

# **FPGA-Based Architectures: An Introduction to Experimental Designs with VHDL**

**Mehmet Sönmez**

**Genel Yayın Yönetmeni / Editor in Chief • C. Cansın Selin Temana**

**Kapak & İç Tasarım / Cover & Interior Design • Serüven Yayınevi**

**Birinci Basım / First Edition • © Aralık 2024**

**ISBN • 978-625-5552-23-5**

**© copyright**

Bu kitabın yayın hakkı Serüven Yayınevi'ne aittir.

Kaynak gösterilmeden alıntı yapılamaz, izin almadan hiçbir yolla çoğaltılamaz.

The right to publish this book belongs to Serüven Publishing. Citation can not be shown without the source, reproduced in any way without permission.

**Serüven Yayınevi / Serüven Publishing**

**Türkiye Adres / Turkey Address:** Kızılay Mah. Fevzi Çakmak 1. Sokak

Ümit Apt No: 22/A Çankaya/ANKARA

**Telefon / Phone:** 05437675765

**web:** [www.seruvenyayinevi.com](http://www.seruvenyayinevi.com)

**e-mail:** [seruvenyayinevi@gmail.com](mailto:seruvenyayinevi@gmail.com)

**Baskı & Cilt / Printing & Volume**

Sertifika / Certificate No: 47083

# **FPGA-Based Architectures: An Introduction to Experimental Designs with VHDL**

**Mehmet Sönmez**

## **CONTENTS**

<b>1. Introduction to Project Design in Quartus Software</b>	<b>1</b>
<b>2. Error Correction Algorithms: An Introduction to Previous Works, Theoretical Analysis and Implementation</b>	<b>18</b>
<b>3. Simulation Results for Digital Designs</b>	<b>33</b>
<b>4. FPGA implementations of Basic Logical Operators and Error Correction Algorithms</b>	<b>58</b>



# **Introduction to Project Design in Quartus Software**

## **1. Introduction**

In many fields, digital designs have attracted attention from many designers to achieve the required systems (Xia et al., 2024; Lata and Cenkeramaddi, 2023; Sharobim, 2023). Especially, it was considered the digital signal processing (Chen et al., 2023; Bureneva and Mironov, 2023), image processing (Wang et al., 2024; Bailey, 2023), communication systems (Ayoub et al., 2024; Krishnamoorthy, 2022), information systems (Ferraz et al., 2021; Devadoss and Ramapackiam, 2024) etc. The type of digital circuit is very crucial for experimental implementations since mentioned systems require very high-speed data processing architectures (McDonald et al., 2023; Kalomiros and Lygouras, 2008). In many tasks the boards were preferred to implement the proposed architectures (Ramírez-Montañez et al., 2023) while it is used specific semiconductor devices to design digital circuits (Yuvaraja, 2023).

## **2. Introduction to Programming**

It has been improved a few software platforms to implement digital designs to FPGA environment and to monitor the proposed algorithms via simulation process (Bruno and Eschemann, 2024). Thanks to these simulation and implementation programs, some parameters such as resource utilization can be estimated before the proposed architecture is implemented to real time digital systems (Koch, 2012; Sklyarov, 2014). There are two software languages to implement the algorithms to FPGA board and to view the behavior of algorithms: Quartus and Vivado FPGA compliers (Tlelo-Cuautle et al., 2016). In this section, it has been a detailed explanation related to the usage of Quartus software complier. Additionally, it is considered a topical software complier that is Quartus II 13.1 Web Edition.

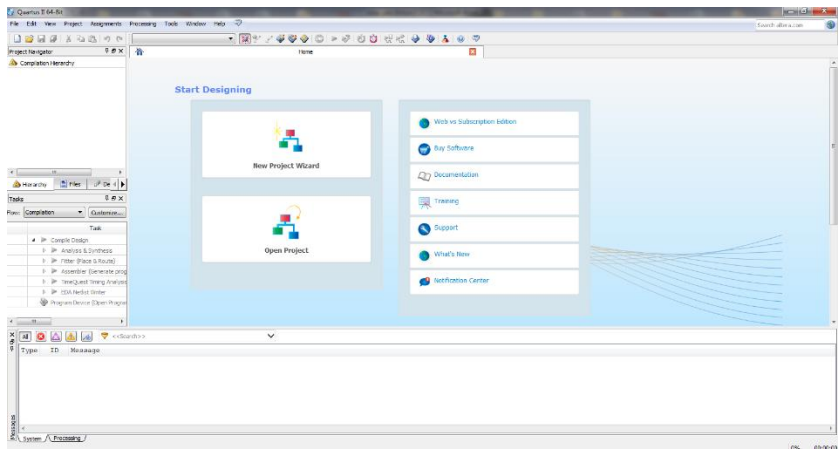


Figure-1 Starting of Quartus II Software

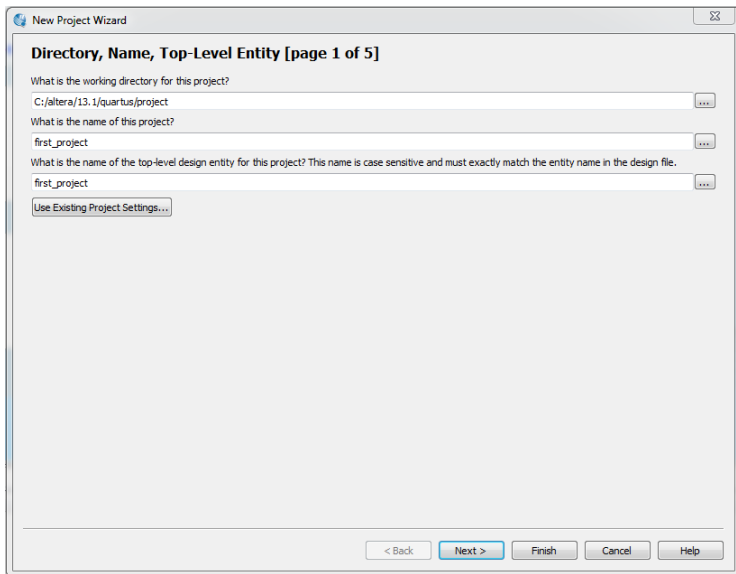


Figure-2 Project name and directory

In Figure 1, it is given a splash screen of Quartus software. As shown in the figure, the last project files can be accessed from splash screen of software. The *New Project Wizard* button can be used to create a new project design. To create a new project, the first stage can be given as shown in Figure 2. While the first line points out the place where the project will be saved, the second line is filled by considering the project name. In this project, the project name is *first\_project*. After the location

and name of project is determined, the new project can be designed. In the first project, a few basic applications can be reported related to logical gates such as AND, OR, XOR, and etc. Therefore, the available blocks in the Quartus compiler are used to implement these basic algorithms.

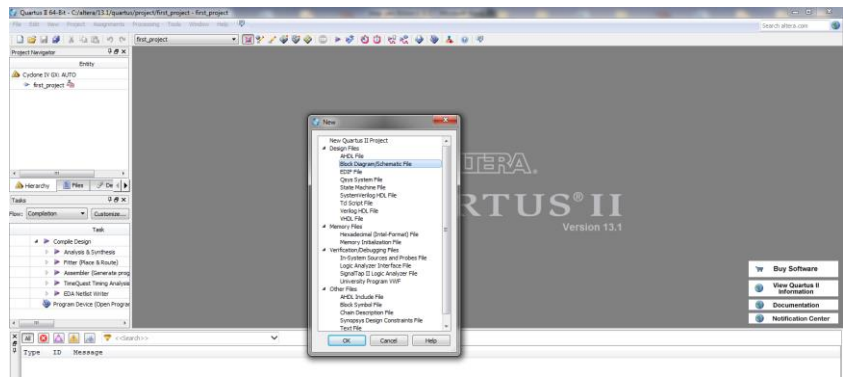


Figure-3 Block Diagram

To use the block diagram that is presented by Quartus compiler, Block Diagram/Schematic File is selected as shown in Figure 3. This window will open after New button is pressed. In the next stage, a schematic file is observed as shown in Figure 4. In this window, available blocks can be selected.

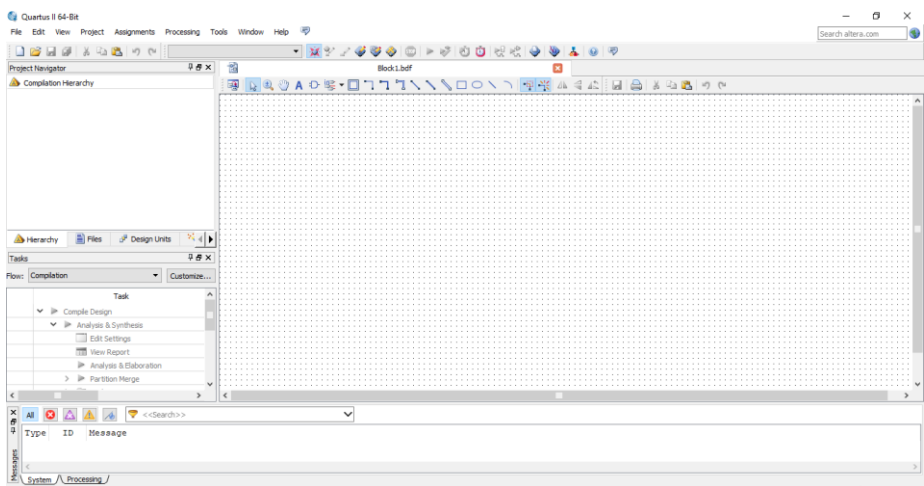


Figure-4 Block Diagram/Schematic File

In Figure 4, a .bdf file interface is illustrated. In the screen given in this figure, VHDL or Verilog blocks can be placed. These blocks are created by using VHDL or Verilog description languages. Because one of the most important issues is synchronization between blocks, the added blocks created by using VHDL or Verilog software languages present many advantages and flexibility in terms of timing adjusting. The proper FPGA family can be selected for experimental applications in the section of Project Navigator. All project designs created by using VHDL or Verilog can be shown in this section. Addition to these, analysis process can be monitored in the Task section which gives compilation percentage. The settings file includes the FPGA device family. The FPGA family contains Arria, Cyclone, Max and Starix processors. In this window, it can be selected FPGA board which is used for experimental systems.

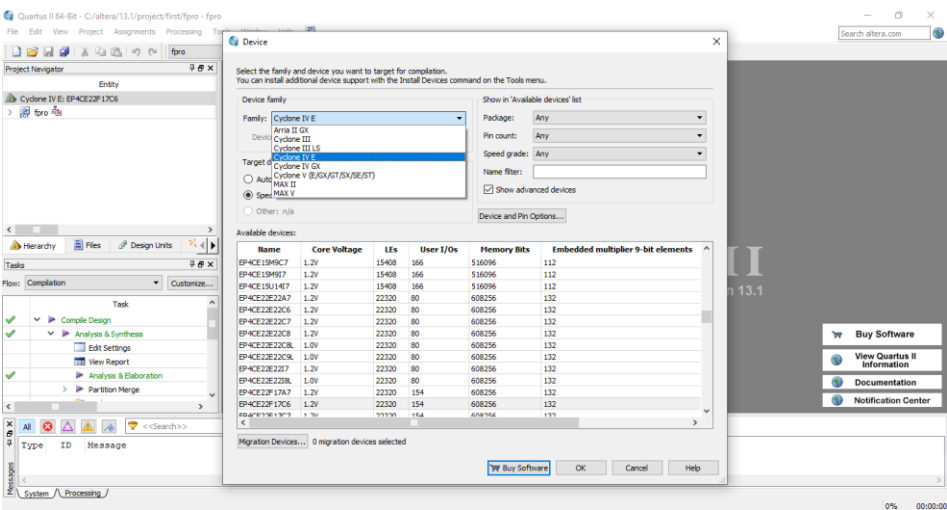


Figure-5 Block Diagram/Schematic File

As shown in the figures, there are many options in terms of the FPGA board. The most efficient board can be selected among these families by taking account for resource amount.

It must be given some theoretical framework related to basic logical gates before the first implementation is presented in this section.

Therefore, it can be given some information for logical gates such as OR, AND, XOR, switching element such as Mux and register such as D-type flip-flops.

### **3. An Overview of Logical Gates**

In logical circuits, OR, AND, and XOR gates may be defined as basic gates (Parr, 2013). Additionally, inverters of gates have very simple mode. These gates are included for many complex systems such as VLSI based electronic circuits although these operation elements have basic structures. In specifically, they are extensively used for communication systems which are based on VLSI to implement error correction codes in experimental systems (Leung, 2011).

In many FPGA designs, it can be considered some important issues such as maximum operating frequency, resource utilization (Lawal et al., 2015). The minimum operating period of a specific implementation determines directly maximum operating frequency. Since this operating period also affects data processing ratio of logical elements, the gate delay is a special and a significant term in VLSI designs (Panda et al., 2020). Moreover, critical path delay determines maximum operating frequency of all systems in the sequential gate design. This means that the maximum logical path is used to define maximum system period. In Figure 6, it is shown a schematic description for basic gates which can be added to Quartus software (Intel-Quartus, 2018). In this scheme, OR, AND, XOR gates and inverters of these gates is observed.

As shown in the figure, all gates excluding NOT gates have two inputs and one output. However, some gates can include more than two inputs in Quartus software. When it is evaluated by considering basic logical systems, the presented gates have more than two inputs. Therefore, it can be considered multiple inputs for basic logical gates.

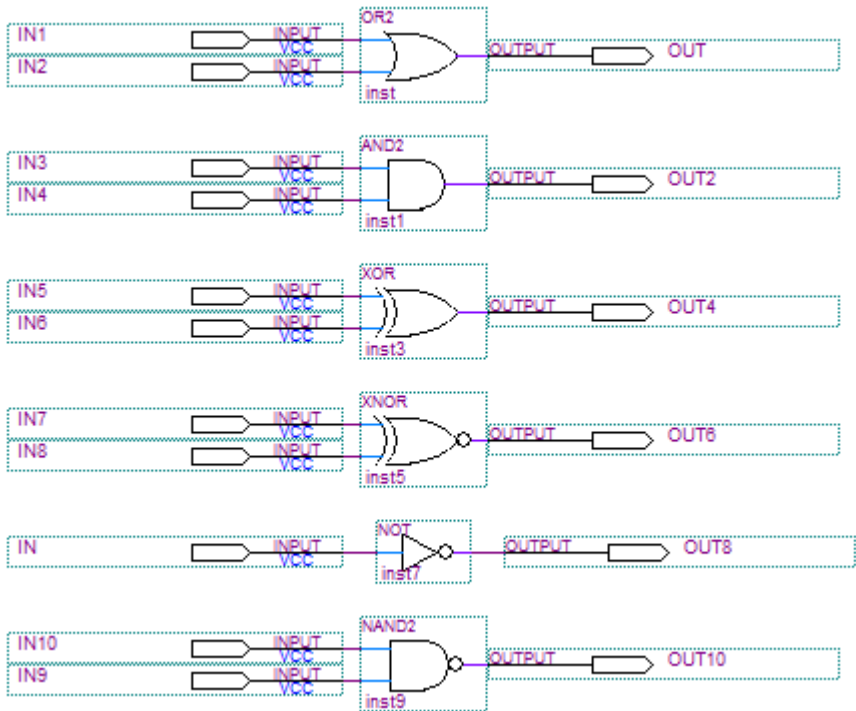



Figure-6 The schematic presentation of basic logical gates

The basic logical gates can be defined as a truth table which observes output case versus input signal cases. In another sections, it is given simulation results for these logical gates. In addition to this, some framework is described.

3.1 NOT Gate (Inverter)

Table-1 The truth table for NOT gate (Inverter)

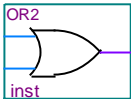
Symbol	I	O
	1	0
	0	1



According to the truth table given in Table 1, the not gate takes the inverse of the input. If the input bit is 1, the output takes '0'. Otherwise, the output is assigned as 1.

**3.2 OR Gate**

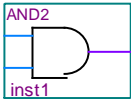
Table-2 The truth table for OR gate

Symbol	I <sub>1</sub>	I <sub>2</sub>	O
	0	0	0
	0	1	1
	1	0	1
	1	1	1

The OR gate is one of the most known logical gates due to its simple structure. The OR gate is given in the Table 2. As shown in the table, if any input variable takes a logical '1', the output level is determined at logical '1' level. If both inputs are being logical '0', the output has logical '0' level.

**3.3 AND Gate**

Table 3 The truth table for AND gate

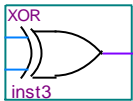
Symbol	I <sub>1</sub>	I <sub>2</sub>	O
	0	0	0
	0	1	0
	1	0	0
	1	1	1

A truth table for the AND gate is given in Table 3. Unlike the OR gate, the output has only logical '1' level when both inputs have logical '1' level.

Otherwise, the output is assigned as logical '0' level. Therefore, both inputs take logical '1' since the output takes logical '1'

**3.4 XOR Gate**

Table 4 The truth table for XOR gate

Symbol	I <sub>1</sub>	I <sub>2</sub>	O
	0	0	0
	0	1	1
	1	0	1
	1	1	0

This gate can be used in many communication systems such as CDMA (Code Division Multiplexing Access), Differential PSK (Phase Shift Keying). The XOR gate consists of OR, AND, and NOT gates. The NOT gate is integrated to AND gate. This logical gate is also defined as NAND gate of which truth table inverts output of AND gate. If both gates are same, the output takes logical '0'. The output has a logical '1' level if output 1 and output 2 is different from each other.

In Figure 7, it is given an equivalent circuit to the XOR gate. As shown in the figure, a XOR gate can be designed by using OR, AND, and NAND logical gates.

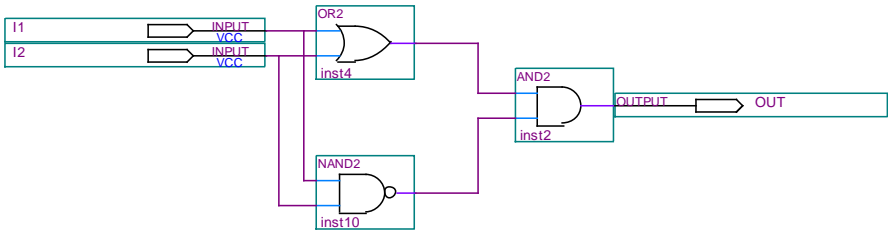


Figure-7 XOR Gate

4. Introduction to Simulation

In this section, it is described the simulation process for Quartus circuit designs. The simulation can be performed in both the Quartus compiler and the Modelsim-Altera program using a file produced by the compiler. This section introduces the simulation processes to be performed in the Modelsim-Altera Program.

In order to start the simulation process in the Modelsim Altera program, some settings must be executed in the compiler. These settings can be given as Figure 8 to generate the simulation file before start the simulation. As shown in the figure, the Tool name must be changed Modelsim-Altera in the simulation section consisted of Eda Tool Setting category.

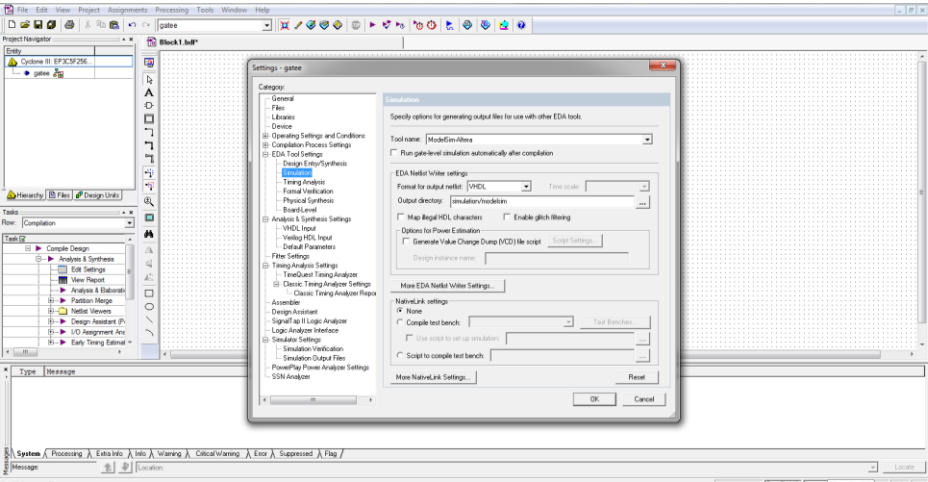


Figure-8 The generating of simulation file

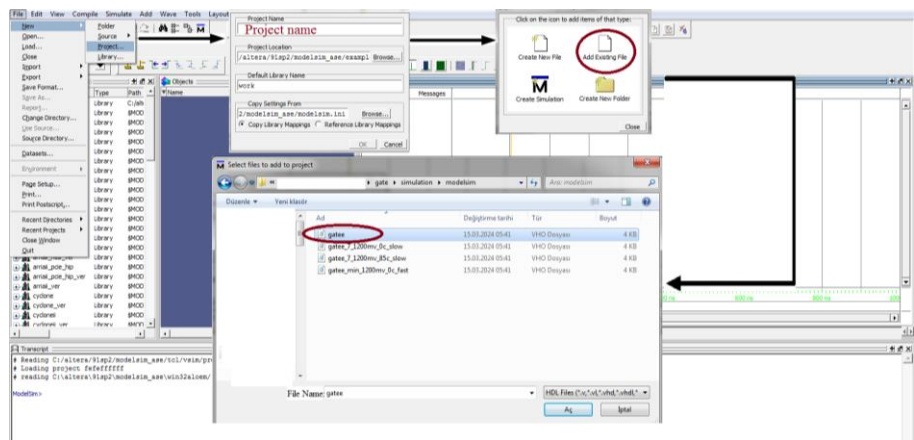


Figure-9 Modelsim-Altera Introduction

After generate the simulation file, the Next-project is chosen at the Modelsim-Altera Software. Afterwards, it is clicked the Add Existing File menu to add the generated file by Quartus compiler to Modelsim-Altera Software as shown in Figure 9. Finally, the simulation process can be started by inserting the generated simulation file.

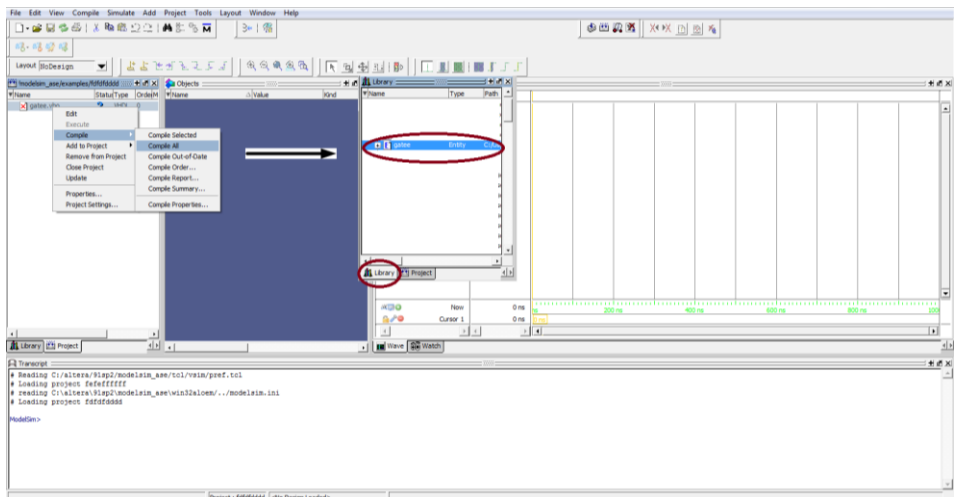


Figure-10 Open the Waveform Window

After insert the simulation file, it can be performed to transfer the objects of project for simulation process. In order to perform this stage, file with extension of .vho inside Project menu is right clicked. Then, this

file is provided the compiling by Modelsim Altera software. In order to observe the Objects, the project file inside Library menu is clicked.

4.1 The Simulation Stage for NOT Gate

In this section, it is given the attained simulation results related to NOT gate. It is shown the scheme of NOT gate created in Quartus software in Figure 11. In the last stage of simulation process, the objects name can be obtained in opened simulation window. The simulation parameters can be entered after these objects hold and release to Messages Menu. The IN and OUT variables is changed with time as shown in Figure 11. To start the simulation, a periodical signal that can be described as a clock signal is assigned for IN variable. Thanks to clock signal, the variation of OUT can be smoothly tracked. As shown in the figure, IN and OUT are inversed in each other.

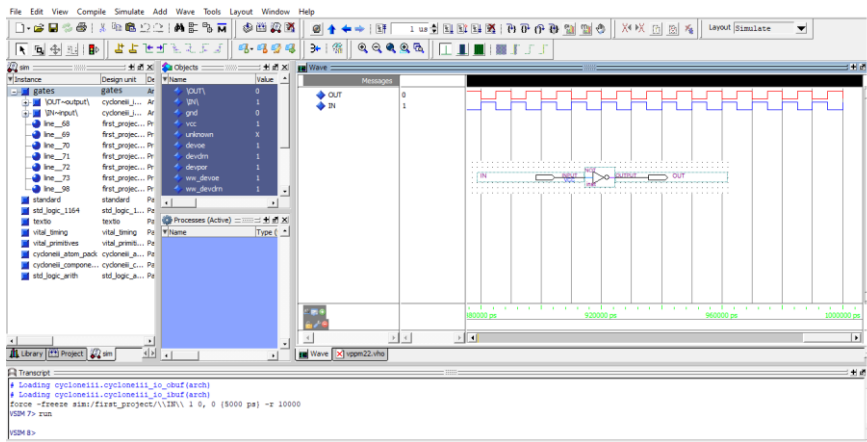


Figure-11 The simulation file for NOT gate

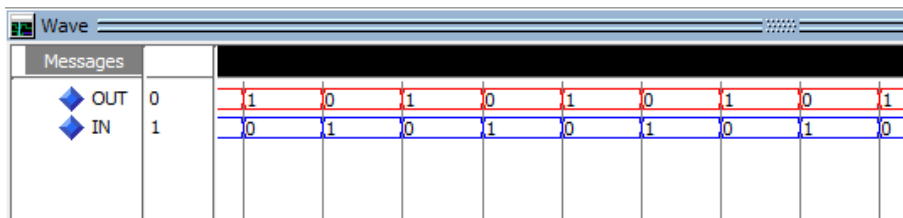
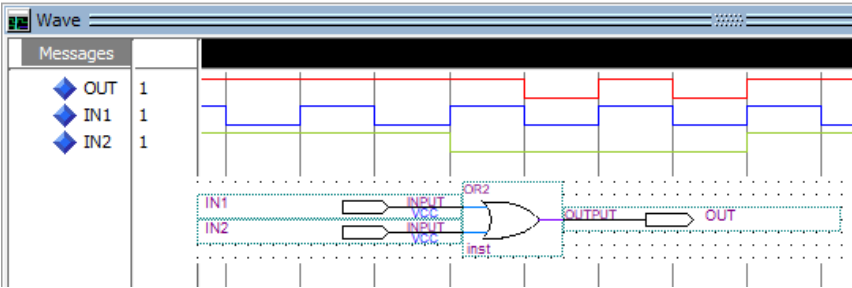


Figure-12 The Literal format results for NOT gate.

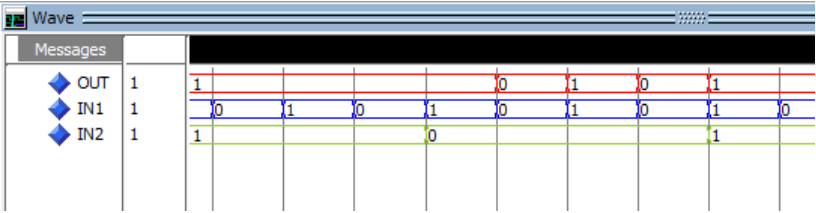
In Figure 12, a type of literal format is given for NOT gate. The inverted input signal can be observed from the output of gate.

**4.2 The simulation of OR Gate**

In this section, the simulation results of OR gate is presented by taking account for truth table given in Table 2. As mentioned previously, the output will be a logical '1' level if any input is assigned as logical '1'. In other case, the output will be logical '0' level if both inputs take logical '0' level.



(a)



(b)

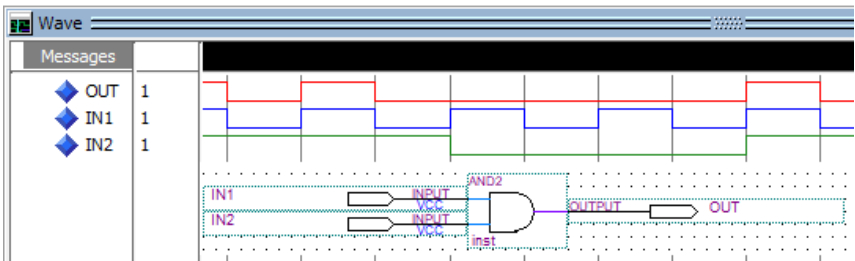
Figure-14 The simulation results for OR gate (a) Logic Format (b) Literal Format

In Figure 12, the simulation results are given for OR logical gate. In the figure, red line defines output signal while blue and yellow lines show input signals. As shown in the Figure, the OUT signal that presents output signal is logical '0' level when IN1 and IN2 indicate input signals logical '0' level.

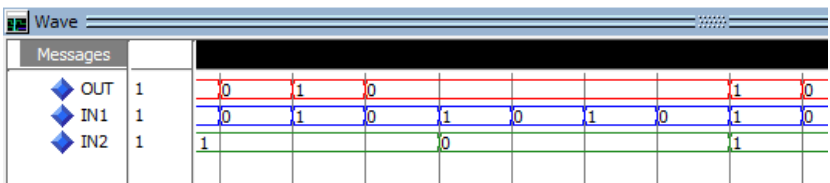


### 4.3 The simulation results for AND Gate

As shown in Table 3, output of AND gate gets logical '1' level if both inputs are logical '1' level. Otherwise, the output will be logical '0' level. In Figure 15, AND gate is simulated in Modelsim waveform software.



(a)

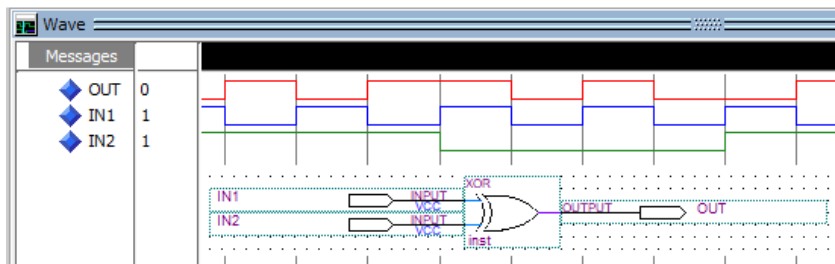


(b)

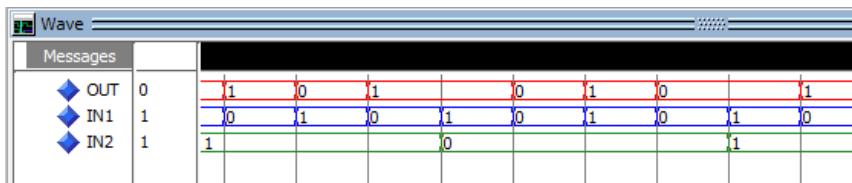
Figure-14 The simulation results for AND gate (a) Logic Format (b) Literal Format

### 4.4 The simulation results for XOR Gate

This simulation results are obtained for XOR gate which is designed by using various techniques. In the first project which is given in Figure 15.a and b, available block is used to observe the behavior of XOR gate. Another project, which consists of OR, AND, and NAND gates, is simulated in Figure 16.a and b. It is shown that there are similar results for both simulations.

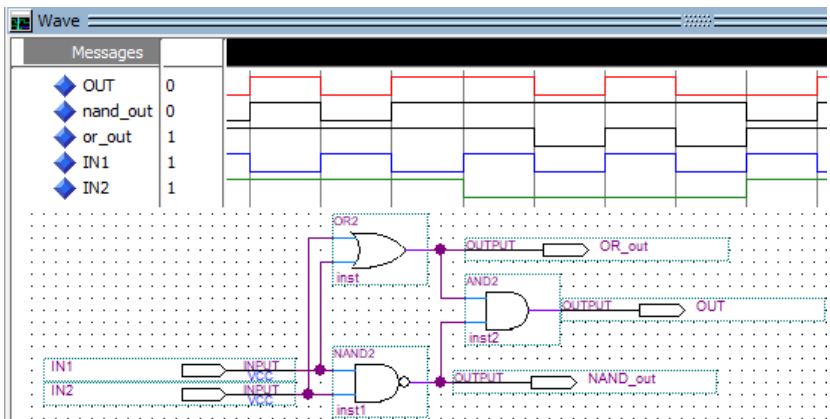


(a)

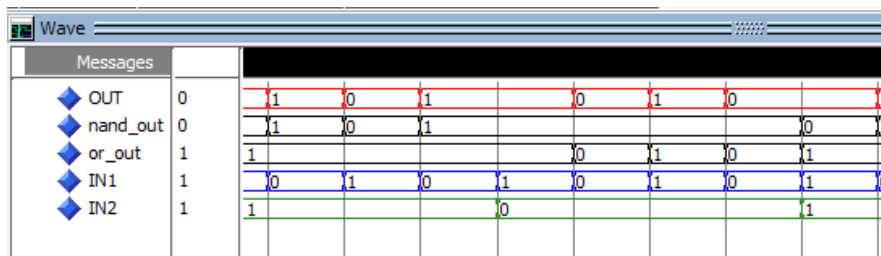


(b)

Figure-15 The simulation results for Quartus XOR gate (a) Logic Format  
(b) Literal Format



(a)



(b)

Figure-16 The simulation results for XOR gate consisted basic gates (a)  
Logic Format (b) Literal Format

**References**

Ayoub, H. G., Abdulrazzaq, Z. A., Fathil, A. F., Hasso, S. A., & Suhail, A. T. (2024). Unveiling robust security: Chaotic maps for frequency hopping implementation in FPGA. *Ain Shams Engineering Journal*, 15(11), 103016.

Bailey, D. G. (2023). *Design for embedded image processing on FPGAs*. John Wiley & Sons.

Bureneva, O., & Mironov, S. (2023). Fast FPGA-based multipliers by constant for digital signal processing systems. *Electronics*, 12(3), 605.

- Bruno, F., & Eschemann, G. (2024). The FPGA Programming Handbook: An essential guide to FPGA design for transforming ideas into hardware using SystemVerilog and VHDL. Packt Publishing Ltd.
- Chen, J., Wang, B., He, S., Xing, Q., Su, X., Liu, W., & Gao, G. (2023). HISP: heterogeneous image signal processor pipeline combining traditional and deep learning algorithms implemented on FPGA. *Electronics*, 12(16), 3525.
- Devadoss, D. K., & Ramapackiam, S. S. (2024). Fully parallel low-density parity-check code-based polar decoder architecture for 5G wireless communications. *ETRI Journal*, 46(3), 485-500.
- Ferraz, O., Subramaniyan, S., Chinthala, R., Andrade, J., Cavallaro, J. R., Nandy, S. K., & Falcao, G. (2021). A survey on high-throughput non-binary LDPC decoders: ASIC, FPGA, and GPU architectures. *IEEE Communications Surveys & Tutorials*, 24(1), 524-556.
- Intel® Quartus® Prime Standard Edition Handbook Volume 1, 2018.
- Kalomiros, J. A., & Lygouras, J. (2008). Design and evaluation of a hardware/software FPGA-based system for fast image processing. *Microprocessors and Microsystems*, 32(2), 95-106.
- Koch, D. (2012). Partial reconfiguration on FPGAs: architectures, tools and applications (Vol. 153). Springer Science & Business Media.
- Krishnamoorthy, R., Soubache, I. D., & Jain, S. (2022). Wireless communication based evaluation of power consumption for constrained energy system. *Wireless Personal Communications*, 127(1), 737-748.
- Lata, K., & Cenkeramaddi, L. R. (2023). FPGA-Based PUF Designs: A Comprehensive Review and Comparative Analysis. *Cryptography*, 7(4), 55.
- Lawal, N., Lateef, F., & Usman, M. (2015, June). Power consumption measurement & configuration time of FPGA. In *2015 Power Generation System and Renewable Energy Technologies (PGSRET)* (pp. 1-5). IEEE.
- Leung, B. (2011). *VLSI for wireless communication*. Springer Science & Business Media.
- McDonald, C., Abrudan, T. E., Cabral, F., Kucera, S., Claussen, H., Farrell, R., & Dooley, J. (2023). FPGA implementation of a sub-400ns 6G free-space optical wireless communications transmitter. *Optics Express*, 31(16), 25933-25942.
- Panda, A. K., Palisetty, R., & Ray, K. C. (2020). High-speed area-efficient VLSI architecture of three-operand binary adder. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(11), 3944-3953.

- Parr, E. A. (2013). *Logic designer's handbook: circuits and systems*. Elsevier.
- Ramírez-Montañez, J. A., Rangel-Magdaleno, J. D. J., Aceves-Fernández, M. A., & Ramos-Arreguín, J. M. (2023). Modeling of Particulate Pollutants Using a Memory-Based Recurrent Neural Network Implemented on an FPGA. *Micromachines*, 14(9), 1804.
- Sharobim, B. K., Yacoub, M. H., Sayed, W. S., Radwan, A. G., & Said, L. A. (2023). Artificial neural network chaotic PRNG and simple encryption on FPGA. *Engineering Applications of Artificial Intelligence*, 126, 106888.
- Sklyarov, V., Skliarova, I., Barkalov, A., & Titarenko, L. (2014). *Synthesis and Optimization of FPGA-based Systems (Vol. 294)*. Springer Science & Business Media.
- Tlelo-Cuautle, E., De La Fraga, L. G., & Rangel-Magdaleno, J. (2016). Engineering applications of FPGAs. *Engineering Applications of FPGAs*.
- Wang, X., He, X., Zhu, X., Zheng, F., & Zhang, J. (2024). Lightweight and Real-Time Infrared Image Processor Based on FPGA. *Sensors*, 24(4), 1333.
- Xia, H., Yu, X., Zhang, J., & Cao, G. (2024). A Review of Advancements and Trends in Time-to-Digital Converters Based on FPGA. *IEEE Transactions on Instrumentation and Measurement*.
- Yuvaraja, S., Khandelwal, V., Tang, X., & Li, X. (2023). Wide bandgap semiconductor-based integrated circuits. *Chip*, 100072.

# **Error Correction Algorithms: An Introduction to Previous Works, Theoretical Analysis and Implementation**

## **1. Introduction**

Communication systems have evolved over time to meet user demands such as high-speed data transmission (Krishna et al., 2021), coverage area (Arum et al., 2020), service continuity (Barzegar et al., 2020) etc (Wei et al., 2022). For this reason, new algorithms have been developed to solve communication problems (Liu et al., 2024; Dai et al., 2020) and to improve transmission performance (Liang and Wang, 2020). Especially thanks to wireless communication systems, data transmission has become easier in places where physical cable installation is difficult (Ramalingam and Shanmugam, 2022). One of the most significant disadvantages of wireless data communication in the free space environment is noise due to interference of signal with undesired frequency. The differences between transmitted and received bits in systems affected by noise have led to the definition of the term bit error rate in communication systems. This noise can be generally called AWGN (Additive White Gaussian Noise) in communication systems (Liu et al., 202) and can be generated and added to evaluate the performance of systems in communication simulations. If the difference between received and transmitted bits increases, Bit Error Rate (BER) of the transmission system will increase. Otherwise, the system's performance will decrease. A sample bit error rate can be given in Figure 1.



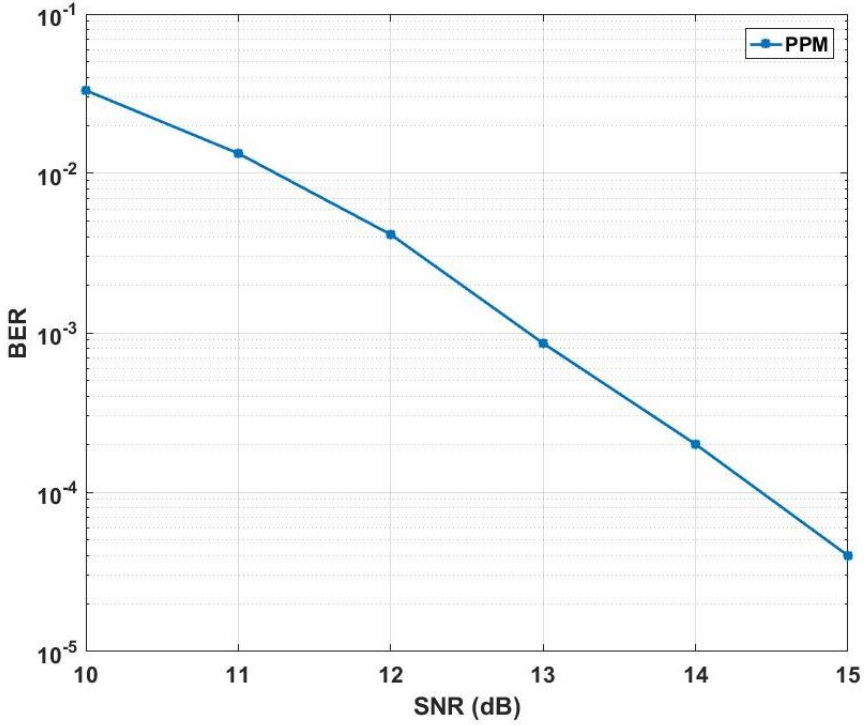


Figure 1. A BER performance for 2-PPM transmission scheme

In Figure, a BER simulation is represented for 2-PPM transmission scheme generally used in optical communication systems. According to the figure, the system requires 13dB Signal-to-Noise Rate (SNR) at the BER performance of  $10^{-3}$ . To decrease this BER, signal power can be re-adjusted. Therefore, as shown in the figure, the BER decreases while the SNR increases.

It is incredibly significant the providing of impeccable data transmission when the data transmission is performed over a communication channel. Therefore, the incorrect bits can be corrected by using improved methods for communication systems (Moon, 2020). These methods can be referred to error correction codes (Clark and Cain, 2013).

Error correction algorithms provide a control unit for data bits while transmitting redundant bits and data bits. Although the transmitting of redundant bits can decrease the data rate of communication system, they are used to correct the false bits (Yin et al., 2024). If the incorrect bits can be corrected by using an error correction method, transmission performance will be enhanced by decreasing the difference between received and transmitted bits. Although there are many error correction schemes (Chowdary et al., 2023; Ke, 2024), a simple error correction scheme can be given in Figure 2.

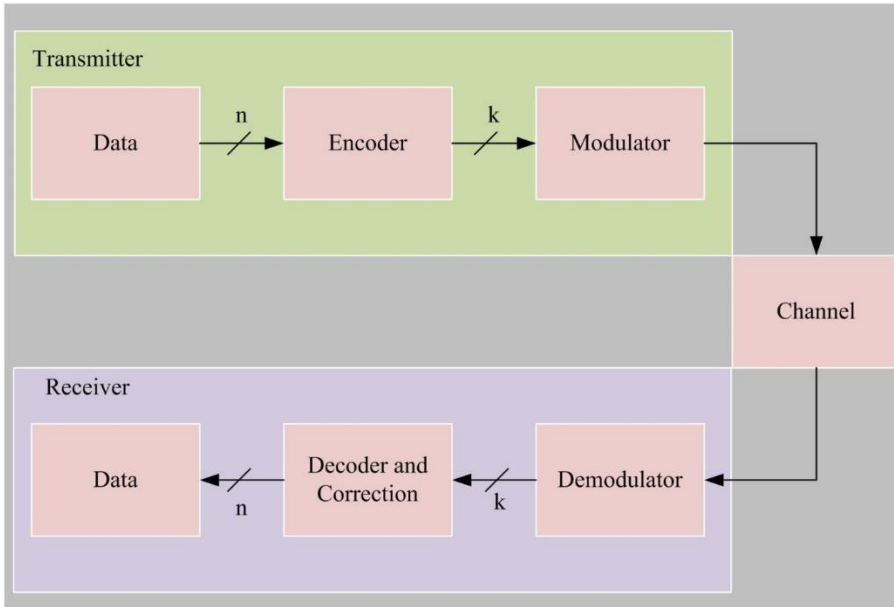
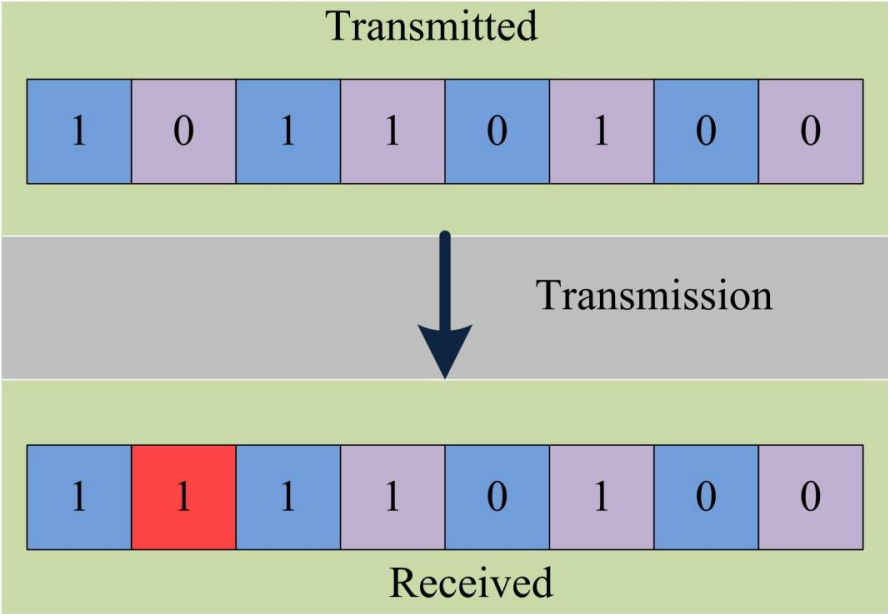


Figure 2. A Basic Error Correction Scheme

In Figure 2, the data is represented as transmitted message bits. The message bit's length is equal to  $n$  bit which is variable in terms of the type of error correction algorithm. The encoder block adds redundant bits to output of data block to shape transmitted signal. The length of data obtained from output of encoder is equal to  $k$ . This signal can also

be called encoded bits. After the coding stage, a modulation process is performed to transfer message signal to communication channel (Yang, 2018; Ryan and Frater, 2002). The transmitter can be considered as a unit where the whole encoding and modulating process can be carried out. In addition to this, the communication channel can be defined as wireless or wire-line medium where noisy signal is added to modulated signal.

The demodulator unit aims to the detecting of bits which are applied on modulator unit at the transmitter side. Hence, the demodulator can be considered as one of the significant units of the Receiver. The bits with length of  $k$  are passed through Decoder to correct the error bits and to remove redundant bits from received bits. In the final stage, the data can be obtained with  $n$  bits. An error correction process is shown in Figure 3.



Şekil 3 An Error Correction Process

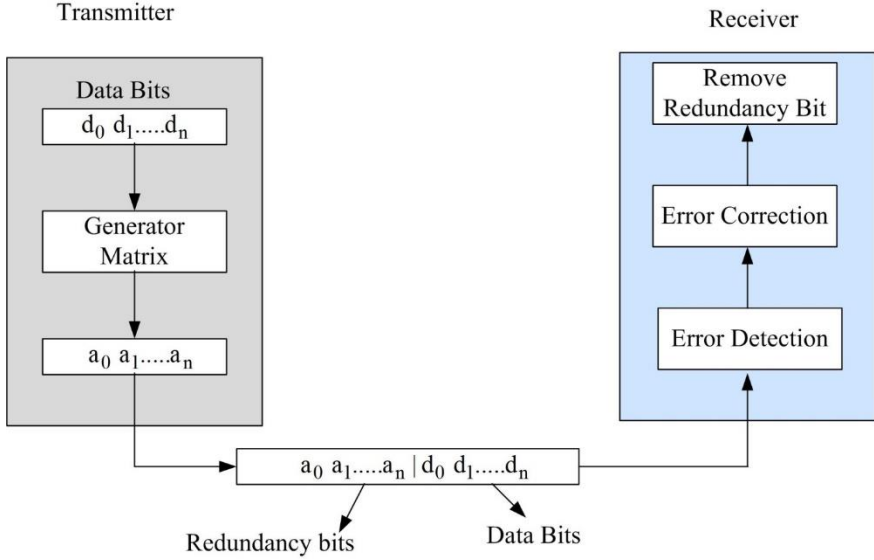


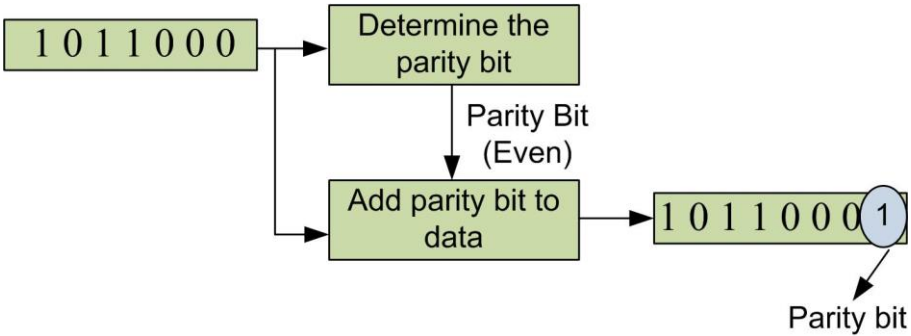
Figure 4. Redundancy Concept

As shown in Figure 3, the transmitted bit vector is "10110100" that consists of 8 bits. However, the received bit vector is obtained as "11110100" after the transmission process. It is shown that there is a bit error in received bit vector due to data transmission. This case can be defined as a single bit error. To detect single bit error in received bit vector, one of the simplest error detection methods is parity check or Vertical Redundancy Check (VRC) algorithm in the literature. In addition to this method, Longitudinal Redundancy Check (LRC), Cyclic Redundancy Check (CRC), and hamming coding (Sunshine, 2013; Agarwall and Tayal 2009) can be used to detect or correct bit errors in communication systems. These algorithms consist of variable complexity architecture due to their different theoretical framework. A redundancy idea must be applied for mentioned techniques (Subramanyam, 2005). The redundancy concept can be given in Figure 4.

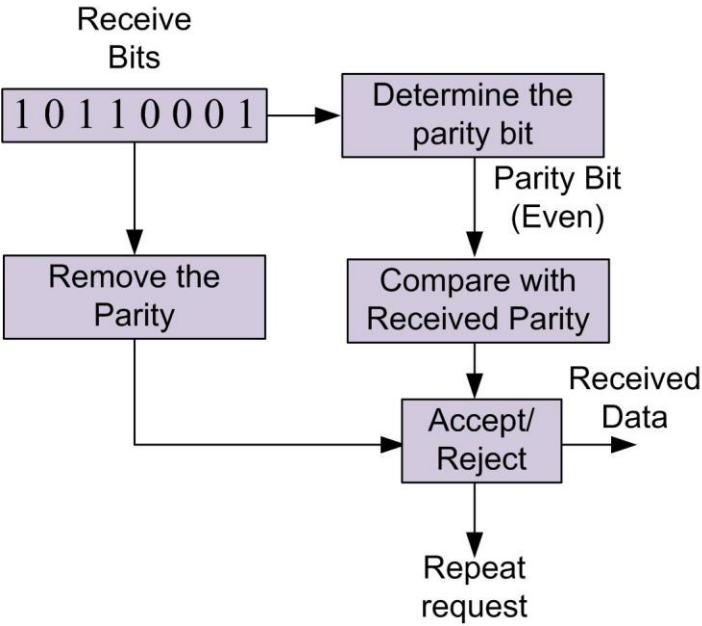
## 2. Vertical Redundancy Check

The Vertical Redundancy Check (VRC), which is also referred to parity check, is one of the most basic error detections. This technique detects

error bits when the number of incorrect bits is odd. This is one of the most significant disadvantages for Vertical Redundancy Check. A sample Vertical Redundancy Check algorithm can be given as shown in Figure 5 (Subramanyam, 2005).



(a)



(b)

Figure 5. A block diagram for VRC algorithm. (a) The transmitter side.  
(b) The receiver side

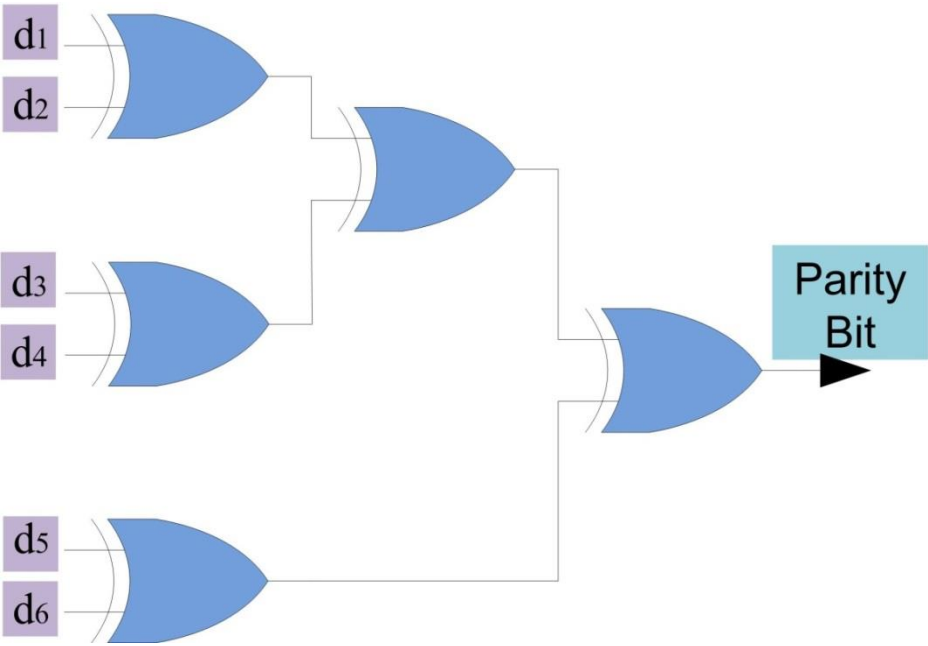


Figure 6. A simple architecture for generating of Parity Bit

In Figure 5 (a), the first stage is determining of parity bit to make a decision related to transmitted bit. The data bit must be transmitted with parity check bit to detect error bit at the receiver side of which structure is given in Figure 5 (b). The receiver determines the parity bit by taking account of the received data bit. Comparing parity bit with data bits, it makes a decision whether the received bit can be accepted. To generate the parity bit, an architecture is given in Figure 6. As shown in the figure, the architecture is based on XOR gates. This structure is referred to even parity scheme.

### 3. Longitudinal Redundancy Check

To determine the location of error bit, it is improved a two-dimension Vertical Redundancy Check architecture called Longitudinal Redundancy Check (LRC). This technique has a complex structure



compared to VRC due to its row length (Gupta, 2006). A sample LRC structure can be given as shown in Figure 7.

1	0	1	0	1	1	0	0
0	1	1	0	1	1	1	1
1	1	0	1	1	0	0	0
0	0	0	0	1	1	0	0
1	1	0	0	0	1	0	1
1	0	1	1	1	1	0	0
0	0	1	0	0	1	1	1
0	1	0	0	1	0	0	1

Figure 7. A LRC sample of data structure

**4. Hamming Coding**

Hamming codes are one of the linear block codes to use to improve the transmission performance of communication system in terms of bit error rate (Desourdis, 2002). Hence, this linear code type has attracted the attention from many researchers in literature. This is because this coding scheme has a very simple algorithm which allows to add error correction codes to data bits.

If  $m$  is defined as the number of data bit that will be conveyed through a communication infrastructure, the number of encoded bits  $k$  with using hamming method can be expressed as  $2^m-1-m$ . If  $n$  is equal to  $2^m-1$ , this hamming system can be defined as  $(n,k)$ . The hamming systems can be arrayed (7,4), (15,11), (31,26) and etc. In this section, we give (7,4) hamming coding (Roman, 1992).

A generator matrix is required to code data bits by using Hamming technique. The generator matrix consists of a  $k \times k$  size of Identity Matrix. An Identity Matrix for  $k$  of 4 can be given by

$$\mathbf{I}_k = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)$$

The generator matrix  $G$  can be obtained by using a matrix of  $4 \times 3$  and Identity Matrix  $\mathbf{I}_k$ . Identity matrix and  $\lambda_{k,n-k}$  matrix can create a sample generator matrix which can be expressed by (Neubauer et al., 2007),

$$\lambda_{k,n-k} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad (2)$$

$$\mathbf{G} = (\mathbf{I}_k | \lambda) \quad (3)$$

$$\mathbf{G} = \left( \begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array} \right) \quad (4)$$

where,  $G$  matrix of which size is  $k \times n$  created by  $\mathbf{I}_k$  and matrix of  $\lambda_{k,n-k}$  (Roman, 1992). According to encoding procedure, encoded bits, which are given in Figure 2 at the input of modulator, are obtained by exclusive-or (XOR). The encoding process can be given as follows:

$$\mathbf{c} = \mathbf{dG} \quad (5)$$

$$\left( \underbrace{c_0 \ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6}_c \right) = \left( \underbrace{d_0 \ d_1 \ d_2 \ d_3}_d \right) \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \quad (6)$$

where, c and d are defined as encoded and data bits, respectively. The d and G matrix are passed through exclusive-or gates. In Equation (5), the c and d matrix have 7 and 4 elements, respectively.

$$\begin{aligned} c_0 &= d_0 \\ c_1 &= d_1 \\ c_2 &= d_2 \\ c_3 &= d_3 \\ c_4 &= d_1 \oplus d_2 \oplus d_3 \\ c_5 &= d_0 \oplus d_2 \oplus d_3 \\ c_6 &= d_0 \oplus d_1 \oplus d_3 \end{aligned} \quad (7)$$

According to equation (7),  $d_0$ ,  $d_1$ ,  $d_2$ , and  $d_3$  bits are directly equal to encoding bits of  $c_0$ ,  $c_1$ ,  $c_2$ , and  $c_3$ . In decoding side, these bits will be directly assigned to data bits if error matrix is equal to 0. The  $c_4$ ,  $c_5$ , and  $c_6$  bits are obtained by using X-OR gates as shown in Figure 8.

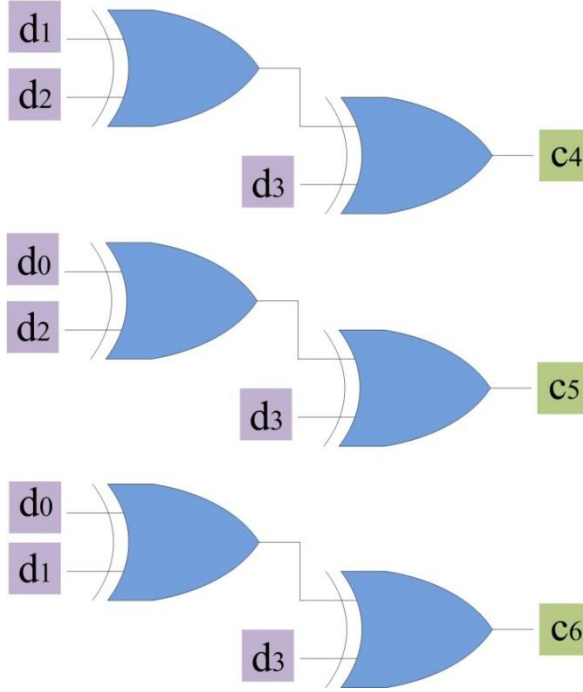


Figure 8. A simple XOR architecture for encoded bits of  $c_4$ ,  $c_5$ , and  $c_6$ .

In the decoder side, it is used an error correction and detection matrix, which is defined as parity-check matrix, by taking account for the type of Generator matrix. This matrix can be given by

$$H = \left( \lambda_{k,n-k}^T \mid I_{n-k} \right) \quad (8)$$

$$\lambda_{k,n-k}^T = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \quad (9)$$

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (10)$$

It must be transposed matrix given in Eq. (10) to achieve the error matrix. To detect error, the encoded bits are multiplied by matrix of  $H^T$ . The error matrix can be defined as follows:

$$(r_0 \ r_1 \ r_2) = cH^T = (c_0 \ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6) \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (11)$$

$$\begin{aligned} r_0 &= c_1 \oplus c_2 \oplus c_3 \oplus c_4 \\ r_1 &= c_0 \oplus c_2 \oplus c_3 \oplus c_5 \\ r_2 &= c_0 \oplus c_1 \oplus c_3 \oplus c_6 \end{aligned} \quad (12)$$

Table 1. r matrix versus Error Bit

Error Bit	$[r_0 \ r_1 \ r_2]$
$c_0$	0 1 1
$c_1$	1 0 1
$c_2$	1 1 0
$c_3$	1 1 1
$c_4$	1 0 0
$c_5$	0 1 0
$c_6$	0 0 1

In Equation 12, r matrix is defined as error matrix. To detect errors among the c matrix, it takes account for  $r_0$ ,  $r_1$  and  $r_2$  values. For example, if  $[r_0 \ r_1 \ r_2]$  is equal to  $[0 \ 1 \ 0]$ , the encoded bit of  $c_5$  is inverted. Therefore, bit error rate performance can be improved by detecting and correcting received bits. As shown in Figure 6, hamming coding is applied on PPM

transmission scheme. In addition to this, a table can be given for correction of error bits. In table 1, a mapping is presented to correct the error bit in terms of  $[r_0 \ r_1 \ r_2]$  values.

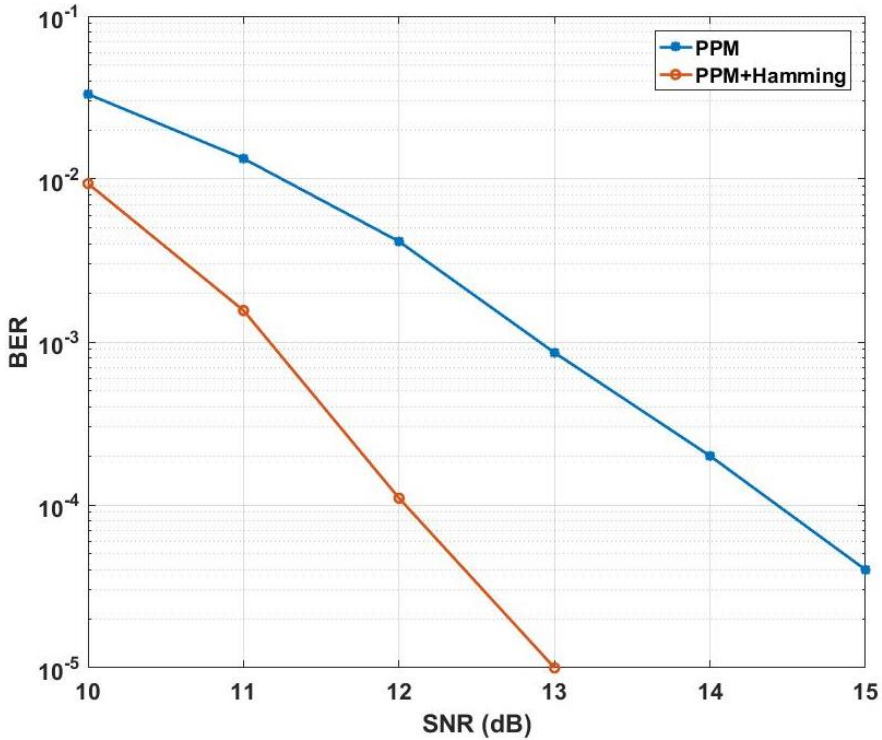


Figure 9. BER performance for 2-PPM with Hamming code

In Figure 9, a bit error rate performance has been given for 2-PPM system. The red line result observes BER performance by using Hamming technique. Other plotting has been obtained by using traditional receiver system. As shown in the figure, the Hamming method can correct the incorrect bits.

## References

- Agarwal, B. B., Tayal, S. P. (2009). Computer Network. India: University Science Press.
- Arum, S. C., Grace, D., & Mitchell, P. D. (2020). A review of wireless communication using high-altitude platforms for extended coverage and capacity. *Computer Communications*, 157, 232-256.
- Barzegar, H. R., El Ioini, N., & Pahl, C. (2020, April). Wireless network evolution towards service continuity in 5G enabled mobile edge computing. In 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC) (pp. 78-85). IEEE.
- Chowdary, M. K., Turaka, R., Alabdullah, B., Khan, M., Babu, J. C., & Kiran, A. (2023). Low-power very-large-scale integration implementation of fault-tolerant parallel real fast fourier transform architectures using error correction codes and algorithm-based fault-tolerant techniques. *Processes*, 11(8), 2389.
- Clark Jr, G. C., & Cain, J. B. (2013). Error-correction coding for digital communications. Springer Science & Business Media.
- Dai, L., Jiao, R., Adachi, F., Poor, H. V., & Hanzo, L. (2020). Deep learning for wireless communications: An emerging interdisciplinary paradigm. *IEEE Wireless Communications*, 27(4), 133-139.
- Desourdis, R. I. (2002). Emerging Public Safety Wireless Communication Systems. United Kingdom: Artech House.
- Gupta, P. C. (2006). Data Communications and computer networks. India: PHI Learning.
- Ke, X. (2024). Error-Correction Coding. In *Handbook of Optical Wireless Communication* (pp. 581-613). Singapore: Springer Nature Singapore.
- Krishna, C. M., Das, S., Nella, A., Lakrit, S., & Madhav, B. T. P. (2021). A micro-sized rhombus-shaped THz antenna for high-speed short-range wireless communication applications. *Plasmonics*, 16(6), 2167-2177.

- Liang, R., Zhao, L., & Wang, P. (2020). Performance evaluations of LoRa wireless communication in building environments. *Sensors*, 20(14), 3828.
- Liu, Y. F., Chang, T. H., Hong, M., Wu, Z., So, A. M. C., Jorswieck, E. A., & Yu, W. (2024). A survey of recent advances in optimization methods for wireless communications. *IEEE Journal on Selected Areas in Communications*.
- Liu, Z., Liu, J., Zeng, Y., & Ma, J. (2020). Covert wireless communication in IoT network: From AWGN channel to THz band. *IEEE Internet of Things Journal*, 7(4), 3378-3388.
- Moon, T. K. (2020). *Error correction coding: mathematical methods and algorithms*. John Wiley & Sons.
- Neubauer, A., Freudenberger, J., Kuhn, V. (2007). *Coding Theory: Algorithms, Architectures and Applications*. Germany: Wiley.
- Ramalingam, S. P., & Shanmugam, P. K. (2022). A comprehensive review on wired and wireless communication technologies and challenges in smart residential buildings. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, 15(9), 1140-1167.
- Roman, S. (1992). *Coding and Information Theory*. Germany: Springer.
- Ryan, M. J., & Frater, M. (2002). *Communications and information systems*. Argos Press P/L.
- Subramanyam, M.V. (2005). *Switching Theory and Logic Design*. (2005). India: Laxmi Publications Pvt Limited.
- Sunshine, C. A. (2013) *Computer Network Architectures and Protocols*. United Kingdom: Springer US.
- Wei, Z., Wang, Z., Zhang, J., Li, Q., Zhang, J., & Fu, H. Y. (2022). Evolution of optical wireless communication for B5G/6G. *Progress in Quantum Electronics*, 83, 100398.
- Yang, S. M. M. (2018). *Modern digital radio communication signals and systems*. Springer.
- Yin, P., Chen, H., Xia, Y., Zhang, J., Liu, M., Gu, C., ... & Tang, F. (2024). High Logic Density Cyclic Redundancy Check and Forward Error Correction Logic Sharing Encoding Circuit for JESD204C Controller. *IEEE Transactions on Circuits and Systems I: Regular Papers*.



## **Simulation Results for Digital Designs**

The FPGA board is used to process data at high-speed switching in many projects (Khedkar and Khade, 2017; Luo et al., 2024; Nagar et al., 2024). In simulation stage, one of the most important issues is the proposed algorithm which is designed for specific tasks. There is a specific period for algorithms which are designed by using any language (Xu et al., 2024; Zhang et al., 2024). Since the period of algorithms can be variable in terms of data processing rate, different algorithms that give same output can be created in software programmer (Hussain and Sarkar, 2024; Hong et al., 2023).

Some experimental systems have been created at gate-based systems which are called VLSI (Very-Large-Scale Integration) (Lin et al., 2015; Lin et al., 2016). In these systems, the data processing rate of system is determined considering the longest path by the compiler (Yuan and Parhi, 2013). Specifically, many researchers focus on the design of error correction codes by using VLSI technologies due to their high-speed data processing rate (Naveen et al., 2023; Singh et al., 2023; Masera et

al., 1999). Compared with FPGA board, VLSI-based systems have more complex structures in terms of experimental systems.

In the previous sections, the applications of the blocks in the Quartus software environment are given. In addition to this, a theoretical framework is presented related to error correction algorithms. Since it is quite difficult the modifications of the codes of the blocks used in the Quartus program, designers have proposed inflexible algorithms. Firstly, it is very important to use the control panel to ensure synchronization in block based digital designs. It is not needed to use a control panel by using pipeline blocks which utilize appropriate number format. This format can be referred to as decimal number format. However, floating point-based blocks that are available in Quartus software require the use of a control panel to be complex. If floating point-based blocks are designed by using VHDL while considering the pipeline system, an output signal can be obtained at each clock signal.

In this section, VHDL code architectures, firstly, are given related to a counter and a comparator. Then, the design of an accumulator block is represented to express the integral operation. Additionally, a Linear Feedback Shift Register (LFSR) and Hamming coding system is given to introduce the random process and the error correction systems. Specifically, the LFSR system can be used to generate noise in wireless communication systems. These systems are designed by using VHDL code structure. The hamming code structure consists of a few VHDL based code systems. However, these blocks can be combined with each other by using VHDL codes. In this section, we give a combination of VHDL blocks to design Hamming code architecture. In addition to this, some simulation results can be given related to the encoding and decoding process. At the first time, an introduction is given according to basic VHDL code architecture.

## **1. Introduction to VHDL**

In this section, it is given a basic VHDL code structure. The VHDL code structure mainly consists of three sections (Pedroni, 2020): The

Library, Port definition and code section. In the Library section, related libraries are called to compile the VHDL code. In the code section, start and outcome variables are added to define the specific operation. This section determines the type of code operating, such as sequential operations.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  = ENTITY name IS
5
6  = PORT (
7
8  END name;
9  = ARCHITECTURE converter OF name IS
10 = BEGIN
11 END converter;
```

Library

Port Definition

Architecture

Figure 1. VHDL code structure

### 1.1. Counter

Firstly, a VHDL code structure is given to design a counter block. The counter block is commonly used for many systems, such as ROM applications and signal processing implementations. In ROM applications, the counter is utilized to point out the address line of related memory block. As shown in Figure 2, a VHDL code structure is presented for counter blocks.

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ENTITY counter IS
4  PORT (
5      clk: IN STD_LOGIC; n: in integer range 0 to 50;
6      counter: buffer integer range 0 to 50);
7  END counter;
8  ARCHITECTURE behv OF counter IS
9  BEGIN
10 PROCESS (clk)
11 BEGIN
12 IF (clk'EVENT AND clk='1') THEN
13 IF counter=n then
14     counter<=0;
15 ELSE
16     counter<=counter+1;
17 END IF;
18 END IF;
19 END PROCESS;
20 END behv;

```

Figure 2. VHDL code for counter block

## 1.2. VHDL to Block Scheme and Simulation Setup

In addition to available blocks in Quartus software, a block scheme can be created by using VHDL or Verilog in compiler environment. In another definition, VHDL code structure can be converted to .bdf file. This process will provide flexibility conditions to design specific architecture created by users. To implement this .bdf file to designs, a starting window can be given as shown in Figure 3.

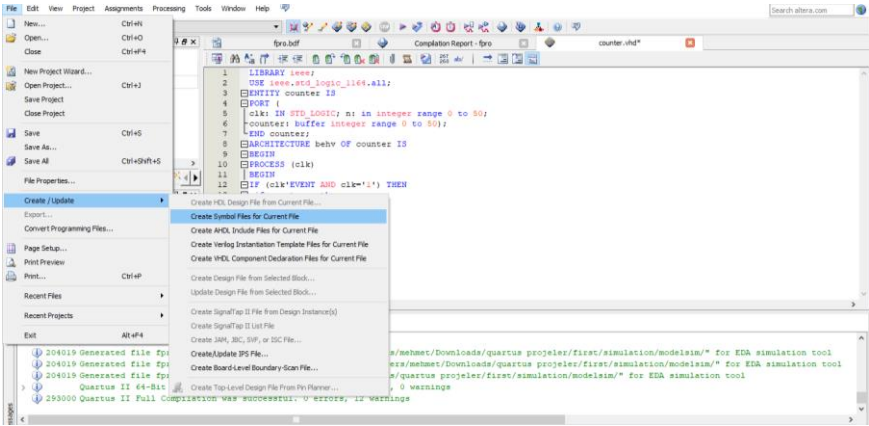


Figure 3. VHDL to .bdf

To start the conversion process, *Create Symbol Files for Current File* as shown in Figure 3. Afterwards, as illustrated in Figure 4, it is shown the created block scheme in .bdf file. This figure represents the counter blocks scheme which consists of the VHDL code given in Figure 3. Block Project menu consists of these block schemes.

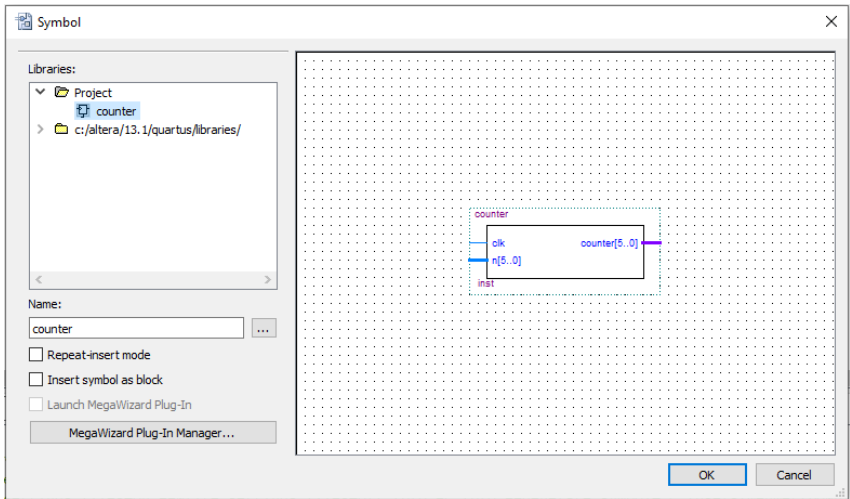


Figure 4. VHDL to .bdf

According to VHDL code given in Figure 2, this block has one input and one output signal as clock signal and counter. Since the counter output is defined between 0 and 20, the number of bits is determined as

5. In Figure 5, it is an image of which pin is assigned by an automatic pin assignment method in Quartus compiler. In the figure, the expression of counter [4..0] indicates the counter output being 5 bits. The simulations given in this section are created by using Vector Waveform File simulator where is located on Modelsim-Altera program which is combined to Quartus software. To open the Vector Waveform File, it must be chosen as the Vector Waveform File button where it is placed in the menu bar.

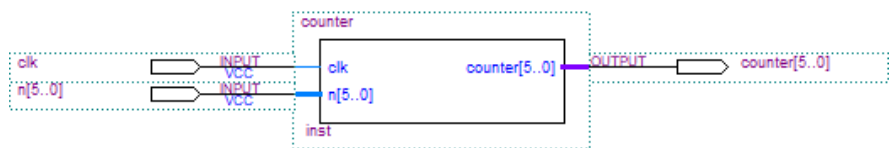


Figure 5. Input and output signals for Counter block

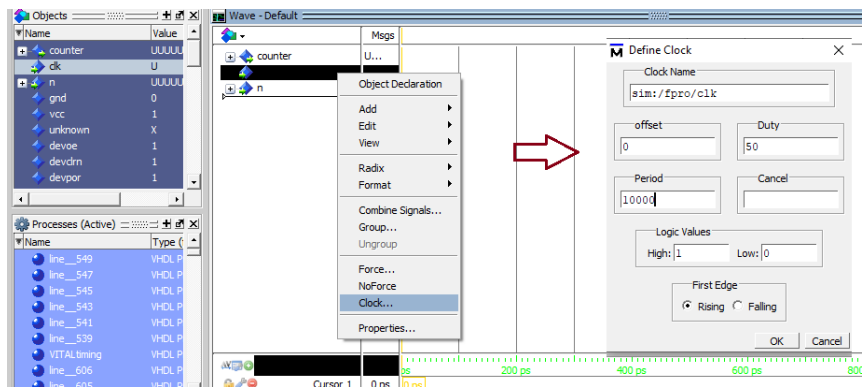


Figure 6. Starting of Vector Waveform File simulation (10000ps)

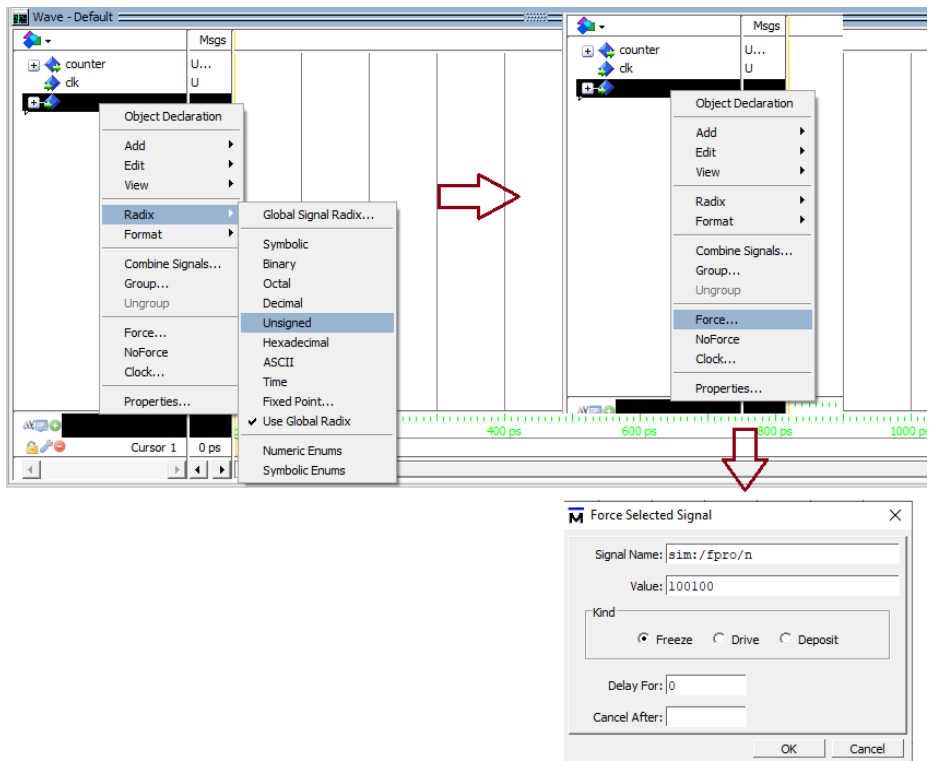


Figure 7. Adjusting of input and output for simulation

To plot the waveform, a clock signal must be assigned by taking account of frequency value. The clock signal has square wave form. During the clock time, the whole code line is processed. As shown in Figure 2 and Figure 3, upper counting is carried out at each clock signal. Hence, it must be assigned a frequency and type of clock signal to observe the simulation results. In Figure 6, firstly, a clock signal of which frequency is adjusted as 10000ps (10ns) is defined. Additionally, the clk input must be defined as clock signal in Define Clock Window.

In simulation stage, another process is assigned input and output as shown in Figure 7. For counter implementation, the  $n$  input indicates the upper limit of counter block. Hence, this available is very significant in this experimental system. The output of counter will give sequential number up to  $n$  value. In order to correct the output of counter, the

number format of output can be changed with another number format. In this implementation, the radix format is selected.

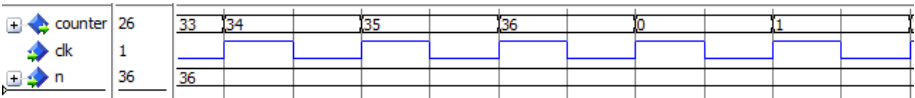


Figure 8. The simulation result for counter block

As shown in Figure 8, the counter output rises to 36. Then, it is made reset to re-count the counter block. The clk, which is called clock signal, has literal form and 50% dimming level.

**1.3. Accumulator**

The accumulator can be used to determine integral results for any system such as signal processing and communication systems. For example, communication systems have utilized the accumulator to take the integral of received signal during one symbol transmission. In these systems, the accumulator adds input signal with output signal at each period. The added signal is transferred to the next block at the end of the period. Finally, the output of the accumulator is cleared to implement the new signal. In this section, it has been given two accumulator designs. The first block continuously generates output signal. Output of others is only activated at the end of the period.

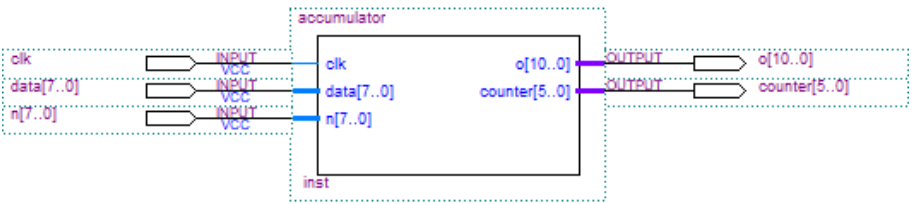


Figure 9. Accumulator block

Appendix A1 represents a VHDL code structure for accumulator block. After this code is converted to .bdf file as shown in Figure 9, compiled design can be simulated by using University Program VWF added to



Quartus 13 software. The VWF file can be created with a new section given in Figure 10.

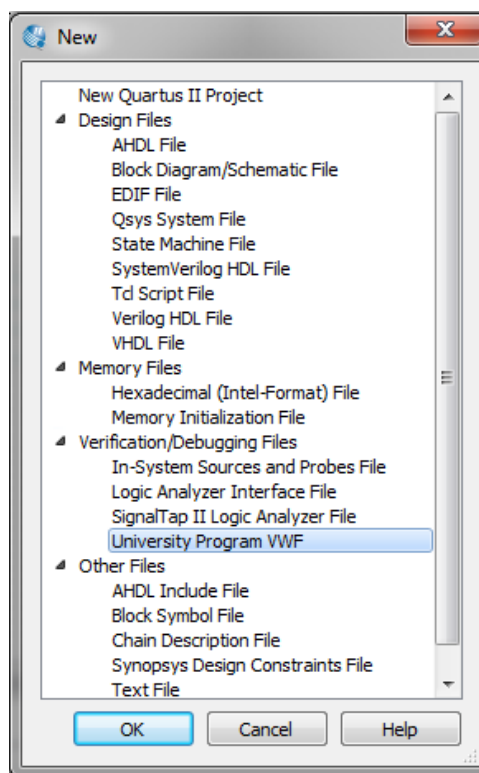


Figure 10. Introductions to Simulation in Quartus 13

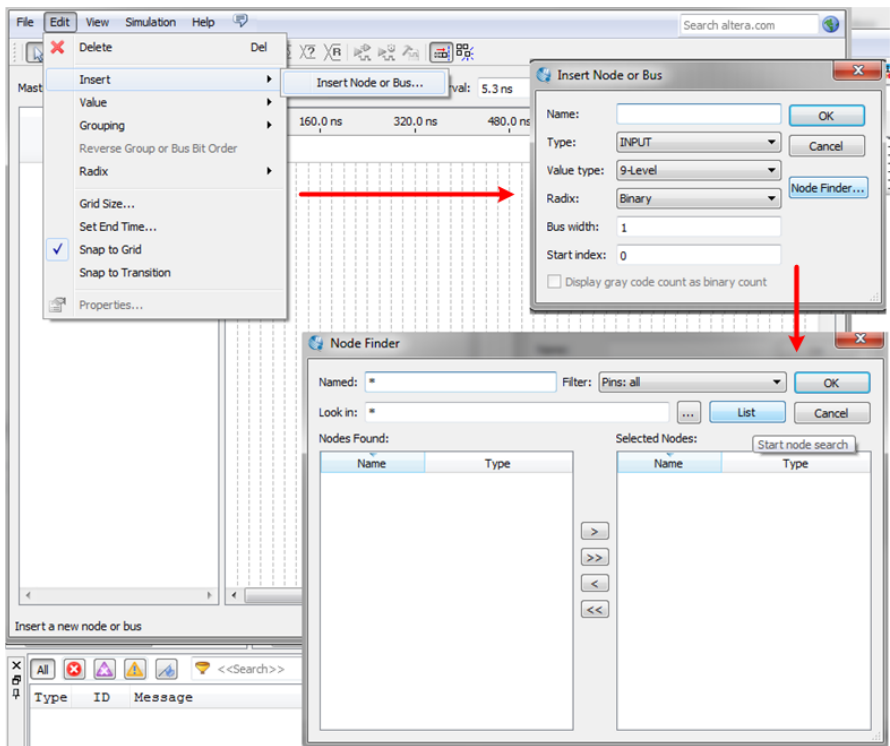


Figure 11. Introduction to Adding variable to VWF file

As observed in Figure 9, there are three inputs and two outputs signal for accumulator block. To make a simulation versus time axes, these inputs must be added to the VWF project simulation file. Hence, the Node Finder will be operated from Insert Node or Bus box. The input and output signals are added to the VWF file from the opened window of Node Finder.

When considering the project complexity, variable project time can be created by using the Set End Time button added in Edit menu. This is a very significant setting to observe whole simulation timing.

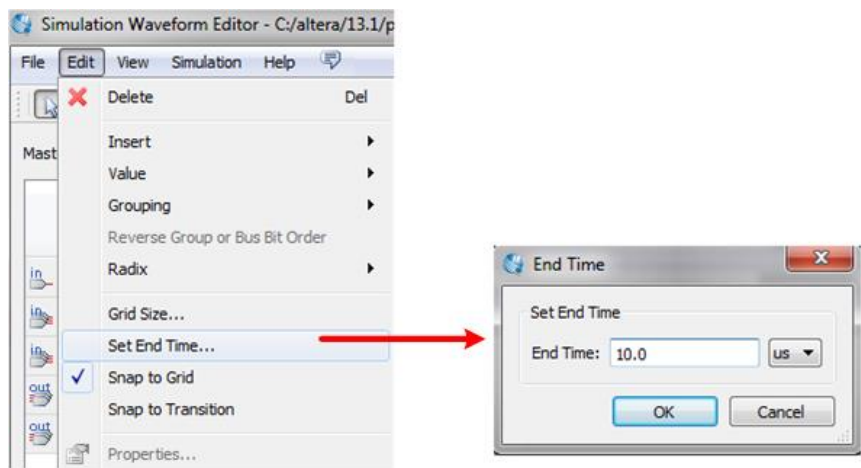


Figure 12. Adjust to the simulation length

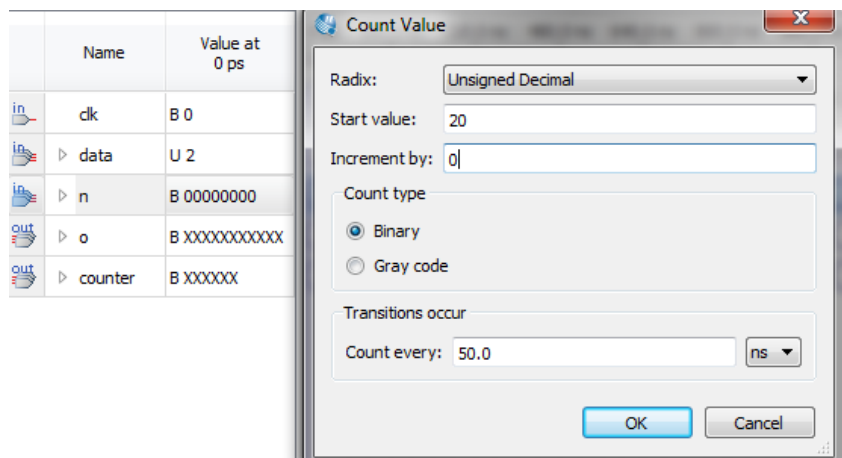


Figure 13. Assign to constant values

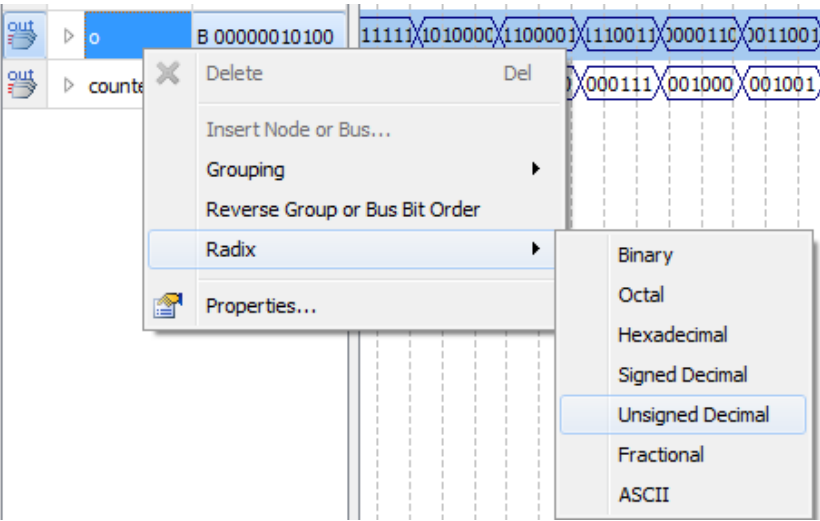
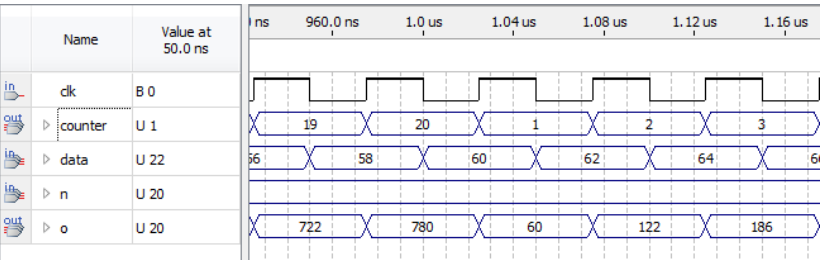
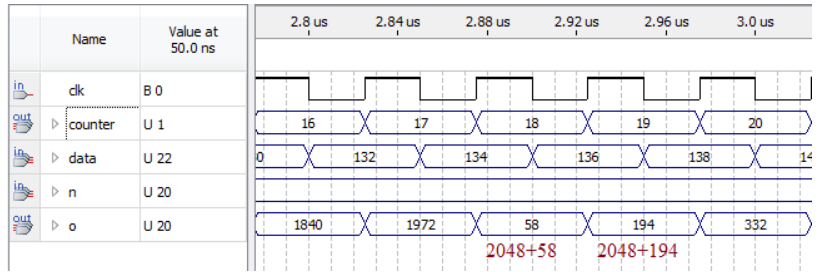


Figure 14. Change to number format



(a)



(b)

Figure 15. Simulation results for Accumulator. (a) non-overflowing case (b) overflowing case

In the last section of adjusting the parameter, it is assigned to constant values and changed to number format from simulation results. According to the accumulator block scheme and VHDL code given in Appendix A-1, a constant value is determined as using period parameter. This parameter is defined as  $n$  in both block scheme of accumulator and Figure 13. The number format of final output, which is referred to variable of  $o$ , can be changed to observe by using decimal format instead of binary format in Figure 14. Additionally, the counter variable counts to  $n$  value in VHDL code of accumulator block. It is shown that simulation results provide the code structure as shown in Figure 15 a. which represents non-overflowing case of output signal. However, Figure 15 b observes an overflowing case since the output is insufficient bit size. After output of 1972, the output signal  $o$  must get 2016. However, it is limited since its bit size is equal to 11 bits.

#### 1.4. LFSR

The LFSR is used for many signal processing implementations including encryption of the data (Wu et al., 2018; Win and Kyaw, 2008), generation of noise (Babu and Anand, 2020; Hazwani, 2014). Although there are several types of LFSR, the complexity of LFSR increases when its order rises. A LFSR VHDL code structure is given in Appendix A-2. According to this VHDL code, 16-bit LFSR is implemented to observe the behavior of LFSR architecture. Additionally, it has been seen that the LFSR structure consists of XOR block which is one of the simplest logic gates. In Figure 16, it is given a LFSR block scheme created by using this code. As shown in the block scheme, LFSR output has 16 bits which are encoded by using XOR gates. The encoded process is realized during one clock signal which is defined as input of  $clk$ .

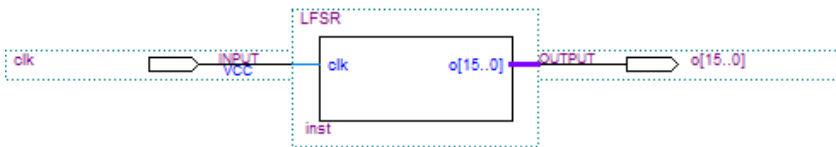


Figure 16. LFSR Block Scheme

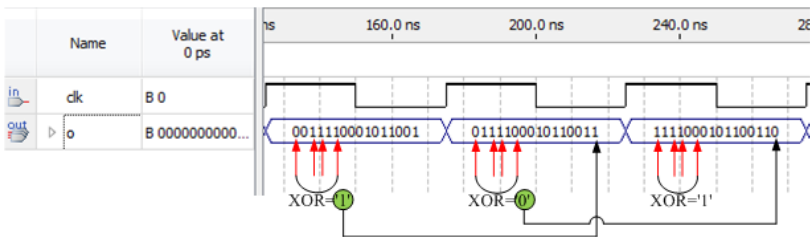


Figure 17. The simulation result for LFSR Block Scheme

In Figure 17, a simulation result for LFSR block scheme is plotted in Simulation Waveform Editor added in the Quartus 13.1 complier. The first bits are "0011110001011001" to encode by using LFSR. After XOR process of 16<sup>th</sup>, 14<sup>th</sup>, 13<sup>th</sup>, and 11<sup>th</sup> bits, the new bit sequent is obtained as "0111100010110011". When investigating in terms of theoretical framework, it is considered that simulation results have a valid bit sequent.

**1.5. The simulation results for Hamming Encoder and Decoder**

This section gives hamming encoder, hamming decoder, and error generator blocks. The error generator block is required to observe the performance of hamming decoder block. The encoder and decoder blocks are designed by using matrix form given in previous chapter. Both theoretical framework and simulation stage consists of (7,4) hamming code architecture. In Figure 18 a and b, it is given a hamming architecture.

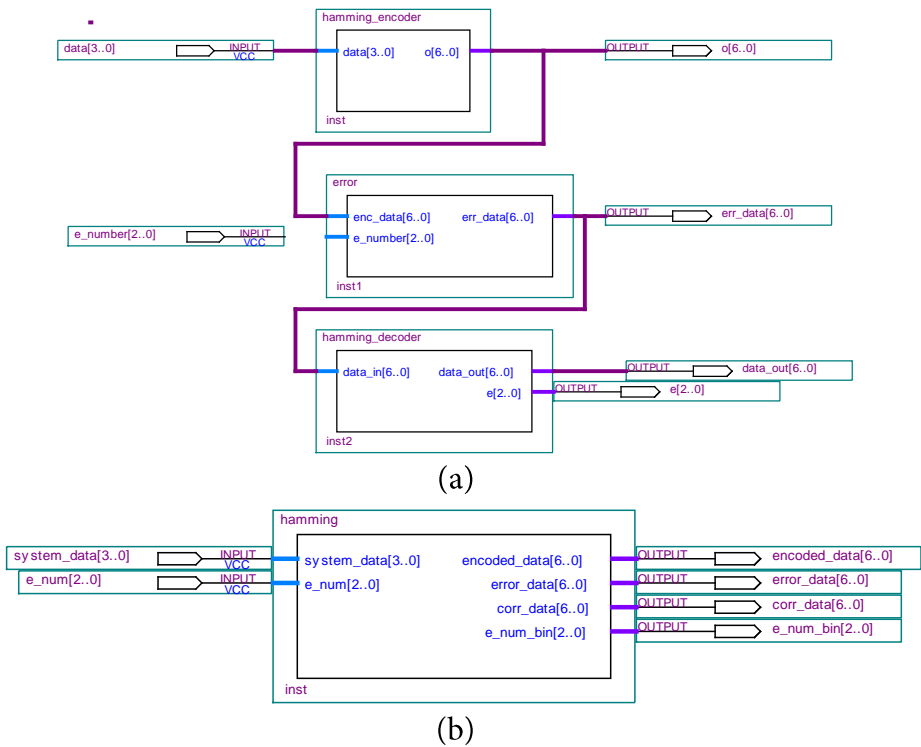
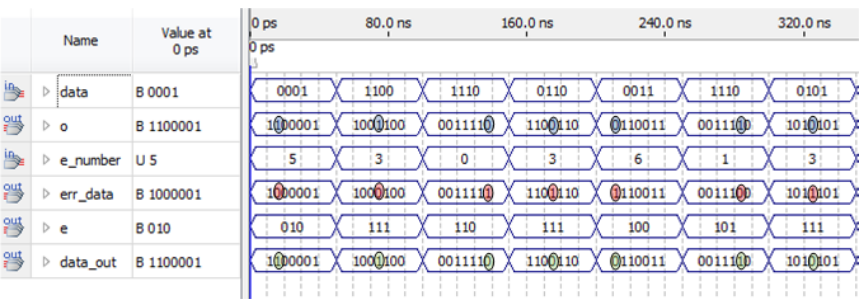
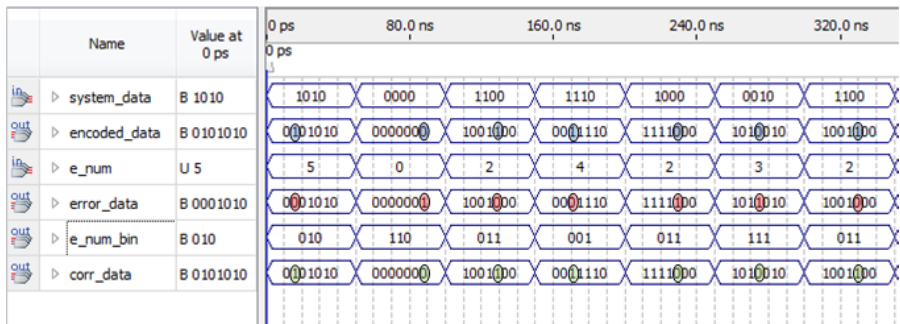


Figure 18. The block schemes for Hamming Encoder, Decoder, and Error generator.





(b)

Figure 19. The simulation results for Hamming Technique. (a)Encoder, Decoder, and Error generator. (b) the simulation result for hamming block

In Figure 19 a, it is given simulation results for Hamming encoder, decoder, and Error generator defined in Figure 18 a. According to simulation results, error generator takes inverse of one bit for output of Hamming encoder block. For Hamming code of "1100001" and e\_number of 5, e output takes "010" value. In the previous chapter, "010" is located in 5<sup>th</sup> line of matrix as expressed in Equation 11. Hence, 5<sup>th</sup> bit of err\_data is corrected by hamming method as shown in Figure 19 a. Similar results are obtained in Figure 19 b where Encoder, Decoder and Error generator are combined in a one block scheme. This project can be achieved by compiling Hamming VHDL code in Appendix A-3. However, the project must consist of all VHDL files of Appendix A-3.



## **Appendix A-1:**

### **Accumulator**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY accumulator IS
PORT (
clk: IN STD_LOGIC;
o: OUT integer range 0 to 2000;
counter: buffer integer range 0 to 50;
data,n: in integer range 0 to 255);
END accumulator;
ARCHITECTURE acc OF accumulator IS
BEGIN
PROCESS (clk,data)
```

```
VARIABLE agen : INTEGER :=0 ;
variable d: integer :=0;
BEGIN
IF (clk'EVENT AND clk='1') THEN
if counter=n then
counter<=1;
d:=data;
else
counter<=counter+1;
d:=data+d;
end if;
end if;
o<=d;
END PROCESS;
END acc;
```

## **Appendix A-2:**

### **LFSR**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY LFSR IS
PORT (
clk: IN STD_LOGIC;
o: OUT std_logic_vector (15 downto 0));
END LFSR;
ARCHITECTURE fib_nc OF LFSR IS
Signal i: STD_LOGIC_vector (15 downto 0):=
"1100111100010110";
Signal data: STD_LOGIC;
BEGIN
PROCESS (clk)
--VARIABLE agen : INTEGER :=0 ;
```

```
BEGIN
IF (clk'EVENT AND clk='1') THEN
i<=i(14 downto 0) & (i(15) xor i(13) xor i(12) xor i(10));
o<=i;
end if;
END PROCESS;
END fib_nc;
```

### **Appendix A-3**

#### **Hamming Encoder:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY hamming_encoder IS
PORT (
data: in std_logic_vector (3 downto 0);
o: OUT std_logic_vector (6 downto 0));
END hamming_encoder;
ARCHITECTURE enc OF hamming_encoder IS
Signal d_1: STD_LOGIC;
Signal d_2: STD_LOGIC;
Signal d_3: STD_LOGIC;
BEGIN
PROCESS (data)
BEGIN
d_1<=data(1) xor data(2) xor data(3);
d_2<=data(0) xor data(2) xor data(3);
d_3<=data(0) xor data(1) xor data(3);
o<=d_3&d_2&d_1&data;
END PROCESS;
END enc;
```

#### **Hamming Decoder:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY hamming_decoder IS
PORT (
data_in: in std_logic_vector (6 downto 0);
data_out: out std_logic_vector (6 downto 0);
e:buffer std_logic_vector(2 downto 0));
END hamming_decoder;
ARCHITECTURE dec OF hamming_decoder IS
--Signal e: STD_LOGIC_vector(2 downto 0);
Signal reg: STD_LOGIC_vector(6 downto 0);
BEGIN
PROCESS (data_in)
BEGIN
reg<=data_in;
e(0)<=data_in(1) xor data_in(2) xor data_in(3)xor data_in(4);
e(1)<=data_in(0) xor data_in(2) xor data_in(3)xor data_in(5);
e(2)<=data_in(0) xor data_in(1) xor data_in(3)xor data_in(6);
IF (e="110") THEN
reg(0) <= not (data_in(0));
ELSIF (e="101") THEN
reg(1) <= not (data_in(1));
ELSIF (e="011") THEN
reg(2) <= not (data_in(2));
ELSIF (e="111") THEN
reg(3) <= not (data_in(3));
ELSIF (e="001") THEN
reg(4) <= not (data_in(4));
ELSIF (e="010") THEN
reg(5) <= not (data_in(5));
ELSIF (e="100") THEN
reg(6) <= not (data_in(6));
end if;
```

```
data_out<=reg;
END PROCESS;
END dec;
```

### **Error:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY error IS
PORT (
enc_data: in std_logic_vector (6 downto 0);
e_number: in integer range 0 to 6;
err_data: OUT std_logic_vector (6 downto 0));
END error;
ARCHITECTURE n_err OF error IS
Signal err_reg: STD_LOGIC_vector (6 downto 0);
BEGIN
PROCESS (enc_data)
BEGIN
err_reg<=enc_data;
err_reg(e_number)<= not enc_data(e_number);
err_data<=err_reg;
END PROCESS;
END n_err;
```

### **Hamming**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY hamming IS
PORT (
system_data: in std_logic_vector (3 downto 0);
encoded_data: buffer std_logic_vector (6 downto 0);
e_num: in integer range 0 to 6;
error_data: buffer std_logic_vector (6 downto 0);
```

```
corr_data: out std_logic_vector (6 downto 0);
e_num_bin: buffer std_logic_vector(2 downto 0)
);
END hamming;
ARCHITECTURE enc_dec_err OF hamming IS
component hamming_encoder
  PORT
  (
data: in std_logic_vector (3 downto 0);
o: OUT std_logic_vector (6 downto 0)
  );
end component;
component error
  PORT
  (
enc_data: in std_logic_vector (6 downto 0);
e_number: in integer range 0 to 6;
err_data: OUT std_logic_vector (6 downto 0)
  );
end component;
component hamming_decoder
  PORT
  (
data_in: in std_logic_vector (6 downto 0);
data_out: out std_logic_vector (6 downto 0);
e:buffer std_logic_vector(2 downto 0)
  );
end component;
BEGIN
block_1: hamming_encoder port map (
  data=>system_data,
  o=>encoded_data
);
block_2: error port map (
```

```
enc_data=>encoded_data,  
e_number=>e_num,  
err_data=>error_data  
);  
block_3: hamming_decoder port map (  
data_in=>error_data,  
data_out=>corr_data,  
e=>e_num_bin  
);  
end enc_dec_err;
```

## References

- Babu, A. S., & Anand, B. (2020). Modified dynamic current mode logic based LFSR for low power applications. *Microprocessors and Microsystems*, 72, 102945.
- Hazwani, S., Khan, S., Siddiqi, M. U., Al-Khateeb, K. A., Habaebi, M. H., & Shahid, Z. (2014, January). Randomness analysis of pseudo random noise generator using 24-bits LFSR. In *2014 5th International Conference on Intelligent Systems, Modelling and Simulation* (pp. 772-774). IEEE.
- Hong, E., Choi, K. A., & Joo, J. (2023). Efficient Two-Stage Max-Pooling Engines for an FPGA-Based Convolutional Neural Network. *Electronics*, 12(19), 4043.
- Hussain, F., & Sarkar, S. (2024, April). Design and FPGA Implementation of Five Stage Pipelined RISC-V Processor. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)* (pp. 1-6). IEEE.
- Khedkar, A. A., & Khade, R. H. (2017). High speed FPGA-based data acquisition system. *Microprocessors and Microsystems*, 49, 87-94.
- Lin, J., Xiong, C., & Yan, Z. (2015). A high throughput list decoder architecture for polar codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(6), 2378-2391.
- Lin, J., Yan, Z., & Wang, Z. (2016). Efficient soft cancellation decoder architectures for polar codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(1), 87-99.



- Luo, Y., Fan, C., Xu, C., & Li, X. (2024). Design and FPGA implementation of a high-speed PRNG based on an nD non-degenerate chaotic system. *Chaos, Solitons & Fractals*, 183, 114951.
- Masera, G., Piccinini, G., Roch, M. R., & Zamboni, M. (1999). VLSI architectures for turbo codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(3), 369-379.
- Nagar, M. S., Mathuriya, A., Patel, S. H., & Engineer, P. J. (2024). High-Speed Energy-Efficient Fixed-Point Signed Multipliers for FPGA-Based DSP Applications. *IEEE Embedded Systems Letters*.
- Naveen, K., Manonmai, V. S. L., Nikhitha, M. S. J., Pradeep, V., & Kumar, G. K. (2023, April). VLSI Architecture of a High Speed Polar Code Decoder using Finite Length Scaling LDPC Codes. In *2023 Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)* (pp. 1-7). IEEE.
- Pedroni, V. A. (2020). *Circuit design with VHDL*. MIT press.
- Singh, S. P., Rai, R., Awasthi, S., Singh, D. K., & Lakshmanan, M. (2023). Vlsi implementation of error correction codes for molecular communication. *Wireless Personal Communications*, 130(4), 2697-2713.
- Win, T. L., & Kyaw, N. C. (2008). Speech Encryption and Decryption Using Linear Feedback Shift Register (LFSR). *International Journal of Electronics and Communication Engineering*, 2(12), 2720-2725.
- Wu, G., Wang, K., Zhang, J., & He, J. (2018). A lightweight and efficient encryption scheme based on LFSR. *International Journal of Embedded Systems*, 10(3), 225-232.
- Xu, B., Zou, S., Bai, L., Chen, K., & Zhao, J. (2024). A general discrete memristor emulator based on Taylor expansion for the reconfigurable FPGA implementation and its application. *Nonlinear Dynamics*, 112(2), 1395-1414.
- Yuan, B., & Parhi, K. K. (2013). Low-latency successive-cancellation polar decoder architectures using 2-bit decoding. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(4), 1241-1254.

Zhang, L., Ye, S., Gou, Z., Yang, X., Dai, Q., Wang, F., & Lin, Y. (2024). An Efficient Parallel CRC Computing Method for High Bandwidth Networks and FPGA Implementation. *Electronics*, 13(22), 4399.

## **4. FPGA implementations of Basic Logical Operators and Error Correction Algorithms**

### **1. Introduction to Experimental Designs**

In literature, many papers have focused on practical systems related to embedded systems, including VLSI, FPGA, and etc (Gdaim et al., 2014; Bakiri et al., 2018; Ishihara and Yasuura, 1997; Camposano and Wolf, 2012). In this section, we give experimental and real time implementations of digital circuits based on FPGA board. One of the most important issues is synchronization problems for real time FPGA implementations because of its high-speed processing capacity (Abdelhalim et al., 2011; Pences et al., 2024; Tang, 2023). In the first stage of implementation, the pin assignment can be provided to observe

input and output signals via Input-Output terminals of FPGA board. In this section of the book, specific implementations are given for DE-0 Nano FPGA board which is very useful for many applications. Its specifications can be given as illustrated in Figure 1.

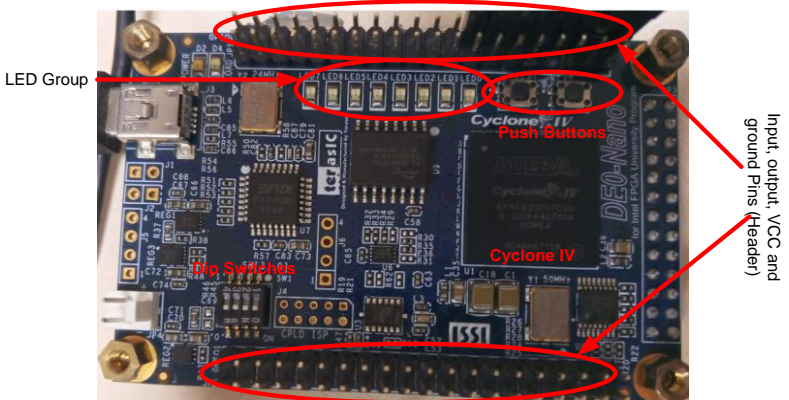


Figure 1. DE0-Nano Board

As shown in the figure, DE0-Nano FPGA board has Cyclone IV processor (EP4CE22F17C6). In addition to this, it consists of an analog-to-digital converter (ADC), 50MHz clock generator, 40 Pin Header, Push buttons, LEDs and Dip switches (Terasic, 2019). One of the most advantages is the number of input-output pins due to observe the performance and the changing of variable versus time. To implement a design created by Quartus software, the first stage is a pin assignment which is given in Figure 2 a. Figure 2 b and 2 c also indicate the pin locations. This information can be provided by the user manual of related FPGA board.

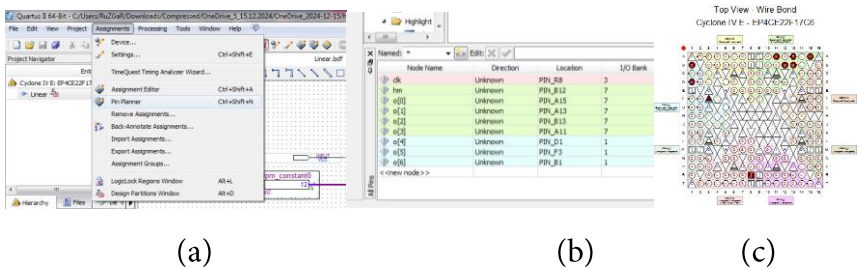


Figure 2. Pin assignment condition.

To transfer .sof file to FPGA board, the programmer must be selected in the menu of tools which is located in Figure 3. Then, the sof. file can be loaded to board by using hardware setup window as given in Figure 4. This window can give information about whether the FPGA board is connected to the server computer which manages and hosts the whole VHDL or Verilog code structure.

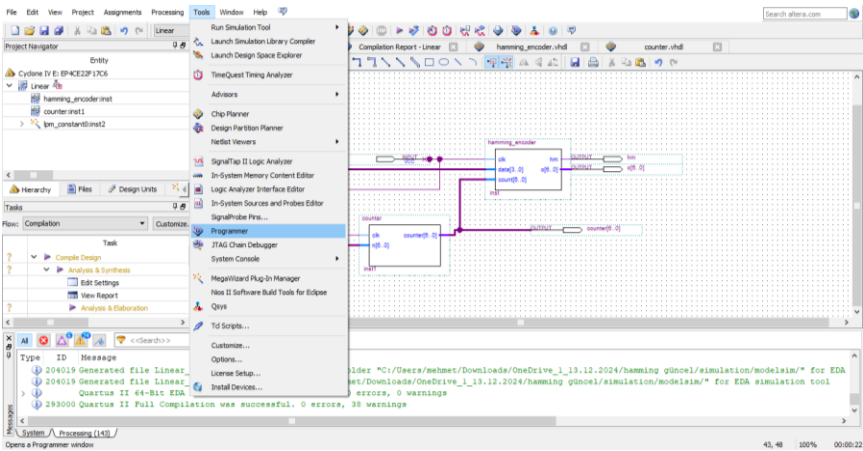


Figure 3. Loading of the program to FPGA board

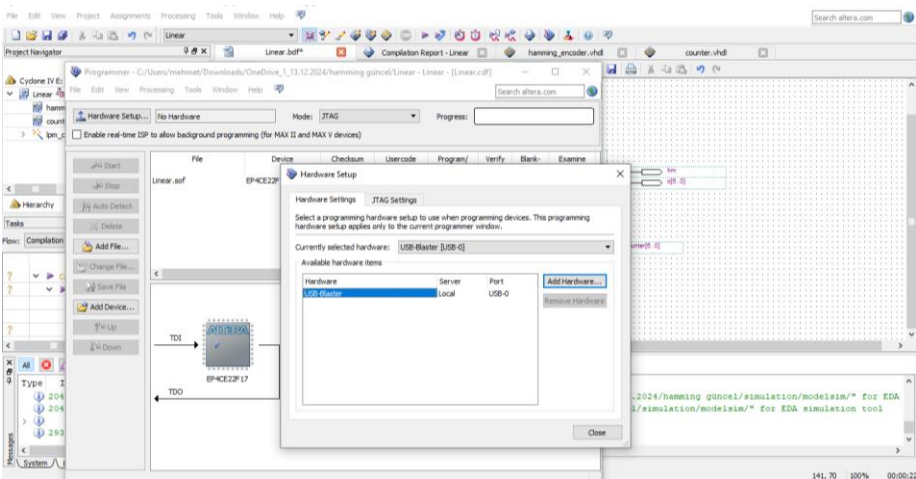


Figure 4. The Test of connection FPGA with Server Computer

As shown in Figure, USB-Blaster is seen in the Hardware Setup window if the connection is done. It can be considered that any hardware

connection cannot be provided under selection of currently selected hardware is empty.

## 2. Experimental results for Basic Gates

### 2.1 NOT Gate

There is very difference between simulation and experimental applications due to their environments. Although the simulation environment is very applicable in terms of assigned of variables, the experimental systems are very limited to observe the variation of system. Hence, some supplementation blocks or inputs in experimental systems must be made.

In this section, experimental results are given for basic gates such as, NOT, OR, XOR. To observe the results during real time application, it is made some modifications about the projects of basic gates mentioned in the first section. In Figure 5, it is given a NOT gate for experiments.

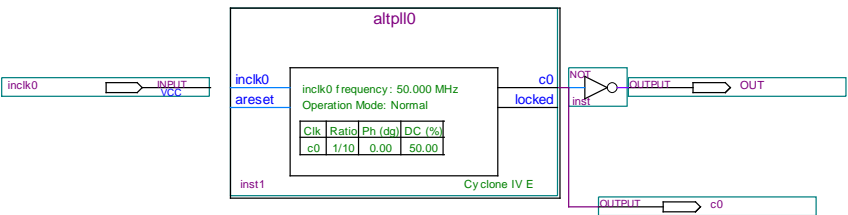


Figure 5. NOT gate for experiment

Figure 5 consists of a PLL (Phase Locked Loop) block to adjust variable input signal at input of NOT gate. To able to track the alteration, both input and output must be assigned by FPGA pin. The oscilloscope output related to NOT gate in real time FPGA board is given in Figure 6.

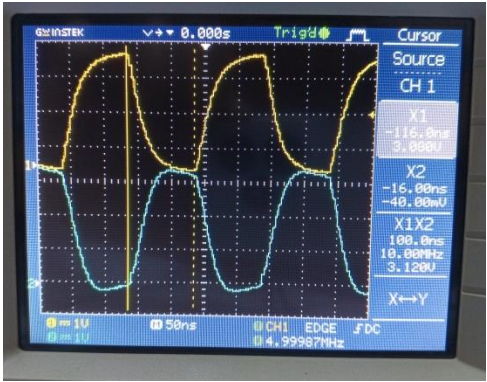


Figure 6 The experimental result for NOT gate

In Figure 6, there are two signals which are input and out signals of NOT gate given in Figure 5. As shown in Figure 5, the PLL generates clock signal by dividing input clock signal with 10. Therefore, the output of PLL must be 5 MHz. According to Figure 6, one period of clock signal is equal to 200ns.

**2.2 OR Gate**

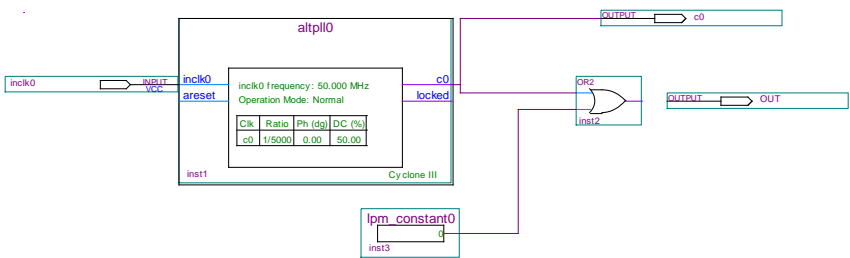


Figure 7 The experimental architecture for OR gate

As mentioned in the previous section, the output of OR gate will be a logical '1' level if any input takes logical '1'. One of inputs of OR gate is

assigned a constant value of which is equal to logical '0' since the change of output signal can be monitored versus clock signal.

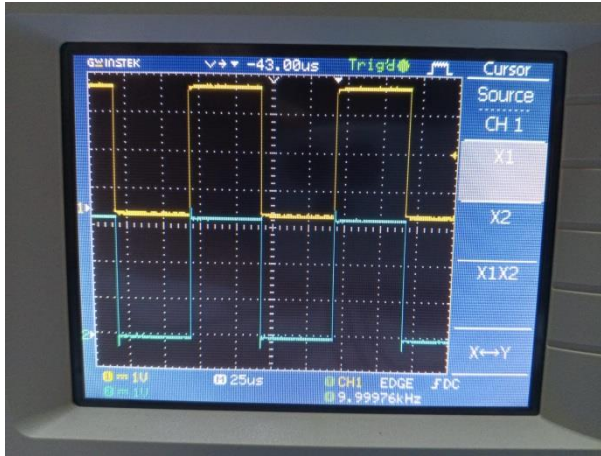


Figure 8. Experimental results for OR gate

In figure 8, it is given real time results obtained by using FPGA board. The yellow line defines the output signal of OR gate while blue line is expressed as input signal of OR gate. Additionally, the input signal is assigned as clock signal which is generated from PLL block. The constant signal is selected as logical '0' to observe the output signal versus clock signal. The yellow line is rising up logical '1' level while the blue line gets to a logical '1'. Otherwise, the yellow line is logical '0' level.

### 2.3 Experimental results for LFSR block

The theoretical framework related to LFSR is mentioned in the previous section. Additionally, it is given simulation results for LFSR timing process. However, LFSR architecture given in the previous section must be improved to achieve the experimental results. The counter block is added to LFSR system to spread in terms of time and to observe the experimental results via oscilloscope display. In this section, experimental results are introduced for LFSR block.

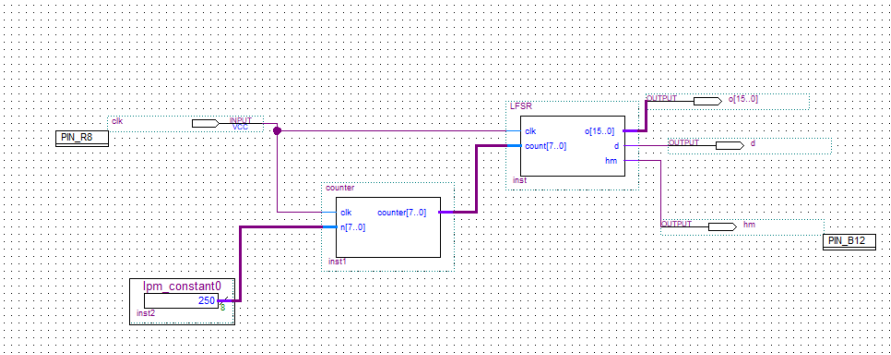


Figure 5. LFSR architecture for experimental applications.

In Figure 6, a sequential bit vector is given in terms of time. According to the figure, 1001101011100111 bits is passed through a LFSR block. Then, 1100110101110011 bits are obtained at the output of LFSR block since 15<sup>th</sup>, 13<sup>th</sup>, 12<sup>th</sup>, and 10<sup>th</sup> bits are used to apply on XOR gates, and result is added to 15<sup>th</sup> bit. The LSB value of bit line is rotated to merge the result bit to bit sequent. Moreover, a 0's group is used to split LFSR packet from each other, and to provide a synchronization between two packets. For LFSR given in this chapter, a code structure is presented in the Appendix section.

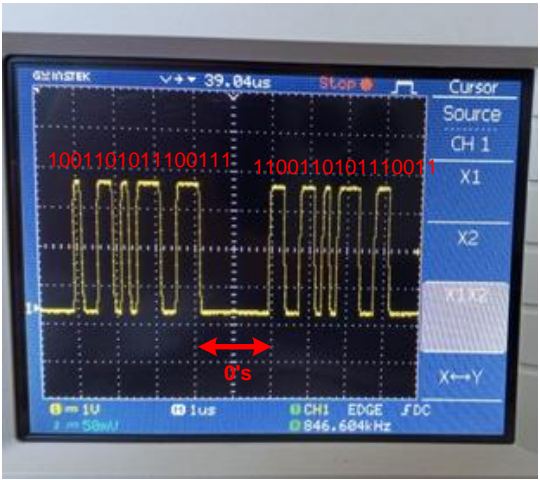
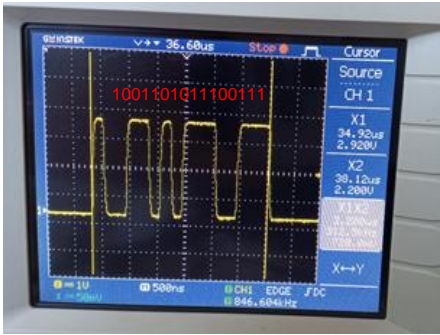
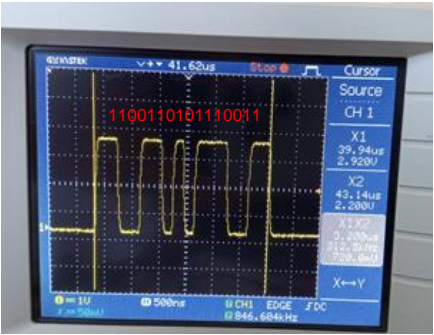


Figure 6. LFSR architecture for experimental applications.

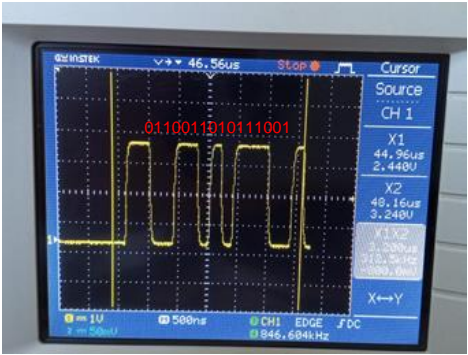




(a)



(b)



(c)

Figure 7. LFSR signal period

In Figure 7, experimental results for LFSR is presented to observe the comparison among LFSR bit packets, and exhibit the period of a LFSR packet. According to experimental results, 16 bits of LFSR are displayed at the oscilloscope screen of which time ranging is adjusted as 500ns for each five nodes. It shows the time of 3.2 us between two cursors. Hence, a bit length is equal to 200ns. As mentioned LFSR code structure, a bit length is selected as 10 clock cycle which is period of 20 ns. A new bit rotates the bit vector to the right direction.

## 2.4 Experimental results for Hamming

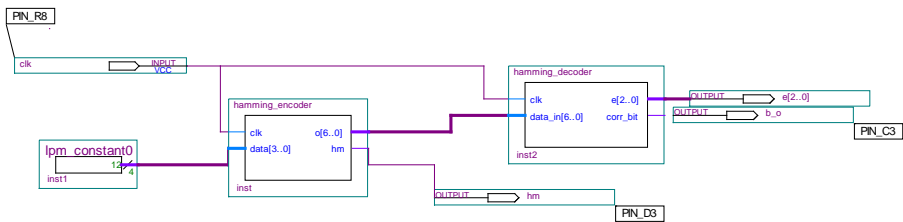


Figure 7. Block scheme for Hamming encoder and decoder

In Figure 7, a block scheme is represented to observe Hamming encoder and decoder architectures for real time implementations which consist of Pin assignments of input and output blocks. According to the scheme, the clock signal is generated via R8 pin which is defined as clk in this architecture.

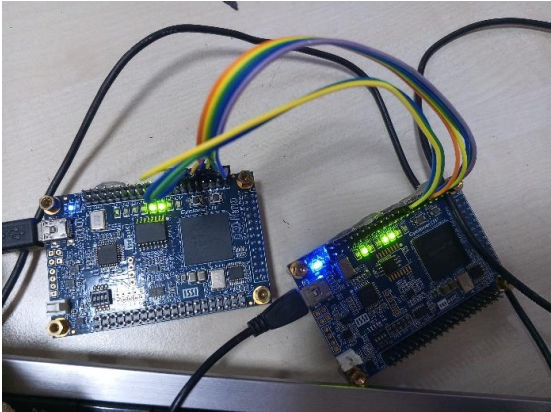


Figure 8. The receiver and transmitter for Hamming experimental system

In Hamming implementation system, the data bit is selected as "1100". Therefore, the output of hamming encoder is equal to "1001100". In Figure 8, it has been given a real time system that includes transmitter FPGA (Hamming encoder) and receiver FPPGA (Hamming Decoder). The FPGA located on the left side shows "0001100" bits at the LEDs group while the FPGA placed in right side presents "1001100". The second FPGA is corrected the 7<sup>th</sup> bit of encoder output.

In figure 9, bits are spread in time axes. The period of bit transmission is selected as 200ns as shown in the figure. The first implementation which is given in 9.a shows is similar as experimental system which consists of transmitter and receiver given in Figure 8. In the second results, the 4<sup>th</sup> bit is inverted to observe the error correction process. In Figure 9.b, transmitter system transfers "1011100" data while receiver detects "1001100".



(a)

(b)

Figure 9. The change over the time for Hamming experimental system

## Appendix A

### VHDL code of LFSR for Real Time Project

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```

```
ENTITY LFSR IS
PORT (
clk: IN STD_LOGIC;
o: OUT std_logic_vector (15 downto 0);d:out std_logic;
count: in integer range 0 to 255;hm: buffer std_logic);
END LFSR;
ARCHITECTURE serial_converter OF LFSR IS
Signal i: STD_LOGIC_vector (15 downto 0):= "1100111100010110";
Signal data: STD_LOGIC;
BEGIN
PROCESS (clk)
--VARIABLE agen : INTEGER :=0 ;
BEGIN
IF (clk'EVENT AND clk='1') THEN
if count=250 then
i<=i(14 downto 0) & (i(15) xor i(13) xor i(12) xor i(10));
o<=i;
end if;
CASE count IS
WHEN 0 => hm<='0';
WHEN 10 => hm<='0';
WHEN 20 => hm<='0';
WHEN 30 => hm<='0';
WHEN 40 => hm<=i(0);
WHEN 50 => hm<=i(1);
WHEN 60 => hm<=i(2);
WHEN 70 => hm<=i(3);
WHEN 80 => hm<=i(4);
WHEN 90 => hm<=i(5);
WHEN 100 => hm<=i(6);
WHEN 110 => hm<=i(7);
WHEN 120 => hm<=i(8);
WHEN 130 => hm<=i(9);
WHEN 140 => hm<=i(10);
```

```
WHEN 150 => hm<=i(11);
WHEN 160 => hm<=i(12);
WHEN 170 => hm<=i(13);
WHEN 180 => hm<=i(14);
WHEN 190 => hm<=i(15);
WHEN 200 => hm<='0';
WHEN 210 => hm<='0';
WHEN 220 => hm<='0';
WHEN 230 => hm<='0';
WHEN 240 => hm<='0';
WHEN OTHERS => hm<=hm;
END CASE;
end if;
END PROCESS;
END serial_converter;
```

## **VHDL code of Counter for Real Time Project**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY counter IS
PORT (
clk: IN STD_LOGIC; n: in integer range 0 to 250;
counter: buffer integer range 0 to 250);
END counter;
ARCHITECTURE behv OF counter IS
BEGIN
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
if counter=n then
counter<=0;
else
counter<=counter+1;
end if;

```

```
end if;  
END PROCESS;  
END behv;
```

## **Appendix B**

### **VHDL code of Hamming Encoder for Real Time Project**

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
ENTITY hamming_encoder IS  
PORT (clk: in std_logic;  
data: in std_logic_vector (3 downto 0);  
o: buffer std_logic_vector (6 downto 0);  
hm: buffer std_logic);  
END hamming_encoder;  
ARCHITECTURE enc OF hamming_encoder IS  
Signal d_1: STD_LOGIC;  
Signal d_2: STD_LOGIC;  
Signal d_3: STD_LOGIC;  
BEGIN  
PROCESS (data)  
variable c: integer:=0;  
BEGIN  
IF (clk'EVENT AND clk='1') THEN  
if c=0 then  
d_1<=data(1) xor data(2) xor data(3);  
d_2<=data(0) xor data(2) xor data(3);  
d_3<=data(0) xor data(1) xor data(3);  
o<= (d_3)&d_2&(not d_1)&data;  
hm<=o(6);  
elsif c=10 then  
hm<= o(5);  
elsif c=20 then
```

```
hm<=o(4);
elsif c=30 then
hm<=o(3);
elsif c=40 then
hm<=o(2);
  elsif c=50 then
hm<=o(1);
  elsif c=60 then
hm<=o(0);
  elsif c=69 then
c:=-1;
end if;
c:=c+1;
end if;
END PROCESS;
END enc;
```

## **VHDL code of Hamming decoder for Real Time Project**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY hamming_decoder IS
PORT (clk: in std_logic;
data_in: in std_logic_vector (6 downto 0);
e:buffer std_logic_vector(2 downto 0);
corr_bit: out std_logic);
END hamming_decoder;
ARCHITECTURE dec OF hamming_decoder IS
--Signal e: STD_LOGIC_vector(2 downto 0);
Signal reg: STD_LOGIC_vector(6 downto 0);

BEGIN
PROCESS (clk)
```

```
variable c: integer:=0;
BEGIN
IF (clk'EVENT AND clk='1') THEN
if c=0 then
    reg<=data_in;
e(0)<=data_in(1) xor data_in(2) xor data_in(3)xor data_in(4);
e(1)<=data_in(0) xor data_in(2) xor data_in(3)xor data_in(5);
e(2)<=data_in(0) xor data_in(1) xor data_in(3)xor data_in(6);

IF (e="110") THEN
reg(0) <= not (data_in(0));
ELSIF (e="101") THEN
reg(1) <= not (data_in(1));
ELSIF (e="011") THEN
reg(2) <= not (data_in(2));
ELSIF (e="111") THEN
reg(3) <= not (data_in(3));
ELSIF (e="001") THEN
reg(4) <= not (data_in(4));
ELSIF (e="010") THEN
reg(5) <= not (data_in(5));
ELSIF (e="100") THEN
reg(6) <= not (data_in(6));
end if;
corr_bit<=reg(6);
elseif c=10 then
    corr_bit<= reg(5);
elseif c=20 then
    corr_bit<=reg(4);
elseif c=30 then
    corr_bit<=reg(3);
```



```
elsif c=40 then
corr_bit<=reg(2);
elsif c=50 then
corr_bit<=reg(1);
elsif c=60 then
corr_bit<=reg(0);
elsif c=69 then
c:=-1;
end if;
c:=c+1;
end if;
END PROCESS;
END dec;
```

## References

- Abdelhalim, K., Smolyakov, V., & Genov, R. (2011). Phase-synchronization early epileptic seizure detector VLSI architecture. *IEEE transactions on biomedical circuits and systems*, 5(5), 430-438.
- Bakiri, M., Guyeux, C., Couchot, J. F., & Oudjida, A. K. (2018). Survey on hardware implementation of random number generators on FPGA: Theory and experimental analyses. *Computer Science Review*, 27, 135-153.
- Camposano, R., & Wolf, W. (Eds.). (2012). *High-level VLSI synthesis* (Vol. 136). Springer Science & Business Media.
- Gdaim, S., Mtibaa, A., & Mimouni, M. F. (2014). Design and experimental implementation of DTC of an induction machine based on fuzzy logic control on FPGA. *IEEE transactions on fuzzy systems*, 23(3), 644-655.
- Ishihara, T., & Yasuura, H. (1997). Experimental analysis of power estimation models of CMOS VLSI circuits. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(3), 480-486.

- Pensec, W., Alari, F. R., Lapotre, V., & GOGNIAT, G. (2024, July). Defending the Citadel: Fault Injection Attacks against Dynamic Information Flow Tracking and Related Countermeasures. In *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (pp. 180-185). IEEE.
- Tang, Z. (2023, February). OFDM communication system based on FPGA. In *2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)* (pp. 666-670). IEEE.
- Terasic. (2019). DE0-Nano-SoC User Manual.