## Flexible Multi-Channel Phase-Coherent RF Source

## $\triangleright$ FlexDDS-NG DUAL

- Two independent output channels for up to 400 MHz ( $1 \mathrm{GS} / \mathrm{s}$ 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 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 ( $>30 \mathrm{MBytes} / \mathrm{s}$ )
- GBit Ethernet: Connect anywhere in the lab, no USB cables, no special OS drivers
- Global trigger inputs that affect all slots simultaneously



## Dual Channel Standalone RF Generator

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

- 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

1 Basic Operation of the Dual Channel AD9910 RF Generator ..... 4
1.1 Frontpanel Elements ..... 4
1.2 Functional Overview ..... 5
2 The USB Serial Interface ..... 7
2.1 Connecting to the USB Serial Interface ..... 7
2.2 USB Command Line Commands ..... 9
3 The DDS Command Processor (DCP) ..... 11
3.1 DCP Instruction Description ..... 11
3.2 DCP Command Line Interface ..... 12
3.2.1 dcp \#: DCP raw command entry ..... 13
3.2.2 dcp spi: Controlling the AD9910 via SPI register writes ..... 14
3.2.3 dcp wr: Modifying internal FPGA registers ..... 17
3.2.4 dcp wait: Timed waits or waiting for up to 2 events ..... 18
3.2.5 dcp update: Update basic settings and pins ..... 19
3.2.6 dcp status: Print current status information ..... 20
3.3 DCP Program Examples ..... 21
3.3.1 Basic: Two outputs with small frequency offset ..... 21
3.3.2 Phase jump by $\pi$ after 2 seconds ..... 21
3.3.3 Using the mirror frequency ..... 22
3.3.4 Wait for BNC inputs; externally triggering DDS changes ..... 22
3.3.5 Frequency ramp up, then down, then again up ..... 22
3.3.6 Phase ramp ..... 23
3.3.7 Synchronizing two channels on the same slot ..... 24
3.3.8 Amplitude ramp followed by a frequency ramp ..... 25
3.3.9 A more complex real-world task with external trigger, amplitude and frequency ramp ..... 26
3.3.10 Working with the AD9910: Frequency ramps ..... 30
3.3.11 Real-world example: Wait for trigger, set 75 MHz , at next trigger ramp down, trigger again, then switch off ..... 30
3.3.12 Example of a Hann shaped chirped pulse ..... 32
3.3.13 Analog frequency modulation with digitally controlled amplitude and phase ..... 35
3.4 DCP Register Description ..... 36
3.4.1 CFG_BNC_A: Configure BNC A ..... 36
3.4.2 CFG_BNC_B: Configure BNC B ..... 37
3.4.3 CFG_BNC_C: Configure BNC C ..... 37
3.4.4 CFG_OSK: Configure Routing to the OSK Pin on the AD9910 ..... 37
3.4.5 CFG_UPDATE: Configure Routing to the IO_UPDATE Pin on the AD9910 ..... 39
3.4.6 CFG_DRCTL: Configure Routing to the DRCTL Pin on the AD9910 ..... 39
3.4.7 CFG_DRHOLD: Configure Routing to the DRHOLD Pin on the AD9910 ..... 40
3.4.8 CFG_PROFILE: Configure Routing to the PROFILE Pins on the AD9910 ..... 40
3.4.9 DDS_RESET: Reset the DDS and Program It to Initial State ..... 41
3.4.10 AM_S0: Analog Modulation, Scale Factor 0 ..... 42
3.4.11 AM_S1: Analog Modulation, Scale Factor 1 ..... 42
3.4.12 AM_O: Analog Modulation, Offset ..... 42
3.4.13 AM_P: Analog Modulation, Offset for Polar Modulation ..... 43
3.4.14 AM_00: Analog Modulation, Offset for Input Channel 0 ..... 43
3.4.15 AM_01: Analog Modulation, Offset for Input Channel 1 ..... 43
3.4.16 AM_CFG: Analog Modulation Configuration Register ..... 44
4 Analog Modulation ..... 45
4.1 Amplitude, Phase and Frequency Modulation ..... 45
4.2 Example: Amplitude Modulation ..... 47
4.3 Example: Phase Modulation ..... 49
4.4 Example: Frequency Modulation ..... 49
4.5 Polar Modulation ..... 50
5 FlexDDS-NG Rack ..... 52
5.1 The Main Slot Front Panel on the FlexDDS-NG Rack ..... 52
5.2 The GBit Network Interface on the FlexDDS-NG Rack ..... 54
5.3 The USB Console on the FlexDDS-NG Rack ..... 55
5.4 FlexDDS-NG Rack Network Interface and FIFO Operation ..... 56
5.5 FlexDDS-NG Rack Text Network Protocol (port 2600x) ..... 58
5.6 FlexDDS-NG Rack Binary Network Protocol (port 2601x) ..... 60
5.7 FlexDDS-NG Rack Phase Synchronization Accross Multiple Slots ..... 61
5.8 FlexDDS-NG Rack Configuration Options ..... 63

## 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.


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.

### 1.1 Frontpanel Elements

The standalone FlexDDS-NG DUAL and the dual channel AD9910 RF generator slot share most of their frontpanel elements. See also the FlexDDS-NG DUAL datasheet available online.
RF Out 0/1 Main (SMA): Main radio frequency (RF) output channels.
RF Out 0/1 Aux (SMA): A copy of the signal on the main output with about -5 dBm signal level. Can be used for monitoring, e.g. to attach a frequency counter or oscilloscope.

Lvl adjustment screw: These are two multi-turn potentiometers that allows to scale the RF power level of the respective main output channel over a range of 60 dB . Turn clockwise to increase the output level. By default set to full amplitude. Use a small screwdriver to adjust.

This sets the amplifier gain covering a full scale RF output power range from below -40 dBm to above
+10 dBm and allows to scale the RF level to your needs without loosing any bit of resolution in the DDS.
RF on/off switch: Master RF switches operating on the OSK funktion of the AD9910. Turn towards the "on" state to enable RF output for the associated RF channel.

Analog In Ch 0/1: Two dedicated analog inputs for analog modulation: These inputs are digitally sampled and allow you to perform amplitude/phase/frequency or even polar I/Q modulation at a rate of $62.5 \mathrm{MS} / \mathrm{s}$ with 12 to 14 bits resolution. Modulation gain and offset are programmed digitally and can be tuned on the fly without the need to change any analog circuits. Full scale voltage is $\pm 1 \mathrm{~V}$ with $50 \Omega$ termination.

BNC A, B, C: Three digital IOs can be configured for various functions including fast on/off, triggering, changing sweep direction, interrupting sweeps, quickly switching output profiles or as outputs e.g. to control post amplifiers or get notified of end-of-sweep.

The IO voltage of the digital IO BNC ports is, by default set to 5 V . It is possible to change that voltage to 3.3 V by removing the slot and setting a jumper on the slot to a different position. Please refer to figure 1.2 on how to do this.
LEDs at BNC A, B: These are configurable. By default the red "Trig" LED flashes each time an I/O Update is issued to the AD9910 and the green "A"/"B" LED indicates the logic level present at the BNC port (LED is on for HIGH level).
uC LED: Power LED, blinks during boot and self-test and should be constantly on during operation.
Err LED: Error LED. Can be on during booting. For firmware newer than 0.82 : This LED blinks to indicate critical hardware error. Immediately after a reset, the LED is on for a short period of time until the slot has locked to the external or backplane reference clock.
USB: See the chapter 2.
Reset pushbutton (red): (Standalone device only) Hardware reset for the generator, will reboot and perform self-test and initialization when pressed.

Power pushbutton (green): (Standalone device only) Push to power up and power down the device. Must be pressed for more than a second to power up.
10 MHz in: (Standalone device only) External 10 MHz reference clock input.
Stby: (Standalone device only) Yellow LED to indicate standby power from the power supply.

### 1.2 Functional Overview

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 .
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: IO voltage selection jumper on the dual channel AD9910 RF generator. Remove the 4 screws holding the slot and pull out the slot to access this jumper. When re-inserting be careful to not damage the cable on the bottom side of the slot. To open up the standalone FlexDDS-NG DUAL, remove the two blue rubber decoration stripes on the left side close to the reset button, then open the 4 screws that appear and remove the side panel. Finally, slide out the top cover metal to access the board.


