Design of DCO-OFDM System for VLC on Chip at the Register-Transfer Level

Syifaul Fuada[1], Angga Pratama Putra[2], and Trio Adiono[3]
[1] Program Studi Sistem Telekomunikasi, Universitas Pendidikan Indonesia, Bandung, Indonesia
[2] VLC Research Group, Pusat Mikroelektronika, Institut Teknologi Bandung, Bandung, Indonesia
[3] School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Bandung, Indonesia
Email: syifaulfuada@upi.edu

Abstract—This paper reports a System-on-Chip (SoC) architecture design for the DC-biased optical OFDM (DCO-OFDM) Visible Light Communication (VLC) transceiver. The proposed SoC comprises several Digital Signal Processing (DSP) blocks, *i.e.,* Fast Fourier Transform (FFT), Convolutional Encoder (CE), Viterbi Decoder, Quadrature Phase-Shift Keying (QPSK) Modulator/Demodulator, Interleaver/Deinterleaver, and Synchronizer. System was designed through the combination of two Intellectual Property (IP) types: designing IP from the scratch (custom-based IP) and employing available core accelerator IP served by the third party, which is Xilinx library (reuse-based IP). All DSP blocks were targeted for the FPGA development board (Xilinx Zynq SoC 7000), then combined with the ARM microprocessor. In this study, ARM microprocessors were used for various tasks, *i.e.,* scheduling process, on-chip memory as a temporary data storage function, Analog-to-Digital Converter (A/D), Digital-to-Analog Converter (D/A), and Ethernet module as communication medium between SoCs and personal computer (PC). Testing was carried out on the Register Transfer Level (RTL) for hardware (H/W) and software (S/W) models implemented on ARM microprocessors. The system performances were measured through the point-to-point data communication scenarios between a PC transmitter and PC receiver. A 77 kbps of data communication speed and 2.9 ms of data processing latency were obtained using 100 MHz clock speed. The VLC on-chip was successfully demonstrated in RTL phase. This system is suitable for a low-rate communication system application, generally for joint VLC and Internet-things (IoT) technology, (then called as VLC/IoT).

Index Terms—Visible Light Communication, DCO-OFDM, System-on-Chip, FPGA, Reuse-based Intellectual Property, custom-based Intellectual Property

I. Introduction

The development of Visible Light Communication (VLC) on-chip has attracted many researchers worldwide in the last five years [1]–[3]. The research topics on the chip development for VLC applications are very diverse, *e.g.,* LED drivers/transmitters circuit [4], [5], the first stage of analog receiver circuit (amplifiers) [6]–[8], Digital Signal Processing (DSP) such as Orthogonal Frequency-Division Multiplexing (OFDM) [9], [10], and many more. The OFDM technique is implemented through a digital computation approach embedded on a DSP [11], [12]. There are various DSPs for VLC, and the Field-Programmable Gate Array (FPGA) is the best solution due to its wide range of capabilities that can meet the system design requirements.

The FPGA can process complex computations and can speed up numerous functions. The processing on FPGA can be done on software (S/W) level or hardware (H/W) level with the trade-off. The S/W implementation has a lower complexity and more flexible to be modified than the H/W implementation [13]. However, the computation run on H/W level is more efficient in consuming power than S/W implementation. There are various benefits of using System-on-Chip (SoC) approach. The H/W implementation can be realized on an Application-Specific Instruction Set Processor (ASIP), where each H/W processing element (PE) is S/W-programmable. In addition, there is free reusable Intellectual Property (IP) provided by third parties to accelerate SoC design and development. This underlies why it needs to employ a SoC as a core in a digital processor for a VLC system [15]–[18]. The OFDM consists of several computational blocks, which are quite time-consuming such as Fast-Fourier Transform (FFT) and Viterbi Decoder [14]. Accordingly, some computations would be better implemented on a H/W level to minimize processing latency and other processes implemented as S/W running on a microprocessor.

Many researchers studied OFDM design on FPGA for VLC, focusing on different issues. Yu et al. developed an FPGA-based OFDM adaptive system for VLC with Quadrature Amplitude Modulation (QAM). The system was claimed to be able to dynamically (adaptively) control the data transfer rate on OFDM according to the Signal-to-Noise Ratio (S/N) value of optical channels. The simulation showed that the OFDM data format (*M*-Subcarrier) has succeeded in converting 16 bits to 1024 bits so that the communication speed can be increased from 24.11 to 60.28 Mbps [19]. Their study was proved on a simulation only. Huo et al. developed a VLC system with RGB LEDs as transmitters and positive-intrinsic-negative diodes (PINs) as receivers. Simple modulation, On-off Keying (OOK), was implemented. The demonstration results showed that the FPGA can be used for high-speed data transmission: a 120 Mbps at 1 meter of optical

Manuscript received February 22, 2022; revised July 13, 2022.
Corresponding author email: syifaulfuada@upi.edu
doi:10.12720/jcm.17.8.608-624

channel with a Bit-Error Rate (BER) $<10^{-3}$ was reached [20]. It is showed that FPGA-based VLC systems can be used as solutions for real-time applications and contribute to system miniaturization if further explored intensively (towards specific chips). However, in [20] study used a classical modulation, OOK, which is more bandwidth-intensive than the OFDM technique. Figueiredo et al. successfully demonstrated OFDM-based VLC where the system's physical layer (PHY) was implemented on a Xilinx Virtex-6 FPGA. Through 25 MHz of bandwidth and 64-QAM modulation, a successful transmission speed of 150 Mbps (BER $<3.8 \times 10^{-3}$) has been obtained in a real-time at 50 cm distance [21]. These results indicated that FPGA platform is a digital processor solution for high-speed indoor VLC systems by utilizing ready-to-use LEDs (which are often found in standard lighting applications). However, in [21] did not reveal the OFDM architecture design in specific discussion, includes the strategy to develop SoC whether to use reuse-based IP (Third party IP) or create it from scratch. Chui-hu *et al*. explored OFDM for high-speed and stable real-time VLC applications. System verification has been carried out on Register-Transfer Level (RTL) simulation and real implementation. RTL simulation was done using ModelSim and implementation was conducted on FPGA (Xilinx Kintex-7) with Verilog language [22]. System testing was limited to signal delivery. Moreover, their research focuses on time-domain synchronization and channel equalization. Astharini *et al.* and Yuniati *et al.* modified OFDM implemented on a FPGA (Xilinx Artix-7). Simulations on RTL have been successfully carried out to evaluate the proposed design [23], [24]. Levent et al. designed an FPGA-based VLC system that focuses on the PHY layer with DC-biased Optical OFDM (DCO-OFDM) modulation [25]. However, it was not explained in detail the crucial part, which is H/W & S/W portion embedded on the SoC side.

By looking at the opportunities and potential for further development of this research areas, we still want to explore the FPGA-based OFDM for VLC system. The VLC models have been made in previous studies, but only simulated on MATLAB. This study is a continuation of previous work, which is to realize the OFDM models that have been successfully computed on MATLAB [14], [26], [27]. This research aims to design an SoC functioned to run a digital computing process using DCO-OFDM modulation on a directional VLC system (one link communication). The PE modules on the FPGA are designed through two approaches: IP core available in Xilinx environment as the third party (reuse-based IP) and custom-made IP. The 64-point FFT was implemented in this study. As a research limitation, the proposed system is prioritized for a digital computing architecture at the transmitter and receiver in which the data communication was carried out between two personal computers (PCs). A Prior work reported a Wi-Fi/VLC hybrid network strategy for auto-sharing data on Wi-Fi to light-fidelity (Li-Fi) channel or vice versa [28]. They have successfully proven the proposed architecture in a simulation phase, which is RTL. Instead, in this study, we will exhibit the RTL simulation to examine the proposed SoC transceiver design for OFDM-based VLC system.

II. Architecture Design

In this study, the SoC system is designed to modulate input data using DCO-OFDM for VLC application. The proposed system is divided into several primary components: Data transmission application on source PC, SoC platform for transmitter (SoC Tx), Digital-to-Analog Converter (D/A) [29], analog LED driver [30], [31] and Analog Front End (AFE) receiver [32], [33], that are separated module from this study, Analog-to-Digital Converter (A/D) [34], SoC platform for receiver (SoC Rx), and a specific application for receiving data on a destination PC. Fig. 1 illustrates the connection between these blocks in the proposed system.

Data transmission begins by sending a data bitstream on the PC source to the SoC Tx platform through an Ethernet cable. The data is then modulated digitally using DCO-OFDM. Afterward, the modulated data is transformed into an analog signal by a D/A module. The LED driver converts the analog signal into an optical signal. The data is transmitted by LED and received by the photodiode [35]. Afterward, the received signal enters the AFE circuit before being converted to the digital domain by an A/D module. Data from the A/D sampling are then demodulated on the SoC Rx platform and sent as a data bitstream to the PC destination. The PC displays the received data.

The SoC transceiver platform was designed to modulate data following DCO-OFDM standard employing FPGA as a DSP. The DSP was built by the integration of H/W (FPGA accelerator) and S/W (the firmware driver of the proposed system).

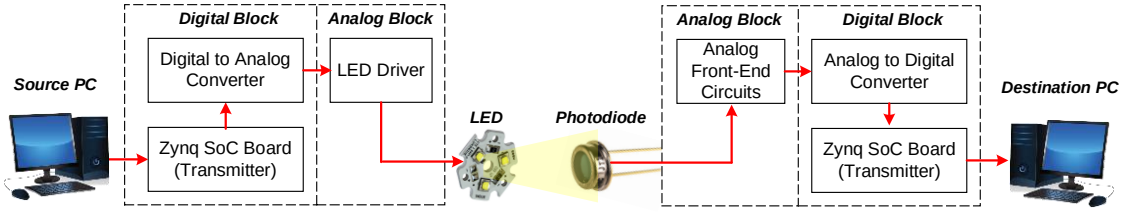


Fig. 1. Illustration of a system model designed for data transmission

A. FPGA Accelerator Design

The accelerator design on the FPGA was intended to speed up the computation process in DCO-OFDM modulation. In this system, several modular accelerators were used for SoC Tx and SoC Rx, in which it can be divided into two types: accelerators that are already available in the Xilinx IP library and custom-made IP that are designed from scratch because they are not available on the library. Besides, several custom IPs were also designed to perform as support functions, *e.g.,* interface signals activation.

The proposed SoC platform consists of two types: 1) H/W-based SoC (FPGA accelerator), 2) S/W-based SoC (applied on source PC and destination PC). The H/W – S/W partition model should be identified and defined carefully to obtain the best performance with a less trade-off, include: 1) all DCO-OFDM modules, 2) Ethernet communication modules as clients on the SoC Tx platform and on destination PC, and 3) Ethernet communication modules as servers on the SoC Rx platform and on source PC. There are several considerations in designing these mentioned models: computational complexity of the module, resource availability, processing speed requirements, computing time of the model and the model's criticality level. These factors are considered to determine whether a module function is implemented on programmable H/W as a PE or as a function in the S/W system run on the processor.

In this study, the scheduling process between modules will be run on the S/W. Some modules also need to be considered in the relationship between the S/W and interfaces available on the H/W (*e.g.*, modules on Ethernet communication) and other modules that require external interfaces (*e.g.*, UART communication).

