SIGTRONA

FlexDDS-HD

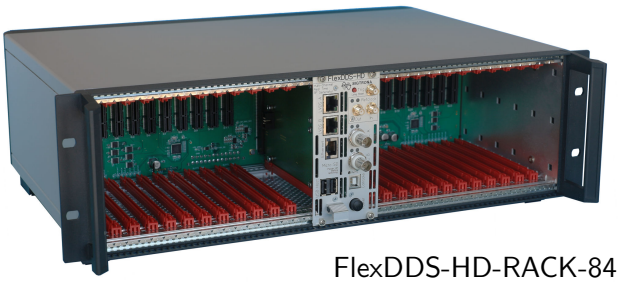
Low Noise Multi-Channel Phase-Coherent Radio Frequency Source

Operator's Manual

Multi-Channel Phase-Coherent Radio Frequency Source

▷ FlexDDS-HD-RACK-84

- Up to 24 slots, each one can be fitted with a FlexDDS-HD-AD9910 dual RF generator module
- Up to 48 channels in total, all synchronized with precisely known and adjustable phase relationship between channels
- GBit Ethernet interface with high speed data streaming capability (> 30 MBytes/s)
- External reference clock input and output
- 1 GHz direct low phase noise clock output
- Global trigger inputs that affect all slots simultaneously



FlexDDS-HD-RACK-84
Up to 48 RF Generator Channels

- GBit Ethernet: Connect anywhere in the lab, no USB cables, no special OS drivers
- Extensible: Populate slots later as needed

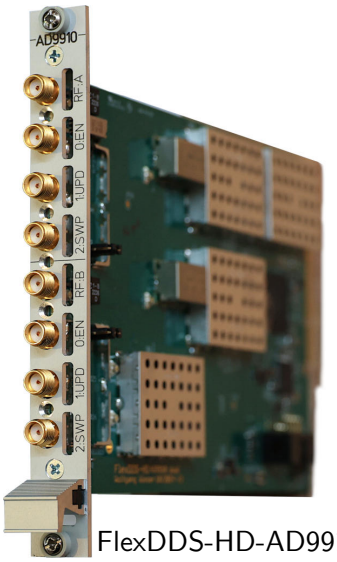
▷ FlexDDS-HD-RACK-ULN-1G (optional)

- Temperature-controlled ultra-low phase noise reference oscillator
- Aging < ±5 ppb/day, < ±300 ppb/year
- 1 to 100 MHz external input (GPSDO, Rubidium)
- 1 GHz ultra-low noise clock output
- 1 to 100 MHz configurable output

Image missing.

▷ FlexDDS-HD-AD9910

- Two independent output channels for up to 400 MHz (1 GS/s sampling rate)
- Excellent signal quality
- Dual-channel operation with precisely known and adjustable phase relationship between channels
- Real-time control of all signal parameters
- Option **AMP**: Dual analog inputs for analog modulation with digitally controlled gain and intercept
- Phase-continuous frequency, phase and amplitude tuning
- Per-channel high speed command processor with 8 ns timing resolution
- 4-6 programmable digital IOs for trigger, fast on/off, ramp control, fast profile switching, ...



FlexDDS-HD-AD9910

General Description

The FlexDDS-HD is a multi-channel phase-coherent radio frequency (RF) source. It is especially targeted for QBit control in quantum computers but could also be useful in radar simulations and other applications. The design deliberately targets the needs of experimental physicists who want to control all signal parameters in real-time from a computer.

Initially, a series of actions (like changes in amplitude or frequency, start of frequency sweeps,...) is compiled into commands which are then transferred to the FlexDDS-HD. Each time a (real-time asynchronious) trigger input is activated, the FlexDDS executes one or several commands and waits for the next trigger event. There is no limit on the number of successive commands as they can be streamed continuously from the host computer.

One outstanding feature of the FlexDDS-HD is its defined and known phase relationship between channels. For example, two channels can easily be set up to produce an RF output at the same frequency and with equal phase. Slightly detuning the frequency of one channel will then linearly increase the phase difference between the two channels.

Comparison Between the FlexDDS-NG and the FlexDDS-HD

The FlexDDS-HD improves over the FlexDDS-NG in the following ways:

- Lower phase noise
- Improved spectral purity
- Higher stability reference oscillator
- Supports up to 24 modules per rack (compared to 6; hence the name "HD" for "high density")
- Matched propagation times on the backplane, so no phase buildup from left to right in the rack
- RF outputs can be configured as either grounded or floating; in the floating state, the SMA connector's shield is disconnected from ground (transformer coupled)
- Choice of 3 different output amplifiers with up to FIXME dBm output power
- A total of 4 or 6 digital IOs per RF generator module (compared to 3; with or without option -ADC, respectively)
- Direct 19" rack mounting (rack width is full 84 divisions instead of only 63)
- Increased independence of RF channels and their data streams on a slot module
- Built-in web server for config and firmware updates
- Network admin interface to reboot slot modules, control reference oscillator etc.
- RF generator modules are hot-pluggable
- RF generator modules are virtually independent and can be reset individually; this allows certain modules to run continuously providing a stable RF source while other modules can be reset frequently
- The per-module USB port found on the FlexDDS-NG is exposed as indivitual TCP network ports on the FlexDDS-HD
- Extensive hardware health monitoring

The FlexDDS-NG provides the following features which were removed in the FlexDDS-HD:

- RF on/off hardware switches on the frontpanel
- Auxiliary RF output ports which provide lower power copies of the main RF outputs for monitoring

- The ability to fine tune the phase/delay between channels on an RF generator module (command `lmkdly A`). Tunable delays introduce a phase noise penalty and the FlexDDS-HD has well matched phases on the generator modules out of the box.
- RF level adjustment screws (the FlexDDS-HD also provides the option of the same type of variable RF power amplifier but the power level is controlled digitally rather than via an adjustment screw)
- An USB device port on each module (the USB is routed internally in the FlexDDS-HD and exposed as a TCP network port)

Contents

Chapter 1

Getting Started with the FlexDDS-HD

So you just got your shiny new FlexDDS-HD and simply want to get started without reading a long manual? Here are are your first steps:

1. Hook up a network cable to the "Ethernet" port on the main control module in the center of the rack. Do not plug anything into the "LVDS" receptables. If you wish to configure a static IP address, follow the steps in section 2.3 on page 14.
2. Power up the FlexDDS-HD, wait for it to boot and check that all RF generator modules have a green light in the center LED and no red blinking light.
3. You need the device's IP address now so if you have not set a static one, either ask your DHCP server or connect to the USB "Console" on the main control module. Take a terminal program that can deal with USB/serial ports (e.g. PuTTY, minicom), set it up for 115 200-8N1 and press enter to see the IP address.
4. Take some network terminal program (e.g. telnet, PuTTY) and connect to TCP port 27000 on the FlexDDS-HD. This is the admin/control interface. Type `help` to get a list of commands and then type `slot p` to view information about the installed RF generator modules. Modules sit in *slots* and those are numbered 0 to 23 from left to right in the rack.
5. If e.g. slot 3 has a module inserted, type `slot 3 bq:cmd ftest f=125M` The two red LEDs next to the RF outputs on the modules should flash quickly. You should now see a clean sine wave at 125 MHz generated on both RF channels on module 3. When entering the command, note exactly where are spaces and where not.
6. Now try `slot 3 bq:cmd ftest wide_sweep` and you should see a repeated sweep from close to DC up to 450 MHz and back.
7. OK, the last two were just test/demo commands which come handy for testing the RF signal chain but now some real stuff: Open a second similar network terminal to port 262xx (where xx is an index of an installed module, they are counted 0...23 from left to right. Without first pressing enter or any other key, enter this:

```
dcp dds_reset
dcp 0 spi:stp0=0x3fff00002147ae14
dcp 1 spi:stp0=0x3fff00002147ae15
dcp update:u
dcp flush
```

After that, you should see a 130 MHz waveform on both RF:A and RF:B outputs which are detuned by 0.23 Hz relative to each other.

1.1 Best Practices

Here are a couple of best practices:

- Open one TCP connection for each RF generator module to a DCP network port (262xx or 263xx). You can make an exception for those RF generator modules with are mainly static and do not see many instructions; for those it is fine if they are controlled via the command `dcp` on the admin/-control interface).
- Use `ddsman` *NNN* `dds_reset` from the admin/control interface (port 27000, 27001) to re-initialize a RF generator module whenever the need arises. Avoid using the `dcp start/stop/reset/dds_reset` (or `dds reset`) commands from in the DCP network ports because those network streams can stall if you send a lot of commands thereby blocking the possibility to reset from within the stream.
- Stream DCP instructions early and keep FIFOs populated to ensure that instructions are available for execution whenever external triggers need them.
- Pre-load changes onto the synthesizer chips *before* waiting for a trigger and then update immediately after the trigger. Then pre-load the changes for the next trigger already. This gives the fastest deterministic timing. If you want frequency X, do *not* wait for a trigger and *then* start writing the changes for X onto the synthesizer *after* the trigger thereby losing a lot of time before you can actually update.
- Look for firmware updates on `www.sigtrona.com/flexdds-hd` and keep your device up to date.

Chapter 2

FlexDDS-HD Rack

The FlexDDS-HD Rack (order code: *FlexDDS-HD-RACK-84* is a 19" enclosure that can hold up to 24 RF generator slot modules. In 19" metrics, it has a height of 3 units and the full width of 84 units enabling easy rack mounting.

Each of the modules can be equipped with an AD9910-based FlexDDS-HD-AD9910 dual channel RF generator.

The FlexDDS-HD Rack provides a GBit network interface that can be used to control all the slot modules using a single and easy to use high speed connection. No specific drivers are needed and the GBit network allows access from multiple computers over greater distance than e.g. a USB interface.

Each RF generator module provides its own DDS command processor (DCP). Those modules are controlled mainly by sending a sequence of DCP commands. For each slot module, the FlexDDS-HD Rack provides a FIFO buffer capable of holding millions of DCP instructions per slot. An unlimited amount of instructions can be streamed in real time.

2.1 Rack Backpanel: Power, Fuse, Ventillation

The FlexDDS-HD backpanel has a standard power inlet with a mechanical 2-phase power switch and a built-in fuse. Please only replace the fuse with a new one of same current rating. The current rating is written on the backpanel.

Please ensure sufficient ventillation and do not block/obstruct the fan outlets.

2.2 Rack Control Module: Front Panel Elements

On the FlexDDS-HD, the rack control module is located in the middle of the rack with 12 slots left of it and 12 slots to the right. This layout eases signal routing between the rack control module and the RF generator slots.

Power pushbutton: Push to switch on/off the FlexDDS-HD rack. The button can show the following power states:

- **No light:** Rack is powered down, no standby power (check power switch on the back).

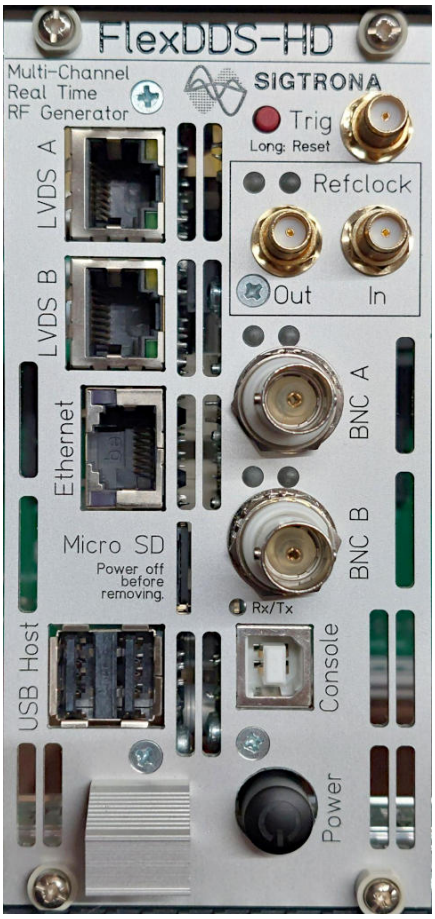


Figure 2.1: Frontpanel elements of the FlexDDS-HD rack control module which is installed in the middle of the rack.

- **Constant yellow:** Rack powered down, standby power available. Push to power up the rack.
- **Blinking green:** Rack booting up (low blink frequency), application starting up (fast blinking).
- **Constand green:** Device is up and running.
- **Alternating yellow/green:** Firmware update in progress. Do not switch off.
- **Blinking yellow:** Device is shutting down.

A short press in the running state will gracefully shut down the device. You can use a long press to force it off.

Red pushbutton: A short press acts as a trigger which can be routed to the slots via the rack trigger bus. A long press performs a configurable type of reset action (see config variable `ms_trig_long_press_action`).

BNC A, BNC B: These are two independent DIOs which can be programmed to do different things (see admin command `dio`). Most commonly they are routed to the rack trigger bus which is accessibly to the slots. The voltage level is 5 V logic by default. In order to opt for 3.3 V voltage levels, refer to figure 2.2.

Refclock in (SMA connector): This is a reference clock input. To lock the complete FlexDDS-HD rack and all RF generators to an external reference clock, apply a high quality signal here. There are various config options, most notably `clkgen_ext_refclk_Hz` specifies the expected reference clock frequency (default: 10 MHz). A sine wave amplitude of 100 mV is sufficient for locking althought for improved phase noise, higher amplitudes are recommended (up to beyond 2 V amplitude).

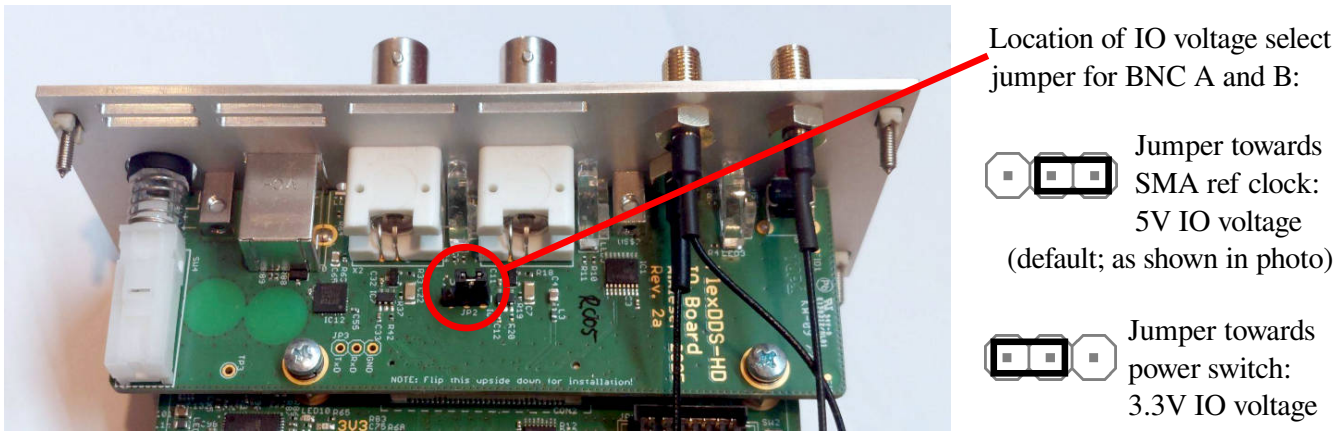


Figure 2.2: IO voltage selection jumpers for the rack control module. To remove the control module, open the 4 screws holding it and pull it out *using the provided handle*. The BNC A and B have one common IO voltage setting (3.3 V or 5 V). When installing, be sure to push *using the provided handle* to avoid bending the board.

When the rack detects an external reference, the green "LED" next to "Refclock" will start blinking. The red LED denotes an error in the clocking but is also activated as a user feedback for a long press of the "Trig" pushbutton.

Refclock out (SMA connector): This is a reference clock output with programmable frequency, see config option `clkgen_refclk_out_freq_MHz`. The output is divided down from the internal 1 GHz oscillator and *not* a copy of the Refclock in.

SMA next to Trig: This is a buffered copy of the 1 GHz low noise reference clock distributed to all RF generator slots on the rack. Use config option `clkgen_ext_1ghz_out_enable` to enable.

Ethernet connector (RJ-45): Connection for GBit Ethernet to send commands to the rack.

LVDS connectors (RJ-45): These allow to communicate via LVDS signals with the rack. Do not connect to network hardware, it may result in hardware damage. This is not yet implemented.

USB host connectors (USB-A): These are recognized as USB host ports by the embedded Linux running on the main control module. Currently unused.

USB console (USB-B): This is a virtual COM port using a FT232 USB/serial converter chip (settings: 115 kbaud-8n1) and it shows the boot console of the embedded Linux and application status messages. It can also be used to display the network address and firmware version. The Tx/Tx LED next to it blinks red if data is sent/transmitted (only if USB connected).

Micro SD: Contains a micro-SD card with 3 partitions:

- FAT partition for firmware updates
- FAT partition for configuration files (`flexdds_hd.cfg`, `ethernet.txt`)
- EXT4 partition for log files and crash dumps, also accessible via the built-in web server

Gently press to remove. Only remove when powered off. The SD card contacts face towards the text.

2.3 The GBit Network Interface on the FlexDDS-HD Rack

The FlexDDS-HD Rack provides a GBit Ethernet port on the control module labeled “Ethernet”. When a network cable is plugged in, the yellow LED indicates carrier detection. The green LED blinks upon network activity.

Note: Attention LVDS Ports

Do not plug any network cables into the receptables labeled “LVDS”. This may harm your router and/or the FlexDDS-HD Rack. Network has to be connected to the receptable labeled “Ethernet”.

Configuring the IP address:

By default, the FlexDDS-HD Rack expects to receive an IPv4 network address via DHCP. As soon as a network cable is connected, it will automatically broadcast DHCP queries to configure its network address. It is recommended to configure the DHCP server in the network to hand out the appropriate IPv4 address based on the MAC address of the FlexDDS-HD Rack. See chapter 2.4 on how to obtain the MAC and network addresses.

You can also set a static IP address. In order to do so, you need to edit a configuration file which is stored on the micro-SD card installed on the rack control module. Here is a step-by-step instruction on how to do this:

1. Power down the FlexDDS-HD.
2. Remove the micro-SD card. It is accessible from the rack control module and labeled ”Micro SD”. Gently press in the card (e.g. with a coin) until you hear a quiet “click” sound. The card then comes back out and you can remove the card.
3. Put the card in a card reader. It has a FAT (VFAT) file system on it which can be read by any current Windows, Linux and Mac OS.
4. Edit the file called `ethernet.txt` with a standard text editor such as Notepad on Windows. Do not use Office or Word as editor.
5. Eject the micro-SD card from the card reader and put it back into the FlexDDS-HD. Again, press gently until you hear a “click” sound. The card is now again locked and cannot be removed simply by pulling it. The electrical contacts on the SD card face towards the frontpanel text “Micro SD”.
6. Power up the FlexDDS-HD again. The network address is now configured.

The sample content of the `ethernet.txt` file looks like this:

```
# Comment out all lines for DHCP.
# Enter all of the following (address, netmask, broadcast) to configure
# a static IP address.
#address 192.168.11.99
#netmask 255.255.255.0
#broadcast 192.168.11.255
#gateway 192.168.11.2

# You can also configure a MAC address if needed.
hwaddr 00:0A:35:00:01:23

# If gigabit speed or auto-negotiation do not work, you can set the speed
```

```
# manually (e.g. 100 MBit):
# speed 100
```

Note: Insecure Networks

The FlexDDS-HD Rack is *not* meant to be operated in public networks. Do not allow the FlexDDS-HD Rack to be world-accessible over the internet. Always operate in local networks behind routers or firewalls that provide protection.

2.4 The USB Console on the FlexDDS-HD Rack

The USB console on the rack is a virtual COM port and needs to be configured for a baud rate of 115 200, with 1 stop bit and no parity (commonly called "115200 8N1").

The port makes use of the FT232 chip, so the FTDI drivers need to be installed. All current Linux distros ship those and Windows will typically install them via network automatically.

You can connect to this console using terminal emulators like PUTTY (Windows) or minicom (Linux).

If you press return/enter on the console a status text is written which indicates the IP address and firmware version:

```
IP: 192.168.11.99     MAC: 00:0A:35:02:00:03     (network up)
FlexDDS-HD v1.2c (Tue Jan  6 19:50:21 CET 2026)
Welcome to FlexDDS-HD
```

2.5 DCP Network Ports and FIFO Operation

The FlexDDS-HD Rack opens multiple TCP ports, as detailed in the next sections. There is one port for each slot and each protocol. E.g. for the text based protocol, slots 0...23 correspond to ports 26200...26205, respectively (see config option `net_dcp_text_port0`). Note that the FlexDDS-NG uses different ports.

You need to open a dedicated (independent) network TCP connection to the FlexDDS-HD Rack for each slot module. Over this network connection, the FlexDDS-HD Rack is fed with DCP instructions and other commands.

The DCP instructions are queued into a large per-slot FIFO which can hold, by default, up to 1 million DCP instructions (per slot). The FIFO content is streamed to the slot modules over the backplane. Each RF generator module has a smaller DCP instruction FIFO (typically 4096 instructions per channel) to avoid effects caused by transmission latency within the rack (see Figure 2.3).

All FIFOs implement flow control which propagates back to the network TCP connection: Once the FIFOs are full, the network transfer is stalled, so the TCP connection will simply not take more data. As soon as instructions are executed by the slot modules and there is available space in the FIFO, the TCP connection accepts more data. This way you can open a connection, keep it open and stream an infinite amount of data over the connection.

Each port can accept up to 1 connection at the same time. If a connection is active and a second connection is made, then the old connection is closed and the new one takes over. This helps dealing with certain environments (e.g. LabView) which do not always properly close connections.

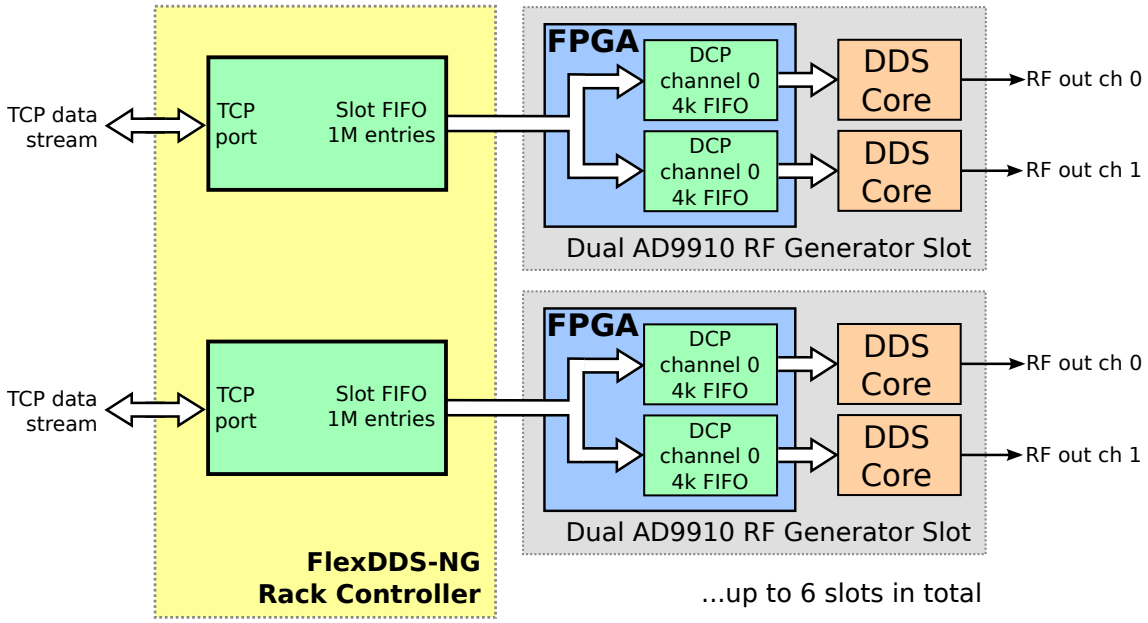


Figure 2.3: Data streams and FIFOs in the FlexDDS-HD Rack. Each slot has its own TCP port, TCP data stream and large slot FIFO within the rack controller. Each slot module has its own smaller FIFO per channel. There is one data stream per slot which is divided into multiple channels on the slot module. Hence, if on a RF generator module, one 4k FIFO runs full, *both* FIFOs on the module can no longer be supplied with instructions.

If a connection is closed and opened again or if a new connection replaces an old one, the content of the large DCP FIFO is preserved.

The admin shell on the FlexDDS-HD allows to disconnect network connections and reset DCP FIFOs. Also, a long press of the red pushbutton can be configured to reset all slots (see config option `ms_trig_long_press_action`).

Note: Association Between Slots and TCP Ports

It is important to understand that each slot (0...23) is associated with a specific TCP port and has a dedicated FIFO buffer in the rack. Hence, each slot module is fed with its own independent data stream. However, each slot module can have multiple channels and the instructions for these channels are in the same FIFO in the rack. See Figure 2.3.

This has one important consequence: Each slot module has a DCP instruction FIFO per channel (usually 4096 instructions per channel). As soon as *one* of these per-channel FIFOs is full, the data stream from the rack FIFO corresponding for that particular module is stalled. Now, if e.g. channel 0 executes instructions much faster than channel 1, then the DCP FIFO in the slot module's FPGA for channel 0 may run empty while the FIFO for channel 1 is still full. (E.g. channel 1 is blocked at a long `wait` instruction.) The DCP for channel 0 will then not be able to execute instructions in time because the slot module is considered "full" by the rack. The rack has a single FIFO per slot and cannot re-order instructions.

Solution: Ensure that DCP instructions for different channels of the same slot are queued in approximately the order in which they will be executed. You can deviate from the true order by up to the size of the per-channel slot module FIFOs. If timing is unclear, consider re-arranging the setup so that different

modules are being used.

2.5.1 DCP Text Network Protocol (TCP Port 262xx)

The TCP port for the text based DCP command interface is 262xx where the xx has to be replaced by the slot number 0...23. The port number can be changed via the configuration option `net_dcp_text_port0`.

Only TCP ports corresponding to functional slots are actually open, those slots left empty do not have an open TCP port.

After opening a TCP connection, the first 16 bytes to be sent are the ASCII representation of the authentication token. This is sort of a "fixed password" as the most basic means to prevent unauthorized access. The authentication token is `31546049441b0d`*xx* where the last two digits, *xx*, have to be replaced by the slot number (converted to hex notation, hence `00` to `17`; e.g. slot 15 has `0f`). This token is set via the configuration option `net_dcp_text_auth_token` and if set to zero, no auth token is required.

After this authentication step, text based commands are read.

Each command is terminated by a CR ('\r') or LF ('\n') character (or both). From a Linux shell, you can use `telnet` or `netcat` to access the FlexDDS-HD Rack. On a Windows host, Putty can be used when choosing the connection type "Telnet". **NOTE: In putty, you must set "negotiation mode" to "Passive" in the configuration under Connection → Telnet.** In general, any tool or programming language (LabView, Matlab,...) that can open a TCP connection and send text over it will be able to send commands to the FlexDDS-HD Rack.

The following network commands are supported in text mode:

`dcp [0|1] ...`	Issue a DCP command (e.g. to feed a DCP instruction into the FIFO). The DCP text mode commands are described in section 4.2.
`dds reset`	Same as `dcp dds_reset`; for compatibility with the FlexDDS-NG.
`set` *VAR*=*VALUE*	Configure certain properties; see below.
`quit`	Close the current network connection.

All commands refer only to the associated slot module.

The `set` command supports the following variables:

`set dcp_dump_isn=[0|1]`	If set to 1, the raw DCP instructions are echoed back.
`set resp_suppress_ok=[0|1]`	If set to 1, an "OK" response is not sent .

