A Traceable Workflow For Software-Defined Radio Development

Travis F. Collins and Andrei Cozma

***Abstract*—In this paper we discuss a workflow for software-defined radio development, which uses modern software and hardware tools to accelerate the design process. Specifically, this work addresses the historical disconnects in the flow from initial algorithm design and conception, through to final HDL and hardware deployment. This flow is demonstrated through an example modem design and custom hardware for deployment, utilizing the Analog Devices AD9361-Z7035 System-on-Module. The main advantages of this proposed workflow is that it provides traceability in a design and naturally segments tasks to an engineer's domain of expertise, but also provides convenient areas of convergence between disjoint group members.**

***Index Terms*—SDR, FPGA, AD9361, SoM.**

I. Introduction

THE development cost and time of a modern communication or radar system is substantial, and the teams designing such systems are becoming smaller and less hardware averse [1]. A typical wireless system itself can be broken into sections including: analog front-end conditioning at RF, conversion stages to baseband, digital conversion, and baseband processing. A large majority of the analog front-end and conversion sections have been addressed with modern transceivers, such as the popular Analog Devices AD9361 [2], but downstream baseband processing is still left of to the radio developers. For a general purpose transceiver it can be difficult to integrate any of these components since they are waveform and application specific. However, implementing such functionality in custom application specific integrated circuits (ASIC) can be a costly endeavor. Even implementing a system with a programmable interface, such as a field programmable gate array (FPGA), can still be complex. Due to the interface understanding to talk to a complex converter or transceiver, as well as the signal processing expertise to implement a wireless system.

To reduce development risk current industry trends for RF systems is to move from ASICs to programmable Systems on Module (SoM) or Systems on Chip (SoC), integrating RF transceivers with programmable logic and CPU cores. An example of such a system is the Analog Devices ADRV9361-Z7035, which bundles the AD9361 and a Xilinx FPGA. This device provides an off-the-shelf solution to directly interface with a transceiver, but which is also field deployable unlike many FMC based evaluation kits. HDL reference designs are also provided with these boards so users can start from a functional design and focus on their application specific pieces.

With this focus on FPGA devices the downstream signal processing blocks that traditionally were implemented in ASICs are now provided as intellectual property (IP) units, comprising both HDL and software. These components are deployed in the programmable logic and the processing units of the new RF systems yielding much more flexibility in terms of functionality, application specificity and update/upgrade capabilities. Therefore, the same device can be used in an array of applications, as well as easily upgraded over its life span.

Developing these IPs requires deep knowledge of communications and signal processing algorithms, as well as the caveats when such algorithms are deployed into hardware and face impairments that are commonly overlooked in theoretical analysis. This disconnect coupled with traditionally hardware centric designs is departing to new software focused implementations where software-defined radio (SDR) systems have become the dominant approach for any wireless prototyping and development. SDRs tightly couple hardware and software, making a device highly programmable but also highly complex. Therefore, implementors must have the necessary skills to manage software control, algorithm design, and FPGA constraints. A rare combination of experience for any engineer.

This paper focuses on a design flow for SDRs that exploits current tooling paradigms and provides an avenue for traditionally separate development teams to work in this new converged methodology. Instead of requiring a single developer competent in all areas, this design flow breaks down an implementation into segments that complement engineer's skill and allow a team to work more effectively. This is approached through an example design of a software defined modem targeting the Analog Devices ADRV9361-Z7035 SoM, based on the AD9361 agile RF transceiver and the Xilinx Zynq SoC. The paper discusses the tools, the design and deployment processes and brings to light the challenges encountered to successfully run the system in the real world. In this work we explicitly define design stages to complement tool flow and design goals, from initial simulation, through prototyping and finally into production.

A. Contributions

The main contributions of this paper is the introduction of a workflow which:

- Provides early access to hardware and real-world signals to algorithms engineers.
- Enables consistent algorithm validation on prototyping and production hardware.
- Reduces design time and missed requirements by test driven implementations.
- Allows engineers to focus on areas in their domain knowledge.

