IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A Universal, Low-Delay, SEC-DEC-TAEC Code for State Register Protection

MENG DONG[1], WEITAO PAN[1], ZHILIANG QIU[1], XIAOXIN QI[1], LING ZHENG[2], AND HUAN LIU[1]

[1]State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China
[2]School of Communication and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, China

Corresponding author: Weitao Pan (wtpan@mail.xidian.edu.cn)

This work was supported in part by the National Key Laboratory Foundation (HTKJ2019KL504012) and National Natural Science Foundation (61502204).

ABSTRACT Finite State Machine (FSM) is widely used in electronic systems and its reliability is critical to the system. Ionizing radiation induced soft error is one of the major concerns in the design of electronic systems, especially in avionics or space applications. Nowadays, the majority of electronic systems relies on single-error correction, double-error detection (SEC-DED) codes to mitigate soft errors. However, the presence of multiple bit upsets is becoming more prevalent as CMOS technology scales down. In addition, state registers in FSMs usually have variable bit-widths and have strict requirement on encoding and decoding delay, which poses challenges for error mitigation techniques. This paper presents an Error Detection and Correction (EDAC) code for state register protection, which can achieve single-error correction, double-error correction and triple-adjacent-error correction (SEC-DEC-TAEC) ability. The proposed code can be used to protect data with $4n$ bit-width ($n = 2, 3, 4, ...$) using one common encoder and decoder code block and introduces minimal delay. Experiment results show that the proposed code has better error correction ability than most existing MCU correction codes. Besides, it reduces area occupation by 30% and delay by 15% compared with Orthogonal Latin Square (OLS) code in the case of 8 bit-width data.

INDEX TERMS Finite State Machine, State Register, Error Correction Codes, Double Error Correction, Triple Adjacent Error Correction.

I. INTRODUCTION

In the design of electronic systems, reliability is one of the major concerns. One reliability issue is the ionizing radiation induced soft errors, which is common in avionics or space applications. These errors can cause single event upset, which could be harmful to the functionality of the electronic systems. Error detection and correction (EDAC) codes have been traditionally used to protect electronic systems from single event upset problems. Single-error correction, double-error detection (SEC-DED) codes are one of the most commonly used EDAC schemes [1]–[4]. These codes have few redundant check bits, simple logic, and fast encoding and decoding speed. However, as CMOS technology scales down, the number of transistors per unit area increases and the operation voltage of transistors decreases. Therefore, a single high-energy radiating particle or ionizing radiation will cause multiple errors, resulting in multiple cells unpsets (MCU) [5]–[9]. The traditional SEC-DED codes cannot correct this kind of error. Therefore, methods to detect and correct multiple errors are needed to protect electronic systems from data corruption.

With the development of programmable ASIC technology, the implementation of electronic systems using programmable ASICs is becoming prevalent. The application of programmable ASICs greatly reduces the development cost and cycle of electronic systems. In an electronic system using programmable ASICs, finite state machines (FSMs) are the brain of the system and are utilized to generate control signals and schedule other function units. State registers in FSMs are responsible for storing the current state. The protection of state registers is of great importance to the reliability of the system. However, the characteristics of state registers pose challenges to the design of error correction schemes. First, state registers usually have variable bit-width, which is determined by the number of states that need to be represented. Second, to ensure the correctness of the FSM operation, the encoding and decoding process should typically be completed within one clock cycle. Therefore, an

error correction scheme that supports variable bit-widths and low delay is urgent.

In this paper, we propose an EDAC code based on Hsiao code and parity check code to protect state registers from data corruption. The proposed code has the following features:

1) It achieves single-error correction, double-error correction and triple-adjacent-error correction (SEC-DEC-TAEC) ability.
2) It can support variable bit-widths using one common encoder and decoder code block. Specifically, it can protect data with bit-width of $4n$ ($n = 2, 3, 4, ...$).
3) It has minimal encoding and decoding delay, which is appealing to state register protection.
4) It can correct errors in both data bits and check bits.

Extensive experiments are conducted to verify the effectiveness of the proposed code. The results show that the error correction ability of the proposed code is greatly improved compared with most existing MCU correction codes. In the case of 8-bit data (which is the commonly used instruction length), compared with OLS code, the hardware area occupied by the proposed code is reduced by 30% and the calculation delay is reduced by 15%. In addition, compared with the commonly used triple-mode redundancy technology in satellite communication systems, the proposed scheme has more obvious advantages in area, delay and reliability.

The rest of this paper is organized as follows. In Section II, the existing proposals for MCU correction are reviewed. Besides, a brief introduction to Hsiao code and parity check code is presented. In Section III, the design of the proposed SEC-DEC-TAEC code is elaborated and its error correction ability is analysed. In Section IV, the proposed code is evaluated in terms of error correction ability and implementation cost, including delay, power consumption and area occupation. Comparation with existing codes are conducted. Finally, the conclusion is drawn and further work is outlined in Section V.

II. RELATED WORK

In this section, we first present related work on MCU correction schemes. Then, we give a brief introduction to parity check code and Hsiao code, based on which the proposed code is designed.

A. MCU CORRECTION SCHEMES

There are many proposals in the literature to solve the MCU problem in order to detect and correct multiple bit errors. These proposals can be divided into three categories. The first category aims to complete encoding and decoding without adding additional check bits [10]–[12], but the generation time of the check matrix is long. For example, it takes nearly a week to generate the check matrix in [11]. The second category stores data in the form of a two-dimensional matrix, called matrix code [13]–[18], and adds error correction codes to the data in row and column to achieve correction of the bit flip situation. To improve the encoding and decoding efficiency, the One-Step Majority of Logic Decoding (OS-MLD) technique is proposed, including Euclidean Geometry codes (EG) [19], Difference Set codes (DS) [20] and Orthogonal Latin Square codes (OLS) [21], among others. However, these methods can only support specific application scenarios. For example, OLS codes can only be applied to codewords with length of m^2 (m is a positive integer). Besides, for different bit-widths, the size of the corresponding check matrix is different. Therefore, if the data to be protected has various bit-widths in an electronic system, multiple encoder and decoder code blocks are needed, complicating the overall system. At present, researchers are working on applying OLS to 32-bit codeword [22] and reducing check bits [23], [24]. The last category is triple-mode redundancy, which is commonly used in satellite communications. There has been a lot of research on FPGA-based implementation of triple-mode redundancy [25]–[27].

