



UNIVERSITÀ DI PISA

Masters in Cybersecurity (LM-66)
Hardware and Embedded Security

2-stage Caesar Cipher

Student:

Marwan Haruna

Submitted to:

Prof. Sergio Saponara

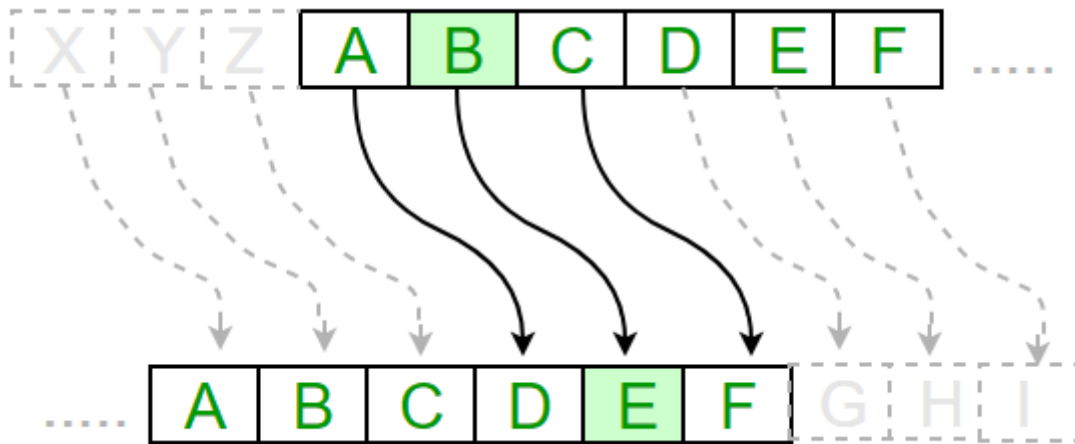
PhD. Stefano Di Matteo

SPECIFICATION ANALYSIS

The project is the implementation of an advanced Caesar cipher in this case the Two Stage Caesar Cipher. requires the development of a hardware module capable of implementing an advanced version of the Caesar's cipher mechanism.

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

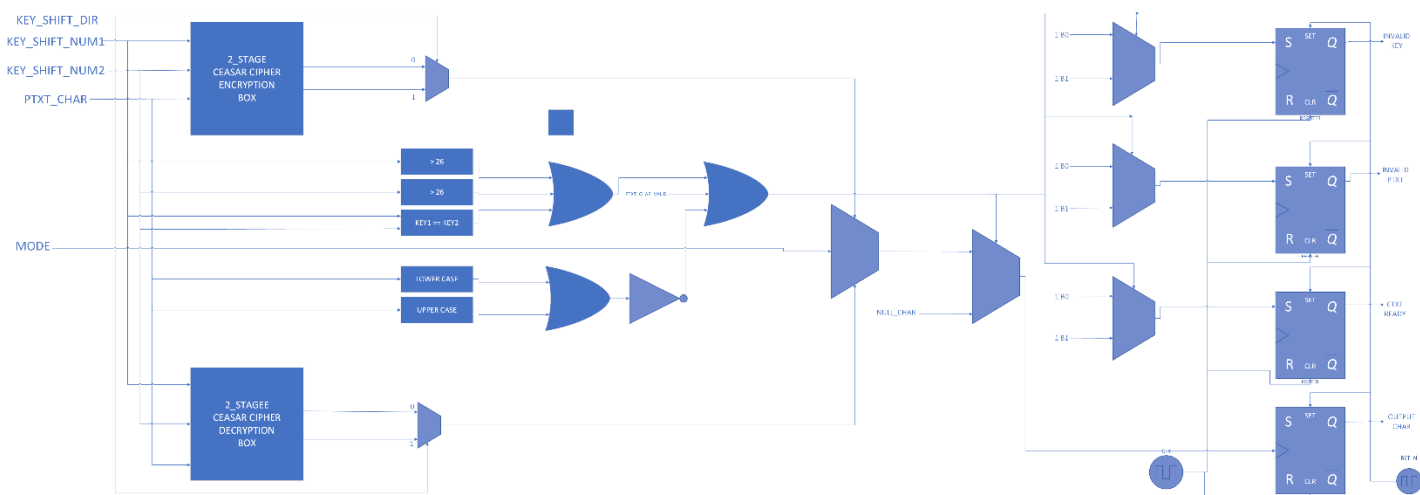
The two stage Caesar cipher takes a plaintext of 8 bits which must be of ASCII characters representing lower- or upper-case letters, each character will be moved by a specific direction with the help of the value d (which is our direction key) and by $k1$ and $k2$ positions in the first and second stage respectively.



