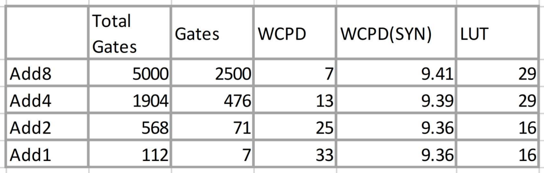
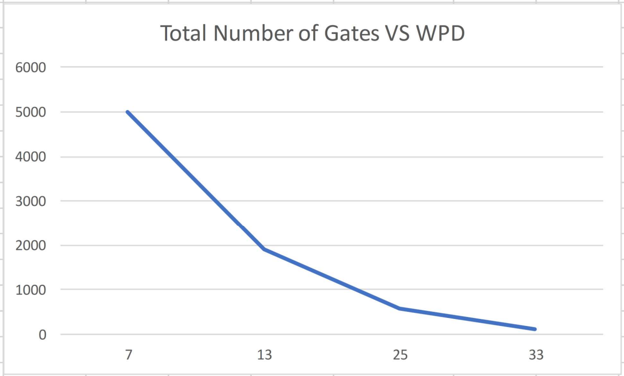
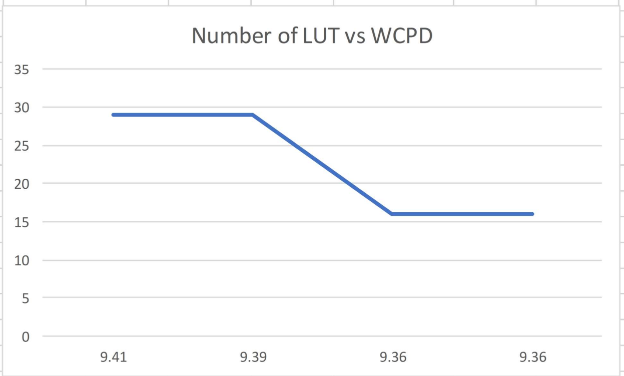
Lab 1 Report

Tuesday, February 15, 2022 2:56 PM



	Total Gates	Gates	WCPD	WCPD(SYN)	LUT
Add8	5000	2500	7	9.41	29
Add4	1904	476	13	9.39	29
Add2	568	71	25	9.36	16
Add1	112	7	33	9.36	16





How does this data compare to the data found in Lab 1 (where we used 1 unit gate delay per logic gate and area of one gate for each gate)?

This compares to data find in Lab 1 because there is no relatable trend to the data that we found in the simulation to the synthesis

Is the data similar (similar trends)? Why or why not?

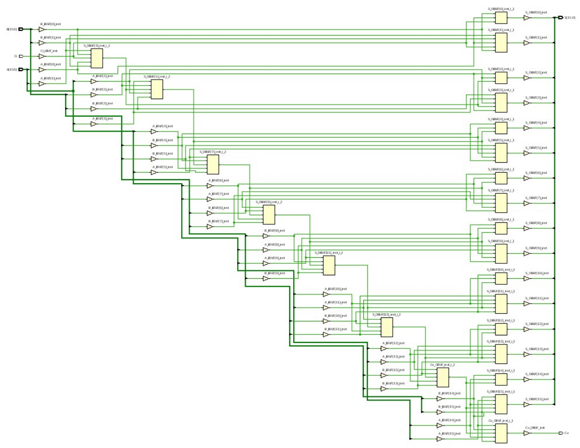
There is no real relatable trend to the data because the amount of bits added in the adder doesn't change so the number of lookup tables should remain the same. The number of lookup tables may have changed in our case because how it was implemented.

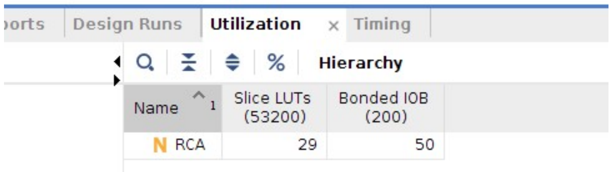
How could you make the design faster than what is reported?

You could make the design faster by decreasing the amount of bits added in the circuit thus making the circuit smaller. You could also make a dedicated 16 bit adder and have the propagation delay of 3 for only one gate needed but designing this would be a nightmare to do with our methods that we have.

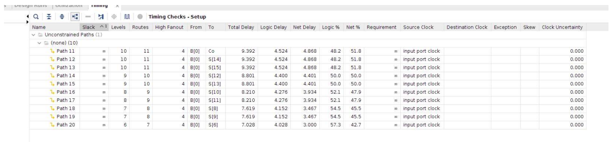
How could you make it smaller?

