**Capstone: Implementation of Ceaser Cipher for FPGA – Part 1**

***Introduction:***

In [cryptography](https://en.wikipedia.org/wiki/Cryptography), a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code, or Caesar shift, is one of the simplest and most widely known [encryption](https://en.wikipedia.org/wiki/Encryption) techniques. It is a type of [substitution cipher](https://en.wikipedia.org/wiki/Substitution_cipher) in which each letter in the [plaintext](https://en.wikipedia.org/wiki/Plaintext) is replaced by a letter some fixed number of positions down the [alphabet](https://en.wikipedia.org/wiki/Alphabet). For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after [Julius Caesar](https://en.wikipedia.org/wiki/Julius_Caesar), who used it in his private correspondence.

The encryption step performed by a Caesar cipher is often incorporated as part of more complex schemes, such as the [Vigenère cipher](https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher), and still has modern application in the [ROT13](https://en.wikipedia.org/wiki/ROT13) system. As with all single-alphabet substitution ciphers, the Caesar cipher is easily broken and in modern practice offers essentially no [communications security](https://en.wikipedia.org/wiki/Communications_security).

***Technique:***

The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places, equivalent to a right shift of 23 (the shift parameter is used as the [key](https://en.wikipedia.org/wiki/Key_(cryptography))):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Plain** | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| **Cipher** | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |

When encrypting, a person looks up each letter of the message in the "plain" line and writes down the corresponding letter in the "cipher" line.

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

Deciphering is done in reverse, with a right shift of 3.

The encryption can also be represented using [modular arithmetic](https://en.wikipedia.org/wiki/Modular_arithmetic) by first transforming the letters into numbers, according to the scheme, A → 0, B → 1, ..., Z → 25.[[2]](https://en.wikipedia.org/wiki/Caesar_cipher#cite_note-2) Encryption of a letter x by a shift n can be described mathematically as,

��(�)=(�+�)mod26.

Decryption is performed similarly,

��(�)=(�−�)mod26.

(Here, "mod" refers to the [modulo operation](https://en.wikipedia.org/wiki/Modulo_operation). The value x is in the range 0 to 25, but if x + n or x − n are not in this range then 26 should be added or subtracted.)

The replacement remains the same throughout the message, so the cipher is classed as a type of [monoalphabetic substitution](https://en.wikipedia.org/wiki/Monoalphabetic_substitution), as opposed to [polyalphabetic substitution](https://en.wikipedia.org/wiki/Polyalphabetic_substitution).

***Top-level block diagram:***

You need to design two blocks: one for encryption and another for decryption





The Verilog top-level module for both block can be as shown below:

**module ceaser\_enc(**

**input clock,**

**input reset,**

**input [7:0] data,**

**input data\_en,**

**output [7:0] data\_out,**

**output data\_ready**

**);**

**endmodule**

***Test procedure:***

Write a test bench to test with a simple string data. Examples: “apples”. Convert every character to ASCII equivalent, note it (A) and push it through the encryption block. Collect the output (B)

The collected output (B) should be pushed through the decryption block and the output of the decryption block should be collected (C).

Compare (A) and (C). They should be same if the cipher is working fine.

***Future:***

In future, this block will be implemented along with the UART blocks to me implemented practically on an FPGA board.

***Instructions:***

Refer to online material and technical papers for more information.

Come up with a specification document for these designs, as it will be helpful for final submission.