Also, there is an encryption and decryption option for all valid characters and if the character is detected to be invalid a null character is used to replace it.

BLOCK DIAGRAM AND DESIGN CHOICES (RTL DESIGN MODULE)

Description



As we can from the above image, the design choice was to have both the encryption and decryption module run in parallel. When we feed the 2_stage_caesar cipher encryption/decryption module with the `key_shift_num1`, `key_shift_num2` and the `ptxt_char`, we can have only two case scenarios. One which we either shift to the right or the other which we shift to the left and both cases do get executed. However, on

one case will be selected and it will be determined by the `key_shift_dir`, when it is 0 the right shift will be selected and if it is 1 the left shift will be selected. As mentioned earlier, both the encryption and decryption scheme are both working in parallel, the mode which we specified will be the one to decide if the encrypted or the decrypted character will be select, if our mode is 0 then it is the former and if it is 1 then it is the latter.

Input

We have about 8 inputs and they are as follows:

- **key_shift_dir**: This is the signal that specifies which direction the keys are going to be used. It has a binary datatype and 1'b0 means it is a right shift while value 1'b1 means it is a left shift.
- **Key_shift_num1**: This is the first key of the cipher and it contains 5 bits only and the max value must not be greater than 26.
- **Key_shift_num2**: This is the second key of the cipher and it contains 5 bits only and the max value must not be greater than 26.
- **Ptxt_char**: This is an 8-bit character which we are going to encrypt or decrypt.
- **Mode**: This is the signal that specifies if we are encrypting or decrypting, it has a binary datatype and when it is 1'b0 it means we are encrypting while when it is 1'b1 it means we are decrypting.
- **Clk**: This is the signal used by the registers to provide the outputs.
- **Rst_n**: This is the signal used to reset the registers that provide the outputs.
- **ptxt_valid**: This signal is used to check if the character is ready to be encrypted or decrypted.
-

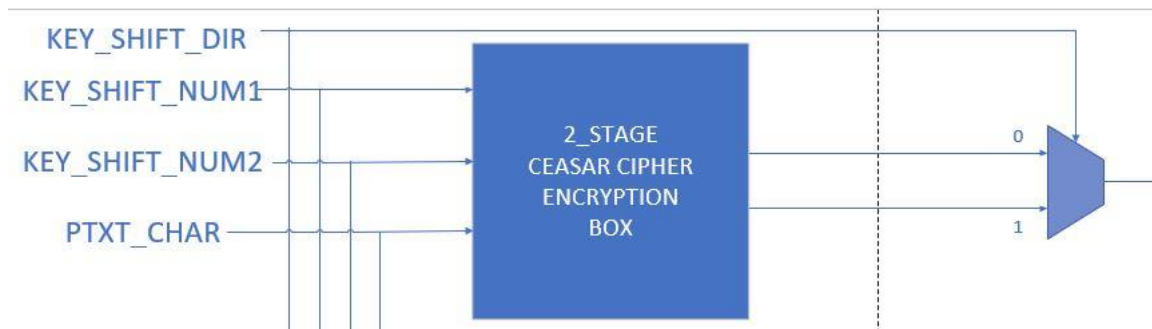
Output

The outputs provided by the system are the following:

- **Invalid_Key**: This is a one bit register that checks if the key is valid, if any of the key is greater than 26 then it is an invalid key or if the keys are both the same. When the register returns 1'b0 then the keys are okay, but if it returns 1'b1 then the keys are not valid.
- **Invalid_ptxt**: This is also a one bit register that checks if the char inserted is an uppercase or lowercase letter and return 1'b0 which is ok else if it returns 1'b1 then it is not an acceptable character.
- **Ctxt_ready**: This is also a one bit register that checks if both the keys and the character inserted, it returns 1'b0 if they are valid or 1'b1 if not.
- **Output_char**: This is an 8 bit register that holds the result of either the encryption or decryption.

Diagrams

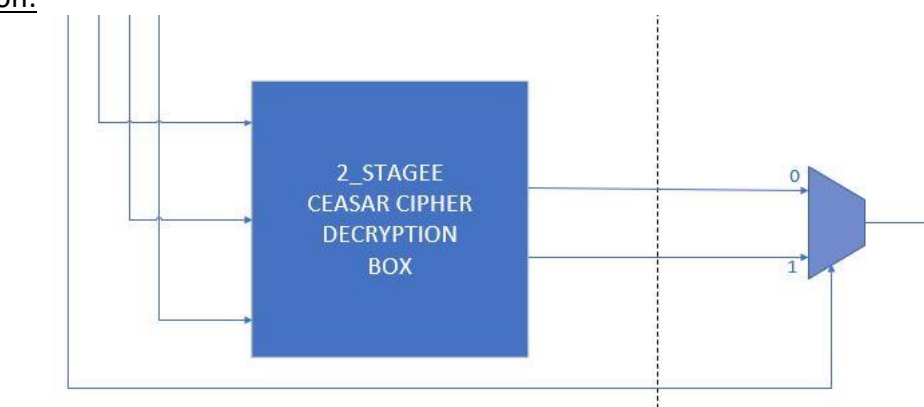
Encryption:



Encryption is done for both the two possible cases but the key_shift_dir determines which to be selected. If the key_shift_dir is 0 then it is a right shift then the ptxt is summed with the key ($Ctxt = ptxt + key_shift_num1 + key_shift_num2$), immediately we check if the ctxt is uppercase and greater than uppercase Z, if it is we subtract the value 26 from it and we perform the second check if it is greater than uppercase Z if it is the we subtract the value 26 from it else we leave the ctxt as it is. Nonetheless we perform the same operation if ctxt is lower case. However, If the key_shift_dir is 1 then it is a left shift then the ptxt is subtracted with the key ($Ctxt = ptxt - key_shift_num1 - key_shift_num2$), immediately after, we check if the ctxt is less than capital A, if it is then we add the value 26 to it and perform the second check if it is greater than capital A if it is we add the value 26 to it again else we leave ctxt at it is. Nonetheless we perform the same operation if ctxt is lower case.

The checking if ctxt is greater than a letter is done to wrap 26 alphabets, so that other values on the ASCII which are invalid characters will not be used as output.

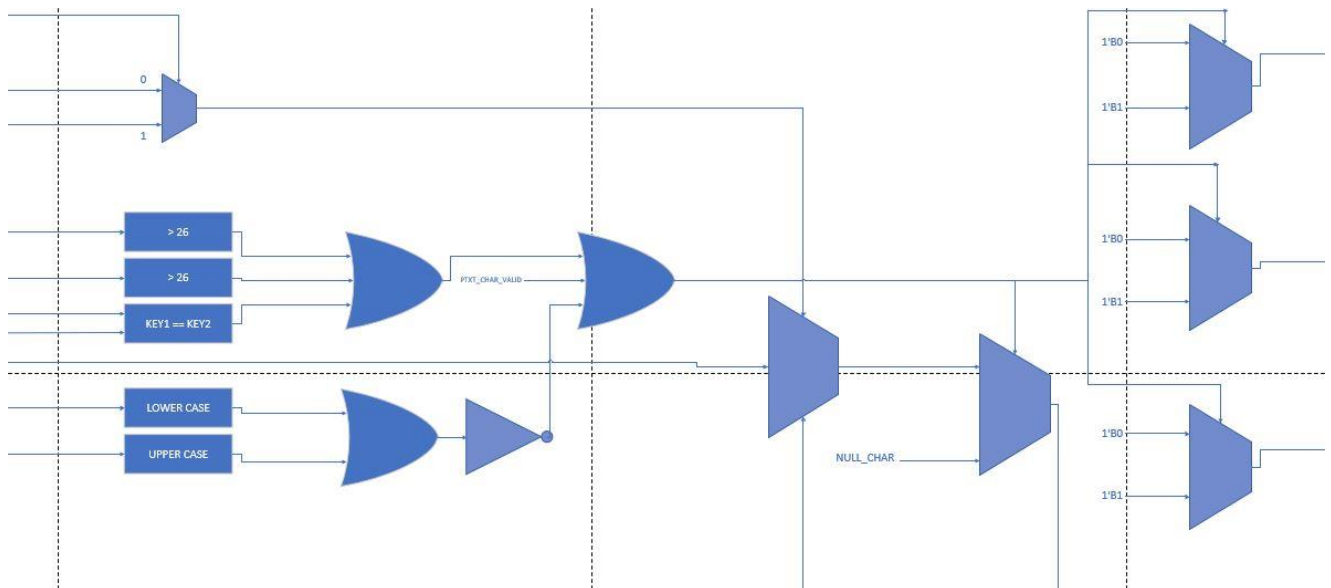
Decryption:



Just as the encryption, both cases of decryption are executing in parallel and the key direction specifies which case is selected. If the shift key is 0 then it will be a right shift and the ptxt and both keys are subtracted from each other ($ctxt = ptxt - key_shift_num1 - key_shift_num2$), immediately we check if the ctxt is uppercase and is less than the value of uppercase A, if so, we add the value 26 to it and perform the second check if it is less than

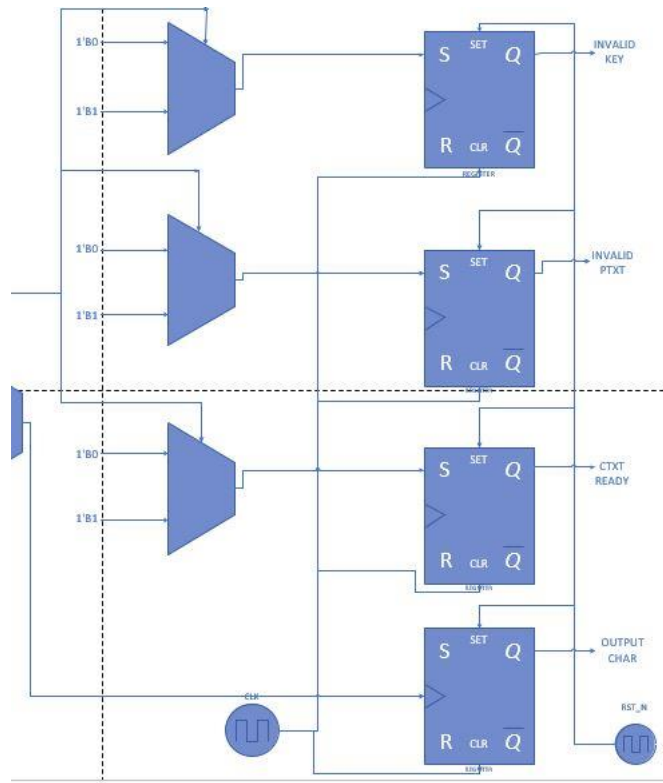
uppercase A then we add the value 26 to it else we leave it as it is. Nonetheless we perform the same operation if ctxt is lower case. However, if the shift key is 1 then it will be left shift and the ptxt is added with both keys ($ctxt = ptxt + key_shift_num1 + key_shift_num2$), immediately we perform the check if ctxt is uppercase and it is greater than uppercase Z, if so, we subtract 26 from ctxt and check if it greater than uppercase Z, if so, we subtract 26 from it else we leave it as it is. Nonetheless we perform the same operation if ctxt is lower case.

Control and Error check:



The error checking mechanism is designed in such a way that if there is a single error the value of the output will be a null character. To enforce this, we take both the keys and check if they are greater than 26 and check if they are the same. If all these conditions check out to be false then a zero signal is sent else a true (1) signal is sent. Also, the ptxt is checked to see if it lies within the accepted ASCII character range which is uppercase A to uppercase B and lowercase a to lowercase z, if it lies within then we send a signal zero else we send one. Both the conditions of the keys and the plaintext must be met in order to output a valid character else it will output a null character.

In addition to the check of the keys, with the signal provided we can get other signals in their registers and output such as the invalid key signal, invalid plaintext, ctxt ready and lastly the output character. Below is a visual representation of the output signals.



TESTBENCH (DESIGN CHOICES, COMMENTS AND C++ MODEL)

C++ model

The implementation of two stage Caesar cipher was used to serve as a test vector to compare and verify that the implementation of the hardware is working correctly. The reason for the selection of C++ is due to its speed and closeness to the hardware as compared to languages such as python. With this we are able to read the exact plaintext file which the hardware implementation reads and we are able to write into a separate file which is used to compare with the hardware implementation.

The C++ program takes three inputs for each plaintext or ciphertext that will be encrypted/decrypted:

- The first is the direction key, which takes 0 or 1 for clockwise and anticlockwise movement.
- The second is Shift key1, which takes a number from 0 to 25 and is checked with the help of $(key1 \% 26)$ to make sure it lies within the range of 0 to 25. Also, this key is used in the first round of shift to specify the length of displacement from the current character.
- The third is Shift key2, which takes a number from 0 to 25 and is checked with the help of $(key1 \% 26)$ to make sure it lies within the range of 0 to 25. Also, this key is used in the second round of shift to specify the length of displacement from the current character.

Upon insertion of the required inputs, the file ptxt.txt is opened and read, its value is stored in a variable which is then sent to the encryption function along with the shift direction and the two shift keys. The encryption takes place and the ciphertext is returned and the program proceeds to save it in the file enc.txt. The same procedure applies for the decryption as it opens the file enc.txt and reads the ciphertext and sends it along with the direction key, the two shift keys and the encryption takes place and the plaintext is returned, which the program proceeds to save it in dec.txt. The above procedure is done twice as recommended in the project specification.

