Tutorial -3

Programming in USRP-FPGA
(USRPN210- Xilinx Spartan 3A-DSP3400)

Objective:

This article gives a description of the main functionality of the FPGA in the Universal Software Radio Peripheral (USRP). It also includes a brief tutorial on how to reprogram the FPGA. This tutorial is belongs to USRP N210 devices having FPGA device of Xilinx Spartan 3A-DSP3400. The design flow may be differ in next generation USRP devices.

Software Defined radios:

Software Defined Radio (SDR) is a "Radio in which some or all of the physical layer functions are software defined". Traditionally, these physical layer functions are implemented in hardware. Later, if there is a need for slight modification in any one of the function, then it should be redesigned and implemented in hardware. The hardware redesign is costly and time consuming. By having these function defined in software, they become flexible and reconfigurable. This saves prototyping time. The signal processing flow for a generic SDR is shown in Figure 1.1.

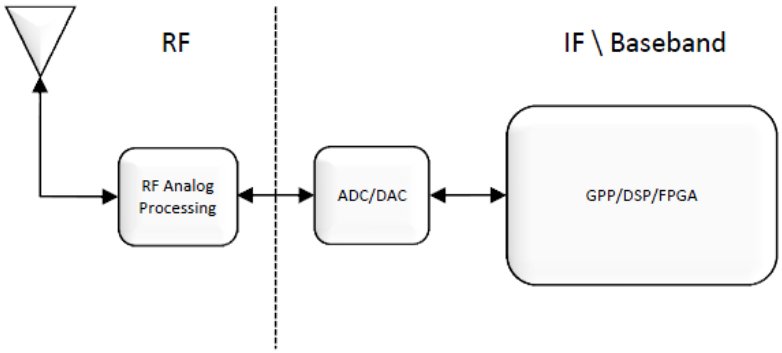

Figure 1.1 a software defined radio.

The most common computational resources on which SDR is implemented are GPP (general purpose processor), DSP (digital signal processor), and FPGA (field-programmable gate array). Each of these platforms has its own advantages and disadvantages. Although GPPs are relatively easy to program, test, and verify, they are often not well-suited for parallel, signal-processing intensive, or real-time constrained operations. DSPs are optimized for certain signal processing tasks, but they are harder to program than GPPs and not optimized for parallel computations.

FPGAs are well-suited for parallel and signal-processing intensive computations, but they are much harder to program and verify. However, FPGAs, with their ability to perform computations in parallel, support high throughput and high sampling rate that GPPs or DSPs are not able to achieve. FPGAs bring significant reduction in NRE (non-recurring engineering) costs. Once a prototype is built and fault is found in the design, FPGAs can be reconfigured with a modified bit-stream and can be tested again. Because of FPGAs are high-performance like ASICs and flexible like GPPs and DSPs, they are an ideal

platform on which prototype can be built and tested. Table 1 compares the different SDR families and choosing these one of the SDR platforms is depends on the applications.

SDR Platform	Frequency range	Computational resource	Maximum signal bandwidth	Supported OS	Supported software	Price (approx)
Microsoft Research SORA	2.4 GHz; 5 GHz	GPP (all processing done by multicore PC)	20 MHz	Windows XP	SORA SDK	$5000
Datasoft's Typhoon SDR DP	400 MHz – 4 GHz	Xilinx Spartan-6 FPGAs; dualcore OMAP	20 MHz	Linux; VxWorks; Android	GNU Radio	$10000
Lyrtech's SFF SDR DP	200 MHz – 1 GHz; 1.6 – 2.7 GHz; 3.3 – 3.8 GHz	Virtex-4 SX35 FPGA; DM6446 DSP	22 MHz	Linux	MATLAB; Simulink; Real-Time Workshop	$14000
CRC's Coral CR Platform	2.4 GHz; 5.8 GHz	FPGA	20 MHz	Linux Fedora Core or equivalent	Linux-OpenWRT with customized MadWIFI driver	$6000
Ettus' USRP-1 with WBX	2.3 – 2.9 GHz	Altera Cyclone FPGA	8 MHz	Linux; Windows; MAC OS X	GNU Radio; Simulink; LabView (test drivers)	$1000
Ettus' USRPN210 with WBX	50 MHz–2.2 GHz	Xilinx Spartan 3A-DSP3400 FPGA	50 MHz	Linux; Windows; MAC OS X	GNU Radio; LabView (test drivers)	$2000

Table.1.1 comparison of different SDR platforms

Universal Software Radio Peripheral (USRP)

The Universal Software Radio Peripheral was designed as a low cost SDR solely for the purpose of running GNU radio applications. Fully developed by Matt Ettus, it is a very flexible platform and can be used to implement real time applications. It is the bridge between the software world and the RF world.

All USRP products are controlled with the open source UHD driver, which is free and open source software called GNU Radio which is used to create complex software-defined radio systems.

Working as a receiver, USRP downconverts the RF signals and sends the baseband samples over a high speed Ethernet link to host system running with GNU Radio software while working as a transmitter USRP upconverts the baseband samples from host system to RF signal to be transmitted. In this transceiver system several RF analog parameters are controlled by the software running on the host system.

It is also possible to reprogram the USRP internal FPGA, using the tool Xilinx ISE Design Suite by utilizing the FPGA resources in order to reduce the intensive work of host system with UHD software (GNU Radio). The following table gives the specification of USRP N210 device.

FPGA	Xilinx Spartan 3A-DSP3400
ADC	14-bits 100 MS/s
DAC	16-bits 400 MS/s
RF Bandwidth	50 MHz of instantaneous RF bandwidth in 8-bit mode 25 MHz of instantaneous RF bandwidth in 16-bit mode
Connectivity	Gigabit Ethernet Interface

Table.1.2 USRP-N210 device specification

FPGA (Field-Programmable Gate Array)

FPGA is a reconfigurable hardware consisting of configurable logic blocks (CLBs) and macro blocks connected via programmable interconnects. The USRP N210 devices have Xilinx Spartan-3A DSP FPGA which consists of four types of macro blocks in addition to CLBs: XtremeDSP DSP48A Slice, Block RAM, Input/Output Blocks, and Digital Clock Manager. CLBs usually consist of look-up tables, flip-flops, and multiplexers, but they can vary among different FPGA devices.

Xilinx Spartan-3A DSP FPGA has four slices in each CLB. A slice consists of two LUTs (Look-Up Tables), two flip-flops, two multiplexers, and a carry-chain. Although CLBs can be used to implement multipliers and adders, Spartan-3A FPGA provides XtremeDSP DSP48A slices that are dedicated for 18-bit by 18-bit multiplication and 48-bit accumulation for MAC (multiply-accumulate) operations. The DSP48A slices are ideal for implementing FIR filters which require adder, multiplier, and storage elements. They can be highly pipelined to provide maximum clock frequencies of 250 MHz

FPGA	System Gates	Equivalent Logic cells	CLBs	Slices	Distributed RAM bits	Block RAM bits	DSP48As	DCMs	Maximum User I/O
xc3sd1800a	1800K	37,440	4160	16,640	260K	1512K	84	8	519
xc3sd3400a	3400K	53,712	5968	23,872	373K	2268K	126	8	469

Table.1.3 FPGAs of USRP N210 (xc3sd3400a) and USRP N200 (xc3sd1800a)

Data Flow between Host system and USRP:

Figure 1.2 explains how signal chain is created between the host PC system with GNU Radio and the USRP N210 with WBX. In this signal chain FPGA takes important role in processing the data flow. The main functionality of USRP-FPGA is shown by figure1.3.

RF Frontend:

In many cases, the RF frontend, the mixers, filters, oscillators and amplifiers required to translate a signal from the RF domain and the complex baseband or IF signals. This job is done by the daughter board (here we use WBX as a daughter board) which is mounted on the USRP.

ADC:

The baseband of IF signals are then sampled by ADCs, and the digital samples are clocked into an FPGA. Here the ADC performs 100MSps with the resolution of 14bits.

FPGA (DDC chain):

The USRP N210 has Xilinx FPGA Spartan 3A-DSP3400. This FPGA is configured as a System on Chip (SOC) where different Intellectual Property (IP) cores of the Ethernet MAC, UART, I2C, Interrupt controller, Buffer pool etc. are connected via the Wishbone bus. On this Wishbone bus, a 32-bit soft-core processor ZPU acts as master while the other fourteen IP blocks are slaves (refer fig1.5).

The default stock FPGA image provides digital down-conversion and some other functionality, which includes fine-frequency tuning and several filters for decimation. Coordinate Rotation Digital Computer

1. **CORDIC:** (CORDIC) block which does the fine frequency tuning for the required center frequency called Digital Down Conversion (DDC).
2. **CIC and Half-band filters:** After the DDC is performed, the samples are decimated with the Cascaded Integrator Comb Filter CIC filter and the two half-band decimators.
3. **Packet Router:** After decimation, the samples are sent to the VRT (VITA Radio Transport) core for packetization. The packets are then sent to the Ethernet MAC to be transported to the host computer.
4. **Host PC with GNU Radio:** In the host computer all the baseband processing is done using the GNU Radio. This includes source coding, channel coding, line coding, modulation/demodulation, filtering, up-sampling etc.

In case of transmitter, reverse of the above procedure occurs. Figure.1.4**.** shows how base band samples from the host are interpolated before being sent to the DAC for transmission and how the samples from the ADC are decimated and transmitted to the host system.

