FlexDDS-NG

(Rev. r4.5, August 2, 2021)

Flexible Multi-Channel Phase-Coherent RF Source

▷ FlexDDS-NG DUAL

- 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
- 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
- External 10 MHz input

FlexDDS-NG DUAL
Dual Channel Standalone RF Generator

- 3 programmable digital IOs for trigger, fast on/off, ramp control, ...
- USB computer interface (Windows, Linux)

▷ FlexDDS-NG Rack

- Up to 6 slots, each one can be fitted with a dual channel RF generator (functionally equivalent to the FlexDDS-NG DUAL above)
- Up to 12 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)
- GBit Ethernet: Connect anywhere in the lab, no USB cables, no special OS drivers
- Global trigger inputs that affect all slots simultaneously

FlexDDS-NG Rack
Up to 12 RF Generator Channels

- External 10 MHz input and output
- Extensible: Add slots later as needed

General Description

FlexDDS-NG is a multi-channel phase-coherent RF source. 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-NG. Each time a (real-time asynchronious) trigger input is activated, FlexDDS-Rack 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 FlexDDS 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.

Contents

Chapter 1

Basic Operation of the Dual Channel AD9910 RF Generator

This description applies to both the FlexDDS-NG DUAL as well as each individual dual channel AD9910 RF generator slot of a FlexDDS-NG Rack.

The dual channel AD9910 RF generator employs 2 independent phase coherent RF synthesizers attached to a single FPGA. The actual RF signal synthesis is performed via 2 Analog Devices AD9910 DDS synthesizer chips clocked at 1 GHz.

Figure 1.1: The FlexDDS-NG DUAL (left) and the dual-channel AD9910 RF generator slots (right) of the FlexDDS-NG Rack are functionally equivalent.

Inside the FPGA, each RF channel has one **DDS command processor** (**DCP**). The DCP is responsible for controlling the AD9910 DDS synthesizer 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. Typically, a small "program" made of DCP instructions for each RF output channel is downloaded to the FlexDDS-NG and then executed in real time. The program can synchronize the FlexDDS-NG waveform generation with the outside world via events and triggers.

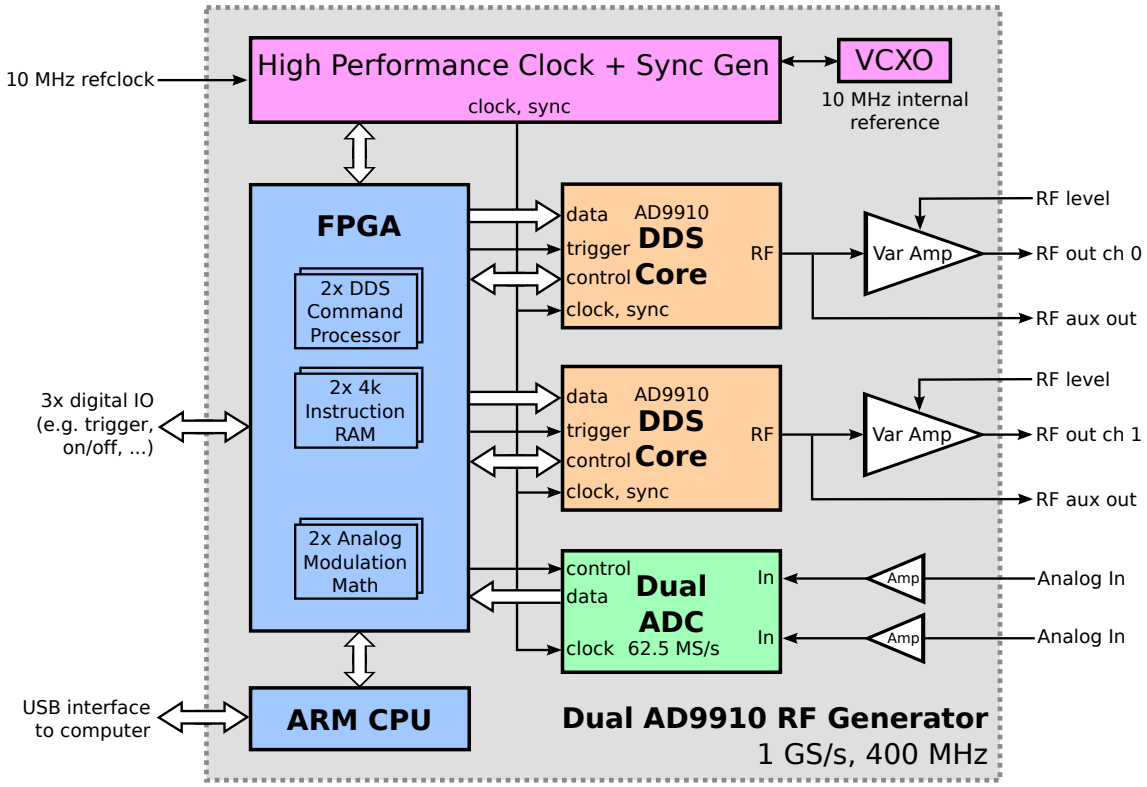


Figure 1.2: FlexDDS-NG overview schematic: Dual-channel AD9910 RF generator.

Chapter 2

The USB Serial Interface

The FlexDDS-NG DUAL as well as each slot of a FlexDDS-NG Rack have a USB interface. For the FlexDDS-NG DUAL, this is the only way of controlling the waveform generator. For the FlexDDS-NG Rack, it is usually not used and commandas are issued via the GBit Ethernet interface (see chapter 6 on page 47). Yet, you *can* mix Ethernet and USB with the Rack version if you like.

Once connected to a computer, it appears as a virtual COM port (VCP; COMx in Windows, `/dev/ttyACM`x in Linux). No drivers are required on Linux. Windows users may need to install the STM32 VCP drivers.

2.1 Connecting to the USB Serial Interface

Windows users can use the Putty program to connect to the virtual COM port. You need to select "Serial" and enter the correct COM port as shown in Figure 2.1 (page 7). No further settings are required, the baud rate and flow control are irrelevant and can be set to anything. A sample session is shown in Figure 2.2 (page 7).

