A NOVEL DESIGN PROCEDURE FOR INCREASED CONFIGURABLE LOGIC BLOCK CAPACITY IN FPGAS

Siddalingesh S. Navalgund[1], Bairu K. Saptalkar[2],
Dr. Mrityunjaya V. Latte[3]

[1,2] Department of Electronics and Communication Engineering
Shri Dharmasthala Manjunatheshwara College of Engineering and Technology
Dharwad, Karnataka, (India)
[3]Principal, JSS Academy of Technical Education
Uttarahalli-Kengeri Road, Bengaluru, Karnataka, (India)

ABSTRACT

Among the three design styles in VLSI, the gate array design style is relatively simple. This simplicity is gained at the cost of rigidity imposed upon the circuit both by the technology and the prefabricated wafers. The number of SRAM locations used to implement a Look-up Table (LUT) based Configurable Logic Block (CLB) plays a crucial role. In this research work, an attempt is made to address the issue of optimizing the utilization of LUTs such that more number of functionalities can be accommodated. Thus this novel procedure reduces the number of LUTs/CLBs/slices required for various functionalities to be implemented. Theoretical calculations clearly show the extent of optimization achieved. The results obtained are very encouraging.

Keywords: Configurable Logic Block, Look-Up Table, Fpgas, Optimization.

I INTRODUCTION

The Field-Programmable Gate Array (FPGA) architecture mainly consists of two parts: the logic blocks and the routing network [1]. A logic block has a fixed number of inputs and one output. A wide range of functions can be implemented using a logic block. Given a circuit to be implemented using FPGAs, it is first decomposed into smaller sub-circuits. Each of the sub-circuits can be implemented using a single logic block. There are two types of logic blocks. The first type is based on Look-Up Tables (LUTs), while second type is based on multiplexers.

Look-up table based logic blocks: A LUT based logic block is just a segment of RAM. A function can be implemented by simply loading its LUT into the logic block at power up. If function f = A'BC + AB'C' needs to be implemented, then its truth table is loaded into the logic block. In this way, on receiving a certain set of inputs, the logic blocks simply 'look up' the appropriate output and set the output line accordingly. Because of the reconfigurable nature of the LUT based logic blocks, they are also called the *Configurable Logic Blocks (CLBs).* It is clear that $2^{I_{max}}$ bits are required in a logic block to represent a I_{max} bit input, 1-bit output combinational logic function. Obviously, logic blocks are only feasible for small values of I_{max}.. Typically, the value of I is 5 or 6. For multiple output and sequential circuits the value of I_{max} is even less [1].

Multiplexer based logic blocks: Typically a multiplexer based logic block consists of three 2-to-l multiplexers and one two-input OR gate. The number of inputs is eight. The circuit within the logic block can be used to implement a wide range of functions. One such function, shown in Fig. 1(a) can be mapped to a logic block as shown in Figure 1(b). Thus, the programming of multiplexer based logic block is achieved by routing different inputs into the block.

