Technical Article

Offloading GNU Radio Processing with FPGA Logic

Nutaq® INNOVATION TODAY FOR TOMORROW®

Offloading GNU Radio Processing with FPGA Logic

Written by Julien Roy, Nutaq FPGA-Embedded Programmer

GNU Radio is an open source software toolkit that accelerates the development of software defined radio (SDR) thanks to the signal processing blocks it provides. GNU Radio digital signal processing blocks run on the computer CPU. When many mathematical operations need to be performed (such as addition, multiplication, filtering, and so on) and when the data rate is high, the computer may be unable to perform all these operations at the speed that's required.

In cases such as these, the developer may seek to offload the CPU processing onto an FPGA, such as the Xilinx Virtex-6 (which is available on Nutaq's PicoSDR solution). While FPGA algorithm development is often slow and painful to debug, developers can choose to work in a model-based environment (see Nutaq's Model-Based Design Kit) which enables faster development while providing easy-to-use simulations by taking advantage of the Simulink® environment from MathWorks®. It also allows designers who aren't familiar with FPGA to quickly understand the signal processing flow that will be implemented inside the FPGA. No VHDL or Verilog knowledge is required.

In this article, we'll implement a digital up-converter (DUC) inside the FPGA. This will decrease both the processing done in GNU Radio as well as the communication bandwidth required to transfer the data between the PC and the FPGA platform.

A Basic Nutaq Model-Based Design Kit (MBDK) Model

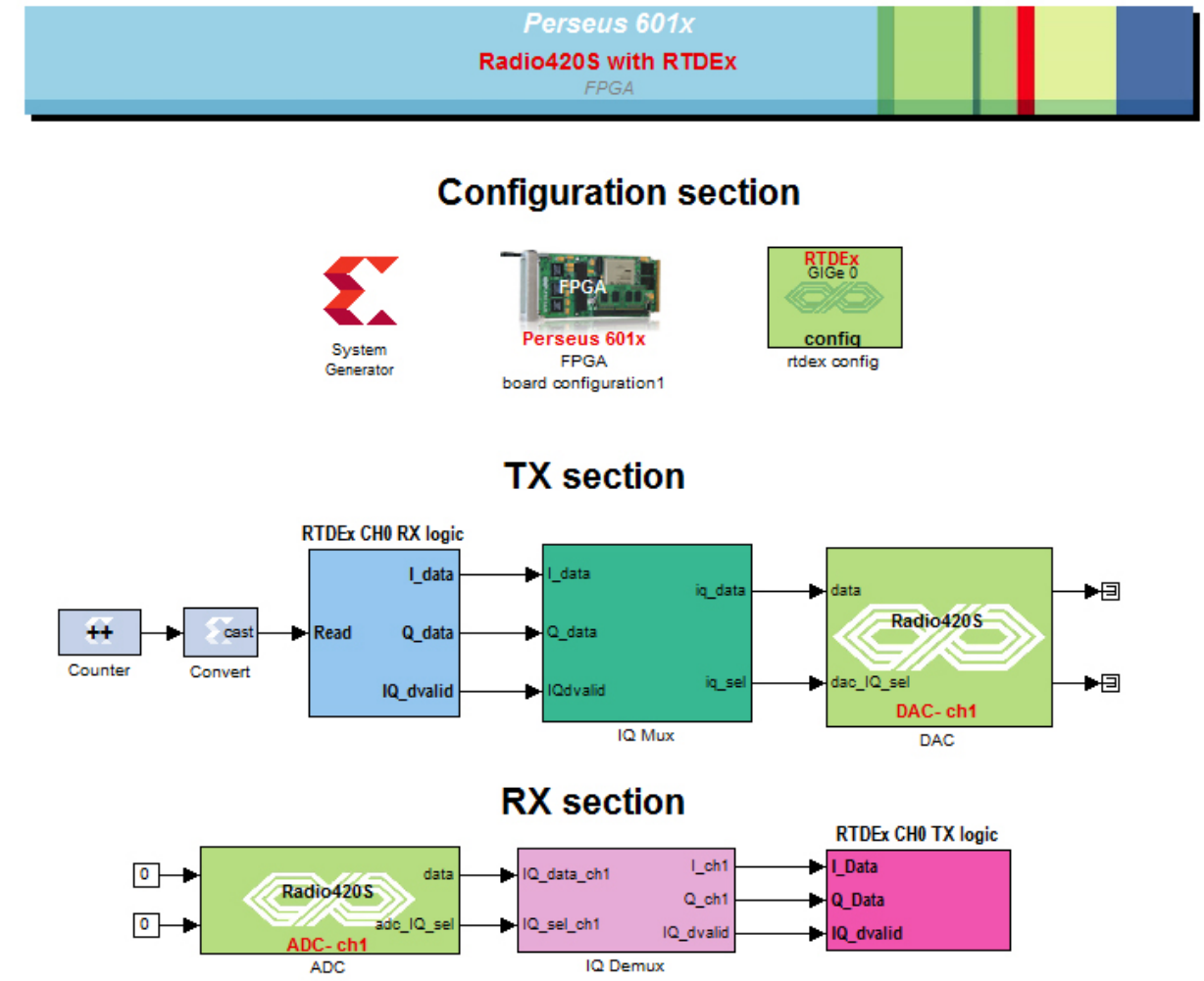

Figure 1: Basic MBDK model

The figure above shows a simple MBDK example using Nutaq's Radio420X and High-Speed Real-Time Data Exchange (RTDEx) IP blocks. This example connects the Radio420X ADC and DAC interfaces to the RTDEx TX and RX channels. Because the Radio420X interfaces use interleaved IQ data, there is either an IQ mux or an IQ demux block between each RTDEx and Radio420X block to convert interleaved IQ data to two independent streams, and vice versa.

A generated bitstream of this model can easily be used in GNU Radio to stream data between the PC and the Perseus 601X card, which houses the Virtex-6 in the PicoSDR system. In this example, we'll modify this simple model to include some real-time signal processing in the FPGA. We'll add a DUC between the RTDEx RX blocks and the Radio420X DAC interface to decrease both the RTDEx data rate required as well as the processing done in GNU Radio.

The Radio420X digital interface can run at up to 80 MHz. At this rate, GNU Radio will need to generate and process 80 megasamples each second and send them through the RTDEx physical layer. Even if your PC can handle this data rate for your given signal processing application in GNU Radio, the physical layer can have a hard time keeping up with this throughput. Assuming 2 bytes per sample, this corresponds to 1.2 Gbps in transmission and in reception. This bit rate exceeds the gigabit Ethernet capacity, and we have only one Radio420X in SISO configuration in our model. In our example, because interpolation will be done inside the FPGA, the data rate will be slower and the Radio420X will be able to run at full speed.