Linux users can use the program *minicom*. You need to open a terminal (e.g. xterm), make the window sufficiently large and start minicom via:

```
minicom -w -c on -D /dev/ttyACMx
```

Again, baud rate and other serial settings are irrelevant and can be set to anything.

Note: When resetting the FlexDDS-NG, it will close and re-open the virtual COM port. Windows users then need to re-start Putty and re-connect. Linux users can just wait for minicom to re-connect automatically. In some cases, a different `/dev/ttyACM`x will be assigned and minicom will not re-connect. A simple way out is by generating an UDEV rule:

Create a file `/etc/udev/rules.d/60_flexdds_acm.rules` with the following content (all in one line, you need root permissions to do this):

```
ATTRS{idVendor}=="0483", ATTRS{idProduct}=="7270", \
ATTRS{serial}=="00240043:51123533:35313135", SYMLINK+="ttyFlexDDS"
```

The serial number has to be replaced with the actual one (will be displayed in `dmesg` after connecting the FlexDDS-NG via USB). Then call (as root):

```
udevadm control -reload
```

and re-plug the USB to the FlexDDS-NG. The FlexDDS-NG will now consistently show up as `/dev/ttyFlexDDS`.

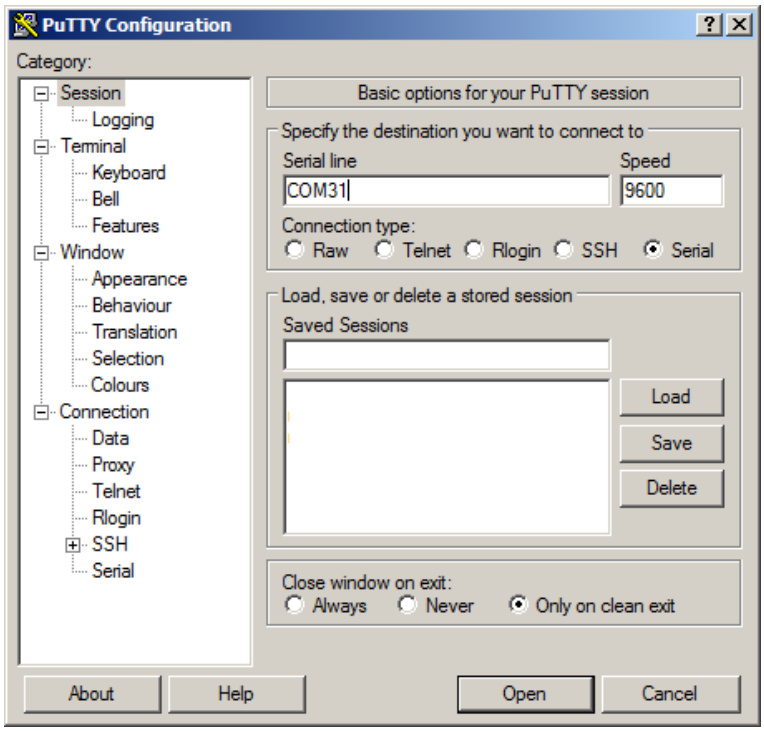


Figure 2.1: Putty connect dialog.

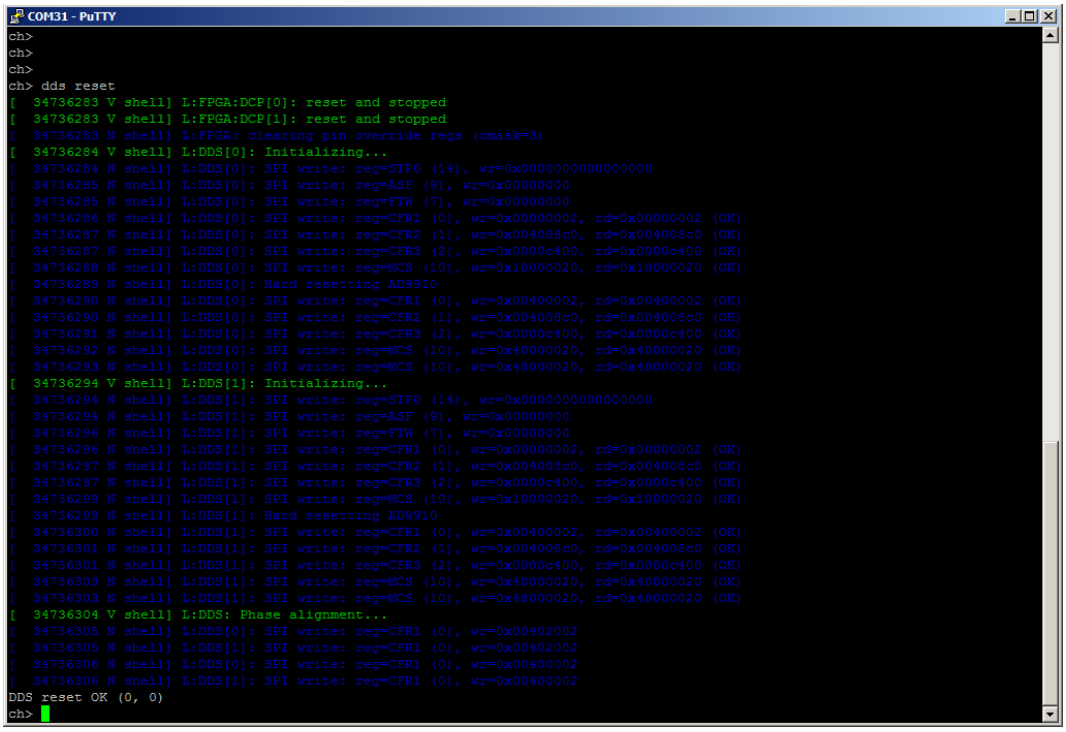


Figure 2.2: Example session in Putty.

2.2 USB Command Line Commands

The FlexDDS-NG accepts text commands. The most important ones are `dcp` and `dds`. Just typing the command name (without any arguments) will print out a short usage description.

`interactive` [`on`|`off`]

Switch interactive mode on or off.

