Design of Fast-SSC Decoder for STT-MRAM Channel

Jianming Cui[1], Zengxiang Bao[1(✉)], Xiaojun Zhang[1,2], Hua Guo[1], and Geng Chen[1]

[1] Shandong University of Science and Technology, Qingdao 266590, China
bzx5699@163.com

[2] State Key Laboratory of High-End Server and Storage Technology, Jinan, China

Abstract. In order to achieve fast decoding and improve the throughput, this paper uses the polar code to encode the spin transfer torque MARAM (STT-MRAM) channel. Based on the Fast-SSC algorithm, a (256, 220) hardware architecture is designed, including the controller, processing element, Kronecker product and memory module. This paper reduces the complexity of data process by splitting the data of nodes, and reduces the memory bandwidth by increasing the reusability of data. The decoder is synthesized on Stratix V 5SGXEA7N2F45C2, the decoding latency is 0.68us, and it can achieve 375 Mbps at 167 MHz.

Keywords: STT-MRAM channel · Polar code · Fast-SSC decoder

1 Introduction

With the development of the memory fields, STT-MRAM is regarded as the most promising replacement for dynamic random access memory (DRAM) in the future due to nanosecond high-speed read and write. In this paper, the spin transfer torque MRAM (STT-MRAM) [1, 2] channel is studied, combined with its reliability, which is susceptible to process changes and thermal fluctuations leading to writing errors, read interference and read decision errors [3, 4]. Researches have shown that for a (1024, 512) polar code under an additive white Gaussian noise (AWGN) and binary phase-shift keying (BPSK) modulation, it has a performance gain of 0.3–0.7 dB than low-density parity-check (LDPC). Polar code with excellent error correction performance is selected for channel coding.

The decoding latency of polar codes increases significantly with the increase of code word length [5], which increased hardware consumption. The successive-cancellation SC [6, 7] decoder hardware architecture, however, due to the serial nature, the hardware resource utilization rate is low and suffers from the high decoding latency. Successive-Cancellation list (SCL) [8] will greatly improve the decoding performance, but the decoding complexity also increases. W. J. Gross et al. proposed the Fast-SSC algorithm [9–11], and they proposed the hardware architecture with the dedicated instruction set. In this paper, polar codes is adopted as coding scheme with Fast-SSC algorithm. The decoder is constructed with (256, 220) polar codes and 32 processing elements (PEs) are used in this paper, which not only achieves better parallelism and high throughput, but also greatly reduces hardware resource consumption.

Q. Wu et al. (Eds.): WiSATS 2020, LNICST 358, pp. 566–575, 2021.
https://doi.org/10.1007/978-3-030-69072-4_46

2 Preliminary

2.1 STT-MRAM Channel

The basic storage unit of STT-MRAM is Magnetic Tunnel Junction (MTJ), control and access to MTJ through N-MOS transistors. In this paper, the Binary Asymmetric Channel - Gaussian Mixture Channel (BAC-GMC) series channel model is used to simulate the STT-MRAM magnetic channel (see Fig. 1), and the writing error P_0, P_1 and read interference error P_r of the STT-MRAM channel are simulated by combining the transition probabilities of the BAC channel.

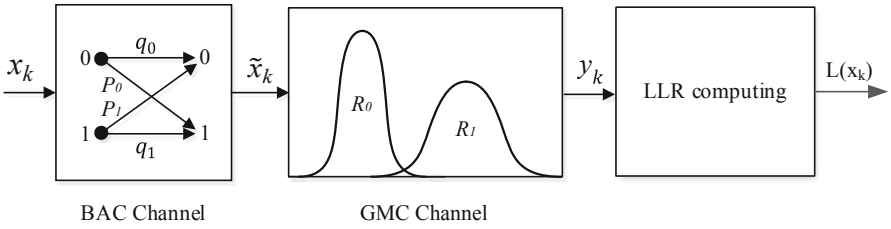

Fig. 1. BAC-GMC channel model

In the GMC channel, R_0 and R_1 present the low-resistance state and the high-resistance state of the MTJ unit respectively, so that R_0 and R_1 follow the Gaussian distribution of mean value μ_0, μ_1 and variance σ_0^2, σ_1^2 respectively.

$$y_k = \begin{cases} R_0 \ if \ (\tilde{x}_k = 0) \\ R_1 \ f \ (\tilde{x}_k = 1) \end{cases} \tag{1}$$

