IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 21, 2021, accepted March 9, 2021, date of publication March 12, 2021, date of current version April 1, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3065776*

# An FPGA-Based LDPC Decoder With Ultra-Long Codes for Continuous-Variable Quantum Key Distribution

**SHEN-SHEN YANG[1,2,4,5], JIAN-QIANG LIU[1,2], ZHEN-GUO LU[1,2], ZENG-LIANG BAI[3], XU-YANG WANG[1,2], AND YONG-MIN LI[1,2]**

[1]State Key Laboratory of Quantum Optics and Quantum Optics Devices, Institute of Opto-Electronics, Shanxi University, Taiyuan 030006, China
[2]Collaborative Innovation Center of Extreme Optics, Shanxi University, Taiyuan 030006, China
[3]School of Information, Shanxi University of Finance and Economics, Taiyuan 030006, China
[4]College of Physics and Information Engineering, Shanxi Normal University, Linfen 041004, China
[5]Key Laboratory of Spectral Measurement and Analysis of Shanxi Province, Shanxi Normal University, Linfen 041004, China

Corresponding author: Yong-Min Li (yongmin@sxu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2016YFA0301403, in part by the National Natural Science Foundation of China (NSFC) under Grant 11774209 and Grant 11804208, in part by the Key Research and Development Project of Shanxi Province under Grant 201803D121065, in part by the Applied Basic Research Program of Shanxi Province under Grant 201801D221010, in part by the Scientific and Technological Innovation Programs of Higher Education Institutions in Shanxi under Grant 2019L0479, in part by the Youth Scientific Research Foundation of Shanxi University of Finance and Economics under Grant QN-2019021, and in part by the Shanxi under Grant 1331KSC.

**ABSTRACT** In this paper, we propose a good decoding performance, low-complexity, and high-speed decoder architecture for ultra-long quasi-cyclic LDPC codes by using the layered sum-product decoding scheme. To reduce implementation complexity and hardware resource consumption, the messages in the iteration process are uniformly quantified and the function $\Psi(x)$ is approximated with second-order functions. The decoder architecture improves the decoding throughput by using partial parallel and pipeline structures. A modified construction method of parity check matrices was applied to prevent read&write conflicts and achieve high-speed pipeline structure. The simulation results show that our decoder architecture has good performance at signal-to-noise ratios (SNRs) as low as $-0.6$ dB. We have implemented our decoder architecture on a Virtex-7 XC7VX690T field programmable gate array (FPGA) device. The implementation results show that the FPGA-based LDPC decoder can achieve throughputs of 108.64 Mb/s and 70.32 Mb/s at SNR of 1.0 dB when the code length is 262,144 and 349,952, respectively. The decoder can find useful applications in those scenarios that require very low SNRs and high throughputs, such as the information reconciliation of continuous-variable quantum key distribution.

**INDEX TERMS** Low-density parity check (LDPC) decoder, field programmable gate array (FPGA), ultra-long codes, layered sum-product decoding, side information.

## I. INTRODUCTION

Low-density parity check (LDPC) codes, a class of forward error correction codes proposed by Gallager [1], have attracted extensive attentions over the past few decades. They have been shown to facilitate information rates very close to the Shannon limit, which has motivated the inclusion of LDPC codes in many modern communications standards, such as wireless communications, digital video broadcasting, as well as data storage systems. In practical applications, dedicated hardware are often employed to improve the data throughput, such as field programmable gate arrays (FPGAs) [2], [3], graphics processing units (GPUs) [4]–[6], CMOS integrated circuits [7]–[10], and so on.

FPGAs are a class of large-scale programmable integrated devices and particularly well suited to facilitate rapid prototyping. The designer can configure the degree of parallelism flexibly to control over the trade-off between the algorithm throughput and hardware resource requirements. Owing to their high processing speed, parallelism, re-programmability,

The associate editor coordinating the review of this manuscript and approving it for publication was Rui Wang.

and low power consumption, a great deal of research effort has been invested into FPGA-based LDPC decoders over the past few decades [2]. The decoders can be characterized by some key features: decoding performance, decoding throughput, the consumption of hardware resources, implementation complexity, and flexibility.

Decoding performance of LDPC codes mainly depends on the decoding algorithm, code length, and constructions of check matrices, and so on. In [11], researchers used the sum-product algorithm (SPA) to implement the FPGA-based LDPC decoder and effectively maintained the decoding performance. The survey presented in [2] shows that most existing decoders used the min-sum algorithm (MSA). Although the MSA is easier to implement in hardware, it will cause degradation of the decoding performance. On the other hand, long code length is beneficial to improve decoding performance. In [12], [13], FPGA-based LDPC decoders with long code length of 65,000 and 32,643 were designed. An LDPC decoder presented in [8] can implement a code length of 96,000 based on the CMOS process.

Four major parameters affect decoding throughput: the parallelism parameter, the number of bits in the codeword, code rate, and the number of iterations [14]. Parallel processing is an effective method to improve the decoder throughput. Fully parallel decoders can achieve very high decoding throughput. However, this is achieved at the cost of excessive hardware resource consumption [15] and therefore only applicable to the decoders for small mapping matrices [16], due to the limited logical resources provided by the state of the art hardware. In comparison, partial parallel decoders are a recommended strategy to obtain a good trade-off between computational complexity and decoding throughput [17]–[20].

The consumption of hardware resources mainly includes look-up tables (LUTs), memory, flip-flops (FFs), and DSP slices. It is a challenge for designers to achieve a good trade-off between the throughput and resource consumption. Some memory system optimization schemes have been proposed to reduce the memory requirement [19], [21].

The implementation complexity depends on two major factors: the complexity of computations at each processing node, and the complexity of interconnection. In [22], a novel technique is presented to simplify the check node operation. Ref. [23] presents a low routing complexity LDPC decoder design by reducing the required interconnections in the critical path of the routing network.

Note that previous FPGA-based decoders cannot meet the needs of some special applications, such as the information reconciliation of continuous-variable quantum key distribution (CV QKD) systems [24], [25]. In a CV QKD system, the sender and receiver obtain a set of correlated Gaussian symbols after the quantum signal is modulated, transmitted and detected. After quantizing the Gaussian signal into bit strings, the initial bit error rate is larger than traditional telecommunication signal, owing to the system runs in a Gaussian channel with a very low signal-to-noise ratio (SNR). Therefore, this special scenario requires error correction codes with good decoding performance at low SNR. In practical applications, it also requires high throughput to match the secret key generation rate. Further, the minimum SNR that can be successfully decoded is also crucial.

Combining LDPC codes with the good decoding performance and FPGA devices with powerful parallel processing, we first proposed an FPGA-based LDPC decoder based on the side information (SI) [26] layered sum-product scheme to meet the needs of the CV QKD systems. Another key point we utilized to improve the decoding performance is to increase the code length above 200,000. However, this puts forward high demands on the hardware resource consumption. To improve the throughput, the partially parallel and pipeline structure was applied to the LDPC decoder. The pipeline technique only requires a small number of clock cycles and no empty clocks are needed to insert between layers and nodes, thus has high clock efficiency. To prevent read&write conflicts in pipeline structure, a modified parity check matrix construction method is proposed. We employ two methods to reduce the hardware resource consumption and implementation complexity, one is the uniformly quantified fixed-point number format for log-likelihood ratios (LLRs) and node messages, the other is piecewise approximation of the complicated function $\Psi(x)$.