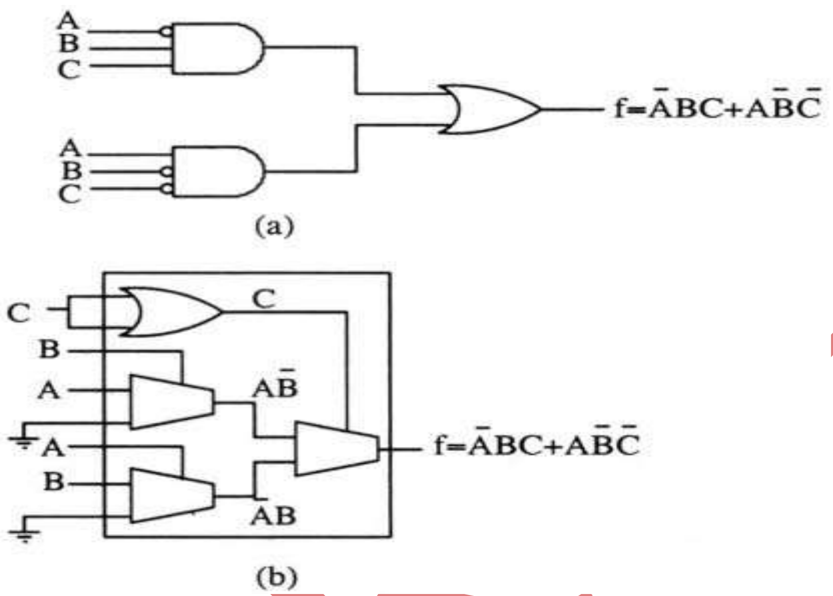

Fig. 1. A logic function mapped to a mux based logic block

1.1 Related Work

Satwant Singh et. al. [2] explores the effect of logic block architecture on the speed of a FPGA. Four classes of logic block architecture are investigated: NAND gates, multiplexer configurations, lookup tables, and wide-input AND-OR gates. An experimental approach is taken, in which each of a set of benchmark logic circuits is synthesized into FPGA's that use different logic blocks. The speed of the resulting FPGA implementations using each logic block is measured. While the results depend on the delay of the programmable routing, experiments indicate that five- and six-input lookup tables and certain multiplexer configurations produce the lowest total delay over realistic values of routing delay. The primary reason is that these blocks can implement typical logic using the fewest levels of logic blocks, and thus incur a small number of stages of the slow programmable routing present in all FPGA's. The secondary reason is that their inherent combinational delay is not excessive. The fine grain blocks, such as the two-input NAND gate, exhibit poor performance because these gates require many levels of logic block to implement the circuits and hence require a large routing delay [2].

Thus it can be seen that the amount of logic which is necessary for a configurable logic block is crucial for the performance of the FPGAs [3][4][5]. It is true that the FPGAs must be selected in such a way that the least number of CLBs must be used, while mapping a design, it is interesting to look into the aspects of the size of CLBs for specific design cases. i.e. if the design consists of majority of similar blocks (example, half adders, full adders, multiplication units etc.), then an attempt can be made to theoretically find the minimum capacity of CLBs necessary[6][7][8].

II PROBLEM FORMULATION

In a 5-input CLB, all 2^5 combinations have to be stored. The problem definition is to suggest a method which stores only those entries which generate either a logic 0 or logic 1 output, whichever is greater without loss of functionality [1]. If the scope of problem is expanded, then the above problem instance can be made to store the relevant values of logic 0 and/or logic 1 for a total of 2^n combinations of outputs corresponding to a given function.

This problem instance is clearly a case of optimization, where the objective function is to reduce the number of storage components used. i.e.

$J = \min(C_i); i = 1 \text{ to } n$……………………………………………………………………………………….(1)

Where C_i indicates the capacity of ith CLB. 'n' indicates the total number of CLBs considered in the design.

III PROPOSED SOLUTION METHODOLOGY

The proposed solution methodology consists of analyzing the functionalities of basic computational structures like gates, adders etc. to evolve the method to implement more complex functionalities.

TABLE 1. TRUTH TABLES OF BASIC GATES

Input		Output					
A	**B**	**AND Gate**	**OR Gate**	**NAND Gate**	**NOR Gate**	**XOR Gate**	**XNOR Gate**
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

Case 1: Analysis of AND gate: It can be easily seen that the number of times a logic 0 occurring is 3 out of 4 combinations. In terms of percentage, it is 75%. Hence it may be chosen to be stored in a CLB, so that the CLB need not have all those combinations which map to logic 0 output.

Case 2: Analysis of OR gate: For OR gate, the number of times a logic 1 occurring is 3 out of 4 combinations. In terms of percentage, it is 75%. Hence it may be selected to be stored in a CLB. This implies that the CLB need not have all those combinations which map to logic 0 output.

Case 3: Analysis of NAND gate: By referring to the Table I, it can be easily seen that the number of times a logic 1 occurring is 3 out of 4 combinations. In terms of percentage, it is 75%. Hence it may be chosen to be stored in a CLB, so that the CLB need not have all those combinations which map to logic 1 output. This equals the number of occurrences of logic 1 output in OR gate, however, the input combinations are different for the production of logic 1 output.

