Modeling Considerations for the Hardware-Software Co-design of Flexible Modern Wireless Transceivers

Benjamin Drozdenko, Matthew Zimmermann, Tuan Dao, Kaushik Chowdhury, and Miriam Leeser
School of Electrical and Computer Engineering, Northeastern University, Boston, Massachusetts 02115
Email: bdrozdenko@coe.neu.edu

***Abstract*—Software-defined radios have introduced new platforms for dynamically modifying wireless system designs, and heterogeneous computing has opened up implementing such designs on different computing elements. Our goal is to develop a modeling environment that captures reusability of various processing blocks at the physical layer for several modern protocols, and makes decisions regarding whether each processing block should be part of reconfigurable hardware or embedded processor software. Our approach creates several different MathWorks Simulink model variants for both the transmitter and the receiver, each with a different boundary between hardware and software components. Using the 802.11a standard as an example, we use these models to generate a bitstream for the FPGA and executable code for the ARM processor on a Xilinx Zynq system-on-chip. Our results collect such metrics as data path delay, resource utilization, and power usage and demonstrate how to further enhance the SDR design.**

I. Introduction

Target hardware for the next generation of Software Defined Radios (SDRs) includes a mix of CPU and FPGA components. In this paradigm, a complex protocol can be broken down into functional blocks, with their execution distributed across the available hardware resources. A barrier associated with designing real-time wireless transceivers is the need for an environment in which both hardware (HW) and software (SW) components in the processing chain can be effectively modeled. Such an environment must enable tunable control of radio parameters and have a means of measuring execution time and energy usage on both HW and SW. In this paper, we introduce our modeling environment, which uses commercially available tools, MathWorks Simulink and Xilinx Vivado. We demonstrate our approach using IEEE 802.11a transmitter (Tx) and receiver (Rx) Simulink models and ensure correctness by comparing against Annex G of the 802.11a specification [1]. These models require modification to best target execution on HW or SW. We demonstrate our models on the Xilinx Zynq SoC, but our high-level designs can be ported to different SoC HW. We generate HDL code and IP core blocks for the components targeted for execution in HW, and also generate C code to be compiled into an executable that runs on the ARM processor. We present information on timing, resource utilization, and power consumption for each of the different HW/SW co-designs. Our approach assumes a *HW-SW divide point* to limit communications between FPGA and embedded ARM processor. We set parameters to control the frame size, sample time for real-time execution on FPGA, and frame time for execution on CPU. The data types and sizes required for HW-SW data transfer differ in each model variant.

Other researchers have developed high-level SDR frameworks that automatically generate low-level implementations. ATOMIX builds applications on wireless infrastructure by defining *atoms* as fixed-time computations, but is intended only for synthesis on a variety of DSPs [2]. CODIPHY can use automatic C and VHDL code generation to build an 802.11a/g transceiver, but its testing is all emulated, not done on live hardware devices [3]. Airblue introduces an FPGA-based SDR platform for the PHY and MAC layers, adopting certain techniques such as data-driven control and annotated streaming of data samples, but this platform cannot be used for studying HW-SW co-design tradeoffs [4].

Our approach advances the state of the art by providing (1) a modeling environment for prototyping the HW-SW divide point, (2) a library of wireless processing blocks implemented in both HW and SW, and (3) an approach that encourages reuse of blocks among different protocols. It provides a platform on which we can prototype MIMO and Wi-Fi/LTE coexistence. This approach is superior to a component-based one because it allows developers to analyze the processing chain as a whole and supports variations in data types for adaptability.

II. HW-SW Prototyping Platform

Our *HW-SW prototyping platform* (Fig. 1) consists of an RF front end (ADI FMComms3), a Xilinx Zynq-based system, and a host computer. It supports the following capabilities: (1) an RF front end that enables wireless transception, (2) a HW platform that combines FPGA and embedded processor, (3) an efficient bus for transfer between HW and SW, and (4) a user interface for specifying algorithms, automatic creation of HDL and C code, compilation into processor-specific executables, and generation of a compatible FPGA bitstream. The Xilinx Zynq SoC has an embedded ARM processor and FPGA fabric, called a Processing System (PS) and Programmable Logic (PL) in Xilinx terminology. We target two different Zynq boards, the ZC706 with a Z7045 chip and the Zedboard with the less capable Z-7020. Internal to the Zynq SoC, an AXI bus connects the PL and the PS. The Zynq boards connect via Ethernet to a host computer, which we use to send start and stop signals to the Zynq PS. The 802.11a transceiver system contains both a Transmit path, from PS to PL to FMComms3, and a Receive path, from FMComms3 to PL to PS.