Figure 1.3: 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 5 on page 52). 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; COM $x$ 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 8). 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 8).
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/ttyACM }
```

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: Interactive Mode

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 noninteractive 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 $F R E Q$ 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) $62.5 \mathrm{MHz}(1 \mathrm{GHz} / 16)$ with deterministic timing for precise real-time control. Each DCP has a FIFO buffer holding 2048 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 | $39 \ldots 32$ | 31 | 16 | 15 | 0 | Name | Description |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 0000 |  |  |  |  |  |  | NOP | No-op (does nothing) |
| 0001 | 00W0 | FEOAAAAA | DDDDDDDD | DDDDDDDD | dddddddd | dddddddd | SPI_WRITE | SPI write to AD9910 |
| 0001 | 00W1 | DDDDDDDD | DDDDDDDD | DDDDDDDD | DDDDDDD |  | SPI_WRITE | SPI long write to AD9910 |
| 0010 |  | AAAAAAAA | DDdddddd | dddddddd | dddddddd | dddddddd | REG_WRITE | Write to DCP register |
| 0011 | OURH | 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 RRRRRRR 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 $(1.024 \mu \mathrm{~s}$ 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 nonzero. The U bit pulses the IO_UPDATE pin to the AD9910. The 00, 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 1 xyz , 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.

### 3.2.1 dcp \#: DCP raw command entry

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

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

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

For example:

| 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 <br> Hex | Symb. <br> 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 |  |
| $0 \times 03$ | ADAC | Auxiliary DAC Control | full | 0x0000007F |
| 0x04 | IOUR | I/O Update Rate | full | OxFFFFFFFF |
| 0x07 | FTW | Frequency Tuning Word | full | $0 \times 0$ |
| $0 \times 08$ | POW | Phase Offset Word | full | 0 x 0 |
| 0x09 | ASF | Amplitude Scale Factor | full | 0x0 |
| 0 xOA | MCS | Multichip Sync | denied |  |
| 0x0B | DRL | Digital Ramp Limit | full | 0x0 |
| Ox0C | DRSS | Digital Ramp Step Size | full | 0x0 |
| OxOD | DRR | Digital Ramp Rate | full | 0x0 |
| Ox0E | STPO | Single Tone Profile 0 | full | 0x0 |
| 0x0F | STP1 | Single Tone Profile 1 | full | 0x0 |
| 0x10 | STP2 | Single Tone Profile 2 | full | 0x0 |
| $0 \times 11$ | STP3 | Single Tone Profile 3 | full | 0x0 |
| $0 \times 12$ | STP4 | Single Tone Profile 4 | full | 0 x 0 |
| $0 \times 13$ | STP5 | Single Tone Profile 5 | full | 0x0 |
| 0x14 | STP6 | Single Tone Profile 6 | full | $0 \times 0$ |
| 0x15 | STP7 | Single Tone Profile 7 | full | 0x0 |
| 0x16 | RAMB | RAM Begin (no data) | full | N/A |
| $0 \times 17$ | 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.

### 3.2.2 dcp spi: Controlling the AD9910 via SPI register writes

```
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.)
$R E G$ denotes the AD9910 register and can be specified either as symbolic name or as register address ( 0 to $0 x 16$ ), see Table 3.2 on page 14 . 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 $0 x$ prefix or in decimal or in binary with a 0 b 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: Immutable Bits in Registers

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 $0 \times 1111111122222222$ first stores $0 \times 11111111$ 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
dcp 0 spi:RAM64C=0x00000000_0009de7d:c
dcp 0 spi:RAM64C=0x00277872_0058c94b:c
dcp 0 spi:RAM64E=0x009dc970_00f66e3c
```

begin writing to the SRAM; dummy value 0 not stored
write 2 words, 0 and then 0 x 9 de 7 d , continue
write 2 more words, continue
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:

> | dcp 0 | spi:RAMB=0:c |
| :--- | :--- |
| dcp 0 | spi:RAM32E=0x009dc970 writing to the SRAM; dummy value 0 not stored |

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:RAM32E=0x33333333 | write the word and end writing to SRAM |

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

| Adr <br> Hex | Symbolic <br> Name | Register Description | Acc ess | 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_0 | Analog Modulation, Offset | = | PC |
| 0x103 | AM_00 | Analog Modulation, Offset Input Channel 0 | = | PC |
| 0x104 | AM_01 | 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 36 . The access column describes register access modes: ' $=$ ' for write, ' + ', '-', '^' to set, clear, toggle bits. 'PC' means one dedicated register per DCP channel.

### 3.2.3 dcp wr: Modifying internal FPGA registers

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

$R E G$ denotes the DCP register address and can be specified either as symbolic name or as register address, see Table 3.7 on page 17. Symbolic names are case-insensitive. A detailed register description is given in a separate chapter on page 36 .
$V A L$ 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 $0 x$ 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): |  |  |
| $32(48)$ | (0_)SPI_FIFO_FLUSHED | SPI FIFO into AD9910 empty; all SPI writes finished |
| $33(49)$ | (0_)SPI_FIFO_EV0 | SPI FIFO write event 0 [not yet documented] |
| $34(50)$ | (0_)SPI_FIFO_EV1 | SPI FIFO write event 1 [not yet documented] |
| $35(51)$ | (0_)DROVER | AD9910 ramp complete (DROVER pin) |
| $36(52)$ | (0_)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 $0_{-}$.

### 3.2.4 dcp wait: Timed waits or waiting for up to 2 events

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

TIME is the wait timeout in units of $1.024 \mu \mathrm{~s}$. Valid range is 0 to $2^{24}-1=16777215$, giving up to $\approx 16$ seconds with about $1 \mu$ s 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.

EVO 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 18) 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 O wait:1000:
dcp 0 wait:1000h:
dcp 0 wait:1000:DROVER,36
dcp 0 wait::BNC_IN_B_RISING
dcp 0 wait:1000h:u
dcp 0 wait::BNC_IN_C_RISING:u
wait for about 1024 us (on channel 0)
wait for about 8000 ns
wait for event 35 or 36 with a 1024 us timeout
wait for rising edge on BNC input B
wait for about }8000\textrm{ns}\mathrm{ , trigger IO_UPDATE afterwards
wait for rising edge on BNC input C, then trigger IO_UPDATE
```


### 3.2.5 dcp update: Update basic settings and pins

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

SPEC specifies what to update or change. Multiple settings can be concatenated and will all be carried out simultaneously. The prefix symbol specifies whether to set/increment $(+)$, or to clear/decrement (-) or to toggle ( ${ }^{\wedge}$ ).
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.
$+/-/{ }^{\wedge}$ h Set/clear/toggle the DRHOLD pin of the AD9910.
$+\mathrm{p} \quad$ Increment the value at the PROFILE2:0 pins of the AD9910.
-p Increment the value at the PROFILE2:0 pins of the AD9910.
$=N \mathrm{p} \quad$ Set the value at the PROFILE2:0 pins of the $N(0 \ldots 7)$.
+/-/~a Set/clear/toggle the BNC A pin of the DCP channel. (*)
+/-/^b Set/clear/toggle the BNC B pin of the DCP channel. (*)
$+/-/{ }^{\text {c }}$ 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 3.4, registers CFG_BNC_A, etc.

Examples:

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

### 3.2.6 dcp status: Print current status information

```
dcp status
```

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 | $0 \mathrm{E}--\mathrm{x}$ | $0 \mathrm{E}-\mathrm{x}$ | 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 DCP Program Examples

DCP execution follows a deterministic timing. Hence, instructions are typically fed into the DCPs (DDS Command Processors) first and then executed by starting the DCP.
This chapter presents a couple of examples how to use the FlexDDS-NG-1GS AD9910 RF generator slots.

## Note: USB versus network interface

Please note that when using the USB interface, the dcp start command is mandatory while when using the network interface for the rack, the DCPs start up running and will execute right away and there is no "start" command. This means, in order to synchronize to the external world, a program fed into the rack would typcially have something like a dcp wait: :BP_TRIG_A as the first command to wait for a rising edge on the BNC A input on the main rack control slot.

## Note: dds reset via network interface

When sending a dds reset over the network interface, one has to physically wait at least 100 ms before transmitting the following commands over the network.

Please also see chapter 5.5 on page 58 .

### 3.3.1 Basic: Two outputs with small frequency offset

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

```
dds reset
dcp 0 spi:stp0=0x3fff00002147ae14
dcp 1 spi:stp0=0x3fff00002147ae15
dcp update:u update AD9910 (both channels)
reset and initialize the DDS and also the DCP
set freq (FTW in STP0) to 130 MHz for ch 0
set freq (FTW in STP0) to 130 MHz + 0.23 Hz for ch 1
(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
```


### 3.3.2 Phase jump by $\pi$ after 2 seconds

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

```
dds reset
dcp 0 spi:stp0=0x3fff000033333333
dcp 1 spi:stp0=0x3fff0000333333333
dcp update:u
dcp 0 spi:stp0=0x3fff7fff33333333
dcp 1 spi:stp0=0x3fff000033333333
```

ch 0 , set freq. to 200 MHz , phase to 0 deg. ch 1 , set freq. to 200 MHz , phase to 0 deg. update both AD9910 to make the STPs effective ch 0 , set freq. to 200 MHz , phase to 180 deg. ch 1 , set freq. to 200 MHz , phase to 0 deg .
been loaded into the AD9910 already but is not yet effective, 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

### 3.3.3 Using the mirror frequency

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

| dds reset |  |
| :--- | :--- |
| dcp 0 spi:stp0 $=0 \times 3 f f f 000033333333$ | normal frequency $(200 \mathrm{MHz})$ |
| dcp 1 spi:stp0 $=0 \times 3 f f f 0000$ cccccccd | mirror frequency $(800 \mathrm{MHz})$ |
| dcp update:u |  |
| dcp start |  |

### 3.3.4 Wait for BNC inputs; externally triggering DDS changes

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 18).

```
dds reset
dcp 0 spi:CFR2=0x1000080
dcp O spi:stp0=0x3fff0000028f5c29
dcp O update:u
dcp 0 spi:stp0=0x3fff0000051eb852
dcp 0 wait::3
dcp 0 update:u
dcp 0 spi:stp0=0x1fff000007ae147b
dcp 0 wait::BNC_IN_A_RISING
dcp O update:u
dcp start
```

> All this is only done on channel 0.
> set CFR2 to matched latency and ASF from STP set STP0 to 10 MHz , full amplitude flush settings to AD9910, 10 MHz now at RF output set STP0 to 20 MHz , full amplitude wait for rising edge on BNC A input (event 3) IO_UPDATE the AD9910, 20 MHz now at RF output set STP0 to 30 MHz , half amplitude wait for rising edge on BNC A input (same as wait::3) IO_UPDATE the AD9910, 30 MHz , half ampl. at RF out

