Transport Triggered Polar Decoders

Mathieu Léonardon*†, Camille Leroux*, Pekka Jääskeläinen‡, Christophe Jégo* and Yvon Savaria†
*Univ. Bordeaux, Bordeaux INP, IMS Lab, UMR CNRS 5218, France
†École Polytechnique de Montréal, QC, Canada
‡Tampere University of Technology, Tampere, Finland

***Abstract*—In this paper, the first transport triggered architecture (TTA) customized for the decoding of polar codes is proposed. A first version of this programmable processor is optimized for the successive cancellation (SC) decoding of polar codes while a second architecture is further specialized to also support Soft CANcellation (SCAN) decoding. Both architectures were fully validated on FPGA device by prototyping. The first architecture was also synthesized in 28nm ASIC technology. It runs at a frequency of 800 MHz and reaches a throughput of 352 Mbps for a (1024, 512) polar code decoded with the SC algorithm. Compared to previous work, the energy consumption is reduced by one order of magnitude (0.14 nJ / bit) and the throughput is increased fivefold. Compared to an optimized software implementation on a general purpose processor (x86 architecture), the throughput is 37 % higher and the energy consumption is two orders of magnitude lower. TTA can be seen as a way to reduce the gap between programmable and dedicated polar decoders.**

I. Introduction

Invented a decade ago [1], polar codes are included in the next generation of wireless communication standard (5G). This singular family of channel codes will be used for the enhanced mobile broadband control channel [2]. The polar codes are also considered for other 5G scenarios. Shortly after their invention, the first dedicated hardware decoders were proposed [3], [4]. Such polar decoders, implemented on ASIC or FPGA device, provide a set of solutions with high data rate, low latency and low energy consumption. On the other side of the design space, some works proposed to implement polar decoding algorithms on programmable architectures (x86, ARM, GPU) [5], [6]. These so-called *software polar decoders* take advantage of the parallelism available on recent high-end processors to reach high throughput with a great deal of flexibility at the price of an increased energy consumption.

Between these two design approaches, Application Specific Instruction-set Processors (ASIP) are usually seen as a way to find a better compromise between the high efficiency of dedicated architectures and the high flexibility of programmable architectures. An ASIP is usually defined as a *specialized programmable processor*, in the sense that it includes some hardware units and instructions dedicated to a family of algorithms with unneeded components removed to save implementation resources.

A first approach is to give the designer the ability to configure and extend a conventional Reduced Instruction Set Computer (RISC) processor. This is the approach proposed by the Cadence Tensilica toolset. Such an ASIP was designed in [7] for the successive cancellation (SC) decoding of polar codes. Despite some customizations, the processor keeps its general purpose programmability. This first study showed some significant improvements in terms of energy consumption and throughput in comparison with current programmable processors (x86, ARM). It also showed that the bottleneck for further improvements in throughput was lying in the core architectural model. Using the RISC programming model typically implies that most of the data accesses pass through the register file. In most cases, the loading, processing and storage of data take 3 instructions while in a dedicated architecture, these memory accesses can be anticipated and masked. This 3 steps process is particularly problematic for channel decoding algorithms because they require a large number of memory transactions.

In this work, we focus on an alternative customizable programmable architecture template: the Transport Triggered Architecture (TTA). An ASIP based on the TTA approach is designed and customized for the decoding of polar codes while retaining a general purpose instruction set. A TTA processor is composed of several Functional Units (FU) connected by a set of configurable buses [8]. The instruction set consists of a single move instruction: in addition to general purpose registers, data can be moved from the output of a FU to the input of any other FU. Among other features, it enables direct data transactions between the memory and the functional units, thus alleviating the cost of memory data transfers. TTA processors have been customized for LDPC [9] and Turbocodes [10] but, to the best of our knowledge, this paper reports the first TTA processor specialized for polar decoding. In order to show the modularity of the TTA model, two transport triggered (TT) architectures were designed, the first one is designed for efficient SC decoding (TT-SC), while the second one also supports Soft CANcellation (SCAN) decoding (TT-SCAN). Both architectures were implemented and validated on a FPGA device. The TT-SC decoder was also synthesized for 28nm FD-SOI CMOS technology, in order to have performance estimations for an ASIC target.

FPGA and ASIC implementation results show that *Transport Triggered Polar Decoders* (TTPD) improve the throughput and the energy efficiency of programmable polar decoders and thus reduce the gap between programmable and dedicated polar decoders. The rest of the paper is organized as follows: Section II details the considered decoding algorithms. Section III presents the proposed TTPD. The implementation results

978-1-5386-7048-4/18/$31.00 ©2018 IEEE

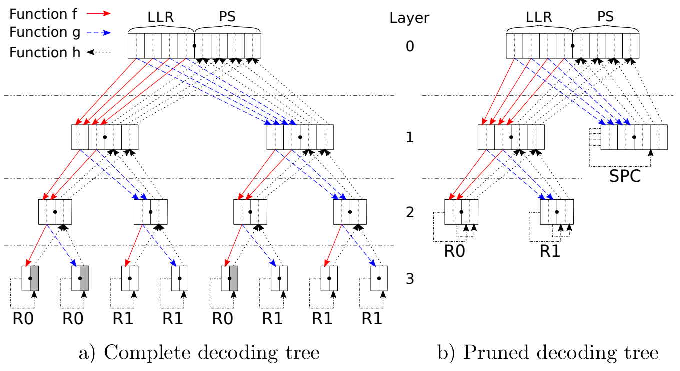

Fig. 1. SC decoding tree

are shown and analyzed in Section IV. Finally, Section V concludes the paper.

II. Polar Decoding

A. Successive Cancellation Decoding

The SC decoding algorithm may be described as the traversal of a binary tree starting from the root node. For a $N = 2^n$ polar code, the tree contains $\log N + 1$ layers. Layers are indexed from $l = 0$ (upper layer) to $l = \log N$ (lower layer). A layer l contains 2^l nodes. Each node contains 2^{n-l} Log-Likelihood Ratios (LLRs), L_i and 2^{n-l} Partial Sums (PSs), s_i. The lowest layer includes $N - K$ frozen bits, represented as gray PSs in Figure 1a, and K information bits. Channel LLRs are stored in the root node. When the decoding is terminated, the decoding result is stored in the root node in the form of PSs. The LLRs of a node are updated when the node is accessed in the descending direction. PSs of a node are updated when the node is accessed in the ascending direction. The LLRs and PSs updating rules are listed in Eq. (1).

$$\begin{cases} f(L_a, L_b) &= sign(L_a \times L_b) \times \min(|L_a|, |L_b|) \\ g(L_a, L_b, s) &= (1-2s)L_a + L_b \\ h(s_a, s_b) &= (s_a \oplus s_b, s_b) \\ \mathrm{R1}(L_a) &= \begin{cases} 0 \text{ if } L_a > 0 \\ 1 \text{ else} \end{cases} \end{cases} \quad (1)$$

In general, when processing a node at layer l, a maximum of 2^{n-l} computations (LLRs or PSs) can be performed in parallel. In other words, the upper layers offer a higher parallelism than the lower layers. Let us define P as the maximum number of computations that a decoder can perform in parallel. Considering a layer l in the tree, as $P \geq 2^{n-l}$ the processing units of the decoder can be fully utilized. This condition is not verified in the lower part of the tree in which some other optimization strategies are necessary in order to speedup the decoding process. One approach is to use unrolled processing units dedicated to the processing of the lower sub-trees [11].

B. Tree Pruning

Figure 1b shows the pruning technique that was proposed in [12] and later improved in [13]. This technique reduces the computational complexity of the SC decoding algorithm. Depending on the *frozen bits* subset, some subtrees can be replaced by a single node. The traversal of these subtrees becomes a specialized function dedicated to the decoding of these nodes. For instance, Rate-0 (R0) and Rate-1 (R1) nodes correspond to nodes holding only frozen bits and only information bits, respectively. Specialized nodes called repetition (REP) and single parity check nodes (SPC) correspond to subtrees representing repetition codes and single parity check codes, respectively. This tree pruning method significantly reduces the number of operations in the algorithm with a negligible impact on the decoding performance.

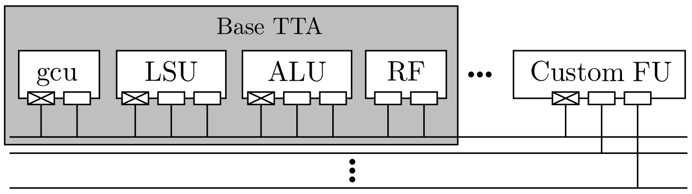

Fig. 2. A base TTA processor with a custom FU.

C. Soft CANcellation algorithm

The SCAN algorithm was introduced in [14]. It is an iterative version of the SC decoding in which PSs are replaced by LLRs. The soft decision nature of SCAN decoding makes it suitable for code concatenation as in [15]. The update rules of SCAN decoding are shown in Eq. (2). In the SCAN algorithm, partial sums are replaced by LLRs. The input variables of the elementary functions are therefore all LLRs (L_a, L_b, L_c).

$$\begin{cases} f_{scan}(L_a, L_b, L_c) &= f(L_a, L_b + L_c) \\ g_{scan}(L_a, L_b, L_c) &= f(L_a, L_c) + L_b \end{cases} \quad (2)$$

III. Transport Triggered Polar Decoders

A. Transport Triggered Architectures

TTA is a modular processor architecture template [8] inspired from the MOVE architecture [16]. A hardware/software development tool chain, called TCE, was designed to ease the implementation of such systems [17]. A TTA is composed of elementary blocks, denoted as function units, connected with a custom number of buses. As shown in Figure 2, the Base TTA consists of a control unit (gcu), that includes the fetch and decode stages, an ALU, a load / store unit (LSU) and a register file (RF). The modularity lies in the possibility to add some customized FUs and to include extra buses interconnecting these FUs.

Building from that very lean foundation, the TTA does not have an explicit instruction set. It simply moves data from the output of an FU to the input of any other FU. All FUs have a particular input port, denoted as its *trigger port*. When a data is written to this trigger port, some processing is performed within the FU. The performed instruction is a side effect of a data transaction. The structure of the TTA is essentially parallel in the sense that several data transactions can happen at the same time. The assembly language expressing the functionality of a TTA system actually consists in specifying, at each clock cycle, the source and destination

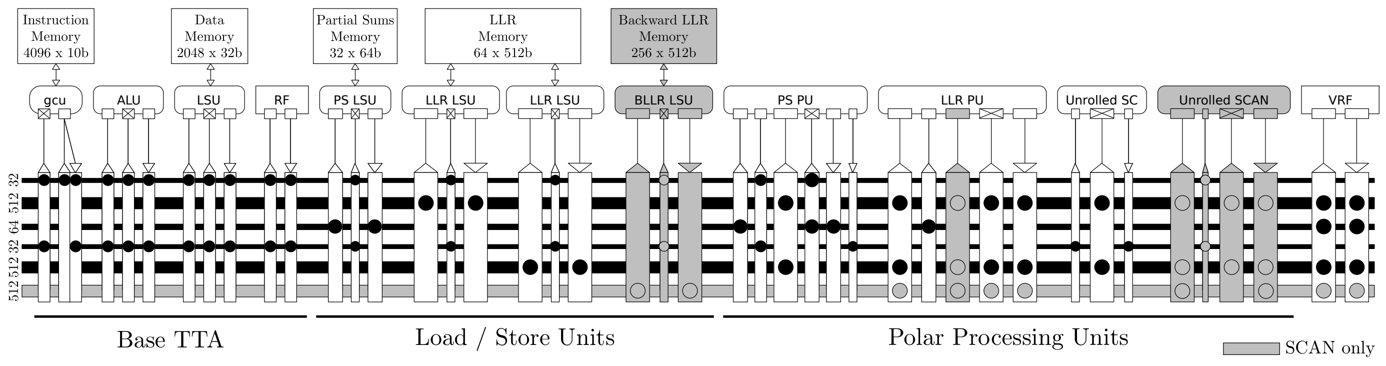

Fig. 3. Transport Triggered Polar Decoder Architecture.

of data transactions performed in parallel: the data path is *exposed* to the programmer.

Despite the great simplicity of the instruction set, programming the TTA assembly would be a tedious task to perform by hand. One would have to specify each data transport at each clock cycle. Fortunately, TCE provides a runtime retargetable compiler. Given an architecture description file, the compiler converts a source code written in a high level description language (C, C++, OpenCL) into an executable file [17]. The efficiency of the compiled program partly stems from the modularity of the LLVM toolset [18]. Front and middle end optimizations are provided by LLVM. Low level optimizations that are specific to TTAs are integrated as a custom TTA code generation library included in TCE.

A complete TTA-based processor, including its custom function units, is generated as generic or optimized RTL (VHDL or Verilog). The resulting customized processor can then be directly synthesized and implemented on either ASIC or FPGA devices. Due to their flexibility and modularity, TTAs have shown to provide very elegant solutions for a broad range of applications.

B. Software optimizations

Assuming the base TTA processor of Figure 2, a C description of a polar decoding algorithm can be readily compiled and executed on the TTA. However, this straightforward implementation is very slow and some software optimizations are required to take advantage of its inherent parallel structure.

As explained in Section II, the pruning technique reduces the amount of computation to be performed during the decoding process. It is thus necessary to determine the type of processing that should be performed at a given stage of the decoding (R0, R1 or UNROLLED). In the main loop of the C source code, this is implemented as a switch case. This conditional jump limits the speedup usually brought by tree pruning. In order to fully benefit from tree pruning, the loop unrolling optimization technique, introduced in [19] for polar codes, was applied. In [20], a library was proposed to produce unrolled source code targeting x86 processors. It was slightly modified to target the TTA architecture described in the following subsection. The drawback of unrolling the code

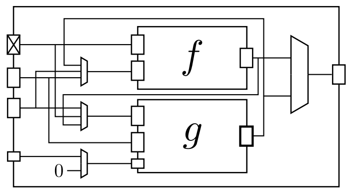

Fig. 4. The LLR Processing Unit supporting SC and SCAN decoding.

is that a specific source code must be compiled for each code rate and frozen bits configuration.

C. Transport Triggered Polar Decoders Architecture

The proposed architecture is depicted in Figure 3. The **Base TTA** allows the proposed architecture to support any algorithm by providing general purpose instructions. It was extended with different kinds of vector custom FUs. A vector size of $P = 64$ was selected, meaning that vector FUs can process a maximum of 64 data (LLR or PSs) in parallel. LLRs are represented as $Q = 8$-bit values in order to guarantee a negligible decoding performance loss. Some **Load / Store Units** provide parallel access to the data memories. The **Polar Processing Units** implement the polar functions that are listed in Eq. 1 and the functions specific to the tree pruning. The **Unrolled SC Unit** is an unrolled SC decoder included to speedup the most sequential part of the decoding algorithms as proposed in [11] and detailed in Section III-C2. Finally, a **Vector Register File** is designed to buffer vector data. The gray-colored blocks are the resources that are only used for the SCAN algorithm. The support of customized SCAN decoding requires an extra 512-bit bus for LLRs, one extra input port in the LLR PU, an LSU for the backward LLRs and a dedicated unrolled SCAN subtree decoder.

1) Load / Store Units: In the proposed architecture, the maximum supported polar code length is $N_{max} = 1024$ which is the size of the largest frozen bit set defined in the 5G standard [2]. This could easily be modified to support larger codes by increasing the size of the memories. The PSs require the storage of N_{max} bits and since it is necessary to access $P = 64$ data in parallel, a 16x64-bit memory is sufficient. However, the system needs some additional addresses, thus the next memory size was selected: 32x64. For similar reasons, the LLR memory is mapped into a 64x512-bit RAM. The backward LLR memory, used in SCAN decoding, requires $QN \log N$ bits and is implemented into a 256x512-bit RAM.

As mentioned in Section I, the efficiency of a channel decoder is strongly determined by memory access latency. The custom LSUs are modified versions of those provided by the TCE. First, the latency of each load and store operation has been reduced from three to one clock cycle by removing some internal registers. It does not affect the critical path that lies in the 32-bit scalar ALU. Then, additional hardware and control resources were added in order to perform unaligned loads and stores. Adding these custom instructions does not affect the unit's latency.