Considering channel symmetry, this paper defines $\alpha(x_k)$ as the LLR of the data bit x_k in the k-th memory cell:

$$\alpha(\tilde{x}_k) = ln(\sigma_0/\sigma_1) + ln((p_0 + q_1)/(q_0 + p_1)) - (y_k - \mu_1)^2/(2\sigma_1^2) + (y_k - \mu_0)^2/(2\sigma_0^2) \tag{2}$$

$$\alpha(x_k) = ln((p_0/(p_0 + q_1))e^{L(\tilde{x}_k)} + (q_0/(q_0 + p_1))) - ln((q_1/(p_0 + q_1))e^{L(\tilde{x}_k)} + (p_1/(q_0 + p_1))) \tag{3}$$

3 Decoding Algorithm

As shown in Fig. 2, V_p represents the parent node of V, V_l and V_r represent the left and right child nodes of V respectively. When the node V is activated, it calculates α_v through (6), and transmits the result α_{vl} to V_l. If V_l is not leaf node, V_l is activated and the f operation is repeated until the left child is leaf node. If V_l is leaf node, β_{vl} is obtained by performing hard decision on α_{vl}. Then transmits it to its parent node V, calculate α_{vr} by (5), and transmits α_{vr} to V_r. If V_r is leaf node, perform hard decision on α_{vr}, calculate bit estimates β_{vr}, and transmits it to V. Further, according to (7), the c

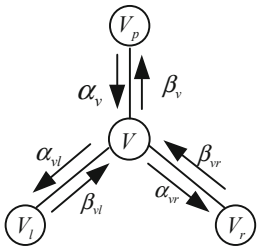

Fig. 2. Fast-SSC algorithm decoding rules

operation is performed to calculate β_v, and transmitted it to V_p. Otherwise, it is activated like the node V.

$$\alpha_{vl}^{(2i-1)}(\alpha_1^N, \beta_1^{2i-2}) = f(\alpha_{N/2}^{(i)}(\alpha_1^{N/2}, \beta_{1,0}^{2i-2} \oplus \beta_{1,\ell}^{2i-2}), \alpha_{N/2}^{(i)}(\alpha_{N/2+1}^N, \beta_{1,\ell}^{2i-2})) \quad (4)$$

$$\alpha_{vr}^{(2i)}(\alpha_1^N, \beta_1^{2i-1}) = g(\alpha_{N/2}^{(i)}(\alpha_1^{N/2}, \beta_{1,0}^{2i-2} \oplus \beta_{1,\ell}^{2i-2}), \alpha_{N/2}^{(i)}(\alpha_{N/2+1}^N, \beta_{1,\ell}^{2i-2}), \beta_{2i-1}) \quad (5)$$

Fast-SSC combines the characteristics of the frozen bit locations without traversing the decoding tree. It contains Rate-0, Rate-1, REP and single-parity check (SPC) nodes [12].

3.1 Rate-0, Rate-1 Nodes

The leaves of the Rate-0 node are all frozen bits, which means the bit estimate of the Rate-0 node is an all-zero vector.

The leaves of the Rate-1 node are all information bits, its estimate can be obtained by hard decision as

$$\beta_v[i] = \begin{cases} 0, \text{ when } \alpha_v[i] \geq 0; \\ 1, \text{ otherwise.} \end{cases} \quad (6)$$

3.2 REP Node

The last leaf of the REP node is an information bit, and the others nodes are frozen bits. It has only two decoding results, all-zero and all-one sequence. Its estimate can be calculated by

$$\beta_v = \begin{cases} 0, \text{ when } (\sum_{i=0}^{N_v-1} \alpha_v[i]) \geq 0; \\ 1, \text{ otherwise.} \end{cases} \quad (7)$$

3.3 SPC Node

The first leaf of the SPC node is a frozen bit, and the others are information bits, it is necessary to perform threshold detection and get the parity check by

$$parity = \bigoplus_{i=0}^{N_v-1} \beta_v[i] \quad (8)$$

Then, if the parity constraint is satisfied, the estimated bit vector is generated by reusing the β_v calculated above. Otherwise, the least reliable bit is founded and flipped if the parity constraint is not satisfied by (9)(10).

$$i = \arg \quad \min(|\alpha_v[i]|) \tag{9}$$

$$\beta_v[i] = \begin{cases} \beta_v[i] \oplus parity, & \text{when } i = j \\ \beta_v[i], & \text{otherwise.} \end{cases} \tag{10}$$

