Pipelined BP Polar Decoder with a Novel Updated Scheme

Xiaojun Zhang[1,2](✉), Na Li[1], Jun Li[1], Chengguan Chen[1], Hengzhong Li[1], and Geng Chen[1]

[1] Shandong University of Science and Technology, Qingdao, China
zhangxiaojun@sdust.edu.cn
[2] State Key Laboratory of High-End Server and Storage Technology, Jinan, China

Abstract. Compared with the SC decoder, BP decoder provides a higher throughput and lower decoding latency for its inherent parallel nature. However, the functional units of existing BP decoders are not fully utilized. In this paper, we propose a new update scheduling scheme and hardware optimization design to improve the hardware efficiency of BP decoder. First, a pipelined decoder architecture is proposed to reduce the consumption of functional units. Then, a new update scheduling scheme is proposed, when updating the messages, both new-value and old-value approaches are used to improve the utilization of functional units and reduce the decoding latency. The analysis and synthesis results have shown that, compared with the existing methods, the proposed decoder suffers from a slight the decoding performance degradation, but the utilization rate of basic computational blocks (BCB) can be increased to 50.58%. The storage resource dissipation can be reduced by 20.1%–34.09% .

Keywords: Polar code · Belief propagation (BP) · VLSI · Pipelined architecture · Hardware efficiency

1 Introduction

Polar code has been proved theoretically to be able to achieve the Shannon capacity over binary-input discrete memoryless channels (B-DMCs), and the codec has a lower algorithm complexity [1]. It is a major breakthrough in the history of channel coding and a research hotspot in the field of coding. There exist mainly two decoding algorithms: successive cancellation (SC) decoding [1] and belief propagation (BP) decoding algorithms [2]. The SC algorithm has low decoding complexity and excellent error-correction performance, but the decoding latency is very high, which limits the system throughput and hinders the wide applications of polar codes, especially for high-speed applications.

Unlike SC decoding algorithm, BP decoding is an iterative algorithm that is highly parallel, which provides high throughput and low decoding latency [3, 4]. In 2013, B. Yuan and K. K. Parhi proposed an improved architecture of polar BP decoder in order to improve throughput and efficiency [5]. The following year, they also utilized early stopping techniques to reduce energy consumption and decoding latency [6]. Subsequently, several architecture transformation techniques are given to further improve the hardware performance [7]. Jin Sha et al. exploited a stage-combined decoding scheme to advance the memory efficiency of BP decoder [8]. S. Sun and Z. Zhang [9] proposed an architecture and optimization of BP decoder that eliminates data dependencies by using forward flood scheduling to improve the error correction performance, throughput and reduce the decoding delay. Junmei Yang et al. [10] presented both feed-forward (FFD) and feed-back (FBK) pipelined architecture for BP decoder to achieve a balance between performance, throughput, latency, area, and utilization. However, owing to the underutilization of the functional units, the hardware performance of the decoder is still not competitive.

Assume that the message that has been updated in current iteration is referred to new value (NV). Otherwise, if the message that has been not updated in current iteration, namely, it is still the value in last iteration, which is defined as old value (OV). In conventional BP decoding, the decoder needs to wait when the new value is not valid, which brings in large decoding latency and costs more memories to store internal messages. To improve hardware efficiency, this paper presents an update scheduling scheme. During the iterations. it updates and calculates the messages with old values directly does without waiting old value. Based on it, a pipelined architecture with new and old value (NOV) based on folding technology is presented. The utilization rate of BCB (basic computational block) can be improved to 100%. Simulations show that the proposed scheme can obtain similar decoding performance as other schemes. Meanwhile, it consumes less hardware resources.

The remainder of this paper is organized as follows. Section 2 briefly reviews polar codes and BP decoding algorithm. In Sect. 3, we present a novel update scheduling scheme and hardware architecture of BP decoder. Section 4 compares the performance among different decoders. Conclusions are drawn in Sect. 5.

