BIRZEIT UNIVERSITY
FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
CHIP DESIGN VERIFICATION
ENCS5337

Phase NO.I: Reference Model
Design Verification of a Hardware Compression and Decompression Chip

---

Prepared By

Doha Hmeid 1190120

Raghad Afaghani 1192423

Instructor

Dr. Ayman Hroub

Section 1

BIRZEIT

April– 2024

Abstract

This report outlines the first phase of a design verification project for a hardware compression and decompression chip. Reference Model Phase, the primary objective of this phase is to create a reference model code that accurately represents the expected behavior and functionality of the hardware compression and decompression chip.

Table of Content

List Of Images

1. Theory

1.1 Compression Algorithm

Compression is a fundamental concept in data processing aimed at reducing the size of data while preserving its essential information. The compression algorithm implemented in a hardware chip typically follows a systematic set of steps to achieve efficient data compression.

When input data is received through the data_in port, it is compared to the existing data stored in the chip's internal memory, known as the dictionary memory. If a match is found between the input data and an entry in the dictionary memory, it indicates that the input data already exists. In this case, the chip writes the corresponding index of the stored data into the compressed_out port. This index value represents a compressed version of the input data, allowing for efficient representation and storage of repeated patterns or information.

In situations where the input data is not found in the dictionary memory, suggesting it is new data, the chip allocates the next available slot within the memory to store the input data. Subsequently, the chip writes the corresponding index of the newly stored data into the compressed_out port and increments the index value to reflect the addition of the new entry.

The physical index register of the chip typically has a width of 32 bits. However, the actual number of bits used from this register depends on the size of the dictionary memory. For instance, if the dictionary memory consists of 256 locations, each occupying 80 bits, the chip utilizes the least significant eight bits of the 32-bit index register. This ensures that the compressed data size is reduced to 8 bits, optimizing storage and transmission efficiency.

To handle scenarios where the internal memory becomes full during the compression process, the chip generates an error signal through the output response. This error signal acts as an indicator that the available memory space is insufficient to accommodate further compression operations, providing feedback to the system for appropriate error handling or memory management.

1.2 Decompression Algorithm

The decompression algorithm in the hardware chip functions as the inverse of the compression algorithm. When the chip receives compressed data through the compressed_in port, it compares the value of the compressed data to the current index register value. If the compressed data value is less than or equal to the index value, the corresponding decompressed data exists in the dictionary memory, and it is outputted through the decompressed_out port. However, if the compressed data value is greater than the index value, indicating the absence of the decompressed data in the dictionary memory, an error is reported.

1.3 Input/Output ports

The input/output ports of the compression/decompression chip are listed in Table 1 as follows:

Port	Default Width (#bits)	Direction	Description
clk	1	Input	
reset	1	Input	Clears the dictionary memory and the index register
command	2	Input	Specifies the chip operation 00: No operation 01: Compression 10: Decompression 11: Invalid command, report an error
data_in	80	Input	Data to be compressed
compressed_in	8	Input	Data to be decompressed
compressed_out	8	Output	Output compressed data
decompressed_out	80	Output	Output decompressed data
response	2	Output	Shows the status of the output 00: no valid output 01: valid compressed_out 10: valid deccompressed_out 11: Error

Table 1. Input / Output Ports

1.4 Block Diagram

The block diagram illustrates a data processing system designed for compression and decompression tasks. It receives various inputs such as raw data, compressed data, commands, a reset signal, and a clock signal. The system consists of two primary components: an Index Register and a Dictionary Memory. These components collaborate to generate compressed and decompressed outputs, while also providing a response output that indicates the operation's status or outcome. The diagram showcases a well-organized data processing flow, incorporating control and synchronization mechanisms to manage the data effectively.

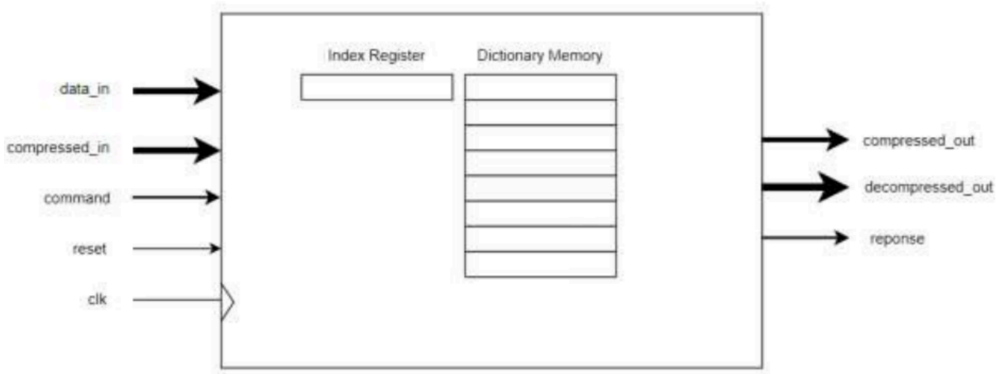

Figure 1. Dictionary Block Diagram

1.5 Reference Model

The reference model implemented in this phase serves as an executable specification, providing a golden model that accurately predicts the correct results based on the given stimulus. The implementation approach for the reference model can vary depending on factors such as the way specifications are written and the specific design requirements. It offers flexibility and adaptability, allowing for different approaches to be employed in implementing the reference model, tailored to the specific needs of the project.

2. Procedure

2.1 Python Demo App

First, a reference model in Python is developed to demonstrate the simplicity and functionality of our reference model. This model is utilized as a foundational representation of our compression and decompression algorithms. Data is simulated using predefined entries. The results are then displayed directly in the console. This initial Python implementation enables the core functionality of our phase to be clearly showcased before proceeding to integration into the system Verilog environment.
The code for the reference model can be found in Appendix A and the results provided in the figures below.

1. The data "Raghad" is compressed successfully, resulting in a compressed version. This compressed data is assigned an index value of 0.

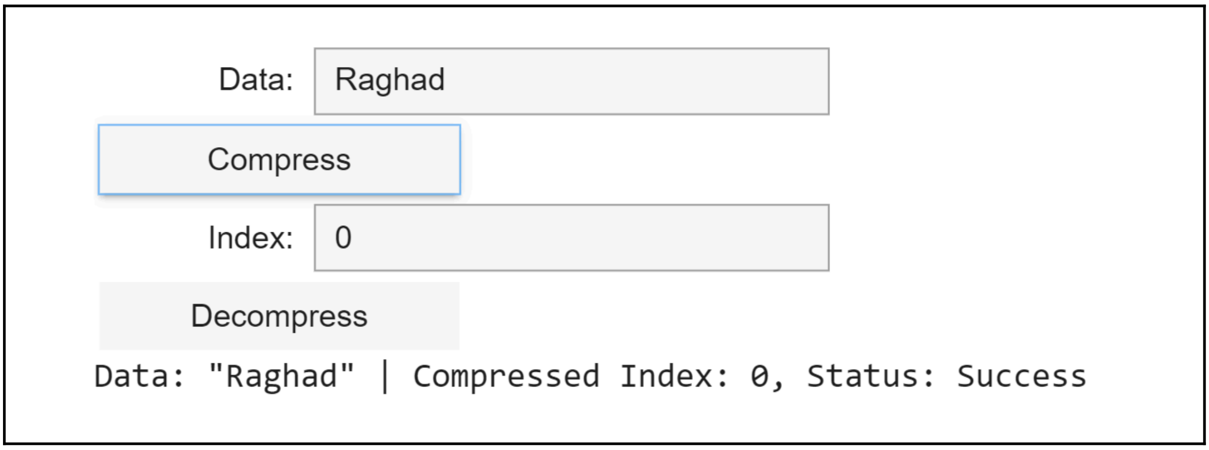

Figure 2. Compression in Python CaseTest I

2. In a similar manner, the data "Doha" is also compressed successfully and given an index value of 1.

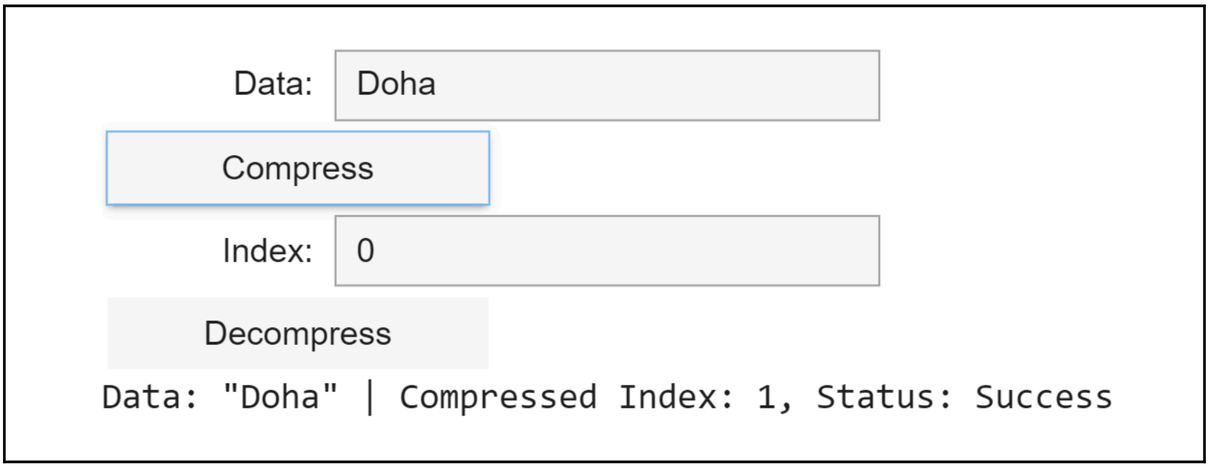

Figure 3. Compression in Python CaseTest II

3. A request is made to decompress the data associated with index 0. The decompression software correctly retrieves the original data, which is "Raghad." This confirms that the data corresponding to index 0 is indeed "Raghad."

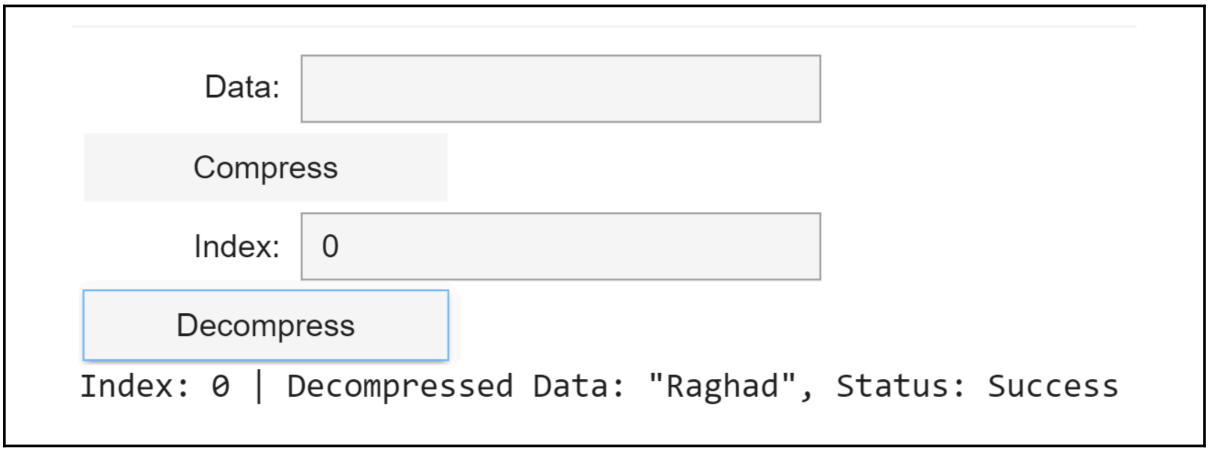

Figure 4. Decompression in Python CaseTest I

4. Another decompression request is made, this time with an index value of 1. The decompression process successfully retrieves the data associated with index 1, which is "Doha."

