PER VICES CORPORATION

NOCTAR USER MANUAL

Contents

Change Log 5

Preface 7

Obligatory Warnings 9

Specifications 13

Installation 17

Command Line Utilities 21

Graphical Utilities 23

Usage Notes 25

System Overview 31

Noctar *GPIO* 33

On Latency and Performance 37

FPGA Implementation 39

FPGA Simulation and Test bench 43

FPGA Programming 47

Driver Implementation 49

Appendix I: Mechanical Drawings 55

Appendix I: Pin Descriptions 57

Appendix II: GPIO Address Offsets 61

Epilogue 65

Change Log

2012-12-12: Rev A: Initial Release

2013-01-04: Rev B: Cleaning up some sections, fixing typos, improving readability, note on input GPIO.

2013-01-23: Rev C: Adding section on testing ADC outputs. Adding IQ matching section. More cleanup and readability.

2013-02-03: Rev D: Adding additional notes on how decimation and interpolation works.

2013-03-06: Rev E: Updating troubleshooting notes, including driver permissions and creation.

Preface

Noctar

Noctar is a wide band, high gain, direct conversion quadrature software defined radio transceiver and signal processing platform. Using analog and digital conversion, it can capture up to 200MHz of bandwidth across up to 4GHz, and rapidly communicate that information over an 8Gbps high speed PCIe bus.

Noctar represents our first step towards building a low cost, distributed, wireless communications infrastructure - something to enable application developers to implement any wireless device in software.

Congratulations!

Congratulations on your purchase of the Per Vices Noctar Transceiver! This manual is intended to provide you with useful information regarding the safe operation and use of your new Transceiver. It may be updated from time to time - you'll always be able to find the latest version on the Per Vices website[1].

[1] http://www.pervices.com

We hope you enjoy using Noctar. In building Noctar, we aimed to provide a highly capable device at the lowest possible price. Our belief is that there are significant resource barriers impeding the widespread adoption of Software Defined Radio (SDR) technology. These barriers come in the form of the high capital cost necessary to purchase specialized equipment, and the extensive experience necessary to successfully operate such devices.

This product is our second step towards reducing the cost barrier. We've tweaked the RF chain, improved shielding, added IO, and improved performance - in short, we have tried to pack in as much functionality as possible at a comparatively low price.

Faced with a number of conflicting requirements, we have endevoured to satisfy the broadest use cases possible, while trying to reduce the complexity necessary to operate Noctar. Our goal is to help you best utilize the flexibility afforded by SDR technology.

Our hope is that you will find NOCTAR to be a useful and dependable companion in your engineering, development, and research efforts.

We welcome your feedback; please feel free to contact us at:
solutions@pervices.com

A note on LANGFORD

When reading this manual, or using the code examples, drivers, or utilities, you may come across references to LANGFORD. You may find yourself wondering who, or what, LANGFORD refers to. The quick answer is NOCTAR is part of the SDR series formally known as φ; for all intents and purposes you can safely assume that LANGFORD refers to NOCTAR.

We originally intended to call our product φ, but we ended up changing our name to avoid stepping on toes... However, to ensure compatibility with our earlier products, and to simplify development efforts, we continue to use 'Langford' within our drivers and utilities.

Obligatory Warnings

The following section contains important safety and regulatory information. Please pay attention to the following disclaimers, warnings, and cautions.

This device is intended for engineering, research, or science laboratory use only - it is not for open office or residential use!

This device has not been tested or approved by any agency or approvals body for Electrical Safety, Electromagnetic Compatibility, or Telecommunications at the time of distribution! You use this device at your own risk.

Disclaimer

This product is provided «As Is». Per Vices is under no obligation to provide updates, upgrades, support, or maintenance of any kind. Per Vices specifically disclaims any and all warranties and guarantees, express, implied or otherwise, arising with respect to the use of this product including, but not limited, to the warranty of merchantability, the warranty of fitness for a particular purpose, and any warranty of non-infringement of the intellectual property rights of any third party. Per Vices neither assumes or authorizes any person to assume for it any other liability.

Your use of this device is at your own risk. Per Vices shall not be liable for you for any damages, direct or indirect, incurred or arising from the use of this product. In no event will Per Vices be liable for loss of profits, loss of use, loss of data, business interruption, nor for punitive, incidental, consequential, or special damages of any kind, however caused, and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise), arising in any way out of the use of this product, even if advised of the possibility of such damages.

Product Functionality

Every effort has been made to ensure that the device you receive is fully functional - each device is fully tested prior to shipping. However, risk of damage or loss is transferred immediately upon delivery to you - we do not generally accept returns or refunds on successfully delivered packages.

If you have any problems, please contact: SOLUTIONS@PERVICES.COM

Specifications

Every effort has been made to test and measure the validity of this equipment. However, we cannot guarantee the accuracy of specifications, and they may change at any time.

Warnings

WARNING
RISK OF ELECTRIC SHOCK

Do not attempt to modify or touch this device while powered. Ensure host computer is properly grounded during operation. Disconnect AC power during installing or removal.

WARNING
HOT SURFACE

This circuit board may become very hot during operation. Contact should be avoided.

WARNING
LABORATORY USE ONLY

This device has not been approved by any agency or approvals body for Electrical Safety, Electromagnetic Compatibility, or Telecommunications at the time of distribution. Research use only!

ATTENTION
OBSERVE ESD PRECAUTIONS

This device contains electrostatically sensitive components: it may be damaged by static discharges. Observe ESD precautions & proper grounding when handling, installing, or removing device.

ATTENTION
RF TRANSMITTER

This device is capable of RF transmission on bands or frequencies subject to regulatory oversight. Operators are responsible to ensure use of this device meets local regulatory and legal standards, as they may apply to you and the band of interest.
This device is intended for test and measurement use only.

Specifications

NOCTAR is a wide band, high gain, direct conversion quadrature transceiver and signal processing platform. Using analogue and digital conversion, it is capable of processing up to signal bandwidths up to 200MHz from 100kHz to 4GHz. NOCTAR is compatible with GnuRadio, and includes open source drivers.

As NOCTAR is capable of Digital Down/Up Conversion, superhet architectures can be implemented using Digital Down/Up Conversion on the FPGA.

Absolute Maximum Ratings

Stresses beyond those listed in table 1, Absolute Ratings, may cause permanent damage to the device. These ratings are stress specifications only; functional operation of the product at these conditions is not implied - exposure to absolute maximum rating conditions for extended periods of time may affect reliability and is not recommended.

Specification	min	max	units
Operating Temperature	5	85	C
Storage Temperature	0	70	C
Input RF Power		15	dBm

Table 1: Absolute Ratings: Exposure or sustained operation at absolute ratings may permanently damage NOCTAR. This device is passively cooled - ensure adequate airflow or cooling during extended operation.

Observed Performance

The specifications listed in table 2 on page 15 detail observed performance under typical conditions. They are intended as a loose guide to what we have observed during internal testing; please contact us if you require specific specifications.

RF Chain

Simulated RF chain performance, based on component specifications, yield the following simulated performance. As both the receive and transmission RF chains have variable stages, table 3 on page 16 uses midpoint references for attenuation and gain stages.

Operating System

We test all devices on a 64 bit Arch Linux system running GnuRadio and the 3.4 kernel. We do not support the Linux 2.6 kernel. We have also successfully used NOCTAR on 64 bit Ubuntu systems.

Some users have reported problems compiling our drivers on VM's or 32 bit machines, especially those using Ubuntu, so we recommend avoiding 32 bit systems or VMs when using our product.

During early testing, we successfully installed our drivers on a 32 bit Arch Linux machine, so it should be possible, if you're willing to get your hands dirty.

Mechanical

A mechanical drawing in included in the Appendix.

Specification	min	nom	max	units
Temperature				
Card Operating Temperature		60		C
Analogue				
RF Tuning (ADF4351)	35		4400	MHz
Dyn. Range (RX,TX)	10		50	dB
Power Gain (RX, High Stage) @2GHz	-23		43	dB
Power Gain (RX, Low Stage) @ 125MHz	-4.5		+55.5	dB
Nominal RF Input Power		-15		dBm
TX Power (Low) @ 10MHz		+15		dBm
Digital				
PCIe x4, v1.1 Data Rate (full duplex)		8		Gbps
ADC resolution		12		bits
ADC Sample Rate (per IQ Channel)		125		MSPS
DAC resolution		16		bits
DAC Sample Rate (per IQ Channel)		250		MSPS
Decimation (2^n), for values of n=[0,5]	0		32	-
Interpolation (2^n), for values of n=[0,5]	0		32	-
Internal Reference (20 MHz)				
Frequency Calibration (20^0C)	-1.0		1.0	ppm
Internal Frequency Stability	-0.28		+0.28	ppm

Table 2: Observed Performance. These specifications reference observations taken during internal use and development. Please note the Dynamic Range section for more information.

2)
We use a 5 bit word to set or clear interpolation bits. Decimation or Interpolation is implemented as a cascade; each stage checks the state of one of the five bits in the decimation word.
If the bit is set, the stage takes the data and carries out one decimation (or interpolation) by two. If the bit is clear, than the stage simply passes the information through.
For example, if the interpolation word is set to (TxIntEn = 5'b01011 = 0xb), then three interpolation stages are set (0, 1, and 3), resulting in a total interpolation of 8 (2^3).
3) I suspect that you're setting the interpolation stages to 4. This corresponds to TxIntEn=5'b00100, which only enables the 2nd stage, and corresponds to an interpolation of 2.
To help figure this out, you can work backwards, using dd, to help set the stages. With 0 interpolation, your maximum bandwidth is limited by the bus (~700MBps), so you're going to lose packets. However, with an interpolation of 0xf, you should be able to view the correct result;
langford_util /dev/langford TxIntEn 0xf
I hope this helps - if I am wrong, and this does not help fix the problem, please let me know! I'm really eager to help work with you to fix this.

Specification	DC	2GHz	units
Input			
RF Input Power		-12	dBm
Analysis Bandwidth		100	MHz
Rx Chain			
SFDR		63.5	dB
SNR		71.6	dB
Input Rx Sensitivity		-73.6	dBm
Input P1dB	1	-7	dBm
Tx Chain			
Power Gain	25	14	dB
SFDR	67	71	dB

Table 3: These specifications are intended to serve as a very general guide, with variable gain and attenuation stages set at midpoints. As variable stages are adjusted, performance may vary considerably.

