A Simplified Decoding Algorithm for Polar Codes

Qiuran Ma [a], Hongfeng Gao [b]

Henan University of Science and Technology, Henan 471000, China.

[a]qiuranma@qq.com, [b]gaohongfenghappy@126.com

Abstract

The successive cancellation (SC) decoding method for polar codes under additive white Gaussian noise (AWGN) channels is performed in the log-likelihood ratio domain. The function   uses a sum-product algorithm based on hyperbolic tangent rules. Hyperbolic tangent requires a large number of multiplication and exponentiation operations, and the decoder has a high complexity. In this paper, a polyline approximation algorithm is proposed to approximate the hyperbolic tangent function (and its inverse function). The hyperbolic tangent function and its inverse function are simplified to a 9-segment polyline function. Simulation results show that for the polar code of   the Bit Error Rate (BER) obtained by the proposed algorithm is basically the same as the original algorithm, but the decoding speed is increased by approximately twice.

Keywords

Polar code, SC decoding, Sum-product algorithm, Polyline approximation algorithm, BER.

1. Introduction

Arıkan proposed the polar code based on the concept of channel polarization [1]. The polar code is the first channel coding method that is rigorously proved to be capable of achieving channel capacity in a binary symmetric discrete memoryless channel (B-DMC) and has clear and simple coding and decoding algorithms.

Under B-DMC, the SC decoding algorithm for polar codes has high reliability. Therefore, the sum-product algorithm based on the hyperbolic tangent rule is used in the log likelihood ratio (LLR) domain in general applications. However, when decoding, the sum-product algorithm involves the hyperbolic tangent function and its inverse function calculation, and the computational complexity is high. For medium and long codewords, the sum-product algorithm is difficult to guarantee real-time performance. In the case of guaranteeing certain accuracy, decoding requires a lot of hardware resources.

Many scholars have studied the sum-product algorithm [2-4,6]. C.Leroux approximates the sum-product algorithm to the min-sum (MS) algorithm and applies it to polar codes decoding [3-4]. He also gives the structure diagram of the sign and magnitude processing elements, and the BER performance curve under the MS algorithm. The BER performance obtained by the minimum sum algorithm is the same as or similar to the sum-product algorithm [3]. In addition, the implementation of the Low Density Parity Check Code (LDPC) decoder also uses the MS algorithm [5]. S.Papaharalabos modified the sum-product algorithm for the LDPC code to a piecewise polyline function, and presented the performance curve of the BER under the sum-product algorithm and the piecewise polyline function algorithm. The simulation results showed that the performance of the BER obtained by the segment polyline algorithm is better than the sum-product algorithm [2,6]. Other decoding algorithms for polar codes are given in [7-11], and the hardware implementation architectures for polar codes decoding are given in [12-15].

Based on the literature [2,3], this paper applies the segment polyline function algorithm in LDPC code to the polar code decoding algorithm, and gives the BER and frame error rate (FER) performance curves. The advantage of using the segment polyline function algorithm is that the linear operation is used instead of the nonlinear operation in the sum-product algorithm, which simplifies the calculation and facilitates the software implementation. The obtained BER performance is basically the same as the BER performance obtained by the sum-product algorithm.

The rest of the paper is organized as follows. Section 2 introduces polar codes and SC decoding processs. Section 3 describes the segment polyline algorithm and its implementation structure. Simulation results are presented and discussed in Section 4. Section 5 gives a simple analysis of complexity. Finally, conclusions are drawn in Section 6.

