VHDL Implementation of Merge Sort Algorithm

Abirami.R
B. Tech in Electronics and Communication Engineering Shiv Nadar University, (Uttar Pradesh) India

Abstract— **Designing Digital Circuits is a broad and an interesting field of study. Implementation of digital circuit design to various real world scenarios may solve many day to day problems. VHDL (VHSIC Hardware Description Language) has become a useful tool in software implementation of digital Circuits. This paper presents the implementation of Merge Sort Algorithm using VHDL. The implementation and functionality test of the algorithm is done using ModelSim Edition 10.3 tool and the circuit model is synthesized using Xilinx 14.5.**

Keywords—Merge Sort; VHDL; Behavioral; Synthesis

I. INTRODUCTION

Sorting is fundamental to many computer based applications. The efficiency of sorting algorithm greatly influences the efficiency of the entire system itself. Notably, many algorithms use sorting as a key sub-routine. A crucial step in sorting is deciding the most efficient sorting algorithm specific to that particular application. There are many well established sorting algorithms. Few such algorithms are Bubble sort, Insertion sort, Selection Sort, Merge Sort, Heap Sort and Radix Sort. Bubble Sort is fairly inefficient, in terms of time constraint than other algorithms. All other algorithms outperform one another depending on the set of inputs. Merge Sort, Heap Sort and Radix Sort are desired when there is huge input data. Sorting is also helpful in searching, selection of data values, finding the closest pair of the desired data, calculating frequency distribution of the given data set. It is also helpful in element uniqueness, where in the algorithm can be slightly modified to remove duplicates in the output data values. The VHDL implementation of merge sort algorithm with the circuit synthesis is discussed in this paper. The merits and demerits of merge sort and other sorting algorithms are also discussed in this paper.

II. MERGE SORT ALGORITHM

A. Overview of the Algorithm

Merge Sort is a stable and an efficient sorting algorithm. It is one of the first few algorithms in which optimum speed up was achieved. The Algorithm was first introduced in 1945 by John Von Neumann [1]. Merge sort algorithm is named so because, it proceeds by merging sorted sub arrays together to make a larger sorted array or sub array. Merge Sort use divide, combine and conquer method of sorting. This methodology helps parallelize the sort implementation. Like most sorting algorithms merge sort is also a comparison based algorithm. Merge sort is stable because the implementation preserves the input order of equal or equivalent data values in the output. The merge sort algorithm can be explained in three steps: Divide Step, Conquer Step and Combine Step. In Divide Step, the given input string or array is divided into many smaller sorted sub arrays. The sub arrays can have one or more data elements. In conquer step the data values in two such sub arrays are sorted. In the combine step the two sub arrays are combined. If all the data values are sorted the process ends, if not conquer and combine steps are repeated until all data values are combined into a single sorted array. The following example illustrates the procedure for merge sort. If the unsorted array contains data values say 3, 6, 1, 9, 2, 4, 6, and 0. Initially each data element is treated as a single sub array as in level: 0 of the Fig. 1. The steps mentioned above are repeated in order and the final sorted array is obtained.

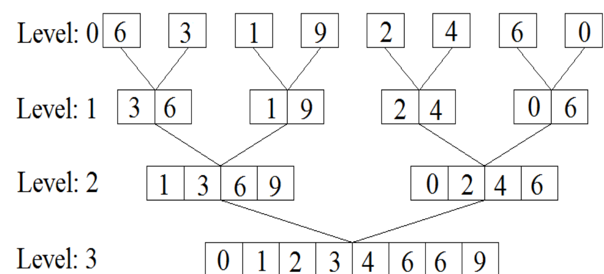

Fig. 1. Merge Sort procedure

The disadvantage of this sorting algorithm is that more memory space is required for storing data values in each level.

B. Comparison of Sorting Algorithms

Selection of sorting algorithm for a specific application can be accomplished by comparing all sorting algorithms. The possible way to analyze the efficiency of a sorting algorithm is by considering the time complexity of the algorithm. Complexity of an algorithm reflects its relative efficiency. Complexity is usually written in the form of Big-O notation. In Big-O notation, O denotes the algorithmic complexity and n denotes the set size for which the algorithm is run against [2]. The following table gives a brief idea about the complexities of various sorting algorithm in its best case and worst case in Big-O notation.

TABLE I.

Sort	**Time Complexity**		
	Average	***Best***	***Worst***
Bubble Sort	O(n^2)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n^2)	O(n)	O(n^2)
Heap Sort	O(n log(n))	O(n log(n))	O(n log(n))
Merge Sort	O(n log(n))	O(n log(n))	O(n log(n))

From the above table, it can be inferred that for large data sets Merge Sort and Heap sort out performs other sorting algorithms. VHDL implementation of heap sort is quite complex than Merge Sort Algorithm.