Testbench in System Verilog

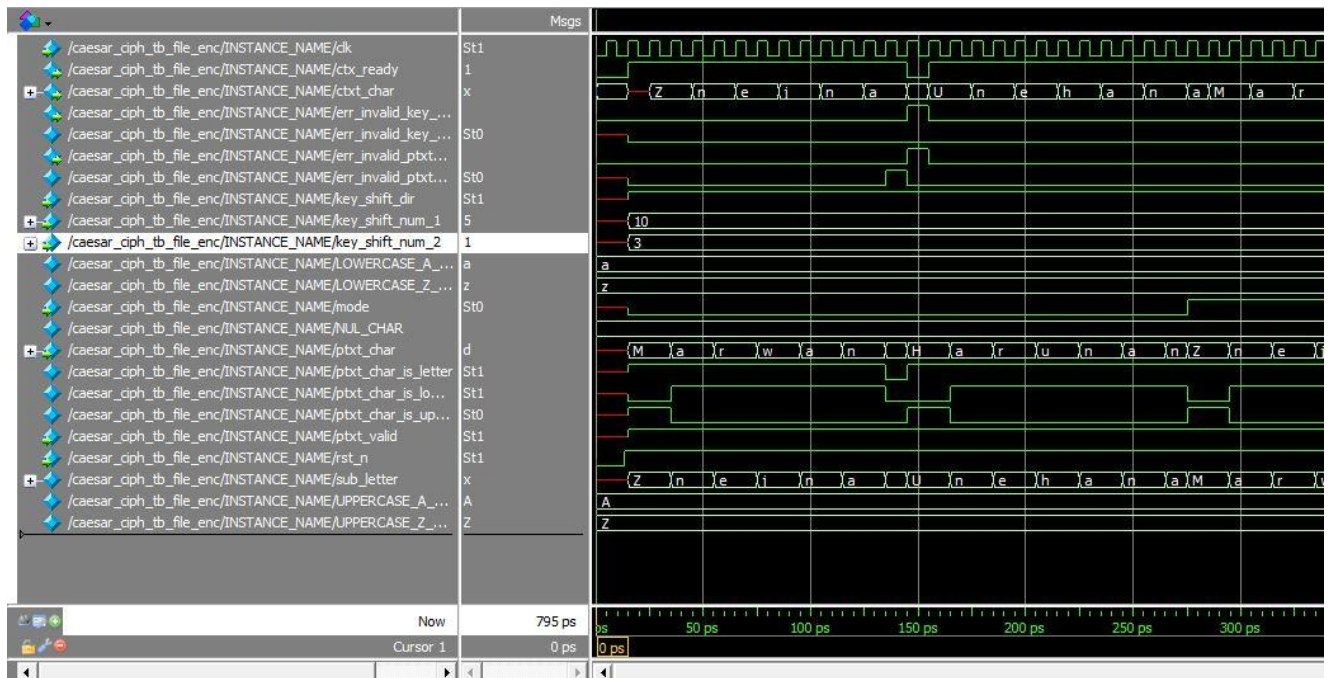
In the hardware implementation, we have two set of testbenches;

- **caesar_ciph_tb_checks:** We test the system by encrypting and decrypting all the characters in the alphabets and display them on the screen. This is done in order to check all possible outcomes are working properly. Some of the
 - This module carries out various tests on all characters of the alphabet by printing on the screen each time the value of the corresponding encryption / decryption in order to observe the behavior of the device and ensure its correct functioning in different cases.
 - The last test in this testbench was specifically designed to test the violation of key size where the size of the key was greater than 26.
- **caesar_ciph_tb_file_enc:** With this module, we are able to read a plaintext from a file known as ptxt.txt and encrypt its content with the available keys. After the encryption we save the encrypted text in a file known as enc_HW.txt. Before moving to the next round a quick comparison is done between the hardware generated ciphertext (enc_HW.txt) with the software generated ciphertext (enc.txt), if they both match a compared done message is displayed else the program stops.
We conduct this test with two set of plaintexts as recommended in the project description.

