Abstract: - Speed and efficiency of sorting algorithms are essential for high speed data processing. FPGA based hardware accelerators show better performance than the general purpose processors. Similarly traditional algorithms may not be always efficient on FPGAs. Sorting networks have come as suitable alternatives which can be implemented on FPGAs efficiently. Each application has its own constraint on latency and throughput. A careful selection of a sorting network with suitable number of pipeline stages performs at higher throughput, without contributing much latency.

Key-Words: - FPGA, Sorting networks.

1 Introduction

With the demand for high speed network and computing, speed and parallel algorithms have become essential tools for development. Many of these operations were performed by a general purpose processor [1]. But now days due to the availability of FPGAs, many researchers try to implement various algorithms on FPGAs more efficiently [2] [3]. FPGAs are often used as hardware accelerators.

One of the commonly used operations in high speed data processing is data sorting. The most commonly used sorting algorithm is Bubble sorting. For efficient and reduced operations implementation of sorting, Batcher proposed a technique of sorting using sorting networks [4] [5] [6]. Many of these are implemented of FPGAs and general purpose processors [7] [8] [9].

In this paper we evaluate various sorting networks based on the complexity and speed, focusing on FPGA implementation. For the analysis purposes all the networks are configured to accept eight unsorted numbers, of eight bit wide each. The result will be the sorted numbers.

2 Sorting Networks

2.1 Bubble Sort

In bubble sort the adjacent pair of data elements are compared and swapped if they are found in wrong order, and this process is repeated until the last two elements of the array are compared. With each pass in the bubble sort, by compare and swap process the smaller elements bubble or move up to their designated locations in the array.

2.2 Shell Sort

Shell Sort is one of the oldest sorting algorithms, which arranges the data sequence in a two dimensional array and then sorts the columns of the array [10], which results in the data sequence being partially sorted. As the process repeats the array becomes narrow. Each time the number of columns will keep decreasing. In the end, the array will have a single column. Shell sort method actually groups the data at each step, rather than sorting the data by itself. At each step, either insertion sort or bubble sort is used to arrange the data. The number of times the data elements need to be rearranged is reduced in this type of sorting method.

Fig.1. Shell Sort, n=8.

2.3 Odd-even transposition sort

In general, odd-even transposition sort compares the adjacent pair of data in an array to be sorted and, if a pair is found to be in the wrong order then those elements are swapped. In the first step, odd index and the adjacent even index elements are compared and are swapped, if found in wrong order. In the next step, even index and the adjacent odd index elements are compared and are swapped, if found in wrong order. This process continues with alternating (odd, even) and (even, odd) phases, until no swapping of data elements are required. Thus the resultant array is a sorted one. This network comprises of the same number of comparator stages as the number of inputs. In each stage either odd or even
index positions are compared with their respective neighbors. Each stage alternates between even and odd.

**Fig.2. Odd-even transposition Sort, n=8.**

### 2.4 Bitonic Sort

Bitonic Sort is a sorting algorithm which is designed for parallel machines. On any arbitrary sequence to be sorted, bitonic sort produces a bitonic sequence by sorting the two halves of the input sequence in opposite directions.

**Fig.3. Bitonic Sort, n=8.**

A bitonic sequence is one, which consists of two sub-sequences, one that monotonically increasing and the other monotonically decreasing. Hence for any arbitrary sequence of length n, in the bitonic sort, first two n/2-element sorts are performed, one increasing and the other is decreasing. This results in an n-element bitonic sequence. This entire sequence is now bitonically sorted to produce a sorted (monotonic) sequence.

### 2.5 Odd-even merge sort

The earlier odd-even transposition sorting algorithm has a complexity of \(O(n^2)\). With such a complexity, for any large sequence with sequence length n the number of steps to perform a complete sort will be very high in real time situations. Odd- even merge sort solves this problem. In odd-even merge sort, all the odd index elements and even index elements are sorted separately and then merged; this step is repeated until we get a completely sorted sequence. Odd-even merge sort is also called as optimal sorting algorithm.

### 3 FPGA Implementation

All the five algorithms are described by Verilog HDL language in two different approaches. One approach is without any pipeline registers and other is with pipeline registers. Each module accepts eight parallel input data of width eight bits each and the clock. The output is the sorted data. The functionality of each of the designed modules is verified by the simulation in ModelSim, as shown in Fig.5.

| TABLE 1
<table>
<thead>
<tr>
<th>SUMMARY OF COMPLEXITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort Network</td>
</tr>
<tr>
<td>Odd-even transposition sort</td>
</tr>
<tr>
<td>Bubble Sort</td>
</tr>
<tr>
<td>Bitonic Sort</td>
</tr>
<tr>
<td>Odd-Even mergesort</td>
</tr>
<tr>
<td>Shellsort</td>
</tr>
</tbody>
</table>

Where \(n\) is the number of inputs.
Each module is then synthesized and implemented on a using Xilinx ISE [11]. Then each module is tested by downloading the bit file to Xilinx Virtex-5 LX50 FPGA.

Each module is then synthesized and implemented on a using Xilinx ISE [11]. Then each module is tested by downloading the bit file to Xilinx Virtex-5 LX50 FPGA.

Table 2 & Table 3 summarize performance of the non pipeline and pipeline stages of designs.

It is observed that pipeline stages based algorithms have better operating frequency than the non pipelined stage algorithms. On the contrary, pipeline stages consume more FPGA resources than the non pipeline stages. The Odd-even merge sort consumes fewer resources and gives optimal operating frequency performance compared to the bitonic sort which not only consumes more resources but also has a poor operating frequency performance. Both modules have a throughput of sorting eight data per clock with a latency of five.

4 Conclusion

It is often critical do decide the best sorting algorithm for a given application. This is based on the tradeoff between pipeline stages, area and speed. It is observed that by adding five pipeline registers for odd even merge sort, throughput can be increased significantly without much increase in area (FPGA resource). To achieve similar performance, other sorting algorithms require more number of pipeline stages.

References: