DIGIMOD: A TOOL TO IMPLEMENT FPGA-BASED DIGITAL FRONT-END FOR SOFTWARE RADIOS

José Marín-Roig, Vicente Torres, Mª José Canet, Asunción Pérez, Trinidad Sansaloni, Fabian Angarita, Javier Valls (Dpto. Electrónica, Universitat Politècnica de València, Gandia, Spain, {jomara, vtorres, macasu, asperez, tmsansal, faanpre, jvalls}@eln.upv.es), Francisco Cardells (Inkjet Commercial Division (ICD) R&D Lab, Hewlett-Packard, Barcelona, Spain, francisco_cardells@spa.exch.hp.com), Felip Vicedo (Dpto. Física y Arquitectura Computadores, Universitat Miguel Hernández, Elche, Spain, felip@umh.es), Vicenç Almenar (Dpto. Comunicaciones, Universitat Politècnica de València, Gandia, Spain, valmenar@dcom.upv.es)

ABSTRACT

DIGIMOD is a software tool that speeds up and makes easy the implementation of wireless communications systems. The tool allows the software radio designer to develop digital upconverters and downconverters and, finally, to generate automatically the VHDL code to implement the system on Xilinx FPGAs. The main characteristic of this tool is that the user can adjust the parameters of an *ad-hoc* interpolation or decimation filter chain composed by CIC, polyphase, pulse shaping, matched filters and a CORDIC-based or ROM-based mixer. By means of its graphical interface the user can choose all the subsystems required, and once the transmitter/receiver is completed, this tool can evaluate the performance of those IP-cores employed. Finally, if the specifications are met, the tool generates the VHDL code for the whole system.

1. INTRODUCTION

In the near future it is expected that new communication systems will provide higher mobility and wider bandwidth than present systems. Then, those firms that develop future communication systems will have to face up to next challenges: continuous evolving standards, fast deployment of new services, higher spectral efficiency, high capacity and mobility data transmissions, and a demand of a high reliability in those services offered.

To meet all theses objectives it will be necessary the use of new methods of design and development. In recent years a new technology called software defined radio (SDR) has come up. The main idea behind SDR is to move the digital part of any communication system towards the antenna. This means that most of the analogue components from transmitters and receivers have been substituted by digital signal processing under FPGA or DSP devices. Working in this way it is possible to change the system configuration without changing the hardware.

This paper presents a tool that speeds up and makes easy the implementation of wireless communications systems. With this tool one can design using SWR a digital transmitter/receiver in intermediate frequency or in base band. By means of its graphical interface the user can choose all the subsystems required, and once the transmitter/receiver is completed, this tool can evaluate the performance of those employed IP-cores. Finally, if the specifications are met, the tool generates the VHDL code for the whole system.

2. DESCRIPTION OF DIGIMOD TOOL

DIGIMOD is a graphical tool that makes easy and fast the design and implementation of a transceiver for digital communications on FPGA. The transmitter, also called upconverter, is composed of several stages: binary data source, symbol mapping, pulse shaping, interpolation filters and mixer. Meanwhile, the receiver (or downconverter) is composed by a chain of elements that performs the reverse operations: a mixer to bring the signal to baseband, decimation filters to cancel the double frequency image and to reduce the sampling rate, a matched filter adapted to the pulse shape, a demapper and a symbol detector.

So, the designer of a digital communication modem can use this tool to:

- select the type of modulation between BPSK, QPSK, and QAM, and the pulse shape parameters;
- evaluate what kind of filters are needed in the interpolation or decimation stages, CIC or polyphase filters can be used;
- design the selected filters by using the toolboxes from MATLAB and importing the final coefficients;

- carry out a floating point simulation where the system performance is evaluated through two kinds of results: the Error Vector Measurement (EVM) of the generated signal, and the bit-error rate (BER) in a standard AWGN channel;
- evaluate the finite precision for each block by comparing the floating point design with the same design using fixed point operators until the implementation loss is minimized;
- generate a VHDL code of the fixed point modelled system in which each block is an area optimized relatively placed macro (RPM).

