

STM32CubeIDE ST-LINK GDB server

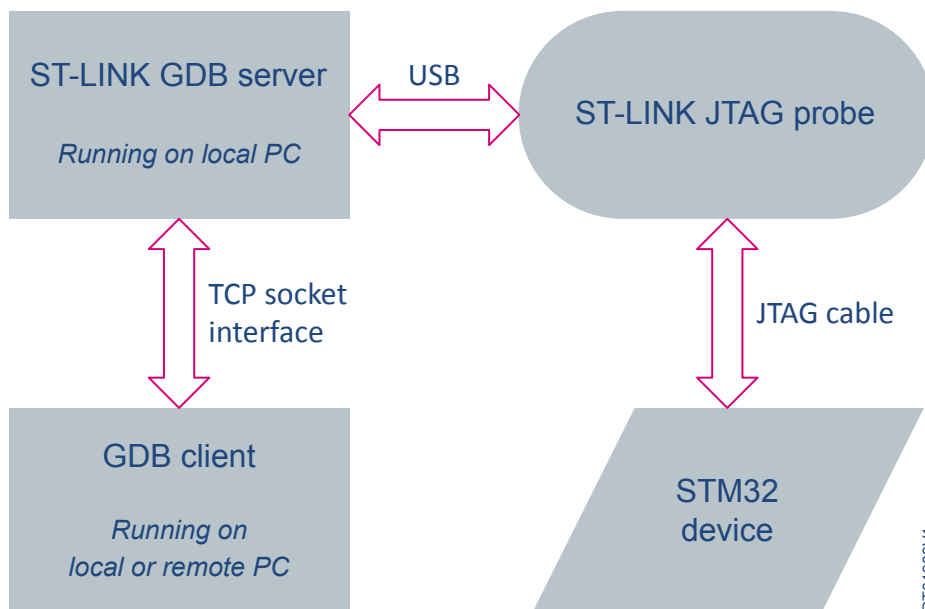
Introduction

The STM32CubeIDE ST-LINK GDB server, also referred to as the GDB server, is a command-line application that runs on a PC connected to the Arm® Cortex®-M target via the ST-LINK JTAG probe.

At startup, the ST-LINK GDB server connects to the STM32 Arm® Cortex®-M target using the ST-LINK JTAG. After establishing the target-side communication, it waits for the client to connect to its TCP-listen socket. Once the client is connected to the TCP-listen socket, the ST-LINK GDB server processes the Remote Serial Protocol (RSP) messages sent by the client, performs the appropriate target-side actions, and sends RSP replies to the client.

Figure 1 shows a typical debug session using the ST-LINK GDB server to debug an Arm® Cortex®-M target using STMicroelectronics ST-LINK probe.

Figure 1. Overview of a debug setup



The figure shows how the GDB client connects to the ST-LINK GDB server via a TCP-socket interface to debug the Arm® Cortex®-M target connected via the ST-LINK JTAG.

1 GDB server usage

The STM32CubeIDE ST-LINK GDB server is a command-line application, which can be started by:

- Entering a set of command-line options
- Instructing the GDB server to load the options from a configuration file

If no options are specified, the GDB server starts with preconfigured default options. The startup options and the default values corresponding to each of them are listed in [Section 1.1: GDB server startup options](#).

The STM32CubeIDE ST-LINK GDB server uses STM32CubeProgrammer (STM32CubeProg) to program the device to debug. The STM32CubeProgrammer software is automatically used by the GDB server when gdb issues a `load` command.

Note: For cases where advanced device control is needed such as erasing flash memory or setting option bytes, STM32CubeProgrammer (STM32CubeProg) can be used. It is included in STM32CubeIDE and can be also downloaded independently from the STMicroelectronics website at www.st.com.

STM32CubeIDE supports STM32 32-bit products based on the Arm® Cortex® processor.

Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries or its affiliates) in the US and/or elsewhere.

The Arm word and logo are trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved.



