A Matlab/Simulink Simulation Approach for Early Field-Programmable Gate Array Hardware Evaluation

Celso Coslop Barbante, José Raimundo de Oliveira
Computing Laboratory (COMLAB)
Department of Computer Engineering and Industrial Automation (DCA), UNICAMP
Campinas, Brazil
e-mail: celsocos@dca.fee.unicamp.br, jro@dca.fee.unicamp.br

***Abstract*— This paper presents a Matlab test bench development for Field-Programmable Gate Array hardware simulation. When a design uses hardware blocks provided by third-part vendors (known as Integration Packages - IP), several options can be set in the block configuration page, inside vendor tool, and affect how the block behaves. These configuration options should be evaluated for any integration package one may be interested in and the test bench proposed facilitates the evaluation of any block-specific configuration parameters, enabling a three times reduction of block configuration time.**

Keywords-Model verification; Matlab; FPGA design.

I. INTRODUCTION

Hardware verification is becoming more challenging as design complexity grows. Verification times have increased with the rising gate count; as overall design complexity grows, ensuring that the system complies with the required specification in early design stage is a desired time saving approach [1].

Textual language can be used to develop a test bench; however, this approach has a degree of complexity similar to design itself and is human-resource intensive. This task can be accomplished easier with a tool like Matlab, demanding less knowledge of vendor specific optimizations [2] to achieve the goal of developing a test bench.

The required computational run-time for simulations is also an important factor to consider because computer resources are limited and costly. Efficient simulation techniques, as presented in this work, collaborates to improve a rational use of computer resources [3].

According to a survey of Collett International Research in 2002, only 39% designs were shipped bug free at first silicon, while 60% contained logic or functional flaws, more than 20% required 3 or more silicon spins. The Collett survey has also shown that nearly 50% of total engineering time was spent on verification [4]. Because the design complexity continually increases, the actual numbers are expected to be worse, being more difficult to verify the design today than in 2002.

Some of these verification challenges can be addressed by using a model-based simulation system, where mathematical aspects and algorithms become a key area in the verification efforts, also incorporating software techniques for formal verification [5-6]. The design methodology that best fits the proposed test bench is a top-bottom design strategy, which can be done using the Matlab and the FPGA tools to generate desired IP models; for example, Fast Fourier Transforms, Arithmetic Logic Units and Decoders, among other blocks that may be provided by Field-Programmable Gate Array (FPGA) vendors.

During design cycles, the model is refined and progressively approaches the hardware behavior, until the hardware IP can be directly used.

In this article, the Xilinx and Matlab integration will be presented with one simple test case, which consists of a Fast Fourier Transform (FFT) processing core, as presented in Section II. A proposed Design Flow is explained in Section III, and the methodology results are presented in Section IV.

II. MATLAB/FPGA SYSTEM INTEGRATION

The first step to achieve the goal to simplify verification using the Matlab is selecting an IP, from available from FPGA vendors, sometimes through third parties companies, but with vendor's support in order to enable support for simulation, hardware models and even synthesis. Two examples of such integration tools are Altera DSP Builder [7] and Xilinx System Generator [8].

Both tools share the same principles, but differ in integration method, capability and support options. The installation procedure details can be found in the vendor's web site and will not be repeated here; however, the process is straightforward once you have checked the Matlab version and FPGA tool version compatibility [9].

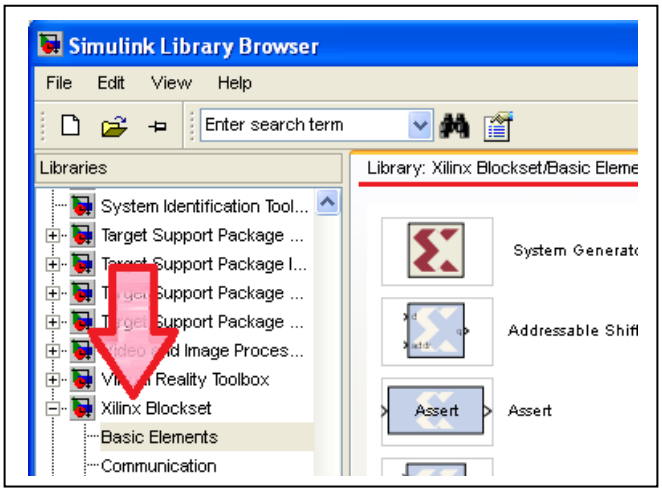

Figure 1. Xilinx Blockset inside Simulink

Each integration package is targeted to a specific Matlab/FPGA vendor tool combination and one must confirm you have correct version of all tools to avoid interoperability problems. The final result will be a Simulink block list inside Matlab, as it is shown in Figure 1.

Under the hood, more changes in the Matlab tool were done and more than a few Simulink [10] blocks are available: a diverse set of FPGA hardware models can be used and a clever use of this integration enable that a test bench developed in Matlab can also be used to validate hardware design, however to use the same Matlab environment for FPGA hardware models you must first generate the models inside the FPGA vendor tool.

During hardware model generation, design choices are required. These choices are made by parameters selections, and each IP has several parameters to be set. Early evaluation methodology to quickly test the model parameters is a major goal for this work, since several blocks can be created, and using the proposed test bench, parameters can be quickly adjusted and compared.

The Xilinx System Generator IPs available in functional categories are, in alphabetical order:

- Automotive & Industrial
- Advanced eXtensible Interface (AXI)
- Base IP (FPGA basic blocks)
- Communication and Networking
- Debug and Verification
- Virtual Input/Output
- Digital Signal Processing
- FPGA Features and Design
- Math Functions
- Memories and Storage Elements
- Standard Bus Interfaces
- Video and Image Processing

For instance, a single Fast Fourier Transform (FFT) core from Digital Signal Processing category should be somewhat simple to use, but this simple IP requires a lot of design choices: The FFT Core can compute from 8 to 65536-point forward or inverse complex transforms. The input data is represented as two's-complement numbers from 8 to 34 bits wide or single precision floating point numbers with 32 bits wide and the phase factors can range from 8 to 34 bits wide.

The FFT IP can use on-chip block RAM or distributed RAM across FPGA; calculation can be done using full-precision unscaled numbers, scaled fixed-point numbers and block-floating point. Some parameters can be configured in run time with additional logic: the point size, the choice of forward or inverse transform, and the scaling schedule.

Finally, four architectures are available to provide a tradeoff between sizes and transform time [11]. With all these options, just for a FFT block, the time required to simulate a hardware level design can be too long, so using hardware models with regular Matlab script files can easier simulate the generated IP.

All major parameters in FFT block generation can be seen in Figure 2, and all this options can be exercised in the proposed test bench.

The test bench is reliable due to modular nature: test cases and model files are separated, thus easier than other graphical-based tools to find issues in the test bench, simulation test cases and add supporting functions. Early simulation in the design flow, before first synthesis, can be used for exercise several design options ahead going further with the project.

