Rapid Prototyping of Wireless Physical Layer Modules Using Flexible Software/Hardware Design Flow

James Chacko, Cem Sahin, Douglas Pfiel, Nagarajan Kandasamy, Kapil Dandekar
ECE Department, Drexel University, Philadelphia, PA 19104
{jjc652,cs486,dsp36,nk78,krd26}@drexel.edu

ABSTRACT

This paper describes a step by step approach in designing wireless physical layer modules starting from a software implementation in MATLAB to a hardware implementation using Xilinx SysGen and ModelSim. The described design flow promotes baseband physical layer research by providing high flexibility and speed to the process of module creation verification and deployment. The novelty introduced into our system lies within the flexible components created using this design flow, which enables on-the-fly modification of multiple parameters to suit various wireless protocols.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: Logic Design - Design Aid—*Automatic synthesis; Verification; Simulation*

Keywords

digital baseband, design flow, MATLAB, Xilinx SysGen, ModelSim

1. INTRODUCTION

Wireless protocols are often implemented in custom hardware in order to satisfy heavy computational requirements within tight time constraints. Hardware implementations can be cumbersome to design and verify and therefore require longer development times. A programmable software implementation of the physical layer, also called Software Defined Radio (SDR), is therefore very advantageous in terms of supporting multiple protocols, faster time-to-market, higher chip volumes and easy modifications. Though convenient, SDR relies on the researcher to choose between the wide array of solutions available that are different in their design language, implementation constraints and flexibility.

Apart from flexible characteristics inherent to SDR's, another factor that takes prominence on the choice of SDR would be the design flow involved, which can vary significantly in the flavor of software/hardware design languages used. For instance, based on the chosen SDR platform designing can be either done with high level languages like MATLAB, Python, C, etc that may or may not translate automatically into hardware. It can involve writing codes in HDL languages such as Verilog and VHDL. Through this paper, we'll discuss the requirements we faced designing wireless physical layer modules and tools that were selected for implementation. This design tutorial is organized as follows: Section 2 discusses some of the major design tools wile Section 3 discusses the requirements in choosing the right one. Section 4 describes our proposed design flow in detail and we will finally conclude with Section 5.

FPGA'15, February 22–24, 2015, Monterey, California, USA.

http://dx.doi.org/10.1145/2684746.2689084.

2. DESIGN TOOLS

Developing any part of the wireless physical layer is easiest when first implemented in software, and MATLAB is the most extensively used tool in this area. MATLAB provides a high level interactive environment for numerical computation and programming. It is a powerful tool in developing wireless applications especially with its built in math functions and application specific toolboxes [1]. As an add-on to its coding environment, MATLAB also provides Simulink which is a graphical programming language tool for modeling dynamic systems, such as wireless systems, that require a wide range of signal processing capabilities. A similar piece of software to Simulink would be LabVIEW with its graphical communication toolsets [2] and its graphical object oriented approach in developing wireless models. LabVIEW is a newer addition to wireless design development tools with provisions to run m-code within its environment to translate models previously written in MATLAB script. Off the shelf SDRs such as WARPLab [3], GNUradio [4] and SORA [14] allow the user to program the baseband/physical layer in MATLAB, Python and C, respectively, and use modular radio front ends for transmission and reception. These SDRs are relevant for conducting wireless baseband level research in software but they do not provide the runtime advantage of having a hardware implementation to generate faster results.

Wireless hardware system development is often implemented through HDL coding languages like Verilog and VHDL at the basic level with common development environments being ModelSim, Altera Quartus and Xilinx ISE [5–9]. ModelSim is more simulation focused, while the latter two are more focused on targeting and running HDL code on hardware. Both Xilinx ISE and Altera Quartus have communication design toolboxes that ease the development of wireless modules. The availability of automatic script or graphical

software design translations to hardware makes hardware development easier for the development of wireless physical layer designs. Examples of such designing platforms would be MATLAB Simulink/SysGen and Parallel Application from Rapid Simulation (PARS) which both translates designs directly to hardware [10, 11]. While PARS requires Sundance hardware, MATLAB Simulink/SysGen can target a wide variety of FPGA boards. In both cases, automatic translation depends largely on the complexity and availability of the hardware libraries to compile hardware equivalent designs. In certain research areas, such as wireless physical layer design, it can be difficult to know which libraries will be required, thought this might be evident later based on how a project matures and branches. In such cases the challenge then is drawn to programming within the constraints of the chosen tool. Switching design flows can be a cumbersome task in any project and therefore choosing the appropriate design environment and tool that can scale in the future is extremely important. Wireless research as described in [12, 13] develops a flexible software-hardware implementation that can be leveraged to research various current and future OFDM based wireless protocols.

No matter which design flow is chosen, there are certain essential requirements in a design environment which can also be relevant and specific to the kind of project being undertaken. In this paper, we are particularly interested in the design and research involving the baseband layer of wireless standards that benefit from the flexibility that is typical of a software implementation as well as real time performance which is typical of a hardware implementation. In the next section we will go over the requirements we saw essential in choosing an appropriate design tool.

3. DESIGN TOOL REQUIREMENTS

Flexible Designing Environment: This is the most significant attribute in choosing a design language and environment. The programming method for building the design can either be script based or GUI based, but must provide granularity and flexibility in the aspect of building complex designs based on functions and libraries that are built-in to the programming language. In the case of designing the wireless baseband/physical layers, programs with built in communication libraries and functions are beneficial and would give the programmer an edge in implementing designs at a much faster rate than otherwise would have taken longer if implemented from scratch. The language should also be flexible enough to have user defined functions/modules integrate well with built in functions/modules, which requires well documented descriptions for ease in integration.

For instance, in order to build a quadrature amplitude modulation (QAM) module, that maps bits into waves, the presence of built in QAM functions and libraries would ease implementation but these libraries may not be available for more complex modules comprising the wireless baseband. QAM itself can vary in its scale setting which decides on how many bits to map at a time: BPSK, 4QAM and 16QAM that maps 1, 2 and 4 bits, respectively. Varying data rates through modules especially in a pipeline are much easier to implement in software than on hardware where implementation relies on data to be predictive. Thus, the need arises for flexibility in the use of available functions/libraries and the programming environment itself.

Debugging and Verification Environment: The debugging and verification environment associated with a programming language is nearly as important as designing the system itself. The debugging environment becomes particularly relevant in complex designs that cannot can not be built in a tier system that are otherwise inherent in smaller designs. Complex system designs involving feedback and control loops, as in wireless baseband/physical layer designs, are far more complex to debug and verify.

The channel estimation block is an appropriate example of a complex module within the wireless baseband layer that provides channel data to rectify the received signal. The complexity in this block is due to the processing speed and data alignment with which it provides feedback. An error within would be harder to locate and often forces the programmer into tracing errors from scratch if not aided with higher level debugging tools like signal captures and waveforms. The effort of debugging and verification also changes based on if the design resides in software or on hardware and it is best to choose an environment that has strengths in debugging in both areas.

Result Capture and Processing: Based on where the design resides, software or hardware, the process of gathering results for processing can be a challenge of its own. A fully software based implementation benefits from being able to capture data from across the entire design easily, whereas significant work needs to be done to capture the same for a hardware based implementation. In the area of wireless research, where designs may reside on software, hardware or consist of a complex integration of both software and hardware cores, the availability of capturing and processing data flexibly is a significant attribute of the right design tool.

For instance, in the case of implementing the QAM module discussed earlier, data going into and out of the module are totally different in it's characteristics, one being bits and the other fixed point decimal values on hardware or floating point in software. Research in this area depends on visualizing channel mapping results and the ability to capture data around the QAM module is necessary irrespective of whether its on software or hardware.

4. DESIGN FLOW

In this section of the paper, we are going to have a detailed discussion of our design flow that starts from utilizing the flexibility provided with a full software implementation to utilizing the speeds provided with a full hardware implementation. We use this design flow in implementing the baseband/physical layer modules targeted for wireless research.

4.1 MATLAB Script Implementation

As the first step towards implementing a module, we write MATLAB m-code that will also be used as a blueprint to build further upon the model if required. We chose MATLAB because of the high level interactive environment for numerical computation and availability of pre-built communication toolboxes and function libraries. At this stage, there are a few relevant points to be kept in mind while writing MATLAB scripts. The first would be to have the option of changing function variables and parameters from outside the function call itself for comparative flexibility which will get more clearer in the next few sections. The second would be to have a structured data source and data