1.1 GDB server startup options

The GDB server is started with the following command:

```
ST-LINK_gdbserver.exe [options]
```

Most options are available in a short and long form. The set of options that the GDB server supports are described below with default modes and configurations in bold:

- `-h, --help`
Causes the `ST-LINK_gdbserver` to display usage information.
- `--halt`
Halts all cores during reset.
[Use this option with multi-core devices to halt all.]
- `-c <config file>, --config-file <config file>`
Instructs the GDB server to read the options from the configuration file. A sample configuration file illustrating the syntax is provided in [Appendix B](#).
- `-f <log file>, --log-file <log file>`
Specifies the name of the log file. The default name is `debug_log.txt`.
- `-l <log level>, --log-level <log level>`
Specifies the logging level. The logging level is between 0 and 31. Specifying a logging level of 0 turns the logging off, whereas specifying a logging level of 31 enables the full logging. The default logging level is **1**. The logging level can be set as per the following list:
 - = 0: Disables logging.
 - ≥ 1: Enables the logging of error messages.
 - ≥ 2: Adds warning messages.
 - ≥ 4: Adds communication-specific messages.
 - ≥ 8: Adds all information messages.
 - ≥ 16: Adds all hardware-specific messages.

Hence, if the options `-f` and `-l` are not specified, the server starts with a default logging level of 1 (error message logging) and the default log file is `debug_log.txt`.
- `-p <port number>, --port-number <port number>`
Specifies the TCP port number at which the server listens for client connection. The default port number at which the server listens is **61234**.

- `-v, --verbose`
 Specifying `-v` in the command line enables the verbose mode; which, in addition, also prints on the standard output the debug log of the server. By default, the verbose mode is **disabled**.
- `-r <refresh delay>, --refresh-delay <refresh delay>`
 Specifies the interval, in seconds, at which the hardware status is updated in the debug log file. Note that updating the status at very small intervals bloats the debug log file. The default status refresh delay is **5 seconds**.
- `-s, --verify`
 Turns on the verification of the flash memory download.
- `-e, --persistent`
 Specifying the option `-e` starts the server in persistent mode. By default, the persistent mode is disabled, meaning that the server starts in **non-persistent** mode.
- `-d, --swd`
 Enables the Serial Wire Debug (SWD) mode. Specifies that SWD is used instead of JTAG.
[Use this option for SWV.]
- `-z <port number>, --swo-port <port number>`
 Specifies the TCP port number at which the server outputs raw SWO data.
[Use this option for SWV.]
- `-a <cpu clock>, --cpu-clock <cpu clock>`
 Specifies the clock speed of the CPU in Hz.
[Use this option for SWV.]
- `-b <SWO CLOCKDIV>, --swo-clock-div <SWO CLOCKDIV>`
 Specifies the SWO clock divider. This operates on the CPU clock speed given by the `-a` option.
[Use this option for SWV.]
Use the `-z`, `-a`, and `-b` options together to specify the speed of the SWO. For example, if the core runs at 168 MHz and the target SWO at 1 MHz, configure these options as follows: `-z 61235 -a 168000000 -b 168`. For a core running at 72 MHz and SWO at 125 kHz, use `-z 61235 -a 72000000 -b 576`.
- `-k, --initialize-reset`
 Initializes the device while under reset condition.
- `-q, --debuggers`
 List connected debuggers, ST-LINK serial numbers.
- `-i <ST-LINK S/N>, --serial-number <ST-LINK S/N>`
 Specifies the ST-LINK serial numbers that the server shall connect to.
- `--frequency <max freq kHz>`
 Specifies the ST-LINK communication frequency in kHz, such as 5, 25, 100, 240, 950, 1800, 4000, 8000, 24000 kHz.
- `-m <apID>, --apid <apID>`
 Defines the apID to debug.
[Use this option for multi-core.]
- `-g, --attach`
 Attaches to the running target.
 Attach is aimed to connect to a running program in the device without doing a reset or downloading a new program.
[STM32_Programmer_CLI is always started with "mode=UR reset=hwRst" so that a device reset is done when loading a new program independent of this option. This ensures that the device is programmed correctly.]
- `-t, --shared`
 Allows two or more programs to connect to the same device simultaneously using one single ST-LINK probe.
[Use this option for the ST-LINK server.]
- `--erase-all`
 Erases all the memories.
- `--memory-map <device id>`
 Shows the memory map for the given device identifier, such as 0x410.
- `--ext-memory-loaders`
 Provides the list of the available external memory loaders.

