PyDualDDS Documentation

Release 1.0

Andreas Fognini

Nov 06, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

Dual channel sine wave Direct Digital Synthesizer (DDS) from 0-4 GHz based on the DAC38RF82EVM.

This Python library allows to interface with the DAC38RF82EVM evaluation board. It is designed to control the numerically controlled oscillators (NCOs) to generate two independent sine wave signals in standalone mode, no FPGA card required. However, it is not limited to this use only, every register on the board can be written and read with this library too. The two output channels can each be independently controlled in frequency (0-4 GHz), phase (0-360 degrees), and amplitude (2048 steps).

1.1 Motivation

For a quantum optics experiment we needed a two channel phase and frequency controllable sine wave generator to drive an electro-optical-modulator at around 400 MHz. The software from TI does not readily allow to control the card through an API. Therefore, we have developed a control library of our own.

1.2 Setting up the card for standalone mode

To use the card in standalone mode, the card has to be configured to use the on board 122.88 MHz oscillator. For that the jumpers have to be set as follows:

Jumper	Position	Description
JP1	Shunt pin 2-3	Disable DAC sleep mode
JP2	Shunt pin 1-2	Enable DAC output
JP3	Close	Enable on board 122.88 MHz oscillator
JP8	Open	Enable VDDDIG1 supply
JP9	Open	Enable VEE18N supply
JP10	Open	Enable PLL clock mode

The library will configure the clock distribution chip (LMK04828) to generate a 1228.8 MHz = 10*122.88 MHz reference clock for the DAC38RF82. Subsequently, the DAC38RF82's internal PLL will be configured to generate the DAC's sampling rate running at 8847.36 MHz = 1228.8 MHz*9*4/5.

1.3 Usage of the library

The usage of the library is outlined in an example of setting both output channels to 397.76 MHz, zero phase, and an amplitude of 1.

```
from pydualdds import DDS

dac = DDS()
dac.config_board()
```

```
dac.start_up_sequence()
```

The board is configured from the configuration file and properly initialized in the startup sequence.

```
#Setting the frequency
dac.nco_freq_a(397.76)
dac.nco_freq_b(397.76)

#Setting the phase
dac.nco_phase_a(0)
dac.nco_phase_b(0)

#Setting the amplitude
dac.amplitude_a(1)
dac.amplitude_b(1)
```

Then both channels (A and B) are set to 397.76 MHz with a relative phase of zero and amplitude 1. To set the phase the SYSREF needs to be triggered to synchronize the two NCOs:

```
dac.nco_sync()
```

This code example in full is found in the file 'example.py'.

CHAPTER

TWO

INSTALLATION

Before you can start the installation of the PyDualDDS library you have to install the pyftdi library.

Please note that on Linux you have to set permissions to be able to access the USB interface, see http://eblot.github.io/pyftdi/installation.html.

Once the pyftdi library is installed you can proceed installing the PyDualDDS library. Open a terminal and navigate in to the PyDualDDS folder.

Then install it either by:

```
sudo python setup.py install
```

or by:

```
sudo python3 setup.py install
```

depending on your python installation. Note, PyDualDDS needs Python 3.5 or higher.

You can test if the installation worked by importing the library:

```
import pydualdds
```

If you not getting an error message, everything worked out.

CHAPTER

THREE

SPECTRA

In Figure 3.1 and Figure 3.2, a broadband spectra of a 397.76 MHz sine wave is depicted. The 1st harmonic is > 50 dB suppressed compared to the main frequency peak. The DAC's sampling rate can be seen in the spectrum at 8847.36 MHz. The wavy background stems mainly from the used spectrum analyzer, compare with Figure 3.3 where the spectrum analyzer's input was terminated with 50 Ohm.