Add 2





Name	Slice LUTs (53200)	Bonded IOB (200)
RCA	29	50

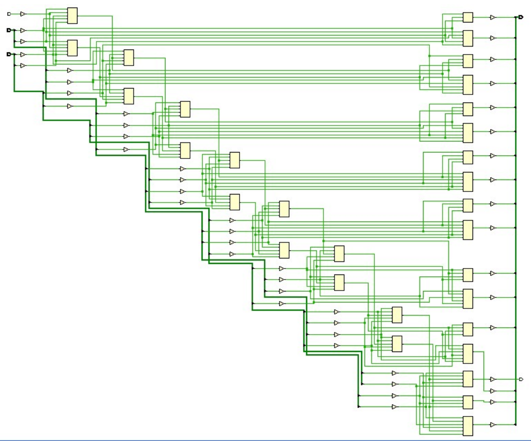


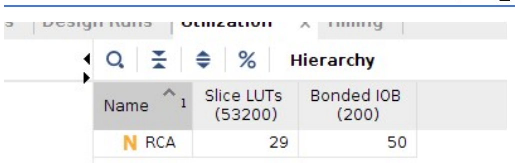
Timing Checks - Setup

Unconstrained Paths (1)
(none) (10)

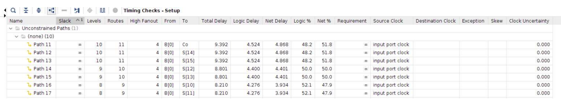
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exception	Skew	Clock Uncertainty
Path 11	∞	10	11	4	B[0]	Co	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 12	∞	10	11	4	B[0]	S[14]	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 13	∞	10	11	4	B[0]	S[15]	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 14	∞	9	10	4	B[0]	S[12]	8.801	4.400	4.401	50.0	50.0	∞	input port clock				0.000
Path 15	∞	9	10	4	B[0]	S[13]	8.801	4.400	4.401	50.0	50.0	∞	input port clock				0.000
Path 16	∞	8	9	4	B[0]	S[10]	8.210	4.276	3.934	52.1	47.9	∞	input port clock				0.000
Path 17	∞	8	9	4	B[0]	S[11]	8.210	4.276	3.934	52.1	47.9	∞	input port clock				0.000
Path 18	∞	7	8	4	B[0]	S[8]	7.619	4.152	3.467	54.5	45.5	∞	input port clock				0.000
Path 19	∞	7	8	4	B[0]	S[9]	7.619	4.152	3.467	54.5	45.5	∞	input port clock				0.000
Path 20	∞	6	7	4	B[0]	S[6]	7.028	4.028	3.000	57.3	42.7	∞	input port clock				0.000

Add 4





Name	Slice LUTs (53200)	Bonded IOB (200)
RCA	29	50



Timing Checks - Setup

Unconstrained Paths (1)
(none) (10)

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exception	Skew	Clock Uncertainty
Path 11	∞	10	11	4	B[0]	Co	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 12	∞	10	11	4	B[0]	S[14]	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 13	∞	10	11	4	B[0]	S[15]	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 14	∞	9	10	4	B[0]	S[12]	8.801	4.400	4.401	50.0	50.0	∞	input port clock				0.000
Path 15	∞	9	10	4	B[0]	S[13]	8.801	4.400	4.401	50.0	50.0	∞	input port clock				0.000
Path 16	∞	8	9	4	B[0]	S[10]	8.210	4.276	3.934	52.1	47.9	∞	input port clock				0.000
Path 17	∞	8	9	4	B[0]	S[11]	8.210	4.276	3.934	52.1	47.9	∞	input port clock				0.000

designing this would be a nightmare to do with our methods that we have.

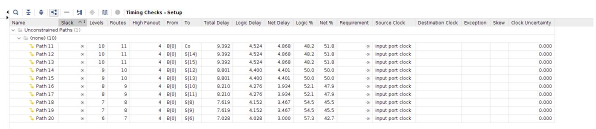
How could you make it smaller?

You could use the ripple carry adder and decrease the amount of adders needed to solve your problem

Given a specification for an N-bit Adder design (for example given a WPD), what would be your approach to meeting the required specifications, area or delay?

It would depend on what is the priority. If speed is the priority then I would use a higher adder circuit for a fast circuit. If I need to design something that doesn't take up a lot of space then I would design the ripple carry adder. Somewhere in the middle of that is a good way to start your design with small area but somewhat fast speed.

