HARDWARE IMPLEMENTATION OF ITERATIVE PROJECTION-AGGREGATION DECODING OF REED-MULLER CODES

Marzieh Hashemipour-Nazari, Kees Goossens, Alexios Balatsoukas-Stimming

Eindhoven University of Technology

ABSTRACT

In this work, we present a simplification and a corresponding hardware architecture for hard-decision recursive projection-aggregation (RPA) decoding of Reed-Muller (RM) codes. In particular, we transform the recursive structure of RPA decoding into a simpler and iterative structure with minimal error-correction degradation. Our simulation results for RM$(7, 3)$ show that the proposed simplification has a small error-correcting performance degradation (0.005 in terms of channel crossover probability) while reducing the average number of computations by up to 40%. In addition, we describe the first fully parallel hardware architecture for simplified RPA decoding. We present FPGA implementation results for an RM$(6, 3)$ code on a Xilinx Virtex-7 FPGA showing that our proposed architecture achieves a throughput of 171 Mbps at a frequency of 80 MHz.

1. INTRODUCTION

Reed-Muller (RM) codes were first proposed in 1954 [1]. Recently, there has been a renewed interest in RM codes because, in some cases, they were shown to be capable of achieving the Shannon capacity of the binary erasure channel (BEC) [2] and the binary symmetric channel (BSC) [3]. The oldest decoding algorithm for RM codes is based on majority voting [1], and it guarantees correction of the error patterns with a weight less than half of the minimum distance. A wide variety of algorithms has been proposed afterward to improve decoding capacity. For example, the Sidel'nikov-Pershakov algorithm [4] corrects most of the corrupted codewords with a number of errors less than $(1 - \varepsilon)n/2$, where n indicates blocklength of the RM codes and $\varepsilon \geq n^{-1/3}$. Some hardware architectures are also available for the aforementioned methods. A parallel decoding architecture for majority-logic decoding algorithms was provided in [5], and a low-area decoder for the Reed decoding method was introduced in [6].

Successive-cancellation (SC) decoding [7] and SC list (SCL) decoding [8], make use of the decomposable structure of RM codes to provide recursive decoding methods with reasonable complexity. The work of [9] improved the performance of SC and SCL decoding methods by exploring several carefully selected permutations of the factor graph of RM codes. The work of [10] exploits the symmetric structure of RM codes and applies an iterative decoding method to provide near maximum likelihood (ML) performance. Other works focused on special cases of RM codes. For example, [11] is a modified version of the Sidel'nikov-Pershakov algorithm that improves error-correcting performance for second-order RM codes. Moreover, the work of [12] provided a new ML decoder with a lower complexity for RM codes of order $m-3$, where $m = \log_2 n$.

The main drawback of the aforementioned algorithms is that they have poor error-correcting performance for short blocklength RM codes. For this reason, the authors of [13] proposed a new algorithm called recursive projection-aggregation (RPA) decoding that improves the error-correcting performance of RM codes in the regimes of interest of ultra-reliable low-latency communications (URLLC) and of the Internet of Things (IoT), i.e., low rate and short blocklength RM codes. The RPA algorithm is highly parallelizable. However, it has a high complexity and its recursive structure is not particularly amenable to hardware implementations. The authors of [14] proposed a collapsed projection-aggregation (CPA) decoding algorithm, which merges multiple recursion levels into a single step. However, this comes at the cost of an increased complexity for the projection step, since more than two bits (or LLRs) are combined at each step. The authors of [14] also introduce recursive puncturing-aggregation (RXA), which is more suitable for high-rate RM codes.

Contributions: In this paper, we present a simplified version of the RPA algorithm to make a trade-off between the error-correcting performance and computations. We simplify the RPA algorithm by carefully removing computations in the recursion levels to make the structure suitable for hardware implementations. Moreover, we propose the first fully parallel hardware architecture for RPA decoding.

