Lab-6 Report

Balasubramanian P - 200050103

March 2022

1 Question

You need to design a data compression circuit using run-length encoding. It replaces continuously repeated occurrences of a byte with a repeat count and the byte value. The circuit receives a fresh byte at every positive transition of an externally supplied clock.

2 Entity

```
entity RLE_Compressor is
        port(clk, rst: in std_logic;
        input: in std_logic_vector(7 downto 0);
        DataValid: out std_logic;
        output: out std_logic_vector(7 downto 0)
    );
end entity;
```

3 Block diagram of Design

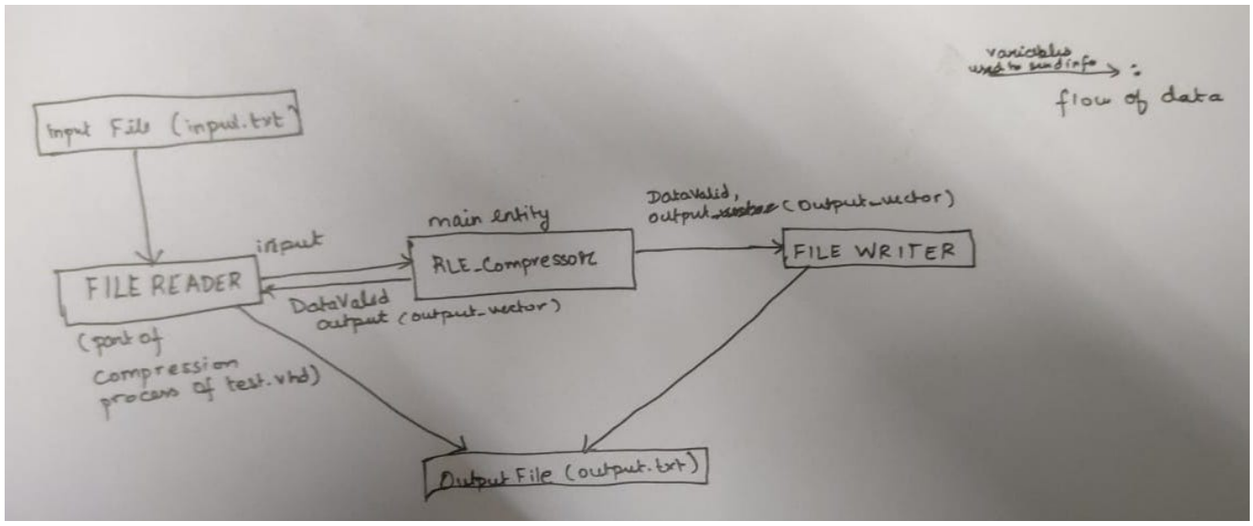

Figure 1: Block diagram of Design

4 RLE_Compressor

All the components are extensively commented in my code. So I am pasting photos of the code and some additional comments for each entity.

- 2 processes used.
- First process to get inputs from the test bench file reader entity.
- Other process is where the compression takes place.

```
begin
    process (clk)                                   --process to get inputs till tot_count (of input characters) reaches 64
    begin
    if(rising_edge(clk)) then
        if(rst = '1') then
        else
            if(tot_count < 64) then                 --tot_count is the index used to write the inputs gotten into the input buffer
                input_buff(tot_count) <= input;
                tot_count <= tot_count + 1;
            end if;
        end if;
    end if;
    end process;
```

Figure 2: Process 1

```
   process (clk) --the main compression circuit
   begin
   if(rising_edge(clk)) then
       if (rst = '1') then     --if rst==1 then the compressor is reset
           curr_count <= ZERO;
           DataValid <= '0';
           output <= ESC;
           curr_char <= ESC;
```

Figure 3: Process 2 rst='1'

```
else --not reset state
    if(state = 0) then  --the compressor has 4 states. when in state==0, the imput is read and processed
                        --to make ready for output character. output char may be generated and state may be changed
                        --or will remain in state=0 till generated.
        DataValid <= '0';
        if(read_count + 1 < tot_count ) then -- read_count is the index with which we read the buffer
            curr_char <= input_buff(read_count);
            curr_count <= curr_count + ONE;
            read_count <= read_count + 1;
            if(input_buff(read_count) /= input_buff(read_count + 1)) then --if the current character and next characters not equal
                if(input_buff(read_count) /= ESC) then  --current character is not equal to ESC
                    if(curr_count = ZERO) then --means the current char is unrepeated so state is set to 1
                        state <= 1;
                    elsif(curr_count = ZERO + ONE) then --means the current char is repeated twice so state is set to 1
                        state <= 2;
                    else    --means the current character is repeated more than twice so state set to 3 where
                            --repeated chars are dealt with
                        state <= 3;
                    end if;
                else --means the current character is ESC so state set to 3
                    state <= 3;
                end if;
            elsif(input_buff(read_count) /= ESC and curr_count = ZERO + "00000100") then --case where character is repeated 5 times,
                        --state set to 3 and the 'ESC 5 curr_char' is outputted
                state <= 3;
            elsif(input_buff(read_count) = ESC and curr_count = ZERO + "00000101") then --case when ESC is repeated 6 times,
                        --state set to 3 and the 'ESC 6 ESC' is outputted
                state <= 3;
            end if;
```

Figure 4: Process 2 rst='0' and state=0

```
        elsif(read_count = 63 and tot_count = 64) then --case when the last input character is read
            curr_char <= input_buff(read_count);
            curr_count <= curr_count + ONE;
            read_count <= read_count + 1;
            if(input_buff(read_count) /= ESC) then  -- if last char is not ESC
                if(curr_count = ZERO) then --unrepeated char
                    state <= 1;
                elsif(curr_count = ZERO + ONE) then -- twice repeated character
                    state <= 2;
                else --multiple times repeated character
                    state <= 3;
                end if;
            else --char is ESC, so set state to 3
                state <= 3;
            end if;
        end if;
```

Figure 5: Process 2 rst='0' and state=0

```
        elsif(state = 1) then --when in state==1,one character output is given out and state set back to 0(read and process state)
            DataValid <= '1';
            output <= curr_char;
            state <= 0;
            curr_count <= ZERO;
        elsif(state = 2) then --when in state==2, one character is given as output and state is set to
                                --1(one more char to be outputted to be done)
            DataValid <= '1';
            output <= curr_char;
            state <= 1;
        elsif(state = 3) then --when in state==3, means that a repetition more than 2 counts is found and the output char must
                            --be ESC. Then state is set to 4(where the count of the repetition is outputted)
            DataValid <= '1';
            output <= ESC;
            state <= 4;
        elsif(state = 4) then --when in state==4, count of the repeated variable is ouputted and then state set to 1,
                                --where the character which was repeated is outputted
            DataValid <= '1';
            output <= curr_count;
            state <= 1;
        end if;
    end if;
end if;
end process;
```

Figure 6: Process 2 rst='0' and states 1,2,3,4

5 File Reader in Testbench (test.vhd)

```
begin
    input_vector <= "00011011";                                 --initialisation
    wait for 200 ps;

    --FILE READER (and some writitng too)
    while LINE_COUNT < 64 loop                                  --read all 64 bytes in 64 lines...
        readLine (INFILE, INPUT_LINE);
        read(INPUT_LINE, input_bit_vec_var);
        input_vector <= to_std_logic_vec(input_bit_vec_var);    --read the input byte
        wait for 120 ps;
        if(DataValid = '1') then                                --when DataValid becomes 1 or high, write the output_vector
            write(OUTPUT_LINE, to_bit_vec(output_vector));                  --onto the output.txt file
            writeline(OUTFILE, OUTPUT_LINE);
        end if;
        wait for 80 ps;
        LINE_COUNT := LINE_COUNT + 1;
    end loop;
```

Figure 7: File Reading Code

6 File Writer in Testbench (test.vhd)

```
    --FILE WRITER
    while LINE_COUNT < 512 loop --waiting for output to be generated and if DataValid is high, then
        input_vector <= "00011011";         --write the output into output.txt
        wait for 80 ps;
        if(DataValid='1') then
            write(OUTPUT_LINE, to_bit_vec(output_vector));
            writeline(OUTFILE, OUTPUT_LINE);
        end if;
        wait for 80 ps;
        LINE_COUNT := LINE_COUNT + 1;
    end loop;
    wait;
end process;
```

Figure 8: File Output Code