We evaluated the decoding on a Xilinx VC709 evaluation board, which is populated with a Virtex-7 XC7VX690T FPGA. The implementation results show that the decoder can achieve throughputs of 108.64 Mb/s and 70.32 Mb/s at SNR $= 1.0$ dB when the code length is 262,144 and 349,952, respectively. The minimum SNR for successfully decoding can reach $-0.6$ dB when the code length is 349,952.

The rest of this paper is organized as follows. In Section II, we present the required building blocks of LDPC codes, including decoding algorithms and matrix construction methods, etc. Then, Section III presents the overall FPGA-based LDPC decoder architecture in detail. The implementation results of our decoder is presented in Section IV. Finally, we give a summary in Section V.

## II. OVERVIEW OF LDPC CODES

In general, LDPC codes need three basic procedures from design to application. First, we need to search for the optimal degree distribution according to the parameters of the channel and decoding algorithm; then, construct parity check matrices (PCMs) according to the degree distribution; finally, implement encoding and decoding. The latter two procedures have a great effect on FPGA-based decoders.

### A. CONSTRUCTIONS OF PCMs

The goal of constructing a PCM is to determine the connection between variable nodes and check nodes, which is called an ''edge''. PCMs can be represented by a factor graph or a matrix, as shown in Fig. 1 and (1), these two representations are equivalent. Some techniques have been proposed for placing edges. Randomly-designed codes potentially achieve better decoding performance, owing to the maximized degree

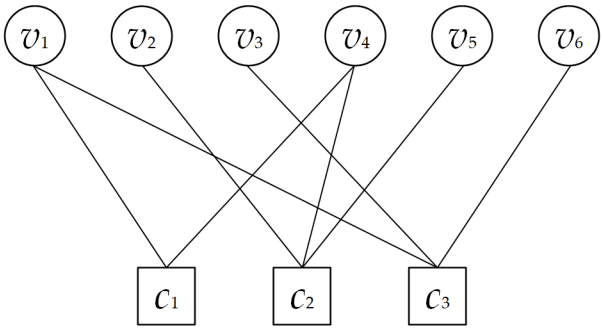



**FIGURE 1.** A factor graph for the LDPC code.

of freedom they afford. The Progressive-Edge-Growth (PEG) algorithm [27] is also an important technique of constructing PCMs. It places an edge in the location that is identified as maximizing the length of cycle, before continuing the algorithm with the selection of different variable nodes. In this way, a factor graph having no short cycles can be created and yielding a strong decoding performance [2].

A class of structured LDPC codes, namely quasi-cyclic LDPC (QC-LDPC) codes, can reduce the implementation complexity when implemented in FPGAs. QC-LDPC codes also facilitate efficient high-speed decoding due to the regularity of their PCMs [11]. The PCMs of QC-LDPC codes are defined by a base matrix $\boldsymbol{H}_b$, where each non-zero elements represent a square submatrix of dimensions $q \times q$. Each submatrix has a different offset factor and all offset factors construct an offset matrix. Fig. 2 presents a PCM of a QC-LDPC code, whose base matrix and offset matrix is given by (1) and (2), respectively ($q = 3$).

$$\boldsymbol{H}_b = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$\boldsymbol{Q} = \begin{bmatrix} 0 & - & - & 1 & - & - \\ - & 1 & - & 2 & 0 & - \\ 2 & - & 0 & - & - & 1 \end{bmatrix} \tag{2}$$

$$H = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc} 1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\ \hline 0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0 \\ \hline 0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0 \end{array}\right]$$

**FIGURE 2.** An expanded PCM with QC construction.

We use the random construction or PEG algorithm to construct a base matrix, then employ the QC method to expand the PCM. According to the characteristics of FPGA data reading and writing, a constraint condition is added when constructing the base matrix, that is, the interval between two adjacent non-zero elements in the base matrix must be large enough. Because the row processor and the column processor work jointly, the rows which have intersection (there are non-zero elements at the same column) cannot be processed simultaneously. Besides, because the processing results will be calculated in several clock cycles later, the intersected rows should be processed successively with several clock cycles' spacing. Therefore, the degree of parallelism is limited. To increase the parallel degree, the code should be specially constructed to break this limit. According to the constraint condition above, the base matrix should not be too small, otherwise, it will cause the matrix construction to fail.

### B. DECODING ALGORITHMS

LDPC decoding algorithms mainly include two classes of algorithms for the messages passing, hard-decision and soft-decision. In the former case, such as the bit-flipping algorithm [28], binary hard decisions are made on the data received from the transmission channel, then the binary sequence obtained by the hard-decision is passed to the iterative process. This class of algorithms has a lower implementation complexity with the cost of poorer decoding performance. In the case of soft-decision based algorithms, such as the MSA [29], [30] and the SPA [31], the input data of a decoder is the channel probabilities represented in the logarithmic ratio which is also known as the log-likelihood ratio (LLR).

In general, the SPA algorithm has the best error rate performance. Although it has been least preferred for hardware realization due to its computational complexity. To obtain the best decoding performance, which is crucial for information reconciliation of CV QKD, we adopt the SPA in our scheme.

### C. MESSAGE PASSING SCHEDULES

LDPC codes can be effectively iteratively decoded using the message-passing schedules. All nodes are processed in an order determined by the LDPC decoder's schedule, it has a significant effect upon the LDPC decoder's decoding capability and other characteristics. The schedule of the LDPC decoding process determines the order variable nodes and check nodes are processed, as well as whether multiple nodes are processed in parallel. The three most common schedules [2] are flooding schedule, layered belief propagation (LBP) schedule, and informed dynamic schedule.

LBP schedule [32] operating in an iterative manner processes the nodes more sequentially within each iteration and activates only one or a specific subset of nodes at a time. It only needs one structure of node processing units (NPUs) when implemented based on FPGA. However, the flood schedule requires two types of NPUs, namely variable node processing units (VNPUs) and check node processing units (CNPUs). LBP has the advantage that the information obtained during an iteration is available to aid the remainder of the iteration. It tends to converge to the correct codeword using fewer iterations [8], [33], [34], resulting in a higher decoding throughput. Furthermore, the LBP schedule only needs to store the message passed by the check node to the variable node and the storage of the message passed

by the variable node to the check node are not required. This reduce half of the required storage space for ultra-long code length. The parallelism of the LBP schedule is sufficient in partial parallel schemes, although it does not have the same high-level of parallelism as the flooding schedule. Considering the above factors, the LBP schedule is adopted in our proposed scheme.

### D. SIDE INFORMATION (SI) DECODING

According to the Slepian-Wolf theorem [35], when the compressed outputs of two correlated sources are jointly decompressed at a decoder, the lossless compression that does not communicate their outputs to each other can be as efficient as if they communicated their outputs. In [26], Liveris *et al*. showed the side information (SI) decoding scheme in which LDPC codes be used to compress the two correlated binary sources. In the SI scheme, the transmitter (Alice) and the receiver (Bob) possess two sets of correlated binary random sequences $\boldsymbol{X}$ and $\boldsymbol{Y}$. Bob first encodes the random number sequence $\boldsymbol{Y}$ and generates a syndrome. Then he sends the syndrome $\boldsymbol{S}$ to Alice. Alice uses her original random number sequence $\boldsymbol{X}$ as side information and decodes the received syndrome to recover Bob's random number sequence. The efficient scheme can be used to information reconciliation of QKD.

### E. SI-LSP ALGORITHM

In summary, the SI-LBP algorithm is employed to implement the FPGA-based LDPC decoder suitable for information reconciliation of CV QKD. Its main steps are summarized as follows:

*Step* (*a*) *Initialization*: Each variable node is assigned an a posteriori LLR:

$$LLR_i^0 = \ln \frac{P_i(0)}{P_i(1)}. \tag{3}$$

*Step* (*b*) *Nodes Update*: For each row, the node processing formula is:

$$M_{ji} = LLR_i^{l-1} - E_{ji}^{l-1} \tag{4}$$

$$E_{ji}^{l} = (1 - s_j) \times \prod_{i \in N(j)/i'} sgn(M_{ji}) \times \Psi\left[\Psi(M_{ji}) - \sum_{i \in N(j)/i'} \Psi(M_{ji})\right] \tag{5}$$

$$LLR_i^l = M_{ji} + E_{ji}^l \tag{6}$$

where $l$ is the number of iterations, and the $\Psi$ function is defined as:

$$\Psi(x) = -\log\left[\tanh\frac{|x|}{2}\right], \tag{7}$$

*Step* (*c*) *Decision*: Quantize $\boldsymbol{X} = [x_1, x_2, \cdots, x_n]$ such that $x_n = 1$ if $LLR(q_i) \geq 0$, otherwise $x_n = 0$. If $\boldsymbol{H}\boldsymbol{X}^T = \boldsymbol{S}$ halt, with $\boldsymbol{X}$ as the decoder output results; otherwise go to Step (b).

## III. FPGA ARCHITECTURE

In this section, we detail the proposed FPGA-based LDPC decoder architecture on a Xilinx FPGA device based on the SI-LBP algorithm. More specifically, the architecture represents a framework for a decoder capable of decoding any QC-LDPC codes. Therefore, the discussion in this section will be presented in a generalized form, where the expansion factor $q$ and the degree of parallelism $p$ are variable parameters.

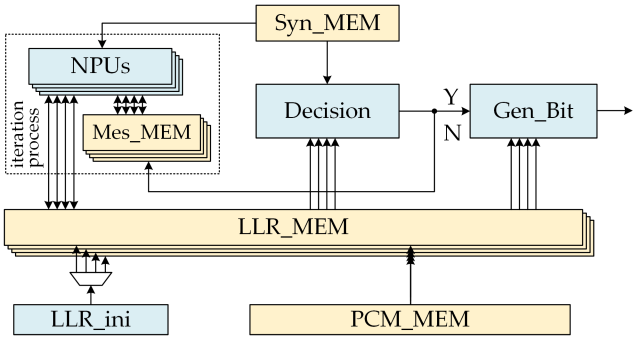



**FIGURE 3.** Logic structure of the proposed FPGA-based LDPC decoder.

Fig. 3 shows the overall architecture of the proposed FPGA-based LDPC decoder. The partially-parallel and pipeline architectures will be describe in Section III-A. The decoder consists of four types of memories: *PCM_MEM*, *LLR_MEM*, *Mes_MEM*, and *Syn_MEM*. Data storage and read&write will be discussed in Section III-C. The calculation of the decoder is divided into four steps: *LLR_ini*, *NPUs*, *Decision*, and *Gen_Bit*, which will be presented in Sections III-D, III-E, and III-G, respectively. In this section, we also discuss some other contents about the proposed architecture, such as fixed-point implementation, test platform based on C language, and decoding throughput.

### A. PARTIALLY-PARALLEL AND PIPELINE ARCHITECTURES

Serial decoders are implemented in hardware by only a single NPU, thereby require a small number of hardware resources. The NPU must be used multiple times per decoding iteration in a time-multiplexed manner, where internal memories are utilized to temporarily store the extrinsic LLRs for each row and column calculated by the NPU over the course of the iterative decoding process. Accordingly, serial decoders can naturally offer full run-time flexibility simply by changing the stored memory address values. However, due to large number of operations required for each decoding iteration, serial decoders suffer from very low decoding throughput, which typically does not meet the requirements of modern communication standards [36]. In the fully parallel strategy, the entire factor graph is implemented in hardware and all variable nodes and check nodes are updated concurrently. Fully parallel decoders are usually implemented to achieve high-throughput decoding of a certain LDPC code at the cost of high area consumption [15].

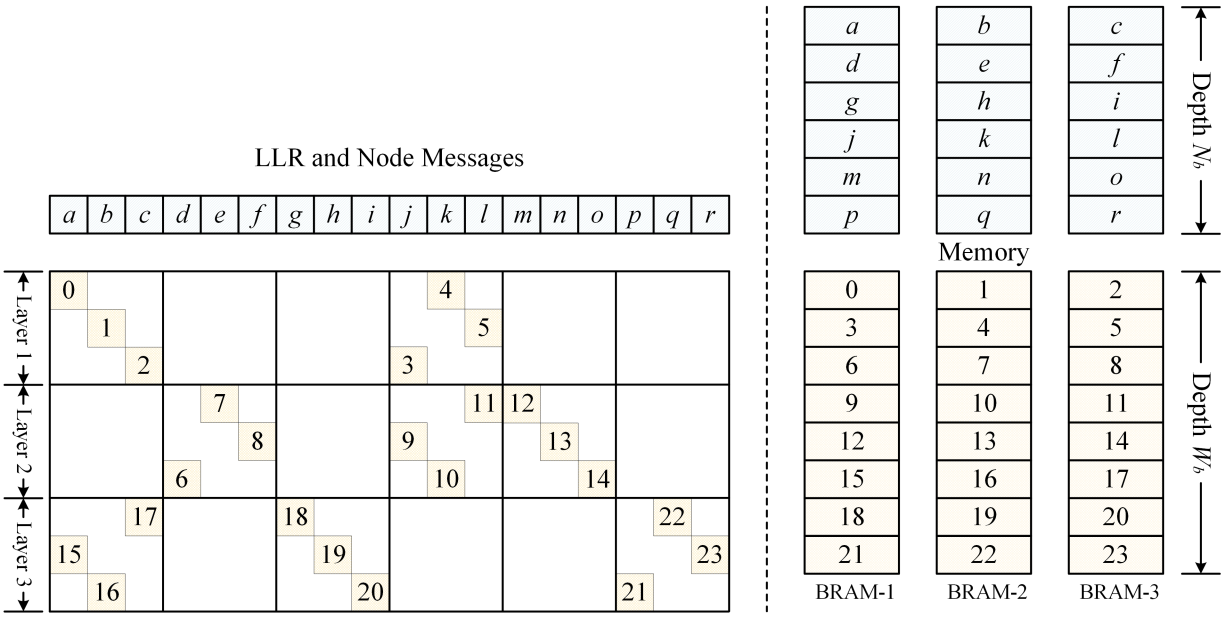



**FIGURE 4.** The memory structure.

Partially-parallel architectures strike a compromise between the serial and fully-parallel architectures by implementing $p$ parallel NPUs. Each decoding iteration is then split into several stages, wherein $p$ NPUs are processed simultaneously. This facilitates higher processing throughputs than those with serial architectures while avoiding the excessive hardware resource requirements of fully-parallel architectures. The increased degree of parallelism means that the distribution of values into BRAMs must be chosen carefully. Several works [11], [18]–[20] have addressed partially parallel decoding architectures for LDPC decoders to obtain a good trade-off between hardware complexity and decoding speed.

The pipeline structure can minimize the number of clock cycles required in the overall decoding process. To reduce the number of clock cycles per processing step, the pipeline structure needs to be applied in both the calculation process and memory access. Our proposed pipeline scheme will be discussed in detail in Sections III-E. In our scheme, partially parallel and pipeline structures are used to improve throughput. To implement a good trade-off between hardware complexity and throughput, the parallel parameter $p$ is set equal to the quasi-cyclic expansion factor $q$, that is, $p = q$.

### B. FIXED-POINT IMPLEMENTATION

The fixed-point number scheme is a method of representing numbers in binary format in hardware. In the scheme, a number can be represented by $(1, I, F)$, where 1 bit for the sign, $I$ bits represent the integer part, and $F$ bits represent the fraction part. Compared to single or double floating-point numbers, fixed-point numbers need fewer bits to represent a number. The word width $w = 1 + I + F$ of messages directly reduces the memory size. The larger the fixed-point number word width, the more accurate the number represented, and the better the decoding performance. However, larger bit width means that more FPGA's on-chip storage resources will be taken up. To find the best trade-off between the storage resources consumption and the decoding performance, accurate simulations were performed using a test platform based on C language. We simulated different types of fixed-point schemes and find (1, 5, 13) is the best choice. We choose the same fixed-point number format for both the LLR and node messages to effectively reduce the implementation complexity.

### C. THE MEMORY STRUCTURE AND DATAPATH

As shown in Fig. 3, the FPGA-based partially-parallel decoder consists of four types of memories: *PCM_MEM*, *LLR_MEM*, *Mes_MEM*, and *Syn_MEM*. They all use the FPGA's on-chip Block RAMs (BRAMs) to store temporary data needed in the decoding algorithm, and facilitate the data read&write through changing addresses only.

The *PCM_MEM* is used to store the PCM and only need to instantiate a BRAM. In order to save the storage space, only the column number of non-zero elements in the base matrix and the corresponding quasi-cyclic expansion factor are stored.

The *LLR_MEM* and *Mes_MEM* are used to store the initial LLRs and the node messages from variable nodes and check nodes. The overall memory of an LDPC decoder is predominantly determined by the size of those memories storing the LLRs and node messages. In order to be able to read out all the data required for parallel processing at the same time, both types of memory need to instantiate $p$ BRAMs to store them separately. Fig. 4 presents the concrete method that LLR and node messages stored. The left side represents the positions of

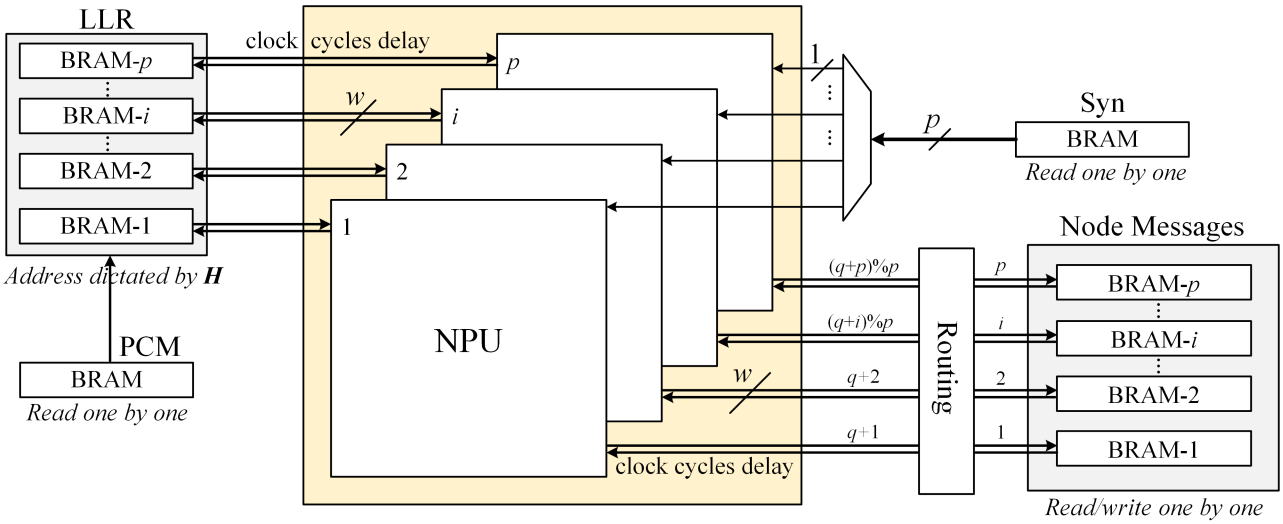



**FIGURE 5.** Datapath.

the data corresponding to Fig. 2, and we have numbered them. The right side shows where the numbered data are stored in BRAMs.

The *Syn_MEM* is used to store syndromes, and its bit width is consistent with the degree of parallelism. Due to the syndrome are required in both the iterative and decision process, the BRAM must be configured as a true dual port RAM. The data in each storage space corresponds to the syndrome required by $p$ layers, so it only needs to be read one by one.

Fig. 5 shows the data read and write structure in the iteration process. For QC-LDPC codes, the address generator of PCM BRAM can be realized with a simple counter, which simplifies the hardware design. Parallel BRAMs are controlled simultaneously by an address lookup table. The reading and writing of LLRs resorts to the addresses of the positions of the non-zero elements in the PCM $\boldsymbol{H}$. The messages in *Mes_MEM* are read one by one and then rearranged according to the quasi-cyclic expansion factor, finally, they enter the NPUs for message processing. After the updating of the messages in NPUs, they are rearranged in the original order and written into the original address space. Messages updating in NPUs require a constant clock cycle delay in the pipeline structure, thereby avoiding read and write conflicts.

### D. INITIALIZATION OF LLRs

As shown in (3), considering the small amount of calculation in the initialization of LLRs step, we use the existing floating-point number IP cores to reduce implementation complexity. As shown in Fig. 6, the scheme just needs five IP cores, including a floating-point number divider, a logarithmic operator, 2 convertors from 32-bits floating-point to fixed-point numbers, and a convertor from floating-point to fixed-point numbers. The proposed scheme has high calculation accuracy and can meet the throughput requirements. Besides, the amount of calculation in this step is relatively

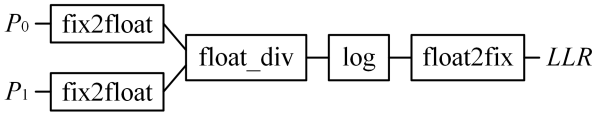



**FIGURE 6.** Logic structure of the initialization of LLRs. *float_div* is a floating-point number divider IP core, and *log* is a logarithmic operator IP core. The module *fix2float* is an IP core that convert the fixed-point number to 32-bits float-point numbers, and the module *float2fix* is the inverse transformation of *fix2float*.

small, and it is sufficient to use pipeline and serial structures. After the LLR is generated, it needs to be written into the corresponding BRAMs.

### E. NODE PROCESSING UNITS (NPUs)

After obtaining LLRs in the initialization step, the iterative decoding step starts. The step requires multiple iterations, massive data storage, and has high complexity. To improve throughput, it is necessary to make full use of the parallel operations characteristics of FPGA, but this will raise another problem: the higher the degree of parallelism, the more data will be read, written and buffered during the iterative process step. We adopts two ways to solve this issue: ($a$) using the fixed-point number representation; ($b$) using the LBP schedule.

The calculations that NPUs perform are shown in (4)-(6). The timing sequence chart is plotted in Fig. 7. The pipeline structure is embodied in four sub-processes and the iteration process. There is no clock delay between layers or nodes. In each iteration, all rows of each block are updated serially in $N_{bm}$ clock cycles. Processing of a node message can be divided into five stages: ($a$) NPUs execute (4) after reading *LLR_read* and *EV2C_read* from multiple BRAMs simultaneously to obtain *MC2V_tep*; ($b$) the *MC2V_tep* is reordered into *MC2V* according to the offset of the cyclic permutation matrix; ($c$) NPUs execute (5). In this stage, NPUs need to calculate the $\Psi(x)$ function and accumulation, etc.,

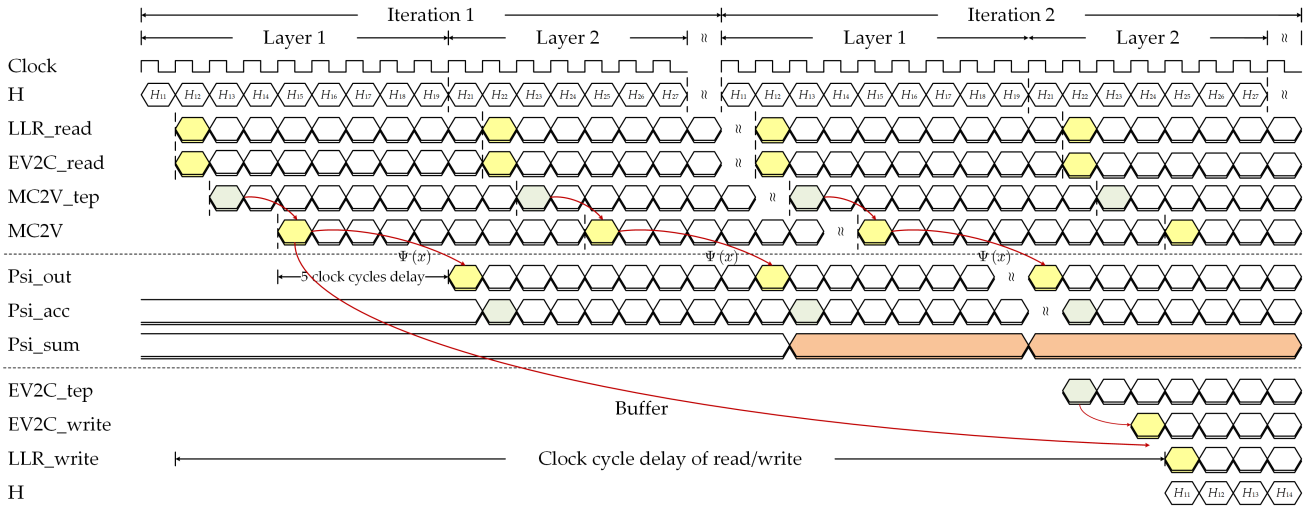



**FIGURE 7.** The timing sequence chart of the iteration procedure.

and finally obtain *EV2C_tep*; (*d*) the *EV2C_tep* is reordered into *EV2C_write*, which is written to original BRAMs; (*e*) NPUs execute (6) to obtain *LLR_write*, which is written to original BRAMs. In these process, *LLR* and *EV2C* is read from their respective BRAMs and rewritten after node messages are updated. The clock delay depends on the $\Psi(x)$ function, accumulation operation, and other inherent delays. To avoid memory access conflicts, it is necessary to add a new constraint when constructing PCMs. The application of pipeline structure makes the method is also applicable to the processing of irregular LDPC codes.

### F. THE APPROXIMATION OF FUNCTION $\Psi(x)$

During the node update process, the extrinsic soft messages are sent to hardware units, which perform the nonlinear function $\Psi(x)$ defined as (7). The calculation of the function based on FPGAs is a computationally complex task and will consume a lot of hardware resources. Notice that $\Psi(x)$ is an even symmetric function and its outputs are positive numbers.

In order to reduce the implementation complexity and remain the calculation accuracy as much as possible, we present a new non-uniform piecewise approximation scheme [11] using the second-order function $y = ax^2 + bx + c$, which can achieve almost identical decoding performance as the standard SPA. To this end, determination of parameters $a$, $b$, and $c$ for each segmentation are critical. Larger number of segments results in better decoding performance, but more LUTs will be consumed. We simulated different types of schemes to find the best trade-off between the resource consumption and decoding performance. Fig. 8 shows the comparison of an example between $\Psi(x)$ function and non-uniform piecewise approximation with 4 segments. Notice that the difference between the value of the original function and its approximation is negligible.

Fig. 9 shows the logic structure of the function $\Psi(x)$ implemented by using on-chip DSP slices with 25 × 18 multiplier and adder/subtracter. The calculation process only lasts for 5 clock cycles by using the specific calculation sequence, as shown in Fig. 7. In our scheme, the fixed-point numbers for the input and output are both in (1,5,13) format. During the calculation process, it is necessary to perform interception and supplement on fixed-point numbers.

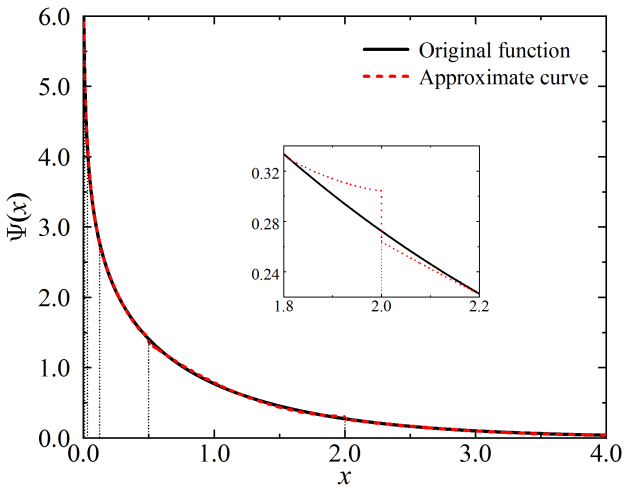



**FIGURE 8.** Comparison between the function $\Psi(x)$ and non-uniform piecewise approximation of second-order function.

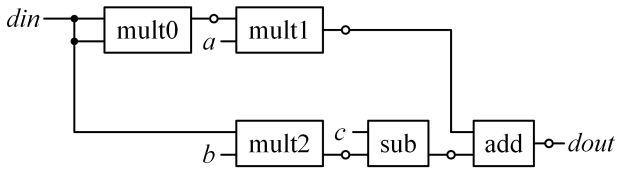



**FIGURE 9.** Logic structure of $\Psi(x)$. *mult*0, *mult*1, and *mult*2 are multipliers, *add* is an adder, and *sub* is a subtractor. The five IP cores are all built using DSP slices to perform fixed-point arithmetic. The circle in the figure indicates the adjustment of the bit width.

### G. DECISION AND BIT SEQUENCE GENERATION

The module *Decision* in Fig. 3 is used to determine whether the decoding succeeds after each iteration. The specific implementation method is to read the sign bits of the LLRs and then perform logical NOT and XOR operation with the corresponding elements of PCM successively. The obtained bit sequence is compared with the syndrome. If they are completely consistent, the decoding is judged to be successful, otherwise the decoding fails and return to NPUs. In order to reduce the unessential calculations, the comparison is performed at the same time as the calculation process. Once the inconsistency is found, the decision step is terminated immediately. Besides, the module *Decision* is also judged whether the predefined maximum number of iterations has been reached. If it is reached, the decoding determined to be failed.

The module *Gen_Bit* in Fig. 3 is used to generate the bit sequence $\boldsymbol{X}$ when the decoding is successful. The module reads the LLRs from *LLR_MEM* in parallel. If a sign bit of LLR is 0, we have $x_i = 1$, otherwise $x_i = 0$.

### H. TEST PLATFORM BASED ON C LANGUAGE

To facilitate the decoder design, a simulation test platform was built based on C language, in which various calculation units completely simulating the FPGA hardware environment are established. There calculation units are included in the test platform: (*a*) decoding performance of PCMs; (*b*) the optimal fixed point number format; (*c*) the number of segments and the accuracy of the coefficients for non-uniform piecewise approximation of function $\Psi(x)$.

## IV. IMPLEMENTATION RESULTS

In this section, we will present the performance of our FPGA-based LDPC decoder following the above architecture. The performance of a decoder can be characterized by four important parameters: bit error rate (BER), the average number of iterations, throughput, and the consumption of hardware resources. For the FPGA-based ultra-long LDPC decoder, the key factor restricting the performance is the on-chip storage resource on an FPGA. To evaluate our proposed scheme, the FPGA-based LDPC decoder is implemented on a Xilinx VC709 evaluation board, which is populated with a Virtex-7 XC7VX690T FPGA with 433,200 LUTs, 866,400 FFs, 3,600 DSP slices, and 52,920 Kb BRAMs.

### A. THROUGHPUT

The high throughput is the key advantage of LDPC decoders implemented by an FPGA device. Its decoding throughput can be calculated by

$$T = \frac{f \cdot L}{N_{bm} \cdot N_{iter} + D}, \tag{8}$$

where $f$ is the clock frequency of FPGAs, $L$ is code lengths, $N_{bm}$ is the number of nodes in a base matrix, $N_{iter}$ is the average number of decoding iterations required to achieve a correct decoding processing, and $D$ is the clock delay of node processing.

In order to more clearly reflect the effect of parallel parameters on throughput, we can change (8) as

$$T \approx \frac{f \cdot q}{(1-R) \cdot N_{ave} \cdot N_{iter}}, \tag{9}$$

where $R$ is the code rate of the LDPC codes, $N_{ave}$ is the average number of nodes in each row of a basic matrix. To derive (9), we have neglected $D$ because it is much smaller than $N_{bm} \cdot N_{iter}$.

From (9), we notice that $R$ and $N_{ave}$ are the intrinsic properties of a PCM, the throughput is attributed to three factors: (*a*) high clock frequency, (*b*) large parallelism parameter $q$ in partially parallel decoders, and (*c*) low average number of iterations $N_{iter}$. The key advantage of FPGA is that it can improve the parallelism parameter of decoders, however, the average number of iterations cannot be reduced by FPGA. Notice that the decoding throughput is not related to the code length.

### B. DECODING PERFORMANCE

In the following, we investigate to verify the performance of the two PCMs with the sizes of $149,504 \times 262,144$ and $309,760 \times 349,952$ using the SI-LBP algorithm where additive white Gaussian noise (AWGN) channel and binary phase-shift keying (BPSK) modulation are assumed. Two different PCMs with code rates of 0.430 and 0.115 have been used in the information reconciliation of CV QKD systems at different SNRs [25]. In Table 1, we show node degree distributions of the two irregular PCMs obtained by the discretized density evolution for SPA algorithm. Using the good degree distributions, high performance PCMs can be constructed.

**TABLE 1. Optimal variable degree distributions for two different code rates.**

| $L$ | 262,144 | | 349,952 | |
|---|---|---|---|---|
| Code rate | 0.430 | | 0.115 | |
| | 2 | 0.176600 | 2 | 0.397963 |
| | 3 | 0.236128 | 3 | 0.262680 |
| | 6 | 0.084050 | 7 | 0.176724 |
| | 10 | 0.207383 | 15 | 0.060988 |
| | 33 | 0.099039 | 20 | 0.101645 |
| | 50 | 0.196801 | | |

Fig. 10 plots the BERs versus the SNRs for the SPA and the offset MSA. From BERs and the minimum SNR for successfully decoding, it can be seen that decoding performance achieved by the SPA is better than the offset MSA. We also compared the performance of SPA in two different cases: (1, 5, 13) fixed-point number and floating-point number. The difference of BERs between the two formats is extremely small (less than $10^{-4}$) under the same number of iterations. Furthermore, Fig. 11 depicted the relationship between SNRs and the average number of iterations. Notice that the average

**TABLE 2.** Performance of the proposed decoder and its comparison with previous work.

| | Wang, *et al.* [11] | Lu, *et al.* [20] | Liu, *et al.* [16] | This work | |
|---|---|---|---|---|---|
| **FPGA device** | Virtex-II 6000 | Kintex-7 XC7K325T | Kintex-7 | Virtex-7 XC7VX690T | |
| **Algorithm** | SPA | RCM-LBP | Modified MSA | SPA | |
| **Schedule** | Flooding | Layered | Layered | Layered | |
| **Code rate** | 0.875 | Compatible | 0.4129 | 0.430 | 0.115 |
| **PCM $H$** | $1,022 \times 8,176$ | $5,544 \times 5,544$ | $93 \times 155$ | $149,504 \times 262,144$ | $309,760 \times 349,952$ |
| **Base matrix $H_b$** | $2 \times 16$ | $693 \times 693$ | $3 \times 6$ | $2,335 \times 4,096$ | $4,840 \times 5,468$ |
| **Number of nodes in $H$** | 32,704 | 1732 | 465 | 1,255,488 | 1,090,624 |
| **Parallel parameter** | 32 VNPUs<br>4 CNPUs | 11 | 31 | 64 | 64 |
| **Width of fixed-point numbers (bits)** | 6 | 6 | 4 | 19 | 19 |
| **LUTs** | 28,229 | 79,961 | 14,00 | 51,451 (11.9%) | 51,451 (11.9%) |
| **Flip-flops** | 26,926 | 89,564 | 10,400 | 35,523 (4.1%) | 35,523 (4.1%) |
| **DSP slices** | — | 264 | — | 1,024 (28.4%) | 1,024 (28.4%) |
| **BRAMs (Kb)** | 2,304 | 1,080 | — | 31,680 (59.9%) | 30,636 (57.9%) |
| **Clock frequency (MHz)** | 193.4 | 218 | 700 | 100 | 100 |
| **Throughput (Mb/s)** [a] | 172 | 1100 | — | 108.64 | 70.32 |
| **Max. initial BER** | — | — | — | 32.76% | 16.05% |
| **Min. SNR (dB)** | 3.2 | 5.0 | 1.0 | 0.6 | -0.6 |

[a] The number of iterations $N_{iter}$ used to calculate throughput is corresponding to $SNR = 1.0$ dB.

number of iterations decreases with increasing SNR. The simulation results show that fixed-point number schemes hardly affect the average number of iterations. Therefore, the effect of fixed-point number schemes on the throughput can be ignored, as shown in (8) and (9).

Table 2 presents the detailed parameters of our FPGA-based LDPC decoder. The decoder occupies about 11.9% of the LUTs, 4.1% of the FFs, 28.4% of the DSP slices, and 59.9% of the BRAMs available on the FPGA device. When implementing the decoder based on FPGA, we use fixed-point numbers format to implement message processing to reduce the decoding complexity and the consumption of storage resources. In our work, the consumption of LUTs and FFs shown in Table 2 corresponds to the case where the parallel parameter is 64 and the fixed-point numbers format is $(1, 5, 13)$. The decoder occupies 31,680 Kb BRAMs that are used to *PCM_MEM*, *LLR_MEM*, *Mes_MEM*, and *Syn_MEM*, of which *Mes_MEM* takes up the largest storage space. In general, the larger the width of fixed-point numbers, the more storage resources must be consumed. But, the storage resources will remain constant with the change of the parallel parameter. DSP slices are used to implement the second-order function in the $\Psi(x)$ function approximation for 64 NPUs, and for each NPU, 16 DSPs are required. Larger parallel parameters will consume more DSP slices. Note that the same consumptions of LUTs, FFs, and DSP slices are observed for the two different PCMs with the same parallel parameter. We provided the minimum BRAMs required for two different PCMs in Table 2. To improve the flexibility of the decoder, larger BRAMs can be used to adapt to different PCMs. The achieved results based on FPGAs show that the minimum SNRs for successfully decoding of the two PCMs can reach 0.6 dB and $-0.6$ dB, respectively. The decoding performance can reach to such low SNRs is due to the ultra-long code length and low code rate. The throughputs presented in Table 1 are evaluated using (9) with the number of iterations $N_{iter}$ corresponding to SNR $= 1.0$ dB. In a word, the proposed decoder can well meet the needs of the CV QKD system in terms of decoding capability, hardware resource consumption, and throughput.

In Table 2, we compare our decoder with three previous works [11], [16], [20]. Obviously, the advantage of our decoder is that it has the minimum SNR that can be successfully decoded. The bit width of the fixed-point numbers is obviously larger than other works. The reason is that the code length is much longer, which leads to a larger number of calculations and higher requirement for calculation accuracy in our work. Compared with the work of Wang and Cui [11], the decoding algorithms of the two works are similar. More LUTs and FFs are consumed in our scheme because the parallel parameter and the bit width of fixed-point numbers are larger. In general, the larger parallel parameter or the bit width of the fixed-point numbers, the more LUTs and FFs must be consumed. In the work of Lu *et al.* [20], a higher throughput is achieved using the RCM-LBP decoding algorithm. Note that this work employs more LUTs and FFs when the parallel parameter is 11, the PCM is much smaller than ours and the clock frequency is higher. Of course, it is very meaningful

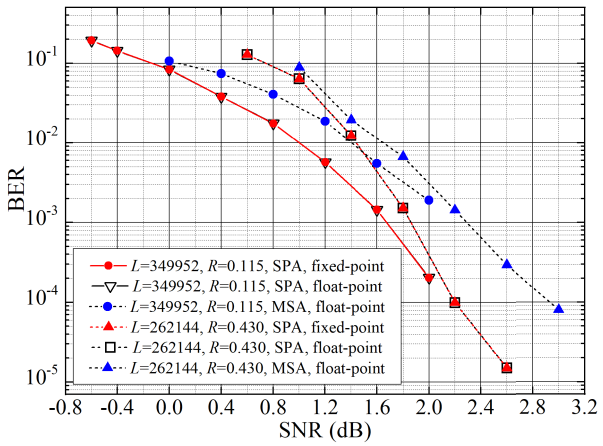



**FIGURE 10.** Comparison of the BER between the offset MSA and the SPA. Two PCMs with size of 149,504 × 262,144 and 309,760 × 349,952 are investigated, the corresponding number of iterations are 5 and 15, respectively.

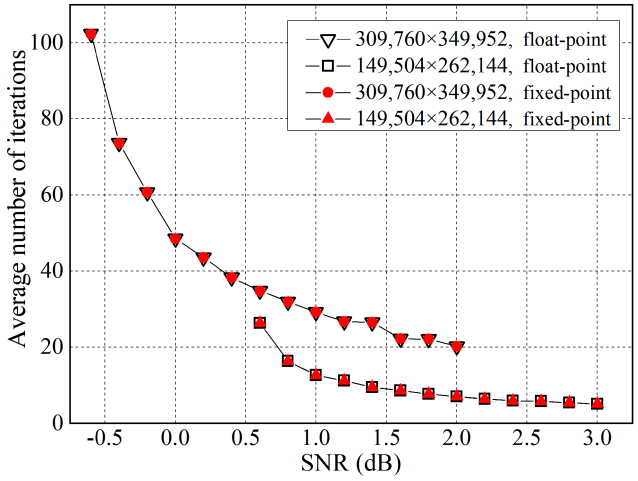



**FIGURE 11.** The simulation results of average number of iterations for LDPC codes versus the SNR.

that the decoder can realize rate adaptive communications, and we also consider to achieve the feature in the future. Very recently, Liu *et al*. [16] implemented the LDPC decoder using MSA in an FPGA. It can be seen that the decoding performance is lower than ours. Compared with the previous works, the decoding throughput of this work is not high enough, but it has been able to meet the needs of the state of the art in information reconciliation of CV QKD. The decoding throughput depends on several parameters such as parallelism parameter, code length, the number of iterations, and clock frequency. The ultra-long code length is the main reason for our low throughput, because it increases greatly the number of nodes that need to be processed is larger than the previous works. In the future, we can improve the decoding throughput further by increasing the clock frequency.

## V. CONCLUSION

In this paper, we have designed and implemented an FPGA-based LDPC decoder using the SI-LBP algorithm that can achieve a better trade-off between the decoding performance and implementation complexity. We developed partially parallel decoder architectures and optimized the pipeline structure to increase the decoding throughput by reducing the clock cycles of the LDPC decoder. A uniform quantization scheme is used to save the consumption of storage resources and reduce implementation complexity. Furthermore, a non-uniform piecewise approximation scheme using the second-order function for the function $\Psi(x)$ is adopted to reduce the implementation complexity greatly. We have demonstrated the advantages of the proposed LDPC decoder architecture with an FPGA implementation. The implementation results on the Xilinx VC709 evaluation board shown that the proposed LDPC decoder with ultra-long code length has good decoding performance and throughput at SNRs as low as -0.6 dB. When SNR = 1.0 dB, the throughputs reach 108.64 Mb/s and 70.32 Mb/s at code lengths of 262,144 and 349,952, respectively. The decoder with superior performance can be readily applied to the information reconciliation in CV QKD, and can also find potential application in the other communication domain.

## REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.
[2] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A survey of FPGA-based LDPC decoders," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1098–1122, 2nd Quart., 2016.
[3] P. Hailes, L. Xu, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "A flexible FPGA-based quasi-cyclic LDPC decoder," *IEEE Access*, vol. 5, pp. 20965–20984, Jul. 2017.
[4] J. Andrade, G. Falcao, V. Silva, and L. Sousa, "A survey on programmable LDPC decoders," *IEEE Access*, vol. 4, pp. 6704–6718, Jul. 2016.
[5] S. Keskin and T. Kocak, "GPU-based gigabit LDPC decoder," *IEEE Commun. Lett.*, vol. 21, no. 8, pp. 1703–1706, Aug. 2017.
[6] J. Yuan and J. Sha, "4.7-Gb/s LDPC decoder on GPU," *IEEE Commun. Lett.*, vol. 22, no. 3, pp. 478–481, Mar. 2018.
[7] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 985–994, Aug. 2009.
[8] M. Zhao, X. Zhang, L. Zhao, and C. Lee, "Design of a high-throughput QC-LDPC decoder with TDMP scheduling," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 1, pp. 56–60, Jan. 2015.
[9] Y. M. Lin, H.-T. Li, M.-H. Chung, and A.-Y. Wu, "Byte-reconfigurable LDPC codec design with application to high-performance ECC of NAND flash memory systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 7, pp. 1794–1804, Jul. 2015.
[10] H.-C. Lee, M.-R. Li, J.-K. Hu, P.-C. Chou, and Y.-L. Ueng, "Optimization techniques for the efficient implementation of high-rate layered QC-LDPC decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 2, pp. 457–470, Feb. 2017.
[11] Z. Wang and Z. Cui, "Low-complexity high-speed decoder design for quasi-cyclic LDPC codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 1, pp. 104–114, Jan. 2007.
[12] C. Beuschel and H.-J. Pfleiderer, "FPGA implementation of a flexible decoder for long LDPC codes," in *Proc. Int. Conf. Field Program. Logic Appl.*, Heidelberg, Germany, Sep. 2008, pp. 185–190.
[13] A. J. Wong, S. Hemati, and W. J. Gross, "A modular architecture for structured long block-length LDPC decoders," *J. Signal Process. Syst.*, vol. 90, no. 1, pp. 29–38, Jan. 2018.
[14] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable LDPC decoder architectures for regular and irregular codes," *J. Signal Process. Syst.*, vol. 53, nos. 1–2, pp. 73–88, Nov. 2008.
[15] S. Sharifi Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, Nov. 2008.

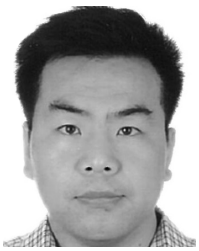
[16] Y. Liu, W. Tang, and D. G. M. Mitchell, "Efficient implementation of a threshold modified min-sum algorithm for LDPC decoders," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 9, pp. 1599–1603, Sep. 2020.
[17] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga, and S. Goto, "Partially-parallel LDPC decoder based on high-efficiency message-passing algorithm," in *Proc. Int. Conf. Comput. Design*, San Jose, CA, USA, 2005, pp. 503–510.
[18] F. Verdier and D. Declercq, "A low-cost parallel scalable FPGA architecture for regular and irregular LDPC decoding," *IEEE Trans. Commun.*, vol. 54, no. 7, pp. 1215–1223, Jul. 2006.
[19] Y. Dai, N. Chen, and Z. Yan, "Memory efficient decoder architectures for quasi-cyclic LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 9, pp. 2898–2911, Oct. 2008.
[20] F. Lu, Y. Dong, and C. W. Chen, "Layered decoding algorithm and two-level quasi-cyclic matrix construction for rate compatible modulation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 8, pp. 3213–3226, Aug. 2019.
[21] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory system optimization for FPGA-based implementation of quasi-cyclic LDPC codes decoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 1, pp. 98–111, Jan. 2011.
[22] V. A. Chandrasetty and S. M. Aziz, "Analysis of performance and implementation complexity of simplified algorithms for decoding low-density parity-check codes," in *Proc. IEEE Globecom Workshops*, Miami, FL, USA, Dec. 2010, pp. 430–435.
[23] H. Lopez, H.-W. Chan, K.-L. Chiu, P.-Y. Tsai, and S.-J.-J. Jou, "A 75-Gb/s/mm$^2$ and energy-efficient LDPC decoder based on a reduced complexity second minimum approximation min-sum algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 926–939, Apr. 2020.
[24] E. Diamanti, H.-K. Lo, B. Qi, and Z. Yuan, "Practical challenges in quantum key distribution," *npj Quantum Inf.*, vol. 2, no. 1, Nov. 2016, Art. no. 16025.
[25] Z. Bai, S. Yang, and Y. Li, "High-efficiency reconciliation for continuous variable quantum key distribution," *Jpn. J. Appl. Phys.*, vol. 56, no. 4, Apr. 2017, Art. no. 044401.
[26] A. D. Liveris, Z. Xiong, and C. N. Georghiades, "Compression of binary sources with side information at the decoder using LDPC codes," *IEEE Commun. Lett.*, vol. 6, no. 10, pp. 440–442, Oct. 2002.
[27] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.
[28] V. A. Chandrasetty and S. M. Aziz, "FPGA implementation of a LDPC decoder using a reduced complexity message passing algorithm," *J. Netw.*, vol. 6, no. 1, pp. 36–45, Jan. 2011.
[29] 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. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.
[30] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.
[31] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.
[32] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," in *Proc. IEEE Workshop onSignal Process. Syst. (SIPS)*, Austin, TX, USA, 2004, pp. 107–112.
[33] H. Ding, S. Yang, W. Luo, and M. Dong, "Design and implementation for high speed LDPC decoder with layered decoding," in *Proc. WRI Int. Conf. Commun. Mobile Comput.*, Leipzig, Germany, Jan. 2009, pp. 156–160.
[34] O. Boncalo, G. Kolumban-Antal, A. Amaricai, V. Savin, and D. Declercq, "Layered LDPC decoders with efficient memory access scheduling and mapping and built-in support for pipeline hazards mitigation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1643–1656, Apr. 2019.
[35] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inf. Theory*, vol. 19, no. 4, pp. 471–480, Jul. 1973.
[36] S. Bates, Z. Chen, L. Gunthorpe, A. E. Pusane, K. S. Zigangirov, and D. J. Costello, "A low-cost serial decoder architecture for low-density parity-check convolutional codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 7, pp. 1967–1976, Aug. 2008.

**SHEN-SHEN YANG** received the B.S. and Ph.D. degrees from Shanxi University, Taiyuan, China, in 2014 and 2020, respectively. He is currently working with Shanxi Normal University. His research interests include field-programmable gate arrays, error correction coding, and post-processing procedure of continuous-variable quantum key distribution.

**JIAN-QIANG LIU** is currently pursuing the Ph.D. degree in optics with Shanxi University, Taiyuan, China. His research interests include field programmable gate arrays and optical design and control system of continuous-variable quantum key distribution.

**ZHEN-GUO LU** is currently pursuing the Ph.D. degree in optics with Shanxi University, Taiyuan, China. His research interests include field programmable gate arrays and quantum random number generation.

**ZENG-LIANG BAI** received the Ph.D. degree in optics from Shanxi University, Taiyuan, China, in 2017. He has been with Shanxi University of Finance and Economics, since 2017. His research interests include error correction coding and post-processing procedure of continuous-variable quantum key distribution.



**XU-YANG WANG** received the Ph.D. degree in optics from Shanxi University, Taiyuan, China, in 2013. He is currently an Associate Professor with Shanxi University. His research interests include integrated photonics and quantum communications.

**YONG-MIN LI** received the Ph.D. degree in optics from Shanxi University, Taiyuan, China, in 2003. Since 2003, he has been a Postdoctoral Fellow with the University of Tokyo, and a Visiting Fellow with Australian National University. He is currently a Professor with Shanxi University. His research interests include quantum communications and quantum optics.