- `-el <file_path>, --extload <file_path>`
Selects a custom external memory-loader.
- `-ei`
Enables the external loader **and** uses the initialization logic during a reset operation. In other words, it selects a client external memory loader and calls its `Init()` function after a reset to make the external memory accessible without the need for the application software to configure the access to the memory-mapped external memory. This option can only be used once. Calling the `Init()` function from several external loaders leads to unpredictable behavior.
- `-cp <path>, --stm32cubeprogrammer-path <path>`
Path to the STM32CubeProgrammer (STM32CubeProg) installation.
- `--incremental`
Enables incremental flash memory programming mode.
- `--pcm-pm-force`
Forces flash memory programming if PCM PM bit is set.
- `--target-sel <targetsel>`
Enables SWD multi-drop mode with target selection:
 - For the STM32 devices compatible with SWD protocol v2, the `targetsel` value is based on the target instance number (`DP_DLPIDR[31:28]`) and device identification (`DBGMCU_IDCODE[11:0]`).
 - For example, for STM32U375xx and STM32U385xx devices with a device identification of 0x454, `targetsel` must be set to 0x04540041.
- `--semihosting [terminal|all]`
Enables semihosting and selects semihosting mode. Mode "terminal" for terminal I/O only and mode "all" for terminal and file I/O.
- `--semihost-console-port <port number>`
Specifies the TCP port number for the semihosting console client.
- `--pend-halt-timeout <Pending halt timeout>`
Specifies the maximum time for ST-LINK GDB server to wait for the core to halt (default time is 2 seconds). When using this option, the GDB client must also be started to use longer timeout values. For instance, use a `.gdbinit` file containing the following two lines to get a 50-second timeout:
 - `set remotetimeout 50`
 - `set tcp connect-timeout 50`
- `--temp-path <path>`
Temporary files for starting a debug session are stored at the path provided.
- `--preserve-temps`
Temporary files are not removed.
- `--licenses`
Provides the list of the tools and licenses used.
- `--, --ignore-rest`
Ignores the rest of the labeled arguments following this flag.
- `--version`
Displays version information and exits.

1.2 GDB server modes of operation

The server can be run either in persistent or non-persistent mode depending upon the configuration options. In persistent mode, the server continues to run even after the client disconnects, and waits for new connections, whereas in non-persistent mode the server exits when the client closes the connection. However, as an exception, if the server encounters errors in the target communication, it closes all connections and shuts down, regardless of the mode.

1.3 Starting the GDB server

The GDB server can be started in a command window in the following way:

1. `cd`
`C:\ST\STM32CubeIDE_1.0.0.19w12patch\STM32CubeIDE\plugins\com.st.stm32cube.ide.mcu.externaltools.stlink-gdb-server.win32_1.0.0.201903011553\tools\bin\ST-LINK_gdbserver`
2. Start the GDB server using `-cp` with the path to `STM32_Programmer_CLI.exe`
For instance: `ST-LINK_gdbserver.exe -d -v -cp "C:\ST\STM32CubeIDE_1.0.0.19w12patch\STM32CubeIDE\plugins\com.st.stm32cube.ide.mcu.externaltools.cubeprogrammer.win32_1.0.0.201903011553\tools\bin"`
3. Then, the GDB server connects to the STM32 device using the ST-LINK JTAG and waits for commands from a GDB debug session.

2 Debugging with GDB

This section describes a minimal set of GDB commands, which are needed to debug a program via a GDB client. A full list of commands can be obtained by typing `help` at the GDB command prompt. Alternatively, the GDB user manual is available in the [STM32CubeIDE](#) information center.

The ST-LINK GDB server is tested against versions v7.0, and 8.1 of the GDB client.

2.1 Launching GDB

A command-line GDB client is started by the command

```
>arm-none-eabi-gdb <program.elf>
```

where `<program.elf>` is the program to debug.