Case 4: Analysis of NOR gate: In a NOR gate, the logic 0 occurs 3 times. Hence only those input combinations need to be mapped for the purpose of storing the logic in CLBs.

Case 5: Analysis of XOR gate: In the case of XOR gate, the percentage of occurrences of logic 0 output and logic 1 output are equal. Hence one may not be able to achieve higher optimization for storing the output in CLBs.

Case 6: Analysis of XNOR gate: The argument as in case 5 holds good here as well, except for the input combinations producing the logic 0 and logic 1 outputs.

Case 7: Analysis of Full Adder: Table 2 provides the truth table of a single-bit full adder.

TABLE 2. TRUTH TABLE OF FULL ADDER

Input			Output	
A	B	C_{in}	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the simplest form, for a 1-bit full adder, the capacity of the CLB is 8 bits. However, a closer examination of the output pattern reveals that since there are two outputs, only 4 combinations of sum and carry exist. In addition to this, the patterns repeat. Table 3 provides the occurrences of these patterns.

TABLE 3. FREQUENCY OF OCCURRENCES OF OUTPUT PATTERNS IN FULL ADDER

Output combinations		Frequency of occurrence (in terms of no. of times)
Sum	Carry out	
0	0	1
0	1	3
1	0	3
1	1	1

Thus, a novel design procedure to increase the capacity or CLBs is suggested in this work. Since there are 4 unique combinations of output patterns, it is sufficient to have a 2-to-4 decoder. The modified truth table is shown in table 3.

TABLE 4. MODIFIED TRUTH TABLE WITH OPTIMISED DECODER.

Input			Decoder Outputs		Output	
A	B	C_{in}	x	y	Sum	Carry out
0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

Thus it is proposed that the single-bit full adder implementation after using the optimized decoder results in reduced number of locations in CLBs for storing the outputs. Fig. 2 shows the block diagram of the optimized decoder.

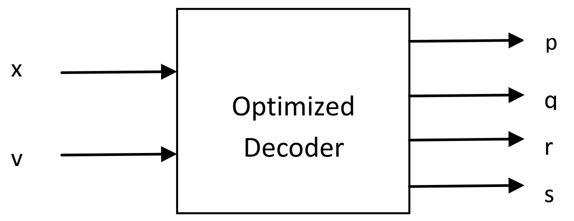

Fig. 2. Block diagram of the optimized decoder

The following Table 5 gives the comparison.

TABLE 5. COMPARISON OF CAPACITY OF CLBS FOR FULL ADDER.

Full adder	No. of locations in CLB required
Without optimized decoder	8
With optimized decoder	4

It is easy to observe a 50% difference in the CLB requirement.

Case 8 : Issues in Implementation of Carry Look-ahead Adder

A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits. Carry-look ahead adder design is a way of reducing the complexity of this ideal, but impractical, arrangement by hardware sharing among the various look ahead circuits. Comparisons can be drawn between CLA and a simpler, but usually slower, ripple carry adder for which the carry bit is calculated alongside the sum bit, and each bit must wait until the previous carry has been calculated to begin calculating its own result and carry bits. The carry-look ahead adder calculates one or more carry bits before the sum. This reduces the wait time to calculate the result of the larger value bits. Fig. 3 shows i^th stage full adder with Propagate signal (P_i) and Generate signal (G_i).