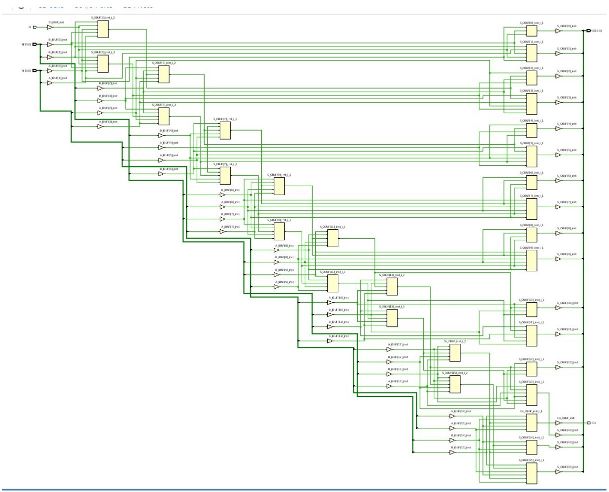
```
-- rca__w_fa.vhd
-- Ripple Carry Adder with full adder subcomponent

-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
library IEEE,WORK;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FA is
        port(   A,B,Ci: in std_logic;
                Co,S: out std_logic);
end;

architecture FA_BEHAV of FA is
begin
  Co <= (A and B) or (A and Ci) or (B and Ci)
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
  S <= (A xor B) xor Ci
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
end;

-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
library IEEE,WORK;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity RCA is
        generic(N:integer:=16);             -- defaults to a 16 bit adder
        port(   A,B: in std_logic_vector(N-1 downto 0);
         Ci: in std_logic;
         Co: out std_logic;
                S: out std_logic_vector(N-1 downto 0));
end;

architecture RCA_STRUCT of RCA is

  -- declarative area

  component FA
    port(A,B,Ci:in std_logic;Co,S:out std_logic);
  end component;

  signal C:std_logic_vector(N downto 0);

begin

  -- instantiation area

  C(0) <= Ci;

  GI: for I in 0 to N-1 generate
    GI:FA port map(A => A(I),B => B(I), Ci => C(I),Co => C(I+1),S => S(I));
  end generate;

  Co <= C(N);

end;
```

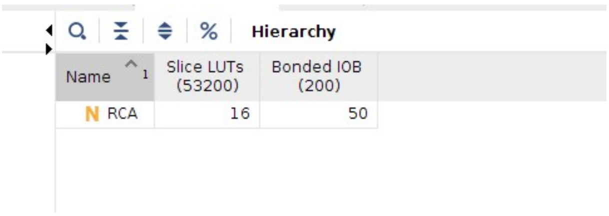


Timing Checks - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exception	Skew	Clock Uncertainty
Unconstrained Paths (1)																	
(none) (10)																	
Path 11	∞	10	11	4	B[0]	Co	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 12	∞	10	11	4	B[0]	S[14]	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 13	∞	10	11	4	B[0]	S[15]	9.392	4.524	4.868	48.2	51.8	∞	input port clock				0.000
Path 14	∞	9	10	4	B[0]	S[12]	8.801	4.400	4.401	50.0	50.0	∞	input port clock				0.000
Path 15	∞	9	10	4	B[0]	S[13]	8.801	4.400	4.401	50.0	50.0	∞	input port clock				0.000
Path 16	∞	8	9	4	B[0]	S[10]	8.210	4.276	3.934	52.1	47.9	∞	input port clock				0.000
Path 17	∞	8	9	4	B[0]	S[11]	8.210	4.276	3.934	52.1	47.9	∞	input port clock				0.000
Path 18	∞	7	8	4	B[0]	S[8]	7.619	4.152	3.467	54.5	45.5	∞	input port clock				0.000
Path 19	∞	7	8	4	B[0]	S[9]	7.619	4.152	3.467	54.5	45.5	∞	input port clock				0.000
Path 20	∞	6	7	4	B[0]	S[6]	7.028	4.028	3.000	57.3	42.7	∞	input port clock				0.000

