Run Length Encoder

Name - Atishay Jain
Roll Number - 210050026

5th February 2023

1 Assumptions

- I have used `IEEE.numeric_std.all` library for conversion of data types
- I assumed that RLE encoder works in such a way that it first takes all the inputs and then produces the outputs one by one, and that is how I implemented it
- Instead of low to high transition of data_valid line, I assumed that it outputs z whenever data_valid line is 1 (because I am outputing all at the end, so the transition is data_valid there didn't make sense as the output was continuous)
- One more small point, that since I was using windows, so I have used ModelSim for testing code. I'm not sure if any issue regarding some flag or something may occur. Please take care of this point.

2 Implementation

- I take the inputs one by one at the beginning on the rising edge of clock
- I used a array of `std_logic_vectors` of length = 100 (assuming it would be enough since the max requirement would be $32 * 3$ in worst case) as a buffer
- I used a variable (named *tmp*) to store the previous input. At each clock cycle when new input arrives, it is compared with this previous value -
  - For first time, when there is no previous value, then previous value variable is assigned the current input. A variable for counting the number of occurences is also used, which is set to 1 here
  - Now, if the same input arrives again, then counting variable is incremented
  - If a new input arrives, then before storing that new input into *tmp*, I first store the previous input along with its count and `esc` character according to the cases based on value of count of that character
- Cases based on count value are such that when we have to store any character are -
  - If $count \geq 2$ and $count < 16$, then store `esc` *count* `character`
  - If count is greater than 15, then store it as assuming different from 15 (rest same as above)

  - If count is 2, then
    - If character is `esc`, then store `esc` 1 `esc`
    - else, store the given character twice
  - else if count is 1, then
    - if character is `esc`, then store `esc` 1 `esc`
    - else store the character one time

- At the end, I have used a dummy signal (00000000) which denotes that all the inputs have processed and by now stored into the buffer. Now, this signal is kept on sending and the outputs stored in buffer are outputed in z one by one.

3 Structure - RLE

Following extra variables and signals are used -

- `storage` - array of `std_logic_vectors` of size 100, used as buffer
- `count` - integer variable to store count of aa input
- `cnt` - integer variable which stores the index of the buffer array at which we should store the current data when needed
- `index_stor` - index from which data from buffer will be outputed at last
- `num` - A logic_vector which is the vector form of count, it is needed when storing 8-bit from of count variable, which was integer
- `tmp` - A variable which stores the previous input when a new input is received on the clock cycle. It is updated/not updated as per need of the code
- `esc` - the sential ESC character
- `all_same` - an integer to denote if all the characters are same or not, it was being used in my implementation for sending initial output signal twice if all are same

4 Structure - Test Bench

- The clock is first set to 0 and rst is set to 1
- Now, the while loop of process begins and it starts taking input from `test.txt` file. The clock values keep on changing to process input
- When the input while loop ends and all the data is taken from `test.txt`, then another loop begins in which a constant input of a special character (00000000 as used by me) is sent to denote RLE encoder to denote that input is over and it starts outputing z by just providing the output from the stored buffer and data_valid is set to 1 for output.
- It writed the output to `out.txt` text file whenever data_valid line is 1 and the value of z is not undefined. The program then ends its output as z is undefined for rest of the simulation time.