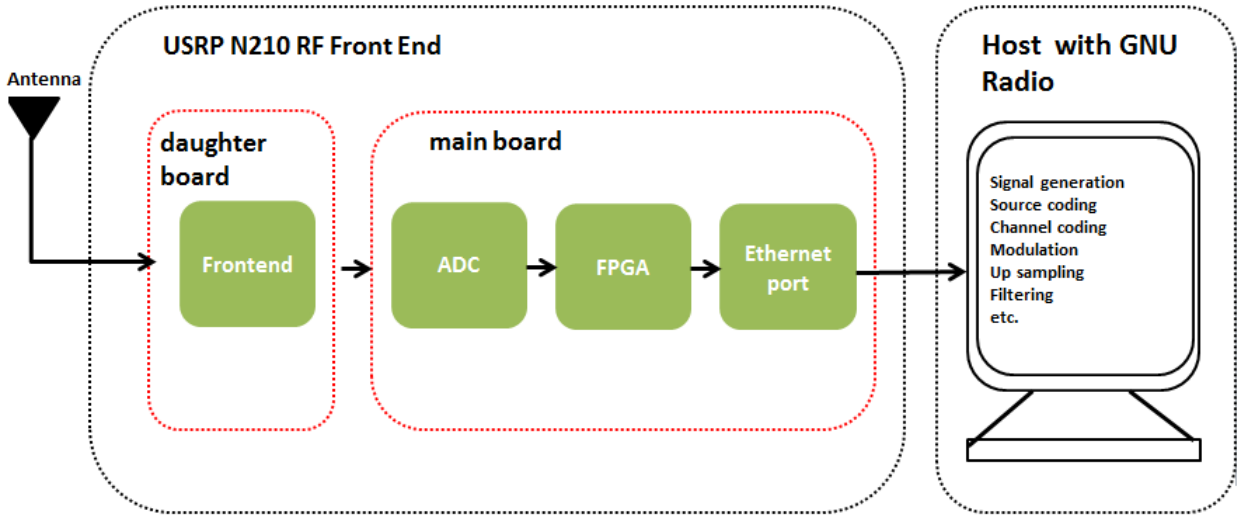

Figure.1.2. Data Flow between GNU Radio and USRP

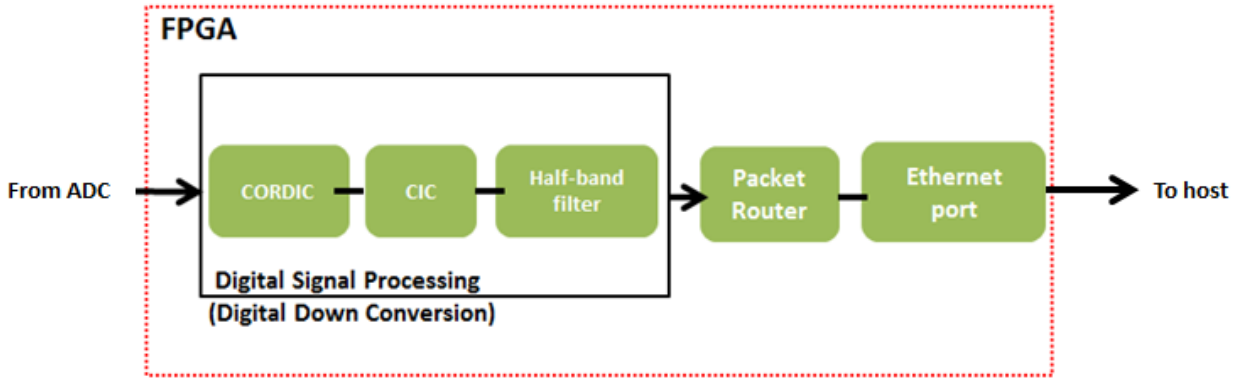

Figure.1.3. USRP-FPGA Functionality

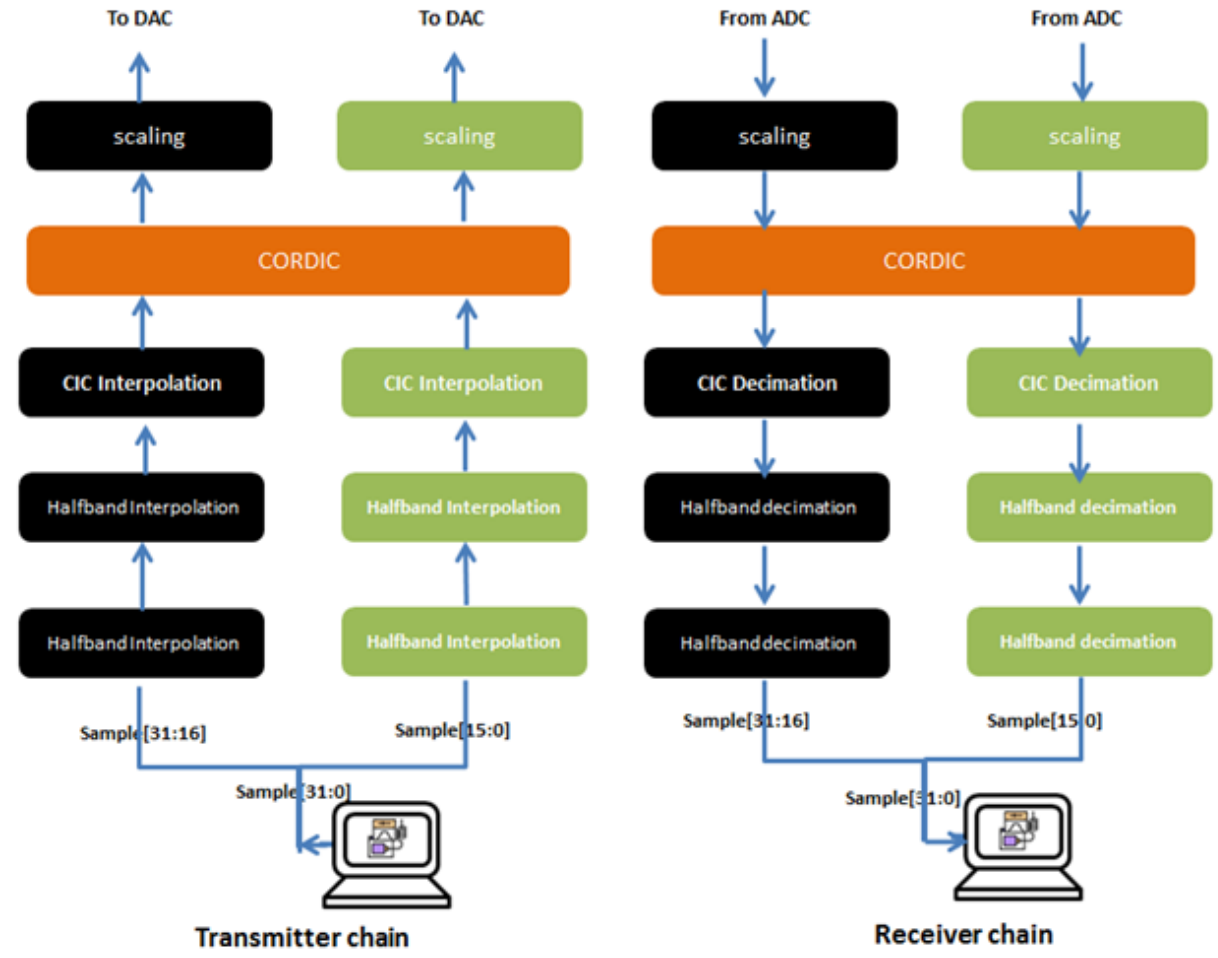

Figure 1.4 Transmitter and receiver signal chain within USRP-FPGA

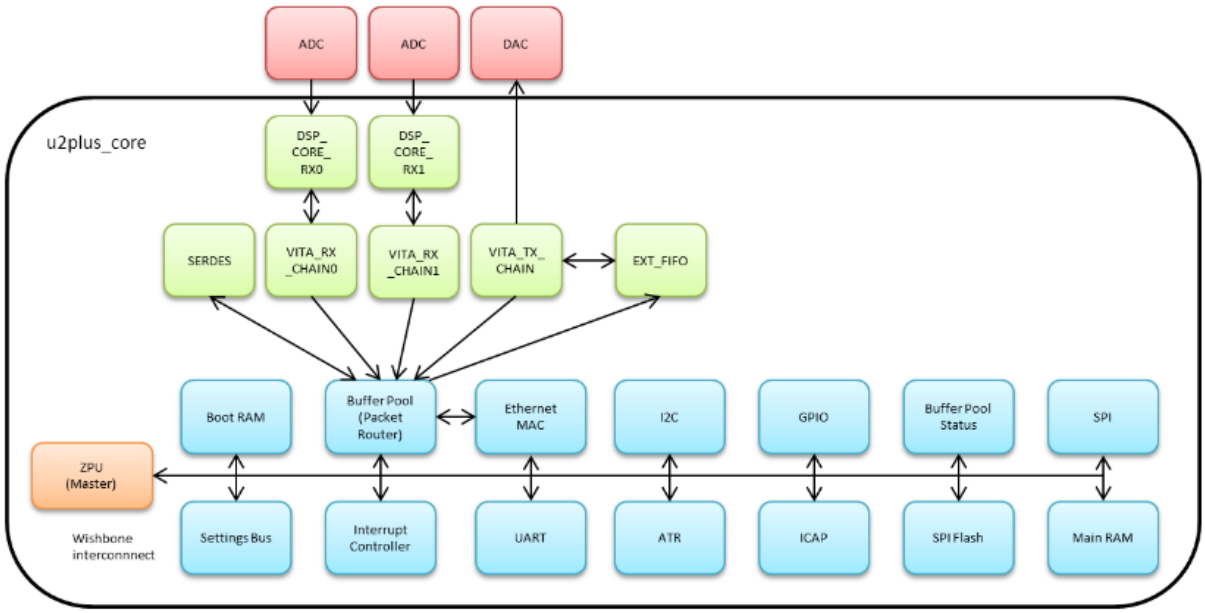

Figure.1.5 SOC in USRP's FPGA

FPGA Source files (Receiver signal processing chain):

The USRP-FPGA source code files are available in the hardware driver repository from following website: **https://github.com/EttusResearch.** The following modules (.v files) fit together to perform the FPGA functions of receiver signal processing chain. Start with the **u2plus.v** and traverse the module hierarchy:

1. **u2plus.v-** This is a Verilog source file which instantiates the u2plus_core to represent the actual signal flow located on the directory "*uhd/fpga/usrp2/top/n2x0".*

2. u2plus_core.v- This file instantiates rx_frontend, ddc_chain and vita_rx_chain, which contains all IP cores and its functionalities which perform the actual signal processing flow in the FPGA (refer figure1.4). This file is located on the same directory where u2plus.v file locates.

3. **ddc_chain.v –** This file is instantiated in the u2plus_core.v to perform the Digital Down Conversion by CORDIC, decimation by CIC and two halfband decimators between the ADC/DAC and the host system (refer figure 1.3). This file is located on the directory"*uhd/fpga/usrp2/sdr_lib".* Also this file enable users to create their own custom DSP logic by instantiating the file called dsp_rx_glue.v

4. **dsp_rx_glue.v** – This file enable users to create their own custom DSP logics by making instantiation in the ddc_chain.v file. This file is located on the same directory where ddc_chain.v file locates.

5. **custom_dsp_rx.v** - This is the module where users should instantiate their own DSP logic block box. By updating the variable **Verilog Macros** in the Makefile.custom file we can enable this custom_dsp_rx.v file become active. This file is located on the directory *"uhd/fpga/usrp2/custom".*

Where to insert our custom DSP block box?

Figure 1.6 illustrates the data flow in the FPGA as it is explained in the custom_dsp_rx.v file. By default the entire module is a simple pass through. The user can implement their custom DSP block by customizing this chain through. The user can insert their DSP block in three places based on their implementation requirements. It is achieved by instantiating the custom DSP logic block box into the custom_dsp_rx.v file.

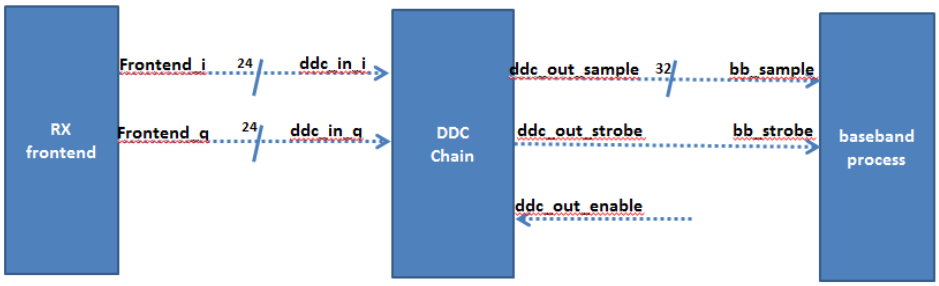

Figure 1.6 Default data flow (simple pass-through)

1. **Implement custom DSP between frontend and DDC input**:

If the user wants their implementation works with the full rate signal they can insert their custom DSP logic block before the DDC.

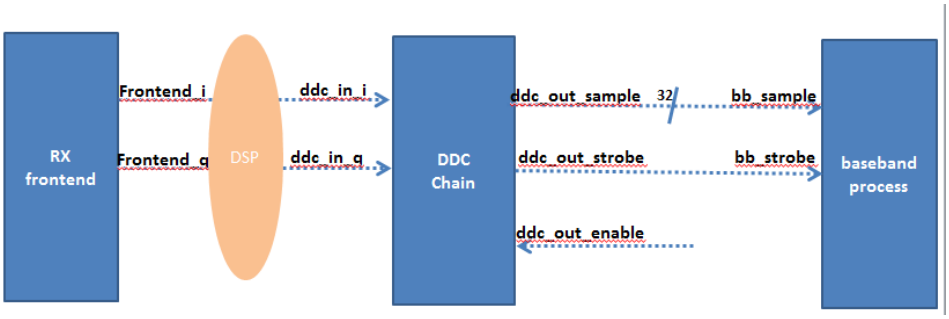

Figure 1.7 Custom DSP block box between frontend and DDC input

2. **Implement custom DSP between ddc output and baseband:**

If the user wants their implementation works with digital down converted signal they can insert their custom DSP logic block before the DDC.

