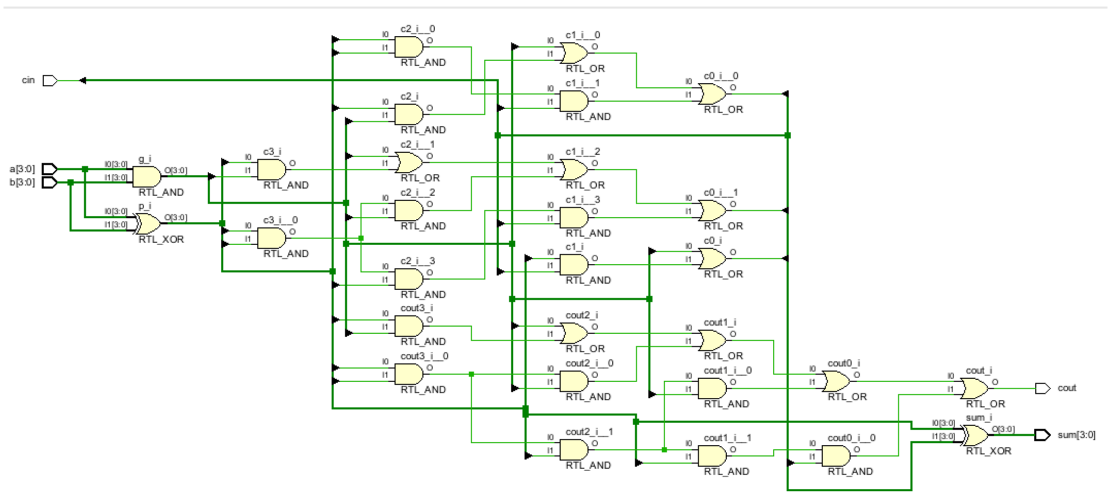
32-bit carry-look-ahead adder

A carry-look ahead adder (CLA) or fast adder is a type of electronics adder used in digital logic. A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits. The carry generated by the full adder in the least significant bit stage must propagate through all the intermediate adders till it reaches the most significant bit adder .The sum and carry out of any stage depends on the carry output of the previous stage. The carry out of previous stage depends on the carry out of the stage prior to that and so on .So if we ensure that the sum and carry out of any stage is not dependent on the results of any previous stages then the ripple effect is eliminated and the speed of addition increases. The carry-look ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger-value bits of the adder.

Step-1 : Implementation of 4 bit carry-look ahead adder

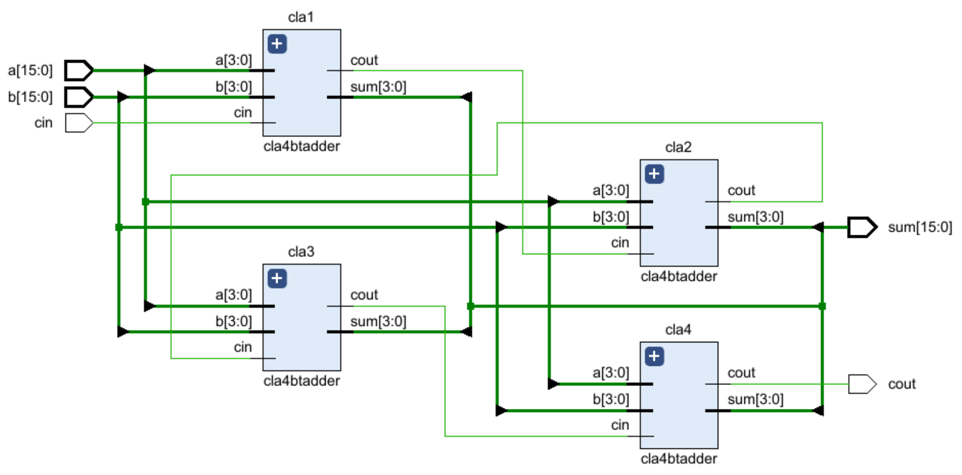
```
module cla4btadder(a,b, cin, sum,cout);
input [3:0] a,b;
input cin;
output [3:0] sum;
output cout;
wire [3:0] p,g,c;
assign p=a^b;
assign g=a&b;
assign c[0]=cin;
assign c[1]= g[0]|(p[0]&c[0]);
assign c[2]= g[1] | (p[1]&g[0]) | p[1]&p[0]&c[0];
assign c[3]= g[2] | (p[2]&g[1]) | p[2]&p[1]&g[0] | p[2]&p[1]&p[0]&c[0];
assign cout= g[3] | (p[3]&g[2]) | p[3]&p[2]&g[1] | p[3]&p[2]&p[1]&g[0] | p[3]&p[2]&p[1]&p[0]&c[0];
assign sum=p^c;
 endmodule
```



Step-2 : Implementation of 16 bit carry-look ahead adder

Instantiation of four 4bit carry-look ahead adder to design 16 bit carry-look ahead adder.

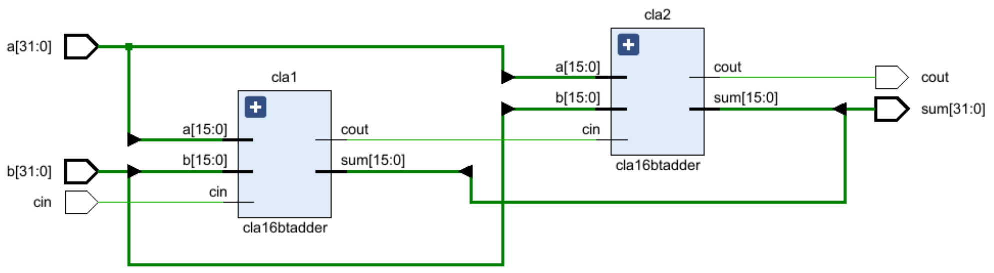
```
module cla16btadder(a,b, cin, sum,cout);
input [15:0] a,b;
input cin;
output [15:0] sum;
output cout;
wire c1,c2,c3;
cla4btadder cla1 (.a(a[3:0]), .b(b[3:0]), .cin(cin), .sum(sum[3:0]), .cout(c1));
cla4btadder cla2 (.a(a[7:4]), .b(b[7:4]), .cin(c1), .sum(sum[7:4]), .cout(c2));
cla4btadder cla3(.a(a[11:8]), .b(b[11:8]), .cin(c2), .sum(sum[11:8]), .cout(c3));
cla4btadder cla4(.a(a[15:12]), .b(b[15:12]), .cin(c3), .sum(sum[15:12]), .cout(cout));
endmodule
```



Step-3 : Implementation of 32 bit carry-look ahead adder

Instantiation of two 16bit carry-look ahead adder to design 32 bit carry-look ahead adder.

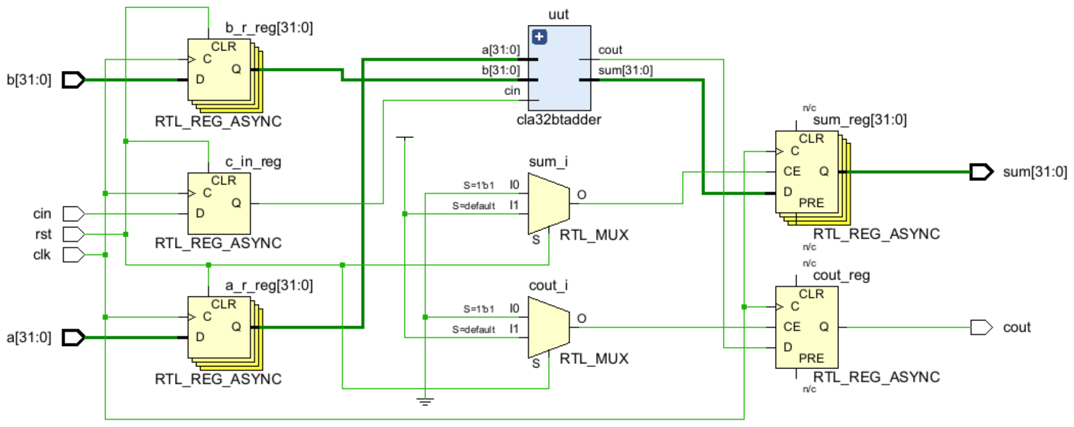
```
module cla32btadder(a,b, cin, sum,cout);
input [31:0] a,b;
input cin;
output [31:0] sum;
output cout;
wire C1;
cla16btadder cla1 (.a(a[15:0]), .b(b[15:0]), .cin(cin), .sum(sum[15:0]), .cout(C1));
cla16btadder cla2 (.a(a[31:16]), .b(b[31:16]), .cin(C1), .sum(sum[31:16]), .cout(cout));
endmodule
```



Step-4 : Implementation of 32 bit pipelined carry-look ahead adder

Pipelined carry-look ahead adder is nothing but inputs and outputs from the adder are get stored in registers. Depending on clock edge or reset pin output will be observed .

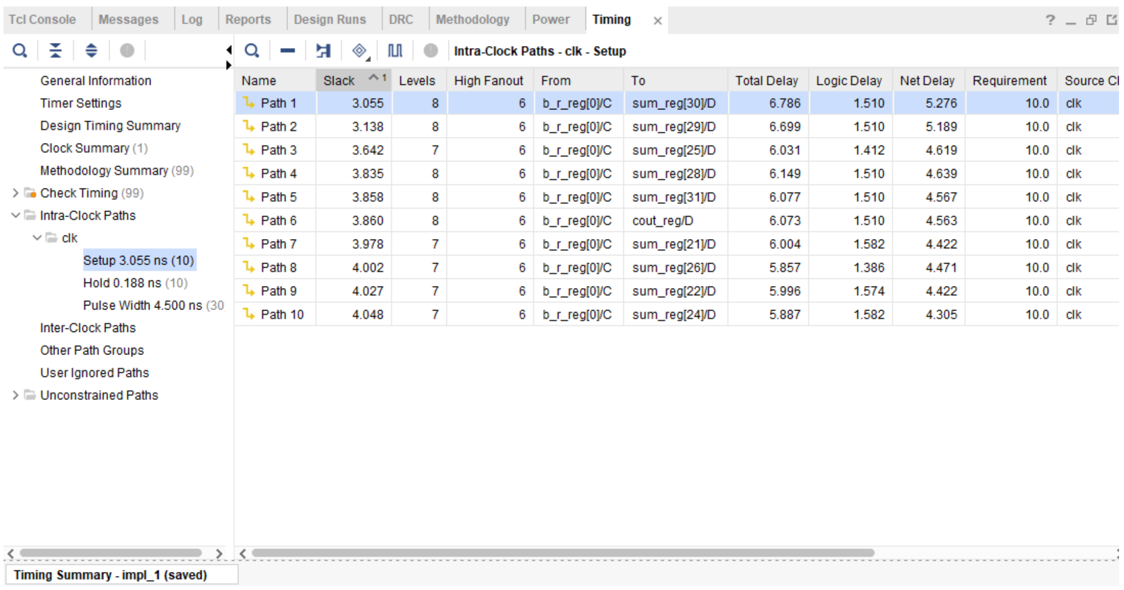
```
module pipelined_cla32btadder(input clk,
input rst,
input  [31:0]a,
input  [31:0]b,
input cin,
output  reg[31:0]sum,
output  reg cout
   );
   reg [31:0] a_r,b_r;
   reg c_in;
   wire [31:0] s_r;
   wire c_r;
   cla32btadder uut(.a(a_r), .b(b_r), .cin(c_in), .sum(s_r), .cout(c_r));
   always @ (posedge clk or posedge rst) begin
   if(rst)
   begin
   a_r<=32'b00000000000000000000000000000000;
   b_r<=32'b00000000000000000000000000000000;
   c_in<=1'b00000000000000000000000000000000;
   end
   else
   begin
   a_r<=a;
   b_r<=b;
   c_in<=cin;
   sum<=s_r;
   cout<=c_r;
   end
   end
endmodule
```



Timing Summary :

There are total 10 setup timing paths and 10 hold timing paths. For the designed shown above the clock is kept constant that is 10ns.(**clock =10ns**)

- (rising edge-triggered cell FDCE clocked by clk {rise@0.000ns fall@5.000ns period=10.000ns})
- (rising edge-triggered cell FDRE clocked by clk {rise@0.000ns fall@5.000ns period=10.000ns})



Intra-Clock Paths - clk - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Cl
Path 1	3.055	8	6	b_r_reg[0]/C	sum_reg[30]/D	6.786	1.510	5.276	10.0	clk
Path 2	3.138	8	6	b_r_reg[0]/C	sum_reg[29]/D	6.699	1.510	5.189	10.0	clk
Path 3	3.642	7	6	b_r_reg[0]/C	sum_reg[25]/D	6.031	1.412	4.619	10.0	clk
Path 4	3.835	8	6	b_r_reg[0]/C	sum_reg[28]/D	6.149	1.510	4.639	10.0	clk
Path 5	3.858	8	6	b_r_reg[0]/C	sum_reg[31]/D	6.077	1.510	4.567	10.0	clk
Path 6	3.860	8	6	b_r_reg[0]/C	cout_reg/D	6.073	1.510	4.563	10.0	clk
Path 7	3.978	7	6	b_r_reg[0]/C	sum_reg[21]/D	6.004	1.582	4.422	10.0	clk
Path 8	4.002	7	6	b_r_reg[0]/C	sum_reg[26]/D	5.857	1.386	4.471	10.0	clk
Path 9	4.027	7	6	b_r_reg[0]/C	sum_reg[22]/D	5.996	1.574	4.422	10.0	clk
Path 10	4.048	7	6	b_r_reg[0]/C	sum_reg[24]/D	5.887	1.582	4.305	10.0	clk

1. **Path-1 :**

   Source(Launch FF): b_r_reg[0]/C
   Destination (Capture FF): sum_reg[30]/D
   Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 5.228ns
   Data path : FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU = 6.786ns.
   Arrival time : 5.228 + 6.786 = **12.014ns.**
   Required time : clock clk rise edge + net + buff + cp + cu = **15.069ns.**

   Slack = Required time - Arrival time
   = 15.069 – 12.014
   = **3.055ns**

2. **Path-2 :**

   Source(Launch FF): b_r_reg[0]/C
   Destination (Capture FF): sum_reg[29]/D
   Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 5.228ns
   Data path : FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU = 6.699ns.
   Arrival time : 5.228 + 6.786 = **11.926ns.**
   Required time : clock clk rise edge + net + buff + cp + cu = **15.064ns.**

   Slack = Required time - Arrival time
   = 15.064 – 11.926
   = **3.138ns**

3. **Path-3 :**

Source(Launch FF):	b_r_reg[0]/C	
Destination (Capture FF):	sum_reg[25]/D	
Source clock path:	clock clk rise edge + net + buffer + FDCE (Prop_fdce_C)	= 5.228ns
Data path :	FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU	= 6.031ns.
Arrival time :	5.228 + 6.031 = **11.259ns.**	
Required time :	clock clk rise edge + net + buff + cp + cu	= **14.901ns.**

Slack = Required time - Arrival time
= 14.901 – 11.259
= **3.642ns**

4. **Path-4** :

Source(Launch FF):	b_r_reg[0]/C	
Destination (Capture FF):	sum_reg[28]/D	
Source clock path:	clock clk rise edge + net + buffer + FDCE (Prop_fdce_C)	= 5.228ns
Data path :	FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU	= 6.149ns.
Arrival time :	5.228 + 6.149 = **11.377ns.**	
Required time :	clock clk rise edge + net + buff + cp + cu	= **15.212ns.**

Slack = Required time - Arrival time
= 15.212 – 11.377
= **3.835ns**

5. **Path-5** :

Source(Launch FF):	b_r_reg[0]/C	
Destination (Capture FF):	sum_reg[31]/D	
Source clock path:	clock clk rise edge + net + buffer + FDCE (Prop_fdce_C)	= 5.228ns
Data path :	FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU	= 6.077ns.
Arrival time :	5.228 + 6.077 = **11.304ns.**	
Required time :	clock clk rise edge + net + buff + cp + cu	= **15.162ns.**

Slack = Required time - Arrival time
= 15.162 – 11.304
= **3.858ns**

6. **Path-6** :

Source(Launch FF):	b_r_reg[0]/C	
Destination (Capture FF):	cout_reg/D	
Source clock path:	clock clk rise edge + net + buffer + FDCE (Prop_fdce_C)	= 5.228ns
Data path :	FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU	= 6.073ns.
Arrival time :	5.228 + 6.073 = **11.300ns.**	
Required time :	clock clk rise edge + net + buff + cp + cu	= **15.160ns.**

Slack = Required time - Arrival time
= 15.160 – 11.300
= **3.860ns**

7. **Path-7** :

Source(Launch FF):	b_r_reg[0]/C	
Destination (Capture FF):	sum_reg[21]/D	
Source clock path:	clock clk rise edge + net + buffer + FDCE (Prop_fdce_C)	= 5.228ns
Data path :	FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU	= 6.004ns.

Arrival time : 5.228 + 6.004 = **11.232ns.**
Required time : clock clk rise edge + net + buff + cp + cu = **15.210ns.**

Slack = Required time - Arrival time
= 15.210 – 11.232
= **3.978ns**

8. **Path-8** :
Source(Launch FF): b_r_reg[0]/C
Destination (Capture FF): sum_reg[26]/D
Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 5.228ns
Data path : FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU = 5.857ns.
Arrival time : 5.228 + 5.857 = **11.084ns.**
Required time : clock clk rise edge + net + buff + cp + cu = **15.086ns.**

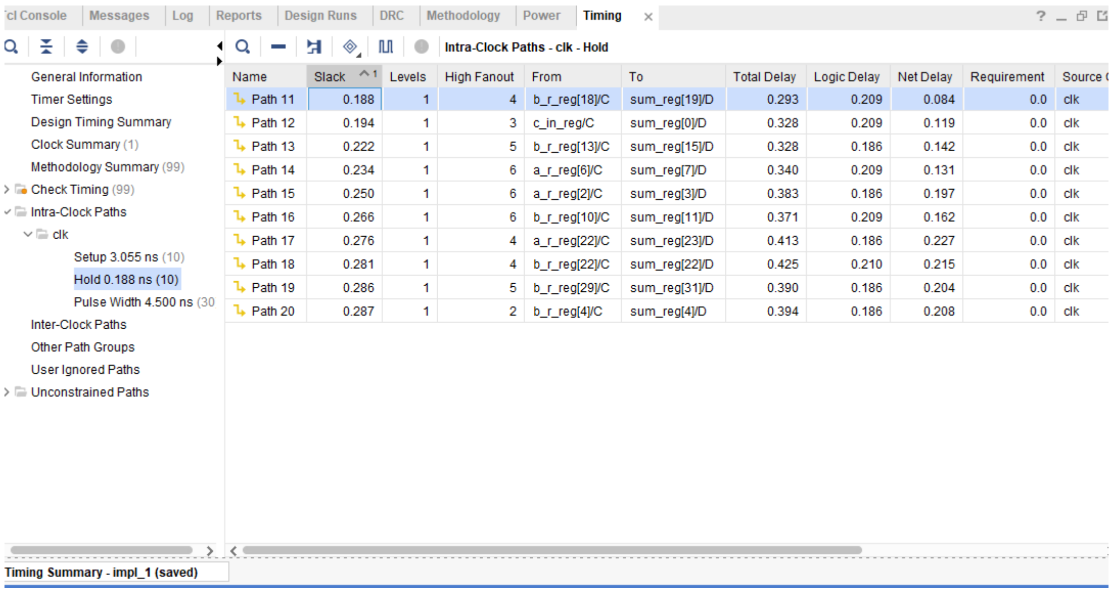
Slack = Required time - Arrival time
= 15.086 – 11.084
= **4.002ns**

9. **Path-9** :
Source(Launch FF): b_r_reg[0]/C
Destination (Capture FF): sum_reg[22]/D
Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 5.228ns
Data path : FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU = 5.996ns.
Arrival time : 5.228 + 5.996 = **11.224ns.**
Required time : clock clk rise edge + net + buff + cp + cu = **15.251ns.**

Slack = Required time - Arrival time
= 15.251 – 11.224
= **4.027ns**

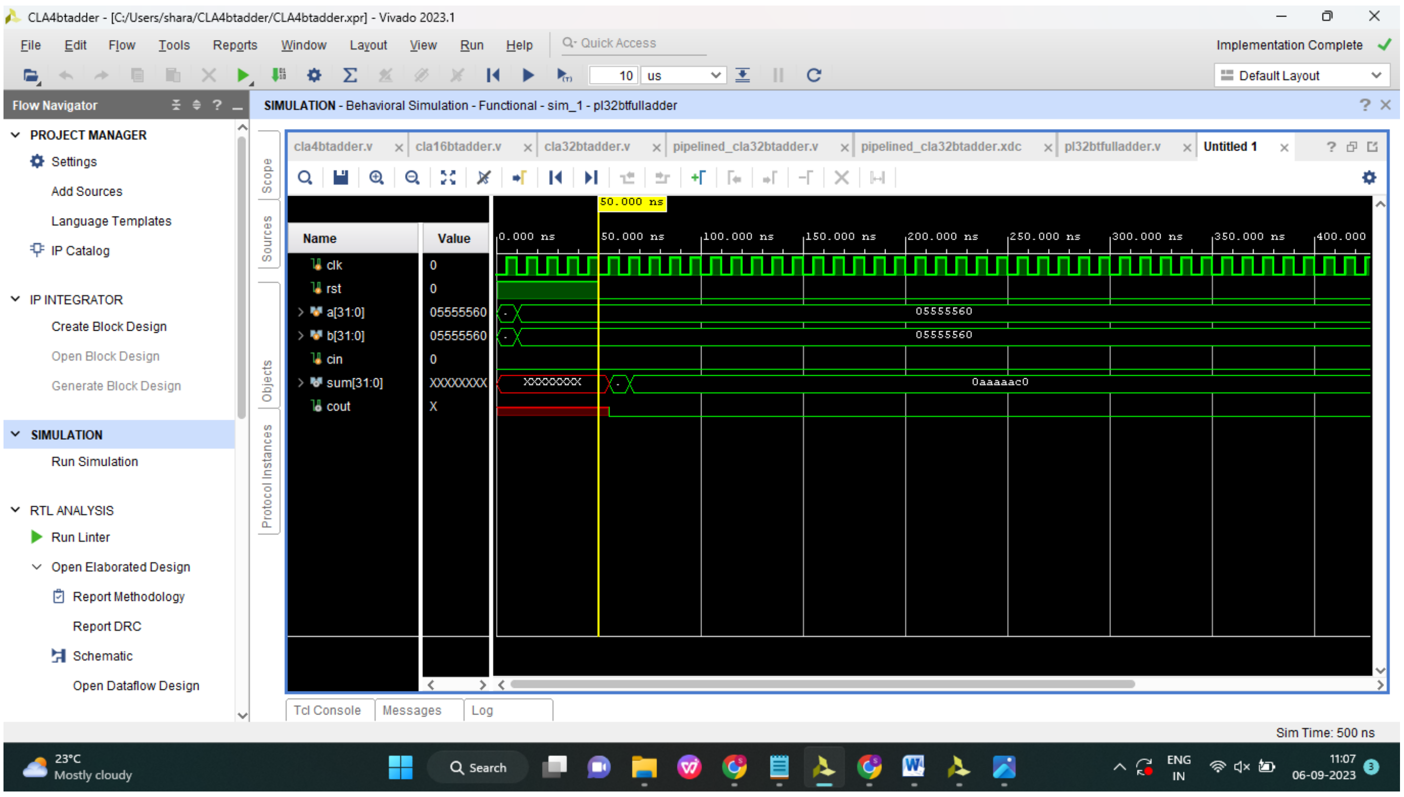
10. **Path-10** :
Source(Launch FF): b_r_reg[0]/C
Destination (Capture FF): sum_reg[24]/D
Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 5.228ns
Data path : FDCE (Prop_fdce_C_Q) + net + buffer + LUT + SU = 5.887ns.
Arrival time : 5.228 + 5.887 = **11.115ns.**
Required time : clock clk rise edge + net + buff + cp + cu = **15.162ns.**

Slack = Required time - Arrival time
= 15.162 – 11.115
= **4.048ns**



Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source
Path 11	0.188	1	4	b_r_reg[18]/C	sum_reg[19]/D	0.293	0.209	0.084	0.0	clk
Path 12	0.194	1	3	c_in_reg/C	sum_reg[0]/D	0.328	0.209	0.119	0.0	clk
Path 13	0.222	1	5	b_r_reg[13]/C	sum_reg[15]/D	0.328	0.186	0.142	0.0	clk
Path 14	0.234	1	6	a_r_reg[6]/C	sum_reg[7]/D	0.340	0.209	0.131	0.0	clk
Path 15	0.250	1	6	a_r_reg[2]/C	sum_reg[3]/D	0.383	0.186	0.197	0.0	clk
Path 16	0.266	1	6	b_r_reg[10]/C	sum_reg[11]/D	0.371	0.209	0.162	0.0	clk
Path 17	0.276	1	4	a_r_reg[22]/C	sum_reg[23]/D	0.413	0.186	0.227	0.0	clk
Path 18	0.281	1	4	b_r_reg[22]/C	sum_reg[22]/D	0.425	0.210	0.215	0.0	clk
Path 19	0.286	1	5	b_r_reg[29]/C	sum_reg[31]/D	0.390	0.186	0.204	0.0	clk
Path 20	0.287	1	2	b_r_reg[4]/C	sum_reg[4]/D	0.394	0.186	0.208	0.0	clk

Waveform:


CLA4btadder - [C:/Users/shara/CLA4btadder/CLA4btadder.xpr] - Vivado 2023.1

SIMULATION - Behavioral Simulation - Functional - sim_1 - pl32btfulladder

Name	Value
clk	0
rst	0
a[31:0]	05555560
b[31:0]	05555560
cin	0
sum[31:0]	XXXXXXXX
cout	X

CONCLUSION

Carry look ahead adder employed the great importance to reducing carry propagation delay of the adder. Though compared with other different logic design approaches CLA logic calculates carry propagating to the next stage as soon as inputs are applied. So the speed of addition process is faster.