# DSD ProJect Report

# Team 14

| | | Member Name | MemberID | Tutoral# |
|---|---|---|---|---|
| | **1** | Moataz Mahmoud | 46-16760 | T69 |
| | **2** | Ayman Maged | 46-15946 | T69 |
| | **3** | Abdelrahman Hany Sabry | 46-12592 | T67 |
| | **4** | Abdelrahman Ehab | 46-6291 | T71 |
| **Compression** | **5** | Kareem Ashraf Okasha | 46-7098 | T65 |
| | **1** | Mohab khaira | 46-2393 | T70 |
| | **2** | Adham EL-Serougy | 46-0537 | T70 |
| | **3** | Mohamed Amr | 46-3561 | T66 |
| | **4** | Mohamed Ahmed Salaheldin | 46-20529 | T67 |
| **Decompression** | **5** | Karim Abdelsalam | 46-6631 | T68 |

# Overview

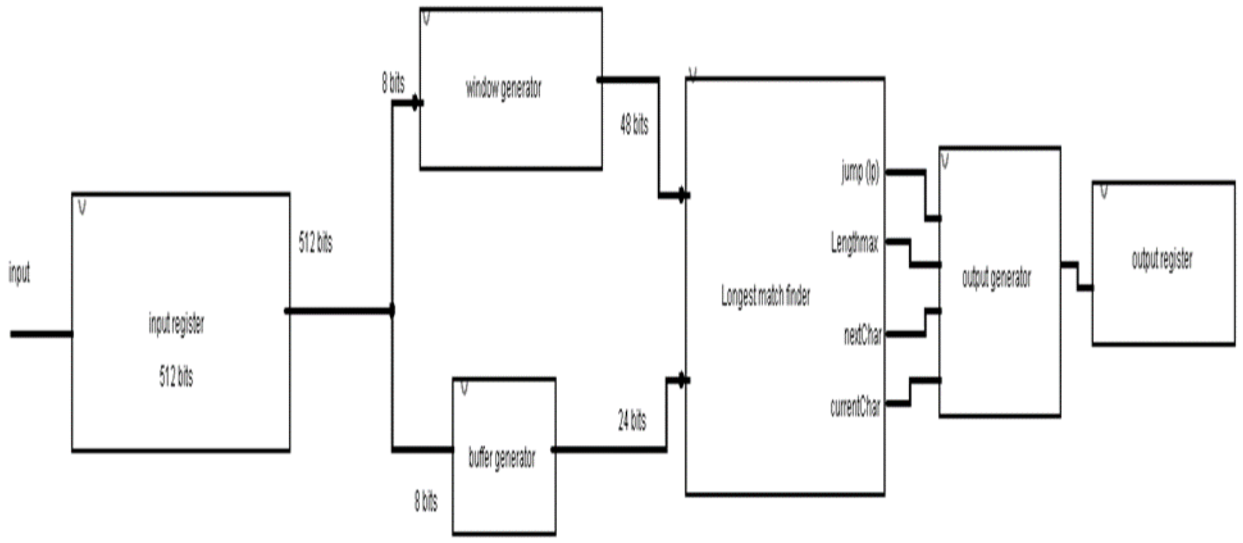
The project is compression using Limpel_Ziv77(LZ77) made in a 13bit format so that the outputted data is in the from of 13 bits divided as follows 3 bits for the Ip 2 bits for the Lmax and 8 bits for the byte that should enter the window.

To run the project simply open it with ISE design suite click on simulation then click on the testbench of the compression_chip and put the input that you want then run the test bench.then after running the test bench take the output from the console and put it in the test bench of the decompression and the original steam of bytes should be outputted

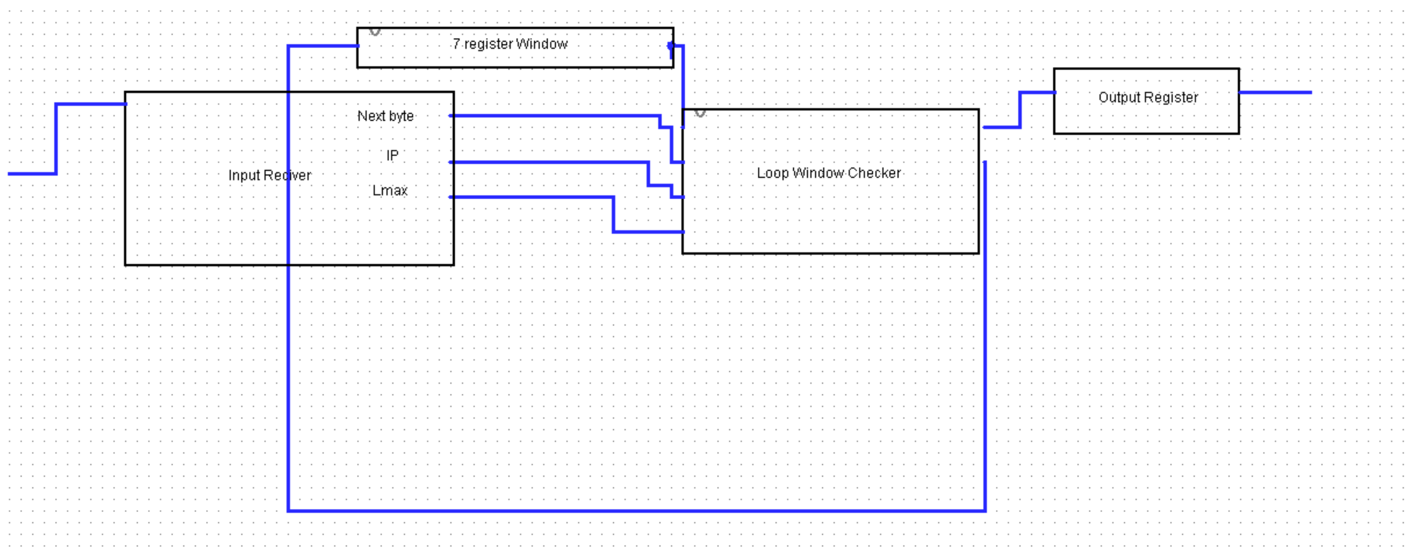
(All photos are found in better quality in the zip file)

# RTL

## Compression



## Decompression



# Design Methodology

## Compression: It’s a combinational circuit that compresses

The input taken is 64 bytes we compress repeated sequences into 13 bits each by loading it into a variable that iterates over the input byte by byte, 3 bits for the jump” location of the first character in the window that was found in the buffer”, 2 bits for the maximum length of the sequence , and 1 byte for the next character and then replacing the repeated sequence with the form (jump,Lmax,”nextcharacter”)

## Decompression:

The output of the compression works on the 13 bit system .By loading it into a variable then splitting it into its components we start looking at the IP then the Lmax while updating the window using a group of registers that shift by the byte then according to the IP and the Lmax we take from the window then finally we add the next character to the output and the window then repeat the loop till its done

## Resources Used:

https://stackoverflow.com/questions/26683335/vhdl-code-to-convert-5-bit-vector-to-integer

https://stackoverflow.com/questions/36953061/how-to-convert-an-integer-to-a-binary-representation-in-vhdl

https://stackoverflow.com/questions/6824838/vhdl-and-using-the-report-statement

https://www.intel.com

https://stackoverflow.com/questions/6824838/vhdl-and-using-the-report-statement

https://www.intel.com

https://stackoverflow.com/questions/42583061/vhdl-arrays-how-do-i-declare-an-array-of-unknown-size-and-use-it

https://www.ics.uci.edu/~jmoorkan/vhdlref/generics.html

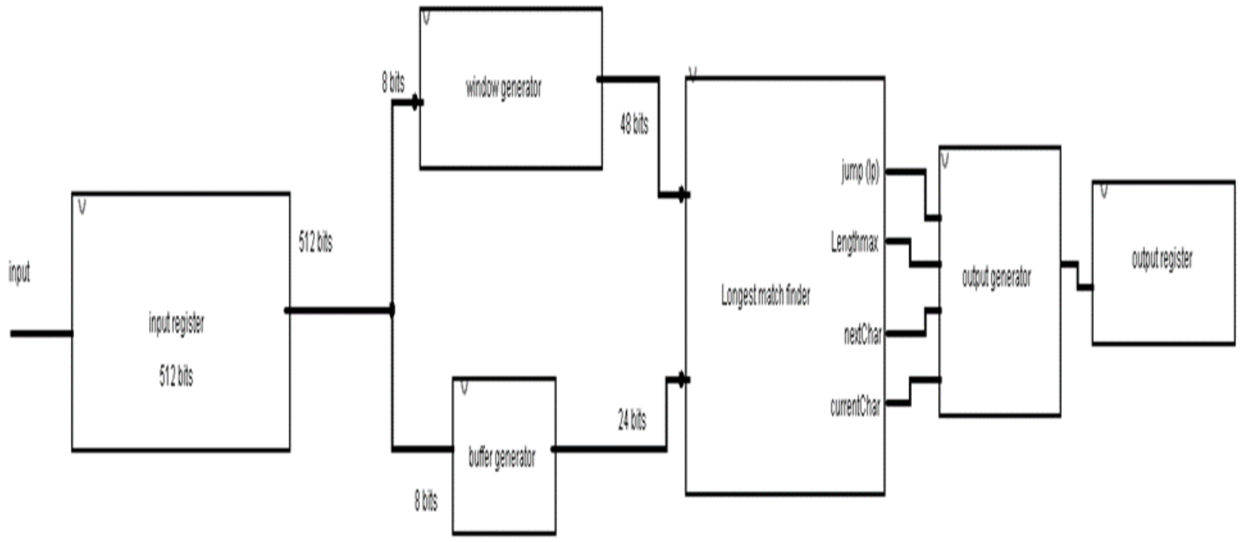
https://www.csee.umbc.edu/portal/help/VHDL/VHDL-Handbook.pdf

https://electronics.stackexchange.com/questions/347282/why-do-we-assign-our-outputs-to-signals-first-in-vhdl

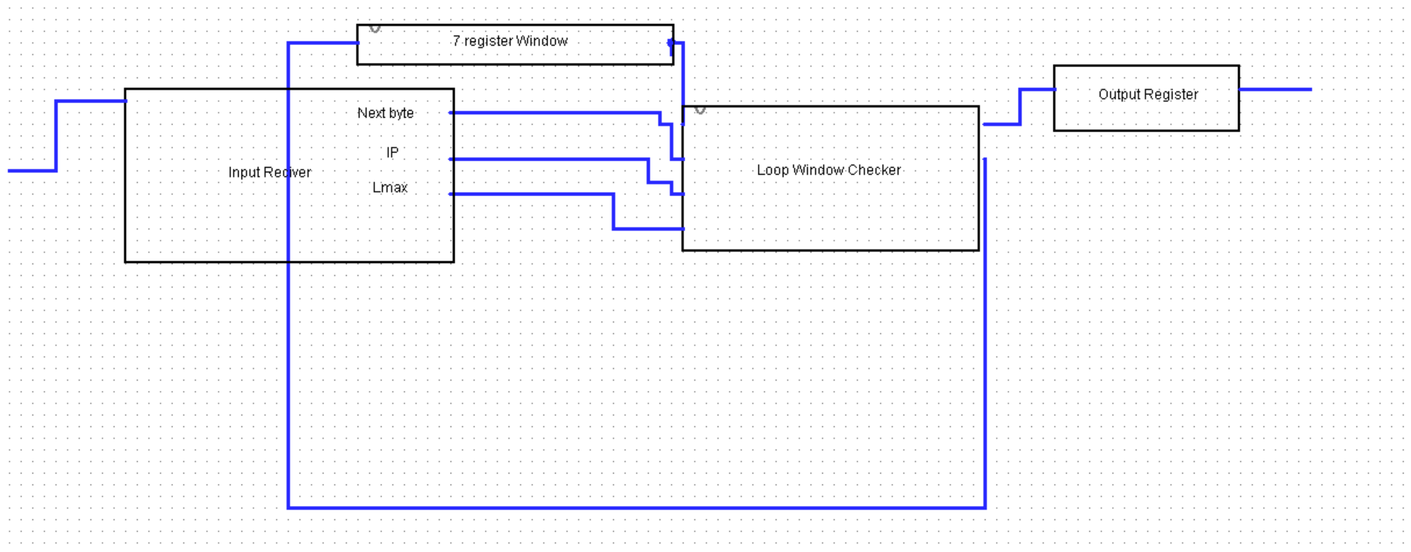
https://stackoverflow.com/questions/6824838/vhdl-and-using-the-report-statement

# Circuit Diagrams

## Compression



## Decompression



# Implementation

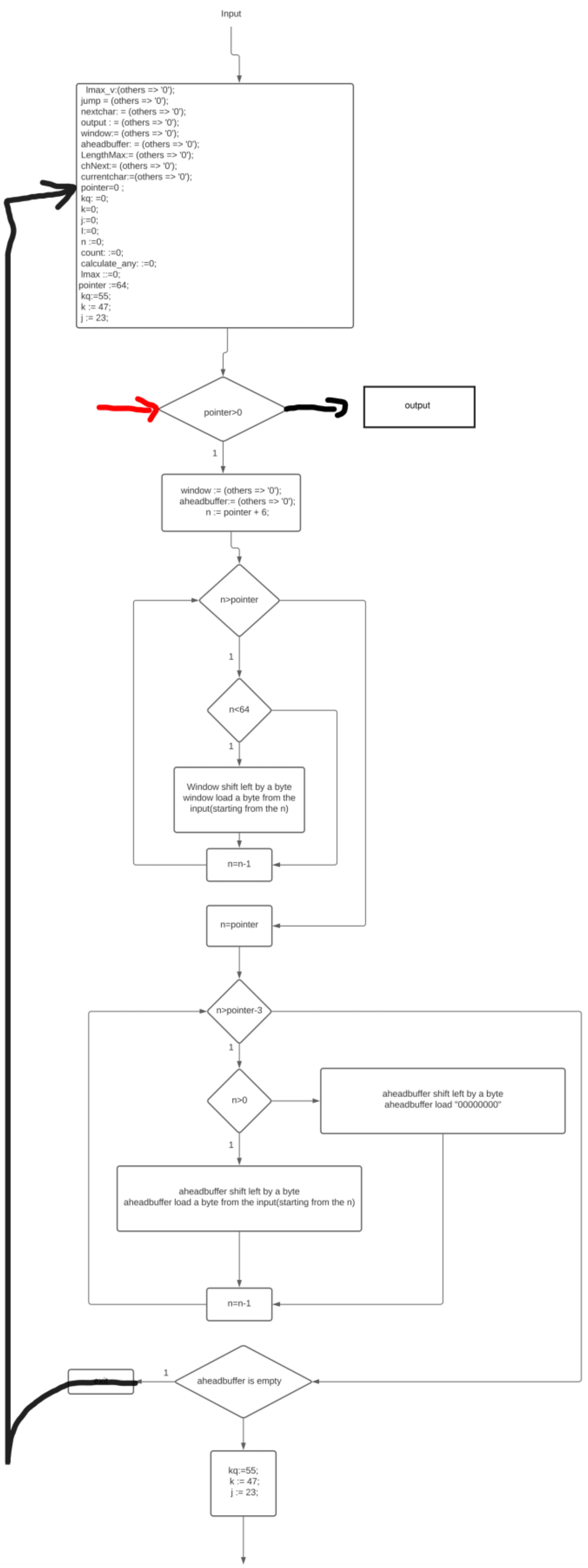
Compression: The input to the compression chip is taken as 512 bits , we implement 9 bytes that iterate over the input byte by byte , the 9 bytes are divided into Window”6 bytes” and the look ahead buffer”3 bytes” with a pointer that points towards the first element in the buffer, in each iteration of the pointer the window and the buffer moves to the right by 1 byte we check if there is a repeated character in both the buffer and the window If we found a repeated character we iterate the Lmax variable by 1 and move to the next character , if the Lmax variable is at least 2 we replace the characters from location of the window + Lmax by a compressed sequence of the form : jump , Lmax , next_character then we concatenate it into the output finally we reset the variables and increment the pointer till we reach the end of the input and then end the sequence with “00000000” in the output

## Decompression:

When out the Computational circuit receives the input. First the input is taken as the 512 bytes some compressed the others are zeroes then it loads them 13 bits by 13 bits into a variable called A then A is separated into Ip Lmax and Next byte. If Lmax and Ip are both equal to 0 then it adds the next byte to the out put and the window and goes to the next loop. Else it will check the IP then Lmax and decide what to do accordingly for example for IP = 3 and Lmax = 2 we go back 3 steps and take 2 bytes so we take input from REG3 and REG 2 and so on.

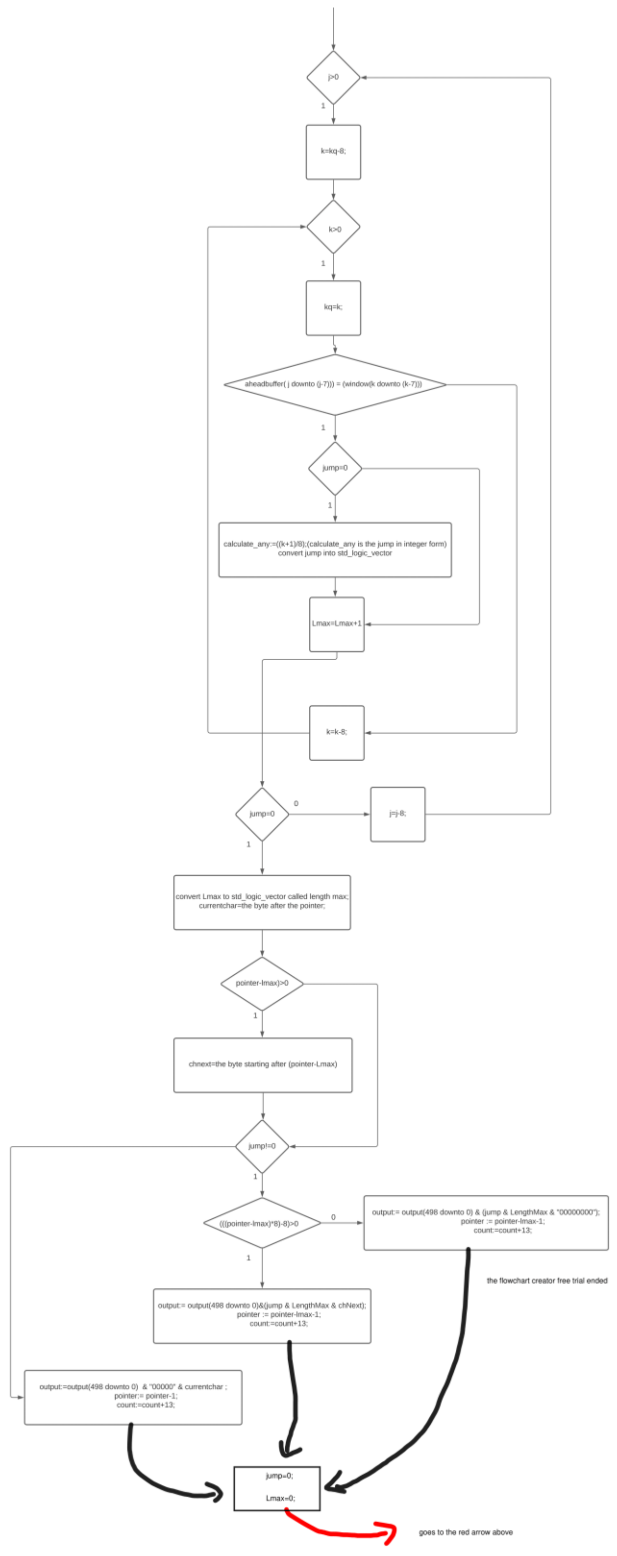
## Flow Charts

## Compression

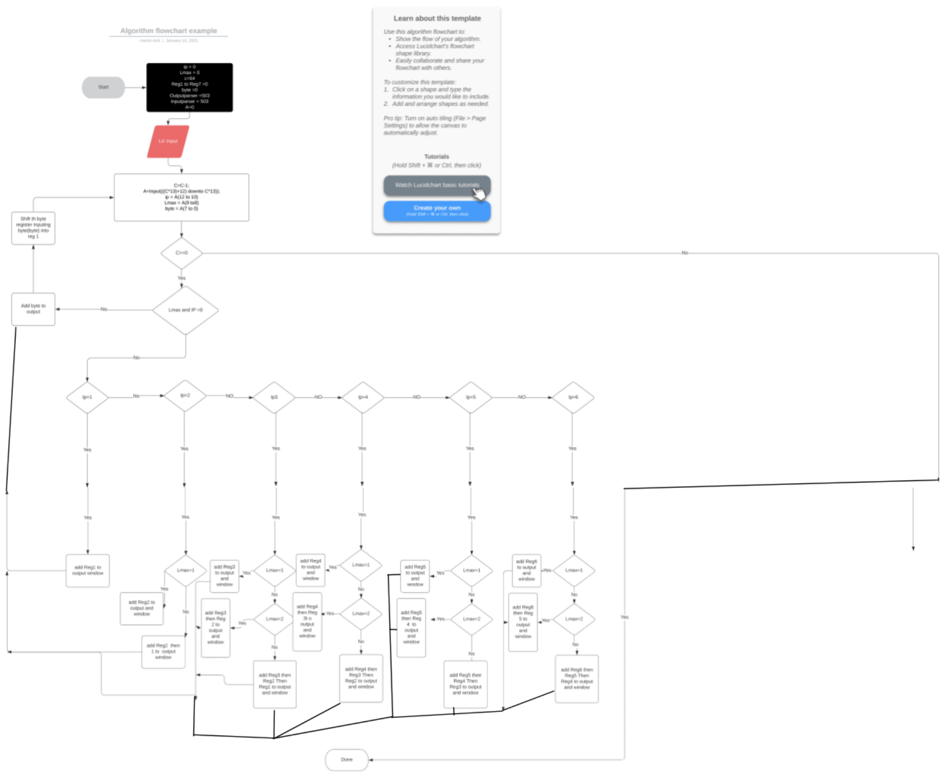


```
  lmax_v:(others => '0');
jump = (others => '0');
nextchar: = (others => '0');
output : = (others => '0');
window:= (others => '0');
aheadbuffer: = (others => '0');
LengthMax:= (others => '0');
chNext:= (others => '0');
currentchar:=(others => '0');
pointer=0 ;
kq: =0;
k=0;
j:=0;
l:=0;
n :=0;
count: :=0;
calculate_any: :=0;
lmax ::=0;
pointer :=64;
kq:=55;
k := 47;
j := 23;
```

