Base Station MIMO Detector Algorithm Implementations

Tuomo Hänninen, Hamid Yadegar Amin and Markku Juntti
Centre for Wireless Communications
P.O. Box 4500, FI-90014 University of Oulu, Finland
Email: tuomo.hanninen@oulu.fi

***Abstract*—In this paper, we implement a high throughput multiple-input multiple-output (MIMO) detector for single-carrier frequency-division multiple access (SC-FDMA) base station. High-level Synthesis (HLS) tool is used for implementing the algorithm on Xilinx Virtex and Zynq FPGAs.**

First, we compare the throughput performance and power consumption results of the different implementations. Second, we evaluate the quality of the results by comparing the HLS results to handwritten register-transfer level (RTL) implementations. In conclusion, the HLS tools have evolved into applicable implementation tools. Furthermore, the possible slight losses in the performance or design complexity with the HLS design method could be counteracted by choosing a higher category FPGA.

I. Introduction

3GPP release 8 and beyond wireless cellular systems [1] use single-carrier frequency-division multiple access (SC-FDMA) as the uplink transmission scheme instead of orthogonal frequency division multiplexing (OFDM) [2]. The reduced peak-to-average power ratio of single-carrier transmission reduces the mobile transmitter cost by allowing cheaper power amplifiers [3]. Multiple-input multiple-output (MIMO) antenna configuration [4], [5] is used in these systems to increase the peak data rates. Similar to any spatial multiplexing based MIMO transmission, a spatial equalizer is required in the receiver to cancel the inter-antenna interference (IAI). Both linear and nonlinear receiver structures have been considered for MIMO receivers with an emphasis on ones operating in OFDM systems, wherein inter-symbol interference (ISI) is not a problem. In single-carrier transmission an equalizer is needed in the receiver to cancel also the ISI. Different variants of *sphere detector* (SD) [6], which calculate the maximum likelihood (ML) solution with reduced complexity, have received a lot of attention in the literature. For systems employing forward error control (FEC) coding, the soft output *list sphere detector* (LSD) [7] is needed. In [8], we proposed a two stage receiver structure *Frequency domain linear MMSE filter with sphere detection* for SC-FDMA uplink transmission. The K-best list LSD algorithm [9] was considered for our receiver. The receiver structure and sphere detector are illustrated in Figure 1 [8].

In [10], we implemented the K-best LSD algorithm on a 40nm Xilinx Virtex 6 field-programmable gate array (FPGA) using an early version of Xilinx Vivado High-level Synthesis (HLS) tool to get a good understanding of its complexity. In [11], we implemented the same MIMO detector algorithm on 28 nm Virtex 7, 20 nm Virtex UltraScale and 16 nm Virtex UltraScale+ FPGA using the latest version Vivado HLS tool to evaluate both the evolution of HLS tools and the evolution of FPGAs. The HLS tools enabled the implementation and comparison of different macroarchitectures in a relatively short time. The architecture optimization was done in the C language level, which gave a clear benefit in terms of design time and effort.

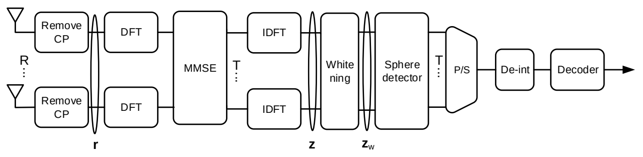


Fig. 1. SC-FDMA receiver with sphere detection.

In this paper, we extend the implementations to cover two different Xilinx Zynq FPGAs. The Zynq architecture differs from previous FPGAs by integrating a complete ARM Cortex-A9 processor. The Zynq SoC integrates the software programmability of a processor with the hardware programmability of an FPGA. This enables hardware acceleration while integrating central processing unit (CPU), digital signal processor (DSP) and mixed signal functionality on a single device. Additionally, we continue the HLS tool evaluation by comparing the HLS results to conventional handwritten RTL design approach.

The rest of this paper is organized as follows. In Section II, introduces the HLS methodology. Section III describes the principles of the sphere detector algorithm used in the implementations. Section IV, compares the throughput performance and power consumption results of the Virtex and Zynq FPGA implementations. Section V evaluates the quality of the results by comparing the HLS results to handwritten register-transfer level (RTL) implementations. The summary and final conclusions are given in Section VI.

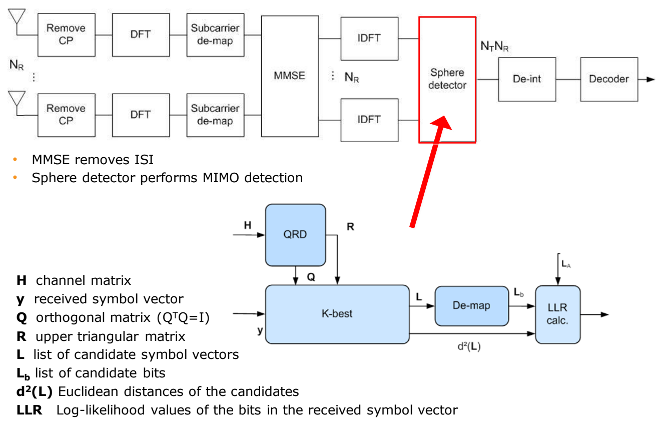
II. High Level Synthesis

HLS design methodology provides an automated path to generate optimized RTL code directly from algorithms written in e.g. C++. Algorithms are developed and verified in C++ using integer and fixed-point bit-accurate data types. HLS is used to generate RTL code from those specifications. The generated RTL code is verified using the C++ algorithm and testbench using the test infrastructure created by the HLS tool. The main goal of using the HLS methodology is to improve productivity by shortening the time to create hardware and to reduce the errors introduced by manual refinement thereby shortening the verification time.

Creating optimized hardware requires integer and fixed-point arithmetic that has been optimized to the application. Native C++ integer types as well as C++ bit-accurate integer and fixed-point data types are supported for synthesis. The designer drives the synthesis process using directives in a highly interactive environment that includes analysis tools that provide feedback on the architecture being generated by the tool. Required directives specify the target technology and the clock period. Optional directives provide the required control to create a wide variety of architectures. For example, some directives control the C++ interface mapping into hardware interfaces, hardware hierarchy and communication between the hierarchy block. Some directives provide control over the architecture and parallelism of each of the blocks. Additional directives also allow fine control over the schedule, the type and number of hardware resources, enabling of power optimizations and many other architectural aspects of the generated hardware. Interface synthesis makes it possible to map the transfer of data that is implied by passing of C++ function arguments to various hardware interfaces such as wires, registers, memories, buses, or more complex user-defined interfaces. All the necessary signals and timing constraints are generated during the synthesis process so that the generated RTL code conforms and is optimized to the desired interfaces.

III. K-best LSD algorithm

The K-best-LSD algorithm [8,9] is a breadth-first type of tree search algorithm, i.e., the search proceeds one layer at a time in the search tree by extending the partial candidates with admissible nodes and calculating the PEDs. The K-best-LSD algorithm keeps always a constant number of candidates in each layer of the tree if no sphere radius constraint is introduced. Therefore the algorithm has a fixed complexity, which is a desirable in implementations. However, the algorithm does not necessarily find the candidates with lowest Euclidean distances (EDs). The K-best LSD algorithm search with K=4 for real-valued 2×2 64-QAM system is illustrated in 3. In this paper, K-best LSD with list size of 8 is used for the implementations. In real-valued 4×4 64-QAM scenario this means there are eight layers and eight PED calculations on each layer. The receiver structure and sphere detector processing blocks considered in this paper are illustrated in Figure 2 [11]. In this paper, we consider two different processing blocks. The more complex algorithm is the K-best LSD tree search algorithm illustrated in Figure 3. The less complex algorithm, used, e.g., for the handwritten RTL vs HLS implementation method comparison, is the log-likelihood ratio (LLR) calculation block. The LLR processing block calculates the soft-output information via max-log-MAP approximation. The LSD algorithm gives a list L_b of candidates, which include the Euclidean distance $d^2(L)$ and the corresponding symbol vector.



Remove CP
DFT
Subcarrier de-map
MMSE
IDFT
Sphere detector
N_T N_R
De-int
Decoder
N_R
MMSE removes ISI
Sphere detector performs MIMO detection
H
QRD
R
Q
K-best
L
De-map
L_b
L_A
LLR calc.
y
d²(L)
H channel matrix
y received symbol vector
Q orthogonal matrix (QᵀQ=I)
R upper triangular matrix
L list of candidate symbol vectors
L_b list of candidate bits
d²(L) Euclidean distances of the candidates
LLR Log-likelihood values of the bits in the received symbol vector

Fig. 2. Receiver with sphere detection.

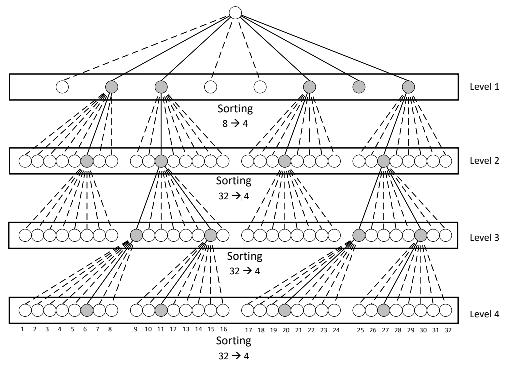


Fig. 3. The K-best LSD tree search.

IV. Implementations on programmable logic

In addition 28 nm 20 nm Virtex 7 FPGA implementations [11] , the K-best LSD MIMO detector was implemented on the 28 nm and 20 nm Zynq FPGAs to get good understanding of the technology influence on the results. Table I describes the implementation results for maximum throughput designs.

Each of the implementations achieve the target throughput for LTE SC-FDMA 4×4 64-QAM uplink with a conventional K-best architecture. As it is seen in Table. I, both the Zynq FPGAs schedule with higher throughput than the Virtex FPGAs. As an instance, the 28 nm Zynq implementation has shown to achieve 20% increase in throughput in comparison to 28 nm Virtex 7 implementation. With 20 nm technology, the performance difference is even higher.

In order to emphasize e.g. the throughput or the energy efficiency of the implementation, Xilinx Vivado provides implementation strategies [12]. Considering the requirement,

TABLE I
K-BEST LSD IMPLEMENTATION - MAXIMUM THROUGHPUT

	Virtex 7	Virtex Ultra-Scale	Zynq	Zynq Ultra-Scale
Technology	28nm	20nm	28nm	20nm
LUT	81445	71960	87896	70982
FF	80742	52930	118501	47089
Freq [MHz]	167	172	200	238
Throughput [Mbps]	501	516	600	714
Dynamic power [mW]	1760	840	1400	1330
Energy [nJ/bit]	3.5	1.6	2.3	1.8

software tries different optimization methods in placing and routing. Table II, presents performance of four different implementation strategies for Virtex 7 which all achieved the maximum throughput of 501 Mbps. The strategy A (PerformanceRefinePlacement), increases the placer effort in the post-placement optimization phase and disables timing relaxation in the router. The strategy B (PerformanceNetDelaylow), compensates for the optimistic delay estimation and adds extra delay cost to distance and high fanout connections. The strategy C (AreaExplore), uses multiple optimization algorithms to get potentially fewer LUTs. The strategy D (Flow run post rout opt), enables physical optimizations in post implementation phase including routing. The result show that in addition to achieve the highest throughput, strategy D manages to perform a better performance in terms of power. Thus, the strategy D results for Virtex 7 were exploited in the power consumption comparison.

Similar iterative approach was exploited to find the lowest power consumption for Virtex UltraScale, Zynq and Zynq UltraScale FPGAs. Table III demonstrate the power consumption of the implementations with the throughputs normalized to 501 Mbps. In addition to capability of achieving a higher throughput, the more advanced FPGAs have shown a better power performance as well.