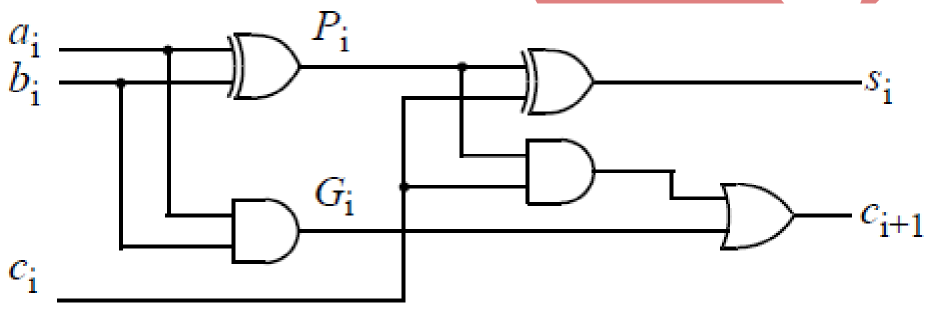

Fig. 3. shows i^th stage full adder with P_i and G_i

Two signals namely Propagate (P) and Generate (G) are used. The signals P_i and G_i are given by:

$P_i = A_i$ XOR B_i……………………………………………………………………………..…(2)

$G_i = A_i$ AND B_i……………………………………………………………………………….(3)

The output sum and carry can be defined as:

$S_i = P_i$ XOR C_i……………………………………………………………………………….(4)

$C_{i+1} = G_i + P_iC_i$……………………………………………………………………………….(5)

G_i is known as the carry Generate signal since a carry (C_i+1) is generated whenever G_i=1, regardless of the input carry (C_i). P_i is known as the carry propagate signal since whenever P_i =1, the input carry is propagated to the output carry, i.e., C_{i+1}. = C_i (note that whenever P_i=1, G_i=0).

Computing the values of P_i and G_i only depend on the input operand bits (A_i& B_i) as clear from the equations.

The Boolean expression of the carry outputs of various stages can be written as follows:

$C_1 = G_0 + P_0.C_0$

$C_2 = G_1 + P_1.C_1 = G_1 + P1.G_0 + P_1.P_0.C_0$

$C_3 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.C_0$

$C_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3P_2.P_1.G_0 + P_3P_2.P_1.P_0.C_0$

TABLE 6. TRUTH TABLE OF A 2-BIT CARRY LOOK-AHEAD ADDER

Inputs		**Outputs**
Y_1Y_0	X_1X_0	$C_{out}S_1S_0$
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

TABLE 7. FREQUENCY OF OCCURRENCES OF OUTPUT PATTERNS IN FULL ADDER

Output combinations			**Frequency of occurrence (in terms of no. of times)**
C_{out}	S_1	S_0	
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	3
1	0	1	2
1	1	0	1
1	1	1	0

The block diagram of a simple 4-bit CLA is shown in Fig. 4. The carry look-ahead network calculates the P_i and G_i terms.

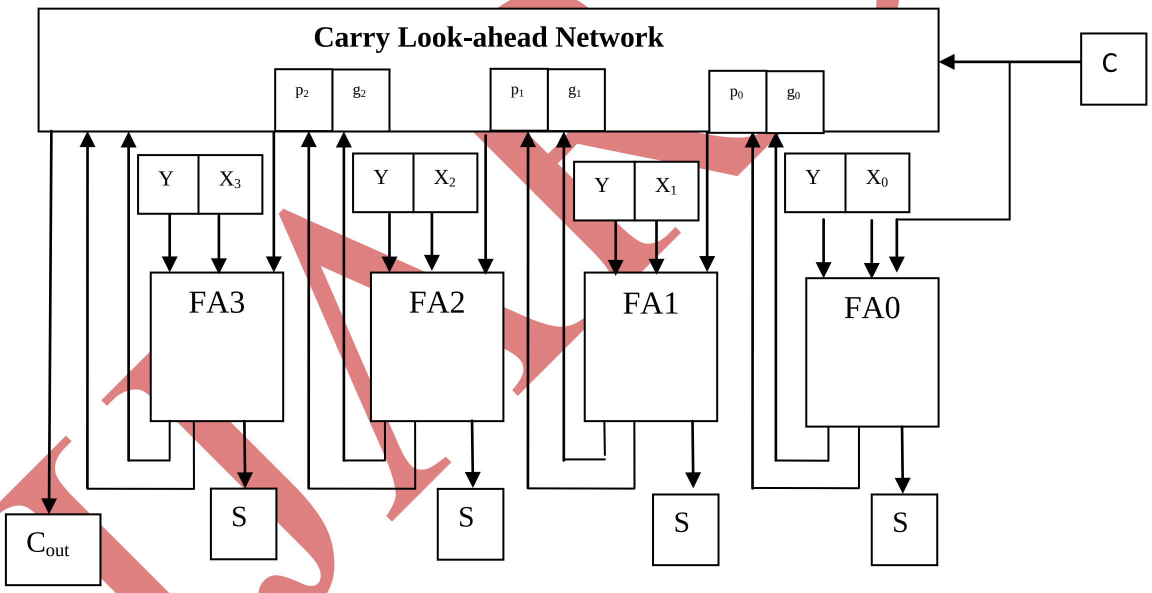

Fig. 4. Shows the Block Diagram of A 4-Bit CLA

IV RESULTS AND DISCUSSIONS

The optimization performed on the basic logic gates shows encouraging results. Table 8 provides a comprehensive summary of the theoretical results for various basic gates. It is interesting to note that for a 2-bit CLA, the output is never driven to "111" ($C_{out}S_1S_0$) combination. Hence the gain is 56.25%. Further, as the number of inputs is increased, the gain in terms of reduced number of locations in CLBs is increased exponentially.

TABLE 8. SUMMARY OF THEORETICAL CALCULATIONS

Function	No. of inputs and bits per input	No. of locations in CLB required		Gain in terms of Reduced no. of locations in CLBs
		Without address mapper	With address mapper	
AND	(2,1)	4	2	50%
OR	(2,1)	4	2	50%
NAND	(2,1)	4	2	50%
NOR	(2,1)	4	2	50%
XOR	(2,1)	4	2	50%
XNOR	(2,1)	4	2	50%
Full Adder	(3,1)	8	4	50%
CLA	(2,2)	16	7	56.25%

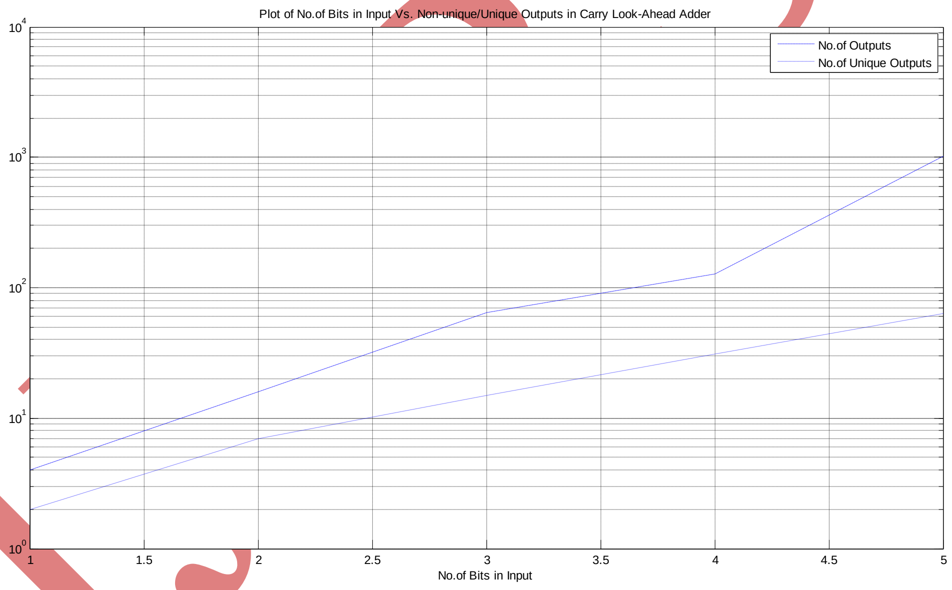

Fig. 5. Plot of variation in number of bits in inputs versus non-unique/unique outputs in CLA

The plot of number of bits in the input versus the gain in capacity of LUT in a CLB for carry look-ahead adder is shown in fig.6 on the following page. For a 2-bit,3-bit,4-bit and 5-bit input implementations, the gains are 50%, 56.25%, 76.53%, 75.78% and 93.85%. With the increasing number of input bits, it is easy to see that the gain values increase exponentially.

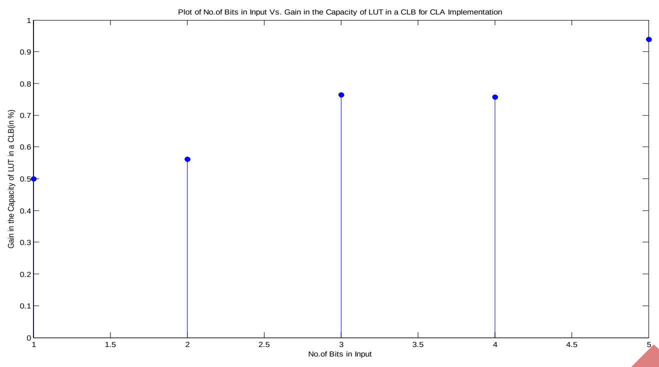

Fig. 6. Plot of number of bits in the input Vs. gain in capacity of LUT in a CLB for carry look-ahead adder

V CONCLUSIONS

It may be observed that with careful analysis of the outputs of the digital circuits, a certain level of optimization can be achieved. This optimization in hardware when used with the design optimization results in smaller and faster implementations. The views presented in this work may be extended with suitable modifications to any other functionality as well. Though the generalization may be difficult, the same line of thinking may be applied to domain-specific problems so that specific FPGA architectures may be evolved for better performance parameters.

The carry look-ahead adder (CLA) related issues are briefed. The results show a small dip in gain (in %) for the number of input bits equal to 4. This can be attributed to the less number of unique outputs for the CLA. Hence, one can also conclude the dependence of gain on the number of input bits.

VI ACKNOWLEDGMENTS

The authors thank the Management, the Principal/Director, Deans, HOD(ECE) and Staff of Sri Dharmasthala Manjunatheshwara College of Engineering and Technology, Dhavalgiri, Dharwad, Karnataka, India for encouraging us for this research work.

REFERENCES

[1]Naveed Sherwani, "*Algorithms for VLSI Physical Design Automation*", Kluwer Academic Publishers, 1999.

[2]Satwant Singh, Jonathan Rose, Paul Chow, David Lewis, "*The Effect of Logic Block Architecture on FPGA Performance*", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 3, March 1992, 0018-9200/92, 1992 IEEE, pp 281-287.

[3]Xilinx, "*Virtex-5 FPGA XtremeDSP Design considerations User Guide*", UG193 (v3.4), June 1, 2010.

[4]Vaughn Betz, Jonathan Rose, "How Much Logic Should Go in an FPGA Logic Block?", *IEEE Design & Test of Computers*, 0740-7475/98, © 1998 IEEE, pp 10-15.

[5]Alberto Sangiovanni-Vincentelli, Abbas El Gamal, Jonathan Rose, "Synthesis Methods for Field Programmable Gate Arrays", *Proceedings of The IEEE*. Vol. 81. No. 7. July 1993, 0018-9219/93, IEEE, pp 1057-1083.

[6]Stephen E. Richardson, "Exploiting Trivial and Redundant Computation", Proceedings of the 11th Symposium on Computer Arithmetic, edited by Swartzlander, Irwin, and Jullien, *IEEE Computer Society Technical Committee on VLSI*, Ontario, June 29–July 2, pp. 220–227,1993.
[7]Elias Ahmed, Jonathan Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density", *FPGA* 2000, Monterey CA USA, Copyright 2000 ACM 0-89791-88-6/97/05.
[8]Ian Kuon, Jonathan Rose, "Area and Delay Trade-offs in the Circuit and Architecture Design of FPGAs", Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada M5S 3G4., FPGA'08, February 24–26, 2008, Monterey, California, USA, Copyright 2008 ACM978-1-59593-934-0/08/02, pp 149-158.