### 3.3.5 Frequency ramp up, then down, then again up

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

```
dds reset
dcp spi:CFR2=0x80
dcp spi:DRL=0x07ae147b051eb852
dcp spi:DRSS=0x0000001a0000000d
```

[^0]continued ...

```
dcp spi:DRR=0x00960096
dcp spi:CFR2=0x80080
dcp update:u+d
dcp wait:3000000:
dcp update:-d
dcp spi:DRL=0x1999999a051eb852
dcp spi:DRSS=0x0000001a00000081
dcp wait:2000000:
dcp update:u+d
dcp start
```

ramp rate 150 (up and down)
enable the ramp generator
do IO_UPDATE, set DRCTL HIGH to start upwards ramp
wait for about 3 seconds for ramp to complete
set DRCTL LOW to start downwards ramp
set upper ramp limit to 100 MHz (effective at next IO_UPDATE)
ramp step size to about 30 Hz up
wait 2 seconds for ramp to complete
set DRCTL HIGH again to sweep up to 100 MHz

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

### 3.3.6 Phase ramp

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

```
dds reset
dcp spi:STP0=0x3fff0000051eb852
dcp spi:CFR2=0x1000080
dcp update:u
dcp wait:500000:
dcp 0 spi:DRR=0x00960096
dcp 0 spi:DRSS=0x0000020000000200
dcp 0 spi:drl=0x7fffffff00000000
dcp 0 spi:CFR2=0x1180080
dcp 0 update:u+d
```

both channels: set STP0 to 30 MHz , full amplitude set CFR2 to matched latency and ASF from STP update; this makes the above appear at the RF outputs wait half a second channel 0 : prepare phase ramp: ramp rate. . . channel $0:$. . ramp step size and. . . channel 0: . . . ramp limits 0 to 180 degrees channel 0: enable ramp generator (destination: phase) update channel 0, DRCTL HIGH (upwards ramp) continued ...

### 3.3.7 Synchronizing two channels on the same slot

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
dcp update:u
dcp wait::BNC_IN_A_RISING
dcp spi:CFR1=0x00402000
dcp update:u
dcp 0 start
dcp 1 start
set CFR2 to matched latency and ASF from STP
update; this makes the above appear at the RF outputs
both DCPs wait for rising edge on BNC A
set autoclear phase bit (for next update)
IO UPDATE both channels simultaneously (re-starts phase accu)
we start the first DCP
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.
```


### 3.3.8 Amplitude ramp followed by a frequency ramp

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

1. Start at 30 MHz and full amplitude (initial state). (This is just for visualization, one could also start with amplitude 0.)
2. Wait for a rising edge trigger on 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
dcp 1 spi:CFR2=0x01000080
dcp 1 update:u
dcp 1 wait::BNC_IN_A_RISING
dcp 1 spi:DRR=0x00960096
dcp 1 spi:DRSS=0x0000020000000200
dcp 1 spi:drl=0x7fffffff00000000
dcp 1 spi:CFR2=0x01280080
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 }\mus\mathrm{ 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:
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
set STP0 to 30 MHz, full amplitude (initial state)
update; this makes the above appear at the RF outputs
wait for BNC A input rising edge trigger
prepare amplitude ramp: ramp rate...
... amp step size and ...
...ramp limits 0 to 50% amplitude
wait 1\mus to ensure frequency ramp has started and DROVER is 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.

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

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

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

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

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

$$
\operatorname{FTW}(f)=\frac{f \cdot 2^{32}}{1 \mathrm{GHz}}=\frac{7 \cdot 10^{6}}{10^{9}} \cdot 2^{32}=30064771=0 \mathrm{x} 01 \mathrm{cac} 083
$$

(This formula can be found in the AD9910 datasheet.)
Similarly, 7.05 MHz are 30279519 or 0x1ce075f.
For the amplitude ramp in dBm , the output of the FlexDDS-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
dcp O update:u
dcp start
```

set STP0 to 7 MHz (0x01cac083), full amplitude (0x3fff) update the AD9910 so that the STP0 takes effect; this starts the 7 MHz output
(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 $(+2 \mathrm{dBm}=$ $0 x 3 f f f=16383$ ), we can hence compute the necessary amplitude setting for any dBm value. This is called the ASF (amplitude scale factor):

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

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

$$
\operatorname{ASF}(-34 \mathrm{dBm})=10^{(-34-2) / 20} \cdot\left(2^{14}-1\right)=10^{-1.8} \cdot 16383=260=0 \times 0104
$$

Similarly, ASF $(-5 \mathrm{dBm})=7318=0 \times 1 \mathrm{c} 96$.
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:

$$
\mathrm{DRL}=\text { ASF }_{14 \mathrm{bits}}^{\text {high }} 0_{18 \text { bits }} \text { ASF }_{14 \mathrm{bits}}^{\text {low }} 0_{18 \text { bits }}=0 \times 72580000 \_04100000
$$

(Note that these numbers can easily be computed by shifting the computed values from before by 2 bits to the left: $0 \times 1 \mathrm{c} 96 \ll 2=0 \times 7258,0 \times 0104 \ll 2=0 \times 0410$.)
Similarly, for the frequency ramp, the DRL value is simply the two 32 -bit FTW values joined together:

$$
\mathrm{DRL}=\mathrm{FTW}_{32 \mathrm{bits}}^{\mathrm{high}} \mathrm{FTW}_{32 \mathrm{bits}}^{\mathrm{low}}=0 \times 01 \mathrm{ce} 075 \mathrm{f} \_01 \mathrm{cac} 083
$$

We now have to compute the ramp speed (i.e. ramp rate and step size). We have some freedom in this: We can either make a ramp with a few large steps or with many small steps. Both approaches have their pros and cons: Many small steps make for a smooth ramp but because the step size has finite precision the final ramp speed might significantly deviate from what we'd like to see. In contrast, a few large steps allow us to specify the step size very precisely but the ramp will look like a staircase. Usually, theres a good compromize in between.
The AD9910 datasheet specifies the ramp rate formula as follows $\left(f_{\text {SYSCLK }}=1 \mathrm{GHz}\right)$ :

$$
\Delta t=\frac{4 \cdot \mathrm{DRR}_{u p, \text { down }}}{f_{\mathrm{SYSCLK}}}=\mathrm{DRR}_{u p, d o w n} \cdot 4 \mathrm{~ns}
$$

The AD9910 datasheet also specifies the amplitude step size as:

$$
\text { AmplitudeStep }=\frac{\mathrm{DRSS}_{u p, \text { down }}}{2^{32}} I_{F S}
$$

where $I_{F S}$ is the electrical full scale amplitude.
It is important to understand that after each time $\Delta t$, the step size gets added to the ramp accumulator. Hence, for a ramp with $n$ steps, the total duration of the ramp will be $\Delta T=n \cdot \Delta t$ and the accumulator will have performed $n$-many steps of size DRSS. Hence,

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

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

Let's say we want to make $n=100000$ steps. With the formulas above we obtain:

$$
\mathrm{DRR}=\frac{\Delta t}{4 \mathrm{~ns}}=\frac{\Delta T}{n \cdot 4 \mathrm{~ns}}=\frac{3 \mathrm{~s}}{100000 \cdot 4 \mathrm{~ns}}=7500=0 \mathrm{x} 1 \mathrm{~d} 4 \mathrm{c}
$$

and

$$
\mathrm{DRSS}=2^{32} \frac{\mathrm{ASF}_{\text {end }}-\mathrm{ASF}_{\text {start }}}{2^{14} \cdot n}=2^{32-14} \frac{7318-260}{100000}=18502=0 \times 00004846
$$

Similarly we could try with $n=10000$ steps or $n=10^{6}$ steps, the results are summarized in the following table:

| $n$ | DRSS | DRR |
| :--- | ---: | ---: |
| $10^{4}$ | 185021 | 75000 |
| $10^{5}$ | 18502 | 7500 |
| $10^{6}$ | 1850 | 750 |
| $10^{7}$ | 185 | 75 |

Note that the first line is technically impossible because the ramp rate is limited at $2^{16}-1=65535$ but all others give very nice ramps, so we have a large freedom concerning the number of steps here.
In a similar fashion the frequency ramp parameters can be computed. Because the frequency change is rather small ( 50 kHz ) and the ramp duration rather long ( 5 seconds), choosing the best step size is a bit tricky this time. For the computation and reasons that will become clear later, let's choose $n=21450$ steps:

$$
\begin{gathered}
\mathrm{DRR}=\frac{\Delta T}{n \cdot 4 \mathrm{~ns}}=\frac{5 \mathrm{~s}}{21450 \cdot 4 \mathrm{~ns}}=58275=0 \mathrm{xe} 3 \mathrm{a} 3 \\
\mathrm{DRSS}=\frac{\mathrm{FTW}_{\mathrm{end}}-\mathrm{FTW}_{\text {start }}}{n}=\frac{30279519-30064771}{21450}=10.01=0 \times 0000000 \mathrm{a}
\end{gathered}
$$

Here is what would happen for different number of steps:

| $n$ | DRSS | DRR |
| :---: | ---: | ---: |
| $10^{4}$ | 21.47 | 125000 |
| 21450 | 10.01 | 58275 |
| $10^{5}$ | 2.14 | 12500 |
| $10^{6}$ | 0.21 | 1250 |