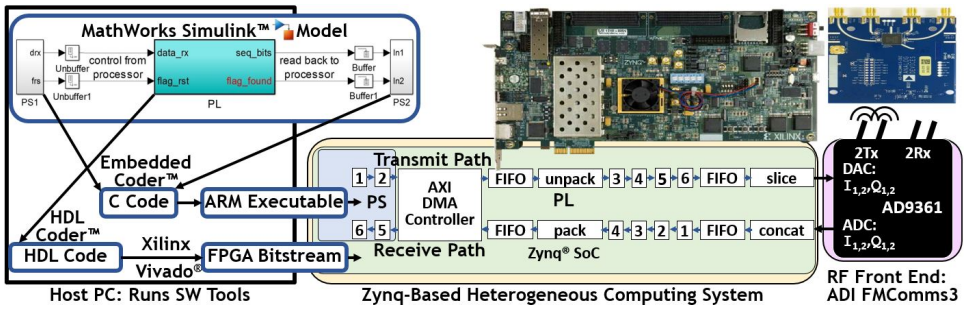


Fig. 1: Prototyping Platform Hardware Components, Software Tools, and Interface

TABLE I: Processing Blocks for Tx Path and Rx Path

Block	Transmit Path	Receive Path
1	Scramble	Preamble Detect
2	Convolutional Encode	OFDM Demodulate
3	Block Interleave	BPSK Demodulate
4	BPSK Modulate	Block Deinterleave
5	OFDM Modulate	Viterbi Decode
6	Preamble Switch	Descramble

The host computer runs the SW tools used to create models. We use MathWorks Simulink to create and simulate synchronous dataflow models. We implement 12 processing blocks, 6 for Tx path and 6 for Rx path, as listed in Table I. We use the MathWorks toolboxes HDL Coder and Embedded Coder to target the PL and PS, respectively. Additional MathWorks HW support packages allow us to interface with the Zynq and the RF front end. Each Simulink model captures all the information about the Zynq transceiver system. The model distinguishes the subsystem targeted for execution on the PL from those components which are targeted to run on the PS. Two IP Cores encapsulate the PL design, one for Tx and one for Rx. A Xilinx Vivado block diagram is generated to combine the IP cores with all the AXI interface components. Vivado synthesizes, implements, and makes a bitstream for the PL. We use MathWorks Embedded Coder support for Zynq to automatically generate C code and compile the executable that targets the PS. When we press the start button on a generated Simulink model on the host, it sends a signal via Ethernet to launch the executable on the Zynq.

III. HW-SW Modeling Environment

Our approach uses commercial tools and can work with multiple HW platforms. Our *HW-SW joint modeling environment* has the following abilities.

1) HW/SW Functional Equivalence: For a set of inputs, each processing block produces the same outputs in the HW implementation and the SW implementation.

2) HW/SW Mapping: Each design variant models the HW-SW divide point for wireless behaviors, mapping each processing block in sequence to either HW or SW.

3) HW/SW Interfacing: Each design variant prepares and translates a fixed number of data bits to between HW and SW as needed for real-time processing.

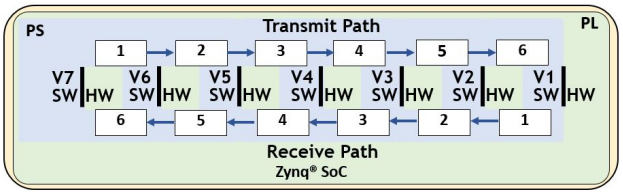


Fig. 2: Modeling HW-SW Divide for Wireless Behaviors

4) Adaptability & Reusability: Each processing block is adaptable depending on the needs of the entire processing chain while remaining reusable and applicable to different wireless standards.

