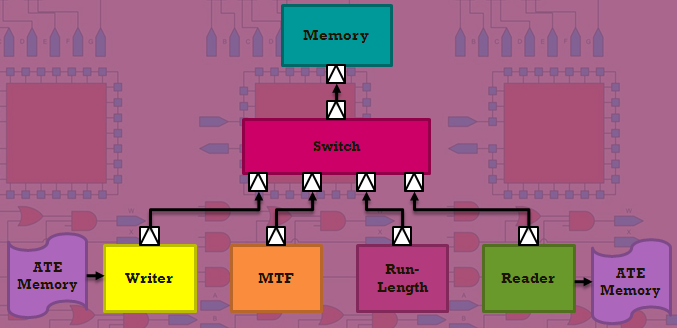
The goal of this assignment is to design a Data Compression System. The elements of this system are to be written in C/C++ and communications are to be handled using TLM (Transaction-Level Modeling) 2.0 interfaces. The system has four processing elements for reading a block of data, transforming the data, compressing it, and writing the compressed block of data. The following is a layout of the system itself:



For this homework, only a selected amount of C++ code/command line output will be shown. All code, with detailed comments, can be located in the ZIP file provided with submission of this assignment.

**System Explanation & High Level Design**

The crux of this Data Compression Systems lies with the Switch structure that is setup to handle all messaging traffic and communication. The system is setup in the following exact configuration:



The C++ code files that are used that interface directly with the High Level Design are:

* common\_header.h – contains all #include<> and TLM libraries used in project.
* memory.h – instantiation of memory which will be written to and read from.
* switch.cpp – router of read/write commands between modules and memory.
* main.cpp – overarching module that links all initiator/target sockets.

These base files were slightly modified to account for the 32 eight (8) bit words. The memory is a total of 1024 bits in size (128x8) in order to account for reading and writing space needed.

**Stage 1: Read from File :: Write to Memory**

The 1st Stage of this assignment requires that 32 eight (8) bit words be read in from an external file, and written to a common shared memory via the Switch.

The C++ code files that are used which construct the primary design of the Writer are:

* writer.cpp – reads from external file and writes information to shared Memory.

The *input.txt* file, which is used to provide the data stream for the program, shows the following data:

**input.txt**

00000000

11111111

00001111

11111111

00000111

11111000

00000000

11110001

11111110

00000011

11100011

11111100

00000111

11111000

00000110

00000011

00000001

11111100

00000111

00001111

11110000

00000111

11111110

00001111

00000011

11111000

00000001

00000000

00000000

00011111

11111111

11111100

The following portion of the *writer.cpp* file is what reads in this information from the external file and writes it into the shared Memory via the Switch – note the few changes made to accommodate this new bit architecture:

**… </beginning of *writer.cpp*, WRITE into Memory>**

cout << "Initiate Writer (thread process):\n\n";

tlm::tlm\_generic\_payload\* trans = new tlm::tlm\_generic\_payload;

sc\_time delay = sc\_time(8, SC\_PS);

if(writeAllowed == true){

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\* BEGIN READ FROM INPUT FILE, WRITE TO MEMORY \*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

// Declare a file from which to read character input:

fstream in\_data ("input.txt");

// Updated for HW 07, since we now need 32, 8-bit words:

for (int i = 0; i < 256; i += 8)

{

int data; string in\_buf;

cout << "Data IN: ";

// Read in each 8-bit word from an external input file:

getline(in\_data,in\_buf);

cout << in\_buf << ",\t";

data = bitset<32>(in\_buf).to\_ulong();

cout << "Data HEX: " << data << endl;

tlm::tlm\_command cmd = tlm::TLM\_WRITE\_COMMAND;

trans->set\_command( cmd );

trans->set\_address( i );

trans->set\_data\_ptr( reinterpret\_cast<unsigned char\*>(&data) );

// Modify data\_length to 8 bits

trans->set\_data\_length( 8 );

// Modify data\_streaming\_width to 8 bits

trans->set\_streaming\_width( 8 ); // = data\_length for no streaming

trans->set\_byte\_enable\_ptr( 0 ); // 0 indicates unused

trans->set\_dmi\_allowed( false ); // Mandatory initial values

trans->set\_response\_status(tlm::TLM\_INCOMPLETE\_RESPONSE );

**… </end of *writer.cpp*>**

In an effort to correctly verify the external read from *input.txt* and the write to the *Memory*, I inserted command line print statements in *switch.cpp* and *writer.cpp*. It is important to note that writing is done to spots 0-255 of the shared Memory (32x8 bit words). The output can be seen below, and maps exactly to the values found in *input.txt*:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*BEGIN READ FROM INPUT FILE, WRITE TO MEMORY\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Data IN: 00000000, Data HEX: 0

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 0 } , data = 0 at time 0 s

Data IN: 11111111, Data HEX: ff

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 8 } , data = ff at time 0 s

Data IN: 00001111, Data HEX: f

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 10 } , data = f at time 0 s

Data IN: 11111111, Data HEX: ff

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 18 } , data = ff at time 0 s

Data IN: 00000111, Data HEX: 7

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 20 } , data = 7 at time 0 s

Data IN: 11111000, Data HEX: f8

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 28 } , data = f8 at time 0 s

Data IN: 00000000, Data HEX: 0

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 30 } , data = 0 at time 0 s

Data IN: 11110001, Data HEX: f1

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 38 } , data = f1 at time 0 s

Data IN: 11111110, Data HEX: fe

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 40 } , data = fe at time 0 s

Data IN: 00000011, Data HEX: 3

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 48 } , data = 3 at time 0 s

Data IN: 11100011, Data HEX: e3

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 50 } , data = e3 at time 0 s

Data IN: 11111100, Data HEX: fc

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 58 } , data = fc at time 0 s

Data IN: 00000111, Data HEX: 7

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 60 } , data = 7 at time 0 s

Data IN: 11111000, Data HEX: f8

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 68 } , data = f8 at time 0 s

