An Ultimate-Shannon-Limit-Approaching Gbps Throughput Encoder/Decoder System

S. Jiang, P. W. Zhang, F. C. M. Lau and C.-W. Sham

***Abstract*—A turbo-Hadamard code (THC) is a type of low-rate channel code with capacity approaching the ultimate Shannon limit, i.e., -1.59 dB. In this paper, we investigate the hardware design of a turbo-Hadamard encoder/decoder system. The entire system has been implemented on an FPGA board. A bit error rate (BER) of 10^{-5} can be achieved at $E_b/N_0 = 0.04$ dB with a throughput of 3.2 Gbps or at $E_b/N_0 = -0.45$ dB with a throughput of 1.92 Gbps.**

I. Introduction

Turbo code, low-density parity-check (LDPC) code, and polar code are the most widely studied and implemented error-correction codes over the last two-and-a-half decades because of their capacity-approaching capabilities [1], [2], [3], [4], [5], [6]. Both turbo code and LDPC code, when used in conjunction with Hadamard code, have been shown to achieve performance very close to the ultimate Shannon limit, i.e., -1.59 dB [7], [8]. Another code with comparable performance is the concatenated zig-zag Hadamard code [9]. Such codes can be used in a multi-user environment such as in a code-division multiple-access or an interleave-division multiple-access (IDMA) [10] system. The codes can also be used to carry embedded messages in point-to-point wireless/wired communications. In [7], [8], [9], simulation results of the aforementioned codes with code lengths of 3.5 Mbits and 22 Mbits have been performed. However, the hardware implementation of the codes has never been investigated. One reason is that the code lengths quoted above are extremely long, making the hardware complexity prohibitively high.

In this paper, we investigate the hardware implementation of turbo-Hadamard code (THC) using FPGA. Our aim is to explore an efficient implementation of an encoder/decoder system that can accomplish high throughput with reasonable complexity. To achieve our goal, we propose a multiple sub-decoders system. By allowing the sub-decoders in the system to decode different codewords simultaneously, we can increase the throughput multiple times. Another challenge to overcome is to design interleavers that can minimize the decoding latency without sacrificing bit error rate (BER) performance. For this issue, we propose applying a fixed inter-window shuffle (FIWS) interleaver [11]. Moreover, each component of the encoding/decoding system is fully optimized to achieve high hardware-utilization rate. Arrangement is made to ensure the data corresponding to different codewords are in an orderly manner such that they can be received/stored/retrieved/processed efficiently at the decoder. Last but not the least, we construct rate-adaptive THCs by puncturing the parity bits of the code.

Sect. II briefly reviews the THC. Sect. III describes our overall design of the THC encoder/decoder system and evaluates the latency of each sub-decoder. Sect. IV shows the FPGA implementation results, including hardware utilization, throughput and bit error rate performance. Finally, Sect. V provides some concluding remarks.

S. Jiang, P. W. Zhang and F. C. M. Lau are with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong (email: sheng.jiang@connect.polyu.hk, pengwei.zhang@connect.polyu.hk, francis.cm-lau@polyu.edu.hk).

C.-W. Sham is with Department of Computer Science, The University of Auckland, New Zealand (email: b.sham@auckland.ac.nz).

II. Turbo-Hadamard Code (THC)

A Hadamard code of order-r can be constructed from a Hadamard matrix of the same order. An order-r Hadamard matrix $\boldsymbol{H}_n$ where $n = 2^r$ can be constructed recursively using

$$\boldsymbol{H}_n = \begin{bmatrix} +\boldsymbol{H}_{n/2} & +\boldsymbol{H}_{n/2} \\ +\boldsymbol{H}_{n/2} & -\boldsymbol{H}_{n/2} \end{bmatrix} \quad (1)$$