Input
pointer>0
output
1
window := (others => '0');
aheadbuffer:= (others => '0');
n := pointer + 6;
n>pointer
1
n<64
1
Window shift left by a byte
window load a byte from the input(starting from the n)
n=n-1
n=pointer
n>pointer-3
1
n>0
aheadbuffer shift left by a byte
aheadbuffer load "00000000"
1
aheadbuffer shift left by a byte
aheadbuffer load a byte from the input(starting from the n)
n=n-1
exit
1
aheadbuffer is empty
kq:=55;
k := 47;
j := 23;


j>0
1
k=kq-8;
k>0
1
kq=k;
aheadbuffer( j downto (j-7))) = (window(k downto (k-7)))
1
jump=0
1
calculate_any:=((k+1)/8);(calculate_any is the jump in integer form)
convert jump into std_logic_vector
Lmax=Lmax+1
k=k-8;
jump=0
0
j=j-8;
1
convert Lmax to std_logic_vector called length max;
currentchar=the byte after the pointer;
pointer-lmax)>0
1
chnext=the byte starting after (pointer-Lmax)
jump!=0
1
(((pointer-lmax)*8)-8)>0
0
output:= output(498 downto 0) & (jump & LengthMax & "00000000");
pointer := pointer-lmax-1;
count:=count+13;
1
the flowchart creator free trial ended
output:= output(498 downto 0)&(jump & LengthMax & chNext);
pointer := pointer-lmax-1;
count:=count+13;
output:=output(498 downto 0) & "00000" & currentchar ;
pointer:= pointer-1;
count:=count+13;
jump=0;
Lmax=0;
goes to the red arrow above

# Decompression


Algorithm flowchart example
Start
ip = 0
Lmax = 0
c=64
Reg1 to Reg7 =0
byte =0
Outputparser =503
Inputparser = 503
A=0
Ld :Input
C=C-1;
A=Input(((C*13)+12) downto C*13));
ip = A(12 to 10)
Lmax = A(9 to8)
byte = A(7 to 0)
Shift th byte register inputing byte(byte) into reg 1
C>=0
Yes
No
Lmax and IP =0
Add byte to output
Ip=1
Ip=2
Ip3
Ip=4
Ip=5
Ip=6
NO
add Reg1 to output window
Lmax=1
Lmax=2
add Reg2 to output and window
add Reg2 then 1 to output window
add Reg3 to output and window
add Reg3 then Reg 2 to output and window
add Reg3 then Reg2 Then Reg1 to output and window
add Reg4 to output and window
add Reg4 then Reg 3t o output and window
add Reg4 then Reg3 Then Reg2 to output and window
add Reg5 to output and window
add Reg5 then Reg 4 to output and window
add Reg5 then Reg4 Then Reg3 to output and window
add Reg6 to output and window
add Reg6 then Reg 5 to output and window
add Reg6 then Reg5 Then Reg4 to output and window
Done
Learn about this template
Tutorials
(Hold Shift + ⌘ or Ctrl, then click)
Watch Lucidchart basic tutorials
Create your own

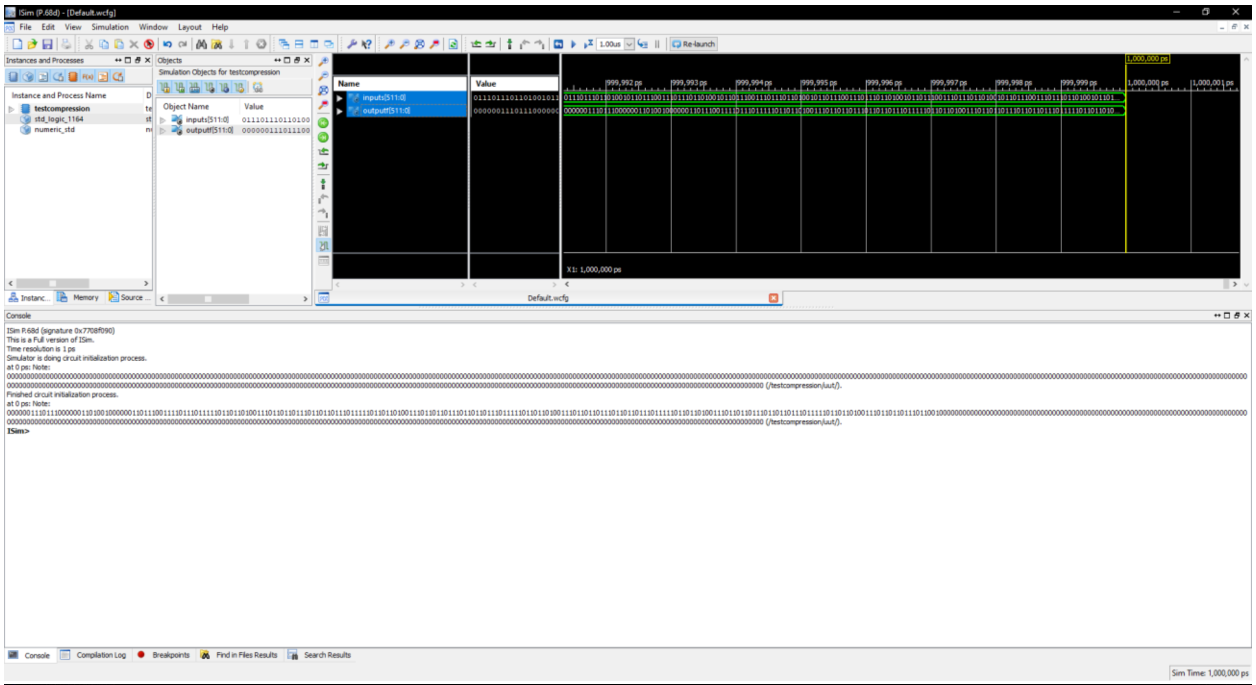
Use this algorithm flowchart to:
- Show the flow of your algorithm.
- Access Lucidchart's flowchart shape library.
- Easily collaborate and share your flowchart with others.

To customize this template:
1. Click on a shape and type the information you would like to include.
2. Add and arrange shapes as needed.

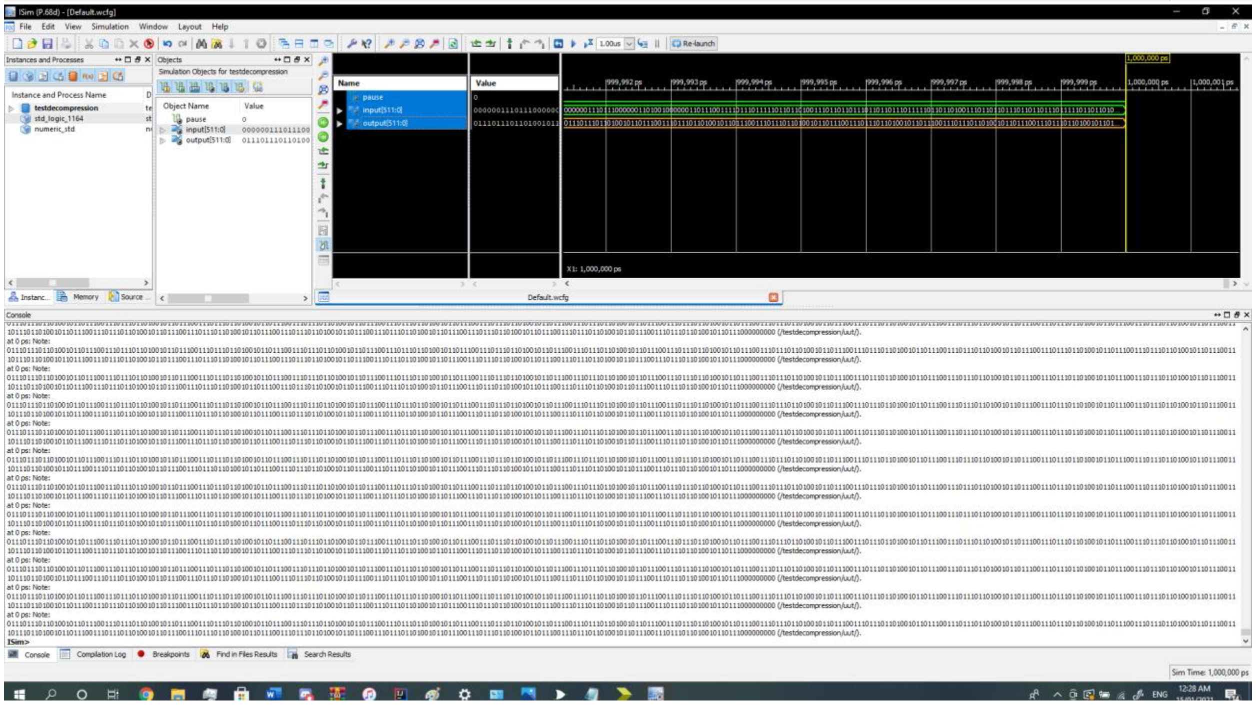
Pro tip: Turn on auto tiling (File > Page Settings) to allow the canvas to automatically adjust.

## Results and Limitation

The results were promising the compression ratio was dependent on the input but for the inputs we tried the compression ratio was between 0.3 and 0.5.As for the limitations there were only two limitations we found when running which were if the input contains all unique bits then it will not be compressed properly  and for some very specific outputs some bytes might get flipped with the byte next to them in the decompression in the next page you will find the compression and decompression test benches



```
ISim P.68d (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
at 0 ps: Note:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 (/testcompression/uut/).
Finished circuit initialization process.
at 0 ps: Note:
00000011101110000001101001000000110111001111011101111101101101001110110110111011011011101111101101101001110110110111011011011101111101101101001110110110111011011011101111101101101001110110110111011011011101111101101101001110110110111011011011101111101101101001110110110111011001000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 (/testcompression/uut/).
ISim>
```



```
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
at 0 ps: Note:
0111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011
1011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100011101110110100101101110001110111011010010110111000111011101101001011011100000000000000 (/testdecompression/uut/).
ISim>
```

