Minnesota State University, Mankato

Electrical and Computer Engineering Department

Lab Assignment 2 – Parity Bit Calculation and Verification

Date of experiment: 01/28/2025

Date of Submission: 02/04/2025

Student name

Nathnael Minuta

TA name

Masamune Oso

EE 363-01 Communication Systems Laboratory

Dr. Yikai Li

Spring 2025

Contents

                                                                  Pages

Abstract ……………………………………………………………………………. 3

Purpose…………………………………………..………………………………… 3

Materials…………………………………………..………………………………  3

Procedure, Data Collection, and Analysis……….…………………   3

Conclusion…………………………………………..……………………………   6

Appendices…………………………………………..……………………………  6

References…………………………………………..……………………………   8

Parity Bit Calculation and Verification

*Abstract*

In this lab, we explored the fundamentals of parity bit calculations and their applications in digital communication systems using Python. Specifically, we implemented functions to calculate, append, and verify parity bits for binary strings based on even and odd parity schemes. The lab introduced the concept of parity as an error-detection mechanism and demonstrated how Python can be utilized to automate this process. By defining functions to handle various parity operations and testing them with example inputs, we gained a deeper understanding of how parity bits ensure data integrity. This exercise highlighted the practical application of Python in designing and verifying basic digital communication protocols.

*Materials*

**Equipment Needed                                        Quantity.**

Laptop (Visual Studio Code)                                 1

**Methods (Procedure, Data Collection, and Analysis)**

**Part A (Task 1 (Calculate Parity Bit))**

1. **Objective**:

Create a Python function that calculates the parity bit for a given binary string and a specified parity type ("even" or "odd").

1. **Implementation**:

a. Define a function named calculate\_parity that accepts two parameters:

* data (binary string)
* parity\_type ("even" or "odd")

b. Return '1' for "even" and '0' for "odd", ignoring data for now.  
c. Maintain the function signature for future activities where data will be used dynamically.

**3. Steps:**

a. Define the function calculate\_parity in Python.  
b. Implement conditional statements to return '1' for "even" and '0' for "odd".  
c. Call the function with test inputs and print the results.

**Part B: Task 2 (Append the Parity Bit)**

**1. Objective:**

Create a Python function that appends a parity bit ('1' or '0') to a binary string based on the specified parity type ("even" or "odd").

**2. Implementation:**

a. Define a function named append\_parity that accepts two parameters:

* data (binary string).
* parity\_type (string: "even" or "odd").  
  b. Use conditional statements to determine the parity bit to append:
* Append '1' for "even".
* Append '0' for "odd".  
  c. Concatenate the parity bit to the input binary string.  
  d. Return the modified binary string with the appended parity bit.

**3. Steps:**

a. Define the function append\_parity in Python.  
b. Use conditional statements to check the parity\_type value.  
c. Concatenate the appropriate parity bit to the binary string.  
d. Call the function with test inputs and display the results using print.

**Part C: Task 3 (Verify Parity)**

**1. Objective:**

Create a Python function to verify if the parity bit in a binary string matches the specified parity type ("even" or "odd").

**2. Implementation:**

a. Define a function named verify\_parity that accepts two parameters:

* data (binary string with a parity bit).
* parity\_type (string: "even" or "odd").  
  b. Extract the parity bit from the binary string (last character).  
  c. Determine the expected parity bit based on the parity\_type:
* "even": Expect 1.
* "odd": Expect 0.  
  d. Compare the extracted parity bit with the expected parity bit.  
  e. Return True if the parity bit matches the expectation; otherwise, return False.

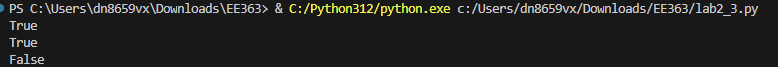
**3. Steps:**

a. Define the function verify\_parity in Python.  
b. Extract the parity bit using string indexing (data[-1]).  
c. Use conditional logic to determine the expected parity bit.  
d. Compare the actual and expected parity bits.  
e. Call the function with test cases and display the results using print.

*Results*







*Conclusion*

In this lab, we successfully implemented and analyzed the principles of parity bit operations using Python. Specifically, we developed functions to calculate, append, and verify parity bits based on even and odd parity schemes. These functions demonstrated how parity bits can be used for error detection in digital communication systems. Challenges included ensuring logical consistency in the verification process and correctly handling various parity types. By overcoming these challenges, we deepened our understanding of how Python can be applied to implement foundational concepts in error detection and digital system design. This exercise reinforced both programming skills and theoretical knowledge of parity in data communication.

*Appendices*

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

*References*

1. <https://rodzah.files.wordpress.com/2011/07/how-to-write-lab-report.pdf>
2. EE343- LAB Assignment 2.pdf