The proposed system model is shown in Fig. 2. It can be seen that all DCO-OFDM modules were implemented on H/W level within the SoC platform instead of the S/W level. The DCO-OFDM modules are a critical function of VLC system. For this reason, DCO-OFDM modules are implemented on a programmable H/W for better processing performance, especially for processing the digital blocks with the highest latency (*e.g.*, FFT/IFFT and Viterbi Decoder blocks). The Ethernet communication module between PC and TCP-based SoC platforms was implemented on S/W. This module was divided into two sections: modules implemented on PCs & on the SoC platform. The S/W on a PC, TCP communication is programmed to access and receive files to be transmitted on a PC. In S/W on the SoC platform, TCP communication was implemented using the *lwIP* library.

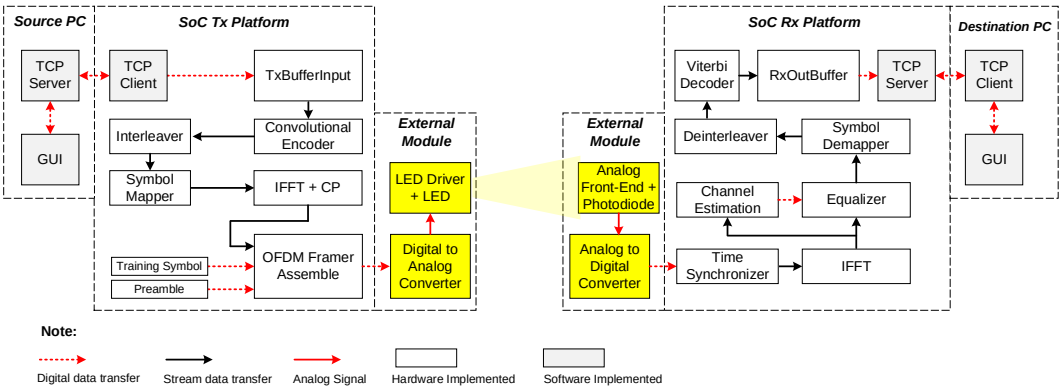


Fig. 2. System model of H/W - S/W partition (color illustration can be obtained and read at the online version of this Journal)

In data communication scheme between PE of DCO-OFDM, the data transfer was done through a stream interface between IPs of each PE. This is due to the PE implementation for this system uses available IPs from the third parties, which is Xilinx Vivado IP (Zynq-7000 SoC). Available interface is the AXI stream interface [36]. It is necessary to create a specific custom-made IP as a link between the AXI stream interface with AXI memory-mapped interface. Thus, it can be connected to the Zynq processing system (PS).

The *TxinputBuffer* IP is custom-based IP lies on the SoC Tx. It has functioned to receive and buffer 7 x 32 bits data from Zynq PS via AXI memory-mapped interface, then stream that data per one bit to the Convolutional Encoder (CE) accelerator via AXI stream interface. Whereas in the SoC Rx, the *RxOutBuffer* IP (custom-based IP type) is functioned to receive and buffer data per one bit from the Viterbi Decoder IP accelerator until it reaches a predetermined number (for interrupt to Zynq PS). Later, the user can access the data via the address on the AXI memory-mapped interface.

The proposed SoC architecture was divided into two parts: SoC Tx and SoC Rx as depicted in Fig. 3(a) and Fig. 3(b) respectively. The connection of each PE in H/W was defined along with the data path and also the interface used to connect with external devices. The main thing that needs attention is the data transfer mechanism between each PE. In this system, PE architecture with stream interfaces was connected to the AXI interconnect stream bus. PE was controlled by scheduling S/W on the ARM microprocessor through Stream-to-Memory-Mapped (S2MM) interfaces. This interface is part of the custom-made IPs, which is *TxBufferInput* IP and *RxOutBuffer* IP. For IP with a stream interface, the processing data needs to be stored first in a buffer until the process for one data is complete because it does not have a temporary register storage. In this study,

the buffer was implemented by Block Random Access Memory (BRAM).

BRAM is realized by several custom-based IPs integrated into one part of the IP within the RTL design. Whereas for the reuse-based IP, the buffering process was done by adding a specific IP, which is AXI stream data FIFO (First-in-First-out) between interfaces to synchronize communication signals. Therefore, data can be received correctly.

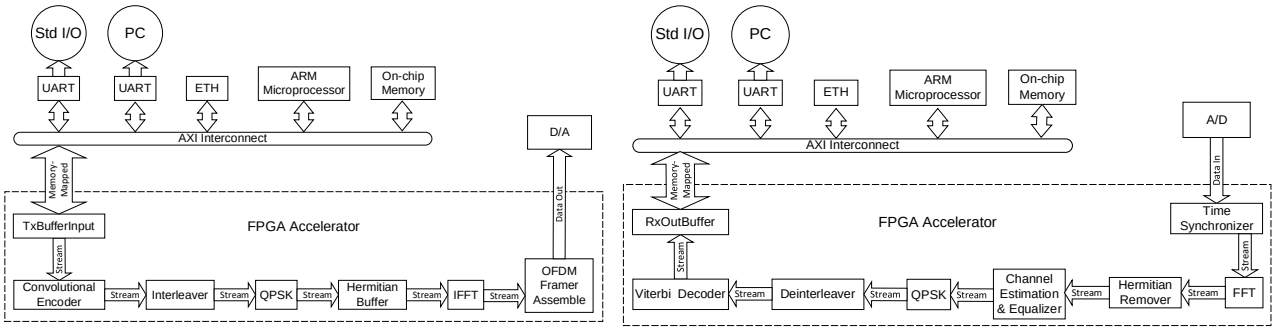


Fig. 3. Architecture of: (a) SoC Tx and (b) SoC Rx

B. Convolutional Encoder (CE) and Viterbi Decoder IPs

In this system, CE IP from Xilinx was used as the convolutional coding accelerator on the SoC Tx. This IP has an AXI stream interface for the master and slave ports and it was employed to do encryption by adding redundant data. Accordingly, the data bits were doubled from the original data according to the IP configuration. The redundant bit was used for forward error correction (FEC) in the SoC Rx to correct the bit error after getting a noise effect when it was transmitted through the channel. In the SoC Rx, these redundant bits were decoded using the Viterbi Decoder IP, which was served by Xilinx.

The decoding process for FEC was done using the Viterbi Decoder IP. Similar to the CE IP, this IP has an AXI stream interface for its master – slave ports. This IP was used to eliminate redundant data bits and correct data if there is an error. A trellis diagram was used as a method. In this study, this IP was configured to disable an optional pin, which is "BER symbol count". Thus, the slave and master ports will not appear on this IP to display the BER status.

C. Interleaver/Deinterleaver IP

The Interleaver/Deinterleaver IP also employs IP available from Xilinx. This IP has an AXI stream interface for its master and slave ports. The configuration of this IP is presented in Table I. The symbol width was set to 2 bits because this IP has 2 bits of input data from the CE IP with a bit rate of 0.5. Therefore, in the rectangular block of this IP, each block contains 2 bits of data.

TABLE I. THE INTERLEAVER IP CONFIGURATION

Variable	Description
Memory Style	Automatic
Symbol Width	2
Mode	Interleaver
Symbol Memory	Internal
Type	Rectangular Block
Number of Rows	28
Number of Columns	8
Block Size	224
Pipelining	Maximum
BLOCK_START Delay	224 samples + 7 cycles
Latency	7

D. IFFT/FFT IP

Multi-carrier modulation of all active subcarrier data signals at the SoC Tx was carried out using an IFFT IP served by Xilinx IP library. It was implemented on the FPGA accelerator. In the SoC Rx, demodulation was done using a FFT IP. Both of these IPs were implemented using the same IP (*i.e.*, xFFT block) in different mode. This IP also uses the AXI stream interface as a port in the master and slave parts. At the SoC Tx, the xFFT IP was configured with *config* mode as "inverse FFT" whereas in the SoC Rx, xFFT IP was configured with *config* mode as "forward FFT". The receiver does not require cyclic insertion prefix because the FFT on SoC Rx functions as a demodulator. Moreover, it follows the xFFT IP configuration on the SoC Tx.

E. AXI Stream Data FIFO IP

Besides using native IP served by Xilinx, custom IP which was not available on the Xilinx IP library was used. Native IP from Xilinx and custom IP should be integrated through appropriate interface communication signal. This study uses AXI standard. Buffering process is needed between these IP. Therefore, the communication process can run properly and also the data can be conveyed entirely. This buffering process was implemented using a native IP, which is *AXI stream data FIFO* IP. This IP was placed as a connector between multiple IP. The configuration of *AXI stream data FIFO* IP is presented in Table II.

TABLE II. CONFIGURATION OF AXI STREAM DATA FIFO IP

SoC Tx		
QPSK Mapper – Hermitian Buffer	FIFO depth	256
	TDATA width (bytes)	2
Hermitian Buffer – IFFT	FIFO depth	512
	TDATA width (bytes)	2
SoC Rx		

ShapiroRudinPark TimeSync – FFT	FIFO depth	1024
	TDATA width (bytes)	2
Hermitian Remover – Channel Estimator Equalizer	FIFO depth	512
	TDATA width (bytes)	2
Channel Estimator Equalizer – QPSK Demapper	FIFO depth	256
	TDATA width (bytes)	2

F. QPSK Mapper and QPSK Demapper IPs

The QPSK mapper IP is categorized as a custom IPs because they are not available on the Xilinx IP library [37]. The IP was designed based on the Gray-code mapping of QPSK. This IP integrates the logic interface for slaves and masters based on the AXI stream interface to communicate following to the standard. There are three sub-modules of QPSK Mappers IP: 1) AXI stream slave, 2) QPSK logic mapper, 3) and AXI stream master. Xilinx Vivado has provided a template code containing the logic of interface signals. The template code is then modified to fit the requirements of the QPSK logic mapper. The abstraction design of the QPSK mapper IP is shown in Fig. 4(a).

It can be seen that only two signals need to be bypassed, *i.e.*, signals from the AXI stream slave interface to the QPSK mapper logic and signals from the QPSK mapper logic to the AXI stream master. The mentioned signal is in the form of bus signal for the output ready signal and data. The output ready signal was used to declare to the AXI stream master that the output data is ready. In this study, the output and input of the top-level IP still follows the AXI interface standard without any additions or reductions. But only few signals used to be connected to the QPSK mapper logic. Fig. 4(b) shows the input data consists of 8 bits. But, the number of bits to be processed is only 2 bits of LSB (Least Significant Bit). The data output consists of 16 bits (8 bits of MSB represents imaginary numbers + 8 bits of LSB represents real numbers). MSB stands for "Most Significant Bit". These real and imaginary numbers determine the QPSK mapping which follows the Gray-code constellation.

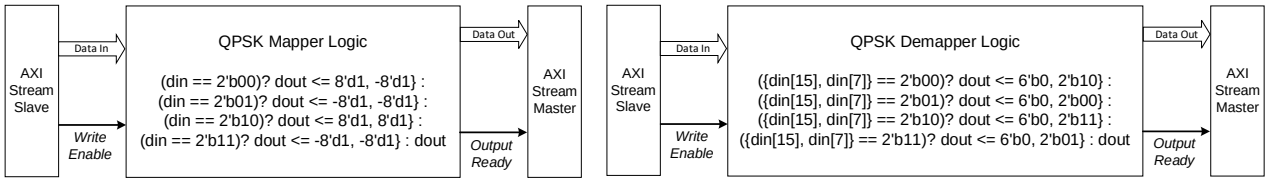


Fig. 4. (a) Top level of QPSK Mapper IP Model and (b) QPSK Demapper IP Model

The QPSK Demapper IP was used to reverse the QPSK Mapper IP effect using the Gray-code constellation also. The QPSK Demapper IP only needs to see the sign bits of the imaginary and real numbers. This process has an advantage, which is the high tolerance of bit errors because it visualizes a sign bit only. Moreover, it is not influenced by changes in magnitude values caused by channel noises. Despite the number of bits represented in a QPSK symbol is limited to only 2 bits, this QPSK modulation is quite robust against errors. The bit width data of the QPSK Demapper IP output is 8 bits in spite of the actual data is only found on 2 bits of LSB. Other bits on MSB (6 bits) are padding filled with zero values; this is due to the IP output is connected to the Deinterleaver IP input which has 8 bits of data width. In this system, the data width to be processed on the Deinterleaver IP and Interleaver IP were set to 2 bits.

G. Hermitian Buffer IP

The transmitted data in the VLC system must be real value (without imaginary components). Several procedures should be done to meet the requirements. Before the data is modulated by the IFFT IP, it must be arranged carefully to ensure the IFFT transformation results are real value. Therefore, the Hermitian Buffer IP was used to implement the Hermitian symmetry characteristic in IFFT resulting the overall output have real values. This IP is categorized as a custom-based IP with the stream-based type AXI interface on the master and slave ports. The hierarchy of this IP is depicted in Fig. 5(a).

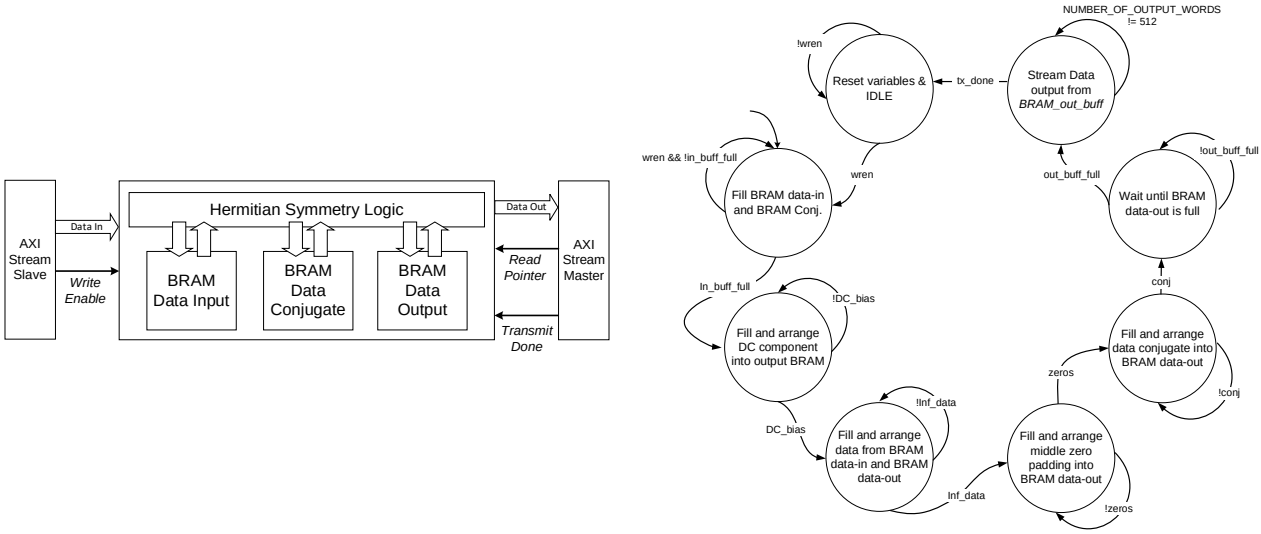
This IP uses BRAM blocks in the data buffering process. Three BRAMs were used: 1) for input data (denoted as BRAM data-in), 2) conjugate input data (BRAM Conj.), and 3) output data (BRAM data-out). The process of data buffering using a memory block requires to follow the rules found in the Xilinx synthesis guide. Accordingly, the synthesizer can distinguish whether the proposed design needs to be synthesized by using BRAM or Lock Up Table (LUT).

In this work, The Hermitian Buffer IP is designed by using BRAM as a memory block as a temporary data storage in the form of memory block. The hardware-description language (HDL) program consists of two core parts working in parallel: 1) a program for regulating the data flow on BRAM data-in along with its conjugates and 2) programs for regulating the data flow on BRAM data-out. The input data is first entered into the BRAM data-in and BRAM Conj. Afterward, the *in_buff_full* flag is then active once the filling data process on these two BRAMs is complete. Finally, the BRAM data-out is filled in certain index in sequence with a DC bias value.

The DC value can be given a zero value on the active subcarrier zero index for each data packet in the burst frame. The distance between the DC bias index with another is 64, includes main data, zero padding, and Hermitian conjugates. In the burst frame, there are 8 data packet symbols in which each symbol has 28 active subcarriers. When *DC_bias* flag is active, the filling data process will begin from BRAM data-in to BRAM data-out on the BRAM index to store active subcarrier data. Along

with it, there are two processes run in parallel: accessing data on the BRAM data-in and copying it on the BRAM data-out. The delay in accessing data needs to be considered until it can be copied to BRAM data-out with 2 clock delay. After 28 active subcarrier indexes for each packet in the burst frame are filled with data from the BRAM data-in, the *inf_data* flag is activated. The next process is to fill zero padding on the BRAM data-out. Zero padding filling is done on an empty index between two data subcarriers: 28 active data and 28 conjugate data.

After the zeros signal flag is active, conjugate data filling from the BRAM Conj. to the BRAM data-out is done in a parallel processing. Afterward, data arranged according to Hermitian symmetry characteristic is then issued by following the signal interface communication between Hermitian Buffer IP with IFFT IP. The address data index is accessed by the *read_ptr* signal. The finite state machine (FSM) of Hermitian Buffer IP is shown in Fig. 5(b).

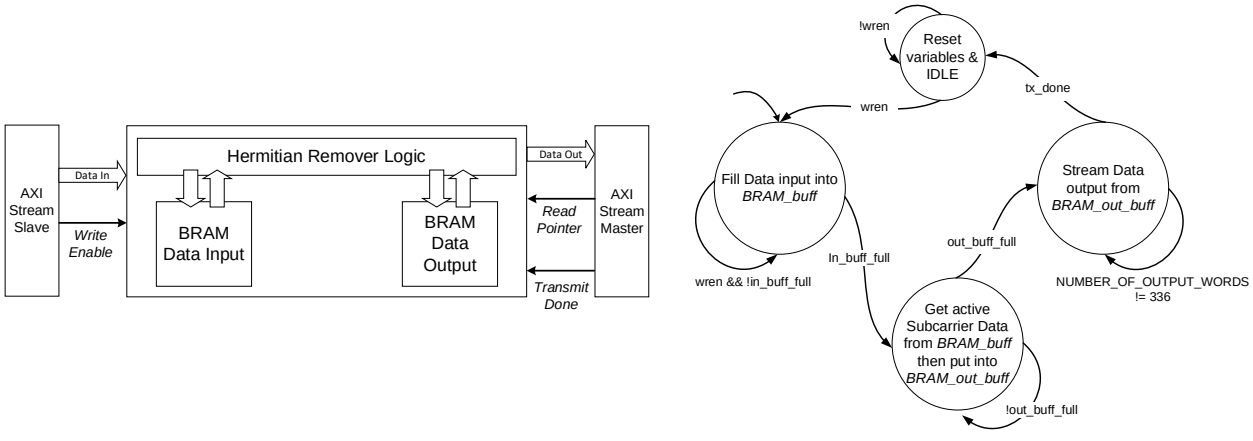


(a) (b)

Fig. 5. (a) Model of Hermitian Buffer IP; (b) FSM of Hermitian Buffer

H. Hermitian Remover IP

Hermitian Remover IP was used to remove the data that contains conjugate component and zero padding after the FFT process. This IP performs a simple data processing, it is only taking data on certain data index, which is 1st to 28th index, to be forwarded as output whereas other indexes are eliminated. Fig. 6(a) shows the hierarchy model of the Hermitian Remover IP.



(a) (b)

Fig. 6. (a) Model of Hermitian Remover IP; (b) FSM of Hermitian Remover

This IP also uses BRAM as a memory for buffering processes. Two BRAMs were used: BRAM data-in and BRAM data-out. The BRAM data-in was used to temporarily store raw data from FFT processing. BRAM data-out was used to temporarily store the data input which has passed two critical cores: 1) process of conjugating data and removing zero padding and 2) process of rearranging the data index. The first process, input data is stored in the BRAM data-in. Then the data on certain index are transferred from BRAM data-in to BRAM data-out. This data transfer is done without including three components, *i.e.*, DC bias, zero padding and conjugate data. Accessing data at the BRAM data-in and filling data at the BRAM data-out were processed on parallel. Thus, the

BRAM data-out only contains main data on 28 active subcarrier signals. The FSM of Hermitian Remover IP is shown in Fig. 6(b).

I. Time Synchronizer IP

Time Synchronizer IP used several BRAM as memory to store the calculation results when the time synchronizing process was performed. The total BRAM used in this IP was 6 units (consists of 2 BRAMs for input data, 1 BRAM for output data, 1 BRAM for calculating *P* value, 1 BRAM for calculating *R* value and 1 BRAM for calculating *M* value). The calculation process requires multiplication between index in the input data. If the BRAM is used at one time, there is only one data that can be taken and will add a delay of two clocks. That is why the input data uses 2 BRAMs so that the calculation process can run faster despite there is a trade-off, which is the number of resources used. The Time Synchronizer IP hierarchy is shown in Fig. 7.

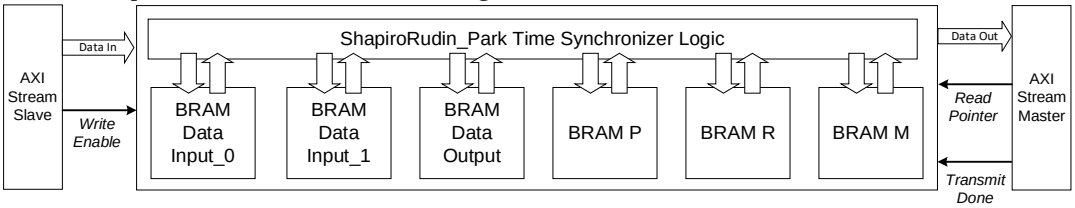
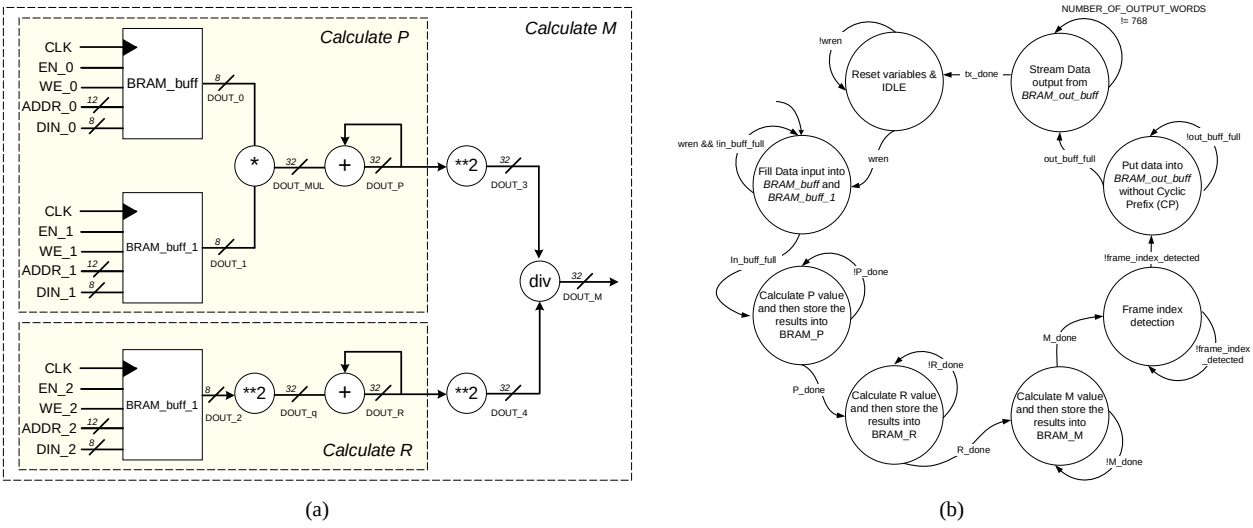


Fig. 7. Model of time synchronizer IP



(a) (b)

Fig. 8. (a) Model for *P*, *R*, and *M* calculation; (b) FSM of time synchronizer IP

The Time Synchronizer IP performs as a core to calculate three variables, which are *P*, *R* and *M* (Fig. 8a). Then the sample index based on the calculation results of the *M* variable is determined and cyclic prefix in that data is eliminated. In the calculation process for the *P* variable, correlation was done by doing dot product multiplication between two-time synchronization symbols. Multiplication process is done from the last index of the first-time synchronization symbol with the initial index of +1 from the next time synchronization symbol. Multiplication continues to be carried out with the first index time synchronization symbol. Later, the second index continues to be added until the size of one window index is equal to the number of FFT points. This result is added to the initial index value of the second time synchronization symbol that is already squared. The *R* value is obtained by multiplying the square between sample data value at the index and then adding the squared results to the previous index until all additions to the window point are complete. Afterward, the index window point increases by one.

The *M* value calculation was done by dividing all data in BRAM *P* with all data in BRAM *R* were previously squared. This value was stored at BRAM *M*. After all calculation operations were completed, the greatest value on BRAM *M* can be obtained. If the largest value in that index is found, it becomes an initial index of the time synchronization symbol. From this index, it can be specified other data indexes. The data is then transferred to the BRAM data-out without involving the cyclic prefix.

From the FSM flow of this IP (Fig. 8b), it can be seen that the input data is loaded into *BRAM_buff* and *BRAM_buff_1* until the BRAM is full. Then, the *in_buf_full* signal becomes active. Finally, the calculations for variables *P*, *R* and *M*, are done. There is a flag signal for each of these calculations indicating the calculation process is complete and ready to proceed to the next state. Afterward, the detection process of the initial frame-index is done. Further, the *frame_index_detected* signal becomes active. The data is then saved to *BRAM_out_buff*. When this BRAM is full, the *out_buff_full* signal becomes active and the data is streamed out up to 768 samples of output.

Then the *tx_done* signal becomes active and enter the IDLE state. Afterward, all variables are re-initialized. The process will recur when there is a *wren* signal (write enable) for input data on the slave port.

J. Channel Estimator & Equalizer IP

The hierarchy of Channel Estimator & Equalizer IP is shown in Fig. 9 This IP uses 8 BRAMs which consists of: 4 BRAMs to store a channel estimation symbol in the OFDM package. The remaining 1 BRAM each for storing real components from input data, storing imaginary components from input data, storing the equalizer coefficient, and storing output data. The reason of BRAM separation for each channel estimation symbol, real and imaginary components from the input data is to speed up the calculation process accompanying with a trade-off, which is resource utilization. This IP has the same purpose as the Time Synchronizer IP.

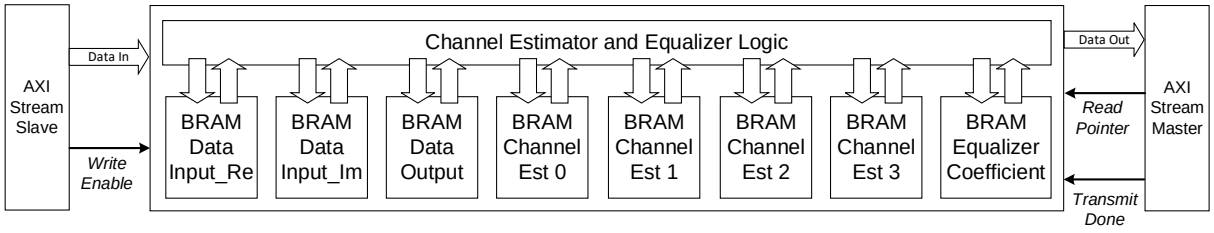
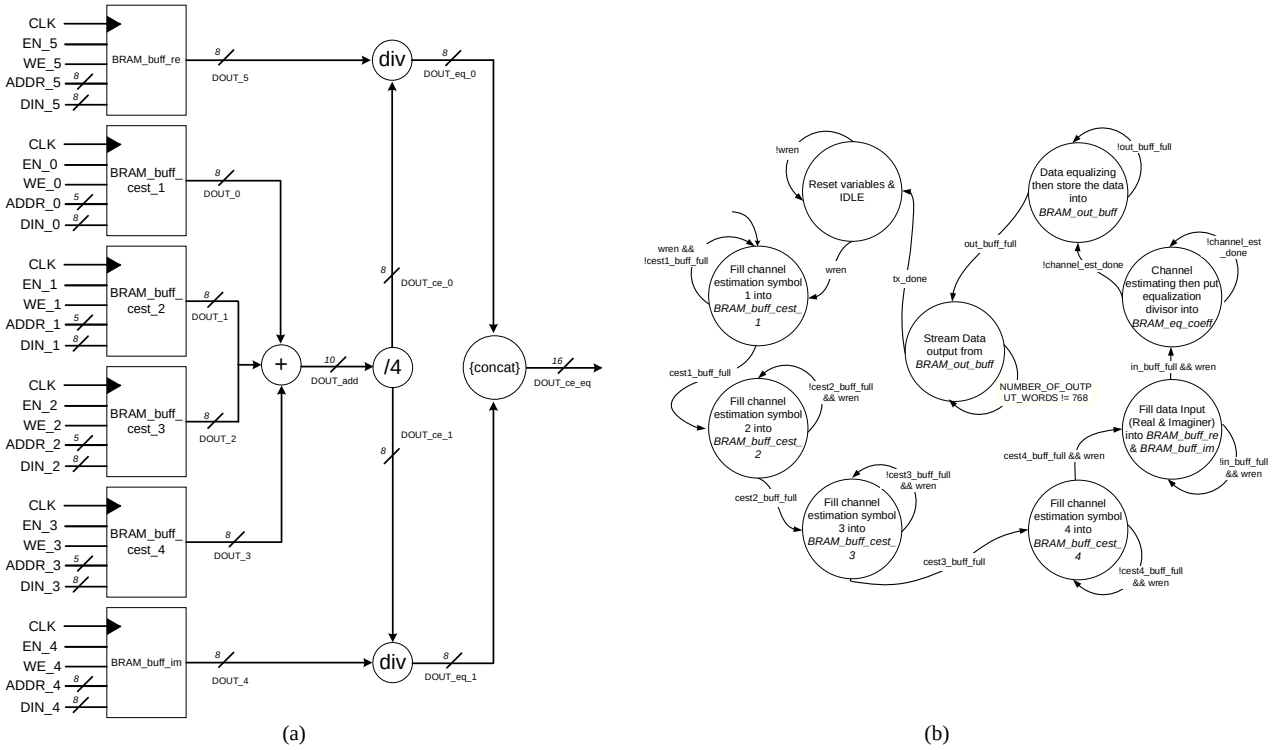


Fig. 9. Model of channel estimator & equalizer IP



(a) (b)

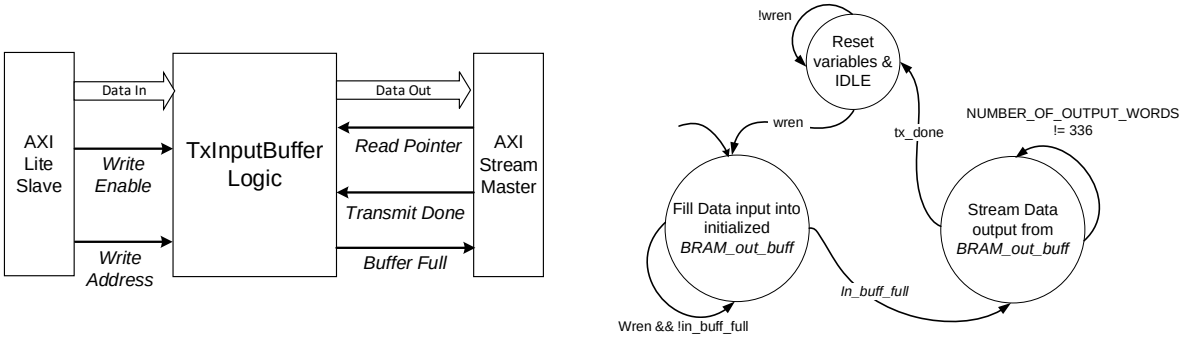
Fig. 10. (a) Model of computation on the channel estimator & equalizer IP; (b) FSM of channel estimator & equalizer IP

The FSM of this IP is depicted in Fig. 10(b). It can be seen that in the first process, the data filled in the BRAM is in the order of channel estimation symbol, *i.e.*, from the first to fourth symbol. Each of these symbols are filled on each BRAM with the reason for the averaging process of these four symbols can be done by accessing BRAM in parallel to save process latency. Symbols filled on each BRAM have own flag signal indicating the filling process was complete. Afterward, it is processed with the input data until the *in_buff_full* flag is activated. Finally, channel estimating and equalizing process are done and *channel_est_done* signal will active and data will be buffered at *BRAM_out_buff* until *out_buff_full* signal is activated. Later, the data is streamed out until the total sample reaches 768 samples. In the next process, the data will enter the IDLE state and all variables will be re-initialized. The process will be repeated when the slave port receives a *wren* signal.

This IP combines two functions: to run the channel estimation process as well as equalizing process (Fig. 10a). The symbol allocation for channel estimation in the OFDM data package is 4 symbol slots. After the time synchronizing process is done and its initial index is obtained, the index of the channel estimation symbols can be known precisely. After this symbol pass the FFT process and the Hermitian data is eliminated well, these 4 symbols go through the channel estimation process on the Channel Estimator & Equalizer IP.

This process is done by finding the average value of these four symbols after passing through the channel

estimating process for each active subcarrier. The average result is then stored in the BRAM equalizer coefficient. The data contents on this BRAM are taken as a dividing factor for each active subcarrier in OFDM data to eliminate the channel characteristics effect. The bit growth effect of FFT process need to pay attention in order to carry out the scaling process on the data distribution with an equalization coefficient. However, this effect is not important because we only need to see the sign bit value. The mentioned effects will be considered for the symbol mapping process using a mapper over QPSK, *e.g.*, 16-QAM, 64-QAM, 128-QAM, 256-QAM, Etc.



(a) (b)

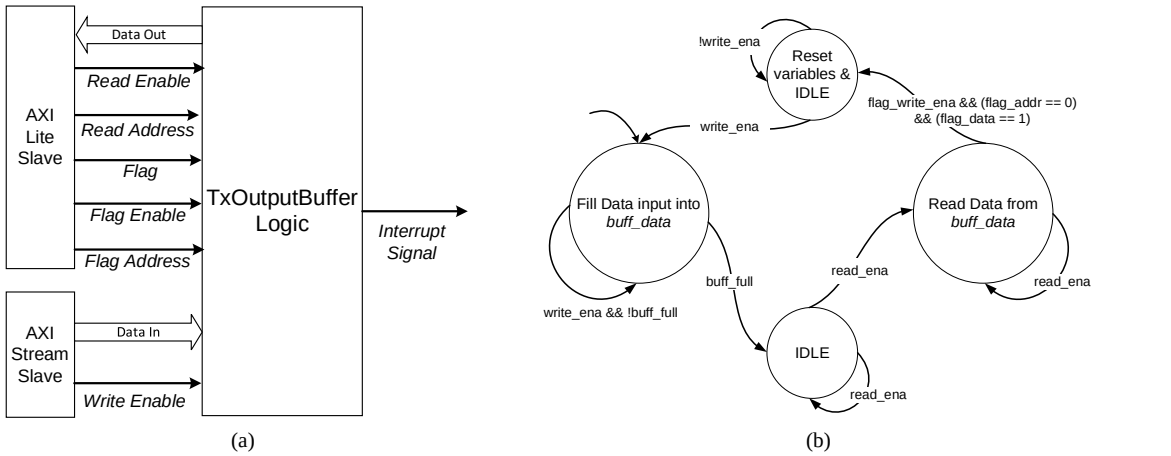
Fig. 11. (a) Model of *TxInputBuffer* IP; (b) FSM of *TxInputBuffer* IP

K. TxInputBuffer and *RxOutBuffer* IP

In this system, data processed by the SoC Tx is received from a PC via Ethernet communication. The data is then transferred to the FPGA accelerator. Finally, the data is modulated using DCO-OFDM. A memory-mapped IP needs to be created for processing the data through a stream-based accelerator. Therefore, data can be transferred by S/W on the ARM microprocessor to the FPGA accelerator. For this reason, a special custom-based IP was designed with the AXI lite interface that has a specific register address as the destination. Besides, it can split the received data into bitstreams to be processed in the accelerator. The hierarchy *TxInputBuffer* IP is shown in Fig. 11(a) while the FSM is depicted in Fig. 11(b).

In the initial process, this IP receives 32 bits × 7 integer data at the input register address. The amount of data 32 bits × 7 was chosen because the OFDM frame design has 28 symbols × 8 numbers of data in one frame, in addition to time synchronization and channel estimation symbols. In this study, QPSK was used as a mapper with 2 bits in one symbol. The QPSK data comes from CE IP with an output rate of 0.5 interleaved by a rectangular block Interleaver with 2 bits of data per column. The output data size is 2 bits for each sequence. Hence, it can be expressed as follows: 28 × 8 = 32 × 7. There are seven times of the integer data sending to IP. The 32 bits value is an integer data type in the S/W.

The vector size of *buff_data* is 224 bits. When the buffer is full, the *buff_full* flag will active. Afterward, the data is sent per bit to slave IP until the total sample of data output is 224 samples. The output data in this vector is accessed by a read pointer that will access the data per bit sequentially, starting from 0 to 223 indexes. When all data have been transmitted, the *tx_done* flag will active and all variables will be re-initialized. Also, the state will enter the IDLE process and wait until the *write_ena* signal appears for data at the *write_address==0*.



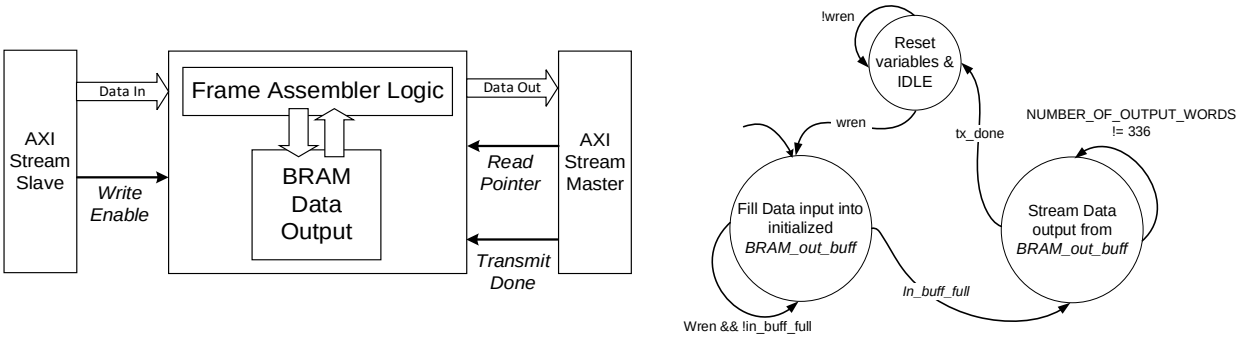
(a) (b)

Fig. 12. (a) Model of *RxOutBuffer* IP; (b) Model of *TxOutBuffer* IP

The *RxOutBuffer* IP is designed to reverse the *TxInputBuffer* IP process before the data is accessed by reading the register address. This IP accepts data per one bit from the Viterbi Decoder IP until 224 bits. The data is then arranged into 32 bits × 7 data sequentially. Once the process is complete, then the IP sends an interrupt signal

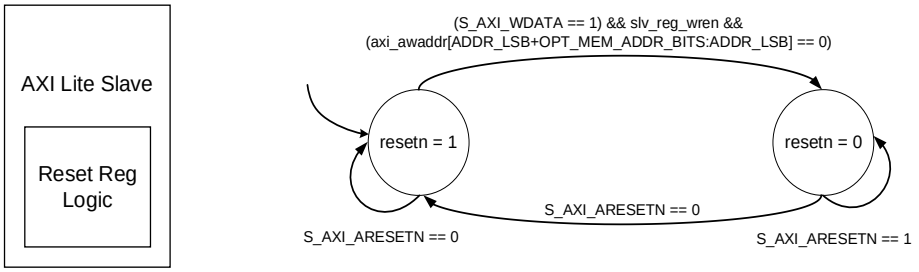
to the ARM microprocessor indicating the process is complete. Later, the data can be accessed. The hierarchy of *RxOutBuffer* IP is shown in Fig. 12(a).

In the first process, data is received by *RxOutBuffer* IP for every one bit of data. When the data filling process is finished, the *buff_full* flag is active indicating the buffer input is full. The *buff_full* flag also marks the interrupt signal to the ARM microprocessor. When the interrupt has been received by a firmware program, then the data can be accessed. This IP detects data requests and they will be streamed out based on the destination address. After all data have been accessed clearly, a signal is generated indicating the process has finished, which is *tx_done*. The *tx_done* signal is used as a sign to reset the register's contents. Then the process enters the IDLE state until it receives a *write_ena* signal. The FSM of the *RxOutBuffer* IP is depicted in Fig. 12(b).



(a) (b)

Fig. 13. (a) Model of frame assembler IP; (b) FSM of frame assembler IP



(a) (b)

Fig. 14. (a) Model of *ResetReg* IP; (b) FSM of reset flag

The FSM of Frame Assembler block is depicted in Fig. 13(b). Firstly, the program will store data at the index after the memory index, which is 480th index, for the time synchronization and channel estimation symbols. The *in_buff_full* flag signal will be activated indicates that the buffer is full and later the data will be streamed out. When the output data has reached 336 samples, the *tx_done* signal comes out indicating the process has finished. Afterward, the state will change to IDLE and all variable contents will be reset. The State will repeat like the previous process when it has received a *wren* signal.

M. ResetReg IP

Systems must be able to run repeatedly in data transmission in which it can be done many times depends on the data sent size. For this reason, a specific IP is needed to reset all accelerators on the FPGA, which can be controlled via firmware. Accordingly, the FPGA accelerator can process data for the next data transmission. When the reset is activated, all variables in the accelerator will be re-initialized to the initial value. The hierarchy model of *ResetReg* IP and its FSM are shown in Fig. 14(a) and 14(b) respectively. This IP works by receiving instructions from the user on the firmware with writing logic high ('1') in the specified address register. If this IP detects '1' logic on that address, then the *RESET_S* signal interrupt will be activated as "active low".

The *RESET_S* signal interrupt is then connected to the processor system reset IP in the input section of the *aux_reset_in*. This processor system reset IP is set so that the *Aux reset logic level* is '0' according to active low on the *RESET_S* signal. Besides, the *Aux reset width* is equal

L. Frame Assembler IP

In this data transmission system, the SoC Tx will continue to send the same OFDM frame to the SoC Rx until it gets a confirmation signal. This signal confirms that the data has been received properly by the SoC Rx. Before that, all burst data from DCO-OFDM modulation processing need to be packaged and integrated with time synchronization and channel estimation symbols that already stored in the memory block. These tasks were carried out by the Frame Assembler IP; this IP was designed to have one BRAM that already initialized first by a time synchronization and channel estimation symbol on the memory block's initial index. The next memory indexes are used to store the burst data from DCO-OFDM modulation results. The hierarchy model of the Frame Assembler IP is shown in Fig. 13(a).

to '1' which indicates the reset signal for all accelerators will be activated when a *RESET_S* signal is detected having '0' logic in at least one clock cycle. The *ResetReg* IP is included in the accelerator which will be reset by the processor system reset IP. When this IP has been reset, the *RESET_S* signal value will return to '1' logic.

III. Firmware Design

The firmware consists of two parts, *i.e.*, transmitter and receiver. In general, the firmware proposed in this study was responsible for receiving/sending data from PC source to PC destination or vice versa through Ethernet, and also in sending and reading data with an FPGA accelerator. This firmware employs the Xilinx SDK environment with a variety of available libraries. Fig. 15 shows the SoC architecture for testing purpose; it can be seen that the entire accelerator for the transmitter part is connected to each other by AXI stream. The accelerator data comes from the ARM microprocessor and then sent to the *TxInputBuffer* IP via Ethernet. This IP has a memory-mapped interface on the slave port. Afterward, data on this IP is then sent to another accelerator via the AXI stream interface on the master port.

The data from all accelerators processing on the DCO-OFDM modulation is fed to the ARM microprocessor through Direct-Memory Access (DMA). The DMA as the feedback system was used for testing purposes. In the actual system, the Frame Assembler IP becomes the output of the latest IP on DCO-OFDM modulation processing. This IP is available on the I/O pin of the Zynq development board. The data output from the Frame Assembler IP is then converted into an analog signal by the D/A. In addition, there is also an input pin to receive the interrupt signal from the receiver board. This interrupt signal indicates the data frame processing that is previously transmitted has been completed and is ready to receive the next OFDM frame. As long as the interrupt signal is not received, Frame Assembler IP will continue to loop the same OFDM frame transmission. This interrupt also functions as an interrupt signal to reset variables, like the *tx_done* signal function on a custom-based IP as elaborated in a previous section.

The SoC Rx has a similar mechanism to the SoC Tx with a little bit different (Fig. 16). The difference point relies that the data is sent from the ARM microprocessor to the FPGA's accelerator by transmitting memory-mapped-to-stream (MM2S) to the DCO-OFDM demodulation accelerator on the FPGA. The demodulation data is collected and received by the *RxOutBuffer* IP. Afterward, the *RxOutBuffer* IP sends an interrupt signal to the ARM microprocessor. This interrupt signal was used as a sign to access register address in the memory-mapped of the *RxOutBuffer* IP slave port.

Fig. 17(a) depicts a firmware design flowchart intended for the real application of DCO-OFDM system. In the system, data is received by the ARM microprocessor from the PC via an Ethernet connection. The entire accelerator on the FPGA is reset to initialize the values in the RTL design variable. Afterward, this reset process is done by setting the logic value in the address register for the slave port available from the *ResetReg* IP. Then the data is sent from ARM to the *TxInputBuffer* accelerator in the register address on the slave port. Then, it will be forwarded to other accelerators for carrying out the modulation process. The modulated data frame will be stored and transmitted repeatedly until the transmitter board receives an interrupt signal from the receiver. The interrupt signal indicates the frame has been demodulated properly. Data from modulation processing is forwarded to the DMA and then to be stored in an on-chip memory. After the DMA has finished receiving all the data, then the data is accessed by the ARM microprocessor via the S2MM port on the DMA. Firmware design for this test is shown in Fig. 17(b).

As shown in Fig. 17(c) the data comes from the A/D which is then connected via the I/O pin to the FPGA's accelerator IP. In this system, Time Synchronizer IP is directly connected to the sample data results of A/D module. When the demodulation process is successful, the ARM microprocessor receives an interrupt signal and then accesses the data on the *RxOutBuffer* IP register address. Then, it will be forwarded to the PC through an Ethernet connection. The process then repeats for the next transmission. Firmware is always in IDLE mode as long as the interrupt signal has not been received. However, for testing purpose, the data received by the accelerator comes from the ARM microprocessor instead of A/D module. The data is sent to the accelerator via DMA and then fed back to ARM microprocessor after it has been completely demodulated (Fig. 17d).

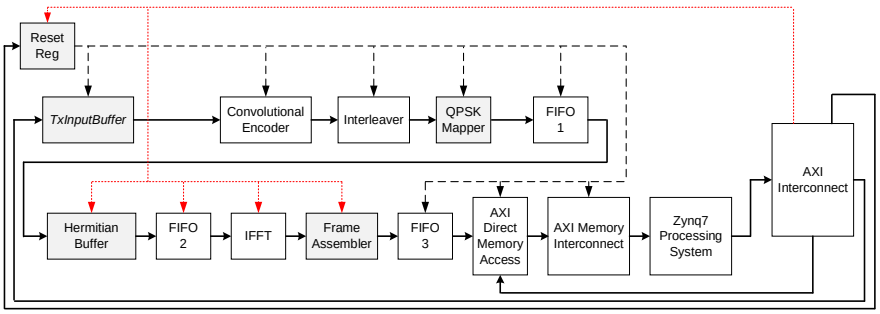


Fig. 15. Loop back testing methodology for SoC Tx

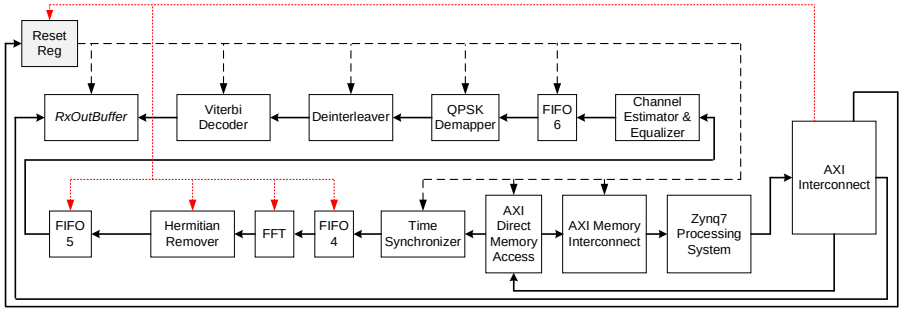


Fig. 16. Loop back testing methodology for SoC Rx

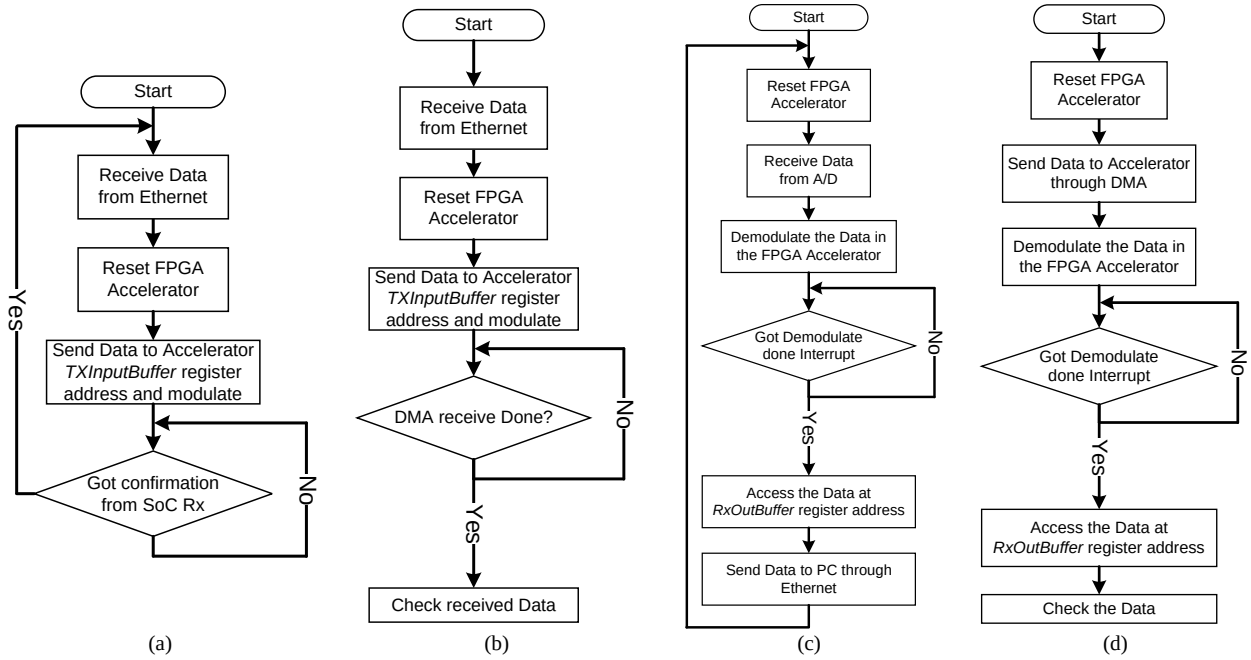


Fig. 17. (a) Firmware design flowchart for a proposed SoC Tx; (b) Flowchart of SoC Tx firmware for testing purpose using DMA; (c) Firmware design flowchart for a proposed SoC Rx; (d) Flowchart of SoC Rx firmware for testing purpose using DMA

IV. Results and Analysis

The main requirement that needs attention in the proposed SoC architecture is the data transfer mechanism between each PE. In this system, the PE architecture with stream interfaces is connected each other to a specific bus, which is AXI interconnect stream. These PE groups are controlled by scheduling S/W on the ARM microprocessor through S2MM as the bridge that are part of the custom-based IP (*i.e.*, *TxBufferInput* IP and *RxOutBuffer* IP). On the IP with stream interfaces, the processing result data needs to be stored first in a Buffer until the process for one stage of data processing is complete because it does not have a temporary storage register. This buffer was implemented using BRAM.

As elaborated in the Section II.A, the BRAM was implemented on several custom-based IPs that already integrated into one part of the RTL of each IP design. Whereas for reuse-based IP, buffering process was done by adding *AXI stream data FIFO* IP between interfaces as a communication signals synchronizer to guarantee that the data can be received validly. This buffering process is highly important due to the two main reasons: the process latency of each PE is different from each other and the communication available on the IP interface is asynchronous.

A. IP Testing

This section reports the testing results of the created IPs (*e.g.*, CE, Viterbi Decoder, Interleaver/Deinterleaver, IFFT/FFT, *AXI Stream Data FIFO*, QPSK Mapper, QPSK Demapper, Hermitian Buffer, Hermitian Remover, Time Synchronizer, Channel Estimator & Equalizer) that were observed from an RTL simulation. The data obtained was latency parameter in clock cycle unit.

The CE block requires a delay of four clocks starting from the first valid data input until the valid output data is first streamed out. In addition, the valid data output requires 2 bits of LSB data from 8 bits, with the 3rd to 8th bit data is zero padding. The same thing applies to input data, where data is only found in 1 bit of 8 bits LSB data, and the rest is zero padding.

The simulation results on the Viterbi Decoder IP showed that data processing requires 198 clock cycles, with each cycle on the clock was 10 ns. Whereas for

Interleaver/Deinterleaver IP, the latency was 231 clock cycles for 224 data samples. This result was in line with the timing diagram design that was taken from the IP summary report.

The simulation results on the IFFT/FFT IP reveal that the latency of this IP was 372 clock cycles. In the initial input data, this IP will take the excess sample data rather than the number of transform points before the data is processed and the ready signal changes to low. With a total transformation point of 64, the number of first data samples taken was 82 samples from all sample data (64 × 8 samples) in one OFDM package. The sampling process of the next package will be taken with the size corresponding to the number of samples. The *AXI stream data FIFO* IP is placed as a connector between IPs which is equal to 3 clock cycles of latency.

The simulation result on the QPSK Mapper IP was 1 clock cycle of latency. This IP has input data consisting of 8 bits that will only process 2 bits of LSB. While, the output data of this IP consists of 16 bits (8 bits of MSB represents imaginary numbers + 8 bits LSB represents real numbers). These real and imaginary numbers determine the QPSK mapping which follows the Gray-code constellation rule.

The QPSK Demapper IP has 1 clock cycle of latency. The QPSK Demapper IP output is connected as an input of the Deinterleaver IP which has an input data width of 8 bits. That is why the bit width of the QPSK Demapper output data is the same with Deinterlever IP, which is 8 bits (main data on 2 bits of LSB + 6 bits of MSB that are padding filled with zero values). For this system, the data width that will be processed on the Deinterleaver IP is set to 2 bits in the symbol width section likewise with the Interleaver IP.

The latency of Hermitian Buffer IP and Hermitian Remover IP were 747 clock cycles and 5137 clock cycles respectively. Hermitian Remover IP needs a large clock cycle because it is waiting for the FFT to finish overall processes in the burst frame. The data reception will continue without pause until it is completed, that is why the clock cycle is larger than Hermitian Buffer IP.

The simulation results on the Time Synchronizer IP showed that the latency is 279951 clock cycles. The latency is very large due to process a lot of complex mathematical operation. In addition, this IP contains a lot of data transfer between BRAM. Lastly, the Channel Estimator & Equalizer IP was 567 clock cycles of latency.

After being tested one by one for each IP, then the entire IP system will be tested by connecting the SoC Tx accelerator to the SoC Rx. It will be discussed in subsequent section.

B. Accelerator System Testing

The output of the Frame Assembler IP on the SoC Tx was connected to the Time Synchronizer IP on the SoC Rx through the Data FIFO IP intermediary. System configuration for this testing is illustrated by Fig. 18; it can be seen a structure of all IPs, starting from the data entered until the data comes out. Data input for the system testbench comes from an Upcounter which will continue to perform the enumeration. Due to data taken only needs 1 bit of LSB, then the input data for testing is a repetitive sequence of '101010 …'. Although this sequence will continue to increase due to each IP accelerator having determined the limit number of input and output data, this test was limited to only transmitting one OFDM frame. However, this does not mean that the proposed system was only limited to process one OFDM frame. In further implementation, if it has been combined with a firmware, then the transmission can be done not only one time but also many times. The reset feature on the firmware should be activated to perform continuous data transmission; it can be accessed via *ResetReg* IP.

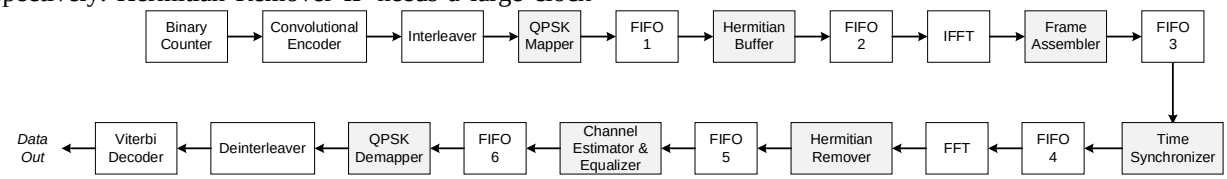


Fig. 18. Accelerator testing block design

Fig. 19. Input data for the proposed system

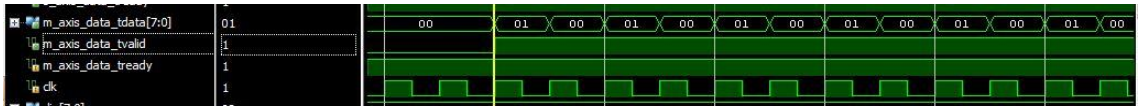


Fig. 20. Output data from the proposed system

The test results showed that the system accelerator can work well as expected. It can be seen that the output data were the same as the input data, *i.e.*, the sequence number of '101010 ...' (Fig. 19). Besides, the overall accelerator latencies for each SoC Tx and SoC Rx can be found, *i.e.*, 4570 clock cycles and 286468 clock cycles respectively (Fig. 20). Afterward, the data bit rate can be calculated using Equation (1). The number of active subcarriers and data burst were 28 and 8 respectively. Data rate symbol mapper and data rate encoder were 2. The total clock used

and frequency clock are 291,038 clock and 100 MHz respectively. Based on the calculation, a 77 kbps of data rate is obtained in which it was quite reasonable used for low speed application, especially for IoT [38], [39].

Authors of [38] proposed a VLC/IoT system that was capable of transferring the data up to 115.2 kbps. This transmission speed was in accordance with the IEEE 802.15.7 PHY I standard (about 100 kbps). The BER in [38] was claimed to be very low and capable of operating at a distance of 7 meters. The basic difference between this study and [38] is the processor unit. It is important to employ SoC for system customization as well as to accelerate the data exchange among VLC/IoT modules within the available network. Authors of [39] reported a novel VLC/IoT system that has been implemented on a prototype with an energy harvesting feature. Hardware in their system involved several important modules, such as photodiodes, flexible electronics, and solar cells. However, authors of [39] did not explain the resulting bit rate in the paper.

In future work, the accelerator design needs to be enhanced by using a more efficient architecture at the basic mathematical operations stages such as power, multiplication, division, etc. to run faster processing. Furthermore, this system needs to be tested in real testbed by connecting external A/D and D/A modules, analog LED driver circuit and AFE. Surprisingly, the graph of SNR against BER in various scenarios (*e.g.*, changing the distance of optical channel, LED power, LED position, channel characteristic, angle, number of photodiodes, etc.) can be plotted. In addition, the throughput against latency (delay) in operational field (actual condition) can be observed. Then, it can be compared to various works related to VLC/IoT use case in terms of aforementioned aspects.

$$Bit\ rate = Number\ of\ active\ subcarriers \times Number\ of\ data\ burst \times \frac{data\ rate\ symbol\ mapper}{data\ rate\ encoder} \times \frac{1}{\left(\frac{total\ clock}{frequency\ clock}\right)} \quad (1)$$

V. Conclusion

The SoC transceiver for the DCO-OFDM-based VLC system has been designed and simulated carefully in RTL. Sending and receiving data between two PCs bridged by the SoC platform was done using an Ethernet module. The SoC platform (Tx & Rx) was embedded on the FPGA development board (Xilinx Zynq SoC 7000). Time synchronizer, channel estimator, and equalizer, have been successfully modeled and implemented as an accelerator. The OFDM was built with three main frames: synchronization signal (2 symbols width), channel estimation signal (4 symbols width), and data (8 symbols width). The OFDM design takes into account the standard frame structure. Several IP core accelerators served from Xilinx (*i.e.*, Convolutional Encoder, Viterbi Decoder, IFFT/FFT, and Interleaver/Deinterleaver) were directly utilized to speed up the development process. In addition, custom-based IP for accelerator modules (*i.e.*, Hermitian Buffer, Frame Assembler, Time Synchronizer, Channel Estimation & Equalizer, Hermitian Remover, *TxInputBuffer*, *RxOutBuffer*, and *ResetReg*) have been created. The reuse-based IP (the 3rd party IP) and custom-based IP (custom-made IP) were passed the functional verification and successfully integrated with AXI bus protocol provided by Xilinx Vivado. Validation process for the RTL accelerator system and firmware were still done separately.

In the next development, a further debugging process will be carried out when the H/W and S/W have been connected. Accordingly, the test result data will be more emphasized. The MAC layer reliability can be later improved to make the system capable to handle the large-scale data transmissions robustly. Also, it can be equipped with several features such as energy-saving mode. Besides, the system implementation on the chip area can be smaller as possible.

List of Abbreviations

A/D = Analog-to-Digital Converter
AFE = Analog Front End
ARM = Advanced RISC Machines
ASIP = Application-Specific Instruction Set Processor
AXI = Advanced eXtensible *Interface*
BER = Bit-Error Rate
BRAM = Block Random Access Memory
BRAM Conj. = BRAM data conjugate
BRAM data-in = BRAM data input
BRAM data-out = BRAM data output
CE = Convolutional Encoder
D/A = Digital-to-Analog Conveter
DCO = DC-biased Optical OFDM
DMA = Direct-Memory Access
FEC = Forward Error Correction
FFT = Fast Fourier Transform
FIFO = First-in-First-out
FPGA = Field-Programmable Gate Array
FSM = Finite-state Machine
H/W = Hardware SoC
HDL = Hardware-description Language
IFFT = Inverse FFT
IoT = Internet-of-Things
IP = Intellectual Property
Li-Fi = Light Fidelity
LSB = Least Significant Bit
LUT = Lock Up Table
MM2S = Memory-Mapped-to-Stream

MSB = Most Significant Bit
OFDM = Orthogonal Frequency-Division Multiplexing
OOK = On-Off Keying
PC = Personal Computer
PE = Processing element
PHY = Physical layer
PIN diode = Positive-Intrinsic-Negative diode
PS = Processing System
QAM = Quadrature Amplitude Modulation
QPSK = Quadrature Phase-Shift Keying
RISC = Reduced Instruction Set Computer
RTL = Register-Transfer Level
Rx = Receiver
S/N = Signal-to-Noise Ratio
S/W = Software in the SoC
S2MM = Stream-to-Memory-Mapped
SoC = System-on-chip
SoC Rx = SoC Receiver
SoC Tx = SoC Transmitter
TCP = Transmission Control Protocol
Tx = Transmitter
UART = Universal Asynchronous Rx-Tx
VLC = Visible Light Communication

Conflict of Interest

The authors declare no conflict of interest

Author Contributions

S.F., A.P.P, and T.A. conceptualized and proposed the system model; S.F. and A.P.P designed each module of the model; A.P.P. implemented, simulated, and verified the system; S.F. analyzed and interpreted the data, summarized the main findings and made suggestions; A.P.P. drafted the research report in brief; S.F. contributed to writing, translating, proof-editing, correcting typos, and revising the manuscript. T.A. supervised the research; S.F. and A.P.P. contributed to drawing all figures within manuscript. All authors had approved the final version.

Acknowledgment

The operational cost of this research was supported by KEMRISTEK DIKTI Republik Indonesia in the Kerjasama Luar Negeri program (No. 009/SP2H/LT/DRPM/IV/2017) while the publication fee was handled by Program Peningkatan Global Competitiveness Perguruan Tinggi Indonesia Universitas Pendidikan Indonesia 2021 Batch II (No SK 1370/UN40/PT.01.02/2021).

References

[1] T. Adiono, "A real-time visible light communication System on Chip (SoC) design for high speed wireless communication," in *Proc. 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Sep. 2019, pp. 4–4.

[2] S. Fuada, T. Adiono, A. P. Putra, and E. Setiawan, "Design of reconfigurable system-on-chip architecture for optical wireless communication," *J. Commun.*, vol. 14, no. 10, pp. 965–970, 2019.

[3] Z. Pan, *et al.*, "Visible light communication cyber-physical systems-on- chip for smart cities," *J. Commun.*, vol. 14, no. 12, pp. 1141–1146, 2019.

[4] F. Che, L. Wu, B. Hussain, X. Li, and C. P. Yue, "A fully integrated IEEE 802.15.7 visible light communication transmitter with On-Chip 8-W 85% efficiency boost LED driver," *J. Light. Technol.*, vol. 34, no. 10, pp. 2419–2430, May 2016.

[5] C. S. A. Gong, Y. C. Lee, J. L. Lai, C. H. Yu, L. R. Huang, and C. Y. Yang, "The high-efficiency LED driver for visible light communication applications," *Sci. Rep.*, vol. 6, no. 1, p. 30991, Aug. 2016.

[6] X. Bi, Z. Gu, and Q. Xu, "Analysis and design of ultra-large dynamic range CMOS transimpedance amplifier with automatically-controlled multi-current-bleeding paths," *IEEE Trans. Circuits Syst. Regul. Pap.*, vol. 66, no. 9, pp. 3266–3278, Sep. 2019.

[7] M. T. I. Badal, M. B. I. Reaz, L. S. Yeng, M. A. S. Bhuiyan, and F. Haque, "Advancement of CMOS transimpedance amplifier for optical receiver," *Trans. Electr. Electron. Mater.*, vol. 20, no. 2, pp. 73–84, Apr. 2019.

[8] R. Y. Chen and Z. Y. Yang, "CMOS transimpedance amplifier for gigabit-per-second optical wireless communications," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 63, no. 5, pp. 418–422, May 2016.

[9] E. Setiawan, T. Adiono, and S. Fuada, "PHY layer design of OFDM-VLC system based on SoC using reuse methodology," in *Proc. International SoC Design Conference (ISOCC)*, Daegu, Korea (South), Nov. 2018, pp. 115–116.

[10] E. Setiawan, T. Adiono, S. Fuada, and W. O. Popoola, "Demodulator IP cores design for OFDM-based visible light communication system-on-chip," in *Proc. 3rd International Conference and Exhibition on Visible Light Communication (ICEVLC)*, Seoul, South Korea, 2019, pp. 54–57.

[11] T. Adiono, A. Pradana, R. V. W. Putra, W. A. Cahyadi, and Y. H. Chung, "Physical layer design with analog front end for bidirectional DCO-OFDM visible light communications," *Optik*, vol. 138, pp. 103–118, Jun. 2017.

[12] M. Figueiredo and C. Ribeiro, "OFDM-Based VLC systems FPGA prototyping," in *Visible Light Communications*, 1st ed., Z. Ghassemlooy, L. N. Alves, S. Zvánovec, and M. A. Khalighi, Eds. London, UK: CRC Press, 2016, pp. 1–36.

[13] T. Adiono and A. P. Putra, "Hardware/software model of DCO-OFDM based visible light communication SoC using DMA," in *Proc. International SoC Design Conference (ISOCC)*, Seoul, Korea (South), Nov. 2017, pp. 92–93.

[14] T. Adiono, Y. Aska, S. Fuada, and A. A. Purwita, "Design of an OFDM system for VLC with a viterbi decoder," *IEIE Trans. Smart Process. Comput.*, vol. 6, no. 6, pp. 455–465, 2017.

[15] E. Setiawan, S. Fuada, and T. Adiono, "Experimental demonstration of OFDM visible light communications based on system-on-chip," in *Proc. International Symposium on Electronics and Smart Devices (ISESD)*, Bandung, Oct. 2018, pp. 1–5.

[16] E. Setiawan, T. Adiono, I. N. O. Osahon, and W. O. Popoola, "Experimental demonstration of visible light communication using white LED, blue filter and SoC based test-bed," in *Proc. International Symposium on Electronics and Smart Devices (ISESD)*, Bandung, Indonesia, Oct. 2019, pp. 1–4.

[17] E. Setiawan and T. Adiono, "Throughput improvement of an autocorrelation block for time synchronization in OFDM-based LiFi," in *Proc. International SoC Design Conference (ISOCC)*, Jeju, South Korea, Oct. 2019, pp. 219–220.

[18] T. Adiono, S. Fuada, and R. A. Saputro, "Rapid development of system-on-Chip (SoC) for network-enabled visible light communications," *Int. J. Recent Contrib. Eng. Sci. IT IJES*, vol. 6, no. 1, Mar. 2018.

[19] Z. Yu, S. Xiao, Y. Hou, M. Ju, and W. Hu, "Simulation of adaptive OFDM White-LED visible light communication system using high-order QAM based on FPGA," in *Proc. Asia Communications and Photonics Conference*, Nov. 2014.

[20] Q. Huo, M. Zhang, W. Xu, D. Han, M. Wu, and Q. Li, "FPGA-based 120Mbps online visible light communication system with RGB LEDs," in *Proc. 15th International Conference on Optical Communications and Networks (ICOCN)*, Hangzhou, China, Sep. 2016, pp. 1–3.

[21] M. Figueiredo, C. Ribeiro, and L. N. Alves, "Live demonstration: 150Mbps+ DCO-OFDM VLC," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, QC, Canada, May 2016, pp. 457–457.

[22] C. Wu, *et al.*, "Realization of OFDM modulation and demodulation for visible light communication based on FPGA," *Optoelectron. Lett.*, vol. 13, no. 1, pp. 58–62, Jan. 2017.

[23] D. Astharini, N. I. H. Pratama, S. Rahardjo, F. R. Triputra, A. Syahriar, and O. N. Samijayani, "Design and Analysis of Generalized LED Index Modulation OFDM on FPGA," in *Proc. International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, Shah Alam, Malaysia, Jul. 2018, pp. 1–7.

[24] Y. Yuniati, "Implementasi sistem hermitian generalized LED index modulation (H-GLIM-OFDM) pada Board FPGA Xilinx Arty Artix-7," *Electrician*, vol. 14, no. 3, Oct. 2020.

[25] V. E. Levent, *et al.*, "FPGA based DCO-OFDM PHY transceiver for VLC systems," in *Proc. 11th International Conference on Electrical and Electronics Engineering (ELECO)*, Bursa, Turkey, Nov. 2019, pp. 418–421.

[26] A. P. Putra and S. Fuada, "Pemodelan sistem komunikasi cahaya tampak berbasis DCO-OFDM dengan viterbi decoder pada MATLAB," *J. SIMETRIS*, vol. 10, no. 2, pp. 529–542, 2019.

[27] E. Setiawan, T. Adiono, and S. Fuada, "Modelling the OFDM-based PHY Layer in SoC for visible light communication," *Int. J. Recent Contrib. Eng. Sci. IT IJES*, vol. 7, no. 3, Sep. 2019.

[28] R. Kashif, F. Lin, O. J. Famoriji, and S. Haider, "An architecture design of auto channel switching unit for hybrid visible light communication system," *J. Commun.*, vol. 16, no. 11, pp. 522–527, 2021.

[29] S. Fuada, A. P. Putra, Y. Aska, A. Pradana, E. Setiawan, and T. Adiono, "Implementasi perangkat digital signal processing untuk sistem visible light communication," *Jetri J. Ilm. Tek. Elektro*, vol. 15, no. 2, Feb. 2018.

[30] S. Fuada, T. Adiono, A. P. Putra, and Y. Aska, "A low-cost Analog Front-End (AFE) transmitter designs for OFDM visible light communications," in *Proc. International Symposium on Electronics and Smart Devices (ISESD)*, Bandung, Indonesia, Nov. 2016, pp. 371–375.

[31] S. Fuada, T. Adiono, A. P. Putra, and Y. Aska, "LED driver design for indoor lighting and low-rate data transmission purpose," *Optik*, vol. 156, pp. 847–856, Mar. 2018.

[32] S. Fuada, "Design and implementation of analog front-end transceiver module for visible light communication system," Master thesis, Department of Electrical Engineering, School of Electrical Engineering and Informatics (SEEI), Bandung, Indonesia, 2017.

[33] S. Fuada and T. Adiono, "Visible light communication kits for education," *J. Educ. Train.*, vol. 5, no. 2, May 2018.

[34] A. P. Putra, S. Fuada, Y. Aska, and T. Adiono, "System-on-Chip architecture for high-speed data acquisition in visible light communication system," in *Proc. International Symposium on Electronics and Smart Devices (ISESD)*, Bandung, Indonesia, Nov. 2016, pp. 63–67.

[35] S. Fuada, A. P. Putra, and T. Adiono, "Analysis of received power characteristics of commercial photodiodes in indoor LOS channel visible light communication," *Int. J. Adv. Comput. Sci. Appl. IJACSA*, vol. 8, no. 7, 2017.

[36] E. Setiawan and T. Adiono, "Design of AXI4-Stream based modulator IP core for visible light communication system-on-chip," *J. INFOTEL*, vol. 10, no. 2, Jul. 2018.

[37] E. Setiawan, M. M. Latin, V. A. Mardiana, and T. Adiono, "Implementation of baseband transmitter design based on QPSK modulation on Zynq-7000 all-programmable System-on-Chip," in *Proc. International Symposium on Electronics and Smart Devices (ISESD)*, Yogyakarta, Indonesia, Oct. 2017, pp. 138–143.

[38] A. Makvandi, Y. S. Kavian, and E. Namjoo, "VLCIoT: Design and implementation of a visible light communication system for indoor Internet of Things applications," *Appl. Opt.*, vol. 60, no. 36, pp. 11094–11103, Dec. 2021.

[39] A. Perera, M. Katz, R. Godaliyadda, J. Häkkinen, and E. Strömmer, "Light-based internet of things: Implementation of an optically connected energy-autonomous node," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, Nanjing, China, Mar. 2021, pp. 1–7.

Syifaul Fuada is with the the Program Studi Sistem Telekomunikasi Universitas Pendidikan Indonesia (UPI) as a young Lecturer and now serving as an assistant professor (Dosen Asisten Ahli) in the same study of program. Mr. Fuada has several has several achievements, such as the most outstanding students of Universitas Negeri Malang in 2013, receiving one of the 106 Indonesia Innovations by BIC-RISTEK DIKTI awards (2014) for Helm Charger product, a top of 10–student travel grant to the IEEE Asia Pacific Conference and Systems (APCCAS 2016) that was held in Jeju, South Korea, receiving one of 108 Indonesia Innovations by BIC-LIPI awards (2016) for Smart Home Product, student winner nominee of NOLTA conference (2017), receiving Best Paper Award from IEEE IGBSG 2018 that was held in Yi-Lan, Taiwan, receiving Best Paper Award from IEEE ICTRuDev 2018 that was held in Bali, Indonesia, receiving the Best paper award from a Scopus-indexed journal, i.e., i-JOE in 2019, receiving Best Paper Award from IEEE IGBSG 2019 that was held in Yichang, China, receiving the 111 Indonesia Innovations by BIC awards (2019) for E-Nelayan and LI-FI products, receiving the 112 Indonesia Innovations by BIC awards (2020) for three innovations: Bidirectional DC/DC Converter for Electric ATV, Smart Home, and Contact/Contactless-based Payment Device, the 3rd place of UPI's *most productive researcher in the SCOPUS-based publication* 2020 (awarded on 2021), the 3rd place of UPI's inventor awarded on 2021. His study interests include analog circuit design and instrumentation, circuit simulation, engineering education, IoT, multimedia learning development, and VLC

Angga Pratama Putra received his B.Sc. degree on Electrical Engineering from School of Electrical Engineering and Informatics, Institut Teknologi Bandung (ITB), Indonesia (2015) and M.Sc in Electrical Engineering with specialization in Computer Engineering, with same campus (2017). His research interests include: Embedded System, Software Engineering, VLSI, System-on-Chip, Internet of Things, VLC

Trio Adiono received a B.Eng. in electrical engineering and an M.Eng. in microelectronics from Institut Teknologi Bandung (ITB), Indonesia, in 1994 and 1996, respectively. He obtained his Ph.D. in VLSI Design from Tokyo Institute of Technology (Titech) in 2002, Japan. In 2005, he was a visiting scholar at MESA+, Twente University, Netherlands. He was also Adjunct Assoc. Prof. NTUST-Taiwan. He has developed several microchips, such as Binary Template Matching Processor, WiMax Baseband Chipset, Smart Card, and IoT. He also has job experience working with Chip Design House in Fukuoka Japan, handling chip design for several multinational companies. He received the "Second Japan Intellectual Property (IP) Award" in 2000 from Nikkei BP. He received Award Karya Lencana Wira Karya from President of Republic Indonesia for his innovation in 4G chip developments. He also holds a Japanese Patent on "High Quality Video Compression System". In the last ten years, he supervised ITB students that received "Communication Society Award" from IEICE-Japan, Xilinx Award, Analog Device Award, and "LSI of the year Award", Japan. He was also chair of IEEE Solid State Circuits Society. His research interests include VLSI design, signal and image processing, VLC, smart cards, and electronics solution design and integration.