RESEARCH ARTICLE OPEN ACCESS

Design Optimization of the 64-Bit Carry Look-Ahead Adder Based on FPGA and Verilog HDL

Isaack Adidas Kamanga*

*(Department of Electronic and Telecommunication Engineering,
Dar es Salaam Institute of Technology (DIT), Dar es Salaam-Tanzania
Email: isaackkamanga@gmail.com)

--************************----------------------------------

Abstract:

Adders are very useful electronic circuits for performing additions in different electronic devices. Adders can be found in computers, Digital Signal Processors (DSPs), graphic processors, and microprocessors. There exist different adder designsand sizes. Different sizes can handlea different number of bits at once. There are different adder topologies such as Ripple Carry Adder (RCA), Carry Save Adder, Carry Look-Ahead Adder (CLA), Carry Increment adder, Carry Skip Adder, Carry Bypass Adder, and Carry Select Adder. Fabrication area, power consumption, and critical path delay are the main design criteria for the improvement of adder circuits. This article presents the synthesis process and algorithm requirements for design improvements of the existing CLA. This study was conducted to improve these three factors by the use of FPGA and Verilog HDL. FPGAs have reduced sizes, improved delay performances, and millions of logic gates in compact areas by VLSI technology. With Verilog HDL the study managed to use short instructions skipping unnecessary iterations to further improve delay performance. In this study, an improved 64-Bit CLA is designed and compared to the same-sized RCA and CLA designs. A Vivado HLX environment is used to simulate the design. ZYNQ-7ZC702 board is used to simulate the FPGA. The proposed design is compared to that of RCA and CLA. Improved results have been observed with 13.1% reduced area, 5% reduces power consumption, and 14.03% system delay performance improvement over CLA.

Keywords —**Adder, Carry-Look ahead adder, Field Programmable Gate Array (FPGA), Verilog, computational time**

--************************----------------------------------

I. INTRODUCTION

The emerging of portable devices continues to demand small sized on board circuits parts such as adders, substractor, encoders and multiplexers. Studies are being conducted so as to meet the demand of small design area, more efficient power consumption while maintaining high processing speed [3], [4]. Adders are the building blocks of the most computational electronic circuits. Most of the electronic circuits such as multiplexers and de-multiplexers, encoders and de-coders, and substractor, can be implemented by using adders, look [1], [3], and [9]. Apart from doing addition, adders can be connected with addition of other parts to perform subtraction, multiplication and division [1], [9]. According to [5], using same electronic circuit to build different types of circuits has an theadvantage of minimizing the overall cost and easy the design and fabrication process. The adder chips manufactured by one vendor can be used by several manufacturers of other electronic devices.

Adder processing time is very crucial in designing high-performance systems. Delay can be caused by a number of logic gates building the adder and the number of input signals to the adder [6]. The latter cannot be avoided, however, the latter can be minimized with the help of hardware description language (HDL). By HDL, the logic operations can be accomplished by programming the specialized programmable device called Field Programmable Gate Arrays (FPGA) [6].

In this article, an improved 64-Bit carry look-ahead adder referred to as Optimized CLA (OCLA) in this article, is presented so as to reduce computation time, reduce design area, and reduce power consumption. The proposed design intends to design an improved system delay performance, design area, and reduced power consumption. The proposed design is compared to the same-sized RA and CLA designs in this article. A Vivado HLX design suite and ZYNQ-7ZC702 evaluation board were used to realize the design.

This article consists of five parts. part 1 provides the introduction to the proposed study and the contributions in the field of electronics. Furthermore, part 1 enlightens the keywords used in this article and the problem statement. Part 2 provides a literature review on different adder designs and the research gap provided by each. Part 3 discusses the methodology employed. Part 4 shows the experimental results and discussion. Finally, the conclusion is provided in part 5. Also, an acknowledgment of key contributors toward successfully of this work and a list of references used have been provided after the conclusion section.

A. *Keywords*

In the electronic field, an adder is a circuit that performs summation [1], [3], [7]- [9]. They are also referred to as summers [10]. There are two types of adders namely half adder and full adder. Adders come in ICs such as the 4008 IC. The desired adder length can be achieved by concatenating several ICs. Half adder takes in two inputs and returns "sum" and "carry-out" [10]. Full adder takes in "carry-in" in addition to half adder's inputs to give "sum" and "carry-out" [10]. According to [12] and [13], there are six adder topologies namely Ripple Carry Adder (RCA), Carry Save Adder (CSA), Carry Look-Ahead Adder (CLA), Carry Increment adder (CIA), Carry Skip Adder (CSA), and Carry Select Adder. Carry look-ahead adder minimizes time wastage in waiting for "carry-out" generation and propagation in lower stages by computing all "carries" at once. Adders can be implemented using VLSI as well as FPGAs. Field programmable gate arrays(FPGAs) are programmable digital ICs in which logic operations can be described by an HDL language. There are two kinds of hardware description languages (HDL) namely VHDL and Verilog. Verilog is based on the C programming language making its syntax easy and fast. Using FPGA and Verilog has been observed to minimize the time between accepting inputs and providing output referred to as computational time.

B. *Problem statement*

According to a literature survey done on various adder topologies [12 & [13], design parameters [1]-[15], there is a need to continue improvement [8]. Furthermore, with the current trend in technological advancement, there is a need to optimize design area, computational time, and power consumption.

II. LITERATURE REVIEW

A. *Adders*

Half adders (HA) are the basics, two can be combined to form a full adder (FA) [11], see Figure 1. Out of six adder topologies, each topology has its advantage and disadvantage [12].

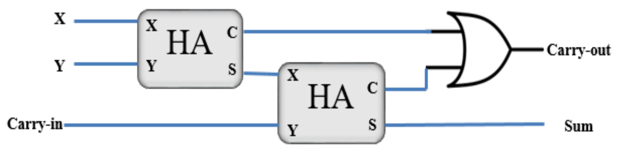


Fig. 1 Using two half adders to build a full adder

1) Ripple carry adder:Fig. 1 shows a single-bit FA block diagram. n FAs can be cascaded to add two numbers (X&Y) of n bit size [10] (see Fig. 2). Figure 2 shows a generalised n-bit ripple carry adder, the name ripple is due to the fact that the carry-out of the lower adder stage becomes the carry-in of the adjacent higher stage. RCA offers a simple design at the expense of serious delay. This delay is highly due to carry propagation delay from lower adder levels [2], [4] & [6]. In design, the first FA in n-bit RCA may be replaced by a HA as there is no carry-in when we start to add two bits.

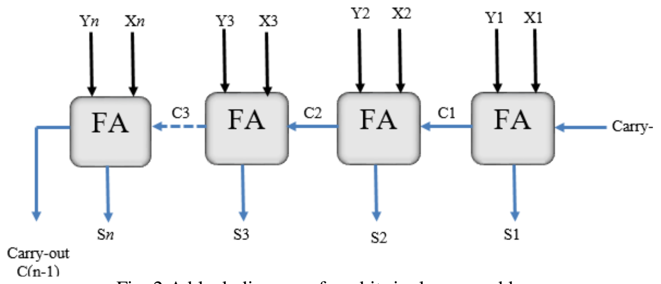


Fig. 2 A block diagram of n -bit ripple carry adder

The worst-case delay Wd of an n-bit RCA can be computed by Equation 1.

$$Wd = (n-1)D_{FA} + D_{HA} \qquad (1)$$

The design of RCA is simple, and fast with an optimized design area however, the delay is the most drawback with regards to communication speed needed in most applications.

2) Carry save adder: Consider the addition of n-bit numbers X,Y,C, Carry Save Adder (CSaA) is built up of n array of FAs [6]. Each FA takes in three binary numbers (x,y,c) and returns sum and carry-out (ps, sc) [10]. Ps is a partial sum while SC is a shift-carry. The overall sum can be computed by the following three steps: Moving one place to the left in the carry sequence sc; Adding a 0 to the partial sum sequence's front most bit, and adding together these two bits with a ripple carry adder to get a value of (n + 1) bits.

$$PSi = ai \oplus bi \oplus ci$$

$$SCi = (ai \wedge bi) \vee (ai \wedge ci) \vee (bi \wedge ci)$$

By creating a propagating carry signal, the carry saver adder reduces the number of gates, hence reducing the carry latency. The main issue is the necessity for several FAs. Additionally, it only partially solves our issue by providing a single result when adding two integers. Instead, it multiplies three integers to get two, whose sum is equal to the sum of the three inputs.

3) *Carry look-ahead adder*:A Carry look-ahead (CLA) is built by connecting a specially designed carry computer to the ripple adder. Two achieve faster addition, carry at each FA stage must be made available as soon as possible. This can be achieved by determining the carry ahead of the addition of lower-bit FA. The carry might come from lower bit position FA, called carry propagated (P), or generated at the current bit position, called carry generated (G), see [8], [9], and [12]. A combinational circuit can be designed as an additional circuit to that of Fig. 2 to make carries C1-C(n-1) available immediately after the numbers X and Y are entered. Using Equation 2 we can derive C1-C4 as follows.

$$\begin{aligned} &Si = Pi \oplus Ci \qquad (2)\\ &C(i+1) = Gi + PiCi\\ &C1 = G0 + P0Cin\\ &C2 = G1 + P1C1 = G1 + P1G0 + P1P0Cin\\ &C3 = G2 + P2C2 = G2 + P2G1 + P2P1G0 + P2P1P0Cin\\ &C4 = G3 + P3C3 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0Cin \end{aligned}$$

We can see from the aforementioned Boolean equations that C 4 does not need to wait for C 3 and C 2 to propagate; rather, C 4 is propagated simultaneously with C 3 and C 2. Since each carry output's Boolean expression is simply the sum of its products, these can be implemented with a single level of AND gates, followed by an OR gate. A n-bit CLA can be depicted with a block diagram in Fig. 3, its complexity is mainly caused by the carry look-ahead circuit.

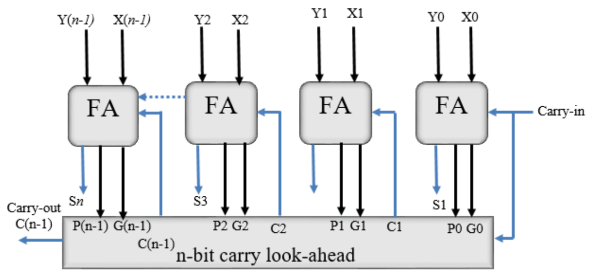


Fig. 3 Carry Look-ahead Adder

The main advantage is the speed of computation at the expense of circuit complexity. Hence the cost is high and the design is complex.

4) ***Carry Skip Adder***:A carry-skip adder (CSkA) is built by a RCA with a special speed up carry chain called a skip chain [12]. Carry Skip Adder reduces the latency of a ripple-carry adder by combining many carry-skip adders into a block-carry-skip adder [12], [13]. The improvement of critical path delay on RCA is minimal with CSkA compared to other adder topologies [10]. When compared to an n-bit ripple-carry adder, a single n-bit carry-skip-adder offers no significant speed advantage. Equation 3 implies no improvement in computational time

$$\tau CSA(n) = \tau CRA(n) \quad (3)$$

Whereby τ represents computation speed.

To build a skip-logic, we need an m-input AND-gate and one multiplexer. Fig. 4 shows a 4-bit CSkA.

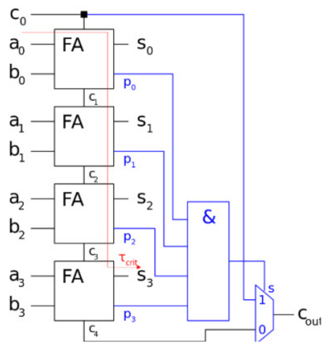


Fig. 4 A 4-bit CSkA

From Figure 4, the carry bit for each block can now "skip" over blocks with a group propagate signal set to logic 1, this significantly reduces the latency of the adder over its critical path.

5) ***Carry Select Adder*** *(CSelA)*:This consists of two ripple carry adders and a multiplexer [12]. It is having several sectors. Each sector of a carry-select adder, with the exception of the least-significant, executes two additions concurrently, one assuming a carry-in of zero and the other a carry-in of one (see Fig. 5). Once the two results from two RCAs are computed, a multiplexer is used to select a correct sum and carry. CSelA reduces RCA delay by 40%-90% [10]. The drawback is that double-sized RCA being used.

6) ***Carry Increment adder***: RCAs and incremental circuitry make up the typical Carry Increment Adder (CIA) [12], [13]. For n-bit RCA, addition is achieved by grouping the bits into two groups. The incremental circuit is a specialized circuit for incrementing the carry of the first group of bits. CIA does compute one partial sum and gets incremented if need be based on the input carry. The incremental circuit in CIA configuration is much smaller compared to the additional RCA and a multiplexer in a CSelA architecture.

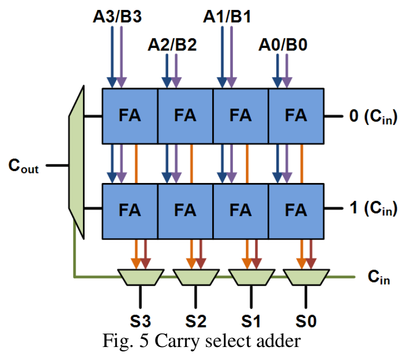


Fig. 5 Carry select adder

B. *Research motivation*

With the invention and development of FPGAs, size of circuit is reduced, speed is enhanced and power consumption is minimized. Furthermore, FPGAs have lower power consumption and they are flexible. HDL. The design of 64 bit FPGA based adder can improve the adder design parameters [6] and form a benchmark for designing higher order adders.

C. *Research gap*

Effort has been put to reduce the computational delay. Literature shows all topologies after RCA, improves computational time on expense of circuit complexity. Hence faster topologies tend to have complex design and need large design area. Out of all topologies, CLA is faster with easy design. FPGAs, have combined optimized traits, we can hence use them to design fast, small sized with low power consumption CLA.

III. METHODOLOGY

Reference is made to Fig. 6 which shows the design process flow.

A. *Design entry*

There are three techniques for specifying the design entry namely schematic-based, Hardware Description Language (HDL), or the combination of the two [7-9], [11]. An HDL based using Verilog HDL is selected due to its flexibility and execution speed [14]. With Verilog, a complex design can be realized in a structural way. Another reason is that this study does not deal more with hardware, it aims at realizing the optimized CLA design in an algorithmic way.

B. *Verilog programming*

We first design and implement a 4-bit CLA then we can call it in a “do while” or a “for” loop for 64 times to realize a 64-bit CLA. This article proposes another trick for reducing processing time. In design, the first FA in n-bit CLA may be replaced by a HA as there is no carry-in when we start to add two bits.

The programming environment provides various options such as various simulation options, performing netlist analysis, running implementation and creating the formatted file for loading to the FPGA hardware, Fig. 7 shows these options.

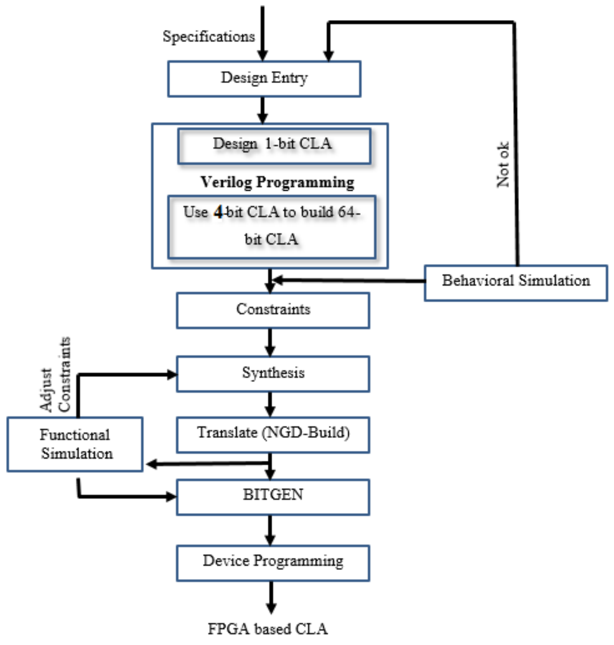
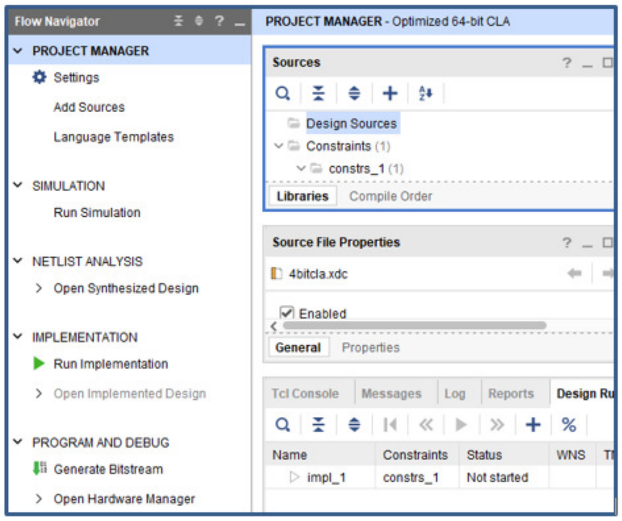


Fig. 6 FPGA device design and implementation flowchart

C. *Constraints*

After Verilog programming the device a behavioral simulation is run to check the compliance of the design to the specifications.When there is no compliance, design specifications can be adjusted until there is compliance.


Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
SIMULATION
Run Simulation
NETLIST ANALYSIS
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
PROGRAM AND DEBUG
Generate Bitstream
Open Hardware Manager

PROJECT MANAGER - Optimized 64-bit CLA

Sources
Design Sources
Constraints (1)
constrs_1 (1)
Libraries Compile Order

Source File Properties
4bitcla.xdc
Enabled
General Properties

Tcl Console Messages Log Reports Design Ru

Name	Constraints	Status	WNS	T
impl_1	constrs_1	Not started		

Fig.7 Vivado HLX project manager panel

Behavioral Simulation often called RTL Simulation is the first simulation performed on Verilog before synthesis process to verify RTL (behavioral) code and to confirm that the design is functioning as intended. During RTL simulation, signals and variables are observed, procedures and functions are traced and breakpoints are set [11]. This is a very fast simulation and so allows the designer to change the HDL code if the required functionality is not met with in a short time period. Since the design is not yet synthesized to gate level, timing and resource usage properties are still unknown. When constraints are defined, physical components of the target device, such as pins, switches, buttons, etc., are assigned to ports in the design and design time constraints are stated. This data is kept in a file called UCF (User Constraints File). PACE, Constraint Editor, and other tools are used to construct or change the UCF. Constraints are created or added just before starting Verilog programming, refer Fig. 8.

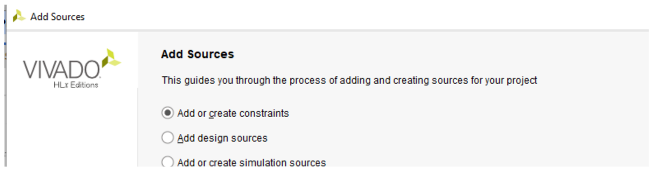
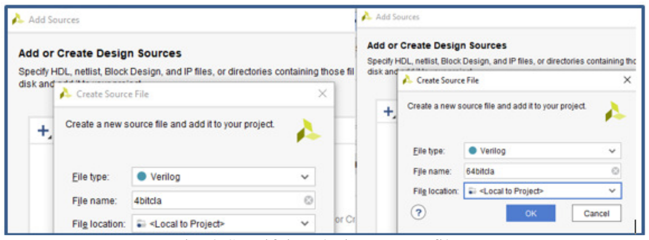

Add Sources
VIVADO HLx Editions
Add Sources
This guides you through the process of adding and creating sources for your project
[✓ Add or create constraints]
Add design sources
Add or create simulation sources

Fig. 8 Creating design constraints

A design source file is created before the simulation source file. Fig. 9 shows the creation of the design file for 4-bit CLA and that for 64-bit CLA.


Add Sources
Add or Create Design Sources
Specify HDL, netlist, Block Design, and IP files, or directories containing those fil
disk and
Create Source File
Create a new source file and add it to your project.
File type: Verilog
File name: 4bitcla
File location: <Local to Project>

Add Sources
Add or Create Design Sources
Specify HDL, netlist, Block Design, and IP files, or directories containing the
disk and
Create Source File
Create a new source file and add it to your project.
File type: Verilog
File name: 64bitcla
File location: <Local to Project>
OK Cancel

Fig. 9 Specifying design source file

D. Synthesis

A translation of Verilog code to a netlist is called synthesis [11]. A netlist contains gates, flip flops, and buffers. It is a complete circuit with all necessary logical elements [7], [11]. In this study, a single bit CLA is a basic logical element used to implement 64-bit CLA. Synthesis process will check code syntax and analyse the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected. The resulting netlist(s) is saved to an NGC (Native Generic Circuit) file. The Synthesis process provides an estimate of the resource utilization; the actual utilization can be found after a MAP process [11]. In the software used, a netlist source is specified at the time of creating the new project, see Fig. 10.

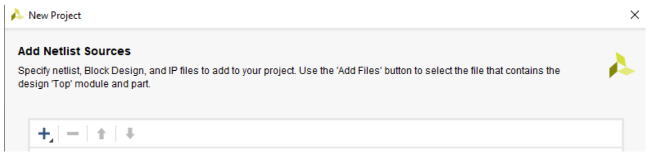

New Project
Add Netlist Sources
Specify netlist, Block Design, and IP files to add to your project. Use the 'Add Files' button to select the file that contains the design 'Top' module and part.

Fig. 10 Specifying netlist sources

E. Implementation process

Implementation is achieved in three key steps namely translate, map, and place and route [6-8], [11]. The translation process combines all the input netlists and constraints into a logic design file. This information is saved as an NGD (Native Generic

Database) file. This can be done using the NGD Build program. The Map process divides the whole circuit with logical elements into sub-blocks such that they can be fit into the FPGA logic blocks. That means the Map process fits the logic defined by the NGD file into the targeted FPGA elements of FPGA. Place and Route (PAR) places the sub-blocks from the mapping process into logic blocks according to the constraints and connects the logic blocks. Functional simulation is an important process to achieve good implementation.

Functional simulation is done twice, Post Translate Simulation (PTS) and Post PAR Simulation (PPS) [7]. PTS is done to find the logic operation of the circuit [7], [11]; this process is useful to verify the functionality of the design [6]. We can change the Verilog code at this stage if the simulation result does not conform. If PTS works but the targeted FPGA does not produce targeted results Post PAR simulation can be run to check the functionality after PAR. It is expected that If the post PAR simulation is ok then the design can be successfully implemented. After a PAR or MAP process it is necessary to perform static timing analysis [8], [11]. The earlier is done to provide a comprehensive timing summary of the design whilst the latter is aimed at listing the signal path delays emanating from the design logic. The total path delay is determined by Equation 4.

$$D(t) = D(dp) + \text{Clock Path Skew} + \text{Clock Uncertainty} + D(rp) \quad (4)$$

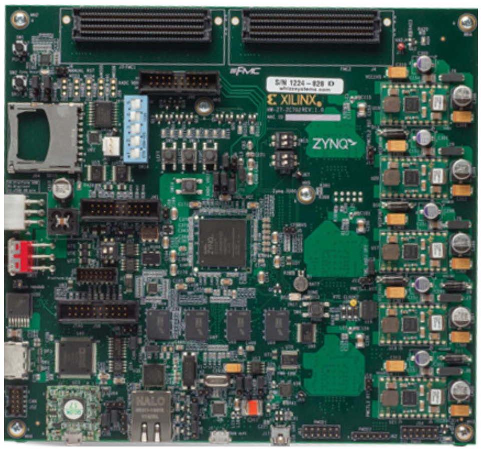
Whereby D(rp) = Routing Path Delay
D(dp)= Data Path Delay
D(t)= Total delay

F. Device Programming

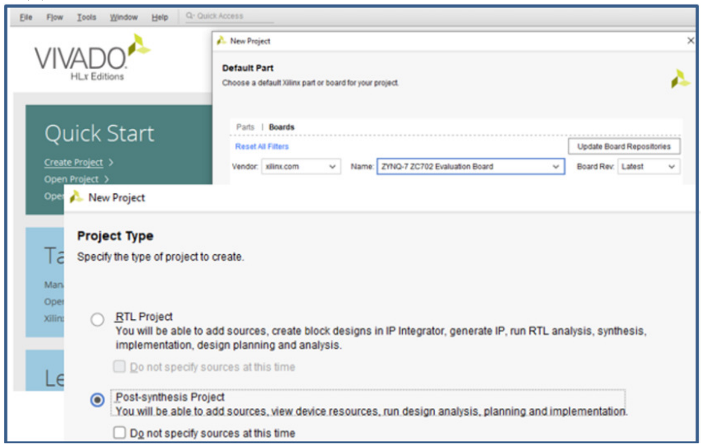
After verifying the design, it is loaded to the FPGA board to realize the design in a hardware. This is normally done by using a cable.Before loading, the design is converted into a well understood format. BITGEN is a program that do the conversion. The routed NCD file is then given to the BITGEN program to generate a bit stream (a. BIT file) which can be used to configure the target FPGA device.

G. Working environment

Vivado HLX design suite one of the Xilinx ISE has been used in this article. Figure 11(b) shows the main window of the software. It gives the option to set design entries and specify the evaluation board as well. It is necessary to select the post-synthesis option when creating a new project if the project shall include up to hardware realization. The formatted file was loaded into the Xilinx board shown in Fig. 11(a). ZYNQ-7ZC702 board supplied by Xilinx has been opted in realizing a design (see Fig. 11 (b)). The selection is done in advance when creating a new project.



(a)



New Project

Default Part
Choose a default Xilinx part or board for your project

Vendor: xilinx.com Name: ZYNQ-7 ZC702 Evaluation Board Board Rev: Latest

New Project

Project Type
Specify the type of project to create.

○ RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time

● Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

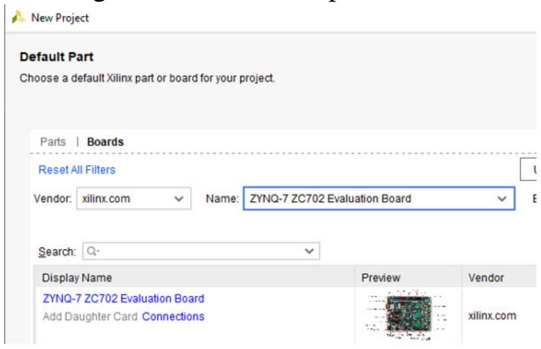
(b)

Fig. 11 Development environment

IV. EXPERIMENT RESULTS AND DISCUSSION

A. *simulation results*

The simulation software provided an option to compare computational delay, power consumption, and design areas among the three designs i.e. RCA, CLA, and the proposed one. Also, the use of FPGA is an advantage to the reduced design area. Fig. 12 In programming, the 4- bit design is recalled 16 times serially or in a loop to implement a 64-bit OCLA. The resulting simulation is shown in Fig. 13. From Fig. 13, each block represents a 4-bit CLA.


New Project

Default Part
Choose a default Xilinx part or board for your project.

Parts | Boards

Reset All Filters

Vendor: xilinx.com   Name: ZYNQ-7 ZC702 Evaluation Board

Search:

Display Name	Preview	Vendor
ZYNQ-7 ZC702 Evaluation Board Add Daughter Card Connections		xilinx.com

Fig. 12 Implementation board selection

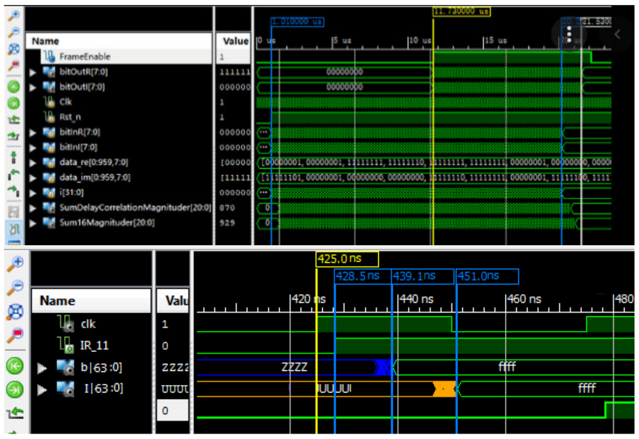


Fig. 13 Simulation waveforms

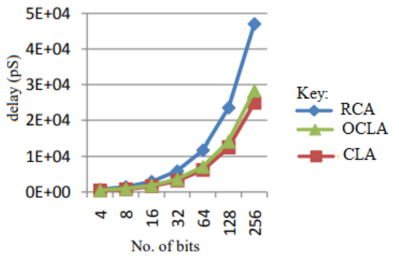
B. *Results comparison*

The simulation results were compared to RCA, ordinary CLA, and the proposed CLA design in terms of the design area, computation time, and power consumption. From Table 1, it is clear that the critical path delay has substantially increased over RCA and ordinary CLA. In a proposed design of optimized CLA (OCLA), Area saving with the proposed design is Fig. 13.1% over CLA.

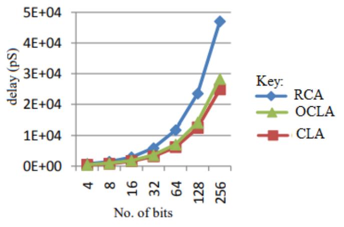
Table 1: Performance comparison

Attribute	RCA	CLA (X)	OCLA	CLA - OCLA (Y)	Percentage (Y/X)*100
Power (µW)	14510.50	18010.85	1711.35	900.50	5%
Area (μm^2)	74.12	127.31	110.63	16.68	13.1%
Computation delay (pS)	728.97	383.54	329.73	53.81	14.03%

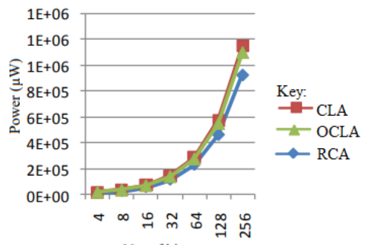
The proposed design has a 5% reduced power consumption. Moreover, the proposed design shows a 14.03% system delay performance improvement over CLA. Even though the proposed design improved over CLA, the design area is larger compared to that of RCA since CLA is more complex due to the carry generator circuity, see Fig. 14 (a). The number of logic gates and the routing for the OCLA has significantly reduced, this can be justified by the green line graph. The RCA shows less area as the design for it is simple compared to CLA and OCLA.



(a)



(b)



(c)

Fig. 14 Performance comparison between RCA, CLA, and OCLA

Furthermore, Fig. 14 (b) depicts a computational delay comparison. For a given number of bits, RCA shows the longest computational delay while OCLA is found to be quicker than CLA. Fig. 14 (c) shows that there is a slight significant improvement in power consumption. The results are due to the minimized computations and worst-case delay of the device.

Fig. 15 shows the simulation results using the gate level description of the 4-bit OCLA while Fig. 16 shows the switch level description of the designed 64-bit OCLA. The gate level description of the 64-bit OCLA would require more space as contains many logic gates.