Here is an example network session with text commands over TCP port 26207 corresponding to 8th slot from left which is a FlexDDS-HD-AD9910 in this example:

`31546049441b0d07`	auth token for slot 7 ("password")
`Auth OK`	response that auth is OK
`dcp 0 spi:stp0=0x3fff00005999999a`	queue a DCP instruction for channel 0 (350 MHz, full amplitude)
`OK`	response from rack
`set resp_suppress_ok=1`	suppress "OK" responses
`dcp 1 spi:stp0=0x3fff00005999999e`	queue a DCP instruction for channel 1 (350.000001 MHz)
`dcp update:u!`	queue DCP update insn for channels 0 and 1
`set dcp_dump_isn=1`	request that DCP instructions are echoed back
`dcp 0 wr:cfg_dio_2=0x300`	set DIO A2 output HIGH on the slot module

`DCP: 0x11d1208200000300`	response: corresponding 64 bit rack DCP NFrame instruction
`dcp 1 wr:cfg_dio_0=0x300`	set DIO B0 output HIGH on the slot module
`DCP: 0x11d2208000000300`	response: corresponding 64 bit rack DCP NFrame instruction
`dcp flush`	ensure commands are flushed to the slot module

Responses sent back from the rack are written in green.

This is a more real-life example for the RF generator module in slot 2, hence to TCP port 26202. It sets two frequencies (on channel 0 and 1), waits a second and then sets the output amplitude to zero on both channels.

`31546049441b0d02`	sent auth token for slot 2 (“password”)
`Auth OK`	response that auth is OK
`set resp_suppress_ok=1`	suppress “OK” responses
`dcp dds_reset`	reset both RF output channels
`dcp 0 spi:stp0=0x2000000006e978d5`	queue a DCP instruction for channel 0 (27 MHz, half amplitude)
`dcp 1 spi:stp0=0x3fff0000072b020c`	queue a DCP instruction for channel 1 (28 MHz, full amplitude)
`dcp spi:CFR2=0x01000080`	next instruction channels 0 and 1 (enable ASF from STP)
`dcp update:u!`	queue DCP update insn for both channels and flush
`dcp spi:stp0=0x0000_0000_00000000`	set zero amplitude on both channels (ineffective until update)
`dcp wait:1000000:`	wait a second
`dcp update:u!`	queue DCP update insn for both channels anf slush
Before sending the next line over the network you must wait until the program above has finished. Otherwise it will be interrupted. (Of course if you want to interrupt and terminate the program because it is stuck, then it’s fine to send the reset any time.)	
`dcp dds_reset`	perform a reset to be ready for the next experiment run

2.5.2 DCP Binary ‘NFrame’ Network Protocol (TCP Port 263xx)

The text based network protocol has the disadvantage of imposing significant network and processing overhead because the DCP instructions have to be parsed on the CPU of the rack control module. This limits the throughput to about 250 000 DCP instructions per second or about 9 MBytes/s. (if `resp_suppress_ok` is set to 1).

FIXME: Check this for the FlexDDS-HD

The binary (‘NFrame’) protocol allows faster instruction streaming with less overhead. It allows to stream 2.5 million network frames (or DCP instructions) per second (20 MBytes/s). However, it requires that DCP instructions are converted to binary form on the controlling host computer.

The binary protocol operates on a separate set of TCP ports, namely 26300…26323 corresponding to slots 0…23. The port can be configured using the config setting `net_dcp_nframe_port0`.

The binary protocol consists of a series of *network frames*. Each network frame has a size of 8 bytes (64 bits) and they are transferred in little endian manner (i.e. native byte order on x86 host computers and ARM processors).

The format of a network frame looks like this:

FIXME: Document NFrame protocol for FlexDDS-HD

2.6 Per-Slot 'USB' Network Console (TCP Port 261xx)

Each of the RF generator modules internally has a microcontroller which communicates with the main control module via USB and offers a text terminal with debug logs, status information and a command line interface.[1]

The rack control module exposes this terminal via one dedicated network port per slot on TCP ports 261xx (where $xx = 0 \ldots 23$ is the slot address; see also config option `net_console_tty_port0`).

These ports are always open for all slots, even for unpopulated ones. You can connect to them using telnet or a similar program. They offer a direct connection to each slot module which is mainly useful for debugging and are *not* meant to be used to operate the FlexDDS-HD in a production environment.

For historical reasons, the commands interpreted directly by the slot module microcontroller are called "USB commands" and are documented in the chapters of the RF generator modules (chapter **??** on page **??** for the FlexDDS-HD-AD9910).

Apart from messages directly from the slot, the rack control module also injects logging messages related to the corresponding slot into the USB network console.

The console supports minimal command editing such as a command history (up/down arrow) and very limited tab completion (command name only).

Note: The rack control module buffers all the communication of all slot modules. So, if you connect to the TCP port after the slot module has booted up, you can sill retrieve most (if not all) of the boot messages which were buffered in the rack control module. Howevr, during startup some slot modules may boot before the USB connection has fully enumerated and their communication will not show up – even if you network connect before booting the slot module. If you need all that communication, see command `slot sm:seqboot=1` on page 25.

2.7 Admin/Control Command Line Interface (CLI) (TCP Ports 27000, 27001)

The FlexDDS-HD provides an interactive command line interface (CLI) for admin and control tasks on TCP port 27000 (see config option `net_admin_cli_port`). This is a fully featured CLI with command history (up/down arrow), full command editing and tab completion. The interactive admin CLI is used mainly for manual control, status display, debugging, maintenance, calibration and similar tasks.

The same interface in a non-interactive fashion is exposed as control interface on TCP port 27001 (config option `net_control_cli_port`). Choose this port to remotely control the FlexDDS-HD via automated scripting or device drivers. The non-interactive port does not echo back keystrokes and has no line editing features and will not save a command history.

You can connect to both ports via telnet or similar programs although many of these programs will not enable the enhanced command editing features. For Linux hosts, Sigtrona provides a program called `mini_telnet` which enables/supports all features the CLI offers when using the interactive port.

Note: You can open several (up to 8) admin/control CLI connections from different hosts at the same time.

The admin/control CLI has 4 authentication levels: None (0), Operator (1), Admin (2) and Vendor (3). When initially connecting, the connection starts out in level 2 (admin), althouh this can be changed using config option `net_admin_cli_initial_auth_level` (valid range is 0 to 2). See command `auth` below for details.

[1] The FlexDDS-NG had this USB exposed directly on the slot module's front panel via a USB-B connector.

The following sections document a subset of the admin/control commands supported by the FlexDDS-HD.

2.7.1 Admin CLI: General Concepts

Argument parsing: The admin/control CLI is a text based command line interface similar to Unix shells. This means, arguments are split at whitespace. This also means that arguments that require assignments such as `mux=17` must not have spaces around the '='.

Slot spec: Many commands accept a "slot spec", usually denoted as *NNN*, specifying which of the 24 slots the command should act on. The slot spec is similar to page ranges when printing documents with some additional special values. Here are some examples:

`present`	all populated slots (this is usually the default)
`all`	all 24 slots
`none`	none of the slots
`0,2,17`	slots 0, 2 and 17 (note again that there must not be any spaces)
`9-12,3,23`	slots 9, 10, 11, 12 as well as 3 and 23

Slot numbering starts with 0 on the left side of the rack, then slot 11 is the one left of the main control module, then comes slot 12 just right of the main control module up to number 23 on the far right of the rack.

2.7.2 `help`: Obtaining Command Usage Information

`help` [*command...*]

Typing `help` lists available commands (also those not supported by the current authentication level).

If one or more commands are specified as arguments, it will display a more detailed help for those specific commands.

2.7.3 `auth`: Authentication

`auth` [*level*]

If no argument is give, print the current authentication level.

Otherwise, try to authenticate for the requested level (operator/admin/vendor).

The passwords for the authentication levels operator and admin are specified in config options `net_password_operat` and `net_password_admin`.

If you prefer not to use any passwords, the best option is to set the option `net_admin_cli_initial_auth_level=2` so that the connection starts out in the 'admin' level (default).

2.7.4 `health`: Display Health Information

`health` [`pwr`] [`hw`] [`tps`] [`slot`] [`all`] [`watch`]

Without arguments, display all available health information.

With arguments, only specific health information is written. This is useful together with the `watch` option, which will print the health values every second until you press 'q'.

The FlexDDS-HD continuously performs and checks many health measurements and the output for a healthy FlexDDS-HD should look similar to the following.

The first block shows the voltage rails on the main control module as well as the power good indicators on the main module and the backplane:

```
PWR:       2197 status messages, last one      0.1 s ago       (OK)
PWR: status: SOM_RUNNING  (11)   slots: AV+ENBL      LED: GREEN
PWR: 12 V:   12.434 V  (12.20 ... 12.80)  [------|---------]   (OK)
PWR: 5.0V:    5.006 V  ( 4.85 ...  5.25)  [------|---------]   (OK)
PWR: 3.3V:    3.386 V  ( 3.20 ...  3.45)  [------------|---]   (OK)
PWR: 2.5V:    2.520 V  ( 2.40 ...  2.60)  [----------|-----]   (OK)
PWR: Temp:   37.7   C  (      ... 75.0 )  Fan setpoint:  80%   (OK)
PWR:      PGOOD: BP_3V3. . : good  |  PSU: good  |  SOM: good  (OK)
MAIN:     PGOOD: 2V5,3V3,5V: good  |  1V8: good  |            (OK)
BP:       PGOOD: 2V5_ANA . : good  |  3V0: good  |            (OK)
BP/SLOTS: PGOOD: 5V5 . . . : good  |  3V8: good  |  2V5: good  (OK)
```

The "SOM" block displays the monitored voltages and the temperature of the computer module on the main control module:

```
SOM: 0_vccint  :  0.98 V ( 0.95 -  1.05)  [-----|----------]   (OK)
SOM: 1_vccaux  :  1.79 V ( 1.75 -  1.85)  [-----|----------]   (OK)
SOM: 2_vccbram :  0.98 V ( 0.95 -  1.05)  [-----|----------]   (OK)
SOM: 3_vccpint :  0.98 V ( 0.95 -  1.05)  [-----|----------]   (OK)
SOM: 4_vccpaux :  1.79 V ( 1.75 -  1.85)  [-----|----------]   (OK)
SOM: 5_vccoddr :  1.35 V ( 1.30 -  1.40)  [------|---------]   (OK)
SOM: 6_vrefp   :  1.25 V ( 1.20 -  1.30)  [------|---------]   (OK)
SOM: 7_vrefn   : -0.00 V (-0.05 -  0.05)  [--------|-------]   (OK)
SOM: 8_vpvn    :  0.61 V (-0.10 -  1.20)  [---------|------]   (OK)
SOM: Temp:   58.3 C  (W: 75.0, SD: 85.0)  [##########      ]   (OK)
```

The next block displays the temperatue of the ULN (ultra low noise) oscillator (if installed), the monitored fan speed as well as information for the three rack voltage rails:

```
ULN: Temp:   60.9 C  (W: 70.0, SD: 75.0)  [############    ]   (OK)
Rack: Fan speed:  58,   0,  59,   0   (min:  50   0  50   0)   (OK)
TPS_2V5: Vin  :  12.55 V ( 12.2 -  12.8)  [---------|------]   (OK)
TPS_2V5: Iout :   0.56 A (  0.0 -  30.0)  [                ]   (OK)
TPS_2V5: Vout :   2.54 V (  2.4 -   2.7)  [---------|------]   (OK)
TPS_2V5: Temp :  32.8  C (      -  80  )  [###             ]   (OK)
TPS_2V5: Flags: ON PGOOD                                      (OK)
TPS_3V8: Vin  :  12.53 V ( 12.2 -  12.8)  [---------|------]   (OK)
TPS_3V8: Iout :   1.04 A (  0.0 -  30.0)  [#               ]   (OK)
TPS_3V8: Vout :   3.84 V (  3.7 -   4.0)  [----------|-----]   (OK)
TPS_3V8: Temp :  33.5  C (      -  80  )  [####            ]   (OK)
TPS_3V8: Flags: ON PGOOD                                      (OK)
TPS_5V5: Vin  :  12.55 V ( 12.2 -  12.8)  [---------|------]   (OK)
TPS_5V5: Iout :   0.27 A (  0.0 -  30.0)  [                ]   (OK)
TPS_5V5: Vout :   5.53 V (  5.4 -   5.7)  [----------|-----]   (OK)
TPS_5V5: Temp :  34.3  C (      -  80  )  [####            ]   (OK)
TPS_5V5: Flags: ON PGOOD                                      (OK)
```

```
TOT: 1 + 3 + 1 = 6 W (3.1 per slot); est. wall power 22 W
```

Finally, the health monitoring for each installed RF genrator slot is listed. In this example, only slots 6 and 7 are installed. The `TEMP` column is the temperature measured on the slot module, all the other 4-digit figures are different voltage rails in mV.

```
------ Health: TEMP VDDA RF_AMP_5V DDSANA1V8 DDSDAC3V3 CORE DDIO FPGA LMK3 G Time ERR ____
S[06]: Health:   24 3327 4974 4996 1849 1851 3309 3307 1848 3318 1211 3335 1  401 000 (OK)
S[07]: Health:   33 3329 4996 5000 1853 1840 3320 3304 1849 3320 1216 3337 1   38 000 (OK)
Check health: 0,0,0(0)
```

The last line (check health) lists the number of errors detected in the health check. Seeing only zeros means no errors found.

2.7.5 `dio`: Configure Digital IOs on the Rack

`dio A|B [mux=`*MUX*`] [invert=`*INV*`]`

Configure the two digital BNC IO ports on the rack control module labeled “BNC A” and “BNC B”.

If *INV* is set to 1, the signal is inverted; the default is 0.

The possible choices for *MUX* are listed in Table 2.3. Either the numeric value or the symbolic name can be used (case-insensitive). Note that there are no spaces around the ‘=’.

The predominant use of those BNC ports will be to serve as inputs for the rack trigger lines. All the other (output) choices are mostly for debugging variuos aspects.

The initial defaults are specified with config option `ms_dio_mux` and default to input. You can use `dio print` to display the current settings.

2.7.6 `bptrig`: Configure Backplane Trigger Bus

`bptrig A|B|RA|RB|LA|LB [mux=`*MUX*`] [invert=`*INV*`]`

This command configures the rack backplane trigger bus from the main control module to all the installed slots.

On the backplane, there are 2 trigger lines (LA, LB) that connect the main control module with all slots to the left of it (slots 0…11) and 2 more trigger lines (RA, RB) that go to the slots on the right side (slots 12…23).

The first argument specifies which of the 4 lines to configure; the choices ‘A’ and ‘B’ are shortcuts for the respective bus line on the left and on the right.

Each of these 4 lines can be configured individually using command `bptrig`. Table 2.4 lists the available choices for *MUX*. Either the numeric value or the symbolic name can be used (case-insensitive). Note that there are no spaces around the ‘=’.

You can use `bptrig print` to display the current settings.

The initial is set in the config option `ms_bp_trig_mux` and defaults to BNC A rising edge event being routed to the trigger lines RA and LA and the BNC B rising edge event being connected to the trigger

Num	Name	Description
-1	input	the BNC port serves as an input
0	LOW	logic LOW value
1	HIGH	logic HIGH value
2	TRIG_SWITCH	monitor of the red trigger switch (HIGH if pressed)
3	CM_SYNC_ASYNC	Generated rack SYNC signal (before)
4	CM_SYNC_PULSE	Generated rack SYNC signal (pulsed)
5	CM_LD1	First stage PLL lock detect (HIGH if locked, LOW if unused)
6	CM_LD2	First stage PLL lock detect (HIGH if locked, LOW if unused)
7	LMK_SYNC_CLK	SYNC signal emitted by the clock generator
8	LVDS_A0	Monitor of LVDS A input, differential pair 0
9	LVDS_A1	Monitor of LVDS A input, differential pair 1
10	LVDS_A2	ditto
11	LVDS_B0	Monitor of LVDS B input, differential pair 0
12	LVDS_B1	ditto
13	LVDS_B2	ditto
63	CLK	125 MHz internal update clock

Table 2.3: Main control module BNC DIO multiplexer choices. The names are case-insensitive. Usually, the DIOs will be used as trigger inputs in a production environment. Most other choices are for debugging various aspects.

lines RB, LB.

2.7.7 ddsman: Reset/Restart DDS and DCP

`ddsman` [*NNN*] [C*NN*] [*operation...*]

DDS management commands.

NNN is the slot spec (see chapter 2.7.1) and defaults to all installed slots.

C*NN* specifies which channels on the slots to act on and defaults to all channels (i.e. C0123). For just channel 0 or 1, use C0 or C1, respectively.

The operations dcp_start, dcp_stop, dcp_reset start, stop, reset the DCP (DDS Command Processor) on the specified slot modules.

With dds_reset, the RF generator channel is completely reset and starts over new. If multiple channels of an AD9910 based RF generator module are affected, the AD9910 synthesizers of these channels are synchronized and phase-aligned.

The following table summarizes the actions of the operations. For start/stop the table denotes the state in which the DCP is after completion of the operation. A "net disconnect" means that DCP network clients connected to the text and 'NFrame' protocol per-slot DCP ports are disconnected and need to re-connect to start over.

Operation	DCP stop	DCP start	FIFO clear	Net disconnect	DDS Synth Reset
dcp_start	-	X (*)	-	-	-
dcp_stop	X	-	-	-	-
dcp_reset	-	X (*)	X	X	-
dds_reset	-	X (*)	X	X	X

Num	Name	Description
0	LOW	static LOW value
1	HIGH	static HIGH value
2	BNC_A_RISE	trigger generated on a rising edge of the rack's BNC A input
3	BNC_A_FALL	same but with falling edge
4	BNC_A_SYNC	a synchronized version of the BNC A on the rack control module
5	BNC_B_RISE	same as 3-5 for BNC B
6	BNC_B_FALL	
7	BNC_B_SYNC	
8-10	LVDS_A0...2	synchronized version of the LVDS A port, differential pairs 0...2
11-13	LVDS_B0...2	same for the LVDS B port
14	TRIG_SWITCH	trigger generated when pressing the red trigger pushbutton
16-18	LVDS_A0...2_RISE	rising edge trigger on LVDS port A, differential pair 0...2
19-21	LVDS_B0...2_RISE	same for LVDS port B
22-24	LVDS_A0...2_FALL	falling edge trigger on LVDS port A, differential pair 0...2
25-27	LVDS_B0...2_FALL	same for LVDS port B

Table 2.4: Main control module backplane trigger bus multiplexer choices. The names are case-insensitive. There are additional undocumented settings for internal debug purposes.

Operations marked with (*) ensure that if multiple channels in an AD9910-based RF generator module are affected, those channels start synchronized.

Note: Rationale and Synchronization

The rationale for using `ddsman` via the admin/control CLI rather than e.g. resetting the DDS via the DCP network port is that the network port can stall when the FIFO is full and require a FIFO drain before further commands can be sent, effectively preventing the reset. However the admin/control interface will not stall and perform a reset and FIFO clear under all circumstances.
Synchronization: None of these operations performs a synchronization or phase alignment accross multiple modules in a FlexDDS-HD rack.

2.7.8 `dcp`: Interfacing to the DDS Command Processor (DCP)

`dcp` *SLOT_ID* [*0123*] [*command...*]

Send a DCP (DDS command processor) command or instruction to the slot specified in *SLOT_ID* (0...23), channel(s) *0123* (or all if not specified).

The primary way to send lots of commands to the DCP is via the dedicated network ports (see chapter 2.5.1). However, especially for mostly static RF outputs, it may be more convenient to use the admin/control interface.

The DCP commands are described in chapter 4 on page 47.

2.7.9 `dcpsrv`: Manage DCP Server

`dcpsrv` [*NNN*] [*operation...*]

DCP server management operations.

NNN is the slot spec (see chapter 2.7.1) and defaults to all installed slots.

The important operations are: `print` which prints DCP server status information and `disconnect` which forces a network disconnect of the clients connected to the DCP text and NFrame ports (typically 262xx and 263xx).

2.7.10 `slot`: Manage/Reset/Update Slots and Query Status

`slot` [*NNN*] [*operation...*]

This provides a set of slot management operations.

NNN is the slot spec (see chapter 2.7.1).

The most important operations are:

`print` – Print a large amount of status information for each slot. A subset of the information can be obtained via `slot hwinfo` `slot version`, `slot health` or by using e.g. `slot print=dcp`. Call `help slot` for details.

`hwreset` – Perform a hardware reset of the slot module; this will make the slot's microcontroller to reboot and re-initialize everything.

`sm:replug` – Similar but more invasive: Simulate that the specified slot modules were removed and plugged in again. Apart from rebooting the module it also resets the slot manager's logic.

`sm:dfu=AD9910_DUAL` – Perform a firmware update on the RF generator modules. This is especially useful if, for some reason, a slot module firmware update was interrupted or failed and the control module is no longer able to obtain information which type of RF generator module is installed. This operation tells the slot manager to perform a firmware update and assume that the specified type of RF generator slot is installed.

`sm:seqboot=1` – Typcically, for reasons of efficiency, all slot modules are initialized in parallel and timing is dicatated by how quickly the USB connections to the individual modules are enumerated. However, this means that certain modules will typcially boot up successfully without their boot messages being captured. To avoid that, you can use this operation to force sequential booting which takes much longer.

`sm:tty` – This operation allows to inject a line of text into the USB console of the respective slots. Max length 56 bytes. This is especially useful for debugging when one wants to send the same USB command to multiple slot modules. Typically the first thing to do is to send a newline (i.e. just `sm:tty` without additional arguments) to ensure that no garbage is left on the USB console's input.

`sm:echo` – Similar but the text only shows up over the network connection and is not sent via USB to the RF generator slots.

`sm:cmd` – Similar to `sm:tty` but it queues slot USB commands explicitly and is less powerful. E.g. `sm:tty` can be used to send a single 'q' to quit an interactive command while `sm:cmd` just sends commands. Max length 56 bytes.

2.7.11 cfg: Query/Modify Configuration Options

```
cfg [dumpall] [load [FILE]] [option=VAL]
```

Display and modify the configuration options.

Note that modifying the configuration options is *not* meant to be used in production environments but meant for debugging and testing. Many configuration options are interpreted at startup and modifying them will not change the running system.

The primary way to set configuration options is via the FlexDDS configuration file `flexdds_hd.cfg` stored on the micro-SD card in the main control module. This file can also be uploaded via the built-in web server. See chapter 2.10 on page 30 for details.

The main use of this command is to display the active configuration options and to tune certain subsystems.

For instance in order to optimize the clock generator settings, uploading a new configuration each time would be tedious. Instead, all config options starting with `clkgen_` can be modified and then a `lmktune start` can be issued to re-initialize the clock generator with the new settings.

2.7.12 usb: Reset USB Hubs

```
usb [print] [reset=RR]
```

The command `usb reset=all` can be used to reset the USB hubs on the main control module and the backplane. This may occasionally be needed if none of the slot modules successfully launch.

`usb print` prints status information for the USB connection to each slot.

2.7.13 lmktune: Tune the Clock Generator

```
lmktune [print|start] [dac[=N]] [cfg:NAME]
```

`lmktune print` prints the current clock genreator settings; this is mostly useful for phase noise optimization.

With `lmktune dac` you can interactively tune the frequency of the 1 GHz reference clock. This is useful for calibration when not locked to an external clock. The new obtained calibration value can then be written into one of the config options `clkgen_uln_freq_tune` or `clkgen_vcxo_freq_tune`.

`lmktune cfg:NAME` you can load a pre-defined configuration. Currently the following ones exist:

- `ULN10M`: Apply settings for best close-in phase noise with a very low noise 10 MHz external reference clock such as a Wenzel 10 MHz oscillator. This is not suitable for a generic 10 MHz GPDSO such as the ones from Leo Bodnar. To activate you need to also use `start`:
  `lmktune cfg:ULN10M start`

With `lmktune start` you can re-initialize and re-start up the clock generator. This has to be done after changing most settings in order for them to go into effect.

2.7.14 threads: Print CPU Usage and Thread Information

```
threads [print] [watch=MSEC]
```

Using print as argument, this command displays all the process threads by the FlexDDS-HD application that are currently running on the main control slot computer. The displayed CPU usage is cumulative since thread start.

Similarly, with watch=2000, keep printing the CPU usage every 2 seconds. The displayed CPU usage in this case is since the last update. The main purpose of this call is to check CPU usage of individual threads.

2.7.15 version: Print Version Information

```
version
```

Print version information, including the serial number.

A sample output looks like this:

```
FlexDDS-HD version: 1.2c  (c1fb2b3fee0f29570e11bf56707959aaa9eca3cf)
Firmware date . . : Thu Jan  8 22:31:08 CET 2026
Serial. . . . . : R001
Mainboard HW rev. : 2a (20251015)
Mainboard HW addr : 00:0A:35:02:00:03
Application uptime: 00:06:17
System uptime . . : 10:44:43
Copyright (c) 2015--2025 Sigtrona GmbH
Use 'slot version' to display version information of slots.
```

2.7.16 quit: Exit Program, Reboot or Power Down

```
quit [exit|poweroff|reboot]
```

Without arguments, simply terminate the admin/control CLI session; this is the same as closing the connection.

With exit, the application exits (will be restarted by the system), with poweroff the rack powers down safely (like when short pressing the power button) and with reboot the control module reboots.

2.8 Built-in Web Server (TCP Port 80)

The FlexDDS-HD has a built-in web server on TCP port 80 (plain HTTP; no SSL/HTTPS support). In order to connect, direct your browser to `http://<Laser_IP>/` and be sure to explicitly use `http` and not `https`.

The web server allows to perform the following tasks:

- Perform a firmware update of the rack (and all slots)

- View the rack configuration file `flexdds_hd.cfg` and upload a new config file
- View the factory configuration file
- View the log files generated by the FlexDDS-HD application, including standard application logs, firmware update logs and crash core dumps
- View device health information

Most pages of the web server are protected using a username/password combination. When prompted by the browser, enter user name `admin` and the password `!FlexDDS-HD!`.

2.8.1 View and Edit the Rack Configuration File

The web server allows you to edit the configuration file `flexdds_hd.cfg` in a browser and upload new settings by clicking on the box "View/Edit User Config" on the initial deck when directing your browser to the laser's IP address.

This is a convenient alternative to removing the micro-SD card and editing the config file there, because it does not require to switch off the FlexDDS-HD.

After having uploaded a new config file, you need to restart the FlexDDS-HD rack application. This can be done via the command `quit restart` from the admin/control interface or by pushing the respective button next to the upload button on the configuration editing web page.

The FlexDDS-HD is shipped with a default user configuration file. There is no way to revert back to the original as-shipped file, but if you leave away or comment out a setting, the built-in default will be used.

2.8.2 View Application Log Files

The main control module keeps application log files including startup self-diagnosis and all sorts of important events. These log files are kept in the third partition of the SD-card which has an `ext4` type Linux file system for reasons of robustness. These log files can be accessed via the built-in web server or by inserting the micro-SD card into any Linux computer or by means of an Ext4 plugin for Windows/MacOS.

To access the application log files, either click on the links or directly go to: `http://<Laser_IP>/log/flexdds/` There are 3 types of application log files:

- `HDW.log` contains hardware related messages
- `NET.log` for network related messages
- `SYS.log` for system / other messages

For each type of log file, 10 versions are kept:

- `HDW.log` contains the log messages of the currently active session.
- `HDW-0.log` contains the logs from the previous application start.
- `HDW-`n`.log` with n up to 9 archives the messages from the $(n+1)$-th last application start.

2.8.3 View Slot Firmware Update (DFU) Log Files

By default, the protocol of the last firmware update on RF generator modules is saved on the SD card and can be obtained via the web server by surfing to `http://<Laser_IP>/log/dfu/`

The files `slot_`nn`_dfu.log` represents the most recent log of a firmware update on slot nn $(0 \ldots 23)$.

2.8.4 Obtaining Crash Core Files

In case the FlexDDS-HD application crashes or terminates abnormally, a core file is generated and saved on the micro-SD card. It can be retrieved by surfing to `http://<Laser_IP>/log/core/` The most recent file is called `flexdds_server.core.latest.gz`

If you experience crashes, especially reproducible ones, please contact Sigtrona. In debugging/fixing the firmware, it is helpful to send along log files, core dump file (if available) and the user config file.

2.9 Phase Synchronization Accross Multiple Slots

The FlexDDS-HD is capable of generating radio frequency signals with a precise and known phase relationship between multiple RF generator slots.

