Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 7428039, 10 pages
https://doi.org/10.1155/2018/7428039

Research Article

High-Throughput Fast-SSC Polar Decoder for Wireless Communications

Xiaojun Zhang,[1] **Xiaofeng Yan**,[1] **Qingtian Zeng**,[1] **Jianming Cui**,[1] **Ning Cao**,[2] **and Russell Higgs**[2]

[1]*Shandong University of Science and Technology, Qingdao 266590, China*
[2]*University College Dublin, Belfield, Dublin 4, Ireland*

Correspondence should be addressed to Xiaojun Zhang; buttern@163.com

Received 4 May 2018; Accepted 4 July 2018; Published 29 July 2018

Academic Editor: Naixue Xiong

Polar code has been proven to achieve the symmetric capacity of memoryless channels. However, the successive cancellation decoding algorithm is inherent serial in nature, which will lead to high latency and low throughput. In order to obtain high throughput, we design a deeply pipelined polar decoder and optimize the processing elements and storage structure. We also propose an improved fixed-point nonuniform quantization scheme, and it is close to the floating-point performance. Two-level control strategy is presented to simplify the controller. In addition, we adopt FIFO structure to implement the α_memory and β_memory and propose the 348-stage pipeline decoder.

1. Introduction

Wireless communication is changing our life and has been applied to many scenarios [1–5], and error-correcting codes are utilized to improve its transmission efficiency and reliability. Polar code is a class of error-correction codes proposed by Arikan [6]. Within the ongoing 5th-generation wireless systems (5G) standardization, polar codes have been adopted as channel coding for the enhanced mobile broadband (eMBB) communication service for its excellent error-correction performance. Especially, for the ultra-reliable low-latency communications (URLLC), it should satisfy the high throughput of several tens Gbps [7], which bring in a challenge for polar decoder. In the past, great research efforts have been made on polar codes in decoding algorithm and hardware architecture since Arikan presented the successive cancellation (SC) decoding algorithm. SC has the advantages of low complexity and simple decoding structure. Although polar codes can theoretically achieve channel capacity when code length is infinite, the performance of SC is mediocre for codes of short and moderate lengths. To address this issue, successive cancellation list (SCL) decoding algorithm is proposed in [8]. Different from the SC algorithm, SCL does not focus on a single candidate codeword; it saves L most reliable candidate codewords at every step. The decoding performance of SCL has been significantly improved. K. Chen and K. Niu proposed CRC-aided SCL (CA-SCL) algorithm [9] based on that the correct codewords can pass the CRC check. And they proposed the successive cancellation stack (SCS) decoding algorithm in [10] and successive cancellation hybrid (SCH) decoding algorithm in [11]. Unlike the SCL decoding which preserves the L most reliable paths in each layer, SCS always extends the most reliable path. Compared with SCL decoding, the performance of SCS is the same as SCL, but the time complexity is lower and the space complexity is higher. The actual time complexity of SCS decoding is far below than that of SCL in the high-SNR regime and is close to SC decoding. SCH algorithm combines the advantages of SCL and SCS, and the performance of SCH is close to that of maximum likelihood (ML) [12]. The researchers of Huawei proposed the adaptive CA-SCL (aCA-SCL) [13] decoding algorithm based on CA-SCL algorithm. aCA-SCL improves the decoding performance by gradually expanding the search width L. aCA-SCL can reduce the software complexity significantly. The above decoding algorithm is proposed for improving the performance, but their throughput is not

ideal. Thus A. Alamdar-Yazdi and F. R. Kschischang propose simplified successive cancellation (SSC) decoding algorithm in [14] based on the location of frozen and information bits. SSC decoding reduces the computational complexity and improves the decoding parallelism by combining some leaf nodes, such as Rate-1 node whose leaf nodes are all unfrozen bits. Simulation results illustrate that the performance is similar to that of SC. G. Sarkis and W. J. Gross divide the leaf nodes into Rate-0, Rate-1, and Rate-R nodes and propose the maximum likelihood SSC (ML-SSC) decoding algorithm in [15] which is mainly to improve the performance of Rate-R nodes decoder in SSC decoding. Compared with the semiparallel SC decoding in [16], ML-SSC decoding improves the decoding throughput by 25 times. In order to further reduce the decoding complexity and improve the throughput, G. Sarkis proposes the fast simplified successive cancellation (Fast-SSC) decoding algorithm, which mainly improves the decoding rules of Rate-R nodes and gives the specific operation for each constituent node [17]. Fast-SSC decoding divides the Rate-R nodes into repetition (REP), single-parity-check (SPC), and REP-SPC nodes and improves the throughput.

In addition, for the decoding hardware architecture of polar codes, a semi-parallel architecture is proposed in [16]. In order to improve resource utilization, this method reuses the processing elements (PE) which effectively reduce the hardware complexity. The overlapped architectures proposed in [18] have advantages in both latency and throughput, which uses precalculation function calculate the possible results firstly and according to the decoded results to choose the corresponding results. It is proved that the decoding latency is reduced by 50% when the code length is larger than 2^7. Then B. Yuan proposed the SCL decoder with multi-bit decision which effectively reduces the decoding latency [19]. And an unrolled hardware polar decoder is proposed in [20] on the basis of Fast-SSC. This decoder loads one frame channel decoding data and outputs a frame of codeword each clock. The PEs are no longer reused and dedicated PEs are assigned to each stage. Graphics processor unit (GPU) provides the flexibility and massive parallel units; the GPU-based polar decoders obtain high throughput [21–23].