2.1. Design Flow

There are several steps during the design process with DIGIMOD. The design flow is shown in figure 1. The first step is the selection of those blocks needed in the design: source, modulation mapping, pulse shaping, interpolation filters, and/or mixer. Then the user must configure their parameters.

Once the filter chain is specified, a simulation using floating point precision is performed to evaluate if the specifications are fulfilled. The tool generates the frequency response of the filter chain, as well as of individual filters, and two quality measurements: BER and EVM. If the obtained results do not match with system specifications the user can aggregate or delete blocks, or adjust block parameters until simulation gives the correct results.

After tuning the floating point design, the user can begin with the fixed point evaluation, this part is performed iteratively: in each step the number of bits (filter coefficients and output signal) of a new block from the chain is adjusted, this process is performed until all the blocks have been analyzed. Simulations are performed in each step in order to asses the user to choose the number of bits needed.

If finite precision simulation accomplishes those required specifications, the tool is ready to generate the VHDL code of the system. After target FPGA device selection, DIGIMOD generates the VHDL code, and it is synthesized with Synplify and implemented with ISE Xilinx tool.

2.2. DIGIMOD Blocks

When the tool is started it shows a screen with the main blocks of a generic digital transmitter or receiver; figure 2 shows this screen for a transmitter design. At this point, the user can choose which components will be used in the final design: it should be noted that it is not necessary to include all the blocks showed in the chain, or that more than one interpolation filter block can be added. Those blocks selected and their parameters appear in a list next to the chain flow diagram (this list is showed empty in figure 2), the order of appearance in the list determines the up/down converter structure since this is the order of connection in the system. Next to the list there is a group of buttons that allows moving the position of an item in the list (Up and Down), editing its parameters or adjusting the number of bits of each stage (Edit). Following sections describe the different blocks available in DIGIMOD tool.

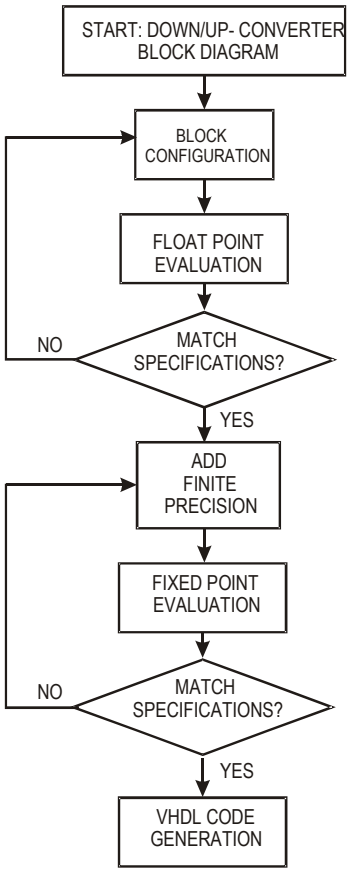

Fig. 1. Design flow diagram of DIGIMOD

Source

This block allows selecting the source of data and the bit rate for simulation. Three options can be selected: *Random*, *File* and *None*. *Random* option generates automatically a random sequence of bits (user is allowed to choose the number of bits transmitted). *File* option reads data stored in a file or a MATLAB workspace variable. One of these two options are required if BER or EVM measurements are needed. When option *None* is selected only the floating point frequency response is available as simulation result, this option obtains a quick result and it is useful during the first steps of the chain design.

The selected option is showed in a box over the chain list (figure 2). This item does not appear in the list because *source* must be always the first stage of the system chain.

