Qiao Gao

ECE 429 Section 03

Final Project: Design and Synthesis of Multi-Operand Adders

12/07/2012

Abstraction

This paper provides some implementation examples of multi-operand adder depending on different structures. More specifically, I design an 8 operands adder with four bits for each operand on three different structures separately: Linear structure of carry propagation adder, linear model of carry save adder and tree structure multi operand adder depending on [4:2] adder. Moreover, a bonus work of a 10 4-bit operands adder is also described in this paper with the use of [5:0] adder and tree structure.

Addition to the implementation of adder, this project also provides the result of simulation, synthesis and place and route with reports of timing efficiency, power consumption and area usage for each level of all the designs.

Keywords: Multi-operand, Adder

1 Introduction

The main purpose of this project is to compare different multi-operand adder structures to find a better implementation among them through actual designs and the analysis after design. As a result, Verilog codes of these designs depending on all the structures are done firstly. After that, the integrated circuit design flow is adjusted to all the design to generate the hardware level implementation and cost report of each design. Depending on these reports, we can tell the differences between adder structures and choose a better one from them.

The challenge of this project focuses on:

1. All of the design should be founded on gate level with structural Verilog in order to make perform the exact structure of specific structure.

2. The bit number of operand, result and internal values in multi-level structures are totally different from each other.

3. For cascading structures, first carry in and final carry out are nor necessary for this project since I am going to make a design without carry in and carry out.

To solve these problems, methods below are used correspondingly:

1. All of the design are made from the most basic level. I design a half adder and a full adder first at gate level description. Then, all the structures can be implemented depending on half adders, full adders and some basic logic gate.

2. Expand all the operands to the highest possible bit of the result so that all the values inside the circuit can be contained in the expended bits with no ignorance.

3. Add '0' to the carry in of the fist level in a cascading structure and add special structure to the end level to make no carry out.

All design details will be served in part 3.

2 Background

Before we start the project, the principle of different adders will be provided in this part to make the thought of our design clear. Among all the adder categories, two operands adder is the

most basic one which is the foundation of all arithmetic algorithm units such as multi-operand adder and multiplier.

2.1 Two operands adder

2.1.1 One bit adder

Two one bit operands adder is the most simple adder which has two different structures known as full adder and half adder. The expressions of these two adders are shown below:

Half adder: s=(x+y)mod2; co=(x+y)/2

Full adder: s=(x+y+ci)mod2; co=(x+y+ci)/2

The only difference between them is that full adders have carry in signals but half adders do not have, which make the implementation of half adders less complicated. As described by Ercegovac and Lang [1], the logic representations of them are:

Half adder: s=xy'+x'y; co=xy;

Hull adder: s=xy'ci'+x'yci'+x'y'ci+xyci; co=xy+xci+yci

Depending on this, we can get the implementation of a full adder below: (Figure 1)

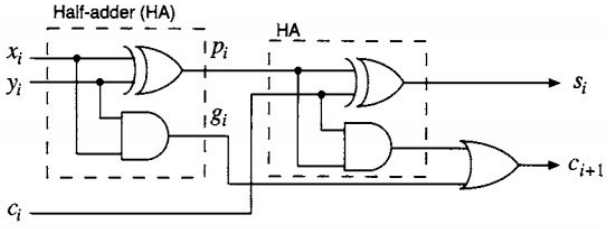


Figure 1

It shows that a full adder can be formed by two half adders and an or gate. We can include them all in a full adder module: (Figure 2)

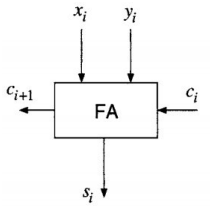


Figure 2

2.1.2 Multi-bit adder

Usually, the input operands of a adder are not in just one bit. In the book written by Ercegovac and Lang [1], some different solutions are provided to achieve two operands adder with more than one bit such as carry propagation adder (CPA), Manchester adder, carry look-ahead adder etc. In this paper, I just cover the most simple one: carry propagation adder or carry ripple adder.

The structure of a carry ripple adder is shown below: (Figure 3)

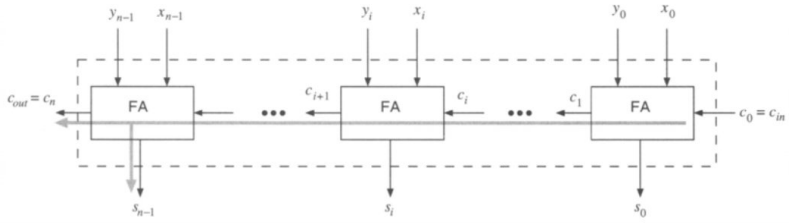


Figure 3

It shows that the sum result of this adder is generated bit by bit due to the delay of carries.

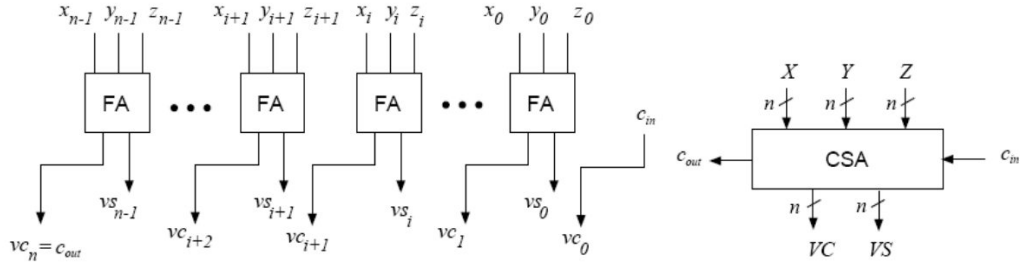
2.2 multi-operand adder

2.2.1 Carry save adder (CSA)

To implement multi-operant adder, the principle of carry save adder must be understand firstly. CSA [1] is a method that performs addition of 3 binary vectors using an array of 1-bit adders but with no propagation. Instead of generating sum and carry directly, the output of every bit is saved in two binary vector called the carry vector and pseudo-sum vector. The expression is shown below:

x+y+z = vc+vs = v

By this method three operands can be transformed to two operands which can be the inputs of a conventional two operands adder. This is the basic thought of a multi-operand adder: change the addition of multiple operands to two operands. Carry save adder is also called [3:2] adder and its structure is shown below: (Figure 4)



(a) (b)

Figure 4

By this structure, three operands can be changed to two with no more propagation delay added.