2.2 Connecting to the server

The server can be started in either persistent or non-persistent mode:

- If the server is started in non-persistent mode, the client must connect to the server using the command `>target remote <ip-address>:<port-number>`. When the client connects to the server in non-persistent mode using this command, the debug session cannot be restarted without closing both client and server.
- If the server is started in persistent mode, the client must connect to the server using the command `>target extended-remote <ip-address>:<port-number>`. When the client connects to the server in persistent mode using this command, the user has the facility to restart the debug session without closing the server or client. In this mode of operation for client and server, the debug session can be repeatedly restarted.

2.3 Loading the program on the target

The executable program is loaded on the target using the GDB command

```
>load <program.elf>
```

This command loads the program on the target. If the code segment of the program lies in the Flash ROM, then the respective pages of the flash memory are erased before the new code is written.

Note: The flash memory programming (the code download into the flash memory) is handled by the GDB server using *STM32CubeProgrammer (STM32CubeProg)* and is transparent to the user.

2.3.1 Loading program to external memory

When a program must be loaded to an external memory, the option `--extload <file_path>` must be used. External memory loader files are delivered for most available STM32 Evaluation and Discovery boards containing external memory. To list the available loader files, use the `--ext-memory-loaders` option.

The *STM32CubeProgrammer software description* user manual ([UM2237](#)) contains detailed information on how to use external flash loader programs and develop customized loaders for external memory. The user manual is available from the [STM32CubeProg](#) webpage on www.st.com.

2.3.2 Memory read/write to external memory

To make external memory accessible for read and write by the debugger, use the `--external-init` option together with external memory loader `--extload <file_path>` option. When `--external-init` is used the `Init()` function in the external memory loader is run by the ST-LINK GDB server after reset. This makes external memory accessible without any need for application software to set up the device for external memory access. The external memory must be memory mapped and directly accessed because normal memory read/write is used by gdb.

Note: When using the `--external-init` option with the ST-LINK GDB server, the stack required by the `Init()` function is limited to 1024 bytes of stack instead of 400 bytes in ST-LINK GDB server v5.4.0.

2.4 Setting breakpoints

Breakpoints are placed using the GDB `break` command

```
> break [LOCATION]
```

where, `LOCATION` is a line number, function name, or "*" followed by an address. If a line is specified, GDB breaks at the start of the code for that line. If a function is specified, GDB breaks at the start of the code for that function. If an address is specified, GDB breaks at the exact address.

The `break` command places a software breakpoint at the specified location. This is valid if the code segment lies in RAM. However, if the code segment lies in the Flash ROM, then the `hbreak` command should be used instead of the `break` command. The arguments for the `hbreak` command are the same as those of the `break` command.

Note: *The maximum number of hardware breakpoints that can be simultaneously placed depends on the debugged STM32 device. Note that there is no such limit on software breakpoints.*

2.5 Setting watch-points

Variable read, write, and access operations are tracked with the commands

```
>rwatch <variable-name>, which halts the program if the variable is read,
```

```
>watch <variable-name>, which halts the program if the variable is written to,
```

```
>awatch <variable-name>, which halts the program if an access to the variable occurs.
```

Note: *The maximum number of watch-points that can be simultaneously placed depends on the hardware configuration.*

2.6 Running the program

Once the program is loaded, the debugger starts execution through the command

```
>continue
```

Note: *There is no mechanism for specifying command-line parameters to the program.*

The user can halt the program by pressing `Ctrl-C` while it is running. The user can step over a statement by using the `next` command. The user can also step inside a function by using the `step` command.

2.7 Exiting a debug session

The user exits the debug session by issuing the GDB command `quit`.

2.8 Debugging on the target STM32 board

The ST-LINK GDB server supports the debug of STM32 Arm[®] Cortex[®]-M-based devices. Both debugging in RAM and flash memory are supported. Flash memory breakpoints are supported by the Arm[®] Cortex[®]-M hardware breakpoints. Application code can be compiled to run from either the RAM or the flash memory. No special commands are needed to distinguish between RAM- and flash-mode codes as GDB handles this internally.

For the debug of code in RAM or flash memory on an Arm[®] Cortex[®]-M microcontroller, the stack pointer (SP, R13) and program counter (PC, R15) must be initialized correctly:

- If the code to debug is located in flash memory, the SP is read from 0x0 and the PC from 0x4.
- If the code to debug is located in RAM, typically the start of the vector is the start of the RAM area used for download. On most devices, this is 0x20000000. This means that the initial value of the stack pointer (SP) can be read from 0x20000000 and the initial value of the PC can be read from 0x20000004.

Use the command

```
>monitor reset
```

after downloading the code to set the SP and PC correctly. [STM32CubeIDE](#) issues this command automatically after downloading the code to the target device.

2.9 Monitor commands

The GDB monitor commands are supported and implement target-specific features, particularly for executing flash memory controller commands.

The following commands are the monitor commands supported for the ST-LINK GDB server:

- `>monitor help`
Print information about the supported monitor commands.
- `>monitor reset`
Reset the system by a software write to a register. This resets the core and peripherals. The whole system can be reset if the reset pin of the target is asserted.
- `>monitor reset core`
Reset the core by a software write to a register (not possible on Cortex[®]-M0, Cortex[®]-M0+ and Cortex[®]-M33 cores). This only resets the core, not the peripherals or the reset pin.
- `>monitor reset hardware`
Reset the hardware activated by the reset pin. The ST-LINK reset line is activated and deactivated with a pulse.
- `>monitor flash mass_erase`
Used to erase all the pages of the flash memory device. Care needs to be taken when using this command since it causes all data stored in the flash memory to be lost.
- `>monitor ReadAP <register index>`
Read CoreSight AP register.

The following commands are not yet supported and always return OK:

- `>monitor ReadDP <register index>`
Read CoreSight DP register.
- `>monitor WriteAP <register index> <data32>`
Write CoreSight AP register.
- `>monitor WriteDP <register index> <data32>`
Write CoreSight DP register.

3 Troubleshooting

In case the server hangs or fails to start, follow these steps:

1. Shut down the GDB server and any client attached to the server.
2. Power off the target board.
3. Disconnect the ST-LINK JTAG USB cable.
4. Reconnect the ST-LINK JTAG USB cable.
5. Power on the target board.
6. Check the debug configuration settings, such as the SWD/JTAG interface
7. Start the GDB server, preferably using a different port number.

A good practice is to try debugging another project to exclude a project-specific issue with the project configuration or startup code.

We also recommend debugging with another USB cable, ST-LINK, or STM32 hardware to exclude hardware failures as the reason for the debugging issues.

It is also advised using some other tool from STMicroelectronics, such as the STM32CubeProgrammer (STM32CubeProg), to connect to the target. If STM32CubeProgrammer can connect to the target, try to:

1. Erase the flash memory with STM32CubeProgrammer.
2. Program a new program with STM32CubeProgrammer.
3. If the programming is successful, erase the flash memory again with STM32CubeProgrammer.
4. Disconnect STM32CubeProgrammer from the target.
5. Try to debug the program using `ST-LINK_gdbserver`.

Appendix A Return/error code information

Table 1. ST-LINK GDB server return/error codes

Code	Definition	Description
0	TARGET_SUCCESS	OK.
1	TARGET_CONNECT_ERR	Port is in use, something is already connected to the board, such as STM32CubeProgrammer.
2	TARGET_DLL_ERR	No ST-LINK connected.
3	TARGET_USB_COMM_ERR	USB communication error.
4	TARGET_NO_DEVICE	ST-LINK is connected but no board is connected.
5	TARGET_UNKNOWN_MCU_TARGET	For instance, not using option <code>-d</code> when only SWD pins are connected to the HW.
6	TARGET_FIRMWARE_OLD	Old ST-LINK_firmware.
7	TARGET_RESET_ERR	Reset error.
8	TARGET_HELD_UNDER_RESET	Reset button on board pressed.
9	TARGET_NOT_HALTED	Target could not be halted.
10	TARGET_CMD_ERR	Command error.
11	TARGET_APP_RESET_ERR	Application reset error.
12	TARGET_VERSION_ERR	Version error.
13	TARGET_GET_STATUS_ERR	Get status error.
14	TARGET_FORCE_HALT_ERR	Force halt error.
16	TARGET_STLINK_SELECT_REQ	Several boards are connected. Need to specify <code>-i <ST-LINK S/N></code> .
17	TARGET_STLINK_SERIAL_NOT_FOUND	The ST-LINK serial number cannot be found when using the <code>-i</code> option.
18	TARGET_DEVICE_UNKNOWN_VENDOR	The vendor information cannot be verified.
255	TARGET_UNKNOWN_ERR	Unknown error.

Appendix B Configuration file format

This appendix describes the configuration file format for starting the ST-LINK GDB server from a configuration file. The configuration file format is briefly depicted by the following rules:

- Each line starting with a # is treated as a comment.
- Each configuration option (refer to [Section 1.1: GDB server startup options](#)) is specified in a single line.

A sample configuration file is shown below:

```
#####
### Sample Configuration File
#####

#####
# -e : Enables persistent mode
#####
-e

#####
# -f <Log-File> : Name of log file.
#####
-f debug_log_v0.txt

#####
# -l <Log-Level> : Logging level between 0 & 31
#####
-l 31

#####
# -p <Port-Number> : TCP-Listen Port-Number.
#####
-p 61234

#####
# -v : Enables verbose mode
#####
-v

#####
# -r <refresh-delay-sec> : Maximum Delay in status refresh
#####
-r 1

#####
# -cp <path> : Path to STM32CubeProgrammer
#####
-cp C:\ST\STM32CubeIDE_1.7.0.21w21\STM32CubeIDE\plugins\
com.st.stm32cube.ide.mcu.externaltools.cubeprogrammer.win32_2.0.0.202105061353\
tools\bin
```

Revision history

Table 2. Document revision history

Date	Revision	Changes
18-Apr-2019	1	Initial release.
11-Oct-2019	2	Updated <i>Section 1.1 GDB server start-up options</i> with more options.
21-Feb-2020	3	Updated <i>Section 1.1 GDB server start-up options</i> with some new options. Added <i>Section 2.3.1 Loading program to external memory</i> and <i>Section 2.3.2 Memory read/write to external memory</i> .
05-Jul-2021	4	Updated <i>Section 2.9</i> with new monitor reset commands and <i>Appendix B</i> with <code>-cp <path></code> option.
14-Nov-2024	5	Updated the list of options in <i>Section 1.1: GDB server start-up options</i> : <ul style="list-style-type: none"> • Added the <code>-ei</code> option • Removed the <code>--external-init</code> option
15-Oct-2025	6	Updated the list of options in <i>Section 1.1: GDB server startup options</i> : <ul style="list-style-type: none"> • Added the <code>--incremental</code> option • Added the <code>--semihosting</code> option • Added the <code>--semihost-console-port</code> option • Updated the <code>--log-level</code> default value Minor text edits across the document.
27-Jan-2026	7	Updated the list of options in <i>Section 1.1: GDB server startup options</i> : <ul style="list-style-type: none"> • Added the <code>--pcm-pm-force</code> option • Added the <code>--target-sel</code> option

Contents

1	GDB server usage	2
1.1	GDB server startup options	2
1.2	GDB server modes of operation	4
1.3	Starting the GDB server	5
2	Debugging with GDB	6
2.1	Launching GDB	6
2.2	Connecting to the server	6
2.3	Loading the program on the target	6
2.3.1	Loading program to external memory	6
2.3.2	Memory read/write to external memory	6
2.4	Setting breakpoints	7
2.5	Setting watch-points	7
2.6	Running the program	7
2.7	Exiting a debug session	7
2.8	Debugging on the target STM32 board	7
2.9	Monitor commands	8
3	Troubleshooting	9
Appendix A	Return/error code information	10
Appendix B	Configuration file format	11
	Revision history	12
	List of tables	14
	List of figures	15

List of tables

Table 1.	ST-LINK GDB server return/error codes	10
Table 2.	Document revision history	12

List of figures

Figure 1. Overview of a debug setup 1

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

The purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of the purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers’ market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2026 STMicroelectronics – All rights reserved