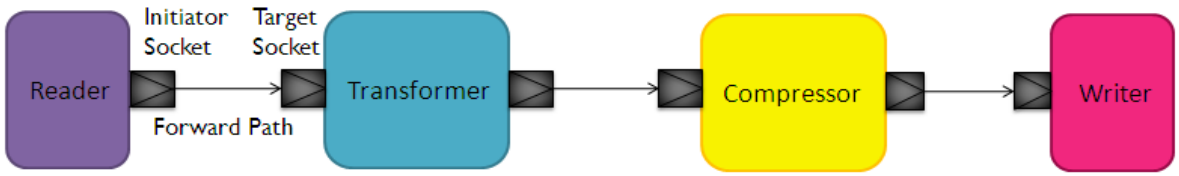
Worcester Polytechnic Institute
Electrical and Computer Engineering Department
Methodologies for System Level Design and Modeling

Online Offering – Fall 2010
Homework 7 – TLM2.0 Basic Concepts

Zainalabedin Navabi
navabi@ece.wpi.edu

In this homework you are to design a data compression system. Processing elements in this system are to be written in C/C++ and the communications are to be handled with TLM 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.

Problem Description:
In this design, the ***reader*** reads 32, 8-bit words from *"input.txt"* (this is considered a block of data). The ***transformer*** module uses BWT algorithm to transform data words to the form that is easier and more efficient to compress. The ***compressor*** uses run length encoding to compress the sorted (transformed) data. The compressed block is written to *"output.txt"* using ***writer.*** Implement this design using TLM blocking transport interface. You should write the description for the ***reader***, ***transformer***, ***compressor*** and ***writer*** in C/C++. You may have to define some internal memory buffers for the operation modules. To sort the table of BWT you can use any sorting algorithm you wish. The C/C++ code of run-length coding and the pseudo code of BWT are made available to you. Diagram below shows the structure of the processing elements and communications of the system you are designing.



Example:
Here is an example for transforming and compressing four 4-bit words. This shows the functionality of your system.

The ***reader*** reads the block shown below from *"input.txt"* and stores to its local memory. This block is transferred to ***transformer.***

1010
1111
0101
1010

The steps of BWT algorithm performed by this module are shown in the following table. As shown, all possible rotations of the input data are obtained first, and they are sorted in ascending order. Output bits from left to right are obtained by taking the right-most bits of the sorted data. The output block will be transferred to the ***compression*** module.

BWT Transform			
Input	All rotations	Sort rows	Output
1010111101011010	1010111101011010	0101010111101011	1111111010000110
	0101011110101101	0101011110101101	
	1010101111010110	0101101010101111	
	0101010111101011	0101111010110101	
	1010101011110101	0110101010111101	
	1101010101111010	0111101011010101	
	0110101010111101	1010101011110101	
	1011010101011110	1010101111010110	
	0101101010101111	1010110101010111	
	1010110101010111	1010111101011010	
	1101011010101011	1011010101011110	
	1110101101010101	1011110101101010	
	1111010110101010	1101010101111010	
	0111101011010101	1101011010101011	
	1011110101101010	1110101101010101	
	0101111010110101	1111010110101010	

The output of the ***compression*** module will be:

71**10**1**1**4**0**2**1**1**0** which translates to: **111100100011100001010010** in binary, assuming length 3 for each run.

The ***writer*** module receives the above block from the ***compression*** module and writes it to *"output.txt"*.

The C source code of run length encoding and the pseudo code of BWT are shown below:

```
String encode(String source) { // assume length of 3.
        StringBuffer dest = new StringBuffer();
        int runLength = 1;
        while (i < source.length()) {
                while (renLength < 7 && i+1 < source.length() && source.charAt(i) == source.charAt(i+1) ) {
                        runLength++;
                        i++;
                }
                dest.append(runLength);
                dest.append(source.charAt(i));
                runLength = 1;
                i++;
        }
        return dest.toString();
}
```

C source code of run length coding

```
function BWT (string s)
  create a table, rows are all possible rotations of s
  sort rows in ascending order
  return (last column of the table)
```

Pseudo code of BWT