Designing and Verifying a Digital Filter Using MATLAB With Eclypse Z7 as FPGA-in-the-Loop

by Pablo Trujillo

Introduction

Digital Filters running on embedded hardware or FPGAs will most likely use fixed-point arithmetic. The effects of quantization need to be checked during development, preferably with hardware-in-the-loop. By using MATLAB and an Eclypse Z7 FPGA board with Zmods for data conversion, that's easy.

When hearing the phrase "software for engineering", most of us engineers immediately think of MATLAB. There are few areas in engineering for which MATLAB does not provide a package, and digital signal processing and FPGA design are definitely not among them. **MATLAB's Signal Processing Toolbox** gives developers all the features they need to create a filter or any signal processing system. The **Fixed-Point Designer** lets them quantize their data and helps them to implement fixed-point and floating-point algorithms in their processing system and to check their effects on the digital system. And packages like **HDL Coder**, **HDL Verifier**, and **FPGA Data Capture** allow them to test their design on a real FPGA as hardware-in-the-loop and to transfer the resulting system to their FPGA board. This way, creating a signal processing system becomes less complex.

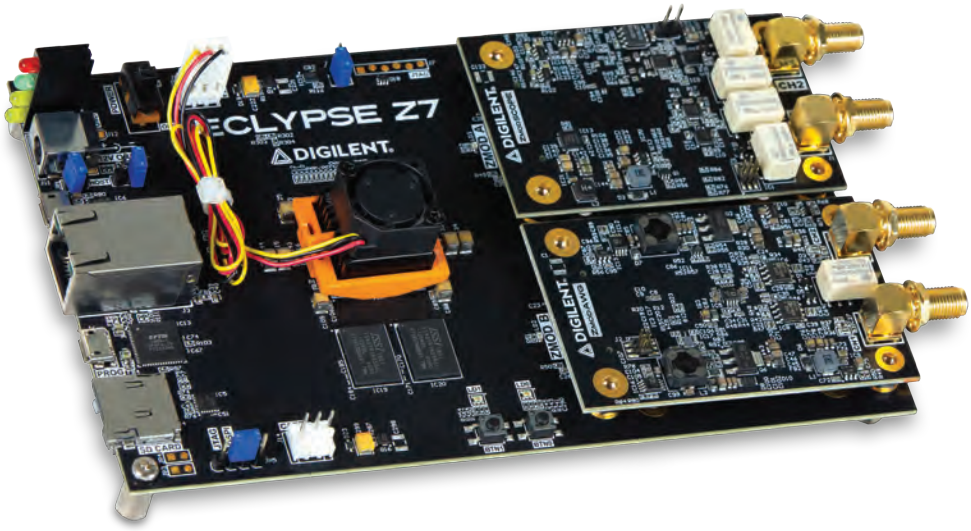

Figure 1: The Eclypse Z7 Zynq-7000 FPGA board is used for the hardware-in-the-loop

Suppose we want to design a bandpass filter for a communications system to extract a certain signal from a line that employs frequency division multiplexing. This technique is used for some time, for example, to transmit different radio stations over the same data line, the air. Another case is Power Line Communication (PLC), where we can transfer

power at grid frequencies of 50Hz or 60Hz, and data at frequencies of several hundred kilohertz.

And once the filter is designed, we want to run it on a real hardware. This hardware system could comprise, for example, an Eclypse Z7 Zynq-7000 SoC (System on Chip) FPGA board from Digilent with one Zmod Scope 1410 2-channel 14-bit oscilloscope module and one Zmod AWG 1411 2-channel 14-bit arbitrary waveform generator module – both from Digilent – mounted on top. With the help of the analog-to-digital converter (ADC) of the Zmod Scope 1410 we can digitize the line signal and we can output both, the sampled and the filtered signal, through the digital-to-analog converter (DAC) on the Zmod AWG, while running the filter itself on the Zynq-7000.

Designing the Filter and Taking Care of Quantization

Figure 2 shows the frequency spectrum of our transmission line. Each channel has a bandwidth of 20kHz, and the channels are spaced 100kHz apart. Our channel of interest is channel 1, and we will design our bandpass filter to avoid interferences from the other channels. To do this, we need to ensure an attenuation of at least –60dB at the bandwidth of the other channels. To get this characteristic, our filter should have a passband between 990kHz and 1010kHz, a lower frequency stopband ranging from 0Hz to 910kHz, and a higher frequency stopband starting at 1090kHz.

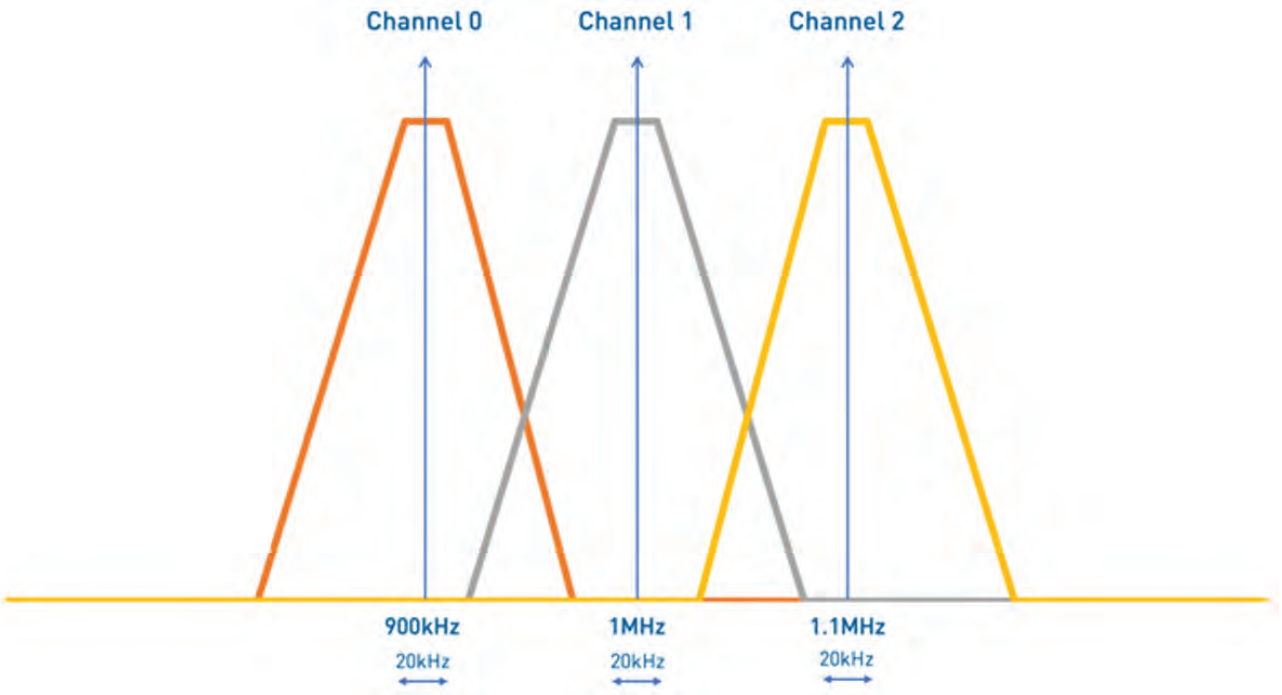

Figure 2: The frequency spectrum of the example transmission line. The signal of interest is channel 1.

With the help of the Filter Designer tool from MATLAB we can configure all these aspects of our digital filter. For our example, we use an IIR (Infinite Impulse Response) Butterworth filter, because it allows us to design a low-order filter with a high attenuation and it has a fairly flat passband. The sampling frequency will be chosen to satisfy both the hardware requirements of our data converters – 105MSPS for the AD9648-105 ADC from Analog Devices on the Zmod Scope and 100MSPS for the AD9717 DAC, also from Analog Devices – and the Nyquist sampling theorem. For our example, 10MSPS are sufficient.

