Ambrosio, Lorenzo Aivin
De Leon, Francis Zaccharie
Gabini, Brian
Verano, Carl Matthew
Villamiel, Denis Leeroi

CSARCH2 - Simulation Project
Analysis Write-up for IEEE-754 Binary-128 Floating-Point Converter

I. Introduction

This application is a Binary-128 floating-point converter designed to handle conversions between decimal and binary representations of numbers following the IEEE-754 standard. It supports special cases like Not a Number (NaN) and Infinities.

The application is a standalone GUI application written in Python that utilizes the `customtkinter` library for the graphical user interface.

The functionalities provided include:

- Selecting the input format (decimal or binary)
- Entering the binary mantissa and base-2 exponent (for binary input)
- Entering the decimal number and base-10 exponent (for decimal input)
- Performing the conversion and displaying the results:
    - Binary output with spaces separating sign bit, exponent, and mantissa
    - The hexadecimal equivalent of the binary representation
- Option to save the results to a text file

II. Background

The IEEE-754 standard defines a format for floating-point numbers used by most modern computers. The binary-128 format specifies a 128-bit representation for a floating-point number, consisting of three main components:

- Sign bit (1 bit): Indicates the sign of the number (0 for positive, 1 for negative)
- Exponent bits (15 bits): Represents the biased exponent value used to scale the significand.
- Significand/Mantissa bits (112 bits): Represents the fractional part of the number.

III. System Design

The application follows a user-friendly design with clear input sections based on the chosen input format (binary or decimal). It utilizes the following functionalities:

- User input handling: Takes the binary mantissa, base-2 exponent (for binary input), decimal number, and base-10 exponent (for decimal input) through text entry fields.
- **Conversion logic:**
    - Convert decimal to binary (If input is decimal)

    - Determine the sign bit based on the input number.
    - Normalize the binary mantissa.
    - Compute the E' for the exponent part, and the significand.
- Special case handling: Identifies and handles NaN and Infinity values based on specific exponent and mantissa bit patterns.

IV. Implementation Details

The application is written in Python and utilizes the `customtkinter` library for creating the Graphical User Interface (GUI). CustomTkinter is a UI library for Python that is based on Tkinter, acting as an extension that provides extra UI elements and more modern design cues to the overall interface. The conversion logic is implemented within the `Binary128Converter` class.

The class attributes (sign_bit, exponent_bits, mantissa_bits) store the individual components of the binary representation. Methods like `convert_decimal_to_binary128` and `convert_binary_mantissa_to_binary_128` handle the conversion processes based on the input format. Additionally, the `normalize_binary_floating_point` method adjusts the exponent and mantissa for proper representation. Other methods such as `is_valid_binary`, `is_valid_decimal`, and `is_valid_exponent` perform input validation. For the special cases, additional if statements were utilized in the `convert_binary_mantissa_to_binary_128` method. Finally, error messages are displayed if invalid input is detected by the program.

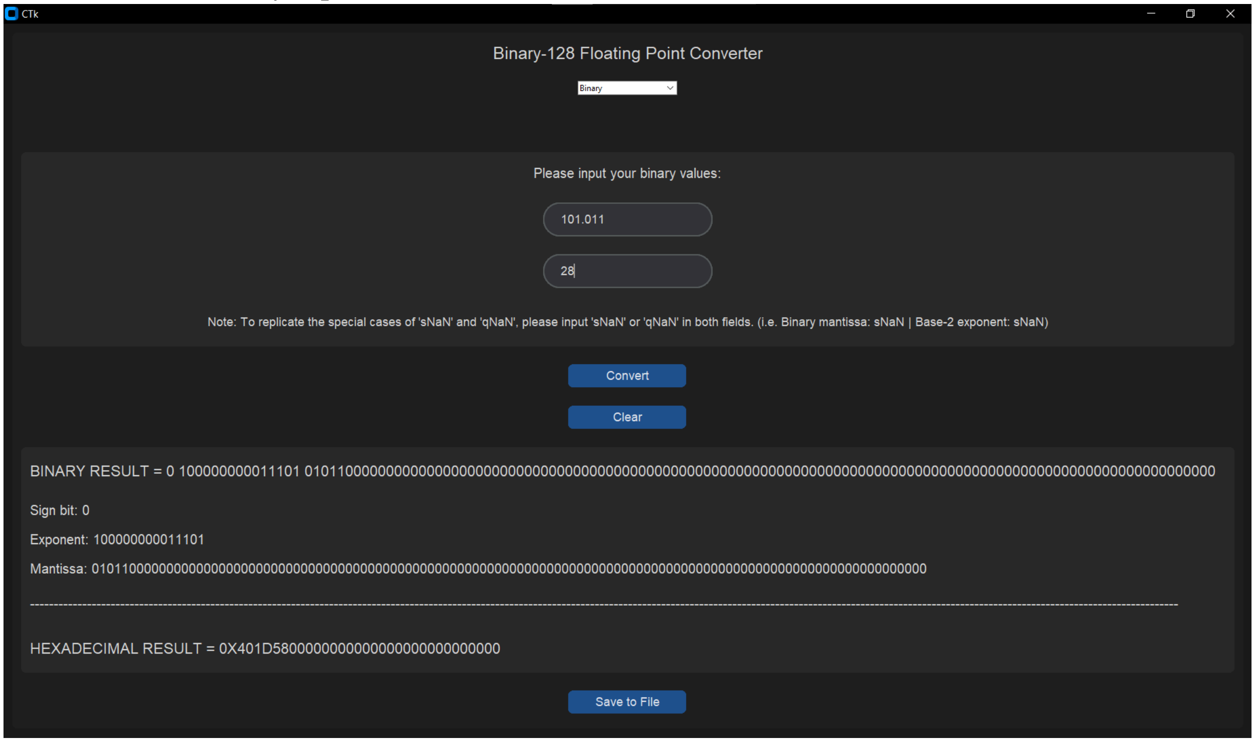
V. Testing and Validation

To ensure the application's correctness, unit tests were conducted under `main_test.py` using a dedicated testing framework, `unittest`. The following tests were conducted:

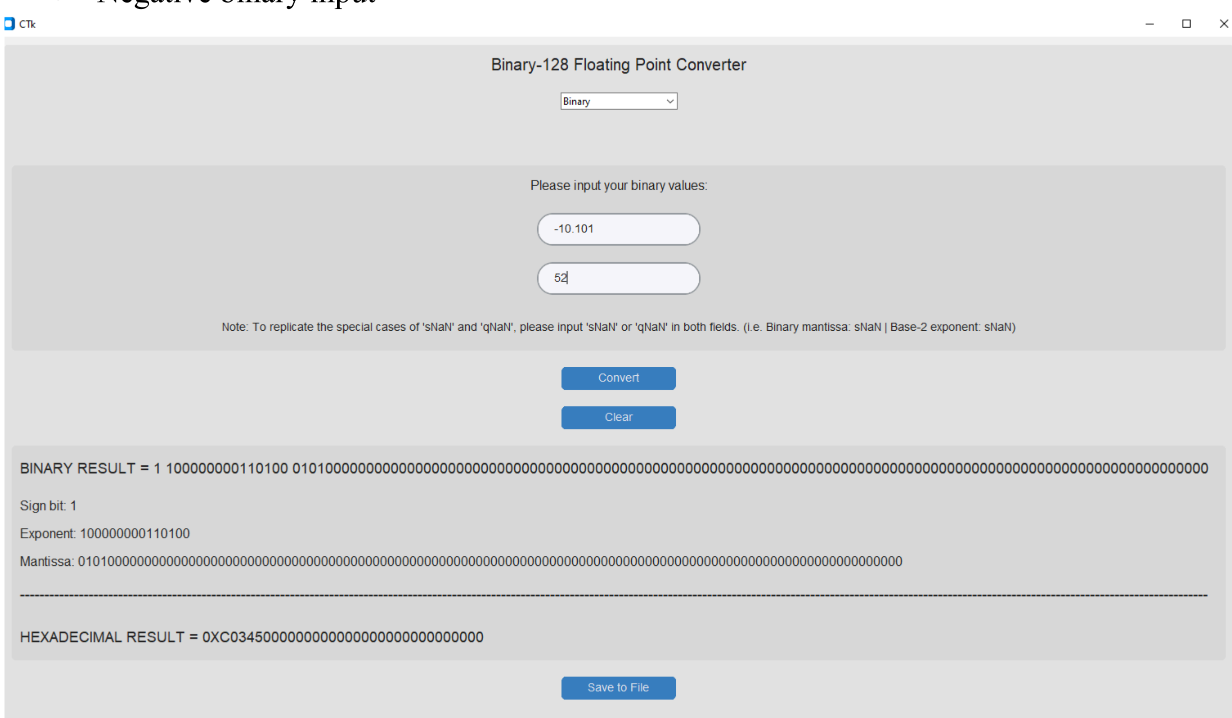
- Converting valid binary mantissa and base-2 exponents to its binary-128 floating point representation.
- Converting valid decimal numbers and base-10 exponents to its binary representation.
- Handling special cases like NaN and Infinities according to the IEEE-754 standard.
- Verifying that invalid input triggers appropriate error messages.

VI. Screenshots

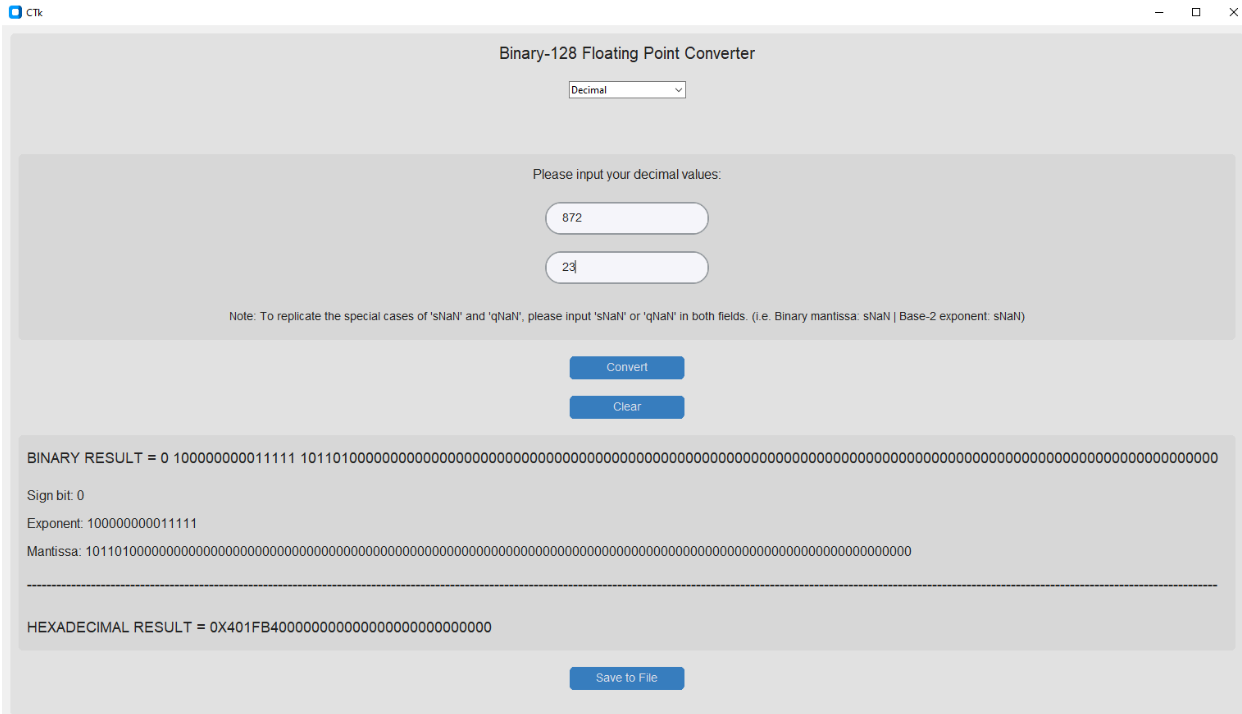
- Normal Input
  - Positive binary input