For protecting data in registers where low delay is required, some codes have been proposed [28]–[30]. In addition, more complex and precise error correction codes [31]–[40] are used in electronic systems for critical applications.

B. PARITY CHECK CODE

Parity check code is the colletive term of odd parity check code and even parity check code, which is one of the most basic error detection codes. By adding a redundant bit, the codeword is ensured to have an odd or even number of 1s. The limitation of parity check code is that it can only detect odd number of errors, but because of its simplicity, parity check code is still widely used for error control in data transmission.

C. HSIAO CODE

Hsiao code is a kind of SEC-DED code with odd weight column. For data bits of the same length, the number of check bits of Hsiao code and Extended Hamming code are the same, but Hsiao code has better performance [2]. Hsiao code reconstructs the check matrix to ensure that the number of 1s in each column is odd, and the number of 1s in each row is equal or as equal as possible (the difference does not exceed one). To minimize the number of 1s in the check matrix, the check matrix of Hsiao code must meet the following conditions:

1) Each column in the check matrix contains an odd number of 1s.
2) The total number of 1s is minimum.
3) The number of 1s in each row is equal as much as possible and the maximum difference is one.
4) There is no column with all 0s.
5) There are no two identical columns.

The first condition ensures that the code spacing is 4. The second and third conditions enable Hasio codes to have a high coding efficiency which reduces overhead. They also enable Hasio codes to have uniform timing, which outperforms extended Hamming codes. The fifth condition ensures that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3178953, IEEE Access

the code spacing is at least 3, and that each corrector is different because the columns in the check matrix represent standard correctors for error correction.

III. THE PROPOSED EDAC CODE

In this section, the design of the proposed EDAC code is elaborated. The encoding process and decoding process are described and the error correction ability is analysed.

A. OVERALL ARCHITECTURE

Fig. 1 depicts the block diagram of the encoder. The encoding process consists of three steps. First, the original word is divided into half-byte segments which are then arranged by row. Then, check bits are generated, which consist of two parts. For each segment, row check bits are computed using Hsiao code; for each column of the segments, a column check bit is generated by the parity check code. Finally, the data and check bits are interleaved to generate the final codeword.

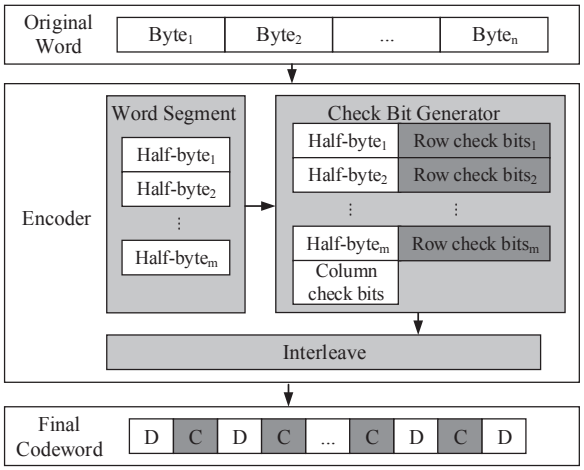


FIGURE 1. Block diagram of the encoder. D and C represent the data bit and check bit respectively. First, the original word is divided into half-byte segments. Then, for each segment, row check bits are computed using Hsiao code; for each column of the segments, a column check bit is generated by the parity check code. Finally, the data and check bits are interleaved to generate the final codeword.

Fig. 2 depicts the block diagram of the decoder. The codeword is first deinterleaved to recover the data bits and check bits. Then, errors in the data bits are detected and corrected based on the check bits. Finally, the data segments are concatenated to recover the original word.

B. THE ENCODING PROCESS

We now elaborate the first two steps of the encoding process, as shown in Algorithm 1. An 8-bit word is used as an example to illustrate the procedure. Firstly, the original word is divided into m half-byte segments. For example, the 8-bit word can be divided as shown in Fig. 3 (a). The half-byte segments are arranged by row to form the data matrix, denoted by $\boldsymbol{D}$. Then, for each row of $\boldsymbol{D}$, the row codeword

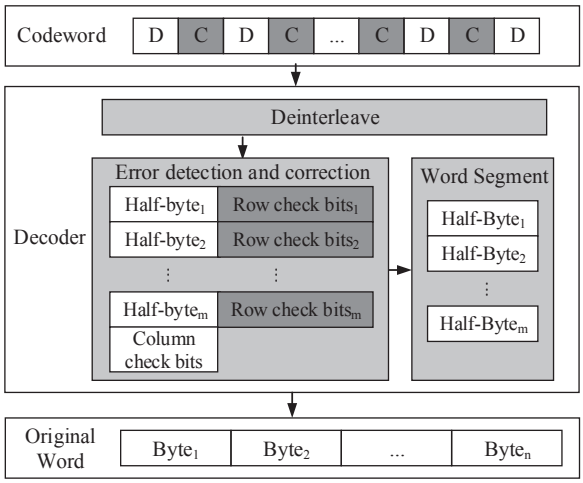


FIGURE 2. Block diagram of the decoder. The codeword is first deinterleaved to recover the data bits and check bits. Then, errors in the data bits are detected and corrected. Finally, the data segments are concatenated to recover the original word.

is computed by multiplying the data bit vector and generator matrix as follows.

$$cw_m = d_m * \boldsymbol{G}, \tag{1}$$

where cw_m is the mth row codeword, d_m represents the data bit vector of the mth segment and $\boldsymbol{G} = \left[\begin{array}{cc} \boldsymbol{I_4} & \boldsymbol{B^T} \end{array} \right]$ is the generator matrix. To reduce the complexity of calculation, $\boldsymbol{B}$ must meet the following restrictions:

1) every column in $\boldsymbol{B}$ contains an odd number of 1s;
2) the number of 1s in $\boldsymbol{B}$ is minimised;
3) the number of 1s in every row is as equal as possible and the difference is no larger than one;
4) there is no column with all 0s;
5) every column is distinct.

We use the following generator matrix which meets the restrictions.

$$\boldsymbol{G} = \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array} \right]. \tag{2}$$

After calculation, all the row codewords are arranged by row to form the row codeword matrix, denoted by $\boldsymbol{C}$, as shown in Fig. 3(b). Then, all bits in each column of $\boldsymbol{D}$ are XORed to generate a column check bit. All the column check bits form the column codeword vector, denoted by $\boldsymbol{O}$, as shown in the last row of Fig. 3 (c).

C. THE DECODING PROCESS

Suppose that the codeword has been deinterleaved and the row codeword matrix $\boldsymbol{C}$ and the column codeword vector $\boldsymbol{O}$ are obtained. The syndrome matrix $\boldsymbol{J}$ of $\boldsymbol{C}$ is calculated using parity check matrix $\boldsymbol{H}$. The mth row of $\boldsymbol{J}$ is:

$$J_m = cw_m * \boldsymbol{H}', \tag{3}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3178953, IEEE Access

	1	2	3	4
1	D_0	D_1	D_2	D_3
2	D_4	D_5	D_6	D_7

(a)

	1	2	3	4	5	6	7	8
1	D_0	D_1	D_2	D_3	C_0	C_1	C_2	C_3
2	D_4	D_5	D_6	D_7	C_4	C_5	C_6	C_7

(b)

	1	2	3	4	5	6	7	8
1	D_0	D_1	D_2	D_3	C_0	C_1	C_2	C_3
2	D_4	D_5	D_6	D_7	C_4	C_5	C_6	C_7
3	O_0	O_1	O_2	O_3				

(c)

FIGURE 3. Example of codeword generation for an 8-bit word.

Algorithm 1 Codeword Generation without Interleaving

Input: original data $\boldsymbol{D_w}[0:n-1]$
Output: row codeword matrix $\boldsymbol{C}$, column codeword vector $\boldsymbol{O}$
1: $j=0, \boldsymbol{D}=[], \boldsymbol{C}=[], \boldsymbol{O}=[]$
2: **for** $i=0$ to $n/4-1$ **do**
3: Put the data from $\boldsymbol{D_w}[i\times 4 : i\times 4+3]$ to the jth row of $\boldsymbol{D}$
4: $j=j+1$
5: **end for**
6: **for** $i=0$ to $j-1$ **do**
7: $\boldsymbol{C}[i,:]=\boldsymbol{D}[i,:]\times\boldsymbol{G}$
8: **end for**
9: **for** $i=0$ to 3 **do**
10: XOR all data in $\boldsymbol{D}[:,i]$ and output the result to the ith column of $\boldsymbol{O}$
11: **end for**
12: Output $\boldsymbol{C}$ and $\boldsymbol{O}$.

where J_m is the syndrome of the mth codeword and $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{I_4} \end{bmatrix}$ as shown in Eq. (4):

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

$\boldsymbol{J}$ and $\boldsymbol{O}$ are jointly used to detect and correct errors in cw_m. We now discuss the following situations:

1) Suppose that there is no error in the data and check bits. Then $\boldsymbol{J}$ is an all-0 matrix. No further processing is needed.
2) Suppose that there is only one error in the data and check bits. Further consider the following two cases:
   a) The error happens in $\boldsymbol{O}$. Then $\boldsymbol{J}$ is an all-0 matrix. No further processing is needed.
   b) The error happens in $\boldsymbol{C}$. Then one row of $\boldsymbol{J}$, say J_m is equal to the pth column of $\boldsymbol{H}$ while other rows of $\boldsymbol{J}$ are $\boldsymbol{0}$. This tells us that the bit in the mth row and pth column of $\boldsymbol{C}$ is wrong. Therefore, the correct codeword can be generated by flipping the pth bit of cw_m.
3) Suppose that there is two errors in the data and check bits. Further consider the following four cases:
   a) The two errors both happen in $\boldsymbol{O}$. Then $\boldsymbol{J}$ is an all-0 matrix. No further processing is needed.
   b) One error happens in $\boldsymbol{O}$ and the other error happens in $\boldsymbol{C}$. Then, the error in $\boldsymbol{C}$ can be corrected, as in the case 2)b).
   c) Both errors happens in $\boldsymbol{C}$ but are in different rows. Then there are two rows in $\boldsymbol{J}$, say J_m and J_n, which are equal to column p and column r in $\boldsymbol{H}$, respectively. This tells us that the bit in the mth row and pth column as well as the bit in the nth row and rth column of $\boldsymbol{C}$ are wrong. Therefore, the correct codeword can be generated by flipping the pth bit of cw_m and the rth bit of cw_n.
   d) Both errors happens in the same row of $\boldsymbol{C}$. Suppose it is row m. Then J_m is equal to the result of XOR of two different columns in $\boldsymbol{H}$. This tells us that two errors happens in c_m. Then, $\boldsymbol{O}$ can be used to correct these two errors because they happen in different columns of $\boldsymbol{C}$, thereby ensuring that each column contains at most one error. Specifically, to get the correct data, the first four bits of cw_m is XORed with $\boldsymbol{O}$. Note that we do not care about error(s) in the last four bits of cw_m, which are the check bits.

Based on the above analysis, it can be seen that the proposed scheme can ensure the correct recovery of data when its storage content encounters a single or double random errors. Hardware implementation of the decoder for 8-bit data is illustrated in Fig. 4.