Data IN: 00000110, Data HEX: 6

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 70 } , data = 6 at time 0 s

Data IN: 00000011, Data HEX: 3

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 78 } , data = 3 at time 0 s

Data IN: 00000001, Data HEX: 1

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 80 } , data = 1 at time 0 s

Data IN: 11111100, Data HEX: fc

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 88 } , data = fc at time 0 s

Data IN: 00000111, Data HEX: 7

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 90 } , data = 7 at time 0 s

Data IN: 00001111, Data HEX: f

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, 98 } , data = f at time 0 s

Data IN: 11110000, Data HEX: f0

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, a0 } , data = f0 at time 0 s

Data IN: 00000111, Data HEX: 7

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, a8 } , data = 7 at time 0 s

Data IN: 11111110, Data HEX: fe

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, b0 } , data = fe at time 0 s

Data IN: 00001111, Data HEX: f

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, b8 } , data = f at time 0 s

Data IN: 00000011, Data HEX: 3

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, c0 } , data = 3 at time 0 s

Data IN: 11111000, Data HEX: f8

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, c8 } , data = f8 at time 0 s

Data IN: 00000001, Data HEX: 1

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, d0 } , data = 1 at time 0 s

Data IN: 00000000, Data HEX: 0

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, d8 } , data = 0 at time 0 s

Data IN: 00000000, Data HEX: 0

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, e0 } , data = 0 at time 0 s

Data IN: 00011111, Data HEX: 1f

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, e8 } , data = 1f at time 0 s

Data IN: 11111111, Data HEX: ff

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, f0 } , data = ff at time 0 s

Data IN: 11111100, Data HEX: fc

ID is: 0 - After calling b\_transport in Switch:

Trans = { W, f8 } , data = fc at time 0 s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* END READ FROM INPUT FILE, WRITE TO MEMORY \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Stage 2: Read from Memory :: Perform BWT :: Write to Memory**

The 2nd Stage of this assignment requires that 32 eight (8) bit words be read in from the shared Memory first into local MTF memory. Afterwards, the BWT algorithm is performed to transform the information. Lastly, the transformed data is written back into the shared Memory. Reading from and writing to the Memory is done via the Switch.

The C++ code files that are used which construct the primary design of the Writer are:

* MTF.cpp – reads from Memory, performs BWT, and writes back to Memory.

In an effort to correctly verify the internal read from the *memory*, I inserted command line print statements in *switch.cpp* and MTF*.cpp*. The output can be seen below, and maps exactly to the values found placed into the *memory* from *writer.cpp*:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BEGIN READ FROM MEMORY, WRITE TO MTF LOCAL MEM\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 0 } , data = 0 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 8 } , data = ff at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 10 } , data = f at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 18 } , data = ff at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 20 } , data = 7 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 28 } , data = f8 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 30 } , data = 0 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 38 } , data = f1 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 40 } , data = fe at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 48 } , data = 3 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 50 } , data = e3 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 58 } , data = fc at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 60 } , data = 7 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 68 } , data = f8 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 70 } , data = 6 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 78 } , data = 3 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 80 } , data = 1 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 88 } , data = fc at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 90 } , data = 7 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, 98 } , data = f at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, a0 } , data = f0 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, a8 } , data = 7 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, b0 } , data = fe at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, b8 } , data = f at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, c0 } , data = 3 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, c8 } , data = f8 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, d0 } , data = 1 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, d8 } , data = 0 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, e0 } , data = 0 at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, e8 } , data = 1f at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, f0 } , data = ff at time 0 s

ID is: 1 - After calling b\_transport in Switch:

Trans = { R, f8 } , data = fc at time 0 s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* END READ FROM MEMORY, WRITE TO MTF LOCAL MEM \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Once the information is read from the Memory and is stored locally into a *buffer*, the BWT algorithm is performed. The following is my implementation of BWT, optimized by the standard template library (STL) string classes inherent to C++:

**… </beginning of *MTF.cpp*, BWT Function Call>**

// Define the MTF function, which now performs BWT:

int\* MTF\_(int\* mem)

{

// 1st, convert the memory received into a 256 bit local string:

string base\_str = "";

for (int i = 0; i < 32; i++)

base\_str += bitset<8>(mem[i]).to\_string(); // Convert each int to str

cout << "Base string from MEM = \n\n" << base\_str << "\n\n";

//2nd, create all rotations for this string. Output them to external file.

// Open external file to output all rotations:

ofstream rt\_out("BWT\_rotations.txt");

vector <string> base\_rt (256);

base\_rt[0] = base\_str;

string itr\_str = base\_str;

rt\_out << "Start BWT Base String Rotations:\n\n";

rt\_out << "Rotation 1:\n" << base\_rt[0] << endl << endl;

for (int i = 1; i < 256; i++)

{

// Create rotation from current itr\_str and set in vector:

rotate(itr\_str.begin(), itr\_str.begin()+1, itr\_str.end());

base\_rt[i] = itr\_str;

rt\_out << "Rotation " << (i+1) << ":\n" << base\_rt[i] << endl << endl;

}

// 3rd, sort all rotations, get BWT Trans output from last element of rows.

sort(base\_rt.begin(), base\_rt.end());

cout << "Sorted BWT Rotations Generated and Output to:BWT\_Rotations.txt\n\n";

rt\_out << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

rt\_out << "\* SORTED BWT Rotations Generated: \*\n";

rt\_out << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

for (int i = 0; i < 256; i++)

rt\_out << base\_rt[i] << endl << endl;

rt\_out << endl << endl;

rt\_out << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

rt\_out << "\* ENCODED BWT ROTATION: \*\n";

rt\_out << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout << "Encoded BWT Output string from vector of sorted strings:\n\n";

// Create encoded BWT Output from the last char of vector of sorted strings:

for (int i = 0; i < 256; i++)

{

if (base\_rt[i][255] == '0')

buffer[i] = 0;

else if (base\_rt[i][255] == '1')

buffer[i] = 1;

else

buffer[i] = 2;

cout << buffer[i];

rt\_out << buffer[i];