2.2.2 Extension methods to more bits

Till now we can finish the addition operation of three operands. However, what can we do if there are more than three operands? The simplest way is using linear structure of carry propagation adder, which is shown below (Figure 5):

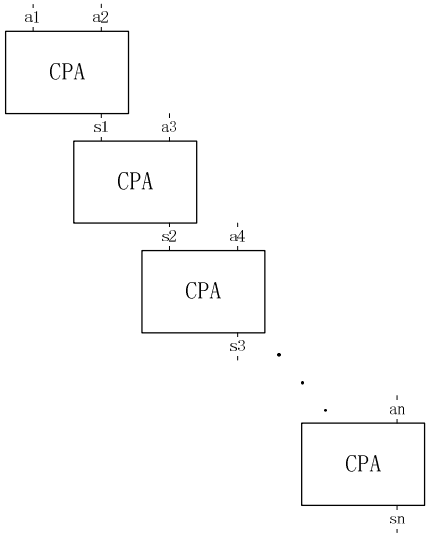


Figure 5

Obviously, the short come of this structure is the long propagation chain within CPA modules

and the interconnections between them. However, we can depend our design on carry save adders to fix this problem.

One method is using linear model [1] consisted of [p:2] adders: (Figure 6)

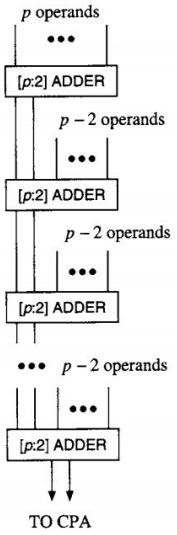


Figure 6

In this structure, delay from inputs to output can be greatly decreased comparing to the former structure due to the thought of carry save adder.

Another method is adjusting tree structure of [p:2] adders: (Figure 7)

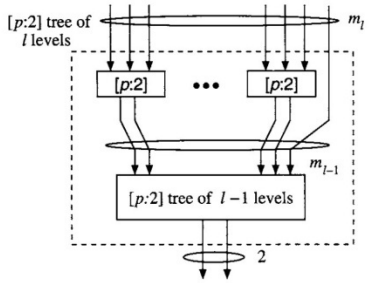


Figure 7

In this structure, all the operands are changed in parallel, which make this implementation faster than the linear model.

The goal of this project is to give the proof of the efficiency of these three structures and provide trade-offs among all these designs. Specific structures for a 8-operand 4-bit adder will be provided in part 3.

2.2.3 Word length extension

Assume that there are m operands which are represented in n bits. As a result, the result have to be perform in more bits in order to handle all expression possibilities. As described by Ercegovac and Lang [1], the range of operands have to be extended to n+p bits, in which p=$\lceil \log_2 m \rceil$. The extension method is repeating the sign bit according to the bit extension rules for 2's complement numbers, as it is shown below: (Figure 8)

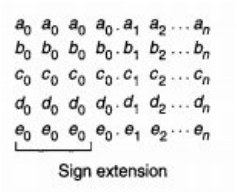


Figure 8

3 Architectural Exploration of Adders

As what has been discussed above, the main purpose of the project is finishing the implementations of an 8-operand 4-bit adder depending on CPA linear structure, CSA linear structure and tree structure and pointing out features of them by analyzing design report. In order to achieve this, firstly, we have to finish the design at RTL level with proper functions.

3.1 Word length extension

Firstly, we have to expand the length of operands. More specifically, in this design, n=4, m=8, so p=$\lceil \log_2 8 \rceil$=3. As a result, the word length of all operands should be n+p=7. So before calculations, an extension module should be adjusted to all the operands. In this situation, all the operands are unsigned number, which means we can achieve this by adding zeros like below:

000a1a2a3a4

And the structure is shown below: (Figure 9)

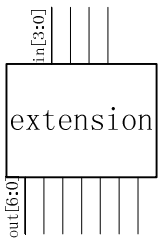


Figure 9

3.2 Full adder and half adder

As it has been described in part 2, we should design a half adder a full adder as basic elements of all the structures. In detail, the structures of full adder and half adder are shown in Figure 1. Referring to this, I made the Verilog descriptions for them as below: (Figure 10)

```
module halfadder(x,y,s,co);
    input x,y;
    output s,co;

    xor(s,x,y);
    and(co,x,y);
endmodule
```

(a) Verilog code of half adder

```
module fulladder(x,y,ci,s,co);
    input x,y,ci;
    output s,co;

    wire p,g,o;

    halfadder HA1(x,y,p,g);
    halfadder HA2(p,ci,s,o);
    or(co,o,g);

endmodule
```

(a) Verilog code of full adder

Figure 10

It is clear that the code shows the exact structure in Figure 1.

3.3 Adder Implementation

3.3.1 CPA linear structure

As indicated in Figure 3, I made the CPA implementation in Figure 11:

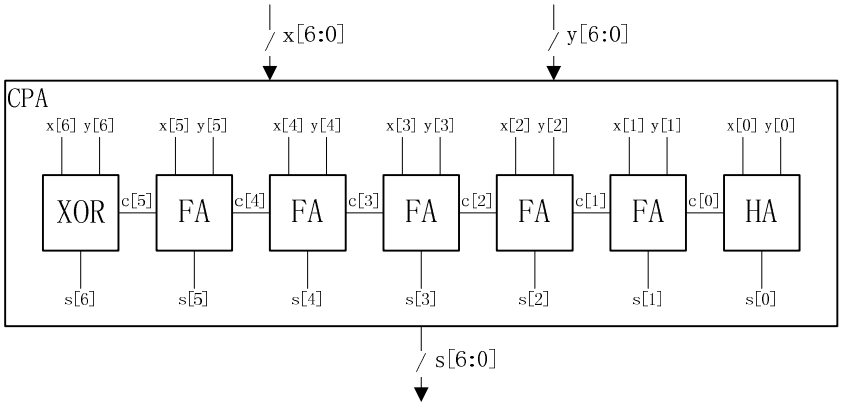


Figure 11

It can be seen that structures of Figure 3 and Figure 11 are almost the same except that there is no carry in and carry out in Figure 11. It is because we don' t need carry in and carry out for this design since there is no carry in input and carry out signals are of no use for external circuit. It will be a less efficient design if there are some ports that have never been used. So in figure 11, at the beginning of the chain, I use a half adder will no carry in and at the end point, I use an XOR to achieve the only sum output and ignore the carry out signal.

After that, I specify the structure in Figure 5 to this design and get the structure below: (Figure 12)

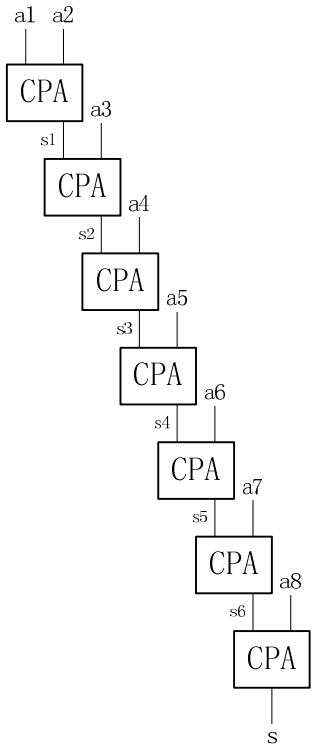


Figure 12

Though this structure may generate more propagation delay, we can get the sum result directly without additional calculation.

3.3.2 CSA linear structure

We should make an carry save adder first, with the same structure in Figure 4. Figure 13 serves my CSA solution for this design:

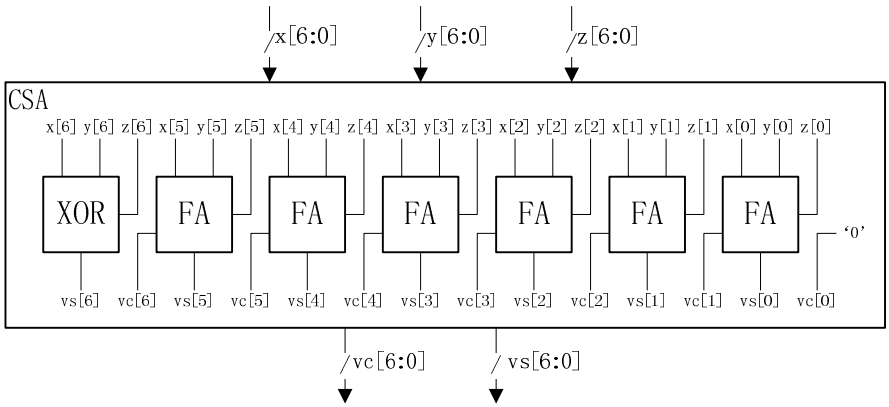


Figure 13

The same with the carry ripple adder, we have to fix the problem of carry in and carry out. In this design, I add a zero as the carry input and adder XOR and the final bit to avoid carry out. Then, I connect CSAs as Figure 6, which is known as the linear model, and get the structure below: (Figure 14)

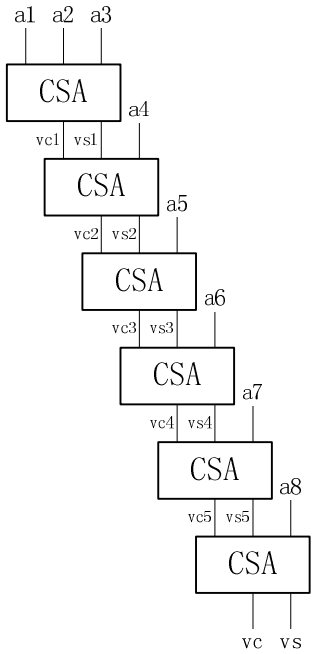


Figure 14

3.3.3 [4:2] tree structure

To make the tree structure with [4:0] adder, it is obviously that we have to make a [4:2] adder first. As it is introduced in [1], the structure of a [4:2] adder can be summarized in the figure below: (Figure 15)

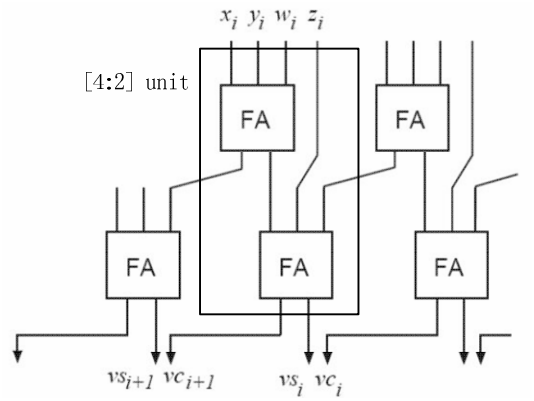


Figure 15

The part of circuit in the block in Figure 15 is the module I made to perform as a basic element of a [4:2] adder, which is named [4:2] unit. This part change one column of all the operands to two bits with a carry in and a carry out. However, this structure will not make extra propagation since the carry out the carry in do not have any logic relationship, which means all the carries can be generate in parallel without any propagation delay. The Verilog description is shown in Figure 16:

```
module adder_4_2_unit(x,y,w,z,ci,vs,vc,co);
    input x,y,w,z,ci;
    output vs,vc,co;

    wire s1;

    fulladder FA1(x,y,w,s1,co);
    fulladder FA2(s1,z,ci,vs,vc);
endmodule
```

Figure 16

Then, I connect them as below to make a [4:2] adder: (Figure 17)

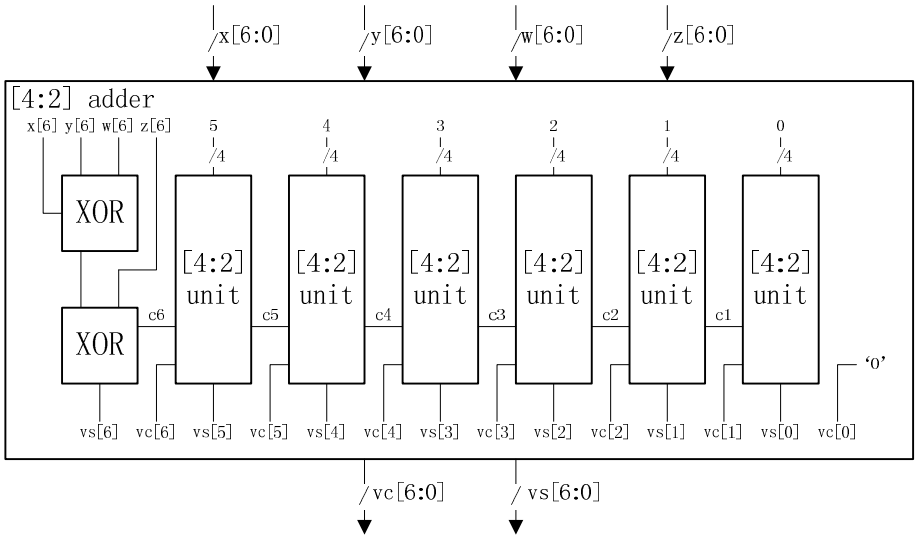


Figure 17

The same with CSA, a '0' is provided as carry input and XOR gates are used to get only sum output. In fact, the thought of using XORs is to replace full adder in the final level with XORs in order to cut down the carry out logic part, as it is shown in Figure 17.