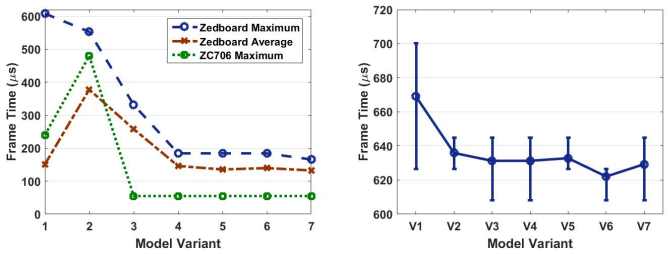
Our main contributions are a library of HW and SW implementations for 802.11a processing blocks, a library of interfacing units for data transfer via AXI, and detailed timing considerations. As a first decision in the modeling process, we minimize the communication by permitting only one interface between HW and SW. We represent alternate HW-SW divide points using 14 model variants, 7 for the Tx path and 7 for the Rx path, as shown in Fig. 2. V1 implements all functionality in SW. Since the RF board connects directly to the FPGA, for each subsequent version we add to HW the processing block that is next closest to the RF board. Thus, V2 adds F6 to HW for the transmit path and F1 for the receive path. This continues to V7, in which all processing blocks are implemented in HW.

Internally, communications on the Zynq chip use an AXI interconnect, which can transfer 32-bit words in a time-synchronous manner between PL and PS. There are two AXI interfaces which we use: AXI-lite for the Rx models and AXI-stream for the Tx models. To support the AXI-stream interface in the Tx models, the Vivado block diagram must contain additional IP Cores such as AXI Direct Memory Access (DMA) Controller, as shown in Fig. 1.

For the purposes of data validation, we run our experiments in an *offline* mode, in which data intended for the RF front end is routed back to the Zynq PS for storage in a file. This method allows us to verify each model variant produces the same output. For each model variant, moving the HW-SW divide line changes the requirements for transferring data between PS and PL. The size and number of elements that must be transferred for each model variant are listed in Table II.

TABLE II: HW-SW Data Transfer for Tx

Variant	Data to Send	Data Type	Size of 1	#Elem
V1	Samples	Signed Fixed Point	16 bits	80
V2	Samples	Signed Fixed Point	16 bits	64
V3	Symbols	Signed Integer	1-8 bits	64
V4	Coded Bits	Boolean	1 bit	48
V5	Coded Bits	Boolean	1 bit	48
V6	Data Bits	Boolean	1 bit	24
V7	Data Bits	Boolean	1 bit	24



(a) Tx on Zedboard (b) Rx on ZC706

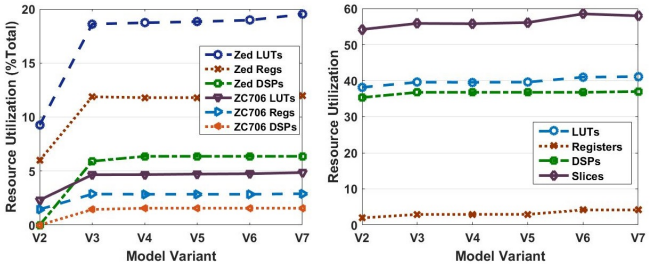
Fig. 3: PS Execution Times per Frame

IV. Experimental Results

For the 802.11a Tx, the execution timing results on the PS are shown in Fig. 3a. Looking at the whole processing chain, it is clear that moving one processing block from SW to HW does not necessarily cause speedup. The increase in Tx frame time on ZC706 from V1 to V2 is proof. Since V1 is a SW-only framework, it requires no AXI communication and saves some time compared to V2, which adds only one small component to HW. The time saved from implementing it in HW is much less than the time spent on the PS-PL data transfer. After V2, the maximum PS frame time decreases as more components are moved onto the PL. The IFFT is the biggest bottleneck in the Tx model, and moving it to PL in V3 results in the largest drop in frame time. For the Rx, the execution timing results on the PS are shown in Fig. 3b. Similar to the Tx, the Rx maximum PS frame time decreases as more components are moved onto the PL. Preamble detection is the biggest bottleneck in the Rx model; moving it to PL in V2 results in the largest drop in frame time. However, there are also significant drops when the FFT is moved in V3 and the Viterbi Decoder is moved in V6. Notably, moving the Descrambler component to PL in V7 does not show a decrease in frame time, suggesting that it may be better placed in SW. For an idea of how long the same operations take to process on the PL, we look for the maximum data path delay of the Tx and Rx, which are shown in Table III. At under 320 ns, the Rx execution time on the PL is much faster than any model variant on the PS. For the Rx to keep up with the Tx, the sum of data path delay and the time to transfer one sample from PL via AXI, $t_{s,AXI}$, must be less than the PL step time. In our trials, since the data path delay is significantly less than the PL sample time, then the Rx is able to operate at the Tx rate. If $t_{s,AXI}$ remains less than 680 ns, which it does, then the Rx can match the Tx.