Note: The FlexDDS-NG boots in *interactive mode*. This mode is intended for terminal sessions at the COM port interface. 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.

`dcp` ...

The main command to control the DDS command processor. See chapter 3.

`dds` ...

Perform certain actions on the AD9910 DDS chip.

`help`

Print short list of commands.

`reset`

Hard reset the device and perform a reboot. It is not recommended to do this frequently, especially on Windows operating systems, because it will disconnect and reconnect the USB port.

`poweroff`

Switch the power off. Same as pressing the power switch on the frontpanel while running.

`version`

Print version information.

`freq2ftw` [*FREQ*]

Convert the frequency *FREQ* in Hz in a frequency tuning word (FTW). Will print both the normal as well as the mirror frequency. Result is given in decimal and in hex.

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

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

Chapter 3

The DDS Command Processor (DCP)

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

The DCP is implemented in the FPGA and is responsible for controlling the AD9910 DDS synthesizer 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.

3.1 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 FE0AAAAA DDDDDDDD DDDDDDDD dddddddd dddddddd	SPI_WRITE	SPI write to AD9910
0001 00W1 DDDDDDDD DDDDDDDD DDDDDDDD DDDDDDDD	SPI_WRITE	SPI long write to AD9910
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.

`SPI_WRITE`: Queue a write to the AD9910 DDS chip via the SPI interface. The `AAAAA` bits specify the 5 bit register address in the AD9910. 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 SPI FIFO is empty and all SPI writes have been carried out. The bits `F` and `E` are completion event bits associated with the 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 `SPI_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 `SPI_WRITE` must have the bit 40 cleared and contains the most significant 32 bits and the register address as for a 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 BNC C, B, and A output from the DCP.

3.2 DCP Command Line Interface

The command `dcp` on the USB command line interface (virtual COM port) controls the DCP. There are several sub-commands described below. Simply entering `dcp` will print out a short usage text.

`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:

`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, the most important operations are also available as more convenient commands:

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 3.2: 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.

`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 3.2 on page 12. 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 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.

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 BNC inputs) 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: Not all bits and not all registers are writable, see the access column in Table 3.2. 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-NG.

Example for storing 1 word in SRAM:

continued ...

`dcp 0 spi:RAMB=0:c`	begin writing to the SRAM; dummy value 0 not stored
`dcp 0 spi:RAM324E=0x009dc970`	write the word and end writing to SRAM

Example for storing 3 words in SRAM:

`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:RAM324E=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.

Adr Hex	Symbolic Name	Register Description	Access	Chan
0x080	CFG_BNC_A	Configure BNC A IO on frontpanel	=+-^	0 only
0x081	CFG_BNC_B	Configure BNC B IO on frontpanel	=+-^	0 only
0x082	CFG_BNC_C	Configure BNC C IO 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	DDS_RESET	Reset the DDS and program it to initial state	=	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 3.7: DCP registers inside the FPGA. Only DCP channel 0 can configure shared hardware (such as configuring the BNC outputs). For a detailed register description, see page 30. The access column describes register access modes: '=' for write, '+', '-', '^' to set, clear, toggle bits. 'PC' means one dedicated register *per* DCP *channel*.

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

Write to a DCP register inside the FPGA.

REG denotes the DCP register address and can be specified either as symbolic name or as register address, see Table 3.7 on page 15. Symbolic names are case-insensitive. A detailed register description is given in a separate chapter on page 30.

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. Registers configuring shared hardware (such as the BNC output configuration) are only accessible from the DCP at channel 0.

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

Num	Name	Description
0	NONE	No event
2	ALL_SPI_FIFOS_FLUSHED	SPI FIFO into AD9910 empty on both channels
3	BNC_IN_A_RISING	Rising edge seen on BNC input A
4	BNC_IN_A_FALLING	Falling edge seen on BNC input A
5	BNC_IN_A_LEVEL	Level (low/high) seen on BNC input A
6,7,8	BNC_IN_B_...	Same as 3,4,5 (rising, falling, level) for BNC B
9,10,11	BNC_IN_C_...	Same as 3,4,5 (rising, falling, level) for BNC C
15	BP_TRIG_A	Backplane trigger A (available only in rack version)
16	BP_TRIG_B	Backplane trigger B (available only in rack version)
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 3.8: 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_.

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

Wait for a specified amount of time and/or up to 2 events.

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. BNC_IN_A_RISING). See Table 3.8 (page 16) 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 AD9910 will be generated when the wait instruction is over. This is particularly useful for triggering an update from an external BNC 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::BNC_IN_B_RISING	wait for rising edge on BNC input B
dcp 0 wait:1000h:u	wait for about 8000 ns, trigger IO_UPDATE afterwards
dcp 0 wait::BNC_IN_C_RISING:u	wait for rising edge on BNC input C, then trigger IO_UPDATE

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

Command to update basic settings and pins.

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 *N* (0...7).
`+/-/^a`	Set/clear/toggle the BNC A pin of the DCP channel. (*)
`+/-/^b`	Set/clear/toggle the BNC B pin of the DCP channel. (*)
`+/-/^c`	Set/clear/toggle the BNC C pin of the DCP channel. (*)

(*) Note: Each channel has a BNC A,B,C output. However, the physical BNC plug will only output that signal when first configured as *output* and when the appropriate signal is selected in the BNC output mux. See chapter 4, registers `CFG_BNC_A`, etc.

Examples:

`dcp 0 update:u!`	update the AD9910 on channel 0
`dcp 0 update:+o-dh!`	set the OSK pin HIGH and clear the DRHOLD and DRCTL pins
`dcp 0 update:^o=3p`	toggle OSK and select profile 3 (no flush)

```
dcp status
```

Print current status information related to the DCPs.

The output looks similar to the following:

```
DCP    : INST   DCP_FIFO  SPI_FIFO  LOCAL
       :        4096       256        128
DCP[0]: wait    324 ---x     0 E--x     0
DCP[1]: idle      0 E--x     0 E--x     0
```

This means that there are 324 instructions in the instruction FIFO of the DCP on channel 0 and none in channel 1. The SPI and local 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.

3.3 USB Session Examples

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

The following example configures both outputs for roughtly 130 MHz. One output frequency is 0.23 Hz higher giving 2 RF waveforms that "move" with respect to each other.

`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
`dcp 1 spi:stp0=0x3fff00002147ae15`	set freq (FTW in STP0) to 130 MHz + 0.23 Hz for ch 1
`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 start`	flush locally buffered instructions and start DCP

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

`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 start`	

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.

`dds reset`	
`dcp 0 spi:stp0=0x3fff000033333333`	normal frequency (200 MHz)
`dcp 1 spi:stp0=0x3fff0000cccccccd`	mirror frequency (800 MHz)
`dcp update:u`	
`dcp start`	

The next example shows how to use the `wait` instruction to **trigger actions from a BNC input**. For demonstration, it is required to send a signal into the the BNC input A. 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 BNC A input, and then switch to 30 MHz with half amplitude after the second rising edge of the BNC A input. For a list of events, see Table 3.8 (page 16).

`dds reset`	**All this is only done on channel 0.**

continued ...

```
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::3                         wait for rising edge on BNC A input (event 3)
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::BNC_IN_A_RISING           wait for rising edge on BNC A input (same as wait::3)
dcp 0 update:u                        IO_UPDATE the AD9910, 30 MHz, half ampl. at RF out
dcp start
```

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.

```
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 start
```

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.

```
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::35                        wait indefinitely for ramp to complete (event 35)
dcp 0 update:u                        IO_UPDATE channel 0 to transition to STP0 (no glitch)
dcp start
```

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.

```
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 start
```

Here's an example to show **how to synchronize two channels** on the same FlexDDS slot (or on the DUAL standalone device) 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 BNC inputs. In this example, we use the rising edge of the BNC A input which has to be applied by an external source.

```
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::BNC_IN_A_RISING                both DCPs wait for rising edge on BNC A
dcp spi:CFR1=0x00402000                  set autoclear phase bit (for next update)
dcp update:u                             IO UPDATE both channels simultaneously (re-starts phase accu)
dcp 0 start                              we start the first DCP
dcp 1 start                              and then the second DCP
```

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 BNC A rising edge event, they will be synchronized.

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 BNC A.
3. Perform a linear amplitude ramp from zero to 50% amplitude.
4. Wait for a second rising edge trigger on BNC A.
5. Perform a frequency ramp from the initial 30 MHz to 50 MHz.

We do all this on channel "RF Out 1".

```
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::BNC_IN_A_RISING           wait for BNC A input 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 amplitude 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::BNC_IN_A_RISING           wait for BNC A input 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 frequency 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 start
```

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

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 (BNC A).
2. Upon receipt of the trigger, the device ramps amplitude up from -34 dBm to -5 dBm over 100 ms (at 7 MHz).
3. Then the device ramps the frequency from 7 MHz to 7.005 MHz over 50 ms.
4. Then device jumps phase by 90 deg.
5. Then device continues to run at 7.005 MHz for 1 second.
6. Device turns the output off.

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, 7.005 MHz are `0x01cb1466`.

For the amplitude ramp in dBm, the output of the FlexDDS-NG 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.

There is a small 10-turn potentiometer on the front, called "Lvl" that can be turned with a screw driver to calibrate the amplitude for each output. 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-NG to full scale ampliude at our desired calibration frequency (e.g. 7 MHz):

`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 start`	

(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, ASF(−5 dBm) = 7318 = `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:

$$\mathrm{DRL} = \mathtt{ASF}^{\mathrm{high}}_{\mathrm{14bits}}\ 0_{\mathrm{18bits}}\ \mathtt{ASF}^{\mathrm{low}}_{\mathrm{14bits}}\ 0_{\mathrm{18bits}} = \mathtt{0x7258000004100000}$$

(Note that these numbers can easily be computed by shifting the computed values from before by 2 bits to the left: `0x1c96` ≪ 2 = `0x7258`, `0x0104` ≪ 2 = `0x0410`.)

FIXME: To be documented: Compute the amplitude ramp rate. The example below uses very slow ramp rates that can be seen with the bare eye on a spectrum analyzer.

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

$$\mathrm{DRL} = \mathtt{FTW}^{\mathrm{high}}_{\mathrm{32bits}}\ \mathtt{FTW}^{\mathrm{low}}_{\mathrm{32bits}} = \mathtt{0x01cb146601cac083}$$

The necessary instructions to perform this task are listed below. All these instructions are fed into the FlexDDS-NG at the beginning. Once the last instruction ("dcp start") is received, the program is executed.

`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=0x7258000004100000`	set amplitude ramp limits (`0x1c96` ≪ 2 = `0x7258`, see above)
`dcp 0 spi:DRSS=0x0000011a0000011a`	set ramp step size (see above)
`dcp 0 spi:DRR=0x00960096`	set ramp rate (see above)
`dcp 0 spi:CFR2=0x00280080`	enable ramp generator with destination amplitude
`dcp 0 wait::BNC_IN_A_RISING`	wait for rising edge TTL trigger on BNC input A
`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=0x01cb146601cac083`	set the frequency ramp limits (see above)
`dcp 0 spi:DRSS=0x0000000100000001`	set ramp step size (see above)
`dcp 0 spi:DRR=0x30963096`	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=0x1c96800001cb1466`	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

continued ...

`dcp 0 update:u-d`	update to the new settings to perform the phase jump
	pre-load the next settings:
`dcp 0 spi:stp0=0x0000800001cb1466`	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 start`	

Here are a couple of frequency ramp examples.

Ramp up-then-down: Normal frequency.

`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 start`	

Ramp down-then-up: Mirror frequency. This is basically the same but with the mirror frequencies, an 'upwards' ramp actually goes downwards in frequency.

`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 start`	

Ramp down, normal frequency

`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 start`	

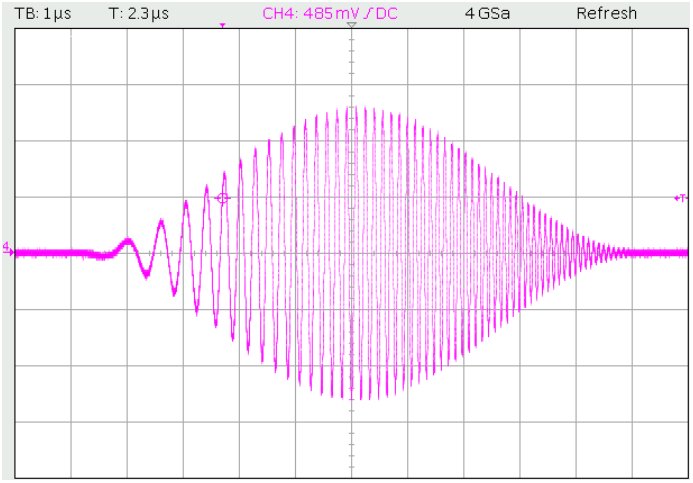
Example of a Hann shaped chirped pulse. This makes use of the SRAM modulation to shape the amplitude and the ramp generator to linearly sweep the frequency. This makes use of only 128 words (amplitude samples) of the total of 1024 available just to keep the code listing short (see (**) below). To use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

```
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: 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
```

continued ...

```
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 start
```

The final waveform looks like this:



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

```
dds reset
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.
                                                              continued ...
```

```
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 start
```

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 BNC A input to the `PROFILE 0` pin (inverted) and also route it to the DRCTL pin:

```
dds reset
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 BNC A to DRCTL
dcp 0 wr:CFG_PROFILE=0b000_000_010_001 route BNC A to PROFILE0, inverted
dcp start
```

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 BNC A input**.

For falling edge triggered pulses, you need to invert both `DRCTL` and `PROFILE0`.

Chapter 4

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.

4.1 CFG_BNC_A: Configure BNC A

Address: 0x080
Access: Write, Set, Clear, Toggle; DCP 0 only

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 BNC input/output on the frontpanel.

Register content description:

`WSCT`	Access mode: 00 to write, 01 to set, 10 to clear, 11 to toggle bits.
`DIR`	BNC 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 BNC output port. See Table 4.1.

Example: Note that only DCP channel 0 can access this register. Writes to this register from channel 1 will be silently ignored.

```
dds reset
dcp 0 wr:CFG_BNC_A=0x200      configure BNC A port as output (DIR=1), LOW (OUT_MUX=0)
```

continued ...

```
dcp 0 wait:1000:              wait about 1 ms
dcp 0 wr:CFG_BNC_A=0x300      configure BNC A as output, 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_bnc_a=^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_bnc_a=^0x100     toggle the INV bit again back to 0; port goes LOW
dcp 0 wait:200:
dcp 0 wr:cfg_bnc_a=+0x100     set the INV bit; port goes HIGH
dcp 0 wait:200:
dcp 0 wr:cfg_bnc_a=-0x100     clear the INV bit; port goes LOW
dcp start
```

4.2 CFG_BNC_B: Configure BNC B

Address: 0x081
Access: Write, Set, Clear, Toggle; DCP 0 only

See the description for `CFG_BNC_A` above.

4.3 CFG_BNC_C: Configure BNC C

Address: 0x082
Access: Write, Set, Clear, Toggle; DCP 0 only

See the description for `CFG_BNC_A` above.

Example: Wait for trigger on BNC B before switching frequencies.

```
dds reset
dcp 0 wr:CFG_BNC_A=0                (not needed as 0 is the initial value)
dcp 0 spi:stp0=0x3fff000010000000   configure single tone profile
dcp 0 wait::BNC_IN_B_RISING         wait for BNC B input rising edge
dcp 0 update:u                      update the AD9910, makes waveform appear at RF output
dcp 0 spi:stp0=0x3fff000020000000   immediately pre-configure the next frequency
dcp 0 wait::BNC_IN_B_RISING         wait for BNC B input rising edge
dcp 0 update:u                      after rising edge, update the AD9910 again (next frequency at output)
dcp start
```

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

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

Num	Name	Description
0...63	(Events)	Same as the events in Table 3.8 (page 16)
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_BNC_A,B,C	BNC A,B,C output from the channel 0 DCP
103 ...105	DCP1_BNC_A,B,C	BNC 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 4.1: Output mux choices: Values for BNC_OUT_MUX. Choose which signal is routed out via the BNC connector.

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 AD9910 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 BNC input).

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 3.8 (page 16), values 0 to 31.

Example: How to use the external BNC input "A" to quickly switch on/off the RF output, i.e. to gate the RF output via the OSK functionality of the AD9910.

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 amplitude to use because when OSK is enabled in the AD9910, the amplitude scale factors from the STP registers are ignored.

Note that BNC A must be configured as input for this to work (this is the default; see register CFG_BNC_A.

The CFG_OSK register has to be configured to disconnect the DCP and route local MUX output to the AD9910. The MUX is configured to choose the BNC A input (Table 3.8: BNC_IN_A_LEVEL, has value 5, i.e. binary 00101).

```
dds reset
dcp 0 spi:cfr1=0b0000000_11000001_00000010_00000000    set OSK bits (see text above)
dcp 0 spi:asf=0xffffffff                               set full amplitude
dcp 0 spi:stp0=0x3fff00001999999a                      ch 0, set STP0 to 100 MHz (ampl. irrelevant)
dcp 0 update:u                                         flush settings in the AD9910
dcp 0 wr:CFG_OSK=0b010_0_000_00101                     route BNC A to the OSK pin (*)
dcp start
```

(*) 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`)

4.5 CFG_UPDATE: Configure Routing to the IO_UPDATE Pin on the AD9910

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 IO_UPDATE pin will prevent normal operation of the DCP SPI updates into the AD9910.

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

4.6 CFG_DRCTL: Configure Routing to the DRCTL Pin on the AD9910

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 that generates **frequency ramps ("sweeps") with external control via BNC input**: The following instructions generate a ramp from 20 MHz to 100 MHz with external control via the BNC A input. When BNC A is HIGH, it ramps up and stays at the end frequency (100 MHz). Once BNC

A 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 BNC A.

```
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_00000101       route BNC A input to DRCTL pin of AD9910
dcp 0 update:u                           IO_UPDATE to commit register changes
dcp start
```

Here's a similar example: **Triggered upwards frequency ramps ("sweeps")**
Unlike the previous example, each rising edge of the BNC A 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 BNC A input. The only difference is that we set the no-dwell bit this time.

```
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_00000101       route BNC A input to DRCTL pin of AD9910
dcp 0 update:u                           IO_UPDATE to commit register changes
dcp start
```

4.7 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.

4.8 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 BNC input *n* (A, B, C) 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.

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

```
dds reset
dcp 0 spi:stp0=0x3fff00001999999a          ch 0, set STP0 to 100 MHz, phase to 0 deg. full amplitude
dcp 0 spi:stp1=0x1fff00004ccccccd          ch 0, set STP1 to 300 MHz, phase to 0 deg. half amplitude
dcp 0 update:u                             flush settings in the AD9910
dcp 0 wr:CFG_PROFILE=0b000_000_010_000     route BNC A to the PROFILE0 pin (*)
dcp start
```

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

4.9 DDS_RESET: Reset the DDS and Program It to Initial State

Address: 0x08a
Access: Write; Per-channel (one for each DCP

NOTE: Requires slot version 0.62 or higher!

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

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

When the DDS_RESET 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 `dds reset` USB command. 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 `dds reset` from the command line 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.

`dcp 0 spi:STP0=0x3fff0000051eb852`	set STP0 to 30 MHz, full amplitude
`dcp 0 spi:CFR2=0x01000080`	set CFR2 to matched latency and ASF from STP
`dcp 0 update:u`	update; this makes the above appear at the RF outputs
`dcp 0 wait:1000000:`	wait a second
`dcp 0 wr:DDS_RESET=1`	now reset channel 0
`dcp start`	

4.10 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.

4.11 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.

4.12 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.

4.13 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.

4.14 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.

4.15 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_O0`.

4.16 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.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	MODF	
Defl.															0	

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.

`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)

Chapter 5

Analog Modulation

The FlexDDS-NG supports analog modulation of the RF outputs using signals applied to the RF In 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.

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

5.1 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:

$$D = \mathrm{coerce}_{16}\left(\frac{(A_0 - O_0)\cdot S_0 + (A_1 - O_1)\cdot S_1}{2^{12}} + O\right) \tag{5.1}$$

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 are clipped to zero while values above $2^{16} - 1$ are clipped to $2^{16} - 1 = 65535$.

$$\text{coerce}_{16}(x) := \begin{cases} 0, & \text{if} \quad x < 0 \\ x, & \text{if} \quad 0 \leq x < 2^{16} \\ 2^{16} - 1, & \text{if} \quad x \geq 2^{16} \end{cases} \tag{5.2}$$

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_OFF`. 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.

5.2 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:

$$D = \frac{A_0}{1} + 2^{15} = \frac{(A_0 - 0) \cdot 2^{12} + (A_1 - 0) \cdot 0}{2^{12}} + 2^{15} \tag{5.3}$$

By comparing coefficients with equation 5.1, we find that:

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

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

```
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 start
```

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 $50\,\Omega$; 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:

```
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
```

continued ...

```
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 start
```

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:

```
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)
dcp start
```

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`

5.3 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.

```
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 start
```

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.

5.4 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.

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 5.1 need to cover the range from $\texttt{0x028f5c29}/2^{12} = \texttt{0x28f5}$ to $\texttt{0x07ae147b}/2^{12} = \texttt{0x7ae1}$.

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 5.1 then simplifies to

$$D = \text{coerce}_{16}\left(\frac{A_0 \cdot S_0}{2^{12}} + O\right) \tag{5.4}$$

(Here, the 2^{12} in the denominator comes from equation 5.1 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:

$$\begin{aligned} D(A_0 = -2^{15}) &= \texttt{0x28f5} = 10485 \\ D(A_0 = +2^{15} - 1) &= \texttt{0x7ae1} = 31457 \end{aligned} \tag{5.5}$$

$$\begin{aligned} S_0 &= \frac{(31457 - 10485) \cdot 2^{12}}{2^{15} - 1 + 2^{15}} = 1310.77 = \texttt{0x51f} \\ O &= \frac{10485 \cdot (2^{15} - 1) - 31457 \cdot (-2^{15})}{2^{15} - 1 + 2^{15}} = 20971.16 = \texttt{0x51eb} \end{aligned} \tag{5.6}$$

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

```
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 start
```

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.

5.5 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 5.1 but instead performs the following computation:

$$\begin{aligned} D_{7\ldots0} &= \text{coerce}_{16}\left(\frac{(A_0 - O_0) \cdot S_0}{2^{12}} + O\right)/2^8 && \text{phase bits} \\ D_{15\ldots8} &= \text{coerce}_{16}\left(\frac{(A_1 - O_1) \cdot S_1}{2^{12}} + P\right)/2^8 && \text{amplitude bits} \end{aligned} \tag{5.7}$$

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.

```
dds reset
dcp 0 spi:stp0=0x0000000003000000              frequency 11.7 MHz; phase and amplitude zero
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp 0 spi:CFR2=0b00000000_00000000_01010000    enable parallel data port
```

continued ...

`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 0 update:u`	update AD9910 (make CFR* effective)
`dcp start`	

Chapter 6

FlexDDS-NG Rack

The FlexDDS-NG Rack is a 19” enclosure that can hold up to 6 slots.

Each of the slots can be the heart of an FlexDDS-NG DUAL with the same frontpanel elements except the 10 MHz input/output and the power and reset pushbuttons because these elements are on the rack controller rather than on individual slots.

The FlexDDS-NG Rack provides a GBit network interface that can be used to control all the slots 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 USB.

For each slot, the FlexDDS-NG Rack provides a FIFO buffer capable of holding up to 1 million DCP instructions. An unlimited amount of instructions can be streamed in real time.

Even though the network on the rack is the recommended way to communicate, you can still plug a USB cable into any individual slot to obtain information of the slot and to feed it with DCP commands.

6.1 The GBit Network Interface on the FlexDDS-NG Rack

The FlexDDS-NG Rack provided a GBit Ethernet port on the control slot (leftmost slot) labeled “Ethernet” When a network cable is plugged in, the yellow LED indicates carrier detection. The green LED blinks upon network activity.

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

Configuring the IP address:

By default, the FlexDDS-NG 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-NG Rack. See chapter 6.2 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 main slot. Here is a step-by-step instruction on how to do

this:

1. Power down the FlexDDS-NG.
2. Remove the micro-SD card. It is accessible from the main slot 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 `flexdds_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-NG. 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-NG again. The network address is now configured.

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

6.2 The USB Console on the FlexDDS-NG Rack

See instructions about the USB console in the firmware update instructions. This also explains how to obtain the IP network and the MAC addresses.

6.3 FlexDDS-NG Rack Network Interface and FIFO Operation

The FlexDDS-NG Rack opens a TCP port for each slot and each protocol. E.g. for the text based protocol, slots 0...5 correspond to ports 26000...26005, respectively.

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

The DCP instructions are queued into a large per-slot FIFO holding, by default, up to 1 million DCP instructions (per slot). The FIFO content is streamed to the slots over the backplane. Each slot has a smaller DCP instruction FIFO (typically 4096 instructions per channel) to avoid effects caused by transmission latency within the rack (see Figure 6.1).

The per-slot FIFO sizes can be configured and made much larger. In order to do this, you need to edit the file called `flexdds.cfg` on the micro-SD of the FlexDDS-NG Rack. Follow the same 6 steps as explained on page 47 for changing the IP address. However, this time, edit the file `flexdds.cfg`.

This is the sample content of the file.

```
# ** FlexDDS-NG Config File **

# Memory allocation
# =================

# FIFO size in kilo bytes for each slot.
# Each FIFO entry consumes 8 bytes (64 bit network frame).
# Minimum is 64 kBytes.
# Total sum must not exceed 786432 kBytes (768 Mbytes)
# Examples:
#  64 10000 64 64 64 64
#      -> Slot 1 has 10000 kBytes, all others have 64 kBytes
#  131072 131072 131072 131072 131072 131072
#      -> All slots have 131 MBytes which is 16 million DCP insns
#  655360 16384 16384 16384 16384 16384
#      -> SLot 0 has 81.92 million DCP insns, all others only about 2 million.
fifo_buf_size_kb = 8192 8192 8192 8192 8192 8192

# EOF
```

The total available memory is 768 MBytes, i.e. 786432 kBytes. This allows for 16 million DCP instructions per slot for each slot or asymmetric distributions like 80 million for one slot and "just" 2 million for other slots.

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 slots 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.

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.

If you press the red "Reset" pushbutton on the FlexDDS-NG Rack or supply a HIGH pulse (at least 50 ms) into the Reset BNC input, a full reset is performed: All network connections are closed, all FIFO contents are discarded and all slots are reset.

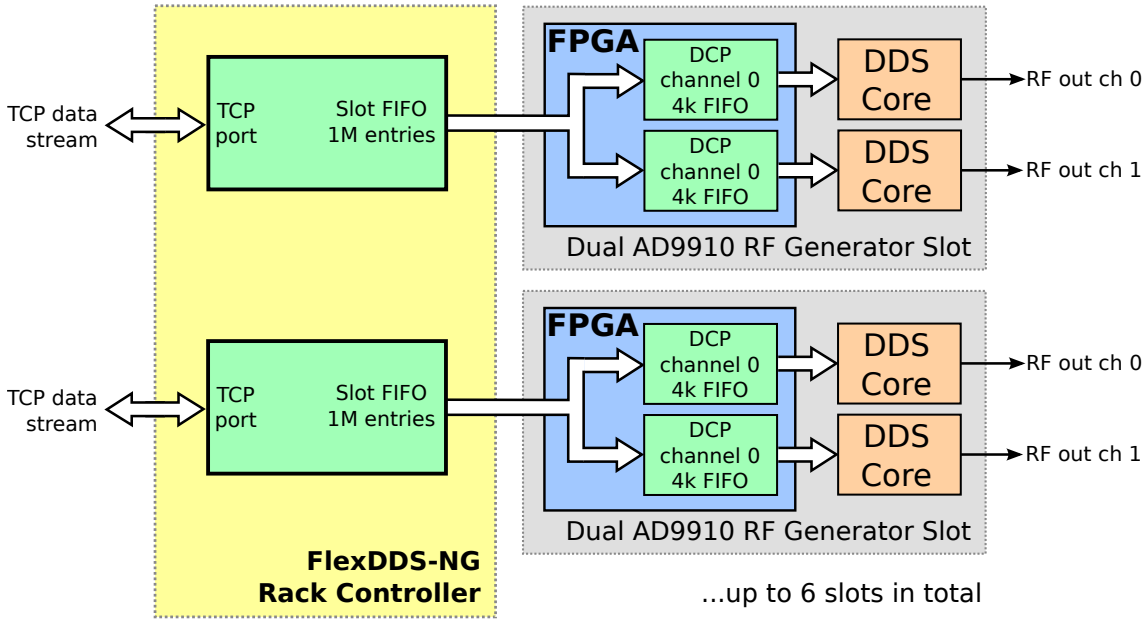


Figure 6.1: Data streams and FIFOs in the FlexDDS-NG Rack. Each slot has its own TCP port, TCP data stream and large slot FIFO within the rack controller. Each slot has its own smaller FIFO per channel. There is one data stream per slot which is divided into multiple channels on the slot. Hence, if on a slot, one 4k FIFO runs full, *both* FIFOs on the slot can no longer be supplied with instructions.

Note: It is important to understand that each slot (0...5) is associated with a specific TCP port and has a dedicated FIFO buffer in the rack. Hence, each slot is fed with its own independent data stream. However, each slot can have multiple channels and the instructions for these channels are in the same FIFO in the rack. See Figure 6.1.

This has one important consequence: Each slot 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 slot is stalled. Now, if e.g. channel 0 executes instructions much faster than channel 1, then the DCP FIFO in the slot 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 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 FIFOs. If timing is unclear, consider re-arranging the setup so that different slots are being used.

(For users familiar with the "old" 8-channel FlexDDS Rack this is a relaxation of the requirements. The old 8-channel rack required that instructions are strictly ordered by time and then combined into a single data stream. This was not always easy to ensure.)