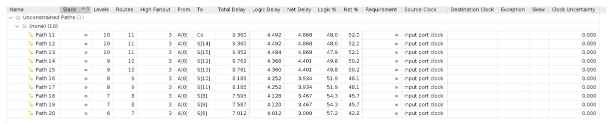
Add 1





Hierarchy

Name	Slice LUTs (53200)	Bonded IOB (200)
RCA	16	50



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exception	Skew	Clock Uncertainty
Unconstrained Paths (1)																	
(none) (10)																	
Path 11	∞	10	11	3	A[0]	Co	9.360	4.492	4.868	48.0	52.0	∞	input port clock				0.000
Path 12	∞	10	11	3	A[0]	S[14]	9.360	4.492	4.868	48.0	52.0	∞	input port clock				0.000
Path 13	∞	10	11	3	A[0]	S[15]	9.352	4.484	4.868	47.9	52.1	∞	input port clock				0.000
Path 14	∞	9	10	3	A[0]	S[12]	8.769	4.368	4.401	49.8	50.2	∞	input port clock				0.000
Path 15	∞	9	10	3	A[0]	S[13]	8.761	4.360	4.401	49.8	50.2	∞	input port clock				0.000
Path 16	∞	8	9	3	A[0]	S[10]	8.186	4.252	3.934	51.9	48.1	∞	input port clock				0.000
Path 17	∞	8	9	3	A[0]	S[11]	8.186	4.252	3.934	51.9	48.1	∞	input port clock				0.000
Path 18	∞	7	8	3	A[0]	S[8]	7.595	4.128	3.467	54.3	45.7	∞	input port clock				0.000
Path 19	∞	7	8	3	A[0]	S[9]	7.587	4.120	3.467	54.3	45.7	∞	input port clock				0.000
Path 20	∞	6	7	3	A[0]	S[6]	7.012	4.012	3.000	57.2	42.8	∞	input port clock				0.000

```vhdl
-- rca__w_add2.vhd
-- ripple carry adder with 2-bit adder (ADD2) subcomponent
-- RCA generic integer N must be even (a multiple of 2)
-- what would be the limitation of N for RCA with ADD4 subcomponent

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
library IEEE,WORK;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- ADD2 is a two-level logic (ignoring inverters) circuit that adds two, 2-bit numbers
-- you must design this block
entity ADD2 is
        port(   A,B: in std_logic_vector(1 downto 0);
                Ci: in std_logic;
                Co: out std_logic;
                S: out std_logic_vector(1 downto 0));
end;

architecture ADD2_BEHAV of ADD2 is
begin
  Co <= (A(1) and B(1)) or (Ci and A(0) and B(1)) or (Ci and A(1) and A(0)) or (A(1) and A(0) and B(0)) or (Ci and A(1) and
B(0)) or (Ci and B(1) and B(0)) or (A(0) and B(1) and B(0))
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
  S(0) <= (not(Ci) and not(A(0)) and B(0)) or (not(Ci) and A(0) and not(B(0))) or (Ci and not(A(0)) and not(B(0))) or (Ci
and A(0) and B(0))
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
  S(1) <= (not(Ci) and not(A(1)) and not(A(0)) and B(1)) or (not(Ci) and not(A(1)) and B(1) and not(B(0))) or (not(A(1))
and not(A(0)) and B(1) and not(B(0))) or (not(Ci) and A(1) and not(A(0)) and not(B(1))) or (not(Ci) and A(1) and not(B(1))
and not(B(0))) or (A(1) and not(A(0)) and not(B(1)) and not(B(0))) or (not(A(1)) and A(0) and not(B(1)) and B(0)) or (Ci
and not(A(1)) and not(B(1)) and B(0)) or (Ci and not(A(1)) and A(0) and not(B(1))) or (A(1) and A(0) and B(1) and B(0)) or
(Ci and A(1) and B(1) and B(0)) or (Ci and A(1) and A(0) and B(1))
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
end;

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
library IEEE,WORK;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity RCA is
        generic(N:integer:=16);              -- defaults to a 16 bit adder
        port(   A,B: in std_logic_vector(N-1 downto 0);
         Ci: in std_logic;
         Co: out std_logic;
                S: out std_logic_vector(N-1 downto 0));
end;

architecture RCA_STRUCT of RCA is

  -- declarative area

  component ADD2
    port(A,B:in std_logic_vector(1 downto 0);Ci:in std_logic;Co:out std_logic;S:out std_logic_vector(1 downto 0));