with $\boldsymbol{H}_1 = [+1]$. The codeword set of an order-r Hadamard code is denoted by $\{\pm\boldsymbol{h}^j : j = 0, 1, 2, \ldots, 2^r - 1\}$, where $+\boldsymbol{h}^j$ and $-\boldsymbol{h}^j$ represent the $(j+1)$-th columns of $+\boldsymbol{H}_n$ and $-\boldsymbol{H}_n$, respectively. Denoting the code-bit positions by $\{0, 1, 2, \ldots, 2^r - 1\}$, the information bits are located at $\{0, 1, 2, 4, 8, \ldots, 2^{r-1}\}$ and the parity-check bits are at the remaining positions. Fig. 1 shows the encoder block diagram and code structure of a convolutional-Hadamard code and Fig. 2 shows the code structure of a THC [7]. A convolutional-Hadamard code is a concatenation of a single-parity-check code, an S-state recursive convolutional code and a Hadamard code. A THC is the combination of a number of convolutional-Hadamard codes, say M convolutional-Hadamard codes, carrying the same but interleaved message bits.

We refer to Fig. 1(b). In a convolutional-Hadamard code, each block message $\boldsymbol{D}$ contains L bits and is segmented into K sub-blocks where each sub-block $\boldsymbol{d}_k$ $(k = 1, 2, \ldots, K)$ contains r bits, i.e., $L = rK$. The parity bit $q_k{}'$ of $\boldsymbol{d}_k$ is computed and sent through an S-state rate-$1/2$ systematic recursive convolutional encoder producing convolutional codes denoted as $(q_k{}', q_k)$. Finally $(\boldsymbol{d}_k, q_k)$ is encoded into an order-r Hadamard code $\boldsymbol{c}_k = (\boldsymbol{d}_k, q_k, \boldsymbol{p}_k)$, where $\boldsymbol{p}_k$ represents the parity bits in the Hadamard code. The code length and code rate of an order-r THC are therefore given by $l = rK + MK(2^r - r)$ and $r_c = \frac{rK}{rK+MK(2^r-r)} = \frac{r}{r+M(2^r-r)}$, respectively.

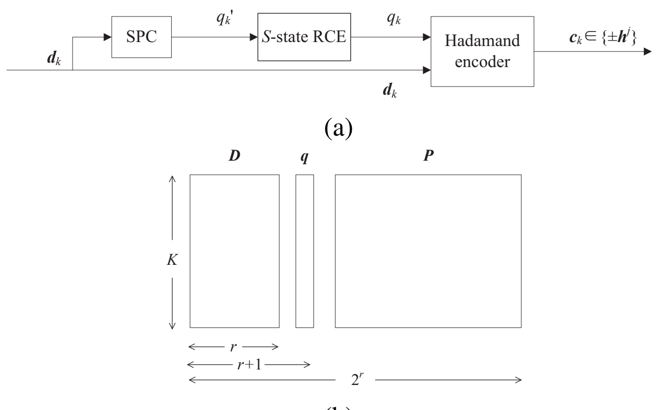

Fig. 1: Convolutional-Hadamard code. (a) Encoder block diagram and (b) code structure. SPC: single-parity check; RCE: recursive convolutional encoder.

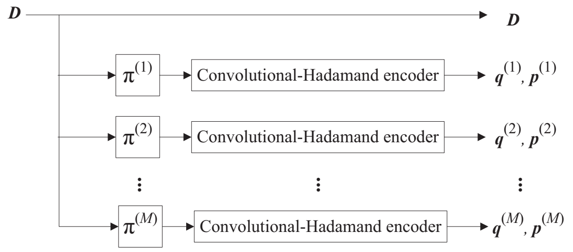

Fig. 2: Structure of turbo-Hadamard code.

Suppose a convolutional-Hadamard codeword $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, ..., \boldsymbol{c}_K]$ is transmitted in an additive white Gaussian noise (AWGN) channel with noise variance σ^2 and a vector $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, ..., \boldsymbol{x}_K]$ is received. Considering the *a-posteriori-probability* (APP) decoding of convolutional-Hadamard code, the logarithm-likelihood ratio (LLR) of the ith bit in the kth sub-block is given by [7]

$$L_k[i] = \log \frac{\sum\limits_{H[i,j]=\pm 1} \gamma_k(\pm \boldsymbol{h}^j)\alpha(\boldsymbol{s}_k)\beta(\boldsymbol{s}_{k+1})}{\sum\limits_{H[i,j]=\mp 1} \gamma_k(\pm \boldsymbol{h}^j)\alpha(\boldsymbol{s}_k)\beta(\boldsymbol{s}_{k+1})} \tag{2}$$

