IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 10, 2020, accepted April 1, 2020, date of publication April 8, 2020, date of current version April 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2986510

USB Transceiver With a Serial Interface Engine and FIFO Queue for Efficient FPGA-to-FPGA Communication

GUO-MING SUNG[1,2], (Member, IEEE), LI-FEN TUNG[1], HSIN-KWANG WANG[1,3], AND JHIH-HAO LIN[1]

[1]Department of Electrical Engineering, National Taipei University of Technology, Taipei 10608, Taiwan
[2]Research and Development Centre for Smart Textile Technology, National Taipei University of Technology, Taipei 10608, Taiwan
[3]Bepro Digital Audio Ltd., Taipei 22180, Taiwan

Corresponding author: Guo-Ming Sung (gmsung@ntut.edu.tw)

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Contract MOST 108-2221-E-027-092, and in part by the Chip Implementation Center, Taiwan, for Fabricating the Test Chip.

ABSTRACT This paper presents a universal serial bus (USB) transceiver with a serial interface engine (SIE) and an asynchronous first-in first-out (FIFO) queue for packet transformation and data transmission in field-programmable gate array (FPGA)-to-FPGA communication. The SIE block receives the data to be transmitted from the central processing unit of a PC and transfers those data to the universal transceiver macrocell interface, which handles data serialization and deserialization, bit stuffing, clock recovery, and clock synchronization. An asynchronous FIFO queue of 2 kilobits is designed to guarantee correct communication between two FPGA development boards. A parallel-in serial-out block converts parallel input data into serial data. A product identification (PID) check block determines whether the serial data are in the USB packet format. The cyclic redundancy check (CRC) checksums, namely CRC5 and CRC16, are presented with data check statements. After passing through the NRZI decoder, bit-unstuffing, PID check, and CRC16 blocks, the received serial data are converted into parallel output data by using a serial-in parallel-out block. The FPGA-to-FPGA communication design operates correctly. An application-specific integrated circuit (ASIC) of the USB transceiver is implemented using TSMC 0.18-μm CMOS technology. The gate counts, power consumption, operating frequency, and chip area of the ASIC are 14,547, 2.6742 mW, 50 MHz, and 0.7×0.67 mm^2, respectively, at a supply voltage of 1.8 V and total pin number of 38.

INDEX TERMS Universal serial bus (USB), first-in first-out (FIFO), serial interface engine (SIE), field-programmable gate array (FPGA), application-specific integrated circuit (ASIC), USB physical layer (PHY), cyclic redundancy check (CRC), CMOS process.

I. INTRODUCTION

The universal serial bus (USB) interface is an interface for data transmission over the Ethernet. USB is not only convenient to use but also transmits data rapidly [1]. The field-programmable gate array (FPGA) implementation of a USB transceiver macrocell interface (UTMI) with a half-speed (HS) or full-speed (FS) transmission rate and USB 2.0 specifications is described in [2]. A vendor can easily select the transceiver that meets their requirements [2]. In [3], simple interface protocols, such as USB and RS485, were used for achieving communication between PCs to develop a packet bridge between an Ethernet network and a Synchronous Optical Network (SONET) in a smart sensor system. The USB interface is preferred due to its power capacity. A payment device can be powered from the USB interface to reduce the overhead size [4]. In [4], an initial implementation of the USB interface was realized to exclusively present a protection method and to evaluate the performance of the interface. Prospective implementation can be realized using a test platform with an FPGA development board [5].

USB interfaces are generally present in PCs that have been designed by various companies, such as Intel, Compaq, NEC, Digital, Northern, IBM, and Microsoft. The USB Implementers Forum (USB-IF) is a support organization responsible for the advancement and adoption of USB tech-

The associate editor coordinating the review of this manuscript and approving it for publication was Tawfik Al-Hadhrami.

nology. The mission of the USB-IF is to develop compliant USB products as per USB specifications [6]. A USB can configure the address and implement the phase to automatically achieve hot swapping or hot plugging. USB is convenient for connecting electronic devices to a computer without stopping or shutting down the system, and it provides considerable business opportunities. An embedded converter that converts RS232 to USB is convenient for achieving the advantages of USB without making major changes to a system that relies on RS232. A first-in first-out (FIFO) logic interface was developed to bridge the data rate differences between the USB and RS232 protocols [7], [8]. That FIFO logic interface is also a useful guide for implementing other USB devices.

Studying the implementation of the USB standard with an FPGA-based development board is crucial. An FPGA-based development board is used to connect a peripheral device to a computer through a USB cable. The USB interface comprises two units—the UTMI and parallel interface engine (PIE) [9]. The UTMI can connect to USB cables and can be used for time frame synchronization, and serial data transmission. PIE is responsible for packet construction or extraction and communication with the peripheral device. The aforementioned modules were designed using a finite state machine and implemented using Verilog hardware description language (HDL), which is synthesized using Xilinx ISE tools [10], [11]. However, a low-cost programmable logic device has a low speed. A fast version of the USB is always in demand because the communication speed of the FPGA circuit is rapidly increasing [9]. The plug and play operation of a peripheral interface is the primary target. Moreover, FPGA circuits have important application value for realizing automatic detection by using USB 2.0. The USB circuit transfers a digital image, which is stored in a FIFO queue, to a computer for image processing. The FPGA-based USB interface has high detection efficiency and accuracy in an aircraft engine blade detection system [12].

A general schematic of the serial interface engine (SIE) block is displayed in Fig. 1 [13]. The SIE block can be divided into two types of sub-blocks—the SIE control logic and endpoint logic sub-blocks. The SIE control logic sub-block contains USB product identification (PID) logic, address recognition logic, and other sequencing logic for managing USB packets and transactions. The endpoint logic sub-block contains the endpoint number recognition, FIFO, and FIFO control logic. The SIE block receives data from the central processing unit of a PC and sends the transferred data to the UTMI [13], [14]. The UTMI block handles the low-level USB protocol and data signing. The UTMI block performs tasks such as data serialization, data deserialization, bit stuffing, clock recovery, and clock synchronization. The primary task of the UTMI is to transfer data from a USB to another USB compatible with the physical logic, such as a differential signal—D+ and D−. The SIE module can be developed using peripheral vendors or purchased from intellectual property (IP) vendors. The standardization of the UTMI allows compatible SIE HDL to be integrated into an ASIC with a macrocell.

In this study, a USB transceiver ASIC was implemented with an SIE and a FIFO queue by using TSMC 0.18-μm CMOS technology. The integration technology of the proposed ASIC can shorten the delay times and improve the data transportation speed. Simulations and measurements were performed to verify the correctness and flexibility of the proposed USB transceiver. The aim of this study was to guarantee correct communication between two FPGA development boards. Section II describes the circuit design of the USB transceiver with an SIE and a FIFO queue. The simulation results and FPGA verification are presented in Section III, and the conclusion is presented in Section IV.

II. CIRCUIT DESIGN OF A USB TRANSCEIVER WITH AN SIE AND A FIFO QUEUE

Figure 2 displays the system architecture of a USB transceiver with an SIE and a FIFO queue for efficient FPGA-to-FPGA communication. The transceiver comprises two Altera FPGA development boards (DE2-70) with a general purpose input or output (GPIO) port connection, two data read-only memory (ROM) modules, and two external USB port physical layers (PHYs) [11]. One FPGA development board is used

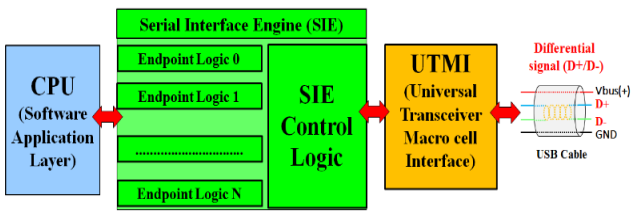

FIGURE 1. General schematic of the SIE block.

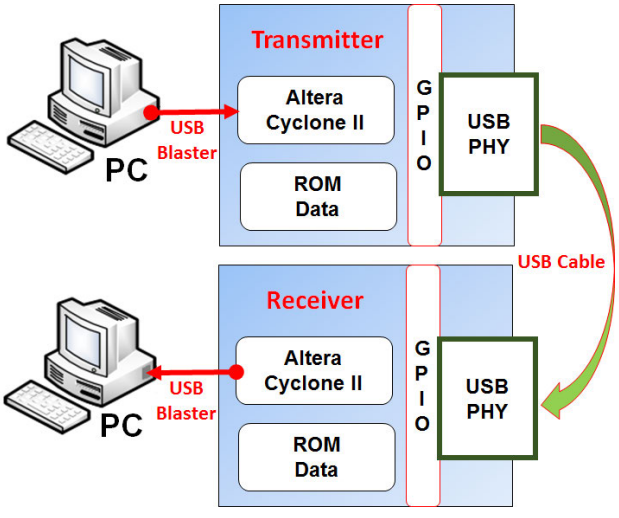

FIGURE 2. System architecture of the USB transceiver with an SIE and a FIFO queue for efficient FPGA-to-FPGA communication.

as the transmitter, and the other is used as the receiver. The SIE deals with the data transmitted between the ROM and the PHY and is programmed in Verilog HDL. The FPGA development board is used to verify the proposed function through a USB blaster, which is connected to a PC. Thus, data transmission can be completed between a transmitter and a receiver after the proposed function is verified. The transmitter reformats the received data into a serial code by using the SIE block and sends the code to the following receiver via a USB cable. The proposed SIE block should guarantee the correctness of data transportation through a USB PHY.

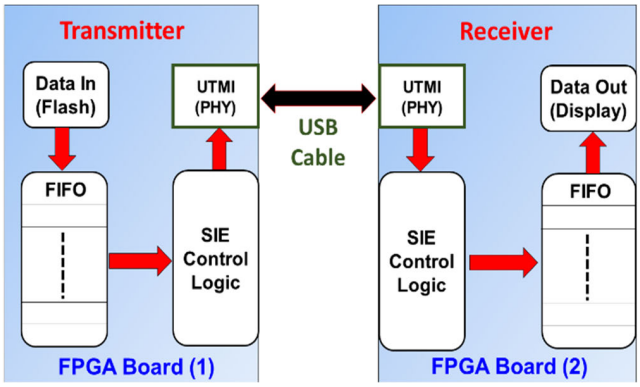

FIGURE 3. System topology of the USB transceiver with an SIE block and a FIFO queue.

Figure 3 presents the system topology of the USB transceiver with an SIE block and a FIFO queue. In the first FPGA board (Transmitter), the input data are stored in a flash memory and then sent to an asynchronous FIFO queue. The SIE block, which includes the SIE control logic and UTMI, transforms the input data into a serial packet according to the USB standard and sends it to the next FPGA board through a USB cable. The second FPGA board (receiver) receives the serial code packet from the USB cable and reassembles the code into the desired data format by using the proposed SIE block. After passing through the FIFO queue, the received data can be displayed on a seven-segmented display on the second FPGA board. If the output data, which are displayed on the second FPGA board, are the same as the input data, which are stored in the flash memory on the first FPGA board, the proposed transceiver operates accurately. If not, the source code must be rewritten in Verilog HDL until the designed function performs correctly. The regular operating frequency of the FIFO queue was 50 MHz; however, the frequency of the FIFO queue for the USB PHY was 26 MHz, which was realized using the CY7C68003 commercial chip (Cypress Semiconductor Corporation, San Jose, CA, USA).

Figure 4 illustrates the operating frequencies in the relative blocks of the USB transceiver. The FPGA board provides an operating frequency of 50 MHz, which is used in the flash memory (ROM) and FIFO module. By passing through the phase-locked loop (PLL) IP and frequency divider circuit,

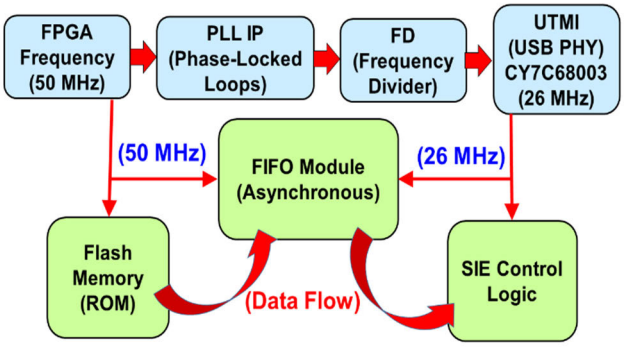

FIGURE 4. Operating frequencies in the relative blocks of the USB transceiver.

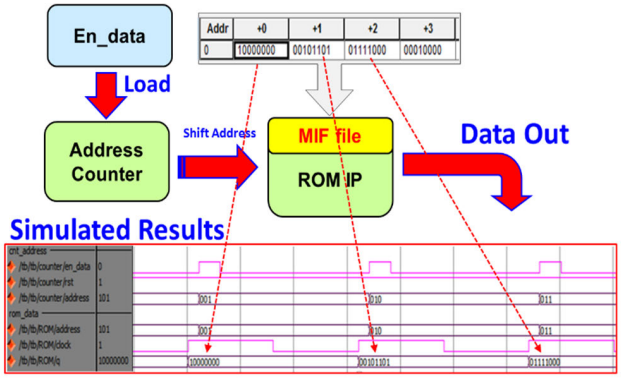

FIGURE 5. Schematic of the ROM data module.

an operating frequency of 26 MHz can be obtained. The down-converted clock frequency of 26 MHz is used in the SIE control logic, UTMI (USB PHY), and FIFO module. The asynchronous FIFO module accepts both the operating frequencies (i.e., 50 and 26 MHz). The flash memory sends the received data to the FIFO module at 50 MHz, and the FIFO module transmits the stored data to the SIE control logic at 26 MHz.

A. ROM DATA MODULE

As depicted in Fig. 2, the ROM data module provides the address to store the transmitted data. Figure 5 displays the schematic of the ROM data module. This module includes the En_data, address counter, and ROM IP submodules. The dominant function of the ROM IP submodule is to store the transmitted data in a memory initialization file (MIF). The address counter submodule provides an address control signal to specify the address of the ROM data. The En_data submodule enables data transmission and counts the ROM operation times in the address counter submodule. A simulation is conducted to guarantee that the ROM data module operates normally. As displayed in Fig. 5, the data points 10000000, 00101101, and 01111000 can be displayed correctly in the MIF file with the addresses +0, +1, and +2 in the simulation results, respectively.

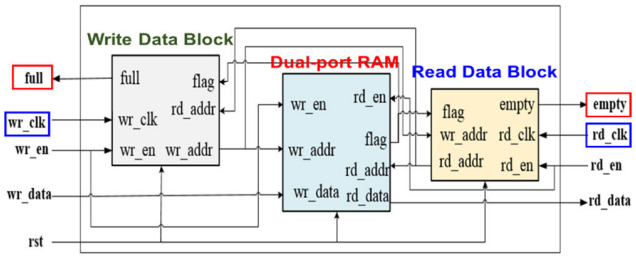

FIGURE 6. Schematic of the asynchronous FIFO module.

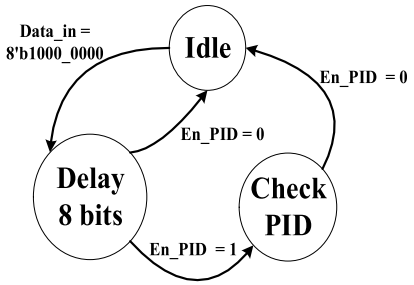

FIGURE 8. State diagram of the PID check block.

B. ASYNCHRONOUS FIFO MODULE

Figure 6 displays the schematic of the asynchronous FIFO module used in this study. An asynchronous FIFO module was adopted in this study because the read clock rd_clk is different from the write clock *wr_clk*. The proposed asynchronous FIFO module includes three submodules. The middle submodule is a dual-port RAM for accessing the write data (wr_data) and write address (wr_addr). The left submodule controls the write address (wr_addr) by using the write enable signal (wr_en) and read address (rd_addr). The right submodule is a read data block, which primarily controls the read enable signal (rd_en), read address (rd_addr), and null signal (empty). The full signal (full) is obtained in the left submodule by comparing the read address (rd_addr), whereas the null signal (empty) is obtained in the right submodule by comparing the write address (wr_addr).

C. SIE MODULE

The system architecture of the proposed SIE module is illustrated in Fig. 7. The input data are input to the transmitter end. Packet format analysis and packet classification should be conducted in advance for the incoming data. After the aforementioned processing is completed, the input data are sent to the receiver end. The receiver end is responsible for verifying whether the received data are correct. If the received data are correct, the data are sent to the UTMI (PHY). If the data are incorrect, the received data are disposed. As displayed in Fig. 7, the proposed SIE module can be divided into many blocks: the data control, parallel-in serial-out (PISO, PID check, cyclic redundancy check 5 (CRC5), CRC16, bit-stuffing, NRZI encoder, NRZI decoder, bit-unstuffing, and serial-in parallel-out (SIPO) blocks [15].

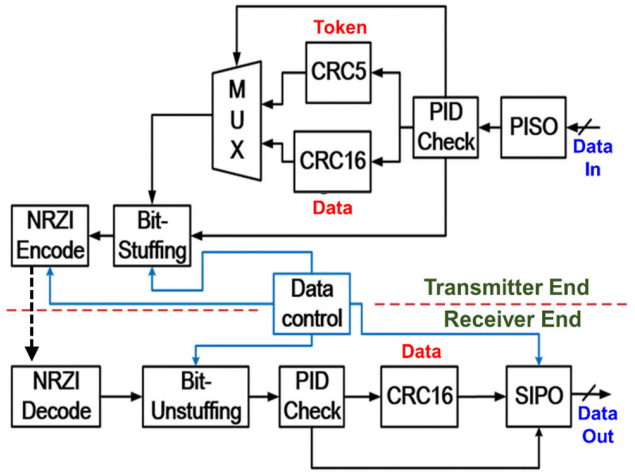

FIGURE 7. System architecture of the proposed SIE module.

D. PID CHECK BLOCK WITH THE PISO FUNCTION

The data control block controls the data processing function by providing the enable signal to blocks such as the bit-stuffing, NRZI encoder, bit-unstuffing, and SIPO blocks. If these blocks do not receive the enable signal, they would be in an idle state without action. In the PISO block, parallel communication is transformed into serial communication. Initially, the input parallel datum is stored in a register. The stored datum is sent to the next block (PID check) bit by bit until the register is empty. The least significant bit (LSB) is transmitted first. In general, the PISO block is always integrated with the PID check block.

Figure 8 presents the state diagram of the PID check block. If the input datum is 8'b1000_0000, an 8-bit delay is required to complete the PISO function. Then, the PID check is initiated with an enable signal En_PID that has a "high" value (1). After the PID check is completed, the signal En_PID becomes low (0) and the PID check block is idle. Figure 9 illustrates the functional flowchart of the PID check block. First, the 8-bit input data (Data_in) are stored in a register with the symbol Data_out[7:0]. If the enable signal of the PID check block is high (i.e., En_PID = 1), the high half-byte (i.e., Data_out[7:4]) is compared with the 1's complementary of low half-byte (i.e., ~Data_out[3:0]). If the high and low half-bytes are the same, packet analysis and classification are conducted using the lookup table, which is established in advance. If the two half-bytes are not the same, the error signal is set to high (Error = 1) and the input data are discarded.

E. CRC5 AND CRC16 BLOCKS

The CRC was proposed by Peterson and Brown [16]. The CRC calculation is only applied to a predetermined serial data field. In general, CRC5 is used to check the token field, whereas CRC16 is used to check the input data field.

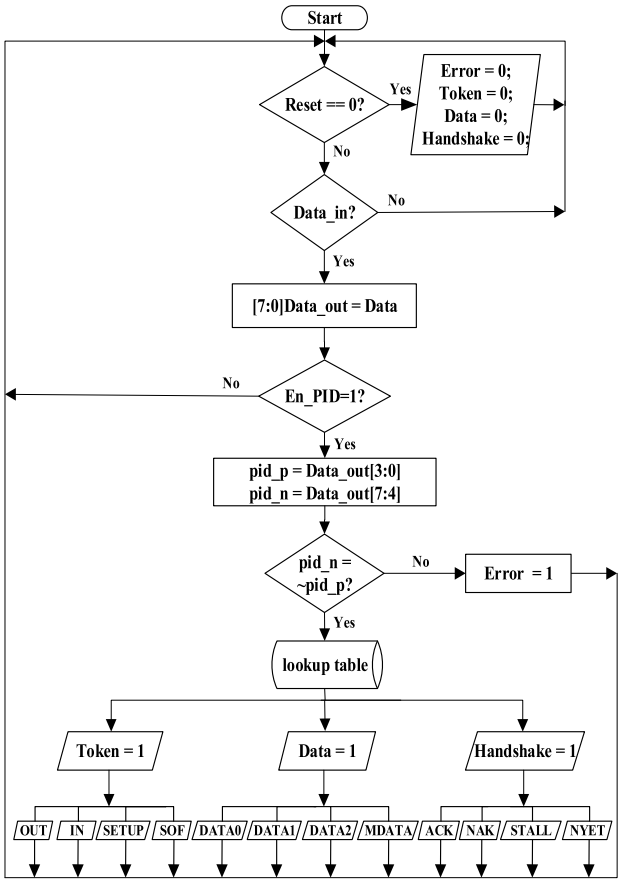

FIGURE 9. Functional flowchart of the PID check block.

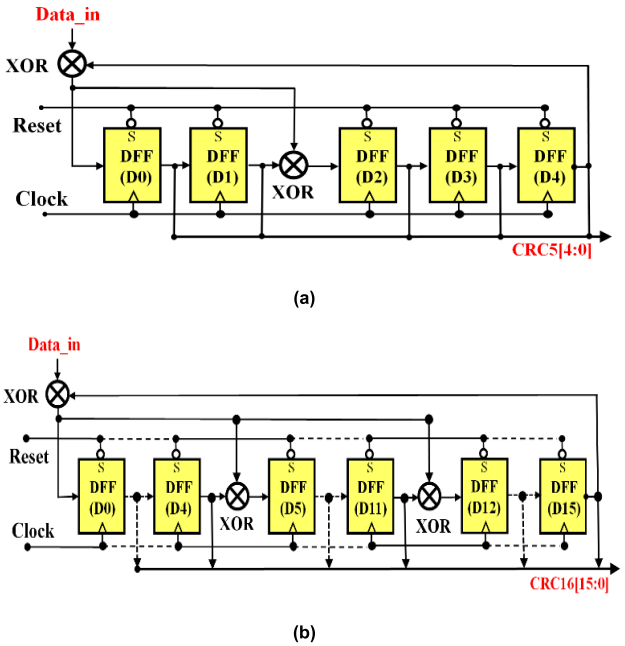

FIGURE 10. General structure of the CRC block: (a) CRC5: $x^5 + x^2 + 1$ block and (b) CRC16: $x^{16} + x^{15} + x^2 + 1$ block.

CRC5 calculation is applied only to all the USB address and endpoint data fields except the SYNC, USB command, and CRC data fields. A hash function is used to reduce the data size by mapping data with an arbitrary size to data with a fixed size. The hash value, which is returned by the hash function, is a word string that comprises random characters and numbers. Figure 10 displays the general structure of the CRC block, where the polynomial functions of the CRC5 and CRC16 blocks are $x^5 + x^2 + 1$ and $x^{16} + x^{15} + x^2 + 1$, respectively. The check procedure is completed bit by bit. First, an exclusive OR (XOR) calculation is conducted using the input data (Data_in) and the most significant bit (MSB), which is 0 for CRC5 [6] and 1 for CRC16 [16] in the initial state. Second, 5 and 16 D-type flip-flops (DFFs) are used to generate the output data CRC5[4:0] and CRC16[15:0], respectively, by delaying 5 and 16 clocks, respectively.

F. BIT-STUFFING AND BIT-UNSTUFFING BLOCKS

A long series of no-transition bits can be difficult for a receiver to count accurately; thus, some methods are generally used for forcing a transition at reasonable intervals. A USB uses bit stuffing by inserting an additional bit of "0" after six consecutive bits of "1." The bit-stuffing block is designed to block input data by holding the data at a high-level state for a long time. The highest value of 6-bit

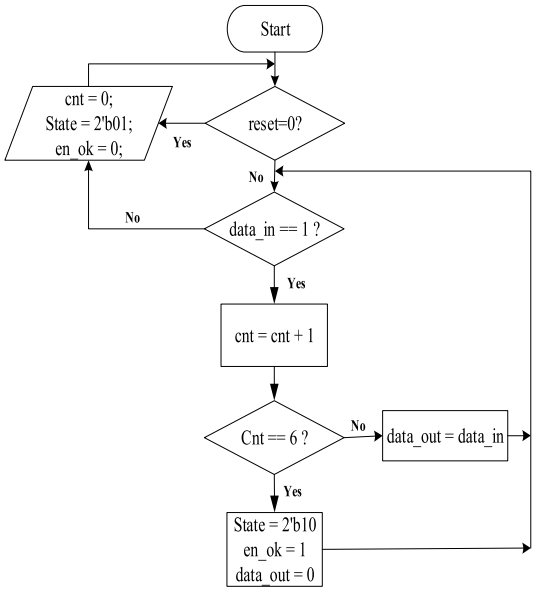

FIGURE 11. Functional flowchart of the bit-stuffing block.

data (111111) causes clock disorder in a transceiver. The bit-stuffing block in this study is designed to overcome this fault. Figure 11 presents the functional flowchart of the bit-stuffing block used in this study. First, the reset signal is set to zero, and the initial state is status 0, which is numbered as "01." In this initial state, the counter value (cnt) is zero, and the enable signal (en_ok) is zero. Second, if the input data (data_in) are high (1), the output data (data_out) are the same as the input data and the counter is incremented. Because the counter value is equal to 6, the state is changed

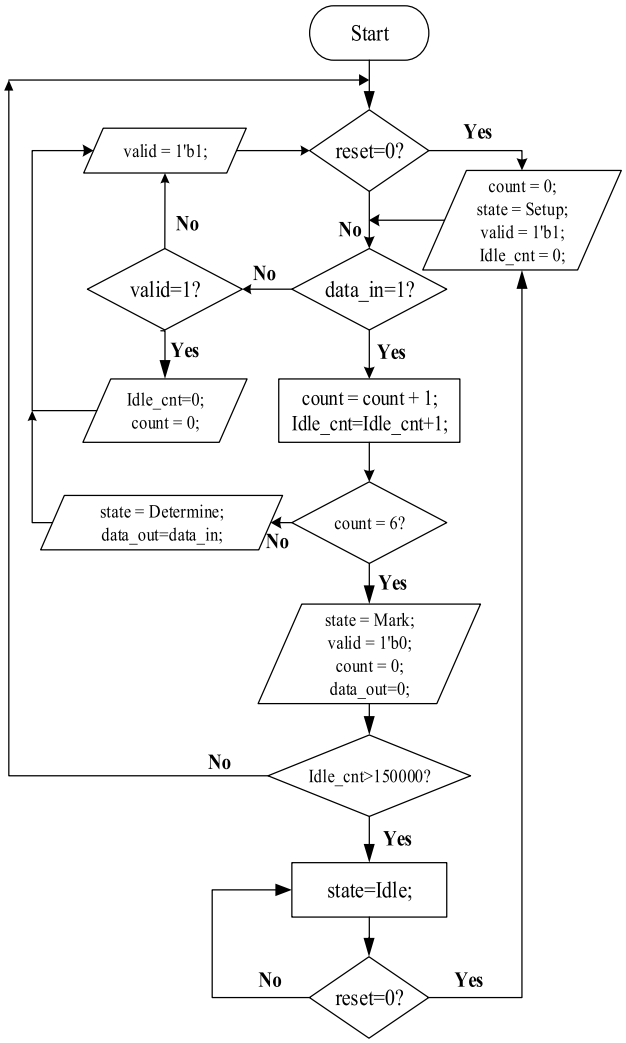

FIGURE 12. Functional flowchart of the bit-unstuffing block with an idle detection.

to status 1, which is numbered as "10." Then, the enable signal is changed to high (1) and the output data are clear to empty (0). In other words, the output data are changed to low (0) after six consecutive bits of "1" (111111). The aforementioned procedure is continued at status 1 but is terminated at status 0.

The bit-unstuffing block is designed to remove the additional zero bit (0), which is added in the previous bit-stuffing block to resolve the clock disorder. Figure 12 displays the functional flowchart of the bit-unstuffing block. First, the reset signal is set to zero (reset = 0), and the initial state is "Setup." Second, the count value is zero (count = 0), count value of the idle state is zero (Idle_cnt = 0), and valid bit is high (valid = 1). Third, if the input data (data_in) are high, the count and idle values are incremented. Because the count value is equal to 6, the state is changed to "Mark." When the valid bit is marked as 0, the bit is considered to be invalid, and the count value is set to 0. Because the count value is less than six, the state is changed to "Determine," and the bit is valid. Conversely, the input data are discarded when the input data are zero (data_in = 0) and the bit is invalid. If the input data are zero and the bit is valid, the count and idle values are cleared. A state diagram for idle time detection is displayed in Fig. 13. The state diagram is used to avoid long-term high levels, such as 3 ms. If the clock frequency is 50 MHz, 150,000 clock pulses are completed in 3 ms. Because the number of idle clock pulses is more than 150,000, a reset signal is set to zero (rest = 0) and the initial state is "Setup." The bit-unstuffing function is then restarted. The bit-stuffing function causes a variable data rate because it requires a marginally longer time to send a long string of "1" bits than to send a long string of "0" bits.

G. NRZI ENCODER AND DECODER BLOCKS

NRZI is a method of mapping a binary signal to a physical signal for transmission [17]. A logical 1 is transmitted for a transition, and a logical 0 is transmitted for no transition. In the NRZI encoder, the output signal is transmitted if the input signal (data_in) is logical 0. Conversely, if the input signal is logical 1, no transition occurs in the output signal. In the NRZI decoder, the output signal is logical 0 when a transition occurs in the input signal. Conversely, the output signal is logical 1 when no transition occurs in the input signal. The DFF circuit can be used to overcome the problems associated with glitch waves in the NRZI encoder and decoder blocks.

III. SIMULATION RESULTS AND FPGA VERIFICATION

As displayed in Fig. 7, the PISO block is integrated with the PID check block. If the incoming data point is 8'b1000_0000 (SYNC field), a PID check is initiated to check the packet type by using the control signal En_PID. When En_PID returns to the low level (0), the PID check block is idle. In other words, the SIE module enters the suspended state whenever the received data are continuously at a high level. In the suspended state, the valid data control line becomes low (0), and the received data are unchanged. When the full flag appears and the reset signal turns high (1), the SIE module restarts and continues the data processing steps. When the transmitted data are valid, the SIE module operates in the idle state without receiving the next input data point. The PISO block converts the parallel input data into serial data, and the PID check block determines whether the serial data are in accordance with the USB packet format. If the serial data are certificate packets (token), the CRC5 check is conducted; otherwise, the CRC16 checksum is presented with the information packet (data). Thus, data are passed through the bit-stuffing and NRZI encoder blocks after passing through the multiplexer [18]. After the aforementioned checks are completed, the serial data are sent to the receiver end. The received data are passed through the NRZI decoder, bit-unstuffing, PID check, and CRC16 blocks. Then, the SIPO block converts the received serial data into parallel data and outputs the data (Data_out).

The data transmission function from a computer to the FPGA development board should be confirmed.

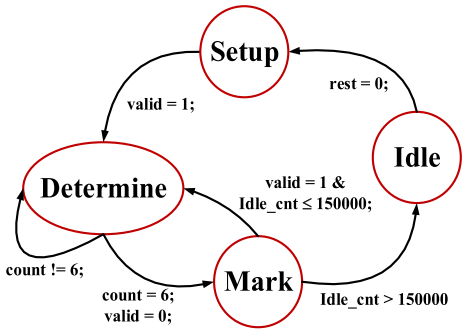

FIGURE 13. State diagram of the idle detection. Because the idle time is larger than 3 ms, a reset signal is required to restart the bit-unstuffing function.

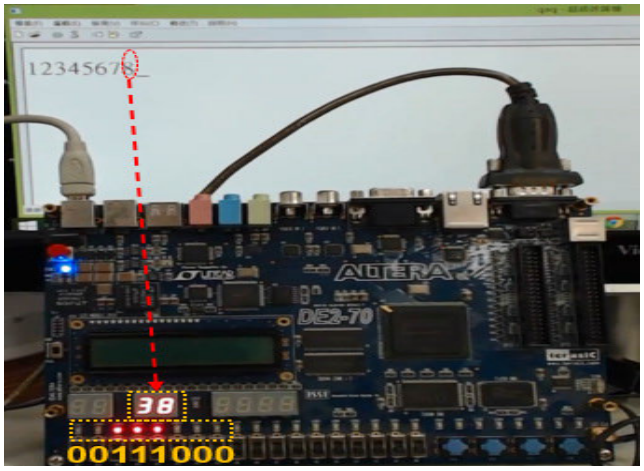

FIGURE 14. Transmitted data of the USB blaster.

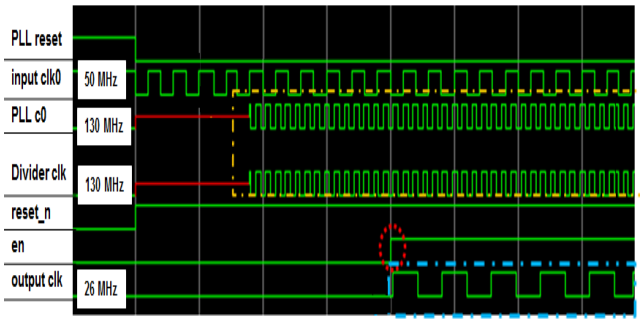

FIGURE 15. Simulated waveforms for the clock frequencies of 50, 130, and 26 MHz. A clock frequency of 26 MHz is input into the USB PHY when the enable signal has a high level (1).

Address	0000	0001	0010	0011
Memory Data	3f	4d	55	66

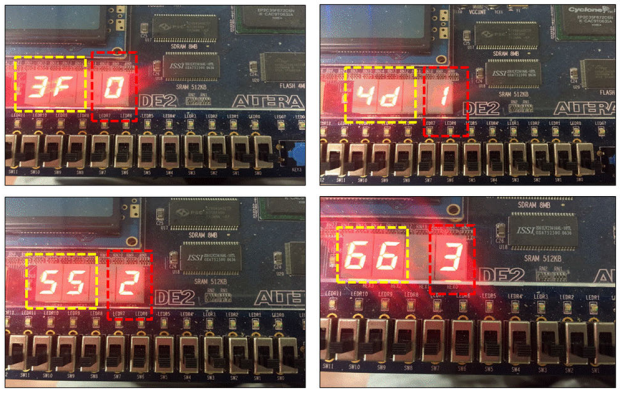

FIGURE 16. Experimental results for the address and memory data on the FPGA development board.

A test number, 8, is sent to the FPGA board from the computer through a USB blaster. Figure 14 displays the data transmission function of the USB blaster. The number 38, which corresponds to the American Standard Code for Information Interchange (ASCII) code of 8, is displayed on the FPGA board. An LED array displays the same ASCII code with the binary code "00111000," which corresponds to the number 38. Both the seven-segmented display and LED array verify that the data transmission from the computer to the FPGA board is correct. Thus, input data can be correctly transferred from a computer to the FPGA board.

To increase the speed of the proposed procedure, numerous IPs can be taken from Quartus II software (Altera Corporation, San Jose, CA, USA). A PLL IP is used to generate an operating frequency of 26 MHz for the USB PHY. Originally, a clock frequency of 50 MHz is provided by the FPGA board. After data are passed through the PLL IP, a synchronous clock frequency of 130 MHz is obtained. Subsequently, the frequency of 130 MHz is divided by 5, which is a clock divider. Thus, a frequency of 26 MHz is obtained. Figure 15 displays the simulated waveforms for the clock frequencies of 50, 130, and 26 MHz. A clock frequency of 26 MHz is input into the USB PHY when the enable signal has a high level (1).

Subsequently, the FPGA board inputs the address and data into the ROM module. Figure 16 displays the experimental results for the address and memory data on the FPGA board. The memory data are 3F, 4d, 55, and 66, which correspond to the addresses 0, 1, 2, and 3, respectively. The experimental results are correct. Furthermore, an asynchronous FIFO block is used to complete the queue function with two clock frequencies: wr_clk and rd_clk. Figure 17 displays the simulation results of the asynchronous FIFO block, which includes the write, memory, and read parts. Because the write enable signal (wr_en) and read enable signal (rd_en) are high, the data write and read operations are completed using the same addresses, due to which the same data are presented at both clock frequencies. When the full or empty signal is high, the enable signal is low not only in the write part but also in the read part.

Figure 18 displays the simulation results of the PISO and SIPO blocks. When a parallel input word "00101101" is input into the PISO block, the LSB "1" is output first and the MSB "0" is output last. Then, the serial data are input into the SIPO block, and an output word "00101101" is finally reconstructed. The input word of the PISO block is the same

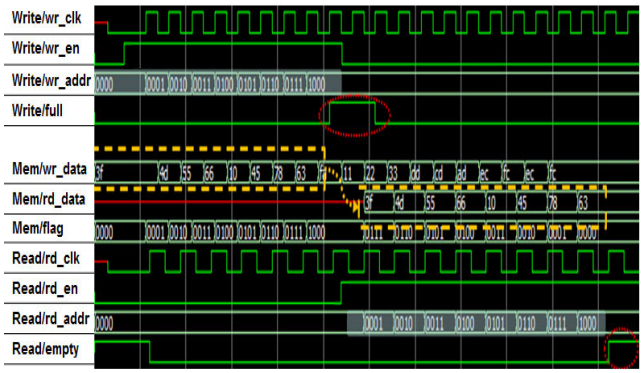

FIGURE 17. Simulation results for the asynchronous FIFO block.

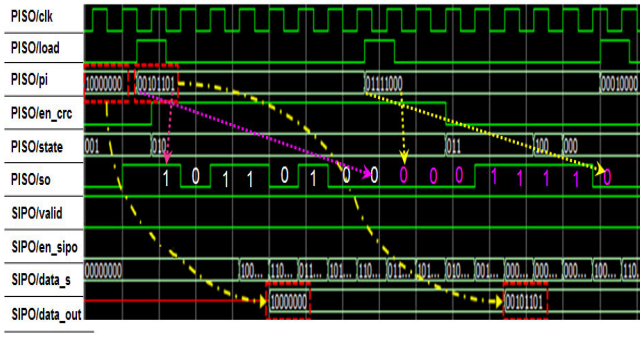

FIGURE 18. Simulation results for the PISO and SIPO blocks.

as the output word of the SIPO block. The simulation results verified that the PISO and SIPO blocks operated successfully.

Figure 19 displays the simulation results of the PID check block. As presented in Fig. 19, the data point "10010110" is displayed if the reset signal (tb/rst) is high (1). Then, the two check signals, pid_p and pid_n, are assigned to "1001" and "0110," respectively. If the 1's complement of pid_p is equal to pid_n, the lookup table provides the following parameters: the token, data, and handshake packets. The term pid_p[0:1] indicates the type of packet. For example, "01," "11," "10," and "00" represent the token, data, handshake, and special packets, respectively. Moreover, pid_p[2:3] represents the packet name. For example, "00," "10," "01," and "11" represent OUT, IN, SOF, and SETUP, respectively, for a token packet. Other packet names can be deduced by analogy by considering the names in the data and handshake packets. Thus, "00," "10," "01," and "11" represent DATA0, DATA1, DATA2, and MDATA, respectively, for the data packet and ACK, NAK, STALL, and NYET, respectively, for the handshake packet.

An input binary code "10101010" is considered to verify the functions of CRC5 and CRC16, which are displayed in Fig. 10. For the CRC5 data, the power is 5; the divisor is "100101,"; and the dividend is "1010101000000," which is generated by combing the input code "10101010" and 5-bit zeros ("00000"). If the dividend is divided by a divisor, we obtain a remainder of "11000." Thus, the CRC5 code is "11000." The CRC16 code can be deduced by analogy with the CRC5 code. When the input code is "10101010", the CRC16 code is "0001010010100000."

Figure 20 displays the functions of the bit-stuffing and bit-unstuffing blocks. If the input data are high (1), the count function is initiated. For the bit-stuffing block, an additional zero bit (0) is added to avoid the clock disorder after six consecutive bits with a high value (1). This function is verified in the data_out signal presented in Fig. 20. Because the counter value (cnt_6_high) is 6, the state is changed to status 1, which is numbered as "10." Then, the enable signal (en_ok) is changed to high (1) and the output data signal (data_out) is changed to empty (0) in the next clock. The bit-unstuffing block is designed to remove the additional zero bit (0), which is added in the previous bit-stuffing block to resolve the clock disorder. The input data signal (data_in) is the same as the output signal (data_out) of the bit-stuffing block. If the input data (data_in) are high, then the count value is incremented. Because the count value is 6, the state is changed to "10." When the valid bit is marked as 0, the bit is considered to be invalid, and the count value is set to 0. As presented in Fig. 20, the zero bit (0) in the output data signal (data_out) is removed when the valid bit is zero (0). This implies that the real output data are a string of high values (1); thus, the output data are the same as the input data (data_in) in the bit-stuffing block.

The NRZI blocks are the most important blocks for mapping a binary signal to a physical signal for transmission. In the NRZI encoder block, the output signal (DP) is changed from high (1) to low (0) if the input signal (data_in) is low (0). Conversely, if the input signal (data_in) is high (1), no transition occurs in the output signal (DP). This implies that the output signal (DP) is zero, which is the same as the previous zero value ($0 \rightarrow 0$). In the NRZI decoder block, the output signal (data_out) is low (0) when a transition ($1 \rightarrow 0$ or $0 \rightarrow 1$) occurs in the input signal (DP_in). Moreover, the output signal (data_out) is high (1) if the input signal (DP_in) has no transition ($1 \rightarrow 1$ or $0 \rightarrow 0$). The simulation results are presented in Fig. 21. The output data (data_out) of the NRZI decoder block are maintained to be the same as the input data (data_in) of the NRZI encoder block by delaying the clock by one cycle.

After the designed functions have been verified, an SIE module is established using the proposed NZRI blocks. Figure 22 displays the functional block diagram of the proposed SIE module for register-transfer-level (RTL) simulation. A control block is proposed using many enable signals, including en_nrzi, en_sipo, en_stuf, and en_unstuf, for the NRZI, SIPO, bit-stuffing, and bit-unstuffing blocks, respectively. When the proposed control block was used, the RTL simulation of the SIE module operated correctly. In addition to the control block, an additional input (crc_in[15..0]) is considered to transform the parallel data in the bit-unstuffing block into serial data (data_out). The serial data are then input into the SIPO block.

Figure 23 presents the verification of the designed USB transceiver with an SIE module and a FIFO queue.

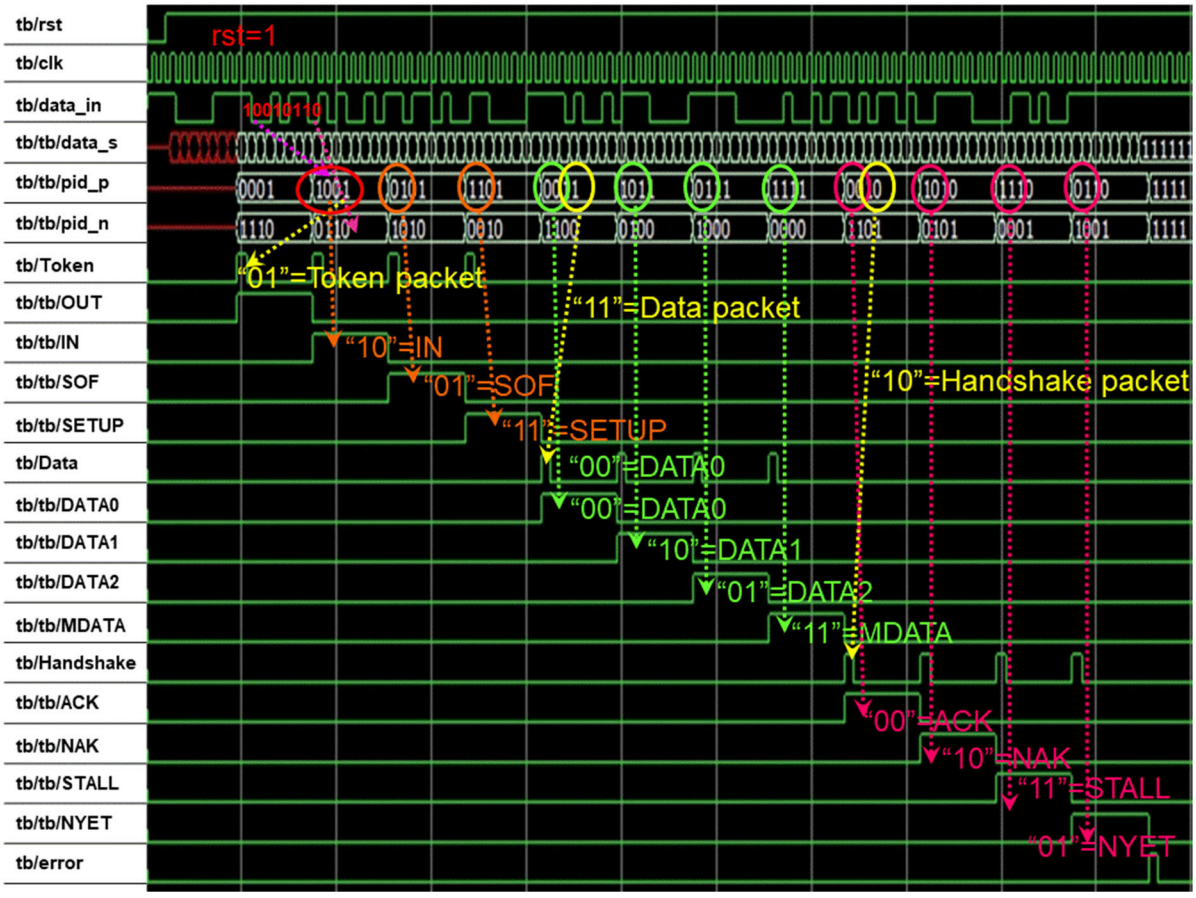

FIGURE 19. Simulation results for the PID check block.

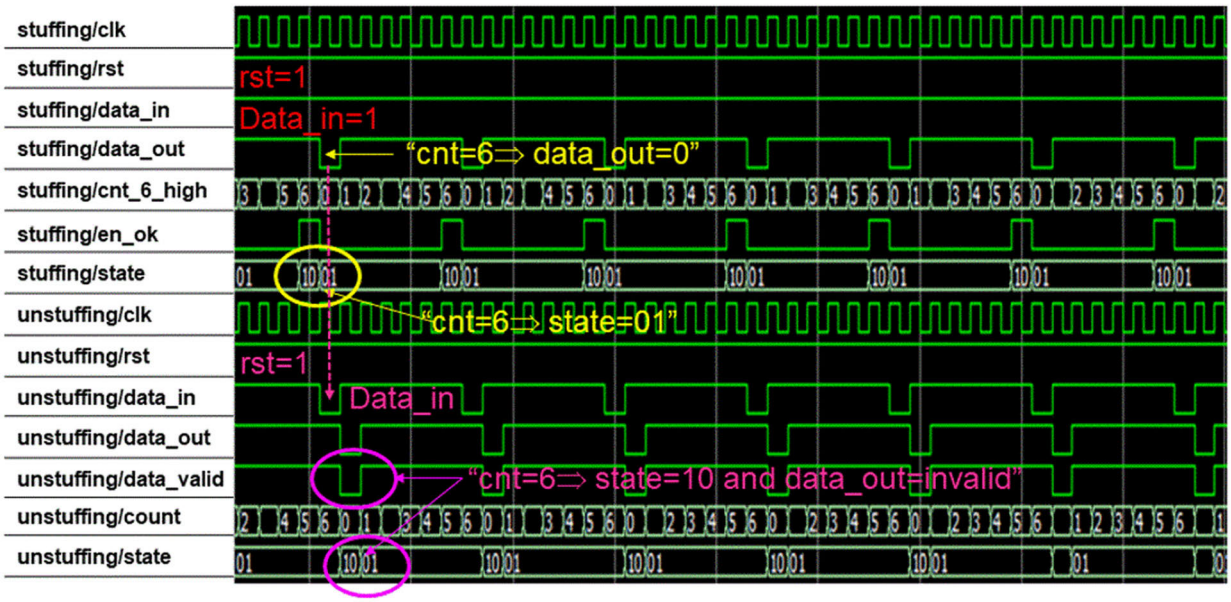

FIGURE 20. Simulation results for the bit-stuffing and bit-unstuffing blocks.

The transceiver includes two Altera FPGA development boards (DE2-70) with a GPIO port connection and two external USB port PHYs. The upper board acts as the transmitter, and the lower board acts as the receiver. A clock divider is used to reduce the clock frequency for visually examining the correctness of data transmission. The transmitted data of

TABLE 1. Performance Summary and Difference with Other USB Transceivers.

Parameters	[2] (2008)	[19] (2016)	[20] (2018)	This Work
FPGA Chip	Xilinx SPARTAN II	Xilinx Vivado	Xilinx Artix-7	Altera Cyclone II
Operating Frequencies	50 MHz/200 MHz	60 MHz/480 MHz	318.8 MHz/200 MHz	50 MHz/130 MHz
Gate Counts	1,344	52,932	91,020	14,547
Data Rate (Mbps)	1.5/12/480	1.5/12/480	1.5/60/600	1.5/12/480
On-Chip Power (mW)	–	370	495	204.69
ASIC Power (mW)	–	–	–	2.6742
ASIC area (mm^2)	–	–	–	0.7×0.67

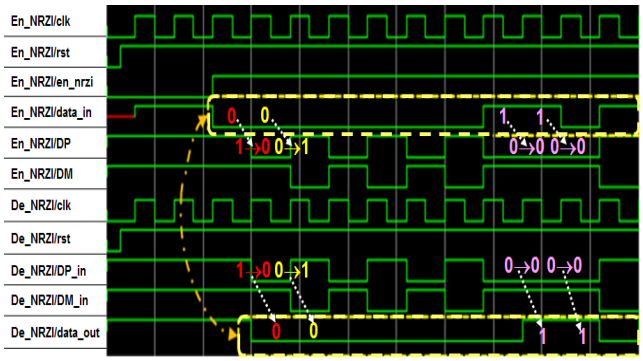

FIGURE 21. Simulation results for the NRZI block.

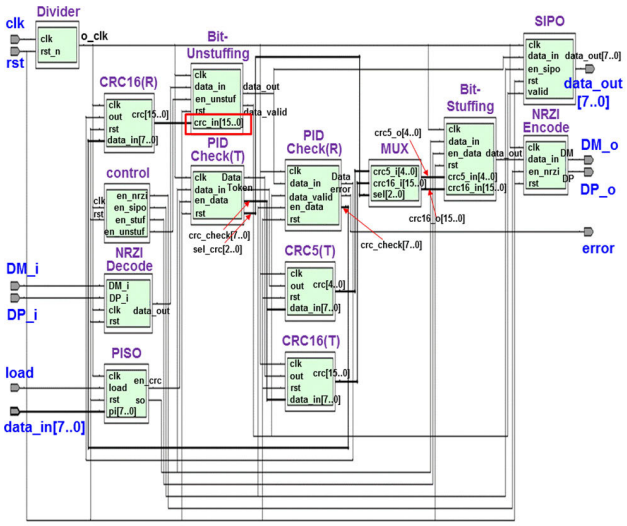

FIGURE 22. Functional block diagram of the proposed SIE module.

"Cb," which is stored in the ROM, is sent to the receiver board through a USB cable. The green LEDs display the ROM address "11100000" (E0) on the upper FPGA development board. The received data point, "11001011" (Cb), is displayed on the red LEDs of the lower FPGA development board. The measurement results indicated that the designed SIE module operated successfully.

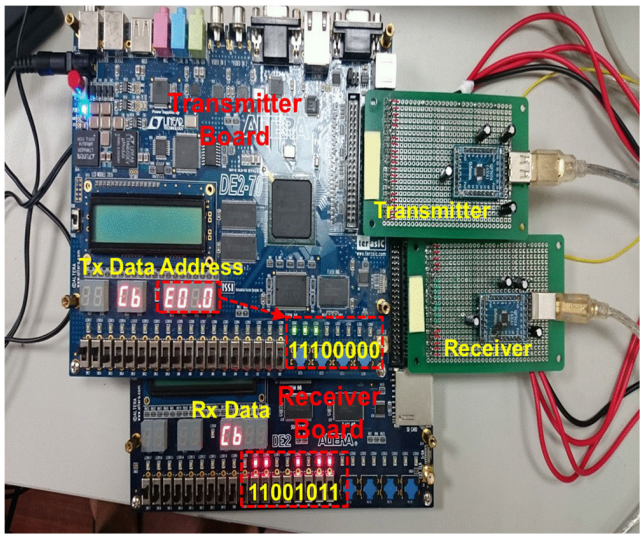

FIGURE 23. FPGA verification of the designed USB transceiver with an SIE module and a FIFO queue.

After the FPGA-to-FPGA communication had been verified by burning the Verilog HDL code into the Altera DE2-70 development board, the fabrication process of the ASIC was started. The RTL and gate level designs and simulations were completed with source code that had been designed with NC-Verilog software (Cadence Company, San Jose, CA, USA). Logic synthesis and chip layout were completed using the Design Compiler and IC Compiler tools, respectively (Cadence Company). The NC-Verilog software was used again to perform the post-layout simulation. After design rule check and layout versus schematic steps were completed using the Calibre RVE software (Cadence Company), the designed ASIC was taped out and fabricated using TSMC 0.18-μm CMOS technology. Figure 24 displays a microphotograph of the designed ASIC of the USB transceiver with an SIE module and a FIFO queue.

The performance comparison presented in Table 1 reveals that the maximum operating frequency and the data rate of the designed USB transceiver with an SIE module and a FIFO queue are lower than those of the models proposed

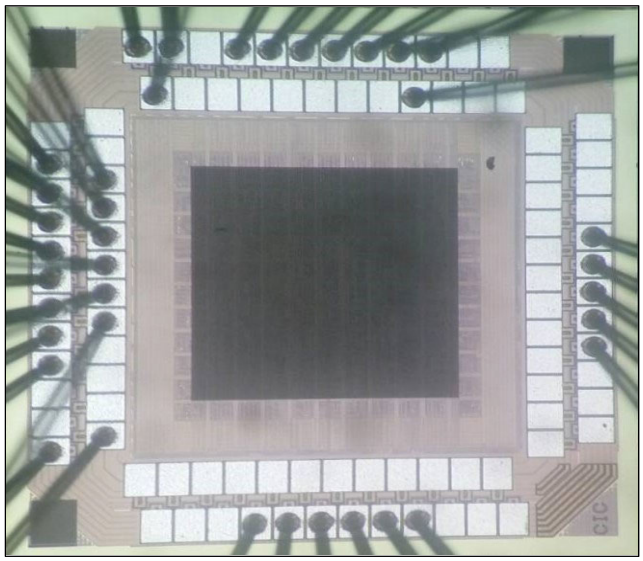

FIGURE 24. Microphotograph of the designed USB transceiver with an SIE module and a FIFO queue.

in [2], [19], and [20]. However, the gate counts and on-chip power consumption of the proposed transceiver are superior to those of the USB topologies presented in [19] and [20]. In this study, the designed USB transceiver is integrated with an SIE module and a FIFO queue into an ASIC. The small chip size and low power consumption of the proposed ASIC improve its performance and allow it to compete commercially with existing models. The asynchronous FIFO queue is used to guarantee correct communication between two FPGA development boards. After the FPGA verification had been completed, the proposed ASIC performed correctly because the performance of the fabricated ASIC was similar to that of the FPGAs.

IV. CONCLUSION

In this study, a USB transceiver with an SIE module and a FIFO queue was designed using Verilog HDL. Two FPGA development boards (Altera DE2-70) and two external USB port physical layer boards (PHYs) were introduced to verify the designed function. The simulation results and FPGA verification proved that the FPGA-to-FPGA communication design operated correctly. A clock frequency of 50 MHz was used for the FPGA development board, and a synchronous clock frequency of 130 MHz was obtained using the PLL IP. A clock divider of 5 was introduced to generate a clock frequency of 26 MHz, which is used in the USB PHY. After the designed USB transceiver was verified using the FPGA development board, the Verilog HDL code could be implemented into an ASIC by using TSMC 0.18-μm CMOS technology. The ASIC had a chip area of 0.7×0.67 mm^2 (including pads) and power consumption of 2.6742 mW. The small chip size and low power consumption of the ASIC grant the ASIC relatively high performance and enable it to compete commercially with existing models. The architecture of the proposed USB transceiver is more complicated than those of previously proposed USB topologies. The proposed architecture was designed to guarantee correct data transmission with a trade-off of large gate counts. The merits of the proposed ASIC, including its low power consumption, high speed, small size, and short delay, enhance its industrial applicability considerably.

APPENDIX

The authors have shared their HDL codes and RTL models online on GitHub BitLocker (link: https://github.com/Yan-Zhang-Yi/source_code.)

ACKNOWLEDGMENT

This manuscript was edited by Wallace Academic Editing.

REFERENCES

[1] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaaniche, and Y. Laarouchi, "Survey on security threats and protection mechanisms in embedded automotive networks," in *Proc. 43rd Annu. IEEE/IFIP Conf. Dependable Syst. Netw. Workshop (DSN-W)*, Budapest, Hungary, Jun. 2013, pp. 1–12.

[2] K. Babulu and K. S. Rajan, "FPGA implementation of USB transceiver macrocell interface with USB2.0 specifications," in *Proc. 1st Int. Conf. Emerg. Trends Eng. Technol.*, Nagpur, India, 2008, pp. 966–970.

[3] G.-M. Sung, H.-K. Wang, and Z.-Y. Li, "Hardware design on FPGA for Ethernet/SONET bridge in smart sensor system," in *Proc. 7th Int. Symp. Next Gener. Electron. (ISNE)*, Taipei, Taiwan, May 2018, pp. 1–4.

[4] A. Bouhraoua and M. Al-Shammari, "A fundamentally secure payment device interfaced to regular PCs," in *Proc. IEEE Region Conf.*, Kansas, MO, USA, Apr. 2008, pp. 1–5.

[5] G.-M. Sung, H.-K. Wang, and J.-H. Lin, "Serial interface engine asic with usb physical transceiver based on fpga development board," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Banff, AB, Canada, Oct. 2017, pp. 410–413.

[6] L. Nardozza, "USB-IF offers guidance to industry for design compliance of USB devices," in *Proc. USB Implementers Forum (USB-IF)*, Barcelona, Spain, Feb. 2016, pp. 1–3. [Online]. Available: http://www.usb.org/

[7] V. Vijaya, R. Valupadasu, B. R. Chunduri, C. K. Rekha, and B. Sreedevi, "FPGA implementation of RS232 to universal serial bus converter," in *Proc. IEEE Symp. Comput. Informat.*, Kuala Lumpur, Malaysia, Mar. 2011, pp. 237–242.

[8] N. F. Jusoh, M. A. Haron, and F. Sulaiman, "An FPGA implementation of shift converter block technique on FIFO for RS232 to universal serial bus converter," in *Proc. IEEE Control Syst. Graduate Res. Colloq.*, Shah Alam, Malaysia, Jul. 2012, pp. 219–224.

[9] P. M. Szecowka and K. J. Pyrzynski, "USB receiver/transmitter for FPGA implementation," in *Proc. Int. Conf. Signals Electron. Syst. (ICSES)*, Wroclaw, Poland, Sep. 2012, pp. 1–6.

[10] (2011). *Xilinx ISE Web Pack, Ver 12.3*. [Online]. Available: www.xilinx.com

[11] *Xilinx Spartan-3 Family Complete Data Sheet*, Xilinx, San Jose, CA, USA, 2007.

[12] K. Yu, "The aircraft engine blade detection system based on USB2.0 and FPGA," in *Proc. Int. Conf. Comput. Intell. Commun. Netw.*, Tetovo, Macedonia Republic, Nov. 2014, pp. 1001–1005.

[13] S. S. Shriramwar, G. A. Jichkar, and N. S. Panchbudhe, "FPGA implementation of USB 2.0 receiver protocol," *Int. J. Adv. Electron. Eng.*, vol. 1, pp. 199–203, May 2011.

[14] *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, Version 1.05*, Intel Corporation, Chennai, India, Accessed: Mar. 29, 2001. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/Technical-specifications/usb2-transceiver-macrocell-interface-specification.pdf

[15] *Universal Serial Bus 3.0 and 2.0 Specifications*, Intel Corporation, Chennai, India, 2014. [Online]. Available: http://www.intel.com/content/www/us/en/io/universal-serialbus/universal-serial-bus-specifications.html

[16] W. Peterson and D. Brown, "Cyclic codes for error detection," *Proc. IRE*, vol. 49, no. 1, pp. 228–235, Jan. 1961.
[17] A. M. Patel, "5. Signal and error-control coding," in *Magnetic Recording. II: Computer Data Storage*, 1st ed., C. D. Mee and E. D. Daniel, Eds. New York, NY, USA: McGraw-Hill, 1988.
[18] N. S. Panchbudhe, S. S. Shriramwar, and G. A. Jichkar, "FPGA implementation of USB 2.0 receiver protocol," in *Proc. Int. Conf. Adv. Comput., Commun. Netw.*, Chandigarh, India, Jun. 2011, pp. 758–762.
[19] R. Dwivedi and M. S. Narula, "USB 2.0 transceiver macrocell interface implementation on xilinx vivado," in *Proc. Int. Conf. Signal Process. Commun. (ICSC)*, Noida, India, Dec. 2016, pp. 419–424.
[20] O. V. Drozd and D. V. Kapulin, "The device of secure data transmission based on magma crypto algorithm with implementation on FPGA," in *Proc. Moscow Workshop Electron. Netw. Technol. (MWENT)*, Moscow, Russia, Mar. 2018, pp. 1–5.

GUO-MING SUNG (Member, IEEE) was born in Zhanghua, Taiwan, in 1963. He received the B.S. and M.S. degrees in biomedical engineering from Chung Yuan Christian University, Taoyuan, in 1987 and 1989, respectively, and the Ph.D. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2001.

In 1992, he joined the Division of Engineering and Applied Sciences, National Science Council, Taiwan, where he was an Associate Researcher, in 1996. Since 2001, he has been with the Department of Electrical Engineering, National Taipei University of Technology, where he is currently a Professor and the Chairman. His research interests include magnetic sensor, universal serial bus/RS485 communication, analog-to-digital converter IC, motor control IC, radio-frequency harvester IC, and mixed-mode ICs for use by the Internet of Things (IoT).

LI-FEN TUNG received the M.S. degree in electrical engineering from the National Taipei University of Technology, in 2019, where she is currently pursuing the Ph.D. degree with the Department of Electrical Engineering. She has been with Semiconductor Industry, since 2000. Her research interests include semiconductor devices, motor control, and the IoT with emphasis on practical applications.

HSIN-KWANG WANG received the B.S. and M.S. degrees in electrical engineering from the National Taipei University of Technology, Taipei, in 2007 and 2009, respectively. He is currently a General Manager with Bepro Digital Audio Ltd., where he has served for a period of 28 years. His research interests include audio signal processing and recognition, analysis and implementation of human sound, magnetic speaker, and DSP applications.

JHIH-HAO LIN received the B.S. degree in electrical engineering from the National Formosa University of Technology, Taipei, Taiwan, in 2013, and the M.S. degree in electrical engineering from the National Taipei University of Technology, Taipei, in 2015. His research interests include serial data processing of protocol, digital front end ICs, and PC peripherals interface applications.