Note that the DRSS values are integers, the fractional values are only shown to illustrate the rounding error. Here we see that because of the long and shallow ramp, the ramp generator is close to its limits and the number of steps has to be chosen carefully: With too few steps, the ramp rate would overflow (first line). With too many steps, the step size gets very small (last two lines). A value of 0 disables ramping altogether but even very small values like 2.14 have a very big rounding error ( $\mathrm{DRSS}=2$, error $7 \%$ ). Hence, the number 21450 was chosen empirically to obtain a nice figure for the step size with very little rounding error (0.1\%).
The necessary instructions to perform this task are listed below. All these instructions are fed into the FlexDDS-NG at the beginning. Once the last instruction ("dcp start") is received, the program is executed.

```
dds reset
dcp 0 spi:CFR2=0x01000080
dcp 0 spi:stp0=0x0104000001cac083
dcp O update:u-d
dcp 0 spi:DRL=0x72580000_04100000
```

set CFR2 to matched latency and ASF from STP
set STP0 to 7 MHz (0x01cac083), -34 dBm amplitude (0x0104) 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: set amplitude ramp limits ( $0 \times 1 \mathrm{c} 96 \ll 2=0 \times 7258$, see above)

```
dcp O spi:DRSS=0x00004846_00004846
dcp 0 spi:DRR=0x1d4c_1d4c
dcp 0 spi:CFR2=0x00280080
dcp O wait::BNC_IN_A_RISING
dcp O update:u+d
dcp 0 spi:CFR2=0x01000080
dcp 0 spi:stp0=0x1c96000001cac083
dcp 0 wait::DROVER
dcp O update:u-d
dcp 0 spi:DRL=0x01ce075f01cac083
dcp O spi:DRSS=0x0000000a0000000a
dcp 0 spi:DRR=0xe3a3e3a3
dcp O spi:CFR2=0x01080080
dcp O update:u+d
dcp O spi:CFR2=0x01000080
dcp 0 spi:stp0=0x1c96800001ce075f
dcp O wait::DROVER
dcp O update:u-d
dcp 0 spi:stp0=0x0000800001ce075f
dcp 0 wait:1000000:
dcp O update:u
dcp start
```

set ramp step size (see above) set ramp rate (see above) enable ramp generator with destination amplitude wait for rising edge TTL trigger on BNC input A 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: disable the ramp again and enable ASF from STP again tune STP0 to match current output ( $7 \mathrm{MHz},-5 \mathrm{dBm}$ ) wait until the amplitude ramp completes and immediately activate the pre-loaded settings (output stays unchanged) start configuring the frequency ramp set the frequency ramp limits (see above) set ramp step size (see above) set ramp rate (see above) enable ramp generator with destination frequency update the AD9910, start the ramp via +d (DRCTL set HIGH) immediately pre-load the new settings while ramp is running: disable the ramp again set the STP0 to the settings after the ramp but with a phase offset word of $\pi$ (i.e. POW $=0 \times 8000$ ) wait until the frequency ramp is over update to the new settings to perform the phase jump pre-load the next settings:
turn output off by setting the ASF to zero wait for 1 second update; this will switch off the output

### 3.3.10 Working with the AD9910: Frequency ramps

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
dcp spi:CFR2=0x80080
dcp update:u+d
dcp wait:2000000:
dcp update:-d
dcp start
```

set ramp rate
enable ramp
do IO_UPDATE, set DRCTL HIGH to start upwards ramp
wait for the ramp to sweep the frequency
change direction to downwards, DRCTL LOW

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 | we start with taking DRCTL HIGH |
| dcp update:u+d | this delay can be left out |
| dcp wait:2000000: | take DRCTL LOW for downwards ramp |
| dcp update:-d |  |
| dcp start |  |

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

This is a real world example: We want to wait for a trigger on BNC A, once it is received we switch on channel 1 at 75.5 MHz , then wait for the next trigger on BNC A. Once received, we perform a frequency ramp to 83.0 MHz and finally after the next trigger switch off.
This can be repeated multiple times without reset in between. Two important steps are: We set the downwards ramp rate and step size to something very large so that it can immediately jump back to the start of the ramp. And we take the DRCTL pin low at the end of the ramp.

```
dds reset
dcp 1 spi:CFR1=0x400000
dcp 1 spi:CFR2=0x01000080
dcp 1 spi:STP0=0x0FFC00001353F7CF
dcp 1 wait::BNC_IN_A_RISING
dcp 1 update:u
dcp 1 wait:1h
dcp 1 spi:DRL=0x153F7CEE1353F7CF
dcp 1 spi:DRSS=0x10000000000006FF
dcp 1 spi:DRR=0x0001008B
dcp 1 spi:CFR2=0x01080080
dcp 1 wait::BNC_IN_A_RISING
dcp 1 update:u+d
dcp 1 wait:1h:
dcp 1 wait::DROVER
dcp 1 spi:CFR1=0x400000
dcp 1 spi:CFR2=0x01000080
dcp 1 spi:STPO=0x0000000013333333
dcp 1 wait::BNC_IN_A_RISING
dcp 1 update:u-d
dcp 1 wait:1h:
dcp start
```

set 75.5 MHz
wait for trigger
update to have 75.5 MHz at the output
setup ramp from 75.5 to 83.0 MHz
set step size (downwards: very large steps!)
set ramp rate (downwards: very quick!)
enable ramp
wait for trigger
actually start frequency ramp after trigger
wait until ramp is over
disable ramp
set amplitude to zero
wait for trigger
after trigger, update and take DRCTL low

### 3.3.12 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 $\left({ }^{* *}\right)$ below). To use more samples within the same time, the step rate has to be reduced accordingly.

```
dds reset
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
    0, zero crossing: 0, mode control: 1 (UP)
```

dcp 0 update:=1p
dcp 0 update:=0p

``` 0 , zero crossing: 0 , mode control: 1 (UP)
```

dcp 0 spi:CFR1=0xc0416002
dcp 0 spi:CFR2=0x004e0cc0
dcp 0 spi:RAMB=0:c
dcp 0 spi:RAM64C=0x00000000_00277872:c
dcp 0 spi:RAM64C=0x009dc970_0162aa03:c
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
dcp 0 spi:RAM64C=0x1d0dfe53_21288376:c
dcp 0 spi:RAM64C=0x257d8665_2a0a5b2d:c
dcp 0 spi:RAM64C=0x2ecc336b_33c0200c:c
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:ASF=0 set zero amplitude (needed for OSK)
step rate: $0 x 14=20\left(65535\right.$ max) , start adr: 0 , end adr: $127\left({ }^{* *}\right)$ (max. is 1023) no dwell high:
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.
set up the RAM profile 0 (i.e. STP0):
set zero amplitude (needed for OSK)
enable OSK, inverse SINC filter, sine output, autoclear phase
program the ramp into the DDS ramp generator 0 to 15 MHz
set up the RAM profile 0 (i.e. STP0):
step rate: $0 \mathrm{x} 14=20(65535 \mathrm{max})$, start adr: 0 , end adr: $127\left(^{* *}\right)$ (max. is 1023) no dwell high:
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.

## profile also acts as a trigger and would otherwise enable RAM playback too early.

switch profile forth and back; probably not needed for...
$\ldots$ profile 0 ; done before updating CFR1 as changing the
disable OSK and enable RAM; we do this before storing
the RAM content but will not take effect until UPDATE enable ramp generator (no-dwell low and high)
begin storing the amplitude shape in SRAM
first 2 words of the Hann shape
next 2 words of the Hann shape...

Note that the way the profile is set up, the playback in the AD9910 will be bottom-to-top and not top-to-bottom, so the listing is "reversed". For the Hann shape it does not make a difference because it's symmetric.

32

```
dcp O spi:RAM64C=0xf641af3a_f8848411:c
dcp 0 spi:RAM64C=0xfa7d0559_fc29fbec:c
dcp 0 spi:RAM64C=0xfd8a5f3d_fe9d55fa:c
dcp 0 spi:RAM64C=0xff62368d_ffd8878b:c
dcp O spi:RAM64C=0xfffffffe_ffd8878b:c
dcp O spi:RAM64C=Oxff62368d_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 O 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 O 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 O spi:RAM64C=0x38e31318_33c0200c:c
dcp 0 spi:RAM64C=0x2ecc336b_2a0a5b2d:c
dcp 0 spi:RAM64C=0x257d8665_21288376:c
dcp 0 spi:RAM64C=0x1dOdfe53_19307edf:c
dcp O 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
dcp O update:u
dcp start
```

The final waveform looks like this:

last 2 words to store in SRAM (END); no ":c" at the end UPDATE to start ramp and SRAM playback

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

```
dds reset
dcp 0 spi:ASF=0
dcp 0 spi:CFR1=0x00412202
dcp 0 spi:FTW=0x147ae148
dcp 0 spi:stp0=0x141fc0000001
dcp 0 update:=1p
dcp 0 update:=0p
dcp 0 spi:CFR1=0xc0416002
dcp 0 spi:CFR2=0x00_40_0c_c0
set constant frequency }82\textrm{MHz
do NOT enable ramp generator
                                    continued ...
```

```
dcp 0 spi:RAMB=0:c
```

...

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

```
dds reset
dcp 0 spi:ASF=0
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
dcp 0 wait:1000000:
dcp 0 update:=1p-d
dcp 0 update:=0p+d
dcp 0 wait:1000000:
dcp 0 update:=1p+d
dcp 0 update:=0p-d
dcp 0 wait:1000000:
dcp 0 update:=1p-d
dcp 0 update:=0p+d
dcp start
copied from above...
```

... until here, first puse generated. wait for 1 second switch profile and DRCTL forth... ... and back again to trigger next pulse wait for 1 second switch profile and DRCTL forth...
$\ldots$ and back again to trigger next pulse and so on... note that the DRCTL is switched alternatingly

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
dcp 0 spi:CFR1=0x00412202
dcp 0 spi:drss=0x700f4240000f4240
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 spi:CFR1=0xc0416002
dcp 0 spi:CFR2=0x004c0cc0
dcp 0 wr:CFG_DRCTL=0b010_0_000_00101
dcp 0 wr:CFG_PROFILE=0b000_000_010_001
dcp start
```

most of this is copied from above...
here we make the downwards ramp step size huge
this avoids a glitch at the first pulse
here we set no-dwell only for HIGH
route BNC A to DRCTL
route BNC A to PROFILE0, inverted

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 PROFILEO.