Installation

Installation comprises four steps;

1. Physically disconnecting mains power to your computer and inserting the card into a PCIe slot.
2. Installing the software drivers and utilities.
3. (Optionally) Copying the initialization routines to a location where it will automatically load with your computer.
4. (Optionally) Installing GnuRadio[2]. This allows you to immediately start playing with NOCTAR.

[2] If you are using ArchLinux, you'll find a build script for GnuRadio within the folder called 'arch'. Use it by running:
makepkg -p gnuradio.PKGBUILD

Physical Installation

NOCTAR comes in a PCIe form factor, and comes programmed.Turn off your computer and physically disconnect the AC power cable. Open up your computer, and install the card in a free x4 (or greater) PCIe slot. See your computer manufacturer for specific instructions. Reverse these steps for removal.

If you're wondering why we're emphasizing the disconnection of mains power, it's because we speak from experience. Learn from our mistakes!

Driver and Utility Installation

The installation procedure is relatively straightforward;

1. Obtain a copy of the Noctar drivers. You can download them from the Per Vices site, or use the ones that may have been shipped with your device.[3]
2. Run make to build the drivers, then switch to a root user (or sudo) and run make install.

   This will install the kernel module, and creates the necessary character device (/dev/langford) which shall be used for RX/TX.

Rarely, make install may fail due to a known race condition (mknod running before insmod is complete). If this is the case, try running make install again.

[3] We recommend downloading the latest drivers from:
http://www.pervices.com.

```
> cd /langford_driver
> make
> su -l
Password:
# make install
```

Figure 1: Sample installation routine; make the drivers, then install them as root. On Ubuntu systems, we've found compilation sometimes requires root priviledges (otherwise you get an error). To get around this, log on as root prior to running make, and make install.

Auto loading Initialization Routines

You will have to initialize the device (by running langford_init) every time you restart your computer! You can make this easier by copying this file to your init scripts.

GnuRadio

The easiest way to start using NOCTAR requires you to have a complete GnuRadio installation. GnuRadio may be including in your distributions package manager. If you want to use the utilities, ensure that you pay special attention to satisfying all necessary (and optional) dependencies.[4]

[4] Some good initial tests of whether things are working properly can be found using the scripts in the tests/ directory.

Troubleshooting

First, ensure that you've initialized the driver;

```
# langford_init
```

It may be necessary to rebuild the langford drivers after updating or upgrading your kernel. If you encounter runtime errors indicating the file doesn't exist, ie;

```
failed to open '/dev/langford': No such file or directory
    Runtime Error: can't open file
```

Then you may have to initialize the character device, using the langford_init script;

```
#langford_init
```

If something's not working, confirm that the computer detects the card and has loaded the correct module;

```
$ lspci |grep Altera -A9
    XX:YY.0 Unassigned class [ff00]: Altera Corporation Device 0004
(rev 01)
    Subsystem: Altera Corporation Device 0004
    Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop-
ParErr+ Stepping- SERR+ FastB2B- DisINTx-
    Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort-
<TAbort- <MAbort- >SERR- <PERR- INTx-
    Latency: 0, Cache Line Size: 256 bytes
    Interrupt: pin A routed to IRQ 16
    Region 0: Memory at fbcf0000 (32-bit, non-prefetchable) [size=64K]
    Region 1: Memory at fbcec000 (32-bit, non-prefetchable) [size=16K]
    Capabilities: <access denied>
    Kernel driver in use: langford
```

The important points are that the card exists and your detects it (evidenced by the fact you get this output), and that it has loaded the correct driver (evidenced by the last line, «Kernel driver in use: langford».

If you don't get this output, confirm the card is properly inserted into the computer.

If you don't see the langford kernel driver, confirm that you've installed the driver properly.

Remember; you need to load the driver using langford_init prior to using the card!

Command Line Utilities

This section discusses the utilities available to read and write data from Noctar, how to use the control utility, and an illustration of the python IO application.

Command Line Utilities

Multiple command line utilities are used to control the functionality of the Noctar device. We describe these utilities here.

Location of Utilities

These utilities exist in two locations:

1. langford_driver
2. langford_driver/libs

Building the Utilities

For both locations (langford_driver, libs), a straightforward make, and make install should work.

Types of Utilities

In general, all utilities can be sorted into 3 categories by level of abstraction:

1. Raw register access – langford_util

   This is the lowest level of abstraction. This utility allows you to manipulate the hardware registers on the PCIe device directly. This utility only accepts raw values (e.g. on/off) and does not perform any computation. This utility is useful for debugging where the finest control is required[5].

2. Serial protocol manipulation – langford_driver/test/langford_*_util

   This is the second lowest level of abstraction. These utilities allow you to write values to serial interfaces (e.g. to the frequency

[5] A detailed list of the parameters, including net names, allowed values, and descriptions, can be found in the Driver Implementation section of this guide.

synthesizer), abstract the bit banging, and automatically configure on board chips. These utilities are also useful for debugging the calculation of values for the serial interface peripherals. For example, to debug the frequency synthesizer, you can use langford_adf4350_util to write the register values in 32 bit hex values directly as a program argument. The utility takes care of toggling all the appropriate bins necessary to perform the serial data transfer. Remember that hex values start with 0x.

3. Task oriented – libs/langford_rx_rf_bb_vga, libs/langford_RX_bb_vga, langford_adc_util

   These are the next level up in abstraction. These utilities calculate the raw values based on target parameters and program the appropriate peripherals. For example, you can use fsynth to set the frequency synthesizer to a certain frequency. fsynth calculates the register values for the peripheral and toggles all the pins necessary for serial data transfer.

4. GUI wrappers - noctar_io/pv_noctar_io.py

   These utilities provide a GUI wrapper to some of the lower level utilities.

Using Utilities

If you run any of these utilities without any arguments, a help message will be displayed. Typically, these utilities will require the name of the character device (/dev/langford by default). For example, to tune the RX RF stage to 500MHz, you can run:

./langford_rf_fsynth /dev/langford 0 500e6

A complete listing of parameter options and legal values is included in the Driver Implementation section.

Graphical Utilities

There are two graphical utilities; a configuration GUI, and some gnuradio companion scripts to get you started.

Noctar IO GUI

To illustrate how everything works, we've developed a python GUI that allows you to easily adjust the options available to you. Figure 2 on the next page shows a screen shot of the utility program. The source code is available for reference.

GnuRadio Integration

If everything has gone well, you should be able to run scripts in the test folder. These scripts automatically tune the front end, and launch an example gnuradio script. This allows you to immediately start playing with the device, as shown in Figure 2 on the following page.

We need the following things to fall into place; physical installation, software driver installation, driver initialization (run langford_init to create the /dev/langford device), and a working gnuradio installation.

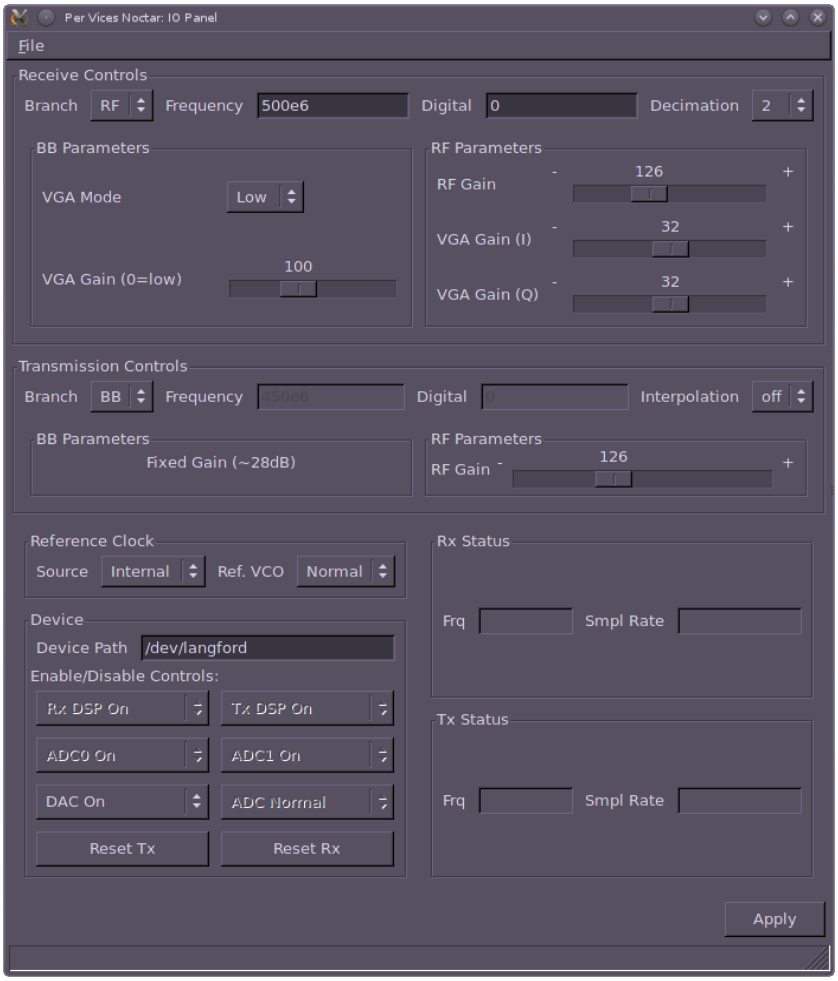


Figure 2: Screen capture of Noctar GUI controlling the reception of a simple GNU Radio Companion flow chart.