where $\gamma_k(\pm \boldsymbol{h}^j) = \Pr(\boldsymbol{x}_k | \boldsymbol{c}_k = \pm \boldsymbol{h}^j)$ is the *a priori* information and is calculated based on the channel log-likelihood ratio (LLR) $\boldsymbol{L}_k = \frac{2\boldsymbol{x}_k}{\sigma^2}$; $\alpha(\boldsymbol{s}_k)$ and $\beta(\boldsymbol{s}_{k+1})$ are calculated using the forward-backward recursion algorithm in the Bahl-Cocke-Jelinek-Raviv (BCJR) decoder. Consequently, a convolutional-Hadamard decoder consists of three main stages.

1) Generate the *a priori* information $\{\gamma_k(\pm \boldsymbol{h}^j)\}$ based on the input $2\boldsymbol{x}_k/\sigma^2$.
2) Perform BCJR decoding based on $\{\gamma_k(\pm \boldsymbol{h}^j)\}$.
3) Update the *a posteriori* LLR of the information bits based on the results from the first two stages.

The minimum hardware requirement of a THC decoder consists of one convolutional-Hadamard decoder, M interleavers, plus appropriate memory storage and control logics. For such an implementation, the same convolutional-Hadamard decoder will be used M times to complete one THC decoding iteration. In a particular THC decoding iteration, the most updated *a posteriori* LLR information of the information bits (output from the latest convolutional-Hadamard decoder), subtracted by the extrinsic information of current component code from the last THC decoding iteration, replaces the corresponding values in $2\boldsymbol{x}_k/\sigma^2$ and becomes the input of the current component code.

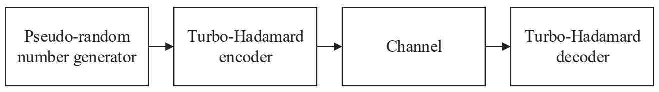

Fig. 3: Data flow of the THC encoder/decoder system.

III. Decoder Design

Fig. 3 shows the data flow of our THC encoder/decoder system. As the implementations of the encoder and channel simulator are relatively straightforward, we only focus on the decoder design. Our THC decoder design consists of M convolutional-Hadamard decoder (each called a sub-decoder). Moreover, consecutive sub-decoders (also the Mth one with the first one) are connected via an appropriate interleaver. Compared with the design using only one convolutional-Hadamard decoder, our design can decode M THC codewords simultaneously in a pipeline manner using the same number of interleavers, thus increasing the throughput by M times. Moreover, the latency can be reduced because simpler control logics can be used. Fig. 4 shows an $M = 3$ THC decoder.

Referring to lower part of Fig. 4, a sub-decoder consists of three main processing units: fast-Hadamard-transform (FHT), BCJR and dual FHT (DFHT). We use a fast Hadamard transform (FHT) to compute the *a priori* information $\{\gamma_k(\pm \boldsymbol{h}^j)\}$ and a DFHT (also called a-posteriori-probability-FHT (APP-FHT)) to compute the *a posteriori* LLR of the information bits because both transforms [7] are simple to be implemented on hardware. Details of each processing unit is described below.

1) (i) The output LLRs of the message bits from the *previous* sub-decoder (after interleaving), (ii) the extrinsic LLRs of the message bits produced by the current sub-decoder in the previous iteration, and (iii) channel LLRs of the parity bits of the current sub-decoder (stored in RAM), are input to the sub-decoder. The extrinsic LLRs of the current sub-decoder are subtracted from the output LLRs from the previous sub-decoder. Then together with the channel LLRs of the parity bits, these signals are input in a pipeline manner to the FHT unit. The FHT unit calculates the *a priori* information $\{\gamma_k(\pm \boldsymbol{h}^j)\}$. Note that the FHT unit contains r stages. After each stage, the number of quantization bits in FHT unit is increased by 1 to avoid data overflow.
2) According to the encoding of the convolutional-Hadamard code, the outputs of the FHT unit represent the probabilities of a transition between the states in a convolutional code based on the trellis of the code. These outputs are sent to the BCJR unit, which is responsible for calculating $\{\gamma_k(\pm \boldsymbol{h}^j)\alpha(\boldsymbol{s}_k)\beta(\boldsymbol{s}_{k+1})\}$ shown in (2), where $\alpha(\boldsymbol{s}_k)$ and $\beta(\boldsymbol{s}_{k+1})$ are calculated by forward and backward recursion.
3) Once the BCJR unit finishes computing all trellises, its output is passed to the DFHT unit to compute the *a posterior* LLRs in (2).

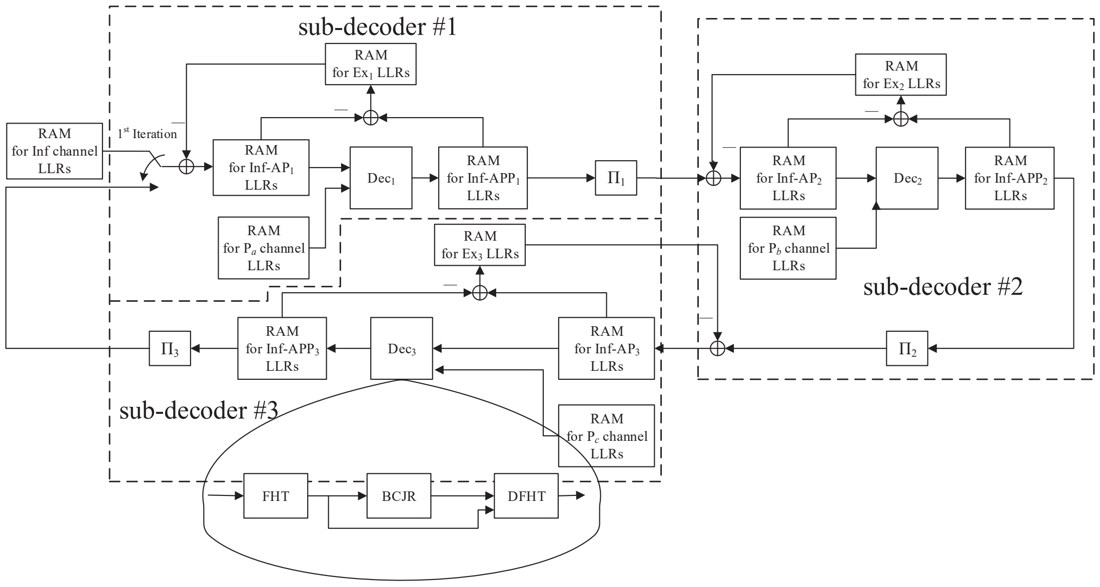

Fig. 4: Overall Decoder Structure.

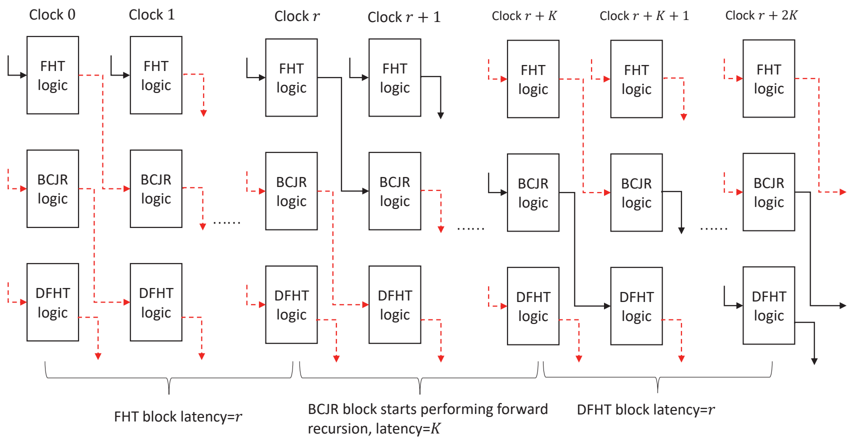

Fig. 5: Pipeline structure within a sub-decoder. Solid line represents valid input/output, dash line represents invalid input/output.

4) The output data from the DFHT unit (the updated LLRs) are interleaved and passed to the next sub-decoder.
5) The extrinsic LLRs of the message bits in this iteration are generated and stored at the same time.

The pipeline illustration of a sub-decoder is shown in Fig. 5. At clock 0, LLRs are input to the FHT unit. It takes K clock cycles to completely input all LLRs. At clock r, the FHT unit outputs the first valid trellis information to the BCJR unit and the BCJR unit starts performing forward recursion and calculating $\alpha(s_1), \alpha(s_2), \ldots, \alpha(s_K)$. At clock $r+K$ the forward recursions for all $\alpha(s_k), k = 1, 2, ..., K$ are completed and backward recursions for $\beta(s_{K+1}), \beta(s_K), ..., \beta(s_2)$ begins. The BCJR unit starts outputing $\gamma_k(\boldsymbol{c}_h)\alpha(s_k)\beta(s_{k+1}), k = K, K-1, ..., 1$ to the DFHT unit in the next K clock cycles. Similar to the FHT unit, the latency in the DFHT unit is also r, so the total time taken by a sub-decoder to compute all calculations is $2r + 2K$ clock cycles. Note that the outputs of the DFHT unit are stored in an interleaver and are to be used by the next sub-decoder.

The interleavers used in THC not only shuffle the order of the message bits, but also change the subsequent SPCs and Hadamard codes. The implementation of large-size random interleavers increases the latency/complexity of encoder/decoder significantly. Our goals are to keep the random property of the interleavers, and to perform interleaving in parallel and thus reducing latency/complexity the latency/complexity of the whole system. The main idea of performing interleaving in parallel is by dividing a size-N interleaver into m windows of size W (i.e., $N = mW$). During encoding/decoding, contention may happen if two or more units try to access the same memory window. Fixed inter-window shuffle (FIWS) is a type of contention-free interleaver [11]. FIWS interleavers fix the inter-window shuffle pattern and m different sub-interleaver patterns need to be designed for m windows. Usually FIWS interleavers are not suitable for turbo or turbo-like codes. The reason is that it is equivalent to dividing a big interleaver into m smaller interleavers. The whole codeword is also divided into m smaller concatenated codewords, leading

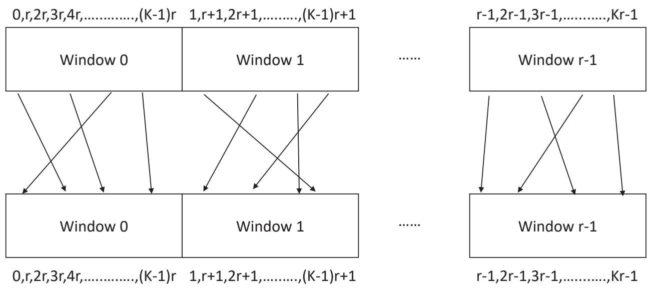
0,r,2r,3r,4r,............,(K-1)r
1,r+1,2r+1,.........,(K-1)r+1
r-1,2r-1,3r-1,..........,Kr-1
Window 0
Window 1
Window r-1
......

Fig. 6: Illustration of an FIWS interleaver used.

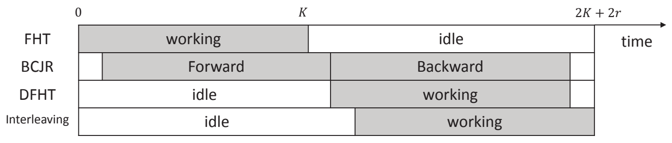

Fig. 7: Component utilization of each sub-decoder

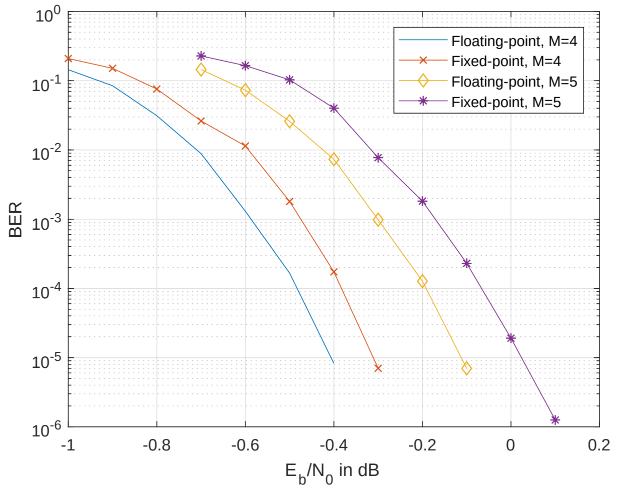

Fig. 8: BER curves of the THC code with $M = 4, 5$ component codes.

to an increase of low-weight codewords. However in a turbo-Hadamard code, the situation is different and hence FIWS interleavers become applicable. (Details are not given due to shortage of space.)

For a THC, we have $m = r$ and $W = K$. Fig. 6 shows the structure of the FIWS interleaver used in our system. (To compute SPCs more efficiently in the encoder, the bit order in Fig. 6 is different from that in [11], [12].) In addition, each FIWS interleaver consists of r depth-K RAMs and r depth-K ROMs. After the ith sub-decoder ($i = 1, \ldots, M$) has completed storing the APP information (from the outputs of DFHT unit) to the RAMs in natural order (see Fig. 4), the RAMs start sending the information to the next sub-decoder in an interleaved order, the pattern of which is stored in the r ROMs. Our simulation results also indicate that there is no obvious performance degradation between FIWS random interleavers and traditional random interleavers.

IV. Implementation Results and Analysis

We consider a turbo-Hadamard encoder/decoder system with the following parameters.

- Number of component codes $M = 3, 4, 5$
- Each message block $\boldsymbol{D}$ contains $L = 4095$ message bits
- Order of Hadamard code is $r = 7$
- Number of sub-blocks per message $K = \frac{L}{r} = 585$
- Number of convolutional code states $S = 2$
- Code length $l = 216450, 287235$ and 358020
- Code rate $r_c \approx 0.0189, 0.0143$ and 0.0114
- Channel LLRs are quantized by $N_{ch} = 6$ bits
- Inputs to FHT unit are quantized by $N_{FHT} = 10$ bits
- Data in BCJR unit are quantized by $N_{BCJR} = 7$ bits
- Data in DFHT unit are quantized by $N_{DFHT} = 10$ bits
- 10 iterations used for decoding each codeword
- Operating frequency $f_c = 100\text{MHz}$
- FPGA board Xilinx Virtex UltraScale+ VCU118

Fig. 7 shows the working flow of each components inside the sub-decoder. Denoting the hardware utilization rate of the FHT, BCJR, DFHT and interleaver as U_{FHT}, U_{BCJR}, U_{DFHT} and U_π, respectively, we have

$$\begin{aligned} U_{\text{FHT}} = U_{\text{DFHT}} = U_\pi &= \frac{K}{2K+2r} = \frac{1}{2}\frac{1}{1+\frac{r}{K}} \approx \frac{1}{2}; \\ U_{\text{BCJR}} &= \frac{2K}{2K+2r} = \frac{1}{1+\frac{r}{K}} \approx 1. \end{aligned} \tag{3}$$

The BCJR unit works almost all the time during decoding while the other units operate approximately half of the time.

Recall that the THC has a trellis length of K and each sub-decoder takes $2K+2r$ clock cycles to complete all operations. Since the number of component codes is M, the decoding latency for one iteration is therefore $2M(K+r)$. Assuming the number of iterations is I, we need $2IM(K+r)$ clock cycles to decode one THC. With an operating frequency of f_c and a codeword length of l, the number of code bits decoded in one second equals $lf_c/2IM(K+r)$. Moreover, our decoder is designed to decode M codewords in parallel. Therefore, the throughput T of our decoder is given by

$$\begin{aligned} T &= \frac{M \times lf_c}{2IM(K+r)} = \frac{[rK + MK(2^r - r)]f_c}{2I(K+r)} \\ &= \frac{[(1-M)r + 2^r M]f_c}{2I(1+\frac{r}{K})} \approx \frac{2^{r-1}Mf_c}{I} \end{aligned}$$

where the approximation is made because $r/K \ll 1$ and $(M-1)r \ll 2^r M$. Based on the parameters we used, $T \approx 1.92\text{Gbps}, 2.56\text{Gbps}$ and 3.2Gbps for $M = 3, 4$ and 5, respectively.

Table I shows the FPGA resources utilization for the turbo-Hadamard encoder/decoder system with $M = 3, 4$ and 5. The look-up tables (LUT) used increase little from $M = 3$ to $M = 4$. For codes $M = 5$, LUT usage increases a lot because of the implementation of multi-rate codes (see below). The block RAMs (BRAM) used is almost proportional to the code length.

Fig. 8 shows the bit-error-rate (BER) results of $M = 4$ and $M = 5$ turbo-Hadamard code under floating-point decoder and fixed-point decoder. Both codes show a performance loss of no more than 0.15 dB at $\text{BER} = 10^{-5}$. For example, at a BER of 10^{-5}, the fixed-point implementation of $M = 4$ THC requires an E_b/N_0 value of -0.3 dB, which is within 0.1 dB that of the floating-point result.

M	Code Length	Look-up Table	Look-up Table RAM	Flip-Flop	Block RAM	IO	Global Clock Buffer	Mixed-Mode Clock Manager
3	217,620	413759	2741	146481	720.5	13	7	1
4	287,820	449648	3271	191756	1146.5	13	8	1
5	358,020	614993	3827	240610	1459.5	13	9	1

TABLE I: Resources utilization of THC with $M = 3, 4, 5$

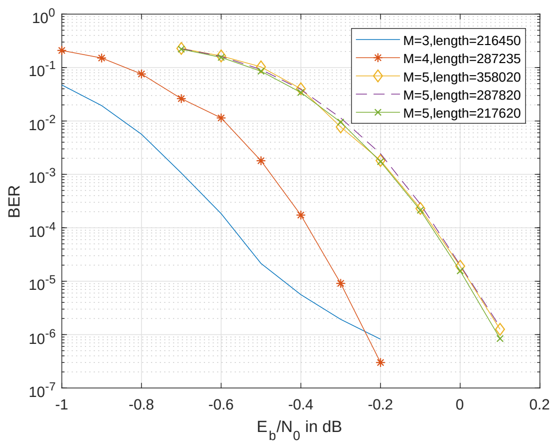

Fig. 9: BER curves of the THC code with $M = 3, 4, 5$ component codes and punctured THC code with $M = 5$ on FPGA.

Index	Code Rate r_c	Message Length N	Code Length	Punctured bits
0	0.0114	4,095	358,020	0 bit
1	0.0142	4,095	287,820	24 bits
2	0.0188	4,095	217,620	48 bits

TABLE II: Number of punctured bits per Hadamard code.

Fig. 9 plots BER curves of the THC for $M = 3, 4, 5$ implemented on the FPGA board. The result shows that THC with $M = 3$ achieves a better BER than $M = 4, 5$ in the low E_b/N_0 regime, but suffers from an error floor at a BER of 10^{-6}. We also consider puncturing to achieve rate-compatible THC. Since only parity bits can be punctured, the puncturing of turbo-Hadamard code is basically the puncturing of Hadamard code, which has been discussed in [13]. In [13], good puncturing patterns are designed by (i) maximizing the minimum Hamming distance or (ii) minimizing the correlations of all codewords. We use the latter method as it has shown better performance. We optimize the puncturing patterns for an $M = 5$ ($l = 358020$) turbo-Hadamard code and generate codes with $l = 287820$ and 217620, respectively. We choose those code lengths because the punctured codes will have similar code lengths and code rates with $M = 4$ ($l = 287235$) and $M = 3$ ($l = 216450$) THCs. The puncturing parameters are shown in Table II and the puncturing patterns are listed in Table III. The punctured THC with $M = 5$ is also implemented on FPGA board (see Table I). The BER results in Fig. 9 shows that the THCs after puncturing have very close performance compared with the original THC.

V. Conclusion

In this paper, an efficient implementation of an ultimate-Shannon-limit approaching encoder/decoder system has been explored using FPGA. It is applicable to future wireless and Internet of Things systems in which a very large number of

Index	Code Rate r_c	Puncturing patterns (Numbers range from 0 to 127)
0	0.0114	NA

1	0.0142	19 32 33 40 42 43 47 48
		51 62 65 68 70 77 79 84
		89 95 97 99 117 118 122 125

2	0.0188	5 8 9 10 12 13 14 19 20 22 23 25 32
		37 40 44 45 46 47 49 51 53 54 56 58 61
		62 66 68 75 78 82 84 86 88 90 93 94 104
		107 111 114 118 120 122 123 124 125

TABLE III: Punctured bits in a Hadamard code.

devices are trying to communicate with an access point at the same time, making the E_b/N_0 values for all transmissions extremely small. The system implemented is based on turbo-Hadamard codes with different lengths and can achieve throughputs up to 3.2 Gbps. Our FPGA implementation design is generic. With some modifications, they can be applied to the encoding/decoding other ultimate-Shannon-limit-approaching codes such as LDPC-Hadamard and concatenated zig-zag Hadamard codes.

References

[1] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. on Commun.*, vol. 44, pp. 1261–1271, 1996.

[2] M. Zhao, X. Zhang, L. Zhao, and C. Lee, "Design of a High-Throughput QC-LDPC Decoder With TDMP Scheduling," *IEEE Transactions on Circuits and Systems II*, vol. 62, no. 1, pp. 56–60, Jan 2015.

[3] I. Tsatsaragkos and V. Paliouras, "Approximate Algorithms for Identifying Minima on Min-Sum LDPC Decoders and Their Hardware Implementation," *IEEE Transactions on Circuits and Systems II*, vol. 62, no. 8, pp. 766–770, Aug 2015.

[4] A. K. Pradhan, A. Thangaraj, and A. Subramanian, "Construction of near-capacity protograph LDPC code sequences with block-error thresholds," *IEEE Trans. on Commun.*, vol. 64, pp. 27–37, 2016.

[5] Q. Lu, J. Fan, C. W. Sham, W. M. Tam, and F. C. M. Lau, "A 3.0 Gb/s throughput hardware-efficient decoder for cyclically-coupled QC-LDPC codes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 1, pp. 134–145, Jan 2016.

[6] V. Guruswami and P. Xia, "Polar Codes: Speed of Polarization and Polynomial Gap to Capacity," *IEEE Trans. on Inform. Theory*, vol. 61, no. 1, pp. 3–16, Jan 2015.

[7] L. Ping, W. K. Leung, and K. Y. Wu, "Low-rate turbo-Hadamard codes," *IEEE Trans. on Inform. Theory*, vol. 49, no. 12, pp. 3213–3224, Dec 2003.

[8] G. Yue, L. Ping, and X. Wang, "Generalized Low-Density Parity-Check Codes Based on Hadamard Constraints," *IEEE Transactions on Information Theory*, vol. 53, no. 3, pp. 1058–1079, March 2007.

[9] W. K. R. Leung, G. Yue, L. Ping, and X. Wang, "Concatenated zigzag Hadamard codes," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1711–1723, April 2006.

[10] Li Ping, Lihai Liu, Keying Wu, and W. K. Leung, "Interleave division multiple-access," *IEEE Transactions on Wireless Communications*, vol. 5, no. 4, pp. 938–947, April 2006.

[11] A. Nimbalker, T. K. Blankenship, B. Classon, T. E. Fuja, and D. J. Costello, "Contention-Free Interleavers for High-Throughput Turbo Decoding," *IEEE Trans. on Commun.*, vol. 56, pp. 1258–1267, 2008.

[12] T. Blankenship, B. Classon, and V. Desai, "High-throughput turbo decoding techniques for 4G," *Proc. Int. Conf. 30 Wireless and Beyond*, pp. 137–142, 2002.

[13] S. Jiang, F. C. M. Lau, W. M. Tam, and C. W. Sham, "Design and error performance of punctured Hadamard codes," in *2017 23rd Asia-Pacific Conference on Communications (APCC)*, Dec 2017, pp. 1–5.