In this paper, we investigate the characters of LLRs for different stages of Fast-SSC polar decoder and propose an improved nonuniform fixed-point quantization method. It adopts (6,5,1) quantization scheme; the decoding performance is close to the floating-point decoding performance. The proposed decoder employs deeply pipelined architecture and optimizes the REP decoder and G operation. To simplify the deeply pipelined control, the controller is divided into global controller and local controller. *α_memory* and *β_memory* use FIFO architecture to reduce the control logic. Finally, a 348-stage pipeline architecture is devised, which is implemented on Altera Stratix V 5SGXEA7N2F45C2. To test the decoding performance, we design a platform based on FPGA.

The remainder of this paper is organized as follows. A brief review of Fast-SSC decoding algorithm and analysis of the quantization schemes are shown in Section 2. Section 3 depicts the deeply pipelined architecture and the PEs. Performance is evaluated in Section 4 and conclusions are drawn in Section 5.

2. Review of Fast-SSC

2.1. Polar Codes. A polar code can be represented by $P(N, K)$, where N denotes the code length and K/N is the code rate. Polar code of length N can be constructed by concatenating two polar codes of length $N/2$. The construction method can be denoted by $x = uG^{\otimes n}$, where $u = \{u_0, u_1, \ldots, u_{N-1}\}$ is the input sequence that to be encoded, and $x = \{x_0, x_1, \ldots, x_{N-1}\}$ denote the codewords. $G^{\otimes n}$ is the n-th Kronecker power of the generator matrix $F = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. Polar codes select K most reliable channels to transmit information bits, and the other N-K channels transmit frozen bits.

2.2. Fast-SSC Decoding Algorithm. The binary decoding tree of Fast-SSC nodes is divided into four types: Rate-0, Rate-1, REP, and SPC. Compared with SC decoding tree, Fast-SSC has less leaf nodes. Since the polar decoder traverses the entire binary tree during iterations, Fast-SSC decoding algorithm has low latency. Figure 1 shows SC decoding tree and corresponding Fast-SSC decoding tree for a (16, 8) polar code. For instance, the REP node consists of leaf node $\{4, 5, 6, 7\}$ and SPC node includes leaf node $\{8, 9, 10, 11\}$. The leaf nodes of Rate-0 node are all frozen bits. Therefore, its output will be the zero vector. The leaf nodes of Rate-1 node are all information bits. The decoding result of such nodes is obtained by

$$\beta_v[i] = \begin{cases} 0, & \alpha_v[i] \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{1}$$

For the REP node, only the last bit of it is information bit, and others are frozen bits. The REP node adds all the input α first and then makes a hard decision as

$$\beta_v = \begin{cases} 0, & \text{when } \left(\sum_{i=0}^{N_v-1} \alpha_v[i]\right) \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

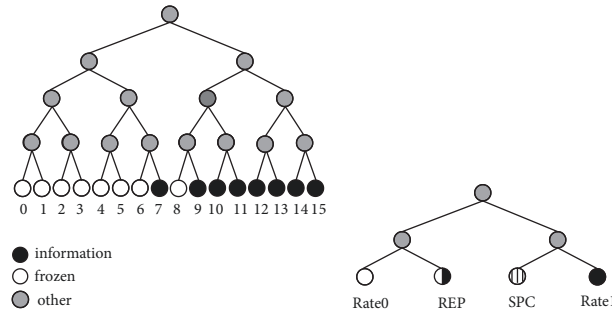
where N_v denotes the code length of the node.

The SPC node, of which only the first leaf node is frozen bit, performs threshold detection by (3) on the input LLRs firstly. The parity of all the inputs is calculated by (4). Then the least reliable bit is founded and flipped if the parity constraint is not satisfied. The threshold detection can be written via

$$\beta_v[i] = \begin{cases} 0, & \alpha_v[i] \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

The parity of the input is calculated as

$$parity = \bigoplus_{i=0}^{N_v-1} \beta_v[i] \tag{4}$$



(a) SC decoding tree (b) Fast-SSC decoding tree

Figure 1: Decoding tree for $P(16, 8)$.

Finally, the output of the SPC node is

$$\beta_v[i] = \begin{cases} \beta_v[i] \oplus parity, & \text{when } i = j \\ \beta_v[i], & \text{otherwise} \end{cases} \quad (5)$$
$$i = \underset{j}{\operatorname{argmin}} \left(\left|\alpha_v[i]\right|\right)$$

In addition to the above four type nodes, the rest colored in grey is referred to as other node, as shown in Figure 1. The decoding method of other nodes uses standard SC algorithm as in Figure 2. When node v is activated, it will receive α_v from its parent node p_v and then calculate the soft-valued input to its left child, α_l, which is calculated using the F operation.

$$\begin{aligned} \alpha_l[i] &= F\left(\alpha_v[i], \alpha_v\left[i + \frac{N_v}{2}\right]\right) \\ &= \operatorname{sign}(\alpha_v[i]) \operatorname{sign}\left(\alpha_v\left[i + \frac{N_v}{2}\right]\right) \\ &\quad * \min\left(\left|\alpha_v[i]\right|, \left|\alpha_v\left[i + \frac{N_v}{2}\right]\right|\right) \end{aligned} \quad (6)$$

Once β_l of the left child node is estimated, it is used to calculate the input to the right child node α_r with G operation.

$$\begin{aligned} \alpha_r[i] &= G\left(\alpha_v[i], \alpha_v\left[i + \frac{N_v}{2}\right], \beta_l[i]\right) \\ &= \begin{cases} \alpha_v\left[i + \frac{N_v}{2}\right] + \alpha_v[i], & \text{when } \beta_l[i] = 0 \\ \alpha_v\left[i + \frac{N_v}{2}\right] - \alpha_v[i], & \text{otherwise} \end{cases} \end{aligned} \quad (7)$$

Finally, β_l and β_r are combined to calculate β_v as

$$\beta_v[i] = \begin{cases} \beta_l[i] \oplus \beta_r[i], & \text{when } i < \frac{N_v}{2} \\ \beta_r\left[i - \frac{N_v}{2}\right], & \text{otherwise} \end{cases} \quad (8)$$

Table 1 lists the number of constituent nodes of the decoding tree for a (1024, 512) polar code. It can be seen that the total number of constituent nodes is 104 of the Fast-SSC decoding, which decreases from 1024 of the SC decoding tree. The decoder does not need to traverse the entire decoding tree, it just traverses the pruned tree. Thus Fast-SSC algorithm improves the decoding efficiency and throughput and decreases the latency.