}

cout << endl << endl;

rt\_out.close();

return buffer;

}

**… </end of *MTF.cpp* >**

All of the individual string rotations are output to a file called BWT\_rotations.txt and can be seen in the project folder if needed. Output from this BWT section on the Command Line shows the following:

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!! ENTER BWT Calculation !!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Base string from MEM =

00000000111111110000111111111111000001111111100000000000111100011111111000000011

11100011111111000000011111111000000001100000001100000001111111000000011100001111

11110000000001111111111000001111000000111111100000000001000000000000000000011111

1111111111111100

Sorted BWT Rotations Generated and Output to: BWT\_Rotations.txt

Encoded BWT Output string from vector of sorted strings:

10000000101010000100100000011011100000000010000000001000010000000000000010001000

00000000001100000000000000000000000000000000000000000111111111111111111111101101

11111111111111111101111111111101110111111111111011111111111101110111110100001101

1110111101111110

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!! EXIT BWT Calculation !!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Once the BWT Algorithm is performed, the Encoded BWT Output string is contained in the initial *buffer* pointer that was sent to the \_MTF function. The following portion of the *MTF.cpp* file is what takes the generated string and writes it into the shared Memory via the Switch:

**… </beginning of *MTF.cpp*, WRITE into Memory>**

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\* BEGIN WRITE TO MEMORY, AFTER MTF (BWT ALGO) DONE \*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

for (int i = 0; i < 256; i += 8)

{

// Begin by converting each 8bit word into an 8bit string.

string base\_str = "";

for (int j = i; j < i+8; j++)

base\_str += bitset<1>(buffer[j]).to\_string(); // Convert int to str.

data = bitset<32>(base\_str).to\_ulong();

cout << "DATA TO WRITE: " << base\_str << "\tIn HEX: " << data << endl;

cmd = tlm::TLM\_WRITE\_COMMAND;

trans->set\_command( cmd );

trans->set\_address( i );

trans->set\_data\_ptr( reinterpret\_cast<unsigned char\*>(&data) );

// Modify data\_length to 8 bits

trans->set\_data\_length( 8 );

// Modify data\_streaming\_width to 8 bits

trans->set\_streaming\_width( 8 ); // = data\_length to indicate no streaming

trans->set\_byte\_enable\_ptr( 0 ); // 0 indicates unused

trans->set\_dmi\_allowed( false ); // Mandatory initial value

trans->set\_response\_status( tlm::TLM\_INCOMPLETE\_RESPONSE );

**… </end of *MTF.cpp*>**

In an effort to correctly verify the write to the shared *Memory*, I inserted command line print statements in *switch.cpp* and *MTF.cpp*. It is important to note that writing is done to spots 256-511 of the shared Memory (32x8 bit words). The output can be seen below:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BEGIN WRITE TO MEMORY, AFTER MTF (BWT ALGO) DONE \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DATA TO WRITE: 10000000 In HEX: 80

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 100 } , data = 80 at time 0 s

DATA TO WRITE: 10101000 In HEX: a8

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 108 } , data = a8 at time 0 s

DATA TO WRITE: 01001000 In HEX: 48

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 110 } , data = 48 at time 0 s

DATA TO WRITE: 00011011 In HEX: 1b

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 118 } , data = 1b at time 0 s

DATA TO WRITE: 10000000 In HEX: 80

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 120 } , data = 80 at time 0 s

DATA TO WRITE: 00100000 In HEX: 20

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 128 } , data = 20 at time 0 s

DATA TO WRITE: 00001000 In HEX: 8

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 130 } , data = 8 at time 0 s

DATA TO WRITE: 01000000 In HEX: 40

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 138 } , data = 40 at time 0 s

DATA TO WRITE: 00000000 In HEX: 0

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 140 } , data = 0 at time 0 s

DATA TO WRITE: 10001000 In HEX: 88

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 148 } , data = 88 at time 0 s

DATA TO WRITE: 00000000 In HEX: 0

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 150 } , data = 0 at time 0 s

DATA TO WRITE: 00110000 In HEX: 30

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 158 } , data = 30 at time 0 s

DATA TO WRITE: 00000000 In HEX: 0

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 160 } , data = 0 at time 0 s

DATA TO WRITE: 00000000 In HEX: 0

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 168 } , data = 0 at time 0 s

DATA TO WRITE: 00000000 In HEX: 0

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 170 } , data = 0 at time 0 s

DATA TO WRITE: 00000000 In HEX: 0

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 178 } , data = 0 at time 0 s

DATA TO WRITE: 00000111 In HEX: 7

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 180 } , data = 7 at time 0 s

DATA TO WRITE: 11111111 In HEX: ff

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 188 } , data = ff at time 0 s

DATA TO WRITE: 11111111 In HEX: ff

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 190 } , data = ff at time 0 s

DATA TO WRITE: 11101101 In HEX: ed

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 198 } , data = ed at time 0 s

DATA TO WRITE: 11111111 In HEX: ff

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1a0 } , data = ff at time 0 s

DATA TO WRITE: 11111111 In HEX: ff

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1a8 } , data = ff at time 0 s

DATA TO WRITE: 11011111 In HEX: df

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1b0 } , data = df at time 0 s

DATA TO WRITE: 11111101 In HEX: fd

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1b8 } , data = fd at time 0 s

DATA TO WRITE: 11011111 In HEX: df

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1c0 } , data = df at time 0 s

DATA TO WRITE: 11111110 In HEX: fe

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1c8 } , data = fe at time 0 s

DATA TO WRITE: 11111111 In HEX: ff

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1d0 } , data = ff at time 0 s

DATA TO WRITE: 11110111 In HEX: f7

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1d8 } , data = f7 at time 0 s

DATA TO WRITE: 01111101 In HEX: 7d

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1e0 } , data = 7d at time 0 s

DATA TO WRITE: 00001101 In HEX: d

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1e8 } , data = d at time 0 s

DATA TO WRITE: 11101111 In HEX: ef

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1f0 } , data = ef at time 0 s

DATA TO WRITE: 01111110 In HEX: 7e

ID is: 1 - After calling b\_transport in Switch:

From MTF:

trans = { W, 1f8 } , data = 7e at time 0 s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* END WRITE TO MEMORY, AFTER MTF (BWT ALGO) DONE \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Stage 3: Read from Memory :: Perform RLE :: Write to Memory**

The 3nd Stage of this assignment requires that 32 eight (8) bit words be read in from the shared Memory first into local RLE memory. This information contains the transformation from the previous part of the assignment, post-BWT algorithm. Afterwards, the Run Length Encoding (RLE) algorithm is performed to compress the information. Lastly, the compressed data is written back into the shared Memory. Reading from and writing to the Memory is done via the Switch.