  end component;

  signal C:std_logic_vector(N/2 downto 0);

begin

  -- it helps to draw this out and label the signal lines
  -- instantiation area

  C(0) <= Ci;

  GI: for I in 0 to N/2-1 generate
    GI:ADD2
      port map(A(1)=>A(2*I+1),A(0)=>A(2*I),B(1)=>B(2*I+1),B(0)=>B(2*I),Ci=>C(I),Co=>C(I+1),S(1)=>S(2*I+1),S(0)=>S(2*I));
  end generate;

  Co <= C(N/2);

end;
```

```vhdl
-- rca__w_add4.vhd
-- ripple carry adder with 2-bit adder (ADD4) subcomponent
-- RCA generic integer N must be even (a multiple of 4)
-- what would be the limitation of N for RCA with ADD4 subcomponent

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
library IEEE,WORK;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- ADD2 is a two-level logic (ignoring inverters) circuit that adds two, 2-bit numbers
-- you must design this block
entity ADD4 is
        port(   A,B: in std_logic_vector(3 downto 0);
                Ci: in std_logic;
                Co: out std_logic;
                S: out std_logic_vector(3 downto 0));
end;

architecture ADD4_BEHAV of ADD4 is
begin
  Co <= (A(3) and B(3)) or (A(3) and A(2) and B(2)) or (A(2) and B(3) and B(2)) or (A(3) and A(2) and A(1) and B(1))
or (A(2) and A(1) and B(3) and B(1)) or (A(3) and A(1) and B(2) and B(1)) or (A(1) and B(3) and B(2) and B(1)) or
(A(2) and A(1) and A(0) and B(3) and B(0)) or (A(3) and A(2) and A(1) and A(0) and B(0)) or (A(2) and A(1) and A(0)
and B(3) and Ci) or (A(3) and A(2) and A(1) and A(0) and Ci) or (A(3) and A(1) and A(0) and B(2) and B(0)) or (A(3)
and A(1) and A(0) and B(2) and Ci) or (A(3) and A(2) and A(0) and B(1) and B(0)) or (A(3) and A(2) and A(0) and B(1)
and Ci) or (A(3) and A(2) and A(1) and B(0) and Ci) or (A(2) and A(1) and B(3) and B(0) and Ci) or (A(3) and A(1) and
B(2) and B(0) and Ci) or (A(3) and A(2) and B(1) and B(0) and Ci) or (A(2) and A(0) and B(3) and B(1) and B(0)) or
(A(2) and A(0) and B(3) and B(1) and Ci) or (A(3) and A(0) and B(2) and B(1) and B(0)) or (A(3) and A(0) and B(2) and
B(1) and Ci) or (A(2) and B(3) and B(1) and B(0) and Ci) or (A(3) and B(2) and B(1) and B(0) and Ci) or (A(1) and
A(0) and B(3) and B(2) and B(0)) or (A(1) and A(0) and B(3) and B(2) and Ci) or (A(1) and B(3) and B(2) and B(0) and
Ci) or (A(0) and B(3) and B(2) and B(1) and B(0)) or (A(0) and B(3) and B(2) and B(1) and Ci) or (B(3) and B(2) and
B(1) and B(0) and Ci)
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
  S(0) <= (not(A(0)) and not(B(0)) and Ci) or (not(A(0)) and B(0) and not(Ci)) or (A(0) and not(B(0)) and not(Ci)) or
(A(0) and B(0) and Ci)
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
  S(1) <= (not(A(1)) and not(A(0)) and B(1) and not(B(0))) or (A(1) and not(A(0)) and not(B(1)) and not(B(0))) or
(not(A(1)) and not(A(0)) and B(1) and not(Ci)) or (A(1) and not(A(0)) and not(B(1)) and not(Ci)) or (not(A(1)) and
B(1) and not(B(0)) and not(Ci)) or (A(1) and not(B(1)) and not(B(0)) and not(Ci)) or (not(A(1)) and A(0) and
not(B(1)) and B(0)) or (not(A(1)) and A(0) and not(B(1)) and Ci) or (not(A(1)) and not(B(1)) and B(0) and Ci) or
(A(1) and A(0) and B(1) and B(0)) or (A(1) and A(0) and B(1) and Ci) or (A(1) and B(1) and B(0) and Ci)
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
  S(2) <= (not(A(2)) and not(A(1)) and B(2) and not(B(1))) or (A(2) and not(A(1)) and not(B(2)) and not(B(1))) or
(not(A(2)) and A(1) and not(B(2)) and B(1)) or (A(2) and A(1) and B(2) and B(1)) or (not(A(2)) and not(A(1)) and