TABLE II
VIVADO HLS 2017 IMPLEMENTATION STRATEGIES

Strategy	A	B	C	D
Technology	28nm	28nm	28nm	28nm
LUT	81450	81446	83105	81445
FF	80742	80742	80742	80742
DSP	565	565	565	565
Freq (MHz)	167	167	167	167
Throughput(Mbps)	501	501	501	501
Power(mW)	1770	1800	2270	1760
Energy(nJ/bit)	3.53	3.60	4.50	3.5

TABLE III
K-BEST LSD IMPLEMENTATION - POWER CONSUMPTION WITH NORMALIZED THROUGHPUT

	Virtex 7	Virtex Ultra-Scale	Zynq	Zynq Ultra-Scale
Technology	28nm	20nm	28nm	20nm
LUT	81445	71922	87809	68004
FF	80742	52930	118445	47089
Freq [MHz]	167	167	167	167
Throughput [Mbps]	501	501	501	501
Dynamic power [mW]	1760	840	1140	950
Energy [nJ/bit]	3.5	1.7	2.2	2.0

V. HANDWRITTEN RTL VS HLS RESULTS

The main benefit of the HLS design method is the reduced time to create the hardware compared to manually generated RTL. Two additional LLR implementation comparisons were carried out to study this implementation aspect. In the first comparison, the design effort for the handwritten RTL was not limited. Significantly more time was spent to create an optimal manual reference design compared to HLS implementation. The implementations were targeted on ASIC which does not require any vendor specific DSP exploitation optimization. The synthesis results for the LLR processing block are summarized in Table IV. The HLS tool implementation of the LLR block is approximately 9% larger compared to the handwritten implementation and the difference in power consumption is 18.5%. The results are somewhat expected for a large design, where the compiler is not able to extract all necessary information from a high level language. The design of both implementations required several iterations until a sufficient performance was achieved. However, the workload per iteration differed significantly between the handwritten RTL and the HLS tool for the benefit of the HLS tool. In addition, finding an optimal tradeoff between the area and latency by changing the pipelining structure and the level of parallelism was much faster with the HLS tool. The same is true for changing the design frequency.

In the second comparison, the same design is implemented on an FPGA using Vivaldo HLS tool and target technology independent behavioral VHDL. In this case, the design effort for the C source code and handwritten RTL were more comparable. In both design methods, there are various optimization options for the synthesize and implementation phase. In HLS method, the optimization tools are presented by directives such as pipelining and, in case of VHDL, optimization tools are provided by various synthesize and implementation strategies. Targeting Virtex 7 as the hardware and benefiting all optimization tools, both designs are accomplished. As shown in Table V, the three different HLS implementations schedule with much higher frequency. With the same throughput, the HLS implementation has approximately 51% lower complexity and 90% lower power consumption compared to the handwritten implementation. Once the VHDL code is provided, synthesis tool considers its architecture as optimal and does not effectively contribute to architecture optimizations process.

However, HLS tries various architectures in gate level process to achieve the optimal solution. The results suggest that using the RTL for describing the design, the RTL needs to be completely optimized for the target technology to achieve decent results. C source code and HLS tool design method enables to write target independent behavioral description of the algorithm and later optimize the design for different technologies.

TABLE IV
LLR IMPLEMENTATION - OPTIMIZED HANDWRITTEN RTL VS HLS TOOL

Design method	Optimized VHDL	Catapult C
Technology	180nm CMOS	180nm CMOS
Complexity [kGE]	15.5	16.9
Frequency [MHz]	150	150
Throughput [Mbps]	121	121
Power [mW]	27	33

,

TABLE V
LLR IMPLEMENTATATION - BEHAVIORAL RTL VS HLS TOOL

Design method	Behavioral VHDL	Vivado HLS		
		I	II	III
Technology	Virtex 7	Virtex 7	Virtex 7	Virtex 7
FF	47	180	117	86
LUT	561	310	243	211
Complexity [Slices]	608	490	360	297
Frequency [MHz]	76	217	230	320
Throughput [Mbps]	76	217	115	80
Power [mW]	30	12	3	2

VI. CONCLUSION

Different target FPGAs were compared when implementing MIMO detector processing blocks for SC-FDMA uplink receiver. Furthermore, HLS design method was evaluated by comparing implementation results to conventional design method.