2. Polar Codes

2.1 Polar Code Encoding

A polar code (N,K) is a linear block code of length $N=2^n$ and rate $R=K/N$ constructed. The recursive concatenation process can be represented by a modulo-2 matrix multiplication as

$$\mathbf{x}=\mathbf{u}\mathbf{F}^{\otimes n} \tag{1}$$

Where $\mathbf{u}=\{u_0,u_1,\dots,u_{N-1}\}$ is the input vector, $\mathbf{x}=\{x_0,x_1,\dots,x_{N-1}\}$ is the coded vector, and the generator matrix $\mathbf{F}^{\otimes n}$ is the n-th Kronecker product of the polarizing matrix $\mathbf{F}=\begin{bmatrix}1&0\\1&1\end{bmatrix}$. The input vector $\mathbf{u}$ is composed of N bits, K of which carry information and are assigned to the K most reliable bit channels. The remaining $N-K$ bits of $\mathbf{u}$ are set to a predefined value (typically 0) known by the decoder. They are thus known as frozen bits, and their set is identified as $\mathbb{F}$. Vector $\mathbf{x}$ is transmitted through the channel, and the decoder receives the vector $\mathbf{y}=\{y_0,y_1,\dots,y_{N-1}\}$.

2.2 SC Decoding in LLR domain

As shown in [3], the SC decoding algorithm successively evaluates the LLR of each bit $\hat{u}_i$. Arıkan showed that these LLR computations can be efficiently performed in a recursive manner by using a data flow graph which resembles the structure of a fast Fourier transform. That structure, shown in Fig.1, is named a butterfly based decoder. As shown in Fig.1, in the LLR domain, messages passed in the decoder are LLR values denoted as $LL_{l,i}$, where l and i correspond to the graph stage index and row index, respectively. Thus, the function f and g nodes in the decoder graph calculate the messages using the following two functions:

$$\lambda_f(\lambda_a,\lambda_b)=2\tanh^{-1}(\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})) \tag{2}$$

$$\lambda_g(\hat{s},\lambda_a,\lambda_b)=\lambda_a(-1)^{\hat{s}}+\lambda_b \tag{3}$$

where for the f function λ_a denotes $LL_{l+1,i}$, λ_b denotes $LL_{l+1,i+2^l}$,for the g function λ_a denotes $LL_{l+1,i-2^l}$, λ_b denotes $LL_{l+1,i}$, $\hat{s}$ denotes $\hat{s}_{l,i-2^l}$, $0\le l<n$, $0\le i<N$.

In Fig.1, y_0^3 is the received signal of the channel after Binary Phase Shift Keying (BPSK) modulation, and the log-likelihood ratio $LL_{2,i}$ of the channel is obtained after initialization. The SC decoder calculates $LL_{1,0}$, $LL_{1,1}$ by two f functions according to the value of $LL_{2,i}$, and then use $LL_{1,0}$, $LL_{1,1}$ calculates $LL_{0,0}$ through a g function. When $LL_{0,0}\ge 0$, $\hat{u}_0$ is judged as 0. When $LL_{0,0}<0$, $\hat{u}_0$ is judged as 1, and at this time, the first element to be judged is activated (the red line in Fig.1 represents the decoding process of $\hat{u}_0$), and pass the decision result to the element to be judged later. Then the SC decoder calculates $LL_{0,1}$ by one g function according to the value of $LL_{1,0}$, $LL_{1,1}$, when calculating the g function, we need to use the part sum $\hat{s}_{0,0}$, $\hat{s}_{0,0}=\hat{u}_0$. When $LL_{0,1}\geq 0$, $\hat{u}_1$ is judged as 0. When $LL_{0,1}<0$, $\hat{u}_1$ is judged as 1, and at this time, the second element to be judged is activated, and pass the decision result to the element to be judged later. By analogy, the decision elements are sequentially activated in the order of 1 to N, and the source bits are sequentially decoded.

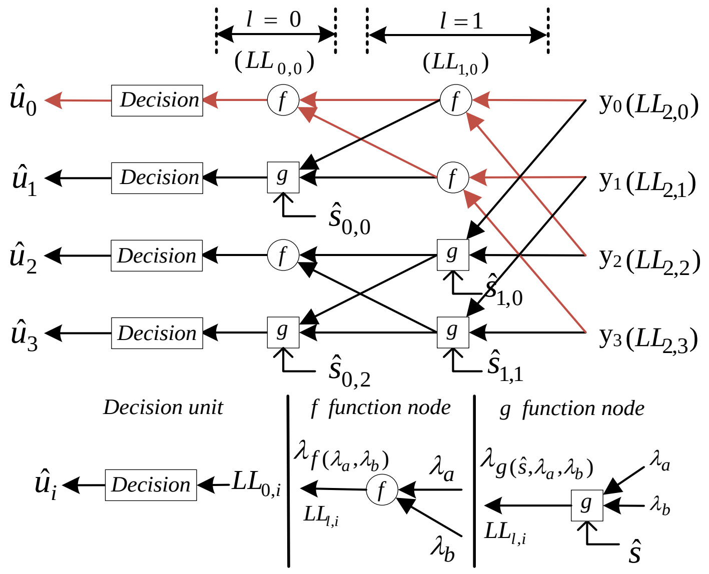

Fig. 1 Butterfly-based SC decoder with $N=4$.

Given a received vector $\mathbf{y}$ corresponding to a transmitted codeword $\mathbf{x}$, the SC decoder successively estimates the transmitted bits u_0 to u_{N-1}. The first step is to decode u_0, the second step is to decode u_1, and the i step is to decode u_{i-1}. At step i, if i is not in the frozen set, the SC decoder estimates $\hat{u}_i$ such that:

$$\hat{u}_i=\begin{cases}0, & if\ \log(\dfrac{\Pr(y,\hat{u}_0^{i-1}\mid u_i=0)}{\Pr(y,\hat{u}_0^{i-1}\mid u_i=1)})\geq 0\\ 1, & otherwise\end{cases} \tag{4}$$

where $\Pr(y,\hat{u}_0^{i-1}\mid u_i=a)$ is the probability that $\mathbf{y}$ was received and the previously decoded bits are $\hat{u}_0^{i-1}$, given the currently decoded bit is a, where $a\in\{0,1\}$. The ratio of probabilities in (2) represents the LLR of bit $\hat{u}_i$.

3. Polyline Approximation Algorithm

3.1 Polyline Approximation Algorithm Principle

It can be seen from formula (2) , the function f uses a sum-product algorithm based on hyperbolic tangent rules, but hyperbolic tangent requires a large number of multiplication and exponentiation operations, and the decoder has a high complexity. In this paper, a polyline approximation algorithm is proposed to approximate the hyperbolic tangent function (and its inverse function).

The polyline approximation algorithm converts nonlinear functions into piecewise linear functions. When the polyline algorithm is implemented, the structure is simple, and the continuous function is estimated with a small number of parameters. The algorithm has been widely used in the fields of modeling, analysis and estimation of a large number of nonlinear dynamic systems.

When implementing a hyperbolic tangent function with an FPGA, if a shift-and-add method is used instead of a multiplication operation, more clock cycles will be consumed and the decoding delay will increase. If the look-up table method is used, the value of the hyperbolic tangent function needs to be stored in the ROM first, and then the input is converted into the look-up table address to obtain the approximate value of the hyperbolic tangent function. However, the increase in the precision will increase the size of the look-up table exponentially. Therefore, the table lookup method is only applicable to applications with low precision requirements. If high-order polynomial approximations are used, more adders and multipliers will be used, consuming too much hardware resources. In this paper, the polyline approximation algorithm is used. This algorithm selects a part of the hyperbolic tangent function and its inverse function to segment it, and the expression of each polyline function is obtained according to the minimum mean square error. This method achieves a good balance between speed, resources, and accuracy.

The hyperbolic tangent function is an odd function about the symmetry of the origin, the definition domain is $(-\infty,+\infty)$,the value range is (-1,1). In this paper, the hyperbolic tangent function is approximated, and a 9-segment piecewise polyline function is obtained. Its expression is formula (5). Fig.2 shows the comparison of the curves of $h_1(x)$ and $\tanh(x)$ functions. It can be seen from Fig.2 that the curves of $h_1(x)$ and $\tanh(x)$ functions are very close, so using $h_1(x)$ instead of $\tanh(x)$ function ,the hardware implementation is simple and does not affect the decoding error rate.

