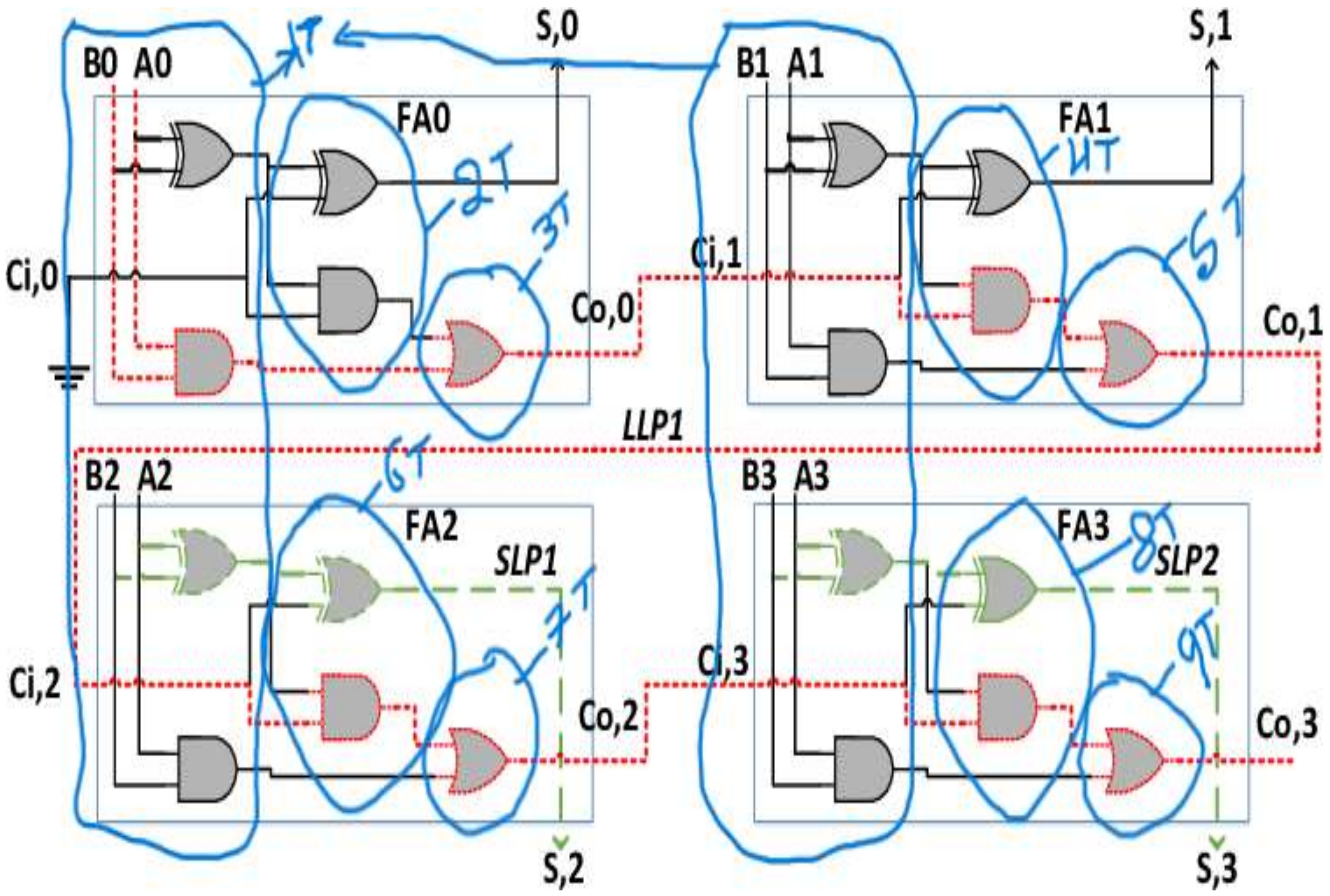
1) **Ripple carry adder**

A Ripple Carry Adder it is one of the simplest forms of adders and is often used in basic arithmetic units. The term "ripple carry" refers to the way the carry bit ripples or cascades from one stage of the adder to the next.



In the above diagram we can see the first stage gates of every single adders performs the operation parallelly, lets consider that time as T, but the next stage gates of the adders have to wait for carry of the previous adder to be come so that it will take time to complete the operation of first adder and give a carry out, this will continue up to the last carry out so that time is more in ripple carry adder.

We can calculate the propagation delay of N bit ripple carry adder using the formula
Propagation delay = (2N+1)T
If we want to perform the operation of 4 bit adder,
PD = (2*4+1) = 9T

2) **Carry look head adder**

A Carry Look-Ahead Adder, also known as a carry-ahead adder, Unlike the Ripple Carry Adder, which propagates carries through each stage sequentially, a Carry Look-Ahead Adder is designed to generate carry signals for multiple stages simultaneously, resulting in faster addition.

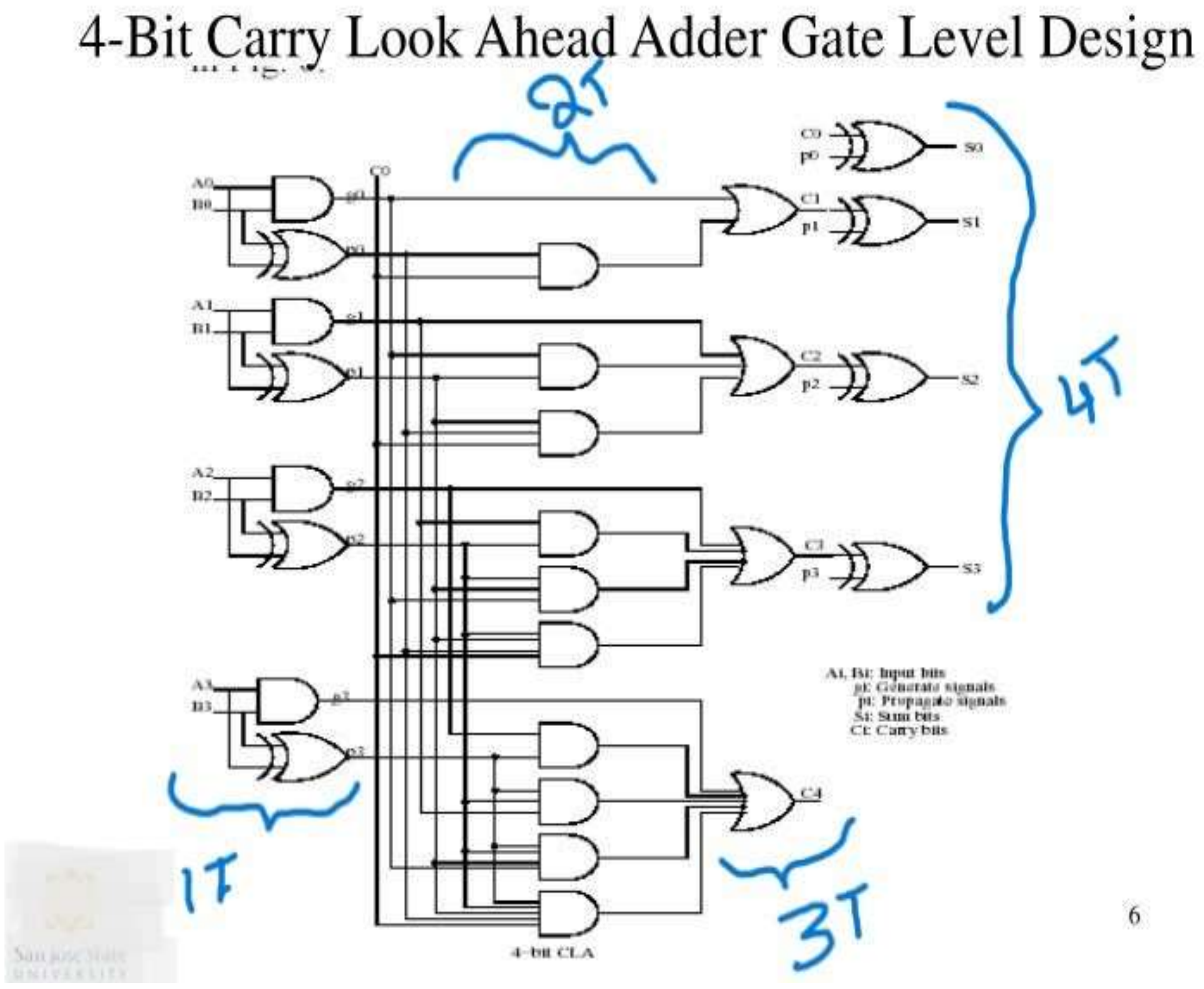
The key feature of a Carry Look-Ahead Adder is its ability to "look ahead" at the input carries and calculate the carry-out for each stage independently without waiting for the carry to ripple through the lower-order stages. This parallel carry generation improves the speed of addition and reduces the critical path delay, making CLA suitable for high-speed arithmetic operations.