Creating a DUC in the Transmission Path

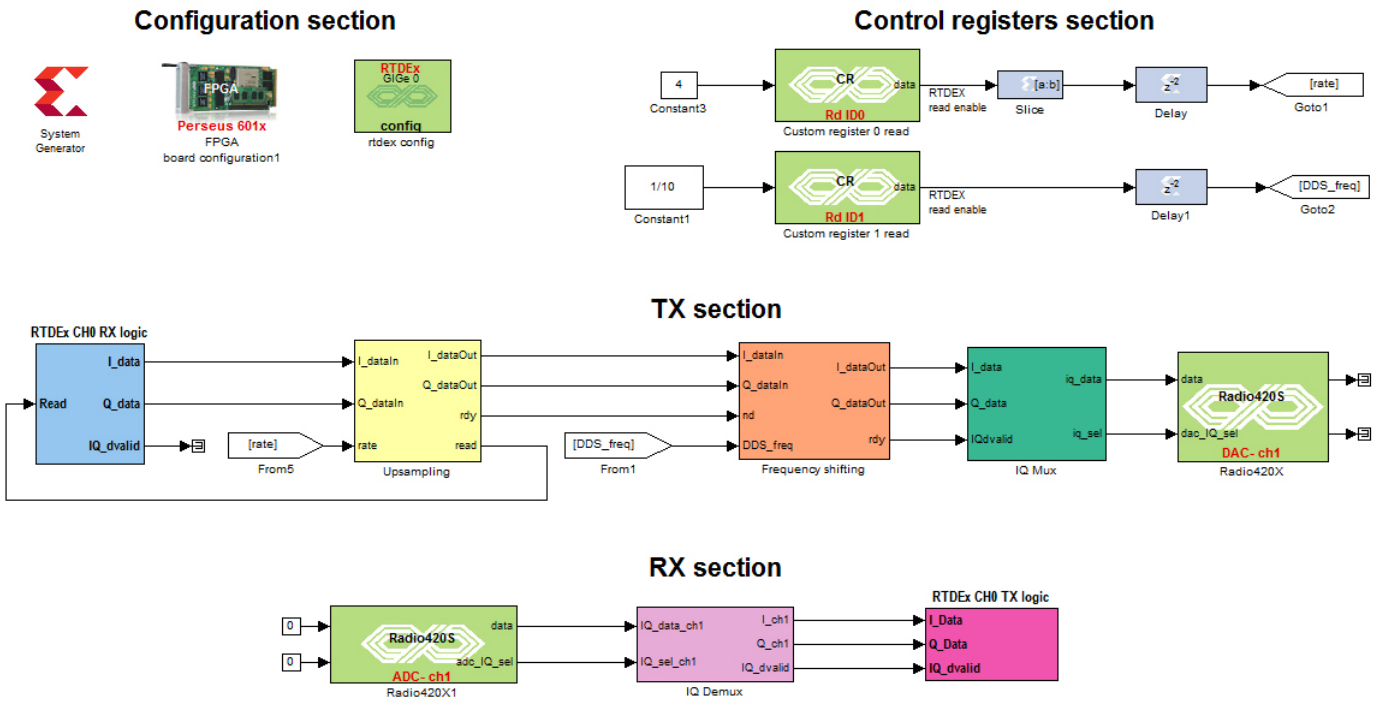

Figure 2: MBDK model including a DUC

The figure above shows the MBDK model including a DUC. Upsampling and frequency shifting blocks have been added between the RTDEx RX blocks and the IQ Mux. These new blocks are configured by custom registers 0 and 1. Custom register 0 drives the upsampling rate, and custom register 1 determines the DDS (direct digital synthesizer) frequency used to shift the upsampled signal.

Upsampling:

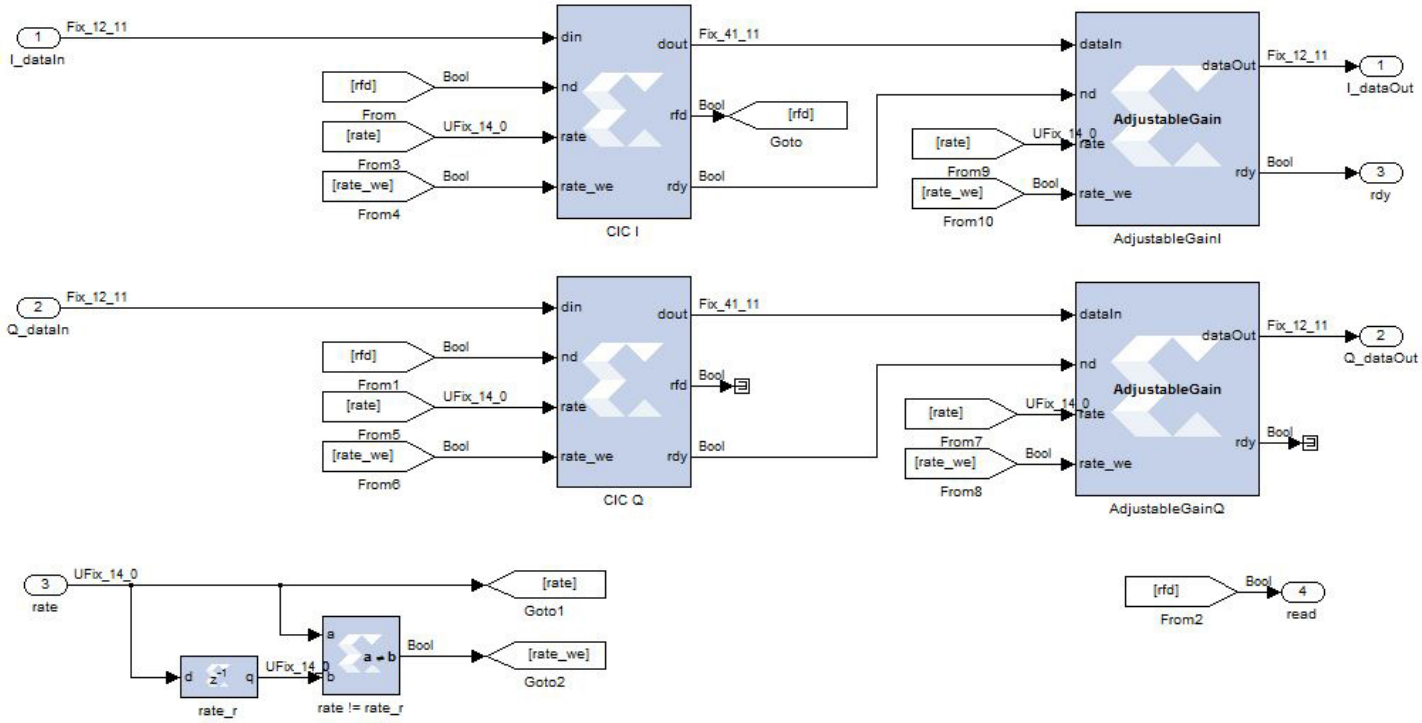

Figure 3 : Upsampling block

The figure above shows the inside of the Upsampling block. The Xilinx® CIC (Cascaded Integrator Comb) Compiler 2.0 is used to implement an interpolation filter with a variable upsampling rate. When the input rate value changes, a pulse on **rate_we** ports is generated. Each time the CIC filters are ready to receive new data, the current input data is sent to CIC filters and new input data is requested with the **read** port. This Upsampling **read** port is connected to the RTDEx RX **read** port. So, each time a new sample is requested, a sample is read from the RTDEx RX FIFO (First In, First Out memory).

Here is the configuration of CIC filters:

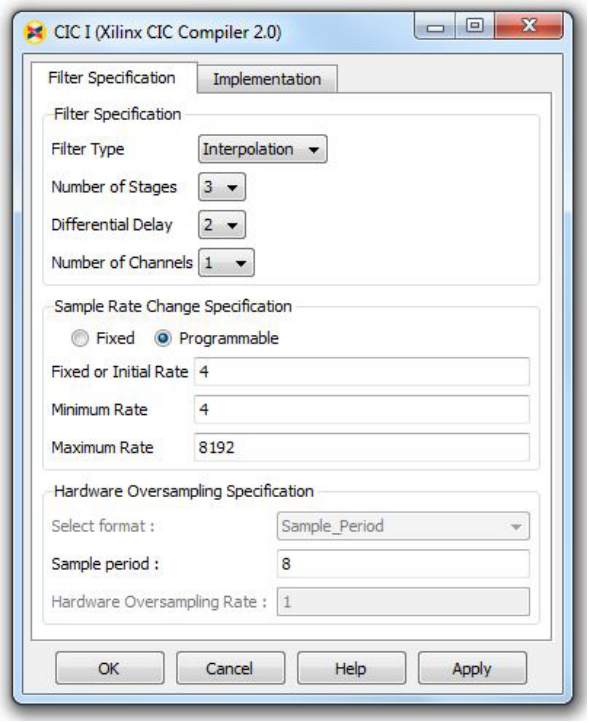

Figure 4: CIC filter configuration

We set the **Number of Stages** to 3 and the **Differential Delay** to 2 to achieve the desired filter response. The **Programmable** interpolation rate is set to handle value between 4 and 8192 and the sample period is set two times higher than the slowest interpolation rate because there will be two CIC filters (one for I data and one for Q data) working at half the sampling frequency.

With this configuration, the outputs of the CIC filters require 41 bits. Depending on the upsampling rate, the gain of the CIC filter can change, and 41 is the number of bits required for the highest rate, 8192.

Since the gain change is dependent on the upsampling rate, we've added an **AdjustableGain** block. Its purpose is to always output the data on 12 bits and with the highest dynamic range. To avoid using a division operation in the FPGA, only bit truncation (same as dividing with powers of two) will be used. Since many cases are possible, we've described the FPGA logic using MATLAB® code for System Generator for DSP™, as shown below.

```
function [dataOut, rdy] = AdjustableGain(dataIn, nd, rate, rate_we)

persistent nd_r1, nd_r1 = xl_state(0,{xlBoolean});
persistent nd_r2, nd_r2 = xl_state(0,{xlBoolean});

rdy = nd_r2;
nd_r2 = nd_r1;
nd_r1 = nd;

persistent dataMux, dataMux = xl_state(0,{xlSigned,13,12});
persistent dataMuxRounded, dataMuxRounded = xl_state(0,{xlSigned,12,11});
persistent rate_r, rate_r = xl_state(4,{xlUnsigned,14,0});

dataOut = dataMuxRounded;

dataMux_LSB = xl_force(xl_concat(xfix({xlUnsigned,12,0},0),xl_slice(dataMux,0,0)),xlSigned,12);
dataMuxRounded = (dataMux + dataMux_LSB);

if rate_r > 5792
    dataMux = xl_force(xl_slice(dataIn,40,28),xlSigned,12);
elseif rate_r > 4096
    dataMux = xl_force(xl_slice(dataIn,39,27),xlSigned,12);
elseif rate_r > 2896
    dataMux = xl_force(xl_slice(dataIn,38,26),xlSigned,12);
```

Figure 5: AdjustableGain code overview

Depending on the upsampling rate (`rate_r`), 13 bits from the input data are chosen (`dataMux`). Then, `dataMux` is rounded based on its least significant bit (LSB) value (`dataMuxRounded`) and assigned to the data output.

It's important to always round the value instead of just truncating the LSB. When truncating, a small offset is created in the output signal which will produce a small tone at the local oscillator frequency.

Frequency shifting:

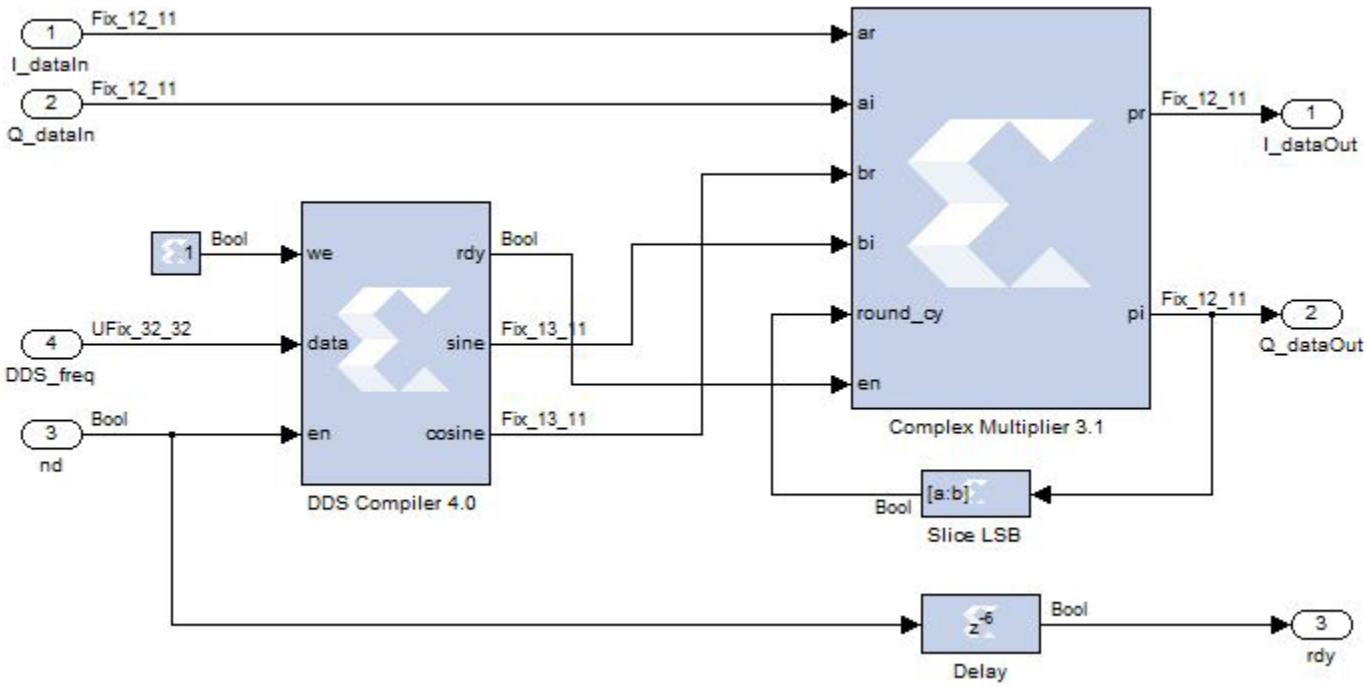

Figure 6: Frequency shifting block

As shown in the diagram above, to shift the frequency of the transmitted signal, we multiply the data by a DDS output signal. At the output of the complex multiplier, the transmitted signal is shifted by the DDS frequency. A 32-bit data port is available to modify the DDS frequency.

For example, to shift the transmitted signal higher in frequency by 1 MHz:

$$\text{data} = \frac{1\,MHz}{Fs} \cdot 2^{32}$$

In our application, when the Radio420X runs at 76.80 MHz, the value of F_S is 38.4 MHz. For these operating values, the data input port of the DDS block must be equal to 111 848 107.

Finally, we configure the complex multiplier to have a 12-bit rounded output in order to have a unit gain for the frequency shifting block.

Running the New Bitstream in GNU Radio

Once the bitstream is generated for this model-based project, it can be easily used in a GNU Radio development environment with the Nutaq GNU Radio420X plug-in.

Here, we'll reuse the GNU Radio project from this blog post. In this new project, we'll use a cosine signal source instead of audio source since it's easier to work with.

