Synthesizing Efficient Hardware from High-Level Functional Hardware Description Languages

Mahshid Shahmohammadian, Geoffrey Mainland
Department of Computer Science
Drexel University
{ms4323,mainland}@drexel.edu

***Abstract*—Functional hardware description languages (FHDL) provide powerful tools for building new abstractions that enable sophisticated hardware system to be built from the composition of smaller, reusable parts. Raising the level of abstractions in hardware designs means the programmer can focus on high-level circuit structure rather than mundane low-level details. The language features that facilitate this include high-order functions, rich static type system with type inference, and parametric polymorphism. We use hand-written structural and behavioral VHDL, Simulink, and the Kansas Lava FHDL to re-implement several components taken from a Simulink model of orthogonal frequency-division multiplexing (OFDM) physical layer (PHY). Our development demonstrates that an FHDL can require fewer lines of code without sacrificing performance.**

***Index Terms*—circuit generation, functional hardware description languages, high-level hardware design, field-programmable gate array**

I. Introduction

The current standard tools for low-level hardware designs are VHDL and Verilog. They provide concurrent description of circuits, multipurpose modules, a strongly typed language (in the case of VHDL), and the ability to model as well as simulate a circuit before translating the design into real hardware in the synthesis process. Simulink provides a much higher-level "box and wires" view of hardware designs. However, Simulink components cannot be parameterized over signal types, and writing reusable modules is difficult. Functional hardware description languages (FHDLs) have a long history, beginning with μFP [1], which was based on Backus's FP [2] language. These languages emphasize building complex designs from core primitives and combining forms; their power derives in part from the abstractions they provide for defining new combining forms.

Hardware implementations written using FHDLs typically require writing fewer lines of code than low-level languages like VHDL and Verilog, and testing is easier—in Kansas Lava, the FHDL we use in this paper, all FHDL programs are also valid Haskell programs, and programs compute identical results whether synthesized or run in a Haskell environment. Because Haskell is pure (and lazy), programmers can apply powerful reasoning techniques, like equational reasoning, directly to Kansas Lava programs. Standard functional language features like polymorphism, type inference, and static type-checking allow modular programming and catch common programmer errors at compile time.

We demonstrate these benefits be re-implementing several components of a Simulink model of orthogonal frequency-division multiplexing (OFDM) physical layer (PHY) using Kansas Lava and VHDL. Our Kansas Lava implementations are significantly shorter without sacrificing performance.

The work described in this paper was supported by the National Science Foundation through grant CCF-1717088.

II. Background and Motivation

The benefits of the functional style of programming apply not only to the software domain, but also to hardware design. Backus [2] argued for these benefits, encouraging programmers to focus on building new abstractions and reasoning about programs algebraically. Functional languages facilitate this style or programming by providing a core language with a flexible set of combining forms for constructing larger programs from smaller programs with clear algebraic properties. In μFP [1], a FHDL based on FP, both the behavior and layout of a hardware circuit are described using primitive functions and combining forms. This leads to a circuit description that is concise and that obeys identifiable algebraic laws.

Embedding a FHDL in a functional language leverages the feature of the host language and avoids having to write a new language implementation from scratch. Lava [3], a FHDL embedded in the functional language Haskell, supports both hardware simulation and verification. Using Haskell's rich type system simplifies hardware descriptions. Other Haskell features, like combinators, type classes, and Monads are also leveraged in Lava—combinators enable the composition of smaller circuits to construct larger circuits, and Monads allow the interpretation of a single circuit description as either a Haskell program or a VHDL generator.

Sheeran [4] has reviewed many useful FHDLs and argued that hardware designers may not be aware of the power of the features unique to FHDLs, such as type inference, polymorphism, higher-order functions, and type classes. This is likely because FHDL designers focus on the language features, whereas hardware designers are concerned primarily with performance. In this paper, we are concerned with performance and show that one can often get the best of both worlds.

Examples of generating "clever circuits" to bring low-level circuit information such as circuit delays and wiring into higher levels at the time of specification are presented in [5] and [6], respectively.

We focus on the Kansas Lava [7] FHDL, which is a domain specific language (DSL) embedded in Haskell. Kansas Lava is the most recent incarnation of Lava [3], itself a spiritual successor to μFP. Rather than creating a language and compiler from scratch, embedding a language in Haskell provides all the benefits of the host language, such as access to its rich type system. Higher-order functions simplify the creation of reusable patterns for combining circuits. Because Haskell is lazy and pure, programmers can use equational reasoning to reason about correctness. Parametric polymorphism simplifies abstracting designs over attributes like signal width. Kansas Lava uses Haskell's sized types to enforce signal compatibility at compile time. As well as supporting VHDL generation, a Kansas Lava program is also a valid Haskell program, which makes it easy to test a design before synthesis.

