IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 10, 2022, accepted February 21, 2022, date of publication February 24, 2022, date of current version March 2, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3154009*

# An Overhead-Reduced Key Coding Technique for High-Speed Serial Interface

**JAEPIL BAK[1], TAEK-JOON AN[1], YOUNGWOO KIM[2], (Member, IEEE), AND JIN-KU KANG[1], (Senior Member, IEEE)**

[1]Department of Electronic Engineering, Inha University, Incheon 22212, South Korea
[2]Department of System Semiconductor Engineering, Sangmyung University, Cheonan 31066, South Korea

Corresponding author: Jin-Ku Kang (jkang@inha.ac.kr)

This work was supported by Inha University.

**ABSTRACT** This paper describes a packet-based overhead-reduced (OR) key coding technique for a high-speed serial interface. The 8B10B code is a de facto standard coding technique in the application but its bit-overhead is 25%. The proposed key coding technique is to reduce the coding overhead and still provides enough bit transition to facilitate clock and data recovery in the receiver. After a key pattern is generated from a certain data stream, input data are encoded and framed as packets along with the generated key for transmission. The packets are transmitted and then decoded as original data in the receiver. Using the proposed coding scheme, 4-, 6-, and 8-bit key coding systems are designed and compared. When a 6-bit key coding encoder/decoder is tested, a packet is composed of a 6-bit OR key header followed by 30 encoded sub-packets, in which each sub-packet has a 6-bit data. In the 6-bit case, the bit overhead is only 3.33% and the maximum continuous run length is 10 bits. To control the running disparity for the AC coupling interface, a logic for selecting the optimal key is implemented to keep the running disparity as low as possible. The running disparity of the encoded data with 6-bit key code is controlled within +/−12.

**INDEX TERMS** High-speed serial interface, clock and data recovery (CDR), 8B10B code, coding overhead, run-length, running disparity, overhead-reduced key coding.

## I. INTRODUCTION

As technology advances, higher data rate transmission is required in optical communications, digital video, memory, data storage systems, and other high-speed serial interfaces. Currently 8B10B data coding is most widely accepted as a de facto standard for data transmission and receiving in most serial communication systems [1], [2]. The 8B10B code has a maximum run length of 5 bits to guarantee enough transitions to facilitate clock and data recovery. Its running disparity is controlled within +/−2; thus, it is suitable as an AC coupling interface. However, the transmission overhead of the 8B10B code is 25%. This means more data bits are sent than in the original. Recently, 64B66B code is being used in optical communication systems [3] and it has the bit-overhead of 3.12%, but the maximum run length is 66 bits, which requires a tighter clock and data recovery (CDR) circuit design. Furthermore, the running disparity of 64B66B coding could be very large. A large DC blocking capacitance or a baseline wandering correction circuit may be needed in AC-coupled channels [4, 5]. Work was done on a 24B27B code with DC balancing for display interface [6]. This scheme reduces the overhead to 12.5% with running length to 8 bits and adds a block to control the running disparity. Recently, as PAM-4 signaling has gotten attention in high-speed interface applications, DC-balancing PAM-4 coding techniques have been published [7], [8]. However, their overhead ratio stayed the same as that of 8B10B, which is 25%.

This paper describes a packet-based key coding technique to reduce the bit transmission overhead drastically down to 3.33%. Running disparity control is also a factor to be considered in the design, especially in an AC coupling interface. With the proposed coding technique, the running disparity is increased compared to 8B10B coding. However, DC level wandering from the increased running disparity can be attenuated by the use of a larger DC blocking capacitor. In this paper, the principle of the proposed coding is described, first. Then the design and measurement results are presented, and conclusions follow.

The associate editor coordinating the review of this manuscript and approving it for publication was Wen-Sheng Zhao.

## II. PRINCIPLE OF OPERATION AND ARCHITECTURE

The proposed overhead-reduced (OR) key coding technique is designed to minimize the bit transmission overhead in a high-speed serial interface. For the proposed technique, input data blocks of multiple $N$-bit sized sub-blocks are processed for each dedicated $N$-bit OR key coding. A single encoded sub-packet is generated by XORing the OR key pattern with one sub-block of input data. A single packet consists of the OR key in the header followed by encoded multiple $N$-bit sub-packets. With the proposed OR key coding, there is at least a single bit data transition (either 0 to 1 or 1 to 0) in every encoded sub-packet, by which a stable clock and data recovery function can be performed on the receiver side. The block diagrams of the encoder and decoder of the $N$-bit OR key coding technique are illustrated in Fig. 1(a) and (c). A serial-to-parallel converter (S-P) at the front-end is placed assuming that input data is in serial. If input data are available in $N$-bit parallel, S-P block is unnecessary. A single packet format is shown in Fig. 1(b).

