BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
Dept. of Electrical and Electronics Engg.

Burrows–Wheeler transform & Inverse Burrows–Wheeler transform

By

Vemparala Sai Kumar :- 2021H1400155P

Rangu Suhas Rahul :- 2021H1400162P

Instructor In-Charge :-
Karri Babu Ravi Teja
Assistant Professor, EEE Department,
BITS Pilani, Pilani Campus

Epitome of Project

FPGA Used:- Zynq-7000 xc7z020 clg484-1

Representation used: - Burrows-Wheeler Transform (BWT)
And its Inverse.

Key points/ salient Features:-

- BWT and IBWT heavily dealt with sorting and comparing the ASCII value of the characters in a string.
- The maximum length of characters in a string is taken as 1023.

Modules Used:-

1. BWT module (Length of string included)
2. IBWT module (Length of string included)

Module descriptions :-

1. Burrows-Wheeler transform:-

Name of the module: BWT.v
Inputs: [(1023*8)-1:0]str;
Output: reg [9:0]n;

Approach :-

In the beginning we have taken the input of the string from a text file which is then stored in a reverse order fashion into RAM memory and indices for every clock pulse and counting the length of the string. Thereby accessing memory individually and sorting them based on the ASCII value and then sorting the characters that are similar by comparing with the next character analogous to the sorting array concept but without any memory matrix into the RAM_sort memory including their indices in a separate memory as sa_sort using control signals. Finally the allotted indices are then added with +1 as we have stored in a reverse order fashion then compared the indices values in the original RAM memory and accessing the character from it and thereafter storing into RAM_bwt memory. The whole process has been done without for loops.

Problems faced :-

- Felt difficulty with If loops using control signals to operate depending on the compilation of an operation sequentially.
- Encountered difficulty with the sorting of the characters in a memory logic and then storing into another memory.
- In finding BWT there was no way to make any control signal high in the code so as included BWT logic inside the sorting code.
- Major issue faced in finding BWT without sorting array logic.

Utilization analysis:-

Resource	Utilization	Available	Utilization %
IO	32	200	16

2. Inverse Burrows-Wheeler transform:-

Name of the module: IBWT.v
Inputs: [7:0] char_bwt, [9:0] length, write_enble, clk, [9:0] addr
Outputs: [7:0] char_ibwt, [9:0] addr_ibwt, done

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Block RAM Tile (140)	Bounded IOB (200)	BUFCTRL (32)
IBWT	**42417**	**16523**	**5747**	**2424**	**0.5**	**57**	**1**

Approach:-

In the beginning we have taken the BWT string as an input from a text file which is then stored one by one character with their index values from testbench to the main module into RAM memory along with the length of the string at the positive edge of the clock pulse. Thereby in the main module, accessing memory and sorting them based on the ASCII value with swapping of their initial indexes and then storing in the new memory block. Starting from the first character in the sorted memory, the swapped index gives the address of the first character of the IBWT string. This character is stored in a new memory at the first address. The swapped index of this character gives the second character which is again stored in that memory at the second address. This process is continued till the $ character is read. This new memory consists of the IBWT string which is sent back from the main module to the testbench which is printed in the text file one by one. The whole process has been done without for loops.

Problems faced:-

- Faced issue in writing loops sequentially without using “for” statements as nested looping is required for sorting, as they are utilizing a large hardware and unable to get synthesized.
- We are able to do the transform for only 256 characters. For

strings with more than 256 characters, the first 256 characters are only getting transformed and repeated again.

Utilization analysis:-

Resource	Utilization	Available	Utilization %
LUT	**42417**	**53200**	**79.73**
LUTRAM	**256**	**17400**	**1.47**
FF	**16523**	**106400**	**15.53**
BRAM	**0.5**	**140**	**0.36**
IO	**57**	**200**	**28.5**

TESTING STRATEGY ADOPTED:-

Initially shorter strings were tested, later the same was tested on larger strings. For strings having length below 256, they are getting transformed perfectly, but it is not working for string length more than 256.

S.No.	Module	Behavioral Simulation	Synthesizable
1.	BWT	Yes	Yes
2.	IBWT	Yes	Yes