The C++ code files that are used which construct the primary design of the Writer are:

* Run-Length.cpp – reads from Memory, performs RLE, and writes back to Memory.

In an effort to correctly verify the internal read from the *memory*, I inserted command line print statements in *switch.cpp* and *Run-Length.cpp*. The output can be seen below, and maps exactly to the values found placed into the *Memory* from *MTF.cpp*:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BEGIN READ FROM MEMORY, WRITE TO RLE LOCAL MEM \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 100 } , data = 80 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 108 } , data = a8 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 110 } , data = 48 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 118 } , data = 1b at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 120 } , data = 80 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 128 } , data = 20 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 130 } , data = 8 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 138 } , data = 40 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 140 } , data = 0 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 148 } , data = 88 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 150 } , data = 0 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 158 } , data = 30 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 160 } , data = 0 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 168 } , data = 0 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 170 } , data = 0 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 178 } , data = 0 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 180 } , data = 7 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 188 } , data = ff at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 190 } , data = ff at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 198 } , data = ed at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1a0 } , data = ff at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1a8 } , data = ff at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1b0 } , data = df at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1b8 } , data = fd at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1c0 } , data = df at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1c8 } , data = fe at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1d0 } , data = ff at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1d8 } , data = f7 at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1e0 } , data = 7d at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1e8 } , data = d at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1f0 } , data = ef at time 0 s

ID is: 2 - After calling b\_transport in Switch:

From RLE: trans = { R, 1f8 } , data = 7e at time 0 s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* END READ FROM MEMORY, WRITE TO RLE LOCAL MEM \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Once the information is read from the Memory and is stored locally into a *buffer*, the RLE algorithm is performed. The following is my implementation of RLE, optimized by the standard template library (STL) string classes inherent to C++:

**… </beginning of *Run\_Length.cpp*, RLE Function Call>**

// Program the Run-Length compression algorithm, using STL string class:

string RunLength\_(int\* mem){

// Convert the memory received into a 256 bit local string:

string base\_str = "";

for (int i = 0; i < 32; i++)

base\_str += bitset<8>(mem[i]).to\_string(); //Convert each int to str

cout << "Base string is:\n" << base\_str << endl << endl;

// Perform Run-Length Encoding (RLE) on the base string:

bool IS\_0 = false;

char IS\_1 = false;

int runLength = 1;

int ptr = 0;

while (ptr < 256)

{

stringstream convStr;

string concat = "";

// Determine whether this current run is of 0's or 1's:

if (base\_str.at(ptr) == '0') IS\_0 = true;

else IS\_1 = true;

// Begin accumulation of runs:

while (((ptr + 1) < 256) && (base\_str.at(ptr) == base\_str.at(ptr+1)))

{

runLength++;

ptr++;

}

// Now append runLength and actual character to the compressedSt:

convStr << runLength;

concat = convStr.str();

compressedSt += concat;

if (IS\_0) compressedSt += "0";

if (IS\_1) compressedSt += "1";

// Reset all variables and increment pointer variable.

IS\_0 = false; IS\_1 = false; runLength = 1; ptr++; convStr.clear();

}

cout << "\nFINAL Compressed string is: \n" << compressedSt << endl << endl;

return compressedSt;

}

**… </end of *Run-Length.cpp* >**

Output from this RLE section on the Command Line shows the following **correct** compression scheme:

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!! ENTER RLE Calculation !!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Base string is:

10000000101010000100100000011011100000000010000000001000010000000000000010001000

00000000001100000000000000000000000000000000000000000111111111111111111111101101

11111111111111111101111111111101110111111111111011111111111101110111110100001101

1110111101111110

FINAL Compressed string is:

11701110111011401120116021103190119011401114011301113021410221102110191101111031

1012110121103110511011402110411041106110

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!! EXIT RLE Calculation !!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Once the RLE Algorithm is performed, the compressed RLE Output string is contained in the string that was sent to the \_RLE function. The following portion of the *Run-Length.cpp* file is what takes the generated string and writes it into the shared Memory via the Switch:

**… </beginning of *Run-Length.cpp*, WRITE into Memory>**

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\* BEGIN WRITE TO MEMORY, AFTER RLE \*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

// In this case, send one hexadecimal bit at a time:

for (int i = 0; i < compressedSt.length(); i++)

{

data = bitset<4>(compressedSt[i]).to\_ulong();

cmd = tlm::TLM\_WRITE\_COMMAND;

trans->set\_command( cmd );

trans->set\_address( i );

trans->set\_data\_ptr(reinterpret\_cast<unsigned char\*>(&data) );

trans->set\_data\_length( 1 );

trans->set\_streaming\_width( 1 ); // = data\_length to indicate no streaming

trans->set\_byte\_enable\_ptr( 0 ); // 0 indicates unused

trans->set\_dmi\_allowed( false ); // Mandatory initial value

trans->set\_response\_status( tlm::TLM\_INCOMPLETE\_RESPONSE );

fout << "From RLE: at "<< sc\_time\_stamp()<<" "<< " delay= "<<delay << address:" << dec << i << " " << name() << " new, cmd=" << (cmd ? 'W' : 'R')<< ", data=" << hex << data << endl;

socket->b\_transport( \*trans, delay ); // Blocking transport call

if ( trans->is\_response\_error() ) {

char txt[100];

sprintf(txt, "Error from b\_transport, response status = %s",

trans->get\_response\_string().c\_str());}

if ( trans->is\_response\_ok()){

char txt[100];

sprintf(txt, "ok from b\_transport in process 1 , response status = %s",

trans->get\_response\_string().c\_str());}

cout << "From RLE: trans = { " << (cmd ? 'W' : 'R') << ", " << dec << trans->get\_address() << " } , data =" << hex << data << " at time " << sc\_time\_stamp()+ delay << endl;}

compressedLen = compressedSt.length();

// Store size of compression string to pull from memory.

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\* END WRITE TO MEMORY, AFTER RLE \*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

}

**… </end of *Run-Length.cpp*>**

In an effort to correctly verify the write to the shared *Memory*, I inserted command line print statements in *switch.cpp* and *RLE.cpp*. It is important to note that writing is done to spots 512-(length of string) of the shared Memory. The length of the compressed string is being stored in a global variable called *compressedLen*. The output can be seen below:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BEGIN WRITE TO MEMORY, AFTER RLE \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Length of COMPRESSED string from RLE = 78

From RLE: trans = { W, 512 } , data =1 at time 0 s, i = 0

From RLE: trans = { W, 520 } , data =1 at time 0 s, i = 1

From RLE: trans = { W, 528 } , data =7 at time 0 s, i = 2

From RLE: trans = { W, 536 } , data =0 at time 0 s, i = 3

From RLE: trans = { W, 544 } , data =1 at time 0 s, i = 4

From RLE: trans = { W, 552 } , data =1 at time 0 s, i = 5

From RLE: trans = { W, 560 } , data =1 at time 0 s, i = 6

From RLE: trans = { W, 568 } , data =0 at time 0 s, i = 7

From RLE: trans = { W, 576 } , data =1 at time 0 s, i = 8

From RLE: trans = { W, 584 } , data =1 at time 0 s, i = 9

From RLE: trans = { W, 592 } , data =1 at time 0 s, i = 10

From RLE: trans = { W, 600 } , data =0 at time 0 s, i = 11

From RLE: trans = { W, 608 } , data =1 at time 0 s, i = 12

From RLE: trans = { W, 616 } , data =1 at time 0 s, i = 13

From RLE: trans = { W, 624 } , data =4 at time 0 s, i = 14

From RLE: trans = { W, 632 } , data =0 at time 0 s, i = 15

From RLE: trans = { W, 640 } , data =1 at time 0 s, i = 16

From RLE: trans = { W, 648 } , data =1 at time 0 s, i = 17

From RLE: trans = { W, 656 } , data =2 at time 0 s, i = 18

From RLE: trans = { W, 664 } , data =0 at time 0 s, i = 19

From RLE: trans = { W, 672 } , data =1 at time 0 s, i = 20

From RLE: trans = { W, 680 } , data =1 at time 0 s, i = 21

From RLE: trans = { W, 688 } , data =6 at time 0 s, i = 22

From RLE: trans = { W, 696 } , data =0 at time 0 s, i = 23

From RLE: trans = { W, 704 } , data =2 at time 0 s, i = 24

From RLE: trans = { W, 712 } , data =1 at time 0 s, i = 25

From RLE: trans = { W, 720 } , data =1 at time 0 s, i = 26

From RLE: trans = { W, 728 } , data =0 at time 0 s, i = 27

From RLE: trans = { W, 736 } , data =3 at time 0 s, i = 28

From RLE: trans = { W, 744 } , data =1 at time 0 s, i = 29

From RLE: trans = { W, 752 } , data =9 at time 0 s, i = 30

From RLE: trans = { W, 760 } , data =0 at time 0 s, i = 31

From RLE: trans = { W, 768 } , data =1 at time 0 s, i = 32

From RLE: trans = { W, 776 } , data =1 at time 0 s, i = 33

From RLE: trans = { W, 784 } , data =9 at time 0 s, i = 34

From RLE: trans = { W, 792 } , data =0 at time 0 s, i = 35

From RLE: trans = { W, 800 } , data =1 at time 0 s, i = 36

From RLE: trans = { W, 808 } , data =1 at time 0 s, i = 37

From RLE: trans = { W, 816 } , data =4 at time 0 s, i = 38

From RLE: trans = { W, 824 } , data =0 at time 0 s, i = 39

From RLE: trans = { W, 832 } , data =1 at time 0 s, i = 40

From RLE: trans = { W, 840 } , data =1 at time 0 s, i = 41

From RLE: trans = { W, 848 } , data =1 at time 0 s, i = 42

From RLE: trans = { W, 856 } , data =4 at time 0 s, i = 43

From RLE: trans = { W, 864 } , data =0 at time 0 s, i = 44

From RLE: trans = { W, 872 } , data =1 at time 0 s, i = 45

From RLE: trans = { W, 880 } , data =1 at time 0 s, i = 46

From RLE: trans = { W, 888 } , data =3 at time 0 s, i = 47

From RLE: trans = { W, 896 } , data =0 at time 0 s, i = 48

From RLE: trans = { W, 904 } , data =1 at time 0 s, i = 49

From RLE: trans = { W, 912 } , data =1 at time 0 s, i = 50

From RLE: trans = { W, 920 } , data =1 at time 0 s, i = 51

From RLE: trans = { W, 928 } , data =3 at time 0 s, i = 52

From RLE: trans = { W, 936 } , data =0 at time 0 s, i = 53

From RLE: trans = { W, 944 } , data =2 at time 0 s, i = 54

From RLE: trans = { W, 952 } , data =1 at time 0 s, i = 55

From RLE: trans = { W, 960 } , data =4 at time 0 s, i = 56

From RLE: trans = { W, 968 } , data =1 at time 0 s, i = 57

From RLE: trans = { W, 976 } , data =0 at time 0 s, i = 58

From RLE: trans = { W, 984 } , data =2 at time 0 s, i = 59

From RLE: trans = { W, 992 } , data =2 at time 0 s, i = 60

From RLE: trans = { W, 1000 } , data =1 at time 0 s, i = 61

From RLE: trans = { W, 1008 } , data =1 at time 0 s, i = 62

From RLE: trans = { W, 1016 } , data =0 at time 0 s, i = 63

From RLE: trans = { W, 1024 } , data =2 at time 0 s, i = 64

From RLE: trans = { W, 1032 } , data =1 at time 0 s, i = 65

From RLE: trans = { W, 1040 } , data =1 at time 0 s, i = 66

From RLE: trans = { W, 1048 } , data =0 at time 0 s, i = 67

From RLE: trans = { W, 1056 } , data =1 at time 0 s, i = 68

From RLE: trans = { W, 1064 } , data =9 at time 0 s, i = 69

From RLE: trans = { W, 1072 } , data =1 at time 0 s, i = 70

From RLE: trans = { W, 1080 } , data =1 at time 0 s, i = 71

From RLE: trans = { W, 1088 } , data =0 at time 0 s, i = 72

From RLE: trans = { W, 1096 } , data =1 at time 0 s, i = 73

From RLE: trans = { W, 1104 } , data =1 at time 0 s, i = 74

From RLE: trans = { W, 1112 } , data =1 at time 0 s, i = 75

From RLE: trans = { W, 1120 } , data =1 at time 0 s, i = 76

From RLE: trans = { W, 1128 } , data =0 at time 0 s, i = 77

From RLE: trans = { W, 1136 } , data =3 at time 0 s, i = 78

From RLE: trans = { W, 1144 } , data =1 at time 0 s, i = 79

From RLE: trans = { W, 1152 } , data =1 at time 0 s, i = 80

From RLE: trans = { W, 1160 } , data =0 at time 0 s, i = 81

From RLE: trans = { W, 1168 } , data =1 at time 0 s, i = 82

From RLE: trans = { W, 1176 } , data =2 at time 0 s, i = 83

From RLE: trans = { W, 1184 } , data =1 at time 0 s, i = 84

From RLE: trans = { W, 1192 } , data =1 at time 0 s, i = 85

From RLE: trans = { W, 1200 } , data =0 at time 0 s, i = 86

From RLE: trans = { W, 1208 } , data =1 at time 0 s, i = 87

From RLE: trans = { W, 1216 } , data =2 at time 0 s, i = 88

From RLE: trans = { W, 1224 } , data =1 at time 0 s, i = 89

From RLE: trans = { W, 1232 } , data =1 at time 0 s, i = 90

From RLE: trans = { W, 1240 } , data =0 at time 0 s, i = 91

From RLE: trans = { W, 1248 } , data =3 at time 0 s, i = 92

From RLE: trans = { W, 1256 } , data =1 at time 0 s, i = 93

From RLE: trans = { W, 1264 } , data =1 at time 0 s, i = 94

From RLE: trans = { W, 1272 } , data =0 at time 0 s, i = 95

From RLE: trans = { W, 1280 } , data =5 at time 0 s, i = 96

From RLE: trans = { W, 1288 } , data =1 at time 0 s, i = 97

From RLE: trans = { W, 1296 } , data =1 at time 0 s, i = 98

From RLE: trans = { W, 1304 } , data =0 at time 0 s, i = 99

From RLE: trans = { W, 1312 } , data =1 at time 0 s, i = 100

From RLE: trans = { W, 1320 } , data =1 at time 0 s, i = 101

From RLE: trans = { W, 1328 } , data =4 at time 0 s, i = 102

From RLE: trans = { W, 1336 } , data =0 at time 0 s, i = 103

From RLE: trans = { W, 1344 } , data =2 at time 0 s, i = 104

From RLE: trans = { W, 1352 } , data =1 at time 0 s, i = 105

From RLE: trans = { W, 1360 } , data =1 at time 0 s, i = 106

From RLE: trans = { W, 1368 } , data =0 at time 0 s, i = 107

From RLE: trans = { W, 1376 } , data =4 at time 0 s, i = 108

From RLE: trans = { W, 1384 } , data =1 at time 0 s, i = 109

From RLE: trans = { W, 1392 } , data =1 at time 0 s, i = 110

From RLE: trans = { W, 1400 } , data =0 at time 0 s, i = 111

From RLE: trans = { W, 1408 } , data =4 at time 0 s, i = 112

From RLE: trans = { W, 1416 } , data =1 at time 0 s, i = 113

From RLE: trans = { W, 1424 } , data =1 at time 0 s, i = 114

From RLE: trans = { W, 1432 } , data =0 at time 0 s, i = 115

From RLE: trans = { W, 1440 } , data =6 at time 0 s, i = 116

From RLE: trans = { W, 1448 } , data =1 at time 0 s, i = 117

From RLE: trans = { W, 1456 } , data =1 at time 0 s, i = 118

From RLE: trans = { W, 1464 } , data =0 at time 0 s, i = 119

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* END WRITE TO MEMORY, AFTER RLE \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Stage 4: Read from Memory :: Write to External File**

The 4th and final Stage of this assignment requires that the information written into memory from the RLE algorithm be read, via the *reader*, and output to an external file (memory). Reading from the Memory is done via the switch.

The C++ code files that are used which construct the primary design of the Writer are:

* reader.cpp – reads from Memory and outputs encoded string to external file.