Concerning phase synchronization, it is important to understand that a known phase relationship can only be established if all slots to be synchronized are triggered by one and the same the backplane trigger bus line (RA, RB, LA, LB). This means an external signal that is received via one of the BNC A or BNC B or LVDS A/B ports on the main control module is mux'ed to the respective backplane trigger bus (see command bptrig).

Here is an example how to test phase synchronization between multiple slots.

First, using the admin/control interface, ensure that the red trigger pushbutton on the control module is routed to the backplane trigger bus line A:

```
flexdds> bptrig A mux=TRIG_SWITCH
```

For each slot to be synchronized, send the following text commands to DCP (text protocol) network port (after sending the auth frame if required):

`dcp X spi:CFR1=0x402000`	set auto-clear phase on update (bit 13) (X is 0, 1 or empty)
`dcp X spi:CFR2=0x1000080`	set single tone ASF bit and matched latency
`dcp X spi:stp0=0x3fff000020000000`	full amplitude 125 MHz output
`dcp wait::BP_TRIG_A:u!`	wait for backplane trigger A, then update

The channel spec X would be 0 or 1 if you would like to sync RF channel 0 or 1 (respectively) on the respective slot module. Leave away X to have both channels of the module synchronized.

After having sent those commands, perform a short press on the red trigger pushbutton on the control module. You should see the red trigger LEDs flash for each output on each slot module you would like to synchronize. The output waveforms of all those outputs should now emit a 125 MHz sine wave with known and constant phase relationship between each other.

How this works: The update trigger is routed accross the backplane to all slots and channels selected and arrives synchronously. If you wire digital inputs on slots independently, synchronization cannot be guaranteed. It is important to set the auto-clear phase accumulator bit (bit 13) in the CFR1 register. This ensures that all sine waveforms are reset to the same phase by the update event. The general concept is the same for more complex waveforms: Make sure all channels to be synchronized have the auto-clear phase bit (CFR1 bit 13) set and that they wait for an external backplane trigger (A or B) and then send that trigger signal.

Here is a more fun example that first generates a moving phase and then synchronizes the channels:

`dcp X spi:CFR1=0x402000`	set auto-clear phase on update (bit 13) (X is 0, 1 or empty)
`dcp X spi:CFR2=0x1000080`	set single tone ASF bit and matched latency
`dcp X spi:stp0=0x3fff00002000000N`	slightly different freq for each slot (use random digits for N)
`dcp wait::BP_TRIG_A:u`	wait for backplane trigger A, then update
`dcp X spi:stp0=0x3fff000020000000`	slightly different freq for each slot
`dcp wait::BP_TRIG_A:u!`	wait for backplane trigger A, then update

In this example, after the first trigger signal you will observe outputs at about 125 MHz with slightly different frequency for each slot. After the second trigger, all frequencies are set to the same 125 MHz and the phase accumulators are reset to achieve phase synchronization.

2.10 Rack Configuration Options

The FlexDDS-HD has a lot of configuration options that cover many aspects ranging from factory calibration over passwords to default power-up settings.

Many of these options can be configured by editing the file `flexdds_hd.cfg` on the micro-SD of the main rack control module.

There are multiple ways to do that:

- You can use the built-in web server to edit the user configuration file in the broweser. After editing, you need to restart the FlexDDS-HD application, e.g. by issuing the command `quit exit` on the admin/control CLI or by pressing the restart button in the web interface.
- You can switch off the FlexDDS-HD and remove the SD card to edit the file directly. Follow the same 6 steps as explained on page 14 for changing the IP address, just edit a different file.

During startup of the application, all configuration options are first initialized with build-in default values. Then, the factory configuration file is read (this file can be viewed using the built-in web server). Finally, the user configuration file `flexdds_hd.cfg` on the micro-SD is read.

The default user configuration file does not list all available configuration options. So you can add new entries to the file if the need arieses. Note however, that the user configuration file is not allowed to change certain factory/vendor settings and attempts to do so will be rejected. Note also that care has to be taken when editing as changing certain settings can make the device malfunction.

Errors in the user config file are reported in the log but the application will start anyway. You can inspect the log and use the `cfg` command in the admin/control CLI to check if values were parsed correctly.

2.10.1 Config File Format (`flexdds_hd.cfg`)

When editing the configuration file it is important to understand the file format and general concepts:

- The configuration file is a simple text file. Only edit with a text editor like Emacs, Notepad, Notepad++, Vim, VisualStudioCode and similar. Do not use office programs like Word or LibreOfficeWriter for editing.
- The file is fundamentally line-based. One config option occupies exactly one line.
- Lines starting with `#` are treated as comments and ignored. Empty lines are also ignored.

- Each config option has the form
  `name = VALUE`
  Spaces before/after the assignment as well as at the begining or end of the line are ignored.
- Each config option has a fixed type. Supported types are:
  int32, uint32, int64, uint64, double, char
- Certain options are vectors of pre-determined length. The value of a vector consists of *space* separated entries (*not* comma separated):
  `ms_bp_trig_mux = 2 5 2 5`
- Integer values can be written in decimal (`177`), hexadecimal (`0xb1`) or binary (`0xb10110001`). Strings are character vectors and must be enclosed in double quotes:
  `net_password_operator = "PaS5w0rd"`
- The file is read top-to-bottom and when assigning options multiple times, later entries overwrite the values from previous entries.
- Some options can only be set by the factory and cannot be overwritten in the user configuration file.
- This is a plain ASCII file, it does not have UTF-8 or any other multi-character encoding.

The following sections provide a non-exhaustive list of available configuration options.

The values listed next to each config option is their default; for some options the default is set during factory calibration and then only a generic range is given.

2.10.2 Clock Generator Configuration Options

`clkgen_refclk_mode` = **-1** (int32)

Master mode switch for the clock generator.

-1 automatically choose mode 0 or 2 if the ULN option is installed
0 (`IVCO_DUAL_PLL`) Use non-ULN oscillator.
1 (`ULN_DIST`) Use ULN oscillator free-running without external reference (requires ULN option)
2 (`ULN_PLL1`) Use ULN oscillator and permit locking to external reference if an external reference is detected (requires ULN option)
3 (`ULN_DUAL_PLL`) Testing purposes only (requires ULN option)
4 (`IVCO_PLL2`) Testing purposes only

`clkgen_have_uln_1ghz` = **0/1** (int32; factory only)

Whether the ULN option is physically installed.

`clkgen_ext_refclk_Hz` = **10 000 000** (int32)

Frequency in Hz of the external reference clock input. Choose such that 1 GHz is a multiple of this value. Values below 100 kHz are not supported.

`clkgen_uln_freq_tune` = **0...1023** (int32; calibration)

DAC value to fine-tune the frequency of the ULN oscillator (if installed).

`clkgen_vcxo_freq_tune` = **0...1023** (int32; calibration)

DAC value to fine-tune the frequency of the non-ULN oscillator (if installed).

`clkgen_pll1_uln_pdf_Hz` = **100 000** (int32)

Phase detector frequency in Hz of the ULN PLL. Must be a fraction of the reference clock frequency. Typical values will be 100 kHz to 10 MHz.

`clkgen_pll1_uln_cp_gain` = **0** (int32)

Charge pump current for the ULN PLL, range 0...15; this is used to tune the PLL bandwidth, lower values mean lower bandwidht.

`clkgen_fb_clk_freq_kHz` = **-1** (int32)

Feedback clock frequency for the 1 GHz PLL in kHz. A value of -1 automatically chooses between 1 MHz and 10 MHz. You may want to tune this value for low phase detector frequencies to work around a limited divider range in the PLL.

`clkgen_refclk_out_freq_MHz` = **-1** (int32)

Frequency in MHz of at the reference clock SMA output labeled "Refclk Out" on the main control module frontpanel. Must be an integer fraction of 1 GHz. A value of -1 disables the output.

`clkgen_ext_1ghz_out_enable` = **0** (int32)

Whether to enable (1) or disable (0) the 1 GHz reference clock output available on the main control module frontpanel next to the "Trig" pushbutton.

`clkgen_pll1_wnd_size` = **-1** (int32)

Window size of the digital lock detector. Lower phase detector frequencies typically need a larger window, otherwise a lock will never be detected. Value range is 0...3, the default of -1 chooses one automatically but that heuristic is quite limited.

`clkgen_pll1_dld_cnt` = **2000** (int32)

Digital lock detect counter: The phase detector must be within the `clkgen_pll1_wnd_size` for this many cycles for the digital lock detect. Value range is 0...16383.

The following set of config values has been found to provide the lowest close-in phase noise when locking the ULN 1 GHz oscillator to an external 10 MHz oscillator with very low phase noise (e.g. Wenzel):

```
clkgen_pll1_uln_cp_gain = 15
clkgen_pll1_uln_pdf_Hz = 1000000
clkgen_fb_clk_freq_kHz = 10000
clkgen_pll1_wnd_size = 2
```

The default value provide a reasonable result for reference clock sources with higher phase noise such as simple GPS disciplined oscillators.

2.10.3 General Rack Configuration Options

`ms_dio_mux` = **-1 -1** (int32[2])

Initial (default) value for the BNC A and BNC B digital IOs on the main control module. Supported values are listed in Table 2.3 on page 23.

`ms_bp_trig_mux` = **2 5 2 5** (int32[4])

Initial (default) value for the 4 backplane trigger lines, specified in the order LA, LB, RA, RB. Possible choices are listed in Table 2.4 on page 24.

`ms_trig_long_press_action` = **1** (int32)

Action to take on long press of the red "Trig" pushbutton on the main control module:

0 do nothing

1 reset all slots, discard all data stored in DCP FIFOs and disconnect all DCP network clients

2 restart the FlexDDS-HD server application; this will terminate all network connections and force a reboot of all RF generator slots

`hw_reset_usb_hubs` = **0** (int32)

Whether to reset the USB hubs inside the rack upon application start. Values are 0 (no), 1 (only reset the backplane hubs), 2 (reset all hubs). If you experience unsuccessful startups where many slots are not recognized, resetting the USB hubs might solve it. However, it prolongs the startup by a few seconds.

2.10.4 Network Configuration Options

`net_console_tty_port0` = **26100** (int32)

The RF generator modules provide a USB console which is exposed to the network by the main control module (see section 2.6). Each slot has one network port. The default of 26100 will set the TCP port range 26100...26123 for the slots 0...23. A value of -1 disables the port.

`net_dcp_text_port0` = **26200** (int32)

Starting TCP network port for the DCP text mode protocol (see section 2.5.1). The default of 26200 will set the TCP port range 26200...26223 for the slots 0...23. A value of -1 disables the port.

`net_dcp_nframe_port0` = **26300** (int32)

Starting TCP network port for the DCP binary 'NFrame' protocol (see section 2.5.2). The default of 26300 will set the TCP port range 26300...26323 for the slots 0...23. A value of -1 disables the port.

`net_admin_cli_port` = **27000** (int32)

Network TCP port for the interactive admin command line interface (CLI). There is only one interface for the rack, not one per slot. A value of -1 disables the port.

`net_control_cli_port` = **27001** (int32)

Network TCP port for the non-interactive control command line interface. There is only one interface for the rack, not one per slot. A value of -1 disables the port.

`net_admin_cli_initial_auth_level` = **2** (int32)

Initial authentication level for new connections to the admin and to the control CLI ports. Possible choices are 0 (none), 1 (operator), 2 (admin). The default of 2 allows connections to perform all admin tasks without prior password authentication which is quite insecure. If a lower auth level is specified, new connections must first use the `auth` CLI command to authenticate e.g. as admin.

`net_tcp_nodelay` = **1** (int32)

This refers to the DCP command ports. If non-zero, this option disables the TCP Nagle algorithm. This may produce faster responses at the cost of higher fragmentation and potentially lower overall throughput (assuming some sort of pipelining or command batching is in place and the client does not wait for an "OK" response to each command sent before sending the next one).

`net_dcp_text_auth_token` = **0x31546049441b0d00** (uint64)

Authentication token for DCP text protocol connections. This is essentially the first line to be sent over the DCP text mode connection and serves as a sort of password. Note that the last 2 digits are the slot number, so they must be specified as "00" here. Be sure to use hex notation (starting with 0x) and use exactly 16 digits. A value of 0 disables the use of an auth token, so none will be required.

`net_dcp_nframe_auth_token` = **0x31546049441b0d00** (uint64)

See the previous option. This is the same thing for the binary 'NFrame' protocol.

`net_password_operator` = **"Eroh0gei"** (char[32])

Password for the operator user. Will be asked by the `auth` command of the admin/control CLI when attempting to authenticate as operator. The first character of the string must be a letter ('a-z' or 'A-Z'). The max length is 31 characters. Do not use UTF-8 special characters.

`net_password_admin` = **"giRae7Di"** (char[32])

Password for the admin user. See the previous option for further details.

`usb_console_tty_s2m_buffer_size` = **32768** (int32)

All the console messages sent by the RF genreator slots to the main control module via the USB console are captured by the control module CPU in a ring buffer. This is the size of that buffer (one for each slot). Even if you connect to the TCP port for the USB console after the slot module has booted, the boot messages are preserved up to this size. The USB console has no "back pressure" on the slots modules, so if data is not read quickly enough via the network and more than the specified size accumulates, messages will be discarded.

`usb_console_tty_s2m_buffer_size` = **8192** (int32)

See the previous config option. This is the same thing, just for the opposite data flow direction, i.e. from the main control module to the RF generator modules.

2.11 Rack Temperature/Health Monitoring

The FlexDDS-HD performs internal health and especially temperature monitoring and will attempt to shut itself off before overheating. If you find a device switched off and suspect a emergency shutdown, please analyze the log files from the previous start which can be obtained via the web server (see chapter 2.8.2). Especially look at the last messages in `HDW-0.log` and `HDW-1.log`.

2.12 Firmware Update

In the FlexDDS-HD, firmware is present inside the main control module as well as inside each RF genrator module. It is important that the firmware versions of all components match. The main control module automatically installs the appropriate firmware version on each RF generator module upon startup or when the module is hot-plugged.

The newest firmware is available online at `www.sigtrona.com/flexdds-hd` and is delivered as a file named `flexdds_hd_firmware.zip`

The present firmware version can be obtained either from the main web page of the FlexDDS-HD rack in question or by issuing the command `version` on the admin/control interface.

2.12.1 Firmware Update Instructions: Web Interface

This is the most convenient way to update the firmware: Load to the FlexDDS-HD built-in web page in a browser by surfing to `http://<Laser_IP>` (do not use `https`!), click the firmware update card on the deck, select the file `flexdds_hd_firmware.zip` (do not unpack it), press upload and wait for the device to restart.

During the firmware update, the log is written on the web page showing progress. Do not restart or switch off during the update. The device will re-start automatically. The first reboot might take longer, because it may install new firmware on the RF generator modules.

While the firmware update is in progress, the power LED will blink yellow/green alternatingly.

2.12.2 Firmware Update Instructions: SD Card

This method can be used to recover a failed update via the web server or if the device does not boot properly.

Follow these steps:

1. Power down the FlexDDS-HD.
2. Remove the micro-SD card from the main control moduly by gently pressing in and letting the card come out.
3. Open the micro-SD card in any Windows/Linux/Mac computer. It has 3 partitions, 2 of them have a FAT filesystem and will be visible with Windows/Mac. Locate the partition which has one file

named `PLACE_FIRMWARE_UPDATE_HERE`. Unpack the `flexdds_hd_firmware.zip` firmware archive onto that partition. It is important that all files are placed directly into the partition, not in any subfolder.

4. Safely remove the micro-SD card from the computer and re-insert into the FlexDDS-HD. The micro-SD contacts face towards the text "Micro SD".
5. Power up the FlexDDS-HD and wait. There will be no indication on the power button of a firmware update in progress. You can watch the progress on the USB/serial console of the main control module.

2.12.3 Recovering a Failed Firmware Update

If the main control module does no longer boot after a firmware update, you can install a new/different/old firmware using the SD card, see the previous section.

If the main control module boots up but does no longer recognize the RF generator modules,[2] it may help to manually update the latter. To do so, open a connection to the admin/control CLI (see section 2.7 on page 19) and enter the following commands:

```
slot p=
slot sm:dfu=AD9910_DUAL
```

The first command lists all installed modules and their firmware version if known. Non-respondig modules are listed as present and with unknown version/type. The second line assumes that all populated slots have modules of the type FlexDDS-HD-AD9910 and forces a firmware update on those modules. If only certain modules need to be updated, specify their slot IDs (0...23) as a comma-separated list (without space between):

```
slot 0,4,6-12,14 sm:dfu=AD9910_DUAL
```

The admin/control interface shows no feedback of the firmware update, so refer to the output of `slot p` and `slot dfu_status`. The power LED will blink yellow/green alternatingly during the update. Do not exit the application during the update.

2.13 Calibration

2.13.1 ULN Frequency Calibration 1 GHz

The internal ultra low noise clock (option "ULN-1G") can be calibrated to 1 GHz. This is useful when not locked to an external reference clock. Perform these steps:

1. Disconnect external reference clock.
2. Use either the external 10 MHz output or the 1 GHz output and hook it up to a frequency counter. The frequency counter must be very precise, ideally locked to a GPDSO, Rb reference or other high stability frequency source.
3. Enable the clock output on the FlexDDS-HD. Use the following commands in the admin/control CLI:

```
cfg clkgen_ext_1ghz_out_enable=1 clkgen_refclk_out_freq_MHz=10
lmktune start dac
```

[2] This can happen e.g. when the device is powered down or the application exits or crashes while a RF generator module firmware update was being installed, leaving a partly installed firmware on the modules.

and press the keys '+' and '-' until the frequency counter displays the correct frequency. Press 'q' when done.

4. Note the DAC value from the previous step and assign it to `clkgen_uln_freq_tune` in the user config file `flexdds_hd.cfg` The easiest way to do this is by editing the config via the web interface (see section 2.10).
5. After uploading a new config file, restart the application, either via the restart button on the web interface or by issuing the command `quit restart` on the admin/control CLI.

Do *not* try to set an RF generator output to 10 MHz because you that frequency cannot precisely be represented with a FTW. You *may* use an RF genrator module output tuned to 125 MHz or 62.5 MHz for calibration, though.

Chapter 3

FlexDDS-HD-AD9910 RF Generator Module

The actual RF signal synthesis inside the FlexDDS-HD rack is performed by up to 24 hot-pluggable and largely independent RF genenerator modules such as the FlexDDS-HD-AD9910. All these RF generator modules employ one or more independent phase coherent RF synthesizers attached to a single FPGA which is supported by an on-board microcontroller.

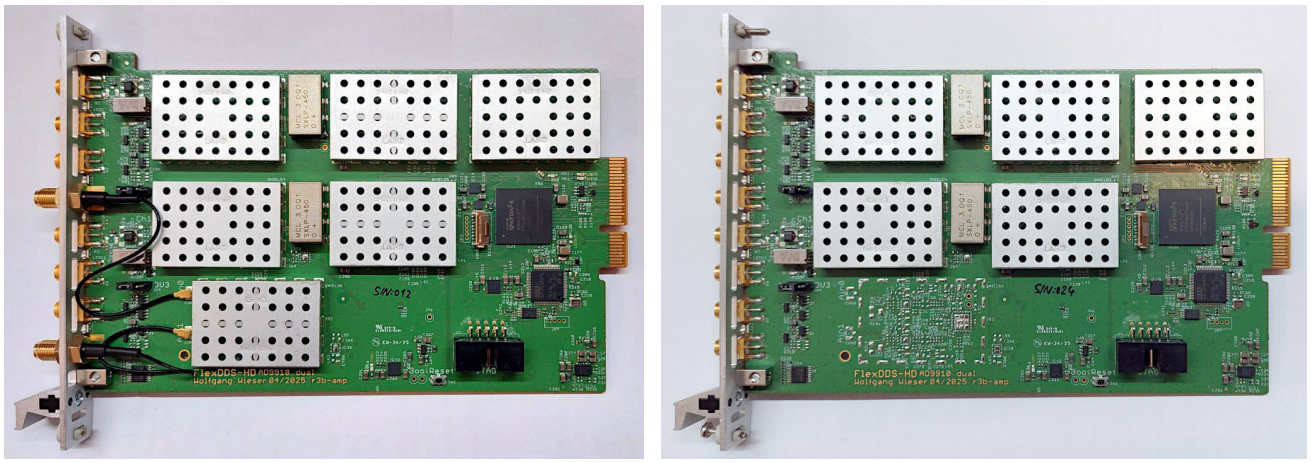


Figure 3.1: FlexDDS-HD-AD9910 RF generator slot modules. Left: Option ADC includes two analog inputs but has only 2 DIOs per RF output channel. Right: Without option ADC there are 3×2 DIOs.

3.1 Functional Overview

In the FlexDDS-HD-AD9910, the actual RF signal synthesis is performed by two Analog Devices AD9910 DDS synthesizer chips clocked at 1 GHz.

Inside the FPGA, each RF channel has one **DDS command processor** (**DCP**). The DCP is responsible for controlling the associated DDS synthesizer chip as well as performing time delays, waiting for events, triggers and generating digital outputs.

DCP instructions can be queued from the USB serial interface via the `dcp` command or can be fed from the FlexDDS-HD Rack via a high speed backplane connection. The rack typcially receives commands via the GBit Ethernet. Typically, a small "program" made of DCP instructions for each RF output channel

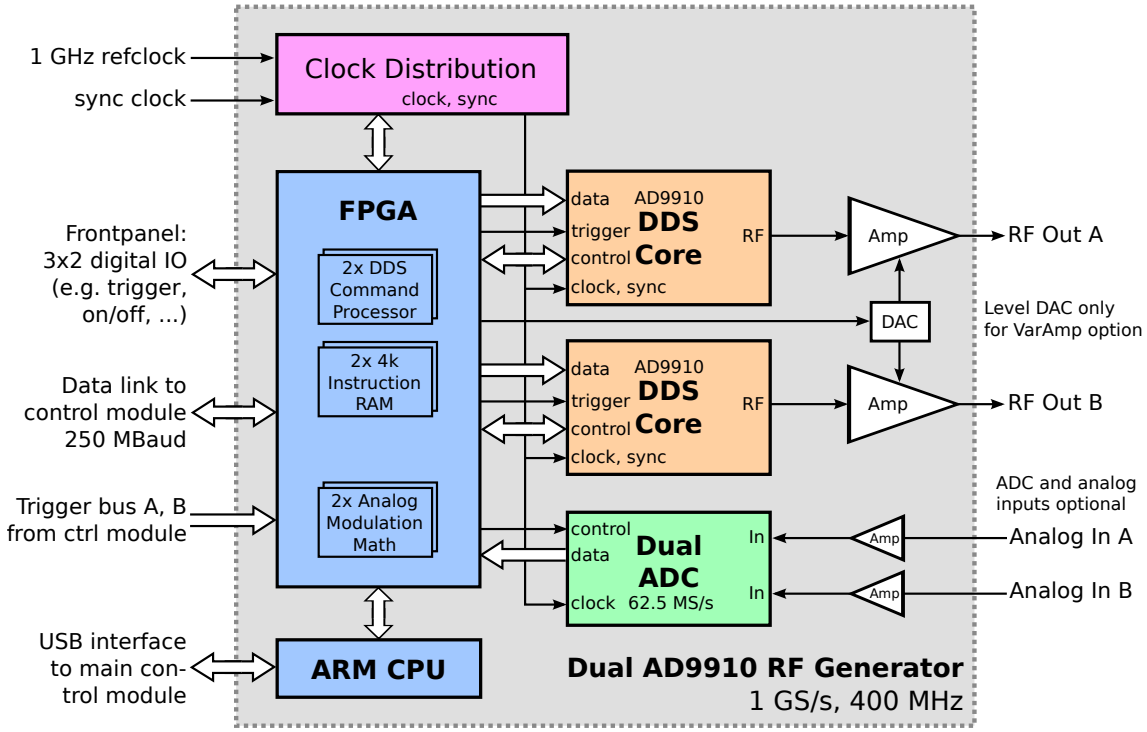
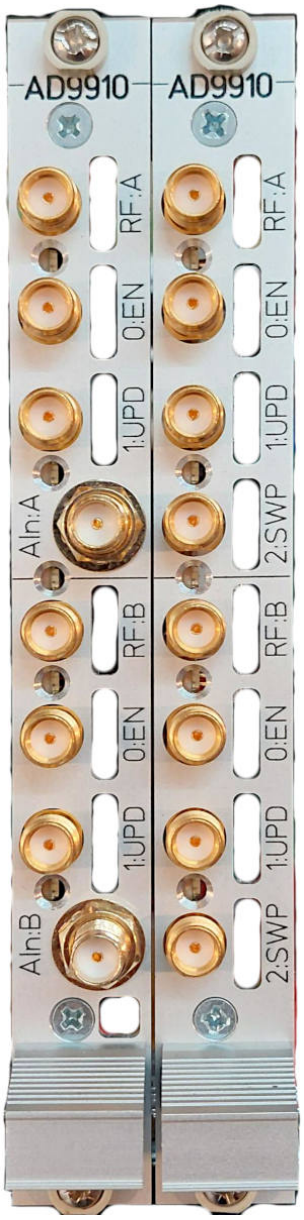


Figure 3.2: Overview of the dual-channel FlexDDS-HD-AD9910 RF generator module.

is downloaded to the FlexDDS-HD and then executed in real time. The program can synchronize the FlexDDS-HD waveform generation with the outside world via events and triggers.

3.2 FlexDDS-HD-AD9910 Frontpanel Elements



FlexDDS-HD-AD9910 RF generator modules: Left/right with/without ADC option. FIXME: Better photo coming soon.

RF:A, RF:B (SMA): Main radio frequency (RF) output channels. The shield of these can be disconnected from ground by means of an RF relay and it is not connected to the frontpanel/metal enclosure.

AIn:A, AIn:B: Two dedicated analog inputs for analog modulation: These inputs are digitally sampled and allow you to perform amplitude/phase/frequency or even polar I/Q modulation at a rate of 62.5 MS/s with 12 to 14 bits resolution. Modulation gain and offset are programmed digitally and can be tuned on the fly without the need to change any analog circuits. Full scale voltage is ±1 V with 50 Ω termination. Only present with the ADC option.

0:EN, 1:UPD, 2:SWP: Up to three digital IOs per RF output channel which can be configured for various functions including fast on/off, triggering, changing sweep direction, interrupting sweeps, quickly switching output profiles or as outputs e.g. to control post amplifiers or get notified of end-of-sweep.

The IO voltage of the digital IO ports is, by default set to 5 V. It is possible to change that voltage to 3.3 V by removing the slot and setting a jumper on the slot to a different position. Please refer to figure 3.3 on how to do this.

FIXME: Add arrows to those:

uC LED: (Center LED) Green when powered up, red when an error on the slot is detected.

RF LEDs A,B: (Next to RF:A,B SMA plug) A red flash indicates trigger operation; green (FIXME: to be documented).

DIO LEDs A,B: (Next to DIO/Ain:A,B) FIXME: to be documented

3.3 The USB Console Interface

The microcontroller on the FlexDDS-HD RF generator modules provides a USB console which provides a command line and logging/status messages. This USB console is internally connected to the main control module and exposed as one network port per slot (see chapter 2.6).

Although not recommended for production use, it can be helpful to connect to this port and issue so-called "USB Commands" or "USB Console Commands", e.g. to display the current status of the AD9910 IO pins and to perform actions like floating/grounding the RF outputs.

The "USB Commands" are regular text commands. For all non-trivial commands, just typing the command name (without any arguments) will print out a short usage description.

3.3.1 USB Command `dcp`

```
dcp ...
```

This is the primary command to control the DDS command processor (DCP) and described in chapter 4.