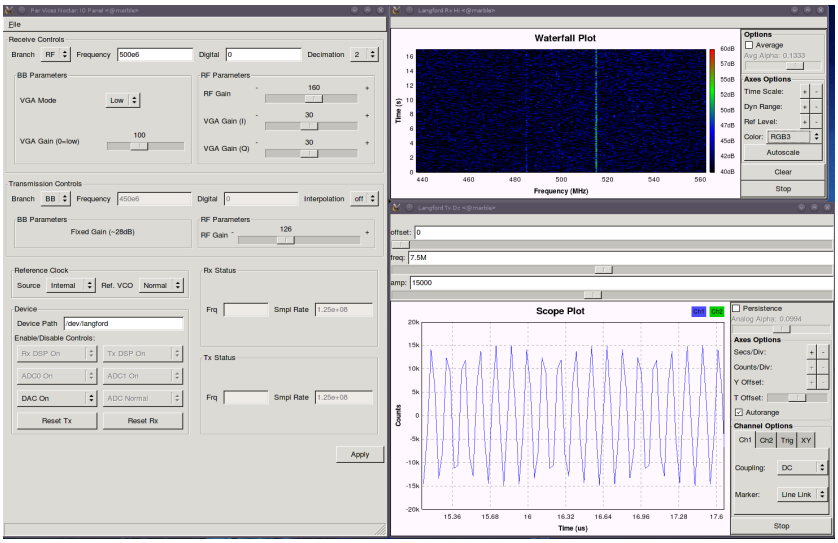


Figure 3: Screen capture using GnuRadio Companion, and Noctar IO panel to view waterfall plot.

Usage Notes

Receive Chain

- This device is capable of very high gain - you might find yourself saturating or overload the receiver. This can be easily solved by turning down the gain.

  The RF Front End has about 40dB dynamic range at the RF Attenuators.

- If you are working with the High branch, look at whether the IQ channels are balanced using a scope view (our test directory contains this). If not, you can adjust each RX VGA channel independently.
- You can implement very reasonable Automatic Gain Controls using the RX Base band VGA controls.

  One day, we hope to push automatic gain controls to the driver, along with IQ balancing.

- Later revisions of the device interleave the ADC samples to provide a real valued, base band signal, at 250MSPS with anti aliasing filters with a 110MHz cut off. This effectively provides base band scope functionality, at the expense of giving up DSP (so no FPGA down or up conversion). This usually works out well, as you can implement batch applications to collect and filter data (including down conversion and decimation) in software, over the PCIe bus, without necessarily compromising anything. If using the base band stage, pay attention to the Receive Data (Base band Branch) section on on page 53.
- You can adjust the ADC gain. By default (power on), both ADCs are automatically configured to digitize a 2Vp-p signal. However, with the introduction of this new feature, you can adjust the full swing ADC input from 1-2Vpp. This may be configured in 1dB steps, from 0 to 6dB. This may be useful in achieving better IQ balance.
  - For example, to adjust ADC A, use the Langford ADC utility, and run the following command;

    ```
    langford_adc_util /dev/langford A IntRef 2
    ```

    The A refers to ADC A which digitizes the Q signal component (choose ADC B for the I signal). The 2 specifies 2dB of gain on the apparent signal swing.

  - When you want to turn off this feature to the default 2Vp-p, pass in -1 as the last parameter;

    ```
    langford_adc_util /dev/langford A IntRef -1
    ```

  - There is a catch; you *may* have to remove RC10. It was originally placed in to act as a short which improves matching and balance between I and Q.

    If you suspect this is a problem, contact us and we'll guide you through it.

- At full RX bandwidth (no decimation), you may sometime lose bytes under GNU Radio, especially with waterfall plots. This is a large problem for RX since integer sizes are 16 bits. A loss of 8 bits will cause incomprehensible data corruption (because you'll only have the latter half of a sample). Increasing decimation generally avoids the problem on lower end hardware.

Constraining ADC Output

It may sometimes be useful for you to constain the ADC output - at the ADC itself. For example, you've developed some neat FPGA code, and would like to confirm that your FPGA logic is correctly handling inputs. Perhaps you don't have an signal generator, and would like to have a known input at the ADCs.

We can use the langford_adc_utility to program the ADC's to output test pattern outputs. To do this, we need to write to address 0x0014h of the appropriate ADC with a single additional byte. The value of this byte, along with some other neat tricks, may be found in Table 26 of the ADC1210S series 12-bit ADC data sheet. For convenience, we've reproduce the relevant table in Table 4 on the facing page.

What this essentially means is that to set the output of ADC B to its midpoint, we type the following;

```
langford_adc_util /dev/langford B Raw 0x1401
```

Alternatively, to set the output of ADC A to its maximum value, we type;

```
langford_adc_util /dev/langford A Raw 0x1403
```

After you've finished playing with ADC A, you can set it back to normal by typing;

```
langford_adc_util /dev/langford A Raw 0x1400
```

You may find more commands and ADC fine tuning in the user manual - including how to use these codes for digital offsets, or timing correction (adjusting for IQ mismatch), in the datasheet.

Bit	Symbol	Access	Value	Description
7 to 3	-		00000	not used
2 to 1	TESTPAT_SEL[2:0]	R/W		digital test pattern select
			000	**off**
			001	midscale
			010	-FS (Full Swing) (MIN)
			011	+FS (Full Swing) (MAX)
			100	toggle '11...11'/'00..00'
			101	custom test pattern
			110	'1010..1010'
			111	'010..1010'

Table 4: Test pattern register 1 (address 0x0014) bit description. From the ADC1210S series datasheet (NXP Semiconductors / TI).

Transmission

- There is a race condition that sometimes manifests itself when writing to the driver. We're not yet sure what, exactly, is happening, but sometimes when you start writing to the device, you get garbage output. This only seems to manifest after the TX branch has been used, and we're still trying to isolate the problem. In the meantime, to fix the problem, type the following:

  $ langford_util /dev/langford TxDspEn 0
  Places the Tx DSP chain in reset (output will stop).
  $ langford_util /dev/langford TxDspEn 1
  This takes the Tx DSP out of reset (output will resume).

We think the problem may relate to how the DMA transfer is completed - a byte may be getting dropped somewhere.

- There is no VGA on the DAC outputs; If you want to balance IQ output, do so (in software) by adjusting the phase and amplitude of the I/Q signal going into the DAC.
- The low frequency stage is attached to a single DAC output (you can't interleave DAC outputs). The other channel is dangling, but you may access it by directly probing the two exposed test ports, shown in 4 on the next page.

Driver Notes

- If you are running into a permission denied problem when accessing the character device, use chmod or chown to set the permissions appropriately.

Full Rate Transception

- At full bandwidth (no decimation), some computers have trouble running GNU Radio without dropping bytes, especially with waterfall plots. This is a problem as we use 16 bit integer sizes

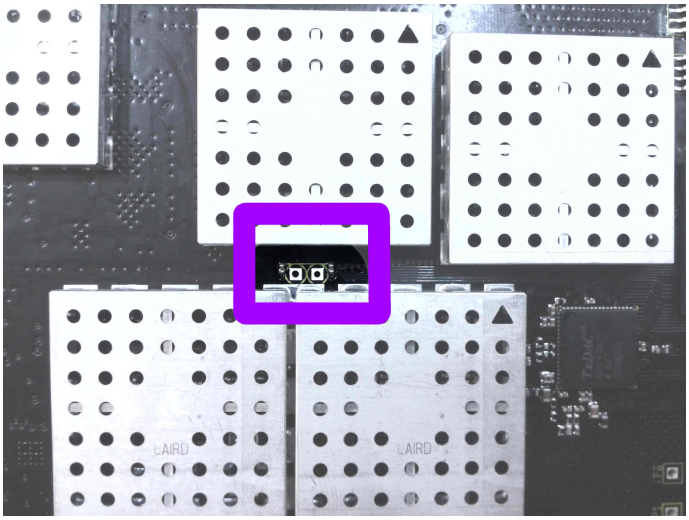


Figure 4: These test points (found about the top left corner of the shield that is immediately to the left of the DAC) directly access the second DAC channel while in baseband mode.

- dropping a byte (8 bits) irrevocably corrupts data. If you find yourself encountering this problem, increase the decimation to 2 or higher will avoid this problem.

Decimation and Interpolation

We use a 5 bit word to set or clear decimation and interpolation bits. The actual decimation or interpolation is implemented as a cascade; each stage checks the state of one of the five bits in the decimation word. If the bit is set, the stage takes the data and carries out one decimation (or interpolation) by a power of two. If the bit is clear, than the stage simply passes the information through to the next chain.

For example, if the interpolation word is set to (TxIntEn = 5'b01011 = 0xb), then three interpolation stages are set (0, 1, and 3), resulting in a total interpolation of 8 (2^3).

- Make sure that you use hex when specifying the number of decimation or interpolation bits you want to set. To set a decimation of 8, you need to set three bits;

  langford_util /dev/langford TxIntEn 0xb

- To conserve FPGA resources, decimation and interpolation are currently implemented for powers of two from zero to five (2^n, where $n \in \mathbb{N}^0, 0 \leq n \leq 5$) , where n corresponds to the number of bits set in the decimation or interpolation register[6]. The decimation values are implemented as a five bit block; for every bit that is enabled, another factor-of-two decimation stage is introduced to the signal chain.

[6] Referencing the langford_util io program, the decimation register is implemented as RxDecEn register, and interpolation as TxIntEn.

- To calculate Receive sample rate, after decimation, use the following formula; $f_{rx,SR} = \frac{F_{sample,adc}}{2^n}$, where $F_{ADC,SR} = 125MSPS$. To illustrate, if you seek to reduce the sample rate by a factor of 8, three bits of decimation would be set ($n = 3$), such that the ultimate sample rate is, $F_{sample,adc} = \frac{125MSPS}{2^3} = 15.625MSPS$.

Three bits of decimation correspond to setting RxDecEn = 0x7.

- To calculate transmission sample rate, prior to interpolation, use the following formula; $f_{tx,SR} = \frac{F_{DAC,SR}}{2^n}$, where $F_{DAC,SR} = 250MSPS$. As above, if you want to send data at 62.5MSPS, you would set 2 bits of interpolation ($n = 2$), the sample rate works out to be; $F_{sample,adc} = \frac{250MSPS}{2^2} = 62.5MSPS$.

Two bits of interpolation require setting TxIntEn to 0x3.