2 Conventional BP Decoding of Polar Code

2.1 Polar Codes

Polar codes can be defined by a parameter vector (N, K, A, u_{A^c}), where $N = 2^n$ denotes the codeword length, K denotes the number of information bits, A and A^c represents the set of information bits' indices and its complement, u_{A^c} represents frozen bits. The source vector $u_1^N = (u_1, u_2, \cdots, u_n)$ can be obtained by mixing K information bits and $N - K$ frozen bits. The codeword vector $x_1^N = (x_1, x_2, \cdots, x_n)$ is generated as follows. Subsequent paragraphs, however, are indented.

$$x_1^N = u_1^N G_N = u_1^N B_N F^{\otimes n} \tag{1}$$

Where G_N is a generator matrix, $F^{\otimes n}$ is the n-th Kronecker power of $F = \begin{bmatrix} 10 \\ 11 \end{bmatrix}$, and B_N is a bit-reversal permutation matrix.

2.2 Polar BP Decoding Algorithm

The BP algorithm can be illustrated through factor graph. Figure 1 depicts the factor graph of an (8, 4) polar code, which consists of $n = \log_2 N$ stages and $N(n+1)$ nodes. Each stage contains $N/2$ 2×2 BCBs [8], which has two inputs and two outputs as shown in Fig. 2.

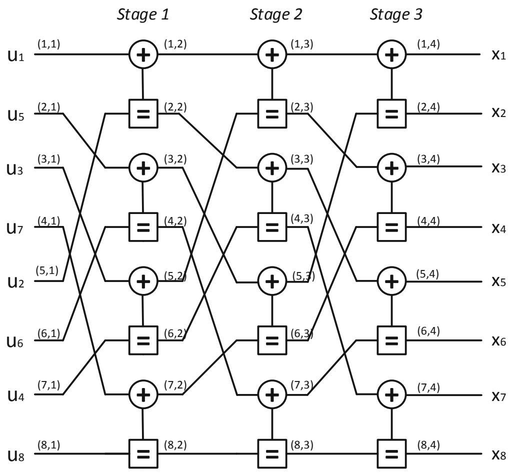

Fig. 1. Factor graph of (8, 4) polar code.

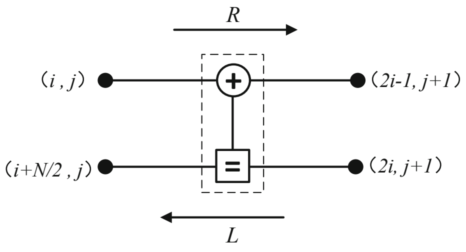

Fig. 2. Factor graph of 2×2 BCB.

The node (i, j) represents the *i*-th input bit of the *j*-th column, which is associated with two types of messages: left-to-right messages $R_{i,j}$ and right-to-left messages $L_{i,j}$. During the decoding iterations, these soft messages are updated and propagated among adjacent nodes. The decoder can be initialized as follows.

$$L_{i,n+1} = \ln\left(\frac{p(y_i|x_i = 0)}{p(y_i|x_i = 1)}\right) \tag{2}$$

$$R_{i,1} = \begin{cases} 0, & \textit{if } i \in A \\ \infty, & \textit{if } i \in A^c \end{cases} \tag{3}$$

In BP decoding, messages are passed iteratively from left to right and then from right to left through the factor graph. In each iteration, the BCB calculates $R_{i,j}$ and $L_{i,j}$ messages according to (4).

$$\begin{aligned} L_{i,j} &= f\left(L_{2i-1,j+1}, L_{2i,j+1} + R_{i+N/2,j}\right) \\ L_{i+N/2,j} &= f\left(L_{2i-1,j+1}, R_{i,j}\right) + L_{2i,j+1} \\ R_{2i-1,j+1} &= f\left(R_{i,j}, R_{i+N/2,j} + L_{2i,j+1}\right) \\ R_{2i,j+1} &= f\left(R_{i,j}, L_{2i-1,j+1}\right) + R_{i+N/2,j} \end{aligned} \tag{4}$$

Where $f(x, y)$ can be further simplified to (5) by using the min-sum approximation [11], and the scaling factor is set to 0.9375.

$$f(x, y) \approx 0.9375 \cdot sign(x) sign(y) \min(|x|, |y|) \tag{5}$$

After the BP decoder reaches the preset maximum number of iterations $I_{\max}$, $\hat{u}_1^N$ can be determined by calculating the LLR values of the leftmost node according to (6).