Table 1: The number of constitute nodes.

Node Types	*Rate-0*	*Rate-1*	*REP*	*SPC*	total
Numbers	14	40	24	26	104

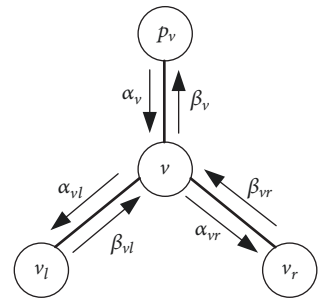


Figure 2: Local decoder of node v.

2.3. Quantization Scheme. The quantization scheme is divided into uniform and nonuniform quantization. The uniform quantization is simple, but the consumption of resources is more than that of nonuniform scheme. The nonuniform quantization employs different quantization bits in different decoding stages and uses less storage resources, but the memory structure is not regular [27]. Unlike the conventional SC decoder which memory is shared for the nodes of different stages, the PEs of deeply pipelined decoder in each stage are equipped with a separate memory. In order to reduce the memory consumption, the nonuniform scheme is adopted to quantitate channel LLRs and internal LLRs. In [20], it adopts the all-integer quantization method, where channel LLR is 4 bits and internal LLR is 5 bits. In this paper, an improved quantization scheme is proposed based on LLR distribution of different stages. At the beginning, the internal LLRs is small, and it is quantitated with the same bits as channel LLRs. To avoid catastrophic overflow, the internal LLRs of latter stages are quantitated with larger bits. Let (Q_i, Q_{cf}, Q_f) denote the quantization scheme, where Q_{cf} presents the

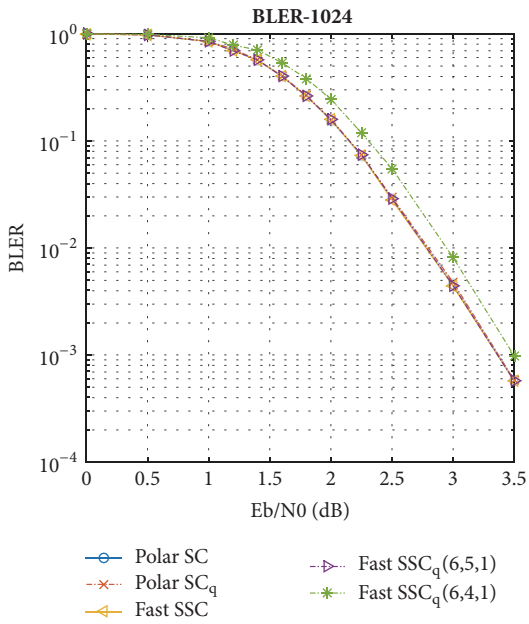


Figure 3: The BLER performance of different decoding schemes.

quantization bits of channel LLRs and that of LLRs for the former stages of the decoder, Q_i denotes the internal LLRs of other stages, and Q_f is the fractional bits. Figure 3 shows the block error rate (BLER) performance of SC, Fast-SSC algorithm, and different quantization schemes. The floating decoding performance of Fast-SSC is close to that of SC. In the quantization schemes of Fast-SSC, it can be seen that the performance of (6,5,1) quantization is close to the floating-point performance, but (6,4,1) quantization results in less than 0.2dB performance loss in high E_b/N_0. Therefore, this paper adopts (6,5,1) quantization scheme.

3. Architecture of Fast-SSC Decoder

The decoder is implemented in deeply pipelined architecture to improve the decoding throughput. This paper optimizes the PEs, storage, and control modules to lower the latency.

3.1. Architecture. The structure of Fast-SSC decoder is depicted in Figure 4, which consists of PE, memory, and controller. The PE is composed of various functions, such as *F*, *G* function, and Kronecker power module. Memory is divided into *α_memory* and *β_memory*, which are utilized to store the channel and internal LLRs and the hard decision of each constituent node, respectively. Because the decoding result of every constituent node needs to multiply G_N, the Kronecker power module contains G_N (N = 4, 8, 16, 32, 64) matrix for the leaf nodes of different length. The entire decoding process is manipulated by the controller module. When the channel LLRs signal (*en_cha_alpha*) is valid, the decoder starts to load one new frame into the decoder, and it outputs the codeword estimates. If the results of current stage are not used immediately by the next stage, it will be stored into *α_memory* or *β_memory*.

3.2. Deeply Pipelined. The deeply pipelined architecture for a (1024, 512) polar code is illustrated in Figure 4. The dotted lined rectangles represent the PEs such as REP128, where REP denotes the operation type; the subscript 128 represents the input length of the node. The spotted rectangles represent the RAMs, which are used to store the internal results to give the latter pipelined stages when the current results are not used immediately and the data is larger than 16. The solid lined rectangles represent registers. When the two-stage operation using a certain data is closer and the amount of data is small, the registers are used to store the data temporarily to reduce the memory control signal. The deeply pipelined architecture is designed according to the node activation order of the decoding tree and the operation order of the local decoding of each node. The software simulation which consists of 368 operations and the hardware implementation is a total of 330 operations. In order to achieve high throughput, this paper split the stages with larger latency. For example, REP128 can be split into four stages. The final architecture has 348 stages; thus the decoding delay is 348 clock cycles. Each stage in the pipeline contains one PE.