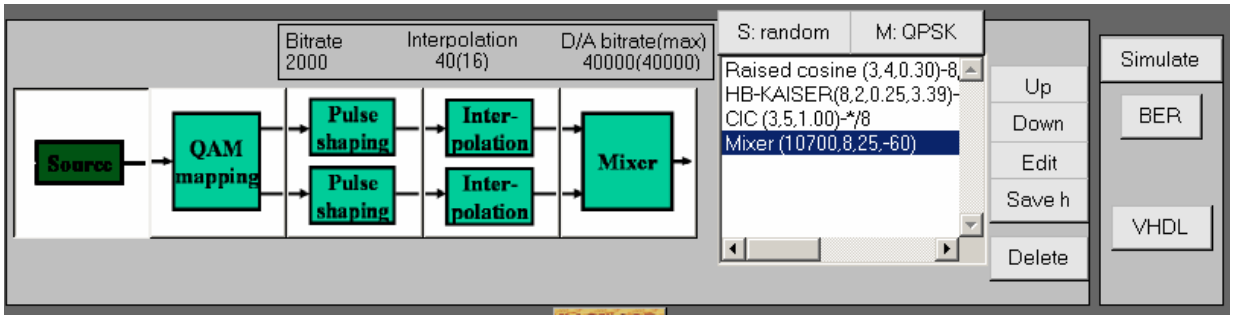

Fig. 2. First screen of DIGIMOD for a transmitter design

QAM mapping

This block allows choosing the modulation format between BPSK, QPSK, and QAM. The mapping is performed using Gray encoding. The selected modulation mapping also appears over the chain list, beside the source box (figure 2).

Pulse shaping and matched filter

This block includes all the parameters needed in the definition of a root rise cosine filter: the delay (in number of symbols), the interpolation factor (in number of samples per symbol), and the roll-off factor (a number between 0 and 1). Figure 3 shows the dialog box for this block when a transmitter is designed.

The user can select between two implementation methods: polyphase filter (by default option) or look-up table [1]. Look-up method makes use of the embedded block select RAM available in a FPGA device. This method allows a higher interpolation factor than polyphase filter at a lower cost and simplifies the interpolation filters that come later in the transmitter chain. In a fixed point implementation of a look-up filter, the only parameter needed is the number of bits at the output.

Polyphase filters are implemented with bit-serial or digit-serial distributed arithmetic [2]. In this case, both the number of bits for the coefficients and the number of bits at the output are needed to implement this filter in fixed point precision.

In a receiver, the matched filter can only be implemented using a polyphase structure.

Multirate filters

This block allows us to select between two kinds of filters: CIC filter or FIR filter. When a CIC filter is chosen, the user must specify these parameters: filter order, interpolation factor and the differential delay [3]. Its fixed point implementation only needs the number of bits at the output.

FIR filters are also called half band filters when the interpolation/decimation factor is 2 or more generally 1/*n*-band filters, where *n* is the change rate factor. These filters are implemented using a polyphase structure and distributed arithmetic. When fixed point design is selected the user must introduce two parameters: number of bits for coefficient quantification and number of bits at the output. The coefficients of the FIR filter can be specified by one of the following ways:

- from a previously designed filter whose coefficients have been saved in a ".mat" file or in a workspace variable;
- from a design using MATLAB SPTOOL, DIGIMOD allows to import the coefficients;
- from a dialog box where three types of filters can be specified: Kaiser window, equiripple, and least squares.

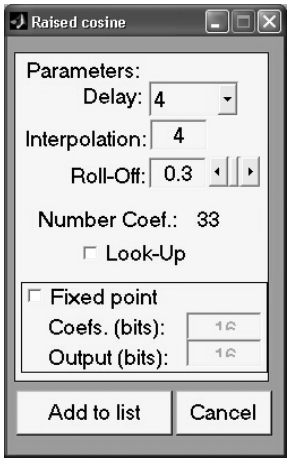

Fig. 3. Pulse shaping dialog.

Mixer

This block only requires the intermediate frequency if a floating point simulation is selected. In order to simulate the precision finite behaviour the output resolution and the phase and frequency resolution must be indicated.

Two methods can be chosen to implement this block. The first performs a CORDIC-based mixer [4] and the second a ROM-based one [5,6]. This last method uses compression techniques to reduce the size of the required memories.

2.3. Implementation Technology

The VHDL code generated by DIGIMOD instantiates all the necessary cores to perform the required operations. All

the cores have been described in VHDL by using relative placed attributes. Furthermore, they are area efficient with respect to same blocks generated with Xilinx Coregenerator system [7].

3. A DESIGN EXAMPLE

For a better understanding of DIGIMOD characteristics and performance, in this section we will present a design example of a digital IF QPSK modulator using DIGIMOD tool. In order to benchmark the performance of our tool we will design the same modulator using Xilinx System Generator [8], and we will compare the results obtained by both tools. The QPSK modulator parameters are:

- Data rate: 2 Mbps
- Pulse shape: root raised cosine with a roll-off of 0.3, and a duration of 8 symbols (4 symbols delay)
- Intermediate frequency: 10.7 MHz
- Output sampling rate: 40 MHz
- DAC resolution: 8 bits

We have implemented this modulator using these block settings in DIGIMOD:

- *Source*: random, 2 Mbps.
- *QAM mapping*: QPSK.
- *Pulse shaping*: delay 4, roll-off 0.3, interpolation 4, (polyphase structure).
- *FIR interpolation filter*: interpolation factor 2, Kaiser design (MATLAB parameters: order 6, cut-off frequency 0.5, beta 3.39).
- *CIC interpolation filter*: interpolation factor 5, filter order 3, differential delay 1.

After some simulations we have checked that good fixed point parameters are: 8 bits for coefficient quantization in both filters: pulse shaping and half band interpolation and 8 bits for signal output for all the blocks in the system: previous filters, CIC filter and mixer. As results of these simulations we show in figure 4 the comparison in BER between floating point design and 8 bits fixed point design (where implementation loss is less than 0.1 dB), and in figure 5 the signal spectrum at CIC filter output in both the passband (above) and the stopband, it is clear that fixed point spectrum is very close to float point spectrum.

3.1. Implementation Results

Table 1 shows the results of three different implementations of the QPSK modulator in a Xilinx VirtexE-8 device. The first one has been performed with Xilinx System generator, the second and third ones use DIGIMOD tool with the pulse shaping filter performed with a polyphase filter or with the look-up table method, respectively.

It can be seen in Table 1 that DIGIMOD implementation is more efficient than System Generator one. It is achieved an area saving of 45% with the polyphase filter option. If the look-up table method is used to implement the pulse shaping, two Block Select RAMs and 27 slices are only required.

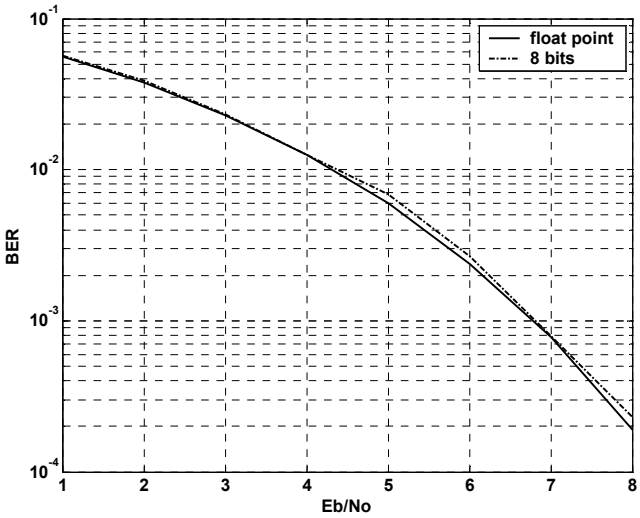

Fig. 4. BER comparison between floating point and fixed point implementation

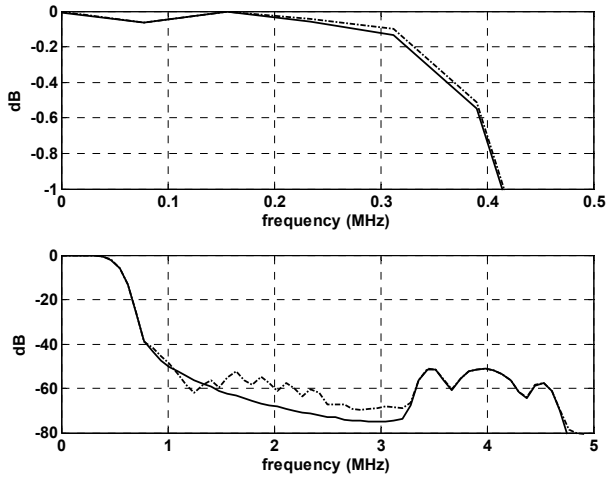

Fig. 5. CIC output signal spectrum: passband (above) and stopband.