Unlike traditional SC decoding, at the end of Fast-SSC decoding, the result of bit estimation needs to be multiplied by the corresponding size of the generator matrix G_{Nv}, and the final bit estimation u_1^{Nv} according to

$$u_1^{N_v} = \beta_1^{N_v} \mathrm{G}_{\mathrm{N}_v} \tag{11}$$

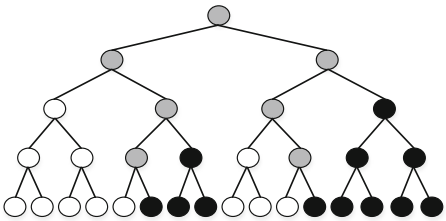

Fig. 3. SC algorithm decoding tree ($N = 16$, $R = 1/2$)

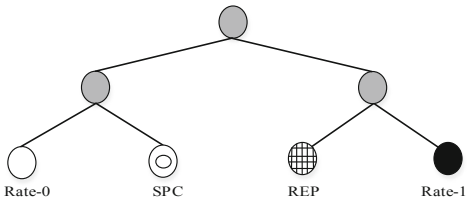

Fig. 4. Fast-SSC algorithm decoding tree ($N = 16$, $R = 1/2$)

Compared with the SC algorithm, the number of nodes in the Fast-SSC decoding tree is significantly reduced (see Fig. 4), which improves decoding the parallelism (Fig. 3).

4 Decoder Architecture

This paper designs a decoder based on the Fast-SSC algorithm (see Fig. 5), which is mainly composed of processing element (PE), controller, Kronecker product and memory module.

We transmit the channel-LLR to the PE, which is used to perform decoding of *f, g, c* and special nodes. The entire decoding process is manipulated by the controller, which decides whether the data transmitted to the PE is read from the register or the memory. If the results of current stage are not used immediately by the next stage, it will be stored into internal-LLR memory or internal-β memory. Due to the decoding result of every constituent node needs to multiply G_{Nv}, we use Kronecker product to implement it, where $Nv = 2, 4, 8, 16, 32$.

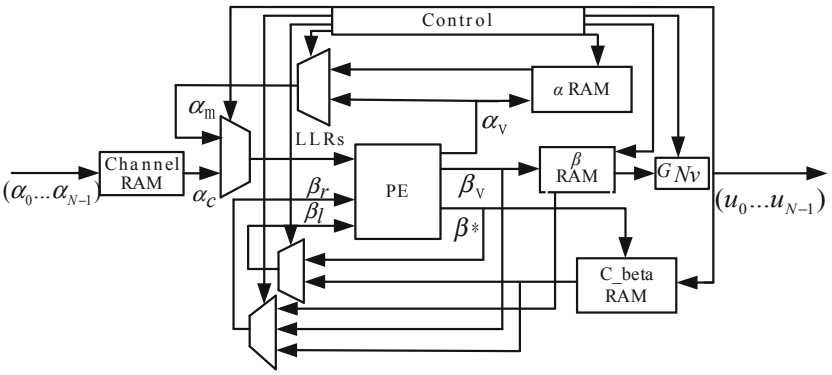

Fig. 5. The overall decoder architecture

LLRs and α_v are the input sequence and output LLR sequence of PE, α_c are LLRs from the channel memory, α_m are LLRs from the internal-LLR memory, respectively, β_v are bit estimations, β^* are the result of *c* operation, β_l and β_r represent bit estimates of the left and right children, respectively.

4.1 Processing Element

For REP nodes (see Fig. 6), they only use the sign of the sum to estimate the codewords. To ensure that the bit width does not affect the accuracy, and avoid the data saturation, they use zero fill method for REP with code length $N_v < 16$ to avoid performance degradation.

According to the character of SPC node (see Fig. 7), the absolute value of the LLRs need to be sorted. When the number of LLRs is large, the sorting complexity is high, which limits the operating frequency of this module, we constrain the length of the SPC to four. The 4MIN1 module selects the smallest LLRs by comparison operation, D0 ~ D3 are calculated to determine which bit to be flipped together with parity. For instance, when D0 = 1, it denotes that the estimate of bit 0 in SPC node is need to flipped.

The *g* operation has two inputs, α and β (see Fig. 8). When β is zero, $\alpha[2i]$ adds $\alpha[2i + 1]$, when β is one, $\alpha[2i + 1]$ subtracts $\alpha[2i]$, and we use two-sign complement operation to avoid overflow during addition.

We use one comparator, one XOR gate and a combination logic to implement *f* operation (see Fig. 9). Where the comparator is used to select the data with the smaller absolute value, and then combine with the sign bit to get the result.

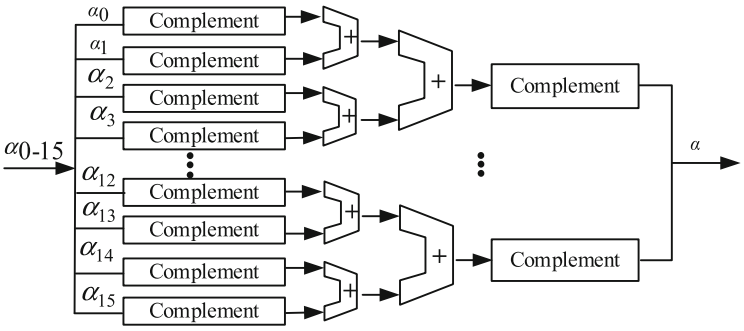

Fig. 6. REP module

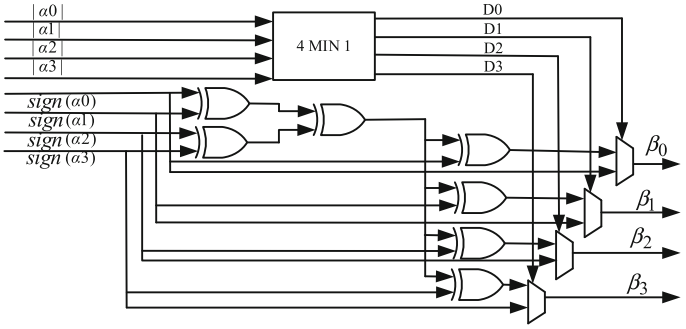

Fig. 7. SPC-4 decoding module

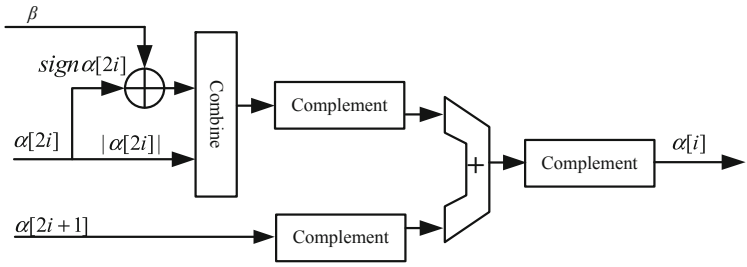

Fig. 8. *g* module

4.2 Memory

The memory mainly includes channel memory, internal-LLR memory, internal-β memory, codeword memory and register. Among them, channel memory stores the channel-LLRs. To reduce the resources of the decoder, the width of this memory is consistent with the data processed by the PE to minimize the memory read and write time. Internal-LLR memory stores the α generated by the f and g operations, therefore this memory is also the most frequent memory for data access, the width of the memory is 192 bits, which is

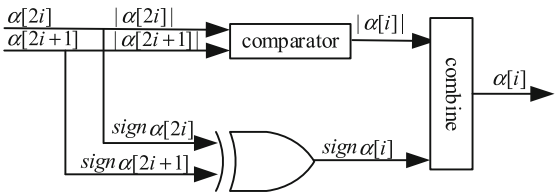

Fig. 9. f operation module

consistent with the output bit width of the. Internal-β memory stores the bit estimation β, we only store the β of the left child node to reduce the decoding latency. Codeword memory stores the bit estimates related to the operation of c, and its bit width is 64. In order to balance the hardware consumption, the right edge Rate-1 node is divided into two Rate-1 during decoding, which can reduce the memory resources of internal-LLR and internal-β by 50%.

4.3 Control Unit

The control unit mainly reads the data from the channel memory, selects the source of the LLRs of the PE from memory or registers, the memory address of the channel memory, the internal-LLR and the hard decisions of each constituent node. It controls the β for the g operation. In addition, the width of β for g operation and c operation is different, and the data between the register and the memory needs to be selected. On the other hand, it also selects PE to perform different decoding operations, controls whether to perform f, g, c operations and outputs the codeword estimates.