3.3. Memory. For node v in Figure 2, the inputs data α_v need to be used twice during the decoding process. Firstly, it is used to calculate α_{vl} of left child node; then it is utilized to calculate α_{vr} of right child node. Similarly, the local decoding result β_{vl} of node v_l needs to be input into Kronecker product module to obtain the final decoded words, and it also needs to calculate the local decoding results β_{vr} of its brother node v_r to obtain the result β_v of node v by C operation. The internal α and β need to be used twice in different stages, so they need to be stored. Since the bit widths of α and β are different, they are stored separately. The memory is divided into *α_memory* and *β_memory* as shown in Figure 4. If a node produces m internal data at t_1 clock and uses them at t_2, assuming $d = t_2 - t_1$, it will require $m \cdot (d + 2)$ memory unit. When the memory unit is less than 16, it can be stored with registers. Otherwise, we will use RAMs to store the internal results. The access timing of RAM is shown in Figure 5, where *Adi* denotes the i-th address of RAM. It is clear that the read sequence is the same as write sequence and they only differ d in clock cycles. Therefore, we can use FIFO to replace RAM.

3.4. Processing Elements. The main PEs of Fast-SSC decoder are listed in Table 2. *F* and *F_with_front_complement* are used to calculate the left child inputs of the activated node. *G*, *G-0R*, *G_without_complement*, *G_without_front_complement*, and *G_without_latter_complement* can calculate the inputs of right child nodes. The *C* operation is used to combine the local decoding results of the left and right children nodes into the local decoding of the active node *v*. *R0-R1*, *REP*, *SPC*, and *R0_SPC* are the decoding operation of the corresponding type of leaf nodes, respectively. In addition to the 330 operations, there are two else stages operations. One exists in the first stage and the other one in the last stage. The first stage is used to cache the channel LLRs and occupies one clock. The results of the local nodes need to multiply G_N matrix to recover the local codeword. After the last constituent node

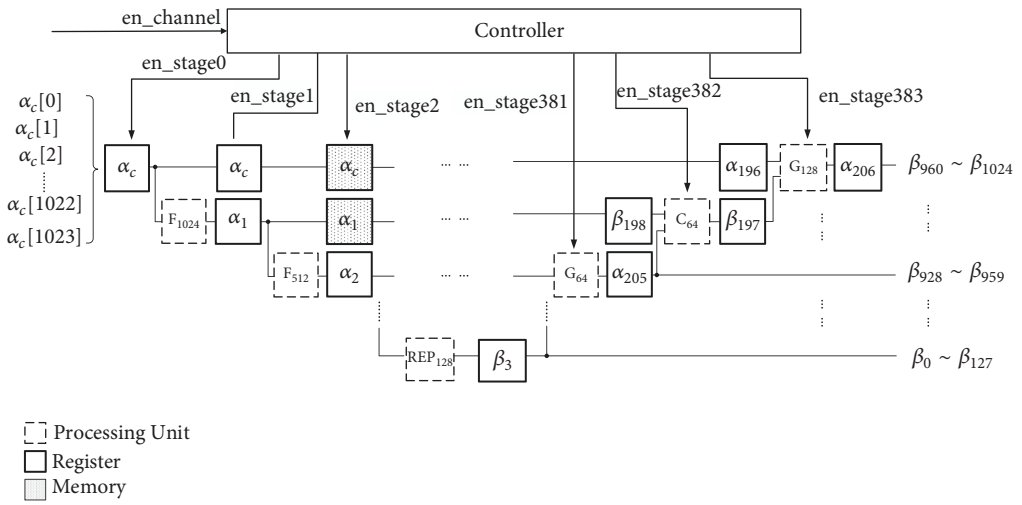


Figure 4: The architecture of Fast-SSC decoder.

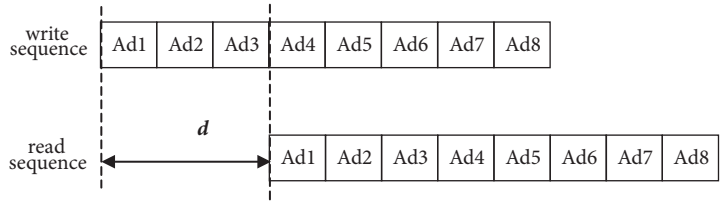


Figure 5: Access timing of RAM.

is computed, the converting operation by G_N for it will take one separate clock cycle. According to the above analysis, the deeply pipelined architecture has a total of 332 stages.

To implement the deeply pipelined architecture, we unfold the overall decoder. The *G-0R*, *R0-R1*, and *R0-SPC* operations are introduced to reduce the number of stages. The decoder can directly active the right child when the left child is Rate-0 node; thus it can reduce the decoding latency and the storage capacity. Moreover, in order to balance the pipeline at all stages and lower wire routing congestion, we refine the *F* and *G* operations into *F_with_front_complement*, *G_without_complement*, *G_without_front_complement*, and *G_without_latter_complement* operations when the inputs are large.

3.5. F Module. *F* operation is used to calculate the α_l of left child nods. According to (4), the sign bit of $\alpha_l[i]$ is obtained by XOR operation, and the numerical bits are the minimum of $\alpha_l[i]$ and $\alpha_l[i+1]$ by compare operation. The structure is shown in Figure 6.

3.6. G Module. The *G* operation calculates α_r of the right child based on β_l of the left child and α_v of the parent node. According to (5), $\alpha[i]$ adds $\alpha[N/2+i]$ when $\beta = 0$, and $\alpha[i]$ subtracts $\alpha[N/2+i]$ when $\beta = 1$. The structure is shown in Figure 7.