$$\hat{u}_i = \begin{cases} 0, \ if \ L_{i,1} \geq 0 \\ 1, \ else \end{cases} \tag{6}$$

3 Proposed Nov Update Scheme and Hardware Architecture

3.1 Existing Pipelined BP Decoder

When designing the hardware architecture of the decoder, each stage needs $N/2$ BCBs for an N-bit fully parallel BP polar decoder. The direction of messages update iteratively is from 1-th stage (RS_1) right to (n-1)-th stage (RS_{n-1}), and then from n-th stage (LS_n) left to the 2-th stage (LS_2). After updating messages ($R_{i,j}$) messages or $L_{i,j}$ messages) of each stage takes only one clock cycle, and each iteration takes a total of $2(n-1)$ clock cycles.

Different from the design of the fully parallel BP polar decoder, the whole decoder needs $2(n-1)$ BCBs when using a BCB at each stage [10]. Based on (4) and the FFD pipelined architecture, each BCB updates one pair of messages (or $L_{i,j}$ messages) every clock cycle. When the messages generated by each $R_{i,j}$ BCB update can not consumed immediately in the next cycle, the generated messages should be delayed. For each BCB, if the required input messages is not generated in time during this cycle, it needs to wait for those messages to arrive before it is activated. FFD architecture uses one BCB at each stage, for the N-bit BP polar decoder, the total number of BCBs is

$$Q_{BCB} = 2(\log_2 N - 1) \tag{7}$$

The decoding latency can be evaluated by

$$D_{\text{latency}} = (\sum_{i=1}^{\log_2 N - 1} 2^i + 2(\log_2 N - 1))I + N/2 \tag{8}$$

The BCB utilization is given by

$$U_{BCB} = \frac{N/2}{\sum_{i=1}^{\log_2 N-1} 2^i + 2(\log_2 N - 1)} \tag{9}$$

3.2 Proposed NOV Updating Scheme

In the FFD pipelined architecture, since the ability of one BCB to update messages is insufficient at each stage, the generated messages need to be delayed if it cannot be immediately consumed. Accordingly, the corresponding register resource will be increased. When BCB updates the messages, it will wait until the required messages is arrived. Subsequently, the overall decoding delay will be increased and the utilization of BCB will also be declined.

For the aforementioned analysis, a novel update scheduling scheme is proposed to overcome those flaws based on the overlapped scheduling method used in the iteration level and codeword level [6]. During the current iteration, the updated messages are called NVs, and the messages that are not updated are called OVs. The OV is essentially the result of last iteration. Similarly, only one pair of $L_{i,j}$ messages (or $R_{i,j}$ messages) can be updated for each BCB in one clock cycle. If the NV that is generated cannot be immediately consumed in the next cycle, it needs to be delayed. However, when the BCB of each stage is updated, if the required messages do not arrive yet, it does not have to wait until those messages are the NV, instead of dealing with the corresponding OV. The BCB of each stage is activated by rotation in the above manner until the preset $I_{\max}$ is reached. In the above decoding process, since two kinds of messages, NV and OV, are always involved, this method is called NOV update scheduling scheme. By adopting this scheme, without considering whether all messages is updated or not. Thus the idle cycle of BCB can be greatly reduced, and the cycles of one iteration will be significantly reduced. On the other hand, the messages that needs to be delayed will be avoided, which can reduce the memories. Similarly, this scheme can also be adopted in the multi-level pipeline architecture.

To further illustrate the NOV scheduling process, an 8-bit polar decoder is shown in Fig. 3. The red dotted line represents the boundary of adjacent iterations, with the first iteration on the left and the second iteration on the right. For RS_1 stage, BCB will be activated in the first cycle to produce right messages: $R_{1,2}$ and $R_{5,2}$, that is NV. Then, $R_{1,2}$ will be consumed by BCB of RS_2 stage in the second cycle, while $R_{5,2}$ is not the right messages required by RS_2 stage in this cycle, so it needs to be delayed. For RS_2 stage, BCB will be activated in the second cycle, and right messages it needs is $R_{1,2}$ and $R_{3,2}$, while $R_{3,2}$ is updated in the third cycle of the first iteration. At this point, instead of waiting for $R_{3,2}$ to be NV, BCB is activated by replacing $R_{3,2}$ with corresponding OV. Each BCB is activated as described above until $I_{\max}$ is reached and decoding is terminated.