4.4 Architecture Optimization

According to the characteristics of polar code, this paper splits the data for nodes with a code length of $N_v > 64$, and which can reduce data waiting time and decoding latency. We design a 32 PE decoding $N = 256$ (see Fig. 10), F_{Nv_i} denotes the f operation on the $(64i \sim 64(i + 1)\text{-}1)$-th LLRs of the current code. The specific expression is as follows: perform F_{256_0} of the root node, and the result is temporarily stored in the register $\alpha*$, and then continue F_{256_1} of the root node, and the result is stored in the internal-LLR memory $\alpha\hat{}$. Then the root node is temporarily not used but activates the next node. After reading the combined data of $\alpha*$ and $\alpha\hat{}$, execute F_{128_0}, that is, obtain α_v of its left child node by the f operation, and store it in $\alpha\hat{}$. Return to the root node, execute F_{256_2} and F_{256_3} respectively, and store the result in $\alpha\hat{}$. Then execute F_{128_1} using the LLR of the left child V of the root node, the resulting LLR is combined with the F_{128_0}, and finally execute F_{64} to obtain α_{vl} with $N_v = 32$ node, and the subsequent lower layers still follow the Fast-SSC for sequential decoding.

In the case of the same hardware consumption, the decoder can reduce the entire decoding process by 10 clocks by splitting the node data. For nodes with $N_v > 64$, the c and g operations use the same rule to split the LLR sequence, and combine the registers to update the data in time to ensure the accuracy of the controller data selection.

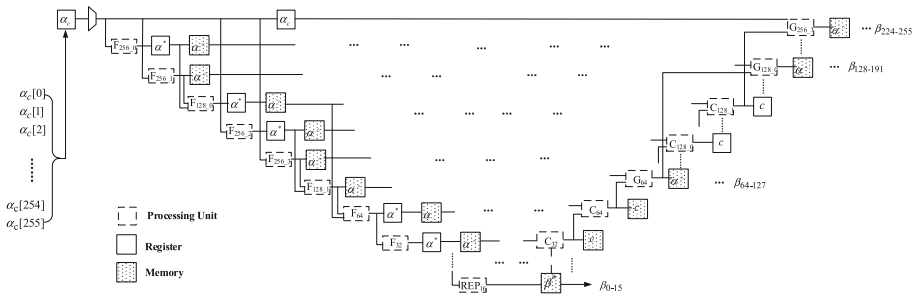

Fig. 10. Decoding flow

In order to read data quickly, it is convenient for the register and the memory to transmit data alternately, and reduce the memory bandwidth. We use three multiplexers to select the data required by each node (see Fig. 11), split the register data to have more flexible selects for the data required by the PE, effectively avoiding memory read-write conflicts, while reducing the number of accesses to the memory. Under the same circumstances, this method does not require additional registers with a bit width of 192.

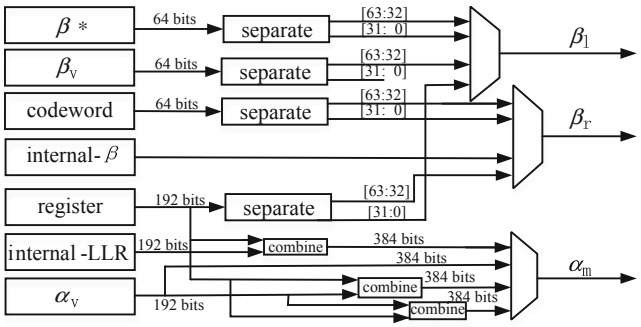

Fig. 11. Data access control

5 Performance Analysis

This paper synthesizes the Fast-SSC decoder on the Stratix V 5SGXEA7N2F45C2, and performs performance comparison between the two decoders. The resource consumption and decoding performance are shown in Table 1 and Table 2.

By comparison, it is found that the operating frequencies of the two decoders have reached 167 M, and the decoding latency of the 32-PE decoder is 0.68 μs, which is 16.18% more than the 128-PE decoder. But the hardware resource consumption is greatly reduced, among them, consumption of total register 782, total block storage bit 4608, logic utilization rate 3150 and total ram blocks 8, resource consumption was reduced by 34.94%, 45.45%, 28.73% and 66.67%, respectively (Fig. 12).

Table 1. Decoder resource consumption

Resources	32PE	128PE
Total registers	782	1202
Total virtual pins	1571	1571
Total block memory bits	4608	8448
Logic utilization	3150	4420
Total RAM blocks	8	24

Table 2. Comparison of decoding latency

	32PE	128PE
Frequency (MHz)	167	166.75
Decoding latency (μs)	0.68	0.57
Throughput (Mbps)	375	449

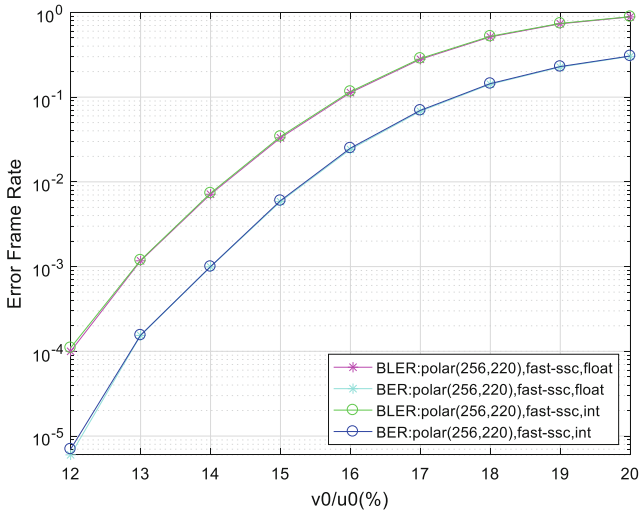

Fig. 12. Performance curve

In this paper, the channel-LLR is quantified with six bits including one fractional bit. When $v0/u0 = v1/u1$, $q0 = 0.999999$, $q1 = 0.999898$, $u0 = 1000$ and $u1 = 2000$, simulations result show that decoding performance with the quantified data is very close to that of the floating-point performance in term of BER and BLER.

6 Conclusion

This paper designs a high-throughput Fast-SSC decoder for STT-MRAM channel, synthesizes it on the FPGA. Simulation results show that the fixed-point decoding performance is very close to that of the floating point algorithm. By splitting the node data, adjusting the data processing order to optimize the architecture, the proposed decoder reduces the complexity of data selection and the decoding latency.

Acknowledgements. This work was supported in part by Joint Fund for Smart Computing of Natural Science Foundation of Shandong Province (ZR2019LZH001), Shandong University Youth Innovation Supporting Program (2019KJN020, 2019KJN024), Shandong Key Research and Development Project (2019GGX101066), the Natural Science Foundation of China (61701284), the Innovative Research Foundation of Qingdao (19-6-2-1-cg).

References

1. Van Beek, S., et al.: Impact of self-heating on reliability predictions in STT-MRAM. In: IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, pp. 25.2.1–25.2.4 (2018)
2. Song, Y.J., et al.: Demonstration of highly manufacturable STT-MRAM embedded in 28 nm logic. In: IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, pp. 18.2.1–18.2.4 (2018)
3. Zhang, M., Wu, F., Huang, H., Xia, Q., Zhou, J., Xie, C.: FPGA-based failure mode testing and analysis for MLC NAND flash memory. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, pp. 434–439 (2017)
4. Yang, K., Zhao, Y., Yang, J., Xue, X., Lin, Y., Bae, J.: Impacts of external magnetic field and high temperature disturbance on MRAM reliability based on FPGA test platform. In: 2015 IEEE 11th International Conference on ASIC (ASICON), Chengdu, pp. 1–4 (2015)
5. Tal, I., Vardy, A.: List decoding of polar codes. IEEE Trans. Inf. Theory **61**(5), 2213–2226 (2015)
6. Arikan, E.: Channel polarization: a method for constructing capacity-achieving codes for symmetric binary- input memoryless channels. IEEE Trans. Inf. Theory **55**(7), 3051–3073 (2009)
7. Hassan, H.G.H., Hussien, A.M.A., Fahmy, H.A.H.: A simplified radix-4 successive cancellation decoder with partial sum lookahead. AEUE – Int. J. Electron. Commun. **96**, 267–272 (2018)
8. Xu, Q., Pan, Z., Liu, N., You, X.: A low-latency list decoder for polar codes. Sci. China Inf. Sci. **61**(10), 1–10 (2018). https://doi.org/10.1007/s11432-017-9312-9
9. Wang, W., Li, L., Niu, K.: An efficient construction of polar codes based on the general partial order. EURASIP J. Wireless Commun. Netw. **2019**(1), 1–12 (2019). https://doi.org/10.1186/s13638-018-1327-7
10. Hashemi, S.A., Condo, C., Gross, W.J.: Fast simplified successive-cancellation list decoding of polar codes. In: 2017 IEEE Wireless Communications and Net-working Conference Workshops, pp. 1–6 (2017)
11. Sarkis, G., Giard, P., Vardy, A., et al.: Fast list decoders for polar codes. IEEE J. Sel. Areas Commun. **34**(2), 318–328 (2016)
12. Zhang, X., et al.: High-throughput fast-SSC polar decoder for wireless communications. Wireless Commun. Mob. Comput. **2018**, 1–10 (2019)