TABLE III: Data Path Delay & Power Consumption

	Data Path Delay		Power Consumption	
	Tx (ns)	Rx (ns)	Tx (W)	Rx (W)
V1	n/a	n/a	1.530	1.566
V2	11.11	313.70	1.819	2.343
V3	16.17	317.43	1.840	2.354
V4	18.33	311.14	1.845	2.111
V5	15.84	313.12	1.844	2.106
V6	16.52	307.73	1.847	2.111
V7	16.04	318.89	1.842	2.115



(a) Tx on Zedboard (b) Rx on ZC706

Fig. 4: PL Resource Utilization by Model Variant

The Tx resource utilization on the Zedboard is shown in Fig. 4a. These results show increasing lookup table (LUT), register, and digital signal processor (DSP) usage as more components are put onto the PL. The number of registers decreases slightly from V2 to V3 due to the different data types involved. The slice registers hold state information that reduces because V2 must transfer data in 32-bit sample form, while V3 holds data in single-bit form. V2 must hold each sample in complex, 16-bit fixed-point format before initiating IFFT processing, and 64 data samples make up a frame. In all model versions, even the PL-only variant, the FPGA is at less than 5% utilization on the ZC706 and 20% on the Zedboard, meaning that many LUTs and registers are available for design enhancements. The Rx resource utilization on the ZC706 is shown in Fig. 4b. Unlike the Tx, the Rx did not fit on the Zedboard, so we had to use the ZC706. Like the Tx, these Rx utilization results show increasing LUT, register, and DSP usage as more components are put onto PL. The largest Rx increase comes from the initial placement of preamble detection on the PL in V2. The Rx uses a significant portion of the FPGA resources, with as much as 60% of the total slices, the main grouping of logic resources. Still, there remain many LUTs and registers available for the next stages of our designs.

In addition to meeting timing and resource requirements, we also want power-efficient designs. The Zynq PS has an embedded ARM processor that uses less power than alternatives like the x86 on the host PC. Since the Zynq platform always provides power to the ARM processor, using the FPGA fabric adds to the overall power usage, even though the FPGA fabric is more power efficient. The FPGA power consumption is directly related to the SoC chip area and resource utilization. The power results were found by running the Vivado power report with fixed environmental settings (e.g.

TABLE IV: Wireless Standard Components Comparison

	802.11a	802.11g	LTE
Scrambling	**(1)**	**(1)**	**(1)**
Convolutional Coding			
1/2 Rate	**(1)**	**(1)**	**(1)**
2/3 Rate	(2)	(2)	(2)
3/4 Rate	(2)	(2)	
Digital Modulation			
BPSK	**(1)**	(D)	
QPSK	(2)	(D)	(2)
16-QAM	(2)		(2)
64-QAM	(2)		(2)
Block Interleaving	**(1)**	**(1)**	**(1)**
OFDM/MA	**(1)**	**(1)**	(DL)
IFFT Size	**64**	**64**	128-2048
Cyclic Prefix (μs)	**0.8**	**0.8**	4.69-33.33
Preamble Detection	**(1)**	(2)	(2)

(1) Implemented & Reusable, (2) Not Yet Implemented, but Reusable

output load 5 pF, ambient temperature 25 °C). The Tx and Rx power consumption on the Zedboard and ZC706, respectively, are shown in Table III. The Tx total power increases from 1.819 to 1.847 W as more components are placed on the PL. However, this small increase of 28 mW is small when compared to the Tx PS consumption, which alone is 1.53 W on the Zedboard. Also as expected, the Rx power increases as preamble detection and FFT are added to the PL. Since each version of our Tx and Rx designs puts more processing onto the FPGA, we would expect monotonic increases in the overall power usage. However, we see a significant decrease when BPSK is placed on the PL in V4. The reason for this drop is mainly due to the data type change from samples to coded bits for data transferred over AXI. Whereas V3 transfers 64 32-bit fixed point samples from PL to PS, V4 must only transfer 48 bits via 2 32-bit integers, reducing the load on AXI by a factor of 32. From V4 to V7, the power increase is 4 mW, which is minor compared to the Rx PS usage of 1.566 W.

V. Discussion

A major benefit of our flexible SDR testbed is the ability to reuse components for alternate 802.11 and mobile standards. A comparison of the protocol settings in 802.11a, 802.11g (for Wi-Fi) and LTE (for mobile phones) standards is given in Table IV. The functional blocks of our 802.11a implementation, especially scrambling and interleaving, can be reused in a number of different standards. However, some modifications would need to be made to support different convolutional encoding rates besides 1/2 and digital modulation schemes besides BPSK. In addition, OFDM requires different IFFT sizes and cyclic prefix lengths, as well as flexible subcarrier allotments to form OFDMA ('MA' for *multiple access*) used in the downlink channel for LTE. This reusability allows us to explore LTE and Wi-Fi coexistence on the same channel, TV whitespace reuse, and switching between different standards. National Instruments also has a testbed for real-time LTE/Wi-Fi coexistence [5], but our research is unique in that it studies coexistence in the context of HW-SW co-design. Considering LTE, larger amounts of control flow exist compared to 802.11. Such control logic may be best placed on the PS. This would require even more PS-PL communication to administer functional changes, introducing *multiple* HW-SW divide points. In this case, while streaming data is best suited for AXI-stream, we may reserve AXI-lite channels for infrequent control messages. Having prototyped on the Zynq, our modeling environment can next be tested on alternate SoC devices. For example, the Altera Arria 10® ARM-based SoC offers performance improvement and power reduction. The Zynq UltraScale+ Multi-Processing SoC (MPSoC) architecture, which is designed for applications such as wireless, has both a Cortex-A and a Cortex-R real-time processor that could improve the SDR's ability to adhere to specification times.

VI. Conclusions

We have demonstrated a method for automating HW-SW co-designs for modern wireless transceivers. Our modeling environment enables profiling of all processing blocks to identify bottlenecks such as preamble detection. The environment explores various HW-SW divide points, and identifies which model variants are most desirable. It details the interfacing necessary at the divide point, and shows when variants consume more time and power as a result of heavy data transfer. While power consumption generally increases as more components are placed on programmable logic, the amount is negligible compared to the embedded ARM.

In the future, we plan to perform tests with online radio transmissions and measure error rates for the different co-designs. This 802.11a PHY layer implementation will be used as a basis for future work in MIMO, higher layers (e.g. MAC), and LTE coexistence. We plan to test our modeling environment on other SoC platforms. Our modeling environment shows that automation of HW-SW co-designs is possible. As a future extension, we intend to automatically decide the HW-SW divide point and bundling of data for transfer, given a user-specified wireless processing chain.

Acknowledgment

The authors would like to thank Analog Devices, MathWorks, and Xilinx for their support and donations.

References

[1] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band*, IEEE Std. 802.11a-1999, 1999.

[2] M. Bansal *et al.*, "Atomix: A Framework for Deploying Signal Processing Applications on Wireless Infrastructure," in *USENIX Symposium on Networked Systems Design and Implementation, NSDI*, 2015.

[3] A. Dutta *et al.*, "CODIPHY: Composing On-demand Intelligent Physical Layers," in *Workshop on Software Radio Implementation Forum*, 2013.

[4] M. C. Ng, K. E. Fleming *et al.*, "Airblue: a System for Cross-layer Wireless Protocol Development," in *Symposium on Architecture for Networking and Communications Systems, ANCS*, 2010.

[5] National Instruments, Inc. (2016) Real-time LTE/Wi-Fi Coexistence Testbed. [Online]. Available: http://www.ni.com/white-paper/53044/en/