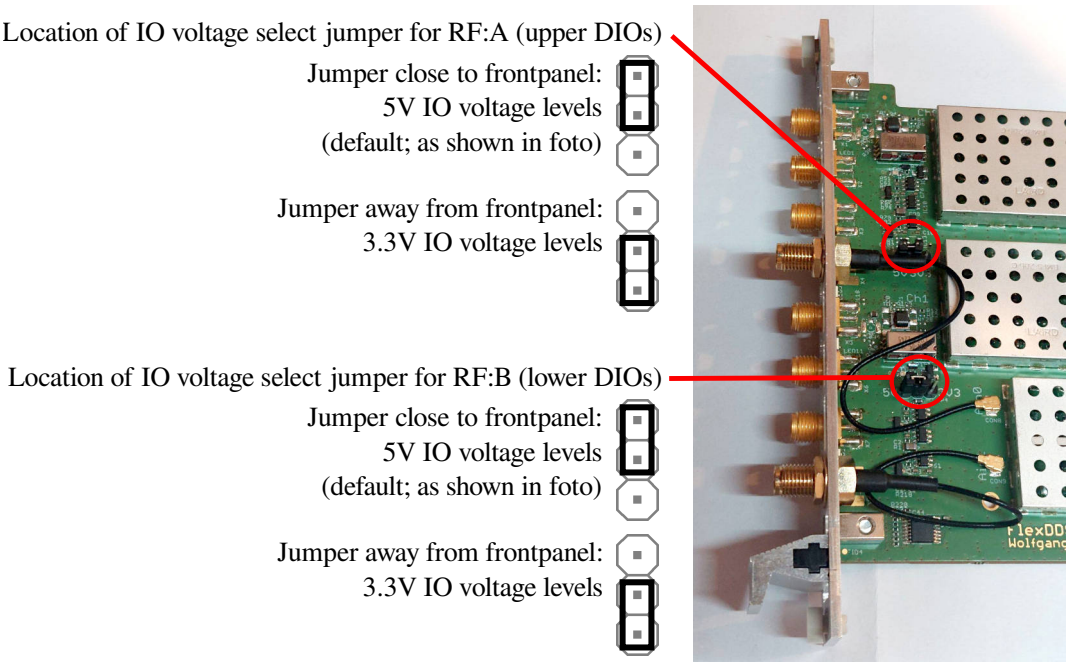


Figure 3.3: IO voltage selection jumpers on the FlexDDS-HD-AD9910 dual channel AD9910 RF generator. Remove the 2 screws holding the slot module and it pull to access the jumpers.

For modules installed in a rack, it is recommended to feed the DCP commands via the rack control module instead of via the USB console.

3.3.2 USB Commands `interactive`, `help`, `reset`, `version`

```
interactive [on|off]
help
reset
version
```

`interactive` can be used to switch interactive mode on or off.

Note: Interactive Mode

The FlexDDS-HD RF generators boot in *interactive mode*. This mode is intended for interactive console sessions. It displays verbose messages and echoes back all typed characters. For remote control software (e.g. via LabView VIs), it is recommended to switch the USB console into non-interactive mode using the command `interactive off`. In non-interactive mode, input is not echoed back and only error messages and query responses are transmitted back.

`help` prints short list of commands.

`reset` hard reset the module and perform a reboot. This is equal to using the admin CLI command `slot NNN hwreset`.

`version` prints version information.

3.3.3 USB Command status

```
status
```

Print various status information. A typical output for the FlexDDS-HD-1GS will look like this:

```
L: STATUS_A=0xe281: BP S0:00 S1:00 CIN0 cin1 HO ld sy
L: STATUS_DDS=0x0101: [0]: SYNC_SMP_ERR pll_lock ram_swp_ovr drover
     [1]: SYNC_SMP_ERR pll_lock ram_swp_ovr drover
L: STATUS_DCP_A0=0x04d0: INST=0, DCP_FIFO:EMPTY,non-full, S2DCP_FIFO:EMPTY,
     SPI_FIFO:EMPTY,non-full, BP_FIFO:EMPTY
L: STATUS_DCP_B0=0x0000: INST fifo: 0 entries, SPI fifo: 0 entries
L: STATUS_DCP_A1=0x04d0: INST=0, DCP_FIFO:EMPTY,non-full, S2DCP_FIFO:EMPTY,
     SPI_FIFO:EMPTY,non-full, BP_FIFO:EMPTY
L: STATUS_DCP_B1=0x0000: INST fifo: 0 entries, SPI fifo: 0 entries
L: STATUS_ISR=0x8000: --- ---
L: CONFIG_DCP=0x0282: run0 EN0 run1 EN1 EN_GLOBAL
```

Long lines have been wrapped for readability. In general, numeric bit fields are also displayed as symbolic names with capital letters if asserted and small letters if deasserted.

3.3.4 USB Command health

```
health
```

Print health information for the RF generator module. A typical output looks like this:

```
FPGA_VCCINT_PG: 1  (OK)
ADC_VREFINT: raw=1490 ==> VDDA=3331 mV  (OK)
MCU_Temp     :   37 C   (raw= 974)  ( -20 ..   60)  [-----------|----]  (OK)
RFAmp0_5V    : 5001 mV  (raw=3075)  (4800 .. 5200)  [--------|-------]  (OK)
RFAmp1_5V    : 4999 mV  (raw=3074)  (4800 .. 5200)  [-------|--------]  (OK)
DDS0_ANA_1V8: 1854 mV  (raw=2280)  (1750 .. 1900)  [-----------|----]  (OK)
DDS1_ANA_1V8: 1842 mV  (raw=2266)  (1750 .. 1900)  [---------|------]  (OK)
DDS0_DAC_3V3: 3321 mV  (raw=2042)  (3200 .. 3450)  [-------|--------]  (OK)
DDS1_DAC_3V3: 3303 mV  (raw=2031)  (3200 .. 3450)  [------|---------]  (OK)
DDS_CORE_1V8: 1849 mV  (raw=2274)  (1750 .. 1900)  [----------|-----]  (OK)
DDS_IO_3V3   : 3322 mV  (raw=2043)  (3200 .. 3450)  [-------|--------]  (OK)
FPGA_INT_1V2: 1216 mV  (raw=1496)  (1100 .. 1300)  [---------|------]  (OK)
LMK_CIN_3V3 : 3339 mV  (raw=2053)  (3200 .. 3450)  [--------|-------]  (OK)
Health problems: 0
```

The first line shows the “power good” indicator for the FPGA core voltage, the `MCU_Temp` is the microcontroller temperature and the follwing lines sho various voltage rails.

3.3.5 USB Command set

```
set [NAME=VALUE]...
```

Set certain variables which control some behavior. Just typing `set` lists all variables and their current

values:

```
USAGE set [variable=value] ...
  loglevel=6                 console log level (panic=0,.. warning=3,.. noise=6)
  interactive=2              interactive mode (0/1/2)
  autoflush_msec=0           auto-flush interval in msec (0 to disable)
  dcp_dump_isn=0             dump instructions as added to DCP memory (0/1)
  dcp_block_msec=-1          block/wait time im msec if FIFO full (-1 for infinite)
  dcp_flush_verbose=0        verbose level for DCP flush (0/1)
  health_check_quiet=0       silence the health check if set
```

loglevel sets the message severity level which will be printed into the console during operation. The lower the value the fewer messages. For automated control scripting, having fewer messages may make it harder to parse the result so using a lower value suppresses messages.

interactive has 3 choices: 2 is the full interactive mode with echo and some line editing (also set by command `interactive on`). 0 is non-interactive "quiet" mode without echo and without prompt set by `interactive off`. The value between, 1, is like the quiet mode but with a shell prompt (like `S[07]:flexdds>`).

autoflush_msec, if non-zero, sets an interval in milliseconds after which DCP commands queued in the microcontroller FIFO are automatically flushed to the DCP on the FPGA. If zero, commands will only be flushed if the FIFO is full or if an explicit flush is requested (`dcp flush` or '!' in DCP commands).

dcp_dump_isn: If non-zero the DCP echos back the raw/hex instruction code of each queued DCP instruction.

dcp_block_msec is the number of milliseconds to wait for for the DCP on the FPGA to make space if the microcontroller FIFO is to be flushed to the FPGA. Usually this should be -1 (infinite) which produces "back-pressure", i.e. the USB console will block and not accept any more commands if the FIFO runs full. However, this also means that the console cannot be used to reset a failed run. So for environments where the controller check if space is available before queuing more commands, this can be set to some positive value. This is similar to using non-blocking IO on file descriptors.

dcp_flush_verbose: If non-zero a log message is written whenever instructions are flushed to the FPGA.

health_check_quiet: If set to 1, no health errors are printed to the console. This is intended for debugging health issues where a new error every second is distracting.

3.3.6 USB Command `eeprom`

`eeprom` [`uc:`*UCNAME*`=`*VALUE*]...
`eeprom wruc`|`reload`

Each RF generator module has an EEPROM storage for calibration, board configuration and user configuration data.

User configuration values can be tuned by setting them via `eeprom uc:`, e.g.: `eeprom uc:rfgain0=65535` (no space around ':' and '=').

Supported values:

- `rfgain0`, `rfgain1`: Represents the analog (DAC) value for tuning the amplification if the variable gain amplifier option was chosen. Valid range 0...65535. The value is silently ignored if the variable gain amplifier is not present.

The command `eeprom wruc` writes the user configuration to the EEPROM.

Using `reload` you can load all the config from the EEPROM again. This will also write the current values if they deviate from the default/unset state.

3.3.7 USB Command `ftest`

```
ftest [CHAN] reset
ftest dio_in|dio_out
ftest [CHAN] aam
ftest [CHAN] f=FREQ[,asf=ASF[iI],pow=POW,auxdac=DAC]
```

This command sets the RF generator slot up for certain functional tests performed during production. It is also very useful if one would simply test the analog inputs or RF outputs in the lab.

`ftest reset` performs a re-initialization of the DDS channel(s); this is equivalent to USB command `dds reset`.

`ftest bnc_in` or `ftest dio_in` (synonym) simply display the logic state of all digital IOs including the backplane trigger lines. Press 'q' to exit.

`ftest bnc_out` or `ftest dio_out` (synonym) toggle all digital IO lines low/high. Press 'q' to exit.

The last line is the most useful one: It configures the DDS output channel for a constant frequency output at *FREQ* Hz. Suffixes 'k' and 'M' are understood but only integer values are allowed. Optionally one can also specify the amplitude via *ASF* in range 0...4095 (default is full amplitude) and the phase offset word via *POW* in range 0...65535 (default: 0). When an 'i' or 'I' is appended to the ASF value, the inverse sinc filter is deactivated or activated, respectively. The *DAC* value can be used to specify the auxiliary DAC on the AD9910 which sets the full scale output current (range 0...255; default: 127 (20 mA))

For example to obtain a 125 MHz signal with 50% full scale amplitude on both RF output channels:

```
ftest f=125M,asf=0x1fff
```

Note that there are no spaces around the '=' and ','.

3.3.8 USB Command `freq2ftw`

```
freq2ftw [FREQ]
```

This is a little helper to convert the frequency *FREQ* in Hz into a frequency tuning word (FTW). Will print both the normal as well as the mirror frequency. The result is given in decimal and in hex.

3.3.9 USB Command dds

```
dds [CHAN] reset **
dds [CHAN] update **
dds [CHAN] status
dds [CHAN] ps
dds [CHAN] readreg:REG **
dds [CHAN] writereg:REG=VALUE **
dds [CHAN] float=-1|1
dds [CHAN] rfgain=VAL
dds [CHAN] rfamp=-1|0|1
```

Note: Direct Synthesizer Access

All those sub-commands marked with ** perform operations on the DDS chip (AD9910) *directly* and will hence interfer with any accesses of the DCP/FPGA at the same time. Therefore, these commands are primarily for testing and debugging. The user has to make sure that the DCP/FPGA is not currently accessing the DDS chip (e.g. AD9910) when executing these commands.

If `dds` is called with no arguments, a short help summary is printed.

CHAN is the DDS channel (0 or 1, both if omitted).

`dds reset` (short: `dds r`) resets and re-starts the DCP and the synthesizer chip. This is equivalent to using `dcp dds_reset` on the network DCP interface of the main control module which is the preferred way of resetting the DDS.

`dds update` (short: `dds u`) sends an IO_UPDATE pulse to the associated DDS synthesizer chip. The preferred way is to do this via the DCP.

`dds ps` prints the digital pin status (LOW/HIGH) of most of the synthesizer chip pins.

A typical output for the AD9910 will look like this:

```
DDS[0]: PDATA=0x0000, F=0, TXEN=1, IO_RESET=1, MASTER_RESET=0
DDS[0]: PROFILE=0, OSK=0, DRCTL=1, DRHOLD=0, RFAMP=1, IO_UPDATE=0
DDS[0]: DROVER=0, RAM_SWP_OVR=0, PLL_LOCK=0, SYNC_SMP_ERR=1
```

`dds readreg` and `writereg` give direct access to the synthesizer chip registers. This is mainly useful to help understand/test what those registers do and read back register contents e.g. to verify the correct operation of DCP instructions.

REG specifies the register address, either as numeric value or as symbolic name (e.g. `CFR2` for the AD9910). Note that there is no space between the ':' and the register name.

`dds readreg` reads the current register value from the AD9910 chip and prints it in hex and in decimal. For the AD9910, this is especially useful together with the `CFR2` bit 16 "read effective FTW" to read back the current frequency of the DDS, i.e. the actual value being in effect. If you need this functionality, it is recommended to initialize `CFR2` with bit 16 set and never clear it. Do not forget that an UPDATE may be required.

`dds writereg` writes the specified *VALUE* into the register *REG*. Note that there is no space around the ':' or '='. Note also that reading back the same register may still give the old value unless an update is performed after the write. The *VALUE* can be specifed in binary, decimal or hex.

`dds float`, if set to '1' opens a HF relay and therby separates the RF output ground from the device ground, keeping only a transformer coupling. This effectively floats the RF output. A value of -1 gives the DCP access to the HF relay. There is no value 0 to force the relay closed.

`dds rfgain` sets the gain/output power level of the variable gain amplifier after the DDS chip. This is *not* available for RF generator modules with a fixed gain amplifier, but only if the variable gain amplifier option was chosen. *VAL* is in range 0...65535 and higher values mean higher power level. The default is 65535.

`dds rfamp` forces on (1) or off (0) the RF amplifier after the DDS synthesizer chip. The default value of -1 gives the DCP the power to en/disable the RF amplifier.

Chapter 4

The DDS Command Processor (DCP)

The FlexDDS-HD contains one DDS Command Processor (DCP) per output channel.

The DCP is implemented in the FPGA on each RF generator module and is responsible for controlling the DDS synthesizer chip as well as performing time delays, waiting for events, triggers and generating digital outputs.

The DCP executes DCP instructions at a rate of (currently) 62.5 MHz (1 GHz/16) with deterministic timing for precise real-time control. Each DCP has a FIFO buffer holding 2 048 instructions.

4.1 DCP Instruction Description

DCP instructions are 48 bits wide. The following table summarizes the instruction format. Bits denoted with '`.`' are "don't care" bits and should be set to 0 to ensure future compatibility. The first 4 bits encode the main instruction selector.

47 ... 40 39 ... 32 31 ... 16 15 ... 0	Name	Description
0000	NOP	No-op (does nothing)
0001 00W0 FEAAAAAA DDDDDDDD DDDDDDDD dddddddd dddddddd	DDS_WRITE	Write to DDS chip
0001 00W1 DDDDDDDD DDDDDDDD DDDDDDDD DDDDDDDD	DDS_WRITE	Long write to DDS chip
0010 AAAA AAAAAAAA DDdddddd dddddddd dddddddd dddddddd	REG_WRITE	Write to DCP register
0011 0URH xXXARRRR RRSSSSSS TTTTTTTT TTTTTTTT TTTTTTTT	WAIT	Wait for event or timeout
0100 0000 ....C CBBAAPPP PHHRROOU	UPDATE	Update

The bit format is described here for completeness and to enable the user to implement his own DCP instruction compiler. It is, however, not necessary to understand the raw instruction format when using the `dcp` command on the USB command line interface as described below.

`NOP`: Instructions starting with 4 zero bits are no-operation instructions. They consume one instruction cycle of execution time and can be used for nanosecond delay purposes. The `wait` instruction should be used for longer delays.

`DDS_WRITE`: Queue a write to the DDS chip. For the AD9910, this works via via the SPI interface. The `AAAAAA` bits specify the 5 bit register address in the AD9910 (only 5 bits used). For a 16 bit register, the data is encoded in the following 16 `DDD...` bits. For a 32 bit register, the data is encoded in the 32 bits `DDD...ddd...`. If the `W` bit is set, the instruction waits until the DDS chip write FIFO is empty and all writes have been carried out (over SPI for the AD9910). The bits `F` and `E` are completion event bits associated with the communication ("SPI") completion event 1 and 0. If set, the respective event is

generated at the time when the register write has been completed.

In order to write a 64 bit register in the AD9910, two successive `DDS_WRITE` instructions have to be carried out: The first one must have bit 40 set to 1 ("long write") and latches the 32 less significant register value bits (`DDD...`). The second `DDS_WRITE` must have the bit 40 cleared and contains the most significant 16 or 32 bits and the register address just like a 16/32 bit write.

`REG_WRITE`: Write to FPGA-internal DCP register. `AAA...` is the 12 bit register address. `DDddd...` encodes the register content which can be up to 32 bits large. Certain registers not only allow to overwrite the old content but also allow to set bits, clear bits or toggle bits. For these registers, the data can be up to 30 bits (`ddd...`) and the first 2 data bits, denoted `DD`, describe the access mode: `11` for toggle, `10` for clear and `01` to set bits.

`WAIT`: Instruction to wait a certain time or for up to 2 events. The 2 events are encoded as 6 bit values `RRRRRR` and `SSSSSS`. If the `A` bit is set ("and"), both events must be present simultaneously to finish the wait, otherwise one of them is sufficient. The timeout is a 24 bit value `TTT...`. The bits `RH` specify the timeout mode: If `00`, no timeout (infinite wait, irrespective of the `TTT...` bits), if `11`, high resolution mode (8 ns per tick), if `10`, normal mode (1.024 μs per tick), if `11`, extended mode (FIXME: Details to come). The `xXX` bits are not used at the moment and must be set to 0.

`UPDATE`: Simultaneously modify the state of several pins. All the bits only peform an action when non-zero. The `U` bit pulses the IO_UPDATE pin to the AD9910. The `OO`, `RR`, `HH` bits modify the OSK, DRCTL and DRHOLD pins into the AD9910. The meaning of the bit pair is as follows: `11` to set HIGH, `10` to set LOW, `01` toggle (and `00` to not change the pin). The `PPPP` bits modify the three PROFILE pins. If set to `1xyz`, PROFILE2 is set to `x`, PROFILE1 is set to `y` and PROFILE0 is set to `z`. If specified as `0011` or `0010`, the profile value is incremented or decremented, respectively. Increment/decrement roll over from 7 to 0 and 0 to 7. The `CCBBAA` pins modify the DIO C, B, and A output from the DCP.

4.2 DCP (Text) Command Line Interface

DCP text commands and instructions can be entered via the DCP (text) network ports on the main control module (preferred way) as well as via the USB console network port and the admin interface. There are subtle difference between those three choices which are described further below together with the various DCP commands.

Simply entering `dcp` will print out a short usage text.

4.2.1 `dcp #`: DCP Raw Command Entry

`dcp` [*CHAN*] `#`*INST*[`!`]

Enters a raw 48-bit DCP instruction. This is intented for higher level software which compiles the desired actions into DCP instructions.

CHAN is the DDS channel (0 or 1, both if omitted) and
INST is the 48 bit instruction in hex notation.

For example (AD9910):

`dcp 0 #0`	channel 0, no operation, just wait one instruction cycle
`dcp 1 #100712345678`	write 0x12345678 into FTW register of channel 1's AD9910
`dcp #400000000001!`	perform IO_UPDATE on both AD9910, flush

Instructions are queued locally on the microcontroller and are not immediately accessible by the DCP. To have them transmitted to the DCP, you need to add the exclamation mark `!` at the end in order to flush the local queue to the DCP FIFO (and no space before it!). It is inefficient to flush each individual instruction, hence when queuing several instructions, it is recommended to flush only on the last one. Flushing occurs automatically when the internal FIFO fills up. Instead of using the exclamation mark (`!`) you can use the command `dcp flush`.

Instead of entering raw DCP instructions, all operations are also available as more convenient commands:

4.2.2 `dcp spi`: Writing Synthesizer Registers via SPI (AD9910)

`dcp` [*CHAN*] `spi:`*REG*`=`*VAL*[`:c`|`w`][`!`]

Write to a register in the AD9910 chip. (Electronically, this is sent over the 4-wire SPI interface at a rate of 62.5 MBaud, hence the name.)

REG denotes the AD9910 register and can be specified either as symbolic name or as register address (0 to 0x16), see Table 4.2 on page 50. Register names are case-insensitive.

VAL is the value to be written into the register. Depending on the register type, this is a 16, 32 or 64 bit value. It can be specified in hex with `0x` prefix or in decimal or in binary with a `0b` prefix.

The register write is put into a dedicated 256-entry SPI communication FIFO and transferred to the AD9910 from that FIFO. By default, the DCP waits until the register write has been performed and the FIFO is empty before continuing with the next instruction. This can be explicitly stated with the `:w` ("wait") suffix (without space) but is also the default if no suffix is specified.

Adr Hex	Symb. Name	Register Description	Access	Reset Value
0x00	CFR1	Control Function Register 1	restricted	0x00410002
0x01	CFR2	Control Function Register 2	restricted	0x004008C0
0x02	CFR3	Control Function Register 3	denied	...
0x03	ADAC	Auxiliary DAC Control	full	0x0000007F
0x04	IOUR	I/O Update Rate	full	0xFFFFFFFF
0x07	FTW	Frequency Tuning Word	full	0x0
0x08	POW	Phase Offset Word	full	0x0
0x09	ASF	Amplitude Scale Factor	full	0x0
0x0A	MCS	Multichip Sync	denied	...
0x0B	DRL	Digital Ramp Limit	full	0x0
0x0C	DRSS	Digital Ramp Step Size	full	0x0
0x0D	DRR	Digital Ramp Rate	full	0x0
0x0E	STP0	Single Tone Profile 0	full	0x0
0x0F	STP1	Single Tone Profile 1	full	0x0
0x10	STP2	Single Tone Profile 2	full	0x0
0x11	STP3	Single Tone Profile 3	full	0x0
0x12	STP4	Single Tone Profile 4	full	0x0
0x13	STP5	Single Tone Profile 5	full	0x0
0x14	STP6	Single Tone Profile 6	full	0x0
0x15	STP7	Single Tone Profile 7	full	0x0
0x16	RAMB	RAM Begin (no data)	full	N/A
0x17	RAM32E	RAM 1 Word, End	full	N/A
0x18	RAM64C	RAM 2 Words, Continue	full	N/A
0x19	RAM64E	RAM 2 Words, End	full	N/A

Table 4.2: AD9910 register names and addresses. Note that the 4 RAM registers are needed to split the RAM access and only the first of these is physically present in the AD9910, the others are pseudo-registers used inside the software. The last column lists the initial values after a `dds reset` command.

In some cases it is desirable to have the DCP continue executing instructions while the SPI transfer from the SPI FIFO is performed in the background. This can be achieved by adding the `:c` ("continue") suffix (without space). This way, up to 256 register writes can be queued in the SPI FIFO and other re-configuration tasks (e.g. configuring digital IOs) can be performed by DCP instructions while the SPI writes are carried out in the background. A `wait` instruction or an `spi` instruction with `:c` suffix has to be performed before attempting an IO_UPDATE (`update`) to ensure that the registers have been completely written.

The SPI FIFO can hold up to 256 register writes of any size. With the `:c` suffix, the DCP executes instructions *much* faster than a SPI register write into the AD9910 (up to 70 times).

Note: Immutable Bits in Registers

Not all bits and not all registers are writable, see the access column in Table 4.2. This is necessary to ensure proper operation of the RF generator. Registers `CFR3` and `MCS` cannot be written to from the DCP. From the registers `CFR1` and `CFR2`, certain bits cannot be written:
`CFR1` bits 7 to 0 are forced to binary 00000010 (all power-up and correct endianness).
`CFR2` bits 23–22, 11–9 and bit 5 cannot be modified.

Examples:

`dcp 0 spi:FTW=0x12345678`	write 0x12345678 into FTW register of channel 0
`dcp 0 spi:7=0x12345678`	same, register as decimal numeric
`dcp 0 spi:0x7=305419896`	same, register as hex, value in decimal
`dcp 1 spi:cfr2=0x01000080`	set CFR2 to enable single tone profile ASF
`dcp 1 spi:stp0=0x17ff00002147ae14`	set STP0 of channel 1 to amplitude 0x17ff=6143
	...and frequency 0x2147ae14 = 130 MHz

Writing to the SRAM in the AD9910. A special procedure must be followed when writing to the 1024 word SRAM in the AD9910:

- First, a write to the `RAMB` register must be performed (without data). `RAMB` stands for "RAM Begin". This instructs the DCP to begin streaming data to the SRAM in the AD9910. When entering the command, a dummy register value of 0 must be supplied which is not stored in SRAM.
- The actual data is stored in the SRAM by writing to pseudo-registers `RAM32E`, `RAM64C` and `RAM64E`. As long as at least 2 words of data remain to be written, `RAM64C` must be used (`C` for "continue"). The 32 bit word in the more significant half of the data is written first, so a DCP SPI write value of `0x1111111122222222` first stores 0x11111111 in SRAM and then 0x22222222.
- The last 1 or 2 words must be stored using a write to `RAM32E` or `RAM64E`, respecitvely (`E` for "end"). This instructs the DCP to end streaming data to the SRAM in the AD9910.
- No other registers may be accessed between `RAMB` and `RAM*E`.

Example for storing 6 bytes in SRAM: Please not the AD9910 datasheet how to set up the profile registers before accessing the SRAM.

`dcp 0 spi:RAMB=0:c`	begin writing to the SRAM; dummy value 0 not stored
`dcp 0 spi:RAM64C=0x00000000_0009de7d:c`	write 2 words, 0 and then 0x9de7d, continue
`dcp 0 spi:RAM64C=0x00277872_0058c94b:c`	write 2 more words, continue
`dcp 0 spi:RAM64E=0x009dc970_00f66e3c`	write the last 2 words, end writing to SRAM

Using the underscore '_' in figures can be used to improve legibility; the underscores have no meaning and are ignored by FlexDDS-HD.

Example for storing 1 word in SRAM:

`dcp 0 spi:RAMB=0:c`	begin writing to the SRAM; dummy value 0 not stored
`dcp 0 spi:RAM32E=0x009dc970`	write the word and end writing to SRAM

Example for storing 3 words in SRAM:

Adr Hex	Symbolic Name	Register Description	Access	Chan
0x080	CFG_DIO_0	Configure DIO 0 on frontpanel	=+-^	0 only
0x081	CFG_DIO_1	Configure DIO 1 on frontpanel	=+-^	0 only
0x082	CFG_DIO_2	Configure DIO 2 on frontpanel	=+-^	0 only
0x084	CFG_UPDATE	Configure routing to IO_UPDATE pin on AD9910	=+-^	PC
0x085	CFG_OSK	Configure routing to OSK pin on AD9910	=+-^	PC
0x086	CFG_DRCTL	Configure routing to DRCTL pin on AD9910	=+-^	PC
0x087	CFG_DRHOLD	Configure routing to DRHOLD pin on AD9910	=+-^	PC
0x088	CFG_PROFILE	Configure routing to PROFILE pins on AD9910	=+-^	PC
0x08a	CFG_CHAN	General channel configuration register	=+-^	PC
0x100	AM_S0	Analog Modulation, Scale Factor 0	=	PC
0x101	AM_S1	Analog Modulation, Scale Factor 1	=	PC
0x102	AM_O	Analog Modulation, Offset	=	PC
0x103	AM_O0	Analog Modulation, Offset Input Channel 0	=	PC
0x104	AM_O1	Analog Modulation, Offset Input Channel 1	=	PC
0x105	AM_CFG	Analog Modulation Configuration	=	PC

Table 4.7: DCP registers inside the FPGA. In contrast to the FlexDDS-NG where only channel 0 could configure the DIO hardware, on the FlexDDS-HD each RF channel (A, B) has 3 associated DIOs that it can configure (only 2 accessible with the ADC option). For a detailed register description, see page 58. The access column describes register access modes: '=' for write, '+', '-', '^' to set, clear, toggle bits. 'PC' means one dedicated register *per* DCP *channel*.

