In today’s modern scientific world, technological changes发生 with a very fast rate. The rapid growth in financial, commercial, Internet-based applications, there is a huge demand for finding out the devices with low latency, power and area along with there is an increasing desire to allow computers to operate on both binary and decimal floating-point numbers. Accordingly, stipulation for decimal floating-point support is being added to the IEEE-754 Standard for Floating-Point Arithmetic. In this research work, we present the design and implementation of a decimal floating-point adder that is acquiescent with the current draft revision of this standard. The adder supports operations on 64-bit (16-digit) decimal floating-point operands. We provide synthesis results indicating the estimated area and delay for our design when it is pipelined to various depths. High performance computing is strongly required in most applications which deal with floating point Numbers. Most workstations used in these fields are adopting a floating point processing unit to accelerate Performance.

Keywords – Binary, Floating Point, Hardware design

I. INTRODUCTION

Various high level encoding languages have a capability for specifying floating-point numbers. The most frequent technique is to stipulate them by a real declaration statement as conflicting to fixed-point numbers, which are specified by an integer declaration statement. Any computer that has a compiler for handling floating point arithmetic operations. The operations are quite often included in the internal hardware. If no hardware available for operation, the compiler must be designed with a package of floating point software subroutines (program or line of logic written once, uses more than once). Although the hardware method is more expensive, it is so much more efficient than the software method that floating point hardware is included in most computers and omitted only in very small ones[4]. Example of floating point hardware devices are Intel 8231, arithmetic processor and AMD’s AM9512 floating point processor. The AM9512 provides add, subtract, multiply, and divide operation for 32-bit and 64-bit operands. It can easily interface to enhance the computational capabilities of the host CPU.

II. PROBLEM OVERVIEW

The user of computer prepares data with decimal numbers and receives results in decimal form. A CPU with an arithmetic logic unit can perform micro operations with binary data. To perform arithmetic operations with decimal data, it is necessary to convert the input numbers to binary, to perform all calculations with binary numbers, and to convert the results into decimal. This may be an efficient method in applications requiring a large number of calculations and relatively smaller amount of input and output data [14]. But user friendly format in which it allows input output operations in decimal but machine operation in binary, so the input decimal data have to convert in binary and in later part after operation converted back to decimal. But it is quite time consuming.

When the application calls for a large amount of input-output and a relatively smaller number of arithmetic calculations, it becomes convenient to do the internal arithmetic directly with the decimal numbers. Computers capable of performing decimal arithmetic must store the decimal data in binary coded form. The decimal numbers are then applied to a decimal arithmetic unit capable of executing decimal arithmetic micro operations.

III. PROBLEM FORMULATION

We implement divide and conquer approach, in which complex logic operations segmented or implementation into various multiple numbers of arithmetic logic blocks. These blocks works independently or dependently means, output of one is input of others. Then, we optimize whole unit, explore boundaries and tradeoff between speed, power, area.
Electronic calculators invariably use an internal decimal arithmetic unit since inputs and outputs are frequent, not seem to be a reason for converting the displayed results to decimal. Since this process requires special circuits and also takes a longer time to execute. Many computers have hardware for arithmetic calculations with both binary and decimal data. Users can specify by programmed instructions whether they want the computer to perform calculations with binary or decimal data. The unit accepts coded decimal numbers and generates results in the same adopted binary code [14].

IV. RESULT

Synthesis Reports : The corresponding circuit hardware realization is carried out by a synthesis tool.

Simulation Reports : The design descriptions are tested for their functionality at every level – behavioral, data flow, and gate. One has to verify here whether all the functions are carried out as expected and resolve them. All such performance is carried out by the simulation tool. The tool also has an editor to carry out any corrections to the source code. Simulation involves testing the design for all its functions, functional sequences, timing constraints, and specifications.

Table 1: Arithmetic Unit for Double Precision Optimized Arithmetic Hardware Design for Binary & Floating Point Operands: Hardware Resource Utilization Summary targeting on xc4vlx40-12ff1148 device

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Device Parameter</th>
<th>Usage Number</th>
<th>Utilization %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of slices</td>
<td>320</td>
<td>1%</td>
</tr>
<tr>
<td>2</td>
<td>Number of slices Flip-Flop</td>
<td>450</td>
<td>1%</td>
</tr>
<tr>
<td>3</td>
<td>Number of 4-input LUTs</td>
<td>502</td>
<td>1%</td>
</tr>
<tr>
<td>4</td>
<td>Number of IOs</td>
<td>196</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Number of bounded IOBs</td>
<td>196</td>
<td>30%</td>
</tr>
<tr>
<td>6</td>
<td>Area Constraint Ratio</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Total memory usage</td>
<td>281640kb</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Arithmetic Unit for Double Precision Optimized Arithmetic Hardware Design for Binary & Floating Point Operand: Comparison in terms of Area & Delay.

<table>
<thead>
<tr>
<th>Name of Design proposals</th>
<th>Number of bits</th>
<th>Delays (ns)</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of Design proposals</td>
<td>Number of bits</td>
<td>Delays (ns)</td>
<td>Area</td>
</tr>
<tr>
<td>Sreehari [28]</td>
<td>32</td>
<td>8.9</td>
<td>-</td>
</tr>
<tr>
<td>Humberto[30]</td>
<td>32</td>
<td>12.1</td>
<td>256</td>
</tr>
<tr>
<td>Haller [29]</td>
<td>32</td>
<td>10.0</td>
<td>305</td>
</tr>
<tr>
<td>Hwang [27]</td>
<td>32</td>
<td>10.5</td>
<td>82</td>
</tr>
<tr>
<td>Fischer [31]</td>
<td>32</td>
<td>10.3</td>
<td>123</td>
</tr>
<tr>
<td>Subhash [14]</td>
<td>64</td>
<td>-</td>
<td>2325</td>
</tr>
<tr>
<td>Taher[32]</td>
<td>64</td>
<td>11.24</td>
<td>-</td>
</tr>
<tr>
<td>P.Karlstrom [18]</td>
<td>64</td>
<td>-</td>
<td>561</td>
</tr>
<tr>
<td>Our Proposal</td>
<td>64</td>
<td>11.2</td>
<td>320</td>
</tr>
</tbody>
</table>

V. CONCLUSION

This research purposed a mixture of hardware compilation, module generators, Floating point arithmetic and automatic interface generation to improve the efficiency, productivity and flexibility when implementing the floating point design on the FPGA. This dual representation is very valuable as allows for easy navigation over all the components of the units, which allows for a faster understanding of their interrelationships and the different aspects of a Floating Point operation. There are several possibilities for improvements to the system. It would be desirable if the coding strategy let the data path share hardware resources for some operation. This coding strategy thus can save area if it is critical for certain application. The parallelism must now be implemented by the user. It would be better if the compiler itself can detect the dependency to reorganize the data path in which the parallelism can be achieved automatically. Our result is high-quality in terms of area, power, speed and trade-off between parameter this is better explained in comparative view.
VI. REFERENCES


[15] Javier and Thomas “LOP for latency improvement in single data path floating point adder”.

[16] Gerald R. Morris and Viktor K. Prasanna “Pipelined Data path for an IEEE-754 64-Bit Floating-Point Jacobi Solver”, Supported by the United States National Science Foundation under award No. CCR-0311823 and in part by award No. ACI-0305763.