### 3.3.13 Analog frequency modulation with digitally controlled amplitude and phase

The following example will perform an analog frequency modulation based on the analog input Ch 0 and at the same time allows to load 8 different amplitude and phase settings from the 8 profile registers via the levels on the 3 BNC inputs.
To test this example, hook up an oscilloscope on RF out 0 , supply a 1 Hz sine wave with $2 \mathrm{~V}_{\mathrm{pp}}$ to analog input 0 and input a TTL rectangle signal into BNC inputs $\mathrm{A}, \mathrm{B}$ and C . The 3 BNC inputs correspond to the 3 profile select bits.

It is important to "enable amplitude scale factors from single tone profiles" (CFR2 bit 24) and disable OSK.

```
dds reset
dcp 0 spi:FTW=0x266663e2
dcp 0 spi:stp0=0x3000_0000_266663e2
dcp 0 spi:stp1=0x2a00_2000_166663e2
dcp O spi:stp2=0x2600_4000_166663e2
dcp 0 spi:stp3=0x2200_6000_166663e2
dcp 0 spi:stp4=0x1c00_8000_166663e2
dcp 0 spi:stp5=0x1800_a000_166663e2
dcp 0 spi:stp6=0x1400_c000_166663e2
dcp 0 spi:stp7=0x1000_e000_166663e2
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp O spi:CFR2=0x01000019
dcp 0 wr:AM_S0=0x1000
dcp 0 wr:AM_00=0x8000
dcp 0 wr:AM_CFG=0x2000_0002
dcp 0 update:u
dcp 0 wr:CFG_PROFILE=0b010_010_010_000
dcp start
```


### 3.4 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.

### 3.4.1 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. <br> Defl. | WSCT | - | - | - | - | - | - | - | - | - | - | - | - | - | - |  |


| 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 3.31.

| Num | Name | Description |
| ---: | :--- | :--- |
| $0 \ldots 63$ | (Events) | Same as the events in Table 3.8 (page 18) |
| 92 | ADC_CHO_SIGN | Sign bit of the ADC channel 0 conversion result |
| 93 | ADC_CH1_SIGN | Sign bit of the ADC channel 1 conversion result |
| $100 \ldots 102$ | DCPO_BNC_A,B,C | BNC A,B,C output from the channel 0 DCP |
| $103 \ldots 105$ | DCP1_BNC_A,B,C | BNC A,B,C output from the channel 1 DCP |
| 126 | FADC_CLK_TGL | ADC clock divided by $2,180^{\circ}$ phase uncertainty (DEBUG ONLY!) |
| 127 | DCP_CLK_TGL | DCP clock divided by $2,180^{\circ}$ phase uncertainty (DEBUG ONLY!) |

Table 3.31: Output mux choices: Values for BNC_OUT_MUX. Choose which signal is routed out via the BNC connector.

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

```
dcp O wr:CFG_BNC_A=0x200
```

dcp 0 wait:1000:
dcp 0 wr:CFG_BNC_A=0x300
dcp 0 wait:1000:
dcp 0 wr: $0 x 080=0 x 200$
dcp 0 wait:1000:
dcp 0 wr:cfg_bnc_a=^0x100
dcp 0 wait:200:
dcp 0 wr:cfg_bnc_a=^0x100
dcp 0 wait:200:
dcp 0 wr:cfg_bnc_a=+0x100
dcp 0 wait:200:
dcp 0 wr:cfg_bnc_a=-0x100
dcp start
configure BNC A port as output (DIR=1), LOW (OUT_MUX=0)
wait about 1 ms configure BNC A as output, inverted; port will go HIGH.
same as first line with numeric register address; port goes LOW again
toggle the INV bit; will go from 0 to 1 ; port goes HIGH wait about $200 \mu \mathrm{~ms}$
toggle the INV bit again back to 0; port goes LOW
set the INV bit; port goes HIGH
clear the INV bit; port goes LOW

### 3.4.2 CFG_BNC_B: Configure BNC B

Address: 0x081
Access: Write, Set, Clear, Toggle; DCP 0 only
See the description for CFG_BNC_A above.

### 3.4.3 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
dcp 0 spi:stp0=0x3fff000010000000
dcp 0 wait::BNC_IN_B_RISING
dcp O update:u
dcp 0 spi:stp0=0x3fff000020000000
dcp 0 wait::BNC_IN_B_RISING
dcp O update:u
dcp start
```


### 3.4.4 CFG_OSK: Configure Routing to the OSK Pin on the AD9910

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

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


| 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 $\left(^{*}\right)$
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 $\left({ }^{* *}\right)$ 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 18), 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
dcp 0 spi:asf=0xffffffff
dcp 0 spi:stp0=0x3fff00001999999a
dcp O update:u
dcp 0 wr:CFG_OSK=ObO10_0_000_00101
dcp start
```

```
set OSK bits (see text above)
set full amplitude
ch 0, set STP0 to 100 MHz (ampl. irrelevant)
flush settings in the AD9910
route BNC A to the OSK pin (*)
```

(*) Remember that the underscores are just for readability, one could also write 0b010000000101 or $0 \times 405$.
The logic can be inverted by setting the INV bit (CFG_OSK=Ob010_1_000_00101)

### 3.4.5 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. <br> Defl. | WSCT | - | - | - | - | - | - | - | - | - | - | - | - | - | - |  |


| 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.

### 3.4.6 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. <br> Defl. | WSCT | - | - | - | - | - | - | - | - | - | - | - | - | - | - |  |


| 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
dcp 0 spi:CFR2=0x1000080
dcp O update:u
dcp 0 wait:500000:
dcp 0 spi:DRL=0x1999999a051eb852
dcp 0 spi:DRSS=0x7000001a00000008
dcp 0 spi:DRR=0x00010008
dcp 0 spi:CFR2=0x10c0080
dcp 0 wr:cfg_drctl=0b0100_00000101
dcp O update:u
dcp start
```

set 30 MHz , full amplitude
set single tone ASF bit and matched latency update AD9910
wait for half a second (for demo purposes)
prepare ramp limits to 30 MHz and 100 MHz ramp step sizes: slow upwards, instant downwards prepare ramp rate
enable ramp and set no-dwell high bit
route BNC A input to DRCTL pin of AD9910
IO_UPDATE to commit register changes

### 3.4.7 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. <br> Defl. | WSCT | - | - | - | - | - | - | - | - | - | - | - | - | - | - |  |


| 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.

### 3.4.8 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. <br> Defl. | WSCT | - | - | - | - | - | - | - | - | - | - | - | - | - | - |  |


| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Desc. | - | - | - | - | OPMODE2 |  | OPMODE1 | OPMODEO | INV2 | INV1 | INVO |  |  |  |  |  |
| 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.
OPMODEn $\quad$ Choose operation mode for profile pin $n(0,1,2)$ : 000: connect the DCP to the PROFILEn directly (default) (*).
001: disconnect DCP and force the PROFILEn pin to 0/LOW (*)
010: disconnect DCP and route the BNC input $n(\mathrm{~A}, \mathrm{~B}, \mathrm{C})$ into the PROFILEn pin $\left(^{*}\right)$
INVn When set, allows logical inversion, one bit for each PROFILE pin. The OPMODE choices marked $\left(^{*}\right)$ 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
dcp 0 spi:stp1=0x1fff00004ccccccd
dcp 0 update:u
dcp 0 wr:CFG_PROFILE=0b000_000_010_000
dcp start
```

ch 0 , set STP0 to 100 MHz , phase to 0 deg. full amplitude ch 0 , set STP1 to 300 MHz , phase to 0 deg. half amplitude flush settings in the AD9910
route BNC A to the PROFILE0 pin $\left(^{*}\right.$ )
(*) Remember that the underscores are just for readability, one could also write 0b000000010000 or $0 \times 010$.

### 3.4.9 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. <br> Defl. | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |


| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Desc. <br> Defl. | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | DDS_RESET <br> 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
dcp 0 spi:CFR2 $=0 \times 01000080$
set STP0 to 30 MHz , full amplitude
set CFR2 to matched latency and ASF from STP
continued ...

## dcp 0 update:u

dcp 0 wait:1000000:
dcp 0 wr:DDS_RESET=1
dcp start
update; this makes the above appear at the RF outputs wait a second
now reset channel 0

### 3.4.10 AM_SO: 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. <br> Defl. | - | - | UPD | - | - | - | - | - | - | - | - | - | - | - |  | $\rightarrow$ |
| $\rightarrow$ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



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.

### 3.4.11 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. Defl. | - | - | UPD | - | - | - | - | - | - | - | - | - | - | - |  | $\rightarrow$ |
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Desc. Defl. | $\leftarrow$ AM_S1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Sets the scaling factor $S_{1}$ for the respective DCP associated with analog input channel 1. For details, see AM_SO.

### 3.4.12 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 | - | - | - | - | - |  |  |  |  |  |  |  | $\rightarrow$ |
| Defl. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |


| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Desc. | $\leftarrow$ AM_0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Defl. | $\leftarrow$ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

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.

### 3.4.13 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. <br> Defl. | - | - | UPD | - | - | - | - | - |  |  |  |  |  |  |  |  |
| $\rightarrow$ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |


| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---: | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| Desc. | $\leftarrow$ AM_P | $\mid$ |  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |
| Defl. | $\leftarrow 0$ |  |  |  |  |  |  |  | 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.

### 3.4.14 AM_00: 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. <br> Defl. | - | - | UPD | - | - | - | - | - | - | - | - | - | - | - |  | $\rightarrow$ |
| $\rightarrow$ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |


| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Desc. | $\begin{array}{ccc}\leftarrow \\ \leftarrow & \text { AM_00 }\end{array}$ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Defl. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

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.

### 3.4.15 AM_01: Analog Modulation, Offset for Input Channel 1

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


Sets the offset value $O_{1}$ for the respective DCP associated with analog input channel 1. For details, see AM_OO.

### 3.4.16 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. <br> Defl. | - | - | UPD | - | - | - | - | - | - | - | - | - | - | - | - | - |


| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Desc. <br> Defl. | - | - | - | - | - | - | - | - | - | - | - | - | - | - | MODF |  |
| 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 $\quad$ Set the analog modulation format, i.e. the F bits of the parallel data bus into the AD9910: MODF Analog modulation format
$00 \quad$ Amplitude modulation (upper 14 bits used)
$01 \quad 16$ 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 4

## Analog Modulation

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

## Note: Minimum Required Firmware Version for Analog Modulation

The FlexDDS-NG slot firmware versions prior to 0.70 (Dec 2022) may experience glitches in the analog modulation. If you intend to use the analog modulation feature, especially on RF channel 1 , it is highly recommended to perform a firmware update in case your firmware version is below 0.70. For users of the FlexDDS-NG rack: The slot firmware update is included in the rack firmware update; so make sure you have rack version 0.70 or above.

The analog signal at each of the RF In ports is digitized with a dedicated ADC operating at $62.5 \mathrm{MS} / \mathrm{s}$ (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 $\pm 0.5 \mathrm{~V}=1 \mathrm{~V}_{\mathrm{pp}}$.
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.

### 4.1 Amplitude, Phase and Frequency Modulation

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

$$
\begin{equation*}
D=\operatorname{coerce}_{16}\left(\frac{\left(A_{0}-O_{0}\right) \cdot S_{0}+\left(A_{1}-O_{1}\right) \cdot S_{1}}{2^{12}}+O\right) \tag{4.1}
\end{equation*}
$$

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 $2^{16}-1=65535$.

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

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_00, AM_01, AM_S0, AM_S1 and AM_0. 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 | $0 x 00000$ | $0 x 000000$ | positive numbers are just like... |
| 1 | $0 x 00001$ | $0 x 000001$ | $\ldots$ regular binary representation |
| 2 | $0 x 00002$ | $0 x 000002$ |  |
| 256 | $0 x 00100$ | $0 x 000100$ |  |
| 131071 | $0 x 1 f f f f$ | $0 x 01 f f f f$ | largest 18 bit signed number $\left(2^{17}-1\right)$ |
| 8388607 | - | $0 x 7 f f f f f$ | largest 24 bit signed number $\left(2^{23}-1\right)$ |
| -1 | $0 x 3 f f f f$ | $0 x f f f f f f$ |  |
| -2 | $0 x 3 f f f e$ | $0 x f f f f f e$ |  |
| -256 | $0 x 3 f f 00$ | $0 x f f f f 00$ |  |
| -131072 | $0 x 20000$ | $0 x f e 0000$ | smallest 18 bit signed number $\left(-2^{17}\right)$ |
| -8388608 | - | $0 x 800000$ | smallest 24 bit signed number $\left(-2^{23}\right)$ |

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: $0 \mathrm{~b} \underline{0} 101 \rightarrow 0 \mathrm{~b} \underline{00000101}$ (positive value), $0 \mathrm{bb} 1101 \rightarrow 0 \mathrm{~b} 11111101$ (negative value)) The 16 bit analog sample values $A_{n}$ are sign-extended to 18 bits before performing the subtraction with offset $O_{n}$.

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

### 4.2 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 $1 \mathrm{~V}_{\mathrm{pp}}$ 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 unsigned range $0 \ldots 65525$, we need to add $65535 / 2=2^{15}$. Hence, the desired linear transfer function must look like this:

$$
\begin{equation*}
D=\frac{A_{0}}{1}+2^{15}=\frac{\left(A_{0}-0\right) \cdot 2^{12}+\left(A_{1}-0\right) \cdot 0}{2^{12}}+2^{15} \tag{4.3}
\end{equation*}
$$

By comparing coefficients with equation 4.1, we find that:

$$
\begin{array}{lll}
O_{0}=0 & S_{0}=2^{12}=0 \times 1000 & O=2^{15}=0 \times 8000 \\
O_{1}=0 & S_{1}=0 &
\end{array}
$$

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
dcp O spi:CFR1=Ob00000000_01000001_00000000_00000000
dcp 0 spi:CFR2=Ob00000000_00000000_00000000_01010000
dcp 0 wr:AM_S0=0x1000
dcp 0 wr:AM_OO=0
dcp 0 wr:AM_D=0x8000
dcp 0 wr:AM_CFG=0x2000_0000
dcp O update:u
dcp start
```

set frequency to 62.5 MHz , full amplitude sinc filter and sine output (not needed) enable parallel data port set scale factor $S_{0}$ set offset $O_{0}$ set global offset $O$ choose amplitude modulation, flush coeff. update AD9910 (make CFR* effective)

Here, the AM_S1 and AM_01 registers are not written so their default value of 0 is used. To view the result, connect a 1 MHz sine wave signal with $1 \mathrm{~V}_{\mathrm{pp}}$ amplitude (into $50 \Omega$; this may require to set an amplitude of $2 \mathrm{~V}_{\mathrm{pp}}$ 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_00 have to be replaced to writes to AM_S1 and AM_01.

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:STPO=0x3fff_0000_10000000
ch 0 : frequency 62.5 MHz , full amplitude continued...

```
dcp 1 spi:STP0=0x3fff_0000_08000000
dcp spi:CFR1=0b01000001_00000000_00000000
dcp spi:CFR2=0b00000000_00000000_01010000
dcp 0 wr:AM_SO=0x1000
dcp 0 wr:AM_O0=0
dcp 0 wr:AM_O=0x8000
dcp 0 wr:AM_CFG=0x2000_0000
dcp 1 wr:AM_S1=0x800
dcp 1 wr:AM_01=0
dcp 1 wr:AM_O=0xc000
dcp 1 wr:AM_CFG=0x2000_0000
dcp update:u
dcp start
```

ch 1: frequency to 31.25 MHz , full amplitude sinc filter and sine output (not needed) enable parallel data port ch 0: set scale factor $S_{0}$ ( ch 0: set offset $O_{0}$ ch 0: set global offset $O$
ch 0 : choose amplitude modulation, flush coeff.
ch 1: set scale factor $S_{1}$ (half the modulation depth)
ch 1: set offset $O_{1}$
ch 1: set global offset $O$ (larger offset)
ch 1: choose amplitude modulation, flush coeff.
update AD9910 (make CFR* effective)

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_01 with AM_S0 and AM_00 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:

$$
\begin{array}{lll}
\text { DCP 0: } & S_{0}=+2^{12}=0 \times 1000 & O=2^{15}=0 \times 8000 \\
\text { DCP 1: } & S_{0}=-2^{12}=0 \times 3 \mathrm{f} 000 & O=2^{15}=0 \times 8000
\end{array}
$$

Note that we could also set $S_{0}$ to 0 xff000 instead of 0 x 3 f 000 because the register is 18 bits wide and the most significant non-zero bits of $0 x f f 000$ would be truncated, leaving an effective register value of $0 x 3 f 000$.

Here's the corresponding instruction listing:

```
dds reset
dcp 0 spi:STP0=0x3fff_0000_10000000
dcp 1 spi:STPO=0x3fff_0000_08000000
dcp spi:CFR1=0b01000001_00000000_00000000
dcp spi:CFR2=0b00000000_00000000_01010000
dcp 0 wr:AM_SO=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 S S S = -2 12 = 0x3f000
dcp 1 wr:AM_O0=0
dcp 1 wr:AM_O=0x8000
dcp 1 wr:AM_CFG=0x2000_0000
dcp start
```

dcp update:u update AD9910 (make CFR* effective)
ch 0 : frequency 62.5 MHz , full amplitude ch 1: frequency to 31.25 MHz , full amplitude sinc filter and sine output (not needed) enable parallel data port
$S_{0}=+2^{12}=0 \times 1000$
$S_{0}=-2^{12}=0 \times 3 f 000$
update AD9910 (make CFR* effective)

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

| DCP $0:$ | $S_{0}=+4000=0 \times f a 0$ | $O=2^{15}=0 \times 8000$ |
| :--- | :--- | :--- |
| DCP 1: | $S_{0}=-4000=0 \times 3 f 060$ | $O=2^{15}=0 \times 8000$ |

### 4.3 Example: Phase Modulation

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