# Code Listing

## Compression Code

```
----------------------------------------------------------------------------------
-- Module Name:    Compression_Chip - Behavioral
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Compression_Chip is
```

```
 port (  inputs:in std_logic_vector(511 downto 0);
                         outputf:out std_logic_vector(511 downto 0));



end Compression_Chip;

architecture Behavioral of Compression_Chip is

 --signal inputs :std_logic_vector(511 downto 0):= (others => '0');

function to_bstring(sl : std_logic) return string is-- function to print an
std logic vector for tracing
  variable sl_str_v : string(1 to 3);  -- std_logic image with quotes around
begin
  sl_str_v := std_logic'image(sl);
  return "" & sl_str_v(2);  -- "" & character to get string
end function;

function to_bstring(slv : std_logic_vector) return string is
  alias    slv_norm : std_logic_vector(1 to slv'length) is slv;
  variable sl_str_v : string(1 to 1);  -- String of std_logic
  variable res_v    : string(1 to slv'length);
```

```
begin
  for idx in slv_norm'range loop
    sl_str_v := to_bstring(slv_norm(idx));
    res_v(idx) := sl_str_v(1);
  end loop;
  return res_v;
end function;



begin
 process (inputs)
 variable lmax_v:std_logic_vector(1 downto 0):= (others => '0');
 variable jump :std_logic_vector(2 downto 0):= (others => '0');
 variable nextchar: std_logic_vector(7 downto 0):= (others => '0');
 variable output : std_logic_vector(511 downto 0):= (others => '0');
 variable window: std_logic_vector(47 downto 0):= (others => '0');-- also
called Dictionary
 variable aheadbuffer: std_logic_vector(23 downto 0):= (others => '0');
 variable LengthMax:std_logic_vector(1 downto 0):= (others => '0');
 variable chNext:std_logic_vector(7 downto 0):= (others => '0');
 variable currentchar:std_logic_vector(7 downto 0):=(others => '0');
 Variable pointer :Integer:=0 ;
 Variable kq: integer:=0;

 variable k:integer:=0;

 variable j:integer:=0;

 variable l:integer:=0;

 variable n : integer;

 variable count: integer:=0;

 variable calculate_any:integer:=0;

 variable lmax :Integer:=0;-- buffer length we can skip 3 characters with
2 bits representing number 3--

 variable input:std_logic_vector(511 downto 0):=(others => '0');

-- each report statement included is to trace the values of the variables

-- to ensure that they work correctly.

 begin

 pointer :=64;

 kq:=55;

 k := 47;

 j := 23;

 l := pointer;

 input := inputs;


 while ( pointer >0)loop-- pointer that loops over the input byte by byte
```

```
        window := (others => '0');

        aheadbuffer:= (others => '0');

        n := pointer + 6; -- value to loop over

        while (n > pointer) loop -- to fill the window

                if (n <= 64) then

--              report to_bstring(input(((n*8)-1) downto ((n*8) - 8)));

                        window := (window((47 - 8) downto 0) & (
input(((n*8)-1) downto ((n*8) - 8)))); -- dictionary.add("" +
input.charAt(i));

--                      dictionary_Size := dictionary_Size + 1;

                end if;


                n := n-1;

        end loop;


        -- Second Loop to fill the look ahead buffer

        n := pointer;

        while (n > pointer -3 ) loop -- for(int i = pointer; i  < pointer + 5 &&
i < input.length(); i++)

              if(n>0) then
```

```
                        aheadbuffer := aheadbuffer((23 - 8) downto 0) &
input(((n*8)-1) downto ((n*8)-8)); --
lookAheadBuffer.add(""+input.charAt(i));


                 else

                        aheadbuffer := aheadbuffer((23 - 8) downto 0) &
"00000000";

--                      lookAheadBuffer_Size := lookAheadBuffer_Size + 1;

      end if;

               n := n-1;

               end loop;

               if((aheadbuffer(23 downto
0))="000000000000000000000000")then exit ;end if;

       --      report "  buffer is  " & to_bstring(aheadbuffer(23 downto
0));



 kq:=55;

 k := 47;

 j := 23;

--      report "j " & integer 'image(j);

--   report "k " & integer 'image(k);

-- nested loops to determine the longest match between the buffer and
the window
```

```
        while( j >0) loop-- iteration over the buffer

                k:=kq-8;-- to save the position of the iterator of the window
to start from the next position of the previous loop

                while( k > 0 )loop-- iteration over the window AKA dictionary

                 kq:=k;

                -- report to_bstring(aheadbuffer);

                        if((aheadbuffer( j downto (j-7))) = (window(k downto
(k-7))))then

                        -- report "here first IF";

                                if(jump = "000")then

                        --      report " k is " & integer 'image(k);

                                        calculate_any:=((k+1)/8);-- determine the
value of the jump in integer form

                                --      report "here 2nd IF";

                                --      report " the jump is  " & integer
'image(calculate_any);

                                        jump :=
std_logic_vector(to_unsigned(calculate_any, 3));-- function that
convert integer into an std logic vector


                                end if;

                                lmax := lmax+1;-- increment the maximum
sequence found length by 1 each matched byte
```

```
                    exit;-- because we found a match thus we move to the
next character in the buffer
                end if;
                    k := k-8; -- decrement the window iterator by 8 bits "
byte = character " each iteration
                end loop;
        --report"second loop";
                if(jump="000") then
                    --report "here 3rd IF";
                    exit;-- the first character in the buffer has no match in
the window so we exit
                    end if;



                j:= j-8 ; -- decrement the buffer iterator by 8 bits " byte =
character" each iteration
        end loop;


        LengthMax := std_logic_vector(to_unsigned(lmax, 2));-- from
integer into std logic vector
        currentchar:=input((((pointer)*8)-1) downto (((pointer)*8)-8));--
we get the current character in case it was not found in the window
        if((pointer-lmax)>0)then
```

```
                chNext := input((((pointer-lmax)*8)-1) downto (((pointer-
lmax)*8)-8));      -- next character to be saved in the compression of the
sequence ( jump , max length ,nextcharacter )

        end if;

        if(not (jump ="000") )then

--      report integer 'image(pointer);


                        if((((pointer-lmax)*8)-8)>0)then -- concatinate the
compressed form into the output

                                output:= output(498 downto 0)&(jump &
LengthMax & chNext);

                                pointer := pointer-lmax-1; -- start from the
position after the compression of the current sequence

                                count:=count+13;-- each time we compress it
takes 13 bits thus we increment the count of bits by 13 each operation

                                --report " First if"&to_bstring(output);


                        else

                                output:= output(498 downto 0) & (jump &
LengthMax & "00000000");-- in case there is no next character
concatinate 8 bits of zeros marking the end of the compression

                                pointer := pointer-lmax-1;

                                count:=count+13;

                                --report " First Else"&to_bstring(output);
                        end if;

        else

                        output:=output(498 downto 0)  & "00000" &
currentchar ; -- when we encounter a character that is not found in the
window the jump and lmax are set to "00000"

                        pointer:= pointer-1;

                        count:=count+13;




        end if;




        jump :="000";

        lmax:=0;


end loop;

output := output( count-1 downto 0) & output( 511 downto count);--
reverse the direction of the compression to put the compressed bits in
the left side " most significant bits"

--report "Count" & integer'image(count);

--report to_bstring(output);

outputf <= output;
```

```
end process;

end Behavioral;
```

## Compression Test Bench:

```
--------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date:   06:42:20 01/14/2021

-- Design Name:

-- Module Name:   C:/Users/Mohamed/FinalProject/Testcompression.vhd

-- Project Name:  FinalProject

-- Target Device:

-- Tool versions:

-- Description:

--

-- VHDL Test Bench Created by ISE for module: Compression_Chip

--

-- Dependencies:
```

```
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY Testcompression IS
END Testcompression;

ARCHITECTURE behavior OF Testcompression IS
```

```
 -- Component Declaration for the Unit Under Test (UUT)

 COMPONENT Compression_Chip
 PORT(
      inputs : IN  std_logic_vector(511 downto 0);
      outputf : OUT  std_logic_vector(511 downto 0)
     );
 END COMPONENT;



--Inputs
signal inputs : std_logic_vector(511 downto 0) := (others => '0');

        --Outputs
signal outputf : std_logic_vector(511 downto 0);
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name




BEGIN

        -- Instantiate the Unit Under Test (UUT)
uut: Compression_Chip PORT MAP (
       inputs => inputs,
       outputf => outputf
     );

-- Clock process definitions




-- Stimulus process
stim_proc: process
begin
        --please input the bits you want to compress Here
        inputs
<="0111011101101001011011100111011101101001011011100111011101100
1011011100111011101101001011011100111011101101001011011100111011
1
0110100101101110011101110110100101101110011101110110100101101110
0
11101110110100101101110011101110110100101101110011101110110100101
10111001110111011010010110111001110111011010010110111001110111011
01001011011100111011101101001011011100111011101101001011011100111
01110110100101101110011101110110100101101110011101110110100101101
11001110111011010010110111001110111011010010110111001110111";
      wait;
end process;

END;
```

# Decompression :

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    19:29:06 01/13/2021
-- Design Name:
-- Module Name:    DecompressionLoop - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DecompressionLoop is
port(pause: in std_logic;
Input : in STD_LOGIC_VECTOR(511 downto 0);
Output : out STD_LOGIC_VECTOR(511 downto 0)
);

end DecompressionLoop;

architecture Behavioral of DecompressionLoop is
function to_bstring(sl : std_logic) return string is
  variable sl_str_v : string(1 to 3);  -- std_logic image with quotes around
begin
  sl_str_v := std_logic'image(sl);
  return "" & sl_str_v(2);  -- "" & character to get string
end function;
```

```
function to_bstring(slv : std_logic_vector) return string is
  alias    slv_norm : std_logic_vector(1 to slv'length) is slv;
  variable sl_str_v : string(1 to 1);  -- String of std_logic
  variable res_v    : string(1 to slv'length);
begin
  for idx in slv_norm'range loop
    sl_str_v := to_bstring(slv_norm(idx));
    res_v(idx) := sl_str_v(1);
  end loop;
  return res_v;
end function;


procedure shiftbyts(variable Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,byte : inout
std_logic_vector(7 downto 0))   is


begin
   Reg7:= Reg6;
            Reg6:= Reg5;
            Reg5:= Reg4;
            Reg4:= Reg3;
            Reg3:= Reg2;
            Reg2:= Reg1;
            Reg1:= byte;
                        report "reg1 " &integer'image(to_integer((unsigned(Reg1))));
            report "reg2 " &integer'image(to_integer((unsigned(Reg2))));
report "reg3 " &integer'image(to_integer((unsigned(Reg3))));
```