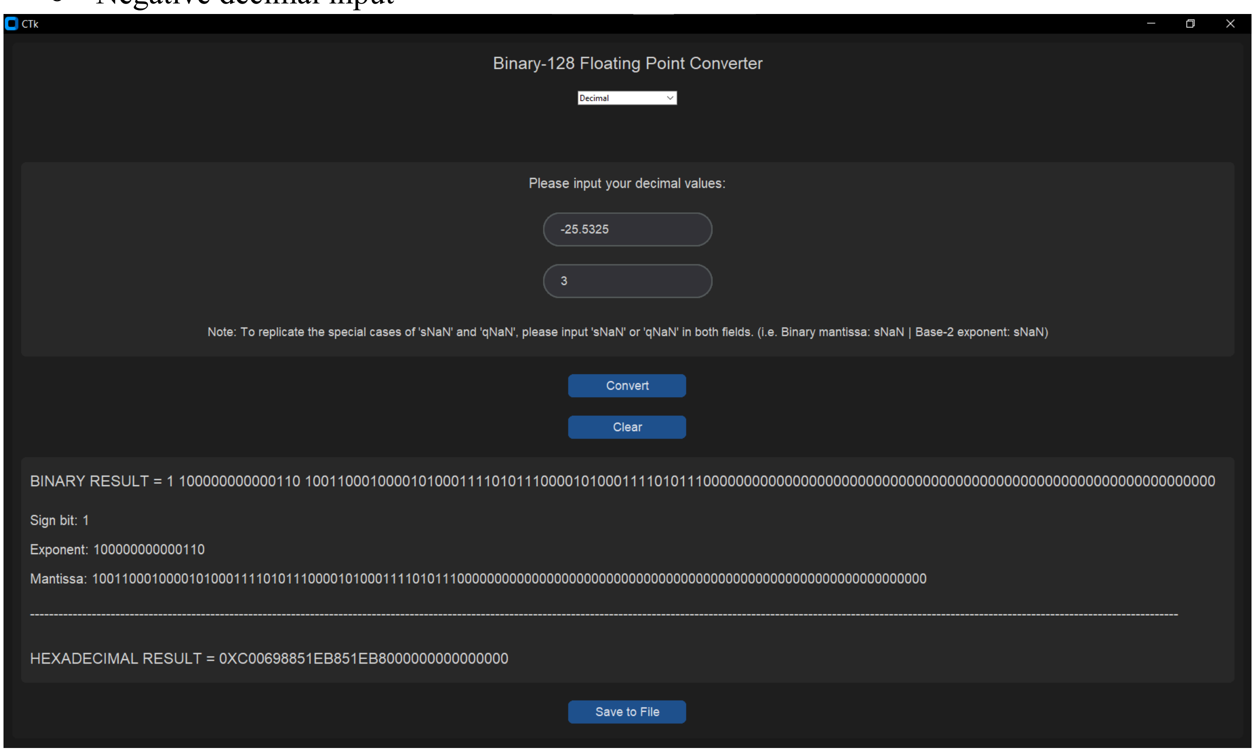
  - Negative binary input



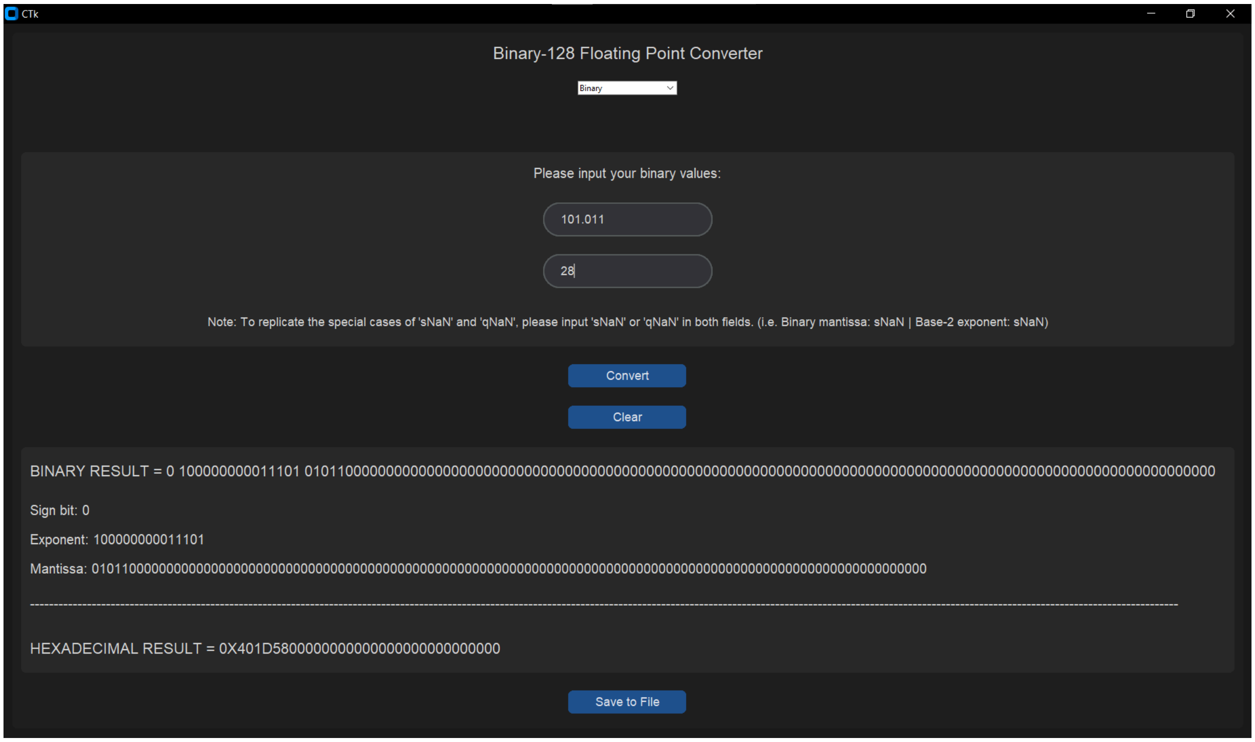
- Positive decimal input



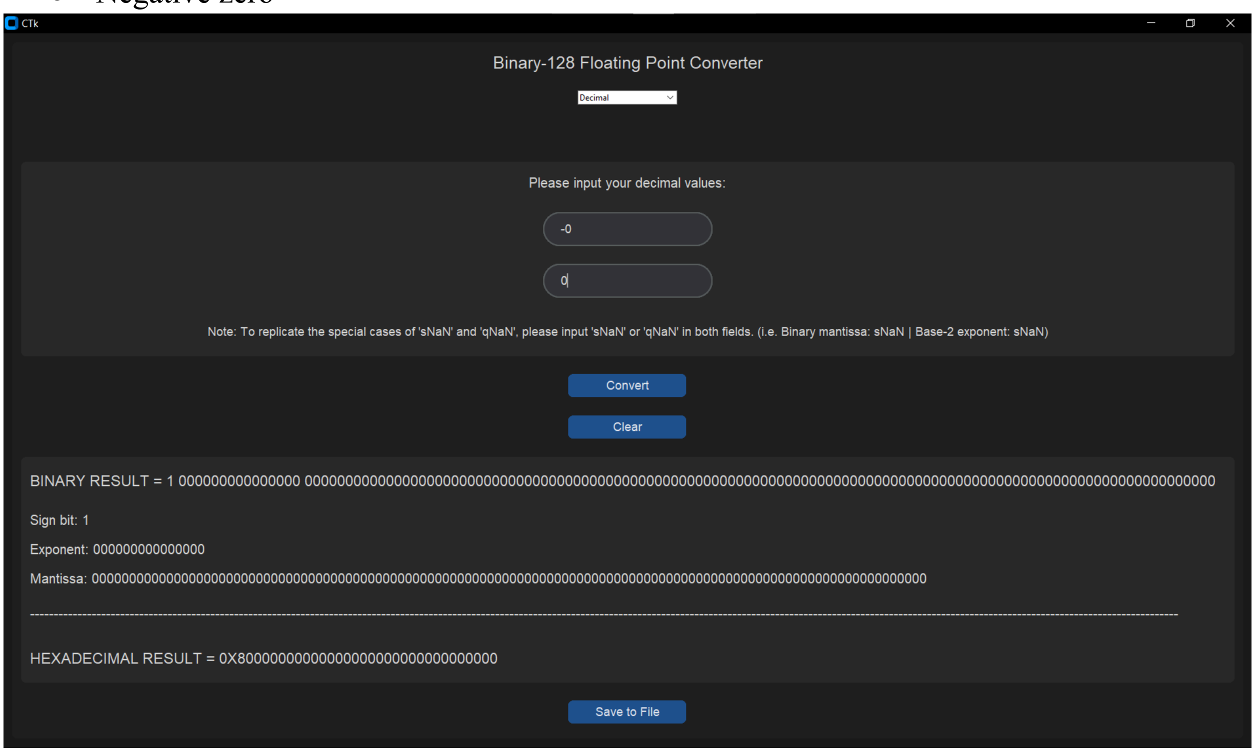
- Negative decimal input



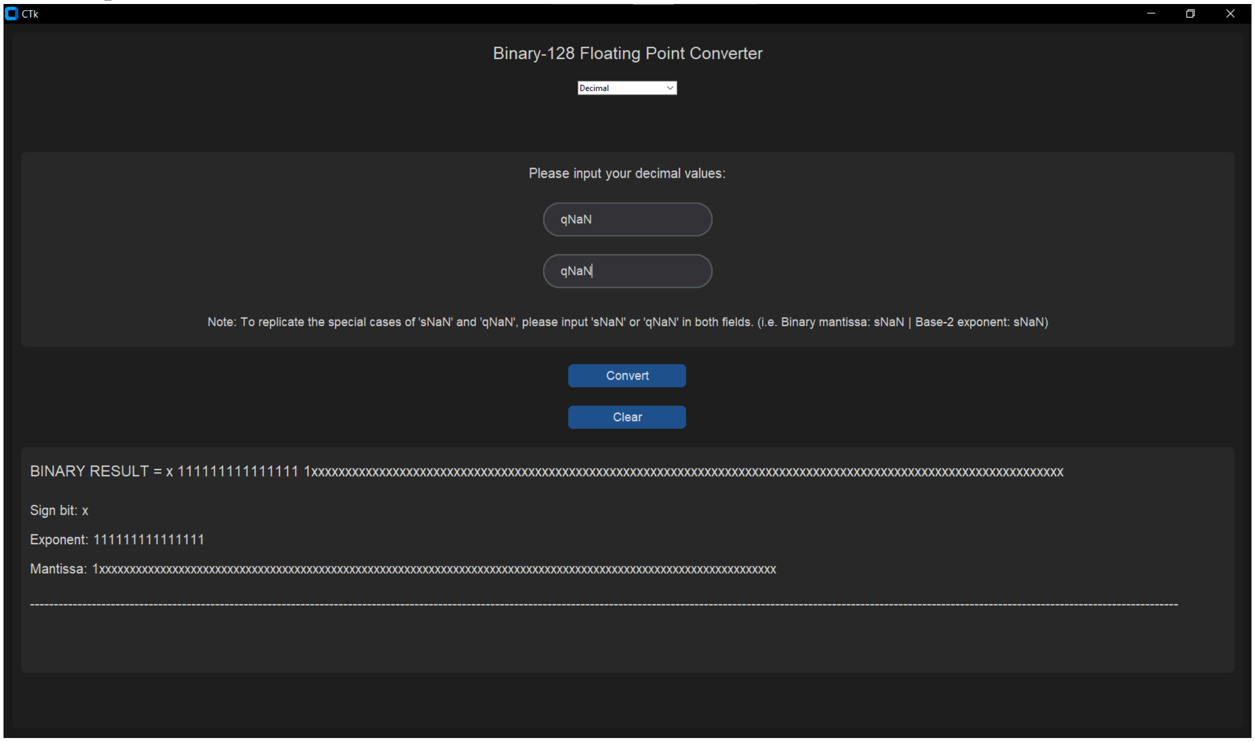
- Special cases
  - Positive zero



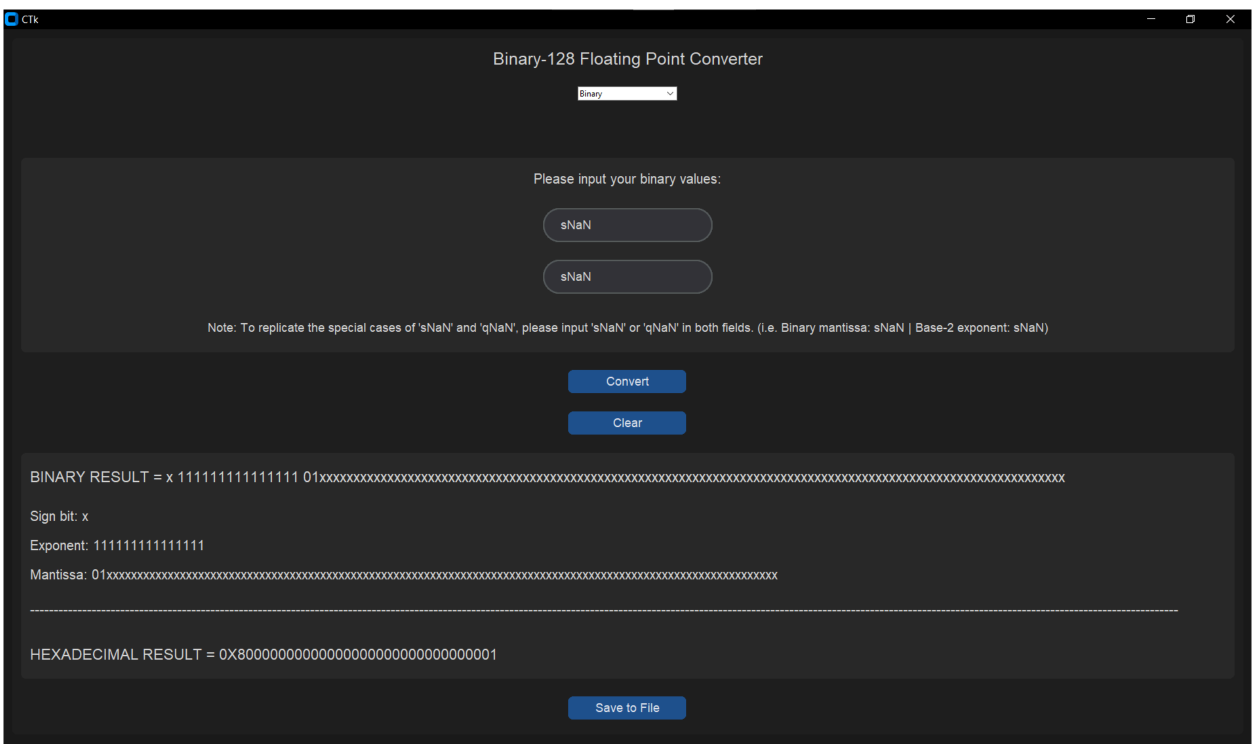
  - Negative zero



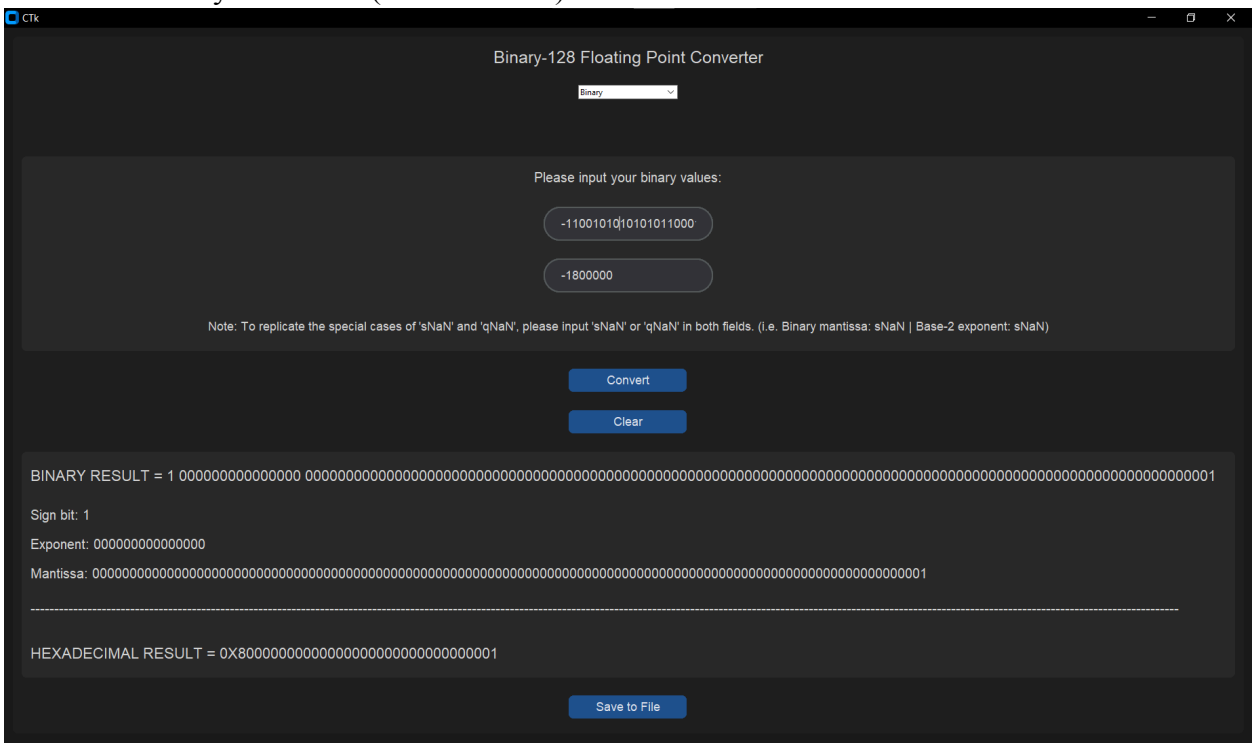
- qNaN


CTk
Binary-128 Floating Point Converter
Decimal
Please input your decimal values:
qNaN
qNaN
Note: To replicate the special cases of 'sNaN' and 'qNaN', please input 'sNaN' or 'qNaN' in both fields. (i.e. Binary mantissa: sNaN | Base-2 exponent: sNaN)
Convert
Clear
BINARY RESULT = x 111111111111111 1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Sign bit: x
Exponent: 111111111111111
Mantissa: 1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

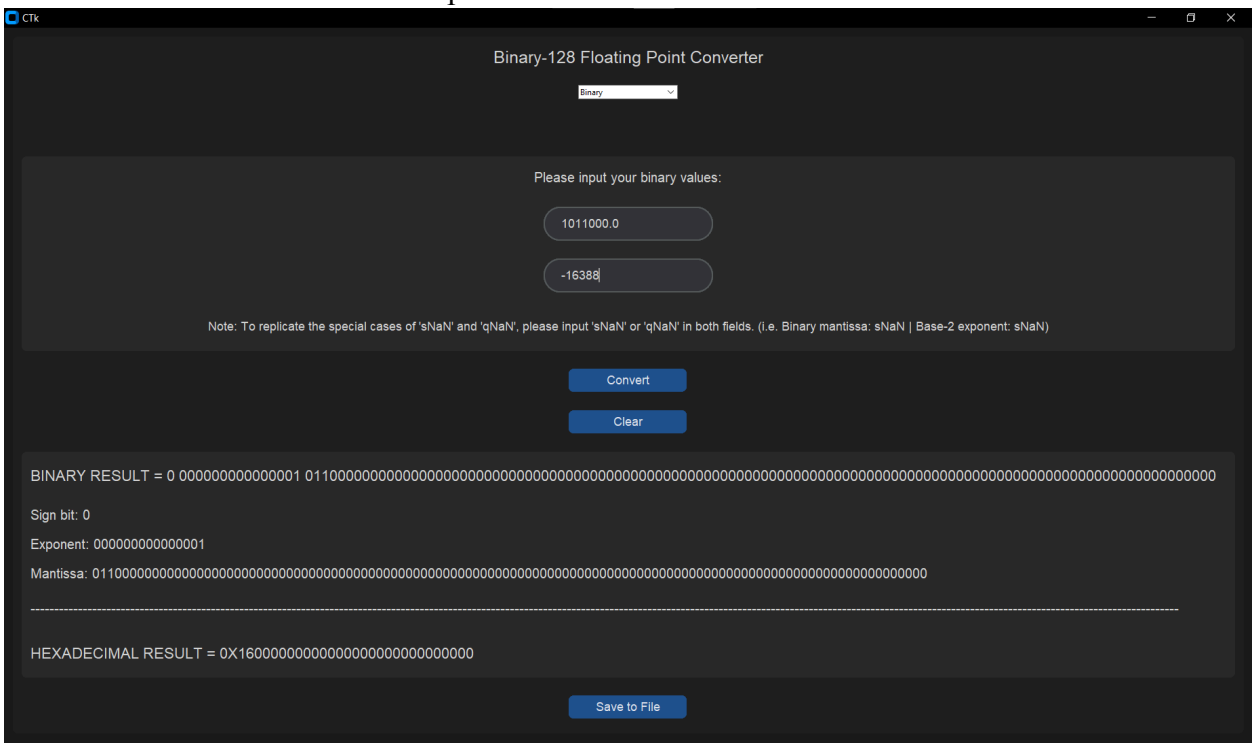
- sNaN


CTk
Binary-128 Floating Point Converter
Binary
Please input your binary values:
sNaN
sNaN
Note: To replicate the special cases of 'sNaN' and 'qNaN', please input 'sNaN' or 'qNaN' in both fields. (i.e. Binary mantissa: sNaN | Base-2 exponent: sNaN)
Convert
Clear
BINARY RESULT = x 111111111111111 01xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Sign bit: x
Exponent: 111111111111111
Mantissa: 01xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
HEXADECIMAL RESULT = 0X80000000000000000000000000000001
Save to File