```
dds reset
dcp spi:STP0=0x3fff_0000_03000000
dcp spi:CFR1=0b01000001_00000000_00000000
dcp O spi:CFR2=Ob00000000_00000000_01010000
dcp 0 wr:AM_SO=0x1000
dcp 0 wr:AM_OO=0
dcp 0 wr:AM_O=0x8000
dcp 0 wr:AM_CFG=0x2000_0001
dcp update:u
dcp start
```

set frequency to 11.7 MHz , full amplitude sinc filter and sine output (not needed) enable parallel data port same full-scale modulation parameters. . .
... as in the example above
choose phase modulation, flush coefficients update AD9910 (make CFR* effective)

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

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

### 4.4 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 $10 \mathrm{MHz}(\mathrm{FTW}=0 \times 028 \mathrm{f} 5 \mathrm{c} 29)$ while positive full scale input $(+0.5 \mathrm{~V})$ should yield 30 MHz $(\mathrm{FTW}=0 \mathrm{x} 07 \mathrm{ae} 147 \mathrm{~b})$.
To cover the required frequency range, we need an FM gain setting of at least 11 (see AD9910 datasheet). We choose 12, although 11 would work just as fine.

In the presence of an FM gain of 12 , the 16 -bit digital modulation values are shifted by 12 bits (i.e. multiplied by $2^{12}$ ). Hence, the required 16-bit $D$ values from equation 4.1 need to cover the range from $0 \mathrm{x} 028 \mathrm{f} 5 \mathrm{c} 29 / 2^{12}=0 \mathrm{x} 28 \mathrm{f} 5$ to $0 \mathrm{x} 07 \mathrm{ae} 147 \mathrm{~b} / 2^{12}=0 \mathrm{x} 7 \mathrm{ae} 1$.

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

$$
\begin{equation*}
D=\operatorname{coerce}_{16}\left(\frac{A_{0} \cdot S_{0}}{2^{12}}+O\right) \tag{4.4}
\end{equation*}
$$

(Here, the $2^{12}$ in the denominator comes from equation 4.1 and is completely unrelated to the FM gain setting). A -0.5 V analog input correspond to an analog ADC value of $A_{0}=-2^{15}$ while a positive full
scale input of +0.5 V correspond to $A_{0}=+2^{15}-1$.
With that information we can now solve the following two linear equations to find $S_{0}$ and $O$ :

$$
\begin{gather*}
D\left(A_{0}=-2^{15}\right)=0 \times 28 \mathrm{f} 5=10485 \\
D\left(A_{0}=+2^{15}-1\right)=0 \times 7 \mathrm{ae} 1=31457  \tag{4.5}\\
S_{0}=\frac{(31457-10485) \cdot 2^{12}}{2^{15}-1+2^{15}}=1310.77=0 \times 51 \mathrm{f}  \tag{4.6}\\
O=\frac{10485 \cdot\left(2^{15}-1\right)-31457 \cdot\left(-2^{15}\right)}{2^{15}-1+2^{15}}=20971.16=0 \times 51 \mathrm{eb}
\end{gather*}
$$

Hence, we can instruct the FlexDDS-NG to perform the requested frequency modulation by executing:
dcp 0 spi:STPO=0x3fff000000000000
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp 0 spi:CFR2=0b00000000_00000000_01011100
dcp 0 wr:AM_S0=0x51f
dcp 0 wr:AM_OO=0
dcp 0 wr:AM_O=0x51eb
dcp 0 wr:AM_CFG=0x2000_0002
dcp 0 update:u
dcp start

```
```

```
dds reset
```

```
```

dds reset

```
choose max amplitude and set FTW to zero
enable parallel data port, set FM gain to 12
\(S_{0}\) as computed above
(zero analog channel offset)
\(O\) as computed above
choose frequency modulation, flush coefficients update AD9910 (make CFR* effective)

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 \(1 \mathrm{~V}_{\mathrm{pp}}\) square wave or a DC value, the frequencies can be measured easily with the oscilloscope.

\subsection*{4.5 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 4.1 but instead performs the following computation:
\[
\begin{array}{rll}
D_{7 \ldots 0} & =\operatorname{coerce}_{16}\left(\frac{\left(A_{0}-O_{0}\right) \cdot S_{0}}{2^{12}}+O\right) / 2^{8} & \text { phase bits } \\
D_{15 \ldots 8} & =\operatorname{coerce}_{16}\left(\frac{\left(A_{1}-O_{1}\right) \cdot S_{1}}{2^{12}}+P\right) / 2^{8} & \text { amplitude bits } \tag{4.7}
\end{array}
\]

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 O spi:stp0=0x0000000003000000
dcp 0 spi:CFR1=0b01000001_00000000_00000000
dcp 0 spi:CFR2=0b00000000_00000000_01010000
dcp 0 wr:AM_SO=0x1000
dcp 0 wr:AM_O0=0
dcp 0 wr:AM_O=0x8000
dcp 0 wr:AM_S1=0x1000
dcp 0 wr:AM_01=0
dcp 0 wr:AM_P=0x8000
dcp 0 wr:AM_CFG=0x2000_0003
dcp O update:u
dcp start

```
frequency 11.7 MHz ; phase and amplitude zero
enable parallel data port
scale factor for phase modulation (analog offset on input channel 0) offset value for phase modulation scale factor for amplitude modulation (analog offset on input channel 1) offset value for amplitude modulation choose polar modulation, flush coefficients update AD9910 (make CFR* effective)

\section*{Chapter 5}

\section*{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 FlexDDSNG 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.

\subsection*{5.1 The Main Slot Front Panel on the FlexDDS-NG Rack}

The mainslot (leftmost slot) of the FlexDDS-NG rack provides the following:
Power pushbutton (green): Push to power up and power down the FlexDDS-NG rack. While blinking, the rack is booting up. When constantly on, the rack has finished booting. When the rack is booting or performing a firmware update, or if it is in a bad state, it cannot be switched off by pressing the power button. In order to forcefully switch off, press the button for several seconds or use the hardware switch at the back. In standby mode, the power button is "blinking" with slowly varying intensity.
Reset pushbutton (red): Perform hardware reset of all slots. This is like power cycling the slots but not the rack. It will take the slots a few seconds to boot, run their power-on diagnosis and lock their clock generators to the backplane.
Reset BNC input: The BNC input next to the reset pushbutton allows to perform a hardware reset by applying a logic HIGH voltage (usually 5 V ). This is equal to pushing the reset button. The voltage level is 5 V logic by default. In order to opt for 3.3 V voltage levels, refer to figure 5.2.
Trig pushbutton (green): Manual trigger, same as applying a trigger to the BNC A port next to the


Figure 5.1: Frontpanel elements of the FlexDDS-NG rack main slot (leftmost slot).
pushbutton.
Trig BNC input: Next to the trig pushbutton. Currently unused. Same voltage level as the reset BNC input.

BNC A, BNC B: These are two independent trigger inputs that are routed along the backplane to all the slots. They are available as backplane trigger A and B in the slots. The voltage level is 5 V logic by default. In order to opt for 3.3 V voltage levels, refer to figure 5.2 .
10 MHz in (SMA connector): This is a 10 MHz reference input. To lock the complete FlexDDS-NG rack and all slots to an external 10 MHz reference, apply a high quality 10 MHz signal here. The maximum allowable frequency deviation is about 10 ppm . A sine wave amplitude of 100 mV is sufficient for locking althought for improved phase noise, higher amplitudes are recommended (up to beyond 2 V amplitude).
When the external 10 MHz reference is active, the green "LED B" will be on (firmware version 0.82 or above). Also, when connecting and disconnecting the external reference clock, the USB console will show messages indicating whether the external clock is detected and used.
Ref out (SMA connector): This is a copy of the internal 10 MHz reference clock used by the FlexDDS-NG rack. If an external 10 MHz input is connected, it will be phase locked to that. The output is an AC coupled square wave with an amplitude of about \(700 \mathrm{mV}_{\mathrm{pp}}\) into \(50 \Omega\).
LED Attn (red): Attention LED. On during boot and firmware updates and when an error occurs that prevents the rack from starting. Will blink on special hardware error.

LED Busy (green): Switched on while the rack is busy, e.g. booting.
LED A (red): Currently unused.

LED B (green): Emits green when the rack is locked to the external 10 MHz input. Requires rack firmware version at least 0.82 .

Ethernet connector (RJ-45): Connection for GBit Ethernet to send commands to the rack.
LVDS connector (RJ-45): This allows (in theory) to attach a special LVDS dongle for long signal transmission without reflection. Currently not implemented. Do not plug any network cable in here, it may destroy your network hardware!

USB host connector (USB-A): Currently unused.
Micro SD: Contains am micro-SD card of size 1 or 2 GBytes with FAT filesystem. Used to store the FlexDDS config files and firmware updates. Gently press to remove. Only remove when powered off. The SD card contacts face towards the text and away from the LVDS connector.


Figure 5.2: IO voltage selection jumpers for the main control slot on the FlexDDSNG rack. To remove the control slot, open the 4 screws holding the slot and pull it out using the provided handle. The BNC A and B have one common IO voltage setting and the BNC Trig and Reset have another independent setting. When installing, be sure to push using the provided handle to avoid bending the board.

\subsection*{5.2 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.

\section*{Note: Attention LVDS Port}

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".

\section*{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 5.3 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

```

\section*{Note: Insecure Networks}

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

\subsection*{5.3 The USB Console on the FlexDDS-NG Rack}

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

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

\subsection*{5.4 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 \(0 \ldots 5\) correspond to ports \(26000 \ldots 26005\), 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 5.3).

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 55 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 O 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


Figure 5.3: 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.
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.

\section*{Note: Association Between Slots and TCP Ports}

It is important to understand that each slot \((0 \ldots 5)\) 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 5.3.

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.)