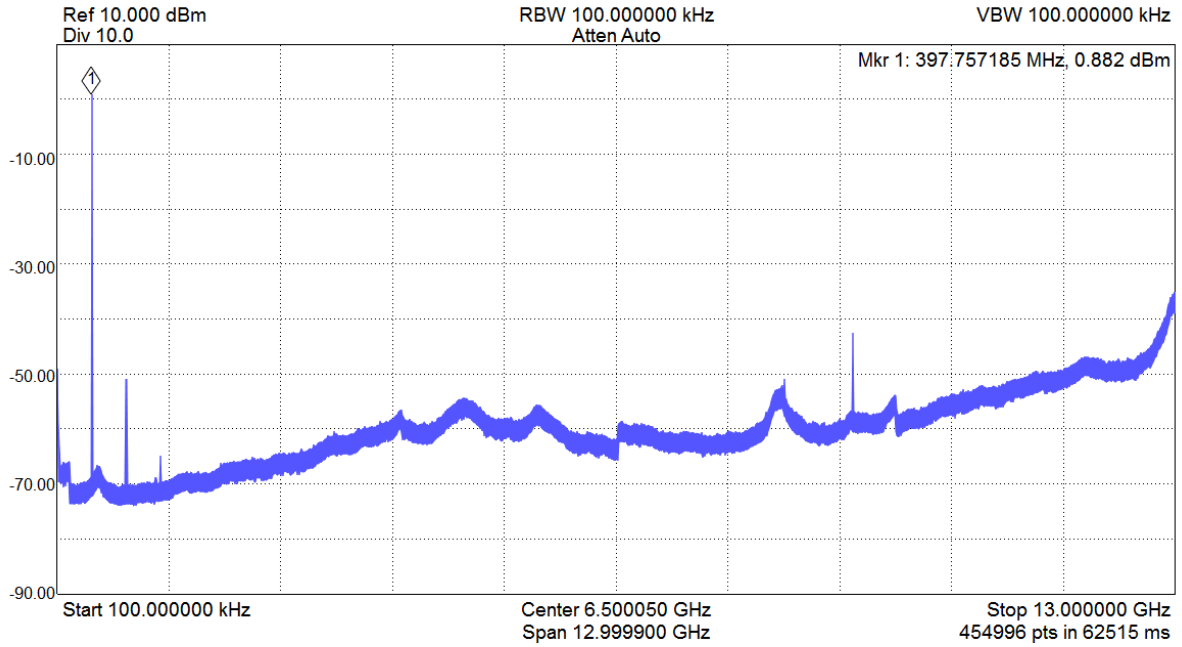

Figure 3.1: Frequency spectrum of channel A set to 397.76 MHz and the amplitude to 1.