III. Overview of SOFDM

The subject of the re-implementations in this paper is SOFDM, a Simulink model of OFDM physical layer, developed by Drexel's Wireless System Laboratory [8]. Fig. 1 shows a complete overview of an OFDM PHY pipeline.

Re-implementation of SOFDM Components

The re-implemented components include:

- Modulation mapping which, given a modulation, maps bit sequences to I/Q constellation points.
- Data interleaving, which permutes transmitted data to enhance robustness to noise.
- Carrier frequency offset (CFO) estimation for synchronization
- Schmidl-Cox algorithm for frequency and timing recovery

All components were re-implemented in hand-written VHDL (structural and behavioral), Simulink, and Kansas Lava. The correctness of the re-implemented components was confirmed by checking for input/output equivalence with the original implementation using random input test vectors.

IV. Evaluation

Components developed using all three approaches were synthesized to a Virtex 7 FPGA. Performance metrics such as area utilization, clock frequency, throughput, and power consumption were measured for each re-implementation effort. The rest of this section provide the re-implementation performance results for the selected components from the SOFDM model.

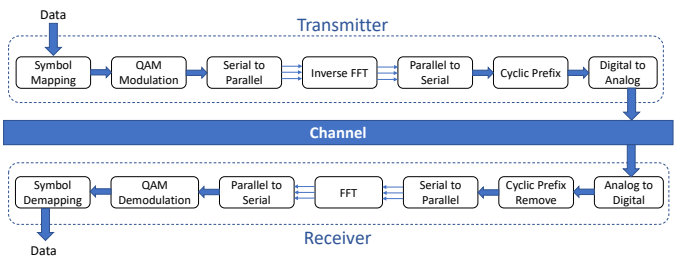

Figure 1. Architecture of transmitter and receiver chain in OFDM

Area Utilization

Table I presents the cell usage of selected components for different re-implementations.

Modulation mapping block mostly contains multiplexers and look-up tables (LUT), memory and digital signal processing (DSP) cell usage are negligible. Data interleaving uses memory units and multiple counter blocks, so the synthesis area utilization reports show a better comparison of FPGA slice usage between different approaches. The carrier frequency offset (CFO) estimation component is a complex design, presenting more of a challenge for a FHDL. This block utilizes DSP logic, both memory and LUTs, and required the use of the COordinate Rotation DIgital Computer (CORDIC) algorithm. The Schmidl-Cox synchronization block also used of DSP and memory.

Implementations of the Schmidl-Cox algorithm [9, 10] in different approaches, helps the receiver in OFDM chain locate the beginning of the transmitted signal for synchronization.

The configurable logic block (CLB) in 7 series FPGAs provides high-performance FPGA logic as various input LUTs, distributed memory along with shift registers, carry logic for arithmetic purposes, and multiplexers. A flip-flop (FF) can register each 5-input LUT output. A slice consists of four 6-input LUTs along with their eight associated flip-flops as well as multiplexers and carry logic, then two slices form a CLB. These CLBs are the primary logic resources for combinational and sequential circuit designs [11].

Table I shows that LUT, FF, and memory unit usage is lowest for Kansas Lava and hand-written behavioral VHDL. In the structural VHDL implementation, which mimics the structure from the Simulink model, the CLB usage is quite similar to Simulink's. The Kansas Lava implementation utilizes far fewer slices than the other re-implementations. It is significant that the VHDL generated from Kansas Lava for the data interleaving block manages to produce Block RAM (BRAM) memory units instead of LUTs and shift registers. This required no special programmer effort.

Power Consumption

Total power consumption of the circuit consists of static and dynamic power. Static power consumption relates to leakage of current in the transistors built in the FPGA. Dynamic power consumption is related to capacitors of the circuit being charged and discharged often. The power consumption results in Table I show the device static and dynamic power consumption for each step after Virtex 7 FPGA VC707 synthesis, implementation, and bit stream generation. Static and dynamic power consumption for different approaches are provided in Table I. The total on-chip power is listed in Table II for better comparison.

The ambient temperature for each step has been measured after bit stream generation to help ensure an accurate static device power consumption. The VHDL generated by Kansas Lava has the smallest total on-chip power for all components.

Table I
CELL USAGE ON FPGA, POWER CONSUMPTION AS STATIC AND DYNAMIC POWER (mW), AND TIMING REPORTS AS SETUP AND HOLD SLACK TIMES (ns) OF SELECTED COMPONENTS FOR DIFFERENT RE-IMPLEMENTATIONS

	Cell Usage						Power		Slack Time	
	I/O	LUT	Memory	DSP	FF	Carry	Static	Dynamic	Setup	Hold
Modulation Mapping										
System Generator	38	27	0	0	0	3	338	26	0.647	2.571
Structural VHDL	39	25	0	0	21	0	331	19	0.748	0.968
Behavioral VHDL	39	25	0	0	5	0	322	15	0.857	1.042
Kansas Lava	39	13	0	0	13	0	299	24	0.812	1.760
Data Interleaving										
System Generator	11	52	4	0	59	17	337	7	0.428	0.103
Structural VHDL	11	72	2	0	48	3	337	7	0.751	0.120
Behavioral VHDL	11	86	24	0	24	4	289	6	0.660	0.170
Kansas Lava	11	68	1	0	39	2	278	7	0.340	0.137
CFO Estimation										
System Generator	175	3409	487	0	3840	364	304	151	0	0.085
Behavioral VHDL	162	2871	0	0	123	432	350	40	0.544	0.020
Kansas Lava	130	1441	0	0	145	349	347	24	0.565	0.330
Schmidl-Cox										
System Generator	297	4498	1913	52	1525	922	326	81	1.997	0.090
Behavioral VHDL	229	3286	17	44	1553	644	327	44	3.529	0.099
Kansas Lava	223	4202	1	61	1285	1018	326	35	0.387	0.159

Timing and Clock Frequency

Setup time in timing reports from synthesis is the minimum time in which the data signal must be stable before the next clock edge happens in order for the data to be reliably sampled by the clock. Hold time is the minimum time in which the data signal must be stable after the next clock edge happens in order for the data to be sampled reliably [12].

Setup and hold slack times as timing reports of selected components in different re-implementations are presented in Table I. The maximum clock frequency is presented in Table II as well. The observed maximum clock frequency of the VHDL generated from Kansas Lava is comparable to other approaches. This timing information is also used to calculate circuit throughput.

Throughput and Throughput over Area

The rate in which the number of bits of output is produced in maximum frequency (or in minimum clock period) is the maximum throughput of the circuit. Based on the clock frequency (or clock cycle period) from the timing reports for each step we can calculate throughput as follows:

$$\text{Number of transmitting bits} \times \text{Maximum clock frequency}$$

The throughput of each re-implementation is given in Table II.

Another useful metric to provide a trade-off between throughput and area utilization is throughput over area that is calculated by dividing the maximum throughput of the component by the number of CLBs used on FPGA for each re-implementation. This performance metric is optimized by targeting maximum throughout along with minimum numbers of slices on board.

The number of CLBs are calculated by the fact that in 7 series FPGAs every CLB contains 8 look-up tables, 16 flip flops and 2 slices [11]. The number of utilized slices in Table II are taken from utilization reports produced during synthesis.

Generated VHDL from Kansas Lava shows comparing throughput results to other re-implementations of different components. Since area utilization from Kansas Lava is the most efficient one, throughput over area for this approach seem quite promising.

Effort of Programming

Finally, programming effort is compared in Table II using lines of code (LOC) for different approaches. The Kansas Lava implementations are the shortest. In our experience, they are also more concise and easier to understand than hand-written VHDL. Kansas Lava programs contain type signature of every circuit function that may not be included as the lines of codes and can be inferred by the compiler.

V. CONCLUSION AND DISCUSSION

High-level FHDLs can assist the hardware developer in designing circuits in a more concise manner by providing a higher level of abstraction without sacrificing performance.

This paper provides a case study of such FHDL called Kansas Lava to re-implement a Simulink model of OFDM PHY layer [8]. Multiple approaches Simulink, structural and hand-written VHDL and Kansas Lava have been presented along with associated performance reports from Virtex 7 FPGA implementations.

Given the results described, we conclude:

- The Kansas Lava FHDL reduces programming effort.
- Kansas Lava makes efficient use of FPGA resources.