WAVEFORMS

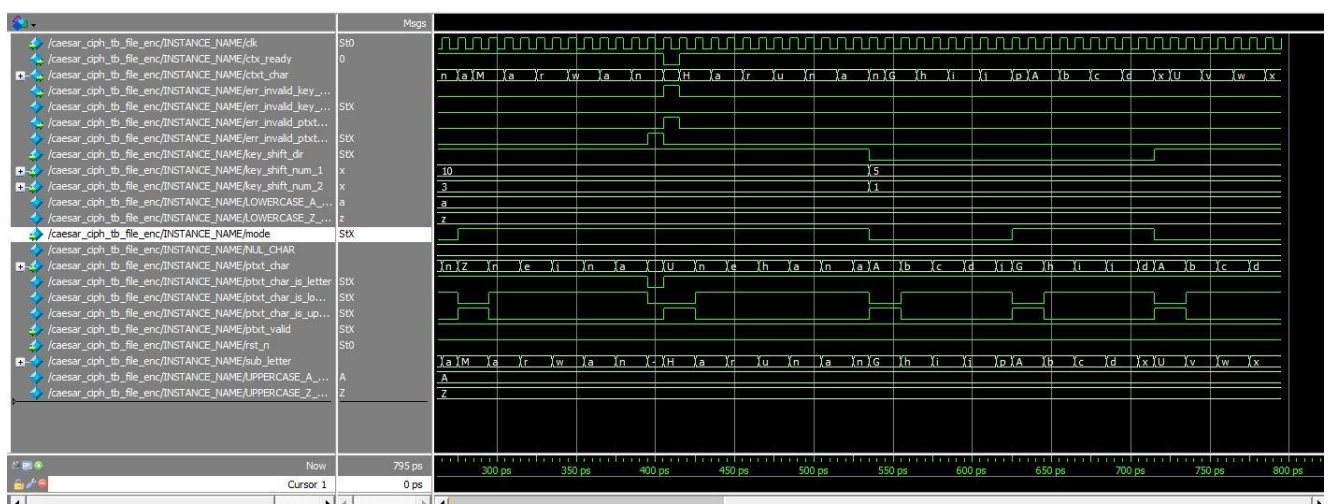
After implementing a Testbench for our module, we could observe the following waveforms.

Encryption waveform



As seen above we are able to encrypt a plaintext with value (Marwan Haruna) with the use of key_shift_dir: 1 as we can see from the waveform it is high, key_shift_num_1: 10 and key_shift_num_2: 3. Also from the waveform we can see that both encryption and decryption is going on parallelly in ctxt_char and ptxt_char. However, the mode decides which the sub_letter get assigned and, in this case, we can see that the mode has a low waveform which means encryption is being selected and we can see the encrypted text in the bottom of the waveform assigned to sub_letter. Finally, we can see that any invalid character has been replaced with a null character.