2. REED-MULLER CODES

The focus of this paper is on the BSC, so all operations and vectors are in $\mathbb{F}_2$. RM codes are denoted by $\mathrm{RM}(m, r)$, where m indicates the code length $n = 2^m$ and r is the order. RM codes are linear block codes with rate $R = \frac{k}{n}$, $k = \sum_{i=0}^{r} \binom{n}{i}$, and with the following recursively defined generator matrix:

$$\mathbf{G}_{(m,r)} = \begin{bmatrix} \mathbf{G}_{(m-1,r)} & \mathbf{G}_{(m-1,r)} \\ \mathbf{0} & \mathbf{G}_{(m-1,r-1)} \end{bmatrix}, \quad \mathbf{G}_{(1,1)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}. \tag{1}$$

Algorithm 1 The RPA decoding of RM codes RM(m, r)

Input: The noisy codeword $\mathbf{y}$, m, r, $N_{\max}$
Output: The decoded codeword $\mathbf{c}$
1: **if** $r = 1$ **do**
2: $c \leftarrow$order-1-decoding $(\mathbf{y}, m)$
3: **else**
4: **for** $j = 1 : N_{\max}$ **do**
5: **for** $i = 1 : 2^m - 1$ **do**
6: $\mathbf{y}_i \leftarrow \mathtt{Proj}(\mathbf{y}, i, m)$
7: $\hat{\mathbf{y}}_i \leftarrow \mathtt{RPA}(\mathbf{y}_i, m-1, r-1, N_{\max})$
8: **end for**
9: $\hat{\mathbf{y}} \leftarrow \mathtt{Agg}(\mathbf{y}, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{n-1}, \hat{\mathbf{y}}_2, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_{n-1})$
10: **if** $\mathbf{y} = \hat{\mathbf{y}}$ **do**
11: **break** *--RPA converges to a fixed point*
12: **end if**
13: $\mathbf{y} \leftarrow \hat{\mathbf{y}}$
14: **end for**
15: **end if**

Algorithm 2 The projection function `Proj`

Input: $\mathbf{y}_{\text{in}}(0 \text{ to } n-1), i, m$
Output: $\mathbf{y}_{\text{out}}(0 \text{ to } n/2-1)$
1: $n \leftarrow 2^m$
2: **if** $i < n/2$ **do**
3: $\mathbf{y}_{\text{out}}(0 \text{ to } n/4-1) \leftarrow \mathtt{Proj}(\mathbf{y}_{\text{in}}(0 \text{ to } n/2-1), m-1, i)$
4: $\mathbf{y}_{\text{out}}(n/4 \text{ to } n/2-1) \leftarrow \mathtt{Proj}(\mathbf{y}_{\text{in}}(n/2 \text{ to } n-1), m-1, i)$
5: **else**
6: **for** j=$1 : n/2-1$ **do**
7: $\mathbf{y}_{\text{tmp}}(2j) \leftarrow \mathbf{y}_{\text{in}}(j)$
8: $\mathbf{y}_{\text{tmp}}(2j+1) \leftarrow \mathbf{y}_{\text{in}}(j \oplus i)$
9: **end for**
10: $\mathbf{y}_{\text{tmp}}(0) \leftarrow \mathbf{y}_{\text{in}}(0)$
11: $\mathbf{y}_{\text{tmp}}(1) \leftarrow \mathbf{y}_{\text{in}}(i)$
12: **for** t=$0 : n/2-1$ **do**
13: $\mathbf{y}_{\text{out}}(t) \leftarrow \mathbf{y}_{\text{tmp}}(2t) \oplus \mathbf{y}_{\text{tmp}}(2t+1)$
14: **end for**
15: **end if**

Algorithm 3 The aggregation function `Agg`

Input: $m, \mathbf{y}_{\text{in}}, \mathbf{y}_1, \mathbf{y}_2 \ldots, \mathbf{y}_{n-1}, \hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_{n-1}$
Output: $\mathbf{y}_{\text{out}}$
1: **for** $z = 0 : 2^m - 1$ **do**
2: $vote(z) \leftarrow 0$
3: **for** $i = 1 : 2^m - 1$ **do**
4: $Ind \leftarrow \mathtt{FindIndex}(z, i, m)$
5: $vote(z) \leftarrow \mathbf{y}_i(Ind) \oplus \hat{\mathbf{y}}_i(Ind) + vote(z)$
6: **end for**
7: $\mathbf{y}_{\text{out}}(z) \leftarrow \mathbf{y}_{\text{in}}(z) \oplus \mathbf{1}\left[vote(z) > \frac{2^m-1}{2}\right]$
8: **end for**

