SCPI Commands

Generated by Doxygen 1.9.5

Chapter 1

SCPI Command Documentation Index

This document provides an overview and quick links to all SCPI command documentation files for the Melexis IO STM32 firmware. Each linked file describes the SCPI command set for a specific subsystem or module.

1.0.1 SCPI Format and Communication

SCPI commands follow a standardized format for communication:

- Each command consists of a command keyword, followed by a space, then one or more arguments separated by commas.
- Commands must be terminated with a line feed (LF, `\n`).
- Example: `:EEPROM:STRing wifi.ssid,MySSID\n`
- The response to a command is sent with a line feed before the prompt, either `\n(OK)>` for success or `\n(Some error)>` for failure, with an error message if applicable.
- Commands are case-insensitive; mixed-case patterns are accepted by the parser.

Syntax:
```
«COMMAND» «argument1»[,«argument2»[,«argument3»...]]\n
Example: :EEPROM:STRing wifi.ssid,MySSID\n
```

Response:
```
\n(OK)>
```

or
```
\n(Some error)>
```

All command and response lines are terminated with LF (`\n`). This is required for correct operation over serial or terminal interfaces.

1.0.2 SCPI Command Modules

- SCPI Commands: System
- SCPI Commands: ADC
- SCPI Commands: GPIO
- SCPI Commands: I2C
- SCPI Commands: PWM
- SCPI Commands: SCPI/IEEE
- SCPI Commands: SPI
- SCPI Commands: UART
- SCPI Commands: EEPROM

1.0.3 How to Use

- Click on a module above to view detailed documentation for its SCPI commands, usage, and examples.

Last updated: October 2, 2025

Chapter 2

SCPI Commands in <tt>commands_adc.c</tt>

This document describes the SCPI-like command interface implemented in `commands_adc.c` for controlling and reading the ADC (Analog-to-Digital Converter) pins on the STM32 microcontroller.

2.1 Overview

The `commands_adc.c` file provides a set of commands for initializing ADC pins, configuring their sample times, and reading their analog voltage values. These commands are accessible via a terminal or remote interface that supports the defined command patterns.

2.2 Supported Commands

Command Pattern	Handler	Tag	Description
`:A0:ADC:INIT`	CMD_AdcInit	ADC_A0_PIN	Initialize ADC A0 pin
`:A1:ADC:INIT`	CMD_AdcInit	ADC_A1_PIN	Initialize ADC A1 pin
`:A3:ADC:INIT`	CMD_AdcInit	ADC_A3_PIN	Initialize ADC A3 pin
`:A0:ADC:SampleTime`	CMD_AdcSampTime	ADC_A0_PIN	Set sample time for ADC A0 pin
`:A1:ADC:SampleTime`	CMD_AdcSampTime	ADC_A1_PIN	Set sample time for ADC A1 pin
`:A3:ADC:SampleTime`	CMD_AdcSampTime	ADC_A3_PIN	Set sample time for ADC A3 pin
`:A0:ADC:SampleTime?`	CMD_AdcSampTimeQ	ADC_A0_PIN	Query sample time for ADC A0 pin
`:A1:ADC:SampleTime?`	CMD_AdcSampTimeQ	ADC_A1_PIN	Query sample time for ADC A1 pin
`:A3:ADC:SampleTime?`	CMD_AdcSampTimeQ	ADC_A3_PIN	Query sample time for ADC A3 pin
`:A0:ADC?`	CMD_AdcRead	ADC_A0_PIN	Read voltage from ADC A0 pin
`:A1:ADC?`	CMD_AdcRead	ADC_A1_PIN	Read voltage from ADC A1 pin
`:A3:ADC?`	CMD_AdcRead	ADC_A3_PIN	Read voltage from ADC A3 pin

» Note: Commands for A2 are present in the code but commented out or marked as TODO.

2.3 Command Details

2.3.1 Initialization

- `:A0:ADC:INIT`, `:A1:ADC:INIT`, `:A3:ADC:INIT`
- Initializes the specified ADC pin for analog input.

2.3.2 Set Sample Time

- `:A0:ADC:SampleTime «cycles»` (cycles: 3, 15, 28, 56, 84, 112, 144, 480)

- Sets the ADC sample time for the specified pin.

2.3.3 Query Sample Time

- `:A0:ADC:SampleTime?`
- Returns the current sample time setting for the specified pin.

2.3.4 Read ADC Value

- `:A0:ADC?`
- Reads and prints the voltage value from the specified ADC pin.

2.4 Handler Functions

- **CMD_AdcInit**: Configures the GPIO and ADC hardware for the selected pin.
- **CMD_AdcSampTime**: Sets the sample time for the selected ADC pin.
- **CMD_AdcSampTimeQ**: Queries the current sample time for the selected ADC pin.
- **CMD_AdcRead**: Reads the voltage from the selected ADC pin and prints the result.

2.5 Examples

2.5.1 Initialize ADC Pin

Command sent:
```
:A0:ADC:INIT\n
```
Response:
```
(OK)>
```

2.5.2 Set Sample Time

Command sent:
```
:A0:ADC:SampleTime 56\n
```
Response:
```
(OK)>
```

2.5.3 Query Sample Time

Command sent:
```
:A0:ADC:SampleTime?\n
```
Response:
```
SampleTime=56
(OK)>
```

2.5.4 Read ADC Value

Command sent:
```
:A0:ADC?\n
```
Response:
```
A0=1.234
(OK)>
```

2.5.5 Error Example (Invalid Pin)

Command sent:
```
:A2:ADC:INIT\n
```
Response:
```
(Some error)>
```

2.6 Notes

- The command interface is extensible. To add new ADC pins or features, update the command table and implement the corresponding handler logic.
- Some features (e.g., A2 pin support) are marked as TODO and may require hardware-specific handling.