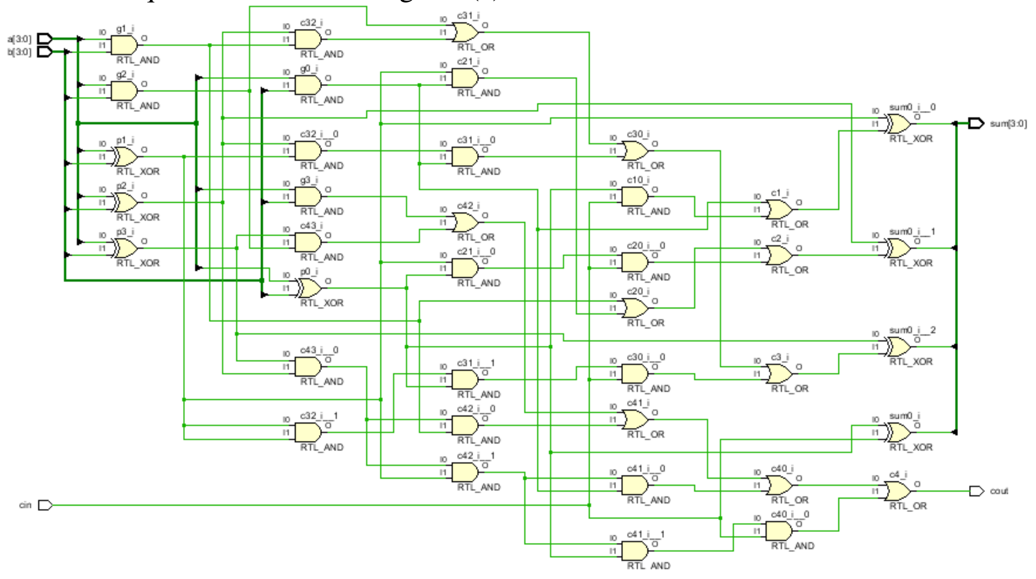


Fig. 15 Gate level description of the 4-bit CLA

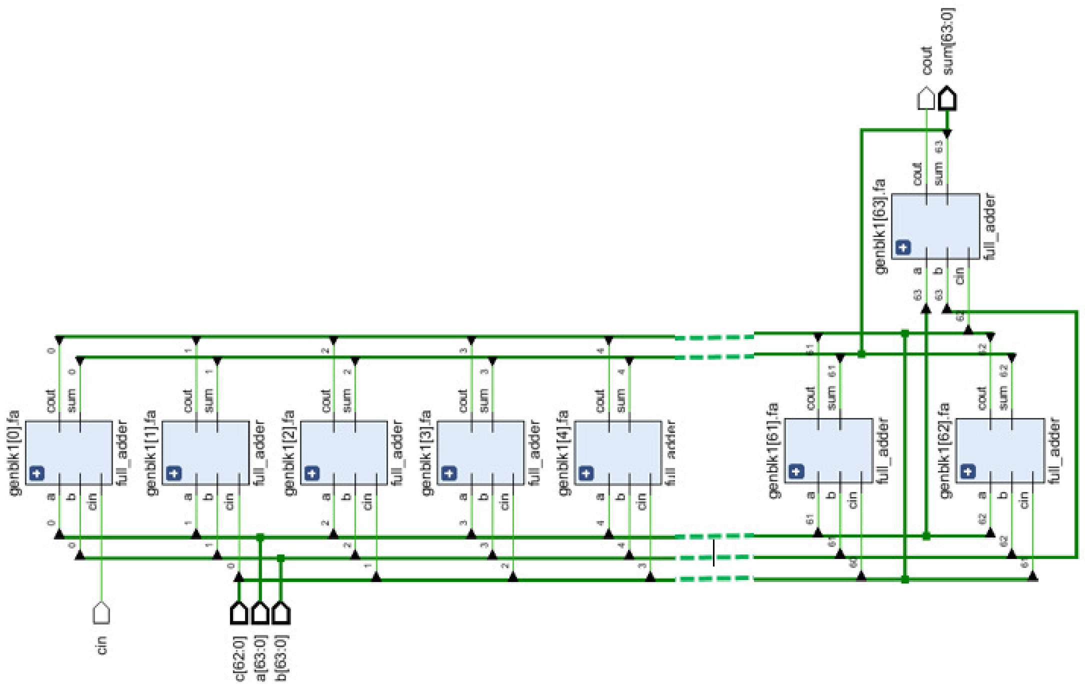


Fig. 16 Switch level description of the 64-bit OCLA

IV. CONCLUSIONS

Ripple carry adders have the simplest architecture with few electronic components in them, having a small design area and low average power consumption. However, this simple design is compromised by a high average worst-case delay. Computation time is very critical in real-time applications requiring computation in this basic circuit to be carried fast with little or negligible delay.
CLA solves this challenge of delay at the expense of large design areas and high power consumption. This study proposed an optimized CLA (OCLA) by designing a CLA using Verilog and FPGA. The Verilog program has been strategically designed to omit unnecessary iterations.

The simulation results show a 5%, 13.1%, and 14.01% improvement in power consumption, minimized design area, and decrease in worst-case delay respectively of the OCLA over CLA. The design has taken the advantage of the promising characteristics of the FPGAs such as reduced sizes, improved delay performances, and small overall design areas by VLSI technology.

ACKNOWLEDGMENT

The author acknowledges the valuable comments of Dr Denis Mwighusa of Nelson Mandela Institute of science and technology which lead to the improvement of this paper.

REFERENCES

[1] V. Thamizharasan& Dr. N. Kasthuri (2022) FPGA implementation of high performance digital FIR filter design using a hybrid adder and multiplier, International Journal of Electronics, DOI: 10.1080/00207217.2022.2098387
[2] Ming-Cui Li &Ri-Gui Zhou (2016) A novel reversible carry-selected adder with low latency, International Journal of Electronics, 103:7, 1202 1215, DOI: 10.1080/00207217.2015.1092595
[3] A.K. Goel& P.S. Bapat (1998) An Algorithm For The Speed Optimization Of Multilevel Carry Skip Adders, International Journal of Modelling and Simulation, 18:3, 208-213, DOI: 10.1080/02286203.1998.11760380
[4]Pudi. V, Sridhara., K, "Low Complexity Design of Ripple Carry and Brent Kung Addersin QCA",Nanotechnology,IEEE transactions on,Vol.11,Issue.1,pp.105-119,2012.
[5] Tripti Sharma, k.G.Sharma, Prof.B.P.Singh, Neha Arora, "High Speed, Low Power 8T Full Adder Cell with 45% Improvement in Threshold Loss Problem", Recent Advances in Networking, VLSI and Signal Processing.

[6] Y. Gong and S. Li, "High-Throughput FPGA Implementation of 256-bit Montgomery Modular Multiplier," 2010 Second International Workshop on Education Technology and Computer Science, 2010, pp. 173-176, doi: 10.1109/ETCS.2010.375.

[7] A. Inamdar, J. Ravi, S. Miller, S. S. Meher, M. ErenÇelik and D. Gupta, "Design of 64-Bit Arithmetic Logic Unit Using Improved Timing Characterization Methodology for RSFQ Cell Library," in IEEE Transactions on Applied Superconductivity, vol. 31, no. 5, pp. 1-7, Aug. 2021, Art no. 1301307, doi: 10.1109/TASC.2021.3061639

[8] I. Im and S. Kang, "Comparative Analysis between Verilog and Chisel in RISC-V Core Design and Verification," 2021 18th International SoC Design Conference (ISOCC), 2021, pp. 59-60, doi: 10.1109/ISOCC53507.2021.9614007

[9] R. Bank and S. Mangaraj, "Design and Implementation of 64-bit Carry Lookahead Adders Using Fixed and Variable Stage Structure," 2018 4th International Conference on Devices, Circuits and Systems (ICDCS), 2018, pp. 143-147, doi: 10.1109/ICDCSyst.2018.8605162.

[10] Jasbir Kaur and LalitSood, “Comparison between various types of adder topologies,” in IJCST, vol. 6, no. 1, Jan.–Mar. 2015.

[11] M. I. Baig, A. Sharif and I. H. Sheikh, "Designing on FPGA for Improved Performance," *2005 Pakistan Section Multitopic Conference*, 2005, pp. 1-5, doi: 10.1109/INMIC.2005.334469.

[12] Deepa Sinha, Tripti Sharma, k.G.Sharma, Prof.B.P.Singh, “Design and Analysis of low Power 1-bit Full Adder Cell”,IEEE, 2011.

[13] Padma Devi, AshimaGirdher and Balwinder Singh, “Improved Carry Select Adder with Reduced Area and Low Power Consumption”, International Journal of Computer Application,Vol 3.No.4, June 2010

[14] MarojuSaiKumar and Dr. P. Samundiswary, “Design and performance analysis of various adders using verilog,” in International Journal of Computer Science and Mobile Computing, IJCSMC, vol. 2, no. 9, pp. 128–138, Sep. 2013.

[15] Ananthakrishnan, Ajit, A., P V, A., Haridas, K., Nambiar, N.M., & S, D. (2019). FPGA Based Performance Comparison of Different Basic Adder Topologies with Parallel Processing Adder. 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), 87-92.

Appendix 1: 4-bit CLA Verilog code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Dar es Salaam Institute of Technology (DIT)
// Engineer: Eng. Isaack Adidas Kamanga
// Create Date: 07/21/2022 10:35:07 AM
// Design Name: Optimized 64-bit CLA
// Module Name: 4bitcla
// Target Devices: ZYNQ-7ZC702
//////////////////////////////////////////////////////////////////////////////////
module bitcla(a,b,cin,sum,cout);
input[3:0] a,b;
input cin;
output [3:0] sum;
output cout;
wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;
assign p0=(a[0]^b[0]),
p1=(a[1]^b[1]),
p2=(a[2]^b[2]),
p3=(a[3]^b[3]);
assign g0=(a[0]&b[0]),
g1=(a[1]&b[1]),
g2=(a[2]&b[2]),
g3=(a[3]&b[3]);
assign c0=cin,
c1=g0|(p0&cin),
c2=g1|(p1&g0)|(p1&p0&cin),
c3=g2|(p2&g1)|(p2&p1&g0)|(p1&p1&p0&cin),
c4=g3|(p3&g2)|(p3&p2&g1)|(p3&p2&p1&g0)|
(p3&p2&p1&p0&cin);
assign sum[0]=p0^c0,
sum[1]=p1^c1,
sum[2]=p2^c2,
sum[3]=p3^c3;
assign cout=c4;
endmodule
```

```
//Testbench code
module bitcla_sim();
reg [3:0] a;
reg [3:0] b;
reg cin;
wire [3:0] sum;
wire cout;
// Instantiate the Unit Under Test (UUT)
bitcla uut (
.a(a),
.b(b),
.cin(cin),
.sum(sum),
.cout(cout)
);
initial begin
// Initialize Inputs
a = 0;
b = 0;
cin = 0;
// Wait 10 ns for global reset to finish
#10;
a = 5;
b = 6;
cin = 1;
// Wait 10 ns for global reset to finish
#10;
end
endmodule
```

Appendix 1: 64-bit CLA Verilog code

```
`timescale 1ns / 1ps
module optimized64cla(x,y,clk,cin,sum,cout);
  //64 bit CLA using 16 4-bit CLA adders
  input [63:0] x,y;
  input clk;
  input cin;
  output cout;
  output [63:0] sum;
  wire [14:0] c;
  reg [63:0] b;
  always@(posedge clk)
  begin
    if(cin==1)
  b<=-y-1;
  else
  b<=y;
  end
 bitcla n1(x[3:0],b[3:0],cin,sum[3:0],c[0]);
 bitcla n2(x[7:4],b[7:4],c[0],sum[7:4],c[1]);
 bitcla n3(x[11:8],b[11:8],c[1],sum[11:8],c[2]);
 bitcla n4(x[15:12],b[15:12],c[2],sum[15:12],c[3]);
 bitcla n5(x[19:16],b[19:16],c[3],sum[19:16],c[4]);
 bitcla n6(x[23:20],b[23:20],c[4],sum[23:20],c[5]);
 bitcla n7(x[27:24],b[27:24],c[5],sum[27:24],c[6]);
 bitcla n8(x[31:28],b[31:28],c[6],sum[31:28],c[7]);
 bitcla n9(x[35:32],b[35:32],c[7],sum[35:32],c[8]);
 bitcla n10(x[39:36],b[39:36],c[8],sum[39:36],c[9]);
 bitcla n11(x[44:40],b[44:40],c[9],sum[44:40],c[10]);
 bitcla n12(x[48:44],b[48:44],c[10],sum[48:44],c[11]);
 bitcla n13(x[52:48],b[52:48],c[11],sum[52:48],c[12]);
 bitcla n14(x[56:52],b[56:52],c[12],sum[56:52],c[13]);
 bitcla n15(x[60:56],b[60:56],c[13],sum[60:56],c[14]);
 bitcla n16(x[64:60],b[64:60],c[14],sum[64:60],cout);
endmodule
```