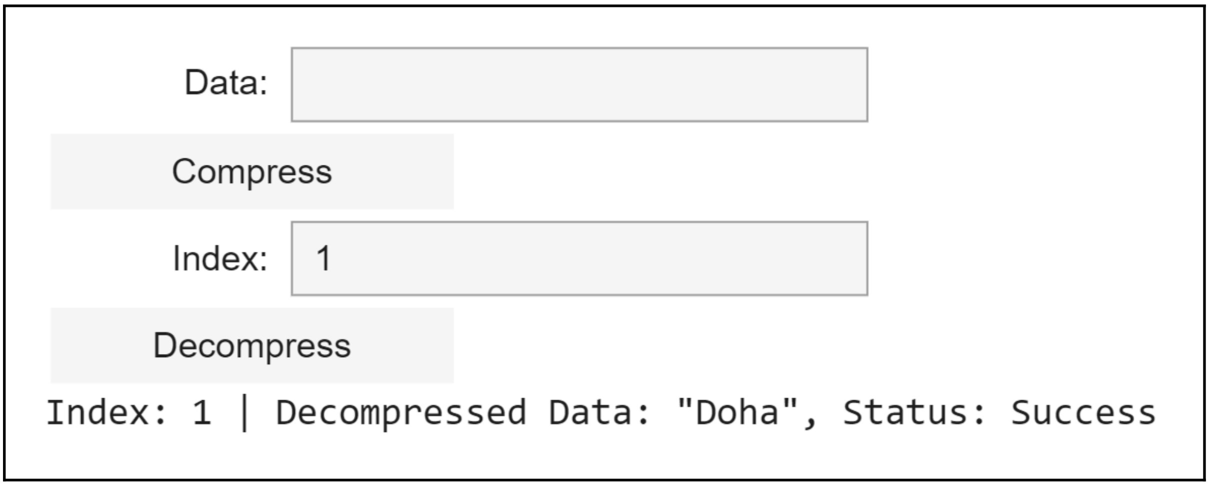

Figure 5. Decompression in Python CaseTest II

2.2 System Verilog

Second, a reference model in System Verilog due to simplicity in integrating it to our project. The code for the reference model can be found in Appendix A. Additionally, the test bench code can be found in Appendix B.

A file named memory.hex was created to simulate the actual memory. The initial values for the memory file were:

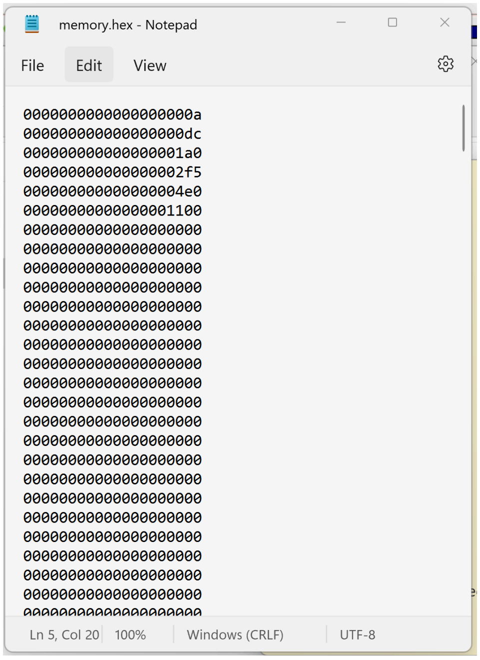

Figure 6. Memory snapshot before the test

After running the test bench, we got the following memory. Notice that when we entered data_in as 0x200 and 0x03. These values were not in the original memory, so after running the program, they were written on the last empty cell from the memory as expected.

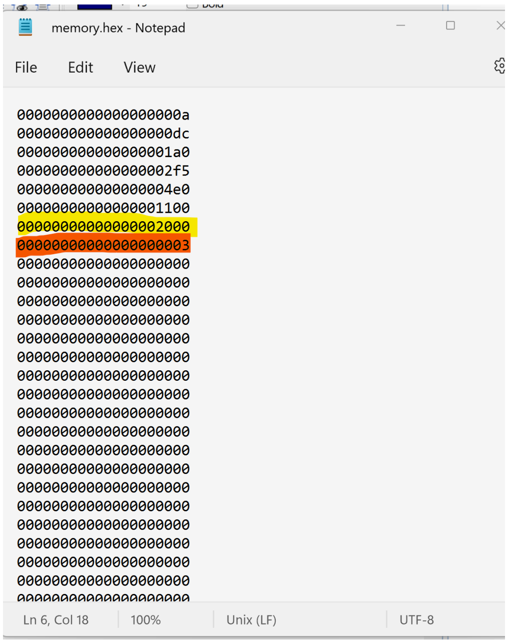

Figure 7. Memory snapshot after the test

The tests and their results are explained below:

1. When the command was 00, this means no operation and the response gave 00 which is no valid output since there is no operation.

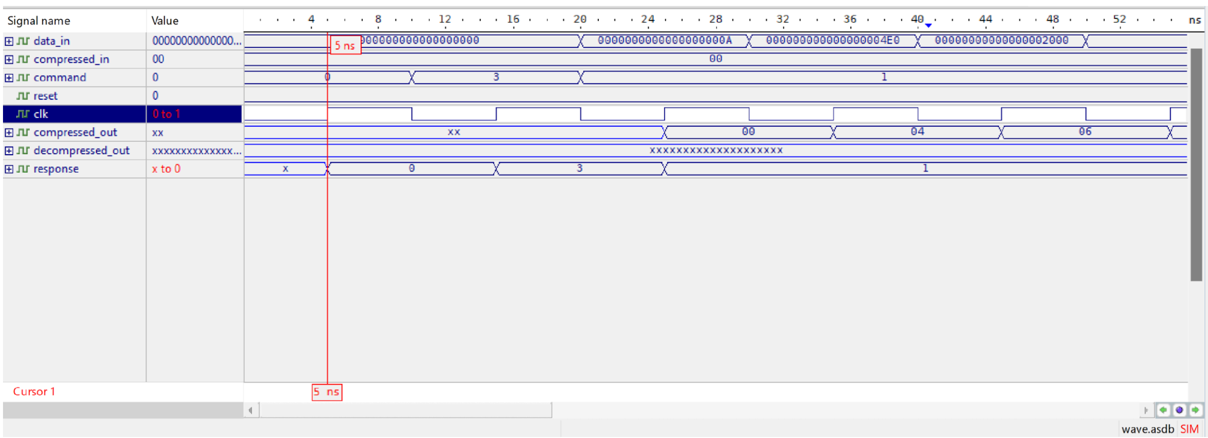

Figure 8. Reference Model CaseTest 1

2. When the command was 3 = 11, this means invalid command and the response gave 3 = 11 which indicates that an error happened.

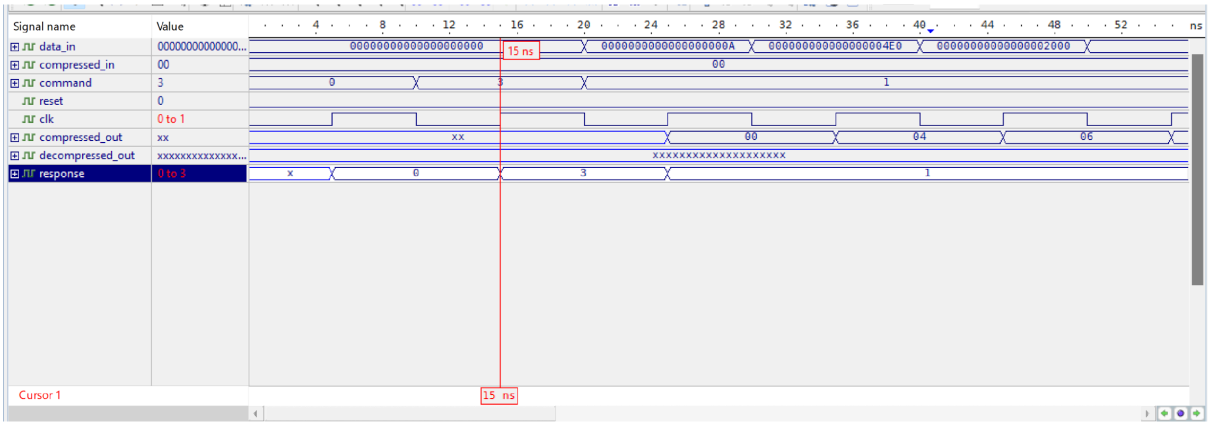

Figure 9. Reference Model CaseTest 2

3. When the command was 1 = 01, this means compression operation and the data_in was 0x0A which exists in index 0 in the memory as shown in the memory snapshot in Figure[6]. We got the response as 1 = 01 which indicates that the data was found in the memory and the compressed_out is 0x00 which refers to the index the data was found in the memory.

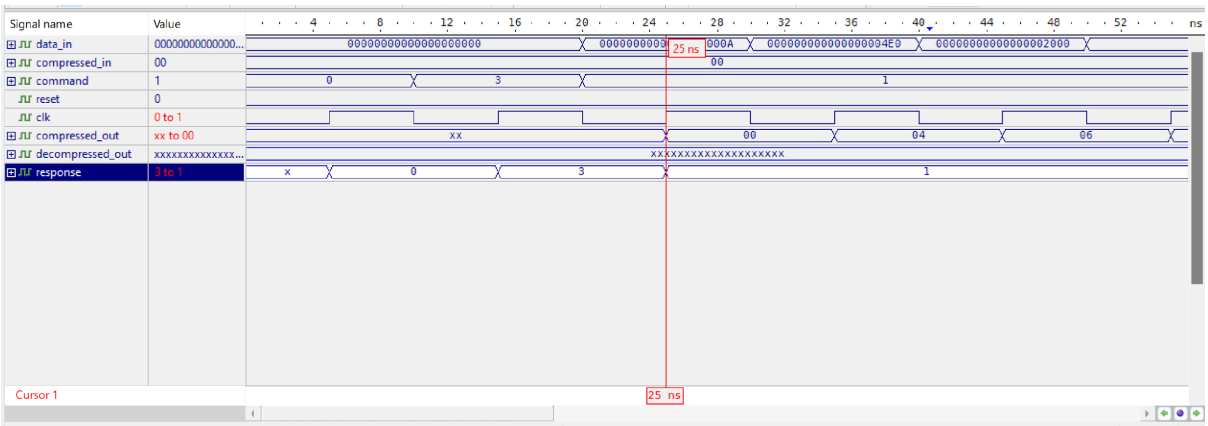

Figure 10. Reference Model CaseTest 3

4. When the command was 1 = 01, this means compression operation and the data_in was 0x4E0 which exists in index 4 in the memory as shown in the memory snapshot in Figure[6]. We got the response as 1 = 01 which indicates

that the data was found in the memory and the compressed_out is 0x04 which refers to the index the data was found in the memory.

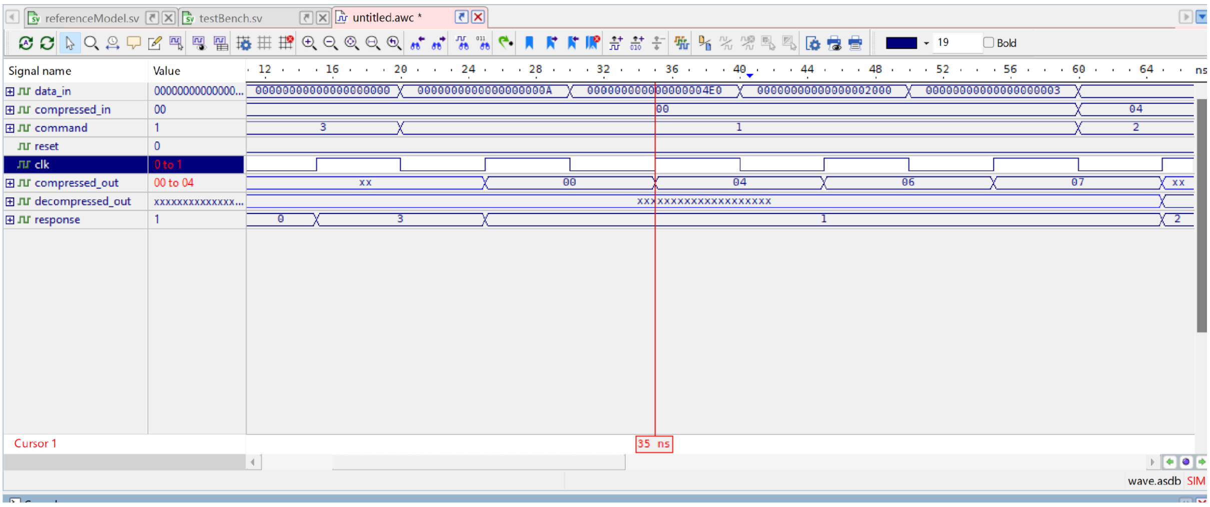

Figure 11. Reference Model CaseTest 4

5. When the command was 1 = 01, this means compression operation and the data_in was 0x2000 which doesn't exist in the memory. The data was written at the current index register which points to the last empty slot in the memory which is at index 0x06. So, compressed_out is 0x06 and the response as 1 = 01. After that the index register was incremented.

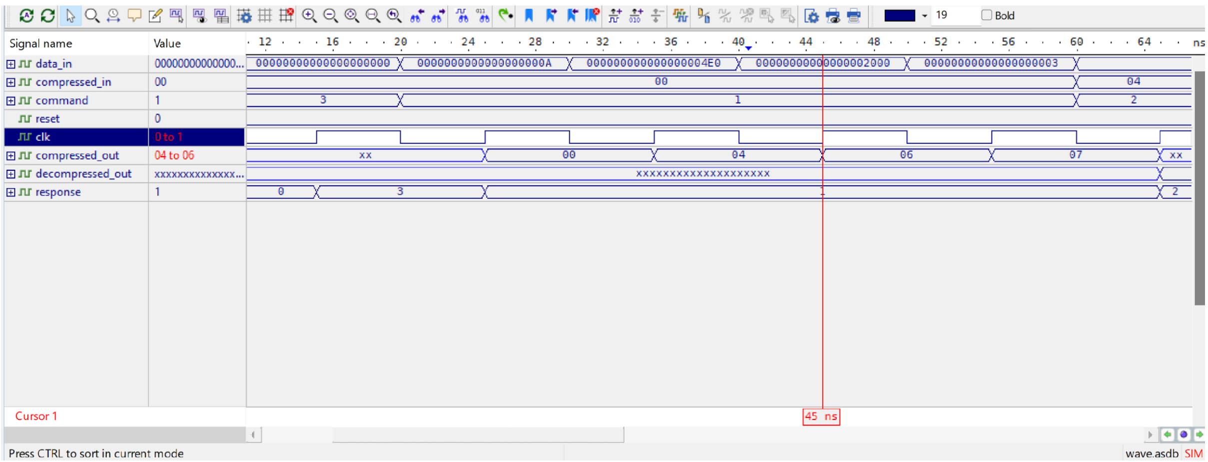

Figure 12. Reference Model CaseTest 5

6. When the command was 1 = 01, this means compression operation and the data_in was 0x03 which doesn't exist in the memory. The data was written at the current index register which points to the last empty slot in the memory

which is at index 0x07. So, compressed_out is 0x07 and the response as 1 = 01. After that the index register was incremented.

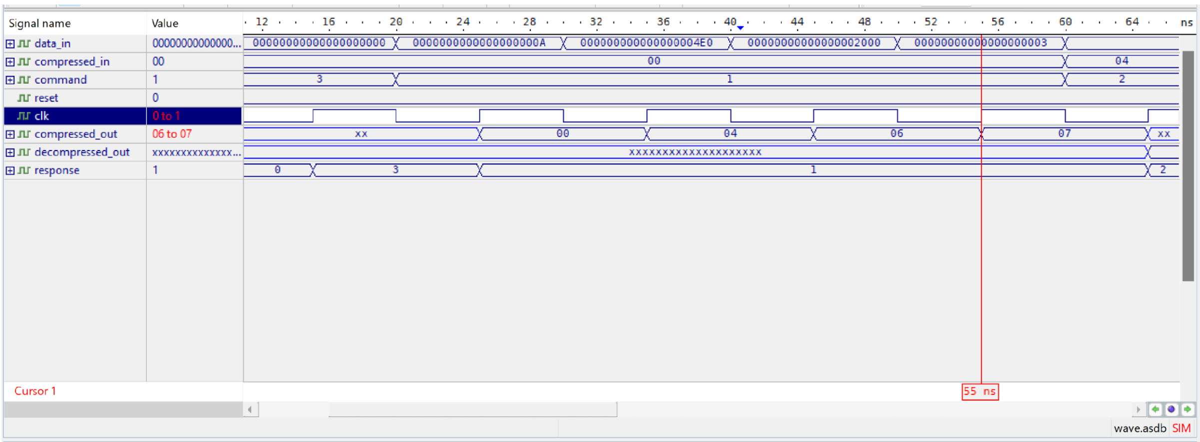

Figure 13. Reference Model CaseTest 6

7. When the command was 2 = 10, this means decompression operation and the compressed_in was 0x04 which exists in the memory and the data at that index is 0x04E0 so, decompressed_out is 0x04E0 and the response is 2 = 10 which indicates valid decompressed output.

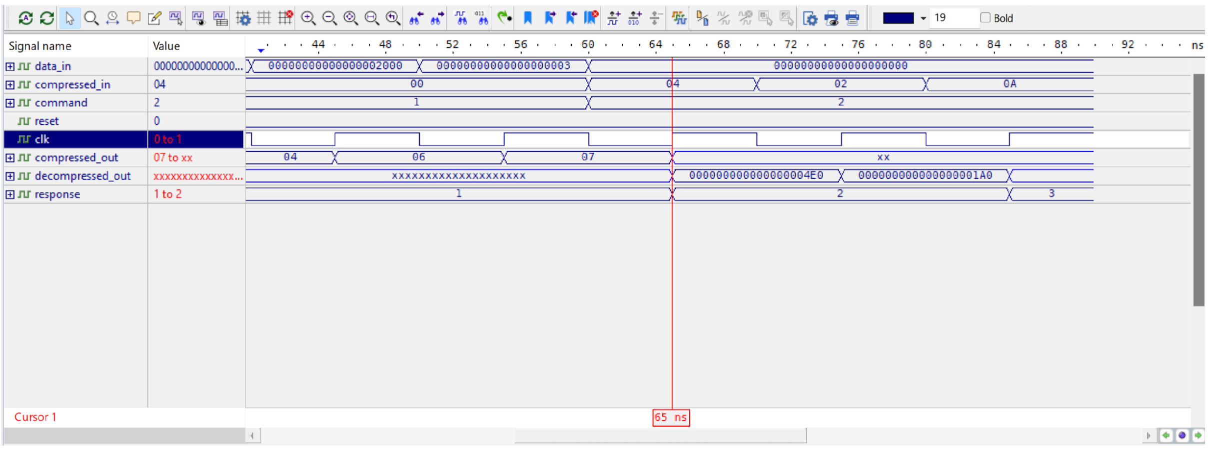

Figure 14. Reference Model CaseTest 7

8. When the command was 2 = 10, this means decompression operation and the compressed_in was 0x02 which exists in the memory and the data at that index is 0x01A0 so, decompressed_out is 0x01A0 and the response is 2 = 10 which indicates valid decompressed output.

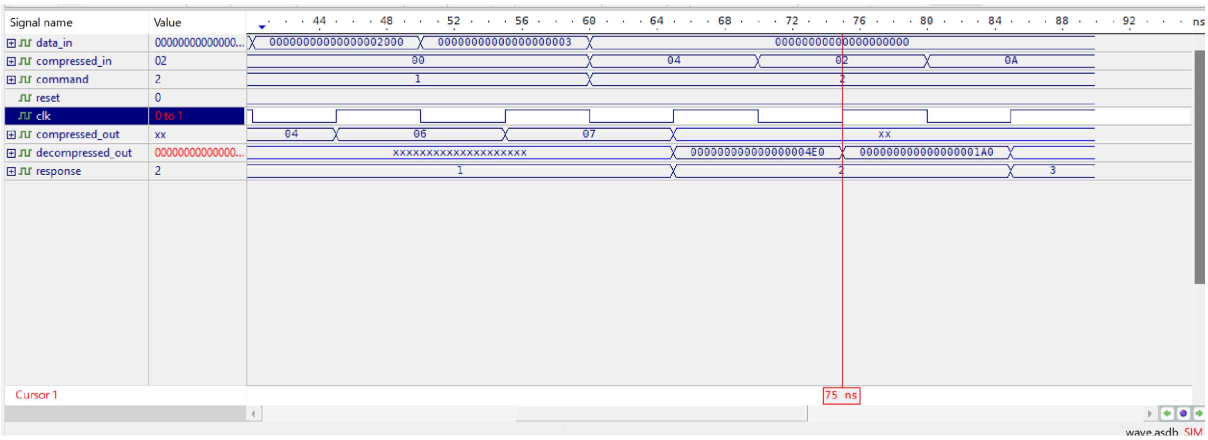

Figure 15. Reference Model CaseTest 8

9. When the command was 2 = 10, this means decompression operation and the compressed_in was 0x0A = 10 which doesn't exist in the memory, the response was 3 = 11 which indicates an error and invalid decompressed output.

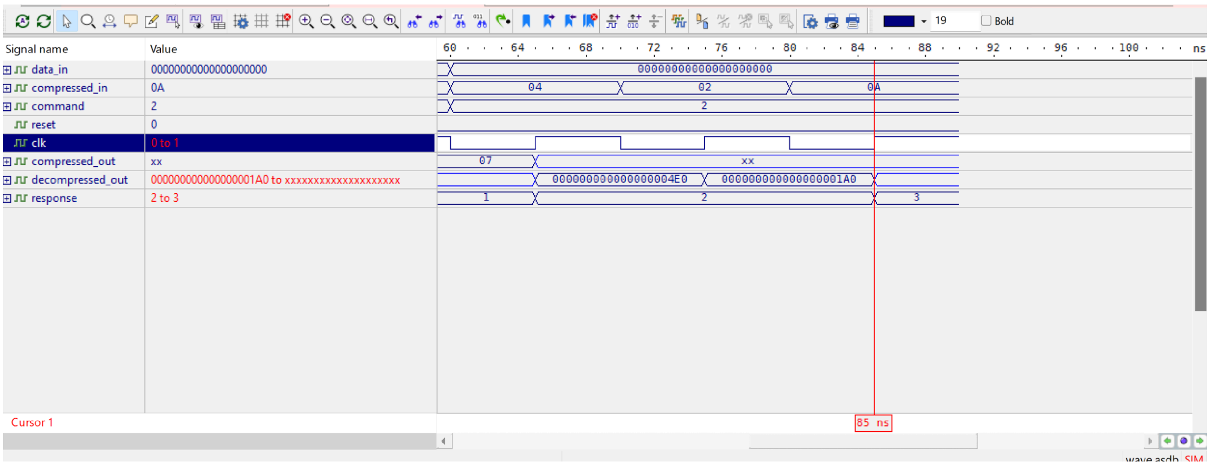

Figure 16. Reference Model CaseTest 9

3. Conclusion

In conclusion, the reference model for the hardware compression and decompression chip has been successfully implemented in both Python and SystemVerilog. The reference model accurately represents the expected behavior and functionality of the chip, providing a golden model for verification and testing. The compression algorithm efficiently reduces the size of data while preserving essential information, utilizing dictionary memory and index registers. The decompression algorithm functions as the inverse of the compression algorithm, retrieving the original data based on the compressed input. The reference model serves as a crucial component in the overall design verification project, ensuring the chip's performance and reliability.

4. References

[1] https://verificationacademy.com/forums/t/reference-model/34936

5. Appendix A

5.1 Python Code

```python
class CompressionDecompressionChip:
    def __init__(self):
        self.dictionary_memory = [None] * 256
        self.index_register = 0
        self.error = False

    def compress(self, data_in):
        if data_in in self.dictionary_memory:
            index = self.dictionary_memory.index(data_in)
            return index, 0b01
        else:
            if None in self.dictionary_memory:
                index = self.dictionary_memory.index(None)
                self.dictionary_memory[index] = data_in
                self.index_register += 1
                return index, 0b01
            else:
                self.error = True
                return None, 0b11

    def decompress(self, compressed_in):
        if compressed_in < self.index_register:
            data_out = self.dictionary_memory[compressed_in]
            return data_out, 0b10
        else:
            self.error = True
            return None, 0b11

    def reset(self):
        self.dictionary_memory = [None] * 256
        self.index_register = 0
        self.error = False

import ipywidgets as widgets
from IPython.display import display, clear_output
chip = CompressionDecompressionChip()

style = {'description_width': 'initial'}
input_layout = widgets.Layout(width='100%')
data_input = widgets.Text(
    description='Data:',
    value=''
)
compress_button = widgets.Button(description="Compress")
index_input = widgets.IntText(
    description='Index:',
```

```python
    value=0
)
decompress_button = widgets.Button(description="Decompress")
output = widgets.Output()
def on_compress_clicked(b):
    with output:
        clear_output()
        if data_input.value:
            compressed_index, status = chip.compress(data_input.value)
            if status == 0b01:
                print(f'Data: "{data_input.value}" | Compressed Index: 
{compressed_index}, Status: Success')
            else:
                print('Error: Memory Full')
        else:
            print('Please enter some data to compress.')

def on_decompress_clicked(b):
    with output:
        clear_output()
        data_out, status = chip.decompress(index_input.value)
        if status == 0b10:
            print(f'Index: {index_input.value} | Decompressed Data: "{data_out}", 
Status: Success')
        else:
            print('Error: Invalid Index')

compress_button.on_click(on_compress_clicked)
decompress_button.on_click(on_decompress_clicked)

input_widgets = widgets.VBox([data_input, compress_button, index_input, 
decompress_button])
display(input_widgets, output)
```

5.2 System Verilog Code

```systemverilog
/*---------------------------------------------------
        Chip Name  -> Compression and Decompression Chip
        File Name  -> referenceModel.sv
        Students   -> Doha Hmeid-1190120 & Raghad Afghani-1192423
        Description -> Reference model made for a Compression and Decompression Chip using system 
verilog
---------------------------------------------------- */

module chip(
        //chip inputs
  input logic [79:0] data_in,
  input logic [7:0] compressed_in,
  input logic [1:0] command,
```

```
    input logic reset,
    input logic clk,

    //chip outputs
    output logic [7:0] compressed_out,
    output logic [79:0] decompressed_out,
    output logic [1:0] response
);

//localparam are constants, can't be changed during instantiation
localparam DICT_SIZE = 256;      //The default size of the dictionary memory is 256 locations
localparam DICT_CELL_SIZE = 80;  //Each cell in memory has 80 bit width

//internal registers
logic [7:0] index_register = 8'b0;
logic [DICT_CELL_SIZE-1:0] dictionary [DICT_SIZE-1:0];             //the dictionary array stores compressed data


always @ (posedge clk or posedge reset)
begin

            //if the reset is on, then clear the dictionary and the index register
            if (reset == 1'b1) begin
      response <= 2'b00;   //no valid output
      index_register <= 8'b0;
              for (int i = 0; i < DICT_SIZE; i++) begin
         dictionary[i] <= 80'h0;
              end
              $writememh("memory.hex",dictionary);
              $display("reset done");
   end

            //otherwise, check the command signal and start processing
            else begin
              //save the last empty slot index in index_register
              for (int j = 0; j < DICT_SIZE; j++) begin
                                if (dictionary[j] == 80'b0) begin
                                            index_register = j;
                                            $display("save the last empty slot index, index_register =%h",
index_register);
                                            $display("j =%h", j);
                                            $display("dictionary[i] =%h", dictionary[j]);
                                            break;
                                end
              end

                        case(command)

                          //No operation
         2'b00: begin
                                            response <= 2'b00; //no vaild output
                                            compressed_out = 8'bx;
                                            decompressed_out = 80'bx;
         end

                          //Compression Algorithm
```

```
        2'b01: begin
                    $readmemh("memory.hex",dictionary);
                    $display("reading the memory.hex to dictionary");
                    response = 0;
                    $display("response %b",response);

                    //search for data_in
                    for (int i = 0; i < DICT_SIZE; i++) begin
                      if (dictionary[i] == data_in) begin  //compares data_in with the stored
data in the chip’s internal memory (dictionary memory)
                        compressed_out <= i;          //data_in found, write the index on
compressed_out port
                        response = 1;           //response = 01, means valid compressed_out
                              $display("found the data_in");
                            break;
                      end
                    end
                    $display("response %b",response);

                    //if dictionary is not full and data_in not found, write it on the last empty
slot in dictionary memory
                    if (index_register < DICT_SIZE) begin
                          if (response == 0) begin
                              dictionary[index_register] = data_in;
                              $writememh("memory.hex",dictionary);
                              $display("data_in not found - write in file");
                              compressed_out <= index_register;  //write the index
on compressed_out port
                              index_register++;             //increment the index
                              response <= 2'b01;              //response = 01, means
valid compressed_out
                          end
                    end
                    // Dictionary full
                    else begin
            response <= 2'b11; //response = 11, means there is error
                          compressed_out = 8'bx;
                          $display("full memory");
         end
                    decompressed_out = 80'bx;
                end

            //decompression Algorithm
                2'b10: begin
         if (compressed_in <= index_register) begin
                    $readmemh("memory.hex",dictionary);
              decompressed_out = dictionary[compressed_in];
                  response <= 2'b10; //response = 10, means valid compressed_out
             end
             else begin
              // Report error if compressed data is out of range
                    decompressed_out = 80'bx;
                  response <= 2'b11; //response = 11, means there is error
             end
                 compressed_out = 8'bx;
        end
```

```
                        //invalid command, report an error
        2'b11: begin
                                response <= 2'b11; //error
                                compressed_out = 8'bx;
                                decompressed_out = 80'bx;
        end

      endcase
          end
end
endmodule
```

6. Appendix B

```
/*-------------------------------------------------
        File Name   ->  testBench.sv
        Students    ->  Doha Hmeid-1190120 & Raghad Afghani-1192423
        Description ->  The testbench for the Compression and Decompression Chip
-------------------------------------------------- */
`timescale 1ns / 1ps
module chip_tb;

        //declaring the input and output signals
        logic [79:0] data_in;
        logic [7:0] compressed_in;
        logic [1:0] command;
        logic reset;
        logic clk;
        logic [7:0] compressed_out;
        logic [79:0] decompressed_out;
        logic [1:0] response;

  //DUT instance of the chip module, signals are connected to the DUT ports
  chip DUT (
    .data_in(data_in),
    .compressed_in(compressed_in),
    .command(command),
    .reset(reset),
    .clk(clk),
    .compressed_out(compressed_out),
    .decompressed_out(decompressed_out),
    .response(response)
  );

  //generate initial values
  initial begin
                data_in = 80'b0;
                compressed_in = 8'b0;
```

```
                        command = 2'b00;
            reset = 1'b0;
                        clk = 0;
    end

    //generate a new clock cycle every 5ns
    always #5 clk = ~clk;

    // Stimulus - the tests
    initial begin
      //test command=00: no operation
              command = 2'b00;
              #10;

              //test command=11: error
              command = 2'b11;
              #10;

              //test compression Algorithm
              $display("Compression test1");
              command = 2'b01;
              compressed_in = 8'b0;
      data_in = 80'h0000000000000000000A; //this data is in index 0 from memory.hex
      #10;

              $display("Compression test2");
      data_in = 80'h000000000000000004E0; //this data is in index 4 from memory.hex
      #10;

              $display("Compression test3");      //this data is not inside memory.hex - must be added at index 6
      data_in = 80'h00000000000000002000 ;
      #10;

              $display("Compression test4");      //this data is not inside memory.hex - must be added at index 7
      data_in = 80'h00000000000000000003 ;
      #10;

              //test decompression Algorithm
              $display("Decompression test1");
              command = 2'b10;
              data_in = 80'b0;
      compressed_in = 8'h04 ; //at index 4 from the memory.hex, we have 000000000000000004E0
      #10;

              $display("Decompression test2");
      compressed_in = 8'h02;            //at index 7 from the memory.hex, we have 00000000000000000003
(the new added value from last steps)
      #10;

              $display("Decompression test3");
      compressed_in = 8'h0A;            //at index 10 from the memory.hex, we don't have any value - must
return error
      #10;

              //test reset pin - must reset the memory
              /*reset = 1'b1;
```

```
            #10;
            reset = 1'b0;
            */
            //end simulation
            $display("end");
    $finish;
  end

  //Monitor - to check the results
  always @(posedge clk) begin
      $display("Time=%0t data_in=%h, compressed_in=%h, command=%b, reset=%b, compressed_out=%h,
decompressed_out=%h, response=%b",
                          $time, data_in, compressed_in, command, reset, compressed_out,
decompressed_out, response);
  end

endmodule
```