D. INTERLEAVING FOR TAEC ABILITY

As discussed above, with the first two encoding steps, the proposed scheme can ensure the correct recovery of data when it encounters single error bit or two error bits. We now further examine the decoding process to see if it is possible to correct triple bit errors. Consider the following conditions:

1) If the three errors happen in different rows, then $\boldsymbol{J}$ is able to detect and correct all errors that happen in $\boldsymbol{C}$ and the error (if any) in $\boldsymbol{O}$ does not matter.
2) If two errors happen in $\boldsymbol{O}$ and one error happen in $\boldsymbol{C}$, then the error in $\boldsymbol{C}$ can be corrected, as in case 1).
3) If two errors happen in the same row in $\boldsymbol{C}$ and the other error happens in a different column in $\boldsymbol{O}$, then the first two errors can be detected by $\boldsymbol{J}$ and corrected by $\boldsymbol{O}$.
4) If two errors happen in the same row in $\boldsymbol{C}$ and the other error happens in a different row and different column in $\boldsymbol{C}$, then the first two errors can be detected by $\boldsymbol{J}$ and corrected by $\boldsymbol{O}$, while the other error can be corrected by $\boldsymbol{J}$.

Based on above analysis, we can see that if the three error bits fulfill some certain patterns, then they can be detected and corrected. Therefore, triple adjacency error correction ability can be achieved without adding additional check bits using interleaving technique. That is, if we can ensure that any triple adjacency error that happens after interleaving fulfills one of the above conditions after deinterleaving, then the triple adjacency error can be corrected. To achieve triple adjacency error correction, the data bits and check bits are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3178953, IEEE Access

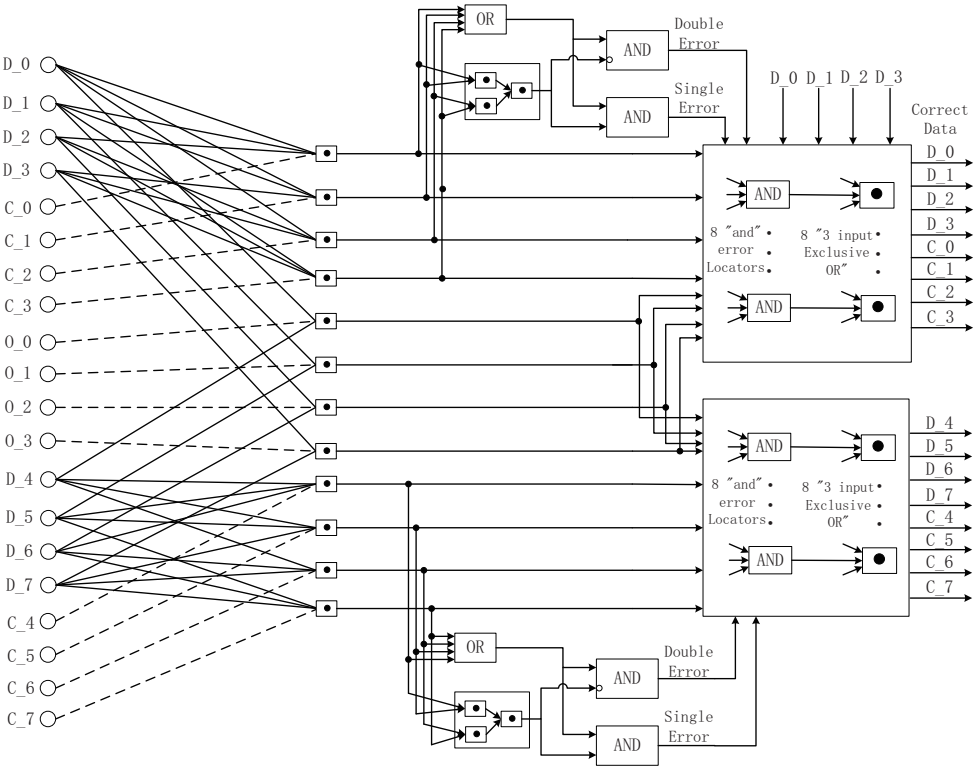


FIGURE 4. Decoding circuitry of the proposed method for 8-bit data.

interleaved according to the following rules.

1) Any triple adjacent bits cannot be composed of data bits and check bits in the same row of $\boldsymbol{C}$.
2) Any triple adjacent bits cannot be composed of one column check bit (denoted by O_i), one data bit in the same column with O_i (denoted by D_i), and another bit in the same row with D_i.

To fulfill the above rules, we propose the following interleaving scheme. Define $\boldsymbol{I}$ as the interleaving matrix, where the data and check bits are put into. $\boldsymbol{I}$ is a k by 4 matrix, where $k = 2 \times m + 1$. Two cases are seperately considered, where m is even and odd, respectively.

1) m is even

Put $\boldsymbol{O}'$ into the $(m+1)$th column of $\boldsymbol{I}$. For each row i with odd number (i.e., $i = 1, 3, \dots$) in $\boldsymbol{C}$, put the first four bits (data bits) into the ith of column $\boldsymbol{I}$, put the last four bits (check bits) into the $(m+1+i)$th column of $\boldsymbol{I}$. For each row j with even number (i.e., $j = 2, 4, \dots$) in $\boldsymbol{C}$, put the first four bits (data bits) into the $(m+1+j)$th column of $\boldsymbol{I}$, put the last four bits (check bits) into the jth column of $\boldsymbol{I}$. The interleaving process for a 16-bit data word is illustrated in Fig. 5.

2) m is odd

Put $\boldsymbol{O}'$ into the $(m+2)$th column of $\boldsymbol{I}$. For each row i with odd number (i.e., $i = 1, 3, \dots$) expect the mth row in $\boldsymbol{C}$, put

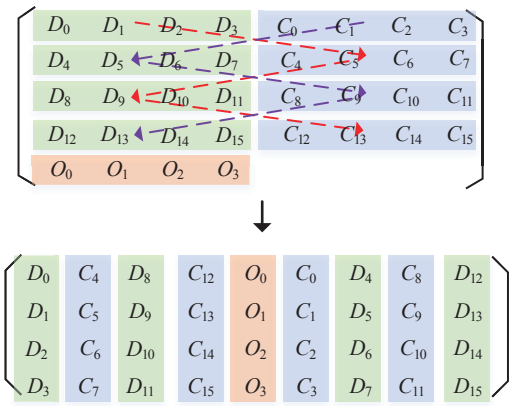


FIGURE 5. Interleaving process for a 16-bit data word. The upper part is $\boldsymbol{C}$ and $\boldsymbol{O}$. The lower part is $\boldsymbol{I}$. Red lines in the upper part indicate the order by which data and check bits are put into the left part of $\boldsymbol{I}$. Purple lines in the upper part indicate the order by which data and check bits are put into the right part of $\boldsymbol{I}$.

the first four bits (data bits) into the $(i+2)$th of column $\boldsymbol{I}$, put the last four bits (check bits) into the $(m+2+i)$th column of $\boldsymbol{I}$. For each row j with even number (i.e., $j = 2, 4, \dots$) in $\boldsymbol{C}$, put the first four bits (data bits) into the $(m+2+j)$th column of $\boldsymbol{I}$, put the last four bits (check bits) into the jth column of $\boldsymbol{I}$. For row m in $\boldsymbol{C}$, put the first four bits (data bits) into the first column of $\boldsymbol{I}$, put the last four bits (check bits) into the $(m+1)$th column of $\boldsymbol{I}$. The interleaving process for a 20-bit

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3178953, IEEE Access

data word is illustrated in Fig. 6.

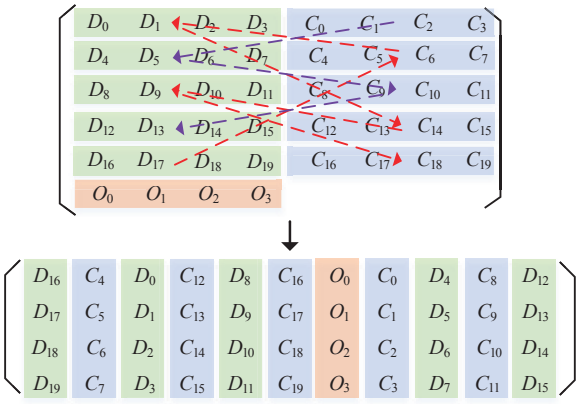


FIGURE 6. Interleaving process for a 20-bit data word. The upper part is $\boldsymbol{C}$ and $\boldsymbol{O}$. The lower part is $\boldsymbol{I}$. Red lines in the upper part indicate the order by which data and check bits are put into the left part of $\boldsymbol{I}$. Purple lines in the upper part indicate the order by which data and check bits are put into the right part of $\boldsymbol{I}$.

After generating $\boldsymbol{I}$, the rows of $\boldsymbol{I}$ are concatenated to form the final codeword. The deinterleaving process in the decoding phase is a reverse process of the interleaving process. With interleaving, the proposed code achieves SEC-DEC-TAEC ability.

IV. PERFORMANCE EVALUATION

In this section, we evaluate the error correction ability and implementation cost of the proposed code. The encoder and decoder are implemented in HDL according to given $\boldsymbol{G}$ and $\boldsymbol{H}$. It should be noted that in actual chips, both data bits and check bits will have problems such as single particle flipping. Therefore, in the evaluation of the code's error correction ability, we treat the check bits and the data bits equally.

A. ERROR CORRECTION ABILITY

The first part of the evaluation focuses on evaluating the error correction ability of the proposed code. The circuit under test (CUT) is an FSM controlled waterfall light circuit implemented using Verilog, the code of which has been uploaded to Github [41]. To simulate the SEU and MBU failures occurring in the circuit, the bit flip is chosen as the basic fault injection model. Therefore, the FPGA platform is used to evaluate the code correction ability by flipping the bit/bits of specified register in the CUT. To better simulate different error scenarios, all registers in the CUT are replaced by the structure as shown in Fig. 7, where the EIE signal is the enable of bit fault injection. When the EIE signal is deactivated, the register is in normal operation since the normal signal is directly selected to the input of the register. On the contrary, when the EIE signal is activated, the reverse of the normal signal value will be selected to the input of the register and the single bit fault will be injected to the circuit.

According to the run state of the CUT, the result of error injection can be divided into three categories as follows.

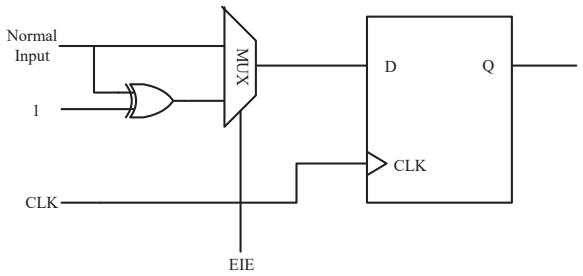


FIGURE 7. The new structure of register. The input of the register is the output of a multiplexer whose inputs are normal input and the xor result of normal input and "1". In addition, the select signal of the multiplexer is error inject enable (EIE), which controls whether the register is injected with fault.

- **Silent State**. For a given runtime, the output of the CUT and all register state in the CUT are normal.
- **Latent State**. For a given runtime, the output of the CUT is normal, but the state of some registers in the CUT is different from that when the error is not injected. In this state, although the injected error has no effect on the circuit function, it may result in a functional error in the subsequent operation.
- **Failure State**. For a given runtime, the output of the CUT is different from that when the error is not injected.

In this paper, to maximize the proper operation of the circuit, the error correction code is considered valid only when the circuit is in the silent state under error injection. The LFSR is used to generation all possible combinations of bit flip in the specified register according the injected error type such as single error, double error, double-adjacent error and so on. For example, for the case where the original data is 4-bit and the injected error type is double error, there will be 6 different combinations of two flip bits. We can generate the LFSR circuit as shown in Fig. 8. If the initial state of LSFR is "1000", the state sequence of it is $1000 \rightarrow 1100 \rightarrow 1110 \rightarrow 0111 \rightarrow 0011 \rightarrow 0001 \rightarrow 1000$. Thus, all possible combinations of flip bits can be generated according to the different output states of the LFSR circuit.

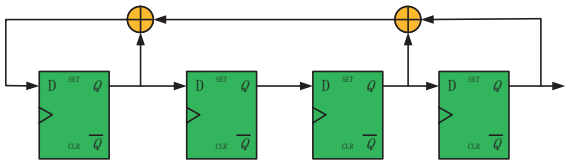


FIGURE 8. An example of LFSR circuit, which the number of DFFs equals the bits number of data and the number of output states equals the number of all possible combinations of flip bits.

As our goal in this section is to measure the correction coverage, we have not injected errors according the error occurrence probability. We have injected single error, double errors, double-adjacent errors, triple errors and triple-adjacent errors in all bits of the data by the corresponding LFSR circuit. For each error type, 1,000,000 errors are injected. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3178953, IEEE Access

TABLE 1. Error correction ability of different codes.

Error types	Code in [14]	Code in [16]	Code in [33]	Code in [34]	Code in [35]	Proposed Code
Single Error	100%	100%	100%	100%	100%	**100%**
Double Adjacent Error	91.7%	100%	100%	100%	100%	**100%**
Double Error	80%	48.6%	16.2%	29%	13%	**100%**
Triple Adjacent Error	79.31%	100%	100%	100%	100%	**100%**
Triple Error	25.7%	22.5%	39.7%	23.8%	27.2%	**50.93%**

metric is calculated as

$$\text{Error\_correction\_ability} = \frac{\text{No. of Corrected}}{\text{No. of Injected}} \times 100\% \quad (5)$$

The code proposed in this paper is compared with the codes proposed in [14], [16], [33], [34] and [35]. The results are shown in Table 1. Reference [14] combines Hamming codes and parity codes to achieve single-bit error correction, but cannot achieve 100% correction for double-bit errors. The codes in [16], [33], [34] and [35] can correct single error, double-adjacent errors, triple-adjacent errors, but cannot correct 100% of double-bit random errors. The decoding accuracy of the proposed code is 100% in the case of single, double and triple adjacent error(s) and 50.93% in the case of random triple errors. This demonstrates the superior error correction ability of the proposed code.

B. IMPLEMENTATION COST

In electronic systems, especially in space electronic systems, minimization of the implementation cost in terms of area, power, and delay is very important. This part of the evaluation focuses on evaluating the cost of the encoder and decoder of the error correction code. The length of state registers in electronic systems is generally 8-bit. Therefore, we focus on the case where the original data bit is 8. It has to remarked that the codes in [14], [16], [33], [34] and [35] do not support data with bit width of 8. Meanwhile, the Hamming code [1] and OLS code [23] support 8-bit data and have relatively simple encoder and decoder circuit. Therefore, the proposed code is compared with the Hamming code [1] and the OLS code [23] in this section. To estimate the cost, we have implemented the encoder and decoder circuit for the three codes in Verilog, and used the Synopsis Design Compiler (DC) configured to minimize the circuit area with the SMIC 180 nm library.

The results are summarized in Table 2. It can be seen that although the proposed code has larger power, area and delay compared with the Hamming code, it has much stronger error correction ability. In addition, compared with OLS code, the proposed code has better performance in power, area and delay. Specifically, the area occupation is reduced by 30% and delay is reduced by 15%.

C. THE IMPACT OF BIT-WIDTHS

The third part of the evaluation focuses on the impact of different bit-widths on the area, power and delay for the proposed code. To make the evaluation more representative, the most commonly used data bit widths in FPGA/ASIC are selected, including 8, 12, 16, 20, 32 and 64 bits. The encoder and decoder are mapped to the SMIC 180 nm device library with DC. The results are summarized in Table 3. It can be seen that as the bit-width to be protected increases, the power and area of the proposed code increase to a large extent, but the increase in delay is small, which demonstrates the low-delay property of the code.

In order to further evaluate the proposed code in actual environment, the encoder and decoder are mapped to the SMIC 180 nm device library with the Synopsys IC Compiler software (ICC). The results are summarized in Table 4. It can be seen that the delay increase is minimal. In addition, we compare the proposed code with the triple-mode redundancy (TMR) scheme in a practical project, and the results show that the occupied area using the proposed scheme is 43.5% of the TMR scheme.

TABLE 2. Implementation cost of different codes.

		Hamming Code [1]	OLS Code [21]	Proposed Code
Encoder	Power(μw)	10.59	21.73	14.57
	Area(μm^2)	266.12	638.67	425.78
	Delay(ns)	0.48	0.39	0.32
Decoder	Power(μw)	38.70	89.41	69.59
	Area(μm^2)	861.54	1663.20	1423.7
	Delay(ns)	0.97	1.37	1.11
Error Correction Ability		SEC-DED	DEC	SEC-DEC-TAEC
Parity Bits		4	12	12

TABLE 3. The implementation cost under different bit-widths for the proposed code (results in DC).

	Encoder			Decoder		
	Power (μw)	Area (μm^2)	Delay (ns)	Power (μw)	Area (μm^2)	Delay (ns)
8-bit	14.57	425.78	0.32	69.59	1423.7	1.11
12-bit	25.26	691.9	0.32	108.03	2155.50	1.21
16-bit	35.31	958.0	0.32	148.54	2874.0	1.25
20-bit	46.95	1224.12	0.44	189.73	3592.51	1.25
32-bit	81.22	2022.45	0.45	323.83	5794.58	1.26
64-bit	197.08	4251.35	1.1	693.85	11569.22	1.75

V. CONCLUSION

In this paper, we present an EDAC code to achieve low-delay SEC-DEC-TAEC ability for state register protection. The proposed code is based on Hsiao code and parity check bit to achieve SEC-DEC, while the introduction of interleaving

TABLE 4. The implementation cost under different bit-widths for the proposed code (results in ICC).

	Encoder			Decoder		
	Power (μw)	Area (μm^2)	Delay (ns)	Power (μw)	Area (μm^2)	Delay (ns)
8-bit	35.40	95987	1.12	107.4	96102	1.78
12-bit	54.59	137129	1.42	167.8	137303	1.80
16-bit	76.77	178280	1.44	229.5	178506	1.81
20-bit	95.23	219420	1.47	298.4	219711	1.83
32-bit	160	342870	1.5	499.2	343338	1.83
64-bit	321.7	672044	1.71	1089.4	673017	2.15

further provides TAEC ability. Besides, variable data bit widths are supported by the code. Compared with existing coding schemes, the proposed code has more advantages for state register protection with its low-delay property and enhanced error correction ability. In the future, we will focus on how to further reduce the check bits of the proposed code.

REFERENCES

[1] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147-160, Apr. 1950.

[2] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395-401, Jul. 1970.

[3] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 124-134, Mar. 1984.

[4] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applications*, 1st ed., Wiley, NY, USA, 2005.

[5] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule," *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527-1538, Jul. 2010.

[6] G. Tsiligiannis, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Todri, et al., "Multiple cell upset classification in commercial SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 4, pp. 1747-1754, Aug. 2014.

[7] G. I. Zebrev, K. S. Zemtsov, R. G. Useinov, M. S. Gorbunov, V. V. Emeliyanov and A. I. Ozerov, "Multiple cell upset cross-section uncertainty in nanoscale memories: Microdosimetric approach," in *Proc. 15th Eur. Conf. Radiat. Effects Compon. Syst. (RADECS)*, pp. 1-5, Sep. 2015.

[8] N. Chechenin and M. Sajid, "Multiple cell upsets rate estimation for 65 nm SRAM bit-cell in space radiation environment," in *Proc. 3rd Int. Conf. Exhib. Satell. Space Missions*, pp. 77, May 2017.

[9] N. N. Mahatme, B. L. Bhuva, Y.-P. Fang and A. S. Oates, "Impact of strained-Si PMOS transistors on SRAM soft error rates," *IEEE Trans. Nucl. Sci.*, vol. 59, no. 4, pp. 845-850, Aug. 2012.

[10] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code," in *Proc. IEEE VLSI Test Symp. (VTS)*, pp. 349-354, 2007.

[11] L. J. Saiz-Adalid, P. Reviriego, P. Gil, S. Pontarelli and J. A. Maestro, "MCU tolerance in SRAMs through low-redundancy triple adjacent error correction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 10, pp. 2332-2336, Oct. 2015.

[12] A. Neale and M. Sachdev, "A new SEC-DED error correction code subclass for adjacent MBU tolerance in embedded memory," *IEEE Trans. Device Mater. Rel.*, vol. 13, no. 1, pp. 223-230, Mar. 2013.

[13] C. Argyrides, H. R. Zarandi and D. K. Pradhan, "Matrix codes: Multiple bit upsets tolerant method for SRAM memories," in *Proc. 22nd IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, pp. 340-348, Sep. 2007.

[14] C. Argyrides, D. K. Pradhan and T. Kocak, "Matrix codes for reliable and cost efficient memory chips," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 3, pp. 420-428, Mar. 2011.

[15] H. S. de Castro, J. A. N. da Silveira, A. A. P. Coelho, F. G. A. e Silva, P. D. S. Magalhães and O. A. de Lima, "A correction code for multiple cells upsets in memory devices for space applications," in *Proc. 14th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, pp. 1-4, Jun. 2016.

[16] J. Gracia-Moran, L. -J. Saiz-Adalid, J. -C. Baraza-Calvo and P. Gil, "Correction of adjacent errors with low redundant matrix error correction codes," in *Proc. 8th Latin-American Symp. on Dependable Computing*, pp. 107-114, 2018.

[17] F Silva, W Freitas, J Silveira, C Marcon and F Vargas, "Extended matrix region selection code: An ECC for adjacent multiple cell upset in memory arrays," *Microelectron. Rel.*, vol. 106, pp. 113582, 2020.

[18] D. C. C. Freitas, D. F. M. Mota, C. Marcon, J. A. N. Silveira and J. C. M. Mota, "LPC: an error correction code for mitigating faults in 3D memories," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 2001-2012, 1 Nov. 2021.

[19] S. Ghosh and P. D. Lincoln, "Dynamic low-density parity check codes for fault-tolerant nanoscale memory," in *Proc. Found. Nanosci.*, pp. 1-5, 2007.

[20] P. Reviriego, M. F. Flanagan, S. F. Liu and J. A. Maestro, "Multiple cell upset correct ion in memories using difference set codes," *IEEE Trans. Circuits Syst. I Reg. Papers*, vol. 59, no. 11, pp. 2592-2599, Nov. 2012.

[21] H. Y. Hsiao, "Orthogonal Latin square codes," *IBM J. Research and Development*, vol. 14, no. 4, pp. 390-394, July 1970.

[22] S. Liu, L. Xiao and Z. Mao, "Extend orthogonal Latin square codes for 32-bit data protection in memory applications," *Microelectron. Rel.*, vol. 63, pp. 278-283, Aug. 2016.

[23] P. Reviriego, S. Liu, A. Sánchez-Macián, L. Xiao and J. A. Maestro, "A scheme to reduce the number of parity check bits in orthogonal Latin square codes," *IEEE Trans. Rel.*, vol. 66, no. 2, pp. 518-528, Jun. 2017.

[24] P. Reviriego, S. Liu, A. Sánchez-Macián, L. Xiao and J. A. Maestro, "Reduction of parity overhead in a subset of orthogonal Latin square codes," in *Proc. 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, pp. 1-5, 2020.

[25] C. Ramamurthy, S. Chellappa, V. Vashishtha, A. Gogulamudi and L. T. Clark, "High performance low power pulse-clocked TMR circuits for soft-error hardness," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 6, pp. 3040-3048, Dec. 2015.

[26] K. Morgan, D. McMurtrey, B. Pratt and M. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 6, pp. 2065-2072, 2007.

[27] R. Shuler, B. Bhuva, P. O'Neill, J. Gambles and S. Rezgui, "Comparison of dual-rail and TMR logic cost effectiveness and suitability for FPGAs with reconfigurable SEU tolerance," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 1, pp. 214-219, Dec. 2009.

[28] K. Namba and F. Lombardi, "A single and adjacent error correction code for fast decoding of critical bits," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1525-1531, Oct. 2018.

[29] J. Li, P. Reviriego, L. Xiao, Z. Liu, L. Li, and A. Ullah, "Low delay single error correction and double adjacent error correction (SEC-DAEC) codes," *Microelectron. Rel.*, vol. 97, pp. 31-37, Jun. 2019.

[30] L. Saiz-Adalid, J. Gracia-Morán, D. Gil-Tomás, J. -. Baraza-Calvo and P. Gil-Vicente, "Ultrafast codes for mltiple adjacent error correction and double error detection," *IEEE Access*, vol. 7, pp. 151131-151143, 2019.

[31] A. Sánchez-Macián, P. Reviriego, J. Tabero, A. Regadío and J. A. Maestro, "SEFI protection for nanosat 16-bit chip onboard computer memories," *IEEE Trans. Device Mater. Rel.*, vol. 17, no. 4, pp. 698-707, Dec. 2017.

[32] S. Ahmad, M. Zahra, S. Z. Farooq and A. Zafar, "Comparison of EDAC schemes for DDR memory in space applications," in *Proc. Int. Conf. Aerosp. Sci. Eng. (ICASE)*, pp. 1-5, Aug. 2013.

[33] A. Neale, M. Jonkman and M. Sachdev, "Adjacent-MBU-tolerant SEC-DED-TAEC-yAED codes for embedded SRAMs," *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 62, no. 4, pp. 387-391, Apr. 2015.

[34] J. Gracia-Morán, L. J. Saiz-Adalid, D. Gil-Tomás and P. J. Gil-Vicente, "Improving error correction codes for multiple-cell upsets in space applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 10, pp. 2132-2142, Oct. 2018.

[35] S. Vijayakumaran and D. Pal, "On the minimum redundancy of SEC-DAEC-TAEC binary linear block codes," *IEEE Communications Letters*, vol. 20, no. 4, pp. 652-655, Apr. 2016.

[36] A. Pinheiro, D. Tavares, F. Silva, J. Silveira and C. Marcon. "Optimized buffer protection for network-on-chip based on Error Correction Code," *Microelectronics Journal*, vol. 100, 2020.

[37] N. Sridevi, K. Jamal and K. Mannem, "Implementation of error correction techniques in memory applications," *Proc. 2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 586-595, 2021.

[38] L.-J. Saiz-Adalid, J. Gracia-Morán, D. Gil-Tomás, J.-C. Baraza-Calvo, and P.-J. Gil-Vicente, "Reducing the overhead of BCH codes: new double error correction codes," *Electronics*, vol. 9, no. 11, p. 1897, Nov. 2020.

[39] R. K. Maity, S. Tripathi, J. Samanta, and J. Bhaumik. "Lower complexity error location detection block of adjacent error correcting decoder for SRAMs," *IET Computers & Digital Techniques*, vol. 14, no. 5, pp. 210-216, 2020.

[40] L. Ramasethu, P. Poongodi, "A class of SEC-DAED-TAEC codes for fault secure SRAM," *Journal of Computational and Theoretical Nanoscience*, vol. 14, no. 12, pp. 5897-5900, 2017.

[41] M. Dong (2022). An Efficient Universal Low Delay SEC-DEC-TAEC Code. [Online]. Available: https://github.com/Melvin-Dong/An-Efficient-Universal-Low-Delay-SEC-DEC-TAEC-Code.

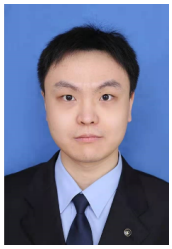
MENG DONG was born in 1995. He received the B.S. degree in school of information communication engineering from Beijing Information Science & Technology University, Beijing, China, in 2016. He is currently pursuing the Ph.D. degree with the Xidian University, Xi'an, China. His research interests include hardware security.

WEITAO PAN received B.S. degree from School of Technical Physics of Xidian University in 2004. His Ph.D. degree was received from School of Microelectronics of Xidian University in 2010. Now he is an associate professor in State Key Laboratory of Integrated Service Networks of Xidian University. His current research interests include VLSI design methods and post-silicon verification.

ZHILIANG QIU is a Professor with the State Key Laboratory of Integrated Services Networks (ISN), Xidian University, Xi'an, China. He received the B.S. degree in communication engineering and the M.S. and Ph.D. degrees in communication and information systems from Xidian University, in 1986, 1989, and 1999, respectively. His research interests include broadband network and switching technology.

XIAOXIN QI was born in 1994. He received the B.S. degree in telecommunication engineering from Xidian University, Xi'an, China, in 2016. He is currently pursuing the Ph.D. degree in information and telecommunication engineering at Xidian University. His research interests include internetworking and routing in satellite networks.

LING ZHENG received the M.S. degree in computer science and technology in 2014, and the Ph. D. degree in information and communication engineering in 2019 respectively, from Xidian University, Xi'an, P.R.China. He is currently a Lecturer with the School of Communications and Information Engineering, Xi'an University of Posts and Telecommunications. His research interests include high performance switching and routing, software-defined networking and deterministic network.



HUAN LIU is currently pursuing the Ph.D. degree with the Xidian University. His research interests include reconfigurable hardware and networked systems. He has rich research experience in dataplane architectures of the network device.