$$h_1(x)=\begin{cases} -1 & x\le -7.0 \\ 0.0012x-0.9914 & -7.0<x\le -3.0 \\ 0.0524x-0.8378 & -3.0<x\le -1.6 \\ 0.322x-0.4064 & -1.6<x\le -0.8 \\ 0.83x & -0.8<x\le 0.8 \\ 0.322x+0.4064 & 0.8<x\le 1.6 \\ 0.0524x+0.8378 & 1.6<x\le 3.0 \\ 0.0012x+0.9914 & 3.0<x\le 7.0 \\ 1 & x>7.0 \end{cases} \tag{5}$$

Each line in the polyline approximation function $h_1(x)$ can be expressed as:

$$h_1(x)=K_1(i)*x+b_1(i) \qquad i=1,...,M_1 \tag{6}$$

where $K_1=\{0,0.0012,0.0524,0.322,0.83,0.322,0.0524,0.0012,0\}$,

$b_1=\{-1,-0.9914,-0.8378,-0.4064,0,0.4064,0.8378,0.9914,1\}$, $M_1=9$, $K_1(i)$ is the slope of the straight line, x is the log likelihood ratio values of the channel, $b_1(i)$ is a constant, M_1 is the number of piecewise functions. In addition, let $X_1=\{-\infty,-7.0,-3.0,-1.6,-0.8,0.8,1.6,3.0,7.0,\infty\}$, X_1 represents the set of interval turning points.

The inverse hyperbolic tangent function is also an odd function about the symmetry of the origin, the definition domain is (-1,1), the value range is $(-\infty,+\infty)$. In this paper, the inverse hyperbolic tangent function is approximated, and obtained a 9-segment piecewise polyline function $h_2(x)$. Its expression is formula (7). Fig.3 shows the comparison of the curves of $h_2(x)$ and $\tanh^{-1}(x)$ functions. It can be seen from Fig.3 that the curves of $h_2(x)$ and $\tanh^{-1}(x)$ functions are very close, so using $h_2(x)$ instead of $\tanh^{-1}(x)$ function ,the hardware implementation is simple and does not affect the decoding error rate.

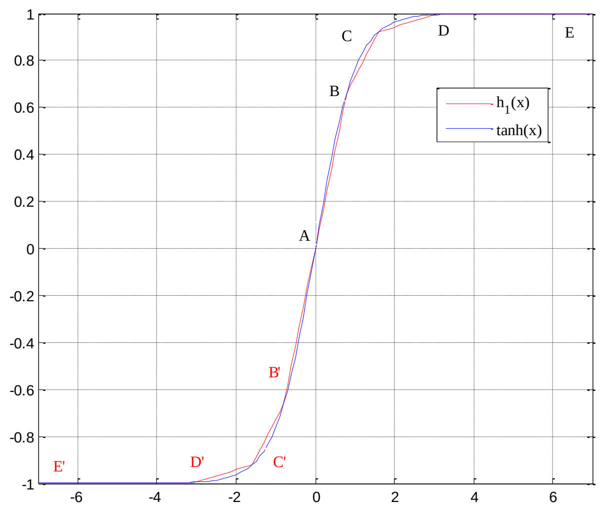

Fig. 2 $h_1(x)$ versus $\tanh(x)$ function .

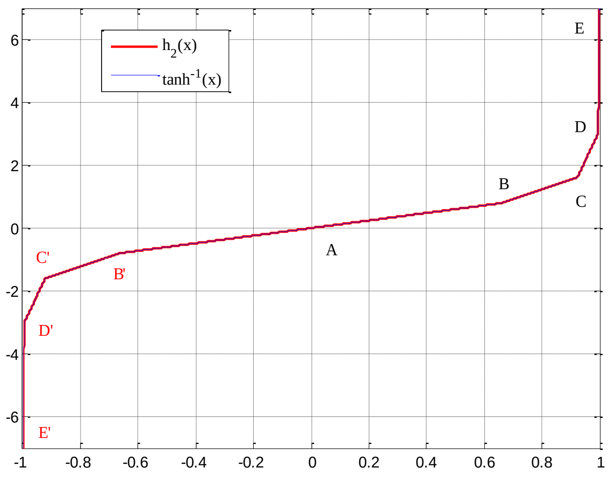

Fig. 3 $h_2(x)$ versus $\tanh^{-1}(x)$ function .

$$h_2(x') = \begin{cases} -7 & x' \le -0.999998 \\ 833.3333x' + 826.1667 & -0.999998 < x' \le -0.9951 \\ 19.0840x' + 15.9885 & -0.9951 < x' \le -0.9217 \\ 3.1056x' + 1.2621 & -0.9217 < x' \le -0.6640 \\ 1.2048x' & -0.6640 < x' \le 0.6640 \\ 3.1056x' - 1.2621 & 0.6640 < x' \le 0.9217 \\ 19.0840x' - 15.9885 & 0.9217 < x' \le 0.9951 \\ 833.3333x' - 826.1667 & 0.9951 < x' \le 0.999998 \\ 7 & x' > 0.999998 \end{cases} \tag{7}$$

Each line in the polyline approximation function $h_2(x')$ can be expressed as:

$$h_2(x') = K_2(j) * x' + b_2(j) \qquad j = 1, \ldots, M_2 \tag{8}$$

where $K_2 = \{0, 833.3333, 19.0840, 3.1056, 1.2048, 3.1056, 19.0840, 833.3333, 0\}$,

$b_2 = \{-7, 826.1667, 15.9885, 1.2621, 0, -1.2621, -15.9885, -826.1667, 7\}$, $M_2 = 9$, $K_2(j)$ is the slope of the straight line, x' is the log likelihood ratio values of the channel, $b_2(j)$ is a constant, M_2 is the number of piecewise functions. In addition, let

$X_2 = \{-\infty, -0.999998, -0.9951, -0.9217, -0.6640, 0.6640, 0.9217, 0.9951, 0.999998, \infty\}$,

X_2 represents the set of interval turning points.

From formula (5) and (7), it can be seen that the segment polyline functions $h_1(x)$ and $h_2(x')$ of $\tanh(x)$ and $\tanh^{-1}(x')$ do not involve logarithmic and exponential operations and only involve addition and multiplication operations. The segment polyline function stores the value of $\tanh(x)$ and $\tanh^{-1}(x')$ by calculating the value of any point on the segment by storing the coordinate of the turning point, the slope and intercept of each segment, and applying coefficient multiplication and addition.

3.2 The Implementation Structure Of Polyline Approximation Function

The polyline approximation function is the main implementation elements of the f function node processing elements.

The implementation structure of $h_1(x)$ is shown in Fig.4. It includes an interval detection circuit, a parameter memory, a receive buffer, an adder, and a multiplier. The parameters of $h_1(x)$ are K_1, b_1, X_1. A set X_1 of interval turning points is stored in the RAM of the interval detection circuit. The coefficients $K_1(j)$ and constant $b_1(j)$ of each segment are stored in the parameter memory, and $(-\infty, \infty)$ is divided into 9 intervals according to the set X_1, where $x_1 \in (X_1(i), X_1(i+1)], i = 1, ..., 8$ is the i interval, $(X(9), X(10))$ is the 9th interval.The input value x is compared with the value in the set X_1 one by one to determine the corresponding interval i. The corresponding $K_1(i)$ and $b_1(i)$ are taken out in the parameter memory. Then $K_1(i)$ is multiplied by x in the data buffer and then added to $b_1(i)$ to get the output value.

The implementation steps are:

a) The input value x is simultaneously sent to the receiving buffer and the interval detection circuit, and is compared with the values in the set X_1 one by one in the interval detection circuit. If $x \in (X_1(i), X_1(i+1)]$, $i = 1, ..., M_1$, then x belongs to the interval i. If $x \notin (X_1(i), X_1(i+1)]$, $i = 1, ..., M_1$, it is judged whether or not $x_1 \le X_1(1)$ is established. If it is true , then let $h_1(x) = -1$, otherwise, let $h_1(x) = 1$.

b) According to the value of i, take the values of $K_1(i)$ and $b_1(i)$ in the parameter storage element;

c) Multiply $K_1(i)$ and x in the receiving buffer and add it to $b_1(i)$ to get the output value $h_1(x)$.

The $h_1(x)$ is computed as shown in algorithm 1.

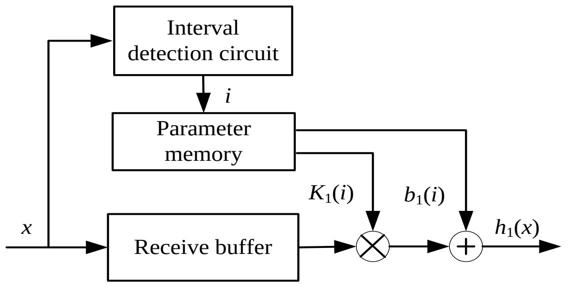

Fig. 4 The implementation structure of $h_1(x)$.

Algorithm 1:

```
  input : $x$
  output : $h_1(x)$
1 for $i = 1$ to $7$ do
2   if $X_1(i) < x \le X_1(i+1)$ then
3       $h_1(x) = K_1(i) * X_1(i) + b_1(i)$
4   elseif
5       $x \le X_1(1)$
6       $h_1(x) = -1$
7   else
8       $h_1(x) = 1$
9   end
10 end
```

The realization structure of the polyline approximation function $h_2(x)$ is the same as $h_1(x)$.The implementation steps are:

a) The input value x is simultaneously sent to the receiving buffer and the interval detection circuit, and is compared with the values in the set X_2 one by one in the interval detection circuit. If $x \in (X_2(j), X_2(j+1)]$, $j = 1,...,M_2$,then x belongs to the interval j. If $x \notin (X_2(j), X_2(j+1)]$, $j = 1,...,M_2$, it is judged whether or not $x \le X_2(1)$ is established. If it is true , then let $h_2(x) = -7$, otherwise, let $h_2(x) = 7$.

b) According to the value of j, take the values of $K_2(j)$ and $b_2(j)$ in the parameter storage element;

c) Multiply $K_2(j)$ and x in the receiving buffer and add it to $b_2(j)$ to get the output value $h_2(x)$.

The $h_2(x)$ is computed as shown in algorithm 2.

Algorithm 2:

```
  input : $x$
  output : $h_2(x)$
1 for $j = 1$ to $7$ do
2   if $X_2(j) < x \le X_2(j+1)$ then
3       $h_2(x) = K_2(j) * X_2(j) + b_2(j)$
4   elseif
5       $x \le X_2(1)$
6       $h_2(x) = -7$
7   else
8       $h_2(x) = 7$
9   end
10 end
```

When $LL_{l+1,i}$ and $LL_{l+1,i+2^l}$ are known, $h_1(\frac{LL_{l+1,i}}{2})$ and $h_1(\frac{LL_{l+1,i+2^l}}{2})$ are first determined according to the function $h_1(x)$, and then the two are multiplied to obtain the parameters of the function $h_2(x)$. Finally, the result obtained by using the f function is $LL_{l,i} = 2h_2(h_1(\frac{LL_{l+1,i}}{2}) * h_1(\frac{LL_{l+1,i+2^l}}{2}))$.

3.3 The Implementation Of f Function

The implementation of f function is shown in Fig.5, including LLR memory, implementation elements of $h_1(x)$ and $h_2(x)$, and some multipliers.

The specific implementation steps of the f function are:

a) Read two log likelihood ratios $LL_{l+1,i}$ and $LL_{l+1,i+2^l}$ from LLR memory and multiply these two values by 1/2, respectively;

b) The two values obtained in step a) are send to the $h_1(x)$ implementation element of Fig.5, and the values of $\tanh(\frac{LL_{l+1,i}}{2})$ and $\tanh(\frac{LL_{l+1,i+2^l}}{2})$ are obtained, then multiply these two values;

c) Send the value obtained in step b) to the implementation element of $h_2(x)$, and then calculate the value of $\tanh^{-1}(x)$. Multiply this value by 2 to obtain the value of $LL_{l,i}$.

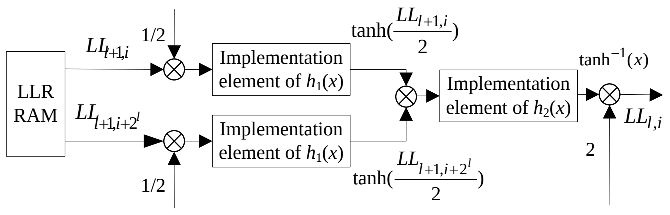
LLR RAM
$LL_{l+1,i}$
$LL_{l+1,i+2^l}$
1/2
1/2
Implementation element of $h_1(x)$
Implementation element of $h_1(x)$
$\tanh(\frac{LL_{l+1,i}}{2})$
$\tanh(\frac{LL_{l+1,i+2^l}}{2})$
Implementation element of $h_2(x)$
$\tanh^{-1}(x)$
2
$LL_{l,i}$

Fig. 5 The implementation structure of f function.

3.4 The Implementation Of g Function

The implementation of g function is shown in Fig.6. It requires an LLR memory, an partial sum register, some adders and some multipliers. The LLR memory stores the log-likelihood ratio values, and the partial sum register stores the partial sum $\hat{s}$. The implementation of partial sum reference [3]. The implementation steps of the g function are:

a) Read two log likelihood ratios values $LL_{l+1,i-2^l}$ and $LL_{l+1,i}$ from LLR memory.

b) Select the desired partial sum $\hat{s}$ from the partial registers.

c) Multiply the partial sum by -2 and add the result to 1, which is to calculate the value of $(-1)^{\hat{s}}$.

d) Multiply the result obtained in step c) by $LL_{l+1,i-2^l}$ and add it to $LL_{l+1,i}$ to obtain the value of $LL_{l,i}$.

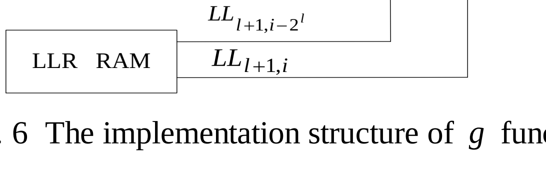

Fig. 6 The implementation structure of g function.

4. Simulation Results

Fig.7 shows the simulation performance curves obtained under the sum-product algorithm (SPA) and polyline approximation algorithm (PAA) for the polar codes with the code lengths of $N = 64$ and $K = 32$ under the AWGN channel. It can be seen from Fig.7 that the proposed polyline approximation algorithm has the same or similar BER or frame error rate (FER) performance as the sum-product algorithm.

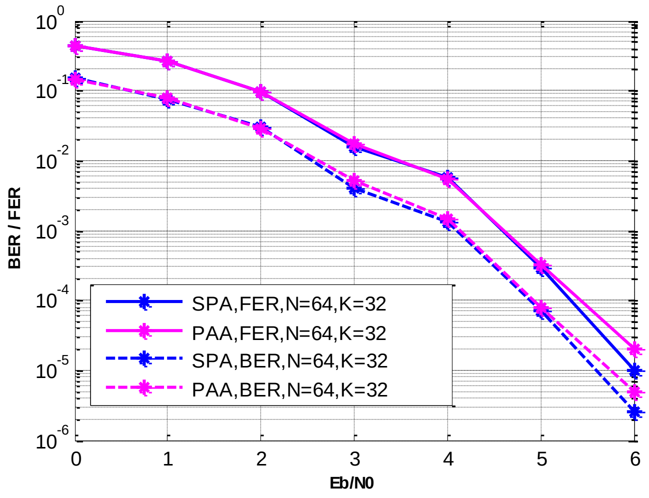

Fig. 7. BER/FER performance of the sum-product algorithm and polyline approximation algorithm for an (64,32) polar code.

5. Complexity analysis

In the decoding stage of the polar code,when the variance obtained by comparing the polyline approximation algorithm with the original SC algorithm is equal to the variance obtained by comparing the look-up table method with the original SC algorithm, the polyline approximation algorithm only use 5 comparison operations, 1 multiplication, and 1 addition, however, if the hyperbolic tangent function (inverse hyperbolic tangent function) is performed using the lookup table method, the quantization interval is 0.007, requiring about 2,000 comparison operations. Table I shows a comparison between the proposed method with the SC decoder when $N = 8$. Therefore, using the polyline approximation algorithm proposed in this paper greatly improves the decoding speed.

Table 1 Complexity Comparison Between Sc and Polyline Approximation Algorithm Methods When $N = 8$.

decoding algorithm	addition operations	multiplication operations	comparison operations
Quantized hyperbolic tangent function	0	0	2000
Quantized inverse hyperbolic tangent function	0	0	2000
$h_1(x)$	1	1	5
$h_2(x)$	1	1	5

6. Conclusion

In this paper, a polyline tangent function (and its inverse function) is proposed to approximate the polyline tangent function. The hyperbolic tangent function and its inverse function are simplified as a 7-segment polyline function. Simulation results show that the proposed algorithm has the same or similar BER performance as the original algorithm for $N = 64, K = 32$ polar codes, but it increases the decoding speed by approximately two times.

References

[1] E. Arıkan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” IEEE Trans. Inform. Theory, vol. 55, pp. 3051-3073, July 2009.

[2] S. Papaharalabos, P. Sweeney, B.G. Evans, “Modified sum-product algorithms for decoding low-density parity-check codes,”. IET Commun., 2007, 1, (3), pp. 294-300.

[3] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, “A semi-parallel successive-cancellation decoder for polar codes,” IEEE Trans. Signal Process., vol. 61, no. 2, pp. 289-299, Jan. 2013.

[4] [4] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware architectures for successive cancellation decoding of polar codes,” in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2011, pp. 1665–1668.

[5] M.P.C. Fossorier, M. Mihaljevic, and H. Imai, “Reduced complexity iterative decoding of low-density parity check codes based on belief propagation,” IEEE Trans. on Comm., vol. 47, no. 5, pp. 673 –680, May. 1999.

[6] Papaharalabos, S., Sweeney, P., Evans, B.G., Albertazzi, G., Vanelli-Coralli, A., and Corazza, G.E.:“Performance evaluation of a modified sum-product decoding algorithm for LDPC codes,”. Proc.2nd Int. Symp. Wireless Commun. Systems (ISWCS), Siena, Italy, September 2005, pp. 800–804.

[7] A. Alamdar-Yazdi and F. R. Kschischang, “A simplified successive-cancellation decoder for polar codes,” IEEE Commun. Lett., vol. 15, no. 12, pp. 1378-1380, Dec. 2011.

[8] K. Niu and K. Chen, “CRC-Aided decoding of polar codes,” IEEE communications letters, vol. 16, no. 10, pp. 1668-1671, Oct. 2012.

[9] 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, May 2014.

[10] A. Balatsoukas-Stimming, M. Bastani Parizi, and A. Burg, “LLR-based successive cancellation list decoding of polar codes,” IEEE Trans. Signal Process., vol. 63, no. 19, pp. 5165-5179, Oct 2015.

[11] I. Tal and A. Vardy , “List decoding of polar codes,” IEEE Transactions on Information Theory, vol. 61, no. 5, pp. 2213-2226, May 2015.

[12] A. Pamuk, “An FPGA implementation architecture for decoding of polar codes,” in International Symposium on Wireless Communication Systems (ISWCS), Nov. 2011, pp. 437-441.

[13] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, “Hardware architectures for successive cancellation decoding of polar codes,” in Proc. IEEE Conf. Int. Acoust., Speech, Signal Process. (ICASSP), 2011, pp. 1665-1668.

[14] C. Leroux, A. J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W. J. Gross, “Hardware implementation of successive-cancellation decoders for polar codes,” Journal of Signal Processing Systems , vol. 69, no. 3, pp. 305-315, Dec. 2012.

[15] Y. Fan and C.-Y. Tsui, “An efficient partial-sum network architecture for semi-parallel polar codes decoder implementation,” IEEE Transactions on Signal Processing, vol. 62, no. 12, pp. 3165–3179, Jun. 2014.