\subsection*{5.5 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 \(75 f 4 \mathrm{a} 4 \mathrm{e} 10 \mathrm{dd} 4 \mathrm{~b} 6 \mathrm{~b} x\) 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 indivitual slots (or like the USB interface of the FlexDDS-NG DUAL).
Each command is terminated by a CR (' \(\backslash \mathrm{r}\) ') or LF (' \(\backslash \mathrm{n}\) ') 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 \(\rightarrow\) 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:
\[
\begin{array}{lll}
\text { dcp [0|1] } & \ldots & \text { Feed DCP instructions into the FIFO buffer of the slot. } \\
\text { dcp flush } & & \text { Flush DCP commands; done automatically after about } 1 \text { second. } \\
\text { dds [0|1] reset } & \text { Reset DDS channel(s); New in rack version } \mathbf{0 . 4 8} \text { (slot version } \mathbf{0 . 6 2} \text { )! } \\
\text { dds [0|1] r } & \text { Short form of the above. } \\
\text { set VAR=VALUE } & \text { Configure certain properties; see below. } \\
\text { quit } & \text { Close the current network connection. } \\
\text { reset } & \text { Like red pushbutton: Reset all slots, FIFO buffers and close all TCP connections. }
\end{array}
\]

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.

\section*{Note: Rack Limitation}

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.

\section*{Note: Resetting All Rack Channels}

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.

\section*{Note: Wait Time Required After Rack Reset}

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.

\section*{Note: Flushing Commands}

The FlexDDS-NG Rack with its network interface 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
Auth OK
dcp 0 spi:stp0=0x3fff00005c54943a
OK
set resp_suppress_ok=1
dcp 1 spi:stp0=0x3fff00005264943a
dcp update:u!
set dcp_dump_isn=1
dcp 0 wr:cfg_bnc_b=0x300
DCP: 0x0031208100000300

```
sent auth token for slot 1 ("password")
response that auth is OK
queue a DCP instruction for channel \(0(\operatorname{slot} 1)\)
response from rack
suppress "OK" responses
queue a DCP instruction for channel 1 (slot 1 )
queue DCP update insn for channels 0 and 1 (slot 1 )
request that DCP instructions are echoed back
set BNC B output HIGH on slot 1
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
Auth OK
set resp_suppress_ok=1
dcp 0 spi:stp0=0x3fff0000051eb852
dcp 0 spi:stp0=0x3fff0000071eb852
dcp spi:CFR2=0x01000080
dcp update:u!

```
sent auth token for slot 2 ("password")
response that auth is OK
suppress "OK" responses
queue a DCP instruction for channel 0
queue a DCP instruction for channel 1
next instruction channels 0 and 1
queue DCP update insn for channels 0 and 1
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.)

\subsection*{5.6 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 250000 DCP instructions per second or about \(9 \mathrm{MBytes} / \mathrm{s}\). (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 ( \(20 \mathrm{MBytes} / \mathrm{s}\) ). 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 \(26010 \ldots 26015\) 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 x 86 host computers and ARM processors).

The format of a network frame looks like this:


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

\section*{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 \(0 \mathrm{x} 75 \mathrm{f} 4 \mathrm{a} 4 \mathrm{e} 10 \mathrm{dd} 4 \mathrm{~b} 6 \mathrm{~b} x\) 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!)

\section*{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 11.

\subsection*{5.7 FlexDDS-NG Rack Phase Synchronization Accross Multiple Slots}

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

\section*{Note: Minimum Required Firmware Version for Multi-Channel Phase Sync}

The FlexDDS-NG rack firmware versions prior to 0.70 (Dec 2022) may show issues with phase synchronization between multiple slots. If you require phase synchronization between multiple slots, it is highly recommended to perform a firmware update if your version is below 0.70.

Concerning phase synchronization, it is important to understand:
- Known phase relationship can only be established if all slots to be synchronized are triggered by one of the the backplane trigger signals, labeled "BNC A" and "BNC B" on the main control slot next to the green pushbutton labeled "Trig". See example below.
- The waveform outputs from slot 0 to slot 5 will acquire an increasing delay of about 250 ps per slot position. This delay is caused by the signals travelling along the backplane in the rack. It is a constant delay in time and hence not a constant phase offset in degrees.
This delay can be compensated by chosing about \(45-50 \mathrm{~mm}\) of additional cable length for each slot position. E.g. if you are using slots 0 and 3, then the signal from slot 3 will be about 750 ps after those from slot 0 . To compensate, use about 15 cm more cable for the signal out of slot 0 . Precise slot-to-slot delays differ between racks, so to be certain it is recommended to verify timing with an oscilloscope. See example below. It can also be compensated by computing the corresponding phase offset word (which is frequency dependent) and programming it into the AD9910.

Here is an example how to test phase synchronization between multiple slots. When executed, it should generate the output shown in Figure 5.4.

For each slot \(\mathrm{N}(0 \ldots 5)\) to be synchronized, send the following text commands to TCP network port 2602N:
```

75f4a4e10dd4b6bN
Auth OK
set resp_suppress_ok=1
dcp X spi:CFR1=0x402000
dcp X spi:CFR2=0x1000080
dcp X spi:stp0=0x3fff000020000000
dcp wait::BP_TRIG_A:u!

```
sent auth token for slot \(\mathrm{N}(0 \ldots 5)\)
response that auth is OK
suppress "OK" responses
set auto-clear phase on update (bit 13) ( X is 0,1 or empty) set single tone ASF bit and matched latency full amplitude 125 MHz output wait for backplane trigger A, then update

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


Figure 5.4: Phase aligned output generated by the above exmple showing channel 0 of slots \(0,1,2\) and 3 (left) as well as slots \(0,1,4,5\) (right).

After having sent those commands to all respective slots (TCP ports), perform one rising edge trigger on the BNC A input of the control slot on the left or alternatively press once the green "Trig" pushbutton next to it. The green "Trig" pushbutton is internally wired to the same backplane trigger bus signal as the BNC A input. You should see the red "Trig" LEDs flash for each output on each slot you would like to synchronize. The output waveforms of all those outputs should now emit a 125 MHz sine wave with known and constant phase relationship between each other.
How this works: The update trigger is routed accross the backplane to all slots and channels selected and arrives synchronously. If you wire BNC inputs on slots independently, synchronization cannot be guaranteed. It is important to set the auto-clear phase accumulator bit (bit 13) in the CFR1 register. This ensures that all sine waveforms are reset to the same phase by the update event. The general concept is the same for more complex waveforms: Make sure all channels to be synchronized have the auto-clear phase bit (CFR1 bit 13) set and that they wait for an external backplane trigger (A or B) and then send that trigger signal.
In order to test repeatability, reset all slots using the red "Reset" pushbutton on the main control slot or by applying a reset signal to the "Reset" BNC input below that pushbutton. Then, repeat the above procedure.
Please note that the phase delay between slots might be larger or smaller than the 250 ps per slot position described above. No matter, it will still be constant and reproducible.
Here is a more fun example that first generates a moving phase and then synchronizes the channels:
```

75f4a4e10dd4b6bN
Auth OK
set resp_suppress_ok=1
dcp X spi:CFR1=0x402000
dcp X spi:CFR2=0x1000080
dcp X spi:stp0=0x3fff00002000000N
dcp wait::BP_TRIG_A:u
dcp X spi:stp0=0x3fff000020000000
dcp wait::BP_TRIG_A:u!

```
sent auth token for slot N (0...5)
response that auth is OK
suppress "OK" responses
set auto-clear phase on update (bit 13) ( X is 0,1 or empty)
set single tone ASF bit and matched latency
slightly different freq for each slot
wait for backplane trigger A, then update
slightly different freq for each slot
wait for backplane trigger A , then update

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

\subsection*{5.8 FlexDDS-NG Rack Configuration Options}

This chapter describes all the config options in the file flexdds.cfg on the micro-SD of the FlexDDS-NG Rack.

Here is a default config file with documentation in comments.
```


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

### Do not touch.

persistent_version_magic = 1

### Memory allocation

### =================

### FIFO size in kilo bytes for each slot.

### Each FIFO entry consumes 8 bytes (64 bit network frame).

### Minimum is }64\mathrm{ kBytes.

### Total sum must not exceed 786432 kBytes (768 Mbytes)

### Example: 64 10000 64646464

### -> Slot 1 has 10000 kBytes, all others have 64 kBytes

### May be changed by the user.

fifo_buf_size_kb = 8192 8192 8192 8192 8192 8192

### Ref clock frequency fine tuning if no external reference is used.

### Range 0..1023, default 518. Total range ca. 20 ppm (9.9999 to 10.0001 MHz)

### On racks with serials R25 and up, this has been factory tuned.

### Oder racks require a firmware update to have that option.

### May be changed by the user.

clkgen_freq_fine_tune = 518

### Do not change this:

clkgen_fast_lock_detune = 150

### Clock gen delay in half steps, starting with 0.

### 100 is 20ns; 1 is 200ps (5 GHz) approx 4 cm FR4. 0..4 is 1 cycle at 1 GHz

### DO NOT CHANGE!

clkgen_bp_out_delay = 0 0 0 0
clkgen_user_delay = 0
clkgen_fpga_delay = 25
clkgen_fpga_delay_50MHz = 40

### Debugging. DO NOT CHANGE!

dbg_dump_config_to_stdout = 0
dbg_skip_slot_init = 0
dbg_ignore_slot_mask = 0
dbg_trig_freq_force_50MHz = -1
dbg_exit_early = 0
dbg_dump_clkgen_status = 0

### EOF

```
```


[^0]:    All the following is done by both channels simultaneously:
    set matched latency (not needed) and ramp destination frequency ramp limits 20 MHz (low) and 30 MHz (high)
    ramp step size to about 6 Hz down and 3 Hz up