When the input size of *G* is larger than 256, high decoding latency will be brought. It is clear that when the input length is long, the next stage after *G* is usually *F* operation. Since the complexity of *F* operation is less than that of *G* operation, to balance the frequency of the two operations, the complement operation of *G* is moved into the *F* operation. The optimized architecture is depicted as Figure 8. The left and right sides of the dotted line are the PEs of two stages, respectively. The complement operation of *G* colored in grey is performed in the next stage.

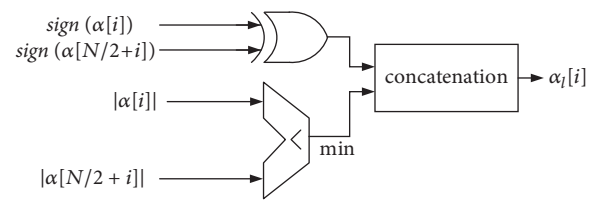


Figure 6: Structure of *F* module.

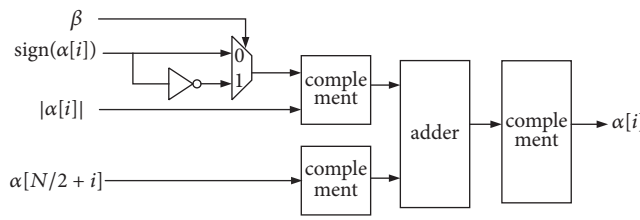


Figure 7: Structure of *G* module.

Table 2: The number of each operation for one frame.

Types	Description	numbers
F	Calculating α_l. (4)	82
F_with_front_complement	Calculating α_l, input complement.	7
G	Calculating α_r. (5)	78
G-0R	Calculating α_r. where $\beta_l = 0$	14
G_without_complement	Calculating α_r, input and output complement	3
G_without_front_complement	Calculating α_r, input complement.	4
G_without_latter_complement	Calculating α_r, output complement.	4
C	Calculating β_v. (6)	85
R0_R1	Decoding Rate-1 nodes, where $\beta_l = 0$.	3
REP	Decoding REP nodes	24
SPC	Decoding SPC nodes	23
R0_SPC	Decoding SPC nodes, where $\beta_l = 0$	3
Total		330

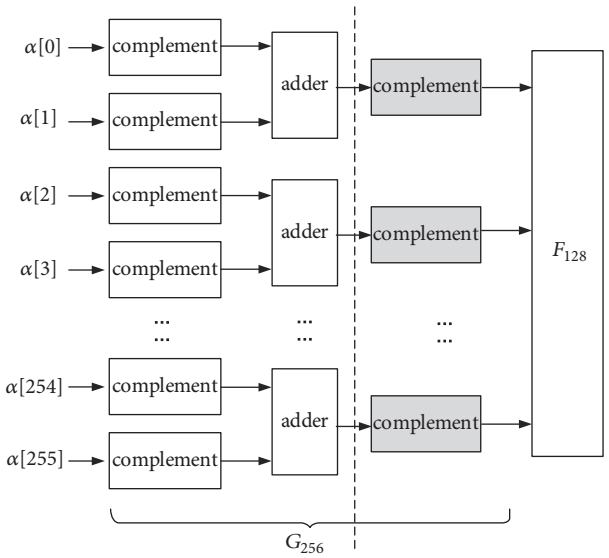


Figure 8: Optimized structure of G_{256}.

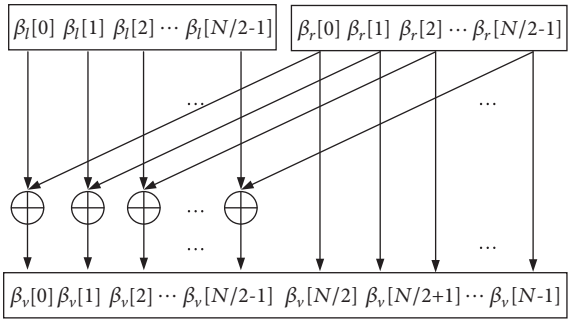


Figure 9: Structure of *C* module.

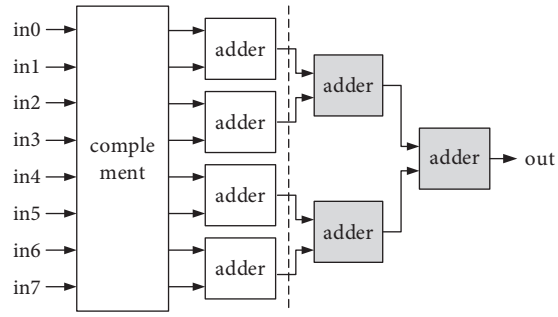


Figure 10: Optimized 8-input architecture of REP module.

3.7. C Module. The *C* module combines β_l of left child and β_r of right child to calculate β_v of parent node. According to (6), the first half of β_r is obtained by β_l XOR β_r, and the latter half is equal to β_r directly. The structure of *C* module is shown in Figure 9.

3.8. REP Module. The number of input ports in REP module is 4, 8, 16, 32, and 64. The 4-input module is the basic REP; other types can be decomposed into 4-input type. The hardware architecture for 8-input REP is presented in Figure 10. When the input length of REP nodes increases, the decoding latency also increases. To improve the working frequency, the 8-input REP is divided into two stages; thus it will use two clock cycles. The first stage translates the input data to complement and generates four internal results. The second stage adds them, and the REP module outputs the result of all the 8-inputs data. The hardware architecture is shown in Figure 10. The dotted line indicates that the original 8-input REP module is divided into two stages.

Similarly, REP module of other lengths is divided into different stages. For instance, the 16-, 64-, and 128-input REP modules are divided into 2, 3, and 4 stages, respectively.