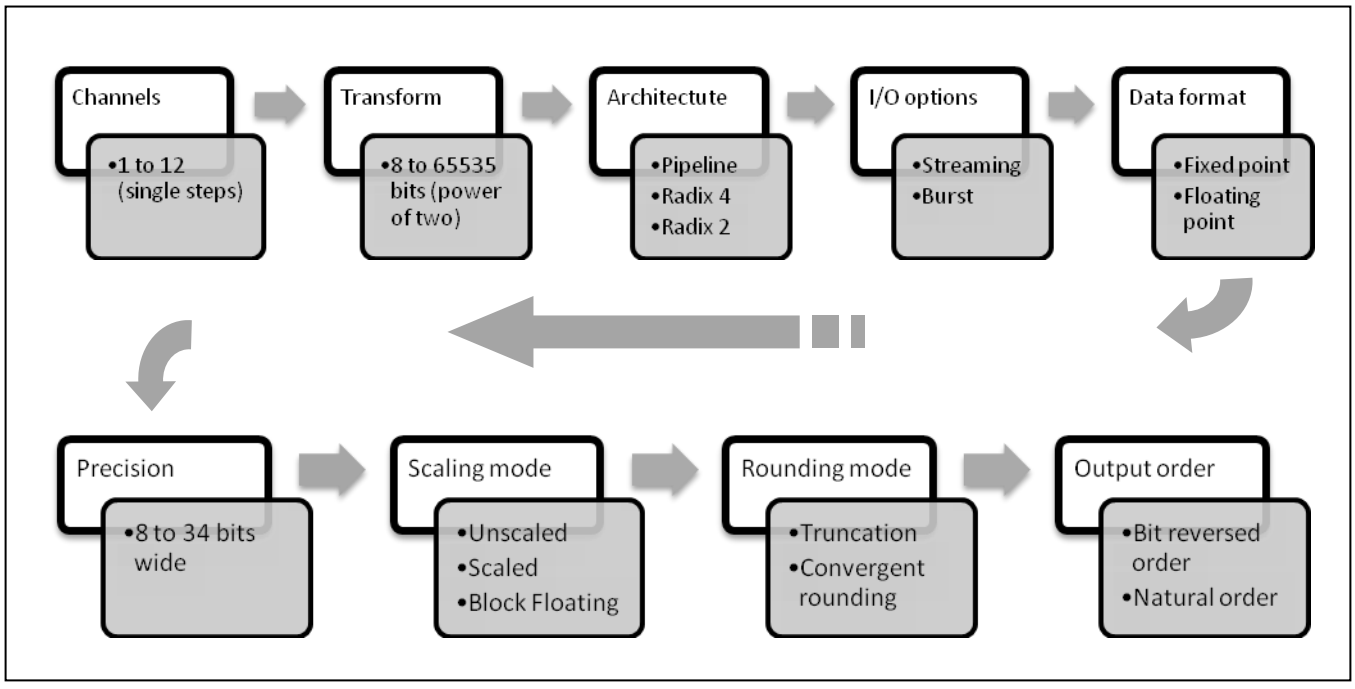

Figure 2. Main options for FFT model generation using Xilinx ISE Core Generator

 ISBN: 978-1-61208-308-7

III. MATLAB/FPGA DESIGN FLOW

A. Developing the matlab test bench

The Matlab is well known as a high-level language and interactive environment for numerical computation, visualization and programming [12]. New features like Hardware Description Language Coder (HDL Coder) and Hardware Description Language Verifier (HDL Verifier) allow modeling, simulating and exploring algorithms by Matlab and Simulink tools and FPGA vendor companion allows generating either target-independent or target-optimized hardware code and program Xilinx and Altera FPGAs.

Figure 3 shows the proposed methodology with a Matlab software model in step one, a hardware model introduced in step two, before hardware verification and reusing Matlab scripts and Simulink diagrams, saving time and test bench code. The final step in the design flow is the real hardware simulation.

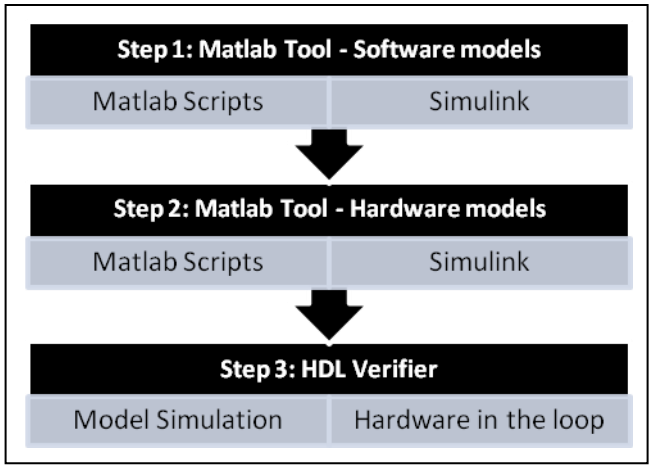

Figure 3. Proposed design flow

To check the FPGA design parameters against the system-level specifications a set of test cases must be developed. The initial test case could use only Matlab functions to model the system under study without adding hardware complexity, to validate the design idea before investing time in device selection, pin-out and others synthesis related issues. The initial test case is very simple and can easily simulate and compare the generated IP against Matlab floating point and full precision functions.

This allows design space analysis and numerical precision evaluation and also keeps the data for latter comparison between hardware and Matlab implementation, creating a figure of merit for quality and easily showing tradeoffs impact and artifacts that may arise due to several design choices made in IP generation.

The Xilinx provided C-model is cycle-accurate and has been demonstrated that results from Matlab, FFT model and System Generator model are all equivalent [13].

To reproduce the results of this work, please note you are not able to use the LCC compiler shipped with Matlab because it will not compile some IP models. Xilinx recommends using Microsoft Software Development Kit (SDK) for windows platform or Gnu Compiler Collection (GCC) for Linux platforms. You can refer to Xilinx user guide [14] and Matlab documentation [15] to create the function models whether is needed.

The test bench starts with a set of Matlab scripts that contain the global variables to control verbosity (to facilitate test bench debug) and design parameters. The test bench calls a set of test cases which can instantiate different models of device under test, for example the first one with the Matlab models and a second one with hardware models provided by FPGA tool.

B. File structure

A test bench top file was created with global variables to control run-time parameters, data sizes and script verbosity. There are parameters to controls text displayed messages during the test bench run and it is useful to debug the test bench itself, but once the environment is working, less debug messages can be displayed to concentrate the focus on the device under test. Utility file functions keep test bench organized and are a good location to place common functions brought by test cases. These are the test bench root files and a set of test cases files to exercise the model.

Once the main scripts and test cases were developed, a high level model using only built-in Matlab functions was made to mimic the desired hardware behavior. These high level models uses all the Matlab capabilities to reduce design time and the result will be compared to the hardware model and to latter validate the IP.

During the development of this work, a small set of utility routines were split into test bench top file and utility functions to keep top file small and easy to change. Moreover, the utility functions can be easily expanded and currently are used to display graphics and store personal preferences in a place which makes more sense than test bench top.

The hardware model can be instantiated by using the FPGA blocks available in Simulink but pay attention that hardware models can also be generated from FPGA vendor tool and embedded in Matlab code by using precompiled functions in the same fashion as regular function is used.

- Test bench top
  - control variables
  - data initialization
  - verbosity control
  - test case selection
- Utility functions
  - draw graphics
  - evaluate errors
  - formatted text output
  - other used test bench functions
- Test cases
  - call software and hardware models
  - execute desired tests and comparisons
  - use of utility functions and model files

 ISBN: 978-1-61208-308-7

- FFT Model direct calculation using Matlab function
  - golden model for floating point FFT
- FFT Model using fixed point calculation
  - golden model for fixed point FFT
- FFT Model using FPGA hardware model A
  - hardware model with "A" parameter set
- FFT Model using FPGA hardware model B
  - hardware model with "B" parameter set
- FFT Model using FPGA hardware model C
  - hardware model with "C" parameter set
- More FFT models can easily be included.
  - create as many models as required

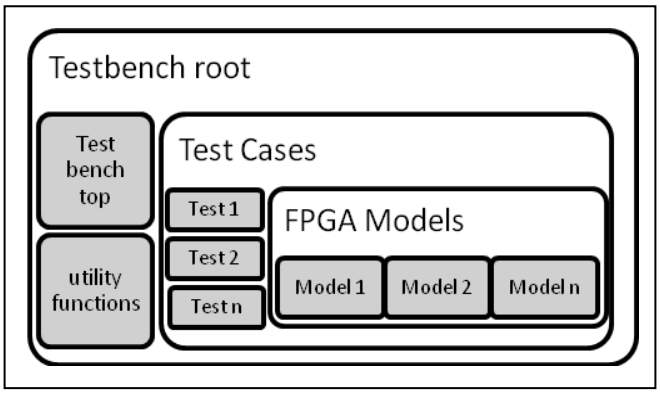

Figure 4. File structure example

When a FPGA IP model is generated by using the Core Generator tool, the model file is also generated. This model is placed inside IP directory tree, created by Core Generator tool and can be used in Matlab, but the integration between Matlab and FPGA vendor tool must be up to date because the Matlab model will call a pre-compiled file (for Altera) or bit accurate C models (for Xilinx).

