# TMC9660 Configuration and Bootstrapping

by Florian Voelker and Lenard Hess

## BOOTSTRAPPING THE TMC9660 IC USING UBLTOOLS SOFTWARE

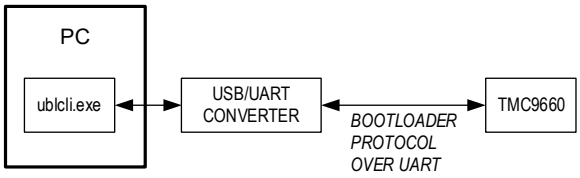
This application note details how to bootstrap the TMC9660 IC using the UblTools software package. It offers a guide on how to go from first communication to evaluating the boot configuration options to programming the part for production usage.

The TMC9660 IC offers extensive configuration to fit a wide range of application needs using a configurable bootloader. With the UblTools package, this configuration effort is reduced to simple configuration files that can be ephemerally applied to the IC during evaluation or permanently burned into onboard one-time programmable (OTP) memory for production. The optional external memory that the TMC9660 utilizes for extra functionality can also be fully bootstrapped through the bootloader and the UblTools.

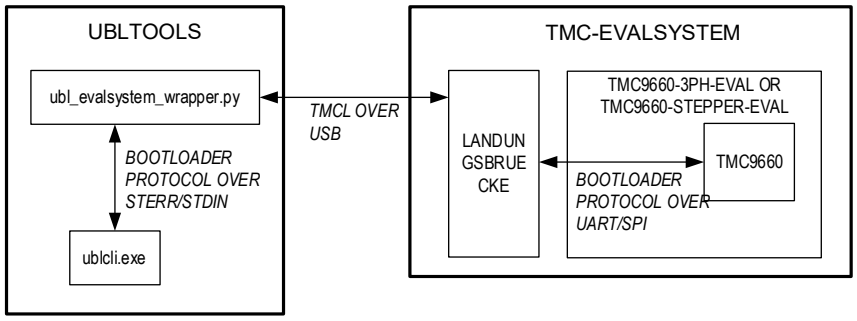
## UBLTOOLS FEATURES

- High-Level Access to TMC9660 Bootloader
- Simple Step-by-Step Bring-Up
- Examples for TMC9660 Evaluation Boards
- Support for Integration into Custom Communication Stacks

## DIRECT CHIP COMMUNICATION




## SUPPORT FOR TMC-EVALSYSTEM




Analog Devices is in the process of updating documentation to provide culturally appropriate terminology and language. This is a process with a wide scope and will be phased in as quickly as possible. Thank you for your patience.


One Analog Way, Wilmington, MA 01887-2356, U.S.A. | Tel: 781.329.4700

# Quick Start Guide: Getting Started

This application note contains a quick reference on how to get started using UblTools to bootstrap the TMC9660 IC. The application note details the steps taken once you have a not-yet-bootstrapped TMC9660 IC powered on, pulled out of reset, and have a connection to the TMC9660 established.

This application notes details how to connect to the TMC9660 through a direct serial Universal Asynchronous Receiver-Transmitter (UART) connection as well as how to connect to the evaluation kit (EV kit) version of the TMC9660. Note that the TMC9660 EV kit does not offer a direct serial connection but instead offers an indirect connection through the Landungsbruecke connector board.

Note that this quick start section does not contain the full reference for all possible command options, but rather just a suggested sequence for bootstrapping the TMC9660 for evaluation. For the full reference of UblTools, see the *UblTools Command Reference* section.

## Setting Up UblTools

To set up UblTools, download the UblTools installer from the Analog Devices, Inc., website at **https://www.analog.com/en/products/TMC9660.html** and run it. The installer creates a folder at the user's chosen target installation directory containing the following files:

- ublcli.exe—the program used to communicate with the TMC9660 bootloader. No installation is required, and the user can simply run it directly using a command line/CLI (explained in the following sections).
- ubl_evalsystem_wrapper.py—a Python wrapper script to connect ublcli.exe with the TMC9660 on the evaluation boards TMC9660-STEPPER-EVAL and TMC9660-3PH-EVAL. Using this requires an installation of Python to run this script.
- ioconfig_tmc9660-stepper-eval.toml—the default boot configuration file for the TMC9660-STEPPER-EVAL.
- ioconfig_tmc9660-3ph-eval.toml—the default boot configuration file for the TMC9660-3PH-EVAL.
- partition_tmc9660-eval.toml—the default partition file for the external SPI Flash available on the TMC9660-STEPPER-EVAL and TMC9660-3PH-EVAL.

UblTools is currently only supported on Windows. Support for Linux is planned but not yet available.

## Using ublcli.exe

**Note**: All the following explanations are for using ublcli.exe on Windows with PowerShell. Any shell capable of running commands can be used to run ublcli.exe instead as well. Furthermore, the explanations in this section assume little to no prior knowledge of using PowerShell. If the user is already familiar with PowerShell or any other shell, the following steps can be accomplished in other ways.

To open a PowerShell window, first press the *Windows* key and the *R* key simultaneously. This opens the *Run* window. In that window, type "*powershell*" (without quotation marks) and press *Enter*. This closes the *Run* window and opens a PowerShell window.

Then, navigate to the location of the downloaded ublcli.exe file. For example, if the file is in the "*C:\Downloads*" folder, the corresponding PowerShell command would be as follows:

*cd C:\Downloads*

Now, the ublcli.exe program can simply be run in the PowerShell:
*.\ublcli.exe*
This prints out a compact usage guide of the command.

To print a more detailed usage guide, run:
*.\ublcli.exe --help*

To print the version of ublcli.exe, run:
*.\ublcli.exe --version*

**Note**: For any other command descriptions in this document, the leading ".\" characters are omitted for readability.

## Connecting to the TMC9660

For all commands that communicate with a TMC9660, the connection must be configured.

Following are two connection configuration examples that are valid for an unprogrammed TMC9660 chip connected through a serial port. The "inspect chip" command that is being used in these examples query and display basic chip information. For the full reference of all connection settings, see the *TMC9660 Boot Configuration File* section.

For a UART-based connection, only the connection port must be specified, e.g., to run the "inspect chip" command on a TMC9660 connected through the exemplary COM123 serial port:
*ublcli.exe --port COM123 inspect chip*

For an RS485-based connection, the RS485 TX_EN pin and delays must also be specified, e.g., to connect through COM123 with the TMC9660 using GPIO 2 as the TX_EN pin with 20 ticks of pre- and post-delay:
*ublcli.exe --port COM123 --tx-en-pin 2 --tx-en-delays 20 inspect chip*

### Connecting to the TMC9660 Evaluation Kits

The TMC9660 has two evaluation boards as part of the TMC-EvalSystem - the TMC9660-3PH-EVAL and TMC9660-STEPPER-EVAL. These evaluation boards can be connected directly through the USB-UART serial adapter or through the Landungsbruecke board that is part of the EV kits. Connecting through the Landungsbruecke does not offer a serial port to directly communicate with the TMC9660 IC. Instead, the communication with the TMC9660 bootloader is facilitated through multiple commands sent to the Landungsbruecke.

To connect to these evaluation boards, the Python helper script ubl_evalsystem_wrapper.py is included as part of the UblTools. To use the script, an installation of Python is required.

To connect to the TMC9660 of the evalsystem, run:
*python ubl_evalsystem_wrapper.py COM123 inspect chip*
Behind the scenes, this causes the wrapper script to call ublcli.exe as follows:
*ublcli.exe --use-pipes --ignore-crc inspect chip* (Do not run this manually!)
The following steps are done by the script:

- The script receives the command requests from ublcli.exe.
- The script sends TMCL-based requests to the Landungsbruecke.
- The Landungsbruecke performs the bootloader command request to the connected TMC9660 IC.
- The script receives the TMC9660 reply from the Landungsbruecke.

- The script sends back the reply to ublcli.exe.

### Configuring ublcli.exe Using Environment Variables

When using UblTools in a typical setting, the configuration of the connection does not change between different invocations of ublcli.exe. To simplify usage, all connection settings are also configurable using environment variables. For example, instead of specifying the port with the "*--port*" argument every time, it can also be specified once with the *UBL_PORT* environment variable.

Setting an environment variable can be done in PowerShell using the following command:

*$Env:UBL_PORT = "COM123"*

This sets the "*UBL_PORT*" variable to the value "*COM123*", causing ublcli.exe to use *COM123* for connections to the TMC9660.

**Note**: All subsequent command examples assume the connection is configured using environment variables – no connection configuration is written in the following examples.

**Note**: This method of setting environment variables only applies to the PowerShell prompt it is being run in. Once closed, the values are no longer set.

## Creating a Configuration

To configure the TMC9660 chip using UblTools, a configuration file is needed. For the full reference of the configuration file format, see the *TMC9660 Boot Configuration File* section.

To generate a blank configuration file, run:

*ublcli.exe setup config*

To load the currently active configuration from a connected TMC9660 instead, run:

*ublcli.exe inspect config*

Both commands print out a configuration toml file.

The generated configuration file can then be edited to the desired configuration.
For example, to configure the VEXT1 LDO to generate a 5V output, edit the file to contain the following lines:

```
[ldo]
vext1_voltage = 5.0   # 0.0, 2.5, 3.3, 5.0
```

***Figure 1. Configuration snippet for VEXT1 LDO.***

The updated configuration can then be applied to the TMC9660 IC with the following command:

*ublcli.exe write config my_configuration.toml*

This causes the TMC9660 bootloader to apply the updated settings.

## Setting Up External Memory

The TMC9660 supports using an external memory to enable additional features—loading motor parameters from external storage when launching the motor control and running a TMCL script from external memory. If these features are not desired, this section is not needed and can be skipped.