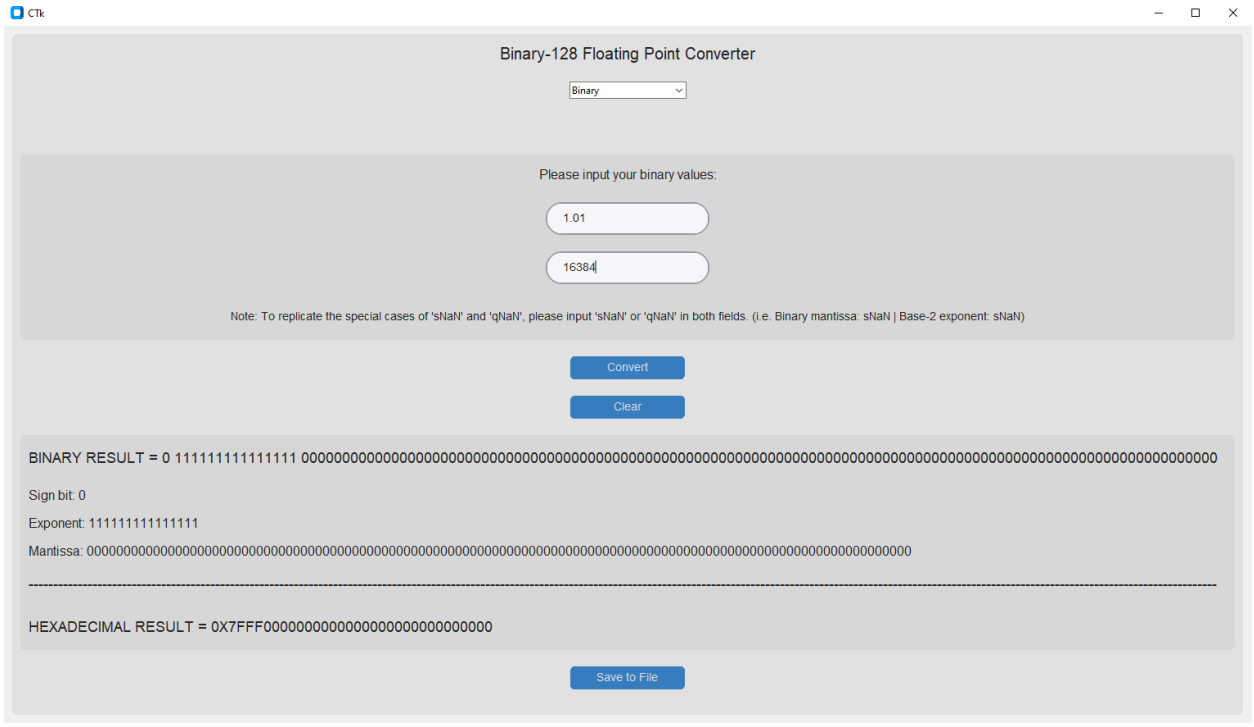
- Extremely low value (denormalized)