2) Polar Processing Units: The PS PU and the LLR PU perform the specific operations of polar decoding in parallel. One can see them as Single Instruction Multiple Data (SIMD) units dedicated to polar decoder processing. An extended version of the TCE, called TCEMC [21], was chosen to facilitate the SIMD description. Figure 4 shows the architecture of the LLR PU. Since both SCAN and SC algorithms need f and g functions, it is possible to share the associated blocks. It reduces the complexity overhead induced by the support of the SCAN decoding.

In [11], the lower stages of the decoding tree are processed using an unrolled subtree decoder in which registers were added in order to split the critical path. In the proposed TTA processor, two similar unrolled subtree decoders were implemented as FUs. The unrolled SC and SCAN decoders can process 8 LLRs in 6 and 10 clock cycles, respectively. These unrolled subtrees do not limit the flexibility of the decoder as the frozen bit pattern is given as an input. Unlike the 3-phase decoder in [11], unrolled subtree decoders do not include registers but instead use a multi-cycle timing strategy. Unnecessary registers are consequently removed, and the critical path remains in the general purpose ALU. Multi-cycle paths are possible in the TCE, by configuring an FU to have a certain latency.

IV. Implementation Results

It is worth mentioning that a fine grain comparison with other existing decoders is difficult, since many parameters change from an implementation to another, such as technology node, power supply, supported algorithms, tree pruning strategies, quantization, frozen bits location. The purpose of this section is to show that TTA stands as a natural compromise between programmable and dedicated architectures in the case of polar decoding. We also show that compared to our previous work [7], the proposed TTA processor offers a significant improvement in terms of throughput and energy efficiency.

The TT-SC decoder was customized for SC decoding only while the TT-SCAN version includes some extra resources for optimized SC and SCAN decoding.

A. TT−SC Architecture

The Transport Triggered Polar Decoder (TTPD) customized for SC decoding was synthesized with a 0.9V, 125°C ST-28nm FD-SOI standard cells library. The memories were implemented with SRAM blocks using the same technology. Table I compares the TT-SC decoder with state-of-the-art programmable processors implementations using the intra-frame parallelization method. The unrolled software description is provided by the AFF3CT tool chain [24]. The Intel i7-4712HQ processor provides a high throughput solution but with a rather high energy consumption. The ARM implementation reduces the energy consumption while it offers a lower throughput. The LX7 polar SC decoder improves the energy consumption of the ARM without significantly affecting the speed. With a (1024, 512) polar code, the TT-SC decoder achieves a throughput of 352 Mbps, which is 37 % higher than the i7 while the energy consumption of 0.14 nJ / bit is two orders of magnitude lower.

Even if the TT-SC decoder is closer in terms of functionality to the general purpose processors mentioned above, it is interesting to compare it to dedicated architectures. The proposed TT-SC processor was implemented and validated on a Xilinx Artix-7 FPGA. Table II shows that the flexibility cost appears in the number of clock cycles needed to decode a frame: 1161 cycles are needed for the proposed TTPD whereas 222 are needed in [13]. The two main factors explaining this difference are i) a more complete tree pruning, with the usage of SPC and REP nodes and ii) in the TT-SC decoder, each input of a FU

TABLE I
Comparison of programmable processors running the SC decoding algorithm for R=0.5 polar codes

Architecture	N	Latency [μs]	Throughput [Mb/s]	E_b [nJ/bit]
i7-3.3GHz	1024	2.0	257	41
	512	1.2	210	49
(GPP)	256	0.7	179	59
	128	0.4	143	73
A57-1.1GHz	1024	10.7	48	17
	512	5.3	48	17
(GPP)	256	2.8	46	17
	128	1.6	41	20
LX7-835MHz	1024	7.2	71	1.6
	512	3.9	66	1.7
(ASIP)	256	1.9	65	1.7
	128	1.0	62	1.8
TTPD-800MHz	1024	1.4	352	0.14
	512	0.8	313	0.15
(ASIP)	256	0.4	304	0.16
	128	0.2	284	0.17

TABLE II
FPGA implementations of dedicated SC decoders for a (1024,512) polar code