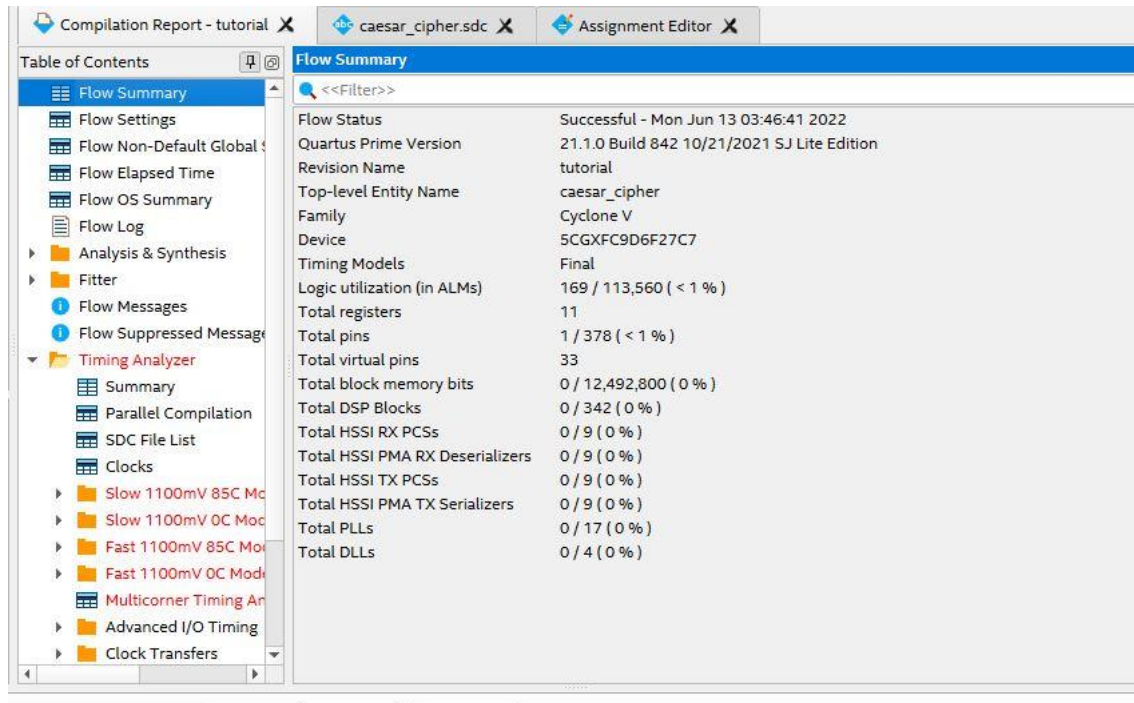
Decryption waveform



As we have seen from the encryption, we also decrypted a ciphertext with value (Zneina Unehana) with same key and direction as the encryption which translates back to (Marwan Haruna), also invalid characters such as spaces, &, * are replaced with a null character.

IMPLEMENTATION OF RTL DESIGN ON FPGA

Quartus summary



The screenshot shows the Quartus IDE interface with the 'Flow Summary' window open. The left sidebar contains a 'Table of Contents' with various categories like 'Flow Summary', 'Analysis & Synthesis', 'Fitter', 'Timing Analyzer', etc. The main area displays the 'Flow Summary' for the project 'caesar_cipher.sdc'. The summary indicates a successful compilation on June 13, 2022, using Quartus Prime Version 21.1.0. The target device is a Cyclone V (5CGXFC9D6F27C7). Key resource utilization statistics are provided, showing that logic utilization is less than 1%, there are 11 registers, 1 pin, and 33 virtual pins.

Flow Summary	
Flow Status	Successful - Mon Jun 13 03:46:41 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	tutorial
Top-level Entity Name	caesar_cipher
Family	Cyclone V
Device	5CGXFC9D6F27C7
Timing Models	Final
Logic utilization (in ALMs)	169 / 113,560 (< 1 %)
Total registers	11
Total pins	1 / 378 (< 1 %)
Total virtual pins	33
Total block memory bits	0 / 12,492,800 (0 %)
Total DSP Blocks	0 / 342 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 17 (0 %)
Total DLLs	0 / 4 (0 %)

From the flow summary we can see that we are utilizing a CYCLONE V FPGA, the logic utilization (in ALMs) is less than 1% of the total resources of the FPGA. Furthermore, we can see we have total of 11 registers which 8 are dedicated for our `ctxt_char`, while the remaining 3 are for `err_invalid_ptxt_char`, `err_invalid_shift_num` and `ctxt_ready`. In addition, we have a total number of pins as 1 which is our clock as it is the only connected signal, we have a total virtual pin as 33.

STATIC TIMING ANALYSIS

Below is the SDC file in Quartus that contains the timing constraints.

```
1 create_clock -name clk -period 5 [get_ports clk]
2
3 set_false_path -from [get_ports rst_n] -to [get_clocks clk]
4
5 set_input_delay -min 1 -clock [get_clocks clk] [all_inputs]
6 set_input_delay -max 2 -clock [get_clocks clk] [all_inputs]
7 set_output_delay -min 1 -clock [get_clocks clk] [all_outputs]
8 set_output_delay -max 2 -clock [get_clocks clk] [all_outputs]
9
```

Inside the SDC file we have a clock period, an asynchronous reset and a minimum and

maximum input and output delays. To get close to our target frequency (100MHz) we had to map our pins to the virtual pins of the FPGA and with that the path delay reduced and the frequency increased.

As you can see below, we at slow 1100mV 85C we got a value of 92.76 MHz while with slow 1100mV 0C we got a value of 89.39 MHz

Slow 1100mV 85C Model Fmax Summary					Slow 1100mV 0C Model Fmax Summary				
<<Filter>>					<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note		Fmax	Restricted Fmax	Clock Name	Note
1	92.76 MHz	92.76 MHz	clk		1	89.39 MHz	89.39 MHz	clk	