Chapter 3

SCPI Commands in <tt>commands_application.c</tt>

This document describes the SCPI command interface implemented in `commands_application.c` for system-level operations on the Melexis IO STM32 platform.

3.1 Overview

The `commands_application.c` file provides SCPI-like commands for system information, help, time management, benchmarking, bootloader access, and pin layout display. These commands are accessible via a terminal or remote interface that supports the defined command patterns.

3.2 Supported Commands

Command Pattern	Handler	Description
`?SYSTEM`	(none)	System commands group header
`:SYSTem:INFO`	CMD_Info	System information
`:SYSTem:HELP:LIST`	CMD_HelpList	List of commands
`:SYSTem:HELP`	CMD_Help	Brief help for particular commands
`:SYSTem:TIME?`	CMD_DateTimeQ	Get Date/Time (if RTC enabled)
`:SYSTem:TIME`	CMD_DateTime	Set Date/Time (if RTC enabled)
`:SYSTem:TIME:MS?`	CMD_TimeMsQ	Get system ticks in milliseconds
`:SYSTem:TIME:US?`	CMD_TimeUsQ	Get system ticks in microseconds
`:SYSTem:TIME:MSSleep`	CMD_TimeSleepMs	Sleep for specified milliseconds
`:SYSTem:TIME:USSleep`	CMD_TimeSleepUs	Sleep for specified microseconds
`:SYSTem:BenchMark`	CMD_Benchmark	System benchmark tests
`:SYSTem:DFU`	CMD_Bootloader	Update application (DFU mode)
`:SYSTem:Pins`	CMD_Pins	Display pins layout

» Note: Some commands are only available if certain features (e.g., RTC) are enabled at compile time.

3.3 Command Details

3.3.1 System Information

- `:SYSTem:INFO`: Prints system statistics, including runtime stats and free heap size.

3.3.2 Help

- `:SYSTem:HELP:LIST`: Lists all available commands for auto-complete and reference.
- `:SYSTem:HELP`: Prints help for all commands, including descriptions.

3.3.3 Time and Date

- `:SYSTem:TIME?`: Gets the current date and time (if RTC is enabled).
- `:SYSTem:TIME`: Sets the current date and time (if RTC is enabled).
- `:SYSTem:TIME:MS?`: Gets the system tick count in milliseconds.
- `:SYSTem:TIME:US?`: Gets the system tick count in microseconds.
- `:SYSTem:TIME:MSSleep «ms»`: Sleeps for the specified number of milliseconds (1 ms to 30 sec).
- `:SYSTem:TIME:USSleep «us»`: Sleeps for the specified number of microseconds (1 us to 30 sec).

3.3.4 Benchmark

- `:SYSTem:BenchMark`: Runs memory, flash, and floating-point operation benchmarks and prints timing results.

3.3.5 Bootloader

- `:SYSTem:DFU «value»`: Enters DFU (Device Firmware Update) mode if the value is 42.

3.3.6 Pins Layout

- `:SYSTem:Pins`: Prints a diagram of the board's pin layout for reference.

3.4 Examples

3.4.1 Get System Information

Command sent:
```
:SYSTem:INFO\n
```
Response:
```
System: STM32F446
Heap: 123456 bytes free
(OK)>
```

3.4.2 List All Commands

Command sent:
```
:SYSTem:HELP:LIST\n
```
Response:
```
:EEPROM:INIT
:EEPROM:ERASE
:EEPROM:STRing ...
... (list continues)
(OK)>
```

3.4.3 Get System Time (RTC enabled)

Command sent:
```
:SYSTem:TIME?\n
```
Response:
```
2025-10-02 14:23:45
(OK)>
```

3.4.4 Set System Time (RTC enabled)

Command sent:
```
:SYSTem:TIME 2025-10-02 14:23:45\n
```
Response:
```
(OK)>
```

3.4.5 Run Benchmark

Command sent:
```
:SYSTem:BenchMark\n
```
Response:
```
RAM: 1234 us
FLASH: 5678 us
FPU: 910 us
(OK)>
```

3.4.6 Error Example (Invalid Command)

Command sent:
```
:SYSTem:TIME:INVALID\n
```
Response:
```
ERR: unknown command
(Some error)>
```

3.5 Notes

- Commands are case-insensitive.
- The command interface is extensible; new system commands can be added by updating the command table and implementing the corresponding handler logic.
- Some commands require specific hardware features (e.g., RTC) to be enabled.

Chapter 4

EEPROM SCPI Command Reference

Compact reference for the EEPROM emulation interface. The module stores a JSON document in RAM (`eeprom_buffer`) and persists versioned snapshots as binary records in a dedicated Flash sector.

> On system startup / reset the firmware loads the last valid persisted record (latest intact) into the RAM buffer automatically.

4.1 </blockquote>

4.2 1. Binary Record Layout

Each persisted record (little endian):

```
uint32 magic       = 0x1504
uint32 json_len    = N (bytes, excluding terminating NUL)
uint32 crc32       = zlib_crc32(JSON, start=ZLIB_CRC32_START)
uint8  json[N]     // UTF-8 JSON payload
uint8  0x00        // terminating NUL
padding 0..3 bytes (0x00) to 4-byte alignment
```

Records are appended sequentially until no space remains or corruption is detected. An erased word reads 0xFFFFFFFF.