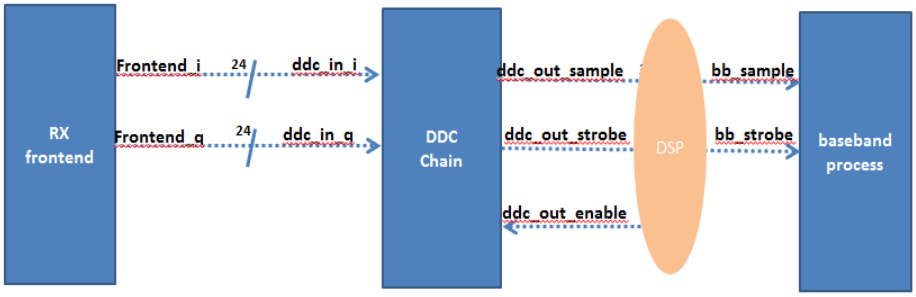

Figure 1.8 Custom DSP block between DDC output and baseband

3. **Implement custom DSP between frontend and baseband:**

If the user wants to bypass the DDC with their custom logic they can replace the DDC by their implementation**.**

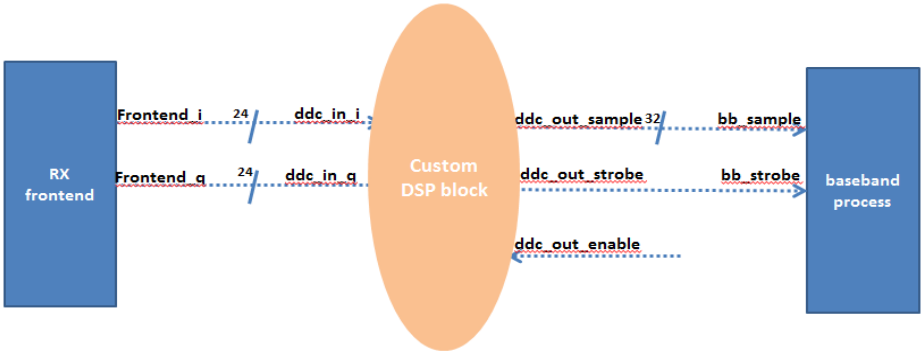

Figure 1.9 Custom DSP block box between frontend and baseband

How to compile the Verilog code and reprogram the FPGA?

1. Add your Verilog source file (.v file) into the **custom** folder which is located in *uhd/fpga/usrp2.* If your Verilog module contains any IP cores then copy the .v and .ngc files of each IP cores as well.

2. Update the Makefile variable **CUSTOM_SRCS** by including the Verilog source files (.v file) and add .xco files if you have used any IP cores in your logic.

3. Instantiate your source file (.v file) into your **custom_dsp_rx.v** file.

4. Before compilation do clear all the files located within the folder called **build_custom** otherwise the compiler will show the error because after compilation the resulted files will be stored in build_custom folder only. The folder is located in the directory *uhd/fpga/usrp2/top/n2x0*

5. After you added all necessary files in the corresponding directory open the terminal and type

cd /opt/Xilinx/13.4/ISE_DS
source ./settings32.sh
cd /home/usrp/uhd/fpga/usrp2/top/N2x0
make –f Makefile.custom 2>&1 | tee logfile

6. Compilation takes 30-50minutes. For compilation it requires paid version of Xilinx ISE (for USRP N210).

7. After compilation the **u2plus.bin** file will be generated in the folder called build_custom. You can also view your timing constraint of your DSP logic in the **logfile** which is located in the directory *uhd/fpga/usrp2/top/N2x0*.

How to burn your customized FPGA image into USRP-FPGA:

1. Connect the USRP device with host system by Gigabit Ethernet cable.

2. Rename the FPGA image file **u2plus.bin** to **USRP_n210_r4_fpga.bin** before you burn this into USRP-FPGA.

2. Run the **usrp_n2xx_net_burner_gui.py** in terminal and select the firmware image **usrp_n210_fw.bin** from the directory *usr/local/share/uhd/images* then select the FPGA image from the directory where you stored the **USRP_n210_r4_fpga.bin** file.

3. Click the **burn Images** button to burn your own FPGA images into the USRP-FPGA. Then do reset the USRP devices to complete the image burning process.

References:

[1]. http://www.ettus.com/kb

[2]. http://www.xilinx.com/support/documentation/data_sheets/ds610.pdf Spartan-3A DSP FPGA Datasheet,

[3]. Jeong-O-Jeong. “Hybrid FPGA and GPP Implementation of IEEE 802.15.4 Physical Layer”. MA thesis. Virginia Polytechnic Institute and State University, 2012.

[4]. usrp-users@lists.ettus.com for suggestions in implementing USRP SDR.