Benchmarking Throughput

- You can figure out the maximum sustainable transfer rates by using dd;

```
$ dd if=/dev/langford of=/dev/null bs=4092
  $ dd if=/dev/zero of=/dev/langford bs=4092
```

- Due to the character device buffer, transfer rates reported by dd may not be perfectly accurate due to the overhead in flushing/filling the buffer and waiting for remaining DMA transfers to complete. For RX, the reported transfer rate will be less than the actual transfer rate. For TX, the reported transfer rate will be more than the actual transfer rate. Running dd for a longer period of time will yield on more accurate results, ultimately, converging on the correct answer.

DMA Transfers

- DMA read and write bandwidth over the PCIe bus is not symmetric. Write bandwidth is larger since writing to system memory is a "fire and forget" event. When data is sent over the PCIe bus, it is assumed to have completed successfully. Reading from system memory requires the writing of a read command and waiting for the data to come back, leading to lower bandwidths.
- Are the input I and Q appearing incorrectly in user mode applications? Do you get base band signals that are exactly the negative of the frequency of what you expect? This is easy to fix, find the following line in langford_system.v: .iAdcDI(iAdcAD), .iAdcDQ(iAdcBD), And change them to: .iAdcDI(iAdcBD), .iAdcDQ(iAdcAD), The same logic applies to TX and the DACs.
- If you decide to add a SignalTap instance to the top level design, you might have to disable either the RX or TX DSP chain to free

up some resources. This is easy to do while maintaining system (software, driver & RTL) integrity. Look for the null sink or the null source commented out near the FIFO and DSP chains. Simply replace the FIFO and DSP chain with the null source/sink.

FPGA Code

- The top level Qsys file needs to be modified slightly after its automatically generated. There is a known issue with Qsys where signals cannot be used both inside the Qsys design and exported form it simultaneously. There is nothing conceptually or practically wrong with doing so, but it is not supported by Qsys, hence this must be done manually.
  - Open the langford_qsys/synthesis/langford_qsys.v file.
  - Remove the line:
    - wire pcie_hard_ip_0_pcie_core_clk_clk;
  - Find the line:
    - module langford_qsys (
  - Add the following line immediately after the line that you found above:
    - output wire pcie_hard_ip_0_pcie_core_clk_clk,
- When building the FPGA image, you may encounter timing errors. There are multiple false paths between the GPIO pins (PCIe clock domain) and ADC/DAC pins which are on different clock domains. These are false positives and can be safely ignored.

System Overview

Overview

NOCTAR has independent RX and TX chains. Within each chain, there are two branches, a High branch, supporting RF frequencies (generally about 150-4000MHz), and a Low branch, supporting base band applications (usually from several hundred kHz through to 110MHz). Figure 5 broadly describes the RF chain (along with relevant device information) for the RF System.

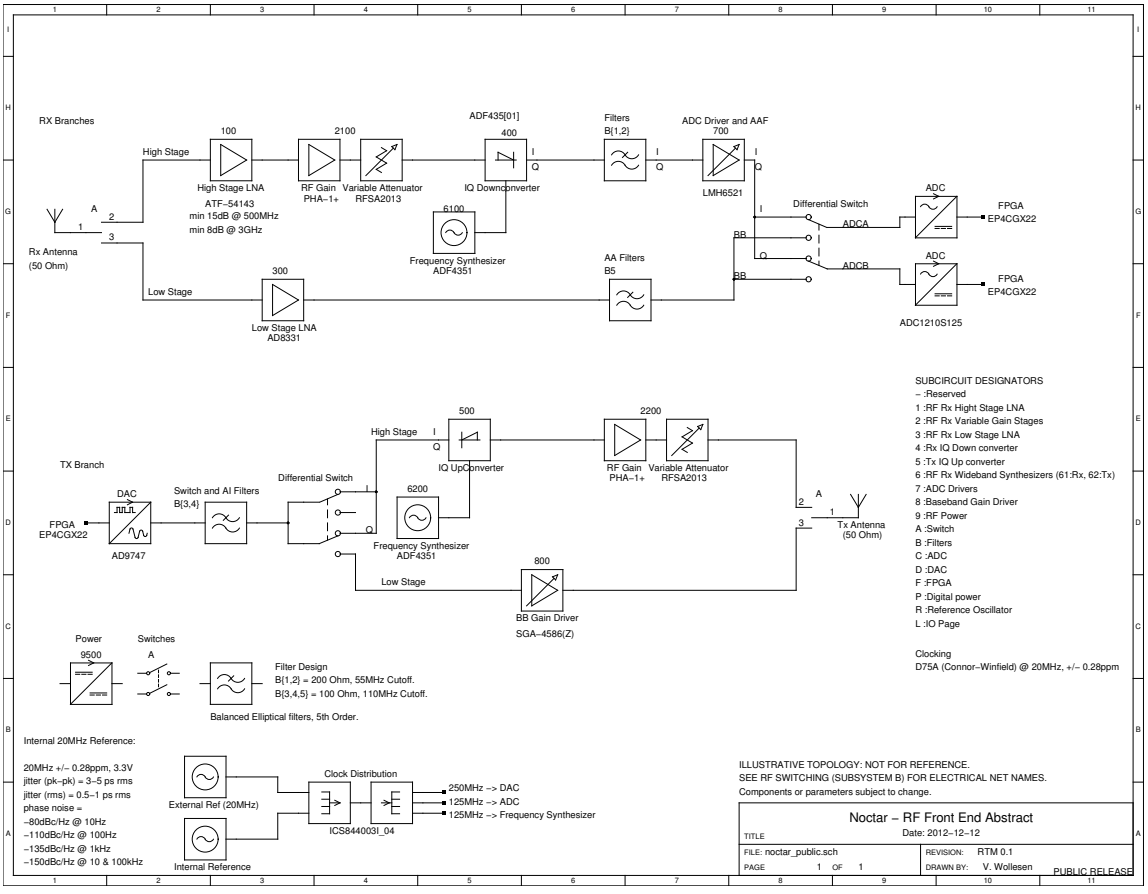


Figure 5: System Diagram

Noctar *GPIO*

Noctar has ten user assignable IO pins. These pins, accessible through an IO header protruding from the bracket, allows you to build programs capable of communicating with outside equipment. By default, we assign pins 1-5 as inputs, and pins 6-10 as outputs. The pin numbering system is shown in Figure 6.

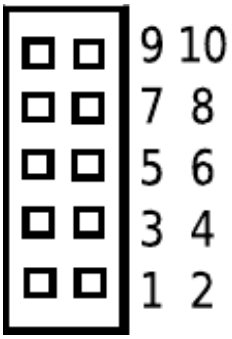


Figure 6: Noctar IO header pin out.

Warning

These pins connect directly to the FPGA - be really, really, careful about what you plug into the ports, and ensure it meets with the IO standards you've specified.

The inputs are directly coupled to the FPGA - if you're not careful about what you attach, it may very well be the last thing you attach.

Default IO Standard

By default, all user access able GPIO pins on the header use a 2.5V IO standard.

Reading and Writing to IO

You can read, and set, the state of the GPIO pins using the langford utility.

```
$> langford_util /dev/langford GPIOin
```

The inputs (pins 6-10) are weakly pull-up, so without anything to pull them down, the default state should be high.

Modifying the IO Pin Configuration

So you're all set with the next big thing... But you just need a couple more pins - maybe you aren't using the outputs, and really want to convert some of the IO inputs into outputs.

With Noctar, this is easy! There are two parts to this - adding the pins to the FPGA, and including them in the driver.

FPGA Modification

Follow these instructions:

More information on the FPGA configuration may be found in the FPGA implementation section.

[7] You can download Quartus II gratis from the Altera website - https://www.Altera.com

1. Download and install Quartus II 12.0 SP2[7]
2. In Quartus, click File > Open Project.
3. Select and open the langford_system.qpf project file.
4. Choose any bit(s) that is available on any PIO register. Reference the FPGA implementation section for assigned PIO pin outs.
   (a) Pretend you choose Lets say you choose bit 18 on PIO0 (output only), with the name oTestPin.
5. Choose an appropriate register based on the direction of the data, PIO2 can be used for input only, all other registers can be used for output only.
6. Open the langford_system.v file.
7. Find the line:

   ```
   module langford_system(
   ```

8. Add the following line immediately after it (change output to input if necessary):
9. output oTestPin,
10. Find the line:

    ```
    endmodule
    ```

11. Add the following line immediately before it you are adding an output pin:

    ```
    assign oTestPin = wPIO5[22];
    ```

12. Alternatively, if this was an input pin, this would work:

    ```
    assign wPIO2[5] = oTestPin;
    ```

13. Click File > Save.

14. If this is a pin internal to the FPGA, then continue and make any other changes to the FPGA design as necessary. The remaining steps except for the last are not necessary. If this pin is to be accessed outside the FPGA, follow the remaining steps.
15. Click Processing > Start > Start Analysis and Synthesis.
16. When the process has complete, click Assignments > Pin Planner.
17. Find the pin that you just added in the list. For the Location column, enter the pin number that you want the signal to appear at. Modify the I/O Standard as necessary.
18. Add the pins to the FPGA design.
19. Follow the driver instructions in the Driver Implementation document to add driver support to these pins.

Driver Modification

1. Add a new IOCTL_GET and/or IOCTL_SET definition to langford_ioctl.h. Ensure that the numerical value hasn't been already been assigned. IOCTL_GET commands are for reads, IOCTL_SET commands are for writes. If you have read only or write only pins, you only have to define one and not the other.
2. In langford.c, there are two switch blocks. The first switch block defines the offsets for the pin. The second switch block performs the actual IO. Modify the 2 switch blocks in the following fashion:
   (a) Add the IOCTL commands to the first block together. Assign the AddrOffset variable to the appropriate Avalon-MM memory address that you set for the GPIO peripheral in Qsys.
   (b) The second switch block is grouped by data access width. Some pins are accessed 1 bit at a time. Others may be accessed as 8 bit bytes or even 32 bits at once. There are different groups for reads and writes. Find the appropriate group for your pins and add your IOCTL commands to them.
3. Add the pin to langford_util.cpp. There 2 to 3 places which you have to modify:
   (a) Add the pin name to the help message
   (b) For pins which can be read, add a new else if statement to the read handler which has the pin name which you put in the help message and the appropriate IOCTL_GET command.
   (c) For pins which can be written, add a new else if statement to the write handler which has the pin name which you put in the help message and the appropriate IOCTL_SET command.