Carry-lookahead adder

- First compute carry propagate, generate:
  - $P_i = a_i$ xor b_i
  - $G_i = a_i\ b_i$
- Compute sum and carry from P and G:
  - $s_i =\ a_i$ xor b_i xor $c_i = P_i$ xor c_i
  - $c_{i+1} = G_i + P_i c_i$

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

$$
\begin{aligned}
C_1 &= G_0 + P_0C_0 \\
C_2 &= G_1 + P_1C_1 = G_1 + P_1\,(G_0 + P_0C_0) \\
&= G_1 + P_1G_0 + P_1P_0C_0 \\
C_3 &= G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \\
C_4 &= G_3 + P_3C_3 \\
&= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0
\end{aligned}
$$



In the above diagram we can see that the gates output of first stage in every adders is taken to calculate the sum and carry of veery single adders, so if we consider 1st stage is taking time T, after that every and gates perform the operation at a time T like that the or and Exor will take time 2T so that the total time of 4 bit look head carry adder is 4T

A Carry Look-Ahead Adder is generally considered better than a Ripple Carry Adder in terms of speed and performance for larger bit widths.

Advantages

1. Parallelism: A CLA can compute all carry bits in parallel, while an RCA computes them sequentially. This parallelism leads to faster addition because all the carry bits are available at the same time, enabling faster clock speeds.

2.Reduced Propagation Delay: In RCA, the carry bit ripples through each full adder stage. This ripple effect introduces a delay in computing the final result, which is proportional to the number of bits being added. In CLA, the propagation delay is reduced since all carry bits are generated simultaneously, resulting in a constant delay regardless of the bit width.

3. Better Scalability: As the bit width increases, the RCA's delay increases linearly with the number of bits. CLA's delay grows logarithmically, making it a more scalable choice for large bit widths.

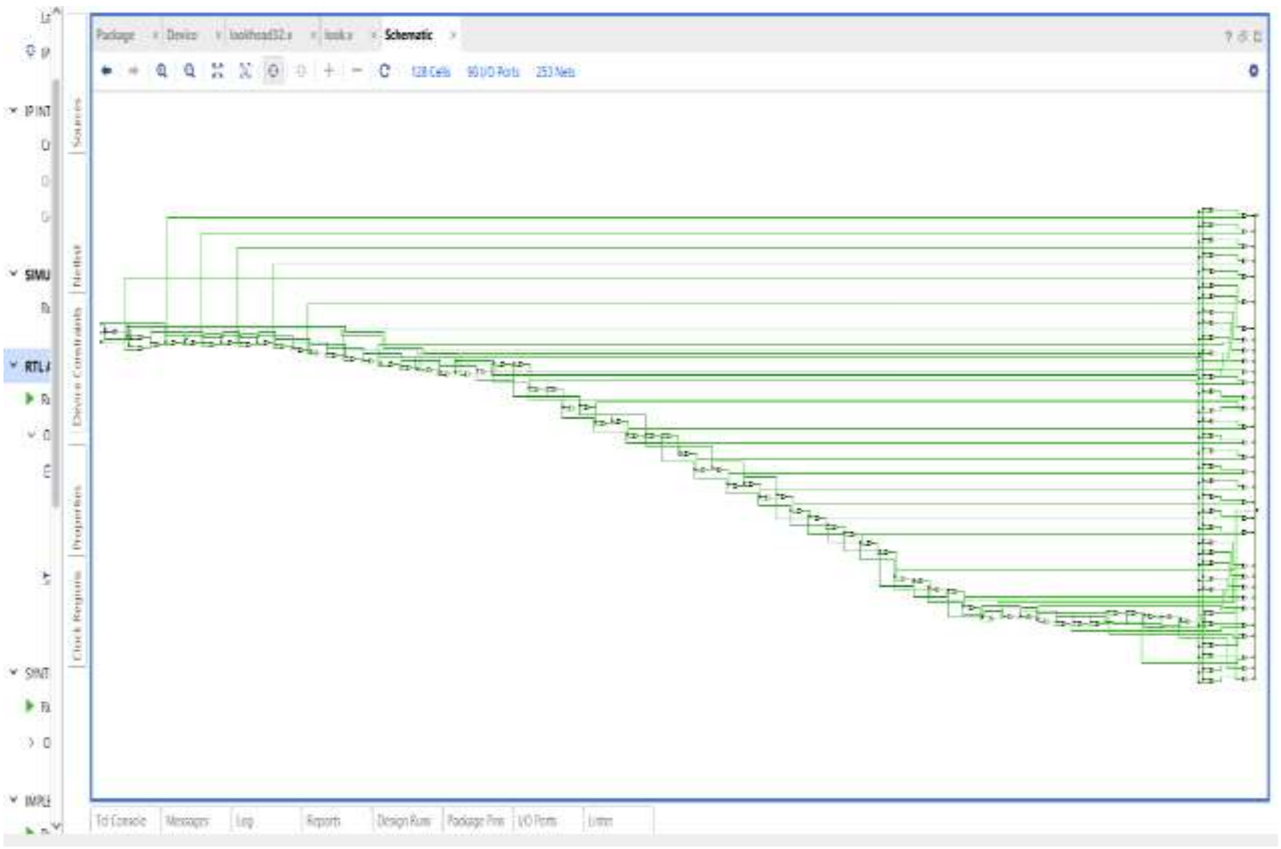
4. Efficiency: CLA is more efficient in terms of gate count and transistor count than RCA for large bit widths, which can lead to power savings and more compact implementations in modern integrated circuits.

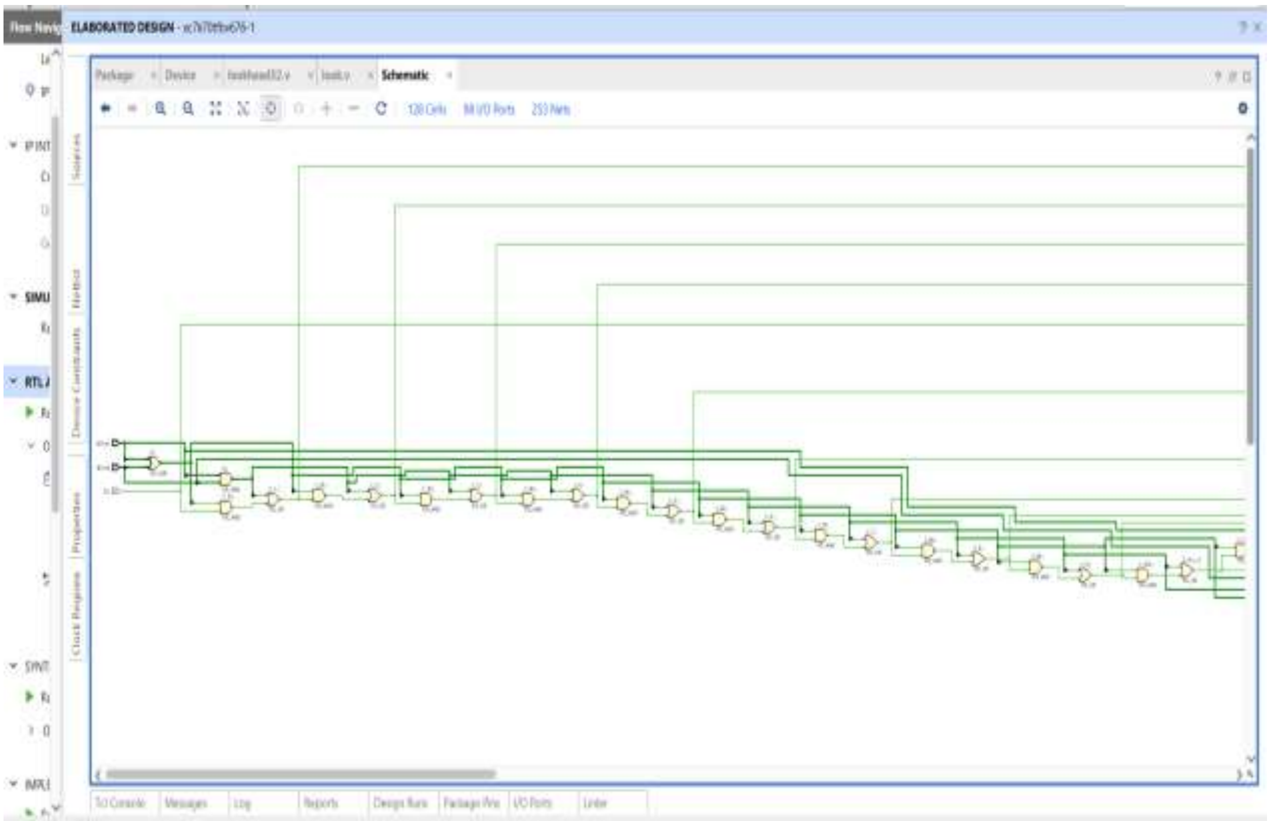
5. Carry Chain: RCA is limited by the carry chain, where the carry from one stage is dependent on the previous stage's output. In contrast, CLA breaks down this dependency, allowing for more efficient hardware utilization.

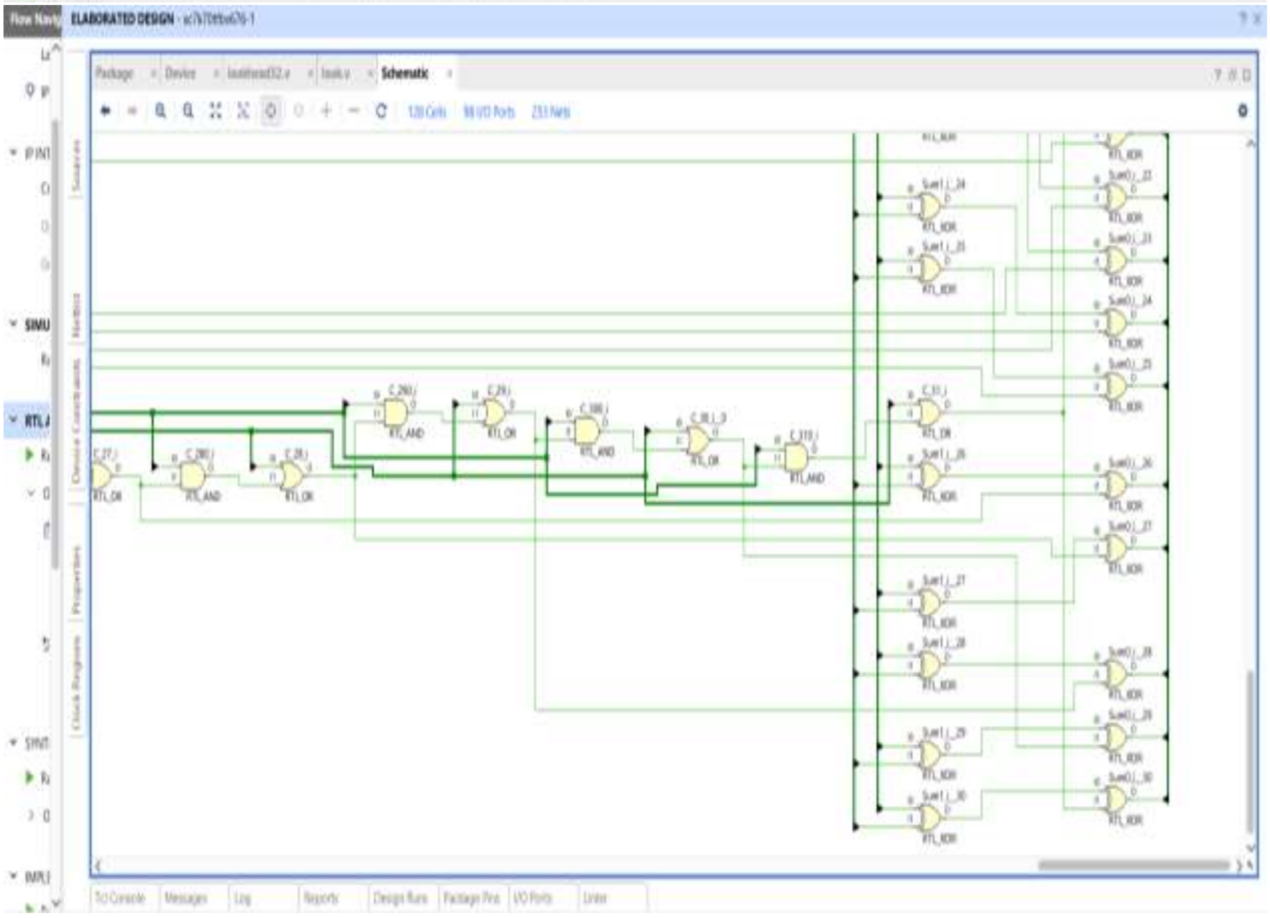
6. Predictable Timing: CLA provides predictable and constant timing characteristics regardless of the input values, making it suitable for high-speed applications.

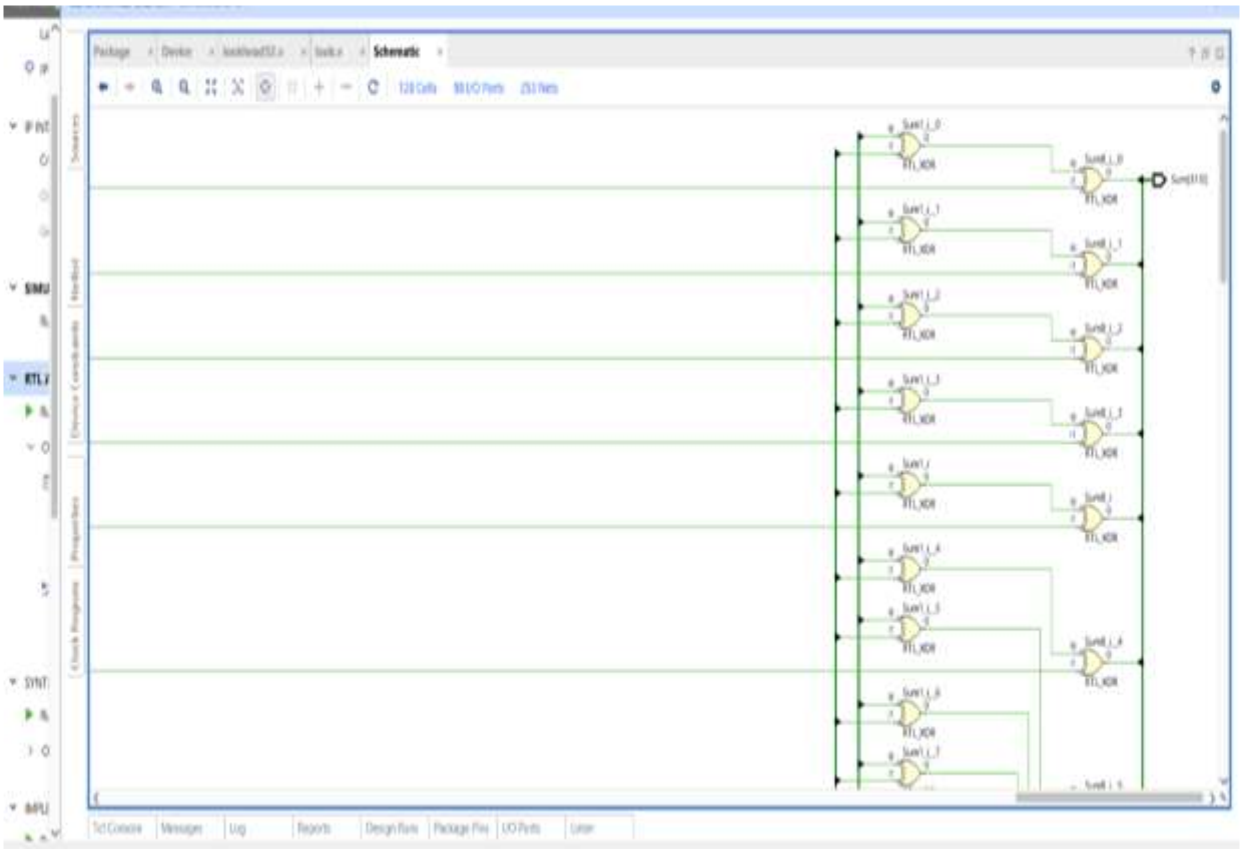
CLA has some drawbacks, such as higher hardware complexity and potentially higher power consumption for smaller bit widths compared to RCA. Therefore, the choice between CLA and RCA depends on the specific requirements of the design, including speed, area, power consumption, and available resources.

Elaborated design of 32 bit carry look head adder without input and output register

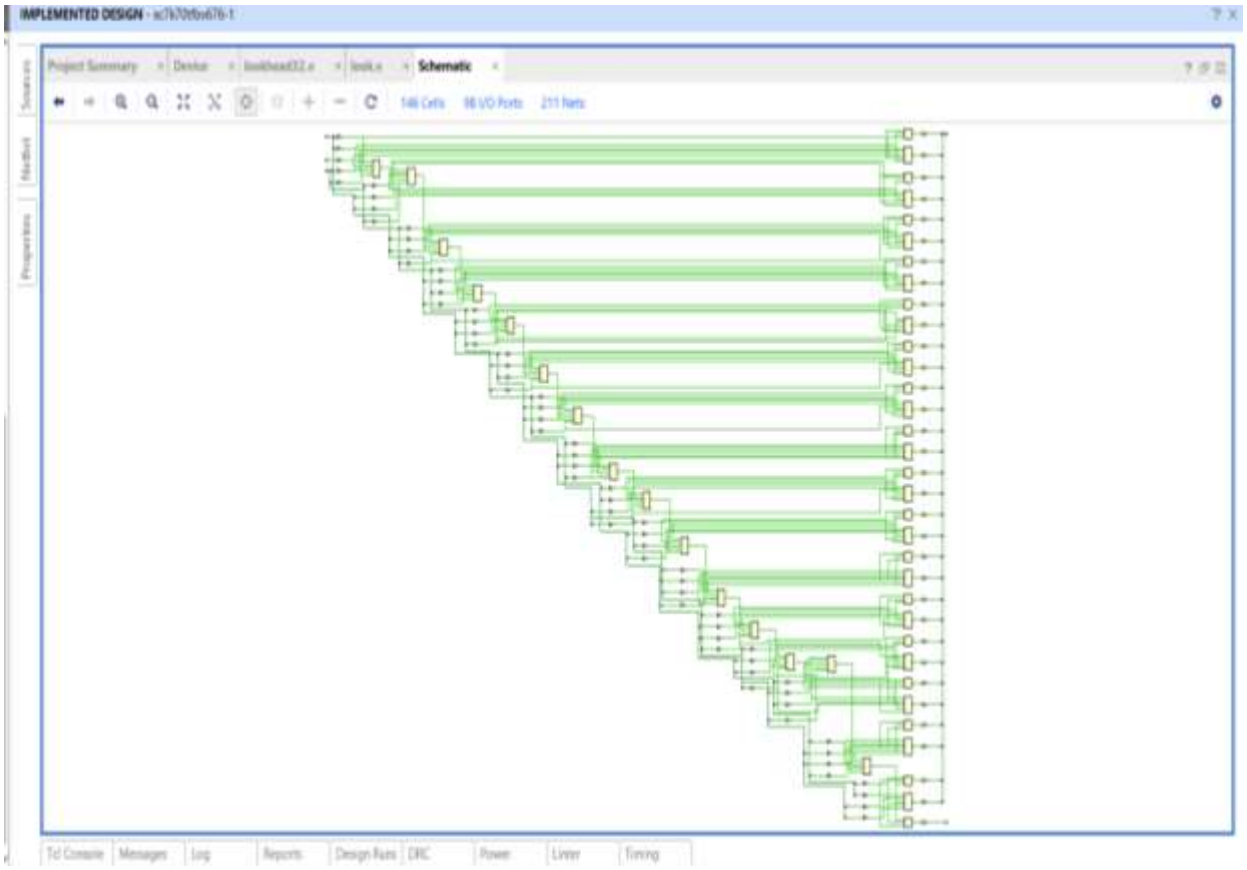



Flow Navigator
ELABORATED DESIGN - xc7k70tfbv676-1
Package
Device
lookahead32.v
look.v
Schematic
128 Cells
98 I/O Ports
253 Nets
Sources
Netlist
Device Constraints
Properties
Clock Regions
Tcl Console
Messages
Log
Reports
Design Runs
Package Pins
I/O Ports
Linter

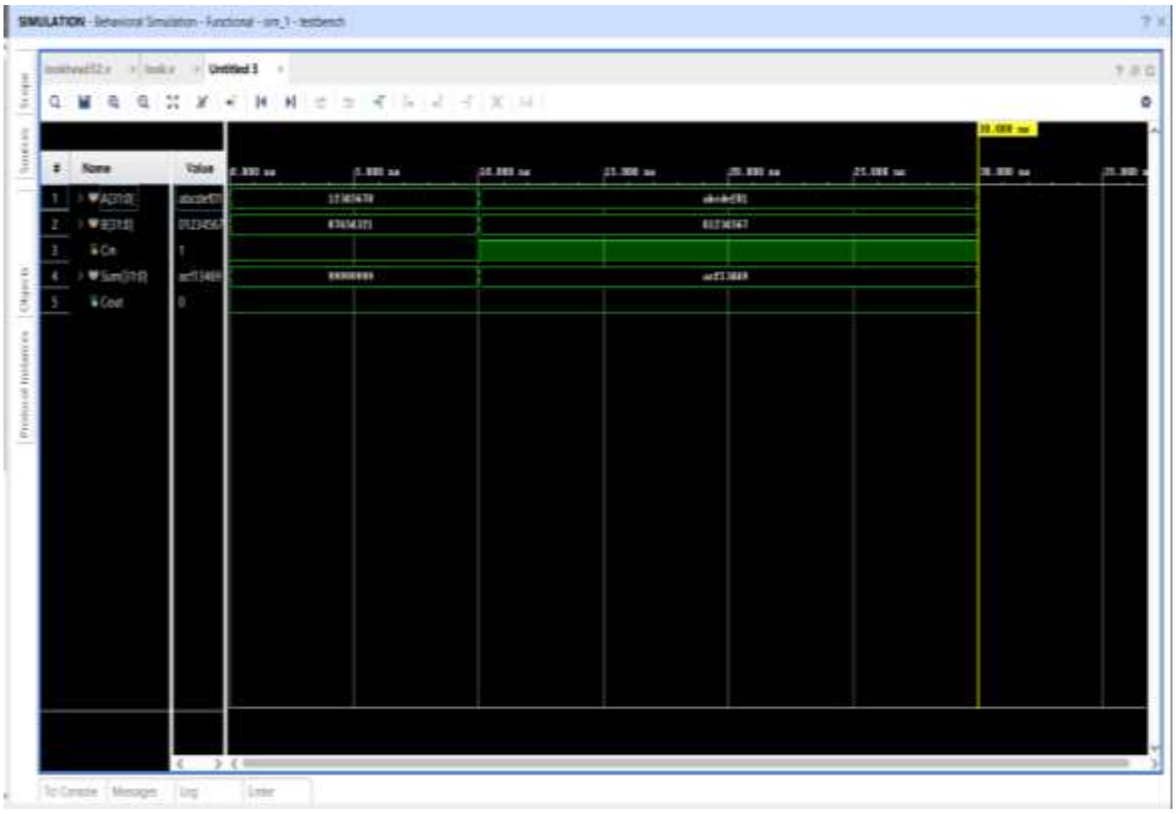

Flow Navigator
ELABORATED DESIGN - xc7k70tfbv676-1
Package
Device
lookahead32.v
look.v
Schematic
128 Cells
98 I/O Ports
253 Nets
C_27_i
RTL_OR
C_28_i
RTL_AND
C_28_i
RTL_OR
C_29_i
RTL_AND
C_29_i
RTL_OR
C_30_i
RTL_AND
C_30_i_0
RTL_OR
C_31_i
RTL_AND
C_31_i
RTL_OR
Sum1_i_24
Sum1_i_25
Sum1_i_26
Sum1_i_27
Sum1_i_28
Sum1_i_29
Sum1_i_30
Sum0_i_22
Sum0_i_23
Sum0_i_24
Sum0_i_25
Sum0_i_26
Sum0_i_27
Sum0_i_28
Sum0_i_29
Sum0_i_30
RTL_XOR
Tcl Console
Messages
Log
Reports
Design Runs
Package Pins
I/O Ports
Linter



Implemented design



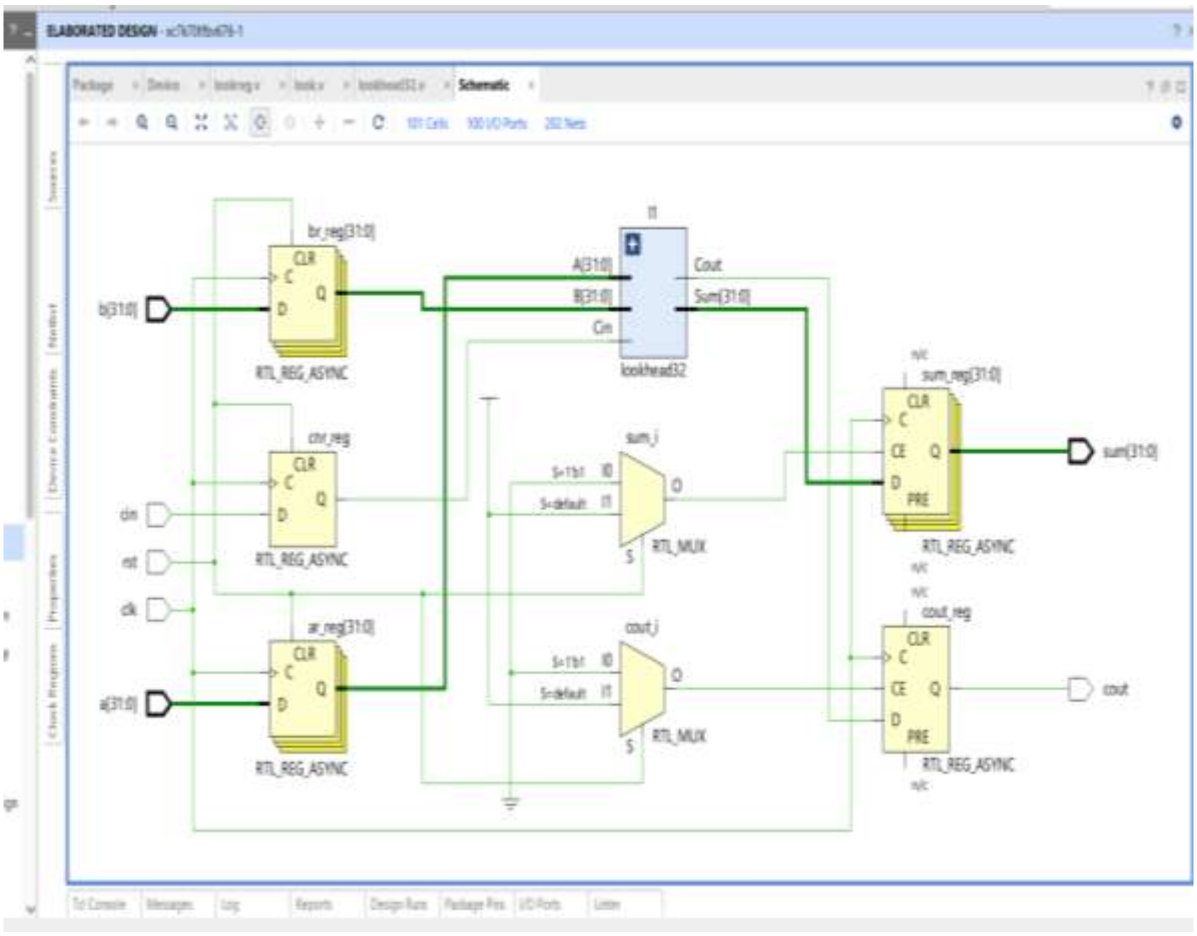
Bheavioural simulation



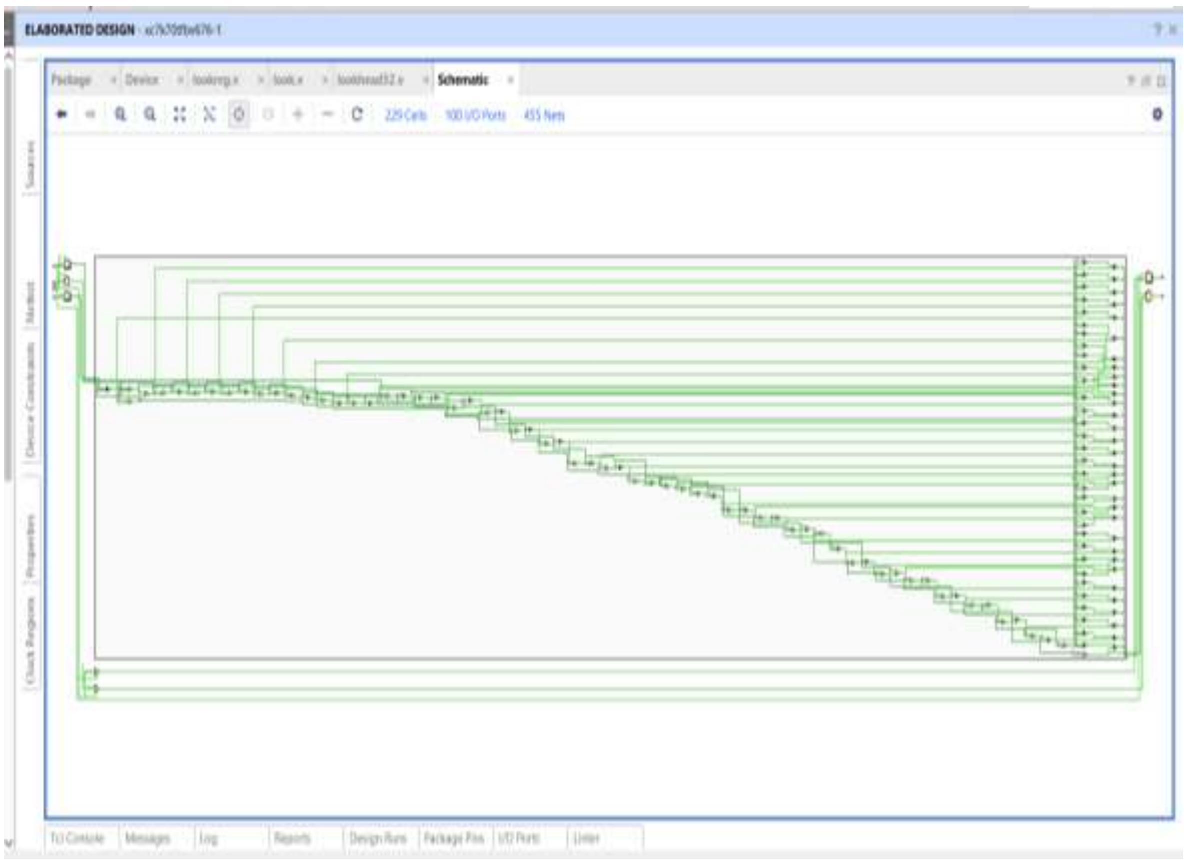
SIMULATION - Behavioral Simulation - Functional - sim_1 - testbench

#	Name	Value
1	A[31:0]	abcdef01
2	B[31:0]	01234567
3	Cin	1
4	Sum[31:0]	acf13469
5	Cout	0

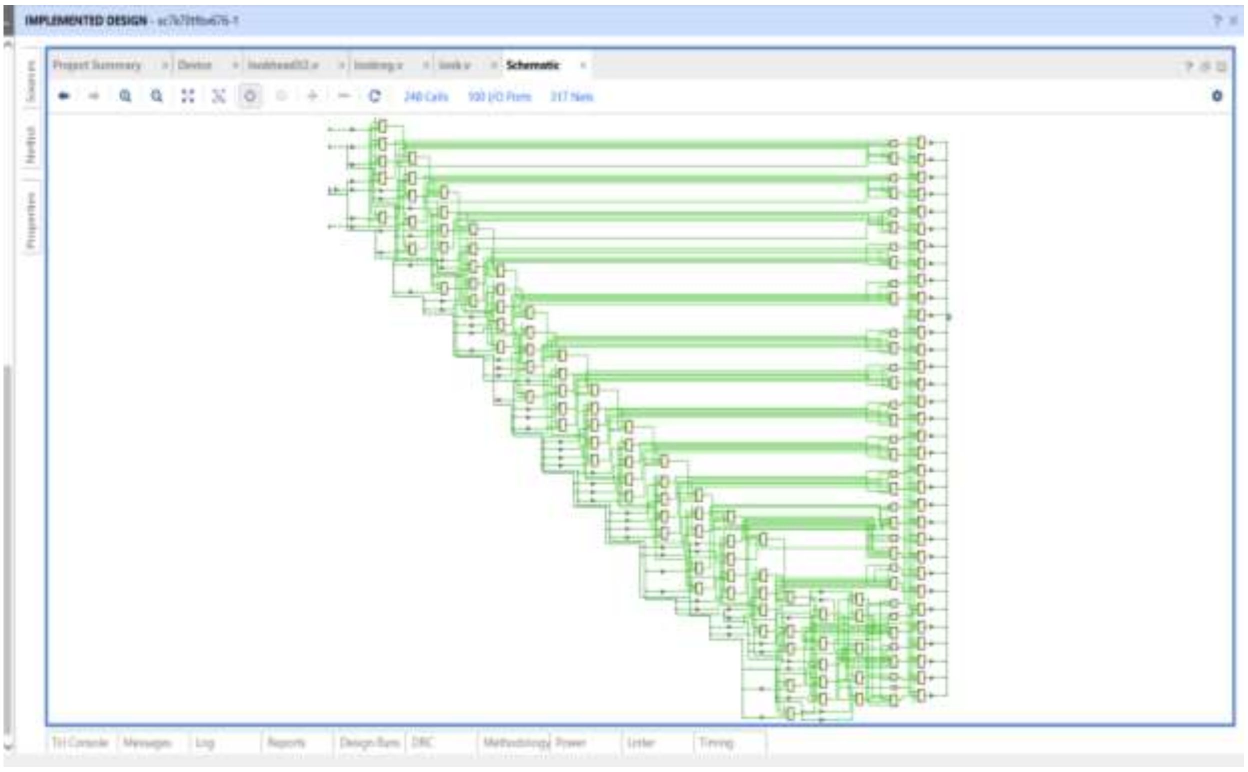
Elaborated design of 32bit lookhead adder with registers



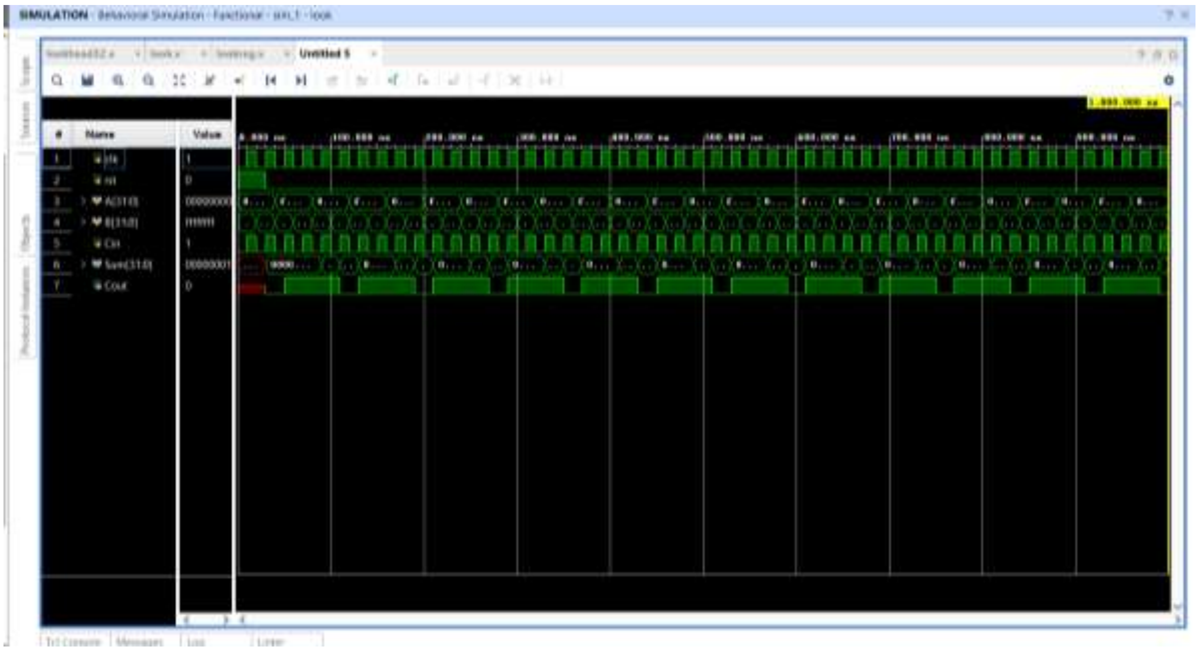
ELABORATED DESIGN



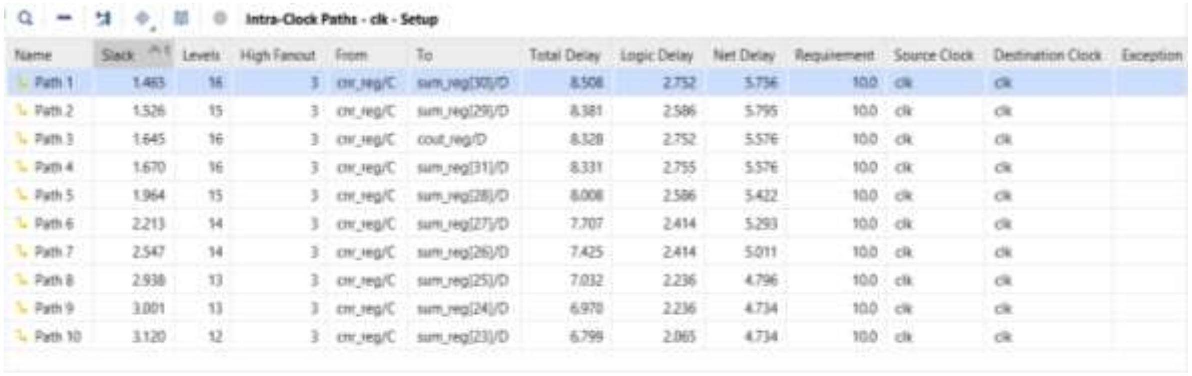
Implemented design



Simulation waveform


SIMULATION - Behavioral Simulation - Functional - sim_1 - look
Untitled 5
Name
Value
clk
rst
A[31:0]
B[31:0]
Cin
Sum[31:0]
Cout

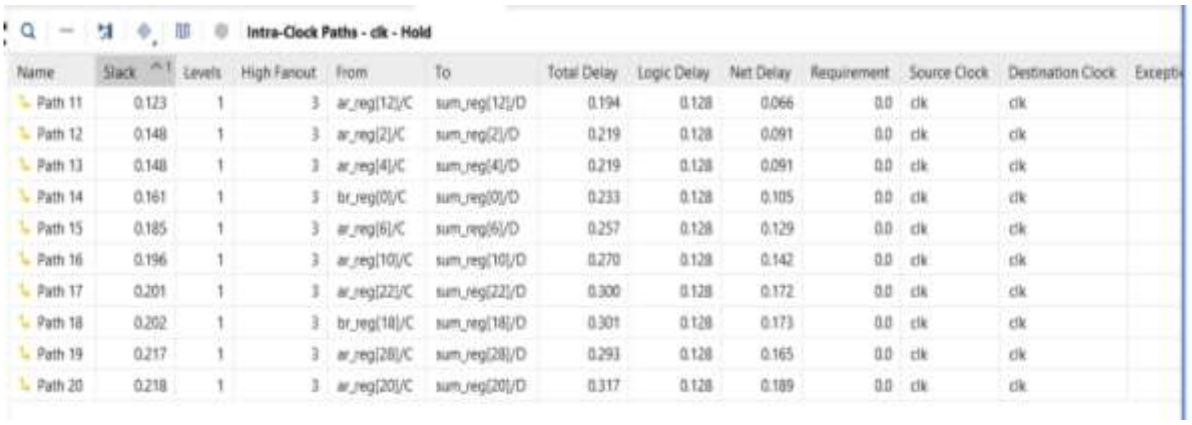
Setup slack



Intra-Clock Paths - clk - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	1.465	16	3	cnr_reg/C	sum_reg[30]/D	8.508	2.752	5.756	10.0	clk	clk	
Path 2	1.526	15	3	cnr_reg/C	sum_reg[29]/D	8.381	2.586	5.795	10.0	clk	clk	
Path 3	1.645	16	3	cnr_reg/C	cout_reg/D	8.328	2.752	5.576	10.0	clk	clk	
Path 4	1.670	16	3	cnr_reg/C	sum_reg[31]/D	8.331	2.755	5.576	10.0	clk	clk	
Path 5	1.964	15	3	cnr_reg/C	sum_reg[28]/D	8.008	2.586	5.422	10.0	clk	clk	
Path 6	2.213	14	3	cnr_reg/C	sum_reg[27]/D	7.707	2.414	5.293	10.0	clk	clk	
Path 7	2.547	14	3	cnr_reg/C	sum_reg[26]/D	7.425	2.414	5.011	10.0	clk	clk	
Path 8	2.938	13	3	cnr_reg/C	sum_reg[25]/D	7.032	2.236	4.796	10.0	clk	clk	
Path 9	3.001	13	3	cnr_reg/C	sum_reg[24]/D	6.970	2.236	4.734	10.0	clk	clk	
Path 10	3.120	12	3	cnr_reg/C	sum_reg[23]/D	6.799	2.065	4.734	10.0	clk	clk	

Hold slack



Intra-Clock Paths - clk - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Excepti
Path 11	0.123	1	3	ar_reg[12]/C	sum_reg[12]/D	0.194	0.128	0.066	0.0	clk	clk	
Path 12	0.148	1	3	ar_reg[2]/C	sum_reg[2]/D	0.219	0.128	0.091	0.0	clk	clk	
Path 13	0.148	1	3	ar_reg[4]/C	sum_reg[4]/D	0.219	0.128	0.091	0.0	clk	clk	
Path 14	0.161	1	3	br_reg[0]/C	sum_reg[0]/D	0.233	0.128	0.105	0.0	clk	clk	
Path 15	0.185	1	3	ar_reg[6]/C	sum_reg[6]/D	0.257	0.128	0.129	0.0	clk	clk	
Path 16	0.196	1	3	ar_reg[10]/C	sum_reg[10]/D	0.270	0.128	0.142	0.0	clk	clk	
Path 17	0.201	1	3	ar_reg[22]/C	sum_reg[22]/D	0.300	0.128	0.172	0.0	clk	clk	
Path 18	0.202	1	3	br_reg[18]/C	sum_reg[18]/D	0.301	0.128	0.173	0.0	clk	clk	
Path 19	0.217	1	3	ar_reg[28]/C	sum_reg[28]/D	0.293	0.128	0.165	0.0	clk	clk	
Path 20	0.218	1	3	ar_reg[20]/C	sum_reg[20]/D	0.317	0.128	0.189	0.0	clk	clk	

Most critical paths

Path 1

Setup slack = 1.465

From = cnr_reg/C,   To = sum_reg30/D

Requirement = 10 ns

Source clock path = 4.224   = ((IBUF IO =0.892) + (NET = 1.764) + (BUFG I O = 0.120) + (NET = 1.448))

Data path(arival time) = 12.732 =4.224+((FDCE =0.308) + (NET = 0.696 + 0.314 + 0.311 + 0.202 + 0.556 + 0.324 + 0.315 + 0.202 + 0.552 + 0.364 + 0.425 + 0.225 + 0.310 + 0.211 + 0.359 + 0.390) + (LUT 5 = 0.064 + 0.183 + 0.174 + 0.173 + 0.168 + 0.053 + 0.062 + 0.173 + 0.171 + 0.177 + 0.187 + 0.175 + 0.177 + 0.173 + 0.169) +

(LUT 3 = 0.165))

Destination clock path(required time) = 14.197 = (CLK RISE EDGE = 10)+(IBUF I O =0.815)+(NET =1.679+1.339)+(BUG I O = 0.113)+(CLK PERMISSION = 0.251)+(CLK UNSERTINITY = -0.035)+(FDRE=0.035)

Setup salck = required time – arrival time

1.465   =   14.197   -   12.732

Applications

The Carry Lookahead Adder (CLA) is a type of digital circuit used for performing fast addition of binary numbers. It has applications in various fields where fast arithmetic operations are essential.

1. Microprocessors and CPUs: CLAs are often used in the arithmetic logic unit (ALU) of microprocessors and central processing units (CPUs) to perform addition operations quickly. The ALU is responsible for executing arithmetic and logical operations in a computer.

2. Digital Signal Processors (DSPs): DSPs are specialized microprocessors designed for efficient digital signal processing tasks. CLAs can be integrated into DSPs to accelerate addition operations, which are common in signal processing applications.

3. Computer Arithmetic Units: In various computer systems, such as scientific and engineering workstations, CLAs can be employed in arithmetic units for high-speed computation.

4. Graphics Processing Units (GPUs): Modern GPUs are highly parallel processors used for graphics rendering, scientific computing, and machine learning. CLAs can enhance the arithmetic capabilities of GPUs, enabling faster numeric calculations.

5. FPGA and ASIC Designs: CLAs can be implemented in field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) to perform addition operations efficiently in custom hardware.

6. High-Performance Computing (HPC): In scientific simulations and simulations of physical phenomena, CLAs can be used to speed up numerical computations, reducing simulation time.

7. Cryptography: Many cryptographic algorithms, such as those used in encryption and decryption, involve arithmetic operations on large binary numbers. CLAs can be employed to accelerate these calculations.

8. Network Processors: In networking equipment like routers and switches, CLAs can be used to perform packet processing and routing calculations quickly.

9. Digital Filters: Digital filters, commonly used in signal processing applications, involve multiplication and addition of binary numbers. CLAs can accelerate these operations, improving filter performance.

10. Image and Video Processing: In image and video processing, addition operations are performed to enhance or manipulate images and videos. CLAs can be used to speed up these tasks.

11. Scientific Computing: Scientific simulations and computations in fields like physics, chemistry, and engineering often require high-speed arithmetic operations. CLAs can be beneficial in these applications.

12. Automated Control Systems: In control systems for industrial automation, robotics, and automotive applications, CLAs can be used for fast computation of control signals.

References

1) A novel implementation of 4bit carry lookhead adder

Published in: 2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC)

2) ACLA: An Approximate Carry-Lookahead Adder with Intelligent Carry Judgement and Correction
Published in: 2021 22nd International Symposium on Quality Electronic Design (ISQED)

3) Factorized Carry Lookahead Adders
Published in: 2019 International Symposium on Signals, Circuits and Systems (ISSCS)