2-stage Caesar Cipher

Referent

Email stefano.dimatteo@phd.unipi.it
Teams s.dimatteo1@studenti.unipi.it

Project

Design and implement a 2-stage Caesar Cipher that encrypts each character of the plaintext (only 8-bit ASCII characters representing lower- or upper-case letters) using two different keys for each stage.

The encryption law of the 2-stage Caesar Cipher can be expressed as it follows:

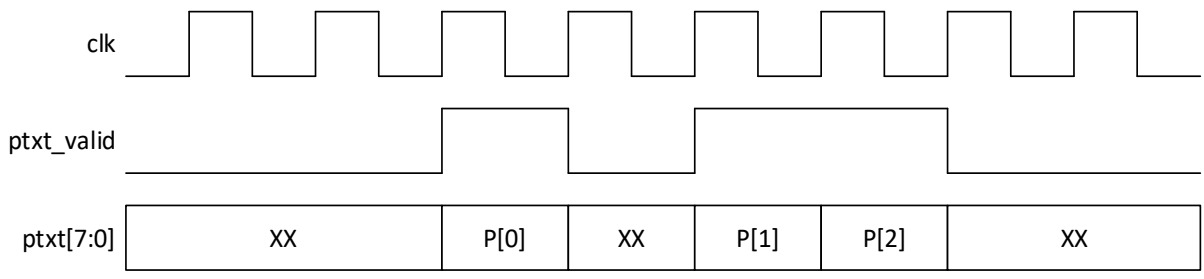
$$C[i] = CS_{K2,\bar{d}}(CS_{K1,d}(P[i]))$$

where:

$C[i]$	is the 8-bit ASCII code of the ith character of ciphertext
$P[i]$	is the 8-bit ASCII code of the ith character of plaintext
CS	is the Caesar Cipher substitution algorithm
$K1$	is the Key to be used for the first stage
$K2$	is the Key to be used for the second stage
d	is the shift direction: 0 = right, 1 = left

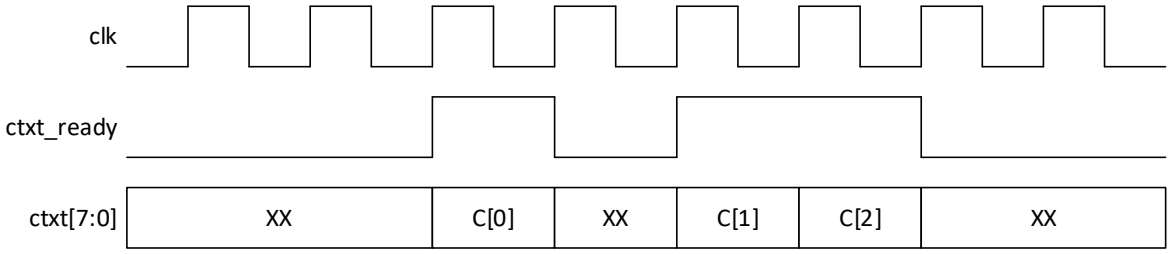
Additional design specifications

- The 2-stage Caesar Cipher shall encrypt one plaintext character per clock cycle;
- The 2-stage Caesar Cipher shall have an asynchronous active-low reset port;
- The plaintext character can be only 8-bit ASCII character representing lower- or upper-case letters (this condition must be checked);
- The keys *K1* and *K2* must be different and comprised between 0 and 26 (these conditions must be checked);
- The 2-stage Caesar Cipher shall feature an input port which has to be asserted when the plaintext character is valid (*ptxt_valid* port): 1'b1, when input plaintext character is valid and has to be consumed by the 2-stage Caesar Cipher, 1'b0, otherwise; the following waveform is expected at input interface of the 2-stage Caesar Cipher;



- The 2-stage Caesar Cipher shall feature an output port which is asserted when the ciphertext character is available at the corresponding output port (*ctxt_ready* port): 1'b1, when output ciphertext character is valid and has to be consumed by the logic resources linked to 2-stage Caesar

Cipher, 1'b0, otherwise; this flag shall be kept to logic 1 at most for one clock cycle; the following waveform is expected at the output interface of the 2-stage Caesar Cipher;



Hints

Develop a testbench that encrypts two plaintext messages (from file or from string): for each plaintext message P, store the corresponding ciphertext C into proper object (file or string), then decrypt C and store the corresponding plaintext P' into proper object (file or string); compare P' with P and check they match (character by character or using string methods). At least one of the two plaintext messages should be greater than 256 characters (around 300, for instance).