the host.

9.2 Clone

```
git clone <repository-url> SmartManipulator-X-Controller
cd SmartManipulator-X-Controller
```

The repository has a `.gitattributes` file, so shell scripts keep LF line endings on Windows too. If you cloned before it was added, see Line endings (p. 38).

9.3 Open the Dev Container

1. Open the folder in VS Code.
2. Press **F1** and run **Dev Containers: Reopen in Container**.

The first start builds the image `Dockerfile`, which takes several minutes. The repository is mounted at `/workspace` and the compiler cache at `/ccache`.

9.4 Fetch Zephyr and build

Press **Ctrl+Shift+B** to open the **Zephyr: Workspace Menu**:

1. **Initialize + update** runs `west init -l .` and `west update`. Do this once; it downloads Zephyr `v4.5.0-rc1`.
2. Pick the firmware variant:
3. **Build simulation firmware** builds into `build-sim/`. It needs no motors or drivers, and every axis is simulated in the firmware. **Start with this one** to check the board, the USB/UART link, and the protocol.
4. **Build firmware** builds the target firmware into `build/`. It drives the six TMC5160 modules over SPI.

The same steps from the container terminal:

```
west init -l .        # first time only
west update
west build -b m2s_hello_fpga_kit/m2s010 -s apps/SmartManipulator_Controller -d build-sim -
- -DFILE_SUFFIX=sim
west build -b m2s_hello_fpga_kit/m2s010 -s apps/SmartManipulator_Controller -d build
```

The images are written to `build-sim/zephyr/` and `build/zephyr/` (`zephyr.elf`, `zephyr.hex`, `zephyr.bin`). The firmware variants (p. 41) table compares the two.

9.5 Program the board

The SmartFusion2 hardware design (MSS, clocks, fabric, pins) comes from the Libero project, and the Libero tools program it on the host. Load the Zephyr image into the configured design with your Microchip programming or debug setup. Use `zephyr.elf` for debugging. This repository does not yet define a scripted programming workflow.

9.6 Test the axes with the GUI (on the host)

On **Windows** in VS Code, outside the container, press **Ctrl+Shift+B** and choose:

- **Create .venv**: creates `.venv` and installs `requirements.txt`. Do this once.
- **Run motor tester GUI**: starts the GUI with the board.
- **Run motor tester GUI (simulator)**: starts it without hardware.

From a terminal instead:

```
py -m venv .venv
.\.venv\Scripts\python.exe -m pip install -r requirements.txt
.\.venv\Scripts\python.exe tools\dxl_test_gui\dxl_gui.py          # add --sim without
hardware
```

```
python3 -m venv .venv && .venv/bin/python -m pip install -r requirements.txt
.venv/bin/python tools/dxl_test_gui/dxl_gui.py
```

Select the kit's COM port (`/dev/ttyACM0` on Linux), keep 460800 bps (or choose **Auto**) and **PIC32 bridge (&)** ticked (the kit's PIC32 forwards data only after `&`, see Host link (p. 64)), and click **Connect**. The tool pings IDs 1-6, the controller and the gripper servo (ID 7). With the simulation firmware, the device list shows **SIM ·** for every axis (for example `SIM · 17HS19-2504S`), the axes move in the simulation, and the telemetry (velocity, current, voltage, temperature) is filled in. See the GUI guide (p. 54) for a complete per-axis test.

To run the automated protocol check (49 checks with the simulation firmware; protocol-only on the target firmware):

```
.\.venv\Scripts\python.exe tools\dxl_test_gui\firmware_check.py --port COM5
```

> **Safety:** the motor currents and ramps come from the motor profiles (`include/motor/motor_profiles.h`), the motor of each axis from `include/motor/motor_axes.h`. Before you drive a motor, check that the profile on its axis matches the motor that is actually connected (see Motor profiles and axis assignment (p. 78)), and wire the motor with VM off as shown in Motor connection (p. 75).

9.7 Next steps

- Read the firmware architecture (p. 41) before changing code.
- Run the protocol unit tests after codec changes. The commands are in Tests (p. 37).
- Agent and engineering rules for the firmware workspace are in the `zephyr-smartfusion2` skill of the shared `sfs-smartmanipulator-x-skills` repository.

CHAPTER 10

Firmware development environment

This page describes the controller firmware repository (SmartManipulator-X-Controller). The Robot PC repository has its own development page (p. 12). The firmware repository uses separate environments:

Environment	Runs where	Contains
Firmware	Docker image (Dev Container, Compose, or CI)	Zephyr, west, Zephyr SDK with `arm-zephyr-eabi`, CMake, Ninja, ccache, and the DynamixelSDK host library
Host tools	Repository `.venv` on Windows or Linux	Python dependencies of `tools/dxl_test_gui` (`requirements.txt`)
FPGA / hardware	Host only	Microchip Libero SoC and licenses: MSS, clocks, fabric, pins, programming

Libero and Microchip licenses are never installed in the image or committed to the repository.

10.1 Repository layout

```
.
├── apps/SmartManipulator_Controller/   Zephyr application (product firmware)
│   ├── boards/                         target overlay, _sim overlay, board .conf
│   ├── include/{dynamixel,motor,servo}/ public module headers
│   ├── src/                            main, transport, codec, XW430 and servo models,
│   │                                   motor workers and backends, servo output
│   ├── tests/protocol2_test.c          host unit tests for the codec
│   ├── prj.conf, prj_sim.conf, Kconfig, CMakeLists.txt
├── tools/dxl_test_gui/                 Protocol 2.0 motor tester (Python/Tkinter)
├── .devcontainer/devcontainer.json     Dev Container (uses ../Dockerfile)
├── .vscode/                            tasks, workspace menu scripts, settings
├── Dockerfile, compose.yaml            firmware build image
├── west.yml                            Zephyr manifest (pinned revision)
├── requirements.txt                    host Python dependencies (.venv) of the GUI
└── .gitattributes                      line-ending rules (LF; CRLF for .bat/.ps1)
```

Generated directories (`build/`, `.west/`, `.ccache/`, `.venv/`, `zephyr/` checkouts) are ignored and must not be committed.

10.2 Pinned tool versions

Component	Version
Zephyr (`west.yml`)	`v4.5.0-rc1` (first release with `m2s_hello_fpga_kit/m2s010`)
Base image	`python:3.12.15-slim-bookworm`, digest-pinned, `linux/amd64`
west	1.4.0

Component	Version
Zephyr SDK	1.0.0 (ARM toolchain in `/opt/zephyr-sdk/gnu/arm-zephyr-eabi`)
CMake	3.30.5
ccache	4.12.3
jsonschema / pyelftools	4.26.0 / 0.33
DynamixelSDK (host library only)	4.1.0, commit `f838bc90f72fcf5b9c279432d6e12fc24969daa8`
pyserial (`.venv`)	3.5
Image tag (`compose.yaml`)	`zephyr-sf2:1.0.0`

Versions are `ARG`s in the `Dockerfile`. The SDK, ARM toolchain, and ccache archives are verified by SHA-256. Do not use `latest` or a moving branch for build-critical dependencies.

10.3 Dev Container

1. Install Docker (with BuildKit) and the VS Code **Dev Containers** extension.
2. Open the repository and run **F1 → Dev Containers: Reopen in Container**.

`.devcontainer/devcontainer.json` builds the repository `Dockerfile` (the single source of truth for tool versions). It mounts the repository at `/workspace` and `.ccache/` at `/ccache`, sets `BOARD=m2s_hello_fpga_kit/m2s010`, and installs the Zephyr IDE, C/C++, CMake, Cortex-Debug, and Serial Monitor extensions. After you change the `Dockerfile` or build arguments, run **Dev Containers: Rebuild Container**. Reopening the container does not update the tools.

Check the toolchain inside the container:

```
west --version
arm-zephyr-eabi-gcc --version
cmake --version && ninja --version
```

10.4 VS Code tasks

Ctrl+Shift+B opens **Zephyr: Workspace Menu**. Its options depend on the operating system:

OS	Action	Implementation
Linux (Dev Container)	**Initialize + update**: `west init -l .` if needed, then `west update`	`.vscode/workspace-menu.sh`
	Update dependencies: initializes if needed, then `west update`	
	Build firmware: checks the workspace (`ensure-west-workspace.sh`), then `west build`	
	Clean: removes `build/`, `build-sim/`, `build-native/`	
	Clean and Build: pristine build (`-p always`)	
	Build simulation firmware / **Clean and Build simulation firmware**: `-d build-sim -- -DFILE_SUFFIX=sim`	

OS	Action	Implementation
	Run simulation on PC (native_sim): builds the simulation for `native_sim/native/64` and runs it; the UART is a `/dev/pts/N`	
Windows host	**Create .venv**: creates `.venv`, upgrades pip, installs `requirements.txt`, checks Tkinter	`.vscode/host-menu.ps1`
	Run motor tester GUI / **(simulator)**: creates `.venv` if needed, then starts `tools/dxl_test_gui`	

Individual tasks are available under **Tasks: Run Task**: *Zephyr: Build*, *Zephyr: Pristine Build*, *Zephyr: Build (simulation)* (all depend on *Zephyr: Ensure workspace*), *Zephyr: Clean*, *Zephyr: Ccache Stats*, *Zephyr: Ccache Clear*. Keep shell logic in the scripts under `.vscode/`, not inline in `tasks.json`.

10.5 Building

```
# target firmware (TMC5160 over SPI)
west build -b m2s_hello_fpga_kit/m2s010 -s apps/SmartManipulator_Controller -d build
# simulation firmware (no motor hardware; prj_sim.conf + _sim overlay)
west build -b m2s_hello_fpga_kit/m2s010 -s apps/SmartManipulator_Controller -d build-sim -
- -DFILE_SUFFIX=sim
# simulation firmware as a Linux executable (UART on a pseudo-terminal)
west build -b native_sim/native/64 -s apps/SmartManipulator_Controller -d build-native --
-DFILE_SUFFIX=sim
```

Use a separate build directory per variant. The variants are described in Firmware variants (p. 41). Run these commands from the workspace root. The root is a west workspace, and `-s` selects the application. Make a **pristine build** after changes to `prj.conf`, Kconfig, DeviceTree overlays, CMake files, the Zephyr version, or the toolchain:

```
west build -p always -b m2s_hello_fpga_kit/m2s010 -s apps/SmartManipulator_Controller -d
build
```

Zephyr picks the board overlay from `boards/` automatically: `m2s_hello_fpga_kit_m2s010.overlay` for the target, and `m2s_hello_fpga_kit_m2s010_sim.overlay` with `FILE_SUFFIX=sim`. Passing `-DDTC_OVERLAY_FILE=...` replaces it; use `-DEXTRA_DTC_OVERLAY_FILE=...` to add one.

Build artifacts

File	Use
`build/zephyr/zephyr.elf`	Debugging (symbols)
`build/zephyr/zephyr.hex` / `zephyr.bin`	Programming tools. Check the eNVM and hardware configuration first.
`build/compile_commands.json`	IntelliSense (`C_Cpp.default.compileCommands`)

ccache

The cache lives in `.ccache/` on the host and is mounted at `/ccache`, so it survives container rebuilds. Use `ccache --show-stats` and `ccache --clear`, or the matching VS Code tasks. The cache is only an optimization. If a build behaves strangely, make a pristine build first.

10.6 West and Zephyr

`west.yml` pins Zephyr `v4.5.0-rc1` and imports its modules. Run `west update` after changing the manifest. Keep the revision pinned; never use `main`. A change of the Zephyr version must be verified with a build, a pristine build, the host tests, and a test on the kit.

10.7 Host Python environment (.venv)

`requirements.txt` in the repository root lists the host dependencies: `pyserial==3.5` for the GUI. Tkinter comes with Python and cannot be installed with pip.

```
# Windows (PowerShell, repository root)
py -m venv .venv
.\.venv\Scripts\python.exe -m pip install -r requirements.txt
```

```
# Linux (repository root); first: sudo apt install python3-tk python3-venv
python3 -m venv .venv
.venv/bin/python -m pip install -r requirements.txt
```

On Windows, the **Create .venv** menu action does the same. `tools/dxl_test_gui/run_gui.bat` creates the environment on first use.

10.8 Tests

Test	Command	Where
Firmware codec (C)	`gcc -std=c11 -Wall -Wextra -Werror -I include src/protocol2.c tests/protocol2_test.c -o /tmp/protocol2_test && /tmp/protocol2_test` in `apps/SmartManipulator_Controller`	Container or any gcc host
GUI codec, bus, simulator, and **firmware cross-check**	`python -m unittest discover -s tests -v` in `tools/dxl_test_gui`	`.venv` or the container
Protocol conformance of the real firmware	`python tools/dxl_test_gui/firmware_check.py --port <COMx or /dev/pts/N>`	Board with the simulation (or target) firmware; or `native_sim` in the container

The cross-check compiles `src/protocol2.c` into a shared library. It verifies that the firmware and the Python codec produce identical packets and that the firmware parser accepts the Python packets. It is skipped when `gcc` is not available.

`firmware_check.py` sends a fixed sequence of transactions and reports PASS or FAIL per check: - PING and broadcast PING, identity registers, and Realtime Tick; - every error code; - SYNC_READ, SYNC_WRITE, REG_WRITE, and ACTION; - CRC and stuffing robustness; - the gripper servo device, if the controller reports one (protocol, limits, ramp, SYNC_WRITE); - on the simulation firmware only: position and velocity motion and telemetry.

It never moves motors on the target firmware unless `--allow-motion` is given. To test the complete firmware stack without a board, inside the container:

```
west build -b native_sim/native/64 -s apps/SmartManipulator_Controller -d build-native --
-DFILE_SUFFIX=sim
build-native/zephyr/zephyr.exe &          # note the /dev/pts/N it prints
python3 tools/dxl_test_gui/firmware_check.py --port /dev/pts/N
```

This needs `pyserial` in the container Python; the image does not include it (`pip install pyserial==3.5`). The motor backends have no unit tests; `motor_tmc5160.c` must be tested on hardware.

10.9 Documentation

The system documentation, including these firmware pages, is maintained in the Robot PC repository. See Documentation (p. 13) there.

10.10 Code style

`clang-format.txt` holds the formatting configuration (LLVM-based, with embedded-systems customizations). clang-format looks for a file named `.clang-format`, so either pass the file explicitly (`clang-format --style=file:clang-format.txt`) or copy it to `.clang-format`.

10.11 Line endings

`.gitattributes` makes every text file LF in the working tree, including on Windows. The exceptions are `*.bat`, `*.cmd`, and `*.ps1`, which use CRLF. Bash in the container fails on CRLF scripts (`$'\r': command not found`). If an old clone still has CRLF files, commit your work and then run:

```
git rm -r --cached -q . && git reset --hard
```

10.12 Docker without VS Code

```
docker build -t zephyr-sf2:1.0.0 .
docker run --rm -it -v "$PWD:/workspace" -v "$PWD/.ccache:/ccache" -w /workspace zephyr-
sf2:1.0.0
# or, using compose.yaml (runs the default build):
docker compose run --rm zephyr
```

10.13 Debugging

Debug with `zephyr.elf` through VS Code, Cortex-Debug, GDB, and a SmartFusion2-compatible debug probe. The probe configuration depends on the board setup and is not yet defined here. The host UART carries binary Protocol 2.0 traffic, so there is no text console. Use a debugger, or route diagnostics to the spare `uart1` with an explicit DeviceTree and Kconfig change.

10.14 Updating tool versions

1. Change the `ARG` in the `Dockerfile` (and its SHA-256 for downloaded archives).
2. Rebuild the image (`docker build --no-cache ...` or **Rebuild Container**).
3. Build, make a pristine build, and run the host tests.
4. Test on the M2S Hello FPGA Kit.
5. Bump the image tag and update the version tables (this page and the `zephyr-smartfusion2` skill).

10.15 CI

CI should use the same `Dockerfile`: build the image, run `west update`, build both variants (`build` and `build-sim`), and archive `zephyr.elf/.hex/.bin`. Run the codec tests as well. Run the `native_sim` build together with `firmware_check.py` as an end-to-end protocol test. Persist `/ccache` through the CI cache. A firmware-only pipeline needs no Libero and no Microchip licenses.

10.16 Licenses and secrets

Never commit or bake into images: Microchip license files, license server credentials, tokens, keys, or proprietary binaries. Keep Libero on the host.

10.17 Troubleshooting

Symptom	Fix
`$'\r': command not found` in a `.vscode/*.sh` task	CRLF line endings; see Line endings
`west: command not found`	Not inside the container; `which west` should print `/opt/venv/bin/west`
`source directory "." does not contain a CMakeLists.txt`	Add `-s apps/SmartManipulator_Controller`, or use the menu
`arm-zephyr-eabi-gcc: command not found`	`PATH` must contain `/opt/zephyr-sdk/gnu/arm-zephyr-eabi/bin`; rebuild the container
Strange build after `prj.conf` / overlay changes	Pristine build (`-p always`)
IntelliSense errors	Build once so `build/compile_commands.json` exists
ccache shows no hits	`echo $CCACHE_DIR` should print `/ccache`; check that `.ccache/` exists on the host

Symptom	Fix
GUI: `No module named serial`	Use the `.venv` interpreter, or run **Create .venv**
GUI: `No module named tkinter`	Reinstall Python from python.org with Tcl/Tk, or `apt install python3-tk`
GUI: timeouts on every ID	Wrong COM port or baud rate (kit USB port: 460800; choose **Auto** to detect it), PIC32 not switched to UART (`&`, checkbox **PIC32 bridge**), board not powered, firmware not running, or the UART still has a text console

CHAPTER 11

Firmware architecture

The application in `apps/SmartManipulator_Controller` of the controller repository (SmartManipulator-X-Controller) turns the SmartFusion2 Cortex-M3 into six virtual DYNAMIXEL XW430-T333-R devices. Protocol 2.0 packets from the Robot PC arrive on the host UART. They are decoded and applied to a per-axis Control Table model, then executed by a **motor backend**: either six TMC5160 drivers on SPI (target firmware) or a simulated axis model (simulation firmware).

11.1 Firmware variants

	Target firmware	Simulation firmware
Purpose	Drive the robot arm	Verify the DYNAMIXEL protocol implementation on the board, without motors
Build	`west build -b m2s_hello_fpga_kit/m2s010 -s apps/SmartManipulator_Controller -d build`	same command with `-d build-sim --DFILE_SUFFIX=sim`
VS Code (Linux menu)	**Build firmware** / **Clean and Build**	**Build simulation firmware** / **Clean and Build simulation firmware**
Kconfig / overlay	`prj.conf`, `boards/m2s_hello_fpga_kit_m2s010.overlay` (6× `adi,tmc51xx`)	`prj_sim.conf`, `boards/m2s_hello_fpga_kit_m2s010_sim.overlay` (no SPI devices)
Motor backend	`src/motor_tmc5160.c` (Zephyr `stepper` / `stepper_ctrl`, SPI mode 3)	`src/motor_sim.c` (trapezoidal ramp generator per axis)
Firmware Version (reg. 6)	1	129 (bit 7 = simulation)
Position scale	Per axis from `include/motor/motor_axes.h`: full steps × 16 microsteps × gear (now 3200 per motor revolution)	3200 microsteps per output revolution
Telemetry (reg. 70, 124–146)	From TMC5160 registers: velocity (VACTUAL), trajectory (XTARGET, VMAX), current (CS_ACTUAL, the current setting in use, not a measurement), errors (DRV_STATUS, GSTAT); supply voltage and temperature are nominal values	Present PWM, Current, Velocity, Velocity/Position Trajectory, Input Voltage, Temperature (modelled)
Size (Zephyr v4.5.0-rc1, SDK 1.0.0)	31.2 KiB FLASH, 20.6 KiB RAM (with the gripper servo)	23.0 KiB FLASH, 18.9 KiB RAM (measured before the gripper servo was added)

Zephyr's `FILE_SUFFIX` mechanism picks `prj_sim.conf` and the `_sim` board overlay automatically. Everything above the motor backend (UART transport, codec, Control Table, threads, queues) is the same code in both variants, so a protocol that passes on the simulation firmware behaves the same on the target.

The simulation variant also builds for `native_sim/native/64` (a Linux executable whose UART is a pseudo-terminal). It is used for automated end-to-end tests of the real firmware code on a PC (**Run simulation on PC (native_sim)** in the Linux menu):

```
west build -b native_sim/native/64 -s apps/SmartManipulator_Controller -d build-native --
-DFILE_SUFFIX=sim
build-native/zephyr/zephyr.exe      # prints "uart connected to pseudotty: /dev/pts/N"
python tools/dxl_test_gui/firmware_check.py --port /dev/pts/N
```

11.2 Source layout

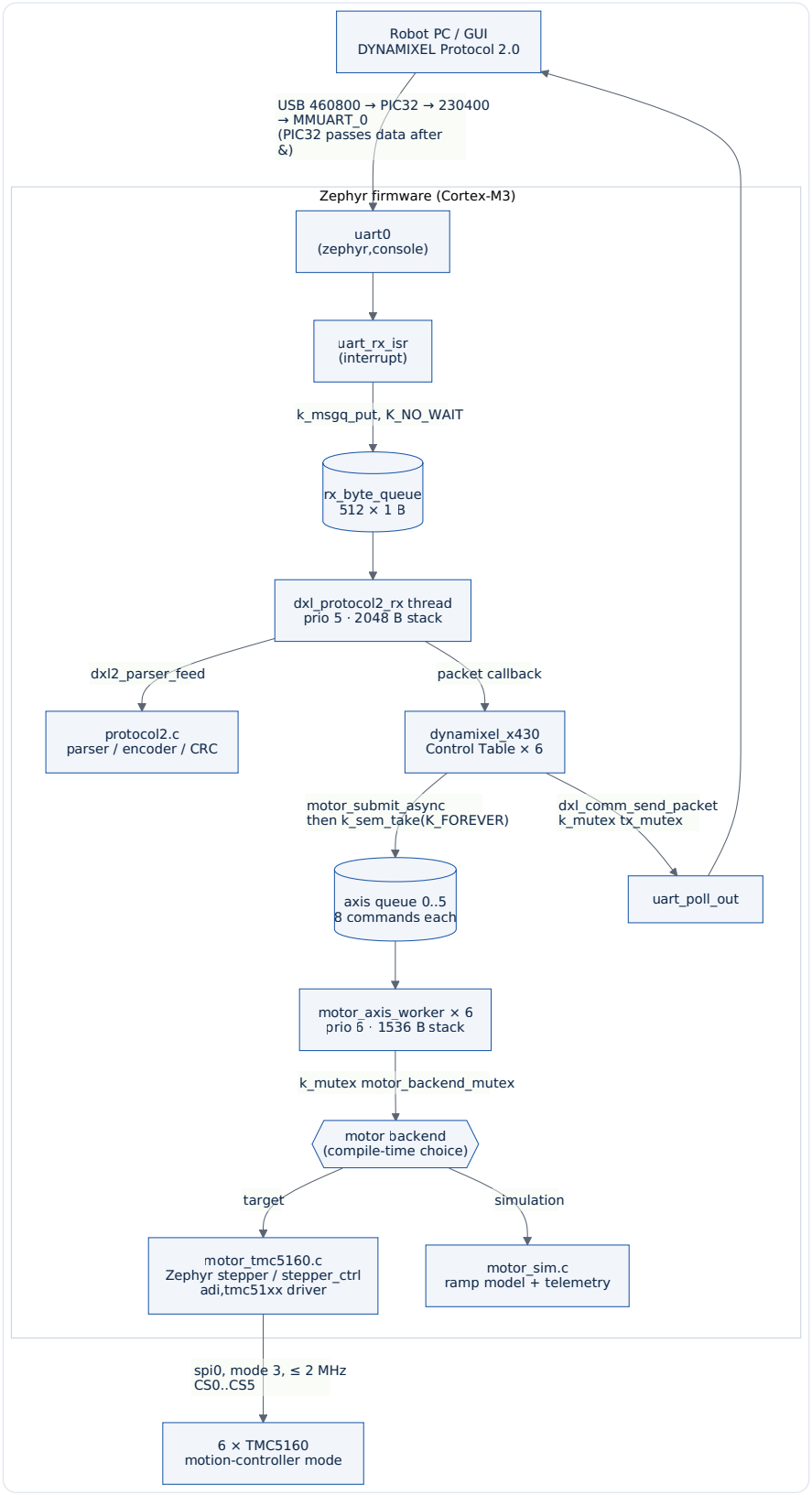
Path	Module	Responsibility
`src/main.c`	Composition root	Binds DeviceTree devices to the axes, configures the XW430 model, and starts the services in order
`include/dynamixel/protocol2.h`, `src/protocol2.c`	Protocol codec	CRC-16 (polynomial `0x8005`, initial value 0), incremental parser, encoder with byte stuffing. Plain C11 with no Zephyr headers, so it is unit-testable on the host.
`include/dynamixel/communication.h`, `src/communication.c`	Transport	UART RX interrupt, byte queue, `dxl_protocol2_rx` thread, packet dispatch, serialized TX, diagnostic counters, MMUART fractional baud divisor
`include/dynamixel/host_link.h`	Transport	Host UART node and bit rate (`DXL_HOST_BAUDRATE`) from the devicetree
`include/dynamixel/x430.h`, `src/dynamixel_x430.c`	Device model	Six virtual XW430-T333-R devices: instructions, Control Table, unit conversion, status packets
`include/motor/motor.h`, `src/motor_core.c`	Motor layer	Backend-independent API: six per-axis worker threads, bounded command queues, one backend mutex
`include/motor/motor_backend.h`	Backend interface	`motor_backend_init()`, `motor_backend_execute()`; exactly one backend is compiled
`src/motor_tmc5160.c`	Target backend	Binds the `tmc5160_axisN_driver/_controller` devices and executes `stepper` / `stepper_ctrl` calls
`src/motor_sim.c`	Simulation backend	Ramp-generator model per axis (position, velocity, run/stop) with telemetry
`include/motor/motor_profiles.h`	Motor data	One `MOTOR_<KEY>_<FIELD>` block per motor type: name, frame, rated current, IRUN/IHOLD, ramp. The GUI parses this file
`include/motor/motor_axes.h`	Axis assignment	Motor profile, direction (`INVERT`) and gear ratio per axis 0-5; `SMX_MICROSTEPS`
`boards/m2s_hello_fpga_kit_m2s010.overlay`	Hardware description (target)	Six `adi,tmc51xx` nodes on `spi0` (`reg` = chip select 0-5), with driver and motion-controller children; currents and ramps from `motor_axes.h`
`include/dynamixel/servo.h`, `src/dynamixel_servo.c`, `src/servo_*.c`	Gripper servo	Virtual DYNAMIXEL device for the gripper servo, see Gripper servo

Path	Module	Responsibility
`boards/m2s_hello_fpga_kit_m2s010_sim.overlay`	Hardware description (simulation)	Empty: only the host UART is used
`Kconfig`	Application options	`SMX_MOTOR_BACKEND` choice, `SMX_FIRMWARE_VERSION`, `SMX_MICROSTEPS_PER_OUTPUT_REVOLUTION` (0 = per axis from `motor_axes.h`), `SMX_SERVO_*`, `SMX_SIM_*`
`prj.conf` / `prj_sim.conf`	Kconfig per variant	Binary UART (no console, printk, or boot banner), interrupt-driven UART; target adds SPI and stepper
`tests/protocol2_test.c`	Tests	Host tests for the codec

The documentation sometimes uses target module names. They map to the code like this:

Target module	Code
`transport`	`communication.c`
`protocol`	`protocol2.c`
`device`	`dynamixel_x430.c`
`motor`	`motor_core.c` + `motor_tmc5160.c` / `motor_sim.c`
`fpga`	`servo_corepwm.c` (CorePWM registers over FIC_0/APB); the pin routing is in the Libero design

11.3 Layering



Firmware layers: from the host UART through the protocol thread and the axis workers to the TMC5160 drivers

11.4 Start-up order

`main()` runs on the Zephyr main thread. Every step must succeed before the next one starts; on the first error, `main()` returns that errno.

1. `motor_init()` calls `motor_backend_init()` under the backend mutex, then creates six worker threads, each with its own message queue.
2. The target backend checks that all `tmc5160_axisN_driver` and `tmc5160_axisN_controller` devices are ready. It returns `-ENODEV` otherwise, which usually means an SPI or DeviceTree problem.
3. The simulation backend resets the six axis models.
4. `dynamixel_x430_init(&x430_config)` initializes the device model. IDs 1-6 map to axes 0-5. The firmware version and position scale come from Kconfig. The configuration requires protocol type 2, a host bit rate with a Baud Rate code (230400 → 23), and unique IDs of at most 252. It also starts the gripper servo device (ID 7, pulse output off).
5. `dxl_comm_init(handle_packet, NULL)` registers the UART ISR, starts `dxl_protocol2_rx`, and enables RX. This step comes last, so no packet can reach a model that is not ready.

The main thread then exits. The console UART is binary, so start-up errors are not printed; debug them with a debugger.

11.5 Threads

Thread / context	Priority	Stack	Blocks on	Created in
`uart_rx_isr`	ISR	–	never	`dxl_comm_init`
`dxl_protocol2_rx`	`K_PRIO_PREEMPT(5)`	2048 B	`rx_byte_queue` (K_FOREVER), the per-call semaphore, `tx_mutex`	`dxl_comm_init`
`motor_axis_worker` × 6	`K_PRIO_PREEMPT(6)`	1536 B each	its axis queue (K_FOREVER), `motor_backend_mutex`	`motor_init`
`main`	Zephyr default	Zephyr default	– (exits after start-up)	Zephyr

In Zephyr a lower number means a higher priority, so the protocol thread preempts the motor workers. It does not starve them: while a motor command runs, the protocol thread is blocked on the command's semaphore.

11.6 Queues

Object	Type	Capacity	Producer → consumer	When full
`rx_byte_queue`	`k_msgq`	512 × `uint8_t`	`uart_rx_isr` → `dxl_protocol2_rx`	Byte dropped and counted by `dxl_comm_rx_overruns()`; the next packet fails CRC
`workers[n].queue` (`motor_core.c`)	`k_msgq`	8 × `motor_queue_item` (command, handler, user data)	`motor_submit_async` (any thread) → worker *n*	Returns `-ENOMSG` at once (`K_NO_WAIT`); the XW430 model answers with Result Fail

11.7 Mutexes and semaphores

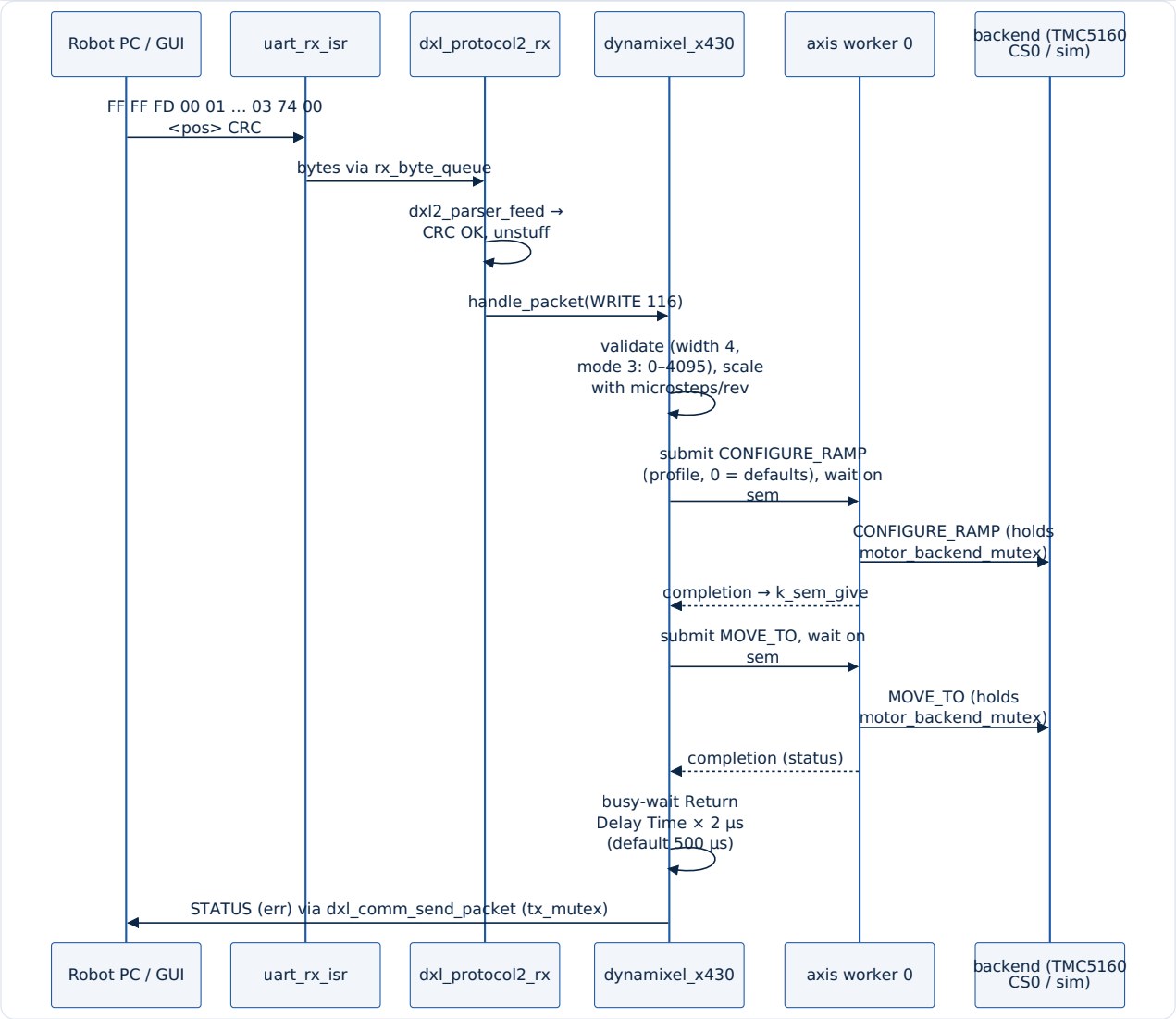
Object	Protects	Held by	Held during
`tx_mutex` (`communication.c`)	The UART TX byte stream, so packets never interleave	`dxl_comm_send_packet`	Polling out one encoded packet
`motor_backend_mutex` (`motor_core.c`)	Every backend call: the shared `spi0` bus and driver state of the six TMC5160s (target), or the shared model state (simulation)	Workers; `motor_init` (backend init)	One backend call: one stepper API call with its SPI transfers, or one model update
`x430_tmc_wait.completed` (`k_sem`, on the stack of each call)	Completion of one motor command	Given by the worker's completion handler, taken by `dxl_protocol2_rx`	Until the worker finishes the command (`K_FOREVER`)

Lock rules

- No code path holds two locks at once. `tx_mutex` and `motor_backend_mutex` are never nested, and the completion handler runs after the worker has released `motor_backend_mutex`. Keep it this way to stay deadlock-free.
- Only the backend files may call `stepper_*` / `stepper_ctrl_*` or touch SPI or the model state. Everything else submits commands through `motor_submit_async`. Backends need no locks of their own.
- The XW430 state (`axes[]` in `dynamixel_x430.c`) has no lock. It is accessed only from the `dxl_protocol2_rx` thread. Calling `dynamixel_x430_*` from another thread needs a lock first.
- The ISR uses only `k_msgq_put(..., K_NO_WAIT)` and atomics.

11.8 Request flow

A WRITE of *Goal Position* to ID 1 with torque on:



Request flow of a WRITE of Goal Position to ID 1

`SYNC_READ` and `SYNC_WRITE` repeat the per-axis part for each listed ID, one after another. `SYNC_WRITE` therefore does not start the axes at the same time.

11.9 Motor detection

DYNAMIXEL Protocol 2.0 detects devices with **PING**. The specification describes PING as the instruction used to check whether a device exists and to read its model number and firmware version. A device that is not present does not answer. The firmware follows this model:

Mechanism	Behavior
Driver probe (`MOTOR_OPERATION_PROBE`)	Target firmware: reads the TMC5160 `IOIN` register over SPI. The axis counts as present when `VERSION` is 0x30. Simulation firmware: every axis is present, except those listed in `SMX_SIM_ABSENT_AXIS_MASK`.
When it runs	At start-up, on every PING (unicast and broadcast), and on a READ of the controller's register 2

Mechanism	Behavior
Axis without a detected driver	**Does not answer at all**, like an unplugged DYNAMIXEL: no PING reply, no status packets, skipped in SYNC_READ/SYNC_WRITE and broadcast WRITE/ACTION
Model Information (address 2, 4 bytes, read-only)	Byte 0: detected driver (0x30 TMC5160, 0xFE simulated). Byte 1: configured motor frame (14/17/23, bit 7 = gearbox). Bytes 2–3: configured rated current in 10 mA units. The motor data comes from the profile assigned in `include/motor/motor_axes.h`; the driver byte is measured. The GUI maps frame and current back to the motor name and warns on a mismatch.
Controller device (ID `SMX_CONTROLLER_ID`, default 200)	Always answers PING (model number 0x5358, same Firmware Version) and READ 0–7: model, detected-axis bit mask (register 2, re-probed on read), axis count, backend (1 = TMC5160, 2 = simulation), firmware version, ID. Writes return Access Error. Hosts use it to tell "board alive, no drivers" from "board gone". Set it to 0 to disable it.

The TMC5160 logic is supplied from VM. **Without the 24 V motor supply no driver is detected**, and only the controller device answers. The SPI read of `IOIN` is serialized by `motor_backend_mutex`. The upstream driver does no SPI on its own, because no event callbacks or DIAG0 interrupts are configured.

11.10 Virtual device summary

Item	Value
Model	XW430-T333-R, model number 1270; firmware version 1 (target) or 129 (simulation)
IDs → axes	1 → A0 base, 2 → A1a shoulder, 3 → A1b shoulder, 4 → A2 elbow, 5 → A3 forearm roll, 6 → A4 wrist (CS0..CS5)
Instructions	PING, READ, WRITE, REG_WRITE (≤ 4 data bytes), ACTION, SYNC_READ, SYNC_WRITE; any other returns Instruction Error
Detection	Only axes with a detected driver answer; plus the controller device on ID 200 (see Motor detection)
Defaults	Operating mode 3, Status Return Level 2, Return Delay 250, Moving Threshold 10
Operating modes	1 velocity, 3 position (0–4095), 4 extended position (±1048575); changing the mode requires torque off
Writes	Must start at a register address and match its width exactly; volatile (no EEPROM)
Error mapping	`-EINVAL` → Data Range, `-EMSGSIZE` → Data Length, `-ENOTSUP/-ENODEV/-EACCES` → Access, other → Result Fail
Scaling	`motor = (dxl − homing_offset) × microsteps_per_rev / 4096`, rounded to the nearest step; velocity unit 0.229 rpm; acceleration unit 214.577 rev/min²
Telemetry	Hardware Error Status (70), Present PWM (124, 885 = 100 %), Present Current (126, 2.69 mA), Present Velocity (128), Velocity/Position Trajectory (136/140), Present Input Voltage (144, 0.1 V), Present Temperature (146, °C). Read through `MOTOR_OPERATION_GET_TELEMETRY`; Access Error when the backend returns `-ENOTSUP`. See Mapping to the stepper axes for what the TMC5160 can and cannot measure.

Item	Value
Real-time Tick	Milliseconds, 0–32767 (wraps)

11.11 Mapping to the stepper axes

The host sees XW430-T333-R servos; each one is a stepper motor driven open loop by a TMC5160. The registers are mapped as follows. "Motor profile" means the profile assigned in `include/motor/motor_axes.h` (data in `motor_profiles.h`).

Register	Meaning on the stepper axis
Goal / Present Position (116 / 132)	4096 units per **output** revolution = full steps × 16 × NUM / DEN microsteps (3200 per motor revolution at gear 1:1). Present Position is the ramp generator counter `XACTUAL`, not a measurement (no encoder yet): correct as long as the motor follows every step
Profile Velocity (112)	Maximum speed of position moves, 0.229 rpm of the output. **0 = Velocity Limit**. Must not exceed the Velocity Limit (Data Range Error)
Profile Acceleration (108)	Acceleration and deceleration, 214.577 rev/min². **0 = AMAX of the motor profile** (a stepper cannot accelerate without limit). Larger values are cut to the TMC5160 register limit
Velocity Limit (44, RW, Torque OFF)	Per axis, default `MAX_RPM` of the motor profile (bench values: 120 rpm NEMA17/14, 60 rpm NEMA23) converted to the output. Range 0–1023. Limits Goal Velocity and Profile Velocity
Current Limit (38, RW, Torque OFF)	**Motor run current** (TMC5160 IRUN), 2.69 mA units. Default IRUN of the motor profile (about 85 % of the rated current); writes up to the rated current (Data Range Error above). The driver sets the current in 1/32 steps of 3.06 A, so the value read back is rounded down to the next step. IHOLD stays at the profile value (at most IRUN)
Present Current (126)	`(CS_ACTUAL + 1) × 95.8 mA`: the current setting in use (IRUN while moving, IHOLD at standstill), signed with the direction; 0 with torque off. Not a measured current
Present Velocity / Velocity Trajectory / Position Trajectory (128 / 136 / 140)	`VACTUAL`; `±VMAX` in the direction of the move; `XTARGET` in position modes
Present PWM (124)	0: no duty-cycle readout in spreadCycle mode
Present Input Voltage (144)	Nominal `CONFIG_SMX_MOTOR_SUPPLY_MV` (24 V): the TMC5160 does not measure VM
Present Temperature (146)	25 °C nominal; 120 °C when DRV_STATUS reports the overtemperature warning, 150 °C at overtemperature shutdown
Hardware Error Status (70)	Bit 0 input voltage: charge pump undervoltage or a **driver reset** after a VM interruption. Bit 2 overheating: DRV_STATUS otpw / ot. Bit 4 electrical shock: short to ground or supply. Latched until the next Torque Enable = 1
Moving (122)	`VACTUAL ≠ 0` (no standstill delay)
Max / Min Voltage Limit (32 / 34)	40 V / 8 V, the range of the TMC5160 modules

Register	Meaning on the stepper axis
Drive Mode, Homing Offset (10, 20)	As on a DYNAMIXEL; like Current and Velocity Limit they can only be written with Torque OFF
Raw TMC5160 registers (1024 + 4 × register, project extension)	Read-only window for bring-up: READ of exactly 4 bytes at an aligned address returns that driver register (e.g. 1024 + 4 × 0x6F = DRV_STATUS). Write-only registers read 0. Access Error on the simulation firmware. `tools/dxl_test_gui/tmc_diag.py` decodes the important ones (IOIN pins, CHOPCONF/MRES, DRV_STATUS, MSCNT)

Behaviour that differs from a plain register store:

- **Torque OFF** stops the ramp generator at once (VMAX 0 with the largest AMAX, under 1 ms at bench speeds) before the power stage is switched off. With the normal soft stop the counter would keep counting through the deceleration while the unpowered motor no longer follows. After Torque OFF the shaft is free: if it is turned by hand or by gravity, Present Position is wrong until the axis is referenced again.
- **Position mode (3)**: the counter keeps the turns made in velocity or extended position mode. Before the power stage is switched on in mode 3, the firmware moves the counter by whole output revolutions so that Present Position (without the modulo) lies in 0–4095. The shaft does not move, and a Goal Position afterwards never unwinds earlier turns.
- **Operating Mode change** forgets the stored Goal Position and Goal Velocity (they belong to the old mode), so Torque ON afterwards does not start a move.
- **Driver reset** (VM interrupted, the TMC5160 restarts with reset values): every backend command first reads GSTAT. After a reset the firmware writes the complete configuration again (GCONF, CHOPCONF with MRES, currents, ramp, SW_MODE, COOLCONF), reports Torque OFF and Hardware Error bit 0, forgets the goals, and refuses the move that found the reset. Present Position restarts at 0. Switch Torque ON again to continue.
- **Zero position**: `XACTUAL` is 0 at power-up, so the pose of the arm at power-up is position 0 (2048 is not the centre unless Homing Offset says so). Place the arm in a known pose before switching on, or set Homing Offset.

The full register list with access rules is in `smartfusion2` skill (shared `sfs-smartmanipulator-x-skills` repository). Shoulder axes A1a and A1b drive the same joint. Configure them identically and command both goals back-to-back (wiring note 8).

11.12 Configuration

- **`prj.conf`**: the host UART is binary. Do not enable logging, `printk`, or the shell on it.
- **Host UART**: `DT_CHOSEN(zephyr_console)`, which is `uart0` on this board. A `BUILD_ASSERT` checks that it is enabled.
- **Host link bit rate**: 230400 bps, set as `current-speed` of `uart0` in both overlays (`boards/m2s_hello_fpga_kit_m2s010.overlay` and `..._sim.overlay`). Change it only there. `include/dynamixel/host_link.h` turns it into `DXL_HOST_BAUDRATE`, which sets the Baud Rate register (code 23 for 230400, a project code because the ROBOTIS X table has none; 46 for 460800; 2 for 115200). This is the PIC32 ↔ SmartFusion2 segment and must match the PIC32 (230400). The PC talks to the PIC32 at 460800 bps, and the PIC32 forwards data only after it has received `&` (the GUI and `firmware_check.py` send it); see Host link (p. 64). The Zephyr ns16550 driver programs only the integer divisor, which at the 50 MHz MSS peripheral clock gives 223214 bps (-3.1 %). `dxl_comm_init()` therefore sets the MMUART fractional divisor

(DLR/DMR + DFR/64, `MM0.EFBR`): 13 + 36/64 = 230414 bps (+0.006 %). A `BUILD_ASSERT` stops the build if a rate would be more than 0.5 % off. On `native_sim` the UART is a pseudo-terminal and the rate only sets the register value.

- **Overlay**: includes `include/motor/motor_axes.h`. Per axis it takes `irun`, `ihold`, `iholddelay`, the ramp (`vstart` … `vstop`), `micro-step-res` and `shaft` (when `SMX_AXISn_INVERT` is 1) from the assigned motor profile. The common `SMX_TMC5160_*` values come from the manufacturer documents: SPI 2 MHz (the MSS SPI driver turns any request below 1.67 MHz into a 25 MHz SCK), clock 12 MHz (CLK is tied to GND, so the internal clock is used), and TCOOLTHRS 0 (the driver always enables stop on stall). After any change to the headers, rebuild pristine (`west build -p`). See Motor profiles and axis assignment (p. 78).
- **Chopper and Moving**: after `stepper_enable()` the backend writes the chopper bits of CHOPCONF to `0x000100C3` (TOFF 3; the driver's TOFF 15 gave a loud, low chopper frequency). Moving (register 122) is read from `VACTUAL` over raw SPI. `DRV_STATUS.stst` (used before) reports standstill only 2^20 clocks (87 ms) after the last step.
- **Kconfig `SMX_MOTOR_SUPPLY_MV`** (target, default 24000): the value reported in Present Input Voltage.
- **Kconfig `SMX_MICROSTEPS_PER_OUTPUT_REVOLUTION`**: 0 in the target firmware, which means "per axis from `motor_axes.h`" (full steps × `SMX_MICROSTEPS` × `GEAR_NUM` / `GEAR_DEN`; with gear 1:1 this is 3200 per motor revolution). A non-zero value overrides it for all axes. The simulation firmware uses 3200.
- **Simulation parameters**: `SMX_SIM_DEFAULT_SPEED` (3200 µsteps/s), `SMX_SIM_DEFAULT_ACCELERATION` (6400 µsteps/s²), `SMX_SIM_HOLD_CURRENT_MA`, and `SMX_SIM_ACCEL_CURRENT_MA` shape the simulated motion and current.

11.13 Gripper servo

The gripper hobby servo (MG996R: about 120° for 1–2 ms, 11 kgf·cm at 6 V, 50 Hz signal on Arduino D3 / N7; see Gripper servo (MG996R) (p. 83)) is a separate virtual DYNAMIXEL device, **ID 7** (`CONFIG_SMX_SERVO_ID`). It answers PING, READ, WRITE, SYNC_READ and SYNC_WRITE, also as part of a broadcast PING, a broadcast WRITE and SYNC packets together with the axes. REG_WRITE and ACTION return Instruction Error. The controller device reports the servo ID in register 5 (0 = firmware built without `CONFIG_SMX_SERVO`).

Path	Responsibility
`include/dynamixel/servo.h`, `src/dynamixel_servo.c`	Device model: Control Table, software ramp (`k_timer`, 20 ms), status packets
`include/servo/servo_pwm.h`	Pulse output interface: `servo_pwm_init()`, `servo_pwm_set_pulse_us()` (0 = off)
`src/servo_corepwm.c`	Target: CorePWM channel 1 at `CONFIG_SMX_SERVO_COREPWM_BASE` (`0x5000_0000`), PRESCALE 99 (1 µs), PERIOD 19999 (20 ms), NEGEDGE = pulse width, `SYNC_UPDATE` after every edge write
`src/servo_sim.c`	Simulation: no output

Kconfig (`SMX_SERVO_*`):

Option	Default	Meaning
`SMX_SERVO`	y	Builds the servo device

Option	Default	Meaning
`SMX_SERVO_ID`	7	DYNAMIXEL ID; must differ from the axes and the controller
`SMX_SERVO_PULSE_MIN_US` / `_MAX_US`	1000 / 2000	Pulse width at position 0 and at the end of the travel
`SMX_SERVO_TRAVEL_DEG`	90	Rotation between the two pulse widths. 90 is a nominal value; the MG996R datasheet gives about 120° for 1–2 ms. Measure it on the gripper
`SMX_SERVO_INITIAL_DEG`	45	Goal Position after startup
`SMX_SERVO_COREPWM_BASE`	`0x50000000`	CorePWM address from the Libero memory map (target only)

Control Table (XW430 addresses, position mode only):

Register	Meaning on the servo
Model Number (0)	`0x5356` ("SV")
Model Information (2)	Byte 0 output driver (`0x50` CorePWM, `0xFE` simulated), byte 1 travel in degrees
Operating Mode (11)	3 only (other values: Data Range Error)
Goal / Present Position (116 / 132)	0.088° units, 0 … range (`TRAVEL_DEG` → 1024 units at 90°). Position maps linearly to the pulse width. Present Position is the **commanded** ramp position: the servo reports nothing back
Min / Max Position Limit (52 / 48, RW, Torque OFF)	Default 0 … range. Goal Position outside: Data Range Error; a stored goal is clamped into new limits
Profile Velocity (112)	Software ramp, 0.229 rpm (= 1.374 °/s per unit), 0–1023. **0 = no ramp**: the pulse jumps to the goal and the servo moves at its own speed
Torque Enable (64)	0 = no pulses (servo limp, also at startup). 1 = pulses start at Present Position and the ramp runs to the goal
Moving (122), Moving Status (123)	Ramp position ≠ goal while torque is on
Velocity Limit (44)	1023, read-only
Hardware Error Status (70)	0 (nothing is measured)
1024 / 1026 / 1028 / 1030 (project, 2 bytes, read-only)	Pulse Min, Pulse Max, Present Pulse (µs, 0 = off), Position Range

The servo position is unknown at power-up. The first Torque Enable = 1 moves the servo to the start position (`SMX_SERVO_INITIAL_DEG`) at full speed.

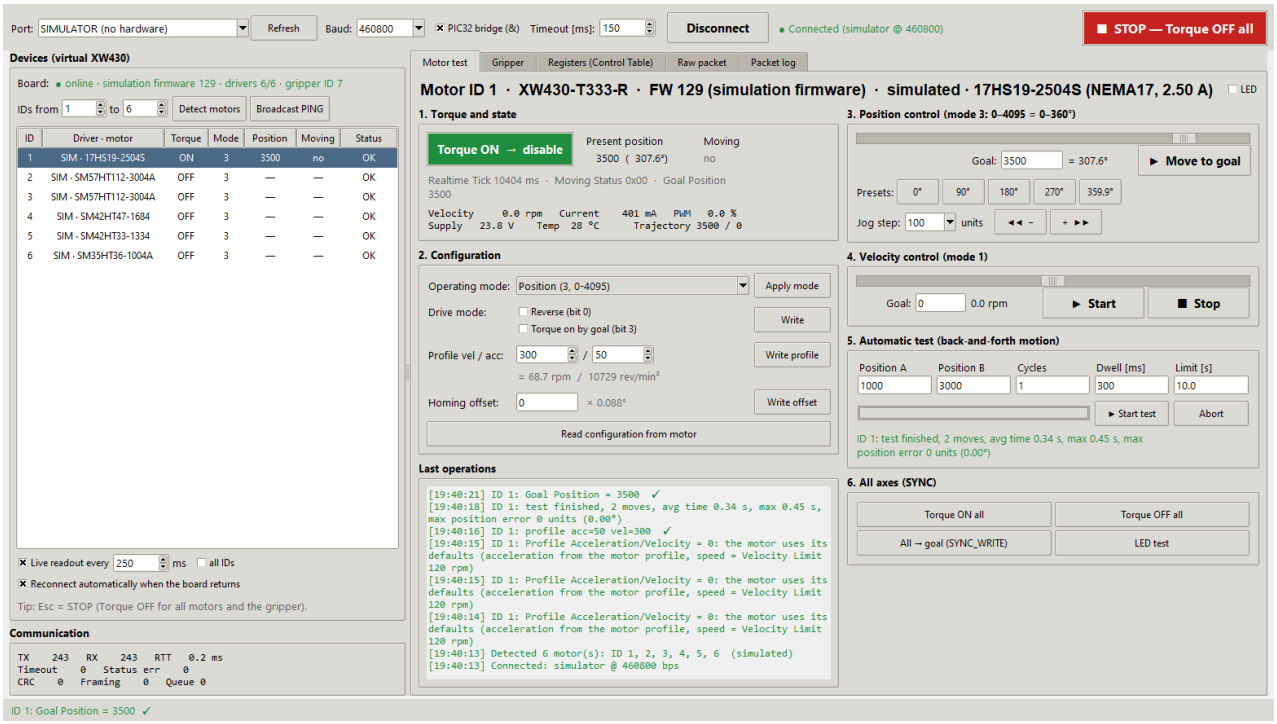
11.14 Known gaps

- Target firmware: host link, TMC5160 detection, register mapping, telemetry and the ramp generator verified on axis A0 (2026-10-11). The shaft does not turn yet, because the module runs in STEP/DIR mode (`SD_MODE=1`, finding 10 in TMC5160 driver modules (p. 80)). Gear ratios are not entered (scale is 1:1 to the motor shaft). The simulation firmware passes `firmware_check.py` (49/49 checks on `native_sim`, including the gripper servo).
- No encoders yet: Present Position is the commanded position (`XACTUAL`). A stall, a too high speed or acceleration for the load, or turning the shaft with torque off is not detected.
- No homing or reference at boot. The wiring concept expects a hand-placed start pose and `XACTUAL = 0` at power-up; check this when the axes are brought up.
- The gripper servo (CorePWM at `0x5000_0000`) follows Goal Position on the kit, but the travel of 90° for 1000–2000 µs is a nominal value and is not calibrated yet.
- GPIO LEDs, `uart1`, and I²C are not used yet.
- Health counters are not exported; there is no diagnostics channel.
- Bulk Read/Write, Fast Sync Read, Reboot, Factory Reset, Clear, and Backup are not implemented.

CHAPTER 12

Motor tester GUI

`tools/dxl_test_gui` in the controller repository (SmartManipulator-X-Controller) is a desktop application for testing the firmware and every stepper axis over **DYNAMIXEL Protocol 2.0** (ROBOTIS specification (docs.robotis.com/docs/dxl/protocol/protocol2/)). It acts as the bus master, the same role the Robot PC has. It runs on Windows or Linux and needs only Python, Tkinter, and `pyserial`.



Motor test tab with the simulator after an automatic A↔B test

12.1 Starting

Way	Command
VS Code on Windows	**Ctrl+Shift+B → Run motor tester GUI** or **Run motor tester GUI (simulator)**
Windows, no VS Code	double-click `tools\dxl_test_gui\run_gui.bat` (creates `.venv` on first run)
Any OS	`.venv` Python: `python tools/dxl_test_gui/dxl_gui.py [--port COM5] [--sim] [--sim-uncalibrated]`

Set up `.venv` first; see Development environment (p. 37).

Option	Meaning
`--port COM5`	Preselects a serial port

Option	Meaning
`--baud 460800` / `--baud Auto`	Preselects the PC port bit rate (default 460800, the kit's USB side)
`--no-pic32-bridge`	Does not send `&` after opening the port (direct USB-UART cable instead of the kit's PIC32 bridge)
`--sim`	Built-in Python simulator that behaves like the **simulation firmware** (motion and telemetry); no board needed
`--sim-uncalibrated`	Python simulator of the **target firmware** with an axis whose position scale is 0 (position and telemetry return Access Error)

With the real board, flash the **simulation firmware** first. The firmware runs on the kit, answers over the same UART, and simulates the axes, so you can test the whole protocol path without motors. The device list then shows **SIM ·** for every axis, and the board line and motor title show *simulation firmware* (Firmware Version 129).

12.2 Connecting

1. Connect the kit's USB Mini-B to the PC. The port shows up as `COMx` (Windows) or `/dev/ttyACM0` (Linux).
2. Pick the port and keep **Baud 460800**, the rate between the PC and the kit's PIC32 (the PIC32 talks to the firmware at 230400). Keep **PIC32 bridge (&)** ticked: the PIC32 passes data to the SmartFusion2 only after it has received `&`, which the tool sends after opening the port and after every rate change (see Host link (p. 64)). **Auto** tries 460800, 230400, 115200, 57600, 1000000, 2000000, and 9600 bps and keeps the first rate at which a device answers a broadcast PING. If you pick another rate while connected, the tool switches the open port and detects the motors again. The connection label shows the rate in use. *Timeout* is the time the tool waits for each status packet (default 150 ms).
3. Click **Connect**. The tool runs **Detect motors**: it pings IDs 1-6, the controller (ID 200) and the gripper servo (ID 7), and lists the axes with their driver and motor (see below).

Scan (PING) searches any ID range. **Broadcast PING** asks every device to answer at once.

12.3 Motor detection and connection monitoring

Detect motors pings the chosen ID range and the controller device (ID 200). It then reads each motor's **Model Information**. The device list shows:

- **Driver · motor** for every detected axis, for example `TMC5160 · 23HS2430 (NEMA23, 3.00 A)`. The driver is detected by the firmware; the motor frame and current are configured in the firmware. The GUI looks them up in the project's motor profiles (`apps/SmartManipulator_Controller/include/motor/motor_profiles.h`) to show the motor name. If several profiles share frame and current (23HS2430 and SM57HT112-3004A are both NEMA23, 3.0 A), the GUI shows the motor that `motor_axes.h` assigns to that ID.
- ⚠ **config: …** after the motor when the board reports a different motor than `motor_axes.h` in this checkout assigns to that ID, for example after flashing a firmware built from another configuration. The log then says *Motor configuration differs from this project*; rebuild and flash, or update the checkout.

- **— not detected / absent** for axes whose driver did not answer, for example a missing module or no 24 V motor supply.
- **Board** line: `online · <variant> firmware <version> · drivers n/6 · gripper ID 7`. Green means all drivers were found, orange means some are missing. The gripper part is missing when the firmware has no servo.

This uses only standard DYNAMIXEL Protocol 2.0 transactions (PING and READ).

Connection monitoring runs continuously:

Situation	What the GUI shows	What it does
USB cable or board unplugged (port disappears or I/O fails)	● **Connection lost**; rows marked *port lost*; controls disabled	Closes the port. With *Reconnect automatically* checked, it reopens the port when it reappears and detects the motors again.
Port still open, but the board stops answering (power off, reset, firmware hang)	● **Port open, board not responding**, after 3 cycles without an answer	Keeps checking every second. When the board answers again, it re-detects the motors.
One axis stops answering while the board is alive	Row status *driver lost*	The rest keeps working; click **Detect motors** to update.

When live readout is off, a one-second heartbeat (PING of the controller) keeps the monitoring running.

12.4 Window layout

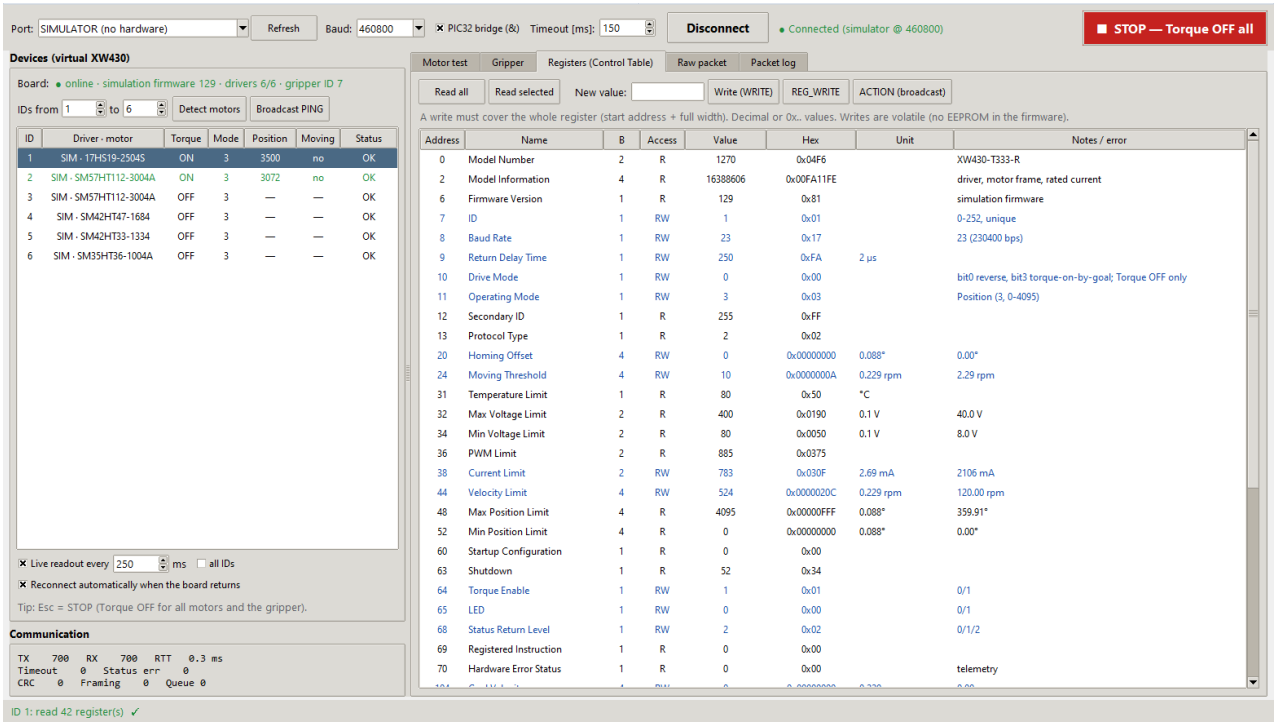
Area	Function
Top bar	Port, baud rate, **PIC32 bridge (&)**, timeout, Connect/Disconnect, connection state, and the red **STOP — Torque OFF all** button (key **Esc**; also switches the gripper servo off)
Device list	**Board** line, then one row per axis: ID, *Driver · motor* (`SIM · …` on the simulation firmware), torque, mode, present position, moving, status. With **Live readout** on, the selected motor (or all, with *all IDs*) is polled every N ms. *Reconnect automatically* reopens a lost port.
Communication	TX/RX counters, last round-trip time, timeouts, status errors, CRC and framing errors, worker queue
Motor test tab	Step-by-step test of the selected axis (below)
Gripper tab	Gripper servo (ID 7): torque, goal in degrees within the position limits, Min/Center/Max presets, Profile Velocity (ramp), position limits; live position, goal, pulse width (µs) and Moving while the tab is open (see Testing the gripper servo)
Registers tab	Control Table browser: Read all, Read selected, WRITE, REG_WRITE, ACTION
Raw packet tab	Any instruction with hex parameters; stuffing and CRC are added for you
Packet log tab	Every TX/RX packet with timestamp, decoding, and hex dump; *Save to file…*

12.5 Testing one stepper axis

> Before you drive a motor, check that the device list shows the right motor for the axis (no ⚠ config warning) and follow the bring-up checklist (p. 81). **Detect motors** works without motor current: it only needs VM on the module (the TMC5160 logic is powered from VM).

1. **Select the motor** in the device list. The tool reads its configuration.
2. **Section 2, Configuration:**
3. *Operating mode*: Position (3), Extended position (4), or Velocity (1). Click **Apply mode**. The mode can change only with torque off; the tool offers to switch torque off first.
4. *Drive mode*: **Reverse** (bit 0) inverts the direction. **Torque on by goal** (bit 3) enables torque automatically on a new goal.
5. *Profile vel / acc*, for example 100 / 20, then **Write profile**. 0 means the motor defaults: speed = Velocity Limit (address 44), acceleration from the motor profile. Profile velocity above the Velocity Limit returns Data Range Error.
6. *Homing offset* shifts the zero (units of 0.088°).
7. **Section 1, Torque**: click the button to switch torque ON (green) or OFF. Below the present position, the telemetry line shows velocity (rpm), current (mA), PWM (%), supply voltage, temperature, and the position and velocity trajectory. On the target firmware the current is the TMC5160 current setting in use, voltage and temperature are nominal values, and PWM is 0.
8. **Section 3, Position control**: use the slider (0–4095 = 0–360° in mode 3, ±4 turns in mode 4) or type a goal, then **Move to goal**. You can also use the 0°/90°/180°/270° presets or jog with − / + by the chosen step.
9. **Section 4, Velocity control** (mode 1): set a speed within ± the axis' Velocity Limit (× 0.229 rpm; the slider follows it), then **Start** / **Stop**.
10. **Section 5, Automatic test**: enter positions A and B, cycles, dwell, and a time limit, then **Start test**. The tool enables torque, moves A↔B, waits for *Moving = 0*, and reports each move's time and the final position error. **Abort** stops after the current step.
11. **Section 6, All axes**: Torque ON/OFF for all, all axes to the current goal with **SYNC_WRITE** (the firmware runs them one after another), and an LED test.

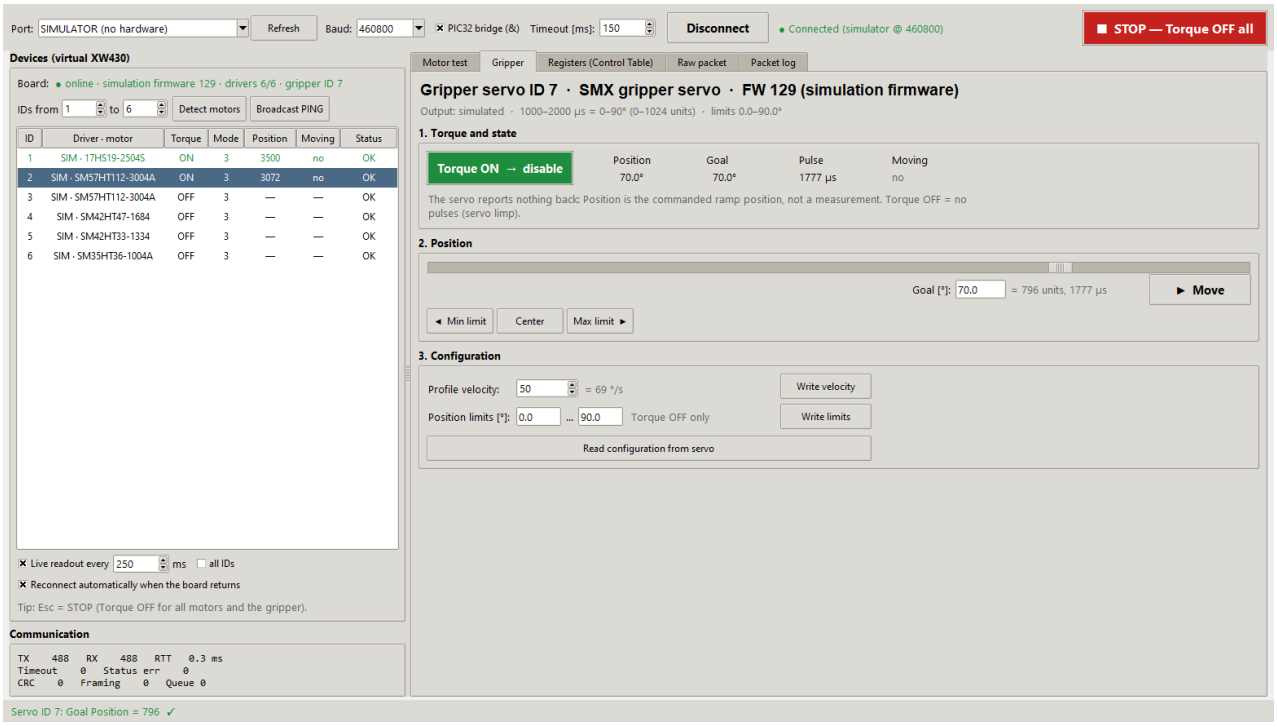
Every action is listed under **Last operations**: green for success, red for an error, with a hint. For example, Access Error on position means the axis has no position scale or the operating mode is wrong.



Registers tab

12.6 Testing the gripper servo

The gripper servo (MG996R on CorePWM) is a separate virtual DYNAMIXEL device, ID 7 by default. It has its own **Gripper** tab, not a row in the device list.



Gripper tab with the simulator: servo at 70° after a move with Profile velocity 50

Detection. Detect motors asks the controller (ID 200) for the servo ID (controller register 5) and then pings that ID. The **Board** line then ends with *gripper ID 7*. If register 5 is 0, the firmware was built without `CONFIG_SMX_SERVO`, and the tab shows *Gripper servo: not detected* with all controls disabled. Without a controller answer, the tool tries the default ID 7.

Header. The title shows the ID, the model (*SMX gripper servo*, model number `0x5356`) and the firmware variant. The line below shows the output (CorePWM on the target firmware, *simulated* on the simulation firmware), the pulse range and travel, for example *1000–2000 µs = 0–90° (0–1024 units)*, and the position limits in use.

Section	Controls	Registers
1. Torque and state	**Torque OFF → enable** / **Torque ON → disable**. Torque ON starts the 50 Hz pulses, Torque OFF stops them (the servo goes limp). Live values: *Position*, *Goal*, *Pulse* (µs, *off* with torque off) and *Moving*.	Torque Enable (64), Goal Position (116), Present Position (132), Moving (122)
2. Position	Slider and *Goal [°]* (Enter = Move), with the matching units and pulse width next to it. **Move** writes the goal. **Min limit**, **Center** and **Max limit** move to the position limits or their middle.	Goal Position (116)
3. Configuration	*Profile velocity* (0 = no ramp, the servo moves at its own full speed; each unit is 0.229 rpm, so 50 ≈ 69 °/s) and **Write velocity**. *Position limits [°]* and **Write limits** (Torque OFF only). **Read configuration from servo** loads velocity, limits, goal and torque into the tab.	Profile Velocity (112), Max/Min Position Limit (48/52)

The units are the axis units, 0.088° each (4096 = 360°); the servo travel 0–90° is 0–1024.

Steps:

1. Open the **Gripper** tab. **Detect motors** has already read the servo configuration into it. While the tab is open, the live readout polls the servo instead of the selected axis.
2. Set **Profile velocity**, for example 50, and click *Write velocity*.
3. Click **Torque OFF → enable**. The first time after power-up the servo moves to the start position (45°) at full speed, because its real position is not known.
4. Enter a goal in degrees, drag the slider, or use *Min limit*, *Center*, *Max limit*, then click **Move**. The firmware ramps the pulse in 20 ms steps; *Moving* shows *YES* until the ramp ends.
5. Restrict the travel to what the gripper can do: switch torque off, enter the limits in section 3 and click *Write limits*. The slider then covers only the new range.

Rules and messages:

- A goal outside the position limits is refused in the GUI with a hint; the firmware would answer Data Range Error.
- A goal written with torque off is only stored (the log says so); the servo moves to it when torque is switched on.
- Position limits written with torque on return Access Error.
- **STOP** (Esc) also switches the servo torque off; on closing, the tool offers to switch torque off on all motors and the gripper.
- *Position* is the commanded ramp position, not a measurement: the MG996R has no feedback. Check the real position by eye.

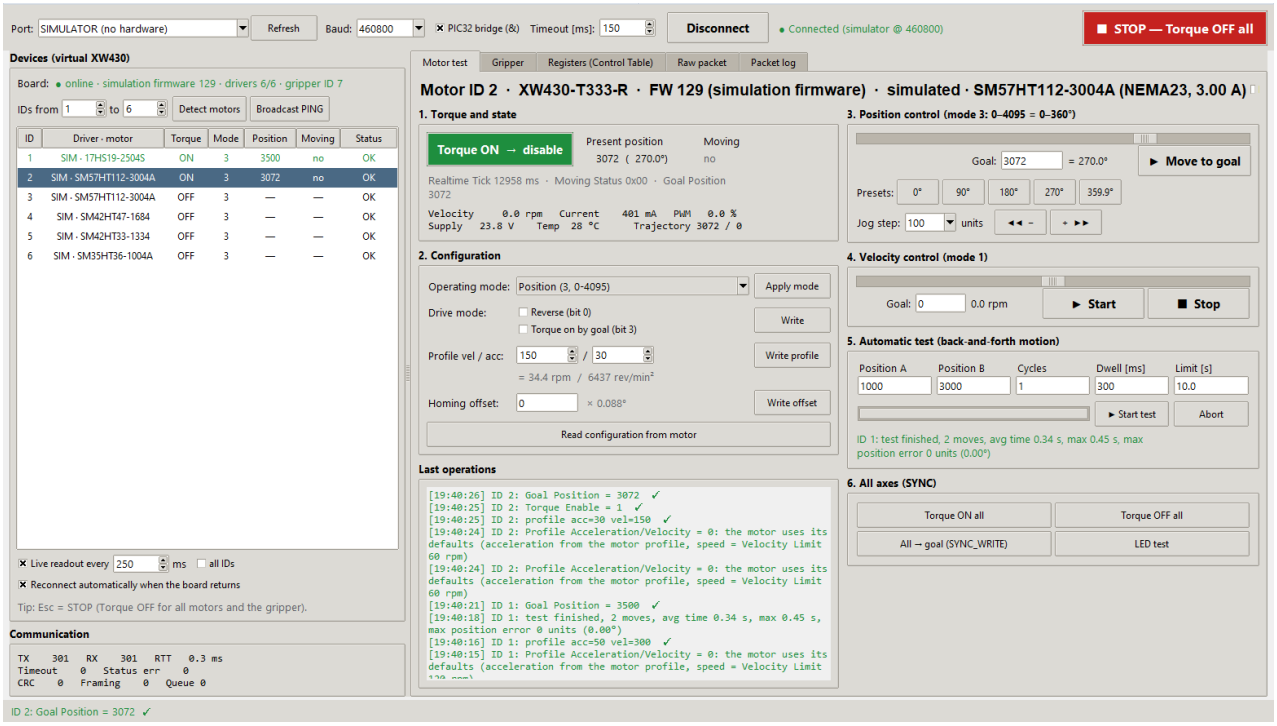
12.7 Messages you will see with the current firmware

Message	Cause
Present position *unavailable* / Access Error on Goal Position	The axis has no position scale (only in `--sim-uncalibrated` or a firmware built without motor profiles)
Access Error on Drive Mode, Homing Offset, Current Limit, Velocity Limit	Torque is ON; these registers are written with Torque OFF only
Hardware Error Status bit 0 and Torque OFF by itself	The TMC5160 lost VM and restarted; the firmware configured it again. Position restarted at 0: switch Torque ON again
Access Error on Goal Velocity write in mode 3	Wrong mode; Goal Velocity is written only in mode 1
Access Error on *Operating Mode*	Torque is ON
Data Length Error	The write did not start at a register address or did not match its width
Values in the wrong fields, one step behind (only with an old GUI)	The PIC32 held back a final `%` byte, see Host link; current `DxlBus` adds a `0x00` after such packets
Timeout on every ID	Wrong port or baud rate (kit USB port: 460800; try **Auto**), **PIC32 bridge (&)** unticked, no power, firmware not running, or a text console active on the UART
Baud Rate register shows 23, not 460800	Correct: the register reports the firmware UART (PIC32 ↔ SmartFusion2, 230400 bps; project code 23, 46 would be 460800). The PC port runs at 460800

12.8 Automated check

`firmware_check.py` runs the same protocol transactions automatically and prints PASS/FAIL. On the simulation firmware it runs 49 checks, including motion, telemetry and the gripper servo. On the target firmware it runs the protocol checks only; motion runs only with `--allow-motion`.

```
.\.venv\Scripts\python.exe tools\dxl_test_gui\firmware_check.py --port COM5
.\.venv\Scripts\python.exe tools\dxl_test_gui\firmware_check.py --sim     # Python
simulator
```



GUI with the simulator (FW 129, like the simulation firmware): telemetry line of axis 2 in section 1

12.9 Internals

File	Role
`dxl_gui.py`	Tkinter user interface. A single worker thread runs all bus transactions; the user interface never blocks.
`dxl/protocol.py`	Protocol 2.0 codec: CRC-16, byte stuffing, stream parser, packet descriptions
`dxl/bus.py`	Thread-safe master: PING, broadcast PING, READ, WRITE, REG_WRITE, ACTION, SYNC_READ, SYNC_WRITE; statistics and packet log callback
`dxl/control_table.py`	XW430 register subset, servo and controller registers, units, operating modes
`dxl/motors.py`	Reads `motor_profiles.h` and `motor_axes.h`, names the motors and detects a configuration mismatch
`dxl/sim.py`	Simulator with the same instructions, register rules, and error codes as `dynamixel_x430.c` and `dynamixel_servo.c` (both variants)
`tmc_diag.py`	TMC5160 bring-up diagnostics through the raw register window (IOIN/SD_MODE, CHOPCONF, DRV_STATUS, MSCNT), optionally during a move
`firmware_check.py`	Automated conformance check against a port or the Python simulator
`tests/test_protocol.py`	Unit tests, including the firmware codec cross-check

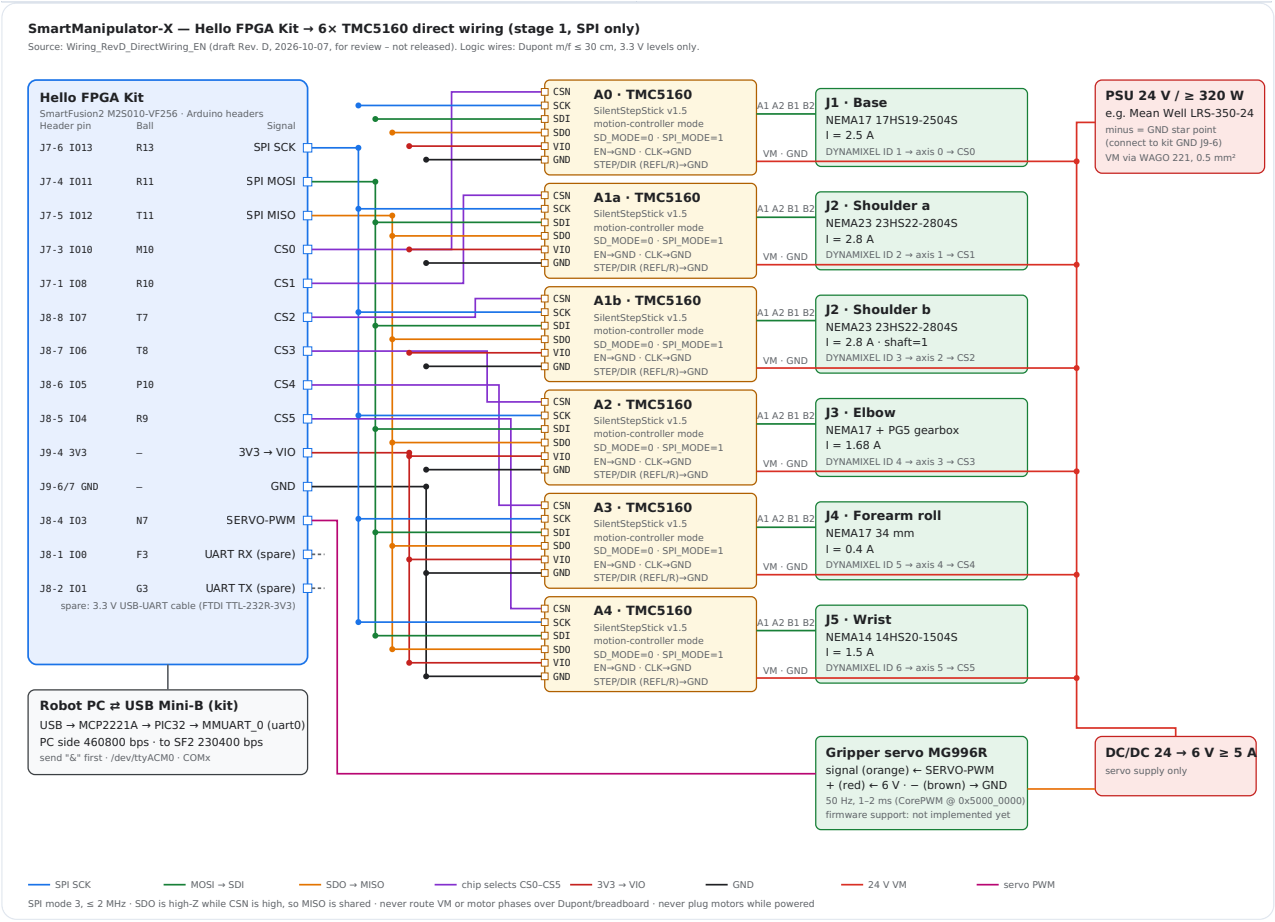
When the firmware's instructions, register rules, or error codes change, update `dxl/sim.py` and `dxl/control_table.py` to match, then run the tests.

CHAPTER 13

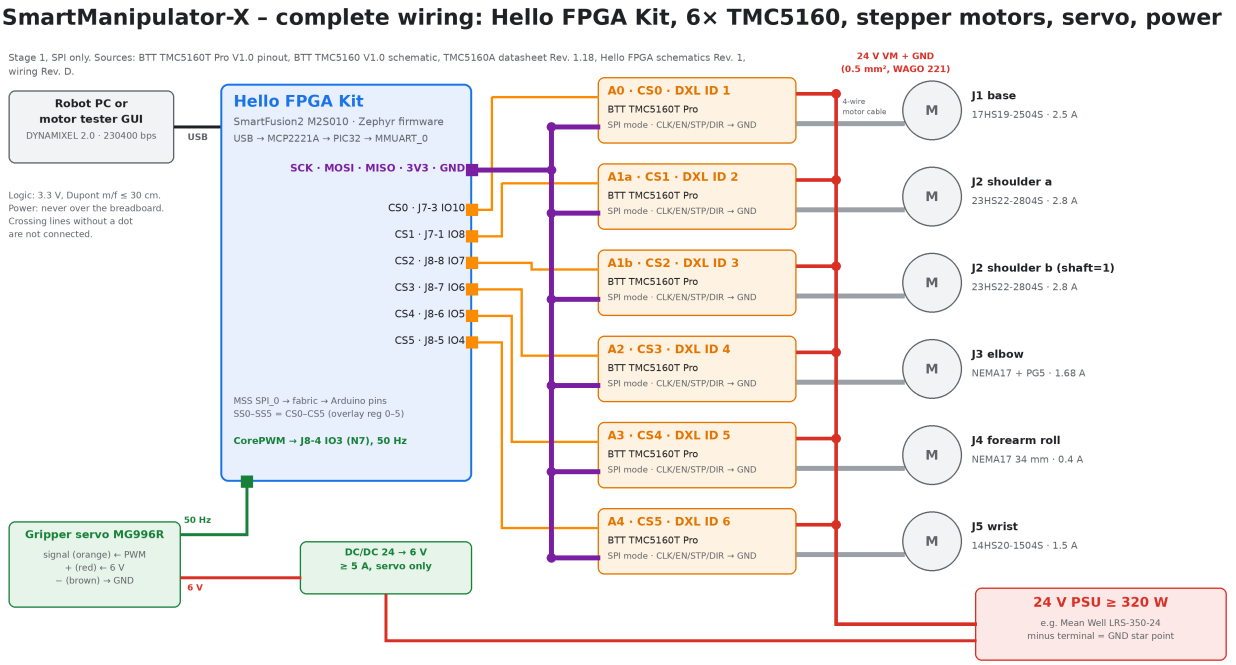
Hello FPGA Kit wiring

This page covers stage 1: SPI-only direct wiring of the M2S Hello FPGA Kit (SmartFusion2 M2S010-VF256) to six TMC5160 modules, the gripper servo, and the power supply.

> **Source and status:** *Robot Arm Demonstrator: Direct Wiring Hello FPGA Kit → 6× TMC5160 (Stage 1, SPI only)*, draft Rev. D, 2026-10-07, bavariamatic. It is **for review and not released**, and it supersedes Rev. C (which had home switches). The original is in `Wiring_RevD_DirectWiring_EN.pdf` (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/Wiring_RevD_DirectWiring_EN.pdf). Pin source: Hello FPGA schematics Rev. 1, p. 11. If this page and the PDF disagree, the PDF is authoritative; update this page to match it.
>
> **Manufacturer documents** (in `datasheets/` (p. 82)): TMC5160A datasheet Rev. 1.18 (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/TMC5160A_datasheet_rev1.18.pdf), Hello FPGA Kit schematics Rev. 1 (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/Hello_FPGA_Kit_schematics_rev1.pdf), BTT TMC5160T Pro pinout (global.bttwiki.com/TMC5160T%20Pro%20V1.0.html) (module in use), and the reference module BTT TMC5160 V1.0 schematic (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/BIGTREETECH_TMC5160-V1.0_schematic.pdf) and manual (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/BIGTREETECH_TMC5160-V1.0_manual.pdf).
>
> **Verified on hardware (2026-10-10/11):** host link (PIC32 ↔ SmartFusion2 at 230400 bps, see Host link (p. 64)); SCK, MOSI, MISO, CS0, 3V3 and GND to one TMC5160T Pro (axis A0 detected, registers and ramp generator work); the servo signal on J8-4 (the servo follows Goal Position). The A0 shaft does not turn yet because the module runs in STEP/DIR mode (finding 10 in TMC5160 driver modules (p. 80)). CS1–CS5 and the other motors are not tested yet.
>
> **Gripper servo:** connection, supply, signal level and bring-up are on Gripper servo (MG996R) (p. 83).



Hello FPGA Kit to 6× TMC5160 wiring diagram



Complete stage-1 wiring: Hello FPGA Kit, six TMC5160 modules, stepper motors, servo, and power

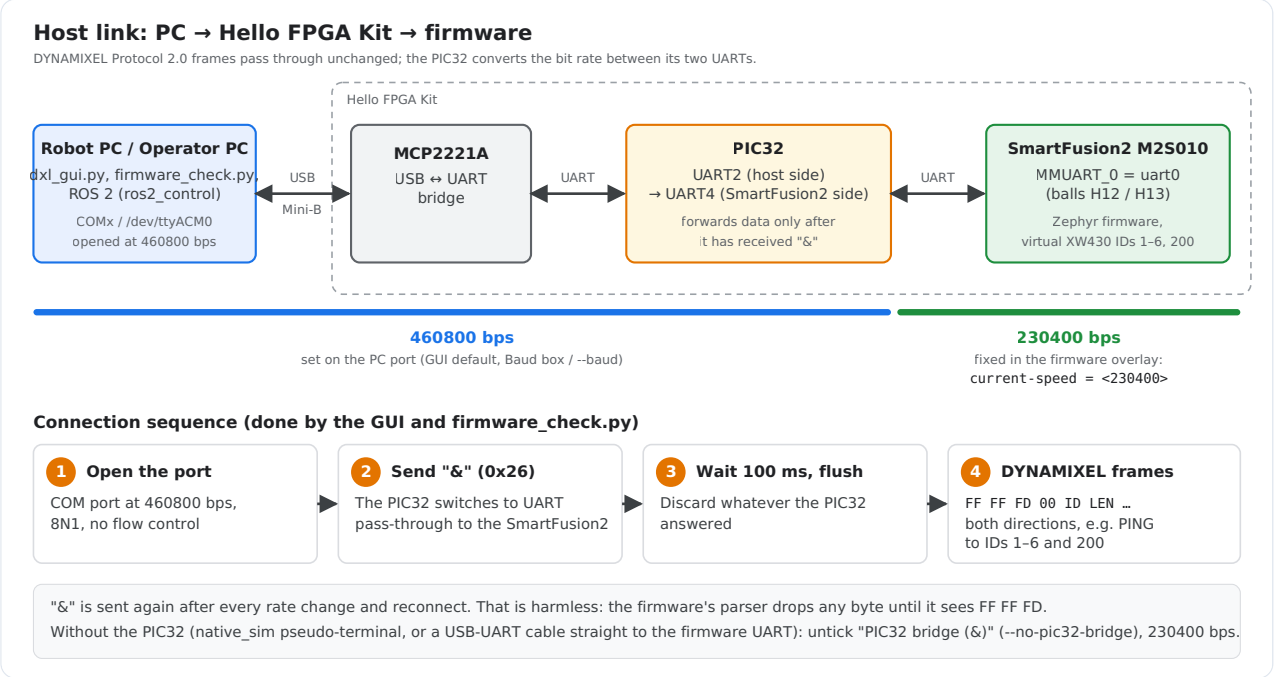
The module pinout, the motor connection, the check against the manufacturer documents, and the firmware review are on TMC5160 driver modules (p. 71).

13.1 Concept

- Each TMC5160 runs in **motion-controller mode** (`SD_MODE=0`, `SPI_MODE=1`). The ramps run inside the driver. The BTT TMC5160T Pro on A0 currently reads `SD_MODE=1` and must be switched (finding 10 in TMC5160 driver modules (p. 80)). The kit drives only **SPI plus six chip selects**; it provides no STEP/DIR, EN, or DIAG signals.
- **No limit switches** in stage 1, because the mechanics have none. As on the original Moveo, positioning is open-loop:
- With the power off, move the arm by hand to the defined start pose.
- At power-up, every TMC5160 starts with `XACTUAL = 0` (reset value), so the start pose is position 0. The firmware does not home.
- The joint ranges are enforced as software limits (firmware or `ros2_control`).
- **Host link:** USB Mini-B → MCP2221A → PIC32 → SmartFusion2 MMUART_0. The link carries DYNAMIXEL Protocol 2.0 and appears on the Robot PC as `/dev/ttyACM0`. PC side 460800 bps, firmware side 230400 bps; the PIC32 passes data only after the host has sent `&`. See Host link (p. 64).

13.2 Host link (USB → PIC32 → SmartFusion2)

The PC does not talk to the SmartFusion2 directly. The kit's USB Mini-B port leads to an MCP2221A USB-UART bridge, then to a PIC32, and only the PIC32 is wired to MMUART_0 of the SmartFusion2. The two UART segments run at **different bit rates**; the PIC32 converts between them.



Host link of the Hello FPGA Kit: bit rates of the two UART segments and the connection sequence

Segment	Bit rate	Where it is set
PC ↔ MCP2221A ↔ PIC32 (UART2)	**460800 bps**	Serial port on the PC: GUI *Baud* box (default 460800), `--baud` of `dxl_gui.py` and `firmware_check.py`
PIC32 (UART4) ↔ SmartFusion2 MMUART_0	**230400 bps**	Firmware: `current-speed` of `uart0` in both board overlays; the PIC32 firmware uses the same rate on its side

The PIC32 forwards nothing until it has received the character `&` (0x26). After opening the port, the host therefore:

1. opens the port at 460800 bps (8N1, no flow control),
2. sends `&`, which switches the PIC32 to UART pass-through,
3. waits about 100 ms and discards any bytes received so far,
4. sends DYNAMIXEL Protocol 2.0 frames (`FF FF FD 00 …`) in both directions.

The GUI and `firmware_check.py` do this automatically (checkbox **PIC32 bridge (&)**, on by default; `--no-pic32-bridge` turns it off). They send `&` again after every rate change and reconnect. That is harmless: the firmware's parser drops any byte until it sees the `FF FF FD` header. A ROS 2 driver or any other host program on the Robot PC must send `&` once after opening the port in the same way.

The PIC32 holds back a `%` (0x25) that ends a packet until the next byte arrives (measured on the kit, 2026-10-11: only this value, only towards the SmartFusion2, `%` inside a packet passes normally). A request whose CRC ends with 0x25 (for example READ 120/4, Realtime Tick + Moving, which the GUI polls) reached the firmware only together with the next request, so every reply arrived one transaction late. The host therefore sends one `0x00` byte after any packet that ends with 0x25; the firmware's parser drops it. `DxlBus` in `tools/dxl_test_gui/dxl/bus.py` does this, and also ignores a status packet whose data length does not match the request (a late reply). Any other host program must do the same.

Consequences:

- The Baud Rate register (address 8) of the virtual servos reports code 23 = 230400 bps, the firmware side. It does not show the 460800 of the PC port.
- Changing the PC rate does not change the firmware rate, and the reverse. `current-speed` in the overlays must stay at the rate the PIC32 uses towards the SmartFusion2 (230400).
- Without the PIC32 (firmware running as `native_sim`, or a USB-UART cable straight to the firmware UART), untick **PIC32 bridge (&)**. On a cable, use 230400 bps; on the `native_sim` pseudo-terminal the rate does not matter.

13.3 Wiring list (logic, Dupont m/f, ≤ 30 cm)

Kit pin	Arduino	Ball	Signal	Destination
J7-6	IO13	R13	SPI SCK	→ SCK of all 6 modules
J7-4	IO11	R11	SPI MOSI	→ SDI of all 6 modules
J7-5	IO12	T11	SPI MISO	← SDO of all 6 modules
J7-3	IO10	M10	CS0	→ CSN A0 (J1 base)
J7-1	IO8	R10	CS1	→ CSN A1a (J2 shoulder a)
J8-8	IO7	T7	CS2	→ CSN A1b (J2 shoulder b)
J8-7	IO6	T8	CS3	→ CSN A2 (J3 elbow)
J8-6	IO5	P10	CS4	→ CSN A3 (J4 forearm roll)
J8-5	IO4	R9	CS5	→ CSN A4 (J5 wrist)
J8-4	IO3	N7	SERVO-PWM	→ servo signal (orange), see Gripper servo

Kit pin	Arduino	Ball	Signal	Destination
J8-1	IO0	F3	UART RX	← TXD of the USB-UART (spare)
J8-2	IO1	G3	UART TX	→ RXD of the USB-UART (spare)
J9-4	3V3	–	VIO	→ VIO (J2-7) of all 6 modules
J9-6/7, J7-7	GND	–	GND	→ GND of the modules, servo, UART, and PSU minus

Use a Dupont splitter or a breadboard for the SPI bus to the six modules. **Logic signals only** go over the breadboard.

13.4 Axes and motors

Motor connection order

Connect the motors in this order. Each motor goes to its own TMC5160 module, and each module's CSN goes to the chip select in the same row. Swapping two motors gives them the wrong current and ramp, and swapping two CS lines moves the motor to another DYNAMIXEL ID.

No.	Module	CS (kit pin)	Joint	Motor	Rated current	DYNAMIXEL ID → firmware axis
1	A0	CS0 (J7-3)	J1 base	NEMA17 17HS19-2504S	2.5 A	ID 1 → axis 0
2	A1a	CS1 (J7-1)	J2 shoulder (a)	NEMA23 23HS2430	3.0 A	ID 2 → axis 1
3	A1b	CS2 (J8-8)	J2 shoulder (b)	NEMA23 23HS2430	3.0 A, `shaft=1`	ID 3 → axis 2
4	A2	CS3 (J8-7)	J3 elbow	NEMA17 17HS19-1684S-PG5 (gearbox 5.18:1)	1.68 A	ID 4 → axis 3
5	A3	CS4 (J8-6)	J4 forearm roll	NEMA17 17HS13-0404S1	0.4 A	ID 5 → axis 4
6	A4	CS5 (J8-5)	J5 wrist	NEMA14 14HS20-1504S	1.5 A	ID 6 → axis 5

- Both shoulder motors (rows 2 and 3) are the same type and drive the same joint. A1b faces A1a, so it runs mirrored: set `SMX_AXIS2_INVERT` to 1 once the mechanics confirm it (see Implications for the firmware (p. 70)).
- The J3 elbow motor has a built-in planetary gearbox (5.18:1, at most 2 N·m at the output). `motor_axes.h` sets the gear ratio 57/11 on axis 3, so the position of ID 4 refers to the gearbox output shaft, not the motor shaft.
- The 17HS13-0404S1 (J4) is a 0.4 A motor. Never connect it to a module that runs a profile for a larger motor: it would get several times its rated current and overheat.
- Coil wiring: one coil to A1/A2, the other to B1/B2. The wire colours differ between the motor types (the 23HS2430 has red A+ and **yellow B+**), see Wire colours of the other motors (p. 75). Never plug or unplug a motor while VM is on.

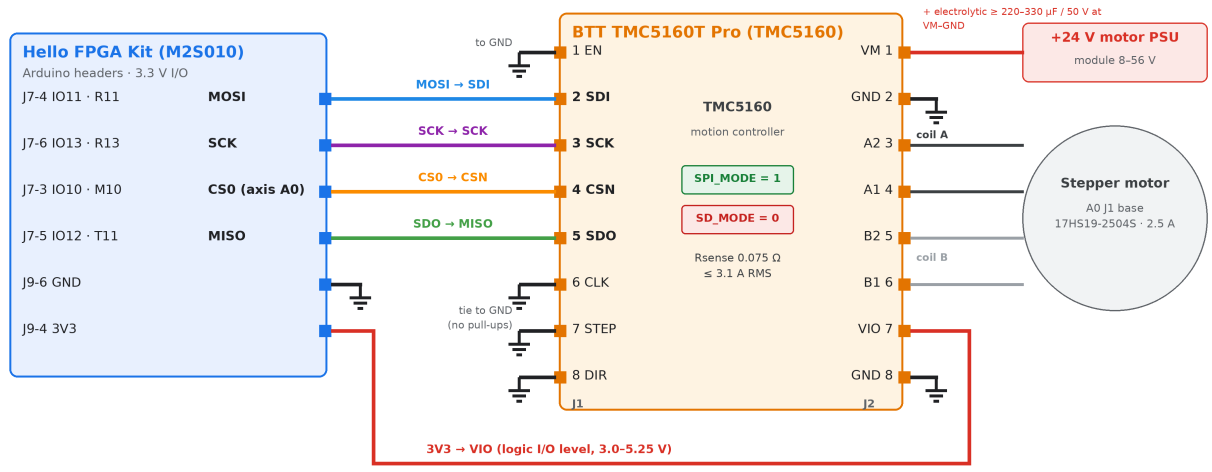
The ID → axis → CS mapping comes from `main.c` (IDs 1-6) and the overlay (`reg = <0>…<5>`). Which motor sits on which axis is set in `apps/SmartManipulator_Controller/include/motor/motor_axes.h`; the motor data (current, ramp) is in `motor_profiles.h` next to it. See Motor profiles and axis assignment (p. 78). The table shows the current assignment. The earlier planned motors (SM57HT112-3004A, SM42HT47-1684, SM42HT33-1334, SM35HT36-1004A) remain in `motor_profiles.h` as alternatives; they are not assigned to any axis.

13.5 Fixed wiring on each TMC5160 module

The modules are **BIGTREETECH TMC5160T Pro** (R_{SENSE} 0.075 Ω, VM 8–56 V, SPI mode by default). Rev. D used Watterott SilentStepStick pin names; the BTT names and header pins are below. Full pinout and the differences to the BTT TMC5160 V1.0: TMC5160 driver modules (p. 71).

Hello FPGA Kit ↔ TMC5160 module (BTT TMC5160T Pro) – SPI wiring of one axis

Axis A0 shown. All six modules share SCK, MOSI, MISO, 3V3 and GND; each has its own chip select. Sources: BTT TMC5160T Pro V1.0 pinout, BTT TMC5160 V1.0 schematic, TMC5160A datasheet Rev. 1.18, Hello FPGA schematics Rev. 1.



- Mode: SPI_MODE = 1, SD_MODE = 0 (internal ramp generator) – default on the TMC5160T Pro (BTT TMC5160 V1.0: SPI pad closed, SD pad open).
- EN, CLK, STEP, DIR have no pull resistors on the module: tie them to GND with short wires.
- SPI mode 3, SCK ≤ 2 MHz over ≤ 30 cm Dupont wires; SDO is tristate while CSN is high, so all modules share MISO.
- Other axes: identical, only CS changes – CS1 J7-1 IO8, CS2 J8-8 IO7, CS3 J8-7 IO6, CS4 J8-6 IO5, CS5 J8-5 IO4.
- TMC5160 logic is powered from VM: switch 24 V on before the kit, and use one common GND (star point at PSU −).

One TMC5160T Pro axis wired to the Hello FPGA Kit

Module pin	Connection
J1-4 CSN (CFG3)	its own CS line (table above)
J1-3 SCK (CFG2), J1-2 SDI (CFG1), J1-5 SDO (CFG0)	shared SPI bus
J2-7 VIO	3V3 (J9-4)
J2-8 GND	common GND
J1-1 EN	→ GND (must not float). The power stage stays off after reset (`CHOPCONF.TOFF=0`) until the firmware enables it over SPI.
J1-6 CLK	→ GND, short wire (internal 12 MHz clock). A floating CLK prevents detection.
J1-7 STP, J1-8 DIR (= REFL, REFR)	→ GND; `SW_MODE` stops disabled

Module pin	Connection
J2-1 VM, J2-2 GND (power)	24 V via WAGO 221, 0.5 mm² stranded wire soldered to the module pins
VM bulk capacitor	electrolytic at VM/GND of every module: ≥ 330 µF / 50 V low-ESR on the 3 A axes, ≥ 220 µF / 50 V on the others
J2-3/4 A2 A1, J2-5/6 **B2 B1**	motor phases: coil A on A1/A2, coil B on B1/B2. Note the order B2, B1.

Motor wiring for the 17HS19-2504S (black A+ → A1, green A− → A2, red B+ → B1, blue B− → B2): TMC5160 driver modules (p. 75). For a BTT TMC5160 V1.0 the headers are J3/J4, coil B is B1, B2, and the SPI pad must be closed and the SD pad open.

13.6 Power and servo

Item	Specification
Motor supply	24 V / ≥ 320 W, for example Mean Well LRS-350-24. The minus terminal is the GND star point; connect it to the kit GND (J9-6).
Servo supply	DC/DC 24 → 6 V, ≥ 5 A, for the servo only (MG996R: 4.8–7.2 V, stall current 2.5 A at 6 V). Its minus goes to the GND star point
Gripper servo	MG996R: signal (orange) ← SERVO-PWM (J8-4, IO3/N7, 50 Hz, 3.3 V), + (red) ← 6 V, − (brown) → DC/DC minus. Never from the kit's 5 V or 3V3. Details: Gripper servo (MG996R)
Spare UART	3.3 V USB-UART cable (for example FTDI TTL-232R-3V3): TXD → J8-1, RXD ← J8-2, GND → J9-7. Connect it only if the host link through the PIC32 is not enough.

13.7 Kit header reference

J8	Arduino	Ball	Use
1	IO0 / RX	F3	UART RX (spare)
2	IO1 / TX	G3	UART TX (spare)
3	IO2	P6	–
4	IO3 ~	N7	SERVO-PWM
5	IO4	R9	CS5 → A4
6	IO5 ~	P10	CS4 → A3
7	IO6 ~	T8	CS3 → A2
8	IO7	T7	CS2 → A1b

J7	Arduino	Ball	Use
1	IO8	R10	CS1 → A1a
2	IO9 ~	P9	–
3	IO10 / SS	M10	CS0 → A0

J7	Arduino	Ball	Use
4	IO11 / MOSI	R11	SPI MOSI
5	IO12 / MISO	T11	SPI MISO
6	IO13 / SCK	R13	SPI SCK
7	GND	–	GND
8	AREF	–	do not use
9	IO14 / SDA	P8	– (I²C later)
10	IO15 / SCL	R8	– (I²C later)

J9	Pin	Use
1	NC	–
2	IOREF	3.3 V (I/O level)
3	RESET (R12)	do not use
4	3V3	VIO of the drivers
5	5V	do not use
6, 7	GND	GND
8	VIN	NC

The Arduino headers are female sockets, so use m/f Dupont jumpers. The I/O level is 3.3 V (IOREF = 3P3V). Arduino A0–A5 are connected to the PIC32; **do not use them**.

13.8 Safety rules

1. Send only 3.3 V signals to the kit. Leave J9-5 (5 V) and J9-3 (RESET) unconnected. Do not power the servo from the kit, and do not pull the servo signal up to its supply.
2. Never route VM or motor phases over a breadboard or Dupont wires. Never plug or unplug a motor while the power is on.
3. SPI runs in mode 3 at no more than 2 MHz (Dupont wires). SDO is high-Z while CSN is high, so the six modules can share MISO.
4. Configure the shoulder modules A1a and A1b identically, and write their target positions back-to-back.
5. Do not enable the TMC5160 power stages until the firmware parameters are verified (next section).
6. Switch on the 24 V motor supply **before** the kit. The TMC5160 logic runs from VM; if VM comes later or drops, the chips lose their configuration. The firmware detects this (GSTAT.reset) before the next command, writes the configuration again and switches torque off; the position restarts at 0. Place the arm in the start pose again before you switch torque on.

13.9 Implications for the firmware

Wiring fact	Firmware or Libero impact	Status
CS0–CS5 on IO10, IO8, IO7, IO6, IO5, IO4	Overlay `reg = <0>…<5>` = SPI_0 SS0–SS5. This matches the Libero contract (`docs/firmware_interface.md` in the Libero repo).	Consistent
SPI mode 3, ≤ 2 MHz	Mode 3 is correct. The MSS SPI driver cannot divide below PCLK/30 = 1.67 MHz: the old 1 MHz setting fell back to **25 MHz** SCK.	Fixed: `SMX_TMC5160_SPI_HZ` = 2 MHz (1.79 MHz); verified on hardware
CLK → GND (internal oscillator, 12 MHz)	`clock-frequency` was 16 MHz	Fixed: `SMX_TMC5160_CLOCK_HZ` = 12 MHz
No StallGuard2 tuning	The pinned driver always sets `SW_MODE.sg_stop`; above TCOOLTHRS an untuned StallGuard2 can stop an axis	Fixed: `SMX_TMC5160_TCOOLTHRS` = 0
Per-axis currents of 0.4–3.0 A	`irun`, `ihold` and the ramp of each axis come from the motor profile in `motor_profiles.h`	Fixed (table in TMC5160 driver modules)
A1b runs mirrored (`shaft=1`)	`SMX_AXIS2_INVERT` in `motor_axes.h` adds `shaft;` to `tmc5160_axis2`; it is still 0	**Set to 1** once the mechanics confirm the motors face each other
TMC5160 logic powered from VM	After a VM power-up or dropout the chip has reset values	Fixed: `motor_tmc5160.c` reads GSTAT before every command and writes the configuration again after a reset (Torque OFF, Hardware Error bit 0, position 0). Still switch 24 V on first
EN → GND, `TOFF=0` after reset	The power stage starts only once the firmware enables it over SPI (Torque ON → `stepper_enable`)	As designed
No switches; `XACTUAL = 0` at power-up; software limits	No homing. Limits must be enforced in firmware or `ros2_control`. Position scale is 1:1 to the motor shaft (200 × 16 = 3200 microsteps per revolution) until the gear ratios are entered in `motor_axes.h`. The J3 gearbox (57/11) is entered; the joint transmissions are not.	**Open** (limits, joint gear ratios)
Shoulder = two motors (A1a/A1b, IDs 2 and 3)	The host must command both IDs; `SYNC_WRITE` is sequential, not synchronized	Open (consider pairing in firmware)
MSS SPI_0 / MMUART_0 pads are hard-wired to the PIC32 (wiring note 5)	In Libero, route the MSS SPI and UART through the fabric to the Arduino pins, or use CoreSPI/CoreUART. Stage 2 of the Libero design routes SPI_0 through the fabric and keeps MMUART_0 as the host link through the PIC32 (fabric-routed to H12/H13).	Consistent (Libero design checked 2026-10-09; SPI_0 routing and the host link confirmed on hardware 2026-10-10)
Servo on IO3 / N7, 50 Hz	Libero CorePWM at `0x5000_0000` (PRESCALE 99, PERIOD 19999, 1–2 ms pulse)	Firmware: `src/servo_corepwm.c`, DYNAMIXEL ID 7 (Gripper servo); works on the kit (2026-10-11). MG996R datasheet: ~120° for 1–2 ms, the firmware default scale is 90° until calibrated (Calibration)

CHAPTER 14

TMC5160 driver modules

This page checks the wiring concept and the firmware's TMC5160 interface against the manufacturer documents, records what has been verified on the hardware, and lists what is still open before the motors are driven.

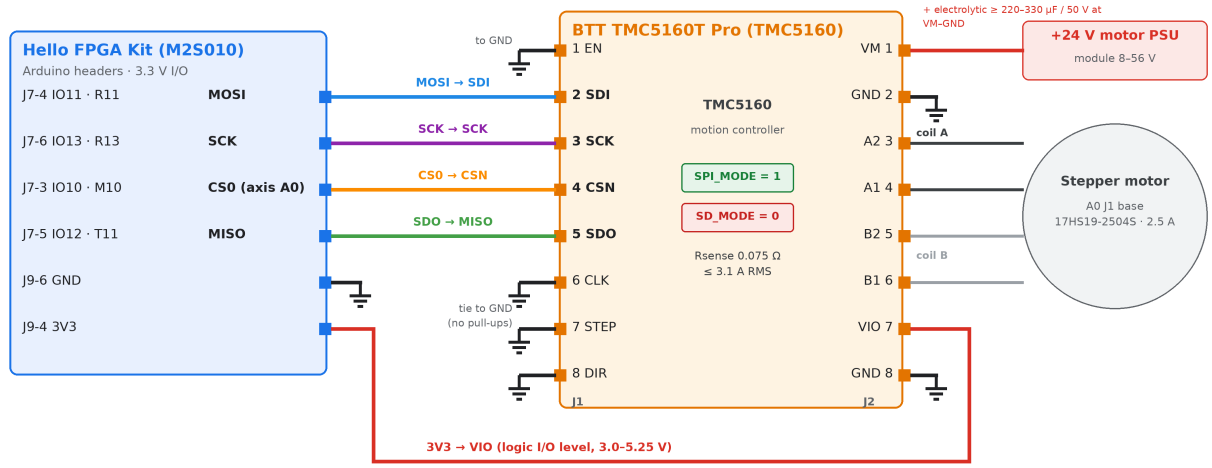
> **Hardware status (2026-10-11):** one BTT TMC5160T Pro (axis A0, motor 17HS19-2504S) is wired to the kit and detected over SPI (`IOIN.VERSION` = 0x30) by the target firmware through the GUI (460800 bps to the PIC32, 230400 bps to the SmartFusion2). Detection needed SPI 2 MHz in the overlay and CLK tied to GND. Registers, telemetry, Current Limit, the ramp generator and mode changes work on A0, but **the shaft does not turn**: the module reads `SD_MODE=1` (STEP/DIR mode), see finding 10. Currents and ramps come from per-motor profiles (`include/motor/motor_profiles.h`).

Source	Revision	File
BIGTREETECH TMC5160T Pro V1.0 pinout (module in use)	BTT wiki, checked 2026-10-10; module silkscreen	bttwiki.com (global.bttwiki.com/TMC5160T%20Pro%20V1.0.html)
ADI Trinamic TMC5160/TMC5160A datasheet	Rev. 1.18, 2023-03-01	`TMC5160A_datasheet_rev1.18.pdf` (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/TMC5160A_datasheet_rev1.18.pdf)
Microchip Hello FPGA schematics (DVP-100-000518-001)	Rev. 1, dated 02-07-2019 on the schematic	`Hello_FPGA_Kit_schematics_rev1.pdf` (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/Hello_FPGA_Kit_schematics_rev1.pdf)
17HS19-2504S motor datasheet (wire colours)	–	pin 1 BLK A+, 2 GRN A−, 3 RED B+, 4 BLU B−
BIGTREETECH TMC5160 V1.0 schematic (`TMC5160.SchDoc`), reference module	2019-03-27	`BIGTREETECH_TMC5160-V1.0_schematic.pdf` (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/BIGTREETECH_TMC5160-V1.0_schematic.pdf)
BIGTREETECH TMC5160 V1.0 manual, reference module	V1.0	`BIGTREETECH_TMC5160-V1.0_manual.pdf` (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/BIGTREETECH_TMC5160-V1.0_manual.pdf)
Direct wiring Hello FPGA Kit → 6× TMC5160	Rev. D draft, 2026-10-07	`Wiring_RevD_DirectWiring_EN.pdf` (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/Wiring_RevD_DirectWiring_EN.pdf)
Zephyr `adi,tmc51xx` and `microchip,mpfs-spi` drivers	v4.5.0-rc1 (pinned in `west.yml`)	source in the west workspace

The manufacturer PDFs are kept unchanged in `docs/hardware/datasheets/` (see Reference documents (p. 82)). The copyright stays with the manufacturers. No schematic of the TMC5160T Pro is in the repository; where this page needs board-level details (pull resistors, capacitors), it uses the BTT TMC5160 V1.0 schematic as the closest reference and says so.

Hello FPGA Kit ↔ TMC5160 module (BTT TMC5160T Pro) – SPI wiring of one axis

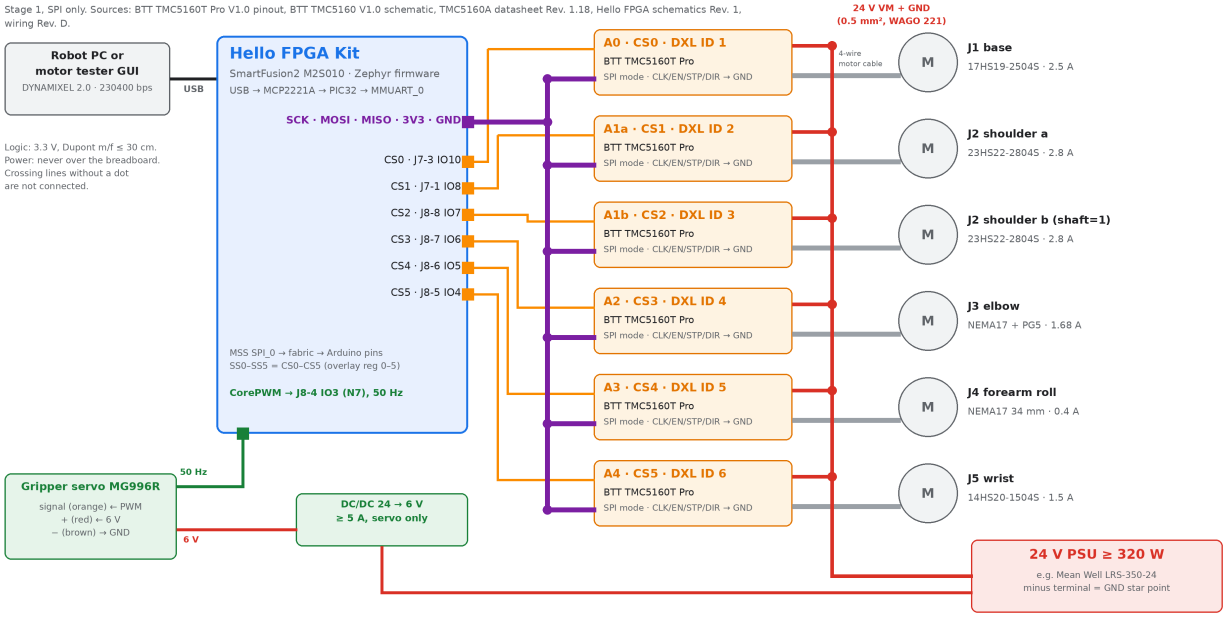
Axis A0 shown. All six modules share SCK, MOSI, MISO, 3V3 and GND; each has its own chip select. Sources: BTT TMC5160T Pro V1.0 pinout, BTT TMC5160 V1.0 schematic, TMC5160A datasheet Rev. 1.18, Hello FPGA schematics Rev. 1.



- Mode: SPI_MODE = 1, SD_MODE = 0 (internal ramp generator) – default on the TMC5160T Pro (BTT TMC5160 V1.0: SPI pad closed, SD pad open).
- EN, CLK, STEP, DIR have no pull resistors on the module: tie them to GND with short wires.
- SPI mode 3, SCK ≤ 2 MHz over ≤ 30 cm Dupont wires; SDO is tristate while CSN is high, so all modules share MISO.
- Other axes: identical, only CS changes – CS1 J7-1 IO8, CS2 J8-8 IO7, CS3 J8-7 IO6, CS4 J8-6 IO5, CS5 J8-5 IO4.
- TMC5160 logic is powered from VM: switch 24 V on before the kit, and use one common GND (star point at PSU −).

One BTT TMC5160T Pro axis wired to the Hello FPGA Kit (axis A0 shown; the other axes differ only in the CS line)

SmartManipulator-X – complete wiring: Hello FPGA Kit, 6× TMC5160, stepper motors, servo, power



Complete stage-1 wiring: Hello FPGA Kit, six TMC5160 modules, stepper motors, servo, and power

14.1 The module in use: BIGTREETECH TMC5160T Pro

The modules fitted are the **BTT TMC5160T Pro** (silkscreen "TMC5160T pro"): TMC5160 with external MOSFETs, R_{SENSE} 0.075 Ω, 3.1 A RMS / 4.4 A peak, VM 8–56 V. BTT lists "Operating Mode: SPI" and STP/DIR as "Pulse Input" / "Direction Input": the module is made for 3D printer boards (Marlin), i.e. TMC5160 **mode 2**, SPI for configuration plus STEP/DIR for the steps (SD_MODE = 1). This project needs **mode 1**, the internal motion controller (SD_MODE = 0); see finding 10.

TMC5160 mode (datasheet ch. 1)	SD_MODE	SPI_MODE	Steps from
1 Motion controller and driver (this project)	0	1	internal ramp generator (XTARGET over SPI)
2 Step and direction driver (BTT "SPI mode")	1	1	STEP/DIR pins; SPI only configures
3 Simple step and direction	1	0	STEP/DIR pins; no SPI

J1 pin	Signal	Connection	J2 pin	Signal	Connection
1	EN	→ GND	1	VM	motor supply (+)
2	CFG1 / SDI	← kit MOSI	2	GND	motor supply (−), common GND
3	CFG2 / SCK	← kit SCK	3	A2	coil A
4	CFG3 / CSN	← kit CS*n*	4	A1	coil A
5	CFG0 / SDO	→ kit MISO (tristate while CSN is high)	5	**B2**	coil B
6	CLK	→ GND, short wire (internal 12 MHz clock)	6	**B1**	coil B
7	STP (= REFL)	→ GND	7	VIO	← kit 3V3 (J9-4)
8	DIR (= REFR)	→ GND	8	GND	common GND

- EN is the pin in the white-marked box. DIAG0, DIAG1, ENCB, ENCA and ENCN (top pads) are not used.
- EN, CLK, STP and DIR must not float. On the BTT TMC5160 V1.0 these pins have no pull resistors, and the TMC5160 has none on CLK; a floating CLK prevented detection during the first test. Tie all four to GND.
- Coil B is in the order **B2, B1** (the V1.0 has B1, B2). Wire motors by pin name, not by position.

Reference module: BIGTREETECH TMC5160 V1.0

The datasheets folder holds the schematic and manual of the older BTT TMC5160 V1.0. It has the same logic header, but:

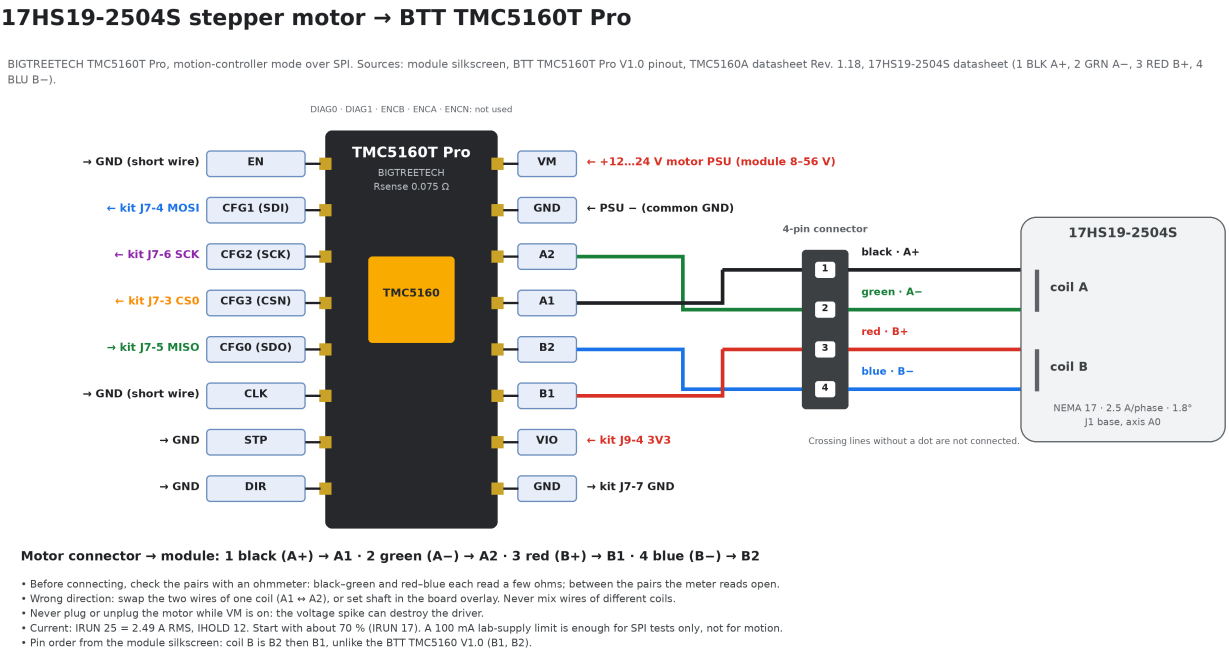
	BTT TMC5160 V1.0	BTT TMC5160T Pro (in use)
Headers	J3 (logic), J4 (power)	J1 (logic), J2 (power)
Logic header	EN, SDI, SCK, CSN, SDO, CLK, STEP, DIR	same order
Power header	VM, GND, A2, A1, **B1, B2**, VCC_IO, GND	VM, GND, A2, A1, **B2, B1**, VIO, GND
VM	8–40 V (FDMC8030 40 V MOSFETs)	8–56 V
R_{SENSE}, current	0.075 Ω, 3.1 A RMS / 4.4 A peak	same (same current table)
VM capacitors	2 × 10 µF + 470 nF	not documented
Extra pads	DIAG0, DIAG1	DIAG0, DIAG1, ENCB, ENCA, ENCN
Mode selection	solder pads, see below	SPI mode by default

If a V1.0 module is used, check its mode pads before wiring:

Pad	Schematic	Connects	Required
SPI	W2	SPI_MODE (22) to VCC_IO. Open = internal pull-down = standalone mode, SPI is off.	**Closed**
SD	W1	SD_MODE (21) to VCC_IO. Open = 10 kΩ pull-down R5 = internal motion controller.	**Open**

With **SD** closed, the chip expects STEP/DIR pulses and the firmware's ramp commands do nothing. With **SPI** open, the chip does not answer on SPI and the firmware reports the axis as absent.

14.2 Motor connection: 17HS19-2504S



17HS19-2504S stepper motor wired to the BTT TMC5160T Pro (axis A0)

Motor connector pin	Wire	Phase (motor datasheet)	Module pin
1	black (BLK)	A+	A1
2	green (GRN)	A−	A2
3	red (RED)	B+	B1
4	blue (BLU)	B−	B2

- Coil A is black/green, coil B is red/blue. Check with an ohmmeter before you connect: each pair reads a few ohms, and between the pairs the meter reads open.
- Wrong direction: swap the two wires of **one** coil (A1 ↔ A2), or set `SMX_AXISn_INVERT` to 1 in `motor_axes.h` (the overlay then sets `shaft`). Never mix wires of different coils.
- Never plug or unplug the motor while VM is on.
- Rated current 2.5 A. The motor profile `M17HS19_2504S` sets IRUN 21 (2.1 A RMS, about 85 %) and IHOLD 12 (1.25 A). A lab supply limited to 100 mA is enough for SPI tests only, not for motion.
- The other motors use different colour codes. The rule is always "one coil to A1/A2, the other to B1/B2". The connection order of all six motors is in Motor connection order (p. 66).

Wire colours of the other motors

Axis	Motor	A+ → A1	A− → A2	B+ → B1	B− → B2	Source
A1a, A1b	23HS2430	red	green	**yel-low**	blue	RATTMMOTOR datasheet

Axis	Motor	A+ → A1	A− → A2	B+ → B1	B− → B2	Source
A2	17HS19-1684S-PG5	black	green	red	blue	Seller page (silnikkrokowy.pl)
A3	17HS13-0404S1	black	green	red	blue	StepperOnline standard, not checked against a datasheet
A4	14HS20-1504S	black	green	red	blue	StepperOnline standard, not checked against a datasheet

- **23HS2430: red is A+, not B+.** On this motor coil A is red/green and coil B is yellow/blue. Wiring it with the 17HS19 colour code puts red on B1 and splits the coils.
- Before you connect a motor, check the pairs with an ohmmeter: 1.6 Ω per coil on the 23H-S2430, about 1.7 Ω on the 17HS19-1684S-PG5, 30 Ω on the 17HS13-0404S1 and 2.8 Ω on the 14HS20-1504S.

14.3 Design assumptions compared with the sources

Item	Project assumption (Rev. D)	Manufacturer documents	Result
Operating mode	Motion controller, `SD_MODE=0`, `SPI_MODE=1`	TMC5160T Pro: "Operating Mode: SPI" (BTT wiki) = SPI + STEP/DIR, mode 2; the wiki documents no SD/SPI selection. V1.0 schematic: jumpers W1 (SD_MODE) and W2 (SPI_MODE)	**Not met on A0** (2026-10-11): IOIN reads `SD_MODE=1`, the module runs SPI + STEP/DIR. SPI works (detection), but the internal ramp generator does not drive the motor. See finding 10
Module	Pin names from the Watterott SilentStepStick	BTT TMC5160T Pro: J1/J2, VIO, coil B order B2, B1	Wiring page updated
Logic level	3.3 V from kit J9-4 to VIO	VCC_IO (VIO) 3.00–5.25 V; all digital pins use this level	OK; verified on hardware
Logic supply	TMC5160 logic is powered from VM	VCC_IO "does not supply internal logic"; VCC comes from 5VOUT, which comes from VSA = VM	OK; see power-up order below
Motor supply	24 V	TMC5160T Pro 8–56 V (V1.0: 8–40 V); chip 10–55 V for full specification	OK
Bulk capacitance	Not specified	Datasheet: at least 100 µF per ampere of coil current near the power stage; the V1.0 has only 2 × 10 µF ceramics; the T Pro values are not documented	**Add an electrolytic per module** (≥ 330 µF / 50 V low-ESR for the 3 A axes, ≥ 220 µF for the others)
Motor current	1.0–3.0 A	0.075 Ω sense resistor: 3.06 A RMS full scale (3.1 A RMS / 4.4 A peak per BTT)	OK; all motors are below the module limit. The 3 A NEMA23 axes run near it (IRUN 26 = 2.6 A): keep a heatsink and airflow on those modules

Item	Project assumption (Rev. D)	Manufacturer documents	Result
EN	→ GND	DRV_ENN, no pull resistor; low = power stage enabled. CHOPCONF.TOFF is 0 after reset, so the bridges stay off until the firmware enables them	OK; **must not float**
CLK	→ GND, internal clock	"Tie to GND using short wire for internal clock"; internal oscillator 12.0 MHz typ. (11.5–12.5 MHz); no pull-down on CLK	OK; must not float. Overlay fixed to 12 MHz
STEP, DIR	→ GND, stops disabled	In motion-controller mode these are the left/right reference inputs	OK
ENCN_DCO (pin 26)	–	"With SD_MODE=0, pull to GND or VCC_IO if not used"; not connected on the BTT V1.0; ENCN is a free pad on the T Pro	Low risk: only matters if the encoder interface is enabled (it is not)
SPI	Mode 3, ≤ 2 MHz, shared MISO	Mode 3, 40-bit datagrams; $f_{SCK} \leq 4$ MHz with the internal clock; SDO tristate while CSN high	OK; overlay fixed to 2 MHz (1.79 MHz SCK), verified on hardware
CS mapping	IO10, IO8, IO7, IO6, IO5, IO4 → SS0–SS5	Kit balls M10, R10, T7, T8, P10, R9 match the schematic	OK

Kit pins checked against the Hello FPGA schematic

Signal	Header	Arduino	Ball	Schematic net
SCK	J7-6	IO13	R13	`MSIO118NB4_R13_ARD_SCK/IO13`
MOSI	J7-4	IO11	R11	`MSIO115NB4_R11_ARD_PWM_MOSI/IO11`
MISO	J7-5	IO12	T11	`MSIO115PB4_T11_ARD_MISO/IO12`
CS0 → A0	J7-3	IO10	M10	`MSIO114NB4_M10_ARD_PWM_SS_IO10`
CS1 → A1a	J7-1	IO8	R10	`MSIO113NB4_R10_ARD_IO8`
CS2 → A1b	J8-8	IO7	T7	`MSIO110PB4_T7_ARD_IO7`
CS3 → A2	J8-7	IO6	T8	`MSIO110NB4_T8_ARD_PWM_IO6`
CS4 → A3	J8-6	IO5	P10	`MSIO109PB4_P10_ARD_PWM_IO5`
CS5 → A4	J8-5	IO4	R9	`MSIO106NB4_R9_ARD_IO4`
Servo PWM	J8-4	IO3	N7	`MSIO104NB4_N7_ARD_INT1_PWM_IO3`
3.3 V	J9-4	3V3	–	`3P3V` (MCP1727 LDO, 1.5 A, from USB 5 V)
GND	J9-6, J9-7, J7-7	GND	–	`GND`

The MSS SPI_0 and MMUART_0 pads are wired to the PIC32 (SPI_0 on J12–J14/K12; the host UART on H12/H13, `UART_TX_SF2` / `UART_RX_SF2`). The Libero design therefore routes SPI_0 through the fabric to the Arduino pins, as assumed. The J2 ICSP header shares R11, T11, R13 and RESET but is not fitted.

SCK, MOSI, MISO, CS0, 3V3 and GND were confirmed on the hardware on 2026-10-10 (axis A0 detected). The servo pin (N7) works since 2026-10-11 (the servo follows Goal Position). CS1–CS5 are not tested yet.

The host link runs USB → MCP2221A → PIC32 (UART2) → PIC32 (UART4) → SF2. The PC side runs at 460800 bps, the PIC32 → SmartFusion2 side at 230400 bps (works on the hardware, 2026-10-10, GUI talking to the target firmware; Microchip's own Hello FPGA demo uses the same rate on this path). The PIC32 forwards data only after the host has sent `&`. See Host link (p. 64).

14.4 Motor current settings

With GLOBALSCALER = 0 (full scale, the reset value) and R_{SENSE} = 0.075 Ω:

I_{RMS} = (CS + 1) / 32 × 0.325 V / 0.075 Ω / √2 = (CS + 1) × 0.0958 A

These values are set in the motor profiles (`motor_profiles.h`):

Axis	Motor profile	Rated	IRUN	IHOLD
A0 J1 base	`M17HS19_2504S`	2.5 A	21 → 2.11 A	12 → 1.25 A
A1a, A1b J2 shoulder	`M23HS2430`	3.0 A	26 → 2.59 A	15 → 1.53 A
A2 J3 elbow	`M17HS19_1684S_PG5`	1.68 A	14 → 1.44 A	8 → 0.86 A
A3 J4 forearm roll	`M17HS13_0404S1`	0.4 A	3 → 0.38 A	1 → 0.19 A
A4 J5 wrist	`M14HS20_1504S`	1.5 A	12 → 1.24 A	7 → 0.77 A

IRUN is about 85 % of the rated current, IHOLD about 50 %. The 0.4 A 17HS13-0404S1 is the exception: the nearest steps are CS 2 (0.29 A, 72 %) and CS 3 (0.38 A, 96 %), and the profile uses CS 3. With such small CS values the current resolution is coarse (the datasheet recommends CS ≥ 16); a lower GLOBALSCALER on that module would fix it but is not implemented. Raise IRUN only if an axis loses steps and the motor stays below about 80 °C.

14.5 Motor profiles and axis assignment

The motor data lives in two headers in `apps/SmartManipulator_Controller/include/motor/`:

- `motor_profiles.h`: one block of `MOTOR_<KEY>_<FIELD>` macros per motor type: name, NEMA frame, rated current, holding torque, full steps, `MAX_RPM` (default Velocity Limit, a bench value: 120 rpm for NEMA17/14, 60 rpm for NEMA23), IRUN/IHOLD/IHOLDDELAY and the six-point ramp (VSTART, A1, V1, AMAX, VMAX, DMAX, D1, VSTOP).
- `motor_axes.h`: which profile drives which axis (`SMX_AXISn_MOTOR`, n = 0…5 = DYNAMIXEL ID 1…6), direction (`SMX_AXISn_INVERT`, adds `shaft;`) and the gear ratio (`SMX_AXISn_GEAR_NUM` / `_DEN`), plus the common microstep setting (`SMX_MICROSTEPS` = 16).

The board overlay includes `motor_axes.h` and takes every per-axis property from it; `main.c` computes the position scale (full steps × microsteps × NUM / DEN) and reports the frame and rated current in Model Information. The GUI reads the same headers, so **Detect motors** shows the motor name and warns when a module reports a different motor than the project expects.

To move a motor to another joint, change `SMX_AXISn_MOTOR` and rebuild with `west build -p` (pristine, because the overlay changes). To add a motor type, copy a profile block and keep the macro format (`#define MOTOR_KEY_FIELD value`): the GUI parses it.

Ramp units with the 12 MHz internal clock and 16 microsteps (3200 µsteps per motor revolution):

Register	Unit	1 rev/s or 1 rev/s²
VSTART, V1, VMAX, VSTOP	0.715 µsteps/s	4474
A1, AMAX, DMAX, D1	65.5 µsteps/s²	49

The profiles use VMAX 4474 (1 rev/s) and AMAX 98 (2 rev/s²) for the NEMA17/14 motors and VMAX 2237, AMAX 49 for the NEMA23 shoulder. V1 = 0 disables the A1/D1 phase; D1 must still be ≥ 1, and VSTOP must be ≥ VSTART.

14.6 Firmware interface review

Scope: `boards/m2s_hello_fpga_kit_m2s010.overlay`, `src/motor_tmc5160.c`, and the Zephyr drivers they use (`adi,tmc51xx`, `microchip,mpfs-spi`). The code was analysed first; the SPI path and detection were then confirmed on the hardware (2026-10-10), and the register mapping and ramp generator on axis A0 (2026-10-11). The motor has not turned yet (finding 10).

What is correct

- **Datagrams:** 40 bits, MSB first, address byte with bit 7 = write. A read sends the address twice and takes the data from the second reply, as the datasheet requires (the first reply carries the previous request's data).
- **SPI mode 3:** the driver and `motor_tmc5160.c` request `CPOL | CPHA`; the MSS SPI driver maps them to SPO/SPH.
- **Chip select:** each `spi_transceive` is one 5-byte packet with hardware slave select SS0–SS5 (`reg = <0>…<5>`); SS is released after the packet, so the TMC5160 latches each datagram on the rising CSN edge.
- **Detection:** `tmc_read_version()` reads IOIN and checks VERSION (bits 31…24) = 0x30, the value for the TMC5160 and TMC5160A.
- **Power-on safety:** CHOPCONF.TOFF = 0 after reset, so with EN tied to GND the bridges stay off until Torque ON calls `stepper_enable()`.
- **Ramp limits:** VSTOP (20 in every profile) ≥ VSTART (10). Profile Acceleration 0 selects the profile's AMAX, so D1 is never 0 in positioning mode; VMAX and AMAX are cut to their register limits before they are written.

Findings

#	Severity	Finding	Fix	Status
1	**Critical**	**SPI clock.** The MSS SPI driver searches CLK_GEN only from 1 to 15, so it cannot reach rates below PCLK/30 = 1.67 MHz. For the old overlay value of 1 MHz it kept CLK_GEN = 0, and the bus ran at 50 MHz / 2 = **25 MHz** (TMC5160 limit: 4 MHz).	Set `spi-max-frequency = <2000000>`: CLK_GEN = 13, SCK = 50 MHz / 28 = 1.79 MHz. Any value from 1.67 to 4 MHz works; never below 1.67 MHz.	**Fixed**, verified on hardware
2	**Critical**	**Clock frequency.** The overlay had 16 MHz; with CLK on GND the chip runs on its internal 12 MHz. Every speed and acceleration conversion is about 33 % off.	`clock-frequency = <12000000>`.	**Fixed** in overlay
3	**Critical**	**Stop on stall is always on.** The pinned driver writes SW_MODE.sg_stop = 1 at init and re-enables it from a work item, whatever `activate-stallguard2` says. Above the TCOOLTHRS speed (old demo value 100, about 140 rpm motor speed) an untuned StallGuard2 can stop the motor, and the axis stays stopped until sg_stop is cleared. The firmware never clears it.	Set `tcoolthrs = <0>` (disables stop on stall and CoolStep), or clear SW_MODE in `motor_tmc5160.c` after init.	**Fixed** in overlay
4	High	**Motor power switched on after the kit.** TMC5160 logic runs from VM. If VM comes later, or drops, the chip restarts with reset values: the overlay settings (MRES, currents, ramp, GCONF, SW_MODE) are lost. A PING re-detects the driver, but nothing is configured again.	`motor_tmc5160.c` reads GSTAT before every command; after GSTAT.reset it writes the configuration again, reports Torque OFF and Hardware Error bit 0 (input voltage), and refuses the pending move. The position restarts at 0.	**Fixed** in `motor_tmc5160.c` (`tmc_check_reset()`)
5	High	**Currents** were demo values (IRUN 2, IHOLD 1).	Per-axis values from the motor profiles (table above).	**Fixed** (`motor_profiles.h`)
6	High	**Shoulder direction.** The wiring requires `shaft=1` on A1b; the overlay does not set `shaft` on `tmc5160_axis2`.	Set `SMX_AXIS2_INVERT` to 1 in `motor_axes.h` after the mechanics confirm the motors face each other.	Open
7	Medium	**Chopper off time.** `stepper_enable()` sets CHOPCONF bits 3…0 to 1111b (TOFF = 15). That works, but gives a low chopper frequency (audible noise, more ripple); the datasheet example uses TOFF = 3.	After enable, write CHOPCONF = `0x000100C3` with the axis' MRES bits (TOFF = 3, HSTRT = 4, HEND = 1, TBL = 2), or patch the driver.	**Fixed** in `motor_tmc5160.c` (`tmc_enable()`)

#	Sever-ity	Finding	Fix	Status
8	Medi-um	**MSS SPI driver.** `mss_spi_disable_ints()` inverts its mask and clears every CONTROL bit except the interrupt enables after each transfer; the next transfer re-writes them. The SCK idle level between transactions is therefore not guaranteed (it may be low instead of high). The unused `speed` variable is the source of the known build warning.	Works because CSN is high between packets. Confirm SCK/CSN timing with a logic analyser during bring-up; report upstream.	Open
9	Low	`tmc_read_version()` bypasses the driver's register lock. At boot the driver's StallGuard work item can read the same chip between the two data-grams of a probe.	Effect: one wrong probe at boot at most. Probe again after init, or read through the driver.	Open
10	**Critic-al**	**Module in STEP/DIR mode.** On axis A0 `IOIN` = 0x30000040: `SD_MODE=1`. With SD_MODE high the TMC5160 takes its steps from the STEP/DIR pins (tied to GND here), not from the in-ternal ramp generator. Measured dur-ing a move: XACTUAL and VACTUAL change, but the microstep counter MSCNT stays constant, DRV_STATUS.stst = 1 and CS_ACTUAL = IHOLD. The shaft only holds; Present Position changes without motion.	Pull SD_MODE low: on the BTT TMC5160 V1.0 open solder jumper W1 (SD) and keep W2 (SPI) closed; on the T Pro find the equivalent SD/SPI selection (not docu-mented in the sources here). Check with `tools/dxl_test_gui/tmc_diag.py`: it must report `SD_MODE=0`, and MSCNT must change during `--move`.	**Open** (hardware)

Findings 1–3 are fixed in the board overlay (`SMX_TMC5160_SPI_HZ` 2 MHz, `SMX_TMC5160_CLOCK_HZ` 12 MHz, `SMX_TMC5160_TCOOLTHRS` 0). Finding 1 was the cause of the failed first detection: with the old 1 MHz setting the bus ran at 25 MHz.

14.7 Bring-up checklist

#	Step	Status
1	Module in SPI motion-controller mode: SPI pad closed, **SD pad open (SD_MODE = 0)**. Check with `tmc_diag.py` (`SD_MODE=0`).	**Open on A0**: reads SD_MODE=1 (finding 10)
2	EN, CLK, STP, DIR to GND (short wires); VIO to J9-4 (3.3 V).	Done on A0
3	Overlay: SPI 2 MHz, clock 12 MHz, TCOOLTHRS 0.	Done
4	24 V (or a current-limited lab supply for the SPI test) on first, then the kit (USB). **Detect motors** in the GUI: each module reports `TMC5160`.	Done on A0 (12 V / 100 mA)
5	Electrolytic capacitor at VM/GND of every module.	Open
6	Per-axis `irun`/`ihold` and ramps from the motor profiles (done); `SMX_AXIS2_INVERT` for A1b; pristine build.	Partly done
7	Motor wired as in Motor connection, with VM off.	Open

#	Step	Status
8	One axis at a time on the 24 V supply: Torque ON, short moves at low speed, check direction and temperature.	Open

14.8 Reference documents

File	Publisher	Use it for
`TMC5160A_datasheet_rev1.18.pdf` (systemfrom-scratch.github.io/sfs-smartmanipulator-x-robot/hardware/data-sheets/TMC5160A_datasheet_rev1.18.pdf)	Analog Devices (Trinamic)	Register map, SPI datagrams and timing, current scaling, StallGuard2/CoolStep, power-up behaviour, external component rules
`BIGTREETECH_TMC5160-V1.0_schematic.pdf` (systemfrom-scratch.github.io/sfs-smartmanipulator-x-robot/hardware/data-sheets/BIGTREETECH_TMC5160-V1.0_schematic.pdf)	BIGTREETECH	Reference module: headers J3/J4, SPI and SD pads (W2, W1), sense resistors, missing pull resistors
`BIGTREETECH_TMC5160-V1.0_manual.pdf` (systemfrom-scratch.github.io/sfs-smartmanipulator-x-robot/hardware/data-sheets/BIGTREETECH_TMC5160-V1.0_manual.pdf)	BIGTREETECH	Reference module: supply range (8–40 V), current limits (3.1 A RMS, 4.4 A peak), EN pad marking, handling precautions
BTT TMC5160T Pro V1.0 wiki (global.bttwiki.com/TMC5160T%20Pro%20V1.0.html)	BIGTREETECH	Module in use: pinout J1/J2, VM 8–56 V, R_{SENSE} 0.075 Ω
`Hello_FPGA_Kit_schematics_rev1.pdf` (systemfrom-scratch.github.io/sfs-smartmanipulator-x-robot/hardware/data-sheets/Hello_FPGA_Kit_schematics_rev1.pdf)	Microchip	Arduino header nets and balls, 3.3 V supply, PIC32/MCP2221A host link, MSS pads wired to the PIC32
`Wiring_RevD_DirectWiring_EN.pdf` (systemfrom-scratch.github.io/sfs-smartmanipulator-x-robot/hardware/Wiring_RevD_DirectWiring_EN.pdf)	project (bavariamatic)	Stage-1 wiring concept, motor list, power and servo
`Handsontec_MG996R_user_guide.pdf` (systemfrom-scratch.github.io/sfs-smartmanipulator-x-robot/hardware/data-sheets/Handsontec_MG996R_user_guide.pdf)	Handsontec	Gripper servo MG996R: supply 4.8–7.2 V, currents, 1–2 ms / 20 ms signal, pinout; see Gripper servo

When a module or kit revision changes, replace the PDF, update the revision in the sources table (p. 71) and re-check the tables on this page and on Hello FPGA Kit wiring (p. 62).

CHAPTER 15

Gripper servo (MG996R)

The gripper uses an MG996R hobby servo. The SmartFusion2 drives it with a 50 Hz pulse from CorePWM in the FPGA fabric. The firmware makes it a virtual DYNAMIXEL device on ID 7, see Gripper servo (p. 51) in the firmware architecture and the **Gripper** tab in the motor tester GUI (p. 58).

> **Sources:** Handsontec *MG996R Metal Gear Servo Motor* user guide (`Handsontec_MG996R_user_guide.pdf` (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/Handsontec_MG996R_user_guide.pdf), online (handsontec.com/dataspecs/motor_fan/MG996R.pdf)), Hello FPGA Kit schematics Rev. 1 (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/datasheets/Hello_FPGA_Kit_schematics_rev1.pdf), wiring concept Rev. D (systemfromscratch.github.io/sfs-smartmanipulator-x-robot/hardware/Wiring_RevD_DirectWiring_EN.pdf) and the Libero hardware contract (CorePWM). If this page and a source disagree, the source is authoritative; update this page.
>
> **Status (2026-10-11):** firmware and GUI are implemented and tested on the simulation firmware and `native_sim`. On the kit the servo follows Goal Position from the GUI (6 V supply with sufficient current). The travel (`SMX_SERVO_TRAVEL_DEG`) and the gripper limits are **not calibrated yet**.

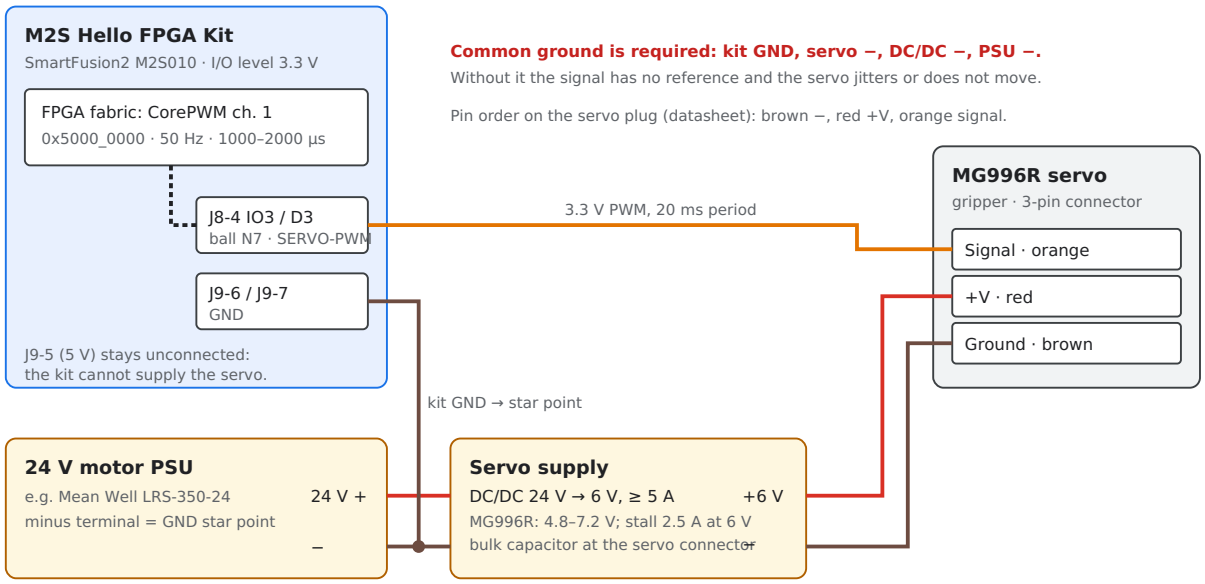
15.1 Datasheet values

Parameter	Value (Handsontec user guide)
Operating voltage	4.8–7.2 V
Stall torque	9.4 kgf·cm at 4.8 V, 11 kgf·cm at 6 V
Operating speed	0.17 s/60° at 4.8 V, 0.14 s/60° at 6 V
Running current	500 mA
Stall current	2.5 A at 6 V
Dead band width	5 µs
Control signal	PWM, 20 ms period (50 Hz), 1–2 ms pulse
Positions	~1 ms = one end ("−90"), 1.5 ms = middle ("0"), ~2 ms = other end ("90")
Rotation	approximately 120° (60° in each direction)
Weight, dimensions	55 g; 40.7 × 19.7 × 42.9 mm (drawing: 40.3 mm between the mounting holes, 53.6 mm over the flanges, 47.6 mm high with the horn)
Connector	3-pin, brown = ground (−), red = +V, orange = signal

Notes on the values:

- Shop listings and this project's earlier notes give "180°" and "13 kg·cm". The user guide gives **about 120°** for the 1–2 ms range and **11 kgf·cm at 6 V**. Plan the gripper with the datasheet values.
- The guide labels positions "−90 / 0 / 90" but also states 120° of travel. Measure the real angle per millisecond on the gripper (see Calibration (p. 86)) instead of relying on either number.
- The signal in the guide's diagram is drawn at the supply level (~5 V). The guide gives no input threshold. The kit outputs **3.3 V** (see Signal level (p. 85)).

15.2 Connection to the Hello FPGA Kit



Gripper servo wiring: SERVO-PWM from J8-4, 6 V from a separate DC/DC converter, common ground

Servo wire	Connects to	Notes
Orange (signal)	Kit **J8-4** = Arduino IO3 / D3, ball **N7**, net `MSIO104NB4_N7_ARD_INT1_PWM_IO3`	CorePWM channel 1, 3.3 V logic. Dupont m/f jumper, ≤ 30 cm
Red (+V)	**+6 V** of the servo DC/DC converter	Never from the kit (J9-5 5 V or 3V3): the kit is powered from USB and the servo draws up to 2.5 A
Brown (−)	**−** of the servo DC/DC converter, and through it the GND star point	The servo current must not flow through the kit or the Dupont wires
–	Kit **J9-6 / J9-7 (GND)** → GND star point (minus of the 24 V PSU)	Common reference for the signal. This connection is already part of the wiring list

Supply:

Item	Specification
Converter	DC/DC 24 V → **6 V, ≥ 5 A**, used for the servo only. 6 V gives 11 kgf·cm (datasheet); stay within 4.8–7.2 V
Bulk capacitor	Recommended: low-ESR electrolytic of about 470–1000 µF / ≥ 10 V at the servo connector, against voltage dips at start and stall (not from the datasheet; project practice for hobby servos)
Wires	+6 V and − of the servo: ≥ 0.5 mm² (stall current 2.5 A). Only the signal goes over a Dupont jumper
Ground	Star point at the 24 V PSU minus. Kit GND, DC/DC minus and the servo − all meet there

Signal level

The kit I/O level is 3.3 V (IOREF = 3P3V); its pins must never see more than 3.3 V. The servo only receives the signal, so no voltage flows back into N7. Most MG996R servos switch reliably at 3.3 V, but the datasheet does not specify the input threshold.

- If the servo jitters or ignores the pulses although the oscilloscope shows a clean 50 Hz signal at J8-4, add a one-way level shifter, for example a 74AHCT125 powered from 5 V (input 3.3 V compatible, output 5 V).
- Do not use a resistor pull-up to 6 V on the signal line: it would drive more than 3.3 V into N7.

15.3 Signal from the firmware

Setting	Value	Where
Period	20 ms (50 Hz): PRESCALE 99 → 1 µs tick at PCLK 100 MHz, PERIOD 19999	`src/servo_corepwm.c`, Libero contract
Pulse width	1000–2000 µs, 1 µs resolution (dead band of the servo: 5 µs)	Kconfig `SMX_SERVO_PULSE_MIN_US` / `_MAX_US`
Position scale	Goal Position 0 … range maps linearly to the pulse width; range = `SMX_SERVO_TRAVEL_DEG` in 0.088° units	Kconfig `SMX_SERVO_TRAVEL_DEG` (default 90)
Start	No pulses after reset and with Torque OFF (servo limp). The first Torque ON moves to `SMX_SERVO_INITIAL_DEG` (default 45° = 1500 µs) at full speed	Kconfig `SMX_SERVO_INITIAL_DEG`
Speed	Software ramp of the pulse width every 20 ms (Profile Velocity); 0 = jump, the servo moves at 0.14 s/60°	GUI **Gripper** tab, register 112

The default `SMX_SERVO_TRAVEL_DEG` of 90 only scales the degrees shown to the host. With the datasheet's ~120° for 1–2 ms the gripper turns about 1.33 times the displayed angle until the value is calibrated.

15.4 Bring-up

Run these steps in order. Mount the servo without the horn or with the gripper open, so a wrong pulse cannot crush anything.

1. **Firmware:** flash the target firmware. In the GUI, **Detect motors** shows *gripper ID 7* in the board line; the **Gripper** tab shows *Output: CorePWM*.
2. **Signal without the servo:** connect only an oscilloscope to J8-4 and GND. With Torque OFF the line stays low. Torque ON in the **Gripper** tab: 50 Hz, pulse 1500 µs (start position 45°). Move to the limits: 1000 µs and 2000 µs. The *Pulse* field in the GUI shows the same values.
3. **Supply:** with the servo unplugged, set the DC/DC output to 6 V and check the common ground (kit GND to the PSU minus).
4. **Servo:** switch the 24 V PSU and the DC/DC off, plug the servo in (brown −, red +, orange signal), switch on, then Torque ON. The servo moves to the middle position at full speed.
5. **Slow test:** set *Profile velocity* to 20–50 (27–69 °/s) and move in small steps. Watch the servo supply voltage during moves.
6. **Limits:** find the open and closed gripper positions (see Calibration (p. 86)), enter them as *Position limits* with Torque OFF, and click *Write limits*. Repeat after every reset: the limits are not stored.

15.5 Calibration

The firmware cannot measure the servo position. Find the gripper's end positions by hand:

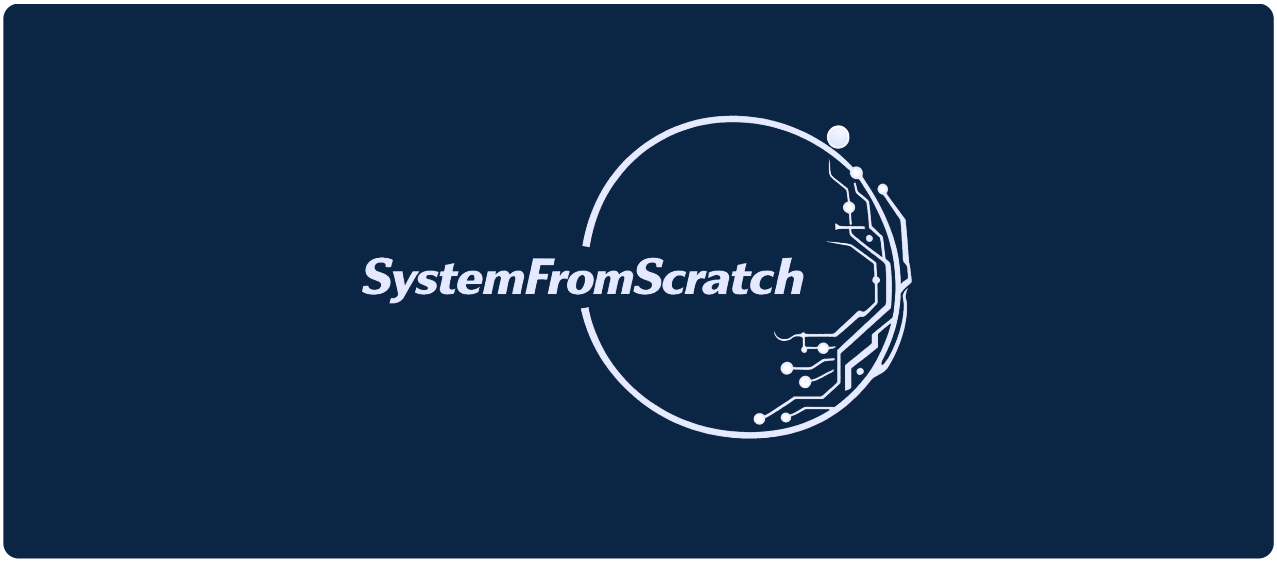
1. Profile velocity about 30, Torque ON.
2. Move in steps of 1–2° towards *closed* until the jaws just touch, then back off by 2–3°. Note the angle. Do not hold the servo against a stop: it then draws up to the stall current of 2.5 A and heats up.
3. Do the same towards *open*.
4. Enter both values as *Position limits*. For a permanent default, change `SMX_SERVO_TRAVEL_DEG`, `SMX_SERVO_PULSE_MIN_US` / `_MAX_US` or `SMX_SERVO_INITIAL_DEG` in Kconfig and rebuild.

If the measured rotation for 1000–2000 µs differs from 90°, set `SMX_SERVO_TRAVEL_DEG` to the measured angle so the GUI shows true degrees.

15.6 Safety

1. Never power the servo from the kit (J9-5 5 V, J9-4 3V3) or over the breadboard.
2. Only 3.3 V signals on J8-4. No pull-up to the servo supply.
3. Plug or unplug the servo only with the DC/DC output off.
4. Torque OFF (or **STOP**, key Esc, in the GUI) removes the pulses: the servo is limp and the gripper may open under load.
5. The servo position after power-up is unknown. The first Torque ON moves at full speed to the start position: keep hands out of the gripper.

Author & Company



SmartManipulator-X is designed, written and maintained by **Anton Vynohradov** and published by **SystemFromScratch®** — secure embedded systems and compliance-ready engineering, from concept to certification.

Author

Anton Vynohradov · ORCID 0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

Embedded Systems Engineer · Secure Product Architect · Founder of SystemFromScratch®

Embedded systems built from scratch — from bare metal to Linux, from first schematic to certification.

Anton designs and builds reliable, secure embedded products end to end: firmware, board support packages and drivers, RTOS and Embedded Linux platforms, hardware bring-up, and the security and compliance work needed to ship them. He is based in Poland and works remotely worldwide.

Area	Focus
Embedded software	Bare-metal, RTOS and Linux firmware in C/C++, built to be maintained for years
BSP & drivers	Bootloaders, Zephyr drivers and Linux kernel work for MCUs, SoCs and SoC FPGAs
Security & compliance	Secure boot, threat modelling, MISRA C/C++, EU Cyber Resilience Act (CRA) readiness
Industries	Industrial IoT, automotive, robotics, medical and consumer electronics

Technology: C · C++ · Python · Zephyr RTOS · FreeRTOS · Embedded Linux · Yocto · Buildroot · ARM Cortex-M · STM32 · Microchip FPGA · RISC-V · CAN / ISO-TP · ROS 2 · Docker · Git · CI/CD · MISRA C/C++ · EU CRA

For SmartManipulator-X, Anton designed the system architecture, the Robot PC image and its ROS 2 configuration, the robot description generated from the BCN3D Moveo CAD, the SmartFusion2 controller firmware with its motor tester, and this documentation.

SystemFromScratch®

SystemFromScratch® was founded by Anton Vynohradov. The company combines deep embedded engineering with cybersecurity and certification-focused product development, for teams in industrial IoT, automotive, robotics, medical and consumer electronics that need a partner who understands the whole stack — silicon, firmware, operating system and the regulations that govern them.

Service	What it covers
Embedded software development	Bare-metal, RTOS and Linux firmware for performance, reliability and long-term maintainability
BSP & device drivers	Board support packages, bootloaders and low-level drivers for MCUs, SoCs, FPGAs and custom hardware
RTOS & Embedded Linux	Zephyr, FreeRTOS and Embedded Linux (Yocto, Buildroot): porting, configuration, integration, reproducible builds
Hardware bring-up & integration	Schematic review, first boot, validation and production support, including SoC FPGA and motor-control platforms
Cybersecurity & CRA readiness	Secure boot, firmware integrity, threat modelling and a secure development lifecycle for the EU Cyber Resilience Act
Compliance & consulting	MISRA C/C++, static analysis, code and architecture reviews, compliance roadmaps

SmartManipulator-X is one of the open-source projects in the SystemFromScratch® portfolio: public engineering work that can be inspected line by line, from the ROS 2 host down to the stepper driver.

Contact

Channel	Address
Website	systemfromscratch.com
E-mail	avynohradov@systemfromscratch.com
GitHub	github.com/AntonVynohradov
GitHub organization	github.com/SystemFromScratch
LinkedIn	linkedin.com/in/vynohradov
ORCID	0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

Channel	Address
Location	Poland — working remotely worldwide

Building a robot, bringing up new hardware, or preparing for the Cyber Resilience Act? Get in touch through systemfromscratch.com or by e-mail.

Support and feedback

- **Bugs and feature requests:** open an issue on GitHub (github.com/SystemFromScratch/sfs-smartmanipulator-x-robot/issues). Please include the repository commit, the host OS and Docker version, the command you ran and its output; for hardware, the firmware variant and the wiring revision.
- **Contributions:** read Development (p. 12) first; keep the documentation in step with the change.
- **Commercial support, integration and custom development:** contact SystemFromScratch® through systemfromscratch.com.

Trademarks and acknowledgements

SystemFromScratch® is a registered trademark. The SystemFromScratch® name and logo are used in this documentation to identify the publisher of SmartManipulator-X; the Apache License 2.0 grants no rights to use them, and they may not be used to endorse or promote products derived from SmartManipulator-X without prior written permission.

ROBOTIS, DYNAMIXEL and OpenMANIPULATOR are trademarks of ROBOTIS Co., Ltd. SmartFusion2 and Libero are trademarks of Microchip Technology Inc. TRINAMIC and TMC5160 are trademarks of Analog Devices, Inc. BIGTREETECH is a trademark of Shenzhen Big Tree Technology Co., Ltd. ROS is a trademark of Open Source Robotics Foundation, Inc. BCN3D is a trademark of BCN3D Technologies. SmartManipulator-X is not affiliated with or endorsed by these companies; their names are used only to identify the products the project builds on.

The mechanical design comes from the BCN3D Moveo (github.com/BCN3D/BCN3D-Moveo) project (MIT license; the license text ships with the robot description meshes). The ROS 2 software concepts come from ROBOTIS OpenMANIPULATOR-X (github.com/ROBOTIS-GIT/open_manipulator) and the DYNAMIXEL SDK and hardware interface.

APPENDIX A

License

The project-owned files of SmartManipulator-X are free and open-source software, licensed under the **Apache License, Version 2.0** (apache.org/licenses/LICENSE-2.0), matching the ROBOTIS Open-MANIPULATOR-X ROS 2 repository.

Item	Value
Copyright	© 2026 Anton Vynohradov
License	Apache License 2.0
SPDX identifier	`Apache-2.0`
Applies to	The Robot PC image files, the ROS 2 packages, the tools and this documentation

Third-party sources keep their own licenses: the ROBOTIS packages that the Robot PC image checks out through `ros2_robot.repos`, and the BCN3D Moveo meshes of the robot description (MIT, text in `meshes/LICENSE-BCN3D-Moveo`).

A.1 What the license allows

In short — the full text below is authoritative:

- **Use** the project in any project, open source or proprietary, commercial or not.
- **Modify** it and **distribute** it, in source or binary form, alone or as part of a product.
- **Patent grant:** each contributor grants you a license to any of their patents that the contribution uses.

A.2 What it requires

When you distribute SmartManipulator-X or a modified version of it:

- Include a copy of the license.
- Keep the copyright, patent, trademark and attribution notices in the source files.
- Mark the files you changed.
- If you redistribute a NOTICE file, keep its attribution notices (SmartManipulator-X has none at the moment).

A.3 What it does not give

- No warranty: the software is provided "as is".
- No trademark rights: the license does not let you use the names of the author or of System-FromScratch® to endorse your product.

A.4 Trademarks

SystemFromScratch® is a registered trademark; the license grants no rights to use the name or logo. The marks of ROBOTIS, Microchip, Analog Devices, BIGTREETECH, Open Source Robotics Foundation and BCN3D are listed on the About (p. 87) page.

A.5 Full license text


```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
```

```
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
```

```
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't include
      the brackets!)  The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```