```
        report "reg4 " &integer'image(to_integer((unsigned(Reg4))));

        report "reg5 " &integer'image(to_integer((unsigned(Reg5))));

        report "reg6 " &integer'image(to_integer((unsigned(Reg6))));

        report "reg7 " &integer'image(to_integer((unsigned(Reg7))));

        report "    ";


end procedure ;




begin





process(Input,pause)

variable A :  STD_LOGIC_VECTOR(12 downto 0);

variable byte : STD_LOGIC_VECTOR(7 downto 0);

variable temp : std_logic_vector(7 downto 0);

variable temp2 : std_logic_vector(7 downto 0);

variable temp3 : std_logic_vector(7 downto 0);

variable Lmax :  STD_LOGIC_VECTOR(1 downto 0);

variable Ip : STD_LOGIC_VECTOR (2 downto 0);

variable Reg1 : std_logic_vector (7 downto 0):= (others => '0');

variable Reg2,Reg3,Reg4,Reg5,Reg6,Reg7 : std_logic_vector (7 downto 0) := (others => '0');

variable Outputf :STD_LOGIC_VECTOR(511 downto 0):= (others => '0');

variable Inputf :STD_LOGIC_VECTOR( 831 downto 0):= (others => '0');
variable Inputparser : integer range 0 to 550;

variable Outputparser : integer range 0 to 511;

variable K : integer range 0 to 512;

variable L : integer range 0 to 512;


begin

--recives input  into inputf    as variable

Inputf (831 downto 320) := Input (511 downto 0);

if (pause = '0' ) then

Inputparser :=63;

Outputparser :=504;

-- loop until you reach 64 bits or until the output is full

for i in 0 to 63 loop

  if (Outputparser >0) then

  --input parser fills the A 13 bits by 13 bits till it is finished

   A:= Inputf(((Inputparser *13)+12) downto (Inputparser *13));

   report "A" &integer'image(to_integer(unsigned(A)));

   byte :=A(7 downto 0);

             Ip:=A(12 downto 10);

             Lmax:=A(9 downto 8);


             -- if the Ip abd lmax =0 sjip the condition and add the byte directly to the output and the
window

             --else according to the ip and lmax fill the output and shift the window

             --input and output parsers start from 511 till zero moving the correct amount of ints
every time

   if (not(Ip ="000" and Lmax = "00")) then
```

```
                if (Ip = "001")then -- I.e the Ip is 1 so we take the value from reg1 and add it to the
window and output

              if (Outputparser <0) then

              Outputparser:= 0;

              end if;

    Output(Outputparser+7 downto Outputparser) <=Reg1;

              Outputf(Outputparser+7 downto Outputparser) :=Reg1;

              Outputparser:=Outputparser -8;

              temp:=Reg1;

              shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);--is a procedure that shifts a
byte into the registers and throws the last byte

              report "1";


     elsif(Ip = "010")then

     report "2";

        if (Lmax ="01")then

if (Outputparser <0) then

              Outputparser:= 0;

              end if;

                   Output(Outputparser+7 downto Outputparser) <=Reg2;

                                  Outputf(Outputparser+7 downto Outputparser) :=Reg2;


                   Outputparser:=Outputparser -8;

           temp:=Reg2;

                   shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

        elsif(Lmax ="10")then

if (Outputparser <0) then

              Outputparser:= 0;
```

```
            end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg2;
                       Outputf(Outputparser+7 downto Outputparser) :=Reg2;

                Outputparser:=Outputparser -8;
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
            Output(Outputparser+7 downto Outputparser) <=Reg1;
            Outputf(Outputparser+7 downto Outputparser) :=Reg1;

            Outputparser:=Outputparser -8;
                   temp:=Reg2;
            temp2:=Reg1;
            shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

            shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);
        end if ;
    elsif(Ip = "011")then
             report "3";
        if (Lmax ="01")then
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg3;
                                       Outputf(Outputparser+7 downto Outputparser) :=Reg3;
                                              temp:=Reg3;
```

```
                                    shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);
                     Outputparser:=Outputparser -8;
                                    report "1LL";
        elsif(Lmax ="10")then
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg3;
                                Outputf(Outputparser+7 downto Outputparser) :=Reg3;

                    Outputparser:=Outputparser -8;
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                    Output(Outputparser+7 downto Outputparser) <=Reg2;
                                Outputf(Outputparser+7 downto Outputparser) :=Reg2;

                    Outputparser:=Outputparser -8;
                                temp:=Reg3;
                                temp2:=Reg2;
                                shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                                shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);

                                report "2LL";
        elsif (Lmax ="11")then
if (Outputparser <0) then
```

```
            Outputparser:= 0;
            end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg3;
                                    Outputf(Outputparser+7 downto Outputparser) :=Reg3;


                  Outputparser:=Outputparser -8;
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                  Output(Outputparser+7 downto Outputparser) <=Reg2;
                                    Outputf(Outputparser+7 downto Outputparser) :=Reg2;


                  Outputparser:=Outputparser -8;
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
             Output(Outputparser+7 downto Outputparser) <=Reg1;
                                    Outputf(Outputparser+7 downto Outputparser) :=Reg1;


                  Outputparser:=Outputparser -8;
                                    temp:=Reg3;
                                    temp2:=Reg2;
                                    temp3:=Reg1;
                                    shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                                    shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);
```

```
                                shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp3);
                               report "3LL";
         end if ;


       elsif(Ip = "100")then
              report "4";
         if (Lmax ="01")then
 if (Outputparser <0) then
             Outputparser:= 0;
             end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg4;
                        Outputf(Outputparser+7 downto Outputparser) :=Reg4;


                Outputparser:=Outputparser -8;
                        temp:=Reg4;
                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);
         elsif(Lmax ="10")then
 if (Outputparser <0) then
             Outputparser:= 0;
             end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg4;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg4;


                   Outputparser:=Outputparser -8;
 if (Outputparser <0) then
             Outputparser:= 0;
             end if;
```

```
                                        Output(Outputparser+7 downto Outputparser) <=Reg3;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg3;


                        Outputparser:=Outputparser -8;
                                        temp:=Reg4;
                                        temp2:=Reg3;
                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);


                                         shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);
        elsif (Lmax ="11")then
    if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg4;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg4;


                        Outputparser:=Outputparser -8;
    if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg3;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg3;


                        Outputparser:=Outputparser -8;
    if (Outputparser <0) then
            Outputparser:= 0;
            end if;
```

```
                                        Output(Outputparser+7 downto Outputparser) <=Reg2;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg2;

                        Outputparser:=Outputparser -8;
                                        temp:=Reg4;
                                        temp2:=Reg3;
                                        temp3:=Reg2;
                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);

                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp3);
        end if ;

      elsif(Ip = "101")then
             report "5";
        if (Lmax ="01")then
    if (Outputparser <0) then
             Outputparser:= 0;
             end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg5;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg5;

                        Outputparser:=Outputparser -8;
                                        temp:=Reg5;
                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);
        elsif(Lmax ="10")then
```

```
if (Outputparser <0) then
              Outputparser:= 0;
              end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg5;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg5;



                        Outputparser:=Outputparser -8;
if (Outputparser <0) then
              Outputparser:= 0;
              end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg4;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg4;


                        Outputparser:=Outputparser -8;
                                        temp:=Reg5;
                                                   temp2:=Reg4;
                                         shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);