3.9. SPC Module. In order to improve the frequency, the length of SPC node is constrained to 4 as shown in Figure 11. $|\alpha[i]|$ denotes the absolute value of $\alpha[i]$, and sign($\alpha[i]$) is the sign bit of $\alpha[i]$. The 4MIN1 module selects the smallest LLRs through compare operation. *min01_flag* denotes the index of the minimum of $\alpha[0]$ and $\alpha[1]$, and *min23_flag* denotes the

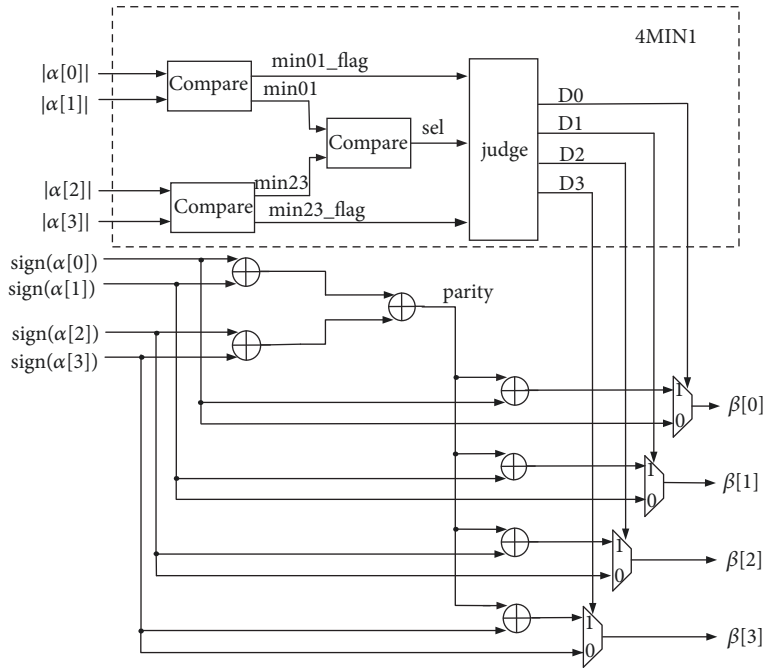


Figure 11: Structure of SPC module.

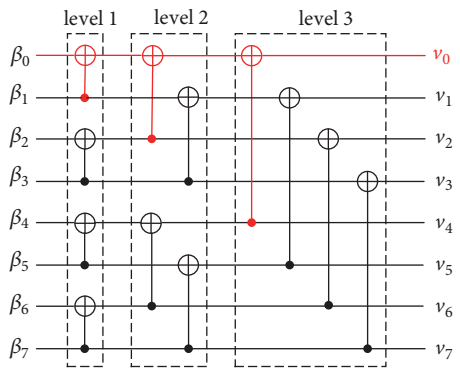


Figure 12: Architecture of Kronecker power module.

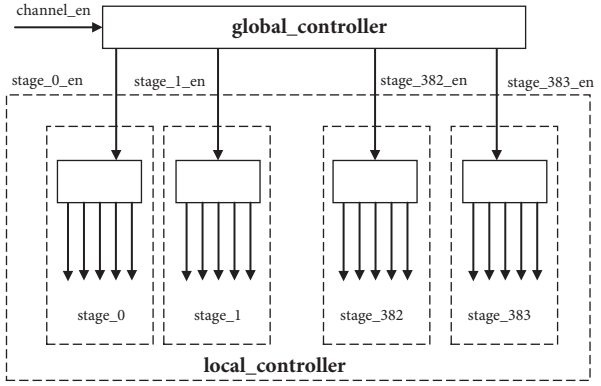


Figure 13: Two-level controller.

index of the minimum of $\alpha[2]$ and $\alpha[3]$. If $\alpha[0]$ is less than $\alpha[1]$ *min01_flag* is set to zero; otherwise *min01_flag* is one. If *min01* is less than *min23*, *sel* is set to zero; otherwise *sel* is one. *D1~D3* is determined by judge module. For instance, if *sel* and *min01_flag* are both zeroes, then *D0* is set to one and others are set to zero. If *sel* is zero and *min01_flag* is one, then *D1* is set to one and others are set to zero.

3.10. Kronecker Power Module. The decoding result of the constitute nodes requires the conversion of *n*th-Kronecker power to get the final result by (9). The code length of the architecture of the Kronecker power module is 8 as shown in Figure 12, where $\oplus$ denotes XOR operation and $\bullet$ denotes that the data is connected directly.

It can be found that if v_i is equal to zero, then it can be obtained as zero directly. As shown in Figure 12, the three XOR operations colored in red can be removed for v_0 is zero.

$$v_1^N = u_1^N F^{\otimes n}, \quad where\ N = 2^n,\ n \geq 0 \tag{9}$$

3.11. Controller. The controller uses a two-level mode to generate control signals. As shown in Figure 13, the first level generates the global control signals, which assigns an enable signal to each stage to determine whether the corresponding stage works or not. For instance, if *stage1_en* is asserted, then stage1 is working; otherwise, stage1 is idle. The second level only generates the local control signals for each stage, such as

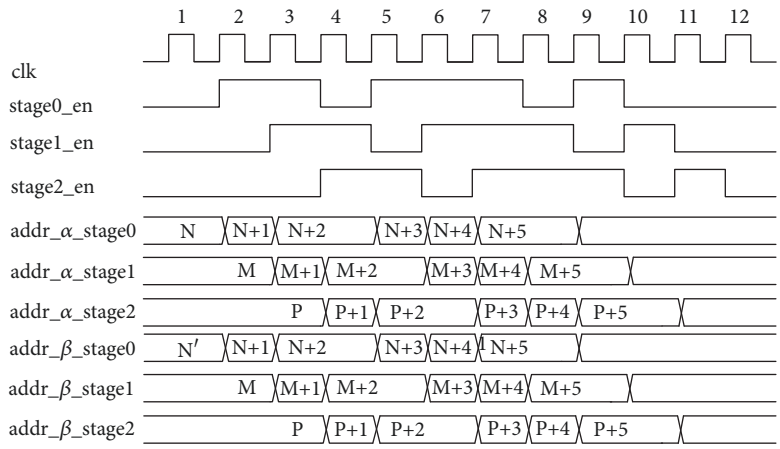


Figure 14: Timing of the controller.