3.3 Hardware Architecture

According to (4), assume that the 4-input ports and the 2-output ports of BCB are *a*, *b*, *c*, *d*, and *out1*, *out2*, respectively. Then (4) can be further simplified to (10). The logical

Clock Cycle	1	2	3	4	5	6	7	8	9	···
RS1	$R_{1,2}$ $R_{5,2}$	$R_{2,2}$ $R_{6,2}$	$R_{3,2}$ $R_{7,2}$	$R_{4,2}$ $R_{8,2}$	$R_{1,2}$ $R_{5,2}$	$R_{2,2}$ $R_{6,2}$	···			
RS2		$R_{1,3}$ $R_{3,3}$	$R_{2,3}$ $R_{4,3}$	$R_{5,3}$ $R_{7,3}$	$R_{6,3}$ $R_{8,3}$	$R_{1,3}$ $R_{3,3}$	$R_{2,3}$ $R_{4,3}$	···		
LS3			$L_{1,3}$ $L_{2,3}$	$L_{3,3}$ $L_{4,3}$	$L_{5,3}$ $L_{6,3}$	$L_{7,3}$ $L_{8,3}$	$L_{1,3}$ $L_{2,3}$	$L_{3,3}$ $L_{4,3}$	···	
LS2				$L_{1,2}$ $L_{3,2}$	$L_{2,2}$ $L_{4,2}$	$L_{5,2}$ $L_{7,2}$	$L_{6,2}$ $L_{8,2}$	$L_{1,2}$ $L_{3,2}$	$L_{2,2}$ $L_{4,2}$	···

Fig. 3. NOV updating schedule of 8-bit polar code. (Color figure online)

architecture of BCB is illustrated in Fig. 4, and its hardware architecture is shown in Fig. 5. Here T2C module converses sign-magnitude (SM) to 2's complement, and C2T is inversely converted. Moreover, SU module performs the scaling function, MIN module can get the minimum value. The BCB updates the $R_{i,j}$ (or $L_{i,j}$) messages from left to right (or right to left) as shown in Fig. 6. Obviously, the BCB is a bidirectional update function.

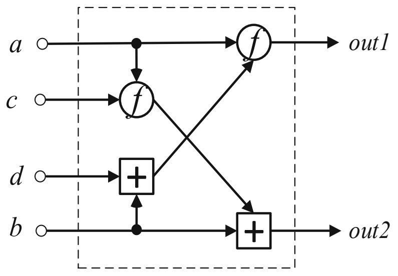

Fig. 4. Architecture of BCB.

$$out1 = f(a, b + d), out2 = f(a, c) + b \tag{10}$$

An N-bit pipelined BP decoder architecture with the NOV update scheme is shown in Fig. 6. The switch module has 4-input ports and 4-output ports, which updates $L_{i,j}$ messages of the LS_1 from right to left for the last iteration and is controlled by the signal I, when $I = I_{\max}$, the switch module is activated. The same BCB can be used with the assistance of the switch module because of RS_1 and LS_1 just update the messages in different directions. The hard decision module is composed of the sign module and the reorder module, which determines the codeword estimation value $\hat{u}$, and is also activated when $I = I_{\max}$. The sign module is used to gain the sign bit of the LLR. The reorder module is employed to perform bit reversal operation. The $\mathrm{MR_n}$ and $\mathrm{ML_n}$ modules are utilized to deposit $R_{i,j}$ and $L_{i,j}$ messages of the n-th stage generated by the BCB respectively, but they are not pure memories, as shown in Fig. 8. Here switch acts

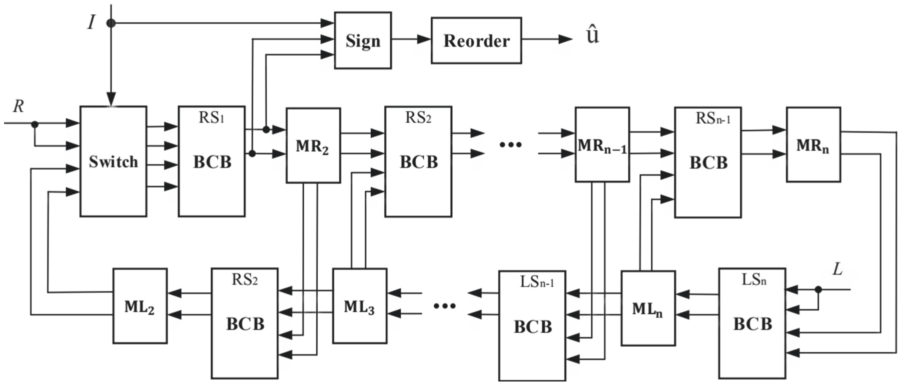

Fig. 5. Hardware architecture of BCB.

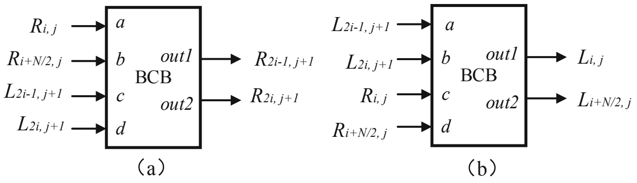

Fig. 6. (a) Update messages from left to right. (b) Update messages from right to left.

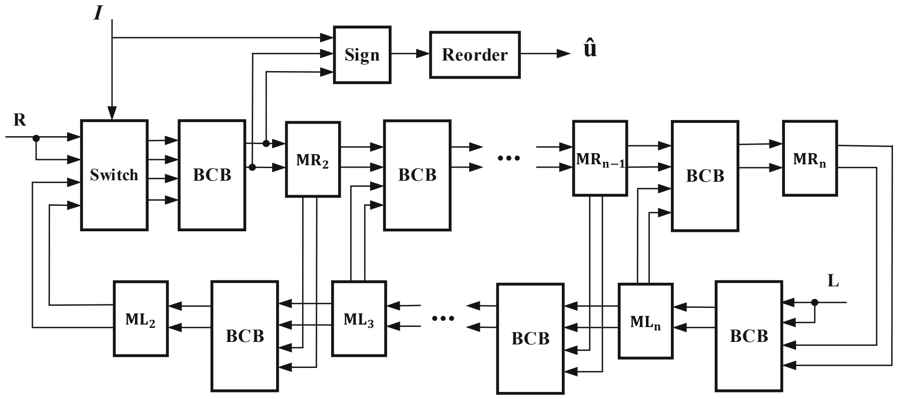

Fig. 7. N-bit pipeline BP decoder architecture with the NOV update scheme.

as a commutator, FIFO is used to store NVs that needs to be delayed, RAM is used to store part of OVs, where the corresponding OVs will be overwritten by the result of the next iteration.

For an 8-bit pipelined BP decoder architecture with the NOV update scheme, one BCB is adopted for each stage, and a total of 4 BCBs is required. Each iteration takes 4 clock cycles. Moreover, BCB takes 4 clock cycles just to update the messages of this

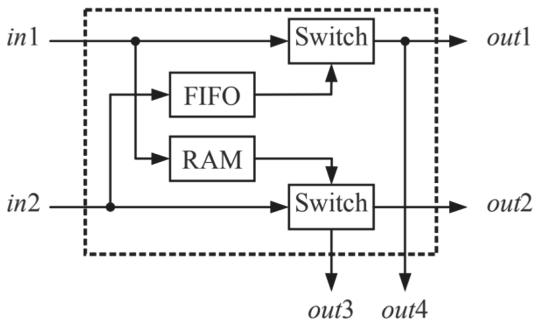

Fig. 8. Memory module.

stage during that time, thus the utilization rate of BCB is 100%. If the decoding result is outputted, it also takes 4 clock cycles, and the decoding latency is $4I + 4$ clock cycles.

In Fig. 7, the total number of BCBs is

$$Q_{BCB} = 2(\log_2 N - 1) \tag{11}$$

The decoding latency is calculated by

$$D_{latency} = \frac{N}{2} I + N/2 \tag{12}$$

The BCB utilization is

$$U_{BCB} = \frac{N/2}{N/2} \tag{13}$$

4 Implementation and Comparison

In this section, we analyze and compare the decoding performance and hardware performance under different polar BP decoder architectures, in order to illustrate the advantages of the proposed architecture with the NOV scheduling scheme.

To evaluate the decoding performance of the proposed scheme, A (1024, 512) polar code is simulated under AWGN (Additive White Gaussian Noise) channel with BPSK(Binary Phase Shift Keying) modulation. Simulation results are depicted in Figs. 9 and 10. Compared with the conventional BP decoding, the proposed has a slight decoding performance degradation in the low SNR regions. This is mainly because we used some old values during iterative process, which lowers the convergence rate of decoding. But it is very close to the performance of the conventional BP decoding in high SNR regions. When $I_{\max} = 80$, the proposed scheme is very close to the conventional one.

Table 1 lists the comparison of BCB number, BCB utilization and decoding latency of FFD and the proposed NOV at different code lengths. From the table, it can be shown that the number of BCB required that of the proposed in this paper is the same as the FFD. Under different code lengths, the BCB is always in active state for the duration of

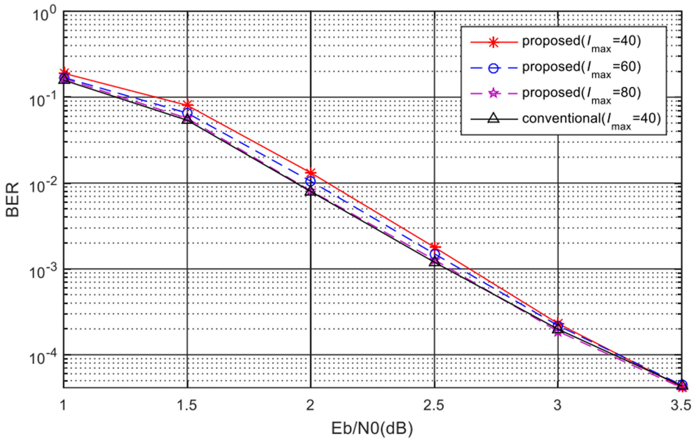

Fig. 9. BER performance for (1024, 512) polar codes with different $I_{\max}$.

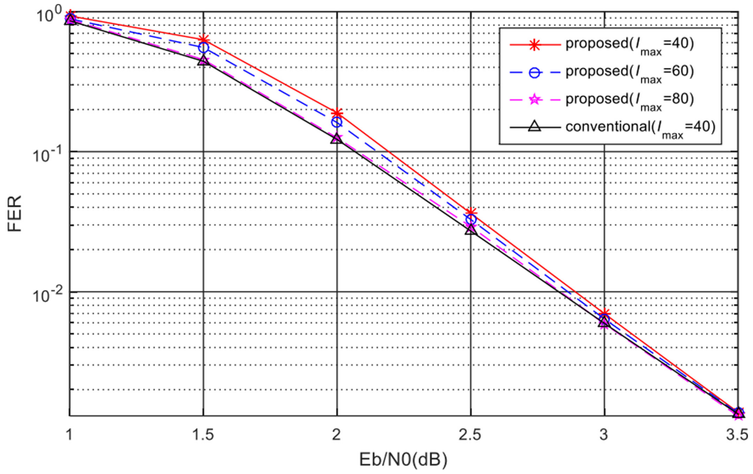

Fig. 10. FER performance for (1024, 512) polar codes with different $I_{\max}$.

iteration with the NOV scheduling strategy. Accordingly, under different code lengths, one iteration and without decoding results, the decoding latency can be reduced by 50.7%, and the utilization ratio of BCB can be increased to 50.6%.

Table 2 shows the comparison of the decoding latency between the proposed NOV and FFD with the average number of iterations with two early stopping schemes (minLLR and G-matrix [8]), when $I_{\max} = 40$ at SNR = 3.5 dB. According to Table 2, compared with FFD, the average number of iterations of NOV increases about three times with two early stopping schemes, this is also caused by using some old values. Furthermore, the decoding latency is increased by 40.7% with the average number of iterations, but the decoding latency can be reduced by 50.2% with the same iteration number.

