Custom Low Power Processor for Polar Decoding

Mathieu Léonardon*†, Camille Leroux*, David Binet†, J.M. Pierre Langlois†, Christophe Jégo* and Yvon Savaria†
*IMS Laboratory, UMR CNRS 5218, Bordeaux INP, University of Bordeaux, Talence, France
†École Polytechnique de Montréal, QC, Canada
Email: {mathieu.leonardon,david.binet,pierre.langlois,yvon.savaria}@polymtl.ca {christophe.jego,camille.leroux}@ims-bordeaux.fr

***Abstract*—Cloud Radio Access Network is foreseen as one of the key features of the future 5G mobile communication standard. In this context, all the baseband processing is intended to be performed on CPUs in order to keep a high level of flexibility. The challenge is then to propose efficient software implementations of baseband processing algorithms that guarantee a sufficient throughput, while limiting the energy consumption. In this paper, as an alternative to general purpose processors, we propose an implementation of an Application Specific Instruction set Processor customized for the Successive Cancellation decoding of polar codes. The resulting software decoder achieves throughputs similar to state-of-the-art ARM processor implementations, while reducing the energy consumption by a factor 10.**

I. Introduction

Polar codes [1] have been selected as the error correction code for the control channels in the fifth generation of mobile networks (5G). Meanwhile, in the context of the 5G standard, Cloud Radio Access Network (Cloud-RAN) is foreseen as one potential disruptive technology in which the physical (PHY) layer is completely virtualized as it is executed on large scale programmable processor arrays. Actually, Cloud-RAN would allow dynamic balancing of the computational effort across the network, as well as inter-cell cooperation [2]. In this context, processing resources must keep a high degree of flexibility, while guaranteeing a sufficient throughput and reducing the energy consumption. Within the PHY layer baseband processing, error correction decoding is one of the most computationnally intensive tasks [3], [4]. As a consequence, future networks based on Cloud-RAN will require efficient software implementations of polar decoders. Recently, optimized software decoders have been proposed on high-end general purpose processors (GPPs) [5], [6]. In [7], [8], embedded ARM processors have shown to be especially energy-efficient for this kind of algorithms. Finally, graphical processing unit (GPU) implementations of polar decoders [9] can reach high throughput, at the expense of a high latency and energy consumption compared to x86 and ARM processors.

In order to address a large set of domains, general purpose processors (x86, ARM, ...) usually include various hardware resources that may not be used in some applications. For instance, in the case of channel decoding, all the floating point units are unused because processing is usually performed on a fixed-point data representation. These unused hardware resources consume expendable leakage power. When profiling optimized software polar decoders, one rapidly observes that most of the time is spent executing a small set of elementary functions. Moreover a significant number of clock cycles are used for loading/storing data in registers. These observations push towards the use of programmable processors that i) only include application relevant hardware to limit power and energy consumption and ii) integrate some dedicated processing resources to increase the throughput. This kind of custom processors is usually called an Application Specific Instruction-set Processor (ASIP).

In this paper, the use of ASIPs as potential candidates for efficient software decoding of polar codes is investigated. More specifically, the architecture of a Cadence Xtensa LX7 processor [10] is customized in such a way that it achieves a throughput similar to ARM processors, while reducing the energy consumption by more than one order of magnitude.

As a first investigation of the potential of ASIPs for polar code decoding, we focus on the Successive Cancellation (SC) decoding algorithm. A natural extension of this work is to address the more complex Successive Cancellation List (SCL) decoding algorithm. It would benefit from the already implemented optimizations, but it would also require some further works especially for the list sorting and management. The polar codes used for benchmarks are the ones recently defined in the 3GPP meetings for the upcoming 5G [11].

After a brief overview of polar coding/decoding in section II, section III describes the base architecture of the Cadence Xtensa LX7 and the different architectural modifications that were performed to improve the SC decoding throughput. In section IV, some throughput, latency and energy estimations are provided in order to compare with ARM and x86 implementations. Some conclusions and future works suggestions are provided in Section V.