In addition, as our data converters are 14-bit fixed-point devices, we need to quantize the data of the filter through the Filter Quantization panel of the Filter Designer Tool to match their word lengths, but also in a way that no loss of resolution will occur. Once we entered all parameters in the Filter Designer and generated the filter, the Filter Designer will show us the DFT (Discrete Fourier Transformation) of the filter's response. Now, we can export it to a Simulink model, where we can run a first test and verify whether the quantized model meets the defined criteria. As simulation input, we generate in MATLAB a sum signal from the three desired frequencies.

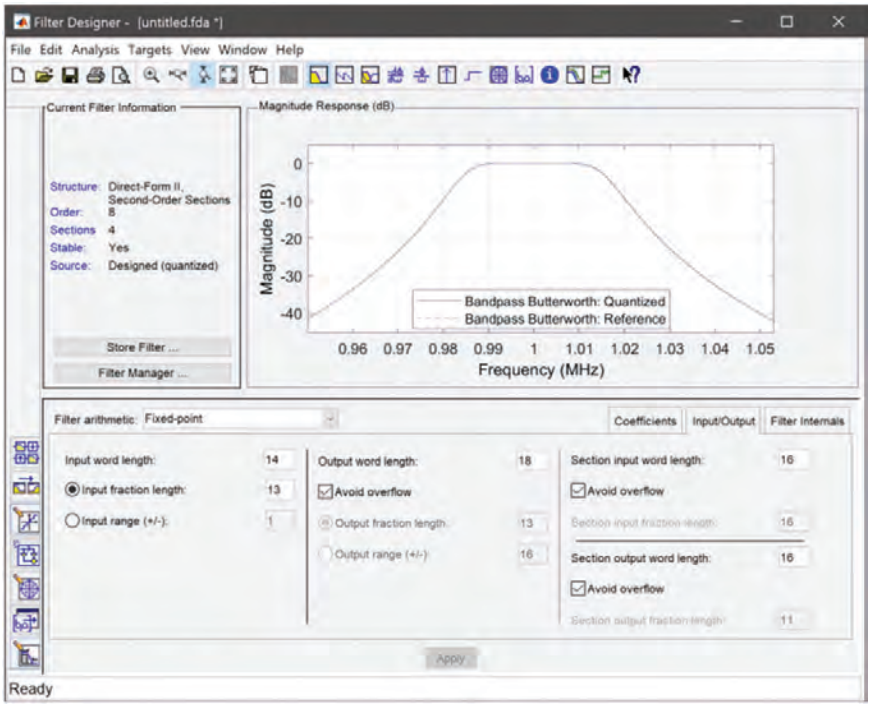

Figure 3: Response of the quantized filter

Bringing the Hardware into the Loop

If the filter behaves to our satisfaction, we can generate the HDL (Hardware Description Language) code with the help of the HDL Coder to test the filter on the Eclypse Z7 FPGA board, after making a few modifications to the Simulink model so that we can generate the HDL code for the whole system.

The HDL Block Properties window allows us to configure the pipeline registers and the architecture in terms of multipliers used. We have to choose one of three different architecture models: a fully parallel, a fully serial, and a partially serial architecture.

With the fully parallel architecture, the output data frequency can be the same as the clock frequency, and each multiplication of the filter will use one DSP slice of the FPGA. The fully serial implementation, however, uses only one DSP slice, but the output frequency will be reduced to (clock speed) / (number of multiplications), slowing down the computation. The partially serial architecture finally is in the middle, with the number of slices selectable. This has the disadvantage that the number of used logic elements increases. As we have enough DSP Slices on the Zynq-7000 SoC on the Eclypse Z7, fully parallel is a good choice. As a last step here, we select the target HDL language, the device used, the reset, and the reports wished on the HDL Coder Properties page. With all the configurations done, we can generate the HDL code for our subsystem.

Now we can simulate the filter with the FPGA in the loop. This means that the filter algorithm will execute directly on the FPGA and not in software on the host workstation. This is where MATLAB's FIL (FPGA-in-the-Loop) tool comes in. It allows us, after creating the board definition, to connect to the board.

Now we can build the project and Vivado, Xilinx' HDL synthesis and analysis tool, will open in the MATLAB command window and synthesize and implement the FIL demo design. Once Vivado finishes the synthesis, a Simulink model will be opened, showing the added FIL block. Now, we can connect the Eclypse Z7 board and load the FIL model into the FPGA.

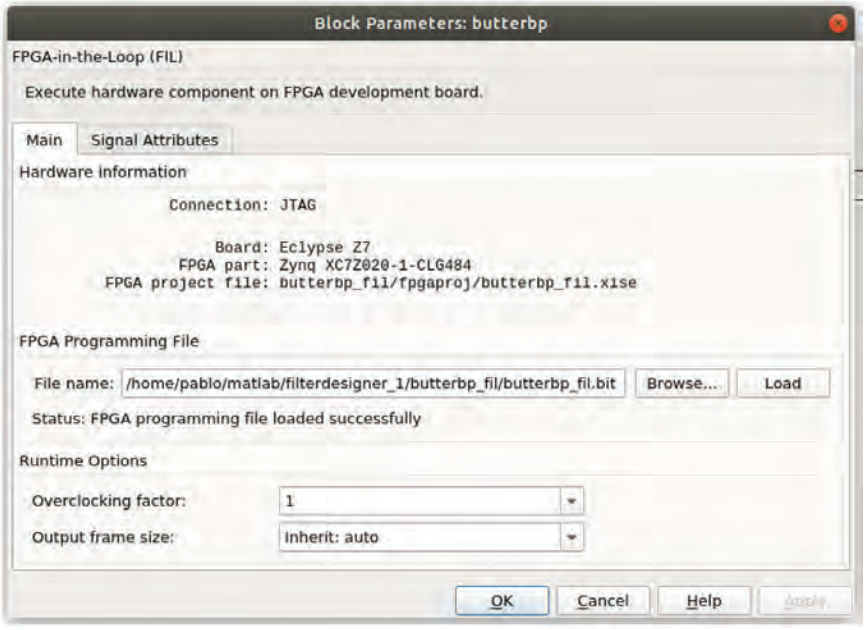

Figure 4: The Block Parameter screen of the FPGA-in-the-Loop tool

To simulate the signal source, three sine-wave signals representing the three bands in the transmission line need to be added, with frequencies of 900kHz, 1MHz, and 1.1MHz. To verify the output, we need a Spectrum Analyzer window. Once the simulation has finished, the output only contains the 1MHz signal.

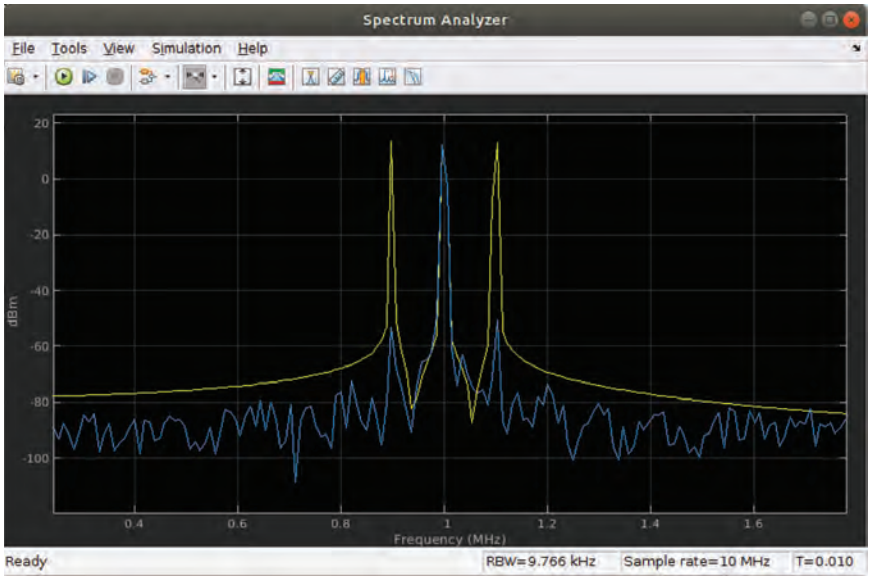

Figure 5: The Spectrum Analyzer window shows that only the signal on channel 1 is left after filtering

Applying Data Capture for Verification

With the filter tested and verified with the FPGA-in-the-loop, we can create the final application, which uses not only the filter but also the ADC and DAC boards. For this, we create a Vivado project where we will add all the blocks necessary, like the Zynq-7000, the drivers for the data converters, and the clock generator.

On the Zynq-7000 SoC, we won't use the Processing System, only the Programmable Logic (PL). As clock source, we will use the 125MHz clock from the Ethernet PHY of the Eclypse Z7 board, which is connected to the PL. An additional clock enable generator to create the filter sampling clock of 10MHz and the system clock of 100MHz is also necessary.

For the ADC board driver, we will use the AXI (Advanced eXtensible Interface)-Stream Interface IP. When adding the filter, we have to take care of the data sizes, because the output of the AXI-Stream IP is 32 bits wide and contains the data of both ADC channels. The data word corresponding to channel 1 is on the lower 14 bits of the word, so it needs to be extracted.

Similar is true for the DAC driver, where the upper output bits of the filter have to be discarded, using a Slice block. Our example project uses both DAC channels: the filter output is connected to channel 1 and the original signal, as sampled by the ADC, is connected to channel 2 for verification through an oscilloscope.

Once the block design is complete, we can validate it, create the HDL wrapper, and generate the bit stream. Once the design is synthesized and implemented, we can test the design on the Eclypse Z7 board, together with a signal generator and an oscilloscope. Using a 1MHz sine wave, the passband can be checked, and with a 910kHz signal, the stopband. This verification indicates that everything works as expected.

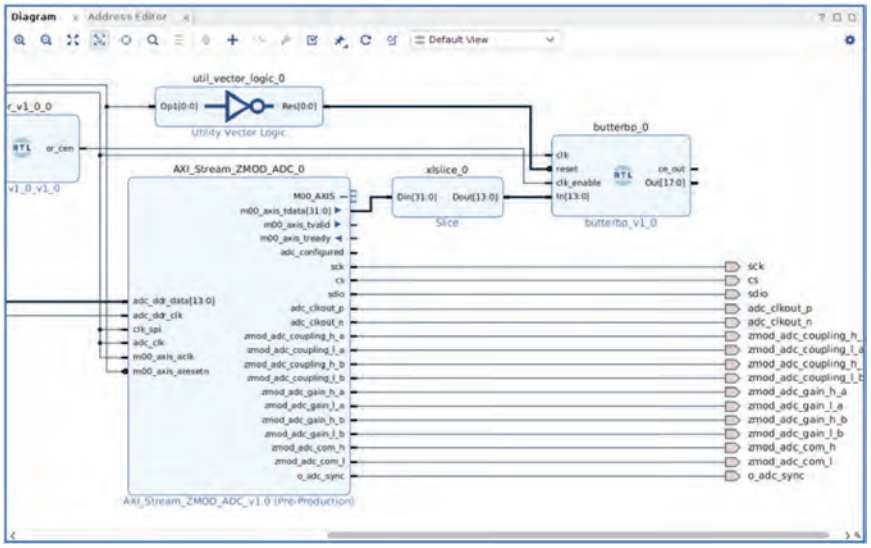

Figure 6: The design block showing the AXI-Stream IP, the data word conversion slice and the bandpass filter

Another possibility is to use a logic analyzer, either an external device or the Integrated Logic Analyzer (ILA) from Xilinx, which can be used to monitor the input and output data inside the FPGA and export it to MATLAB. A third possibility is to use the FPGA Data Capture feature of MATLAB, which auto-generates an IP block that captures live data from the FPGA and sends it to MATLAB or Simulink via JTAG. FPGA Data Capture is part of the HDL Verifier Support Package for Xilinx FPGA boards.

In the window for the FPGA data capture component generation, the filter input and output ports can be added, and the buffer size and maximum sequence depth set. Once done, the tool will generate the HDL IP core, as well as a script to start the data capturing from the MATLAB workspace, and a Simulink model to capture data from Simulink. The last step is to add all files to the project and the data capture module to the block design. Then, we can generate the bit stream.

To test the system, we have to close the hardware server from Vivado to release the JTAG port. Some adjustments to the Simulink data acquisition model, such as removing the

oscilloscope and replacing it with the spectrum analyzer, and configuring data type and format, are required as well.

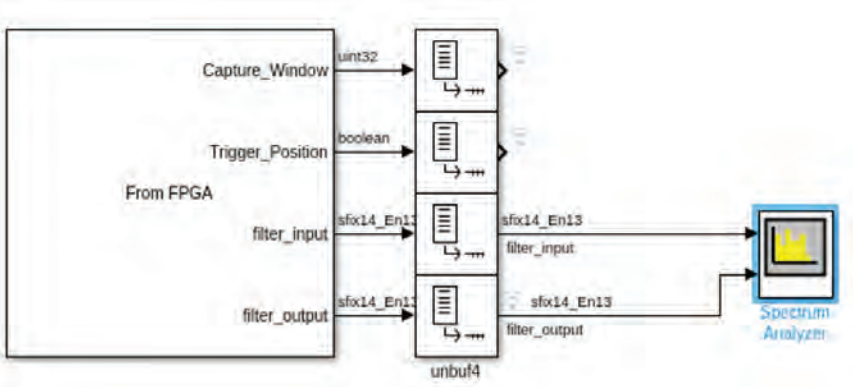

Figure 7: Block diagram for the real-time data capture on the Eclypse Z7 board

With white noise generated by an external signal generator, we see that the filter works as expected in the frequency range from 990kHz to 1010kHz without attenuation. However, there is a small catch: the update rate of the analyzer is 1Hz instead of the 10MHz of the real design.

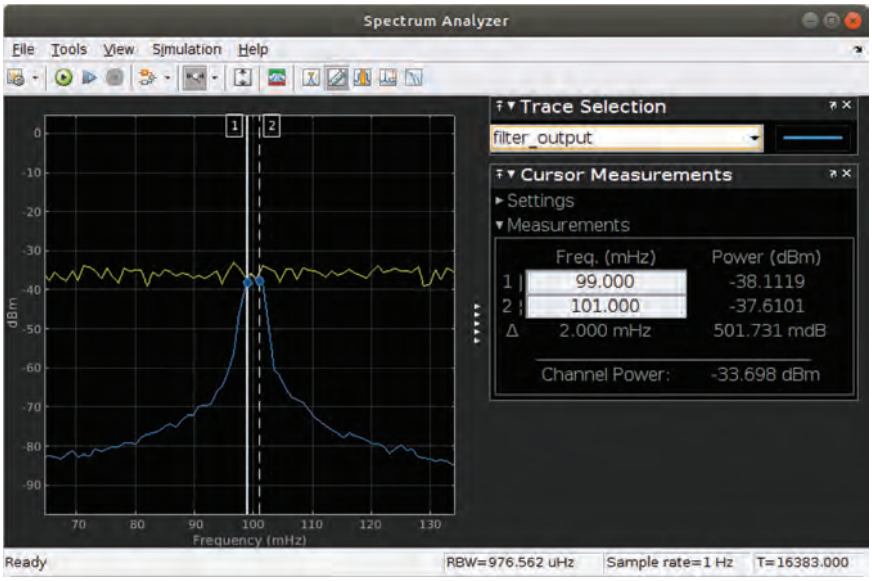

Figure 8: Using a white noise signal as input, the filter lets pass only frequencies from 990kHz to 1010kHz

This project shows that it is possible to easily design a digital filter, check it step by step, and finally verify it with an Eclypse Z7 Zinq-7000 board as hardware-in-the-loop, by using the different packages of MATLAB and Vivado. A full step-by-step tutorial for this project is available here:

https://www.controlpaths.com/2021/11/15/designing-a-filter-on-matlab-and-verifying-it-using-fpga-in-the-loop-and-eclypse-z7/

Pablo Trujillo is an FPGA developer for power electronics devices and a specialist in DSP and power electronics control design.