4.3 2. Command Summary

Command	Parameters	Action
`:EeProm:INIT`	∗(none or ,<index>)∗	Load latest record (no param / -1) or 0-based record index.
`:EeProm:ERASE`	–	Clear RAM buffer only (Flash untouched).
`:EeProm:DUMP`	–	Pretty-print current RAM JSON.

| `:EeProm:SAVE` | ∗(optional ,<erase_all 0\|1>)∗ | Persist RAM JSON (append or force erase). | | `:EeProm:RECords?` | – | List all valid / first corrupt Flash records. | | `:EeProm:Object? <key>` | key | Pretty-print object at top-level key. | | `:EeProm:String <key[.prop]>,<val>` | key,value | Set string. | | `:EeProm:Integer <key[.prop]>,<val>` | key,value | Set integer. | | `:EeProm:Float <key[.prop]>,<val>` | key,value | Set float. | | `:EeProm:Boolean <key[.prop]>,<val>` | key,value | Set boolean (0/1/true/false/on/off/yes/no). | | `:EeProm:String? <key[.prop]>` | key | Get string. | | `:EeProm:Integer? <key[.prop]>` | key | Get integer. | | `:EeProm:Float? <key[.prop]>` | key | Get float. | | `:EeProm:Boolean? <key[.prop]>` | key | Get boolean. | | `:EeProm:DELete <key[.prop]>` | key | Delete key or property. |

Notes:

- Keys are case-sensitive; command mnemonics are case-insensitive.
- Nested access uses `.` (e.g. `net.port`). Missing objects are auto-created on set.

4.4 3. INIT Behavior

`:EeProm:INIT` scans the sector, validating consecutive records:

1. Stop at first 0xFFFFFFFF (free space) or structural/CRC error.
2. Latest fully valid record is chosen by default (this same logic is run automatically at startup).
3. `:EeProm:INIT,<n>` loads record index *n* (0-based) if valid; aborts if out of range.
4. If the newest record is corrupt, the last prior valid record is loaded.

Return messages (examples):

```
EEPROM loaded latest record len=400 crc=0x99887766
EEPROM loaded record 2 len=128 crc=0x1A2B3C4D
EEPROM loaded previous valid record len=256 crc=0x55AA9911 (newest corrupted)
EEPROM record 5 not found
EEPROM empty (no records)
EEPROM corrupted -> cleared
```

4.5 4. SAVE Behavior

Syntax: `:EeProm:SAVE` or `:EeProm:SAVE,<erase>` where `<erase>` is 0 or 1.
Flow:

1. If RAM buffer is empty (`""`), it is converted to `{}` before saving.
2. If `<erase>` = 1: erase sector, write record at offset 0 (no duplicate check).
3. Else: scan for first free slot; validate chain.
4. If last record matches (same length, CRC, and bytes) → skip (no Flash write).
5. If insufficient space → erase sector then write at offset 0.
6. On corruption during scan → erase sector then write at offset 0.

Messages:

```
EEPROM unchanged (skip save) len=400 crc=0x99887766
EEPROM saved: json=420 bytes crc=0xDEADBEEF total=436 bytes @offset=0x0000 (forced erase)
EEPROM saved: json=128 bytes crc=0x12345678 total=140 bytes @offset=0x01B4 (sector erased)
```

Empty JSON example:

```
:EeProm:ERASE
:EeProm:SAVE
--> EEPROM saved: json=2 bytes ...
```

4.6 5. RECords? Output

`:EeProm:RECords?` prints table until free space or first corruption:

```
Idx  Offs  Len   CRC        Status
 0   0x000  256  0x1234ABCD  OK
 1   0x110  300  0x89EF0123  OK
 2   0x23C  ----  --------   CORRUPT (bad magic 0xFFFF0000)
Summary: valid=2 total_scanned=3 (stopped on corruption)
```

Status meanings:

- OK – Structure + CRC valid.
- BADCRC – CRC mismatch (scan stops).
- CORRUPT – Structural issue (magic, length, terminator, overflow).

4.7 6. GET / SET Formats

Outputs:

- String: `key="value"`
- Integer: `key=123`
- Float: `key=1.23`
- Boolean: `key=0|1` On success for set/delete: `OK`.

Error samples:

```
ERR: get 'wifi.ssid' not found
ERR: set 'val' buffer too small
ERR: invalid integer
ERR: delete 'wifi.mode' not found
```

4.8 7. Deletion

`:EeProm:DELete key` removes whole key. `:EeProm:DELete parent.child` removes only `child` from object `parent`. Nonexistent targets report an error.

4.9 8. Edge Cases & Limits

Case	Handling
Empty buffer save	Auto-converted to `{}`.
Duplicate save	Skipped if identical (wear reduction).
Full sector	Auto-erase (unless skip) then write at offset 0.
Corruption mid-chain	Stop scan; latest prior valid used; next save may erase.
Oversized JSON	Rejected with error before Flash write.

Maximum JSON length per record: `EE_SECTOR_SIZE - 12 - 1 - padding`.

4.10 9. Typical Session

```
:EeProm:INIT
:EeProm:String device.name,NodeA
:EeProm:Integer net.port,502
:EeProm:SAVE
:EeProm:String device.name,NodeA   (unchanged)
:EeProm:SAVE        --> skip
:EeProm:RECords?
:EeProm:INIT,0      (load oldest)
:EeProm:SAVE,1      (force new baseline)
```

4.11 10. Return Codes (Internal)

- Success operations return HAL_OK (0) to caller layer.
- Flash / validation failures return HAL_ERROR (non-zero) and print an error if verbose.

4.12 11. Verbosity

Internal helper APIs may suppress printing when `verbose == 0`. SCPI command layer typically prints always.
End of EEPROM SCPI reference.

Chapter 5

SCPI Commands in <tt>commands_gpio.c</tt>

This document describes the SCPI command interface implemented in `commands_gpio.c` for GPIO (General Purpose Input/Output) control on the Melexis IO STM32 platform.

5.1 Overview

The `commands_gpio.c` file provides SCPI-like commands for initializing, deinitializing, reading, and writing to GPIO pins. The commands support both board and connector pins, and allow configuration as input or output.