III. IMPLEMENTATION OF THE ALGORITHM

A. VHDL Code Implementation

The VHSIC Hardware Description Language is a language use to describe hardware design in both abstract and concrete level. In the past decade, the VHDL applications have expanded from a design documentation language to a highly sophisticated design implementation and verification language [3]. VHDL usage has increased tremendously since its inception. VHDL software interface reduces coding complexity and provides a graphical interface of the system. A four data input sorter is implemented in VHDL using merge sort, one of the time efficient algorithm. Into the main VHDL code, comparators of level 1 and level 2 as in Fig.1 is port mapped. Direct recursion cannot be implemented in VHDL hence port mapping is used. Same VHDL program cannot be employed for every level because the number of inputs and outputs keep changing. The VHDL code for the main program is as follows.

```
------------------------VHDL Code for Main Program---------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-------------------------------------------------------------------------------
ENTITY sort IS
PORT
(in1, in2, in3, in4: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
out1,out2,out3,out4: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END sort;
-------------------------------------------------------------------------------
ARCHITECTURE merge OF sort IS

COMPONENT merge1 is
PORT
(in1_1,in1_2: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
out1_1, out1_2: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

COMPONENT merge2 is
PORT
(in2_1,in2_2,in2_3,in2_4: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
 out2_1,out2_2, out2_3, out2_4: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
END COMPONENT;

SIGNAL a,b,c,d: STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN
lev1_1: merge1 PORT MAP (in1, in2, a, b);
lev1_2: merge1 PORT MAP (in3, in4, c, d);
lev2_1: merge2 PORT MAP (a, b, c, d, out1, out2, out3, out4);
END merge;
---------------------------------END-------------------------------------------
```

The port mapped comparator VHDL program for level-1 and level- 2 is as follows.

```
--------------------------VHDL Code for LEVEL-1------------------------
ENTITY merge1 is
PORT
(in1_1,in2_1: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
out1_1,out2_1: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END merge1;

ARCHITECTURE level1 OF merge1 is
BEGIN

PROCESS(in1_1, in1_2) IS
BEGIN
IF (in1_1 <= in1_2) THEN
        out1_1<= in1_1;
        out1_2<= in1_2;
ELSE
        out1_1<= in1_2;
        out1_2<= in1_1;
END IF;
END PROCESS;
END level1;
---------------------------------END-----------------------------------

-------------------------VHDL Code for LEVEL- 2------------------------
ENTITY merge2 IS
PORT (in2_1,in2_2,in2_3,in2_4: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
out2_1, out2_2, out2_3, out2_4: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
END merge2;

ARCHITECTURE level2 OF merge2 IS
SIGNAL flag: std_logic;
BEGIN

PROCESS (in2_1, in2_2, in2_3, in2_4) IS
BEGIN
   IF (in2_1 <= in2_3) THEN
        out2_1 <= in2_1;
   ELSIF (in2_1 > in2_3) THEN
        out2_1 <= in2_3;
        flag <= ' 1 ';
   end if;

   IF (in2_2<=in2_3 and flag<='0') THEN
         out2_2<= in2_2;
         out2_3<= in2_3;
         out2_4<= in2_4;
   ELSIF (in2_2<=in2_4 and flag<='0') THEN
         out2_2<= in2_3;
       out2_3<= in2_2;
       out2_4<= in2_4;
   ELSIF (flag<='0') THEN
       out2_2<= in2_3;
       out2_3<= in2_4;
       out2_4<= in2_2;
   END IF;
```

```
        IF (in2_4<=in2_1 and flag<='1') THEN
                out2_2<= in2_4;
                out2_3<= in2_1;
                out2_4<= in2_2;
        ELSIF (in2_4>=in2_2 and flag<='1') THEN
                out2_2<= in2_1;
                out2_3<= in2_2;
                out2_4<= in2_4;
        ELSIF (flag<='1') THEN
                out2_2<= in2_1;
                out2_3<= in2_4;
                out2_4<= in2_2;
        END IF;

END PROCESS;
END level2;
---------------------------------END-------------------------------------------
--
```

B. Simulation and Synthesis Results

The above VHDL code for Merge sort algorithm is verified and simulated in ModelSim Edition 10.3 tool the simulation results are as follows.

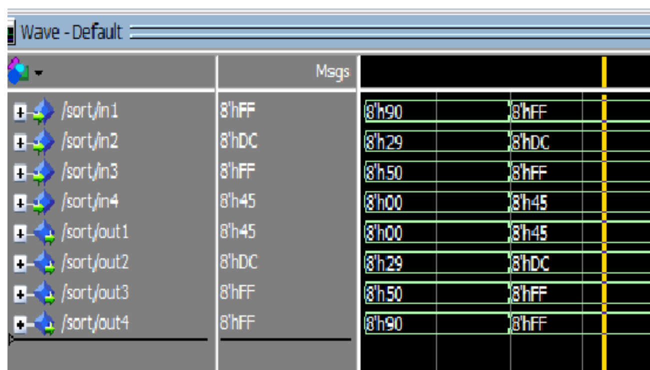

Fig. 2. Simulation Result

ModelSim allows verification of the code by running test bench programs. Test benches are specific to every program. In the test best both inputs and expected outputs are specified. If the output is same as expected output then no error message is shown. Else the mentioned error message will be displayed in the transcript window of ModelSim. The program for merge sort produced expected output when tested. The Synthesis results of the algorithm are obtained from Xilinx 14.5 tool. Fig. 3. shows the high level schematic design

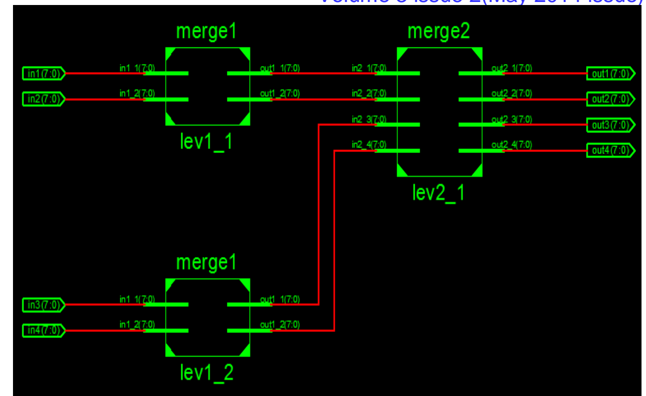

Fig. 3. High Level Design Schema

The digital design of Level- 1 and Level-2 of the merge sort algorithm as discussed earlier is synthesized. The synthesis results of these levels are show in Fig.4. It can be seen that merge sort requires less number of comparators in comparison with bubble sort and selection sort. The number of comparators in the design can be related to the total number of comparisons needed to sort a given set of input.

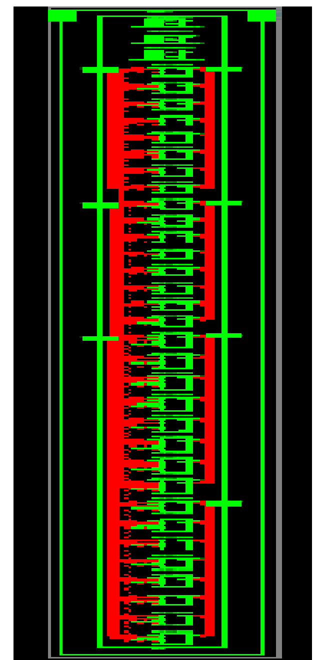

Fig. 4.a. Level-1 Design **Fig.4.b**. Level-2 Design

The design summary of the system is attached below. There were no errors or warnings in the design implementation procedure. Device utilization summary shown below gives the number of devices utilized on the FPGA board and number of devices used.

sort Project Status (04/26/2014 - 20:25:52)			
Project File:	abi.xise	Parser Errors:	No Errors
Module Name:	sort	Implementation State:	Programming File Generated
Target Device:	xc3s50-4cp132	• Errors:	
Product Version:	ISE 14.5	• Warnings:	
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	0 (Timing Report)

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs	24	1,536	1%	
Number of occupied Slices	12	768	1%	
Number of Slices containing only related logic	12	12	100%	
Number of Slices containing unrelated logic	0	12	0%	
Total Number of 4 input LUTs	24	1,536	1%	
Number of bonded IOBs	32	89	35%	
Average Fanout of Non-Clock Nets	2.31			

fIG. 5. SCREEN SHOT OF DESIGN SUMMARY

IV. DISCUSSION OF THE RESULTS OBTAINED

The first 2 level implementation of Merge Sort Algorithm using VHDL is simple, when the logic is extended to higher levels the logic and VHDL coding becomes complex. But once implemented, better results are obtained in comparison with other algorithms. The total circuit delay of the 4 input merge sorter was only 22.3ns unlike other algorithms with comparatively higher circuit delay. The circuitry for merge sort is comparatively smaller than the circuitry required for conventional bubble sort and selection sort algorithms.

V. CONCLUSION

This study helped to understand advantages and disadvantages of various sorting algorithms and implementation of Merge Sort algorithm using VHDL. The results obtained and inferences made reflect the efficiency, dependability and Complexity of Merge Sort digital circuitry.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms," 2nd Ed., Mc Graw-Hill, 2003.

[2] Prabhakar Gupta, Vineet Agarwal, Manish Varshney, "Design and Analysis of Algorithms," 1st Ed., PHI Learning Private Limited, Delhi,2008.

[3] Douglas L. Perry, "VHDL Programming by Example," 4th Ed., Tata McGraw-Hill Education Private Limited, 2002.