II. Polar Coding

A. Polar Encoding

Polar codes are linear block codes with a recursive structure similar to Reed-Muller codes. The encoding of a (N, K) polar code can be seen as an N-dimension linear transformation applied to an N-bit vector $\boldsymbol{u}$: $\boldsymbol{x} = \boldsymbol{u}F^{\otimes n}$, where $\boldsymbol{x}$ is the generated codeword and $F^{\otimes n}$ the n-th Kronecker product of $\left[\begin{smallmatrix}1 & 0\\ 1 & 1\end{smallmatrix}\right]$. The $\boldsymbol{u}$ vector includes K *information bits* and $N-K$ *frozen bits*. The linear transformation is described by the N-by-N matrix $F^{\otimes n}$ where $N = 2^n$ and $n \geq 0$. Within the vector $\boldsymbol{u}$, the location of information bits is defined by a set $\mathcal{A} \subset \{0, ..., N-1\}$, while the complementary subset $\mathcal{A}^c$ contains the location of frozen bits.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

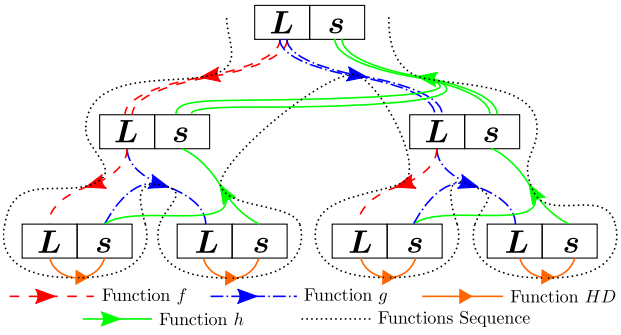


Fig. 1. Successive Cancellation Decoding.

B. Successive Cancellation Decoding

As shown in Figure 1, SC decoding can be seen as a pre-order binary tree traversal of depth n in which each node consists of 2^{n-d} LLRs, which are real values, and 2^{n-d} partial sums, which are binary values, d is the depth of the considered node. The functions that are to be applied to these values are listed in (1).

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

Figure 1 represents SC decoding for $n = 2$. One can observe that multiple functions can be applied simultaneously between the two upper layers of the tree. The number of such possibly parallel operations on a node at depth d is equal to half the number of L values stored at this node i.e. 2^{n-d}. This possible parallelism is denoted as *intra-frame* parallelism. The decoded *codeword* $\hat{\mathbf{u}}$ is equal to the partial sums held in the root node of the tree when the decoding process is complete.

C. Simplified SC decoding

As proposed in [12] and later improved in [13], it is possible to reduce 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, respectively correspond to nodes holding only frozen bits or only information bits. Specialized nodes called repetition (`REP`) and single parity check nodes (`SPC`) correspond to subtrees. They represent repetition codes and single parity check codes. This tree pruning method drastically reduces the number of operations involved to decode the tree, with a negligible impact on the decoding performance.

D. Parallelism

State-of-the art software polar decoders executing on general purpose processors [5]–[9] extensively exploit Single Instruction Multiple Data (SIMD) units to speedup the processing. As first formalized in [6], two parallelization techniques can be applied. The *intra-frame* method consists in using SIMD units to perform several operations (e.g. functions from Eq. (1) in the SC decoding tree). The alternative *inter-frame* method consists in decoding multiple frames in parallel. In comparison with the intra-frame method, the throughput of the *inter-frame* method is higher, because SIMD units are always fully used. However the time spent loading the data in the decoder increases the latency of the decoder. In this paper, the *intra-frame* parallelization strategy is preferred, because of its lower memory footprint. It causes lower power consumption and smaller memory cache allocation in the custom processor. Finally intra-frame parallelization features a lower latency, which is a very important criterion in 5G.

III. Processor Design

The selected base processor is the Cadence XTensa LX7 that was chosen for its genericity and extensibility. The philosophy of the XTensa processor is to propose a RISC-oriented base architecture intended to be simple and energy efficient, but very extensible. Then, by designing custom parallel instructions, the energy consumption and cycle counts of a given application execution can be drastically reduced. The proposed custom processor was designed by following this methodology. Figure 2 shows the architectural model that is composed of three parts: the data cache memory, the hardware resources for the core instructions, and the hardware resources for the custom instructions.

A. Base Architecture

The XTensa instruction set architecture (ISA) has been designed for *configurability* and *extensibility*. The latter is the possibility to add designer-defined custom instructions and the associated custom hardware units, which are described in section III-C. *Configurability* is the possibility to select pre-designed functionalities for a given application to customize the base architecture. One can for example add pre-designed hardware accelerators (multipliers, mac, or floating point units), enable interrupt and exception handling, or use memory protection and local memory. In the proposed custom processor, no pre-designed accelerator is used, but necessary features consuming the least hardware are activated by default. Indeed, in polar decoding, the role of the core instructions is mainly to handle pointers and variables used to access the data and feed them to the custom SIMD units. While no heavy arithmetic is used, a very long instruction word (VLIW) structure called FLIX3 is chosen. It enables a pre-configured 3-way pipeline. With this configuration, a 64-bit complex instruction that includes 3 Xtensa base instructions, each operating on distinct 32-bit data, can be simultaneously executed. The execution of three instructions in parallel helps reducing the cycle counts in multiple parts of the decoding algorithm. The last relevant configuration was to choose to increase the number of general purpose registers from 32 to 64, which also improved performance.

B. Memory of the Architecture

1) Data quantization: It is demonstrated in [14] that 6-bit LLR values enable matching the BER performance of non-quantized decoders. As it is more convenient for a software

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

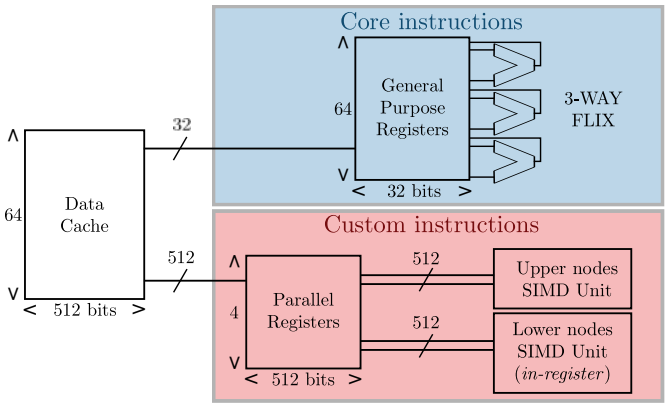


Fig. 2. Processor structure and data path

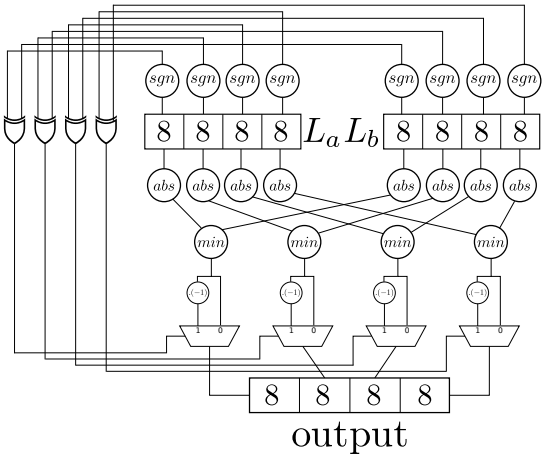


Fig. 3. f function SIMD hardware unit

```
1 inline void f_64(signed char* la,
2                  signed char* lb,
3                  signed char* lc)
4 {
5   p512_a = (VR512*) la; // cast char* to simd ptr
6   p512_b = (VR512*) lb; // cast char* to simd ptr
7   p512_c = (VR512*) lc; // cast char* to simd ptr
8
9   v512_a = *p512_a;     // fill simd register
10  v512_b = *p512_b;     // fill simd register
11
12  // execute custom instruction
13  v512_c = custom_f_64(v512_a, v512_b);
14
15  *p512_c = v512_c;     // store result in cache
16 }
```