On Latency and Performance

If you're reading this section it's because you're likely pushing boundries, or trying to accomplish something very neat. The default firmware uses very conservative settings, so we encourage you to tweak the settings to significantly improve your particular application[8].

The following section aims to provide you with the background, tools, and suggestions you can use to really get the most of what we offer.

[8] We really want to encourage you to get the most out of our hardware, so please don't hesitate to contact us if you have any questions, or face any problems, in getting the most out of your hardware.

Some of the things we talk about may seem daunting - but don't be put off. Try a couple of things, and if you're stuck, please don't hesitate to contact us for help!

Latency

We provide a complete test bench of the entire DSP chain. This test bench indicates that, after considering DMA unpacking, filtering, digital up or down conversion, and decimation and interpolation, the default latency is about 1ms.

Importantly, this latency doesn't reflect the capabilities of the hardware - much of this is based on digital up and down conversion (using multipliers), and the decimation and interpolation filters.

If you have a specific frequency of interest, you can choose to either implement an «IF» architecture using a fixed digital up or down conversion. You may also choose to replace the decimation or interpolation blocks with a filter of your own design.

Finally, for ultra-low latency, you might opt to remove the entire DSP chain, and carry out all processing on the host computer - drastically speeding up the entire process, and likely improving routing and timing[9].

[9] If you decide to pursue this approach, let us know - we'd love to see whether we can help you out.

DMA Transfers

The current driver uses a sub-optimal version of memcopy to implement the DMA transfers[10]. Fixing this would likely improve sustained transfer rate, and possibly latency.

[10] This shouldn't be too hard, but requires a good understanding of linux device drivers and PCIe interupts - we're working on it...

IQ Matching

If you're looking at this section, it's likely because you're looking at trying to get the most performance out of the board. You might even be asking yourself - why did you guys use two discrete ADCs?[11] How do I match them?

You're looking at two different problems - ideally, we want to match the amplitude, and phase at the ADCs.

[11] The answer comes down to cost - we're trying to pack in a lot of performance while still keeping the price low.

Amplitude Matching

If you're working on the high frequency branch, you'll notice that we have two seperate differential VGAs: one for the 'I' component, another for the 'Q' component. This allows for independant adjustment of the I and Q amplitudes prior to hitting the ADC.

You can also adjust the attenuation, within the ADC, by using the langford_adc_utility. This has the impact of changing the full swing potential on the ADC to between 1-2V. To set the fullswing attenuation on ADC A to 1V, type the following;

```
langford_adc_util /dev/langford A IntRef 0
```

You can disable this, and revert to the default behaviour, by typing:

```
langford_adc_util /dev/langford A IntRef -1
```

If you're feeling super adventurous, there's even room for a small resistive attentuation network just past the prior to the ADC.

Finally, you might adjust the amplitude of each channel by modifying the FPGA firmware. There's a convienent location just after the high pass filters in the verilog code.

Phase Matching

So here it becomes a little tricky. We suggest implementing an IQ correction filter in software - we're working on building an example implementation - which would allow you to artificially delay one (or both) of the wave forms.

To Be Continued...

We're likely going to add more to this section once we have a chance to catch our breath. If you have any questions in the meantime, please don't hesitate to contact us.

FPGA Implementation

Your most excellent choice of the Noctar Transceiver means that you'll have access to the FPGA firmware under a noncommercial license[12].

The FPGA firmware is written in Verilog, and uses a combination of Per Vices custom IP, and generated IP from Altera.

[12] If you're looking for a commercial license, please don't hesitate to talk to us. We're open for business!

Convention and Directory Structure

The top level directory contains two reasonably important directories:

system — This is top level system directory. It contains the contains the main Quartus project file, along with the top level verilog including the Quartus Megafunctions, and the Altera PCIe hard IP.

dsp — This is the DSP directory - it contains all the Per Vices IP necessary to implement the decimation, interpolation, filtering, and digital down/up conversion. This directory includes a complete testbenches simulating the entire dsp chain using iVerilog.

We now discuss each section in turn.

System - FPGA System

The FGPA system comprises of the following modules:

1. langford_qsys – main Qsys design

   This design contains the Altera Qsys modules, including the PCIe hard IP (HIP), DMA controllers, and GPIO controllers. This block can be edited with a point and click GUI. To open it, complete the following steps:

   (a) Download and install Quartus II 12.0 SP2, ensuring that you install all necessary modules for the Cyclone IV series. You can download it from: https://www.altera.com

(b) In Quartus, click File > Open Project.

(c) Select and open the langford_system.qpf project file.

(d) Click Tools > Qsys

(e) Qsys will prompt you to open a system. Select and open the langford_qsys.qsys file.

(f) You can now view the connections which make up this module and make changes as you see fit.

2. DcFifo64 – 64 bit dual clock FIFO

   This module serves as extra buffer space as well as clock domain synchronization. The ADC and DAC operate on different clocks than the DMA controller on the FPGA. The FIFO serves as a clock domain crosser for the data which is stream from/to the ADC/DAC and the PCIe DMA controllers.

3. TopRx – rx DSP chain

   All DSP performed on the FPGA for incoming data from the ADC is performed in this module. A detailed description can be found in the DSP Used in Phi document.

4. TopTx – tx DSP chain All DSP performed on the FPGA for outgoing data from the DAC is performed in this module. A detailed description can be found in the DSP Used in Phi document.

5. pcie_gx_reconfig – PCIe self reconfiguration block The Altera PCIe HIP has the capability to reconfigure its electrical properties to suite the host computer. This includes automatic equalization and preemphasis. This module controls this functionality.

A diagram detailing the FPGA system is shown in Figure 7.

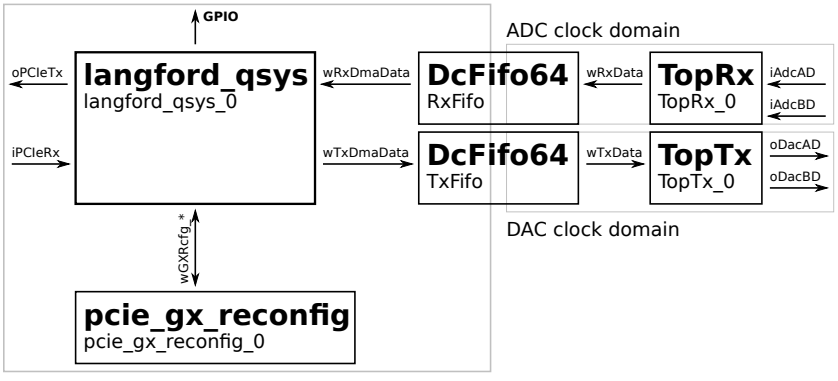


Figure 7: Noctar FPGA system. Control signals have been omitted for the sake of clairity.

DSP - Digital Signal Processing System

We use several DSP cores to deliver you the magic of SDR. Each DSP core occupies its own subdirectory in the dsp directory, with multiple DSP cores being integrated into the rx and tx DSP chains. The rx DSP chain is called TopRx and the tx DSP chain is called TopTx. Each chain is comprised of smaller DSP cores, chained together as building blocks. Let's take a closer look at how the Rx and Tx chains work...

Rx DSP Chain – TopRx

The Rx chain, shown in Figure 8, comprises of the following modules:

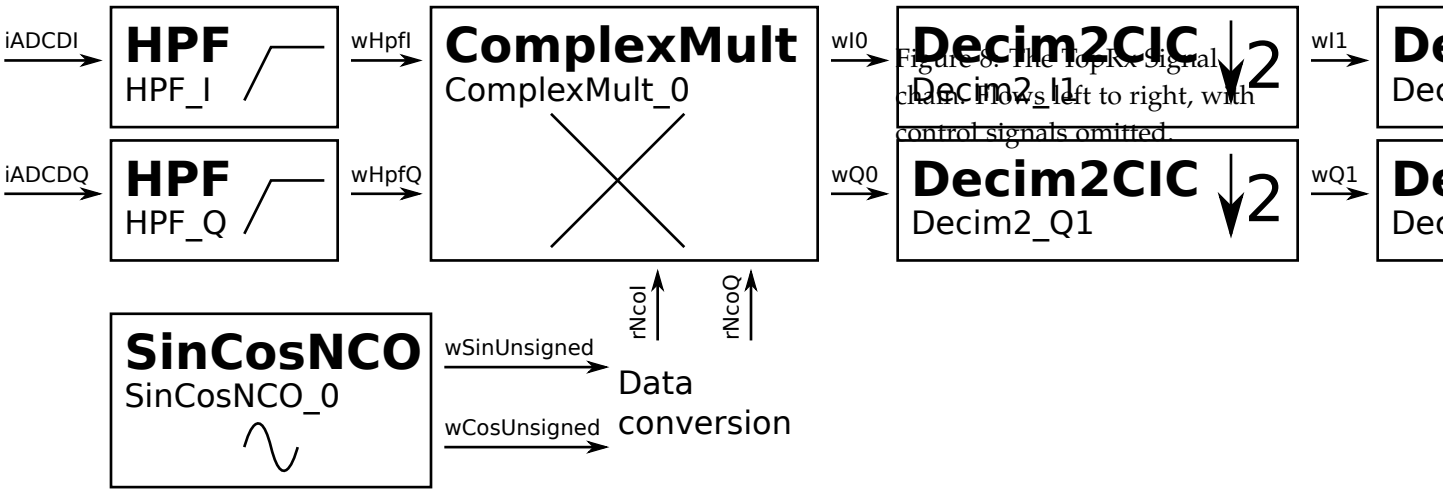


Figure 8: The TopRx signal chain. Flows left to right, with control signals omitted.

1. HPF – high pass filter

   The high pass filter removes DC offset. Cut off frequency is ~30 kHz when sampled at 125 MHz.

2. SinCosNCO – sine and cosine numerically controlled oscillator

   This NCO produces the signal required to perform digital down conversion. The output of this module is always synchronized at 90° apart.

3. ComplexMult – complex multiplier

   The complex multiplier takes two complex numbers and calculates the product, also a complex number. This performs digital IQ down conversion.