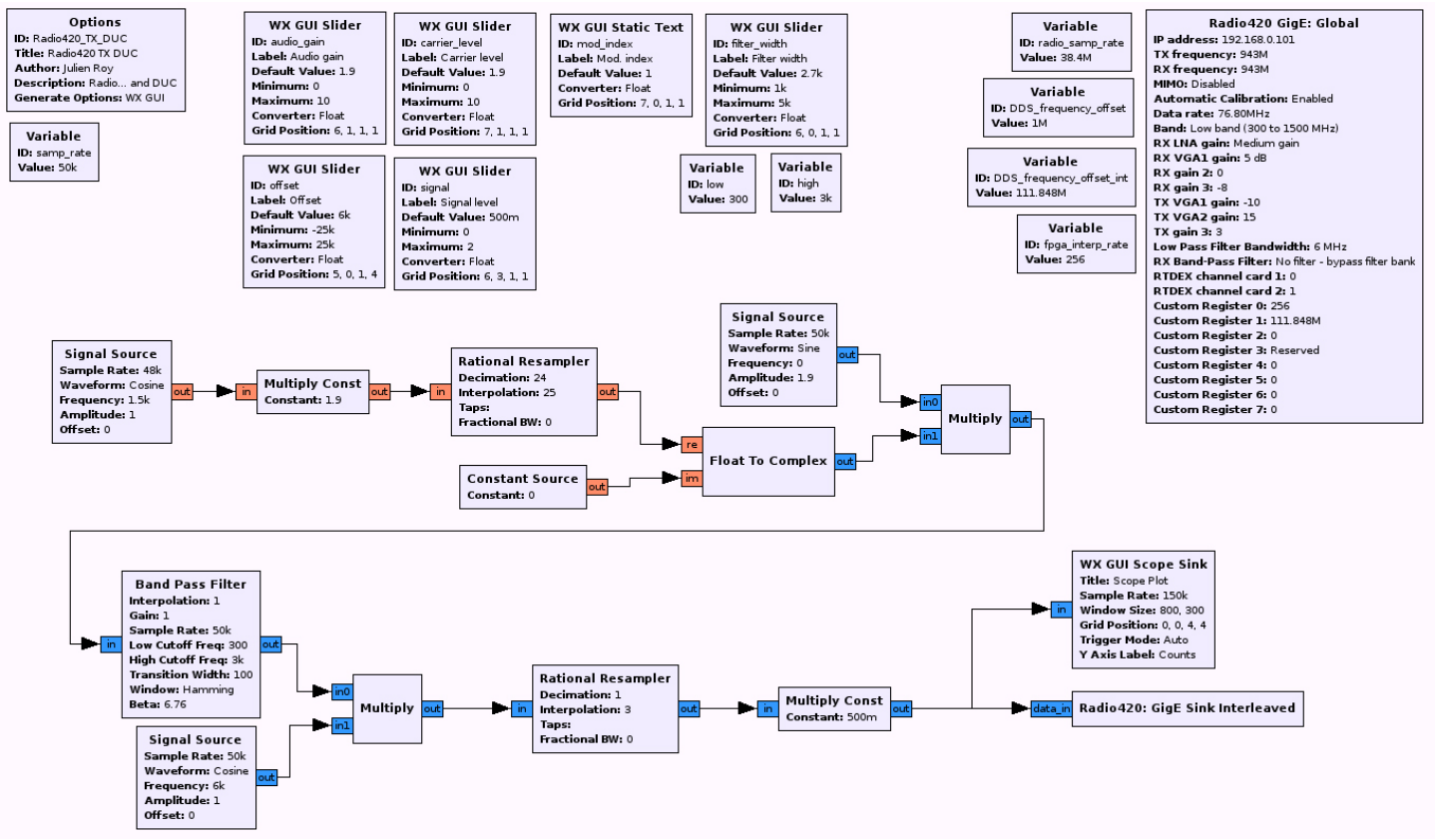

Figure 7: GNU Radio project

In this GNU Radio example, the Radio420X is configured at 76.80 MHz, and the FPGA is configured to have an interpolation rate of 256. We set the **Rational Resampler** to an interpolation rate of 3 to produce an overall sample rate of 38.4 MHz (50 kHz * 3 * 256).

The Radio420 Global configuration is as follows:

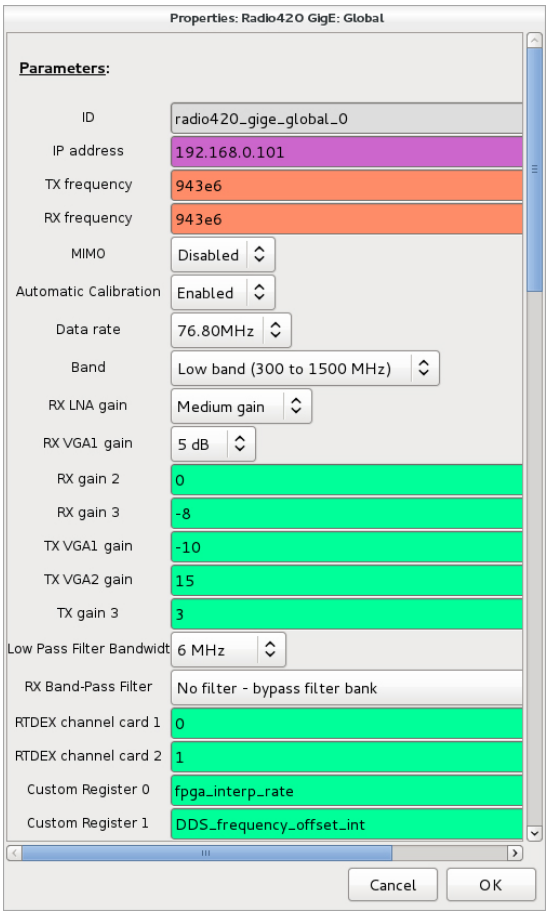

Figure 8: Radio420 Global configuration

We've configured the RTDEx channel for the first radio card (the only one used here) to 0, according to our MBDK project. We link the custom register values to GNU Radio variables by setting Custom Register 0 to the interpolation rate (256), and Custom Register 1 to the DDS frequency offset (111 848 107) to shift the upsampled signal by 1 MHz. In order to run the RTDEx interface at a very slow data rate (~4.8 Mbps), we've changed the RTDEx packet size to 1024 bytes (see the installation instructions for the Nutaq GNU Radio plug-in). If the packet size is too big, the PC interface can send many packets before it knows that the RTDEx FIFO is full, causing the data to overflow.

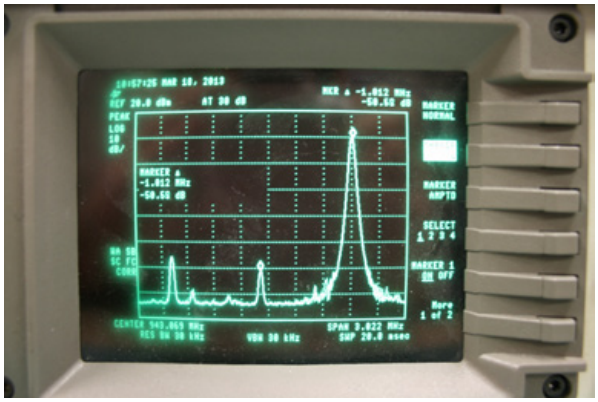

Figure 9: Spectrum of the transmitted signal

We can now launch the demo:

We can see the transmitted signal 1 MHz away from the local oscillator frequency. As expected, the DUC inside the FPGA allows us to fully use the Radio420X output bandwidth, even if the sampling frequency is as low as 150 KHz in GNU Radio. At this interpolation rate (256), only 8% of a 3.4 GHz i7 core is used. In comparison, if the FPGA interpolation rate were set to 4 with a 76.8 MHz data rate, it would require two i7 cores working at 100% to generate the desired data and send it through the RTDEx Ethernet interface. A lot of CPU processing power has been saved by transferring some of the processing to the FPGA.

FPGA Saves CPU Cycles

The Nutaq MBDK makes it easy to develop real-time signal processing in the FPGA, which in turn can significantly decrease the CPU processing needed to run a given GNU Radio application. In this article, we developed a DUC, but almost any signal processing blocks can be migrated from the GNU Radio environment to the Nutaq MBDK. We did all of this without writing any VHDL or Verilog code.

The powerful combination of the processing available in the FPGA fabric coupled with the GNU Radio rapid development environment opens up a wide spectrum of potential applications to the software defined radio developer.

Available files for download:

- MBDK project connecting Radio420S with RTDEx
- MBDK project with the DUC added
- MBDK AdjustableGain function
- MBDK project used to simulate the behavior of the developed DUC
- GNU Radio companion project using the DUC functionalities

Requirements for MBDK related files:

- Windows 7 64-bit
- Matlab R2011B
- ISE Design Suite [System Edition] 13.4
- System Generator for DSP 13.4
- Nutaq ADP Software Tools - µTCA Edition 6.1.0 with MBDK license

Requirements for GNU Radio Companion project file:

- Linux Fedora 17
- GNU Radio v3.6.3
- Nutaq ADP Software Tools - µTCA Edition 6.1.0
- Nutaq GNU Radio plug-in for ADP 6.1

Nutaq products are constantly being improved; therefore, Nutaq reserves the right to modify the information herein at any time and without notice.

Nutaq®

INNOVATION TODAY
FOR TOMORROW®

2150 Cyrille-Duquet, Quebec City (Quebec) G1N 2G3 CANADA
T. 418-914-7484 | 1-855-914-7484 | F. 418-914-9477
info@nutaq.com