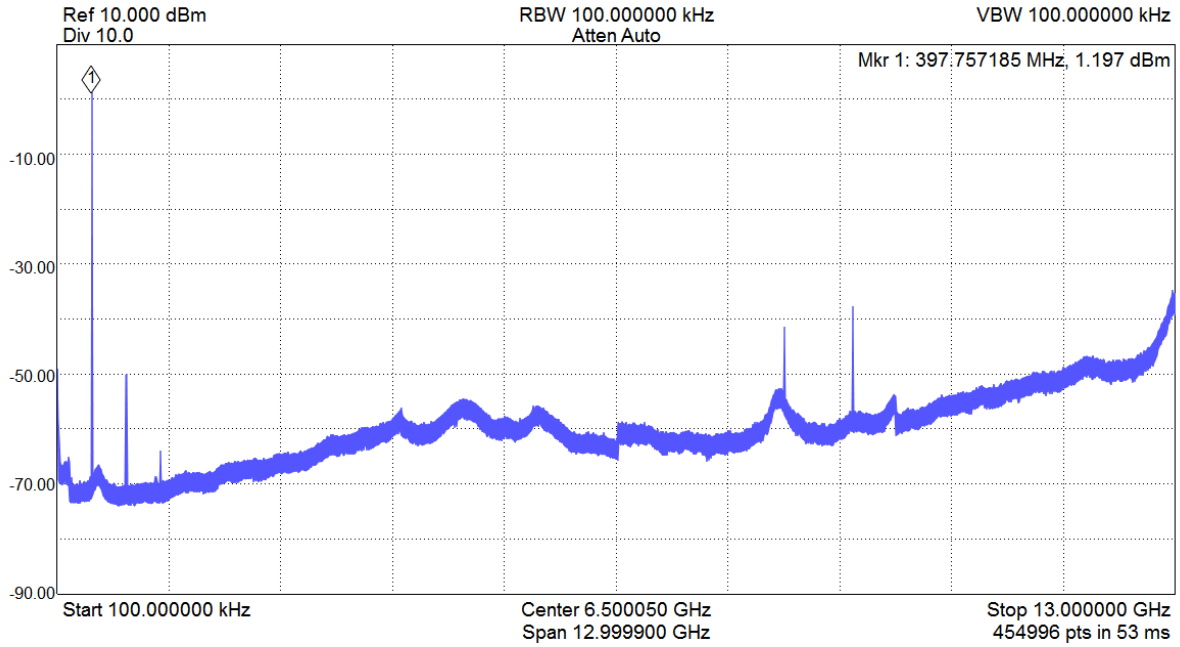

Figure 3.2: Frequency spectrum of channel B set to 397.76 MHz and the amplitude to 1.

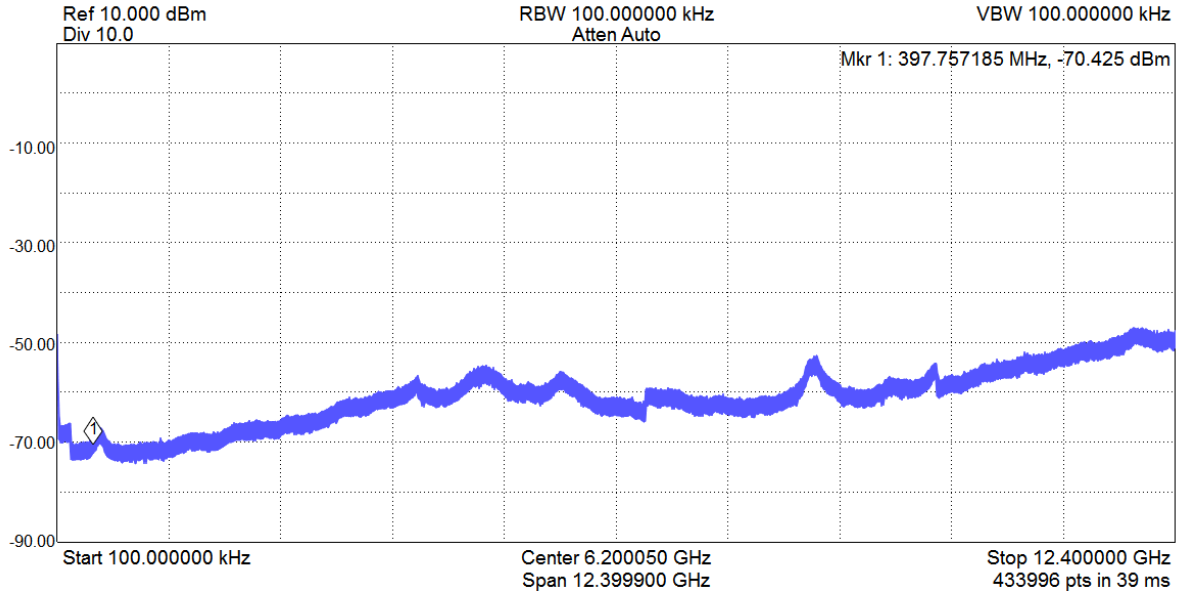

Figure 3.3: Frequency spectrum of the spectrum analyzer with a 50 Ohm terminated input.

CHAPTER

FOUR

MODULES DESCRIPTION

In the following the PyDualDDS module is described. It consists of two classes:

```
* DDS
* DacCom
```

The DDS class is used to control the high level functions of the DAC card such as the frequency, phase, and amplitude. The DacCom class implements the low level communication with the DAC card.

4.1 PyDualDDS module

class pydualdds.**DDS**(*config='./config/PLL_M9N5Ref1228_8MHz_PLLlock.cfg'*)

Bases: object

The DDS library controls the DAC38RF82EVM board and implements a two channel DDS from 0-4 GHz.

The frequency, the phase, and amplitude of each channel can be independently controlled.

The DAC is configured to run at 8.847 Giga samples per second from the on borad's 122.88 MHz clock.

This library allows to use the board in standalone mode without an additional FPGA card.

amplitude_a(*gain*)

Amplitude of channel A.

Parameters **gain** (*float*) – Amplitude setting (0-2)

Above gain 1 distortions may accure. Allows 1024 steps in the gain range from 0-1.

amplitude_b(*gain*)

Amplitude of channel B.

Parameters **gain** (*float*) – Amplitude setting (0-2)

Above gain 1 distortions may accure. Allows 1024 steps in the gain range from 0-1.

config_board()

Configures the DAC and the clock distribution chip on the board

Resets the DAC by its reset pin and loads the configuration file in to the DAC and the clock distribution chip.

nco_freq_a(*freq*)

Set frequency of channel A's NCO.

Parameters **freq** (*float*) – Frequency in MHz

nco_freq_b(*freq*)

Set frequency of channel B's NCO.

Parameters **freq** (*float*) – Frequency in MHz

nco_phase_a(*deg*)
Set phase of channel A's NCO.

Parameters **deg** (*float*) – Phase in degrees

nco_phase_b(*deg*)
Set phase of channel B's NCO.

Parameters **deg** (*float*) – Phase in degrees

nco_sync()
Synchronize NCO's by asserting SYSREF

Call this function after a frequecy change to resynchronize the phase.

start_up_sequence()
Start up sequence

Starts up the PLLs, resets the chips, and synchronizes via SYSREF.

class pydualdds.**DacCom**(*config='./config/PLL_M9N5Ref1228_8MHz_PLLlock.cfg'*)
Bases: object

close()
Close the connection to the DAC card.

dac_change_page(*page*)
Change register mapping page.

The registers are devided in three pages: multi-DUC1 (0b001), multi-DUC2 (0b010), and DIG_MISC (0b100).

Multiple pages can be written at the same time, i.e. 0x011 writes multi-DUC1 and multi-DUC2 at the same time.

Function keeps track of the selected pages in variable self.DAC_PAGE.

Parameters **page** (*int*) – 0b000 - 0b111

dac_configure()
Configure the DAC from the provided configuration file.

Raises **ValueError** – if the read value does not correspond to the set value

dac_read(*address*)
Read a register from the DAC.

Parameters **address** (*int*) – Address of register with page prefix, e.g 0x0128 reads the register 0x28 of page multi-DUC1.

Returns Register value

dac_read_byte(*address*)
Read a register from the DAC without page correction.

Parameters **address** (*int*) – Address of register

Returns Register value

dac_reset()
Reset the DAC

dac_write(*address*, *data*)
Write a DAC register

Parameters

- **address** (*int*) – Address of register to write in
- **data** (*int*) – data to write to register

dac_write_byte(*address*, *data*)
Write a byte to the DAC.

Parameters

- **address** (*int*) – Address of register with page prefix, e.g 0x0328 writes the register 0x28 of page multi-DUC1 and multi=DUC2 at the same time.
- **data** (*int*) – data to write to register (0x00-0xFF)

lmk_configure()
Configure the LMK04828 clock cleaner from the provided configuration file

Raises **ValueError** – if the read value does not correspond to the set value

lmk_read(*address*)
Read data from an address of the LMK04828

Parameters **address** (*int*) – Address of register

Returns Data at address

lmk_write(*address*, *data*)
Write data to a address in the LMK04828.

Parameters

- **address** (*int*) – Address of register
- **data** (*int*) – data

port_flush()
Flush data to port.

read_dac_config(*file_name*)
Read the configuration file and extract the DAC data from it.

Parameters **file_name** (*str*) – Path to the configuration file

Returns DAC configuration dictionary

read_lmk_config(*file_name*)
Read the configuration file and extract the LMK04828 data from it.

Parameters **file_name** (*str*) – Path to the configuration file

Returns LMK04828 configuration dictionary

set_bit(*bits*, *position*, *value*)
Set bit in bit field bits at a given position to a specified value.

Helper function.

Parameters

- **bits** (*int*) – bit field
- **position** (*int*) – bit field e.g. 0b11110000
- **value** (*int*) – number e.g. 5

Returns Changed bit field

set_bit_on_port(*bit*, *value*)
Set a bit of the 8 bit output port

Parameters

- **bit** (*int*) – bit position 0-7
- **value** (*bool*) – True or False

CHAPTER

FIVE

LICENSES

5.1 Source Code

The source code of the PyDualDDS library is licensed under GPLv3:

Copyright (C) 2017 Andreas Fognini

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

5.2 Documentation

This documentation is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

INDEX