Fig. 4. C++ SIMD implementation of the f functions.

implementation, 8-bit integers are selected to store the LLRs in the memory. As first reported in [15], LLR storage requires 2^{n-d} LLRs to be stored at depth d layer. Consequently, the overall LLR memory size is $2^{n+1}-1$ bytes. Although partial sums are actually binary values, they are stored as 8-bit integers and their memory footprint is 2^n. It would be possible to compress this part of the memory at the cost of extra masking operations and more irregularity in the data access.

2) Cache Memory Configuration: As the goal of our design is to decrease the decoding time by increasing parallelism, the maximum available register size is assigned, which is 512 bits. Therefore, in order to load and store data from and to the cache memory in one clock cycle, the cache memory line size is also 512 bits. The size and associativity of the data and instruction memories are customizable. Experiments were performed to select these configurations. In all the tested cases, 4-way associativity offers the best performance. The cache size can be any power of two between 2 and 128 KB. Even if the smallest number of cache misses is reached for the largest memory, a memory of 8 KB for both instruction and data cache is enough to reach a number of cache misses that is less than 5 percents higher than the minimum one.

C. Custom Instructions and SIMD Units

Acceleration of the decoding algorithm relies heavily on the vectorization of the elementary functions of Eq. (1). To this end, custom instructions were defined together with SIMD hardware units that are necessary to execute them. For instance, Figure 4 shows how the f function can be called in the source code. Figure 3 describes the architecture of the corresponding SIMD hardware unit. For clarity, in Figure 3, a parallelism of $P = 4$ is shown, while in the real implementation, we actually have $P = 64$. The other elementary functions of Eq. (1) and the specialized functions derived from the Fast-SSC algorithm [13] are implemented the same way.

With custom instructions, a significant amount of time is dedicated to the loading and storing of the LLRs and partial sums from and to the data cache. In order to reduce the amount of time spent during this data handling, *in register custom instructions* have been developed. The principle is that a complete subtree of the decoding tree is loaded in a single 512-bit register. It is thus possible to work only with the same unique register, without having to perform additional stores and loads. As shown in section IV, this technique enables significant performance gains especially on short polar codes.

D. Software Description

A *Pseudo-Unrolled* technique, inspired from the unrolling technique in [16], is applied for the software description. All the recursive aspects of the code and therefore a significant number of context switches, are removed. Instead, all the elementary functions to be applied are stored in a function pointer array before the decoding process. The input of the functions, e.g. the pointers to the data memory, are precomputed. These pre-computations being done only once, decoding only consists in executing each function stored in the array. The number of clock cycles required for the decoding process is then approximately reduced by a factor of two.

IV. Results and Discussion

Results reported in this section were obtained with polar codes constructed as specified in the draft modulation and coding scheme of the 5G standardization group [11]. The