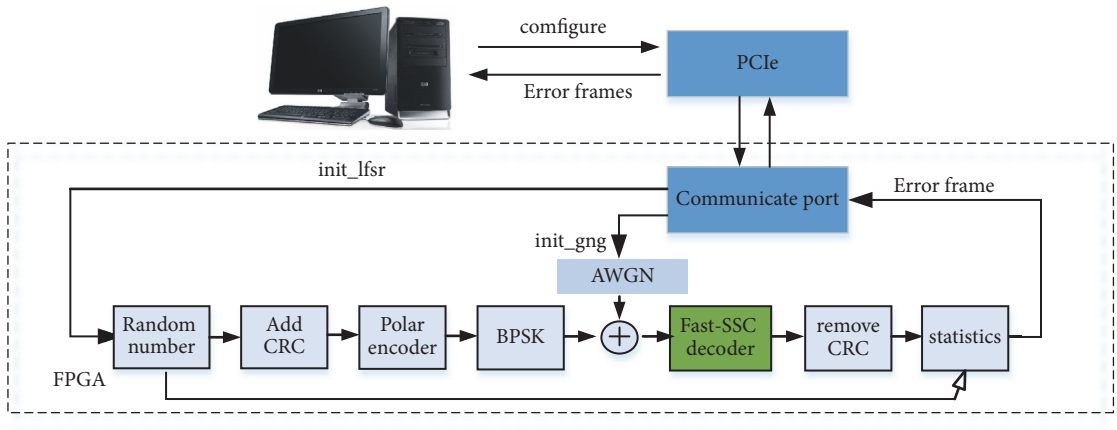


Figure 15: Test platform for polar decoder based on FPGA.

the address bus, data, and control signals of the memory, and the enable signals of PEs.

As depicted in Figure 14, *addr_α_stgae2* denotes the address bus of the *α_memory* in stage2. When *satge2_en* is asserted, the corresponding address adds one. Namely, an internal LLR is stored into α_memory_stage2. Compared with the one-level controller to generate all the control signals, it can reduce the implementation complexity of the controller.

4. Performance Analysis

4.1. Test Platform. To test the high throughput of Fast-SSC decoder, we implement a test platform based on FPGA. The overall platform including generating test data is completed on the FPGA to reduce the communication cost with the host computer. As depicted in Figure 15, the platform consists of random number generator, CRC check, polar encoder, BPSK AWGN, Fast-SSC decoder, and statistics module. PCIe is responsible for the communication between the host computer and FPGA platform. Meanwhile, this paper designs a software platform based on C++ that compares the results of hardware test and software simulation. At the beginning, the host computer generates random number seeds, Gaussian noise seeds, the number of test frames, and start signal and transmits them to FPGA. When the decoding is completed, the statistics module uploads the number of error frames; then the host computer calculates the BLER and displays the test parameters. For the (1024, 512) polar code, simulations show that the test platform takes 19.18s at 300MHz to test data with $1.4*10^{10}$ bits.

4.2. Resource Consumption. The (1024, 512) polar decoder is implemented on Altera Stratix V 5SGXEA7N2F45C2 in Quartus II 15.0. The resources used by the decoder based on FPGA are shown in Table 3. It can be observed that the proposed decoder costs more memory compared with other decoder based on FPGA for that 6 bits is used to quantize the LLRs partly. However, it costs less registers compared with [20]. For the decoder in [24], it costs less resource because it does not adopt deeply pipelined architecture.

4.3. Performance. The latency and throughput are the main performance parameters of the polar decoder. Let *freq_decode* be the frequency of the decoder, and *frame_decoder_clocks*

Table 3: Statistics of resources.

	ALMs/LUTs	Register	Memory (bits)
Proposed	81498	96762	2367488
[20]	156450	152124	285120
[24]	29828	2332	18356

Table 4: Comparison with other polar decoders.

	This work	P. Giard [20]	Park [25]	Dizdar [26]	P. Giard [24]
Decoding. Algo.	Fast-SSC	Fast-SSC	BP	SC	Fast-SSC
IC type	FPGA	FPGA	ASIC	ASIC	FPGA
Tech(nm)	28	40	65	90	40
f(MHz)	300	231	300	2.5	80.6
Latency(us)	1.16	2.4	50	0.4	2.1
T/P(Gbps)	307.2	237	4.68	2.56	0.48

denote the number of clocks to decode one frame. The latency and throughput are calculated by

$$latency = \frac{1}{freq\_decoder} * frame\_decoder\_clocks \tag{10}$$

$$Throughput = N * freq\_decoder \tag{11}$$

In this paper, for the (1024, 512) polar decoder, its working frequency can achieve 300MHz. The decoder requires 348 clocks to decode one frame. By (10) and (11), the latency is 1.16us, and the throughput is 307.2Gbps. Table 4 compares the proposed decoder with other polar decoders. In [20], a deeply pipelined decoder based on FPGA is capable of achieving the throughput over 237 Gbps for a (1024, 512) polar code. The latency of the decoder is twice more than this work. And the throughput of the proposed decoder is 1.3 times greater than that. It shows that either the latency or the throughput of this work is better than that in [20, 24, 25]. O. Dizdar and E. Arikan proposed a deeply pipelined polar decoder based on SC decoding algorithm. That decoder operates at lower clock frequency and costs less dynamic power. The proposed decoder has three times higher latency but is over 119 times faster than that in [26].