5.2 Supported Commands

5.2.1 Initialization

Command Pattern	Description
`:A0:GPIO:INIT:IN`	Init A0 as GPIO Input
`:A0:GPIO:INIT:OUT`	Init A0 as GPIO Output
...	...
`:CON:SCL:GPIO:INIT:IN`	Init Connector SCL as GPIO Input
`:CON:SCL:GPIO:INIT:OUT`	Init Connector SCL as GPIO Output

5.2.2 Deinitialization

Command Pattern	Description
`:A0:GPIO:DEINIT`	DeInit A0 as GPIO
...	...
`:CON:SCL:GPIO:DEINIT`	DeInit Connector SCL as GPIO

5.2.3 Output (Write)

Command Pattern	Description
`:A0:GPIO «1/0»`	Set A0 GPIO Output
...	...
`:CON:SCL:GPIO «1/0»`	Set Connector SCL GPIO Output

5.2.4 Input (Read)

Command Pattern	Description
`:A0:GPIO?`	Read A0 GPIO Input

Command Pattern	Description
...	...
:CON:SCL:GPIO?	Read Connector SCL GPIO Input

» Note: Replace A0, A1, A2, A3, SDA, SCL, TX, RX, MO, MI, SCK, CON:SCL, CON:SDA, CON:SCLK, CON:MO, CON:MI, CON:CS0, CON:CS1 with the desired pin name.

5.3 Command Details

- **INIT:IN/OUT**: Initializes the specified pin as input or output.
- **DEINIT**: Deinitializes the specified pin, returning it to analog mode.
- **GPIO «1/0»**: Sets the output state of the specified pin (1 = high, 0 = low).
- **GPIO?**: Reads the input state of the specified pin and prints 1 (high) or 0 (low).

5.4 Handler Function

All commands are handled by the CMD_GPIO function, which interprets the tag to determine the pin, direction, and action.

5.5 Example Usage

- Initialize A0 as output: :A0:GPIO:INIT:OUT
- Set A0 high: :A0:GPIO 1
- Read A0: :A0:GPIO?
- Deinitialize A0: :A0:GPIO:DEINIT
- Initialize Connector SCL as input: :CON:SCL:GPIO:INIT:IN

5.6 Examples

5.6.1 Initialize Pin as Output

Command sent:
```
:A0:GPIO:INIT:OUT\n
```
Response:
```
(OK)>
```

5.6.2 Set Pin Output High

Command sent:
```
:A0:GPIO 1\n
```
Response:
```
(OK)>
```

5.6.3 Read Pin Input

Command sent:
```
:A0:GPIO?\n
```
Response:
```
A0=1
(OK)>
```

5.6.4 Deinitialize Pin

Command sent:
```
:A0:GPIO:DEINIT\n
```
Response:
```
(OK)>
```

5.6.5 Error Example (Invalid Pin Name)

Command sent:
```
:Z9:GPIO:INIT:OUT\n
```
Response:
```
ERR: unknown pin
(Some error)>
```

5.7 Notes

- The command interface is extensible; new pins can be added by updating the command table.
- Commands are case-insensitive.
- Some connector pins are prefixed with `CON:`.

Chapter 6

SCPI Commands in <tt>commands_i2c.c</tt>

This document describes the SCPI command interface implemented in `commands_i2c.c` for I2C and FMPI2C control on the Melexis IO STM32 platform.

6.1 Overview

The `commands_i2c.c` file provides SCPI-like commands for initializing, deinitializing, configuring, and performing data transfers over the I2C and FMPI2C interfaces. The commands support both the main FMPI2C peripheral and a secondary I2C2 peripheral, with options for memory and direct read/write operations, timing configuration, and advanced I2C features.

6.2 Supported Commands

6.2.1 FMPI2C (I2C1)

Command Pattern	Description
`:I2C:INFO`	Print FMPI2C info (clock, timing, filters, etc.)
`:I2C:INIT`	Initialize FMPI2C (pins SCL/SDA)
`:I2C:DEINIT`	Deinitialize FMPI2C
`:I2C:TimingRegister «val»`	Set timing register (uint32)
`:I2C:TimingRegister?`	Get timing register
`:I2C:FREQuency «freq»`	Set I2C frequency (e.g. 100k, 400k, 1MA)
`:I2C:FastModePlus «bool»`	Set fast mode plus (ON/OFF, 1/0, TRUE/FALSE)
`:I2C:FastModePlus?`	Get fast mode plus status
`:I2C:AnalogFilter «bool»`	Set analog filter (ON/OFF, 1/0, TRUE/FALSE)
`:I2C:AnalogFilter?`	Get analog filter status
`:I2C:DigitalFilter «0-15»`	Set digital filter coefficient
`:I2C:DigitalFilter?`	Get digital filter coefficient
`:I2C:MemReaD «dev,mem,sz,cnt»`	Read memory (dev addr, mem addr, mem addr size, count)
`:I2C:MemWRite «dev,mem,sz,data»`	Write memory (dev addr, mem addr, mem addr size, data bytes)
`:I2C:ReaD «dev,cnt»`	Read from device (dev addr, count)
`:I2C:WRite «dev,data»`	Write to device (dev addr, data bytes)
`:I2C:EXCHange «dev,cnt,data»`	Write and/or read to/from device (dev addr, count, data bytes, ...)

6.2.2 I2C2

Command Pattern	Description
`:I2C2:INFO`	Print I2C2 info

Command Pattern	Description
:I2C2:INIT	Initialize I2C2 (pins A2/A3)
:I2C2:DEINIT	Deinitialize I2C2
:I2C2:ClockSpeed «val»	Set I2C2 clock speed (Hz)
:I2C2:AddressingMode «0/1»	Set addressing mode (0=7BIT, 1=10BIT)
:I2C2:DualAddressMode «bool»	Set dual address mode (ON/OFF, 1/0, TRUE/FALSE)
:I2C2:GeneralCallMode «bool»	Set general call mode (ON/OFF, 1/0, TRUE/FALSE)
:I2C2:NoStretchMode «bool»	Set no stretch mode (ON/OFF, 1/0, TRUE/FALSE)
:I2C2:PullUpDown «0/1/2»	Set GPIO pull (0=No Pull, 1=Pull-Up, 2=Pull-Down)
:I2C2:MemReaD «dev,mem,sz,cnt»	Read memory (dev addr, mem addr, mem addr size, count)
:I2C2:MemWRite «dev,mem,sz,data»	Write memory (dev addr, mem addr, mem addr size, data bytes)
:I2C2:ReaD «dev,cnt»	Read from device (dev addr, count)
:I2C2:WRite «dev,data»	Write to device (dev addr, data bytes)
:I2C2:EXCHange «dev,cnt,data»	Write and/or read to/from device (dev addr, count, data bytes, ...)

6.3 Command Details

- **INFO**: Prints configuration and status information for the selected I2C peripheral.
- **INIT**/**DEINIT**: Initializes or deinitializes the I2C peripheral and its pins.
- **TimingRegister**/**FREQuency**: Configures timing and frequency for FMPI2C.
- **FastModePlus**/**AnalogFilter**/**DigitalFilter**: Configures advanced I2C features.
- **MemReaD**/**MemWRite**: Reads/writes memory from/to a device using memory addressing.
- **ReaD**/**WRite**/**EXCHange**: Reads, writes, or exchanges data with a device.
- **I2C2:ClockSpeed**/**AddressingMode**/**DualAddressMode**/**GeneralCallMode**/**NoStretchMode**/**PullUp**↩ **Down**: Configures I2C2-specific features.

6.4 Examples

6.4.1 Initialize FMPI2C

Command sent:
```
:I2C:INIT\n
```
Response:
```
(OK)>
```

6.4.2 Set I2C Frequency

Command sent:
```
:I2C:FREQuency 400k\n
```
Response:
```
(OK)>
```

6.4.3 Read Device Memory

Command sent:
```
:I2C:MemReaD 0x50,0x00,1,4\n
```
Response:
```
Data: 12 34 56 78
(OK)>
```

6.4.4 Write Device Memory

Command sent:
```
:I2C:MemWRite 0x50,0x00,1,ABCD\n
```
Response:
```
(OK)>
```

6.4.5 Error Example (Invalid Address)

Command sent:
```
:I2C:ReaD 0xZZ,4\n
```
Response:
```
ERR: invalid device address
(Some error)>
```

6.5 Notes

- Commands are case-insensitive.
- The command interface is extensible; new features can be added by updating the command table and implementing the corresponding handler logic.
- Some commands require specific hardware features or configurations.

Chapter 7

SCPI Commands in <tt>commands_pwm.c</tt>

This document describes the SCPI command interface implemented in `commands_pwm.c` for PWM (Pulse Width Modulation) and frequency measurement/control on the Melexis IO STM32 platform.

7.1 Overview

The `commands_pwm.c` file provides SCPI-like commands for initializing, deinitializing, reading, and setting PWM and frequency on various pins. It also supports DMA and IRQ-based PWM input, as well as clock configuration for advanced timing features.

7.2 Supported Commands

7.2.1 DMA/IRQ/Clock/Deinit

Command Pattern	Description
`:A0:PWM:DMA:INIT`	Init A0 for PWM DMA Input
`:A0:PWM:DMA?`	Get A0 PWM DMA Data
`:A0:PWM:IRQ:INIT`	Init A0 for PWM IRQ Input
`:A0:PWM:IRQ?`	Get A0 PWM IRQ Data
`:A0:CLK:INIT`	Clock at A0 Init
`:A0:CLK:FREQ`	Set clock frequency at A0
`:A0:DEINIT`	Deinit A0 pin

7.2.2 PWM/Frequency/Output (A0-A3, SDA, SCL)

Command Pattern	Description
`:A0:PWM:INIT`	Init A0 as PWM input
`:A0:PWM?`	Read A0 PWM
`:A0:FREQ:INIT`	Init A0 as Frequency measure
`:A0:FREQ?`	Read A0 Frequency in Hz
`:A0:OUT:INIT`	Init A0 as PWM output
`:A0:OUT:SET`	Set A0 PWM output in Hz
`:A0:OUT:SET:FREQ`	Set A0 Frequency output
`:A0:PWM:DEINIT`	DeInit A0 PWM Out/In
...	... (same for A1, A2, A3, SDA, SCL)

7.2.3 Example for Other Pins

- Replace `A0` with `A1`, `A2`, `A3`, `SDA`, or `SCL` for similar commands on those pins.

7.2.4 Connector/Other Pins (Commented/Planned)

- The file contains commented-out patterns for connector pins (e.g., `:CON:SCLK:INIT`) and other SPI/↩ UART-related pins, which can be enabled or extended as needed.

7.3 Command Details

- **DMA/IRQ/Clock/Deinit**: Initialize or read PWM using DMA or IRQ, configure clock, or deinitialize.
- **PWM:INIT/FREQ:INIT/OUT:INIT**: Initialize pin for PWM input, frequency measurement, or PWM output.
- **PWM?/FREQ?**: Read PWM or frequency value from the pin.
- **OUT:SET/OUT:SET:FREQ**: Set PWM output duty or frequency.
- **PWM:DEINIT**: Deinitialize PWM on the pin.

7.4 Handler Functions

- `CMD_Pwm_Init`: Handles initialization and deinitialization for PWM/frequency modes.
- `CMD_Pwm_Read`: Reads frequency or PWM values.
- `CMD_Pwm_Set`: Sets PWM duty or frequency.
- `CMD_PwmDmaInit`, `CMD_PwmDmaData`: DMA-based PWM input.
- `CMD_PwmIrqInit`, `CMD_PwmIrqData`: IRQ-based PWM input.
- `CMD_Clk`, `CMD_ClkFreq`: Clock configuration.
- `CMD_DeInit`: Deinitialization of all PWM/clock/DMA resources.

7.5 Example Usage

- Init A0 as PWM input: `:A0:PWM:INIT`
- Read A0 frequency: `:A0:FREQ?`
- Set A1 PWM output: `:A1:OUT:SET 50` (sets duty cycle)
- Set SCL frequency output: `:SCL:OUT:SET:FREQ 1000` (sets frequency)
- Init A0 for PWM DMA: `:A0:PWM:DMA:INIT`
- Get A0 PWM DMA data: `:A0:PWM:DMA? 10` (reads 10 samples)

7.6 Examples

7.6.1 Initialize PWM Input

Command sent:
```
:A0:PWM:INIT\n
```
Response:
```
(OK)>
```

7.6.2 Read PWM Value

Command sent:
```
:A0:PWM?\n
```
Response:
```
A0:PWM=1234
(OK)>
```

7.6.3 Set PWM Output Frequency

Command sent:
```
:A0:OUT:SET:FREQ 1000\n
```
Response:
```
(OK)>
```

7.6.4 Read Frequency

Command sent:
```
:A0:FREQ?\n
```
Response:
```
A0:FREQ=1000
(OK)>
```

7.6.5 Deinitialize PWM

Command sent:
```
:A0:PWM:DEINIT\n
```
Response:
```
(OK)>
```

7.6.6 Error Example (Invalid Pin)

Command sent:
```
:Z9:PWM:INIT\n
```
Response:
```
ERR: unknown pin
(Some error)>
```

7.7 Notes

- Commands are case-insensitive.
- The command interface is extensible; new pins or features can be added by updating the command table and implementing the corresponding handler logic.
- Some advanced or connector pin commands are present as comments and can be enabled as needed.

Chapter 8

SCPI Commands in <tt>commands_scpi.c</tt>

This document describes the SCPI command interface implemented in `commands_scpi.c` for SCPI (Standard Commands for Programmable Instruments) compliance and system-level queries on the Melexis IO STM32 platform.

8.1 Overview

The `commands_scpi.c` file provides handlers for the core SCPI and IEEE 488.2-mandated commands, as well as required SCPI system and status queries. These commands are essential for instrument identification, status reporting, error handling, and basic system control, ensuring compatibility with SCPI-based test and measurement environments.

8.2 Supported Commands

8.2.1 IEEE Mandated Commands (SCPI std V1999.0 4.1.1)

Command Pattern	Handler	Description
`*CLS`	CMD_Stub	Clear Status
`*ESE`	CMD_Stub	Event Status Enable
`*ESE?`	CMD_StubQ	Query Event Status Enable
`*ESR?`	CMD_StubQ	Query Event Status Register
`*IDN?`	CMD_IDN	System ID Query
`*OPC`	CMD_Stub	Operation Complete
`*OPC?`	CMD_StubQ	Query Operation Complete
`*RST`	CMD_RST	Reset Device
`*SRE`	CMD_Stub	Service Request Enable
`*SRE?`	CMD_StubQ	Query Service Request Enable
`*STB?`	CMD_StubQ	Query Status Byte
`*TST?`	CMD_StubQ	Self-Test Query
`*WAI`	CMD_Stub	Wait-to-Continue

8.2.2 Required SCPI Commands (SCPI std V1999.0 4.2.1)

Command Pattern	Handler	Description
`:SYSTem:ERRor?`	CMD_StubQ	Query system error
`:SYSTem:ERRor:NEXT?`	CMD_StubQ	Query next system error
`:SYSTem:ERRor:COUNt?`	CMD_StubQ	Query error count
`:SYSTem:VERSion?`	CMD_Version	Query SCPI version

Command Pattern	Handler	Description
:STATus:OPERation?	CMD_StubQ	Query operation status
:STATus:OPERation:EVENt?	CMD_StubQ	Query operation event
:STATus:OPERation:CONDition?	CMD_StubQ	Query operation condition
:STATus:OPERation:ENABle	CMD_Stub	Set operation enable
:STATus:OPERation:ENABle?	CMD_StubQ	Query operation enable
:STATus:QUEStionable?	CMD_StubQ	Query questionable status
:STATus:QUEStionable:EVENt?	CMD_StubQ	Query questionable event
:STATus:QUEStionable:CONDition?	CMD_StubQ	Query questionable condition
:STATus:QUEStionable:ENABle	CMD_Stub	Set questionable enable
:STATus:QUEStionable:ENABle?	CMD_StubQ	Query questionable enable
:STATus:PRESet	CMD_Stub	Preset status

8.2.3 Custom/Meta

Command Pattern	Handler	Description
?SCPI	NULL	System commands group/help

8.3 Handler Functions

- **CMD_Stub**: Placeholder for unimplemented or NOP commands.
- **CMD_StubQ**: Returns a default value (typically 0) for query commands.
- **CMD_Version**: Prints the SCPI version (e.g., 1999.0).
- **CMD_IDN**: Returns instrument identification string (manufacturer, model, serial, firmware, options).
- **CMD_RST**: Performs a system reset with user feedback.

8.4 Examples

8.4.1 Query Device Identification

Command sent:
```
*IDN?\n
```
Response:
```
Melexis,IO-STM32,123456,1.0
(OK)>
```

8.4.2 Reset Device

Command sent:
```
*RST\n
```
Response:
```
(OK)>
```

8.4.3 Query System Error

Command sent:
```
:SYSTem:ERRor?\n
```
Response:
```
0,"No error"
(OK)>
```

8.4.4 Query SCPI Version

Command sent:
```
:SYSTem:VERSion?\n
```
Response:
```
SCPI V1999.0
(OK)>
```

8.4.5 Error Example (Unknown Command)

Command sent:
```
*FOO\n
```
Response:
```
ERR: unknown command
(Some error)>
```

8.5 Notes

- Most status and error commands are stubs and return default values for SCPI compliance.
- The command interface is extensible; additional SCPI commands can be added as needed.
- All commands are case-insensitive.

Chapter 9

SCPI Commands in <tt>commands_spi.c</tt>

This document describes the SCPI command interface implemented in `commands_spi.c` for SPI (Serial Peripheral Interface) control on the Melexis IO STM32 platform.

9.1 Overview

The `commands_spi.c` file provides SCPI-like commands for initializing, configuring, and performing data transfers over two SPI peripherals (SPI1 and SPI2). It also supports Vdd control, buffer direction, and chip select (CS) management for flexible SPI communication.

9.2 Supported Commands

9.2.1 Vdd Control

Command Pattern	Description
`:VDD?`	Return Vdd state
`:VDD:OFF`	Disable Vdd
`:VDD:3V3`	Set SPI Vdd to 3.3V
`:VDD:5V`	Set SPI Vdd from USB Vbus (5V)

9.2.2 Buffer Direction

Command Pattern	Description
`:SPI:BUFfer`	Set buffer controls DIR_0...DIR↩ _5

9.2.3 SPI1 (Connector X2)

Command Pattern	Description
`:SPI:INFO?`	Get SPI1 settings
`:SPI:Init`	Initialize SPI1 (0-CS0, 1-CS1, 2-Both, 3-None)
`:SPI:DeInit`	Deinitialize SPI1
`:SPI:FRame`	Set frame format (0-Motorola, 1-TI)
`:SPI:DATAsize`	Set data size (0-8bit, 1-16bit)
`:SPI:FirstBit`	Set first bit (0-MSB, 1-LSB)
`:SPI:CPOL`	Set clock polarity (0-Low, 1-High)
`:SPI:CPHA`	Set clock phase (0-1 edge, 1-2 edge)
`:SPI:PRESCaler`	Set prescaler (2, 4, ..., 256)

Command Pattern	Description
:SPI:CS0	Set CS0
:SPI:CS0?	Get CS0
:SPI:CS1	Set CS1
:SPI:CS1?	Get CS1
:SPI:ReaD	Read SPI1 «count 1-256»
:SPI:WRite	Write SPI1 «data»[,«data»...256]
:SPI:WriteReaD	Write/Read SPI1 «data»[,«data»...256]
:SPI:EXCHange	Write/Read SPI1 «count 1-256»,«data»[,«data»...]
:SPI:0:ReaD	Read SPI1 slave 0 «count 1-256»
:SPI:0:WRite	Write SPI1 slave 0 «data»[,«data»...256]
:SPI:0:WriteReaD	Write/Read SPI1 slave 0 «data»[,«data»...256]
:SPI:0:EXChange	Write/Read SPI1 slave 0 «count»,«data»[,...]
:SPI:1:ReaD	Read SPI1 slave 1 «count 1-256»
:SPI:1:WRite	Write SPI1 slave 1 «data»[,«data»...256]
:SPI:1:WriteReaD	Write/Read SPI1 slave 1 «data»[,«data»...256]
:SPI:1:EXCHange	Write/Read SPI1 slave 1 «count»,«data»[,...]

9.2.4 SPI2 (Header H5)

Command Pattern	Description
:SPI2:INFO?	Get SPI2 settings
:SPI2:Init	Initialize SPI2 (0-CS0, 1-CS1, 2-Both)
:SPI2:DeInit	Deinitialize SPI2
:SPI2:FRame	Set frame format (0-Motorola, 1-TI)
:SPI2:DATAsize	Set data size (0-8bit, 1-16bit)
:SPI2:FirstBit	Set first bit (0-MSB, 1-LSB)
:SPI2:CPOL	Set clock polarity (0-Low, 1-High)
:SPI2:CPHA	Set clock phase (0-1 edge, 1-2 edge)
:SPI2:PRESCaler	Set prescaler (2, 4, ..., 256)
:SPI2:CS0	Set CS0
:SPI2:CS0?	Get CS0
:SPI2:CS1	Set CS1
:SPI2:CS1?	Get CS1
:SPI2:ReaD	Read SPI2 «count 1-256»
:SPI2:WRite	Write SPI2 «data»[,«data»...256]
:SPI2:WriteReaD	Write/Read SPI2 «data»[,«data»...256]
:SPI2:EXCHange	Write/Read SPI2 «count 1-256»,«data»[,«data»...]
:SPI2:0:ReaD	Read SPI2 slave 0 «count 1-256»
:SPI2:0:WRite	Write SPI2 slave 0 «data»[,«data»...256]
:SPI2:0:WriteReaD	Write/Read SPI2 slave 0 «count»,«data»[,...]
:SPI2:0:EXCHange	Write/Read SPI2 slave 0 «count»,«data»[,...]
:SPI2:1:ReaD	Read SPI2 slave 1 «count 1-256»
:SPI2:1:WRite	Write SPI2 slave 1 «data»[,«data»...256]
:SPI2:1:WriteReaD	Write/Read SPI2 slave 1 «data»[,«data»...256]
:SPI2:1:EXCHange	Write/Read SPI2 slave 1 «count»,«data»[,...]

9.3 Command Details

- **Vdd Control**: Manage SPI Vdd power (OFF, 3.3V, 5V, query state).

- **Buffer**: Set direction of buffer pins DIR_1 to DIR_5.
- **Init**/**DeInit**: Initialize or deinitialize SPI peripherals and chip select pins.
- **Frame**/**DataSize**/**FirstBit**/**CPOL**/**CPHA**/**Prescaler**: Configure SPI frame format, data size, bit order, clock polarity/phase, and baudrate.
- **CS**/**CS?**: Set or query chip select (CS0/CS1) state or mapping.
- **ReaD**/**WRite**/**WriteReaD**/**EXCHange**: Perform SPI data transfers (read, write, write/read, exchange) with optional slave selection.

9.4 Examples

9.4.1 Initialize SPI1

Command sent:
```
:SPI:Init 0\n
```
Response:
```
(OK)>
```

9.4.2 Set SPI1 Prescaler

Command sent:
```
:SPI:PRESCaler 8\n
```
Response:
```
(OK)>
```

9.4.3 Write Data to SPI1

Command sent:
```
:SPI:WRite 12,34,56\n
```
Response:
```
(OK)>
```

9.4.4 Read Data from SPI1

Command sent:
```
:SPI:ReaD 3\n
```
Response:
```
Data: 12 34 56
(OK)>
```

9.4.5 Exchange Data with SPI1 Slave 0

Command sent:
```
:SPI:0:EXChange 2,AB,CD\n
```
Response:
```
Data: EF 01
(OK)>
```

9.4.6 Error Example (Invalid Prescaler)

Command sent:
```
:SPI:PRESCaler 3\n
```
Response:
```
ERR: invalid prescaler value
(Some error)>
```

9.5 Notes

- Commands are case-insensitive.

- The command interface is extensible; new features or pins can be added by updating the command table and implementing the corresponding handler logic.
- Some commands require specific hardware features or configurations.

Chapter 10

SCPI Commands in <tt>commands_uart.c</tt>

This document describes the SCPI command interface implemented in `commands_uart.c` for UART (Universal Asynchronous Receiver/Transmitter) control on the Melexis IO STM32 platform.

10.1 Overview

The `commands_uart.c` file provides SCPI-like commands for initializing, deinitializing, configuring, and performing data transfers over UART3 and UART4. The commands support baud rate, word length, stop bits, parity, oversampling, and read/write/exchange operations.

10.2 Supported Commands

10.2.1 Information

Command Pattern	Description
`:UART3:INFO`	Get UART3 settings
`:UART4:INFO`	Get UART4 settings

10.2.2 Initialization/Deinitialization

Command Pattern	Description
`:UART3:INIT`	Initialize UART3 TX0/RX0
`:UART4:INIT`	Initialize UART4 TXA0/RXA1
`:UART3:DEINIT`	Deinitialize UART3
`:UART4:DEINIT`	Deinitialize UART4

10.2.3 UART3 Configuration

Command Pattern	Description
`:UART3:BaudRate «val»`	Set UART3 baud rate (e.g. 9600, 115200)
`:UART3:BaudRate?`	Get UART3 baud rate
`:UART3:WordLength «0/1»`	Set UART3 word length (0=8bit, 1=9bit)
`:UART3:WordLength?`	Get UART3 word length
`:UART3:StopBit «1/2»`	Set UART3 stop bit
`:UART3:StopBit?`	Get UART3 stop bit
`:UART3:Parity «0/1/2»`	Set UART3 parity (0=None, 1=Odd, 2=Even)
`:UART3:Parity?`	Get UART3 parity
`:UART3:OverSampling «on/off»`	Set UART3 oversampling (on=16, off=8)

Command Pattern	Description
:UART3:OverSampling?	Get UART3 oversampling

10.2.4 UART4 Configuration

Command Pattern	Description
:UART4:BaudRate «val»	Set UART4 baud rate
:UART4:BaudRate?	Get UART4 baud rate
:UART4:WordLength «0/1»	Set UART4 word length (0=8bit, 1=9bit)
:UART4:WordLength?	Get UART4 word length
:UART4:StopBit «1/2»	Set UART4 stop bit
:UART4:StopBit?	Get UART4 stop bit
:UART4:Parity «0/1/2»	Set UART4 parity (0=None, 1=Odd, 2=Even)
:UART4:Parity?	Get UART4 parity
:UART4:OverSampling «on/off»	Set UART4 oversampling (on=16, off=8)
:UART4:OverSampling?	Get UART4 oversampling

10.2.5 UART Data Transfer

Command Pattern	Description
:UART3:ReaD «count»	Read data from UART3 (1-255 bytes)
:UART3:WRite «data,...»	Write data to UART3 (up to 256 bytes)
:UART3:EXCHange «data,...»	Write then read from UART3
:UART4:ReaD «count»	Read data from UART4 (1-255 bytes)
:UART4:WRite «data,...»	Write data to UART4 (up to 256 bytes)
:UART4:EXCHange «data,...»	Write then read from UART4

10.3 Command Details

- **INFO**: Prints current UART settings (baud rate, word length, stop bits, parity, oversampling).
- **INIT/DEINIT**: Initializes or deinitializes UART3/UART4 and their associated pins.
- **BaudRate/WordLength/StopBit/Parity/OverSampling**: Configures or queries UART parameters.
- **ReaD/WRite/EXCHange**: Reads, writes, or exchanges data with the UART peripheral.

10.4 Examples

10.4.1 Initialize UART3

Command sent:
```
:UART3:INIT\n
```
Response:
```
(OK)>
```

10.4.2 Set UART3 Baud Rate

Command sent:
```
:UART3:BaudRate 115200\n
```
Response:
```
(OK)>
```

10.4.3 Query UART3 Baud Rate

Command sent:
```
:UART3:BaudRate?\n
```
Response:
```
BaudRate=115200
(OK)>
```

10.4.4 Write Data to UART3

Command sent:
```
:UART3:WRite Hello\n
```
Response:
```
(OK)>
```

10.4.5 Read Data from UART3

Command sent:
```
:UART3:ReaD 5\n
```
Response:
```
Data: 48 65 6C 6C 6F
(OK)>
```

10.4.6 Error Example (Invalid Baud Rate)

Command sent:
```
:UART3:BaudRate 123\n
```
Response:
```
ERR: invalid baud rate
(Some error)>
```

10.5 Notes

- Commands are case-insensitive.
- The command interface is extensible; new features can be added by updating the command table and implementing the corresponding handler logic.
- Some advanced features (e.g., timeout, last bit) are present as comments and can be enabled as needed.