Table 1 Implementation results

Resource (slices)	System Generator	DIGIMOD	
		Polyphase PS	Look-up PS
Pulse shaping (PS)	201	118	27 + 2 BSRAM
Half band	90	30	
CIC	67	67	67
DDS+MIXER	281	189	189
Block connections	140	8	4
QPSK modulator	1137	627	354 + 2 BSRAM

Moreover, not only DIGIMOD implementation requires less area per block, but also it does not require extra resources to connect those blocks employed, as System generator does.

Finally, the layout of the QPSK modulator implemented with DIGIMOD is shown in Figure 6. As can it be seen and was mentioned above all the cores are implemented as relative placed macros.

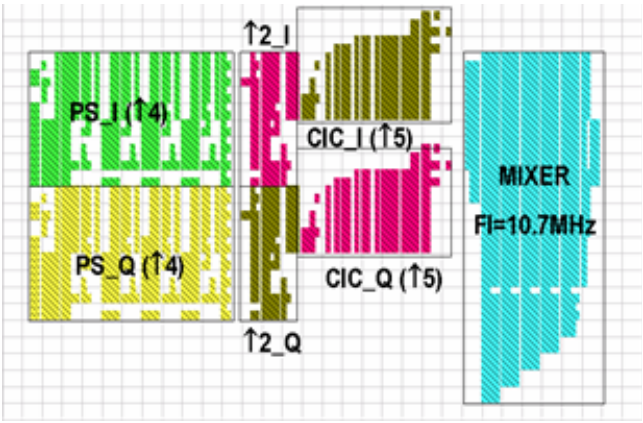

Fig. 6. QPSK modulator layout

4. CONCLUSIONS

In this paper a software tool that allows the software radio designers to develop digital downconverters and, finally, automatically to generate the VHDL code to implement the system on Xilinx FPGAs. A design example of a QPSK modulator has been presented and the results of the implementation on a VirtexE device have been given. The same circuit has been implemented by using Xilinx System Generator and it is shown that our tool leads to an area efficient implementation.

To complete DIGIMOD, our goal is to integrate in this platform all the necessary components of a complete digital modem. So we will add the generation of other type of modulations, the possibility of insertion of known preambles for burst systems, synchronization: phase and time for continuous and burst data transmission, and equalization.

5. ACKNOWLEDGEMENTS

This work was supported by the *Ministerio de Cicencia y Tecnología* under Research Project TIC2001-2688-C03 and in part by the *Universitat Politècnica de València*.

6. REFERENCES

[1] José Marin-Roig, Javier Valls, Vicenç Almenar, "LUT-based Up-converters for FPGA", *Communication Systems, Networks and Digital Signal Processing Conference*, July 2002

[2] Stanley A. White, "Aplications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review", *IEEE ASSP Magazine*, July 1989.

[3] E. Hogenauer, "An Economical class of Digital Filters for Decimation and Interpolation", *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol ASSP-29, nº2, April 1981.

[4] F. Cardells, J. Valls, "Optimisation of direct digital frequency synthesizers based on CORDIC", Electronic Letters, Vol. 37, no. 21, pp. 1278-1280, October 2001

[5] F. Cardells, J. Valls, "Optimized FPGA-implementation of quadrature DDS", *IEEE International Symposium on Circuits and Systems (ISCAS2002)*, May 2002, AZ USA

[6] F. Cardells, J. Valls, "Area-Optimized Implementation of Quadrature Digital Direct Synthesizer on LUT-based FPGAs", *IEEE Trans. on Circuits and Systems II*, Vol. 50, no. 3, pp. 135-138, March 2003

[7] Xilinx Coregenerator, www.xilinx.com.

[8] Xilinx System Generator for DSP v2.2 Reference Guide, www.xilinx.com