`dcp 0 spi:RAMB=0:c`	begin writing to the SRAM; dummy value 0 not stored
`dcp 0 spi:RAM64C=0x1111111122222222:c`	write 2 words, continue
`dcp 0 spi:RAM32E=0x33333333`	write the word and end writing to SRAM

Note: In this example, the `:c` suffix is used in the DCP commands to slightly improve write speed (and also allows to free up the DCP for other operations). The `:c` suffix can also be left away. It is however important to be sure that the SPI queue is flushed before performing an update, so it is highly recommended to *not* use `:c` for the last command. This makes the DCP wait for the transfer of all the SPI commands into the AD9910.

4.2.3 `dcp wr`: Modifying Internal FPGA Registers

`dcp` [*CHAN*] `wr:`*REG*`=`[`+-^`] *VAL*[`!`]

REG denotes the DCP register address and can be specified either as symbolic name or as register address, see Table 4.7 on page 52. Symbolic names are case-insensitive. A detailed register description is given in a separate chapter on page 58.

VAL is the value to be written into the register. Depending on the register type, this value can have up to 32 bits. It can be specified in hex with `0x` prefix or in decimal.

Note: There is one DCP per RF output channel. Each DCP has full write access to its own set of registers and no access to those of the other channel. In contrast to the FlexDDS-NG where only channel 0 had access to the digital IO configuration, on the FlexDDS-HD each RF channel has its own set of 3 digital

Num	Name	Description
0	NONE	No event
2	ALL_SPI_FIFOS_FLUSHED	SPI FIFO into AD9910 empty on both channels
3	DIO_IN_A0_RISING	Rising edge seen on DIO input 0 of RF channel A
4	DIO_IN_A0_FALLING	Falling edge seen on DIO input 0 of RF channel A
5	DIO_IN_A0_LEVEL	Level (low/high) seen on DIO input 0 of RF channel A
6,7,8	DIO_IN_A1_...	Same as 3,4,5 (rising, falling, level) for DIO 1, RF channel A
9,10,11	DIO_IN_A2_...	Same as 3,4,5 (rising, falling, level) for DIO 2, RF channel A
15,16,17	DIO_IN_B0_...	Same as 3,4,5 (rising, falling, level) for DIO 0, RF channel B
18,19,20	DIO_IN_B1_...	Same as 3,4,5 (rising, falling, level) for DIO 1, RF channel B
21,22,23	DIO_IN_B2_...	Same as 3,4,5 (rising, falling, level) for DIO 2, RF channel B
27	BP_TRIG_A	Event on backplane trigger bus A of the rack
28	BP_TRIG_B	Event on backplane trigger bus B of the rack
The following refers to the *same* channel (numbers in brackets to the *other* channel):		
32 (48)	(O_)SPI_FIFO_FLUSHED	SPI FIFO into AD9910 empty; all SPI writes finished
33 (49)	(O_)SPI_FIFO_EV0	SPI FIFO write event 0 [not yet documented]
34 (50)	(O_)SPI_FIFO_EV1	SPI FIFO write event 1 [not yet documented]
35 (51)	(O_)DROVER	AD9910 ramp complete (DROVER pin)
36 (52)	(O_)RAM_SWP_OVR	AD9910 RAM sweep over (RAM_SWP_OVR pin)

Table 4.8: DCP events. The top half shows global event numbers. The bottom half are per-channel event numbers. For per-channel events, the event numbers in brackets refer to events from the *other* channel while those not in brackets refer to the *same channel*. Names for the other channel must be prefixed with O_.

IOs (only 2 of those are accessible with the ADC option).

A DCP register write takes just a single DCP instruction cycle. Hence, there is no need to wait for a register write to complete.

4.2.4 dcp wait: Timed Waits or Waiting For Up To 2 Events

```
dcp [CHAN] wait:[TIME[h|x]]:[EV0[&,EV1]][:u][!]
```

TIME is the wait timeout in units of 1.024 μs. Valid range is 0 to $2^{24} - 1 = 16777215$, giving up to $\approx$ 16 seconds with about 1 μs resoluton. With suffix h, the delay timer is in high-resolution mode and the time unit is 8 ns. The valid range 0 to $2^{24} - 1$ then results in up to 134 ms delay with a resolution of 8 ns. If *TIME* is omitted, the timeout is inifinite and only events will terminate the wait.

EV0 and *EV1* are up to 2 events to wait for. They can be specified numerically or with their symbolic name (e.g. BP_TRIG_A). See Table 4.8 (page 53) for a list of events. If no event is given, only the timeout is active. If one event is given, the wait is terminated as soon as the event occurs. If two events are given, they are separated by either & or ,. If separated by &, *both* events have to occur simultaneously to terminate the wait. Otherwise, *any* of the events terminates the wait.

If the :u flag is set at the end, then an IO_UPDATE of the synthesizer chip will be generated when the wait instruction is over. This is particularly useful for triggering an update from an external digital input.

Examples:

`dcp 0 wait:1000:`	wait for about 1024 us (on channel 0)
`dcp 0 wait:1000h:`	wait for about 8000 ns
`dcp 0 wait:1000:DROVER,36`	wait for event 35 or 36 with a 1024 us timeout
`dcp 0 wait::DIO_IN_A0_RISING`	wait for rising edge on DIO input 0 (RF channel A)
`dcp 0 wait:1000h::u`	wait for about 8000 ns, trigger IO_UPDATE afterwards
`dcp 0 wait::DIO_IN_A1_RISING:u`	wait for rising edge on DIO input 1, then trigger IO_UPDATE

Complete example. In this example, we flush after every instruction using the '!' suffix **which is not recommended for efficiency reasons** but helps see the action of each instruction when executing it.

`dcp dds_reset`	start over clean
`dcp spi:CFR2=0x1000080`	set single tone ASF bit and matched latency on RF channels A and B
`dcp 1 spi:stp0=0x3fff000020000000!`	Load setting 125 MHz, full amplitude for RF channel B
`dcp 1 update:u!`	make new STP0 setting go into effect (RF output appears)
`dcp 1 spi:stp0=0x1fff000014bc6a7f!`	Load setting 81 MHz, half amplitude for RF channel B
`dcp 1 wait::DIO_IN_A0_RISING:u!`	wait for rising edge on DIO in 0 (RF chan. **A!**), then update
`dcp 1 spi:stp0=0x3fff000014bc6a7f!`	Load setting 81 MHz, full amplitude for RF channel B
`dcp 1 wait::DIO_IN_B1_RISING:u!`	wait for rising edge on DIO in 1 (RF chan. **B!**), then update

4.2.5 `dcp update`: Update Settings and Control Pins

`dcp` [*CHAN*] `update:`[`+-=^`]*SPEC*[`!`]

SPEC specifies what to update or change. Multiple settings can be concatenated and will all be carried out *simultaneously*. The prefix symbol specifies whether to set/increment (`+`), or to clear/decrement (`-`) or to toggle (`^`).

`u`	Pulse the IO_UPDATE pin to the AD9910 which makes most of the register writes come into effect.
`+o`	Set the OSK pin of the AD9910 (drive HIGH).
`-o`	Clear the OSK pin of the AD9910 (drive LOW).
`^o`	Toggle the OSK pin of the AD9910.
`+/-/^d`	Set/clear/toggle the DRCTL pin of the AD9910 .
`+/-/^h`	Set/clear/toggle the DRHOLD pin of the AD9910.
`+p`	Increment the value at the PROFILE2:0 pins of the AD9910.
`-p`	Increment the value at the PROFILE2:0 pins of the AD9910.
`=`*N*`p`	Set the value at the PROFILE2:0 pins of the AD9910 to *N* (0...7).
`+/-/^a`	Set/clear/toggle the DIO 0 pin of the DCP channel. (*)
`+/-/^b`	Set/clear/toggle the DIO 1 pin of the DCP channel. (*)
`+/-/^c`	Set/clear/toggle the DIO 2 pin of the DCP channel. (*)

(*) Note: Each channel DCP has a digital A,B,C output. However, the physical SMA DIO plug will only output that signal when first configured as *output* and when the appropriate signal is selected in the DIO output mux. See chapter 4.3, registers `CFG_DIO_A`, etc.

Examples:

`dcp 0 update:u!`	update the AD9910 on channel 0
`dcp 0 update:+o-dh!`	AD9910: set the OSK pin HIGH and clear the DRHOLD and DRCTL pins

`dcp 0 update:^o=3p`	AD9910: toggle OSK and select profile 3 (no flush)

4.2.6 `dcp start|stop|reset|dds_reset`: Control Execution

```
dcp start|stop|reset|dds_reset
```

`dcp start` puts the DDS command processor into running mode so that it starts executing commands and **dcp stop** stops the DCP so that it no longer executes commands. You can `dcp start` it agagin so that it resumes. This is rarely needed when using the DCP network ports on the main control module because in this case the DCPs come up in running state after a reset. A 'start' implicltly also performs 'try_flush'.

`dcp reset` resets the DCP and discards all pending commands. This is especially useful if you like to clear out any remaining commands while avoiding that the RF output is interrupted as it would be when using `dds reset` or a full slot hardware reset.

`dcp dds_reset` (synonym: command `dds reset`) resets the DCP just like `dcp reset` and additionally re-initializes the synthesizer chip (e.g. AD9910). This ensures a clean re-start and is the recommended way to start when the RF generator is in an unknown state.

`dcp dds_reset` is the same as `dds reset` which is only available on the USB console.

Note: There are subtle differences between those commands when entered via the USB console compared to via the DCP network port (text protocol): When controlled via the network port, the DCP starts up in *running* state, so after `dcp reset` and `dcp dds_reset` the DCP is already running and executes the first instruction that arrives. In contrast, when entered via the USB console, those commands will leave the DCP in *stopped* state, so a `dcp start` will be required to start it. You can queue instructions into the FIFO before starting it.

You can check whether the DCP is started/running by using the command `dcp status` on the USB console.

Note: DCP Start/Stop Cannot Be Used For Slot Synchronization

`dcp start/stop` cannot be used to synchronize RF generator modules across multiple slots. Instead, it is is required to use a wait-for-trigger instruction on all slots to be synchronzied. However, when using `dcp start` without channel spec to affect all channels of a single slot module, they will be fully synchronized.

Note: Resetting All Channels in a Rack

To reset multiple RF generator modules, you can use the admin/control CLI command `slot` [*NNN*] `bq:cmd dcp dds_reset start`. In this case the 'start' is required because the command is sent internally over the USB console. Ensure to wait some time until this command was fully processed before starting to queue DCP instructions. In a more invasive way, the `slot` [*NNN*] `hwreset` admin/control CLI command can be used to hardware reset multiple slots. Both these options avoid the pitfall of a stalled network stream preventing to send a reset over the DCP text network connection.

Note: Wait Time Required After Rack Reset

The `dcp dds_reset` empties all FIFOs and cancels all DCP actions! Therefore after such a command, you **must** wait at least 100 ms before sending new commands over the network, otherwise these new commands may be removed as well. It may be useful to send a reset as the last command of a sequence to avoid the waiting time; in this case ensure that you do not send it before all commands have been executed.

4.2.7 `dcp flush|try_flush`: FIFO Flush to DCP

```
dcp flush|try_flush
```

`dcp flush` ensures that instructions queued locally actually reach the DCP inside the RF genrator module's FPGA. Without the flush, they may still be stuck in a local FIFO. In most cases, adding a '!' at the end of a DCP instruction has the same effect.

When the `dcp flush` is performed as USB Command on the USB console, then it flushes the local microcontroller FIFO onto the FPGA. You can use also use the `autoflush_msec` setting on the USB console to ensure that the microcontoller's FIFO is flushed regularly.

When the `dcp flush` is performed on the network DCP port, it flushes the FIFO of the main control module and ensures it is transferred to the RF generator module.

All the flush commands can block if the target FIFO (the 4k FIFO on the generator module) runs full. For this reason, the `dcp try_flush` is available on the USB console. It is not available on the network ports.

4.2.8 `dcp status`: Print Current Status Information

```
dcp status
```

This command is only available via the USB console. The output looks similar to the following:

```
DCP    : INST   DCP_FIFO  SPI_FIFO  LOCAL
       :       4096       256        512
DCP[0]: wait    324 ---x     0 E--x     0  running
DCP[1]: idle      0 E--x     0 E--x     0  running
```

This means that there are 324 instructions in the instruction FIFO of the DCP on channel 0 and none in channel 1. The communication (SPI into AD9910) and local (microcontroller) FIFOs are empty in both cases. The second line gives the total size of the FIFOs (in number of entries).

There are a couple of single-letter flags which are either cleared (dash '`-`') or set (letter). A flag '`E`' denotes *empty*, a flag '`F`' means *full*, '`x`' stands for enabled ("executing/transferring") and '`r`' for "held in reset".

The `inst` column specifies the instruction currently executed by the DCP.

On the FlexDDS-HD similar information can be obtained via the admin/control CLI using the command `slot print=dcp`:

```
______ Global_Errors__ Dly T CBMLAR  uC_FIFO____  FPGA_FIFO_____ EF EF Insn0,Insn1
S[06]: 0x0000 -------- 103 G CBMLA-    0,  0/512   324,   0/4096 E- -- wait ,idle
```

```
S[07]: 0x0000 -------- 111 G CBMLA-    0,  0/512     0,   0/4096 E- E- idle ,idle
```

The FPGA FIFO state is in the column `FPGA_FIFO` and the SPI FIFO is not displayed but it drains so quickly that it rarely has any entries. The microcontroller FIFO is displayed in the column `uC_FIFO`. The empty/full flags are displayed in the `EF` columns and the current DCP instruction for both RF channels is displayed as `Insn0,Insn1`. The FIFO sizes are the figures after the '/' (in number of DCP instructions).

4.3 DCP Register Description

This chapter describes the registes in the DCP.

Registers are up to 32 bit in size although often not all bits are used. Unused bits are denoted with '-' and must be written as zero.

Many registers support not only writing new values but also setting/clearing/toggling bits. These registers are limited to 30 bits. The topmost 2 bits are the write access mode `WSCT`: 00 to write, 01 to set, 10 to clear, 11 to toggle bits. Instead of setting the `WSCT` access bits directly, you can also use the prefix '+' for "set", '-' for "clear", '^' for "toggle" when issuing the `dcp wr:` command, e.g. `dcp wr:cfg_dio_0=-0x100` to clear bit 8.

4.3.1 CFG_DIO_0: Configure DIO 0

Address: `0x080`
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	WSCT		-	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	-	-	DIR	INV	0	OUT_MUX						
Defl.							0	0	0	0						

This register configures the DIO 0 digital input/output on the frontpanel. In the FlexDDS-HD (unlike the FlexDDS-NG), each DCP has its own set of 3 DIOs: DCP channel 0 controls RF channel RF:A and the associated 3 DIOs A0, A1, A2. Similarly DCP channel 1 controls RF:B and the DIOs B0, B1, B2.

Register content description:

`WSCT` Access mode: 00 to write, 01 to set, 10 to clear, 11 to toggle bits.

`DIR` DIO port direction: 1 for output, 0 for input (default).

`INV` When set, invert the port. Inversion will affect input and output equally. Note that inversion does *not* alter the logic behind rising/falling edge detection, i.e. a low to high transition of the input will always generate a rising event even if `INV` is set.

`OUT_MUX` When configured as output (`DIR`=1), choose the signal routed to the DIO output port. See Table 4.12.

Num	Name	Description
0...63	(Events)	Same as the events in Table 4.8 (page 53)
92	ADC_CH0_SIGN	Sign bit of the ADC channel 0 conversion result
93	ADC_CH1_SIGN	Sign bit of the ADC channel 1 conversion result
100 ...102	DCP0_DIO_A,B,C	DIO A,B,C output from the channel 0 DCP
103 ...105	DCP1_DIO_A,B,C	DIO A,B,C output from the channel 1 DCP
126	FADC_CLK_TGL	ADC clock divided by 2, 180° phase uncertainty (DEBUG ONLY!)
127	DCP_CLK_TGL	DCP clock divided by 2, 180° phase uncertainty (DEBUG ONLY!)

Table 4.12: Output mux choices: Values for `OUT_MUX`. Choose which signal is routed out via the SMA DIO connector.

Example:

```
dcp dds_reset
dcp 0 wr:CFG_DIO_0=0x200      configure RF:A's DIO 0 port as output (DIR=1), LOW (OUT_MUX=0)
dcp 0 wait:1000:              wait about 1 ms
dcp 0 wr:CFG_DIO_0=0x300      same as above but now inverted; port will go HIGH.
dcp 0 wait:1000:
dcp 0 wr:0x080=0x200          same as first line with numeric register address; port goes LOW again
dcp 0 wait:1000:
dcp 0 wr:cfg_dio_0=^0x100     toggle the INV bit; will go from 0 to 1; port goes HIGH
dcp 0 wait:200:               wait about 200 µms
dcp 0 wr:cfg_dio_0=^0x100     toggle the INV bit again back to 0; port goes LOW
dcp 0 wait:200:
dcp 0 wr:cfg_dio_0=+0x100     set the INV bit; port goes HIGH
dcp 0 wait:200:
dcp 0 wr:cfg_dio_0=-0x100     clear the INV bit; port goes LOW
dcp flush
```

4.3.2 CFG_DIO_1: Configure DIO 1

Address: 0x081
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

See the description for CFG_DIO_0 above.

4.3.3 CFG_DIO_2: Configure DIO 2

Address: 0x082
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

See the description for CFG_DIO_0 above.

Example: Wait for trigger on DIO A1 before switching frequencies. We're operating on RF channel RF:B. By default after a reset, all DIOs are configured as inputs. Note that even though DCP 1 can only control the DIOs B0, B1, B2 it can nevertheless access the input signal on the DIOs A0, A1, A2 as well.

```
dcp dds_reset
dcp 1 spi:stp0=0x3fff000010000000   configure single tone profile: 62.5 MHz
dcp 1 wait::DIO_IN_A1_RISING        wait for rising edge on DIO 1 of the other RF channel (RF:A)
dcp 1 update:u                      update the AD9910, makes waveform appear at RF output
dcp 1 spi:stp0=0x3fff000020000000   immediately pre-configure the next frequency (125 MHz)
dcp 1 wait::DIO_IN_A1_RISING        wait for next rising edge on DIO A1
dcp 1 update:u                      after rising edge, update the AD9910 again (next freq. at output)
dcp flush
```

4.3.4 CFG_OSK: Configure Routing to the OSK Pin on the AD9910

Address: 0x085
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	WSCT		-	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	OPMODE			INV	0	0	0	SRC_MUX				
Defl.					000			0	0	0	0	0				

This register configures how the OSK pin into the AD9910 is routed.

By default, the OSK pin of the DDS chip is connected to the DCP processor. It can be connected differently by configuring this register.

The pin routing has a local MUX (multiplexer) configured via the `SRC_MUX` which allows to select an alternate source for the OSK pin (e.g. a frontpanel DIO).

Register content description:

`WSCT` Access mode: 00 to write, 01 to set, 10 to clear, 11 to toggle bits.

`OPMODE` 000: connect the DCP to the OSK directly (default) (*).
001: disconnect DCP and force the OSK pin to 0/LOW (*)
010: disconnect DCP and route the local MUX output to the OSK pin (*)
011: logical AND of the local MUX output and the DCP output (**)
100: logical OR of the local MUX output and the DCP output (**)

`INV` When set, allows logical inversion. The `OPMODE` choices marked (*) above are inverted if that bit is set. Those marked with (**) will change to "AND NOT", "OR NOT" when set.

`SRC_MUX` Specifies the local MUX selection. You can choose between any of the global event bus lines as listed in Table 4.8 (page 53), values 0 to 31.

Example: Gating the RF output via DIO: We are going to use the external DIO A/B0 to quickly switch on/off the RF output, i.e. to gate the RF output via the OSK functionality of the DDS chip.

In the `CFR1` register of the AD9910, we need to set the "manual OSK external control" (bit 23) and "enable OSK" (bit 9) but we need to keep "auto OSK" disabled. Also, the ASF register needs to be set to the desired amplitude because when OSK is enabled in the AD9910, the amplitude scale factors from the STP registers are ignored.

Note that the digital IO must be configured as input for this to work (this is the default; see register `CFG_DIO_0`.

The `CFG_OSK` register has to be configured to disconnect the DCP and route local MUX output to the DDS chip. The MUX is configured to choose the DIO 0 input (Table 4.8: `DIO_IN_A0_LEVEL`, has value 5, i.e. binary 00101, `DIO_IN_B0_LEVEL`, has value 17, i.e. binary 10001).

```
dcp dds_reset
dcp   spi:cfr1=0b0000000_11000001_00000010_00000000   set OSK bits (see text above; RF:A and B)
dcp   spi:asf=0xffffffff                              set full amplitude (RF:A and B)
dcp   spi:stp0=0x3fff00001999999a                     set STP0 to 100 MHz (ampl. irrelevant)
dcp   update:u                                        update settings in the AD9910 (RF:A and B)
dcp 0 wr:CFG_OSK=0b010_0_000_00101                    RF:A: route DIO A0 to the OSK pin (*)
dcp 1 wr:CFG_OSK=0b010_0_000_10001                    RF:B: route DIO B0 to the OSK pin
dcp flush                                             (dcp start if controlled via USB console)
```

(*) Remember that the underscores are just for readability, one could also write `0b010000000101` or `0x405`.

The logic can be inverted by setting the INV bit (CFG_OSK=0b010_1_000_00101)

4.3.5 CFG_UPDATE: Configure Routing to the IO_UPDATE Pin on the DDS Chip

Address: 0x084
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	WSCT		-	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	OPMODE			INV	0	0	0	SRC_MUX				
Defl.					000			0	0	0	0	0				

This register configures how the IO_UPDATE pin into the AD9910 is routed. This register normally does not have to be changed. It is higly recommended to use great care when changing it, as disconnecting the DCP from the update pin will prevent normal operation of the DCP register writes into the AD9910.

This register works completely analogous to the CFG_OSK register. Please refer to that register for details.

4.3.6 CFG_DRCTL: Configure Routing to the DRCTLPin

Address: 0x086
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	WSCT		-	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	OPMODE			INV	0	0	0	SRC_MUX				
Defl.					000			0	0	0	0	0				

This register configures how the DRCTL pin into the AD9910 is routed.

This register works completely analogous to the CFG_OSK register. Please refer to that register for details.

Here's an example for the AD9910 that generates **frequency ramps ("sweeps") with external control via digital input**: The following instructions generate a ramp from 20 MHz to 100 MHz with external control via the DIO A1 input. When DIO A1 is HIGH, it ramps up and stays at the end frequency (100 MHz). Once DIO A1 goes low again, it jumps down to the start frequency (20 MHz) again. For demo purposes, connect a rectangular waveform with 0.2 Hz to DIO A1.

```
dcp dds_reset
dcp 0 spi:STP0=0x3fff0000051eb852      set 30 MHz, full amplitude
dcp 0 spi:CFR2=0x1000080               set single tone ASF bit and matched latency
dcp 0 update:u                         update AD9910
dcp 0 wait:500000:                     wait for half a second (for demo purposes)
dcp 0 spi:DRL=0x1999999a051eb852       prepare ramp limits to 30 MHz and 100 MHz
dcp 0 spi:DRSS=0x7000001a00000008      ramp step sizes: slow upwards, instant downwards
dcp 0 spi:DRR=0x00010008               prepare ramp rate
dcp 0 spi:CFR2=0x1080080               enable ramp
dcp 0 wr:cfg_drctl=0b0100_00001000     route DIO A1 (8) to DRCTL pin of AD9910
dcp 0 update:u                         IO_UPDATE to commit register changes
```

`dcp flush`	(`dcp start` if controlled via USB console)

Here's a similar example (also for the AD9910): **Triggered upwards frequency ramps ("sweeps")** Unlike the previous example, each rising edge of the DIO A1 input triggers an upwards frequency ramp; once the ramp reaches its final frequency it immediately jumps down to the start frequency and waits for the next rising edge of the DIO A1 input. The only difference is that we set the no-dwell bit this time.

`dcp dds_reset`	
`dcp 0 spi:STP0=0x3fff0000051eb852`	set 30 MHz, full amplitude
`dcp 0 spi:CFR2=0x1000080`	set single tone ASF bit and matched latency
`dcp 0 update:u`	update AD9910
`dcp 0 wait:500000:`	wait for half a second (for demo purposes)
`dcp 0 spi:DRL=0x1999999a051eb852`	prepare ramp limits to 30 MHz and 100 MHz
`dcp 0 spi:DRSS=0x7000001a00000008`	ramp step sizes: slow upwards, instant downwards
`dcp 0 spi:DRR=0x00010008`	prepare ramp rate
`dcp 0 spi:CFR2=0x10c0080`	enable ramp **and set no-dwell high bit**
`dcp 0 wr:cfg_drctl=0b0100_00001000`	route DIO A1 (8) to DRCTL pin of AD9910
`dcp 0 update:u`	IO_UPDATE to commit register changes
`dcp flush`	(`dcp start` if controlled via USB console)

4.3.7 CFG_DRHOLD: Configure Routing to the DRHOLD Pin on the AD9910

Address: `0x087`
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	WSCT		-	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	OPMODE			INV	0	0	0	SRC_MUX				
Defl.					000			0	0	0	0	0				

This register configures how the DRHOLD pin into the AD9910 is routed.

This register works completely analogous to the `CFG_OSK` register. Please refer to that register for details.

4.3.8 CFG_PROFILE: Configure Routing to the PROFILE Pins on the AD9910

Address: `0x088`
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	WSCT		-	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	OPMODE2			OPMODE1			OPMODE0			INV2	INV1	INV0
Defl.					000			000			000			0	0	0

This register configures how the three PROFILE pins into the AD9910 are routed.

This works analogous to the `CFG_OSK` but the register contents are different:

WSCT	Access mode: 00 to write, 01 to set, 10 to clear, 11 to toggle bits.
OPMODE*n*	Choose operation mode for profile pin *n* (0, 1, 2): 000: connect the DCP to the PROFILE*n* directly (default) (*). 001: disconnect DCP and force the PROFILE*n* pin to 0/LOW (*) 010: disconnect DCP and route the DIO A*n* (0, 1, 2) into the PROFILE*n* pin (*) 011: disconnect DCP and route the DIO B*n* (2, 1, 0) into the PROFILE*n* pin (*)
INV*n*	When set, allows logical inversion, one bit for each PROFILE pin. The OPMODE choices marked (*) above are inverted if that bit is set.

Note that the order for DIO B*n* pins is *reversed*. The reason for this is to make all PROFILE pins available to external DIOs also for the RF generator modules with the ADC option that lack the DIO A2 and DIO B2 frontpanel connectors.

Hence, in order to route the DIO A0 to the PROFILE0 pin, DIO A1 to PROFILE1 and DIO A2 to PROFILE2, you would be using a value of 0b010_010_010_000. In order to just route DIO A0 to PROFILE0 and keep the other two profile pins connected to the DCP, use a value of 0b000_000_010_000.

```
dcp dds_reset
dcp 0 spi:stp0=0x3fff00001999999a        ch 0, set STP0 to 100 MHz, phase to 0° full amplitude
dcp 0 spi:stp1=0x1fff00004ccccccd        ch 0, set STP1 to 300 MHz, phase to 0° half amplitude
dcp 0 update:u                           flush settings in the AD9910
dcp 0 wr:CFG_PROFILE=0b000_000_010_000   route DIO A0 to the PROFILE0 pin (*)
dcp flush                                (dcp start if controlled via USB console)
```

(*) Remember that the underscores are just for readability, one could also write 0b000000010000 or 0x010.

