EE 599
Spring 2020
Homework 1
Assigned: February 20, 2020
Due: March 06, 2020
Total Points: 100

1 Odd-even transposition sort [40 Points]

Odd-even transposition sort algorithm is a parallel sorting algorithm. It sorts n elements in n clocks (n is even), each of which requires *n/2* compare-exchange operations. This algorithm alternates between two phases, called the odd and even phases. Let $< a_1, a_2, ..., a_n >$ be the sequence to be sorted. During the odd phase, elements with odd indices are compared with their right neighbors, and if they are out of sequence they are exchanged; thus, the pairs $(a_1, a_2), (a_3, a_4), ..., (a_{n-1}, a_n)$ are compare-exchanged (assuming n is even). Similarly, during the even phase, elements with even indices are compared with their right neighbors, and if they are out of sequence they are exchanged; thus, the pairs $(a_2, a_3), (a_4, a_5), ..., (a_{n-2}, a_{n-1})$ are compare-exchanged. After n phases of odd-even exchanges, the sequence is sorted. An example sorting instance is shown in Figure 1 and the sequential algorithm is shown in figure 2.

1.1 Implementation : Using Xilinx Vivado Software (Use Zynq-7000 xc7z007sclg225-2 FPGA)

Consider only 8-bit arithmetic. All the values are already stored in the BRAM.

1. Using Verilog, implement **odd-even transposition circuit**, which takes n, 8 bit inputs and sort them.
2. For a 16 elements write a test bench and verify the waveforms.
3. Elaborate the design and include all the schematics' screenshots of the modules in the report.
4. Synthesis the design and include the screenshots.
5. Generate Resource and timing estimations and include them in the report.
6. Redo part 3, 4, 5 for 32, 64, 128.

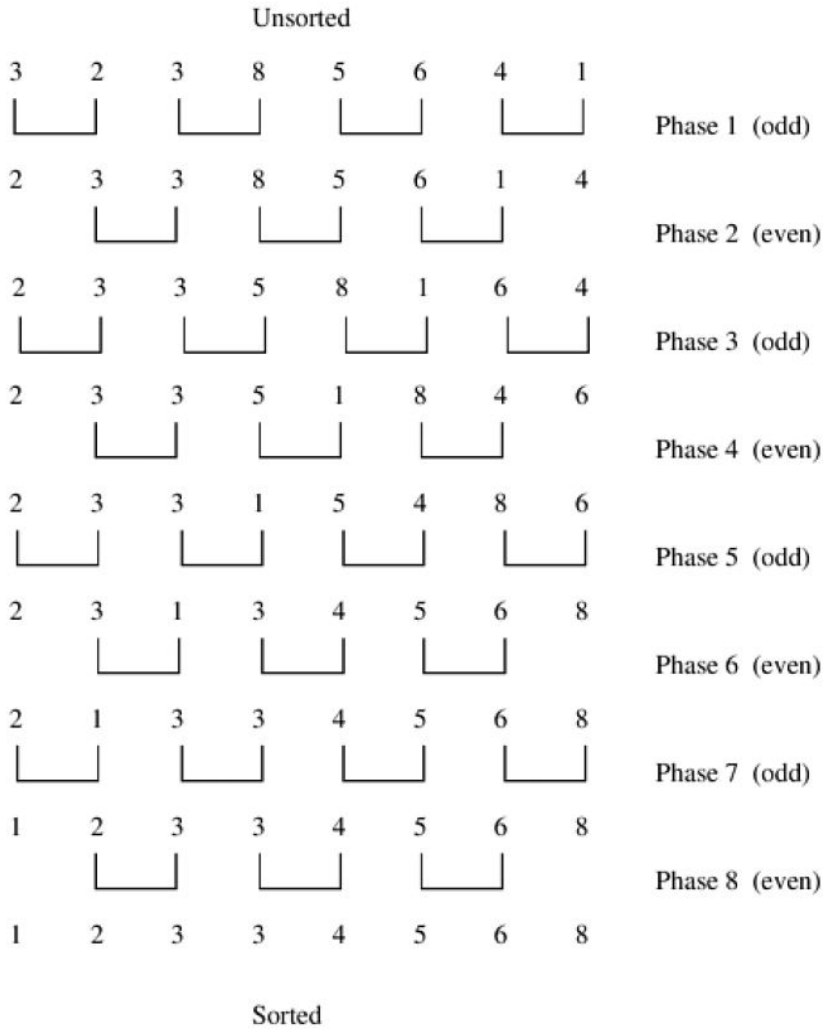

Figure 1: Example of odd-even transposition sort

```
1.   procedure ODD-EVEN(n)
2.   begin
3.      for i := 1 to n do
4.      begin
5.         if i is odd then
6.              for j := 0 to n/2 - 1 do
7.                  compare-exchange(a2j + 1, a2j + 2);
8.         if i is even then
9.              for j := 1 to n/2 - 1 do
10.                 compare-exchange(a2j, a2j + 1);
11.     end for
12.  end ODD-EVEN
```

Figure 2: Serial Algorithm of odd-even transposition sort

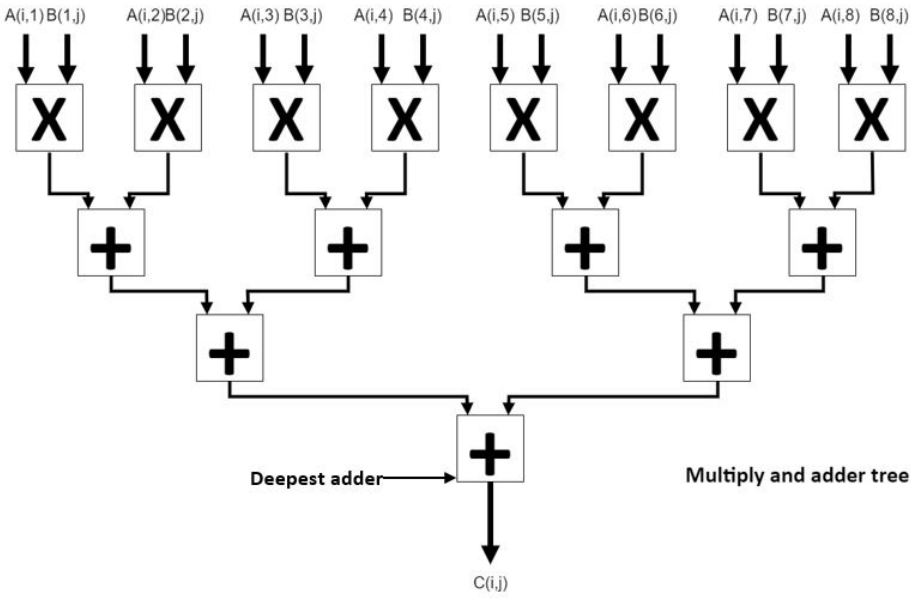

Figure 3: Example multiply and adder tree design

2 Dense Matrix-Matrix Multiplication [60 Points]

2.1 Scalable Multiply and adder tree

If A is an $m \times n$ matrix and B is an $n \times p$ matrix, the matrix product $C = AB$ (denoted without multiplication signs or dots) is defined to be the $m \times p$ matrix such that,

$$c_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + ... + a_{i,n}b_{n,j} = \sum_{k=1}^{n} a_{i,k}b_{k,j};$$

where for i = 1, ..., m and j = 1, ..., p

Consider two matrices A and B, each having the size of $n \times n$ where n = 2^r.

Figure 3 shows an example design of Multiply and Adder Tree. The adder tree consists of a Multiplication Step following Adder Steps. Given the size of matrices is $n \times n$, there are n multipliers in the first stage. Assume that matrix A saved in row order, and matrix B saved in column order in the memory. In the beginning, the first row of A and the first column of B loaded and multiplied together. Then in each Adder Step, partial sums are added together until it produces the final result corresponding to an element in the output matrix. Adder steps consist of 2 element adders as shown in Figure 2. Notice that Multiply and adder tree is a pipeline process. Notice that in each step, after corresponding rows and columns of A and B going through the pipe, it produces one element of the output matrix.

2.1.1 Design Problems

Consider simple Multiply and Adder Tree design with n element multiplication,

1. How many Multiply units needed for the entire design?
2. Consider an adder stage r, how many adder modules needed for that stage (Assume multiplication stage as stage 0)?

3. If all the inputs to the design represented using k bits, how many bits are needed to represent the final result of the Multiply and Adder Tree?

4. How may Adder modules need for the entire multiply and adder tree design?

5. How many clock cycles need to produce the first output element in the adder tree?

6. How many clock cycles need to multiply two $n \times n$ matrices?

2.1.2 Implementation

Consider only 8-bit arithmetic. All the values are already stored in the BRAM.

1. Implement module **Multiply**, which takes two 8 bit inputs, multiply them together, then register them, and finally pass the registered value as output. (k is a parameter to the design)

2. Implement module **Adder**, which takes two k bit inputs, add them together, then register them, and finally pass the registered value as output. (k is a parameter to the design)

3. Using Generate: For loop (Discussion 2: slide 17) method and Multiply module in part 1, implement a scalable multiplication stage which can handle n inputs (n is a parameter)

4. Using Generate: For loop (Discussion 2: slide 17) method and adder module in part 2, implement a scalable adder tree that takes n inputs and output final result (You may need more than one Generate for loops).

5. Implement module **MulandAddTree** by connecting part3 and part 4.

2.1.3 Post Implementation: Using Xilinx Vivado Software (Use Zynq-7000 xc7z007sclg225-2 FPGA)

A. Consider two 4x4 matrices,

1. Write a testbench which provides two 4X4 matrices to the design and takes output final 4X4 resultant matrix.

2. Simulate the design using the simulator and include the waveform in the report (Clearly indicate the locations where final outputs produced)

3. Elaborate the design and include all the schematics' screenshots of the modules (MulandAddTree, adder, and multiply) in the report.

4. Synthesis the design and include the screenshots like part 3.

5. Generate Resource and timing estimations and include them in the report.

6. Generate power estimation reports and include them in the report.

7. How many of parallel *MulandAddTrees* can be implemented in this FPGA (Provide resource utilization reports with parallel *MulandAddTres*)?

8. Redo part 4,5,6 for 8x8, 16x16 and 32x32 matrices.