	Proposed TTPD	[22]	[13]	
Target	Artix 7	Stratix IV	Stratix IV	Virtex 6
Clock cycles	1161	222	165	165
IT/P (Mb/s)	44	238	319	217
Freq (MHz)	100	103	103	70
LUTS	14744	23020	24821	22115
FFs	7354	1024	5823	7941
RAM (Kb)	141	43	36	36
Pruning	R0 & R1	Full	Full	Full

TABLE III
ASIC IMPLEMENTATIONS OF DEDICATED SC DECODERS FOR A (1024,512) POLAR CODE

	Proposed TTPD	[23]	[4]	[4][1]
Target	28nm	28nm	180nm	28nm
Clock cycles	1161	1833	1568	1568
IT/P [Mb/s]	352	94	49	436
Freq [MHz]	800	336	150	1335
Power [mW]	48	18	67	5
$\mathbf{E_b}$ [nJ/bit]	0.14	0.19	1.4	0.011
Area [mm^2]	0.16	0.44	1.71	0.04
Pruning	R0 & R1	First R0	None	None

[1] Scaling factors from 180nm to 28nm of [4] are taken from [23].

has to be registered while in a dedicated architecture, the load - compute - save operations can be performed in a single clock cycle. In terms of FPGA complexity, the TT-SC decoder needs a similar amount of logic (LUT and FF) while increasing the memory cost, as explained in section III-C1. For the record, the Base TTA as represented in Figure 2 is implemented on approximately 1000 LUTs and 700 Flip-Flops in our design.

Table III shows some implementation results of previous ASIC SC decoders. Comparing the TT-SC decoder with ASIC implementations is difficult due the variety of implementation parameters. Indeed, the SC decoder proposed in [23] is the basis of a list decoder, and its power and area could be reduced if only SC was supported. Moreover, a limited pruning strategy is used. This explains the lower throughput compared to the TT-SC processor. The SC decoder in [4] is based on a different technology node and no pruning is implemented. Thus, Table III only aims at showing that the TT-SC processor has a competitive hardware complexity and power consumption. It achieves several hundreds of Mb/s while conserving the great flexibility of a programmable processor.

B. `TT-SCAN` Architecture

In order to demonstrate its extensibility, the proposed TTA SC decoder was further customized to implement the SCAN decoding algorithm. Three modifications were necessary: i) an LSU and the associated RAM were added to handle the extra LLRs required in the SCAN algorithm, ii) the LLR ALU was modified to support SCAN update rules as depicted in Figure 4 and iii) a SCAN specific unrolled sub-tree decoder was designed (”Unrolled SCAN” in Figure 3).

The `TT-SCAN` was implemented on an Artix 7 FPGA. The resulting throughput of our programmable `TT-SCAN` is the same as that of the dedicated decoder reported in [25]. The algorithm is exactly the same, but the parallelism is only 16, and no unrolled subtree is used.

V. CONCLUSION

In this paper, the first TTA-based polar decoders were proposed. Compared to the state-of-the-art ASIP implementation of a (1024, 512) polar code under SC decoding, the throughput of the proposed decoder (352 Mbps) is increased fivefold and the energy consumption (0.14 nJ / bit) is one order of magnitude lower. Compared to an optimized software implementation on a GPP (x86 architecture), the throughput is 37 % higher and the energy consumption is two orders of magnitude lower. The use of TTA processor for polar decoding is shown to be an interesting compromise between the performance of dedicated decoders and the flexibility of programmable processors. Flexibility was demonstrated by transforming a TT-SC decoder into a TT-SCAN decoder. Future works will focus on the list decoding of polar codes.

REFERENCES

[1] E. Arikan, “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels,” *IEEE Transactions on Information Theory*, 2009.

[2] 3GPP, “TS 38.212, Multiplexing and Channel Coding,” 2017.

[3] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware Architectures for Successive Cancellation Decoding of Polar Codes,” in *ICASSP*, 2011.

[4] A. Mishra, A. J. Raymond, L. G. Amaru, G. Sarkis, C. Leroux, P. Meinerzhagen, A. Burg, and W. J. Gross, “A Successive Cancellation Decoder ASIC for a 1024-bit Polar Code in 180nm CMOS,” in *IEEE A-SSCC*, Nov. 2012.

[5] B. Le Gal, C. Leroux, and C. Jégo, “Multi-Gb/s Software Decoding of Polar Codes,” *IEEE Transactions on Signal Processing*, Jan 2015.

[6] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, “Fast Software Polar Decoders,” in *ICASSP*, May 2014.

[7] M. Léonardon, C. Leroux, D. Binet, J. M. P. Langlois, C. Jégo, and Y. Savaria, “Custom Low Power Processor for Polar Decoding,” in *ISCAS*, May 2018.

[8] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. New York, NY, USA: John Wiley & Sons, Inc., 1997.

[9] B. Rister, P. Jääskeläinen, O. Silvén, J. Hannuksela, and J. R. Cavallaro, “Parallel Programming of a Symmetric Transport-Triggered Architecture with Applications in Flexible LDPC Encoding,” in *ICASSP*, 2014.

[10] H. Kultala, O. Esko, P. Jääskeläinen, V. Guzma, J. Takala, J. Xianjun, T. Zetterman, and H. Berg, “Turbo Decoding on Tailored OpenCL Processor,” in *IWCMC*, 2013.

[11] B. Le Gal, C. Leroux, and C. Jégo, “A scalable 3-phase polar decoder,” in *ISCAS*, May 2016.

[12] A. Alamdar-Yazdi and F. R. Kschischang, “A Simplified Successive-Cancellation Decoder for Polar Codes,” *IEEE Comm. Letters*, 2011.

[13] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, “Fast Polar Decoders: Algorithm and Implementation,” *IEEE JSAC*, 2014.

[14] U. U. Fayyaz and J. R. Barry, “Low-Complexity Soft-Output Decoding of Polar Codes,” *IEEE JSAC*, 2014.

[15] Y. Wang, K. R. Narayanan, and Y. C. Huang, “Interleaved Concatenations of Polar Codes With BCH and Convolutional Codes,” *IEEE Journal on Selected Areas in Communications*, 2016.

[16] D. Tabak and G. J. Lipovski, “MOVE Architecture in Digital Controllers,” *IEEE Journal of Solid-State Circuits*, 1980.

[17] O. Esko, P. Jääskeläinen, P. Huerta, C. S. de La Lama, J. Takala, and J. I. Martinez, “Customized Exposed Datapath Soft-Core Design Flow with Compiler Support,” in *IEEE FPL*, 2010.

[18] C. Lattner and V. Adve, “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation,” in *CGO*, 2004.

[19] G. Sarkis, P. Giard, C. Thibeault, and W. J. Gross, “Autogenerating Software Polar Decoders,” in *GlobalSIP*, Dec. 2014.

[20] A. Cassagne, B. Le Gal, C. Leroux, O. Aumage, and D. Barthou, “An Efficient, Portable and Generic Library for Successive Cancellation Decoding of Polar Codes,” in *LCPC*. Springer, Cham, Sep. 2015.

[21] P. Jääskeläinen, T. Viitanen, J. Takala, and H. Berg, *HW/SW Co-design Toolset for Customization of Exposed Datapath Processors*. Springer International Publishing, 2017, pp. 147–164.

[22] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, “A 638 Mbps Low-Complexity Rate 1/2 Polar Decoder on FPGAs,” in *SiPS*, Oct. 2015.

[23] P. Giard, A. Balatsoukas-Stimming, T. C. Müller, A. Bonetti, C. Thibeault, W. J. Gross, P. Flatresse, and A. Burg, “PolarBear: A 28nm FD-SOI ASIC for Decoding of Polar Codes,” *IEEE JESTCS*, 2017.

[24] A. Cassagne, M. Léonardon, O. Hartmann, G. Delbergue, T. Tonnellier, R. Tajan, C. Leroux, C. Jego, B. Le Gal, O. Aumage, and D. Barthou, “Fast Simulation and Prototyping with AFF3CT,” in *SiPS*, 2017.

[25] G. Berhault, C. Leroux, C. Jégo, and D. Dallet, “Hardware Implementation of a Soft Cancellation Decoder for Polar Codes,” in *DASIP*, 2015.