Table II
NUMBER OF FPGA SLICES, MAXIMUM CLK FREQUENCY (MHz), TOTAL ON-CHIP POWER CONSUMPTION (mW), THROUGHPUT (Mbps), THROUGHPUT OVER AREA (Mbps/CLB), AND NUMBER OF LINES OF CODE (LOC) FROM THE SELECTED COMPONENTS FOR DIFFERENT RE-IMPLEMENTATIONS

	Slices	Max. Freq.	Total Power	Throughput	Throughput/Area	LOC
Modulation Mapping						
System Generator	12	125	364	2000	333.3	-
Structural VHDL	6	91	350	1456	485.3	128
Behavioral VHDL	10	77	337	1232	246.4	99
Kansas Lava	8	167	323	2672	668	59
Data Interleaving						
System Generator	25	83	344	332	25.5	-
Structural VHDL	28	83	344	332	23.7	336
Behavioral VHDL	31	100	295	400	25	101
Kansas Lava	19	87	285	348	34.8	42
CFO Estimation						
System Generator	1132	29	455	986	1.7	-
Behavioral VHDL	581	30	390	960	3.3	122
Kansas Lava	464	24	371	768	3.3	123
Schmidl-Cox						
System Generator	1680	26	407	416	0.5	-
Behavioral VHDL	975	22	371	484	0.9	132
Kansas Lava	1580	14	361	224	0.3	101

- Dynamic and on-chip power consumption as well as throughput and clock frequency of all Kansas Lava implementations are comparable if not superior to the other implementations.

VI. FUTURE WORK

The ultimate goal of this work is to implement a full OFDM pipeline in a FHDL. As well as being an interesting extension of the present case study, this would enable the comparison of various implementations of a full OFDM pipeline instead of component-by-component comparisons. One possible objection to the current study is that it does not demonstrate the ability of a FHDL to scale to larger designs; the implementation of a full OFDM pipeline would address such a concern.

We are working on extending Ziria [13] so that it can serve as a FHDL. Ziria, a domain-specific language for wireless physical layer protocols, like OFDM, currently supports compilation to efficient CPU code. The work described in this paper is part of an effort to add a VHDL back-end to Ziria. Because it performs aggressive whole-program optimizations [14], we expect that it can provide much better performance than other FHDLs for larger designs.

REFERENCES

[1] M. Sheeran, "μFP, a Language for VLSI Design," in *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, ser. LFP '84. New York, NY, USA: ACM, 1984, pp. 104–112. [Online]. Available: http://doi.acm.org/10.1145/800055.802026

[2] J. Backus, "Can Programming Be Liberated from the von Neumann Style?: A Functional Style and Its Algebra of Programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978.

[3] P. Bjesse, K. Claessen, M. Sheeran, and S. Singh, "Lava: Hardware Design in Haskell," in *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, ser. ICFP '98. New York, NY, USA: ACM, 1998, pp. 174–184. [Online]. Available: http://doi.acm.org/10.1145/289423.289440

[4] M. Sheeran, "Hardware Design and Functional Programming: a Perfect Match." *J. UCS*, vol. 11, pp. 1135–1158, Jan. 2005.

[5] ——, "Generating Fast Multipliers Using Clever Circuits," in *Formal Methods in Computer-Aided Design*, A. J. Hu and A. K. Martin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 6–20.

[6] E. Axelsson, K. Claessen, and M. Sheeran, "Wired: Wire-Aware Circuit Design," in *Correct Hardware Design and Verification Methods*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Oct. 2005, pp. 5–19.

[7] A. Gill, T. Bull, G. Kimmell, E. Perrins, E. Komp, and B. Werling, "Introducing Kansas Lava," in *Implementation and Application of Functional Languages*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Sep. 2009, pp. 18–35.

[8] J. Chacko, C. Sahin, D. Nguyen, D. Pfeil, N. Kandasamy, and K. Dandekar, "FPGA-based latency-insensitive OFDM pipeline for wireless research," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2014, pp. 1–6.

[9] T. M. Schmidl and D. C. Cox, "Robust frequency and timing synchronization for OFDM," *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1613–1621, Dec. 1997.

[10] P. Morris, "Hardware design and implementation of the Schmidl-Cox synchronization algorithm for an OFDM transceiver," 2015.

[11] "7 Series FPGAs Configurable Logic Block User Guide (UG474)," p. 74, 2016.

[12] V. Expert, ""Setup and Hold Time" : Static Timing Analysis (STA) basic (Part 3a)." [Online]. Available: http://www.vlsi-expert.com/2011/04/static-timing-analysis-sta-basic-part3a.html

[13] G. Stewart, M. Gowda, G. Mainland, B. Radunovic, D. Vytiniotis, and C. L. Agulló, "Ziria: A DSL for Wireless Systems Programming," in *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*, Istanbul, Turkey, Mar. 2015, pp. 415–428.

[14] G. Mainland, "Better Living through Operational Semantics: An Optimizing Compiler for Radio Protocols," *Proceedings of the ACM on Programming Languages*, vol. 1, no. 1, pp. 19:1–19:26, Sep. 2017.