                                                   shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);
        elsif (Lmax ="11")then
if (Outputparser <0) then
              Outputparser:= 0;
              end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg5;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg5;
```

```
                    Outputparser:=Outputparser -8;
if (Outputparser <0) then
          Outputparser:= 0;
          end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg4;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg4;



                    Outputparser:=Outputparser -8;
if (Outputparser <0) then
          Outputparser:= 0;
          end if;
                                        Output(Outputparser+7 downto Outputparser) <=Reg3;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg3;


                    Outputparser:=Outputparser -8;
                                        temp:=Reg5;
                                        temp2:=Reg4;
                                        temp3:=Reg3;
                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);


                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);


                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp3);
      end if ;


    elsif(Ip = "110")then
```

```
            report "6";
            report "Outputparser" &integer'image(Outputparser);
        if (Lmax ="01")then
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg6;
                                    Outputf(Outputparser+7 downto Outputparser) :=Reg6;
                                    temp:=Reg6;
                                    shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                  Outputparser:=Outputparser -8;


        elsif(Lmax ="10")then
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg6;
                                    Outputf(Outputparser+7 downto Outputparser) :=Reg6;


                  Outputparser:=Outputparser -8;
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                                    Output(Outputparser+7 downto Outputparser) <=Reg5;
                                    Outputf(Outputparser+7 downto Outputparser) :=Reg5;
                  Outputparser:=Outputparser -8;
                              temp:=Reg6;
                              temp2:=Reg5;
                              shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                              shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);
        elsif (Lmax ="11")then
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                        Output(Outputparser+7 downto Outputparser) <=Reg6;
                              Outputf(Outputparser+7 downto Outputparser) :=Reg6;

                  Outputparser:=Outputparser -8;
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                              Output(Outputparser+7 downto Outputparser) <=Reg5;
                              Outputf(Outputparser+7 downto Outputparser) :=Reg5;

                  Outputparser:=Outputparser -8;
if (Outputparser <0) then
            Outputparser:= 0;
            end if;
                              Output(Outputparser+7 downto Outputparser) <=Reg4;
                              Outputf(Outputparser+7 downto Outputparser) :=Reg4;
```

```
                    Outputparser:=Outputparser -8;
                                        temp:=Reg6;
                                        temp2:=Reg5;
                                        temp3:=Reg4;
                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);

                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp3);
          end if ;

        elsif(Ip = "111")then
               report "7";
          if (Lmax ="01")then
     if (Outputparser <0) then
              Outputparser:= 0;
              end if;
                         Output(Outputparser+7 downto Outputparser) <=Reg7;
                                        Outputf(Outputparser+7 downto Outputparser) :=Reg7;

                    Outputparser:=Outputparser -8;
                                        temp:=Reg7;
                                        shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

          elsif(Lmax ="10")then
     if (Outputparser <0) then
              Outputparser:= 0;
          end if;
                    Output(Outputparser+7 downto Outputparser) <=Reg7;
                              Outputf(Outputparser+7 downto Outputparser) :=Reg7;

               Outputparser:=Outputparser -8;
if (Outputparser <0) then
          Outputparser:= 0;
          end if;
                              Output(Outputparser+7 downto Outputparser) <=Reg6;
                              Outputf(Outputparser+7 downto Outputparser) :=Reg6;

               Outputparser:=Outputparser -8;
                              temp:=Reg7;
          temp2:=Reg6;
                              shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                              shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);
     elsif (Lmax ="11")then
if (Outputparser <0) then
          Outputparser:= 0;
          end if;
                    Output(Outputparser+7 downto Outputparser) <=Reg7;
                              Outputf(Outputparser+7 downto Outputparser) :=Reg7;

               Outputparser:=Outputparser -8;
if (Outputparser <0) then
          Outputparser:= 0;
```

```
           end if;
                                               Output(Outputparser+7 downto Outputparser) <=Reg6;
                                               Outputf(Outputparser+7 downto Outputparser) :=Reg6;

                  Outputparser:=Outputparser -8;
if (Outputparser <0) then
           Outputparser:= 0;
           end if;
                                               Output(Outputparser+7 downto Outputparser) <=Reg5;
                                               Outputf(Outputparser+7 downto Outputparser) :=Reg5;

                  Outputparser:=Outputparser -8;
                                               temp:=Reg7;
                                               temp2:=Reg6;
                                               temp3:=Reg5;
                                               shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp);

                                               shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp2);

                                               shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,temp3);
      end if ;

   end if;

   end if ;
          if (Outputparser <0) then
           Outputparser:= 0;
```

```
             end if;


            --if ((Outputparser <= 511)) then
            -- adds the next byte into the output and decrements the output parser
            Output(Outputparser+7 downto Outputparser) <=byte;
            Outputf(Outputparser+7 downto Outputparser) :=byte;
            Outputparser:=Outputparser -8;
            --end if;









--          Reg7:= Reg6;
--          Reg6:= Reg5;
--          Reg5:= Reg4;
--          Reg4:= Reg3;
--          Reg3:= Reg2;
--          Reg2:= Reg1;
--          Reg1:= byte;
            shiftbyts( Reg1,Reg2,Reg3,Reg4,Reg5,Reg6,Reg7,byte);
            report "Outputparser" &integer'image(Outputparser);
end if;
Inputparser :=Inputparser -1;
report to_bstring(outputf);
```

```
end loop;
report to_bstring(outputf);
end if ;

end process;

end Behavioral;
```

## Decompression Test Bench:

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   06:42:34 01/14/2021
-- Design Name:
-- Module Name:
C:/Users/Mohamed/FinalProject/testdecompression.vhd
-- Project Name:  FinalProject
-- Target Device:
-- Tool versions:
```

```
-- Description:

--

-- VHDL Test Bench Created by ISE for module: DecompressionLoop

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

-- Notes:

-- This testbench has been automatically generated using types
std_logic and

-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends

-- that these types always be used for the top-level I/O of a design in
order

-- to guarantee that the testbench will bind correctly to the post-
implementation

-- simulation model.

--------------------------------------------------------------------------------

LIBRARY ieee;

USE ieee.std_logic_1164.ALL;
```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY testdecompression IS
END testdecompression;

ARCHITECTURE behavior OF testdecompression IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT DecompressionLoop
    PORT(
         pause : IN  std_logic;
         Input : IN  std_logic_vector(511 downto 0);
         Output : OUT  std_logic_vector(511 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal pause : std_logic := '0';
   signal Input : std_logic_vector(511 downto 0) := (others => '0');

        --Outputs
   signal Output : std_logic_vector(511 downto 0);
   -- No clocks detected in port list. Replace <clock> below with
   -- appropriate port name



BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: DecompressionLoop PORT MAP (
        pause => pause,
        Input => Input,
        Output => Output
      );

   -- Clock process definitions
```

```
  -- Stimulus process
  stim_proc: process
  begin
    pause <= '0';
          --please input the bits from the compression here
          Input <=
"000000110001000000001100001010010110001010010011000011011
101100010101110110001011011011000101101101100010110110110
1101000000110111110100101101101100110110110111011011011011
1001011011000000001100001001010110000011001101100001100110
1100001000000111001100000011001000000000110111100000011011
1000000011001101000101110011000000110100100000001101010000
0001110101110110000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
";
      -- insert stimulus here


      wait;
  end process;

END;
```