To use external memory (SPI Flash or I2C electronically erasable programmable read-only memory [EEPROM]) with the TMC9660, multiple steps are required.

The following example shows how to configure an external SPI Flash memory (W25X40CLSNIG) connected to SPI0 with GPIO12 used as chip select. Both the TMC9660-STEPPER-EVAL and the TMC9660-3PH-EVAL use this memory with these wirings.

First, the TMC9660 connection to the external memory must be configured. Edit your boot configuration file to contain the following lines and then write it to the TMC9660:

```
[spi_flash]
enabled      = true
frequency    = 10000000   # Hz : 40 MHz / (N+1), N: 3-15
pin_cs       = 12         # 0 - 18
spi_block    = "SPI0"     # "SPI0", "SPI1"
pin_spi0_sck = 11         # 6, 11 (Required when spi_block = "SPI0")
```

***Figure 2. Configuration snippet for SPI Flash.***

As an optional step, the SPI Flash can first be fully erased. This can be done with the following command:

*ublcli.exe write SPI erase*

Note that the other UblTools commands writing to SPI automatically erase memory sections before writing to them, making this step optional if the user does not need to ensure that prior memory contents are deleted.

Next, the SPI flash must be partitioned. To partition an external memory, UblTools uses another toml configuration file – the partition file. The following example of this partition file configures two partitions—one for holding motor parameters to be applied when the motor control system starts and one for holding a motor script. See the *UblTools Partition File Format* section for details on the partition file format.

```
partition_config_version = "v1.0"

[part_table]
version     = "v1.1"
size        = 0x00080000    # Memory size: 4Mibit = 512KiB
sector_size = 0x1000        # Sector size: 4KiB

[[part_table.partition]]
name     = "settings"
type     = "TMCL_CONFIG"
offset   = 0x1000
size     = 0x1000
writable = true

[[part_table.partition]]
name     = "script"
type     = "TMCL_SCRIPT"
offset   = 0x2000
size     = 0x2000
writable = true
```

***Figure 3. External memory partition file.***

To write the configuration, run the following command:

*ublcli.exe write SPI partition my_partition.toml*

To verify that the partition has been written correctly, read back the written configuration using:

*ublcli.exe inspect SPI*

**Note:** For an I2C EEPROM, the steps are similar. The relevant configuration section is *"[i2c_eeprom]"* and the commands use "*I2C*" instead of "*SPI*". Also, be advised that the erase command for an I2C EEPROM can take multiple minutes to complete.

## Starting the Motor Control System

To start the motor control system, run the following command:

*ublcli.exe start*

**Note**: After a reset or a power cycle, any runtime configuration from the "*write config*" command is lost and must be re-applied before starting the motor control system. On the other hand, the external memory content is persistent.

## Burning Permanent Configurations

Once the user has finalized the configuration for the TMC9660, it can be burned into the internal OTP memory to have it automatically get applied on power-on.

**Warning**: **This step is permanent.** Only perform this step once the user has fully set up and tested the configuration. The TMC9660 offers four configuration slots in total. A configuration slot can only be invalidated by writing a subsequent slot. If the last slot is written, no new configuration can be burned. A wrong configuration that causes the TMC9660 to not start up correctly (e.g., by misconfiguring the communication interfaces) leaves the user unable to burn corrected configurations!

To burn a configuration, run the configuration write command with the additional "*--burn*" flag. It is suggested to first run the command with the "*--dry-run*" flag to verify that no mistakes were made writing the command:

*ublcli.exe write config my_configuration.toml --burn --dry-run*

The suggested *"--dry-run*" can then be removed to actually perform the OTP burn:

*ublcli.exe write config my_configuration.toml --burn*

The command prompts the user for confirmation before performing the burn. To skip the confirmation, the "*--yes*" flag can be added. It is strongly recommended to only use this in a production setting, and not for manual evaluation.

Once burned, the configuration is applied on any subsequent power-on.

The OTP configuration that is applied on power-on can be inspected using:

*ublcli.exe inspect config OTP*

### Configuration Detail: Motor Control System Start

The configuration file has an option whether to start the motor control system ("*load_rom_code*" in the "*[bootstrap]"* section). UblTools has special behavior for this value for the two main workflows using the configuration file—writing it at runtime and burning it to OTP. A special flag is available ("*--boot*" or "*--no-boot*") to control this special behavior.

**Table 1. Boot Flag Influence on Different Write Commands**

| COMMAND | BOOT FLAG | | |
|---|---|---|---|
| | No flag set | --boot | --no-boot |
| write config | If "load_rom_code = true" is set, exit with an error instead of writing. | If "load_rom_code = true" is set, launch the motor control after writing the configuration. | If "load_rom_code = true" is set, ignore it. |
| write config --burn | If "load_rom_code = false" is set, exit with an error instead of burning. | | Burn the configuration even if "load_rom_code = false" is set. |

For writing configurations, this behavior intends to prevent accidentally booting, requiring the user to explicitly choose whether to boot or not.

For burning configurations, this behavior intends to prevent accidentally burning a configuration that does not start the motor control system. If desired, such a configuration can still be used—in case the design calls for a preconfigured TMC9660 IC that stays in bootloader mode.

# UBLTOOLS Command Reference

This section contains a full reference for all commands.

## General Settings

By default, ublcli.exe only prints basic information about the command it is running. To see more detail, the "*-v*" flag can be added to any command:

*ublcli.exe -v inspect chip*

Up to three "*v*" characters can be added to increase the amount of detail. At the highest level of detail, every datagram sent to the TMC9660 and the response received for it is printed out:

*ublcli.exe -vvv inspect chip*

**Note**: The "*-v*" flag must be placed before the command to run.

The "*--help*" flag can be added to see usage information for ublcli.exe:

*ublcli.exe --help*

It can also be added at the end of any subcommand to view usage information for that subcommand. For example, to see the different subcommands of the "*inspect*" command, run:

*ublcli.exe inspect --help*

## Connection Settings

For all commands that connect to a TMC9660 chip, the connection must be configured.

There are two ways for ublcli.exe to connect to a TMC9660—through a serial port or through an external wrapper program to allow custom connection logic.

### Serial Connections

When connecting through a serial port, the following settings are used. All these settings can be configured through an argument on the command line, or by setting an environment variable. If both are present, the command line argument is used.

**Table 2. CLI Arguments**

| CLI ARGUMENT | ENVIRONMENT VARIABLE | DESCRIPTION |
|---|---|---|
| --port | UBL_PORT | The serial port that ublcli.exe should connect to.<br>This argument is required. |
| --timeout | UBL_TIMEOUT | The amount of time in seconds that ublcli.exe should wait for a reply.<br>Default: 5.0 |
| --baud-rate | UBL_BAUDRATE | The serial baud rate for communicating with the TMC9660.<br>Default: 115200 |
| --chip-addr | UBL_CHIP_ADDR | The chip address to use in the TMC9660 UART communication protocol.<br>Default: 1 |
| --host-addr | UBL_HOST_ADDR | The host address to expect in the TMC9660 UART communication protocol replies. If not set, UblTools detect the host address used in the first reply received and expect this address for subsequent replies. |
| --tx-en-pin | UBL_TX_EN_PIN | The TMC9660 GPIO to use as TX_EN.<br>If this option is set, UblTools send the BOOTSTRAP_RS485 command to set up RS485 communication. |
| --tx-en-delays | UBL_TX_EN_DELAYS | The pre- and post-delay to configure as part of the BOOTSTRAP_RS485 command.<br>Default: 16 |

All connection settings must be placed before the command to run:

Valid: *"ublcli.exe --port COM123 inspect chip"*

Invalid: *"ublcli.exe inspect chip --port COM123"*

## Custom Connections

When the connection to the TMC9660 is not directly available by serial port, ublcli.exe can be connected to a separate program that facilitates the connection instead. This mechanism allows any form of custom connection logic, including but not limited to tunneling the UblTools communication through a different protocol, or connecting to a TMC9660 through an SPI instead of UART. For example, this mechanism is used in the evaluation board that is part of the Trinamic Modular Evaluation System and by the TMCL-IDE.

To create a custom connection, the "*--use-pipes*" argument must be passed to ublcli.exe. This causes ublcli.exe to send request datagrams through the standard error output and await replies through the standard input. The data sent are UART request bytes and the data read back is expected to be a valid UART reply.

When the "*--use-pipes*" argument is supplied, the serial connection arguments are ignored, and no connection is made to any serial port by ublcli.exe.

For an example application, refer to the ubl_evalsystem_wrapper.py script that is part of UblTools.

## Command: setup config

This command prints out a sample boot configuration toml file with the default values and comments listing the possible values as well as what parameters are optional. It is meant to be used as a starting point for creating a TMC9660 configuration file for your design.

See also the *Command: inspect config* to read out the active or stored configuration instead.

This command does not have any optional arguments.

## Command: inspect

This command allows inspecting a connected TMC9660.

With no further arguments, the inspect command defaults to running the inspect chip subcommand.

Following are listed the different subcommands to inspect different parts of the connected TMC9660.

### Command: inspect chip

This command shows basic information about the connected chip. It shows the connected chip (TMC9660) as well as the bootloader version by default. The command supports an optional "*--extended*" flag, which causes it to print out additional information.

In this application note, this command is used as the command to check connectivity as it is one of the simplest commands connecting to the chip and causes no side effects.

### Command: inspect config

This command is used to read out configurations from a connected TMC9660.

It always prints the configuration in the toml format that can be written back using the "*write config*" command.

By default, or when passing the "*active*" subcommand it prints the configuration currently applied to the running TMC9660.

When passing the "*OTP*" subcommand, it instead prints out configuration stored in OTP. By default, this subcommand checks all OTP configuration pages and prints out the page that is applied upon the next power-on. Alternatively, the optional "*--page*" argument can be used to print out the content of a specific OTP page instead.

### Command: inspect SPI

This command is used to read out information about a connected SPI Flash memory. It prints out the partition information if the connected memory is partitioned. Otherwise, it prints out whether an SPI Flash is connected or not—whether it could be detected by the TMC9660.

**Known issue:** In ublcli.exe v1.0.0, this command will not verify the partition table checksum. If the checksum does not match, it will still report the invalid table's contents as if they were valid. This is fixed in v1.0.1, where the command will report the invalid checksum instead.

### Command: inspect I2C

This command is used to read out information about a connected I2C EEPROM memory. It prints out the partition information if the connected memory is partitioned. Otherwise, it prints out whether an I2C EEPROM is connected or not—whether it could be detected by the TMC9660.

**Known issue:** In ublcli.exe v1.0.0, this command will not verify the partition table checksum. If the checksum does not match, it will still report the invalid table's contents as if they were valid. This is fixed in v1.0.1, where the command will report the invalid checksum instead.

## Command: write

This command is used to perform various write actions on the TMC9660.

### Command: write config

This command is used to apply a configuration file to the TMC9660. The bootloader applies this configuration while the IC is running, allowing evaluation of different options without using up the limited OTP memory.

This command translates the configuration file detailed in TMC9660 Boot Configuration File into a sequence of configuration write accesses to the TMC9660.

**Note**: There are two operations that can only be done last for this command—the updating of communication interfaces, which may disconnect the connections of UblTools and the launch of the motor control system, which stops the bootloader operation. When using this command, ensure that at most only one of these two operations happen. Otherwise, you must ensure that the communication does not get severed, or you must configure the TMC9660 in two steps with two invocations of the "*write config*" command.

### Command: write config --burn

Adding the "*--burn*" argument causes the configuration to be burned into OTP memory. **This operation is permanent!**

The TMC9660 offers a total of four burn slots within the IC, with subsequent slots overriding previous ones.

It is suggested to first run this command with the optional "*--dry-run*" argument to ensure that all arguments to the command are correct. Only after confirming, remove the flag to perform the actual burn operation.

By default, the command asks for manual confirmation of the burn operation. This confirmation can be bypassed using the "*--yes*" argument. It is heavily recommended to only use this argument in a production environment, and not during manual evaluation!

### Command: write SPI

This command is used to write to an external SPI Flash.

### Command: write SPI erase

This command erases the full contents of the connected SPI Flash.

### Command: write SPI partition

This command writes a partition table to an external SPI Flash. It automatically erases the area needed to write the partition table. However, it does not erase any old data within the partitions. To ensure correct operation, it is suggested to erase the SPI Flash first.

### Command: write I2C

This command is used to write to an external I2C EEPROM.

### Command: write I2C erase

This command erases the full contents of the connected I2C EEPROM.

**Note**: This command takes a long time, up to multiple minutes with slow connection configurations and large EEPROMs. This is due to the erase operation performing writes to every memory location to clear memory contents.

### Command: write I2C partition

This command writes a partition table to an external I2C EEPROM. It automatically erases the area needed to write the partition table. However, it does not erase any old data within the partitions. To ensure correct operation, it is suggested to erase the I2C EEPROM first.

## Command: start

This command is used to start the motor control system on the TMC9660.

By default, this command starts the motor control system in the mode (parameter or register mode) selected by the active configuration. The optional "*--mode*" argument can be used to override this selection. The "*--mode param"* forces parameter mode, and the "*--mode reg*" forces register mode.

**Note:** The mode selection uses the TMC9660 bootstrap setting BOOT_MODE. This setting defaults to a reserved value in hardware. If this command reads out a reserved value from the hardware, this command, unless overwritten by "*--mode*", changes the BOOT_MODE selection from reserved to parameter mode before launching.

# TMC9660 Boot Configuration File

The bootstrapping through UblTools uses a configuration file. The configuration file is a toml file holding settings to be applied to the TMC9660.

Note that any setting not mentioned in a config file remains unchanged when applying a configuration. However, it is recommended to list the full configuration in a file to simplify the setup procedure.

## Overview of the toml Format

This is a basic overview of the toml file format. It only covers some of the capabilities of the file format.

The toml file format comprises lines of key/value pairs. Keys are simple strings (e.g., enabled) and values can be different types, e.g., booleans (true), integers (255), or strings ("auto16").

Key/value pairs can be grouped using tables (e.g., [uart]) or using a dot to indicate groups.
See the following example config, where the "enabled=true" line is part of the "[uart]" table. This is the same as if writing "uart.enabled=true" without a table declaration.

```
# Example boot configuration: UART settings for a TMC966X
io_config_version = "v0.1"

[uart]
enabled           = true
pin_ic_tx         = 6         # 0, 6
pin_ic_rx         = 7         # 1, 7
baud_rate         = "auto16" # 9600, 19200, 38400, 57600, 115200, 1000000, "auto8", "auto16"
chip_address      = 1         # 1-255
host_address      = 255       # 1-255 (must be different than chip_address)
```

***Figure 4. Example boot configuration***

A full list of available settings can be found in the sections that follow.

## List of Configuration Settings

This section contains the full list of available settings. Any default values are highlighted in **bold**. For some settings, no defaults exist, such as settings disabled by a separate *enable* configuration item.

### App Settings

This setting selects the app mode.

**Table 3. App Setting**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| app_settings. app_type | “param”, “reg” | Selects whether to run the register or parameter mode. |

**Note**: This setting corresponds to the TMC9660 bootstrap setting BOOT_MODE. In the IC, this setting defaults to a reserved value. When using UblTools, the *start* command first reconfigures any reserved setting to the parameter mode selection.

### Bootstrap Settings

These settings concern the application launch by the bootloader.

**Table 4. Bootstrap Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| bootstrap.disable_selftest | **false**, true | Disables internal ROM and RAM self-tests on power-on. This setting only takes effect when burned into the OTP. |
| bootstrap.load_rom_code | **false**, true | Loads and runs the motor control system from ROM application code. |
| bootstrap.fault_on_app_exit | **false**, true | Asserts the FAULTN pin when the motor control application code exits back into the bootloader. |
| bootstrap.fault_on_app_entry | **false**, true | Asserts the FAULTN pin when the motor control application code gets loaded.<br>**Note**: The motor control system de-asserts the FAULTN pin when it is fully initialized. |
| bootstrap.fault_on_config | **false**, true | Asserts the FAULTN pin when the bootloader starts a configuration procedure and de-assert it once completed.<br>**Note**: This allows using the FAULTN pin to identify completion of slower operations such as clock reconfigurations. |

## Low Dropout Voltage Regulators

These settings concern the LDO voltage regulators of the TMC9660.

**Table 5. LDO Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| ldo.vext1_voltage | 0, 2.5, 3.3, 5.0 | Desired output voltage of the VEXT1 pin |
| ldo.vext1_slope_speed | 3.0, 1.5, 0.75, 0.37 | Slope speed in µs of the VEXT1 LDO soft-start. |
| ldo.vext2_voltage | 0, 2.5, 3.3, 5.0 | Desired output voltage of the VEXT2 pin |
| ldo.vext2_slope_speed | 3.0, 1.5, 0.75, 0.37 | Slope speed in µs of the VEXT2 LDO soft-start. |
| ldo.ldo_short_fault | false, true | Assert the FAULTN pin on LDO short. |

## Universal Asynchronous Receiver-Transmitter/UART

These settings configure the UART connection to the TMC9660.

**Table 6. UART Settings**

| **SETTING** | **VALUES** | **DESCRIPTION** |
|---|---|---|
| uart.enabled | false, **true** | Enables UART communication. If enabled, the following settings are required. |
| uart.baud_rate | 9600, 19200, 38400, 57600, 115200, 1000000, "auto8", "**auto16**" | Selects UART communication speed. |
| uart.chip_address | 0 – 255 | Selects UART chip address. Default: 1 |
| uart.host_address | 0 – 255 | Selects UART host address. Default: 255 |
| uart.pin_ic_tx | **6**, 0 | Selects which GPIO to use for Tx. |
| uart.pin_ic_rx | **7**, 1 | Selects which GPIO to use for Rx. |
| uart.pin_ic_txen | 2, 8 | Selects which GPIO to use for Tx enable (optional). |
| uart. txen_pre_delay | 0 – 255 | Required if uart.pin_ic_txen is set. |
| uart. txen_post_delay | 0 – 255 | Required if uart.pin_ic_txen is set. |

### SPI Slave

These settings configure the SPI slave connection to the TMC9660.

**Table 7. SPI Slave Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| spi_slave.enabled | false, **true** | Enables SPI slave connection.<br>If enabled, the following settings are required. |
| spi_slave.spi_block | **"SPI0"**, "SPI1" | Selects whether to use SPI0 or SPI1.<br>SPI0:<br>• MISO: GPIO 9<br>• MOSI: GPIO 10<br>• SCK: GPIO 6 or GPIO 11<br>• CSN: GPIO 12<br>SPI1:<br>• MISO: GPIO 17<br>• MOSI: GPIO 18<br>• SCK: GPIO 14<br>• CSN: GPIO 15 |
| spi_slave.pin_spi0_sck | 6, **11** | Selects whether to use GPIO 6 or GPIO 11 as SCK for SPI0. Required only when spi_flash.spi_block = SPI0. |

### SPI Flash

These settings configure an external SPI Flash memory.

**Table 8. SPI Flash Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| spi_flash.enabled | **false**, true | Enables external SPI Flash memory.<br>When enabled, the additional SPI Flash settings are required. |
| spi_flash.frequency | 2500000, 2666666, 2857142, 3076923 3333333, 3636363 4000000, 4444444 5000000, 5714285 6666666, 8000000, 10000000 | Clock frequency for communicating with the SPI Flash. |
| spi_flash.spi_block | "SPI0", **"SPI1"** | Selects whether to use SPI0 or SPI1.<br>SPI0:<br>• MISO: GPIO 9<br>• MOSI: GPIO 10<br>• SCK: GPIO 6 or GPIO 11<br>SPI1:<br>• MISO: GPIO 17<br>• MOSI: GPIO 18<br>• SCK: GPIO 14 |
| spi_flash.pin_spi0_sck | 6, 11 | Selects whether to use GPIO 6 or GPIO 11 as SCK for SPI0. Required when spi_flash.spi_block = SPI0. |
| spi_flash.pin_cs | 0 – 18 | Selects which GPIO to use as chip select. |

### External Memory Usage

These settings configure how different chip features utilize connected external memory.

**Table 9. External Memory Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| ext_mem.tmcl_script | "spi-flash", "i2c-eeprom" | (optional) Selects where to store tmcl script. |
| ext_mem.parameter_storage | "spi-flash", "i2c-eeprom" | (optional) Selects where to store parameters. |

### HALL
These settings configure hall sensor.

**Table 10. Hall Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| hall.enabled | **false**, true | Enables Hall sensor. If enabled, then the following settings are required. |
| hall. pin_u | 2, 7, 9 | Selects which GPIO to use for Hall U signal. |
| hall. pin_v | 3, 15 | Selects which GPIO to use for Hall V signal. |
| hall. pin_w | 4, 8, 10 | Selects which GPIO to use for Hall W signal. |

### ABN1
These settings configure the ABN1 sensor.

**Table 11. ABN1 Encoder Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| abn1.enabled | **false**, true | Enables the use of ABN1 encoder. |
| abn1.pin_a | 5, 8, 17 | Selects the GPIO for encoder signal A. (Required if abn1.enabled is true) |
| abn1.pin_b | 1, 13, 18 | Selects the GPIO for encoder signal B. (Required if abn1.enabled is true) |
| abn1.pin_n | 14, 16 | Selects the GPIO for encoder signal N. (Optional) |

### I2C EEPROM
These settings configure the external I2C EEPROM memory.

**Table 12. I2C EEPROM Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| i2c_eeprom.enabled | **false**, true | Enables external I2C EEPROM memory. |
| i2c_eeprom.pin_scl | 4, 12, 13 | Selects the GPIO for clock line. |
| i2c_eeprom.pin_sda | 5, 11, 14 | Selects the GPIO for the data line. |
| i2c_eeprom.address | 0 – 7 | Sets the EEPROM address. |
| i2c_eeprom.frequency | 0 – 5 | Hz: 100000 × 2^N. |

### SPI Encoder
These settings configure the SPI Encoder.

**Table 13. SPI Encoder Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| spi_encoder.enabled | **false**, true | Enables the SPI encoder. |
| spi_encoder.spi_block | "SPI0", "SPI1" | Selects whether to use SPI0 or SPI1.<br>SPI0:<br>• MISO: GPIO 9<br>• MOSI: GPIO 10<br>• SCK: GPIO 6 or GPIO 11<br>SPI1:<br>• MISO: GPIO 17<br>• MOSI: GPIO 18<br>• SCK: GPIO 14 |
| spi_encoder.spi_mode | 0 – 3 | SPI mode to use for communication with the SPI encoder |
| spi_encoder.frequency | 2105263, 2222222, 2352941, 2500000, 2666666, 2857142, 3076923, 3333333, 3636363, 4000000, 4444444, 5000000, 5714285, 6666666, 8000000, 10000000 | Clock frequency for communicating with the SPI encoder in Hz |
| spi_encoder.pin_spi0_sck | 6, 11 | Required when spi_block = "SPI0" |
| spi_encoder.pin_cs | 12, 13,<br>16 for spi_block="SPI0", 15 for spi_block="SPI1" | Selects GPIO for chip selection (CS). |

## Phased Locked Loop/PLL

These settings configure the internal PLL.

**Table 14. PLL Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| pll.enabled | false, **true** | Enables the PLL. |
| pll.source | **"IntOsc"**, "ExtOsc", "ExtClk" | Selects the source clock the PLL uses. |
| pll.sys_frequency | 15000000, **40000000** | Selects the system frequency the PLL generates.<br>**Note**: The motor control system requires the system frequency to be 40MHz when launched. |
| pll.ext_frequency | 1000000-32000000 | Frequency in Hz of the external oscillator or clock.<br>Must be a multiple of 1MHz<br>For external oscillator, must be a multiple of 8MHz. |
| pll.xtal_boost | false, true | Whether the maximum drive current gets applied during external oscillator startup. |

## Reference Switches

These settings configure the reference switch mechanism GPIO pins.

**Table 15. Reference Switch Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| ref_switch.pin_left | 2, 12, 16 | (optional) Selects the GPIO for left reference switch. |
| ref_switch.pin_home | 4, 7, 15, 17 | (optional) Selects the GPIO for home reference switch. |
| ref_switch.pin_right | 3, 18 | (optional) Selects the GPIO for right reference switch. |

## Step/Dir

These settings configure the Step/Dir GPIO pins for stepper motor operation.

**Note**: The Step/Dir feature cannot be used together with the ABN2 feature.

**Table 16. Step/Dir Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| step_dir.enabled | **false**, true | Enables the step/dir mode. |
| step_dir.pin_step | 7, 11, 16 | Selects GPIO for step signal. |
| step_dir.pin_dir | 6, 15 | Selects GPIO for direction signal. |

## Watchdog

These settings configure the watchdog.

**Table 17. Watchdog Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| watchdog.enabled | false, **true** | Enables watchdog operation. |
| watchdog.timeout | **250**, 500, 750, 1000, 1250, 1500, 1750, 2000 | Selects timeout duration (ms). |

## Brakechopper

These settings configure the brakechopper.

**Table 18. Brakechopper Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| brakechopper.enabled | **false**, true | Enables brakechopper use. |
| brakechopper.pin_brakechopper | 0 – 18, “Y2” | Selects GPIO or Y2_HS for brake output. |

### Mechanical Brake

These settings configure the mechanical brake.

**Table 19. Mechanical Brake Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| mechanical_brake.enabled | **false**, true | Enables mechanical brake use. |
| mechanical_brake.pin_mech_brake | 3, 10, 18, "Y2" | Selects GPIO or Y2_LS for brake output. |

### ABN2

These settings configure the ABN2 encoder.

**Note**: The ABN2 encoder does not support usage of an N channel.
**Note**: The ABN2 encoder cannot be used together with the Step/Dir feature.

**Table 20. ABN2 Encoder Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| abn2.enabled | **false**, true | Enables ABN2 encoder. |
| abn2.pin_a | 6, 15 | Selects GPIO for encoder Signal A. |
| abn2.pin_b | 7, 11, 16 | Selects GPIO for encoder Signal B. |

### GPIOs

These settings provide configuration for each of the individual GPIO pins (0-18).

**Table 21. GPIO Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| gpioX.type | "input", "output", “analog” | (optional)<br><br>The “analog” option is only available for GPIO 2, 3, 4, and 5 |
| gpioX.output_value | 0, 1 | Required when gpioX.type="output" |
| gpioX.pull_resistor | "disabled", "pulldown", "pullup" | (optional)<br>When the GPIO type is set to “analog,” this may only be set to “disabled.” |

## UBLTOOLS Partition File Format

The memory partitioning through UblTools uses a configuration file that can be applied to TMC9660.

### Partition Table Settings

These settings provide configuration to partition a memory into different sections, each with a dedicated purpose.

See *Overview of the toml Format* for an explanation of the toml format.

The partition description comprises a single toml file in two parts. The first part are values concerning the entire memory. The second part is a list of partitions.

A full example partition file can be found in the Quick Start guide in *Setting Up External Memory*.

**Table 22. Partition Table Settings—Memory Settings**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| partition_config_version | "v1.0" | Version of the partition configuration file. |
| part_table.version | "v1.1" for TMC9660 | Version of the bootloader partition format.<br>Note: This is the version that the bootloader reports with GET_INFO PARTITION_VERSION. |
| part_table.size | e.g., 0x00080000 for 512 KiB | Size of the external memory in bytes. Must be a power of two.<br>The maximum value is 224 for SPI Flash and 216 for I2C EEPROM. |
| part_table.sector_size | e.g., 0x1000 for 4 KiB | SPI Flash: Size of a memory sector in bytes. This refers to the memory that gets erased by a SECTOR_ERASE command.<br>I2C: Set to 1. |

**Table 23. Partition Table Settings—Partitions List**

| SETTING | VALUES | DESCRIPTION |
|---|---|---|
| part_type.partition.name | e.g., “app_settings” | The name of the partition encoded in UTF-8. Can be up to 12 bytes long. |
| part_type.partition.type | “MOTOR_CONFIG”, “TMCL_SCRIPT” | The type of the partition.<br>“MOTOR_CONFIG” allows holding the parameter mode startup values.<br>“TMCL_SCRIPT” allows holding the TMCL script to be run by parameter mode. |
| part_type.partition.offset | e.g., 0x1000 | Offset of the partition in the external memory in bytes. Must be aligned to the sector size. |
| part_type.partition.size | e.g., 0x1000 | Size of the partition in bytes. Must be a multiple of the sector size. |
| part_type.partition.writable | false, true | Whether the partition should be writable at runtime.<br>Note: This bit is just a hint, it is not fully enforced. |

Note: The “part_type.partition” key holds an array of partitions, each with the respective partition keys (“name”, “type”, etc.).