not(A(0)) and B(2) and not(B(0))) or (A(2) and not(A(1)) and not(A(0)) and not(B(2)) and not(B(0))) or (not(A(2)) and
not(A(1)) and not(A(0)) and B(2) and not(Ci)) or (A(2) and not(A(1)) and not(A(0)) and not(B(2)) and not(Ci)) or
(not(A(2)) and not(A(1)) and B(2) and not(B(0)) and not(Ci)) or (A(2) and not(A(1)) and not(B(2)) and not(B(0)) and
not(Ci)) or (not(A(2)) and not(A(0)) and B(2) and not(B(1)) and not(B(0))) or (A(2) and not(A(0)) and not(B(2)) and
not(B(1)) and not(B(0))) or (not(A(2)) and not(A(0)) and B(2) and not(B(1)) and not(Ci)) or (A(2) and not(A(0)) and
not(B(2)) and not(B(1)) and not(Ci)) or (not(A(2)) and B(2) and not(B(1)) and not(B(0)) and not(Ci)) or (A(2) and
not(B(2)) and not(B(1)) and not(B(0)) and not(Ci)) or (not(A(2)) and A(1) and A(0) and not(B(2)) and B(0)) or
(not(A(2)) and A(1) and A(0) and not(B(2)) and Ci) or (not(A(2)) and A(1) and not(B(2)) and B(0) and Ci) or
(not(A(2)) and A(0) and not(B(2)) and B(1) and B(0)) or (not(A(2)) and A(0) and not(B(2)) and B(1) and Ci) or
(not(A(2)) and not(B(2)) and B(1) and B(0) and Ci) or (A(2) and A(1) and A(0) and B(2) and B(0)) or (A(2) and A(1)
and A(0) and B(2) and Ci) or (A(2) and A(1) and B(2) and B(0) and Ci) or (A(2) and A(0) and B(2) and B(1) and B(0))
or (A(2) and A(0) and B(2) and B(1) and Ci) or (A(2) and B(2) and B(1) and B(0) and Ci)
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;
  S(3) <= (not(A(3)) and not(A(2)) and B(3) and not(B(2))) or (A(3) and not(A(2)) and not(B(3)) and not(B(2))) or
(not(A(3)) and A(2) and not(B(3)) and B(2)) or (A(3) and A(2) and B(3) and B(2)) or (not(A(3)) and not(A(2)) and
not(A(1)) and B(3) and not(B(1))) or (A(3) and not(A(2)) and not(A(1)) and not(B(3)) and not(B(1))) or (not(A(3)) and
not(A(1)) and B(3) and not(B(2)) and not(B(1))) or (A(3) and not(A(1)) and not(B(3)) and not(B(2)) and not(B(1))) or
(not(A(3)) and A(2) and A(1) and not(B(3)) and B(1)) or (not(A(3)) and A(1) and not(B(3)) and B(2) and B(1)) or (A(3)
and A(2) and A(1) and B(3) and B(1)) or (A(3) and A(1) and B(3) and B(2) and B(1)) or (not(A(3)) and not(A(2)) and
not(A(1)) and not(A(0)) and B(3) and not(B(0))) or (not(A(3)) and not(A(1)) and not(A(0)) and B(3) and not(B(2)) and
not(B(0))) or (A(3) and not(A(2)) and not(A(1)) and not(A(0)) and not(B(3)) and not(B(0))) or (A(3) and not(A(1)) and
not(A(0)) and not(B(3)) and not(B(2)) and not(B(0))) or (not(A(3)) and not(A(2)) and not(A(1)) and not(A(0)) and B(3)
and not(Ci)) or (not(A(3)) and not(A(1)) and not(A(0)) and B(3) and not(B(2)) and not(Ci)) or (A(3) and not(A(2)) and
not(A(1)) and not(A(0)) and not(B(3)) and not(Ci)) or (A(3) and not(A(1)) and not(A(0)) and not(B(3)) and not(B(2))
and not(Ci)) or (not(A(3)) and not(A(2)) and not(A(0)) and B(3) and not(B(1)) and not(B(0))) or (A(3) and not(A(2))
and not(A(0)) and not(B(3)) and not(B(1)) and not(B(0))) or (not(A(3)) and not(A(2)) and not(A(0)) and B(3) and
not(B(1)) and not(Ci)) or (A(3) and not(A(2)) and not(A(0)) and not(B(3)) and not(B(1)) and not(Ci)) or (not(A(3))
and not(A(2)) and not(A(1)) and B(3) and not(B(0)) and not(Ci)) or (A(3) and not(A(2)) and not(A(1)) and not(B(3))
and not(B(0)) and not(Ci)) or (not(A(3)) and not(A(1)) and B(3) and not(B(2)) and not(B(0)) and not(Ci)) or (A(3) and
not(A(1)) and not(B(3)) and not(B(2)) and not(B(0)) and not(Ci)) or (not(A(3)) and not(A(2)) and B(3) and not(B(1))