Keep in mind that simply copying the model file for another PC running Matlab will not work, because the models rely on FPGA vendor files to work.

In the example provided in Figure 4, three files for FFT test cases and three files for FPGA models are shown, but during development, many files can be created as many options can be evaluated in the IP generation procedure. Use as many files as required to represent different parameters analyses in the test bench.

For a quick analysis of design space and numerical precision loss, this is very convenient, simple to design and to reuse.

IV. RESULTS

Once the test bench is ready and the software level function is working, the simulation is very simple to be repeated because the entire test bench is parameterized.

The Matlab scripts calculate buffer sizes and compare with provided data and other similar tasks, in order to provide a sanity check during simulation run time. This help to avoids mistakes in data format and parameter setup, because the sanity checks try to reproduce the constraints available in the FFT manual.

The result from this test bench development using automatic calculation instead of hard-coded values is a deeper knowledge of the FFT IP and deeper comprehension of the FFT IP manual.

The test bench functions generate warning messages, trigger some double checks in IP specification to confirm if test bench behavior was accurate.

It was possible to simulate the hardware and evaluate design tradeoffs, simply using the model and comparing the results between high level function and the vendor-provided hardware model function.

A lot of experimentations with FFT IP parameters were possible, helping to find the best fit for IP speed, area size and data size. All those experimentations were easy to reproduce by saving the hardware model files, reducing the amount of time compared with traditional Simulink design flow with lower level test benches. The easy reproducible results are welcome result of this methodology.

Compared with traditional Simulink graphical approach, this text-based methodology with multiple model files that can be exercised in the same simulation run, has potentials to give early-results for comparison, enabling for example the analysis of fixed point quantization errors early in design saving synthesis time and reducing efforts in final hardware creation [16-17].

For the FFT IP core generated with parameters presented in Figure 2, the result for simulation with hardware and software models is depicted below. The algorithm uses the FFT block to simulate a power spectrum calculation from a modified sinusoid signal with a peak power at low frequency.

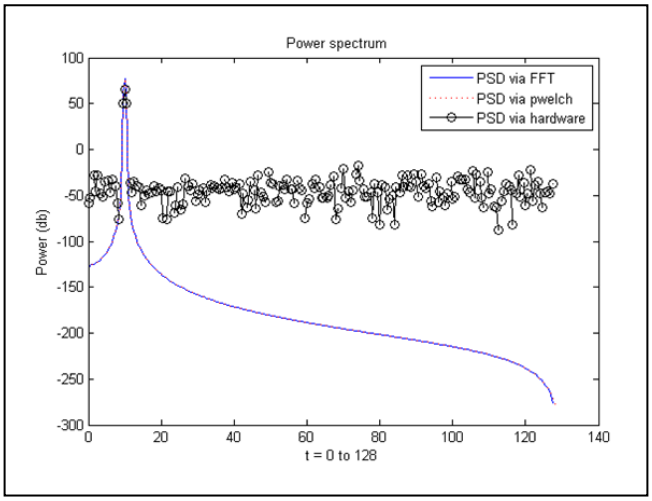

Figure 5. Power spectrun evaluation: Comparing pwelch function, software FFT model and hardware FFT model

Three simulations are shown in the Figure 5. A first one uses software function to calculate the power spectrum coefficients, the second uses Matlab pwelch function, and finally, the same calculation is done with FFT via hardware model.

It is possible to see that pwelch and Matlab FFT functions match to each other. This makes sense because

 ISBN: 978-1-61208-308-7

both functions use the same precision to calculate the power. However, a significant difference between the result from the Matlab software functions and the hardware model equivalent can be found.

This is expected because hardware models use fixed point precision and with the proposed test bench this result was easy to reproduce for several model configurations, resulting a better design space analysis.

The result provided in Figure 5 was found before first FPGA synthesis, directly in Matlab and much faster than traditional Simulink flow with hardware in the loop simulation.