6.4 FlexDDS-NG Rack Text Network Protocol (port 2600x)

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 `75f4a4e10dd4b6bx` where the last digit, *x*, has to be replaced by the slot number (0 to 5).

After this authentication step, text based commands are read much like the USB interface of the individual slots (or like the USB interface of the FlexDDS-NG DUAL).

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-NG 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-NG Rack.

The following network commands are supported in text mode:

`dcp [0|1] ...`	Feed DCP instructions into the FIFO buffer of the slot.
`dcp flush`	Flush DCP commands; done automatically after about 1 second.
`dds [0|1] reset`	Reset DDS channel(s); **New in rack version 0.48 (slot version 0.62)!**
`dds [0|1] r`	Short form of the above.
`set VAR=VALUE`	Configure certain properties; see below.
`quit`	Close the current network connection.
`reset`	Like red pushbutton: Reset all slots, FIFO buffers and close all TCP connections.

All commands refer only to the associated slot with the exception of the "reset" command which performs a global reset.

The `dcp` command works as described in section 3.2 but does not implement `start`, `stop` and `reset`.

Note: DCP start/stop is currently not implemented on the rack. Each slot DCP starts in "running" state and executes the commands as they are fed by the rack controller. To synchronize, it is recommended to start with a wait-for-trigger instruction for all slots.

Note: To reset all channels, you can use the general `reset` command. Starting with rack version 0.48 (slot version 0.62), there is also a `dds reset` network command just as with the USB interface. However note that you **must** wait with sending commands after a `dds reset` until the reset has been processed. You must physically wait with transmission, you cannot use a `dcp wait:` command.

Note: The `dds reset` empties all FIFOs and cancels all DCP actions! Therefore you if you use `dds reset` as the first command, you **must** wait at least 100 ms before sending new commands over the network, otherwise these new commands may be removed as well. Alternatively, you can also send the `dds reset` over the network after an experiment run is complete but before new commands for the next run are transmitted.

Note: The FlexDDS-NG Rack with network directly feeds the DCP on the slot FPGAs without the per-slot microcontroller doing anything. The "!" in DCP commands corresponds to "dcp flush" on the rack.

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 26001 corresponding to slot 1 (i.e. the *second* RF generator slot from left):

`75f4a4e10dd4b6b1`	sent auth token for slot 1 ("password")
`Auth OK`	response that auth is OK
`dcp 0 spi:stp0=0x3fff00005c54943a`	queue a DCP instruction for channel 0 (slot 1)
`OK`	response from rack
`set resp_suppress_ok=1`	suppress "OK" responses
`dcp 1 spi:stp0=0x3fff00005264943a`	queue a DCP instruction for channel 1 (slot 1)
`dcp update:u!`	queue DCP update insn for channels 0 and 1 (slot 1)
`set dcp_dump_isn=1`	request that DCP instructions are echoed back
`dcp 0 wr:cfg_bnc_b=0x300`	set BNC B output HIGH on slot 1
`DCP: 0x0031208100000300`	response: corresponding 64 bit rack DCP instruction

This is a more real-life example for slot 2. It sets two frequencies (on channel 0 and 1), waits a second and then resets both channels.

`75f4a4e10dd4b6b2`	sent auth token for slot 2 ("password")
`Auth OK`	response that auth is OK
`set resp_suppress_ok=1`	suppress "OK" responses
`dcp 0 spi:stp0=0x3fff0000051eb852`	queue a DCP instruction for channel 0
`dcp 0 spi:stp0=0x3fff0000071eb852`	queue a DCP instruction for channel 1
`dcp spi:CFR2=0x01000080`	next instruction channels 0 and 1
`dcp update:u!`	queue DCP update insn for channels 0 and 1
`dcp wait:1000000:`	wait a second
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 `dds reset` any time.)	
`dds reset`	perform a reset to be ready for the next experiment run

6.5 FlexDDS-NG Rack Binary Network Protocol (port 2601x)

The text based network protocol has the disadvantage of imposing significant network and processing overhead. 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).

The binary 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 26010...26015 corresponding to slots 0...5.

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:

63 ... 56 55 ... 48 47 ... 0	Description
0000	NULL frame; ignored
0001 000F SSS1 CCCC DDDDDDDD DDDDDDDD DDDDDDDD DDDDDDDD DDDDDDDD DDDDDDDD	DCP instruction
0001 000F SSS0 0000 lll.....	NOP (ignored)
0001 000F SSS0 0001 00000000 00000000 00000000 aaAAAAAA dddddddd dddddddd	Write slot FPGA reg
0010 0000 0000 0001 lll.....	Rack command
0011 aaaa aaaa aaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa	Network auth token

The first 4 bits (blue) specify the network frame type:

- 0000 NULL frames: These are ignored or can be used for network benchmarking.
- 0001 SLOT frames: These are transferred to a slot as specified via the SSS bits.
- 0010 RACK frames: These are interpreted by the rack.
- 0011 AUTH frames: Used for the authentication token.

Description of the AUTH frame:

After opening a TCP connection, the first 8 bytes to be sent are the binary authentication token. This is sort of a "fixed password" as the most basic means to prevent unauthorized access. The authentication token is `0x75f4a4e10dd4b6b`*x* where the last digit, *x*, has to be replaced by the slot number (0 to 5). The AUTH token has to be sent LSB first (little endian) just as any other network frame.

If the FlexDDS-NG Rack closes the network connection after the AUTH frame, then the authentication failed, i.e. the auth frame was incorrect. (Check port, slot ID and byte order and be sure to transfer 8 binary bytes and not 16 text letters!)

Description of SLOT frames:

The destination slot address is specified via the three `SSS` bits; valid values are 000 for slot 0 to 101 for slot 5.

The 4 `CCCC` bits specify a channel bit mask inside the slot. To address DCP channel 1, use 0001, for channel 2 use 0010, for channel 3 (if available) use 0100 and so on. To address multiple channels, set multiple bits. E.g. to address channels 0 and 1 with the same instruction, use 0011.

The 48 `D` bits represent the DCP instruction as explained in chapter 3.1 on page 9.