and not(B(0)) and not(Ci)) or (A(3) and not(A(2)) and not(B(3)) and not(B(1)) and not(B(0)) and not(Ci)) or
(not(A(3)) and not(A(0)) and B(3) and not(B(2)) and not(B(1)) and not(B(0))) or (A(3) and not(A(0)) and not(B(3)) and
not(B(2)) and not(B(1)) and not(B(0))) or (not(A(3)) and not(A(0)) and B(3) and not(B(2)) and not(B(1)) and not(Ci))
or (A(3) and not(A(0)) and not(B(3)) and not(B(2)) and not(B(1)) and not(Ci)) or (not(A(3)) and B(3) and not(B(2))
and not(B(1)) and not(B(0)) and not(Ci)) or (A(3) and not(B(3)) and not(B(2)) and not(B(1)) and not(B(0)) and
not(Ci)) or (not(A(3)) and A(1) and A(0) and not(B(3)) and B(2) and B(0)) or (not(A(3)) and A(2) and A(1) and A(0)
and not(B(3)) and B(0)) or (not(A(3)) and A(1) and A(0) and not(B(3)) and B(2) and Ci) or (not(A(3)) and A(2) and
A(1) and A(0) and not(B(3)) and Ci) or (not(A(3)) and A(2) and A(0) and not(B(3)) and B(1) and B(0)) or (not(A(3))
and A(2) and A(0) and not(B(3)) and B(1) and Ci) or (not(A(3)) and A(2) and A(1) and not(B(3)) and B(0) and Ci) or
(not(A(3)) and A(1) and not(B(3)) and B(2) and B(0) and Ci) or (not(A(3)) and A(2) and not(B(3)) and B(1) and B(0)
and Ci) or (not(A(3)) and A(0) and not(B(3)) and B(2) and B(1) and B(0)) or (not(A(3)) and A(0) and not(B(3)) and
B(2) and B(1) and Ci) or (not(A(3)) and not(B(3)) and B(2) and B(1) and B(0) and Ci) or (A(3) and A(1) and A(0) and
B(3) and B(2) and B(0)) or (A(3) and A(2) and A(1) and A(0) and B(3) and B(0)) or (A(3) and A(1) and A(0) and B(3)
and B(2) and Ci) or (A(3) and A(2) and A(1) and A(0) and B(3) and Ci) or (A(3) and A(2) and A(0) and B(3) and B(1)
and B(0)) or (A(3) and A(2) and A(0) and B(3) and B(1) and Ci) or (A(3) and A(2) and A(1) and B(3) and B(0) and Ci)
or (A(3) and A(1) and B(3) and B(2) and B(0) and Ci) or (A(3) and A(2) and B(3) and B(1) and B(0) and Ci) or (A(3)
and A(0) and B(3) and B(2) and B(1) and B(0)) or (A(3) and A(0) and B(3) and B(2) and B(1) and Ci) or (A(3) and B(3)
and B(2) and B(1) and B(0) and Ci)
     -- pragma synthesis_off
     after 2 ns
     -- pragma synthesis_on
  ;

end;

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
library IEEE,WORK;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity RCA is
        generic(N:integer:=16);              -- defaults to a 16 bit adder
        port(   A,B: in std_logic_vector(N-1 downto 0);
         Ci: in std_logic;
         Co: out std_logic;
                S: out std_logic_vector(N-1 downto 0));
end;

architecture RCA_STRUCT of RCA is

  -- declarative area

  component ADD4
    port(A,B:in std_logic_vector(3 downto 0);Ci:in std_logic;Co:out std_logic;S:out std_logic_vector(3 downto 0));
  end component;

  signal C:std_logic_vector(N/4 downto 0);

begin

  -- it helps to draw this out and label the signal lines
  -- instantiation area

  C(0) <= Ci;

  GI: for I in 0 to N/4-1 generate
    GI:ADD4
      port map(
      A(0)=>A(4*I),
      A(1)=>A(4*I+1),
      A(2)=>A(4*I+2),
      A(3)=>A(4*I+3),
      B(0)=>B(4*I),
      B(1)=>B(4*I+1),
      B(2)=>B(4*I+2),
      B(3)=>B(4*I+3),
      Ci=>C(I),
      Co=>C(I+1),
      S(0)=>S(4*I),
      S(1)=>S(4*I+1),
      S(2)=>S(4*I+2),
      S(3)=>S(4*I+3));
  end generate;

  Co <= C(N/4);

end;
```

```
-- tb_RCA.vhd
-- this file can be used to simulate RCA adders.
-- instead of exhaustively simulating the adders,
-- this test bench picks random input vectors to test the adders