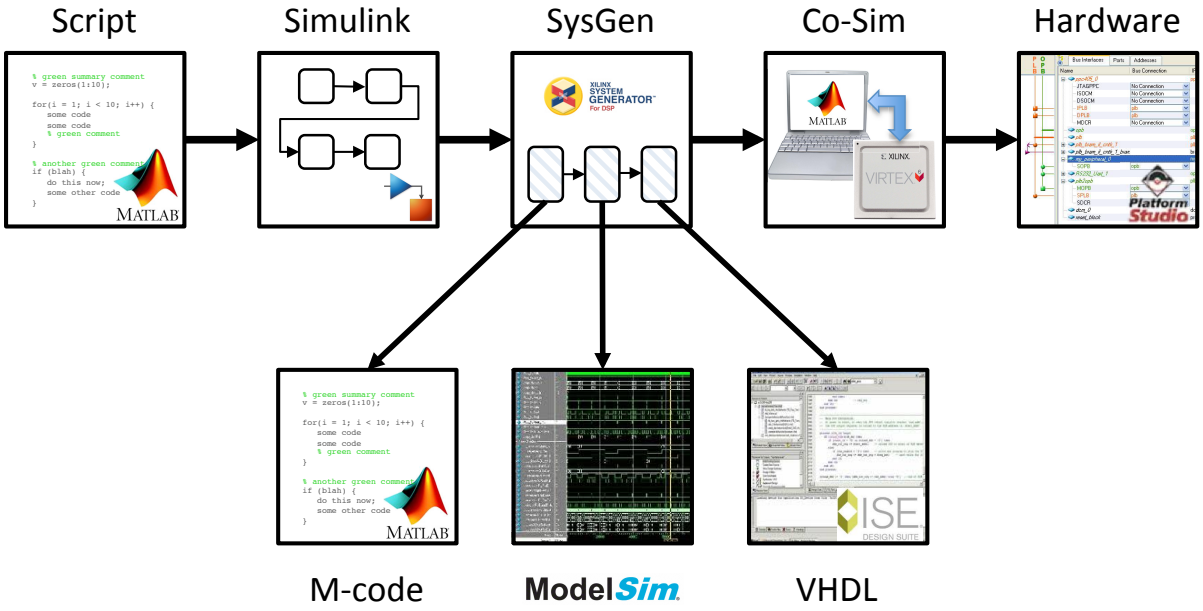

Figure 1: Proposed design flow for projects implementing baseband/physical layer modules for wireless research.

sink plotting/processing output data for quick script level verification of functionality and correctness.

4.2 MATLAB Simulink Implementation

As the next step, we use MATLAB's Simulink graphical design tool and create the functional equivalent of the script that was previously developed. The Communication Systems Toolbox available in Simulink provides a wide array of built-in wireless modules that helps the creation of wireless physical layer modules, but its library implementation may vary with the one in script. To continue, have the results from both the script and Simulink designs compared against each other to ensure script and Simulink design level coherence to the same set of input and output data that can be captured easily within MATLAB's workspace. For those modules that do not have built-in equivalents for the functions to be implemented, it can be put together with the logic level functions/modules Simulink provides.

4.3 SysGen Implementation

In this section, we are going to use Xilinx System Generator (SysGen) Toolkit that is present as an add-on within the Simulink library in MATLAB to create the SysGen functional equivalent of the Simulink design that was created in the previous step. By using building blocks provided in the SysGen library, we will be able to build custom hardware IPs and directly export them to hardware using SysGen's export tools. The implementation change from floating point arithmetic to fixed point arithmetic has to be kept in mind while designing blocks in SysGen. After completing the design it must be crosschecked for errors against its Simulink implementation by having the data to the SysGen modules sourced directly from the Simulink data source and results captured and processed within the MATLAB workspace. This will ensure functional correctness and show the range of error from conversion to fixed point implementation.

4.4 ModelSim Implementation

Complex Verilog/VHDL functions and controls can be created through ModelSim and imported into MATLAB's Simulink environment using the BlackBox SysGen module. This allows building of complex functions much quicker than otherwise would have taken for piecing together SysGen library blocks. Once created and imported into the design, the models can be further studied against the inputs from the Simulink/SysGen design by invoking ModelSim to scope the signal and control flow through waveforms for ease in debugging.

4.5 Software/Hardware Co-Simulation

Wireless communication research benefits greatly from hardware support in the form of accelerators and radio front ends. Since a full baseband implementation in hardware is difficult to implement from scratch, having the ability for software/hardware co-simulation enables software flexibility, hardware speeds and access to the wireless channel.

4.5.1 PC Driven Interface

At this stage, our hardware designed through SysGen will be verified for their correct functionality on an FPGA, a ML605 board in our project, using real data before moving forward. Our Simulink-based design provides capabilities to fulfill this need. Leveraging the features available through the Simulink software, we set up a software-hardware Ethernet co-simulation platform. In this scenario, the experiment master is set up as the host PC, which orchestrates the full experiment.

4.5.2 Data Input/Output

For our co-simulation experiment, we initialize the data using the MATLAB workspace, where each data vector is created and prepared for transmission through our FPGA design. The co-simulation feature feeds data into these entry points at the beginning of the experiment via the Ethernet

connection onto the ML605, allowing them to flow through our FPGA design, and captures the data as the experiment is progressing. The captured data is finally saved back into the MATLAB workspace, where a final script checks the received data against the expected results for correctness. It should be noted that even though the ML605 is the one processing the data, this scenario allows us to tap into the data's flowpath at any point with the use of Simulink scopes and identify the problematic stages on the hardware if there are any error flags.

4.6 Hardware Implementation

Although the time and effort needed to set up a software-hardware Ethernet co-simulation experiment is short, the runtime for each experiment is extremely long. The performance of our custom FPGA design is diminished significantly due to the host PC being the experiment master. The co-simulation framework does prove to be a valid method to verify our hardware design rapidly for smaller data sets. Since it cannot provide results quickly for larger data sets within a reasonable time, we will discuss a faster method that will unleash the full performance of our hardware implementation.

4.6.1 Microblaze Driven Interface

Close-to-real-time measurements are ideal when trying to determine the characteristics of hardware designs. A fully embedded design achieves this goal with few additional design requirements. For a data sourced development process, we complemented our hardware design with an on-board data generator, which also has the capability to keep track of error rates. The control of the experiment has now been moved from the host PC to the on-board processor, MicroBlaze, available on the ML605. In addition to implementing and integrating the on-board data generator, the code loaded on the MicroBlaze initializes the board and starts and controls the experiment throughout its runtime. This design was developed by exporting the MATLAB Simulink/SysGen design directly to Xilinx Embedded Development Kit (EDK) and then using Xilinx Software Development Kit (SDK) that allows to communicate with the FPGA through C code running on the on-board MicroBlaze processor.

4.6.2 Data Input/Output

As introduced above, the data is fed into our design using our custom on-board data generator. This hardware block simply accepts a seed and outputs randomly generated data. Using carefully calculated delays, the same data (using the same seed) is generated at the receive end of the design as well for cross checking. Once the MicroBlaze sends the signal to start the design flow, the data is sent through all the stages of our design serially and looped-back to the receive path. The pre-generated data is then compared against what was captured from the receive path. The total bit count, and the bit error rates are updated in real-time within the data generation block registers, which are accessible from the MicroBlaze at any time for verification. Upon full completion of the experiment (i.e. the total bit count reached the user-preset count set for the experiment), the total bits sent/received, total number of errors, and the bit error rate (BER) data are also printed to the COM port for user analysis.

5. CONCLUSION

This paper throughly describes the requirements we found essential in choosing design tools for designing the baseband/physical layer for wireless research in software and then realizing it in hardware in a step-by-step process for assembling a novel research tool. The described design flow benefits from the flexibility provided with software implementation and hardware runtime speeds. We also described tool techniques that are relevant in SDR research that we used for debugging and verifying complex designs comprising of both software and hardware modules with ease.

Acknowledgment

This project is supported by the National Science Foundation through grants CNS-0854946 and CNS-0923003.

6. REFERENCES

[1] Matlab Communications System Toolbox. http://www.mathworks.com/products/communications/.

[2] LabView Tools: IP for Software Designed Instruments. https://decibel.ni.com/content/docs/DOC-29613.

[3] "Rice University WARP - Wireless Open-Access Research Platform (WARP)." http://warp.rice.edu.

[4] GNU Radio Overview. http://gnuradio.org/redmine/projects/gnuradio.

[5] Xilinx Comprehensive Baseband Solutions. http://www.xilinx.com/applications/wireless-communications/baseband/index.htm.

[6] Xilinx Wireless IP, Reference Designs. http://www.xilinx.com/esp/wireless/.

[7] Altera Software Defined Radio. http://www.altera.com/end-markets/wireless/advanced-dsp/sdr/wir-sdr.html.

[8] ModelSim-Altera Edition Software. http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html.

[9] ModelSim: ASIC and FPGA design. http://www.mentor.com/products/fv/modelsim/.

[10] Matlab HDL Verifier. http://www.mathworks.com/products/hdl-verifier/.

[11] Parallel Application from Rapid Simulation. http://www.sundancedsp.com/development-tools/pars.

[12] J. Chacko, C. Sahin, D. Nguyen, D. Pfeil, N. Kandasamy, and K. Dandekar. Fpga-based latency-insensitive ofdm pipeline for wireless research. In *High Performance Extreme Computing Conference (HPEC '14). 19th Annual HPEC Conference*, Sept 2014.

[13] B. Shishkin, D. Pfeil, D. Nguyen, K. Wanuga, J. Chacko, J. Johnson, N. Kandasamy, T. Kurzweg, and K. Dandekar. Sdc testbed: Software defined communications testbed for wireless radio and optical networking. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2011 International Symposium on*, pages 300 –306, May 2011.

[14] K. Tan et al. Sora: high performance software radio using general purpose multi-core processors. In *Proc. USENIX Symp. Networked Systems Design & Implementation (NSDI)*, pages 75–90, 2009.