4. Decim2CIC – optional CIC half band filter and decimate by 2

This module allows for optional decimation by 2. This allows for lower digital transfer rates and narrower signal bandwidths while avoiding problems relating to aliasing.

Tx DSP Chain -TopTx

The Tx chain, shown in Figure 9, comprises of the following modules:

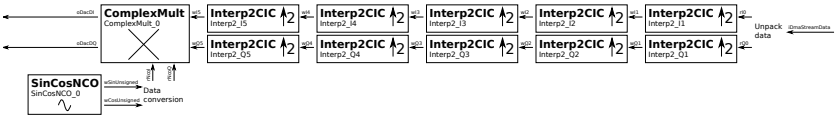


Figure 9: The TopTx Signal chain. Flows right to left, with control signals omitted.

1. Interp2CIC – optional interpolate by 2 and CIC half band filter

   This module allows for optional interpolation by 2. This allows for lower digital transfer rates and narrower signal bandwidths while avoiding problems relating to harmonic spurs.

2. SinCosNCO – sine and cosine numerically controlled oscillator

   This NCO produces the signal required to perform digital up conversion. The output of this module is always synchronized at 90^{o} apart.

3. ComplexMult – complex multiplier

   The complex multiplier takes two complex numbers and calculates the product, also a complex number. This performs digital IQ up conversion.

FPGA Simulation and Test bench

Verifying DSP Cores

Each DSP core can be tested separately or together. Two types of types of tests are done on each DSP core, each with their own purpose:

1. Simulation Quick results for ease of verification Used to check functional accuracy of each core
2. FPGA test bed
   - Check for synthesisability of code
   - Check for functional accuracy after synthesis and P&R
   - Perform timing analysis
   - Obtain rough estimates of resource utilization on FPGA

It is easy to run either test. The following subsections will describe the procedure needed to run both types of tests. Each DSP core is located in its own directory in the dsp directory. Each directory typically contains a tb_iverilog directory for simulations and tb_altera directory for FPGA test beds, with a complete, end-to-end, test available found in the dsp/system_tb directory[13].

[13] If you're playing with the DSP cores, it's worth ensuring that you haven't unintentionally broken anything by running the dsp/system_tb test bench prior to taking the time to synthesize everything!

Running Simulations

1. Install iverilog and gtkwave
2. cd into the tb_iverilog directory for the core of interest, or the system_tb directory to run a test of the entire dsp system.

   For example:

   ```
   $ cd langford/fw/dsp/HPF/tb_iverilog
   ```

3. From a terminal, run:

   ```
   $ make test
   ```

All necessary files will be generated and the simulation will run.

4. After the simulation is complete, gtkwave will launch with the simulation results. You can view these results and compare them against your expectations.

5. If you wish, you can modify the test bench by editing the tb_iverilog.v file.

   For example: git://langford/fw/dsp/HPF/tb_iverilog/tb_iverilog.v
   You can repeat simulation after modifying the test bench by restarting from step 3.

6. You can modify the DSP core by editing the DSP core file. For example: dsp/HPF/HPF.v

   You can repeat simulation after modifying the test bench by restarting from step 3.

Figure 10 shows a screen capture of one such simulation.

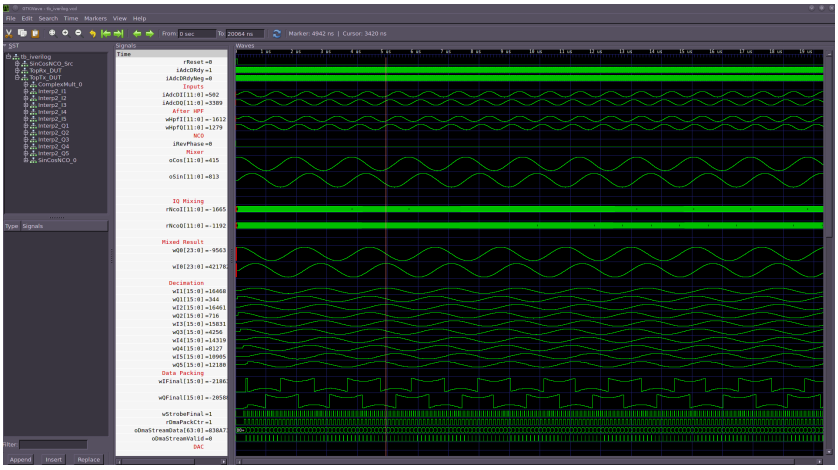


Figure 10: Screencapture of end-to-end DSP testbench using iVerilog, found in the dsp/system_tb directory.

FPGA Test beds

1. Download and install Quartus II 12.0 SP2.

2. Ensure that TalkBack is enabled. This will enable basic SignalTap functionality[14].

3. Launch quartus. On Linux installations, Quartus II is installed to: $PREFIX/quartus/bin/quartus

4. Power up the target FPGA board and connect your USB Blaster Cable.

5. In Quartus, click File > Open Project.

[14] Follow these steps to enable TalkBack: http://www.altera.com/support/kdb/solutions/rdo6202

6. Open the tb_altera.qpf project file located in the tb_altera directory of the DSP core of your choice.
   For example: dsp/HPF/tb_altera/tb_altera.qpf
7. In Quartus, click Processing > Start Compilation. This will generate the design file. All timing and Once this is complete, you will get a message informing you of such.
8. Click Tools > SignalTap II Logic Analyzer
9. Ensure that USB-Blaster is selected under Hardware and the target FPGA is detected.
10. Click on the program button.
11. Once the device is programmed, click Processing > Run Analysis. Captured data will be displayed in the Data tab.

Notes

- If during the synthesis for the FPGA test bed that you get a message that a file is not available, its likely that it relies a file that is generated by the makefile. Run make in the tb_iverilog directory. See section on Running Simulations for more information.
- Some test benches include a variety of stimulus for simulation. You'll see different sections commented out - simply comment or uncomment a block to enable a particular stimulus.

If you have more than one stimulus driving an input, you'll get an error informing you as much.

- If you want to examine other signals with SignalTap, you can do so by clicking on the Setup tab and double clicking on the background of the table. You can select any number of signals to add to your analysis. If the signal is not listed with the SignalTap II: post-fitting filter applied, then it has been removed during synthesis.

FPGA Programming

We've made it easy to program the Noctar FPGA. You're going to require two things - a USB Blaster cable, which costs money, and a copy of the Quartus II software, which is gratis on the Altera website[15].

[15] http://www.altera.com

How to program the FPGA

1. Apply power to the Phi device. You can do this either by inserting this device into a computer with a PCIe slot and turning it on or into a dummy PCIe slot.
2. Connect the 10 pin header of the JTAG programmer to JF1. Align the red wire on the programmer cable to pin 1.
3. Connect the JTAG programmer's USB cable to the JTAG server.
4. Launch Quartus II
5. In Quartus, click Tools > Programmer
6. Ensure that the reported hardware is USB-Blaster. If it does not, plug in the the USB cable of the JTAG programmer, close the Quartus Programmer and repeat step 6.
   (a) (Optionally) If you click on «Auto-detect», it should recognize the FPGA.
   (b) If you do use Auto-detect to test the circuit, remember to delete the FPGA and image - you don't want anything there.
7. From the menu, click Edit > Add File
8. Browse and open your file (output_file.jic)
9. Enable Program/Configure and Verify for the EPCS16 device
10. Click Processing > Start
11. The yellow LED on the board will blink and turn on as the program is downloaded.

(a) When the programming is complete, the Quartus program will indicate as such.

12. Turn off your computer. Note that you should do a cold shut down to ensure the device enumerates properly - a reboot doesn't always do it.

13. Congratulations! You have successfully programmed the FPGA It is now safe to remove power from the device. Next time the device is powered on, the FPGA status LED (DF1) will come on automatically[16].

[16] If the device doesn't show up on lspci, then either the image you burned is no good, or you didn't do a hard power down of the card.

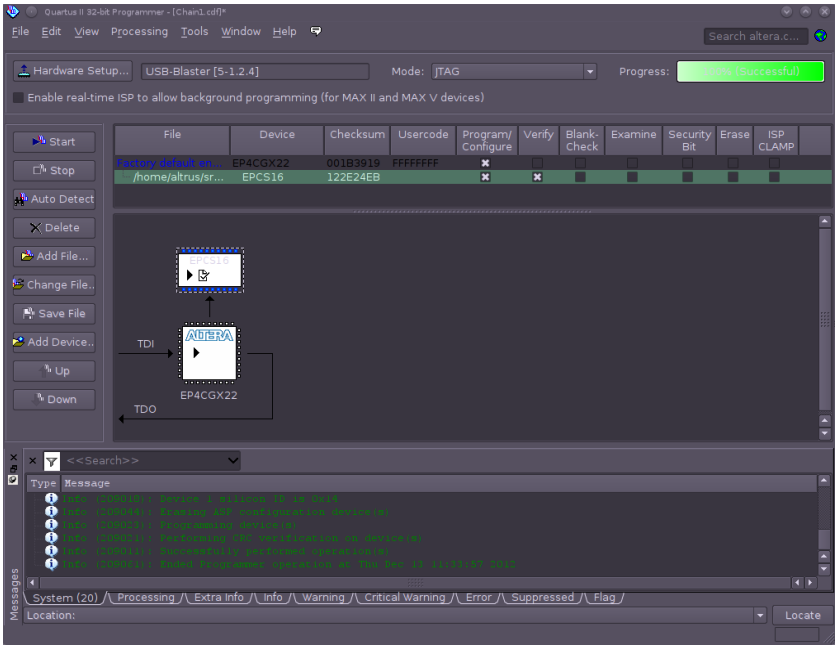


Figure 11: Quartus 12.0sp2 programming window after successfully programming FPGA.

Additional Notes

- You can program the device remotely, if you set up a JTAG server, but this can be a little unwieldy.

Driver Implementation

This document outlines the considerations taken into the design and implementation of the driver for the NOCTAR device. The goal is to allow you to easily add features, modify, or debug, the NOCTAR driver.

The files required to compile the driver are located in the git://langford_driver directory. There are 4 files which are used to make the driver:

1. Makefile

   Contains the rules to compile and load the driver and utilities, and creates the initial character device.

2. langford.c

   Main source file for the driver.