5. Conclusions

In this paper, a decoder in deeply pipelined architecture has been presented based on Fast-SSC decoding algorithm. The proposed decoder can output 1024 bits at each clock. To optimize the critical path, the PEs are decomposed and recombined to balance the latency of two adjacent stages. The fixed-point nonuniform quantization scheme lowers storage capacity and obtains a good decoding performance. The two-level mode is proposed to reduce the complexity of the controller. Moreover, we build a platform based on FPGA to test its performance. Numerical results show that the decoder can achieve high throughput.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the Natural Science Foundation of China (61701284, 61472229, and 61471224), Sci. & Tech. Development Fund of Shandong Province of China (2016ZDJS02A11), project funded by China Postdoctoral Science Foundation (2016M592216), Qingdao Postdoctoral Research Project (2016125), and SDUST Research Fund (2015TDJH102).

References

[1] W. Liu and X. Luo, "Localization Algorithm of Indoor Wi-Fi Access Points Based on Signal Strength Relative Relationship and Region Division , Computers," *Materials Continua*, vol. 55, no. 1, pp. 71–93, January 2018.

[2] Z. Xia, N. N. Xiong, A. V. Vasilakos, and X. Sun, "EPCBIR: An efficient and privacy-preserving content-based image retrieval scheme in cloud computing," *Information Sciences*, vol. 387, pp. 195–204, 2017.

[3] H. Cheng, Z. Su, N. Xiong, and Y. Xiao, "Energy-efficient node scheduling algorithms for wireless sensor networks using Markov Random Field model," *Information Sciences*, vol. 329, pp. 461–477, 2016.

[4] R. Meng, S. G. Rice, J. Wang et al., "A fusion steganographic algorithm based on faster R-CNN," *Computers, Materials & Continua*, vol. 55, no. 1, pp. 1–16, January 2018.

[5] H. Cheng, N. Xiong, A. V. Vasilakos, L. Tianruo Yang, G. Chen, and X. Zhuang, "Nodes organization for channel assignment with topology preservation in multi-radio wireless mesh networks," *Ad Hoc Networks*, vol. 10, no. 5, pp. 760–773, 2012.

[6] 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.

[7] "IMT-2020(5G) PG White Paper on 5G Concept," http://www.imt-2020.org.cn/zh/documents/1?currentPage=2&content=.

[8] I. Tal and A. Vardy, "List decoding of polar codes," in *Proceedings of the 2011 Information Theory*, pp. 1–5, St. Petersburg, Russia, August 2011.

[9] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, 2012.

[10] K. Niu and K. Chen, "Stack decoding of polar codes," *IEEE Electronics Letters*, vol. 48, no. 12, pp. 695–697, 2012.

[11] K. Chen, K. Niu, and J.-R. Lin, "Improved successive cancellation decoding of polar codes," *IEEE Transactions on Communications*, vol. 61, no. 8, pp. 3100–3107, 2013.

[12] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Communications Letters*, vol. 16, no. 12, pp. 2044–2047, 2012.

[13] J. Snyders and Y. Be'ery, "Maximum likelihood soft decoding of binary block codes and decoders for the Golay codes," *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, vol. 35, no. 5, pp. 963–975, 1989.

[14] 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.

[15] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Communications Letters*, vol. 17, no. 4, pp. 725–728, 2013.

[16] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289–299, 2013.

[17] 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, 2014.

[18] C. Zhang and K. Parhi, "Low-latency sequential and overlapped architectures for successive cancellation polar decoder," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2429–2441, 2013.

[19] B. Yuan and K. K. Parhi, "Low-Latency Successive-Cancellation List Decoders for Polar Codes with Multibit Decision," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2268–2280, 2015.

[20] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "237 Gbit/s unrolled hardware polar decoder," *IEEE Electronics Letters*, vol. 51, no. 10, pp. 762-763, 2015.

[21] Y. Li and R. Liu, "High throughput GPU polar decoder," in *Proceedings of the 2nd IEEE International Conference on Computer and Communications, ICCC 2016*, pp. 1123–1127, Chengdu, China, October 2016.

[22] S. Cammerer, B. Leible, M. Stahl, J. Hoydis, and S. Ten Brink, "Combining belief propagation and successive cancellation list decoding of polar codes on a GPU platform," in *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2017*, pp. 3664–3668, New Orleans, La, USA, March 2017.

[23] X. Han, R. Liu, Z. Liu, and L. Zhao, "Successive-cancellation list decoder of polar codes based on GPU," in *Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pp. 2065–2070, Chengdu, December 2017.

[24] F. Ercan, C. Condo, and W. J. Gross, "Reduced-memory high-throughput fast-SSC polar code decoder architecture," in *Proceedings of the 2017 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, Lorient, France, October 2017.

[25] Y. S. Park, Y. Tao, S. Sun, and Z. Zhang, "A 4.68Gb/s belief propagation polar decoder with bit-splitting register file," in *Proceedings of the 2014 IEEE Symposium on VLSI Circuits*, pp. 1-2, Honolulu, Hawaii, USA, June 2014.

[26] O. Dizdar and E. Arikan, "A high-throughput energy-efficient implementation of successive cancellation decoder for polar codes using combinational logic," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 3, pp. 436–447, 2016.

[27] A. Balatsoukas-Stimming, A. J. Raymond, W. Gross, and A. Burg, "Hardware Architecture for List SC Decoding of polar codes," https://arxiv.org/abs/1303.7127.