V. CONCLUSIONS

A reduction of three times in simulation setup time was experienced for the FFT block. Saved time increases proportionally to quantity of simulated models, because once the file structure is deployed and first model is exercised, it is very simple to add more models to the test bench. The test bench development contributed to a better understanding of IP parameters, design arguments and options.

Initial hardware results were ready to analysis without even synthesize the designs and the Simulink graphical interface was avoided and this is an important feature, because it is faster to design a test bench by using text than graphical interface.

ACKNOWLEDGMENT

The authors would like to acknowledge the financial support of CAPES and the support of the Department of Computer Engineering and Industrial Automation (DCA) at State University of Campinas – UNICAMP.

REFERENCES

[1] E. Linehan and S. Clarke, “Managing hardware verification complexity with aspect-oriented model-driven engineering”, [retrieved: September, 2013], Available: http://ulir.ul.ie/handle/10344/666

[2] Synopsys Inc., “Faster simulation without more hardware”, [retrieved: October, 2013], Available: http://www.synopsys.com/Company/Publications/SynopsysInsight/Pages/Art2-FasterSimul-IssQ2-11.aspx

[3] S. Narayanan and L. Rothrock (editors), “Determining the number of simulation runs: Treating simulations as theories by not sampling their behavior”, Human-in-the-loop simulations: Methods and practice, Chapter 5, Springer, 2011, pp. 97-116

[4] P. J. Mosterman, “Model-based design of embedded systems”, Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education, IEEE Computer Society, IEEE Press, June, 2007, pp. 197-199, doi:10.1109/MSE.2007.65

[5] P. S. Kaliappan, “Model based verification techniques”, May, 2008, unpublished paper

[6] V. D. Silva, D. Kroening, and G. Weissenbacher, “A survey of automated techniques for formal software verification”, IEEE Transaction on Computer-Aided Design of Integrated Cricuits and Systems, Vol 27, no 7, July, 2008, pp. 1165-1178

[7] Altera Inc., “DSP builder”, [retrieved: August, 2013], Available: http://altera.com/products/software/products/dsp/dsp-builder.html

[8] Xilinx Inc., “System generator for DSP”, [retrieved: September, 2013], Available: http://www.xilinx.com/tools/sysgen.htm

[9] Xilinx Inc., “Which versions of System Generator for DSP and Accel DSP synthesis tool are compatible with which versions of ISE design tools and MATLAB?”, [retrieved: September, 2013], Available: http://xilinx.com/support/answers/17966.htm

[10] The Mathworks Inc., “Simulink - simulation and model based design”, [retrieved: October, 2013], Available: http://www.mathworks.com/products/simulink/

[11] Xilinx Inc., “Logic core IP: Fast fourrier transform product specification”, [retrieved: August, 2013], Available: http://www.xilinx.com/support/documentation/ip_documentation/ds808_xfft.pdf

[12] The Mathworks Inc., “Matlab (rel. 2011a – 2012b)”, [retrieved: September, 2013], Available: http://www.mathworks.com/

[13] J. Wu, “FFT results from Matlab fft, bit accurate C model and SysGen FFT block”, July, 2010, [retrieved: August, 2013], Available: http:// myfpgablog.blogspot.com.br/2010/07/fft-results-from-matlab-fft-bit.html

[14] Xilinx Inc., “LogiCORE IP fast fourier transform v9.0: product guide for vivado design suite”, [retrieved: August, 2013], Available: http://china.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf

[15] The Mathworks Inc., “Create MEX-files: Build C/C++ and Fortran subroutines into MATLAB functions” [retrieved: September, 2013], Available: http://www.mathworks.com/help/matlab/create-mex-files.html

[16] The Mathworks Inc., “FPGA design and codesign with Matlab”, [retrieved: August, 2013], Available: http://www.mathworks.com/fpga-design/

[17] S. V. Beek and S. Sharma, “Best practices for FPGA prototyping of MATLAB and simulink algorithms”, [retrieved: September, 2013], Available: http://www.eejournal.com/archives/articles/20110825-mathworks/

 ISBN: 978-1-61208-308-7