Algorithm 4 The function `FindIndex`

Input: Index z, branch number i, m
Output: Ind
1: **if** $i \geq 2^{m-1}$ **do**
2: **if** $z < 2^{m-1}$ **do**
3: $Ind \leftarrow z$
4: **else**
5: $Ind \leftarrow$ bi2de(de2bi $(z) \oplus$ de2bi(i))
6: **end if**
7: **else**
8: **if** $z < 2^{m-1}$ **do**
9: $Ind \leftarrow \mathtt{FindIndex}(z, i, m-1)$
10: **else**
11: $Ind \leftarrow \mathtt{FindIndex}(z - (2^{m-1}), i, m-1) + 2^{m-2}$
12: **end if**
13: **end if**

2.1. Recursive Projection Aggregation Decoding

As Algorithm 1 shows, the RPA algorithm has three main steps: projection (line 6), recursive decoding (line 7), and aggregation (line 9). Let us consider a noisy received vector $\mathbf{y}$ of the transmitted codeword $\mathbf{c}$ of length n.

In the *projection* step, $\mathbf{y}$ is transformed into $n-1$ distinct vectors of length $n/2$. For hard-decision decoding, each transformed vector $\mathbf{y}_i, i \in \{1, 2, \ldots, n-1\}$, is obtained by taking the modulo-2 sum over specific coordinates of the input vector corresponding to the i-th projection, as shown on lines 2-11 of Algorithm 2. Next, on line 13, a binary *XOR* operation sums every two adjacent bits to convert each n-bit input vector $\mathbf{y}$ to an $n/2$-bit vector $\mathbf{y}_i$.

In the *recursive decoding* step, each vector $\mathbf{y}_i$, produced in the projection step, is recursively decoded by RPA for RM$(m-1, r-1)$ until first-order RM codes are reached, which can be decoded efficiently using the fast Hadamard transform (FHT) [15]. Each $\hat{\mathbf{y}}_i$ is a decoded vector of $\mathbf{y}_i$.

In the *aggregation* step, for each coordinate $\hat{\mathbf{y}}(z)$, Algorithm 4 finds the corresponding coordinates in $\mathbf{y}_i$ that were originally created with $\mathbf{y}(z)$. These coordinates together with their decoded value in $\hat{\mathbf{y}}_i$ represent $n-1$ estimations for each coordinate of vector $\hat{\mathbf{y}}$ (see line 3-6 in Algorithm 3). Next, in line 7 of Algorithm 3, per-coordinate majority voting is performed to produce an estimate $\hat{\mathbf{y}}$ of the transmitted codeword.

This procedure is repeated for multiple iterations. In [13], the maximum number of iterations is set to $N_{\max} = \lceil m/2 \rceil$.

3. ITERATIVE PROJECTION-AGGREGATION DECODING

As can be seen on line 4 of Algorithm 1, RPA decoding performs multiple iterations at each level of the recursion. After each aggregation, if $\hat{\mathbf{y}} \neq \mathbf{y}$, $\mathbf{y}$ will be updated by $\hat{\mathbf{y}}$, and the whole procedure from projection to aggregation will iterate again. Unfortunately, having iterations on each recursion level makes RPA unsuitable for hardware implementations, as it requires complicated control circuitry and memory structures.

In the case of a noisy received vector $\mathbf{y}$ with only one error, for all projected vectors at each recursion level, it can be verified from Algorithm 2 that there exists exactly one error for every level of the recursion and for all projections, which is corrected in level $r = 1$ because FHT decoding guarantees the correction of one error. However, the condition for skipping the remaining iterations is not satisfied (see line 10 in Algorithm 1), and as a result, RPA runs another iteration at this level. This additional iteration is unnecessary because it performs projection, first-order decoding, and aggregation on the already corrected codewords. More generally, and motivated by the above example, if the iteration loops for a recursion level run more than once but stop before reaching

Algorithm 5 The IPA decoding of RM(m, r) codes

Input: The noisy codeword $\mathbf{y}$, m, r, $N_{\max}$
Output: The decoded codeword $\mathbf{c}$

1: $\mathbf{y}_{(1,0)} \leftarrow \mathbf{y}$; $n_{tmp} \leftarrow 1$
2: **for** $j = 1 : N_{\max}$ **do**
3: **for** $l = 1 : r-1$ **do** *--Projection loop*
4: $n_{tmp} \leftarrow n_{tmp} \times (2^m - 1)$
5: **for** $i = 1 : n_{tmp}$ **do**
6: $\mathbf{y}_{tmp} \leftarrow \mathbf{y}_{(\lceil i/(2^m-1) \rceil, l-1)}$
7: $\mathbf{y}_{(i,l)} \leftarrow$ Proj$(\mathbf{y}_{tmp}, mod(i, 2^m - 1), m)$
8: **end for**
9: $m \leftarrow m - 1$
10: **end for**
11: **for** $t : 1 : n_{tmp}$ **do** *--First-order decoding*
12: $\hat{\mathbf{y}}_{(t,r-1)} \leftarrow$ order-1-decoding $(\mathbf{y}_{(t,r-1)}, m)$
13: **end for**
14: **for** $l = r-2 : 0$ **do** *--Aggregation loop*
15: $m \leftarrow m + 1$; $t \leftarrow 2^m - 1$; $n_{tmp} \leftarrow n_{tmp}/t$
16: **for** $i = 1 : n_{tmp}$ **do**
17: $d \leftarrow (i-1) \times t$
18: $\hat{\mathbf{y}}_{(i,l)} \leftarrow$ Agg$(\mathbf{y}_{(i,l)}, \mathbf{y}_{(d+1,l+1)}, \ldots, \mathbf{y}_{(d+t,l+1)}, \hat{\mathbf{y}}_{(d+1,l+1)}, \ldots, \hat{\mathbf{y}}_{(d+t,l+1)})$
19: **end for**
20: **end for**
21: **if** $\mathbf{y}_{(1,0)} = \hat{\mathbf{y}}_{(1,0)}$ **do**
22: **break** *--IPA converges to a fixed point*
23: **end if**
24: $\mathbf{y}_{(1,0)} \leftarrow \hat{\mathbf{y}}_{(1,0)}$
25: **end for**
26: $\mathbf{c} \leftarrow \hat{\mathbf{y}}_{(0,1)}$

$N_{\max}$, the last iteration always runs only to check the stop condition. We call these iterations *ineffective*.

Based on our simulations of various RM codes, at low channel crossover probabilities, more than 50% of internal iterations are ineffective. Motivated by this observation, we present a simplification of RPA by removing iterations on the internal levels of the RPA recursion. Effectively, our proposed iterative projection-aggregation (IPA) algorithm sets $N_{\max} = 1$ for all recursive decoding steps of the RPA algorithm except for the first one. This can be concluded by comparing Algorithm 5, in which the iterative structure of IPA is shown, with Algorithm 1. The difference is that there exists only one iteration loop around the entire function in Algorithm 5. Moreover, the iterative structure of Algorithm 5 is more convenient for a hardware implementation. As we show in Section 5, this reduces complexity and hardware implementation significantly, with a small error-correcting penalty.

It can be shown that the complexity of RPA decoding with internal iterations is $O(n^r(\log_2 n)^{r+1})$. For the IPA algorithm, the complexity is $O(n^r(\log_2 n)^2)$ as we remove the internal iterations. We also show in the Section 5 that the overall calls to the first-order decoder, which is a more practical complexity measure, are decreased significantly.

4. PROPOSED HARDWARE ARCHITECTURE

Our proposed fully parallel IPA architecture, which is shown in Fig. 1, consists of three main components and a control unit. The first component is the *projection*, including $r-1$ levels of the projection for RM(m, r) codes (line 3 of Algorithm 5). The second component, which we call the *first-order decoder*, has parallel decoders for all RM$(m-r+1, 1)$ codes generated in the innermost level of the RPA (line 11 of Algorithm 5). The third component is the *aggregation* unit performing $r-1$ levels of aggregation (line 14 of Algorithm 5).

The *projection* component performs $r-1$ levels of projection, as described in Section 2.1. Each projection level has parallel projection units, each consisting of a *re-ordering* unit (*ROU*) and an *XOR* unit. The re-ordering unit *ROU(m,i)* finds the coordinates for i-th projection of the input vector $\mathbf{y}$ with length of 2^m based on lines 2-11 of Algorithm 2. Then, an *XOR* unit is assigned to each projection branch for performing the sum operations as described in lines 12-13 of Algorithm 2.

The *first-order decoder* component provides first-order decoders (FODs) for all RM$(m-r+1, 1)$ codes, obtained in the innermost level of projection, in parallel. Each *FOD* was designed based on the decoding method proposed in [15], and consists of three sub-units: *FHT*, *Argmax*, and *Generator matrix*. The first unit gives the vector $\mathbf{l}$, which is the result of the FHT on a binary input vector $\mathbf{y}$:

$$\mathbf{l} = (1 - 2\mathbf{y})\mathbf{H}_{2^m}, \tag{2}$$

where the Hadamard matrix $\mathbf{H}_{2^m}$ is

$$\mathbf{H}_{2^m} = \begin{bmatrix} \mathbf{H}_{2^{m-1}} & \mathbf{H}_{2^{m-1}} \\ \mathbf{H}_{2^{m-1}} & -\mathbf{H}_{2^{m-1}} \end{bmatrix} \text{ and } \mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{3}$$

The architecture of *FHT* unit is derived from [16]. The *Argmax* unit finds the index z of the maximum value of $\mathbf{l}$. The output of the *FOD* unit is:

$$\hat{\mathbf{y}} = \hat{\mathbf{x}}\mathbf{G}_{(m,1)}, \tag{4}$$

where $\hat{\mathbf{x}} = \begin{bmatrix} \frac{1-\text{sign}(\mathbf{l}(z))}{2} & \mathbf{z}_{\text{bin}} \end{bmatrix}$ with $\mathbf{z}_{\text{bin}}$ being the binary representation of z, and where $\mathbf{G}_{(m,1)}$ is the generator matrix of RM$(m, 1)$.

The *aggregation* component provides $r-1$ levels of aggregation, each of which has $\frac{\prod_{i=0}^{r-2}(2^{m-i}-1)}{\prod_{i=1}^{j}(2^{(m-r+1)+i}-1)}$ *AGG* units in parallel (line 16 of Algorithm 5), where j denotes the current level of aggregation. As Fig. 1 shows, each *AGG* unit consists of 2^m-1 *RRUM*s (the hardware implementation of Algorithm 4) and one majority voter to aggregate into a n-bit codeword $\hat{\mathbf{y}}$ calculated in line 7 of Algorithm 3. Finally, XOR gates are used to flip the desired bits of input vector $\mathbf{y}$ in $\hat{\mathbf{y}}$ as described in line 7 of Algorithm 3.

The throughput of the decoder is calculated by:

$$\text{Throughput} = \frac{\text{Frequency}}{N_{\text{iter}} N_{\text{cycles/iter}}} \times n, \tag{5}$$

where $N_{\text{iter}} = N_{\max}$ for the minimum throughput and $N_{\text{iter}} = N_{\text{avg}}$ is the average number of iterations for the average throughput. The data path is pipelined to $N_{\text{cycles/iter}}$ stages.

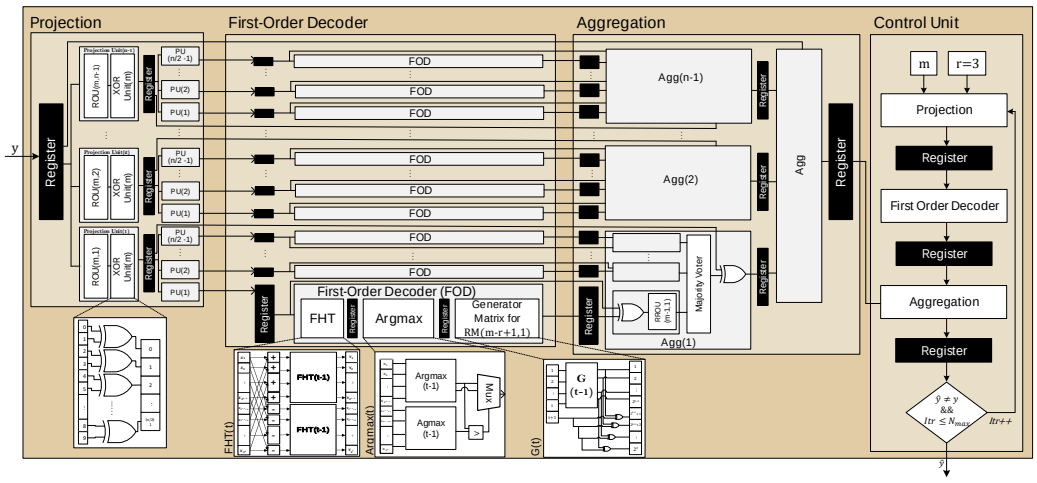

Fig. 1. An example of the proposed hardware architecture for IPA decoding for RM$(m, 3)$ codes.

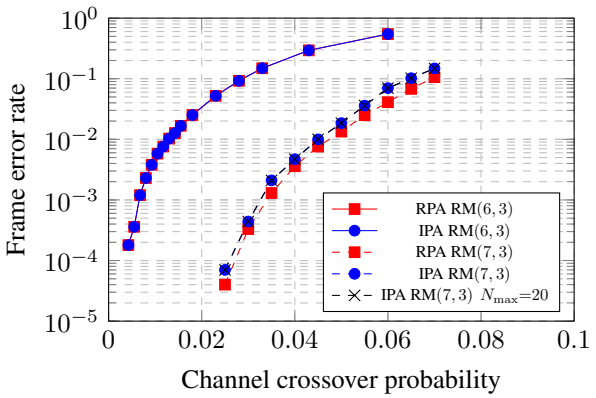

Fig. 2. Frame error rate Comparison between RPA and IPA for RM$(6, 3)$ and RM$(7, 3)$ codes over BSC channels.

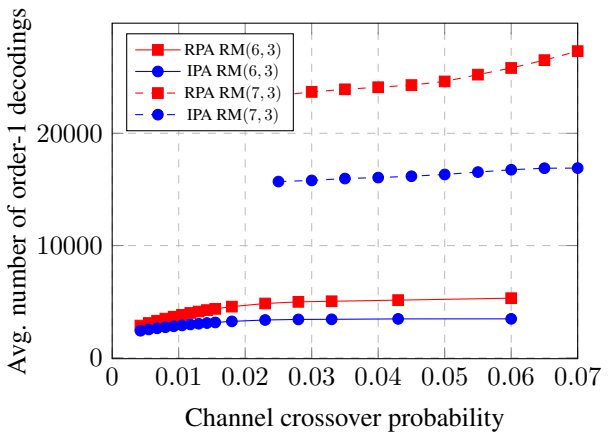

Fig. 3. Comparison of the number of the first-order decoding between RPA and IPA for RM$(6, 3)$ and RM$(7, 3)$ codes.

In particular, $r-1$ and $2(r-1)$ registers are inserted between the projection and aggregation levels, respectively. Additionally, three registers are inserted between the components of the FODs, and one register is used to check the termination condition. As such, we have $N_{\text{cycles/iter}} = 3(r-1)+4$.

5. RESULTS

Simulation results for IPA decoding and RPA decoding for the RM$(6, 3)$ and RM$(7, 3)$ codes over the BSC channel are shown in Fig. 2. We observe that IPA decoding has exactly the same frame error rate (FER) as RPA decoding for RM$(6, 3)$, while there is a minimal error-correcting performance degradation up to 0.005 in terms of channel cross-over probability for RM$(7, 3)$. We also increased $N_{\max}$ to see if this compensates the performance degradation of IPA, but we observed that it unfortunately does not help.

Fig. 3 shows the average number of first-order decodings for IPA and RPA decoding for RM$(6, 3)$ and RM$(7, 3)$ codes over the BSC channel. We observe that the number of the first-order decodings is decreased by up to 40% for RM$(6, 3)$ and up to 50% for RM$(7, 3)$.

Table 1. Post-PAR results for an RM$(6, 3)$ code on a Xilinx Virtex-7 FPGA (xc7vx1140T).

LUTs	$602,111/712,000(84.57\%)$
Flip-flops	$65,699/1,424,000(4.6\%)$
Clock frequency	80 MHz
Min. throughput ($N_{\max}=3$)	171 Mbps
Avg. throughput @ FER=10^{-3}	284 Mbps

We provide post-PAR results of our IPA decoder architecture for RM$(6, 3)$ on a Xilinx Virtex-7 FPGA with a frequency of 80 MHz in Table 1. The resource utilization is high due to the fully parallel nature of the decoder, but the achieved decoding throughput is also relatively high. As there are no other implementations of RPA in the literature, we cannot perform a direct comparison.

6. REFERENCES

[1] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, sep 1954.

[2] S. Kudekar, S. Kumar, M. Mondelli, H. D. Pfister, E. Şaşoğlu, and R. Urbanke, "Reed–Muller codes achieve capacity on erasure channels," *IEEE Trans. Inf. Theory*, vol. 63, no. 7, pp. 4298–4316, July 2017.

[3] O. Sberlo and A. Shpilka, "On the performance of Reed–Muller codes with respect to random errors and erasures," in *Annual ACM-SIAM Symp. on Discrete Algorithms*, 2020, p. 1357–1376.

[4] V. M. Sidel'nikov and A. S. Pershakov, "Decoding of Reed–Muller codes with a large number of errors," *Problemy Peredachi Informatsii*, vol. 28, pp. 80–94, 1992.

[5] J. Bertram, P. Hauck, and M. Huber, "An improved majority-logic decoder offering massively parallel decoding for real-time control in embedded systems," *IEEE Trans. Commun.*, vol. 61, no. 12, pp. 4808–4815, Dec. 2013.

[6] M. Hiller, L. Kurzinger, G. Sigl, S. Muelich, S. Puchinger, and M. Bossert, "Low-area reed decoding in a generalized concatenated code construction for PUFs," in *IEEE Computer Society Annual Symp. on VLSI*, July 2015.

[7] I. Dumer, "Recursive decoding and its performance for low-rate Reed–Muller codes," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 811–823, May 2004.

[8] I. Dumer and K. Shabunov, "Soft-decision decoding of Reed–Muller codes: Recursive lists," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1260–1266, Mar. 2006.

[9] S. A. Hashemi, N. Doan, M. Mondelli, and W. J. Gross, "Decoding Reed–Muller and polar codes by successive factor graph permutations," in *Int. Symp. on Turbo Codes & Iterative Inf. Proc. (ISTC)*, Dec. 2018.

[10] E. Santi, C. Häger, and H. D. Pfister, "Decoding Reed–Muller codes using minimum-weight parity checks," in *IEEE Int. Symp. Inf. Theory (ISIT)*. June 2018, IEEE.

[11] B. Sakkour, "Decoding of second order Reed–Muller codes with a large number of errors," in *IEEE Inf. Theory Workshop*, Oct. 2005.

[12] A. Thangaraj and H. D. Pfister, "Efficient maximum-likelihood decoding of Reed–Muller RM(m-3,m) codes," in *Int. Symp. Inf. Theory (ISIT)*, June 2020.

[13] M. Ye and E. Abbe, "Recursive projection-aggregation decoding of Reed–Muller codes," *IEEE Trans. Inf. Theory*, vol. 66, no. 8, pp. 4948–4965, Aug. 2020.

[14] Mengke Lian, Christian Hager, and Henry D. Pfister, "Decoding Reed–Muller codes using redundant code constraints," in *IEEE Int. Symp. Inf. Theory (ISIT)*, June 2020.

[15] Y. Be'ery and J. Snyders, "Optimal soft decision block decoders based on fast Hadamard transform," *IEEE Trans. Inf. Theory*, vol. 32, no. 3, pp. 355–364, May 1986.

[16] A. Agrawal, R. Bairathi, and A. Joshi, "FPGA implementation of 4-point and 8-point fast Hadamard transform," *International Journal of Computer Applications*, vol. 124, no. 3, pp. 23–28, Aug. 2015.