3. langford.h

   Constants, include statements and other preprocessor macros for the driver.

4. langford_ioctl.h

   Contains all the constants for ioctl calls. Can be included by user mode programs (e.g. langford_*util) too.

The NOCTAR device is represented on a Linux computer as a character device. This device needs to be created prior to accessing NOCTAR[17]:

1. Run langford_init. This program parses your /proc/devices file and automagically creates the character device.

2. View the kernel logs when the driver is loading. The driver will display the exact command needed to created the character device[18].

3. Run make install in the langford_driver directory.

[17] You can create this device using mknod (look in /proc/devices for major number) - see the langford_init source for a script.

[18] There's a lot of useful information to be found by looking at the dmesg file when langford runs.

Variable Naming

Confusion may arise from the notion of read or write and their use in variable names. This is understandable - it arises from the perspective and context within which they are used. To illustrate, to a user mode program, a file write is used to transmit data toward the SDR and out into the real world. However, for NOCTAR[19], bus writes actually move data from the real world into the computer, the exact opposite direction as a software write. It's difficult to maintain a consistent variable naming convention that meets semantic meaning and expectancy. To reduce the confusion, we now outline the variable names used in the driver and which side they are used on.

[19] NOCTAR is a SDR device with an FPGA in bus mastering mode (to perform DMA transfers)

Description

RX Variables in from the real world.

TX Variables out to the real world.

DMA Pointers (RX/TX based)

RX pRxDmaBuffs RxDmaBuffsBusAddr

TX pTxDmaBuffs TxDmaBuffsBusAddr

User mode IO handler (Software-centric)

RX cdev_read

TX cdev_write

DMA transfer thread (Software-centric)

RX dev_read_thread dev_read_task

TX dev_write_thread dev_write_task

Buffer for character device (Software-centric)

RX cdev_read_buff_mutex cdev_read_buff cdev_read_buff_start cdev_read_buff_end

TX cdev_write_buff_mutex cdev_write_buff cdev_write_buff_start cdev_write_buff_end

DMA constants (RX/TX based)

RX DMARXBUFFS DMARXDESCS DMARXDESCDELAY

TX DMATXBUFFS DMATXDESCS DMATXDESCDELAY

BAR constants (FPGA-centric)

RX WRCSR* WRDESC*

TX RDCSR* RDDESC*

Driver Initialization Flow

When the driver is loaded, many events happen. In particular, these functions will be called in the following order:

1. driver_initial

   This function is called when insmod is called. This function performs the following tasks in the following order:

   (a) Registers the driver with the PCI subsystem

   (b) Registers the character device

2. driver_probe

   This function is run when a new PCI device is discovered on the system which may be the NOCTAR device. This function checks the device for compatibility and initializes it. Specifically, this function performs the following tasks in the following order:

   (a) Enables the device via the PCI subsystem.

   (b) Checks device version (in PCI configuration space) for compatibility.

   (c) Maps BAR regions to pointers accessible form kernel space.

   (d) Discovers and sets up DMA settings with the kernel.

   (e) Initialize the SDR peripherals (ADC, DAC) and put them in low power mode.

At this point, the driver is completely loaded and the user can interact with the device via character device reads, writes or ioctl calls. During removal;

1. driver_remove

   This function is executed when a the NOCTAR device is removed or if the driver is unloaded. This function undoes all the tasks performed in driver_probe.

2. driver_exit

   This function is called when the driver is unloaded. This function undoes all the tasks performed in driver_initial.

IO Flow

When the driver has completely loaded, it goes away into the background, waiting for the character device file to be opened. When the device is opened, the driver sets up the computer for DMA transfers. When all IO has complete and the file is closed, DMA transfers are stopped. Specifically, when the character device is being opened, the following tasks are completed by cdev_open:

1. Allocate DMA memory buffers
2. Calculate and write address translation table to the FPGA
3. Allocate character device memory buffer
4. Power on ADC and DAC
5. Spawn DMA read and write device threads

The file close function (cdev_release) undoes all the tasks completed by cdev_open.

After the file is open, various IO can take place. Reads and writes interacts with data which is shared between the kernel space driver and user space application. Because everything is multi threaded, these tasks happen concurrently.

We illustrate the interaction process between the user program, kernel module and FPGA for the Receive and Transmit sides in Figures 12 and 13[20].

[20] This is really important - because everything happens at once, you can't immediately rely on synchronous patterns.

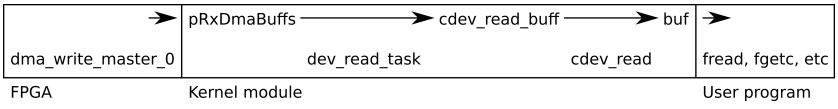


Figure 12: Rx Data flow

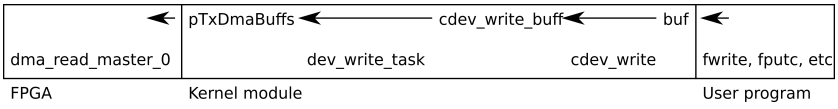


Figure 13: TX Dataflow

Character Device Data Format

Due to asymmetric PCIe DMA transfer bandwidths (see FPGA System Description for explanation), the data format for RX is different than TX. Both are designed to take advantage of as much of the available bandwidth as possible.

Receive Data (High Frequency Branch)

Rx data are provided in IQ pairs. Each IQ pair is 32 bits, with the I and Q components represented by two 16 bit signed integers. Specifically, the format for RX (reading from the /dev/langford device) on the high frequency RX stage is represented in Table 5. To read this data inside GNU radio, you can use a flow chart similar to that shown in Figure 5.

Position	0	1	2	3	4	5	6	7	8
Data	I[0]		Q[0]		I[1]		Q[1]		etc

Table 5: RX HF data structure

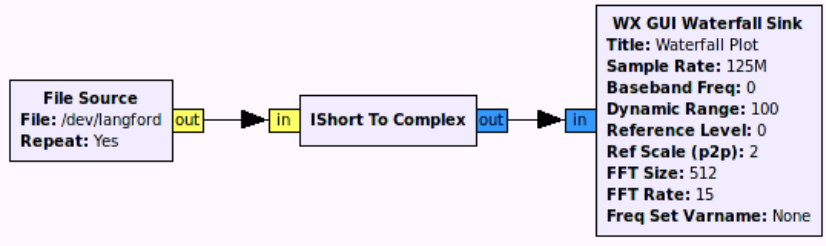


Figure 14: RX HF flow structure

Receive Data (Base band Branch)

We receive base band shorts by interleaving ADC samples independently triggered on the positive and negative clock edges. We don't presently carry out base band DSP. To ensure ADC interleaving, we use the Langford ADC utility;

$langford_adc_util /dev/langford A ClkSel 0

$langford_adc_util /dev/langford B ClkSel 1

The data and flow structure for reading base band Rx data are shown in Table 6 and Figure 15.

Position	0	1	2	3	4	5	6
Data	P[0]	N[0]	P[1]	P[1]	P[2]	N[2]	etc

Table 6: RX BB data structure

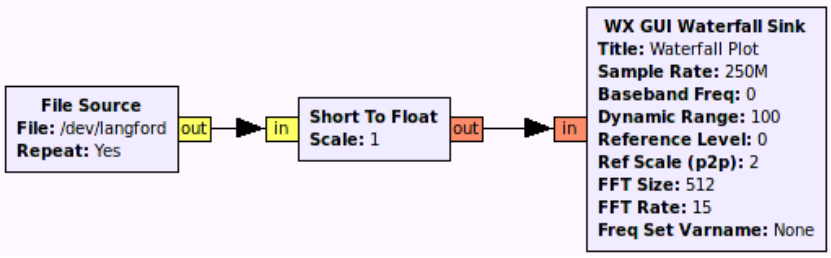


Figure 15: RX BB flow structure

Transmission Data (High Frequency Branch)

TX data are provided in IQ pairs. Each IQ pair is 16 bits. I is represented as a 8 bit signed integer. Following I is Q also represented as a 8 bit singed integer. The data structure for high frequency transmission is shown in Table 7, with the flow chart shown in Figure 16.

Position	0	1	2	3	4	5	6
Data	I[0]	Q[0]	I[1]	Q[1]	I[2]	Q[2]	etc

Table 7: TX HF data structure

Figure 16: Tx HF flow structure

Transmission Data (Low Frequency Branch)

For low frequency output, we only use one end of the IQ pair - the other end is essentially left dangling, although the ports have been exposed As it happens, we use the real part, so base band transmission, using the structure in Figure is as straightforward as building the graph shown in Figure 17.

You can access the DAC output by probing the ports on the top left corner of the shielded section located just to the left of the DAC.

Position	0	1	2	3	4	5	6
Data	I[0]	I[1]	I[2]	I[3]	I[4]	I[5]	etc

Table 8: TX BB data structure

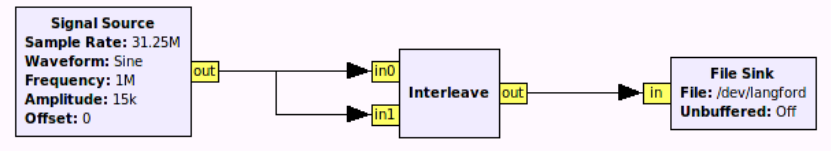
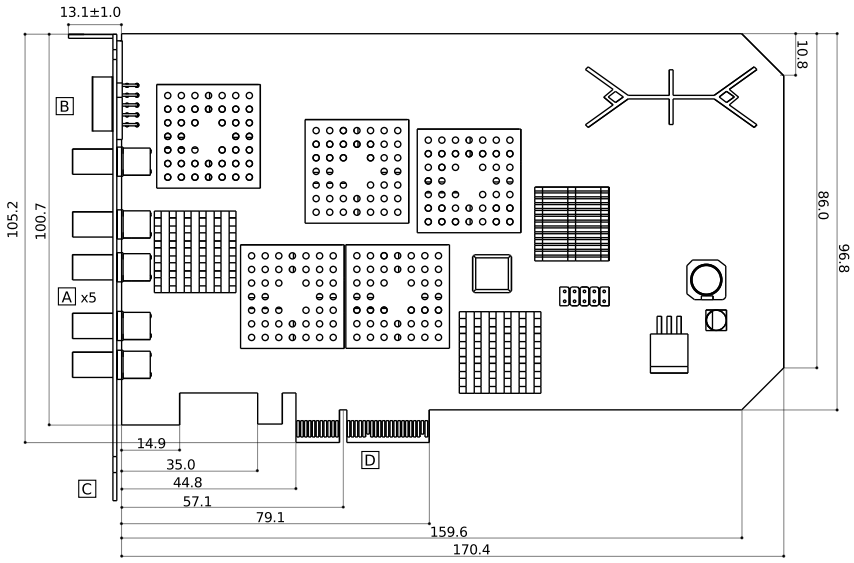
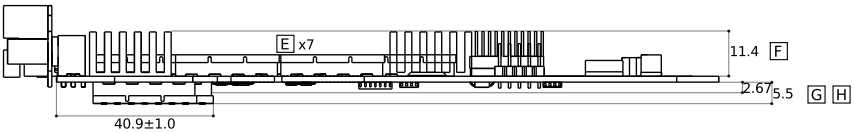
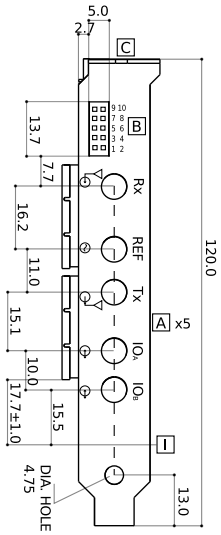


Figure 17: TX BB flow structure

Appendix I: Mechanical Drawings







NOTES

- A Female 50 ohm SMA connectors, MPN: WM5537 (Molex Inc.)
- B Female connector, MPN: PPPC052LJBN-RC (Sullins Connector Solutions)
- C PCIe IO bracket
- D Hard gold fingers
- E EMI frames and cover, MPN: BMI-S-203 (Laird Technologies EMI)
- F Max component height (top of board)
- G PCIe solder side height restriction (single slot requirement)
- H Max component height (bottom of board) (may require 2 PCIe slots)
- I Reference edge of board (component side)

DRAFTED BY: B. Malatest
DATE: 2012-12-10
CHECKED BY:
DATE: NOT FOR PRODUCTION USE

NOCTAR TRANSCEIVER

SIZE: A3

GENERAL TOLERANCES (UNLESS OTHERWISE STATED)
(1) PLC. DECIMAL ±0.5mm
(2) PLC. DECIMAL ±0.05mm
Regardless of feauture size

METRIC
Dimensions in mm

SCALE: 1:1
WEIGHT (kg): 0.4kg
DRAWING NUMBER: PRELIMINARY DRAFT - REVISION 0.1
SHEET: 1 of 1

Appendix I: Pin Descriptions

For the engineers, masochistic, or curious, this section exhaustively lists all the options you can change with langford. Consider this the hardware API that allows you to manipulate the data that NOCTAR collects[21].

[21] If you're not sure how these parameters affect RF performance, see the system overview section to see exactly how they'll effect signal quality.

Pin Name

Allowed Values Description

N21ato, N22ato

0-255 RX and TX high frequency variable gain setting respectively. 0 = 0 V (minimum gain) and 255 = 5 V (maximum gain), other values are linear interpolated.

N3GAIN

0-255 RX low frequency variable gain setting. 0 = 0 V (minimum gain) and 255 = 1 V (maximum gain), other values are linear interpolated.

N61CE, N62CE

0,1 Chip Enable pin of serial interface for RX and TX frequency synthesizer respectively.

N61DATA, N62DATA

0,1 Data pin of serial interface for RX and TX frequency synthesizer respectively.

N61CLK, N62CLK

0,1 Clock pin of serial interface for RX and TX frequency synthesizer respectively.

N61LE, N62LE

0,1 Latch Enable pin of serial interface for RX and TX frequency synthesizer respectively.

N61MUXOUT, N62MUXOUT

0,1 Multiplexor output pin of serial interface for RX and TX frequency synthesizer respectively.

N61LD, N62LD

0,1 Lock detect pin of serial interface for RX and TX frequency synthesizer respectively

N3ENB

0,1 Enable low frequency amplifier.

N3HILO

0,1 Enable High or Low Gain stages (Low gain = -4.5..43.5dB, High gain = 7.5..55.5dB). Overall gain determined in conjunction with N3GAIN.

NASRxA, NASTxA

0,1 Control pin for RF switch. This pin controls the A pin of the switches. The B pin is always guaranteed to take the logical compliment of the A pin.

N7CLK

0,1 Clock pin of the serial interface for the differential ADC VGA.

N7CS

0,1 Chip select pin of the serial interface for the differential ADC VGA.

N7SDI

0,1 Serial data to the serial interface for the the differential ADC VGA.

N7SDO

0,1 Serial data from the serial interface for the the differential ADC VGA.

DACIQSel

0,1 IQ framing pin for the DAC. Used for single port mode.

DACReset

0,1 Reset pin for the DAC. Used also to enable pin mode.

DACCSB

0,1 Enable mix mode for the DAC. Used also as the chip select pin for the DAC's serial interface.

DACSDIO

0,1 Enable unsigned binary data format for the DAC. By default, the data format is signed binary. Used also as the data pin to/from the DAC's serial interface.

DACSClk

0,1 Enable single port mode for the DAC. By default, dual port mode is enabled. Used also as the clock pin for the DAC's serial interface.

DACSDO

0,1 Power down the DAC. Used also as the data pin from the DAC's serial interface.

ADCAPD, ADCBPD

0,1 Power down the ADC.

ADCASClk, ADCBSClk

0,1 Enable signed binary data format for the ADC. By default, the data format is unsigned binary. Used also as the clock pin for the ADC's serial interface.

ADCASDIO, ADCBSDIO

0,1 Enable LVDS outputs for the ADC. By default, the data format is CMOS. Used also as the data pin to/from the ADC's serial interface.

ADCAnCS, ADCBnCS

0,1 Enable pin mode for the ADC. Used also as the chip select pin for the ADC's serial interface.

RXPhase, TXPhase

0-$(2^{32}-1)$ Phase increment for the RX and TX NCO respectively. $f_{NCO} = \frac{Phase\,increment}{2^{32}} * f_{sr,ADC}$, where $f_{sr,ADC} = 125\text{MHz}$.

RxDecEn, TxIntEn

0,1 Enable RX decimation and TX interpolation respectively. To reduce bus bandwidth, you can choose to perform decimation and interpolation. Each bit enables a decimation or interpolation state. For example, 0x03 enables 2 stages. For another example, 0x01 is the same as 0x08.

RXRevFreq, TXRevFreq

0,1 Reverse (negate) DDC (digital down conversion) and DUC (digital up conversion) frequency.

RXDspEn, TXDspEn

0,1 Enable the entire RX or TX DSP chain respectively.

NRXTALSEL

0,1 Crystal select pin of the clock distribution IC. This determines whether the internal or external reference is used. If you opt to use the external reference, ensure that it is a 20MHz reference.

NRVCOSEL

0,1 Bypass the clock distribution IC. If you want to directly drive the frequency dividers from reference, this is the way to do it...

Appendix II: GPIO Address Offsets

The GPIO (general purpose IO) pins are used to provide IO functionality to both control signals within the FPGA as well as other board level peripherals. For example, the frequency for digital down/up conversion and the pins for the serial interface for the ADC VGA driver is set this way.

GPIO pins are grouped into 32 bit registers, all of which reside on BAR0 of the PCIe HIP. In the Qsys design, each 32 bit register is implemented as a PIO device. The PIO device is an Altera IP which can be found under Peripherals > Microcontroller Peripherals in the Component Library. Currently, there are 6 32 bit registers. This following subsections summarizes the assignment of GPIO pins in the Noctar system.

Bit	Assignment
7:0	N21ato
15:8	N22ato
23:16	N3GAIN

Table 9: PIO0 - Output - Address offset: 0x00008000 - Analog pin control pins.

Bit	Assignment
0	N61CE
1	N61DATA
2	N61CLK
3	N61LE
4	N62CE
5	N62DATA
6	N62CLK
7	N62LE
8	N3ENB
9	N3HILO
10	NASRxA
11	NASTxA
12	N7CLK
13	N7CS
14	N7SDI
15	DACIQSel
16	DACReset
17	DACCSB
18	DACSDIO
19	DACSClk
20	ADCAnOE
21	ADCAPD
22	ADCASClk
23	ADCASDIO
24	ADCACS
25	ADCBnOE
26	ADCBPD
27	ADCBSClk
28	ADCBSDIO
30	DACSDO

Table 10: PIO1 - Output - Address offset: 0x00008100 - Physical output pins.

Bit	Assignment
0	N61MUXOUT
1	N61LD
2	N62MUXOUT
3	N62LD
4	N7SDO
5	USER GPIO #1
6	USER GPIO #2
7	USER GPIO #3
8	USER GPIO #4
9	USER GPIO #5

Table 11: PIO2 - Input - Address offset: 0x00008200 - Physical input pins, header GPIO

Bit	Assignment
31:0	RXPhase

Table 12: PIO3 - Output - Address offset: 0x00008300 - Phase increment for rx NCO

Bit	Assignment
31:0	TXPhase

Table 13: PIO4 - Output - Address offset: 0x00008400 - Phase increment for tx NCO

Bit	Assignment
7:0	RXDecEn
15:8	TXIntEn
16	RXRevFreq
17	TXRevFreq
18	RxDspEn
19	TxDspEn
20	NRXTALSEL
21	NRVCOSEL
22	RxFifoClr
23	TxFifoClr
24	RxIsSigned
25	TxIsSigned
26	USER GPIO #1
27	USER GPIO #2
28	USER GPIO #3
29	USER GPIO #4
30	USER GPIO #5

Table 14: PIO5 - Output - Address offset: 0x00008500 - DSP, clock control, header GPIO

Epilogue

«Take chances! Make mistakes[22]! Get messy!»
Ms. Frizzle

[22] Per Vices believes in the value of mistakes as research experience. We encourage, whenever possible, that you try and limit your mistakes to the (mostly) reversible kind. However, we also appreciate those times when you know better, but do it anyways.