T. Collins and A. Cozma are employees are Analog Devices, Inc.

II. Background

A modem design is a complex process requiring development of a set of algorithms to recover a waveform with a specific structure. This typically follows a path of mathematical derivation and then simulation for initial validation. Such simulations are performed with tools like MATLAB, Python, or even C/C++ for mathematical computation. For ease of implementation, this work is done with floating point data types and complex mathematical libraries, which accelerate this validation process. However, these designs cannot be directly mapped into hardware friendly versions, and generally require substantial rewrites and possibly new algorithms. In Figure 1, we outline this design process in detail for many design teams. After the initial MATLAB reference design, the algorithms will be converted to a more multi-threaded friendly language like Python for real-time signal processing with hardware. A popular SDR framework like GNURadio [3] uses this approach to perform signal processing in a dataflow type implementation for performance [4]. Once validated against

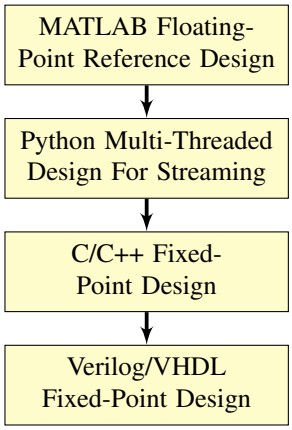

Fig. 1. Design flow for tradition modem design with disjoint stages from initial algorithm conception to final hardware ready HDL design.

with hardware, the algorithms can begin the process of conversion to FPGA friendly fixed-point design. Typically, a design is moved to C/C++ and purely implemented with integers or custom fixed point primitives that closely match FPGA mathematical operations. Proving out fixed-point algorithm performance in C/C++ is much simpler than doing so in Verilog directly.

Many tools have been developed to help aide in this design process, but heavily focus on the algorithmic conversion stages. These include SystemC, Xilinx System Generator (SysGen), Xilinx High-Level Synthesis (HLS), Intel DSP Builder, Intel HSL Compiler, and MathWorks HDL Coder. SystemC and both HLS tools allow automatic conversion of C/C++ code into register transfer level (RTL) for a targeted FPGA. SysGen, DSP Builder, and HDL Coder all rely on Simulink for system construction and numerical simulation. For a system designer, HLS compilers rely on IP integration primarily for system level integration. However, with the Simulink based tools much of this integration can be implemented and tested in Simulink itself, which can be favorable to those less familiar with synthesis tools.

The tools discussed so far, typically produce monolith designs with minimal interfaces to the outside world. However, alternative design patterns use FPGAs as acceleration engines rather than a standalone design for the field. RFNoC [5] fits into this category, which is an extension to GNURadio where processing blocks are placed on the FPGA but can be arranged in any order and in conjunction with CPU processing blocks. However, RFNoC will always require some host control for management and configuration from GNURadio which is undesirable for many production systems.

When considering the workflow for RFNoC, processing block implementation is a disconnected flow which requires switching between many different tools. Making traceability cumbersome for a developer, like the flow presented in Figure 1.

III. Design Workflow

In this paper we will present a flow which complements the available tools, but also simplifies the design process by maintaining traceability from simulation to HDL deployment. The design flow will be demonstrated through an example modem design, which was implemented by two engineers. One full time algorithms engineer and one part-time systems engineer. We will also provide possible deviations from this workflow to utilize other tools, but still maintain the desired traceability throughout the design process.

Separation of Tasks between developers

Figure 2 provides a breakdown of the four stages of the design: algorithm development, design elaboration, prototyping, and finally production. All four stages remain in the MathWorks suite of tools (MATLAB and Simulink), which provides us the traceability through each step in the design assuming some implementation strategies by a user. The first stage in the design process is algorithm development is self evident, and is a common starting point for many designers as discussed in Section II. In the case of this workflow, a designer will create the necessary tests which verify the performance of their simulations. This will be a foundational baseline which will be used to compare futures variations against, sometimes referred to as a golden reference. To simplify this process MATLAB has a built-in test framework called MATLAB Unittest [6], which makes writing repeatable straightforward. Since Simulink can be considered an extension of MATLAB, programmatic control of Simulink models can also be performed within the MATLAB Unittest.

Tests derived in this stage should map to system level requirements as much as possible, which is a efficient test driven coding flow. To complement simulations, real-world data can be streamed into these simulation through Industrial IO (IIO) infrastructure [7], available on all transceivers and many data converters. Utilizing streamed data from hardware provides a more robust level of validation on hardware an algorithm is designed to eventually run upon. Due to MATLAB and Simulink's tight integration with IIO, passing data to and from devices and be implemented with the test cases themselves.

Once the algorithms and general signal processing components have been verified, we can move on to the task of

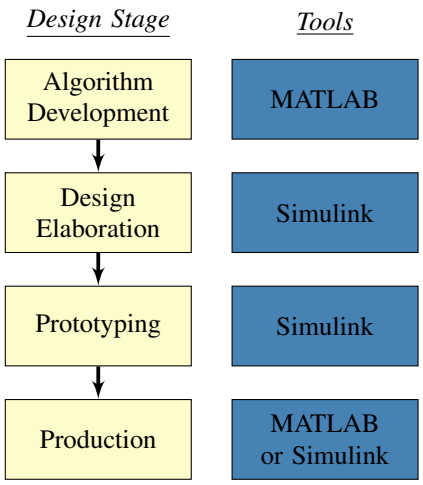

Fig. 2. Proposed design flow for traceable designs, from initial algorithm conception to deployable HDL.

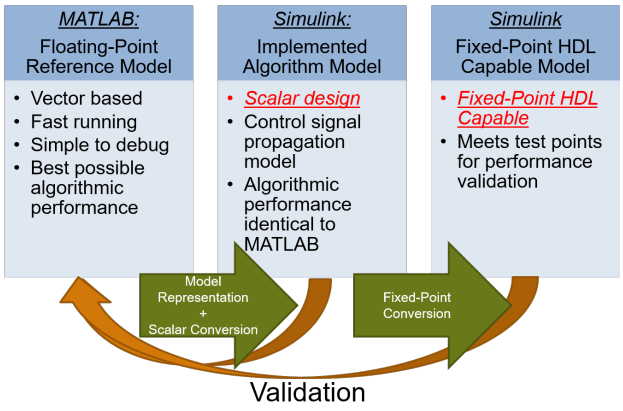

Fig. 3. Staged design variation to get from MATLAB to a HDL capable Simulink model.

getting them to hardware. In this case we will utilize HDL Coder [8] to create Verilog and RTL for our eventual deployed design. Therefore, we must convert our design into a Simulink model. This stage of the workflow is design elaboration, where were are explicitly modeling all the necessary control signals and data paths within our system. MATLAB is a great tool for algorithm implementation; however, Simulink is better at describing system level features and more importantly generating HDL code.

A. Design Elaboration

The eventual goal is to create a model that will generate HDL code, but this requires our algorithms to be based in Fixed-Point data types. A second requirement of HDL Coder specifically, is support for sample or scalar based mathematics. Therefore, any algorithms that rely on vector calculation, which is very common in MATLAB programming, must be converted as well. To ease this process the design process is split into three distinct variation. This simplifies the design process, and makes updates or feature improvements testable and sustainable in the future. The three distinct variations are outlined in Figure 3.

Figure 3 breaks down the three design variations starting from the left with our baseline MATLAB float-point reference model. This is the model that was already constructed in the algorithm development stage. The second design variation, which is the first Simulink model, is called the implemented algorithm model. This model still uses floating-point data types, but uses purely sample based algorithms. This model should have the same algorithmic performance as the MATLAB golden reference since we are not reducing the precision of any calculation. Since Simulink is an extension of MATLAB, we can utilize features like the MATLAB Function Block, shared workspaces, and mirrored block support in both tools to easy the development of this model. This first model accomplishes the first two steps in our requires for HDL code generation, it gets the design into Simulink and makes the design sample based. For consistency to our requirements, the Simulink model should also be integrated into the unit testing harness developed in the algorithm development stage.

Once the implemented algorithm model is complete, a second model is then created from this floating-point model, which instead uses fixed-point data types and blocks which support HDL code generation. Simulink provide several tools to aide in this process include: Fixed-Point Designer, the Fixed-Point Tool, fixed-point specific blocks of common signal processing units, and automated brute-force conversion methods. However, since there will be a precision reduction in the mathematical operation it is important to again validate this design continuously against the generated testing suite. At the completion of this process the design is capable of generating HDL code.

Alternatively, it is possible to utilize Xilinx SysGen [9] or Intel DSP Builder [10] to implement these fixed-point designs. These tools do offer higher performance in some cases of the generated code for their respective FPGA vendors. However, standard Simulink blocks cannot be directly interleaved in each case. SysGen is more flexible than DSP Builder, but a design must be segmented into a SysGen portion and non-SysGen portion for valid generation of HDL. DSP Builder only allows blocks sourced from its own library, no external Simulink or third party blocks.

B. Prototyping

In the third phase, prototyping, the design is deployed to a validated evaluation platform. These will typically be a SoM or even FPGA evaluation kit with attached FMC cards. The purpose of this stage is to validate the design is operating correctly on a deployed system. To simplify this process, Analog Devices provide baseline reference designs, generated IP can be deployed into. This provides a starting point for algorithm designers to focus on their IP rather than worry about interfacing with a transceiver or converter. MathWork's hardware support packages (HSP) automate the insertion of IP into reference designs, so algorithm engineers can almost ignore HDL compiler tools.

MathWorks tools particularly shine here since they provides various mechanisms to investigate a deploy design. These include the following features:

- FPGA-in-the-loop (FIL) uses MATLAB or Simulink to generate stimuli for a deployed IP, which is standalone. This is useful for inspecting a specific function of a specific block on hardware. However, external ADCs or DACs are not used during these simulations.
- External Mode is a Simulink only feature which provides direct register access and low speed data streaming off the FPGA. This tool is useful for tuning parameters in real-time as the design is running with real data coming from an external converter or transceiver.
- FPGA-Capture is a tool that provides functionality similar to ChipScope or SignalTap for signal capture. However, blocks can be directly dropped anywhere into a model at compile time for inspection at runtime. This is useful for traditional FPGA timing diagram debug, but directly provides data back into MATLAB or Simulink. Alternatively, FPGA-Capture can be used to aide in packet error rate (PER) testing, where trigger at connected to CRC or equivalent error signals. Allowing for debugging of rare events of error during this intensive testing. This allows PER testing to be done on the hardware, which can take many order of magnitude less time than in simulation.
- IIO itself can be utilize to collect data since libIIO [11] provide remote procedure call (RPC) functionality over USB and Ethernet interfaces.

Again, this testing should be integrated or related to the testing framework utilized in the previous stages of the design. Maintaining the same level of verification to the desired requirements of a design.

C. Deployment

In the final stage, deployment, the design is moved from prototyping hardware to field ready hardware. In the case of FMC cards, many implementors would move to a fully custom board. However, a SoM can be simply migrated from a debug or evaluation carrier board, to an application specific carrier. In the case of the SoM, the developed design can be directly migrated to the deployment platform. This is true since the FPGA, transceiver, and connectivity between them does not change. Figure 4 is an example of a SoM development cycle from prototyping to field deployment. The device on the bottom of Figure 4 is a PackRF reference platform, which can be field deployed.

At this point the design can be handed off to an FPGA integration engineer who has a fully validated signal chain meeting requirements set out in the initial phase of the design. Since the design process has this consistent validation there is complete traceability, even from Simulink blocks to generated code. Therefore, if a bug exists in the code it can be validated and tested at any stage in the design.

This process also doesn't require valuable FPGA engineer's time to debug a signal chain. Additionally, the FPGA engineer can spend more resources on optimizing the infrastructure of the system level design to provide optimized interfaces with the algorithmic IP and more flexible debugging signal paths to ease the integration and verification of the algorithmic IP in the production design.

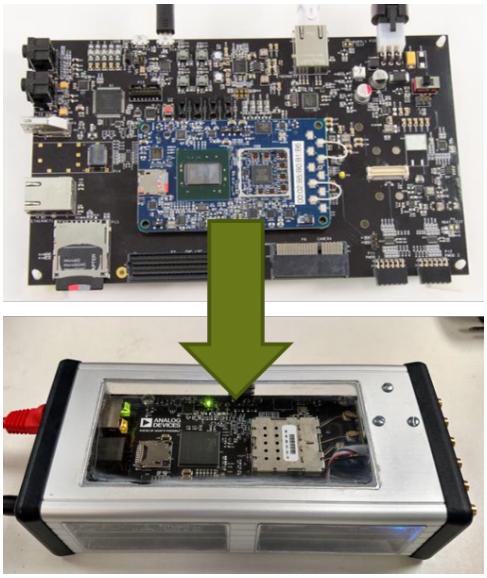

Fig. 4. Example of ADRV9371-Z7035 SoM on prototyping carrier (Top) and deployed in custom enclosure and carrier (Bottom). Example of simplified hardware deployment path from prototyping to a field ready system.

IV. Experimental Setup

Using the flow described previously we implemented an example design of a full stack software defined modem targeting the Analog Devices PackRF platform. Relying on the Analog Devices Linux distribution [12] as an operating system, on open source software and on standard interfaces allowed the design to easily and rapidly integrate with other applications. The outcome of the project was a practical implementation of a wireless point to point link between two radios, with applications in UAV video transmission, high speed wireless data links, internet of things and cognitive radio.

Since the software defined modem design covers all the layers in the OSI network stack, the set of design and implementation tasks was divided with focus on these layers. The physical (PHY) and data link (MAC) layers of the modem were implemented using the design flow described in the previous sections and are based on a QPSK frequency-division duplexing (FDD) communication scheme with two nodes, each node being able to receive and send data at the same time. The network layer was implemented in software based on the TUN/TAP virtual network driver [13] while the upper layers of the stack were left to the default implementation in the Linux operating system. For the physical and data link layers a pure HDL implementation was chosen. All the HDL code was generated from the Simulink models using the MathWorks HDL Coder, resulting into an IP block providing essentially packet based interfaces to the upper layers. As seen in Figure 5, for the final integration into the system HDL design, the PHY and MAC IP needs to connect to a number of other blocks in the design such as the AD9361 transceiver IP and the Rx and Tx DMAs that are being used to transfer the data

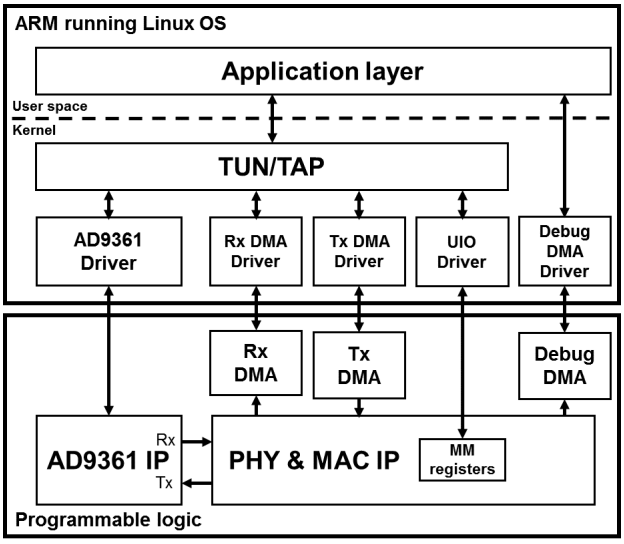

Fig. 5. System block diagram of software and HDL components, as well as their interfaces to the AD9361 controller IP. Showing connections between the Xilinx Zynq SoC ARM and FPGA logic.

to and from the system memory. An additional DMA is used for debug signals allowing large volumes of real time debug data from the IP to be captured and processed in order to track any functionality issues.

Usually, the task of integrating the IP generated from the Simulink model with the rest of the HDL design is a manual process where the algorithm engineer hands it to an FPGA engineer to add into the design. By making use of the PackRF Board Support Package (BSP) for the MathWorks Workflow Advisor [14], the integration process is fully automated by the MathWorks tools enabling the algorithm engineer to generate the entire HDL design to be deployed onto the hardware platform. Figure 6 depicts the traditional integration process where theres a lot of back and forth between teams leading to interface definition mismatch probability and prone to errors between steps.

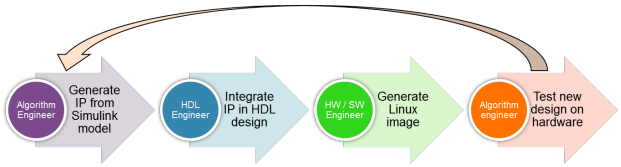

Fig. 6. Traditional design flow

Figure 7 shows the automated process where the same person does all the steps staying in the same flow as for the previous designs. This leads to less errors and a reduced integration time.

The result of the process described before is an IP that handles the physical and data link layers, but, to get to a fully functional system, the rest of the stack needs to be implemented as well. In this particular case the other layers of the stack have all been implemented in software taking advantage of the infrastructure provided by an embedded Linux operating system running on the processing system of the Xilinx Zynq

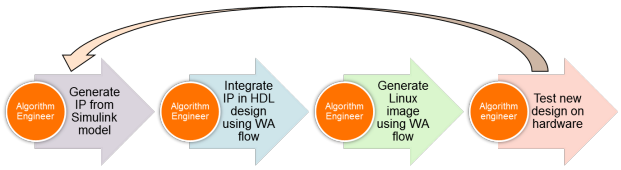

Fig. 7. Integrated design flow

SoC. Depending on the application, there can be different higher level protocols running on top of the data link layer. For this case the decision was to go with TCP/IP and expose the modem as an Ethernet interface to the rest of the system. To do this a TUN/TAP interface was used, which is a standard way to expose a custom communication device in Linux, providing all the other applications access to the communications device through an industry standard Ethernet interface. The choices to use Linux as an operating system and to expose the modem through a standard Ethernet interface moved the design into an industry standard space since embedded Linux is seeing an increasingly wider adoption in industry and any application is able to interface with an Ethernet interface for data transfer.

Fig. 8. Experimental setup

For system verification two Analog Devices PackRFs were used to stream H.264 encoded video via a wireless link from a USB camera attached to one of the devices and display the received video on the screen of the other device. The modem IP was proven capable to sustain a direct peer to peer link at a data rate of 5 MSPS with forward error correction and channel equalization.

V. Conclusions

The design flow presented in Section IV is not perfect but it has three main advantages. First, by staying in the same tools set this directly connects the development stages and makes fixing bugs, which is inevitable in any design, a systematic and traceable process. By maintaining the same testing framework

across these design stages it allows consistency across the implementations and reduces oversights in the design. However, this assumes the built tests have the necessary coverage over all the required features of a design.

Second, this design flow with use of BSPs clearly segments work for different team members with differing skills sets, and more importantly allows them to work together more seamlessly. This is primarily of the result of the BSP forcing strict API boundaries on a design, and allows both FPGA and algorithms developers to work in the same tools with the same assumptions and perspectives on a system.

Finally, by taking advantage of SoMs and vendor authored references designs, developers can start from a functional design and focus on their specific implementation. Rather than wasting valuable time validating connectivity to hardware components and data correctness.

Overall this workflow:

- Provides early access to hardware and real-world signals to algorithms engineers.
- Enables consistent algorithm validation on prototyping and production hardware.
- Reduces design time and missed requirements by test driven implementations.
- Allows engineers can focus on areas in their domain knowledge.

ACKNOWLEDGMENT

The authors would like to thank the Analog Devices Systems Development Group for their support in developing the example design.

REFERENCES

[1] AspenCore, "2017 Embedded Markets Study." [Online]. Available: https://m.eet.com/media/1246048/2017-embedded-market-study.pdf

[2] R. G. Machado and A. M. Wyglinski, "Software-defined radio: Bridging the analogdigital divide," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 409–423, March 2015.

[3] E. Blossom, "Gnu radio: Tools for exploring the radio frequency spectrum," *Linux J.*, vol. 2004, no. 122, pp. 4–, Jun. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=993247.993251

[4] G. F. Zaki, W. Plishker, T. Oshea, N. McCarthy, C. Clancy, E. Blossom, and S. S. Bhattacharyya, "Integration of dataflow optimization techniques into a software radio design framework," in *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, Nov 2009, pp. 243–247.

[5] M. Braun and J. Pendlum, "A flexible data processing framework for heterogeneous processing environments: Rf network-on-chip," in *2017 International Conference on FPGA Reconfiguration for General-Purpose Computing (FPGA4GPC)*, May 2017, pp. 1–6.

[6] The MathWorks, Inc., "Testing Frameworks." [Online]. Available: https://www.mathworks.com/help/matlab/matlab-unit-test-framework.html

[7] Jonathan Cameron, "Industrial I/O Kernel Subsystem." [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/gregkh/staging.git/tree/drivers/staging/iio/Documentation

[8] The MathWorks, Inc., "Generate VHDL and Verilog code for FPGA and ASIC designs." [Online]. Available: https://www.mathworks.com/products/hdl-coder.html

[9] Xilinx, Inc., "System Generator for DSP." [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html

[10] Intel, Corp., "DSP Builder for Intel FPGAs." [Online]. Available: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/dsp-builder.html

[11] Analog Devices, Inc., "What is libiio?" [Online]. Available: https://wiki.analog.com/resources/tools-software/linux-software/libiio

[12] ——, "Analog Devices Linux Distribution." [Online]. Available: https://github.com/analogdevicesinc/linux

[13] Maxim Krasnyansky, "Universal TUN/TAP device driver." [Online]. Available: https://www.kernel.org/doc/Documentation/networking/tuntap.txt

[14] Analog Devices, Inc., "Analog Devices Inc. Board Support Packages." [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/67530-analog-devices-inc-board-support-packages?s_tid=srchtitle

Travis F. Collins Travis holds BS, MS, and PhD degrees in Electrical and Computer Engineering from WPI. He joined ADI in April 2017 in the System Development Group where he focuses on transceiver products and complete signal chain workflows. Travis expertise includes digital signal processing, communications theory, radar, and general high performance compute for software defined radio applications.

Andrei Cozma is an engineering manager at Analog Devices, supporting the design and development of system level reference designs. He holds BS, MS, and PhD degrees in Electrical and Computer Engineering. Andrei has been involved in the design and development of projects from different industry fields such as motor control, industrial automation, software defined radio and instrumentation.