Table VI summarizes the achievable gains in the implementation process. Higher throughput performance with large designs was achieved with the Zynq FPGAs compared to Virtex FPGAs. Similarly, 20 nm technology enabled 17% - 52% lower power consumption compared to 28 nm technology within the same FPGA family. The fully optimized handwritten RTL implementation had only 5% higher throughput and 20% lower power consumption compared to HLS implementation. However, if target independent behavioral VHDL was used, which reduces the design time significantly, the HLS implementation achieved better results. Therefore, HLS design method should be at least seriously considered as an alternative to traditional and more time consuming design method.

In conclusion, the HLS tools have evolved into practical implementation tools. One of the most significant evolution steps is that the modern tools can synthesize even high complex wireless communication algorithms in relatively sort time. Therefore, they enable the iterative design approach required by the HLS method. Furthermore, smaller silicon technology can be exploited to achieve better results without modifying the algorithm or source code. Thus, while choosing between algorithms with different performance-complexity ratio, it should be noted that few tens of percentage difference in performance or complexity falls within the variation of FPGA technology or design method selection. Choosing a higher category FPGA has higher gain than choosing a manual handwritten RTL design approach. Therefore, the possible slight losses in the performance or design complexity with the HLS design method could be counteracted by a higher category FPGA.

TABLE VI
SUMMARY OF THE ACHIEVABLE GAINS

	Performance	Power consumption
Zynq vs Virtex (28 nm)	+20%	-35%
Zynq vs Virtex (20 nm)	+38%	+13%
20 nm vs 28 nm (Virtex)	+3%	-52%
20 nm vs 28 nm (Zynq)	+19%	-17%
Fully optimized RTL vs HLS	+5%	-20%
HLS vs Behavioral RTL	-42%	-90%

REFERENCES

[1] 3rd Generation Partnership Project (3GPP), "Physical layer procedures," 3GPP TS 36.213 V11.0.0, Tech. Rep., 2012.

[2] Z. Zvonar, "Multiuser detection in asynchronous CDMA frequency-selective fading channels," *Wireless Personal Communications*, vol. 2, no. 4, pp. 373–392, 1995.

[3] D. Falconer, S. Ariyavisitakul, A. Benyamin-Seeyar, and B. Eidson, "Frequency domain equalization for single-carrier broadband wireless systems," *IEEE Communications Magazine*, vol. 40, no. 4, pp. 58–66, Apr. 2002.

[4] E. Telatar, "Capacity of multi-antenna Gaussian channels," *European Transactions on Telecommunications*, vol. 10, no. 6, pp. 585–595, Nov. 1999.

[5] D. Gesbert, M. Shafi, D. Shiu, P. J. Smith, and A. Naguib, "From theory to practice: An overview of MIMO space-time coded wireless systems," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 3, pp. 281–302, Apr. 2003.

[6] M. O. Damen, H. E. Gamal, and G. Caire, "On maximum–likelihood detection and the search for the closest lattice point," *IEEE Transactions on Information Theory*, vol. 49, no. 10, pp. 2389–2402, Oct. 2003.

[7] B. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Communications Letters*, vol. 51, no. 3, pp. 389–399, Mar. 2003.

[8] J. Ketonen, J. Karjalainen, M. Juntti, and T. Hänninen, "MIMO detection in single carrier systems," in *Proceedings of the 19th European Signal Processing Conference*, Aug. 2011.

[9] K. Wong, C. Tsui, R. K. Cheng, and W. Mow, "A VLSI architecture of a K-best lattice decoding algorithm for MIMO channels," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 2002.

[10] T. Hänninen, J. Janhunen, and M. Juntti, "Novel detector implementations achieving 3G LTE downlink and uplink requirements," in *SDR-WInnComm*, Jan. 2013.

[11] T. Hänninen, M. S. Saud, H. Y. Amin, and M. Juntti, "MIMO detector implementations using high-level synthesis tools from different generations," in *Asilomar Conference on Signals, Systems, and Computers*, Nov. 2017.

[12] Xilinx, "Vivado design suite user guide implementation," UG904 v2013.4, Tech. Rep., 2013.