Table 1. Comparison of decoding latency, BCB number, and BCB utilization for different code lengths

Code length (bit)	Decoding latency		Number of BCB		BCB utilization	
	FFD [10]	NOV	FFD [10]	NOV	FFD [10]	NOV
512	782	512	16	16	48.67%	100%
1024	1552	1024	18	18	49.23%	100%
2048	3090	2048	20	20	49.56%	100%
4096	6164	4096	22	22	49.76%	100%
8192	12310	8192	24	24	49.87%	100%

Table 2. Comparison of the decoding delay under the average number of iterations

Design	SNR (dB)	Average number of iterations		Decoding latency	
		FFD [10]	NOV	FFD [10]	NOV
Proposed without stopping criteria	3	40	40	42112	20992
	3.5	40	40	42112	20992
Proposed with minLLR	3	6.70	19.53	7480	10511.3
	3.5	5.53	16.45	6263.2	8934.4
Proposed with G-matrix	3	6.20	17.85	6960	9651.2
	3.5	5.12	15.04	5836.8	8212.4

The synthesis results of BP polar decoder with different decoding schemes are shown in Table 3. Let (Q_s, Q_i, Q_f) denote the quantization scheme, where Q_s presents the quantization bits of sign bit, Q_i and Q_f denotes the integer bits and fractional bits respectively. The quantization scheme (1, 8, 3) is adopted to quantize the LLRs of BP decoders. From Table 3, it can be seen that compared with FFD and FBK architectures under the 12-bit quantization scheme, NOV architecture can reduce the consumption of register and block memory by 25.8% to 38.9%, respectively, and the frequency can be increased by 1.9 to 2.1 times.

Table 3. Synthesis and simulation results under different methods

Hardware overheads	Existing methods		NOV
	FFD [10]	FBK [10]	
Logic unit	2,738	2,942	212,408
Storage resource	225,099	185,674	148,364
Throughput (Mbps)	17.28	10.82	10.74
Frequency (MHz)	105.69	96.02	93.75

5 Conclusion

In this paper, we propose a new update scheduling scheme and an optimized hardware architecture to improve the hardware efficiency of polar BP decoder. This research have been implemented and verified at FPGA platform. Synthesis results show that the proposed architecture can significantly improve the hardware utilization, and lower memory consumption compared with the existing methods.

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. Arıkan, 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)
2. Pamuk, A.: An FPGA implementation architecture for decoding of polar codes. In: International Symposium on Wireless Communication Systems, pp. 437–441 (2011)
3. Yu, Y., Pan, Z., Liu, N., You, X.: Belief propagation bit-flip decoder for polar codes. IEEE Access **7**, 10937–10946 (2019)
4. Wang, X., Zheng, Z., Li, J., Shan, L., Li, Z.: Belief propagation bit-strengthening decoder for polar codes. IEEE Commun. Lett. **23**(11), 1958–1961 (2019)
5. Yuan, B., Parhi, K.K.: Architecture optimizations for BP polar decoders. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2654–2658 (2013)
6. Yuan, B., Parhi, K.K.: Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders. IEEE Trans. Signal Process. **62**(24), 6496–6506 (2014)
7. Yuan, B., Parhi, K.K.: Architectures for polar BP decoders using folding. In: Proceedings of the International Symposium on Circuits and Systems (ISCAS), Melbourne, Australia, pp. 205–208 (2014)
8. Sha, J., Xing, L., Wang, Z., Zeng, X.: A memory efficient belief propagation decoder for polar codes. China Commun. **12**(5), 34–41 (2015)

9. Sun, S., Zhang, Z.: Architecture and optimization of high-throughput belief propagation decoding of polar codes. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 165–168 (2016)
10. Yang, J., Zhang, C., Zhou, H., You, X.: Pipelined belief propagation polar decoders. In: IEEE International Symposium on Circuits and Systems (ISCAS) (2016)
11. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Trans. Inf. Theory **47**(2), 498–519 (2001)