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

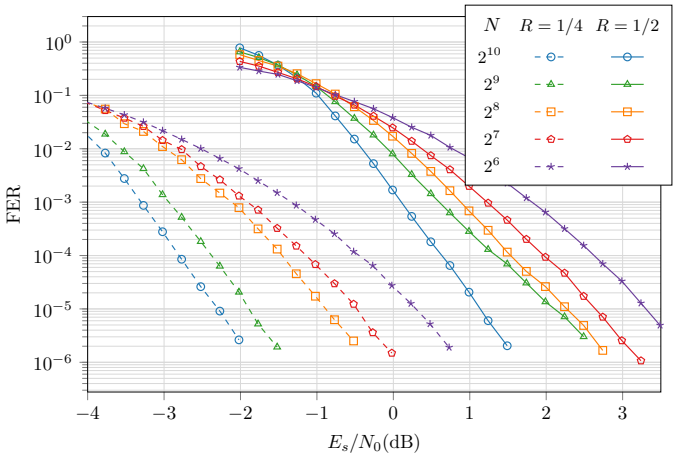
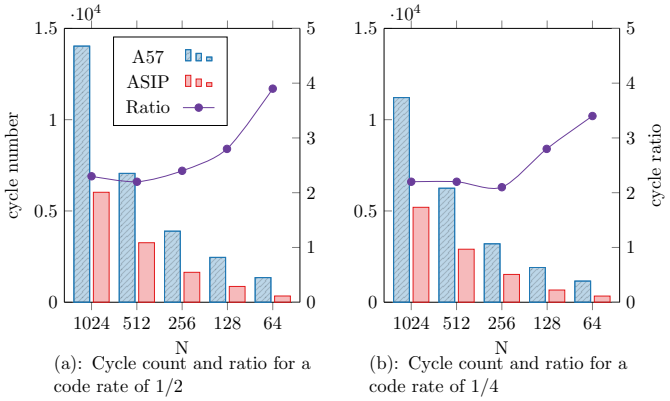


Fig. 5. Frame Error Rate (FER) of the SC decoder for different code lengths (N) and code rates (R).



(a): Cycle count and ratio for a code rate of 1/2

(b): Cycle count and ratio for a code rate of 1/4

Fig. 6. Comparison of the average number of cycles needed to decode one frame on A57 and ASIP different code rates (1/2 on the left, 1/4 on the right)

academic version of the XTensa design tools does not enable the logic synthesis of the processor. Only estimates of the frequency, power consumption and design area are available [17]. By using the TSMC HPM 28 nm technology parameters, the estimated power of the proposed ASIP is 111 mW, and the estimated area is 0.475 mm^2. The estimated frequency is 835 MHz, but it does not take into account the custom instructions datapath. As a consequence, a more pessimistic scenario with a running frequency of 400MHz is considered.

A. Comparison with an ARM processor

The AFF3CT software [18] allows running an optimized software SC decoder on the A57 processor. The reported data used for the ARM implementation were provided by the authors of [8]. The software decoder is not generated but dynamic, as it is the case for the ASIP implementation. The *intra-frame* method is used and 8-bit words are chosen to store the LLRs. The NEON instructions being 128-bit wide, the parallelism level is 16. The maximum running frequency of the Juno board is exploited [8]. In the reported results, the RAM power consumption is not taken into account.

Figure 6 shows the average number of clock cycles to decode one frame for both processors. The use of customized SIMD units reduces the number of clock cycles. This is especially true for the decoding of short polar codes in which *in-register custom instructions* are prevalent. However this lower number of clock cycles is counterbalanced by a lower clock frequency as shown in Table I ($R = 1/2$). Nevertheless, even for the less favorable scenario ($f = 400$ MHz), the ASIP architecture achieves throughputs similar to the ARM processor. The most remarkable benefit is the energy per decoded bit, which is more than 10 times lower for the ASIP.

B. Comparison with an x86 processor

The throughput, latency and power consumption of the same software decoder have also been measured on a Intel i7-4712HQ processor. The power of the core is evaluated with the `powerstat` tool, reporting only the core power. This time, the parallelism level can be fixed to 32 using the 256-bit wide AVX2 SIMD instructions. The CPU was run at the maximum turbo frequency of 3.3 GHz. The results show that the throughput and the latency of the decoder is far better than the other implementations but that the energy efficiency is worse. An *inter-frame* implementation would increase the efficiency at the cost of increased latency and memory footprint as shown in [8].

TABLE I
COMPARISON OF THE LATENCY, THROUGHPUT AND POWER CONSUMPTION

Target	N	Latency [μs]	Throughput [Mb/s]	E_b [nJ]
A57-1.1GHz	1024	13	38	21
	512	6.7	38	21
	256	3.6	35	22
	128	2.1	30	27
i7-3.3GHz	1024	2.3	222	47
	512	1.4	182	57
	256	0.8	155	68
	128	0.5	124	85
ASIP-835MHz	1024	7.2	71	1.6
	512	3.9	66	1.7
	256	1.9	65	1.7
	128	1.0	62	1.8
ASIP-400MHz	1024	15	34	1.4
	512	8.2	31	1.6
	256	4.1	31	1.6
	128	2.1	30	1.7

V. Conclusion

Cloud-RAN is foreseen to be an important feature of the 5G standard. It enables the network to be more flexible and adaptable. In this paper, ASIP processors customized for polar decoding have been studied in the context of network virtualization. The proposed custom processor is 10 times more energy efficient than state-of-the-art software implementations of SC polar decoders on embedded processors, while achieving the same throughput. Relevant future works includes the logic synthesis of the ASIP architecture and the adaptation of the ASIP architecture to the SCL algorithm.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

REFERENCES

[1] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[2] D. Wübben, P. Rost, J. S. Bartelt, M. Lalam, V. Savin, M. Gorgoglione, A. Dekorsy, and G. Fettweis, "Benefits and impact of cloud computing on 5G signal processing: flexible centralization through cloud-ran," *IEEE Signal Processing Magazine*, vol. 31, no. 6, pp. 35–44, 2014.

[3] V. Q. Rodriguez and F. Guillemin, "Towards the deployment of a fully centralized cloud-ran architecture," in *Proceedings of the IEEE International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 1055–1060.

[4] N. Nikaein, "Processing radio access network functions in the cloud: critical issues and modeling," in *Proceedings of the ACM International Workshop on Mobile Cloud Computing and Services (MCS)*, 2015, pp. 36–43.

[5] 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 *The 28th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2015)*., Raleigh, United States, Sep. 2015. [Online]. Available: https://hal.inria.fr/hal-01203105

[6] B. L. Gal, C. Leroux, and C. Jego, "Multi-gb/s software decoding of polar codes," *IEEE Transactions on Signal Processing*, vol. 63, no. 2, pp. 349–359, Jan 2015.

[7] ——, "Software polar decoder on an embedded processor," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2014, pp. 1–6.

[8] A. Cassagne, O. Aumage, C. Leroux, D. Barthou, and B. L. Gal, "Energy consumption analysis of software polar decoders on low power processors," in *2016 24th European Signal Processing Conference (EUSIPCO)*, Aug 2016, pp. 642–646.

[9] P. Giard, G. Sarkis, C. Leroux, C. Thibeault, and W. J. Gross, "Low-latency software polar decoders," *CoRR*, vol. abs/1504.00353, 2015. [Online]. Available: http://arxiv.org/abs/1504.00353

[10] I. U. Cadence Design Systems. Tensilica customizable processors. [Online]. Available: https://ip.cadence.com/ipportfolio/tensilica-ip/xtensa-customizable

[11] 3GPP, "TS 38.212, multiplexing and channel coding (release 15)," Tech. Rep., sep 2017.

[12] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, 2011.

[13] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, May 2014.

[14] ——, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, 2014.

[15] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 1665–1668.

[16] G. Sarkis, P. Giard, C. Thibeault, and W. J. Gross, "Autogenerating software polar decoders," in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec 2014, pp. 6–10.

[17] J. Sanghavi and A. Wang, "Estimation of speed, area, and power of parameterizable, soft ip," in *Proceedings of the 38th Annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 31–34. [Online]. Available: http://doi.acm.org/10.1145/378239.378259

[18] A. Cassagne, M. Leonardon, C. Leroux, R. Tajan, G. Delbergue, B. Petit, and O. Hartmann, "Aff3ct, an open source software dedicated to forward error correction simulations, http://aff3ct.github.io," 2017.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE