INNOVATION TODAY
FOR TOMORROW®

Command Line Interface

Programmer’s Reference Guide

October 2015

nutaq
.com

Revision history

Revision	Date	Comments
0.1	November 2012	First draft.
1.0	December 2012	Added Radio420 CLI functions
1.1	December 2012	Added MI125 and LVDS CLI functions
1.2	February 2013	Ready for revision
1.3	February 2013	Linguistic revision
1.4	February 2013	Added ram_init function
1.5	March 2013	Added PCIe support in RTDEx and Record/Playback commands
1.6	June 2013	Added ADC500 CLI functions
1.7	September 2013	Added PPS Sync and Aurora functions. Corrected Radio 420 gains functions.
1.8	March 2014	Added CCE, U-Boot, Kernel, DTB, JFFS2 functions in Flash commands. Added Bus read and write commands. Added fmcradio_spi functions
1.9	June 2014	Layout modifications and added Mestor LVDS module - Release 6.5
1.10	October 2014	Added MO1000 functions
1.11	November 2014	Added System Monitor functions Modified update_cce command behaviour Up to date for release 6.6
1.12	December 2014	Added i2c bus scan command
1.13	October 2015	New glossary Updated Aurora commands Updated Record/Playback commands Removed RTDEx commands Up to date for release 7

Version 1.13

WARNING

Do not use Nutaq products in conjunction with life-monitoring or life-critical equipment. Failure to observe this warning relieves Nutaq of any and all responsibility.

FCC WARNING

This equipment is intended for use in a controlled environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of personal computers and peripherals pursuant to subpart J of part 15 of the FCC rules. These rules are designed to provide reasonable protection against radio frequency interference. Operating this equipment in other environments may cause interference with radio communications, in which case the user must, at his/her expense, take whatever measures are required to correct this interference.

This page was left intentionally blank.

Table of Contents

1 Introduction 1
1.1 Conventions 1
1.1 Glossary 2
1.2 Technical Support 3

2 CLI Description 5

3 CLI Commands 7
3.1 Main Module 7
3.1.1 Basic Commands 7
3.1.2 Connection Commands 7
3.1.3 Flash Commands 7
3.1.4 Bus Commands 9
3.2 OS Configuration Module 9
3.4 Record/Playback Module 10
3.5 Radio420X Module 11
3.6 ADAC250 Module 13
3.7 MI250 Module 15
3.8 MI125 Module 16
3.9 ADC5000 Module 17
3.10 LVDS-xIn-xOut Module 19
3.11 Aurora Module 20
3.12 PPSSync Module 21
3.13 Mestor LVDS Module 22
3.14 MO1000 Module 23
3.15 System Monitor module 25
3.16 Timestamp module 25

4 Building a CLI Script 26
4.1 Writing a Script 26
4.2 Running a Script on Windows 27
4.3 Running a Script on Linux 27

List of Tables

Table 1 Glossary 3
Table 2 Basic CLI commands 7
Table 3 Connection commands 7
Table 4 Flash commands 8
Table 5 Bus commands 9
Table 6 OS configuration module 9
Table 7 Record/Playback commands 10
Table 8 Radio420X commands 12
Table 9 ADAC250 commands 14
Table 10 MI250 commands 15
Table 11 MI125 commands 16
Table 12 ADC5000 commands 18
Table 13 LVDS-xIn-xOut commands 19
Table 14 Aurora commands 20
Table 15 PPSSync commands 21
Table 16 Mestor LVDS commands 22
Table 17 MO1000 commands 24
Table 18 Sysmon commands 25
Table 19 Timestamp commands 25

This page was left intentionally blank.

1 Introduction

This guide contains a complete list of all the commands supported by Nutaq's CLI (Command Line Interface). In addition to listing all the available functions for each module, it also contains description of all the arguments of the functions as well as a description and an example of utilisation.

Please refer to this guide when creating scripts using the Command Line Interface.

Organization

This guide is organized as follows:

- CLI purpose
- CLI commands
- Building a CLI script

1.1 Conventions

In a procedure containing several steps, the operations that the user has to execute are numbered (1, 2, 3…). The diamond (♦) is used to indicate a procedure containing only one step, or secondary steps. Lowercase letters (a, b, c…) can also be used to indicate secondary steps in a complex procedure.

The abbreviation NC is used to indicate no connection.

Capitals are used to identify any term marked as is on an instrument, such as the names of connectors, buttons, indicator lights, etc. Capitals are also used to identify key names of the computer keyboard.

All terms used in software, such as the names of menus, commands, dialog boxes, text boxes, and options, are presented in bold font style.

The abbreviation N/A is used to indicate something that is not applicable or not available at the time of press.

Note:
The screen captures in this document are taken from the software version available at the time of press. For this reason, they may differ slightly from what appears on your screen, depending on the software version that you are using. Furthermore, the screen captures may differ from what appears on your screen if you use different appearance settings.

1.1 Glossary

This section presents a list of terms used throughout this document and their definition.

Term	Definition
Advanced Mezzanine Card (AMC)	AdvancedMC is targeted to requirements for the next generation of "carrier grade" communications equipment. This series of specifications are designed to work on any carrier card (primarily AdvancedTCA) but also to plug into a backplane directly as defined by MicroTCA specification.
Advanced Telecommunications Computing Architecture (or AdvancedTCA, ATCA)	AdvancedTCA is targeted primarily to requirements for "carrier grade" communications equipment, but has recently expanded its reach into more ruggedized applications geared toward the military/aerospace industries as well. This series of specifications incorporates the latest trends in high speed interconnect technologies, next-generation processors, and improved Reliability, Availability and Serviceability (RAS).
Application Programming Interface (API)	An application programming interface is the interface that a computer system, library, or application provides to allow requests for services to be made of it by other computer programs or to allow data to be exchanged between them.
Board Software Development Kit (BSDK)	The board software development kit gives users the possibility to quickly become fully functional developing C/C++ for the host computer and HDL code for the FPGA through an understanding of all Nutaq boards major interfaces.
Boards and Systems (BAS)	Refers to the division part of Nutaq which is responsible for the development and maintenance of the hardware and software products related to the different Perseus carriers and their different FMC daughter cards.
Carrier	Electronic board on which other boards are connected. In the FMC context, the FMC carrier is the board on which FMC connectors allow a connection between an FMC card and an FPGA. Nutaq has two FMC carriers, the Perseus601x (1 FMC site) and the Perseus611x (2 FMC sites).
Central Communication Engine (CCE)	The Central Communication engine (CCE) is an application that executes on a virtual processor called a MicroBlaze in the FPGA of the Perseus products. It handles all the behavior of the Perseus such as module initialization, clock management, as well as other tasks.
Chassis	Refers to the rigid framework onto which the CPU board, Nutaq development platforms, and other equipment are mounted. It also supports the shell-like case—the housing that protects all the vital internal equipment from dust, moisture, and tampering.
Command Line Interface (CLI)	The Command Line Interface (or CLI) is a basic client interface for Nutaq's FMC carriers. It runs on a host device. It consists of a shell where commands can be typed, interacting with the different computing elements connected to the system.
FPGA Mezzanine Card (FMC)	FPGA Mezzanine Card is an ANSI/VITA standard that defines I/O mezzanine modules with connection to an FPGA or other device with re-configurable I/O capability. It specifies a low profile connector and compact board size for compatibility with several industry standard slot card, blade, low profile motherboard, and mezzanine form factors.
HDL	Stands for hardware description language.
Host	A host is defined as the device that configures and controls a Nutaq board. The host may be a standard computer or an embedded CPU board in the same chassis system where the Nutaq board is installed. You can develop applications on the host for Nutaq boards through the use of an application programming interface (API) that comprises protocols and functions necessary to build software applications. These API are supplied with the Nutaq board.
MicroTCA (or µTCA)	The MicroTCA (µTCA) specification is a PICMG Standard which has been devised to provide the requirements for a platform for telecommunications equipment. It has been created for AMC cards.
Model-Based Design	Refers to all the Nutaq board-specific tools and software used for development with the boards in MATLAB and Simulink and the Nutaq model-based design kits.
Model-Based Development Kit (MBDK)	The model-based development kit gives users the possibility to create FPGA configuration files, or bitstreams, without the need to be fluent in VHDL. By combining Simulink from Matlab, System Generator from Xilinx and Nutaq's tools, someone can quickly create fully-functional FPGA bitstreams for the Perseus platforms.
NTP	Network Time Protocol. NTP is a protocol to synchronize the computer time over a network.
Peer	A host peer is an associated host running RTDEx on either Linux or Windows. An FPGA peer is an associated FPGA device.

Term	Definition
PicoDigitizer / PicoSDR Systems	Refers to Nutaq products composed of Perseus AMCs and digitizer or SDR FMCs in a table top format.
PPS	Pulse per second. Event to indicate the start of a new second.
Reception (Rx)	Any data received by the referent is a reception.
Reference Design	Blueprint of an FPGA system implemented on Nutaq boards. It is intended for others to copy and contains the essential elements of a working system (in other words, it is capable of data processing), but third parties may enhance or modify the design as necessary.
Transmission (Tx)	Any data transmitted by the referent is a transmission. Abbreviated TX.
µDigitizer / µSDR Systems	Any Nutaq system composed of a combination of µTCA or ATCA chassis, Perseus AMCs and digitizer or SDR FMCs.
VHDL	Stands for VHSIC hardware description language.

Table 1 Glossary

1.2 Technical Support

Nutaq is firmly committed to providing the highest level of customer service and product support. If you experience any difficulties using our products or if it fails to operate as described, first refer to the documentation accompanying the product. If you find yourself still in need of assistance, visit the technical support page in the Support section of our Web site at www.nutaq.com.

2 CLI Description

The Command Line Interface (or CLI) is a basic client interface for Nutaq’s FMC carriers. It runs on a host device. It consists of a shell where commands can be typed, interacting with the different computing elements connected to the system. The CLI offers many useful features:

- Programming an FPGA bitstream in the onboard flash memory to be used on subsequent boots.
- Reading and writing at specified addresses on the AXI bus.
- Loading data at specified addresses in the Perseus’ DDR3 SDRAM.
- Using the RTDEx **and** Record/Playback FPGA cores.
- Configuring and controlling Nutaq’s FMC daughter card.

Requirements

- Python 2.7 (supplied)
- Ethernet connectivity to the FMC carrier

3 CLI Commands

The functionalities of an FMC carrier system can be highly customized and the standard software evolves over time. The Command Line Interface is a light interface that helps debug and configure the FPGA cores of various peripherals of the carrier.

The following commands can be used once you are connected to the CCE and when the necessary modules are present.

3.1 Main Module

3.1.1 Basic Commands

Command	Argument	Description	
help	*None*	Shows available commands and describes how to use them.	*help*
version	*None*	Displays the version of the CCE currently connected.	*version*
shell	*None*	Runs shell commands.	*shell echo Hello Worlc*
sleep	Number of seconds to sleep	Sleeps for a number of seconds.	*sleep 10*
exit	*None*	Exits the Command Line Interface.	*exit*

Table 2 Basic CLI commands

3.1.2 Connection Commands

Command	Argument	Description	
connect	Target IP address	Connects to a remote CCE.	*connect 192.168.0.10*
disconnect	*None*	Disconnects from a CCE process.	*disconnect*
reboot	*None*	Reboots the FPGA.	*reboot*
getmac	*None*	Prints the MAC address.	*getmac*
getip	*None*	Prints the IP address.	*getip*
fmc_info	*None*	Prints the information related to the FMC card present on the carrier	*fmc_info*

Table 3 Connection commands

3.1.3 Flash Commands

Command	Argument	Description	
fpgaflash	• Index • Bitstream file to flash • Comment (optional)	Program a bitstream onto the on-board flash memory. The index values supported are 1 and 2. The comment is optional but will be displayed when fpgaflash_get_info will be executed. If no comment is provided, the bitstream file name will be used instead.	*fpgaflash 1 “C* *fpgaflash 1 “C*
fpgaflash_get_info	*None*	Print the bitstream index that will be loaded into the FPGA at system boot up. Print the information of bitstream 1 and bitstream 2 if they are present in the flash.	*fpgaflash_get_*
fpgaflash_set_index	• Index	Set the bitstream index that will be loaded into the FPGA at system boot up. The index values supported are 1 and 2.	*fpgaflash_set_*

update_cce	• CCE file to flash	Program the CCE onto the on-board flash memory. Make sure the filename is “cce”. This command does not require a prior *connect* command.	*update_cce “C:\Nutaq\BA*
update_uboot	• U-Boot file to flash	Program U-Boot onto the on-board flash memory. Make sure the filename has the extension “.bin”.	*update_uboot “C:\Nutaq boot-s.bin”*
update_kernel	• Kernel image to flash	Program a Kernel image onto the on-board flash memory. Make sure the filename has the extension “.ub”.	*update_kernel “C:\Nutaq\BAS\sdk\embe*
update_dtb	• DTB file to flash	Program the DTB onto the on-board flash memory. Make sure the filename has the extension “.dtb”.	*update_dtb “C:\Nutaq\BAS\sdk\embe*
update_jffs2	• JFFS2 file to flash	Program the JFFS2 onto the on-board flash memory. Make sure the filename has the extension “.jffs2”.	*update_jffs2 “C:\Nutaq\BAS\sdk\embe*

Table 4 Flash commands

3.1.4 Bus Commands

Command	Argument	Description	
read	• Internal bus address (available addresses are in the 0×60000000–0×7FFFFFFF and 0×85000000–0×8FFFFFFF ranges)	Reads 32 bits of data from an absolute address in the FPGA AXI bus.	*read*
custom_register_read	• Custom register ID (available IDs are 0-31)	Reads 32 bits of data from a custom register ID in the FPGA AXI bus.	*custo*
write	• Internal bus address (available addresses are in the 0×60000000–0×7FFFFFFF and 0×85000000–0×8FFFFFFF ranges) • Data to write	Write 32 bits of data in an absolute address in the FPGA AXI bus.	*write*
custom_register_write	• Custom register ID (available IDs are 0-31) • Data to write	Write 32 bits of data in a custom register ID in the FPGA AXI bus.	*custo*
I2c_bus_scan	• Bus to scan	Scan a specified i2c bus to detect all available devices.	*i2c_b*

Table 5 Bus commands

3.2 OS Configuration Module

Command	Argument	Description	Example
osconfig_create_static_entry	• Entry name • IP address • Gateway address • Net mask	Creates a static IP configuration entry.	*osconfig_create_static_entry staticentry 192.168*
osconfig_create_dhcp_entry	• Entry name	Creates a DHCP configuration entry.	*osconfig_create_dhcp_entry dhcpentry*
osconfig_useentry	• Entry name	Commits the entry for use on future boots.	*osconfig_create_dhcp_entry staticentry*
osconfig_listentries	*None*	Lists all entries in the memory.	*osconfig_listentries*

Table 6 OS configuration module

3.4 Record/Playback Module

Command	Argument	Description	
PlaybackData	• Trigger Source ○ “SOFT”: Software trigger ○ “EXT”: External trigger • Playback mode ○ “SINGLE”: Single Playback ○ “CONTINUOUS”: Continuous playback • Playback size – in bytes • Start address	Playback data from RAM with specified parameters	*Playbac*
PlaybackStop	*None*	Stop the current playback from RAM	*Playbac*
RecordData	• Trigger Source ○ “SOFT”: Software trigger ○ “EXT”: External trigger • Record size – in bytes • Start address • Trigger delay: in chunks of 64 bytes	Record data to RAM with specified parameters	*recplay*
LoadDataToPlaybackFromFile	• RTDEx channel used to load memory • RTDEx frame size used to load memory • Start address • File name	Sends data from a host device to the Perseus through the RTDEx and writes it in the DDR3 SDRAM. The frame size is in bytes. The transfer size will be the same as the file size. This command performs either on the Gigabit Ethernet or the PCI Express media.	*LoadDa*
RetrieveRecordedDataToFile	• RTDEx channel used to read memory • RTDEx frame size used to read memory • Record size • Start address • Record timeout (ms) • File name	Reads the data from the Perseus DDR3 SDRAM and sends it to a host device through the RTDEx. The transfer size and frame size are in bytes. This command performs either on the Gigabit Ethernet or the PCI Express media.	*Retrieve* *readda*

Table 7 Record/Playback commands

3.5 Radio420X Module

ommand	Argument	Description	Example
reset	*None*	Resets the Radio420X.	*fmcradio_reset*
select	• Selected radio (1 for bottom, 2 for top)	Selects the Radio420X controlled radio.	*fmcradio_select 1*
powerup	*None*	Powers up the FMC site.	*Fmcradio_powerup*
setrevision	• FMC Radio hardware revision (SDR_A, SDR_B, SDR_C, SDR_D)	Sets the revision of the FMC Radio.	*Fmcradio_setrevision SDR_D*
path_enable	• Selected path (TX or RX)	Enables the RF path on the card.	*fmcradio_path_enable tx*
path_disable	• Selected path. (TX or RX))	Disables the RF path on the card. Disabling the RF path can reduce ambient noise.	*fmcradio_path_disable rx*
pll	• Reference frequency (onboard crystal frequency in Hz) • ADC frequency (frequency at which the D/A and the A/D converters operate in Hz) • Lime frequency (clock sent to the Lime Microsystems RF chip in Hz)	Configures the onboard PLL.	*fmcradio_pll 30720000 4096000*
band	• Selected radio band [high (1500 to 3000 MHz) or low (300 to 1800 MHz)]	Switches RF bands.	*fmcradio_band low*
pll_lock	*None*	Displays the current lock status of CDCE620005 PLL.	*fmcradio_plllock*
clkmux	• Clock Destination (PLLIN2, 1PPS, FMCCLK0, or FMCCLK1) • Clock Source (PLLCLKOUT, EXTCLK, FMCCLK2, or FMCCLK3)	Configures the clock multiplexer so that the source clock is connected to the destination clock.	*fmcradio_clkmux 'destination' 's*
lime_pll	• Direction (TX or RX) • Reference frequency (clock fed by the CDCE62005 PLL in Hz) • Carrier frequency (Radio420X mixing frequency in Hz)	Configures the PLL of the Lime Microsystems RF chip.	*fmcradio_lime_pll tx 40960000*
rx	• LMS6002D RX Low-noise amplifier gain (lh or lnamaxgain to configure the low-noise amplifier (LNA) gain to maximum. lm or lnamidgain to configure the LNA gain to medium (default). lb or lnabypass to bypass the LNA) • LMS6002D RX VGA1 gain (vh or vgamaxgain to configure a VGA gain to 30 dB. vm or vgamidgain to configure a VGA gain to 19 dB (default). vl or vgalowgain to configure a VGA gain to 5 dB) • LMS6002D RX VGA2 gain (in dB, must be between 0 dB and 60 dB, above 30 dB is not recommended) • RDA1005LDS RX SPI gain (in dB, must be between –13 dB and 18 dB)	Configures the RX path gains.	*fmcradio_rx --vm --lb 0 -5*
filter	• RX filter value (possible filter names are 1880 MHz, 1950 MHz, 1960 MHz, 2140 MHz, 2495 MHz, 3600 MHz, 837 MHz, 882 MHz, 898 MHz, 943 MHz, NONE, and TESTPOINT)	Configures the RX path RF filter.	*fmcradio_*filter 943MHZ
lpf	• RF path (RX, TX, or RXTX) • Filter cut-off frequency (possible values for the low-pass filter name are 0.75 MHz, 0.875 MHz, 10 MHz, '14 MHz, 1.25 MHz, 1.375 MHz, 1.5 MHz, 1.92 MHz, 2.5 MHz, 2.75 MHz, 3.5 MHz, 3 MHz, 4.375 MHz, 5 MHz, 6 MHz, 7 MHz, and BYPASS)	Configures the low-pass filter inside the Lime Microsystems chip.	*fmcradio_lpf tx 2.5MHZ*
lime_reset	*None*	Resets the Lime Microsystems LMS6002 chip.	*fmcradio_lime_reset*
lpfcalibrate	• Reference frequency (reference fed to the LMS6002 from the CDCE62005 PLL in Hz)	Calibrates the LMS6002 low-pass filter.	*fmcradio_lpfcalibrate 40960000*
rxvga_calibrate	*None*	Calibrates the DC offset of the RX amplifier.	*fmcradio_rxvga_calibrate*
tx	• LMS6002D TX VGA1 gain (in dB, must be between –35 and –4 dB) • LMS6002D TX VGA2 gain (in dB, must be between 0 and 25 dB) • RDA1005LDS TX SPI gain (in dB, must be between –13 and 18 dB)	Configures the TX path gains.	*fmcradio_tx -17 5 -4*
limespi_write	• Address • Data	Writes data to the specified register address of the Lime Microsystems LMS6002 chip.	*fmcradio_limespi_write 8 273*
limespi_read	• Address	Reads data from the specified register address of the Lime Microsystems LMS6002 chip and prints the result.	*fmcradio_limespi_read 8*
rxdc_offset_calibrate	*None*	Performs RX DC offset calibration.	*fmcradio_rxdc_offset_calibrate*

l	Argument	Description	Example
calibrate	• Carrier frequency (in Hz) • Acquisition frequency (in Hz)	Performs LO leakage calibration.	*fmcradio_loleakage_calibrate 943000000 40960000*
te	• Carrier frequency (in Hz) • Acquisition frequency (in Hz)	Performs IQ gain and IQ phase calibrations.	*fmcradio_ssb_calibrate 943000000 409600*
e	• Carrier frequency (in Hz) • Acquisition frequency (in Hz)	Performs RX DC offset, LO leakage, and IQ gain and phase calibrations.	*fmcradio_rf_calibrate 943000000 4096000*
l	• FPGA control, possible values are : ○ 0 : Reference Clock ○ 1 : Radio Frequency ○ 2 : RX Gain ○ 3 : TX Gain ○ 4 : PLL/CPLD (IO expanders) • SPI controller, possible values are : ○ 0 : Host ○ 1 : FPGA	Configures who controls the SPI bus with the different controls.	*fmcradio_spi_control 0 1*
	• Device, possible values are : ○ 0 : LIMEMICRO RF device ○ 1 : TX Gain device ○ 2 : RX Gain device ○ 3 : PLL device ○ 4 TCVXO device • Address	Reads data from the specified register address of the specified device and prints the result.	*fmcradio_spi_read 0 8*
	• Device, possible values are : ○ 0 : LIMEMICRO RF device ○ 1 : TX Gain device ○ 2 : RX Gain device ○ 3 : PLL device ○ 4 TCVXO device • Address • Data	Writes data to the specified register address of the specified device.	*fmcradio_spi_control 0 8 273*

Table 8 Radio420X commands

3.6 ADAC250 Module

Command	Argument	Description	
adac250_presence	*None*	Verifies if the ADAC250 FMC card and FPGA core are present.	*adac2*
adac250_powerup	*None*	Powers up the FMC site for the ADAC250 operation.	*adac2*
adac250_pll_init	• reference frequency (in Hz) • ADC sample frequency (in Hz) • DAC data frequency (in Hz) • Interpolation factor (1, 2, or 4) • clock source (0 for internal, 1 for external ref and 2 for external clock)	Configures the ADAC250 PLL to provide clock to the ADC and DAC at specified frequencies (Hz). The ADAC250 internal reference is at 10 MHz. The DAC data frequency is the frequency of the data to send to the DAC chip and should not take into account the interpolation factor.	*adac2* *adac2*
adac250_pll_setrefClkTuning	• Tuning value (on 16 bits)	Tunes reference frequency generator. Value is 16 bits wide.	*adac2*
adac250_pll_get_status	*None*	Verifies and prints the PLL lock status.	*adac2*
adac250_dac_init	• Sleep mode 0: None, 1: Sleep A, 2:Sleep B, 3: Sleep A & B • Interpolation factor 1: x1, 2: x2, 4: x4 • Mix-mode channel A 0: LP, 1: HP, 2: PosFDac, 3: NegFDac • Mix-mode channel B 0: LP, 1: HP, 2: PosFDac, 3: NegFDac	Initializes the DAC. See the DAC chip datasheet for the Mix-mode details.	*adac2*
adac250_dac_gain	• Channel 0: A, 1: B • Gain (0 to 15)	Changes the DAC gain of the specified channel. See DAC datasheet for conversion formula between value entered and the gain in dB.	*adac2*
adac250_dac_syncsource	• Source 0: from FPGA logic, 1: from DAC register	Changes the synchronization signal source to the DAC. When set to 1, the adac250_dac_sync function must be called to set the DAC sync to 1 to enable the DAC outputs.	*adac2*
adac250_dac_sync	• Sync 0: Disable DAC Sync register, 1: Enable DAC Sync register	Synchronizes the DAC output with data stream. Refer to the datasheet for details. If adac250_dac_syncsource function set the source to 1, this function must be called to set the DAC sync to 1 to enable the DAC outputs.	*adac2*
adac250_dac_calibrate	*None*	Calibrates the DAC data bus timings.	*adac2*
adac250_adc_init	*None*	Initializes the ADC.	*adac2*
adac250_adc_gain	• Channel 0: A, 1: B • Gain (0 to 12 in 0.5-dB steps)	Sets the ADC gain of the specified channel.	*adac2*
adac250_adc_finegain	• Channel 0: A, 1: B • Channel fine gain (0 to 127)	Sets the ADC fine gain of the specified channel.	*adac2*
adac250_adc_pedestal	• Channel 0: A, 1: B • Channel DC offset (-32 to 31)	Sets the DC offset of the specified channel.	*adac2*
adac250_adc_calibrate	*None*	Calibrates the ADC data bus timings.	*adac2*
adac250_mux_configClockOutput	• Clock output 0: ADAC250_CLOCKOUT_REFOUT, 1: ADAC250_CLOCKOUT_PLLCLK1, 2: ADAC250_CLOCKOUT_PLLREF, 3: ADAC250_CLOCKOUT_FMCCLK1 • Clock input 0: ADAC250_CLOCKIN_10MHZ, 1: ADAC250_CLOCKIN_REFIN, 2: ADAC250_CLOCKIN_PLLCLKOUT2, 3: ADAC250_CLOCKIN_FMCCLK3	Routes the specified input to the specified multiplexer output.	*adac2*

Command	Argument	Description	
adac250_adc_settriggerdelay	• Delay 1 to 32	Delay the input trigger by the specified number of clock cycles. This delay can be used to synchronize trigger signal with the data signals and compensate for the ADC propagation delay.	*adac250_adc_se*

Table 9 ADAC250 commands

3.7 MI250 Module

Command	Argument	Description
mi250_select	• Fmc connector position	Selects the MI250 on which FMC connector will be controlled by the nex commands. 'num' starts at 1.
mi250_init	*None*	Resets MI250 and initializes the ADC to default values.
mi250_powerup	*None*	Powers up the FMC site for MI250 operation.
mi250_presence	*None*	Verifies the presence of the MI250 core and FMC daughter card.
mi250_pllconfig	• Reference clock value (in Hz) • ADC clock value (in Hz) • Clock source (0 = inboard reference, 1 = external reference, 2 = external clock)	Configure the MI250 PLL to provide clock to the ADC at the specified frequency.
mi250_pllstatus	*None*	Prints the current status of the MI250 PLL.
mi250_set_gain	• Channel (0 to 7 for channels A-H) • Gain (0 to 12 -- 0.5 dB gain per step)	Sets the ADC channel gain.
mi250_set_finegain	• Channel (0 to7 for channels A-H) • Fine gain (0 to 127 – 0 to 0,134dB)	Sets the ADC channel fine gain.
mi250_set_pedestal	• Channel (0 to 7 for channels A-H) • Offset value (-32 to 31 in ADC increments)	Sets the ADC channel pedestal.
mi250_PLLGetStatus	*None*	Prints the current status of the MI250 PLL.
mi250_adcreset	*None*	Resets all ADCs.
mi250_adcarm	• Armed status (0 or 1)	Sets the armed status for the ADC channels in the FPGA.
mi250_set_trigger_source	• Source (0 for external trigger, 1 for software trigger)	Sets the ADCs trigger source.
mi250_software_trigger	*None*	Triggers the MI250 acquisition if the trigger source is set to software trigger.

Table 10 MI250 commands

3.8 MI125 Module

Command	Argument	Description	
mi125_reset	• MI125 target board (1)	Resets and initializes the MI125 (ADC, clock routing, all options...) with default values.	*Mi125_*
mi125_powerup	• MI125 target board (1)	Powers up the MI125 board.	*mi125_*
mi125_digital_adccalibration	• MI125 target board (1)	Forces the ADC output digital calibration.	*mi125_*
mi125_set_config	• MI125 target board (1) • Channel group • ADC output LVDS • ADC output randomize mode • ADC output data format	Configures board options.	*mi125_* *termon* *twocon*
mi125_set_clksrc	• MI125 target board (1) • Clock source (125 MHz,EXT, BOTTOMFMC,FMCCARRIER)	Configures ADC clock source	*mi125_*
mi125_checkcore	• MI125 target board (1)	Checks for the MI125 FPGA core presence.	*mi125_*
mi125_set_testmode	• MI125 target board (1) • Testmode (TESTMODEOFF, TESTMODE1, TESTMODE2) • Pattern	Configures the ADC test mode or not. In test mode, the digital interface will continuously send the programmed pattern.	*mi125_*
mi125_get_temperature	• MI125 target board (1) • Temperature data output format to use (TEMP1C, TEMP0DOT1C)	Gets the PCB temperature.	*mi125_*
mi125_get_channelcalibstatus	• MI125 target board (1)	Gets the ADC last digital calibration status.	*mi125_*
mi125_get_versions	• MI125 target board (1)	Gets the FPGA core and firmware version of the board.	*mi125_*
mi125_checkadc	• MI125 target board (1) • ADC id to verify (ADCX, x = 0 to 3)	Runs the ADC check.	*mi125_*
mi125_set_trigout	• MI125 target board (1) • Trigger output configuration (TRIGOUTON, TRIGOUTOFF)	Configures the MI125 trigger output.	*mi125_*
mi125_get_clkmaster	• MI125 target board (1)	Gets if the module is a clock master.	*mi125_*
mi125_clockreset	• MI125 target board (1)	Forces an MCM clock reset.	*mi125_*

Table 11 MI125 commands

(1) The MI125 target board parameter specifies which instance of the MI125 is accessed from a “double-stack” configuration stand-point. A value of 1 for this parameter accesses the MI125-16 closest to the carrier and a value of 2 accesses the MI125-16 that is the farthest. When only one MI125-16 is used, this parameter **must** be set to 1.

3.9 ADC5000 Module

Command	Argument	Description
adc5000_presence	*None*	Verifies the presence of the ADC5000 core and FMC daughter carc
adc5000_powerup	*None*	Powers up the FMC site for ADC5000 operation.
adc5000_reset	*None*	Resets and initializes ADC5000(ADC, PLL, clock routing) with defau values
adc5000_configurepll	• Reference clock value (in Hz) • The data clock frequency (in Hz). Must be 2x the desired sampling rate • The frequency outputted to clock ouput connector (in Hz). Must be 2x the desired sampling rate • The clock frequency sent to the FMC connector (in Hz). Must be 1/8 of the desired sampling rate • The frequency of the sync signal (in Hz). Can be 1/8 of the desired sampling rate • The source of the reference clock ([0, 2])	Configures the ADC5000 PLL to provide clock to the ADC at the specified frequency. Clock source: • 0 for using external clock • 1 for using external reference • 2 for using internal 100 MHz reference
adc5000_setmode	• Mode ({1, 2, 4})	Selects the acquisition mode. 4 channels allow data rate up to 1.25 GSps, 2 channels allows data up to 2.5 GSps and 1 channel allow data rate up 5 GSps. These mo select the number of channel and which front panel connector is chosen. • 1 for 1 channel ADC Mode (ADC A), • 2 for 2 channel ADC Mode (ADC A and C). • 4 for 4 channel ADC Mode (ADC A, B, C and D).
adc5000_setsync	• Sync ([0, 3])	Sets the ADC sync source • 0 for external clock • 1 for carrier Sync signal • 2 for PLL Sync output • 3 for no Sync signal
adc5000_plllock	*None*	Prints the current status of ADC5000 PLL
adc5000_settestmode	• testmode ([0, 2])	Configures the ADC so that it output a test signal0 for normal mod • 1 for bitflash test mode • 2 for ramp test mode
adc5000_calibrateiodelay	*None*	Calibrates the digital communication between the FMC ADC5000 and the FPGA carrier board. Sweep IO delay, to find the most stab delay to apply between the data lanes and the ADC clock.
adc5000_getadcstatus	*None*	Prints the current status of ADC5000 ADCs
adc5000_setstandby	• standby ([0, 3])	Selects the standby configuration of the ADC5000 ADC chip. • 0 for full active • 1 for standby A and B • 2 for standby C and D • 3 for full standby
adc5000_setcoding	• coding ([0, 1])	Selects the coding configuration of the ADC5000 ADC chip. • 0 for Binary format • 1 for Grey format
adc5000_setbandwidth	• bandwidth ([0, 1])	Selects the bandwidth configuration of the ADC5000 ADC chip. • 0 for Nominal bandwidth (1.5 GHz) • 1 for Full bandwidth (3.2 GHz)
adc5000_setadjtriggerdelay	• delay ([1, 32])	Adjusts the trigger delay. The trigger from the trig in connector can be delay for 1 to 32 ADC cycles.

Command	Argument	Description	
adc5000_setoffset	• channel ([1, 4]) • offset ([0, 1023])	Adjusts the channel offset for the selected ADC channel. Offset variation range: ~± 40 LSB, 1024 steps. • 0x000 : Maximum positive offset applied, • 0x1FF : Minimum positive offset applied, • 0x200 : Minimum negative offset applied (0 LSB offset), • 0x3FF : Maximum negative offset applied.	*ad*
adc5000_setgain	• channel ([1, 4]) • gain ([0, 1023])	Adjust the channel gain for the selected ADC channel. Gain variation range: ~±10%, 1024 steps (1 step ~0.02%). • 0x000 : Gain shrunk to min accessible value, • 0x200 : Gain at Default value (0 dB), • 0x3FF : Gain Increased to max accessible value.	*ad*
adc5000_setphase	• channel ([1, 4]) • phase ([0, 1023])	Adjust the channel phase for the selected ADC channel. Delay control range for edges of internal sampling clocks: ~±15 ps (1 step ~30 fs). • 0x000 : ~ -15ps correction on selected channel aperture Delay, • 0x200 : 0ps correction on selected channel aperture Delay, • 0x3FF : ~ +15ps correction on selected channel aperture Delay.	*ad*

Table 12 ADC5000 commands

3.10 LVDS-xIn-xOut Module

Command	Argument	Description
fmclvds_reset	*None*	Resets and initializes to default values the fmclvds.
fmclvds_powerup	*None*	Powers up the FMC site for fmclvds operation.
fmclvds_presence	*None*	Verifies the presence of the fmclvds core and the FMC daughter card.
fmclvds_select	• Position (0 for bottom, 1 for top)	Selects the card to access.
fmclvds_set_channel_dir	• Group • Direction (0 for output, 1 for input)	Sets the direction of an LVDS group (according to the hardware).
fmclvds_set_channel_powerdown	• Group • Powerdown (0 for down, 1 for up)	Powers up or down an LVDS group.
fmclvds_set_channel_preemphasis	• Group • Preemphasis (0 for off, 1 for on)	Sets or clears the preemphasis status of an LVDS group.

Table 13 LVDS-xIn-xOut commands

3.11 Aurora Module

Command	Argument	Description	
aurora_set_resets	• 'nb' : Aurora core ID {1,2,...}	Reset Aurora FPGA core.	*aurora_re*
aurora_getchannelstatus	• 'nb' : Aurora core ID {1,2,...}	Get channel status (up or down)	*aurora_ge*
aurora_resetrxfifo	• 'nb' : Aurora core ID {1,2,...}	Reset Aurora RX FIFO content	*aurora_re*
aurora_resettxfifo	• 'nb' : Aurora core ID {1,2,...}	Reset Aurora TX FIFO content	*aurora_re*
aurora_getrxdatacount	• 'nb' : Aurora core ID {1,2,...}	Display the number of bytes received by the RX channel in bytes.	*aurora_ge*
aurora_gettxdatacount	• 'nb' : Aurora core ID {1,2,...}	Display the number of bytes sent by the TX channel in bytes.	*aurora_ge*
aurora_getrxdatarate	• 'nb' : Aurora core ID {1,2,...}	Display the number of bytes received by the RX channel in bytes in the last seconds.	*aurora_ge*
aurora_gettxdatarate	• 'nb' : Aurora core ID {1,2,...}	Display the number of bytes sent by the TX channel in bytes in the last seconds.	*aurora_ge*
aurora_setgtxtxparam	• 'nb' : Aurora core ID {1,2,...} • 'TxDiffCtrl': Driver Swing Control. Value from 0 to 15 • 'TxPostEmphasis': Transmitter Post-Cursor TX Pre-Emphasis Control. Value from 0 to 31 • 'TxPreEmphasis': Transmitter Pre-Cursor TX Pre-Emphasis Control. Value from 0 to 15	Apply the given TX parameters to the Multi-Gigabit Transceiver (MGT) components.	*aurora_se*
aurora_getgtxtxparam	• 'nb' : Aurora core ID {1,2,...}	Display the current Multi-Gigabit Transceiver (MGT) component TX parameters.	*aurora_ge*
aurora_setgtxrxparam	• 'nb' : Aurora core ID {1,2,...} • 'RxEqMix' : Receiver Equalization Control. Value from 0 to 7 • ‘DfeTap1’ : DFE tap 1 weight value control. Value from 0 to 31	Apply the given RX parameters to the Multi-Gigabit Transceiver (MGT) components.	*aurora_se*
aurora_getgtxrxparam	• 'nb' : Aurora core ID {1,2,...}	Display the current Multi-Gigabit Transceiver (MGT) component RX parameters.	*aurora_ge*
aurora_getdfeeyedacmon	• 'nb' : Aurora core ID {1,2,...}	Display the 'Averaged Vertical Eye Height' of the Multi-Gigabit Transceiver (MGT) components.	*aurora_ge*

Table 14 Aurora commands

3.12 PPSSync Module

Command	Argument	Description
ppssync_presence	*None*	Verifies the presence of the PPSYNC core and FMC daughter card
ppssync_ref_dac_init	• DAC value	Initializes the clock disciplining DAC to value
ppssync_ref_dac_limit	• Minimum DAC value • Maximum DAC value	Sets minimum and maximum DAC values
ppssync_start	• ADC and DAC frequency of the FMC card • Integration time • Proportional gain • Integral gain	Starts clock disciplining
ppssync_stop	*None*	Stops clock disciplining
ppssync_read_ref_dac	*None*	Read ref. DAC value from file system
ppssync_save_ref_dac	*None*	Save current DAC value to file system
ppssync_read_pi_profile	*None*	Read PI profile from file system
ppssync_save_pi_profile	*None*	Save current PI profile to file system
ppssync_get_info	*None*	Get status of pps synchronisation

Table 15 PPSSync commands

3.13 Mestor LVDS Module

Command	Argument	Description	
lvds_presence	• Group	Verifies the presence of the lvds core and the Mestor daughter card.	*lvds_prese*
lvds_getmode	• Group	Gets the LVDS mode for the selected group.	*lvds_getm*
lvds_setoutputenable	• Group • Output enable (1 for output, 0 for input, for each GPIO of the group)	Sets the output enable value for the selected group.	*lvds_setou*
lvds_setvalue	• Group • Value (1 or 0 for each GPIO of the group)	Sets the output value for the selected group in GPIO mode.	*lvds_value*
lvds_getvalue	• Group	Gets the input value for the selected group	*lvds_getva*
lvds_reset	• Group	Resets the selected group	*fmclvds_s*
lvds_reset_fifo	• Group	Resets the selected group in synchronous mode	*fmclvds_s*

Table 16 Mestor LVDS commands

3.14 MO1000 Module

Command	Argument	Description
mo1000_powerup	• ID of the MO1000 (1 or 2)	Powers up a MO1000 FMC
mo1000_reset	• ID of the MO1000 (1 or 2)	Resets a MO1000 to its default non operating condition
mo1000_init	• ID of the MO1000 (1 or 2)	Initializes a MO1000 to its default operating condition
mo1000_writereg	• ID of the MO1000 (1 or 2) • Device to access • Register address (in hex) • Value to write (in hex)	Writes a value in specified device register address *WARNING: Use for test purpose only, it may corrupt board behaviour or damage it if used incorrectly.*
mo1000_readreg	• ID of the MO1000 (1 or 2) • Device to access • Register address (in hex)	Reads a value from specified device register address
mo1000_getstatus	• ID of the MO1000 (1 or 2)	Gets MO1000 board status and pattern test errors
mo1000_gettemperature	• ID of the MO1000 (1 or 2) • Output format	Gets board PCB and DACs temperatures
mo1000_getchannelcalibstatus	• ID of the MO1000 (1 or 2)	Gets MO1000 board calibration status
mo1000_dodaccalibration	• ID of the MO1000 (1 or 2)	Performs a MO1000 calibration
mo1000_setdacoutctrl	• ID of the MO1000 (1 or 2) • DAC channel ([1,8]) • DAC output state	Controls the DAC output state for the specified channel
mo1000_setclockconfig	• ID of the MO1000 (1 or 2) • Clock source • Source clock frequency (if clock source is not 125MHz) • DAC clock desired frequency • Master clock mode • Master clock frequency (if clock mode is manual)	Sets up the clocks of the module
mo1000_writeclockconfig	• ID of the MO1000 (1 or 2)	Writes the module clocks setup prepared with mo1000_setclockconfig
mo1000_getpllconfig	• ID of the MO1000 (1 or 2)	Gets the module PLL parameters prepared with mo1000_setclockconfig
mo1000_setpllconfig	• ID of the MO1000 (1 or 2) • Charge pump current • C1 value • R2 value • C2 value • R3 value • C3 value	Sets the module PLL loop filter parameters
mo1000_setdacparinterpolation	• ID of the MO1000 (1 or 2) • Interpolation mode	Configures interpolation mode for all DAC channels
mo1000_setdacpardcoffset	• ID of the MO1000 (1 or 2) • DAC channel ([1,8]) • DC offset ([-32768,32767])	Controls the output dc offset for the specified DAC channel
mo1000_setdacpargain	• ID of the MO1000 (1 or 2) • DAC channel ([1,8]) • Gain (ScaleFactor = gain / 64)	Controls the digital gain for the specified DAC channel *Warning: gain greater than 64 (scale factor of 1) could cause signal saturation*
mo1000_dodacupdate	• ID of the MO1000 (1 or 2)	Configures the DACs for operation with all the current parameters (defaults after 'mo1000_reset')

Command	Argument	Description	
mo1000_setdacparactchannel	• ID of the MO1000 (1 or 2) • Number of active channels	Configures the number of DAC active channels (inactive channels are in power down)	*mo1000_se*
mo1000_getclkmaster	• ID of the MO1000 (1 or 2)	Verifies if this module is a clock master	*mo1000_ge*
mo1000_setdacparisinc	• ID of the MO1000 (1 or 2) • DAC channels pair • Real coefficient 0 • Real coefficient 1 • Real coefficient 2 • Real coefficient 3 • Real coefficient 4 • Imaginary coefficient 0 • Imaginary coefficient 1 • Imaginary coefficient 2 • Imaginary coefficient 3 • Imaginary coefficient 4	Configures inverse sinc filter coefficients for the specified DAC channels pair.	*mo1000_se* *0 0 0 0 0*
mo1000_setdacparisincctrl	• ID of the MO1000 (1 or 2) • DAC channels group • Inv sinc filter state	Controls the DAC inverse sinc filter state for the specified channels group	*mo1000_se*
mo1000_setdacparfinemod	• ID of the MO1000 (1 or 2) • DAC channels group • Frequency tuning word ((uFtw = Fcenter/Fdac * 4294967296) • NCO phase offset (0.0054931640625 deg per count) • Sideband selection	Configures the fine modulation parameters for the specified DAC channels group	*mo1000_se* *122333866*
mo1000_setdacparfinemodctrl	• ID of the MO1000 (1 or 2) • DAC channels group • Fine modulation state	Controls the fine modulation state for the specified DAC channels group	*mo1000_se* *enable*
mo1000_setdacpardataformat	• DAC channels group	Controls the digital data format for the specified DAC channels group	*mo1000_se* *2compleme*
mo1000_setdacparquadphase	• ID of the MO1000 (1 or 2) • DAC channels pair • Real phase correction • Imaginary phase correction	Configure quadrature phase for the specified DAC channels pair.	*mo1000_se*
mo1000_getversions	• ID of the MO1000 (1 or 2)	Gets FPGA core and mo1000 driver versions	*mo1000_ge*
mo1000_checki2c	• ID of the MO1000 (1 or 2)	Verifies i2c bus to detect all module devices for diagnostic purpose	*mo1000_ch*
mo1000_getcorefrequency	• ID of the MO1000 (1 or 2)	Gets FPGA core and DACs reference frequencies	*mo1000_ge*
mo1000_settestmode	• ID of the MO1000 (1 or 2) • Test mode selection • Even 16 bits pattern • Odd 16 bits pattern	Configures DAC test mode	*mo1000_se* *mo1000_se*

Table 17 MO1000 commands

3.15 System Monitor module

Command	Argument	Description	
sysmon_get_info	*None*	Prints all System Monitor information, such as the FPGA core's current temperature.	*s*

Table 18 Sysmon commands

3.16 Timestamp module

Command	Argument	Description	
timestamp_set_time	• value	Set the timestamp value at the next PPS rising edge event	*t*
timestamp_get_time	*None*	Prints the current timestamp value	*t*
timestamp_reset	*None*	Reset the timestamp value	*t*

Table 19 Timestamp commands

4 Building a CLI Script

The CLI is a Python script file, which means that it dynamically parses and runs commands. It is also possible to write commands in text files and have them run as if you wrote the commands they contain directly at the command prompt. These files are known as batch files on Windows systems, and shells on Linux systems. Batch file examples can be found in the e*xamples* folder.

4.1 Writing a Script

The first command in a CLI script should always be the connect command to initiate the connection between the host and the targeted device. Once the connection is made, the CLI verifies automatically the presence of an FMC daughter card and tries to identify it. The CLI will not allow commands for an undetected FMC card.

Following the connection, all the other commands can be called, one at a time. The user should call the commands following the typical configuration order of the system it is controlling. The programmer can insert echo commands to have feedback in the command window.

The following is an example of a typical CLI script.

```
connect 192.168.0.101

shell echo Write 0 to bit 14 of CLK_CTRL register to disable the clock
write 0x70000004 0x10aa

fmcradio_powerup
fmcradio_reset
fmcradio_setrevision SDR_C
fmcradio_pll 30720000 76800000 30720000
fmcradio_pll_lock

shell echo Write 1 to bit 14 of CLK_CTRL register to enable the clock
shell echo now that the PLL is locked
write 0x70000004 0x50aa

fmcradio_band low
fmcradio_filter 943MHZ
fmcradio_lpf rx 2.5MHZ
fmcradio_lpf tx 2.5MHZ
fmcradio_lime_pll rx 30720000 943000000
fmcradio_lime_pll tx 30720000 943000000
fmcradio_rx 0 -5
fmcradio_tx -15 5 0
fmcradio_rxvga_calibrate
fmcradio_lpfcalibrate 30720000
write 0x70000018 57266231
write 0x7000001c 0x1
```

4.2 Running a Script on Windows

To run a script on Windows:

1. Create a new batch file.
2. Specify the path to the CLI and the script file.

   For example,

   ```
   call %BASROOT%\LaunchCLI.bat  C:\myscript.txt.
   ```
3. Run the new batch file.

4.3 Running a Script on Linux

To run a script on Linux:

4. Create a new shell file.
5. Specify the path to the CLI and the script file.

   For example,:

   ```
   set -e
   BASROOT=/opt/Nutaq/bas
   DIRNAME=$(dirname $0)
   sudo bash $BASROOT/LaunchCLI.sh  myscript.txt
   ```
6. Run the new shell file.