4.3.9 CFG_CHAN: Generic Channel Configuration Register

Address: 0x08a
Access: Write, Set, Clear, Toggle; Per-channel (one for each DCP)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	WSCT		-	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	-	-	-	-	-	-	-	RF_FLOAT		DIS_RFAMP		RST
Defl.												00		00		0

Register content description:

`RST` When the `RST` bit is written to 1, it will instruct the microcontroller on the slot to reset the DDS core, program it to the initial state and also reset the DCP. The reset action will take considerable time, so please allow at least 100 ms after the register write. The bit is cleared automatically once the DCP has been resettet.
This is exactly the same as issuing the `dcp dds_reset` command (via USB or the rack network interface). Except that if the reset is the last command and the DCP is blocked (e.g. waiting for a trigger) it might never execute the reset. In contrast, issuing a `dcp dds_reset` from the via USB or network is always possible and will terminate any pending DCP action.
NOTE: This action takes time. Allow this action to complete before feeding new commands. As this command also empties the DCP FIFO, commands fed while the DDS resets itself are silently discarded. This also holds for `wait` instructions.

`DIS_RFAMP` When set to 1, disable the RF amplifier.
The FlexDDS-HD-AD9910 has one main RF amplifier per channel and it is disabled by setting bit 1. (The second bit is reserved for modules with 2 RF amplifiers per channel, e.g. for I and Q outputs.)
Note: It is advisable to use the OSK feature to switch on/off the RF output instead of disabling the amplifier because it is faster and has a cleaner transition. Disabling the RF amplifier is useful if the OSK feature cannot be used or to save power.

`RF_FLOAT` If set to 1, the RF output is configured as floating by opening the GND relay (if installed). This may be used to combat ground loops. Only the lower bit is used, the higher bit is reserved for future use.
NOTE: In some environments, this may degrades phase noise performance for certain offset frequencies, so it is advisable to only do this when necessary.

Here's an example for the FlexDDS-HD-AD9910 that resets one of the DDS channels from the DCP. Note that this is a "final" thing as the DCP resets itself and will discard any further commands queued for it.

```
dcp dds_reset
dcp   spi:STP0=0x3fff0000051eb852    set STP0 to 30 MHz, full amplitude
dcp   spi:CFR2=0x01000080            set CFR2 to matched latency and ASF from STP
dcp   update:u                       update; this makes the above appear at the RF outputs
dcp   wait:1000000:                  wait a second
dcp 0 wr:CFG_CHAN=1                  now reset channel 0 (RF:A), channel 1 (RF:B) keeps running
dcp flush                            (dcp start if controlled via USB console)
```

This is a similar example for the FlexDDS-HD-AD9910 which makes use of the `DIS_RFAMP` bit to disable the channel 0 (RF:A) output amplifier.

```
dcp dds_reset
dcp   spi:STP0=0x3fff0000051eb852    (see above)
dcp   spi:CFR2=0x01000080
dcp   update:u
dcp   wait:1000000:                  wait a second
dcp 0 wr:CFG_CHAN=+0b010             set DIS_RFAMP to disable the RF amplifier for channel 0
dcp flush                            (dcp start if controlled via USB console)
```

Finally, this shows how to open the GND HS relay and float an output.

```
dcp dds_reset
dcp   spi:STP0=0x3fff0000051eb852     (see above)
dcp   spi:CFR2=0x01000080
dcp   update:u
dcp 0 wait:1000000:                   RF:A waits one second
dcp 1 wait:2000000:                   RF:A waits two seconds
dcp 0 wr:CFG_CHAN=+0b1000             set RF_FLOAT to open GND relay for RF:A
dcp 1 wr:CFG_CHAN=+0b1000             set RF_FLOAT to open GND relay for RF:B
dcp flush                             (dcp start if controlled via USB console)
```

4.3.10 AM_S0: Analog Modulation, Scale Factor 0

Address: 0x100
Access: Write; one dedicated register for each DCP

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	-	-	UPD	-	-	-	-	-	-	-	-	-	-	-		→
Defl.																→

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	←	AM_S0														
Defl.	←	0														

Sets the scaling factor S_0 for the respective DCP associated with analog input channel 0. The scale factor is a signed 18 bit value (two's complement).

Note that all the writes to AM_* registers target shadow registers. All the shadow registers are copied to the effective registers at the same time if the UPD bit is set during writing.

4.3.11 AM_S1: Analog Modulation, Scale Factor 1

Address: 0x101
Access: Write; one dedicated register for each DCP

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	-	-	UPD	-	-	-	-	-	-	-	-	-	-	-		→
Defl.																→

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	←	AM_S1														
Defl.	←	0														

Sets the scaling factor S_1 for the respective DCP associated with analog input channel 1. For details, see AM_S0.

4.3.12 AM_O: Analog Modulation, Offset

Address: 0x102
Access: Write; one dedicated register for each DCP

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	-	-	UPD	-	-	-	-	-								→
Defl.																→

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	←								AM_O							
Defl.	←								0							

Sets the offset value O for the analog modulation math of the respective DCP. The offset is a signed 24 bit value (two's complement).

Note that all the writes to AM_* registers target shadow registers. All the shadow registers are copied to the effective registers at the same time if the UPD bit is set during writing.

4.3.13 AM_P: Analog Modulation, Offset for Polar Modulation

Address: 0x106
Access: Write; one dedicated register for each DCP

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	-	-	UPD	-	-	-	-	-								→
Defl.																→

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	←	AM_P							0	0	0	0	0	0	0	0
Defl.	←	0							0	0	0	0	0	0	0	0

Sets the offset value P for the analog modulation math of the respective DCP which is used only for polar modulation. The value has only 16 bits of precision but is logically arranged such that it looks like a 24 bit value (with the least significant 8 bits ignored). This is done to make AM_P compatible to the offset AM_O.

Note that all the writes to AM_* registers target shadow registers. All the shadow registers are copied to the effective registers at the same time if the UPD bit is set during writing.

4.3.14 AM_O0: Analog Modulation, Offset for Input Channel 0

Address: 0x103
Access: Write; one dedicated register for each DCP

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	-	-	UPD	-	-	-	-	-	-	-	-	-	-	-		→
Defl.																→

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	←								AM_O0							
Defl.	←								0							

Sets the offset value O_0 for the respective DCP associated with analog input channel 0. The channel offset is a signed 18 bit value (two's complement).

Note that all the writes to AM_* registers target shadow registers. All the shadow registers are copied to the effective registers at the same time if the UPD bit is set during writing.

4.3.15 AM_O1: Analog Modulation, Offset for Input Channel 1

Address: 0x104
Access: Write; one dedicated register for each DCP

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	-	-	UPD	-	-	-	-	-	-	-	-	-	-	-	→	
Defl.															→	

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	←	AM_O1														
Defl.	←	0														

Sets the offset value O_1 for the respective DCP associated with analog input channel 1. For details, see AM_OO.

4.3.16 AM_CFG: Analog Modulation Configuration Register

Address: 0x105
Access: Write; one dedicated register for each DCP

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Desc.	-	-	UPD	-	-	-	-	-	-	-	-	-	-	-	-	-
Defl.																

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Desc.	-	-	-	-	-	-	-	-	-	-	-	-	OVFQ	OVFN	MODF	
Defl.													0	0	00	

Analog modulation configuration register for the respective DCP.

Register content description:

UPD — Update. All the writes to AM_* registers target shadow registers. All the shadow registers are copied to the effective registers at the same time if the UPD bit is set during writing.

OVFQ — Quadrature result overflow enable, see next bit.

OVFN — Normal result overflow enable. Usually, the computation result saturates when an overflow occures (see $\text{coerce}_{16}()$ in formula 4.1). If the OVFN bit is set, overflow values do not saturate but instead the most significant (overflowing) bits are simply cut off and discarded. This is especially useful for phase modulation because it allows to have a continuous modulation range far beyond 2π.
NOTE: This requires firmware version 0.92 or higher.

MODF — Set the analog modulation format, i.e. the F bits of the parallel data bus into the AD9910:

MODF	Analog modulation format
00	Amplitude modulation (upper 14 bits used)
01	16 bit phase modulation
10	Frequency modulation (16 bit used, see FM gain setting of AD9910)
11	Polar modulation (8 bit phase, 8 bit amplitude)

4.4 Analog Modulation

When the ADC option was chosen, the FlexDDS-HD-AD9910-ADC RF generator module supports analog modulation of the RF outputs using signals applied to the AIn:A and AIn:B analog input ports.

The analog signal at each of the RF In ports is digitized with a dedicated ADC operating at 62.5 MS/s (i.e. 1 GHz divided by 16). The ADC has a redolution of 12 or 14 bit depending on the hardware configuration and an analog input range of $\pm 0.5\,\mathrm{V} = 1\,\mathrm{V_{pp}}$.

The digital samples out of the ADCs are processed by a linear math unit and can then be fed into the 16 bit parallel data port of the AD9910 RF generator. This allows amplitude, frequency, phase and even polar modulation.

Each DCP has one dedicated linear math unit. These two math units (one per RF output channel) are completely independent of each other. Each linear math unit has access to the sample data stream of *both* analog inputs. This means, any of the 2 RF output channels can be modulated by any of the two analog input channels or even by a weighted sum/difference of both the analog input signals. Also, the same analog input channel can be used to modulate both RF outputs simultaneously with the same or different math coefficients. You could even use e.g. analog input 0 to frequency modulate RF channel 0 while using the weighted sum of the input channels 0 and 1 to amplitude modulate the RF channel 1.

Note: Nomenclature: Channels 0, 1 and A, B

On the FlexDDS-HD frontpanel, the RF output channels 0 and 1 are labeled RF:A and RF:B. Similarly, the analog input channels 0 and 1 are labeled AIn:A and AIn:B. The idea behind using A,B for the analog and 0,1,2 for the digital channels was to avoid confusion, especially since each RF channel has its own 2 or 3 associated DIOs. Nevertheless, in this chapter, we will always use indices 0 and 1 for the analog/RF channels A and B, respectively.

For polar modulation, analog input channel 0 provides the phase information while channel 1 provides the amplitude information.

4.4.1 Amplitude, Phase and Frequency Modulation

The 16 bit modulation data D fed into the AD9910 is computed by the linear math unit in the following way:

$$D = \mathrm{coerce}_{16}\left(\frac{(A_0 - O_0)\cdot S_0 + (A_1 - O_1)\cdot S_1}{2^{12}} + O\right) \tag{4.1}$$

Here, A_0 and A_1 are the analog samples generated by the ADC attached to analog input channels 0 and 1, respectively. These are 16 bits wide and MSB aligned (i.e. for a 12 or 14 bit ADC, the last 4 or 2 bits are zero).

O_0 and O_1 are user configurable offsets with a width of 18 bits. These can be used e.g. to compensate offset errors in the ADC. The result of the difference operation is also 18 bits wide.

S_0 and S_1 are user configurable scaling factors and are 18 bits wide. These can be used to control the slope of the two linear transfer functions. The result of the multiplication is 36 bits wide.

The result of the math operations is scaled down by 2^{12} by cutting off least significant 12 bits.

A global offset O (24 bits wide) is then added and can be used to configure the intercept of the bilinear transfer function.

The resulting figure is finally coerced to a 16 bit value, i.e. values below zero saturate to zero while values above $2^{16}-1$ are saturating at $2^{16}-1 = 65535$. This saturation can be disabled by setting the `OVFN`, `OVFQ` bits in the `AM_CFG` register, a feature that allows phase modulation beyond the range of $0 \ldots 2\pi$ (requires firmware version 0.92 or higher).

$$\text{coerce}_{16}(x) := \begin{cases} 0, & \text{if } x < 0 & \text{and } \texttt{OVFN}, \texttt{OVFQ} = 0 \\ 2^{16} - 1, & \text{if } x \geq 2^{16} & \text{and } \texttt{OVFN}, \texttt{OVFQ} = 0 \\ x, & \text{if } 0 \leq x < 2^{16} & \text{and } \texttt{OVFN}, \texttt{OVFQ} = 0 \\ x \bmod 2^{16} & \text{if} & \texttt{OVFN}, \texttt{OVFQ} = 1 \end{cases} \tag{4.2}$$

The 5 coefficients O_0, O_1, S_0, S_1 and O are user configurable by writing to the corresponding analog modulation coefficient registers `AM_O0`, `AM_O1`, `AM_S0`, `AM_S1` and `AM_O`. Note that these registers have **shadow registers** and only the shadow registers are accessible from the DCP. Hence, a write to any of these registers will not immediately take effect. A write with the update bit `UPD` set to 1 is required to transfer *all* the shadow register contents *at once* to the effective registers. So, after setting up all the coefficients, by setting the update bit `UPD` during the last register write, the new set of coefficients instantly replaces the previous set. This is much like the configuration of the AD9910 and the IO_UPDATE pin.

Binary representation: All the coefficients are represented as 18 or 24 bit *two's complement* figures. This is the same representation as internally used by most computers. Here are examples of figures represented in two's complement:

value	18-bit	24 bit	comment
0	0x00000	0x000000	positive numbers are just like...
1	0x00001	0x000001	...regular binary representation
2	0x00002	0x000002	
256	0x00100	0x000100	
131071	0x1ffff	0x01ffff	largest 18 bit signed number ($2^{17} - 1$)
8388607	-	0x7fffff	largest 24 bit signed number ($2^{23} - 1$)
-1	0x3ffff	0xffffff	
-2	0x3fffe	0xfffffe	
-256	0x3ff00	0xffff00	
-131072	0x20000	0xfe0000	smallest 18 bit signed number (-2^{17})
-8388608	-	0x800000	smallest 24 bit signed number (-2^{23})

So, if you keep adding 1 to an n-bit two's complement number, it will eventually roll over from $2^{n-1} - 1$ to -2^{n-1}. All negative values have their most significant bit set, all positive values have it cleared. To extend an n-bit two's complement number to $m > n$ bits, you have to fill up all the $m - n$ *new* most significant bits with the value of the most significant bit of the original figure. (E.g. to extend a 4 bit value to a 8 bit value: `0b0101` → `0b00000101` (positive value), `0b1101` → `0b11111101` (negative value)) The 16 bit analog sample values A_n are sign-extended to 18 bits before performing the subtraction with offset O_n.

Configuration of the AD9910: To use the analog modulation feature, the parallel data port of the AD9910 has to be enabled and the desired modulation scheme (amplitude, frequency, phase, polar) has to be selected via a write to the `AM_CFG` register.

4.4.2 Example: Amplitude Modulation

Say we'd like to configure analog output channel 0 for full scale **amplitude modulation** from analog input channel 0. I.e. the full analog input range of $1\,V_{pp}$ should be translated to a amplitude modulation from zero amplitude to full amplitude.

Since the analog samples in A_0 have 16 bits (full scale) and the output D also has 16 bits (full scale), we need to scale the analog values with a trivial factor of 1. However, as the analog samples are *signed* values

and the output D has to cover the *un*signed range $0 \ldots 65525$, we need to add $65535/2 = 2^{15}$. Hence, the desired linear transfer function must look like this:

$$D = \frac{A_0}{1} + 2^{15} = \frac{(A_0 - 0) \cdot 2^{12} + (A_1 - 0) \cdot 0}{2^{12}} + 2^{15} \tag{4.3}$$

By comparing coefficients with equation 4.1, we find that:

$O_0 = 0$ $S_0 = 2^{12} = \texttt{0x1000}$ $O = 2^{15} = \texttt{0x8000}$
$O_1 = 0$ $S_1 = 0$

Here's the corresponding code (remember, underscores in figures can be inserted to improve readability and are completely ignored):

```
dcp dds_reset
dcp 0 spi:stp0=0x3fff_0000_10000000                    set frequency to 62.5 MHz, full amplitude
dcp 0 spi:CFR1=0b00000000_01000001_00000000_00000000   sinc filter and sine output (not needed)
dcp 0 spi:CFR2=0b00000000_00000000_00000000_01010000   enable parallel data port
dcp 0 wr:AM_S0=0x1000                                  set scale factor S0
dcp 0 wr:AM_O0=0                                       set offset O0
dcp 0 wr:AM_O=0x8000                                   set global offset O
dcp 0 wr:AM_CFG=0x2000_0000                            choose amplitude modulation, flush coeff.
dcp 0 update:u                                         update AD9910 (make CFR* effective)
dcp flush                                              (dcp start if controlled via USB console)
```

Here, the `AM_S1` and `AM_O1` registers are not written so their default value of 0 is used. To view the result, connect a 1 MHz sine wave signal with 1 V_{pp} amplitude (into 50 Ω; this may require to set an amplitude of 2 V_{pp} into high impedance on a function generator) to the analog input channel 0 and view th RF output channel 0 with an oscilloscope.

Similarly, if we would like to modulate RF channel 0 from analog channel 1, the same code as above is valid, just that writes to `AM_S0` and `AM_O0` have to be replaced to writes to `AM_S1` and `AM_O1`.

To modulate RF channel 1 instead, we'd use `dcp 1` rather than `dcp 0` in all code lines.

You can also amplitude-modulate RF channel 0 from analog input 0 and RF channel 1 from analog input 1:

```
dcp dds_reset
dcp 0 spi:STP0=0x3fff_0000_10000000          ch 0: frequency 62.5 MHz, full amplitude
dcp 1 spi:STP0=0x3fff_0000_08000000          ch 1: frequency to 31.25 MHz, full amplitude
dcp spi:CFR1=0b01000001_00000000_00000000    sinc filter and sine output (not needed)
dcp spi:CFR2=0b00000000_00000000_01010000    enable parallel data port
dcp 0 wr:AM_S0=0x1000                        ch 0: set scale factor S0 (
dcp 0 wr:AM_O0=0                             ch 0: set offset O0
dcp 0 wr:AM_O=0x8000                         ch 0: set global offset O
dcp 0 wr:AM_CFG=0x2000_0000                  ch 0: choose amplitude modulation, flush coeff.
dcp 1 wr:AM_S1=0x800                         ch 1: set scale factor S1 (half the modulation depth)
dcp 1 wr:AM_O1=0                             ch 1: set offset O1
dcp 1 wr:AM_O=0xc000                         ch 1: set global offset O (larger offset)
dcp 1 wr:AM_CFG=0x2000_0000                  ch 1: choose amplitude modulation, flush coeff.
dcp update:u                                 update AD9910 (make CFR* effective)
dcp flush                                    (dcp start if controlled via USB console)
```

In this example, the RF channel 1 has half the modulation depth for the same analog input voltage

because the scale factor S_1 is half as large. To obtain full scale amplitude for the maximum analog value, the offset O was increased accordingly by 50%.

If you would like to amplitude-modulate both RF output channels from the same analog input channel 0 (with possibly different scale and offset coefficients), you would replace `AM_S1` and `AM_O1` with `AM_S0` and `AM_O0` for dcp 1 in the example above.

Negative scale factors: We can use that example to amplitude modulate both RF outputs with opposite polarity. I.e. the higher the input voltage, the larger the amplitude on channel 0 and the smaller the amplitude on channel 1. We use analog input channel 0 for both output channels. Hence, we have to set S_0 for DCP channel 1 to a negative value:

DCP 0:	$S_0 = +2^{12} =$ `0x1000`	$O = 2^{15} =$ `0x8000`
DCP 1:	$S_0 = -2^{12} =$ `0x3f000`	$O = 2^{15} =$ `0x8000`

Note that we could also set S_0 to 0xff000 instead of 0x3f000 because the register is 18 bits wide and the most significant non-zero bits of 0xff000 would be truncated, leaving an effective register value of 0x3f000.

Here's the corresponding instruction listing:

```
dcp dds_reset
dcp 0 spi:STP0=0x3fff_0000_10000000          ch 0: frequency 62.5 MHz, full amplitude
dcp 1 spi:STP0=0x3fff_0000_08000000          ch 1: frequency to 31.25 MHz, full amplitude
dcp spi:CFR1=0b01000001_00000000_00000000    sinc filter and sine output (not needed)
dcp spi:CFR2=0b00000000_00000000_01010000    enable parallel data port
dcp 0 wr:AM_S0=0x1000                        S0 = +2^12 = 0x1000
dcp 0 wr:AM_O0=0
dcp 0 wr:AM_O=0x8000
dcp 0 wr:AM_CFG=0x2000_0000
dcp 1 wr:AM_S0=0x3f000                       S0 = -2^12 = 0x3f000
dcp 1 wr:AM_O0=0
dcp 1 wr:AM_O=0x8000
dcp 1 wr:AM_CFG=0x2000_0000
dcp update:u                                 update AD9910 (make CFR* effective)
flush                                        (dcp start if controlled via USB console)
```

If the figures look too "easy", here's another example with a slightly smaller scale factor:

DCP 0:	$S_0 = +4000 =$ `0xfa0`	$O = 2^{15} =$ `0x8000`
DCP 1:	$S_0 = -4000 =$ `0x3f060`	$O = 2^{15} =$ `0x8000`

4.4.3 Example: Phase Modulation

Next is an example for **phase modulation**. The frequency is set quite low to 11.7 MHz to make the effect easily visible. Both channels are configured for the same frequency but only one RF output channel is phase modulated.

```
dcp dds_reset
dcp spi:STP0=0x3fff_0000_03000000            set frequency to 11.7 MHz, full amplitude
dcp spi:CFR1=0b01000001_00000000_00000000    sinc filter and sine output (not needed)
dcp 0 spi:CFR2=0b00000000_00000000_01010000  enable parallel data port
dcp 0 wr:AM_S0=0x1000                        same full-scale modulation parameters...
dcp 0 wr:AM_O0=0                             ...as in the example above
dcp 0 wr:AM_O=0x8000
```

`dcp 0 wr:AM_CFG=0x2000_0001`	choose phase modulation, flush coefficients
`dcp update:u`	update AD9910 (make CFR* effective)
`dcp flush`	(`dcp start` if controlled via USB console)

Set up a function generator to a 1 MHz sine wave with 1 V_{pp} and connect it to the analog input channel 0. To observe the phase modulation, connect both RF outputs to an oscilloscope and compare the normal un-modulated output from RF channel 1 with the modulated output from RF channel 0. Play around with amplitude and frequency of the analog input.

Of course, you can phase-modulate the channel 0 while simultaneously amplitude modulating the channel 1, either from the same analog signal or from different analog input signals.

4.4.4 Example: Frequency Modulation

Frequency modulation works just like amplitude and phase modulation with the only caveat that the frequency tuning word is fundamentally 32 bits wide while the parallel modulation data input into the AD9910 only allows 16 bits of precision. Hence, the FM gain setting in the `CFR2` register has to be set accordingly.

Please note: When using the analog frequency modulation, the AD9910 derives the actual output frequency by starting with the currently active FTW source (STP0 or FTW register) and then *adding* the 16 bit modulation input, shifted by the FM gain setting (CFR2). **This is especially important if the frequency modulation depth is smaller than the carrier frequency. See the next example which demonstrates this case.**

Say we want to modulate the frequency in the following way: Negative full scale input (i.e. −0.5 V) should result in 10 MHz (`FTW` = `0x028f5c29`) while positive full scale input (+0.5 V) should yield 30 MHz (`FTW` = `0x07ae147b`).

To cover the required frequency range, we need an FM gain setting of at least 11 (see AD9910 datasheet). We choose 12, although 11 would work just as fine.

In the presence of an FM gain of 12, the 16-bit digital modulation values are shifted by 12 bits (i.e. multiplied by 2^{12}). Hence, the required 16-bit D values from equation 4.1 need to cover the range from `0x028f5c29`/2^{12} = `0x28f5` to `0x07ae147b`/2^{12} = `0x7ae1`.

We can now compute the scale and offset coefficients for an analog input on channel 0. We know that input channel 1 is not used (i.e. $S_1 = 0$) and set offset $O_0 = 0$ (or to whatever small value is required to cancel the analog input offset error). Equation 4.1 then simplifies to

$$D = \text{coerce}_{16}\left(\frac{A_0 \cdot S_0}{2^{12}} + O\right) \tag{4.4}$$

(Here, the 2^{12} in the denominator comes from equation 4.1 and is completely unrelated to the FM gain setting). A −0.5 V analog input correspond to an analog ADC value of $A_0 = -2^{15}$ while a positive full scale input of +0.5 V correspond to $A_0 = +2^{15} - 1$.

With that information we can now solve the following two linear equations to find S_0 and O:

$$\begin{array}{lcll} D(A_0 = -2^{15}) & = & \texttt{0x28f5} & = & 10485 \\ D(A_0 = +2^{15} - 1) & = & \texttt{0x7ae1} & = & 31457 \end{array} \tag{4.5}$$

$$\begin{aligned} S_0 &= \frac{(31457 - 10485) \cdot 2^{12}}{2^{15} - 1 + 2^{15}} = 1310.77 = \texttt{0x51f} \\ O &= \frac{10485 \cdot (2^{15} - 1) - 31457 \cdot (-2^{15})}{2^{15} - 1 + 2^{15}} = 20971.16 = \texttt{0x51eb} \end{aligned} \tag{4.6}$$

Hence, we can instruct the FlexDDS-HD to perform the requested frequency modulation by executing:

```
dcp dds_reset
dcp 0 spi:STP0=0x3fff000000000000              choose max amplitude and set FTW to zero
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp 0 spi:CFR2=0b00000000_00000000_01011100    enable parallel data port, set FM gain to 12
dcp 0 wr:AM_S0=0x51f                           S0 as computed above
dcp 0 wr:AM_O0=0                               (zero analog channel offset)
dcp 0 wr:AM_O=0x51eb                           O as computed above
dcp 0 wr:AM_CFG=0x2000_0002                    choose frequency modulation, flush coefficients
dcp 0 update:u                                 update AD9910 (make CFR* effective)
dcp flush                                      (dcp start if controlled via USB console)
```

The result can be observed by hooking up an oscilloscope to RF output channel 0 and a function generator to the analog input channel 0. By setting a $1\,V_{pp}$ square wave or a DC value, the frequencies can be measured easily with the oscilloscope.

4.4.5 Example: Frequency Modulation: Small Modulation on Large Offset

In this example we would like to an analog frequency modulation between 93 MHz and 95 MHz, i.e. modulate a carrier of 94 MHz with a modulation depth of 2 MHz.

This could be done by selecting a huge O offset parameter to position the FTW around 90 MHz and a small S_0 to give the rather small modulation depth. This has, however, the big disadvantage of losing lots of bits of precision from the analog input (being scaled down by the small S_0 and results in stepwise frequency tuning).

Instead the appropriate approach is as follows:

A start frequency of 93 MHz corresponds to an FTW of 0x17ced917. We set this into our primary FTW source (FTW register in this case).

A modulation depth of 2 MHz means an FTW change of 0x83126f which is 23.03 bits so we will need 24 bits. Since we have 16 bits of modulation input, we need to shift the input by 8 bits to cover the required 24 bit range. Hence, the FM gain in the CFR2 needs to be set to 8 (0b1000).

If we could live with 1.9 MHz modulation deth, we have less than 23 bits of FTW change and could do with an FM gain of 7 bits (0b0111).

In this case, since we start at 92 MHz and have set that into the FTW, our offset $O = 0$.

As we know $S_0 = 2^{12}$ corresponds to a full scale modulation, i.e. the full analog input voltage range is scaled to the full 16 bit of parallel modulation input data. With an FM gain of 8 bits, that is a frequency range of 24 bits or $1\,\text{GHz} \cdot 2^{24}/2^{32} = 3.906\,\text{MHz}$. Since we only need $\Delta f = 2\,MHz$ (or 23.03 of those 24 bits), we need about half the scale factor:

$$S_0 = \frac{\Delta f}{1\,\text{GHz}} \cdot \frac{2^{32} \cdot 2^{12}}{2^{16} \cdot 2^{\text{FM\_gain}}} = \frac{\Delta f}{1\,\text{GHz}} \cdot \frac{2^{28}}{2^{\text{FM\_gain}}} = \frac{2\,\text{MHz}}{1\,\text{GHz}} \cdot \frac{2^{28}}{2^8} = 2097 \tag{4.7}$$

```
dcp dds_reset
dcp 0 spi:FTW=0x17ced917                          Set base frequency to 93 MHz
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp 0 spi:CFR2=0b00000000_00000000_01011000       set FM gain to 8
dcp 0 wr:AM_S0=2097                               S0 as computed above
dcp 0 wr:AM_O0=0x8000                             (zero analog channel offset)
dcp 0 wr:AM_O=0x0                                 O = 0 as explained above
dcp 0 wr:AM_CFG=0x2000_0002                       choose frequency modulation, flush coefficients
dcp 0 update:u                                    update AD9910 (make changes effective)
dcp flush                                         (dcp start if controlled via USB console)
```

Looking closely with a spectrum analyzer will show that the phase noise is degraded in this case compared to the output signal without frequency modulation. This added phase noise comes from the analog noise in the modulation input as well as the ADC quantization noise.

4.4.6 Polar Modulation

Polar modulation is enabled by writing the `MODF` bits in the `AM_CFG` register to `0b11`.

By doing so, the linear math kernel is altered and it no longer follows equation 4.1 but instead performs the following computation:

$$\begin{aligned} D_{7\ldots0} &= \text{coerce}_{16}\left(\frac{(A_0 - O_0)\cdot S_0}{2^{12}} + O\right)/2^8 && \text{phase bits} \\ D_{15\ldots8} &= \text{coerce}_{16}\left(\frac{(A_1 - O_1)\cdot S_1}{2^{12}} + P\right)/2^8 && \text{amplitude bits} \end{aligned} \tag{4.8}$$

The linear transfer function is similar to the other modulation schemes except that (1) a second offset P is now present, which can be configured via the register `AM_P` and (2) only the 8 most significant bits of the 16 bit result are used (hence the division by 2^8 after coerceing). This way, the same values for the coefficients from amplitude and phase modulation can be used in polar mode with the same effect on the output signal.

Note that for polar modulation, analog input channel 0 is hard wired to *phase* modulation while analog input channel 1 is hard wired for *amplitude* modulation.

The following example performs a polar modulation similar to a combination to the phase and amplitude modulations at the beginning of the section. Notice how all the scale and offset parameters are identical.

```
dcp dds_reset
dcp 0 spi:stp0=0x0000000003000000                 frequency 11.7 MHz; phase and amplitude zero
dcp 1 spi:stp0=0x3fff000003000000                 RF channel 1, just to show the phase modualation
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp 0 spi:CFR2=0b00000000_00000000_01010000       enable parallel data port
dcp 0 wr:AM_S0=0x1000                             scale factor for phase modulation
dcp 0 wr:AM_O0=0                                  (analog offset on input channel 0)
dcp 0 wr:AM_O=0x8000                              offset value for phase modulation
dcp 0 wr:AM_S1=0x1000                             scale factor for amplitude modulation
dcp 0 wr:AM_O1=0                                  (analog offset on input channel 1)
dcp 0 wr:AM_P=0x8000                              offset value for amplitude modulation
dcp 0 wr:AM_CFG=0x2000_0003                       choose polar modulation, flush coefficients
```

```
dcp   update:u                          update AD9910 (make CFR* effective)
dcp flush                               (dcp start if controlled via USB console)
```

In the example above, RF channel 1 was set to a constant carrier at the same frequency to make it easier to see the phase modulation in effect.

4.5 DCP Program Examples for the AD9910

DCP execution follows a deterministic timing. Hence, instructions are typically fed into the DCPs (DDS Command Processors) first and then executed by starting the DCP.

This chapter presents a couple of examples how to use the FlexDDS-HD-AD9910 RF generator slots.

Note: USB versus network interface

Please note that when using the USB interface, the `dcp start` command is mandatory while when using the network interface for the rack, the DCPs start up running and will execute right away and there is no "start" command required. This means, in order to synchronize to the external world, a program fed into the rack would typcially have something like a `dcp wait::BP_TRIG_A` as one of the first commands to wait for an externally triggered event.

Note: `dcp dds_reset` via network interface

When sending a `dcp dds_reset` over the network interface, one has to physically **wait** at least 100 ms before transmitting the following commands over the network.

Please also see chapter 2.5.1 on page 17.

4.5.1 Basic: Two outputs with small frequency offset

The following example configures both outputs for roughtly 130 MHz. One output frequency is 0.23 Hz higher giving two RF waveforms that "move" with respect to each other.

```
dcp dds_reset                          reset and initialize the DDS and also the DCP
dcp 0 spi:stp0=0x3fff00002147ae14      set freq (FTW in STP0) to 130 MHz for ch 0 (RF:A)
dcp 1 spi:stp0=0x3fff00002147ae15      set freq (FTW in STP0) to 130 MHz + 0.23 Hz for ch 1 (RF:B)
dcp update:u                           update AD9910 (both channels)
          (all these DCP instructions are still queued locally; you can flush them to the FPGA via '!' at the
          last dcp instruction or dcp flush. dcp start also flushes.)
dcp flush                              (dcp start if controlled via USB console)
```

4.5.2 Phase jump by π after 2 seconds

The next example sets both outputs to 200 MHz, then waits 2 seconds and then changes the phase of one output by π.

```
dcp dds_reset
dcp 0 spi:stp0=0x3fff000033333333      ch 0, set freq. to 200 MHz, phase to 0 deg.
dcp 1 spi:stp0=0x3fff000033333333      ch 1, set freq. to 200 MHz, phase to 0 deg.
dcp update:u                           update both AD9910 to make the STPs effective
dcp 0 spi:stp0=0x3fff7fff33333333      ch 0, set freq. to 200 MHz, phase to 180 deg.
dcp 1 spi:stp0=0x3fff000033333333      ch 1, set freq. to 200 MHz, phase to 0 deg.
          (Note: The new STP0 has now been loaded into the AD9910 already but is not yet effective,
          because the IO_UPDATE has not yet been triggered.)
dcp wait:2000000:                      wait about 2 seconds
dcp update:u                           finally, update both channels to flip the phase
```

	(Note: Our small program of DCP instructions is now complete. We can now start the DDS Command Processor (DCP). Nothing will happen at the RF outputs before we start the DCP.)
`dcp flush`	(`dcp start` if controlled via USB console)

4.5.3 Using the mirror frequency

The next example sets both outputs to 200 MHz by using the direct frequency for the channel 0 and the mirror frequency for channel 1.

`dcp dds_reset`	
`dcp 0 spi:stp0=0x3fff000033333333`	normal frequency (200 MHz)
`dcp 1 spi:stp0=0x3fff0000cccccccd`	mirror frequency (800 MHz)
`dcp update:u`	
`dcp flush`	(`dcp start` if controlled via USB console)

4.5.4 Wait for DIO inputs; externally triggering DDS changes

The next example shows how to use the `wait` instruction to **trigger actions from a DIO input**. For demonstration, it is required to send a signal into the the DIO A1. This can be done by by hooking up a signal generator set to square wave output at TTL levels and 1 Hz frequency. The below example will start at 10 MHz and then switch to 20 MHz after the first rising edge of the DIO A1 input, and then switch to 30 MHz with half amplitude after the second rising edge of the DIO A1 input. For a list of events, see Table 4.8 (page 53).

`dcp dds_reset`	**All this is only done on channel 0.**
`dcp 0 spi:CFR2=0x1000080`	set CFR2 to matched latency and ASF from STP
`dcp 0 spi:stp0=0x3fff0000028f5c29`	set STP0 to 10 MHz, full amplitude
`dcp 0 update:u`	flush settings to AD9910, 10 MHz now at RF output
`dcp 0 spi:stp0=0x3fff0000051eb852`	set STP0 to 20 MHz, full amplitude
`dcp 0 wait::6`	wait for rising edge on DIO A1 input (event 6)
`dcp 0 update:u`	IO_UPDATE the AD9910, 20 MHz now at RF output
`dcp 0 spi:stp0=0x1fff000007ae147b`	set STP0 to 30 MHz, half amplitude
`dcp 0 wait::DIO_IN_A1_RISING`	wait for rising edge on DIO A1 input (same as `wait::6`)
`dcp 0 update:u`	IO_UPDATE the AD9910, 30 MHz, half ampl. at RF out
`dcp flush`	(`dcp start` if controlled via USB console)

4.5.5 Frequency sweep over the whole output frequency range

This is mostly interesting for assessing the amplitude stability. It performs a continuous frequency sweep over the whole frequency range 300 kHz to over 450 MHz, up and down.

`dcp dds_reset`	
`dcp spi:ASF=0xfffc`	set desired amplitude 0xfffc is max
`dcp spi:CFR1=0x00400200`	use 0x00000200 to disable the inverse sinc filter
`dcp spi:CFR2=0x00060080`	
`dcp spi:DRL=0x766666660013a92a`	frequency range
`dcp spi:DRSS=0x0000001a0000000d`	step size

`dcp spi:DRR=0x00060006`	ramp rate
`dcp spi:CFR2=0x000e0080`	enable the ramp/sweep generator, set both no-dwell bits
`dcp update:u+d`	
`dcp flush`	(`dcp start` if controlled via USB console)

4.5.6 Frequency ramp up, then down, then again up

The following example code performs the same **frequency ramps** on both channels. It starts at 20 MHz, ramps up to 30 MHz in about 2 seconds and stays there for about 1 second before ramping down to 20 MHz twice as fast and then, after 2 seconds sweeps upwards to 100 MHz.

`dcp dds_reset`	**All the following is done by both channels simultaneously:**
`dcp spi:CFR2=0x80`	set matched latency (not needed) and ramp destination frequency
`dcp spi:DRL=0x07ae147b051eb852`	ramp limits 20 MHz (low) and 30 MHz (high)
`dcp spi:DRSS=0x0000001a0000000d`	ramp step size to about 6 Hz down and 3 Hz up
`dcp spi:DRR=0x00960096`	ramp rate 150 (up and down)
`dcp spi:CFR2=0x80080`	enable the ramp generator
`dcp update:u+d`	do IO_UPDATE, set DRCTL HIGH to start upwards ramp
`dcp wait:3000000:`	wait for about 3 seconds for ramp to complete
`dcp update:-d`	set DRCTL LOW to start downwards ramp
`dcp spi:DRL=0x1999999a051eb852`	set upper ramp limit to 100 MHz (effective at next IO_UPDATE)
`dcp spi:DRSS=0x0000001a00000081`	ramp step size to about 30 Hz up
`dcp wait:2000000:`	wait 2 seconds for ramp to complete
`dcp update:u+d`	set DRCTL HIGH again to sweep up to 100 MHz
`dcp flush`	(`dcp start` if controlled via USB console)

Below is a **modified frequency ramp** example from the above one. Both channels to a sweep from 20 MHz to 100 MHz. The waveform has the amplitude; this is set in the single tone profile (requiring bit 24 in CFR2). Both DCPs then wait until the ramp is complete by monitoring the DROVER signal from the AD9910 (event 4). Once the ramp is over, channel 0 switches to the single tone profile with full amplitude while channel 1 stays in ramp mode with half amplitude. One can see that both channels are still phase aligned.

`dcp dds_reset`	
`dcp spi:STP0=0x1fff0000051eb852`	set 30 MHz, HALF amplitude
`dcp spi:CFR2=0x1000080`	set single tone ASF bit and matched latency
`dcp update:u`	update AD9910
`dcp wait:500000:`	wait half a second
`dcp spi:DRL=0x1999999a051eb852`	prepare ramp limits to 30 MHz and 100 MHz
`dcp spi:DRSS=0x0000001a00000008`	prepare ramp step sizes
`dcp spi:DRR=0x00080008`	prepare ramp rate
`dcp spi:CFR2=0x1080080`	enable ramp
`dcp update:u+d`	IO_UPDATE to start upwards ramp (DRCTL HIGH)
`dcp 0 spi:STP0=0x3fff00001999999a`	channel 0: set STP at 100 MHz, full amplitude
`dcp 0 spi:CFR2=0x1000080`	channel 0: disable ramp
`dcp wait::DROVER`	wait indefinitely for ramp to complete (event 35)
`dcp 0 update:u`	IO_UPDATE channel 0 to transition to STP0 (no glitch)
`dcp flush`	(`dcp start` if controlled via USB console)

4.5.7 Phase ramp

The following example will drive a **phase ramp**. Both outputs start with a 30 MHz sine wave signal that is completely *in* phase (i.e. no phase difference). After half a second, the channel 0 makes a smooth phase sweep by 180 degrees.

```
dcp dds_reset
dcp spi:STP0=0x3fff0000051eb852        both channels: set STP0 to 30 MHz, full amplitude
dcp spi:CFR2=0x1000080                 set CFR2 to matched latency and ASF from STP
dcp update:u                           update; this makes the above appear at the RF outputs
dcp wait:500000:                       wait half a second
dcp 0 spi:DRR=0x00960096               channel 0: prepare phase ramp: ramp rate...
dcp 0 spi:DRSS=0x0000020000000200      channel 0: ...ramp step size and...
dcp 0 spi:drl=0x7fffffff00000000       channel 0: ...ramp limits 0 to 180 degrees
dcp 0 spi:CFR2=0x1180080               channel 0: enable ramp generator (destination: phase)
dcp 0 update:u+d                       update channel 0, DRCTL HIGH (upwards ramp)
dcp flush                              (dcp start if controlled via USB console)
```

4.5.8 Synchronizing two channels on the same slot module

Here's an example to show **how to synchronize two channels** on the same FlexDDS RF generator module so that they are phase aligned. The basic trick is to set the autoclear phase bit in CFR1 and then issue an UPDATE *simultanouesly* to both channels to be synchronized. We assume that both DCPs have already executed a complex set of instructions. Because both DCPs are running independent of each other, the commands that each of them has executed in the past means that they are probably not executing the next commands synrchroniously. So, if we simply queue a `dcp update:u`, the two DCPs will generally execute them not at the same time. Hence, we need both DCPs to wait for some external trigger. This can be a "ramp finished" event but most commonly one would use an external trigger via one of the DIOs or the backplane trigger bus. In this example, we use the rising edge of the DIO A0 input which has to be applied by an external source.

```
dcp dds_reset
dcp stop                               stop both channel DCP
dcp 0 start                            we start the first DCP
dcp 1 start                            and then the second DCP
          This stop/start preamble is only here for demonstration purpose. Because we are starting the
          DCPs not at the same time, they will not run simultaneously and the two outputs will show a
          random phase relationship each time we execute this caused by random latency in the USB or
          network protocol. After the DIO A0 rising edge event, they will be synchronized.
dcp spi:STP0=0x3fff0000051eb852        both channels: set STP0 to 30 MHz, full amplitude
dcp spi:CFR2=0x1000080                 set CFR2 to matched latency and ASF from STP
dcp update:u                           update; this makes the above appear at the RF outputs
dcp spi:CFR1=0x00402000                set autoclear phase bit (for next update)
dcp wait::DIO_IN_A0_RISING             both DCPs wait for rising edge on DIO A0
dcp update:u                           IO UPDATE both channels simultaneously (re-starts phase accu)
dcp flush                              make sure we flush the local FIFOs
```

The method above also works across different slots in the same rack by choosing one of the two backplane triggers (e.g. `BP_TRIG_A`) instead of `DIO_IN_A0_RISING`.

4.5.9 Amplitude ramp followed by a frequency ramp

Here's an example for a **amplitude ramp followed by a frequency ramp**. The task is as follows:

1. Start at 30 MHz and full amplitude (initial state). (This is just for visualization, one could also start with amplitude 0.)
2. Wait for a rising edge trigger on DIO B1.
3. Perform a linear amplitude ramp from zero to 50% amplitude.
4. Wait for a second rising edge trigger on DIO B1.
5. Perform a frequency ramp from the initial 30 MHz to 50 MHz.

We do all this on channel 1 (RF:B).

```
dcp dds_reset
dcp 1 spi:STP0=0x3fff0000051eb852     set STP0 to 30 MHz, full amplitude (initial state)
dcp 1 spi:CFR2=0x01000080             set CFR2 to matched latency and ASF from STP
dcp 1 update:u                        update; this makes the above appear at the RF outputs
dcp 1 wait::DIO_IN_B1_RISING          wait for DIO B1 rising edge trigger
dcp 1 spi:DRR=0x00960096              prepare amplitude ramp: ramp rate...
dcp 1 spi:DRSS=0x0000020000000200     ...amp step size and ...
dcp 1 spi:drl=0x7fffffff00000000      ...ramp limits 0 to 50% amplitude
dcp 1 spi:CFR2=0x01280080             enable ramp generator (destination: amplitude)
dcp 1 update:u+d                      update channel 1, set DRCTL HIGH
          Now, the amplitude ramp is running and slowly increases the amplitude from 0 to 50%. Preload
          registers for what we need once the amplitude ramp has completed. This means, we already preload
          the registers with the frequency ramp that we'll do next:
dcp 1 spi:CFR2=0x01000080             disable ramp generator
dcp 1 spi:STP0=0x1fff0000051eb852     set STP0 to 50% amplitude (final ramp amplitude)
dcp 1 spi:DRL=0x0ccccccd051eb852      set ramp limits 30 MHz (low) and 50 MHz (high)
dcp 1 spi:DRSS=0x0000001a0000000d     ramp step size to about 6 Hz down and 3 Hz up
dcp 1 spi:DRR=0x00960096              ramp rate 150 (up and down)
dcp 1 wait:1:                         wait 1 µs to ensure ampl. ramp has started and DROVER is LOW
dcp 1 wait::DROVER                    wait until amplitude ramp has completed (DROVER is HIGH)
dcp 1 update:u-d                      update; preloaded state takes effect, also take DRCTL low!
          Now, the ramp generator is disabled, the 50% amplitude is taken from STP0 and the new frequency
          ramp limits are configured in ramp generator.
dcp 1 spi:CFR2=0x01080080             (preload) enable ramp generator (destination: frequency)
dcp 1 wait::DIO_IN_B1_RISING          wait for DIO B1 rising edge trigger
dcp 1 update:u+d                      update; this starts the frequency ramp
          Preload registers for what we need once the frequency ramp has completed:
dcp 1 spi:CFR2=0x01000080             disable ramp generator
dcp 1 spi:STP0=0x1fff00000ccccccd     preload STP0 with 50% amplitude and 50 MHz (freq ramp final)
dcp 1 wait:1:                         wait 1 µs to ensure freq. ramp has started and DROVER is LOW
dcp 1 wait::DROVER                    wait until frequency ramp has completed (DROVER is HIGH)
dcp 1 update:u-d                      update; preloaded state takes effect (ramp disabled, amplitude and
                                      final ramp frequency from STP0); also take DRCTL low
dcp flush                             (dcp start if controlled via USB console)
```

Note: If you don't have an external trigger, you can use `dcp 1 wait:500000:` instead of `dcp 1 wait::DIO_IN_B1_RISING`.

Note also: We always pre-load the register contents as early as possible to keep trigger latency low. Of

course, one could also load the new ramp limits after the `wait::DROVER` but that would increase the delay between the time the ramp is over and the time we can accept the next DIO trigger.

4.5.10 A more complex real-world task with external trigger, amplitude and frequency ramp

The following example is a **more complex real-world task**:

1. Device sits at 7 MHz at -34 dBm output power and waits for a trigger from a TTL line (DIO A1).
2. Upon receipt of the trigger, the device ramps amplitude up from -34 dBm to -5 dBm over 3 seconds (at 7 MHz).
3. Then the device ramps the frequency from 7 MHz to 7.05 MHz over 5 seconds.
4. Then device jumps phase by 90 deg.
5. Then device continues to run at 7.05 MHz for 1 second.
6. Device turns the output off.

The durations (3 and 5 seconds) are chosen so that it is easy to follow on an oscilloscope or spectrum analyzer. One can make the ramps 3 and 5 milliseconds, respectively by (a) multiplying the ramp step sizes by a factor of 1000, or by (b) dividing the ramp rate registers by 1000, or by (c) a combination of these two.

We are using channel 0. The frequency of 7 MHz corresponds to a frequency tuning word (FTW) of

$$\mathrm{FTW}(f) = \frac{f \cdot 2^{32}}{1\,\mathrm{GHz}} = \frac{7 \cdot 10^6}{10^9} \cdot 2^{32} = 30064771 = \texttt{0x01cac083}$$

(This formula can be found in the AD9910 datasheet.)

Similarly, 7.05 MHz are 30279519 or `0x1ce075f`.

For the amplitude ramp in dBm, the output of the FlexDDS-HD has to be calibrated. This procedure only has to be done once. Also, the amplitude setting is fairly independent of the frequency so the calibration usually does *not* have to be repeated when changing the frequency.

For the FlexDDS-HD-AD9910 with the variable gain amplifier option, a calibration can be performed by switching on the RF output and tuning the RF leve DAC to a suitable value, see these commands on the USB console:

`ftest f=81M`	(turn on RF output)
`dds 0 rfgain=`*VAL*	(set the power level in the amplifier for RF:A; see section 3.3.9)
`dds 1 rfgain=`*VAL*	(same for RF:B, if needed)

Adjust the *VAL* until the observed output has the desired power level. The *VAL* in range 0...65535 is then the calibration to be applied. Once the value has been found, you can store it permanently in the user configuration EEPROM, see USB console command `eeprom` in section 3.3.6.

Alternatively, if a fixed amplifier option is installed in the RF generator module, it is possible to tune the auxiliary level DAC inside the AD9910 by changing the `ADAC` register. On the USB console:

`ftest f=81M,auxdac=`*VAL*

Adjust the *VAL* until the desired power level is reached. You will need to add a DCP command to write that value into the AD9910 initially:

`dcp spi:ADAC=`*VAL*

The `ADAC` has limited adjustment range; if you need more range you can either use fixed external attenuators or scale the amplitude values written into the AD9910 or accordingly.

Depending on the installed amplifier option, a useful full scale calibration is often +10 dBm. However, in this example, as all power levels are below 0 dBm, we will calibrate the full scale to e.g. 2 dBm. (One would rather use 0 dBm in real life but having a non-zero value makes the formulas below easier to follow.)

So we have to set the FlexDDS-HD to full scale ampliude at our desired calibration frequency (e.g. 7 MHz):

`dcp dds_reset`	
`dcp 0 spi:stp0=0x3fff000001cac083`	set STP0 to 7 MHz (0x01cac083), full amplitude (0x3fff)
`dcp 0 update:u`	update the AD9910 so that the STP0 takes effect; this starts the 7 MHz output
`dcp flush`	(`dcp start` if controlled via USB console)

(The STP0 is the Single Tone Profile register 0, i.e. it specifies the amplitude, phase offset and frequency of the output.)

Once we have executed these 4 lines, output 0 will emit a 7 MHz sine wave on the main "RF Out 0" SMA output. We can connect that to a calibrated spectrum analyzer or RF power meter and turn the corresponding "Lvl" with a screw driver until the level is 2 dBm.

The amplitude part of the STP0 register (first 14 bits) scales linearly. Based on our calibration (+2 dBm = `0x3fff` = 16383), we can hence compute the necessary amplitude setting for any dBm value. This is called the ASF (amplitude scale factor):

$$\mathrm{ASF}(a) = 10^{(a-a_{\mathrm{full}})/20} \cdot (2^{14} - 1)$$

Here a is the desired amplitude in dBm and $a_{\mathrm{full}} = 2\,\mathrm{dBm}$ is the just calibrated full scale amplitude. In our example,

$$\mathrm{ASF}(-34\,\mathrm{dBm}) = 10^{(-34-2)/20} \cdot (2^{14} - 1) = 10^{-1.8} \cdot 16383 = 260 = \texttt{0x0104}$$

Similarly, $\mathrm{ASF}(-5\,\mathrm{dBm}) = 7318 = \texttt{0x1c96}$.

For the amplitude ramp, one can use the OSK generator or the ramp generator. Since the ramp is fairly long (100 ms), it is better to use the full ramp generator because the OSK generator can only make short ramps.

We need to set the amplitude ramp limits in the DRL register. This is a 64-bit register that contains the upper limit in the upper 32 bits and the lower limit in the lower 32 bits. Of each limit, only the highest 14 bits are used for the amplitude. Hence, for our two amplitudes, the DRL value is:

$$\mathrm{DRL} = \texttt{ASF}^{\mathrm{high}}_{\mathrm{14bits}}\ 0_{\mathrm{18bits}}\ \texttt{ASF}^{\mathrm{low}}_{\mathrm{14bits}}\ 0_{\mathrm{18bits}} = \texttt{0x72580000\_04100000}$$

(Note that these numbers can easily be computed by shifting the computed values from before by 2 bits to the left: $\texttt{0x1c96} \ll 2 = \texttt{0x7258}$, $\texttt{0x0104} \ll 2 = \texttt{0x0410}$.)

Similarly, for the frequency ramp, the DRL value is simply the two 32-bit FTW values joined together:

$$\mathrm{DRL} = \texttt{FTW}^{\mathrm{high}}_{\mathrm{32bits}}\ \texttt{FTW}^{\mathrm{low}}_{\mathrm{32bits}} = \texttt{0x01ce075f\_01cac083}$$

We now have to **compute the ramp speed** (i.e. ramp rate and step size). We have some freedom in this: We can either make a ramp with a few large steps or with many small steps. Both approaches have their pros and cons: Many small steps make for a smooth ramp but because the step size has finite precision the final ramp speed might significantly deviate from what we'd like to see. In contrast, a few large steps allow us to specify the step size very precisely but the ramp will look like a staircase. Usually, theres a good compromize in between.

The AD9910 datasheet specifies the ramp rate formula as follows ($f_{\mathrm{SYSCLK}} = 1\,\mathrm{GHz}$):

$$\Delta t = \frac{4 \cdot \mathrm{DRR}_{up,down}}{f_{\mathrm{SYSCLK}}} = \mathrm{DRR}_{up,down} \cdot 4\,\mathrm{ns}$$

The AD9910 datasheet also specifies the amplitude step size as:

$$AmplitudeStep = \frac{\mathrm{DRSS}_{up,down}}{2^{32}} I_{FS}$$

where I_{FS} is the electrical full scale amplitude.

It is important to understand that after each time Δt, the step size gets added to the ramp accumulator. Hence, for a ramp with n steps, the total duration of the ramp will be $\Delta T = n \cdot \Delta t$ and the accumulator will have performed n-many steps of size DRSS. Hence,

$$\mathrm{DRSS} = 2^{32} \frac{AmplitudeStep}{I_{FS}} = 2^{32} \frac{\mathrm{ASF}_{\mathrm{end}} - \mathrm{ASF}_{\mathrm{start}}}{2^{14} \cdot n}$$

We know that we need to take 3 seconds ($\Delta T = 3\,\mathrm{s}$) to ramp the ASF from $\mathrm{ASF}_{\mathrm{start}} = 260$ to $\mathrm{ASF}_{\mathrm{end}} = 7318$ as computed above.

Let's say we want to make $n = 100\,000$ steps. With the formulas above we obtain:

$$\mathrm{DRR} = \frac{\Delta t}{4\,\mathrm{ns}} = \frac{\Delta T}{n \cdot 4\,\mathrm{ns}} = \frac{3\,\mathrm{s}}{100\,000 \cdot 4\,\mathrm{ns}} = 7\,500 = \texttt{0x1d4c}$$

and

$$\mathrm{DRSS} = 2^{32} \frac{\mathrm{ASF}_{\mathrm{end}} - \mathrm{ASF}_{\mathrm{start}}}{2^{14} \cdot n} = 2^{32-14} \frac{7318 - 260}{100\,000} = 18\,502 = \texttt{0x00004846}$$

Similarly we could try with $n = 10\,000$ steps or $n = 10^6$ steps, the results are summarized in the following table:

n	DRSS	DRR
10^4	185 021	~~75 000~~
10^5	18 502	7 500
10^6	1 850	750
10^7	185	75

Note that the first line is technically impossible because the ramp rate is limited at $2^{16} - 1 = 65\,535$ but all others give very nice ramps, so we have a large freedom concerning the number of steps here.

In a similar fashion the frequency ramp parameters can be computed. Because the frequency change is rather small (50 kHz) and the ramp duration rather long (5 seconds), choosing the best step size is a bit tricky this time. For the computation and reasons that will become clear later, let's choose $n = 21\,450$ steps:

$$\mathrm{DRR} = \frac{\Delta T}{n \cdot 4\,\mathrm{ns}} = \frac{5\,\mathrm{s}}{21\,450 \cdot 4\,\mathrm{ns}} = 58\,275 = \texttt{0xe3a3}$$

$$\mathrm{DRSS} = \frac{\mathrm{FTW}_{\mathrm{end}} - \mathrm{FTW}_{\mathrm{start}}}{n} = \frac{30279519 - 30064771}{21\,450} = 10.01 = \texttt{0x0000000a}$$

Here is what would happen for different number of steps:

n	DRSS	DRR
10^4	21.47	~~125 000~~
21 450	10.01	58 275
10^5	2.14	12 500
10^6	~~0.21~~	1 250

Note that the DRSS values are integers, the fractional values are only shown to illustrate the rounding error. Here we see that because of the long and shallow ramp, the ramp generator is close to its limits and the number of steps has to be chosen carefully: With too few steps, the ramp rate would overflow (first line). With too many steps, the step size gets very small (last two lines). A value of 0 disables ramping altogether but even very small values like 2.14 have a very big rounding error (DRSS=2, error 7%). Hence, the number 21 450 was chosen empirically to obtain a nice figure for the step size with very little rounding error (0.1%).

The necessary instructions to perform this task are listed below. All these instructions are fed into the FlexDDS-HD at the beginning. Once the last instruction (`dcp flush`) is received, we start executing the program. (In fact depending on flush settings it may start earlier but won't go past the first `wait`.)

`dcp dds_reset`	
`dcp 0 spi:CFR2=0x01000080`	set CFR2 to matched latency and ASF from STP
`dcp 0 spi:stp0=0x0104000001cac083`	set STP0 to 7 MHz (0x01cac083), −34 dBm amplitude (0x0104)
`dcp 0 update:u-d`	update the AD9910 so that the STP0 takes effect; this starts the 7 MHz output (-d sets DRCTL to LOW to be sure)
	we start pre-loading the next settings into the AD9910:
`dcp 0 spi:DRL=0x72580000_04100000`	set amplitude ramp limits (`0x1c96` ≪ 2 = `0x7258`, see above)
`dcp 0 spi:DRSS=0x00004846_00004846`	set ramp step size (see above)
`dcp 0 spi:DRR=0x1d4c_1d4c`	set ramp rate (see above)
`dcp 0 spi:CFR2=0x00280080`	enable ramp generator with destination amplitude
`dcp 0 wait::DIO_IN_A1_RISING`	wait for rising edge TTL trigger on DIO A1
`dcp 0 update:u+d`	immediately update after the trigger (activate pre-loaded settings) and also set DRCTL to HIGH (+d) to start the upwards ramp
	the ramp is now running and we pre-load the next settings:
`dcp 0 spi:CFR2=0x01000080`	disable the ramp again and enable ASF from STP again
`dcp 0 spi:stp0=0x1c96000001cac083`	tune STP0 to match current output (7 MHz, −5 dBm)
`dcp 0 wait::DROVER`	wait until the amplitude ramp completes
`dcp 0 update:u-d`	and immediately activate the pre-loaded settings (output stays unchanged)
	start configuring the frequency ramp
`dcp 0 spi:DRL=0x01ce075f01cac083`	set the frequency ramp limits (see above)
`dcp 0 spi:DRSS=0x0000000a0000000a`	set ramp step size (see above)
`dcp 0 spi:DRR=0xe3a3e3a3`	set ramp rate (see above)
`dcp 0 spi:CFR2=0x01080080`	enable ramp generator with destination frequency
`dcp 0 update:u+d`	update the AD9910, start the ramp via +d (DRCTL set HIGH)
	immediately pre-load the new settings while ramp is running:
`dcp 0 spi:CFR2=0x01000080`	disable the ramp again
`dcp 0 spi:stp0=0x1c96800001ce075f`	set the STP0 to the settings after the ramp but with a phase offset word of π (i.e. POW = `0x8000`)
`dcp 0 wait::DROVER`	wait until the frequency ramp is over
`dcp 0 update:u-d`	update to the new settings to perform the phase jump
	pre-load the next settings:
`dcp 0 spi:stp0=0x0000800001ce075f`	turn output off by setting the ASF to zero
`dcp 0 wait:1000000:`	wait for 1 second
`dcp 0 update:u`	update; this will switch off the output
`dcp flush`	(`dcp start` if controlled via USB console)

4.5.11 Working with the Bugs in the Ramp Generator of the AD9910

Note: Bugs in the AD9910 synthesizer ramp generator

The AD9910 chip unfortunately has bugs in its ramp generator which will not be fixed by Analog Devices and which are unfortunately not documented. (Actually, they were discussed in a forum in AD's Engineer Zone but all old posts were deleted when they switched to their shiny new webpage.) For instance, when programming multiple upwards ramps in succession, the first one will play nicely while the remaining ones will simply jump to the end frequency rather than sweeping. Please refer to the following examples that work around those bugs to achieve ramps that actually work.

Here are a couple of frequency ramp examples that work around bugs in the AD9910.

Ramp up-then-down: Normal frequency. If you need an upwards ramp followed by a downwards ramp, you can do this straighforward.

```
dcp dds_reset
dcp spi:DRL=0x07ae147b051eb852      set ramp limits
dcp spi:DRSS=0x0000001a0000001a     set ramp step size
dcp spi:DRR=0x00960096              set ramp rate
dcp spi:CFR2=0x80080                enable ramp
dcp update:u+d                      do IO_UPDATE, set DRCTL HIGH to start upwards ramp
dcp wait:2000000:                   wait for the ramp to sweep the frequency
dcp update:-d                       change direction to downwards, DRCTL LOW
dcp flush                           (dcp start if controlled via USB console)
```

Ramp down-then-up: Mirror frequency. If you try the downwards ramp followed by an upwards ramp in the straighforward way, this will not work. However, one can use the mirror frequency so that to the ramp generator it looks like two 'upwards' ramps in FTW but these actually go downwards in frequency.

```
dcp dds_reset
dcp spi:DRL=0xfae147aef851eb85      set ramp limits (mirror frequency)
dcp spi:DRSS=0x0000001a0000001a     set ramp step size
dcp spi:DRR=0x00960096              set ramp rate
dcp spi:CFR2=0x80080
dcp update:u+d                      do IO_UPDATE, set DRCTL HIGH to start ramp
dcp wait:2000000:                   wait for the ramp to sweep the frequency
dcp update:-d                       change direction, DRCTL LOW
dcp flush                           (dcp start if controlled via USB console)
```

Note, however, that switching from the frequency to the mirror frequency will not be phase continuous, so this might not always be an option.

Ramp down, normal frequency

```
dcp dds_reset
dcp spi:DRL=0x07ae147b051eb852      set ramp limits (normal frequency)
dcp spi:DRSS=0x0000001a7fffffff
dcp spi:DRR=0x00960000
dcp spi:CFR2=0x80080
dcp update:u+d                      we start with taking DRCTL HIGH
dcp wait:2000000:                   this delay can be left out
```

```
dcp update:-d                    take DRCTL LOW for downwards ramp
dcp flush                        (dcp start if controlled via USB console)
```

Ramp up, then up: Perform intermitted mini downwards ramps

If you need multiple upwards ramps, the idea is to wait until the first ramp is over and then perform a very tiny downwards ramp (by only 0.23 Hz or one FTW step) before taking on the next upwards ramp.

In the following example, we would like to perform a frequency ramp from 5 to 6 MHz in 1 ms, wait for a trigger and then perform another frequency ramp from 20 to 21 MHz in 5 ms and then another one again from 5 to 6 MHz in 193 ms. As for the computation of the step size and ramp rate, please see section 4.5.10 on page 82.

In this example, we trigger the ramps from the backplane via trigger line A (`BP_TRIG_A`) but we could as well use any DIO. To aid in debugging, we use DIO A0 as output for the DROVER pin and DIO A1 as output for the trigger signal from the backplane.

```
dcp dds_reset                          everything here on RF channel 0 (RF:A)
dcp 0 wr:CFG_DIO_0=0b1_00_0100011      set DIO A0 as output for the DROVER pin (35)
dcp 0 wr:CFG_DIO_1=0b1_00_0011011      set DIO A1 as output for the BP_TRIG_A (27)

dcp 0 spi:CFR2=0x1000080               set single tone ASF and matched latency
dcp 0 spi:stp0=0x3fff00000147ae14      set initial frequency to 5 MHz and stay there
dcp 0 update:u                         make 5 MHz appear at the RF output

dcp 0 spi:DRL=0x0189374c_0147ae14      ramp limits for a sweep from 5 to 6 MHz
dcp 0 spi:DRSS=0x00000001_00000225     upwards step size to 128 Hz, downwards to 0.23 Hz
dcp 0 spi:DRR=0x0001_0020              ramp rate 32 (up) and 1 (down)
dcp 0 spi:CFR2=0x80080                 enable the ramp generator
dcp 0 wait::BP_TRIG_A                  wait for trigger (can also use DIO_IN_A0_RISING)
dcp 0 update:u+d                       update registers and set DRCTL HIGH to start upwards ramp
        Now, the upwards frequency ramp from 5 to 6 MHz runs. In the mean time we load the registers
        for an alibi downwards ramp by 0.23 Hz.
dcp 0 spi:DRL=0x0189374c_0189374b      downwards 6 MHz to "6-epsilon" MHz
dcp 0 wait::DROVER                     wait until the upwards ramp to 6 MHz is over...
dcp 0 update:u-d                       ...and immediately start our alibi ramp down 0.23 Hz
                                       by taking DRCTL LOW
        Now we can do the same thing again for a 20 to 21 MHz ramp:
dcp 0 spi:DRL=0x05604189_051eb852      set ramp limits 20 to 21 MHz
dcp 0 spi:DRSS=0x00000001_00000225     step size 128 Hz (up) and 0.23 Hz (down)
dcp 0 spi:DRR=0x000100a0               ramp rate 1 (down) and 160 (up), 5 times the speed as above
dcp 0 wait::BP_TRIG_A                  wait for trigger
dcp 0 update:u+d                       update registers and set DRCTL HIGH to start upwards ramp
        Now, the upwards frequency ramp 20 to 21 MHz runs. In the mean time we load the registers for
        an alibi downwards ramp by 0.23 Hz.
dcp 0 spi:DRL=0x05604189_05604188      downwards 23 MHz to "23-epsilon" MHz
dcp 0 wait::DROVER                     wait until the upwards ramp is over and then...
dcp 0 update:u-d                       ...update and take DRCTL LOW for the alibi down ramp
        Now we can do the same thing again for a 5 to 6 MHz ramp as above, albeit at this time the ramp
        is much slower:
dcp 0 spi:DRL=0x0189374c_0147ae14      5 to 6 MHz
dcp 0 spi:DRSS=0x00000001_0000000d     step 0.23 Hz down, 3 Hz up
dcp 0 spi:DRR=0x00010096               ramp rate 150 (up) and 1 (down)
dcp 0 wait::BP_TRIG_A                  wait for trigger
```

```
dcp 0 update:u+d                        start the slow upwards ramp 5 to 6 MHz
dcp 0 spi:DRL=0x0189374c_0189374b       again prepare an alibi downwards ramp
dcp 0 wait::DROVER                      wait for upwards ramp to complete
dcp 0 update:u-d                        do the alibi ramp
dcp flush                               (dcp start if controlled via USB console)
```

4.5.12 Real-world example: Wait for trigger, set 75 MHz, at next trigger ramp down, trigger again, then switch off

This is a real world example: We want to wait for a trigger on DIO B1, once it is received we switch on channel 1 at 75.5 MHz, then wait for the next trigger on DIO B1. Once received, we perform a frequency ramp to 83.0 MHz and finally after the next trigger switch off.

This can be repeated multiple times without reset in between. Two important steps are: We set the *downwards* ramp rate and step size to something very large so that it can immediately jump back to the start of the ramp. And we take the DRCTL pin low at the end of the ramp.

```
dcp dds_reset
dcp 1 spi:CFR1=0x400000
dcp 1 spi:CFR2=0x01000080
dcp 1 spi:STP0=0x0FFC00001353F7CF       set 75.5 MHz
dcp 1 wait::DIO_IN_B1_RISING            wait for trigger
dcp 1 update:u                          update to have 75.5 MHz at the output
dcp 1 wait:1h:
dcp 1 spi:DRL=0x153F7CEE1353F7CF        setup ramp from 75.5 to 83.0 MHz
dcp 1 spi:DRSS=0x10000000000006FF       set step size (downwards: very large steps!)
dcp 1 spi:DRR=0x0001008B                set ramp rate (downwards: very quick!)
dcp 1 spi:CFR2=0x01080080               enable ramp
dcp 1 wait::DIO_IN_B1_RISING            wait for trigger
dcp 1 update:u+d                        actually start frequency ramp after trigger
dcp 1 wait:1h:
dcp 1 wait::DROVER                      wait until ramp is over
dcp 1 spi:CFR1=0x400000
dcp 1 spi:CFR2=0x01000080               disable ramp
dcp 1 spi:STP0=0x0000000013333333       set amplitude to zero
dcp 1 wait::DIO_IN_B1_RISING            wait for trigger
dcp 1 update:u-d                        after trigger, update and take DRCTL low
dcp 1 wait:1h:
dcp flush                               (dcp start if controlled via USB console)
```

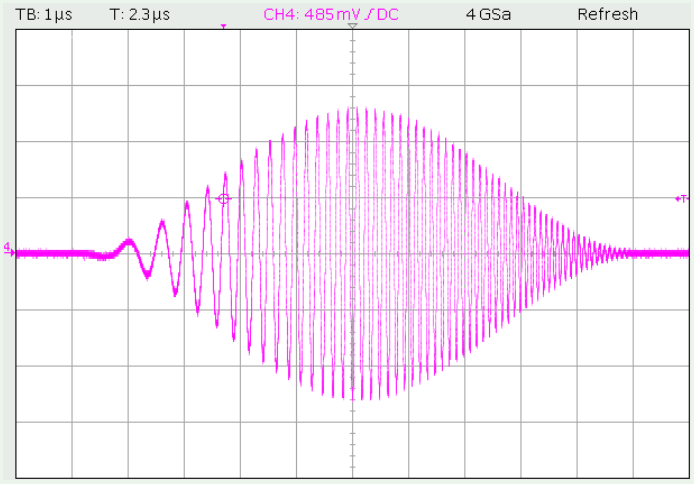
4.5.13 Example of a Hann shaped chirped pulse

This makes use of the SRAM modulation to shape the amplitude and the ramp generator to linearly sweep the frequency. This makes use of only 128 words (amplitude samples) of the total of 1024 available just to keep the code listing short (see (**) below). To use more samples within the same time, the step rate has to be reduced accordingly.

```
dcp dds_reset
dcp 0 spi:ASF=0                                  set zero amplitude (needed for OSK)
dcp 0 spi:CFR1=0x00412202                        enable OSK, inverse SINC filter, sine output, autoclear phase

dcp 0 spi:drss=0x000f4240000f4240                program the ramp into the DDS ramp generator
dcp 0 spi:drl=0x03d70a3d00000000                 0 to 15 MHz
dcp 0 spi:drr=50

dcp 0 spi:stp0=0x141fc0000001                    set up the RAM profile 0 (i.e. STP0):
          step rate: 0x14 = 20 (65535 max), start adr: 0, end adr: 127 (**) (max. is 1023) no dwell high:
          0, zero crossing: 0, mode control: 1 (UP)
          NOTE: We can use a different profile than profile 0 but if we do, we need to select the particular
          profile (using the FPGA) in order to upload the SRAM content.

dcp 0 update:=1p                                 switch profile forth and back; probably not needed for...
dcp 0 update:=0p                                 ... profile 0; done before updating CFR1 as changing the
          profile also acts as a trigger and would otherwise enable RAM playback too early.

dcp 0 spi:CFR1=0xc0416002                        disable OSK and enable RAM; we do this before storing
                                                 the RAM content but will not take effect until UPDATE
dcp 0 spi:CFR2=0x004e0cc0                        enable ramp generator (no-dwell low and high)

dcp 0 spi:RAMB=0:c                               begin storing the amplitude shape in SRAM
dcp 0 spi:RAM64C=0x00000000_00277872:c           first 2 words of the Hann shape
dcp 0 spi:RAM64C=0x009dc970_0162aa03:c           next 2 words of the Hann shape...
dcp 0 spi:RAM64C=0x0275a0c0_03d60411:c
dcp 0 spi:RAM64C=0x0582faa4_077b7bec:c
dcp 0 spi:RAM64C=0x09be50c3_0c4a142e:c
dcp 0 spi:RAM64C=0x0f1d3439_1235f2eb:c
dcp 0 spi:RAM64C=0x1592675b_19307edf:c           Note that the way the profile is set up, the playback in
dcp 0 spi:RAM64C=0x1d0dfe53_21288376:c           the AD9910 will be bottom-to-top and not top-to-bottom,
dcp 0 spi:RAM64C=0x257d8665_2a0a5b2d:c           so the listing is "reversed". For the Hann shape it does not
dcp 0 spi:RAM64C=0x2ecc336b_33c0200c:c           make a difference because it's symmetric.
dcp 0 spi:RAM64C=0x38e31318_3e31e19a:c
dcp 0 spi:RAM64C=0x43a9458f_4945dfeb:c
dcp 0 spi:RAM64C=0x4f043ab2_54e0cb13:c
dcp 0 spi:RAM64C=0x5ad7f3a1_60e60684:c
dcp 0 spi:RAM64C=0x670747c2_6d37ef90:c
dcp 0 spi:RAM64C=0x73742ca1_79b82682:c
dcp 0 spi:RAM64C=0x7ffffffe_8647d97b:c
dcp 0 spi:RAM64C=0x8c8bd35c_92c8106d:c
dcp 0 spi:RAM64C=0x98f8b83b_9f19f979:c
dcp 0 spi:RAM64C=0xa5280c5c_ab1f34ea:c
dcp 0 spi:RAM64C=0xb0fbc54b_b6ba2012:c
dcp 0 spi:RAM64C=0xbc56ba6e_c1ce1e63:c
dcp 0 spi:RAM64C=0xc71cece5_cc3fdff1:c
dcp 0 spi:RAM64C=0xd133cc92_d5f5a4d0:c
dcp 0 spi:RAM64C=0xda827998_ded77c87:c
dcp 0 spi:RAM64C=0xe2f201aa_e6cf811e:c
```

```
dcp 0 spi:RAM64C=0xea6d98a2_edca0d12:c
dcp 0 spi:RAM64C=0xf0e2cbc4_f3b5ebcf:c
dcp 0 spi:RAM64C=0xf641af3a_f8848411:c
dcp 0 spi:RAM64C=0xfa7d0559_fc29fbec:c
dcp 0 spi:RAM64C=0xfd8a5f3d_fe9d55fa:c
dcp 0 spi:RAM64C=0xff62368d_ffd8878b:c
dcp 0 spi:RAM64C=0xfffffffe_ffd8878b:c
dcp 0 spi:RAM64C=0xff62368d_fe9d55fa:c
dcp 0 spi:RAM64C=0xfd8a5f3d_fc29fbec:c
dcp 0 spi:RAM64C=0xfa7d0559_f8848411:c
dcp 0 spi:RAM64C=0xf641af3a_f3b5ebcf:c
dcp 0 spi:RAM64C=0xf0e2cbc4_edca0d12:c
dcp 0 spi:RAM64C=0xea6d98a2_e6cf811e:c
dcp 0 spi:RAM64C=0xe2f201aa_ded77c87:c
dcp 0 spi:RAM64C=0xda827998_d5f5a4d0:c
dcp 0 spi:RAM64C=0xd133cc92_cc3fdff1:c
dcp 0 spi:RAM64C=0xc71cece5_c1ce1e63:c
dcp 0 spi:RAM64C=0xbc56ba6e_b6ba2012:c
dcp 0 spi:RAM64C=0xb0fbc54b_ab1f34ea:c
dcp 0 spi:RAM64C=0xa5280c5c_9f19f979:c
dcp 0 spi:RAM64C=0x98f8b83b_92c8106d:c
dcp 0 spi:RAM64C=0x8c8bd35c_8647d97b:c
dcp 0 spi:RAM64C=0x7fffffff_79b82682:c
dcp 0 spi:RAM64C=0x73742ca1_6d37ef90:c
dcp 0 spi:RAM64C=0x670747c2_60e60684:c
dcp 0 spi:RAM64C=0x5ad7f3a1_54e0cb13:c
dcp 0 spi:RAM64C=0x4f043ab2_4945dfeb:c
dcp 0 spi:RAM64C=0x43a9458f_3e31e19a:c
dcp 0 spi:RAM64C=0x38e31318_33c0200c:c
dcp 0 spi:RAM64C=0x2ecc336b_2a0a5b2d:c
dcp 0 spi:RAM64C=0x257d8665_21288376:c
dcp 0 spi:RAM64C=0x1d0dfe53_19307edf:c
dcp 0 spi:RAM64C=0x1592675b_1235f2eb:c
dcp 0 spi:RAM64C=0x0f1d3439_0c4a142e:c
dcp 0 spi:RAM64C=0x09be50c3_077b7bec:c
dcp 0 spi:RAM64C=0x0582faa4_03d60411:c
dcp 0 spi:RAM64C=0x0275a0c0_0162aa03:c
dcp 0 spi:RAM64E=0x009dc970_00277872     last 2 words to store in SRAM (END); no ":c" at the end
dcp 0 update:u                           UPDATE to start ramp and SRAM playback
dcp flush                                (dcp start if controlled via USB console)
```

The final waveform looks like this:



Side note: Pulse **without chirp:** If you'd like to generate a shaped pulse with *constant* frequency rather than a chrip, all you have to do is set the `FTW=` register to the desired frequency and not enable the ramp generator. You can also remove the writes to the ramp registers (`DRSS`, `DRL`, `DRR`). Hence, for constant frequency, the above script would start like this:

```
dcp dds_reset
dcp 0 spi:ASF=0
dcp 0 spi:CFR1=0x00412202
dcp 0 spi:FTW=0x147ae148          set constant frequency 82 MHz
dcp 0 spi:stp0=0x141fc0000001
dcp 0 update:=1p
```

```
dcp 0 update:=0p
dcp 0 spi:CFR1=0xc0416002
dcp 0 spi:CFR2=0x00_40_0c_c0      do NOT enable ramp generator
dcp 0 spi:RAMB=0:c
...
```

After having executed the above script, additional Hann shaped chirped pulses can be generated without uploading all the RAM content again. For example by executing the following program after the previous one (without resetting the DDS), you can generate 4 additional pulses:

```
dcp 0 spi:ASF=0                     copied from above...
dcp 0 spi:CFR1=0x00412202
dcp 0 spi:drss=0x000f4240000f4240
dcp 0 spi:drl=0x03d70a3d00000000
dcp 0 spi:drr=50
dcp 0 spi:stp0=0x141fc0000001
dcp 0 update:=1p
dcp 0 update:=0p
dcp 0 wait:1000000:
dcp 0 spi:CFR1=0xc0416002
dcp 0 spi:CFR2=0x004e0cc0
dcp 0 update:u                      ...until here, first puse generated.
dcp 0 wait:1000000:                 wait for 1 second
dcp 0 update:=1p-d                  switch profile and DRCTL forth...
dcp 0 update:=0p+d                  ...and back again to trigger next pulse
dcp 0 wait:1000000:                 wait for 1 second
dcp 0 update:=1p+d                  switch profile and DRCTL forth...
dcp 0 update:=0p-d                  ...and back again to trigger next pulse
dcp 0 wait:1000000:                 and so on...
dcp 0 update:=1p-d                  note that the DRCTL is switched alternatingly
dcp 0 update:=0p+d
dcp flush                           (dcp start if controlled via USB console)
```

This could be made externally triggered by using a different wait statement but in this special case there's the opportunity to make the Hann shaped pulse externally triggered inifitely often: We can route the DIO A0 input to the PROFILE0 pin (inverted) and also route it to the DRCTL pin:

```
dcp 0 spi:ASF=0                     most of this is copied from above...
dcp 0 spi:CFR1=0x00412202
dcp 0 spi:drss=0x700f4240000f4240   here we make the downwards ramp step size huge
dcp 0 spi:drl=0x03d70a3d00000000
dcp 0 spi:drr=50
dcp 0 spi:stp0=0x141fc0000001
dcp 0 update:=1p                    this avoids a glitch at the first pulse
dcp 0 update:=0p
dcp 0 spi:CFR1=0xc0416002
dcp 0 spi:CFR2=0x004c0cc0           here we set no-dwell only for HIGH
dcp 0 wr:CFG_DRCTL=0b010_0_000_00101      route DIO A0 to DRCTL
dcp 0 wr:CFG_PROFILE=0b000_000_010_001    route DIO A0 to PROFILE0, inverted
dcp flush                           (dcp start if controlled via USB console)
```

The above example assumes that you have previously programmed the RAM in the AD9910 (e.g. with the

first example code with the RAM64C= commands) and will **generate a Hann shaped pulse on each rising edge of the DIO A0 input**.

For falling edge triggered pulses, you need to invert both DRCTL and PROFILE0.

4.5.14 Analog frequency modulation with digitally controlled amplitude and phase

This example requires a FlexDDS-HD-AD9910-ADC, i.e. with the ADC option, so that it provides the AIn:A,B analog inputs on the frontpanel.

The following example will perform an analog frequency modulation based on the analog input Ch 0 and at the same time allows to load 8 different amplitude and phase settings from the 8 profile registers via the levels on 3 frontpanel DIOs.

To test this example, hook up an oscilloscope on the RF:A output, supply a 1 Hz sine wave with 2 V_{pp} to analog input 0 and input a TTL rectangle signal into DIOs A0, A1 and B0. These 3 digital inputs inputs correspond to the 3 profile select bits.

It is important to "enable amplitude scale factors from single tone profiles" (CFR2 bit 24) and disable OSK.

```
dcp dds_reset
dcp 0 spi:FTW=0x266663e2                          set the base FTW for analog modulation
dcp 0 spi:stp0=0x3000_0000_266663e2               amplitude, phase and frequency for the...
dcp 0 spi:stp1=0x2a00_2000_166663e2               ...single tone profiles 0 to 7, note...
dcp 0 spi:stp2=0x2600_4000_166663e2               ...that the FTW part is irrelevant here as...
dcp 0 spi:stp3=0x2200_6000_166663e2               ...it will not go into effect
dcp 0 spi:stp4=0x1c00_8000_166663e2               we set 8 different amplitudes and...
dcp 0 spi:stp5=0x1800_a000_166663e2               ...8 different phases
dcp 0 spi:stp6=0x1400_c000_166663e2
dcp 0 spi:stp7=0x1000_e000_166663e2
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp 0 spi:CFR2=0x01000019                         enable parallel port and FM gain
dcp 0 wr:AM_S0=0x1000                             convert 2 Vpp into 16 bit full scale
dcp 0 wr:AM_O0=0x8000                             create output starting at 0
dcp 0 wr:AM_CFG=0x2000_0002                       select analog frequency modulation
dcp 0 update:u
dcp 0 wr:CFG_PROFILE=0b011_010_010_000            DIO A0, A1, B0 passthrough to profiles
dcp flush                                         (dcp start if controlled via USB console)
```

Chapter 5

Errata: Known Bugs and Limitations

No reasonable complex device exists without errata.

5.1 Slow DCP instruction transmission to slot modules

In all current versions (at least up to version 1.2), the DCP instructions are not transmitted from the main control module directly into the FPGA of the RF generator modules but instead are sent over the USB into the microcontroller of the RF generator module which, in turn, transmits it into the FPGA. This limits the instruction throughput. This shortcoming will be eliminated in a future firmware release.

5.2 Control interface responses are not easily machine readable

This will be resolved soon. The responses on the control interface will be made easier to parse on a machine.

5.3 No synchronization between multiple modules in a rack

The synchronization between multiple modules in a rack will be implemented soon.

5.4 No lookup table available in analog modulation mode

As an alternative to using a linear transform, you will be able to use a lookup table with linear interpolation. This will be implemented in the future. This feature will then also be made available on the FlexDDS-NG.