In an effort to correctly verify the internal read from the *Memory*, I inserted command line print statements in *switch.cpp* and *reader.cpp*. The output can be seen below, and maps exactly to the values found placed into the *Memory* from *Run-Length.cpp*:

In Reader: trans = { R, 512 } , data = 1 at time 0 s

In Reader: trans = { R, 520 } , data = 1 at time 0 s

In Reader: trans = { R, 528 } , data = 7 at time 0 s

In Reader: trans = { R, 536 } , data = 0 at time 0 s

In Reader: trans = { R, 544 } , data = 1 at time 0 s

In Reader: trans = { R, 552 } , data = 1 at time 0 s

In Reader: trans = { R, 560 } , data = 1 at time 0 s

In Reader: trans = { R, 568 } , data = 0 at time 0 s

In Reader: trans = { R, 576 } , data = 1 at time 0 s

In Reader: trans = { R, 584 } , data = 1 at time 0 s

In Reader: trans = { R, 592 } , data = 1 at time 0 s

In Reader: trans = { R, 600 } , data = 0 at time 0 s

In Reader: trans = { R, 608 } , data = 1 at time 0 s

In Reader: trans = { R, 616 } , data = 1 at time 0 s

In Reader: trans = { R, 624 } , data = 4 at time 0 s

In Reader: trans = { R, 632 } , data = 0 at time 0 s

In Reader: trans = { R, 640 } , data = 1 at time 0 s

In Reader: trans = { R, 648 } , data = 1 at time 0 s

In Reader: trans = { R, 656 } , data = 2 at time 0 s

In Reader: trans = { R, 664 } , data = 0 at time 0 s

In Reader: trans = { R, 672 } , data = 1 at time 0 s

In Reader: trans = { R, 680 } , data = 1 at time 0 s

In Reader: trans = { R, 688 } , data = 6 at time 0 s

In Reader: trans = { R, 696 } , data = 0 at time 0 s

In Reader: trans = { R, 704 } , data = 2 at time 0 s

In Reader: trans = { R, 712 } , data = 1 at time 0 s

In Reader: trans = { R, 720 } , data = 1 at time 0 s

In Reader: trans = { R, 728 } , data = 0 at time 0 s

In Reader: trans = { R, 736 } , data = 3 at time 0 s

In Reader: trans = { R, 744 } , data = 1 at time 0 s

In Reader: trans = { R, 752 } , data = 9 at time 0 s

In Reader: trans = { R, 760 } , data = 0 at time 0 s

In Reader: trans = { R, 768 } , data = 1 at time 0 s

In Reader: trans = { R, 776 } , data = 1 at time 0 s

In Reader: trans = { R, 784 } , data = 9 at time 0 s

In Reader: trans = { R, 792 } , data = 0 at time 0 s

In Reader: trans = { R, 800 } , data = 1 at time 0 s

In Reader: trans = { R, 808 } , data = 1 at time 0 s

In Reader: trans = { R, 816 } , data = 4 at time 0 s

In Reader: trans = { R, 824 } , data = 0 at time 0 s

In Reader: trans = { R, 832 } , data = 1 at time 0 s

In Reader: trans = { R, 840 } , data = 1 at time 0 s

In Reader: trans = { R, 848 } , data = 1 at time 0 s

In Reader: trans = { R, 856 } , data = 4 at time 0 s

In Reader: trans = { R, 864 } , data = 0 at time 0 s

In Reader: trans = { R, 872 } , data = 1 at time 0 s

In Reader: trans = { R, 880 } , data = 1 at time 0 s

In Reader: trans = { R, 888 } , data = 3 at time 0 s

In Reader: trans = { R, 896 } , data = 0 at time 0 s

In Reader: trans = { R, 904 } , data = 1 at time 0 s

In Reader: trans = { R, 912 } , data = 1 at time 0 s

In Reader: trans = { R, 920 } , data = 1 at time 0 s

In Reader: trans = { R, 928 } , data = 3 at time 0 s

In Reader: trans = { R, 936 } , data = 0 at time 0 s

In Reader: trans = { R, 944 } , data = 2 at time 0 s

In Reader: trans = { R, 952 } , data = 1 at time 0 s

In Reader: trans = { R, 960 } , data = 4 at time 0 s

In Reader: trans = { R, 968 } , data = 1 at time 0 s

In Reader: trans = { R, 976 } , data = 0 at time 0 s

In Reader: trans = { R, 984 } , data = 2 at time 0 s

In Reader: trans = { R, 992 } , data = 2 at time 0 s

In Reader: trans = { R, 1000 } , data = 1 at time 0 s

In Reader: trans = { R, 1008 } , data = 1 at time 0 s

In Reader: trans = { R, 1016 } , data = 0 at time 0 s

In Reader: trans = { R, 1024 } , data = 2 at time 0 s

In Reader: trans = { R, 1032 } , data = 1 at time 0 s

In Reader: trans = { R, 1040 } , data = 1 at time 0 s

In Reader: trans = { R, 1048 } , data = 0 at time 0 s

In Reader: trans = { R, 1056 } , data = 1 at time 0 s

In Reader: trans = { R, 1064 } , data = 9 at time 0 s

In Reader: trans = { R, 1072 } , data = 1 at time 0 s

In Reader: trans = { R, 1080 } , data = 1 at time 0 s

In Reader: trans = { R, 1088 } , data = 0 at time 0 s

In Reader: trans = { R, 1096 } , data = 1 at time 0 s

In Reader: trans = { R, 1104 } , data = 1 at time 0 s

In Reader: trans = { R, 1112 } , data = 1 at time 0 s

In Reader: trans = { R, 1120 } , data = 1 at time 0 s

In Reader: trans = { R, 1128 } , data = 0 at time 0 s

In Reader: trans = { R, 1136 } , data = 3 at time 0 s

In Reader: trans = { R, 1144 } , data = 1 at time 0 s

In Reader: trans = { R, 1152 } , data = 1 at time 0 s

In Reader: trans = { R, 1160 } , data = 0 at time 0 s

In Reader: trans = { R, 1168 } , data = 1 at time 0 s

In Reader: trans = { R, 1176 } , data = 2 at time 0 s

In Reader: trans = { R, 1184 } , data = 1 at time 0 s

In Reader: trans = { R, 1192 } , data = 1 at time 0 s

In Reader: trans = { R, 1200 } , data = 0 at time 0 s

In Reader: trans = { R, 1208 } , data = 1 at time 0 s

In Reader: trans = { R, 1216 } , data = 2 at time 0 s

In Reader: trans = { R, 1224 } , data = 1 at time 0 s

In Reader: trans = { R, 1232 } , data = 1 at time 0 s

In Reader: trans = { R, 1240 } , data = 0 at time 0 s

In Reader: trans = { R, 1248 } , data = 3 at time 0 s

In Reader: trans = { R, 1256 } , data = 1 at time 0 s

In Reader: trans = { R, 1264 } , data = 1 at time 0 s

In Reader: trans = { R, 1272 } , data = 0 at time 0 s

In Reader: trans = { R, 1280 } , data = 5 at time 0 s

In Reader: trans = { R, 1288 } , data = 1 at time 0 s

In Reader: trans = { R, 1296 } , data = 1 at time 0 s

In Reader: trans = { R, 1304 } , data = 0 at time 0 s

In Reader: trans = { R, 1312 } , data = 1 at time 0 s

In Reader: trans = { R, 1320 } , data = 1 at time 0 s

In Reader: trans = { R, 1328 } , data = 4 at time 0 s

In Reader: trans = { R, 1336 } , data = 0 at time 0 s

In Reader: trans = { R, 1344 } , data = 2 at time 0 s

In Reader: trans = { R, 1352 } , data = 1 at time 0 s

In Reader: trans = { R, 1360 } , data = 1 at time 0 s

In Reader: trans = { R, 1368 } , data = 0 at time 0 s

In Reader: trans = { R, 1376 } , data = 4 at time 0 s

In Reader: trans = { R, 1384 } , data = 1 at time 0 s

In Reader: trans = { R, 1392 } , data = 1 at time 0 s

In Reader: trans = { R, 1400 } , data = 0 at time 0 s

In Reader: trans = { R, 1408 } , data = 4 at time 0 s

In Reader: trans = { R, 1416 } , data = 1 at time 0 s

In Reader: trans = { R, 1424 } , data = 1 at time 0 s

In Reader: trans = { R, 1432 } , data = 0 at time 0 s

In Reader: trans = { R, 1440 } , data = 6 at time 0 s

In Reader: trans = { R, 1448 } , data = 1 at time 0 s

In Reader: trans = { R, 1456 } , data = 1 at time 0 s

In Reader: trans = { R, 1464 } , data = 0 at time 0 s

Once the information is read from the Memory and is stored locally into a *buffer*, the reader then writes the data into an external file, called *output.txt*. The following is an excerpt from the C++ file in *reader.cpp*, after the reading is done from the Memory:

**… </beginning of *reader.cpp* >**

void thread\_process()

{

cout << "Initiate Reader (thread process):" << endl;

tlm::tlm\_generic\_payload\* trans = new tlm::tlm\_generic\_payload;

sc\_time delay = sc\_time(4, SC\_PS);

int readerAllowed = 1;

int data = 0; int out\_data = 0;

wait(readAllowed.default\_event());

if(readAllowed == true){

// Setup loop to retrieve all relevant information from the memory:

int\* out\_buf = new int[compressedLen];

for (int i = 0; i < compressedLen; i++){

tlm::tlm\_command cmd = tlm::TLM\_READ\_COMMAND;

trans->set\_command( cmd );

trans->set\_address( i\*8 );

trans->set\_data\_ptr( reinterpret\_cast<unsigned char\*>(&data) );

trans->set\_data\_length( 1 );

trans->set\_streaming\_width( 1 ); // = data\_length to indicate no streaming

trans->set\_byte\_enable\_ptr( 0 ); // 0 indicates unused

trans->set\_dmi\_allowed( false ); // Mandatory initial value

trans->set\_response\_status( tlm::TLM\_INCOMPLETE\_RESPONSE );

fout << "In Reader: at "<< sc\_time\_stamp()<<" "<< " delay= "<< delay << " address: "<< dec << i << " " << name() << " new, cmd=" << (cmd ? 'W' : 'R') << ", data=" << hex << data << endl;

socket->b\_transport( \*trans, delay ); // Blocking transport call

cout << "In Reader: trans = { " << (cmd ? 'W' : 'R') << ", " << dec << trans->get\_address()<< " } , data = " << dec << data << " at time " << sc\_time\_stamp()+ delay << endl;

out\_data = bitset<4>(data).to\_ulong();

out\_buf[i] = out\_data;

}

// Now output all data to external file:

ofstream str\_res("output.txt");

str\_res << "THE FINAL COMPRESSED STRING IS:\n\n";

for (int k = 0; k < compressedLen; k++)

str\_res << out\_buf[k];

str\_res.close();

}

}

**… </end of *reader.cpp* >**

**Final Results & Output**

In summary, the following was the output given the provided *input.txt* file:

**Encoded BWT Output string from vector of sorted strings**:

10000000101010000100100000011011100000000010000000001000010000000000000010001000

00000000001100000000000000000000000000000000000000000111111111111111111111101101

11111111111111111101111111111101110111111111111011111111111101110111110100001101

1110111101111110

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!! **ENTER RLE Calculation** !!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Base string is:

**10000000101010000100100000011011100000000010000000001000010000000000000010001000**

**00000000001100000000000000000000000000000000000000000111111111111111111111101101**

**11111111111111111101111111111101110111111111111011111111111101110111110100001101**

**1110111101111110**

FINAL Compressed string is:

**11701110111011401120116021103190119011401114011301113021410221102110191101111031**

**1012110121103110511011402110411041106110**

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!! **EXIT RLE Calculation** !!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Once the computation is done, inspection of the *output.txt* file yields:

**THE FINAL COMPRESSED STRING IS:**

**117011101110114011201160211031901190114011140113011130214102211021101911011110311012110121103110511011402110411041106110**

All data handling and thread processing between all SC Modules works, as expected. The Switch correctly interfaces with the Memory and allows for proper interpretation of these messages as expected.