The proposed encoding technique is done in three steps and the decoding procedure in the receiver is done in a way that reverses the encoding process. The first step in encoding is finding an OR key from a given number of data bits. The number of input data bits to look for in the OR key generation depends on the number of bits of the OR key. If an $N$-bit-sized OR key is to be generated, $(2^{N-1} - 2)$ sub-blocks should be looked up and a single sub-block consists of $N$-bit data. Details are given in Section II-A. Thus, the total input data bits to be looked up for generating a dedicated 8-bit OR key for the proposed coding scheme are $126 \times 8 = 1008$ bits, which means 126 sub-blocks should be checked and each sub-block composed of 8 bits. Once the OR key is found, the encoded sub-packets are produced by XORing each $N$-bit input data sub-block with the generated OR key. Then, a single packet is framed by the OR key placed in the header and followed by encoded $(2^{N-1} - 2)$ sub-packets, as shown in Fig. 1(b). In case an 8-bit OR key scheme is applied, an encoded single packet for data transmission is composed of an 8-bit OR key in the header followed by 126 encoded sub-packets, where each encoded sub-packet is of 8-bit size data. Then, the encoded packets are transmitted in series through a channel. The serialization should be done before transmission.

On the receiver side, once transmitted encoded packets are arrived at the receiver, the data decoding is being processed. In the decoding procedure, firstly the transmitted OR key is extracted from the header in a packet. Then, the received packets are de-framed with the extracted OR key and the sub-packets. The recovered sub-packets are stored in FIFO for further processing. Then, the final decoded data are obtained by XORing the recovered OR key and encoded sub-packets, as shown in Fig. 1(c).

### A. PRINCIPLE OF OR KEY GERNERATION AND PACKET FRAMING

The proposed OR key scheme is derived from the logical XOR operation between two $N$-bit sized data $X$ and $Y$, i.e., $Z = X \oplus Y$. If $Z$ are all zeroes (000…0) or all ones (111…1), that suggests the case of either $X = Y$ (when Z=000…0) or $X = \overline{Y}$(when Z=111…1), respectively. From this relation, in given $(2^{N-1} - 2)$ sub-blocks of input data $X$ (each sub-block is $N$-bit data), there is at least one $N$-bit pattern of $Y$ that makes $X \neq Y$ or $X \neq \overline{Y}$. One of these possible $N$-bit $Y$'s could be an OR key. Basically, the procedure is to find $N$-bit patterns that are not the same as any of the $N$-bit sub-block patterns and their complementary pairs in $(2^{N-1} - 2)$ sub-blocks. Once an OR key is found, then input sub-blocks ($X$) are encoded as sub-packets by XORing the OR key with each sub-block data. Because OR key is not the same as any of the sub-block data, the encoded sub-packet data ($Z$) has at least one-bit data transition. Among possible OR key candidates from a $N$-bit-sized sub-block, the two special bit patterns consisting of all zeros and all ones are excluded to avoid a continuous same bit pattern. We further simplified the process by selecting one of the complementary pair as OR key candidates. Possible OR key candidates can be determined from $(2^{N-1} - 2)$ sub-blocks, where $N$ is the bit-size of one sub-block. If $N = 6$, a 6-bit OR key is generated from 30 6-bit sub-blocks of input data. After a packet framing, a packet is composed by the OR key in the header plus 30 encoded sub-packets. For example, a 4-bit OR key generation for 4-bit sub-packet is explained. When $N = 4$, the number of sub-blocks to look up is $(2^{N-1} - 2) = 6$. This means that each sub-block is a 4-bit pattern and one OR key should be generated for every 6 sub-blocks. Thus, we must look up 24bits (4bits $\times$6) input data to produce a 4-bit-sized OR key for a single packet. One encoded packet is composed of the OR key followed by 6 encoded sub-packets. Let us assume that input data $A$ with 6 4-bit sub-blocks {1111, 0001, 0010, 0011, 1011, 1001} are coming into the OR key generation block. The original 6 4-bit input patterns and their complementary pairs, {0000, 1110, 1101, 1100, 0100, 0110}, are excluded as OR key candidates. The remaining unmatched 4-bit patterns are {0101, 1010, 0111, 1000}. The first and the second patterns are complementary pairs and the third and the fourth patterns are complementary pairs, respectively. Thus, we pick one of the complementary pairs starting with '0' as the final OR key candidates. Therefore, the final OR code candidates in the input pattern are {0101, 0111}. Either of these two can serve as the OR key.

The logic implementation of OR key generation is shown in Fig. 2. Step (1): As $A_i$ is a single $N$-bit sub-block of data given at a time and it is provided as input data, of the which the first read is the most-significant-bit (MSB) of $A_i$. If MSB is equal to '1', the remaining *(N-1)*-bits are inverted, otherwise, the remaining *(N-1)*-bits are accepted as they are. Step (2): After the multiplexor, an *(N-1)*-bits value is assigned as $E_i$, and a $2^{N-1}$-bit-sized $D_i$ value is produced by a decoder, in which $D_i = 2^{E_i}$. Step (3): After logical ORing operation of $D_i$ and $B$ in the previous cycle ($B_{i-1}$), the result is stored in the $B_i$ register (i.e., $B_i = B_{i-1} + D_i$). This step is for recording which bit position becomes '1' for a single unit of sub-blocks to be encoded. The number of iteration cycles are determined

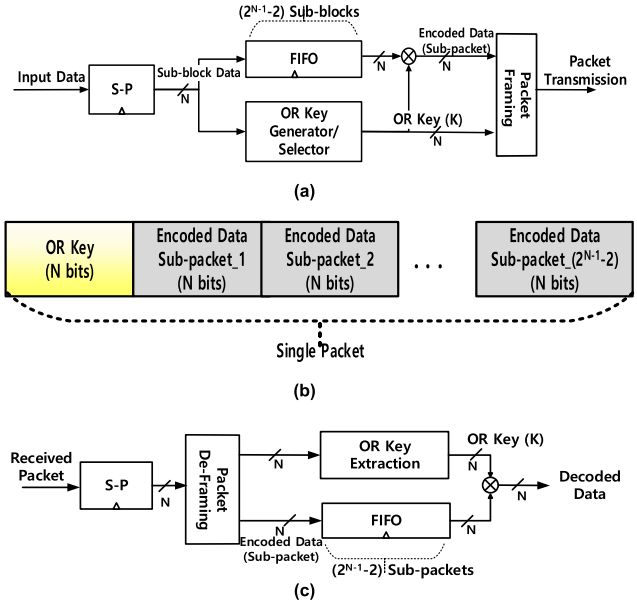


**FIGURE 1.** Block diagram of (a) Encoder, (b) Packet format, and (c) Decoder of the proposed N-bit size OR key coding technique.

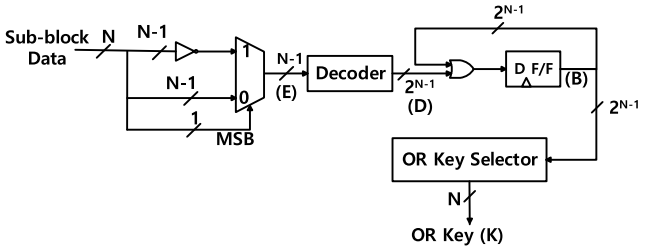


**FIGURE 2.** Logic block diagram of the OR key generator/selector.

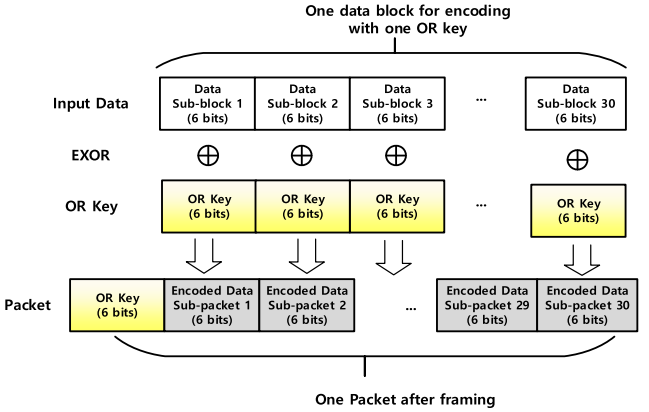


**FIGURE 3.** One packet consisting of a 6-bit OR key in the header plus 30 encoded-sub-packets.

by the bit-size of the OR key. Step (4): Steps (1) and (2) are repeated $(2^{N-1}-2)$ times until the remaining $(2^{N-1}-2)$ sub-blocks are checked. After Step (3), there is at least one zero-value bit in the $B_i$ register. Then, the bit position of zero-value is decoded. The bit position of zero-value can be expressed as an equivalent value of $2^C$, and the corresponding binary value of $C$ becomes the OR key $(K)$ for $(2^{N-1}-2)$ sub-blocks of $A_i$. Once the OR key $(K)$ is found, it is easily verified that $A_i \neq K$ and $A_i \neq \overline{K}$ for all $A_i$, where $i = 1$ to $(2^{N-1}-2)$.

The encoded sub-packet data $Z$ are obtained by XORing the OR key $K$ with $A_i$ (i.e., $Z_i = A_i \oplus K$). Thus, more than one state transition is guaranteed in each encoded sub-packet. There may be more than one OR key candidates for a packet processing. Among those candidates, the one minimizing the running disparity is to be selected through the OR key selector, which will be described later.

Let us check the logic block operation with an input data$A$ with 6 4-bit sub-blocks $\{1111, 0001, 0010, 0011, 1011, 1001\}$ again coming into the OR key generation logic block in Fig. 2. Then the bit patterns at node $E$ (3-bit patterns) become $\{000, 001, 010, 011, 100, 110\}$. As described, if the first bit (MSB) of a sub-block data is '1', the rest of the sub-block bits are inverted. If the first bit (MSB) of a sub-block data is '0', the remaining 3-bits of the sub-block are as they are. Next, the bit patterns at $E$ are decoded using a decoder and the outputs of $D$ are equivalent to $\{2^0, 2^1, 2^2, 2^3, 2^4, 2^6\}$. Because all of six outputs of $D$ are logically added with a logical OR gate, the B register has the values of $\{11111010\}$. There are two '0' values at the 5th and 7th positions from the left and these bit positions are equivalent to binary values of $5 = (0101)_2$ and $7 = (0111)_2$. These two patterns are the OR key candidates, $(0101)_2$ and $(0111)_2$. The inverted values of these two key words also can be the OR key candidates for given input packets. If bit pattern of $(0101)_2$ is selected as an OR key $(K)$ in this example, the encoded sub-packets $Z$ will be $\{1010, 0100, 0111, 0110, 1110, 1100\}$. As shown, each encoded sub-packet has at least one state transition. A packet framed in the example becomes {0101(OR key), 1010, 0100, 0111, 0110, 1110, 1100}.

Fig. 3 shows how the encoded sub-packets are streamed as a packet. In the case where each sub-block is 6-bit data, the OR key is found for every 30 sub-blocks. And after framing the packet, the OR key is placed in the header followed by 30 encoded sub-packets. Therefore, in the $N$-bit OR key system the final framed packet is composed of an OR key in the header followed by $(2^{N-1}-2)$ encoded sub-packets. Therefore, the overhead ratio can be given in Eq. (1) and the overhead ratio is 3.3% in the 6-bit OR key system ($N = 6$ case).

$$Overhead\ ratio = \frac{1}{(2^{N-1}-2)} * 100(\%) \tag{1}$$

The run length is defined as the number of same consecutive bits. Because the maximum run length affects the clock recovery design, it is better to have a lower value for the maximum run length. In the proposed encoding technique, at least one state transition is guaranteed, the maximum run length may occur in worst-case bit patterns when the one encoded sub-packet is 1000…00 and is followed by the encoded sub-packet of 00…0001. It also applies to the inverted bit stream case (0111…11 sub-packet followed by 11…1110 sub-packet). Therefore, the maximum run length is $2 \times (N-1)$ bits and it is 10 bits in a 6-bit OR key system.

As the bit-size $N$ becomes larger, the overhead ratio becomes lower, but the maximum run length affecting the running disparity increases. DC balancing in the data transmission for the AC coupling interface is achieved by the running disparity control. The running disparity is defined as the difference between the numbers of 1's and 0's in a defined block of digits or the instantaneous deviation from the long-term average value of the running digital sum. For AC coupling interface, the running disparity should be controlled under a certain value for minimizing the baseline wandering. Therefore, the bit-size of OR key is to be determined on the overhead ratio and the running disparity. These parameters could be optimized upon applications.

### B. OR KEY SELECTOR DESIGN FOR MINIMIZING RUNNING DISPARITY

Among multiple OR key candidates, the one which minimizes the running disparity is to be selected. In this section, the OR key selector block for running disparity (RD) minimization is described. There is an RD calculator in the selector block from which the maximum RD (*Max_RD*) and the minimum RD (*Min_RD*) values are calculated with all possible OR key candidates. The OR key candidates include non-inverted OR keys (*Noninverted_OR key*) and inverted ones (*Inverted_OR key*). At the beginning, *Initial_RD* value is obtained from the last packet processing. During current packet processing (for example, in a 6-bit OR key system, 30 6-bit units of sub-blocks are to be checked), the RD calculator gives *Max_RD*, *Min_RD*, and *Final_RD,* which is the final value of running disparity in the current OR key generation cycle.

For non-inverted OR key candidates, the larger value is chosen from the absolute of $(initial\_RD + Max\_RD)$ or the absolute of $(initial\_RD + Min\_RD)$ of the corresponding OR key by tracing the RD variation in the current cycle. This can be expressed as in (2).

$$\begin{aligned} &Noninverted\_Max\_RD(ORkey) \\ &\quad = Max\{|initial\_RD + Max\_RD|, \\ &\qquad |initial\_RD + Min\_RD|\} \end{aligned} \tag{2}$$

For inverted OR key candidates, the larger one is chosen from the absolute value of $(Initial\_RD - Max\_RD)$ or the absolute value of $(Initial\_RD - Min\_RD)$ of the corresponding OR key by tracing the RD variation in the packet processing cycle.

$$\begin{aligned} &Inverted\_Max\_RD(ORkey) \\ &\quad = Max\{|initial\_RD - Max\_RD|, \\ &\qquad |initial\_RD - Min\_RD|\} \end{aligned} \tag{3}$$

Then, the minimum is chosen between *Noninverted_Max_RD* and *Inverted_Max_RD,* as represented by the $X$ variable in (4).

$$X = Min\{Noninverted\_MaxRD, Inverted\_MaxRD\} \tag{4}$$

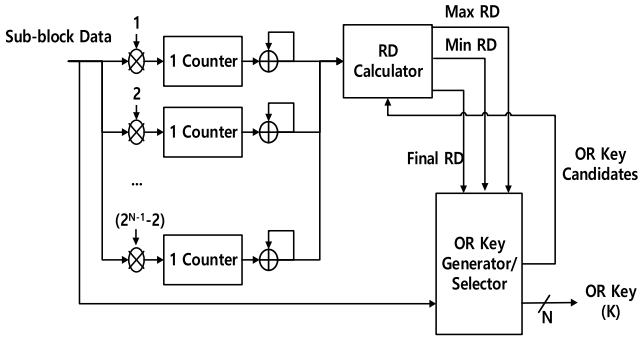


**FIGURE 4.** Block diagram of the OR selector with the RD calculator.

Next, if $X$ is *Noninverted_Max_RD*, the OR key selected is a non-inverted form and *Final_RD* is updated by adding it to *Initial_RD*. Otherwise, the OR key in inverted form is the selected and *Final_RD* is updated by subtracting it from *Initial_RD,* as expressed in the pseudocode below.

$$\begin{aligned} &If\,(X == Noninverted\_Max\_RD), \\ &\qquad \{Final\_RD\_new = Initial\_RD + Final\_RD\} \\ &else\ if\,(X == Inverted\_Max\_RD), \\ &\qquad \{Final\_RD\_new = Initial\_RD - Final\_RD\} \end{aligned}$$

Using the algorithm, for a 6-bit OR key system, the proposed key selection process achieves the running disparity of $+/-12$. Fig. 4 shows a block diagram of the OR key selector with the RD calculator. After checking '1' counters, the RD calculator gives information to select the proper OR key to minimize the running disparity among the OR key candidates generated in the OR key generator block. The OR key selector with RD calculator block is implemented to operate at 125MHz for processing 5Gb/s input data in parallel, as explained in Section III.

Because the OR key coding technique needs a processing step to find the OR key code generation and framing for given input data, the required number of clocks are increased as the bit-size of OR key code increases. The $N$-bit OR key candidate search process requires $(2^{N-1}\text{-}1)$ clock cycles and then up to $2^{N-2}$ clock cycles are needed for finding the final OR key from DC balancing calculation. One more clock cycle is used for XOR operation for data encoding. Thus, $N$-bit OR key coding requires up to $2^{N-1} + 2^{N-2}$ clock cycles. For example, 4-bit OR key coding up to 12 clock cycles are needed compared to typically 2 clock cycles in 8B10B coding [1].

## III. EXPERIMENTAL RESULTS

Using the proposed key coding technique, systems were designed with encoders/decoders for 4-, 6-, or 8-bit OR keys. The three different bit-size OR key systems were explored considering three design parameters which are the overhead ratio, running disparity and hardware consumption. The encoder and the decoder of the proposed OR key coding were built in two separate FPGAs. Two FPGA boards with a data rate of 5 Gb/s were set up for the test. They were AC

**TABLE 1. Performance summary and comparison with 8B10B and 64B66B coding.**

| | | Proposed OR Key Coding | | | 8B10B Coding | 64B66B Coding |
|---|---|---|---|---|---|---|
| | | 4-bit Key | 6-bit Key | 8-bit Key | | |
| Maximum Run Length | | 6 bits | 10 bits | 14 bits | 5 bits | 66 bits |
| Overhead ratio | | 16.67% | 3.33% | 0.81% | 25% | 3.13% |
| Coding Method | | Key Coding (Packet) | | | Coding Table | |
| Baseline Wandering [1)] | | 0.020% | 0.024% | 0.030% | 0.012% | 0.136% |
| Running Disparity | | +/-8 | +/-12 | +/-16 | +/-2 | undefined |
| Hardware (FPGA) | LUTs | 72 | 152 | 620 | 46 | 683 |
| | Regist -ers | 14 | 25 | 67 | 11 | 85 |
| Power Consumption @ 125MHz Clock | | 13mW | 15mW | 22mW | 12mW | 23mW |

1) With 100nF DC blocking capacitor, 100 ohms of differential termination.

coupled with a coupling capacitor of 100nF and a differential termination of 100 ohm. A 30cm RG-8 cable was used for the channel. The embedded clock and data recovery blocks in the receiver FPGA were utilized for the test [9]. The bit overhead ratio was reduced to 0.81% in an 8-bit OR key system compared to 16.67% in a 4-bit key and 3.33% in a 6-bit key. However, in an 8-bit OR key system the running disparity and key selector block was larger.

The block diagram of a 6-bit OR key coding transmitter and receiver implemented in the FPGA are given in Fig. 5(a) and Fig. 5 (b), respectively. To show the capability of a 5Gb/s serial interface, the blocks were processed at 125MHz in parallel. After random bit generation, parallel 40 bits per clock were provided to the encoder block to be operated in 125MHz clock. For 6-bit OR key code system, 30 × 6 bits (180 bits) are to be processed for a single packet frame. The implemented 6-bit OR key encoder block works on every 360 bits (40 bits × 9 clock cycles) for generating two packets (180 bits × 2). Then, the encoder block generates packets, and each packet has 186 bits (6-bit OR key + encoded 180 bits). After packet framing, the bit-width control is added for the following MUXed FIFO and serializer block. As the encoder block provides 120 bits in parallel to MUXed FIFO, every 20 packets (186 × 20 = 3,720 bits) can be processed in 31 clock cycles (120 × 31 = 3,720 bits).

The measured waveforms of the 6-bit OR key encoder/decoder are given in Fig. 5(c). For given input data, the generated OR key (6'h20) and the encoded sub-packets are shown. In the receiver, the OR key is extracted, and the sub-packets are decoded. As shown in Fig. 5(c), the decoded data pattern is the same as the input data pattern. The results also show that every encoded sub-packet has more than one state transition and the maximum run length is 10 bits. During the test, no bit error was detected.

Table 1 summarizes a performance comparison of the proposed 4-, 6-, 8-bit OR key coding with 8B10B coding and 6466B coding. In the proposed OR key designs, the hardware usage is greater than 8B10B due to the OR key selector block for disparity control. When 8B10B coding system is implemented with logic operations described in [1], the hardware consumption is the minimum. The bit overhead

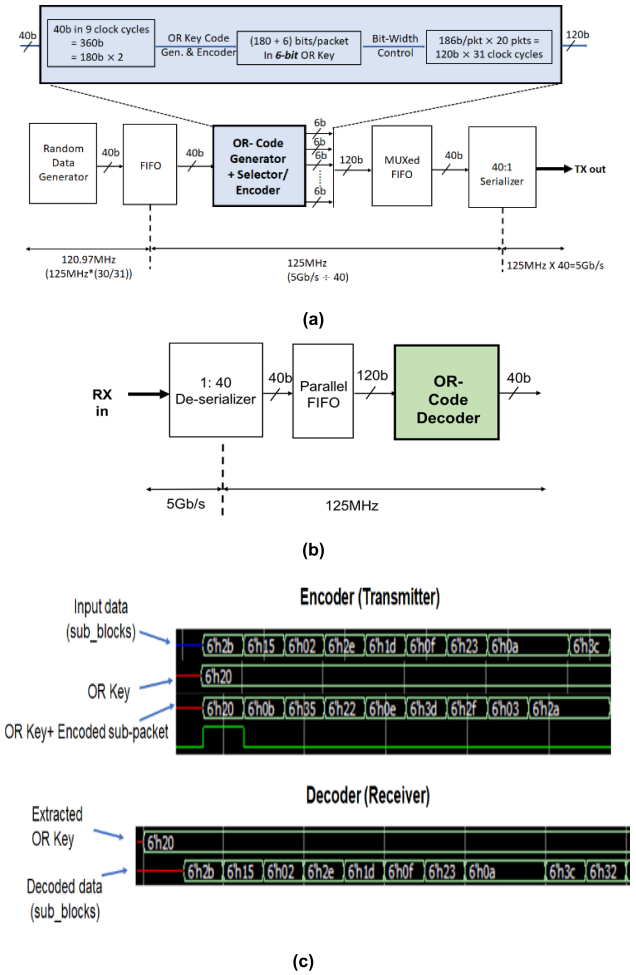


**FIGURE 5. (a) Transmitter, (b) Receiver, (c) and Waveforms from the FPGA of the proposed 6-bit OR key encoder/decoder.**

is reduced to 3.33% from 25% of 8B10B. The 64B66B code has a 3.13% overhead, but a high baseline wondering due to an undefined high running disparity. The maximum run length (10 bits) of the 6-bit OR key coding technique is longer than 8B10B. Because in most high-speed interfaces AC coupling is the major interconnecting scheme between transmitter and the receiver, the DC balance (i.e., running disparity) is another important performance metric in an AC coupled interface. The 8B/10B coding technique is designed to have +/−2 running disparity, which is suitable for an AC coupled interface without using a heavy DC blocking capacitor. The running disparity of +/−12 for a 6-bit OR key system was achieved while minimizing the running disparity using OR key selector logic. The baseline wandering due to the running disparity can be attenuated by increasing the size of the DC blocking capacitor. When used with a 100nF DC blocking capacitor, the baseline wandering of a 6-bit OR key coding is 0.024% compared to 0.012% with 8B10B coding. If the interface needs tighter control of the baseline wandering, a circuit to correct baseline wandering should be added. The logic gate count of the OR key code is higher than 8B10B because of the OR key selector block.

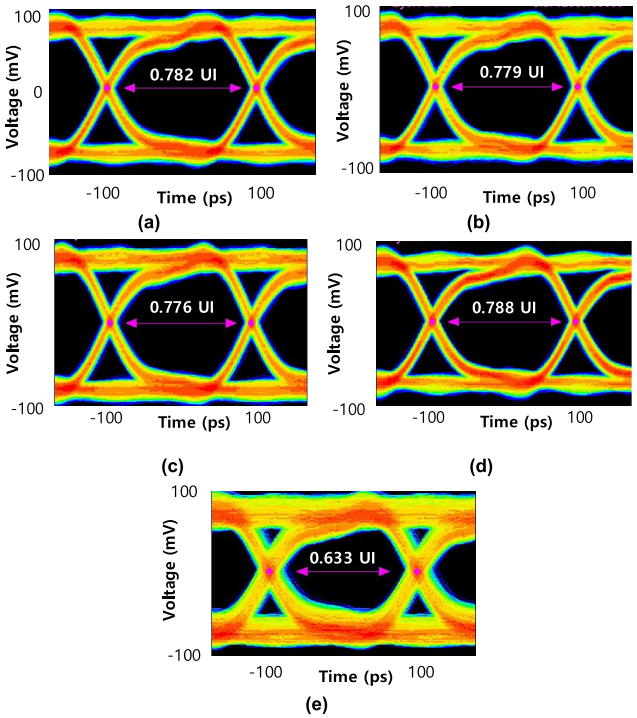


**FIGURE 6.** Measured eye-diagram at the receiver (a) 4-bit OR key coding, (b) 6-bit OR key coding, (c) 8-bit OR key coding, (d) 8B10B coding, and (e) 64B66B coding.

**TABLE 2.** Measured total jitter at receiver (5Gb/s data rate).

| | Proposed OR Key Coding | | | 8B10B Coding | 64B66B Coding |
|---|---|---|---|---|---|
| | 4-bit Key | 6-bit Key | 8-bit Key | | |
| Total Jitter @ BER $10^{-12}$ | 0.2181 UI | 0.2211 UI | 0.2240 UI | 0.2117 UI | 0.3667 UI |

The proposed work is focused on reducing the overhead ratio with a minimal cost of running disparity and hardware consumption.

Fig. 6 shows the measured eye-diagrams in five different coding cases measured at the receiver. As shown, 64B66B coding has a narrower eye than with the OR key coding technique and 8B10B coding due to the large, undefined running disparity.

Table 2 summarizes the total jitter measurement results with measured BER of $10^{-12}$. The proposed OR key coding shows almost the same total jitter as with the conventional 8B10B coding technique. However, 64B66B shows a larger jitter because the running disparity is undefined.

## IV. CONCLUSION

An overhead-reduced key coding technique was proposed. Using the proposed technique, an encoder/decoder with three different bit-size OR key systems were designed. With a 6-bit key coding system design, the OR key is generated in every 30 sub-packets. The overhead is reduced to 3.33% from 25% with 8B10B coding, and the run length is 10bits in a 6-bit key coding system. The OR key code selector was added to maintain the running disparity the minimum in the processing packets, from which the running disparity is controlled within +/−12. By using a DC blocking capacitor of proper size, the baseline wandering of the OR key coding technique can be well controlled with a 100nF DC blocking capacitor.

## ACKNOWLEDGMENT

The authors would like to thank the IDEC Program and for its hardware and software assistance for the design and simulation.

## REFERENCES

[1] A. X. Widmer and P. A. Franaszek, "A DC-balanced, partitioned-block, 8B/10B transmission code," *IBM J. Res. Develop.*, vol. 27, no. 5, pp. 440–451, Sep. 1983.
[2] Y.-W. Kim, B. Shin, and J.-K. Kang, "High-speed 8B/10B encoder design using a simplified coding table," *IEICE Electron. Exp.*, vol. 5, no. 16, pp. 581–585, 2008.
[3] R. C. Walker and R. Dugan, *64b/66b Low-Overhead Coding Proposal for Serial Links*, IEEE 802.3 High Speed Study Group, Jan. 2000, pp. 18–20.
[4] C. S. Thierauf, *High-Speed Cricuit Board Signal Integrity*. Norwood, MA, USA: Artech House, 2004.
[5] Maxim Integrated Products Application Note, *NRZ Bandwidth—LF Cutoff and Baseline Wander*, Maxim Integrated, San Jose, CA, USA, Apr. 2008, pp. 1–7.
[6] Y. Kang, L. W. Chang, Y. C. Wu, and W. T. Chen, "A 5 Gbps/lane intra-panel interface for ultra-high-definition TFT-LCD application," *SID Symp. Dig. Tech. Papers*, vol. 46, no. 1, pp. 1278–1280, Jun. 2015.
[7] H.-Y. Chen, C.-H. Lin, and S.-J. Jou, "DC-balance low-jitter transmission code for 4-PAM signaling," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 9, pp. 827–831, Sep. 2006.
[8] J. T. Stonick, G. Y. Wei, J. L. Sonntag, and D. K. Weinlader, "An adaptive PAM-4 5-Gb/s backplane transceiver in 0.25-$\mu$m CMOS," *IEEE J. Solid-State Circuits*, vol. 38, no. 3, pp. 436–443, Mar. 2003.
[9] A. Athvale, "*High-Speed Serial IO Made Simple Designer's Guide With FPGA Applications*, Connectivity Solution Edition 1.0 Xilinx, San Jose, CA, USA, 2021.