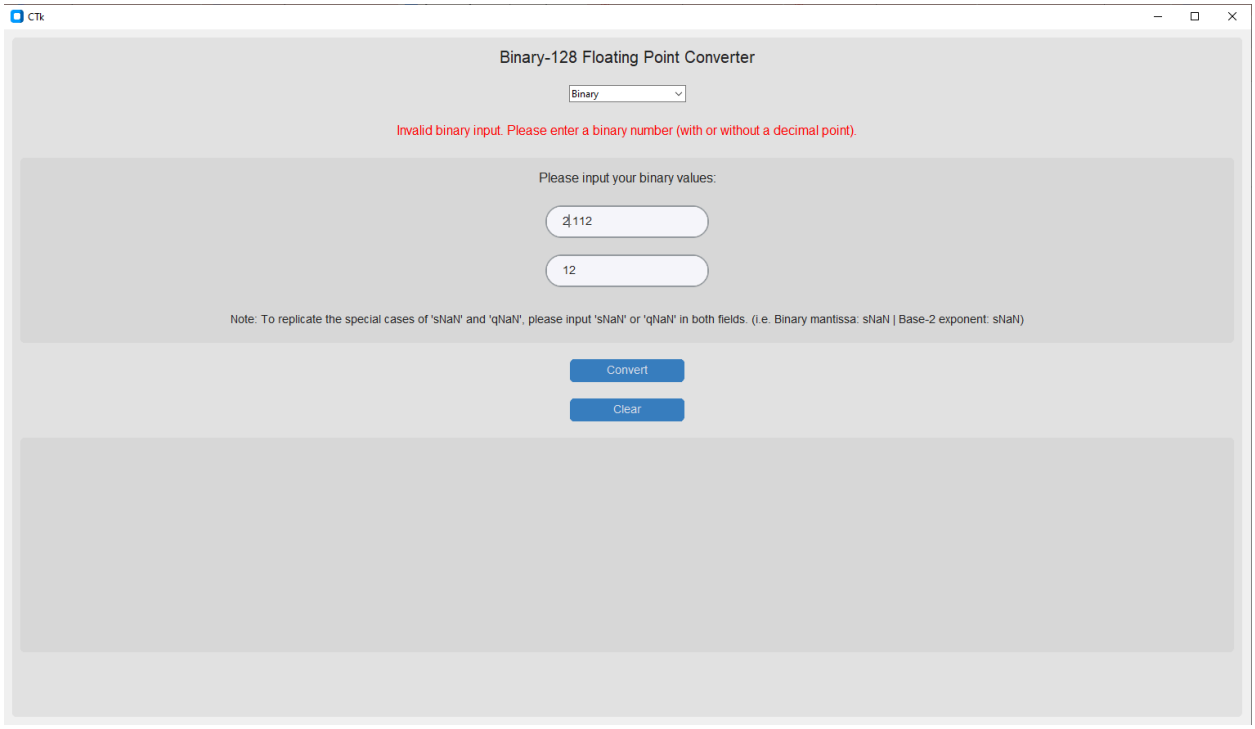
- Borderline denormalized input



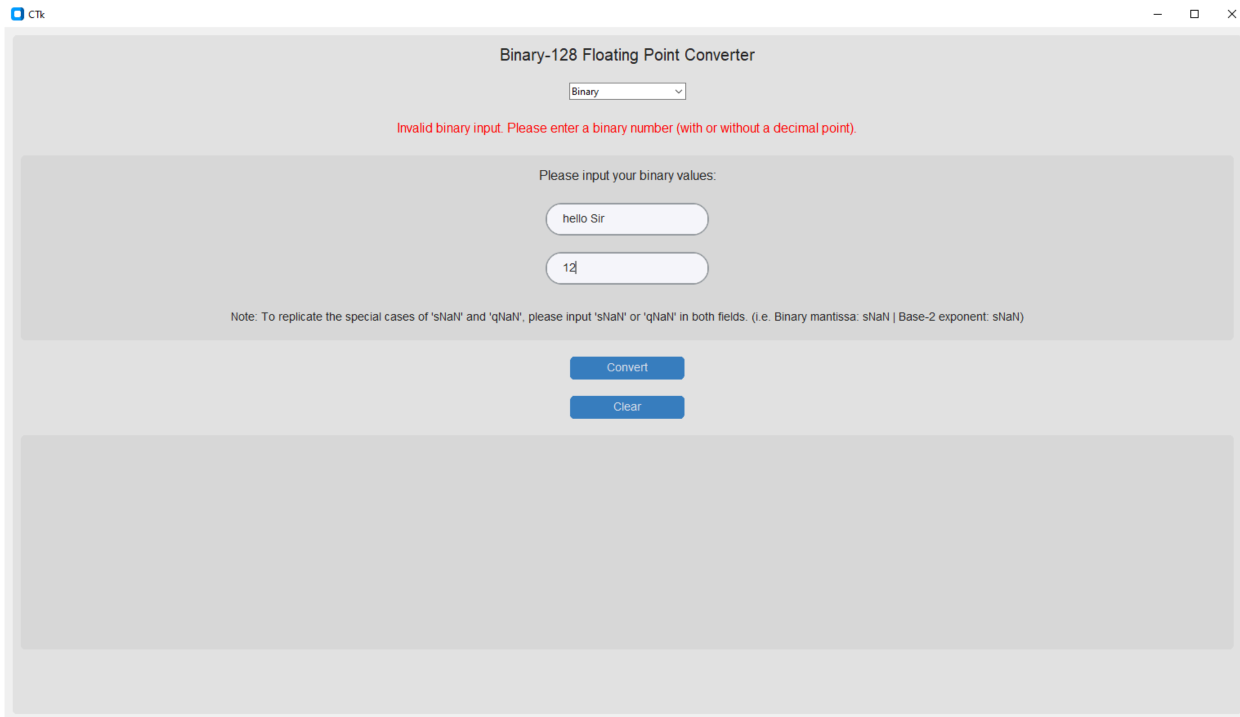
- Infinity input



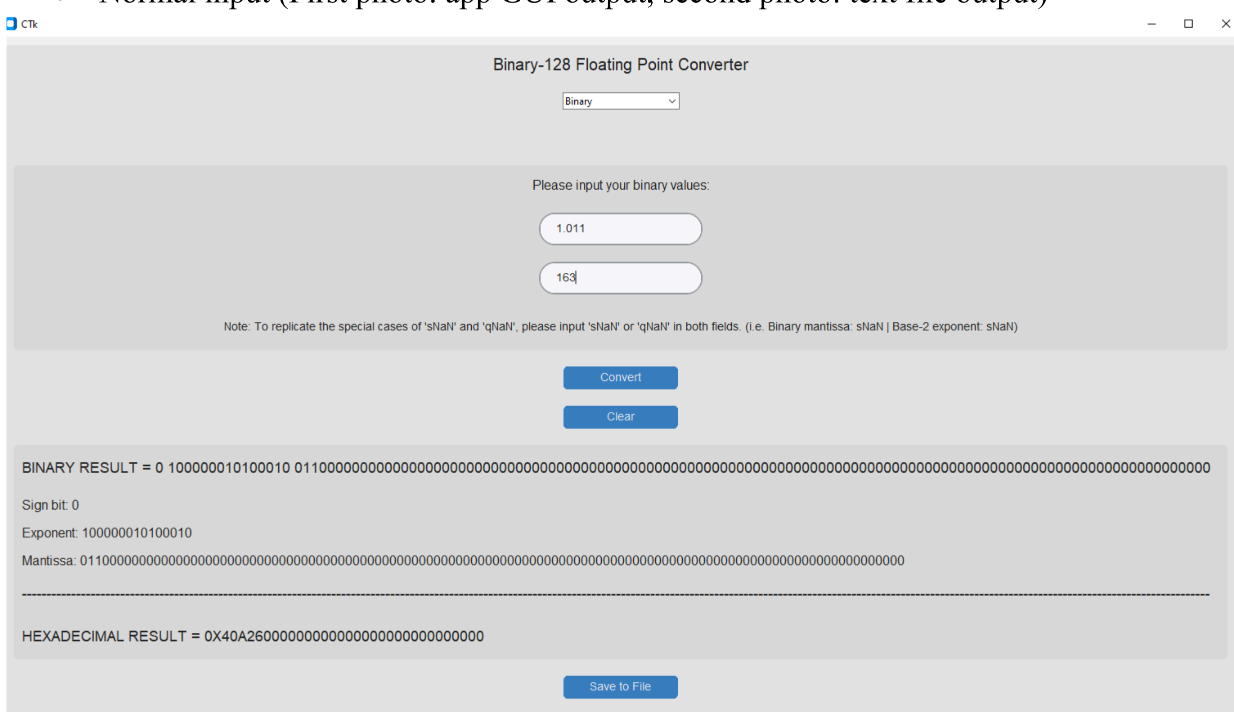
- Forced wrong inputs
  - Input error due to non-binary numerical input in Mantissa



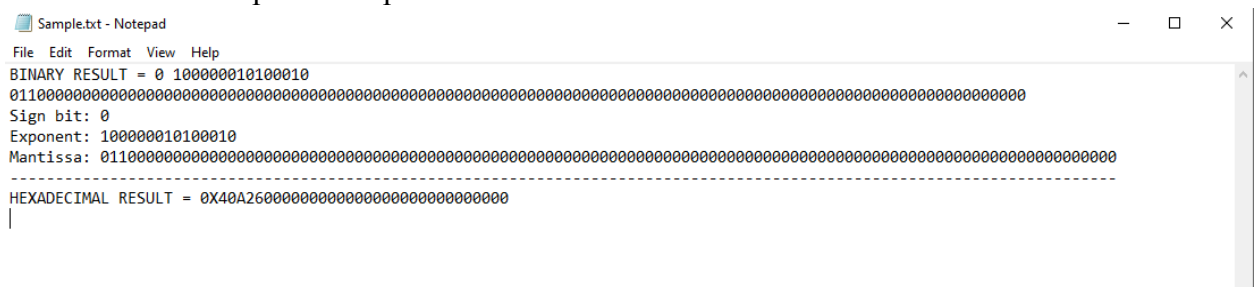
- Input error due to non-binary character input



- Text file output
  - Normal input (First photo: app GUI output, second photo: text file output)



- Text file output of the picture above



VII. Code Repository

The application can be found in a GitHub Repository through the following link: https://github.com/briangabini/csarch2-mco2