-- note limitations on N for the RCA subcomponents ...
```

```vhdl
library IEEE,STD,WORK;
  use IEEE.STD_LOGIC_1164.ALL;
  use IEEE.STD_LOGIC_ARITH.ALL;
  --use IEEE.STD_LOGIC_SIGNED.ALL;
  use IEEE.STD_LOGIC_UNSIGNED.ALL;
  use IEEE.STD_LOGIC_TEXTIO.ALL;
  use STD.TEXTIO.ALL;
  use IEEE.MATH_REAL.ALL;

entity TB_RCA is
  generic (
    N_ITS:integer:= 4000 ;  -- number of random vectors for random TB
    N:integer:= 16 ;        -- number of input bits for multiplier
    WPD:time:= 33 ns
  );
end ;
     -- the only change needed for each test bench is the wcpd above to account dor the different delays of the hardware design

architecture TB of TB_RCA is
  file OUT_FILE: text
    is out "sim_RCA.txt";  -- simulation output file
  component RCA
    generic(N:integer);
    port(A,B: in std_logic_vector(N-1 downto 0);
          Ci: in std_logic;
          Co: out std_logic;
           S: out std_logic_vector(N-1 downto 0));
  end component ;
  signal A,B: std_logic_vector(N-1 downto 0);
  signal S: std_logic_vector(N downto 0);
  signal Ci: std_logic;
begin
  CUT:RCA                           -- Circuit Under Test
    generic map(N=>N)
    port map (A=>A,B=>B,Ci=>Ci,Co=>S(N),S=>S(N-1 downto 0));

  test_VECTOR : process
    variable I,Cin : integer;
    variable SEED_1,SEED_2,RN1,RN2 : integer := 0;
    variable R_RAND : real;
  begin
    SEED_1 := 18; SEED_2 := 28;      -- seeds for random number generator
    I := N_ITS-1; WH_LOOPI : while I > -1 loop
      Cin := 0; WH_LOOPC : while Cin < 2 loop -- 0 to 1
        ----------------------------------
        UNIFORM(SEED_1,SEED_2,R_RAND);
        RN1 := integer((2.0**31.0)**R_RAND);
        RN1 := conv_integer(conv_std_logic_vector(RN1,N));
        UNIFORM(SEED_1,SEED_2,R_RAND);
        RN2 := integer((2.0**31.0)**R_RAND);
        RN2 := conv_integer(conv_std_logic_vector(RN2,N));
        A <= conv_std_logic_vector(RN1,N);
        B <= conv_std_logic_vector(RN2,N);
        if Cin = 1 then Ci <= '1'; else Ci <= '0'; end if;
        ----------------------------------
        wait for WPD;
        Cin := Cin + 1;
      end loop WH_LOOPC;
      I := I-1;
    end loop WH_LOOPI;
    assert (false) report "sim done :)" severity FAILURE;
  end process;

  test_RCA : process
    variable OUTLINE : LINE;
    variable COMMA : string(1 to 3):= " , ";
    variable BLANK3 : string(1 to 3):= "   ";
    variable BLANK1 : string(1 to 1):= " ";
    variable I,Cin : integer;
    variable S_INTEGER: integer;
  begin
    I := N_ITS-1; WH_LOOPI : while I > -1 loop
      Cin := 0; WH_LOOPC : while Cin < 2 loop -- 0 to 1
        wait for WPD;
        S_INTEGER := conv_integer(A) + conv_integer(B) + conv_integer(Ci);
        if conv_integer(S) /= S_INTEGER then
          WRITE(OUTLINE, A);
          WRITE(OUTLINE, '+');
          WRITE(OUTLINE, B);
          WRITE(OUTLINE, '=');
          WRITE(OUTLINE, S_INTEGER);
          WRITE(OUTLINE, '=');
          WRITE(OUTLINE, conv_std_logic_vector(S_INTEGER,N+1));
          WRITE(OUTLINE, BLANK3);
          WRITE(OUTLINE, '=');
          WRITE(OUTLINE, S);
          WRITE(OUTLINE, '=');
          WRITE(OUTLINE, conv_integer(S));
          WRITELINE(OUT_FILE, OUTLINE);
          assert (false) report "error :(" severity FAILURE;
        end if;
        Cin := Cin + 1;
      end loop WH_LOOPC;
      I := I-1;
    end loop WH_LOOPI;
  end process;

end ;

configuration CFG_TB of TB_RCA is
for TB
end for;
end ;
```