AN EFFICIENT ARCHITECTURE FOR PARALLEL ADDERS

CH. CHIMPIRAIAH¹, E. V VIJAY²

¹Dept of electronics and communication engineering, Gudlavalleru Engg College, AP, India
²Assistant professor, Dept of electronics and communication engineering, Gudlavalleru Engg College, AP, India

chintham.486@gmail.com, evvijay@yahoo.co.in

ABSTRACT

Binary addition is the crucial operation in digital circuit design. This paper presents an efficient structure for Parallel adders. This structure and fast performance makes them particularly attractive for VLSI implementations. Area and Delay are key parameters for VLSI designs. The performance of proposed adder gives the better Delay performance Compare to Ripple carry adder, Carry look ahead adder, carry tree adders (kogge-stone adder, sparse kogge-stone adder, spanning tree adder). Key words—Adders, Area, Delay, Parallel adder

1 INTRODUCTION

Binary adders are one of the most basic and widely used arithmetic operations in modern integrated circuits. They tend to play a critical role in determining the performance of the design. Arithmetic operations are the regular common operations in digital integrated circuits. The simplest circuit adds, subtracts, and multiplies or anything. The computation should be fast and the area consumed by the arithmetic units should be small. These are the two basic requirements for any adder.

Area and Time consumed by the circuit are the basic and important requirements. Numbers can be represented in digital circuits in various ways. Hence, developing efficient adder architecture is crucial to improving the efficiency of the design. Generally ripple carry adder uses for binary addition. After the design of ripple carry adder several techniques are used for the computation of parallel adders. Carry look ahead adders are based on parallel prefix computation gives the better performance than ripple carry adder. After many years research continuous to be focused on improving the delay performance of the adder. As such, extensive research continues to be focused on improving the delay performance of the adder. Next, Brent and Kung (BK) design parallel prefix-computation graph In an area-optimal way and the kogge-stone (KS) architecture is optimized for timing. This architecture is proposed based on KS and BK structures. In this paper propose a new design of an efficient structure for parallel adders. Our proposed adder shows marginally faster performance than the regular kogge-stone adder with area saving.

II RIPPLE CARRY ADDER DESIGN

Ripple carry adder is a digital circuit it adds two binary numbers in parallel form. It contains of full adders connected in a chain. The output carry of each full adder is connected to the input carry of next full adder. But, computation of the carry-in signals at every bit is the most critical and time consuming process. Let a, b are the two input signals and c, is the corresponding carry-in signal to produce the sum(S) and carry(C) outputs.

\[ s = (a \oplus b \oplus c) \]

(1)

\[ c = (a \text{ and } b) \text{or} (b \text{ and } c) \text{or} (a \text{ and } c) \]

(2)

Carry look ahead Adder: In carry look ahead adder, the focus is to design a circuit which can efficiently compute the \( n \) carry values. In every given bit positions generate(G) and propagate(P) signals are compute. The carry look ahead adder uses two special signals Generate(G) and Propagate(P) to predict the Propagation of a Carry signal without using a carry chain.

\[ G = (a \text{ and } b) \]

(3)

\[ P = (a \text{ xor } b) \]

(4)

III CARRY-TREE ADDERS DESIGN

It is extends from the idea of carry look ahead Computation’s Classes of Parallel carry look ahead schemes are formed targeting at high performance applications.

These are also called recurrence solver adders. Different Parallel schemes are formed (a) kogge-stone adder (b) sparse Kogge-stone adder (c) spanning tree adder. Parallel-prefix adders are the variation of well known carry look ahead adders. The difference between CLA and PPA is for the generation of carry signals. In PPA prefix operator is introduced and it is used for the carry signal generation.
\[(g_{11},p_{11})o(g_{22},p_{22}) = (g_{1} + p_{1}.g_{2}, p_{1}.p_{2}) \quad (5)\]

Taken the given inputs

Pre-calculation of \( pi, gi \) equations

Pre-fix graphs can be used to perform calculation of carry values

join \( Ci \) and \( Pi \) for every stage to generate the sum bits

Final sum

Fig 1: Three stage structure for carry-tree adder

In this equation is applied on two pairs of bits \((g_{11}, p_{11})\) and \((g_{22}, p_{22})\). These pairs represent generate and propagate signals used in addition. The output of the operator is a new pair of bits generated in equation (5). With the use of this operator a prefix network is constructed and the second stage of the adder is implementation of this network. The structure of the Prefix network determines the type of prefix adder. These operators can be combined in different ways to form various adder structures.

The key advantage of the tree structured adder is that the critical path due to the carry delay is on the order of \( \log_{2}N \) for an \( N \)-bit wide adder. The arrangement of the prefix network gives rise to various families of adders. First the focus is on the Kogge-Stone adder which has minimal logic depth and fan-out.

Kogge-stone adder design:

Step1: First to generate propagation and generation signals for each bit.
Step2: To generate black cell and gray cell equations.
Step3: In each step gray cells are generated.
Step4: By using gray cell equation to generate the carry bits directly.
Step5: To combine propagation and carry bits for each step to generate sum. Here we designate BC as the black cell which generates the ordered pair in equation (5); the gray cell (GC) generates the left signal only. The interconnect area is known to be high.

Another carry-tree adder is known as the sparse kogge-stone adder. The sparse kogge-stone adder uses ripple carry and kogge-stone adder structure. This design terminates with 4-bit RCA and it uses less number of carry values in the method of KS structure. It allows a large adder to be composed of small adders by generating intermediate carries quickly. Power and area of the carry generation is improved and routing congestion is substantially reduced but delay is increase compare to kogge-stone adder. This adder gives intermediate results between ripple carry adder and kogge-stone adder.

Third type of carry-tree adder is known as the spanning tree adder. This design also terminates with 4-bit RCA. It uses minimum no of multi input gates. It gives better results compared to Sparse Kogge-stone adder.

IV ADDER SELECTIONS

Ripple carry adder
Carry-look ahead adder
Kogge-stone adder
Sparse kogge-stone adder
Spanning tree adder
Brent-kung adder

These adders were selected and these are implemented with bit sizes of 16, 32 bits. This variety of sizes gives the performance of adders with area and delay values. The performance of these adders are compared with area, delay, no of logic levels, total no of paths, no of look up tables. The synthesis values are summarized in the given table 1.

V PROPOSED ADDER DESIGN

The proposed adder is based on the combination of kogge-stone (KS) and Brent-kung (BK) Adder. Our proposed adder is compared with kogge-stone adder. This adder gives the better results compare to kogge-stone adder and
of logic levels are also reduced one compared to kogge-stone adder. The proposed adder block diagram is shown in figure 2.

VHDL hardware description language is use to model for each adder. The top level entity for each design is the adder which is composed of three components. The generator components carry signal component, final sum generator component. The Xilinx ISE Foundation version 9.2i software use for simulation and synthesis.

Fig 2: Proposed adder

RTL SCHEMATIC DIAGRAM

Fig 3: RTL diagram

SIMULATION RESULTS FOR PROPOSED ADDER.
V1 RESULTS

Table 1 contains the synthesis results for proposed adder. Proposed adder uses no of slices, no of LUTS and no of bounded IOBs. Table 2 contains the results for ripple carry adder, carry-look ahead adder, carry-tree adders and our proposed adder. The area and delay result for each adder is obtained. In the table area is measured in slices and delay values are measured in nanoseconds (ns).

DEVICE UTILIZATION SUMMARY:

<table>
<thead>
<tr>
<th>Logic utilization factor</th>
<th>No of sources used</th>
<th>No of sources available</th>
<th>Percentage utilization factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices</td>
<td>145</td>
<td>4656</td>
<td>3%</td>
</tr>
<tr>
<td>Number of LUTS</td>
<td>252</td>
<td>9312</td>
<td>2%</td>
</tr>
<tr>
<td>Number of IOs</td>
<td>443</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of bonded IOBS</td>
<td>443</td>
<td>190</td>
<td>233%</td>
</tr>
</tbody>
</table>

Table 2: AREA and DELAY Comparisons of Ripple carry adder, carry-look ahead adder, KS and proposed approach

<table>
<thead>
<tr>
<th>Adder name</th>
<th>Area(slices)</th>
<th>Delay(ns)</th>
<th>No of logic levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripple carry adder 16-bit</td>
<td>17</td>
<td>31.81</td>
<td>16</td>
</tr>
<tr>
<td>Ripple carry adder 32-bit</td>
<td>37</td>
<td>45.79</td>
<td>33</td>
</tr>
<tr>
<td>Carry-look ahead adder 16-bit</td>
<td>35</td>
<td>31.36</td>
<td>16</td>
</tr>
<tr>
<td>Carry-look ahead adder 32-bit</td>
<td>73</td>
<td>43.25</td>
<td>33</td>
</tr>
<tr>
<td>Sparse kogge-stone adder 16-bit</td>
<td>47</td>
<td>16.58</td>
<td>10</td>
</tr>
<tr>
<td>Sparse kogge-stone adder 32-bit</td>
<td>109</td>
<td>17.61</td>
<td>11</td>
</tr>
<tr>
<td>Spanning tree adder 16-bit</td>
<td>41</td>
<td>15.88</td>
<td>10</td>
</tr>
<tr>
<td>Spanning tree adder 32-bit</td>
<td>101</td>
<td>16.05</td>
<td>11</td>
</tr>
<tr>
<td>Kogge-stone adder 16-bit</td>
<td>81</td>
<td>12.27</td>
<td>8</td>
</tr>
<tr>
<td>Kogge-stone adder 32-bit</td>
<td>192</td>
<td>13.85</td>
<td>8</td>
</tr>
<tr>
<td>Proposed adder 16-bit</td>
<td>63</td>
<td>11.04</td>
<td>7</td>
</tr>
<tr>
<td>Proposed adder 32-bit</td>
<td>145</td>
<td>12.17</td>
<td>7</td>
</tr>
</tbody>
</table>

V11 CONCLUSION

In this paper presents an efficient architecture for parallel adders. The results obtained that the performance of proposed adder gives the best performance compared to kogge-stone adder. This adder gives the better area, delay values and no of logic levels compare to previously used adders.

REFERENCES