Finally, connect [4:2] adders in tree structure as it is shown below, and finish the transformation of 8 operands: (Figure 18)

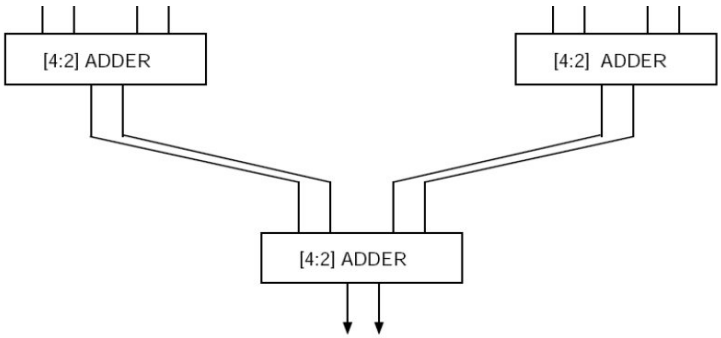


Figure 18

4 Bonus Design

The bonus design is to achieve the implementation of a ten operands 4-bit adder with tree structure. The same with the design above, we have to expend all operands to n+p=4+$\lceil \log_2 10 \rceil$=8 bits. Then a [5:2] adder has to be made due to the tree structure. The structure of a [5:2] unit is shown below: (Figure 19)

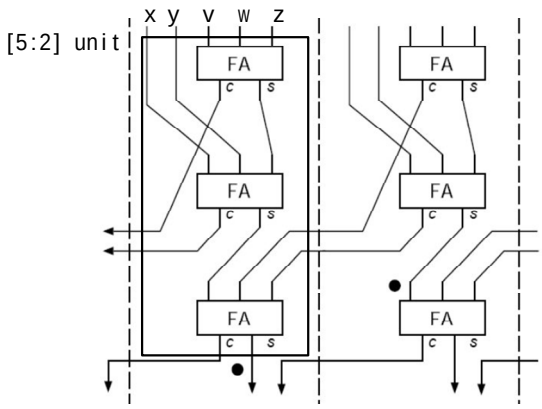


Figure 19

Different from the [4:2] unit, his module has two carry in and two carry out. However, there is still no carry propagation in this design since carry out signals are not generated by carry in signals.

This module is described by the Verilog code below: (Figure 20)

```
module adder_5_2_unit(x,y,v,w,z,ci0,ci1,vs,vc,co0,co1);
    input x,y,v,w,z,ci0,ci1;
    output vs,vc,co0,co1;

    wire s1,s2;

    fulladder FA1(v,w,z,s1,co0);
    fulladder FA2(x,y,s1,s2,co1);
    fulladder FA3(s2,ci0,ci1,vs,vc);
endmodule
```

Figure 20

Then, the structure of a [5:2] adder is served in Figure 21:

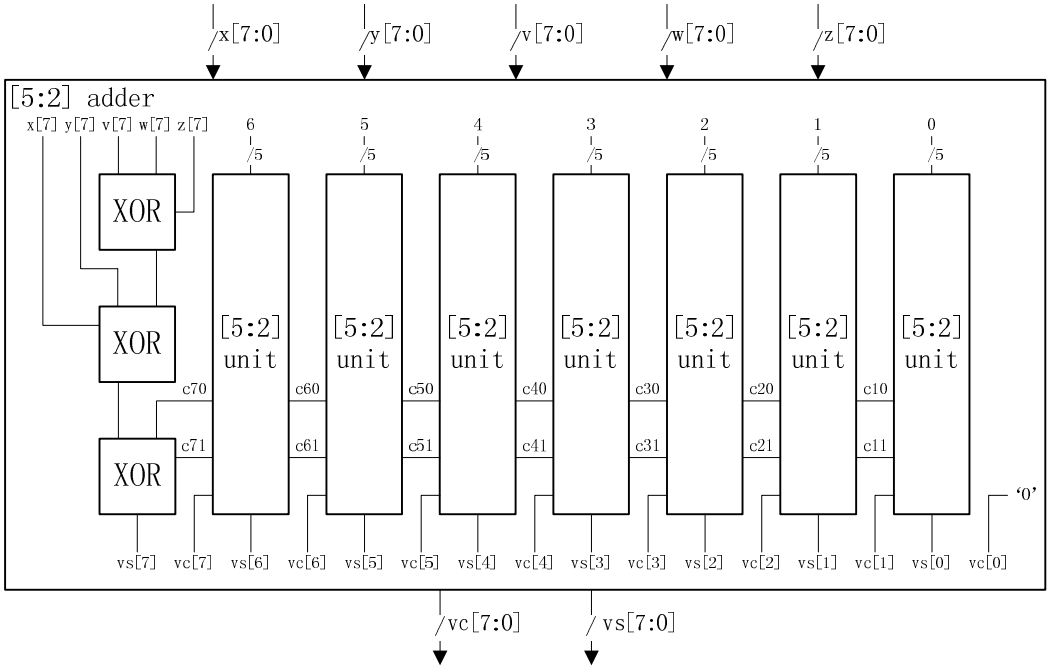


Figure 21

Like all the multi-operand adders above, '0' is served to the carry input and change [5:2] unit with XORs for the final level to generate no carry out. Differently, I use one more [5:2] unit to handle more bit of the operands. Moreover, three XOR gates are used in the last level to take over the add operation of a [5:2] unit, because there are three full adders in one unit.

Finally, finish the design by connect [5:2] adders and a [4:2] adder like below (Figure 22). One thing I have to point out is this [4:2] adder used in this design is slightly different with the design above. It has one more [4:2] unit since the word length of all operands is one bit longer than that in eight operand design.

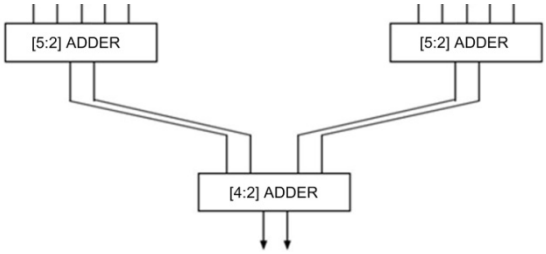


Figure 22

Verilog source code of all of the designs can be found in Appendix Ⅰ.

5 Functional Validation and Verification

5.1 RTL simulation

RTL simulation is the process to ensure that all our designs have the correct functions we want, which is essential before we start logic synthesis. The discussion about of synthesis will be provided in part 6. This part of the paper just covers verification aspect. In order to do RTL simulation, we can add a series of stimulating signals, which is known as a test bench, to the top level of our design. The code of all the test benches can be found in Appendix Ⅰ.

For all the designs of 8-operand 4-bit adder, I use the stimulate signal below:

#100 a1 = 15;a2 = 15;a3 = 15;a4 = 15;a5 = 15;a6 = 15;a7 = 15;a8 = 15;

#200 a1 = 5;a2 = 5;a3 = 5;a4 = 5;a5 = 5;a6 = 5;a7 = 5;a8 = 5;

#300 a1 = 10;a2 = 10;a3 = 10;a4 = 10;a5 = 10;a6 = 10;a7 = 10;a8 = 10;

#400 a1 = 15;a2 = 14;a3 = 13;a4 = 12;a5 = 11;a6 = 10;a7 = 9;a8 = 8;

#500 a1 = 7;a2 = 6;a3 = 5;a4 = 5;a5 = 4;a6 = 3;a7 = 2;a8 = 0;

#600 a1 = 11;a2 = 11;a3 = 11;a4 = 11;a5 = 11;a6 = 11;a7 = 11;a8 = 11;

The simulation result of carry propagation structure is: (Figure 23)

```
Compiling source file "adder_8_4bit_test.v"
Compiling source file "adder_8_4bit_CPA.v"
Highest level modules:
adder_8_4bit_test

At Time:                100 a1=15 a2=15 a3=15 a4=15 a5=15 a6=15 a7=15 a8=15 Sum
=120
At Time:                200 a1= 5 a2= 5 a3= 5 a4= 5 a5= 5 a6= 5 a7= 5 a8= 5 Sum
= 40
At Time:                300 a1=10 a2=10 a3=10 a4=10 a5=10 a6=10 a7=10 a8=10 Sum
= 80
At Time:                400 a1=15 a2=14 a3=13 a4=12 a5=11 a6=10 a7= 9 a8= 8 Sum
= 92
At Time:                500 a1= 7 a2= 6 a3= 5 a4= 4 a5= 3 a6= 2 a7= 1 a8= 0 Sum
= 28
At Time:                600 a1=11 a2=11 a3=11 a4=11 a5=11 a6=11 a7=11 a8=11 Sum
= 88
```

Figure 23

At the same time we can get a wave form with all the signals inside: (Figure 24)

Figure 24

Both of these indicate that the adder design based on carry propagation adder linear structure performs the correct function we need. Actually, all the 8-operand adder designs in this project are proved to be correct after RTL simulation. All simulation reports and wave forms and be found in Appendix Ⅳ.

In addition, the stimulation signals for a 10-operand 4-bit adder are shown below:

#100 a1 = 15;a2 = 15;a3 = 15;a4 = 15;a5 = 15;a6 = 15;a7 = 15;a8 = 15;a9 = 15;a10= 15;

#200 a1 = 5;a2 = 5;a3 = 5;a4 = 5;a5 = 5;a6 = 5;a7 = 5;a8 = 5;a9 = 5;a10= 5;

#300 a1 = 10;a2 = 10;a3 = 10;a4 = 10;a5 = 10;a6 = 10;a7 = 10;a8 = 10;a9 = 10;a10= 10;

#400 a1 = 1;a2 = 2;a3 = 3;a4 = 4;a5 = 5;a6 = 6;a7 = 7;a8 = 8;a9 = 9;a10= 10;

#500 a1 = 15;a2 = 14;a3 = 13;a4 = 12;a5 = 11;a6 = 10;a7 = 9;a8 = 8;a9 = 7;a10= 6;

#600 a1 = 11;a2 = 11;a3 = 11;a4 = 11;a5 = 11;a6 = 11;a7 = 11;a8 = 11;a9 = 11;a10= 11;

The result is shown below: (Figure 25)

```
Compiling source file "adder_10_4bit_test.v"
Compiling source file "adder_10_4bit_tree.v"
Highest level modules:
adder_10_4bit_test

At Time:                 100 a1=15 a2=15 a3=15 a4=15 a5=15 a6=15 a7=15 a8=15 a9=
15 a10=15 Sum=150
At Time:                 200 a1= 5 a2= 5 a3= 5 a4= 5 a5= 5 a6= 5 a7= 5 a8= 5 a9=
 5 a10= 5 Sum= 50
At Time:                 300 a1=10 a2=10 a3=10 a4=10 a5=10 a6=10 a7=10 a8=10 a9=
10 a10=10 Sum=100
At Time:                 400 a1= 1 a2= 2 a3= 3 a4= 4 a5= 5 a6= 6 a7= 7 a8= 8 a9=
 9 a10=10 Sum= 55
At Time:                 500 a1=15 a2=14 a3=13 a4=12 a5=11 a6=10 a7= 9 a8= 8 a9=
 7 a10= 6 Sum=105
At Time:                 600 a1=11 a2=11 a3=11 a4=11 a5=11 a6=11 a7=11 a8=11 a9=
11 a10=11 Sum=110
```

Figure 25

It shows that the design of the 10-operand adder is correct since all the inputs generate accurate results. All the screenshots about this can be found in Appendix Ⅳ.

5.2 Equivalence checking

Though all our designs work properly at RTL level, we cannot guarantee the correctness of them after synthesis and place and route since the features of standard cells and interconnections are included in the final design. The solution is comparing it with the origin RTL level design. This process is called equivalence checking.

For 8-operand 4-bit adder, the screenshot of equivalence checking result is shown below: (Figure 26)

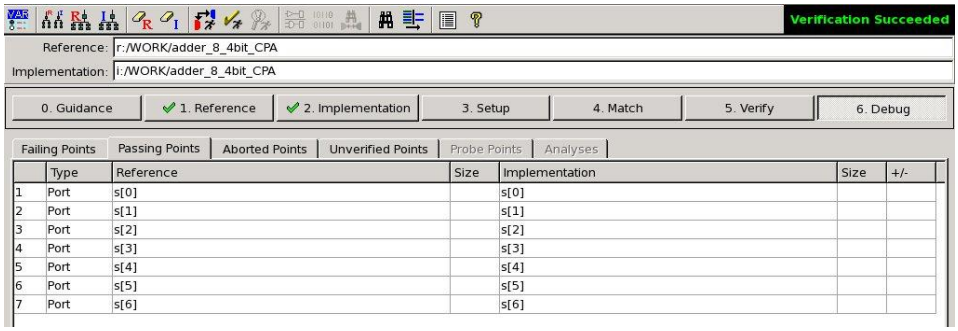


Figure 26

It indicates that the logic functions of the final design and the RTL design are totally the same. The equivalence checking results of other 8-operand adder structures can be found in Appendix Ⅳ

In terms of bonus work, the equivalence checking result is shown below (Figure 27), which is the evidence of correctness for the final design.

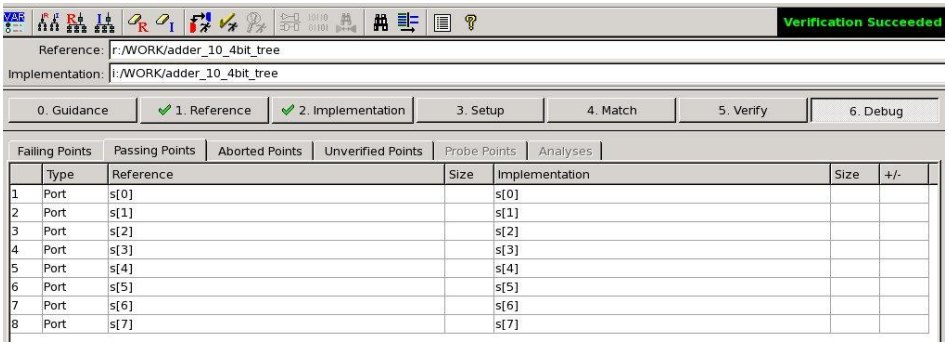


Figure 27

6 Synthesis Result

Depending the standard cell, we can make our design finished automatically by adjusting a design flow in which some design tools will be used. Firstly, we use hardware describe language to build the function in RTL level, which can only describe the logic functions without any implementing details. Then, the RTL design can be changed to gate level directly. This process is called Logic Synthesis. Because gate delays are considered in this level of design, some timing features begin to appear. Next step is called place and route in which standard cell units will be connected and the design will be finished at layout level. After finishing this step, some features of interconnections are also included in the final design, which make it more closer to an actual circuit. As a result, we can analyze trade-offs of different structures based on timing, power and area report for each level of design.

To generate the proper report, we have to built correct synthesis shell and place and route shell code firstly. In these code, we set the clock frequency to 250MHz and output the result of power, area and timing to files as reports. These codes are provided in Appendix Ⅱ.

After that, full reports of timing, power and area are served in Appendix Ⅲ. In this part, I summarize all the results in the table below: (Table 1)

		Synthesis			Place and route		
		Timing slack(ns)	Power (mW)	Area (um^2)	Timing slack(ns)	Power (mW)	Area (um^2)
8-operand 4-bit adder	CPA linear model	2.45	0.1307	490.4	2.197	0.3902	498.4
	CSA linear model	2.37	0.1297	471.2	2.106	0.3515	482.0
	[4:2] tree structure	2.81	0.1055	438.8	2.614	0.1946	450.5
10-operand 4-bit adder	[5:2] tree structure	2.70	0.1410	574.9	2.454	0.2673	589.9

Table 1

It can be seen from this table that timing slack gets smaller, power and area consumption get larger after place and route for all of the adders due the cost of interconnections. Moreover, CSA linear structure have less power and area cost than CPA structure, but the timing slack is worse. Most significantly, the [4:2] tree structure has less power and area consumption with better timing slack at the same time, which make it the best structure among all of these tree structures.

7 Conclusion and future work

In this project, I finish the design flow of the implementation of a 8-operand 4-bit adder in three different structures and a 10-operand 4-bit adder in tree structure. Then, I find that the tree structure is a more efficient design in these three structures after analyzing the report of power, area and timing both in synthesis and place and route level.

However, there are still some methods left to accelerate a multi-operand adder. For example, we can use carry look-ahead adders instead of CPA to achieve the addition of two vectors. Moreover, we can use the method of changing by column with counters to reduce the number of operand step by step rather than transfer all of them to two vectors.

References

[1] M.D. Ercegovac and T. Lang, Digital Arithmetic. Morgan Kaufmann, 2004.

Appendix Ⅰ Source code

8-operand 4-bit adder:

Test bench instance

```
`timescale 1ns/10ps

module adder_8_4bit_test;

reg [3:0] a1,a2,a3,a4,a5,a6,a7,a8;
wire [6:0] s;

adder_8_4bit_CPA adder(a1,a2,a3,a4,a5,a6,a7,a8,s);

initial begin
	$shm_open("shm.db",1); // Opens a waveform database
	$shm_probe("AS");	// Saves all signals to database
	#1000 $finish;
	$shm_close();	// Closes the waveform database
end

initial begin
	a1 = 15;
	a2 = 15;
	a3 = 15;
	a4 = 15;
	a5 = 15;
	a6 = 15;
	a7 = 15;
	a8 = 15;
	#100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d
Sum=%d",
		$time,a1,a2,a3,a4,a5,a6,a7,a8,s);

	a1 = 5;
	a2 = 5;
	a3 = 5;
	a4 = 5;
	a5 = 5;
	a6 = 5;
	a7 = 5;
	a8 = 5;
	#100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d
```

```
Sum=%d",
                $time,a1,a2,a3,a4,a5,a6,a7,a8,s);

              a1 = 10;
        a2 = 10;
        a3 = 10;
        a4 = 10;
        a5 = 10;
        a6 = 10;
        a7 = 10;
        a8 = 10;
        #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d
Sum=%d",
                $time,a1,a2,a3,a4,a5,a6,a7,a8,s);
              a1 = 15;
        a2 = 14;
        a3 = 13;
        a4 = 12;
        a5 = 11;
        a6 = 10;
        a7 = 9;
        a8 = 8;
        #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d
Sum=%d",
                $time,a1,a2,a3,a4,a5,a6,a7,a8,s);
              a1 = 7;
        a2 = 6;
        a3 = 5;
        a4 = 4;
        a5 = 3;
        a6 = 2;
        a7 = 1;
        a8 = 0;
        #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d
Sum=%d",
                $time,a1,a2,a3,a4,a5,a6,a7,a8,s);
              a1 = 11;
        a2 = 11;
        a3 = 11;
        a4 = 11;
        a5 = 11;
        a6 = 11;
        a7 = 11;
        a8 = 11;
```

```
        #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d
Sum=%d",
                $time,a1,a2,a3,a4,a5,a6,a7,a8,s);
end

endmodule

CPA linear model
module halfadder(x,y,s,co);
        input x,y;
        output s,co;

        xor(s,x,y);
        and(co,x,y);
endmodule

module fulladder(x,y,ci,s,co);
        input x,y,ci;
        output s,co;

        wire p,g,o;

        halfadder HA1(x,y,p,g);
        halfadder HA2(p,ci,s,o);
        or(co,o,g);

endmodule

module adder7(a,b,s);

    output [6:0] s;
    input [6:0]         a, b;

    wire   c1, c2, c3, c4, c5, c6;

    halfadder a0(a[0],b[0],s[0], c1);
    fulladder a1(a[1],b[1],c1,s[1],c2);
    fulladder a2(a[2],b[2],c2,s[2],c3);
    fulladder a3(a[3],b[3],c3,s[3],c4);
    fulladder a4(a[4],b[4],c4,s[4],c5);
    fulladder a5(a[5],b[5],c5,s[5],c6);
    xor(s[6],a[6],b[6],c6);

endmodule
```

```
module extension(in,out);
	input [3:0] in;
	output [6:0] out;

	assign out={1'b0,1'b0,1'b0,in};

endmodule

module adder_8_4bit_CPA(a1,a2,a3,a4,a5,a6,a7,a8,s);
	input [3:0] a1,a2,a3,a4,a5,a6,a7,a8;
	output [6:0] s;

	wire [6:0] o1,o2,o3,o4,o5,o6,o7,o8;
	wire [6:0] s1,s2,s3,s4,s5,s6,s;

	extension ext1(a1,o1);
	extension ext2(a2,o2);
	extension ext3(a3,o3);
	extension ext4(a4,o4);
	extension ext5(a5,o5);
	extension ext6(a6,o6);
	extension ext7(a7,o7);
	extension ext8(a8,o8);

	adder7 CPA1(o1,o2,s1);
	adder7 CPA2(o3,s1,s2);
	adder7 CPA3(o4,s2,s3);
	adder7 CPA4(o5,s3,s4);
	adder7 CPA5(o6,s4,s5);
	adder7 CPA6(o7,s5,s6);
	adder7 CPA7(o8,s6,s);
endmodule

CSA linear model
module halfadder(x,y,s,co);
	input x,y;
	output s,co;

	xor(s,x,y);
	and(co,x,y);
endmodule

module fulladder(x,y,ci,s,co);
```

```
        input x,y,ci;
        output s,co;

        wire p,g,o;

        halfadder HA1(x,y,p,g);
        halfadder HA2(p,ci,s,o);
        or(co,o,g);
endmodule

module adder_3_2(x,y,z,vs,vc);
        input [6:0] x,y,z;
        output [6:0] vs,vc;

        assign vc[0]=1'b0;
        fulladder FA1(x[0],y[0],z[0],vs[0],vc[1]);
        fulladder FA2(x[1],y[1],z[1],vs[1],vc[2]);
        fulladder FA3(x[2],y[2],z[2],vs[2],vc[3]);
        fulladder FA4(x[3],y[3],z[3],vs[3],vc[4]);
        fulladder FA5(x[4],y[4],z[4],vs[4],vc[5]);
        fulladder FA6(x[5],y[5],z[5],vs[5],vc[6]);
        xor(vs[6],x[6],y[6],z[6]);

endmodule

module extension(in,out);
        input [3:0] in;
        output [6:0] out;

        assign out={1'b0,1'b0,1'b0,in};

endmodule

module adder7(a,b,s);

      output [6:0] s;
      input [6:0]        a, b;

      wire   c1, c2, c3, c4, c5, c6;

      halfadder a0(a[0],b[0],s[0], c1);
      fulladder a1(a[1],b[1],c1,s[1],c2);
      fulladder a2(a[2],b[2],c2,s[2],c3);
      fulladder a3(a[3],b[3],c3,s[3],c4);
```

```
    fulladder a4(a[4],b[4],c4,s[4],c5);
    fulladder a5(a[5],b[5],c5,s[5],c6);
    xor(s[6],a[6],b[6],c6);

endmodule

module adder_8_4bit_linear(a1,a2,a3,a4,a5,a6,a7,a8,s);
      input [3:0] a1,a2,a3,a4,a5,a6,a7,a8;
      output [6:0] s;
      wire [6:0] o1,o2,o3,o4,o5,o6,o7,o8;
      wire [6:0] vs1,vs2,vs3,vs4,vs5,vs;
      wire [6:0] vc1,vc2,vc3,vc4,vc5,vc;

      extension ext1(a1,o1);
      extension ext2(a2,o2);
      extension ext3(a3,o3);
      extension ext4(a4,o4);
      extension ext5(a5,o5);
      extension ext6(a6,o6);
      extension ext7(a7,o7);
      extension ext8(a8,o8);

      adder_3_2 CSA1(o1,o2,o3,vs1,vc1);
      adder_3_2 CSA2(o4,vs1,vc1,vs2,vc2);
      adder_3_2 CSA3(o5,vs2,vc2,vs3,vc3);
      adder_3_2 CSA4(o6,vs3,vc3,vs4,vc4);
      adder_3_2 CSA5(o7,vs4,vc4,vs5,vc5);
      adder_3_2 CSA6(o8,vs5,vc5,vs,vc);

      adder7 CPA(vs,vc,s);
endmodule

[4:2] tree structure
module halfadder(x,y,s,co);
      input x,y;
      output s,co;

      xor(s,x,y);
      and(co,x,y);
endmodule

module fulladder(x,y,ci,s,co);
      input x,y,ci;
      output s,co;
```

```
        wire p,g,o;

        halfadder HA1(x,y,p,g);
        halfadder HA2(p,ci,s,o);
        or(co,o,g);

endmodule

module adder_4_2_unit(x,y,w,z,ci,vs,vc,co);
        input x,y,w,z,ci;
        output vs,vc,co;

        wire s1;

        fulladder FA1(x,y,w,s1,co);
        fulladder FA2(s1,z,ci,vs,vc);
endmodule

module adder_4_2(x,y,w,z,vs,vc);
        input [6:0] x,y,w,z;
        output [6:0] vs,vc;

        wire c1,c2,c3,c4,c5,c6,s1;

        assign vc[0]=1'b0;
        adder_4_2_unit unit1(x[0],y[0],w[0],z[0],1'b0,vs[0],vc[1],c1);
        adder_4_2_unit unit2(x[1],y[1],w[1],z[1],c1,vs[1],vc[2],c2);
        adder_4_2_unit unit3(x[2],y[2],w[2],z[2],c2,vs[2],vc[3],c3);
        adder_4_2_unit unit4(x[3],y[3],w[3],z[3],c3,vs[3],vc[4],c4);
        adder_4_2_unit unit5(x[4],y[4],w[4],z[4],c4,vs[4],vc[5],c5);
        adder_4_2_unit unit6(x[5],y[5],w[5],z[5],c5,vs[5],vc[6],c6);
        xor(s1,x[6],y[6],w[6]);
        xor(vs[6],s1,z[6],c6);

endmodule

module extension(in,out);
        input [3:0] in;
        output [6:0] out;

        assign out={1'b0,1'b0,1'b0,in};

endmodule
```

```
module adder7(a,b,s);

    output [6:0] s;
    input [6:0]        a, b;

    wire   c1, c2, c3, c4, c5, c6;

    halfadder a0(a[0],b[0],s[0], c1);
    fulladder a1(a[1],b[1],c1,s[1],c2);
    fulladder a2(a[2],b[2],c2,s[2],c3);
    fulladder a3(a[3],b[3],c3,s[3],c4);
    fulladder a4(a[4],b[4],c4,s[4],c5);
    fulladder a5(a[5],b[5],c5,s[5],c6);
    xor(s[6],a[6],b[6],c6);

endmodule

module adder_8_4bit_tree(a1,a2,a3,a4,a5,a6,a7,a8,s);
      input [3:0] a1,a2,a3,a4,a5,a6,a7,a8;
      output [6:0] s;
      wire [6:0] o1,o2,o3,o4,o5,o6,o7,o8;
      wire [6:0] vs1,vs2,vs;
      wire [6:0] vc1,vc2,vc;

      extension ext1(a1,o1);
      extension ext2(a2,o2);
      extension ext3(a3,o3);
      extension ext4(a4,o4);
      extension ext5(a5,o5);
      extension ext6(a6,o6);
      extension ext7(a7,o7);
      extension ext8(a8,o8);

      adder_4_2 level1_1(o1,o2,o3,o4,vs1,vc1);
      adder_4_2 level1_2(o5,o6,o7,o8,vs2,vc2);
      adder_4_2 level2_1(vs1,vc1,vs2,vc2,vs,vc);

      adder7 CPA(vs,vc,s);
endmodule
```

10-operand4-bit adder:

Test bench instance

```
`timescale 1ns/10ps

module adder_10_4bit_test;

reg [3:0] a1,a2,a3,a4,a5,a6,a7,a8,a9,a10;
wire [7:0] s;

adder_10_4bit_tree adder(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);

initial begin
	$shm_open("shm.db",1); // Opens a waveform database
	$shm_probe("AS");	// Saves all signals to database
	#1000 $finish;
	$shm_close();	// Closes the waveform database
end

initial begin
	a1 = 15;
	a2 = 15;
	a3 = 15;
	a4 = 15;
	a5 = 15;
	a6 = 15;
	a7 = 15;
	a8 = 15;
	a9 = 15;
	a10= 15;
	#100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d a9=%d
a10=%d Sum=%d",
		$time,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);

	a1 = 5;
	a2 = 5;
	a3 = 5;
	a4 = 5;
	a5 = 5;
	a6 = 5;
	a7 = 5;
	a8 = 5;
	a9 = 5;
	a10= 5;
	#100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d a9=%d
a10=%d Sum=%d",
		$time,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);
```

```
    a1 = 10;
     a2 = 10;
     a3 = 10;
     a4 = 10;
     a5 = 10;
     a6 = 10;
     a7 = 10;
     a8 = 10;
     a9 = 10;
     a10= 10;
     #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d a9=%d
a10=%d Sum=%d",
          $time,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);

     a1 = 1;
     a2 = 2;
     a3 = 3;
     a4 = 4;
     a5 = 5;
     a6 = 6;
     a7 = 7;
     a8 = 8;
     a9 = 9;
     a10= 10;
     #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d a9=%d
a10=%d Sum=%d",
          $time,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);

     a1 = 15;
     a2 = 14;
     a3 = 13;
     a4 = 12;
     a5 = 11;
     a6 = 10;
     a7 = 9;
     a8 = 8;
     a9 = 7;
     a10= 6;
     #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d a9=%d
a10=%d Sum=%d",
          $time,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);

     a1 = 11;
```

```
        a2 = 11;
        a3 = 11;
        a4 = 11;
        a5 = 11;
        a6 = 11;
        a7 = 11;
        a8 = 11;
        a9 = 11;
        a10= 11;
        #100 $display("At Time:%d a1=%d a2=%d a3=%d a4=%d a5=%d a6=%d a7=%d a8=%d a9=%d
a10=%d Sum=%d",
                $time,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);
end

endmodule

[5:2] tree structure
module halfadder(x,y,s,co);
        input x,y;
        output s,co;

        xor(s,x,y);
        and(co,x,y);
endmodule

module fulladder(x,y,ci,s,co);
        input x,y,ci;
        output s,co;

        wire p,g,o;

        halfadder HA1(x,y,p,g);
        halfadder HA2(p,ci,s,o);
        or(co,o,g);

endmodule

module adder_4_2_unit(x,y,w,z,ci,vs,vc,co);
        input x,y,w,z,ci;
        output vs,vc,co;

        wire s1;

        fulladder FA1(x,y,w,s1,co);
```

```
        fulladder FA2(s1,z,ci,vs,vc);
endmodule

module adder_5_2_unit(x,y,v,w,z,ci0,ci1,vs,vc,co0,co1);
        input x,y,v,w,z,ci0,ci1;
        output vs,vc,co0,co1;

        wire s1,s2;

        fulladder FA1(v,w,z,s1,co0);
        fulladder FA2(x,y,s1,s2,co1);
        fulladder FA3(s2,ci0,ci1,vs,vc);
endmodule

module adder8(a,b,s);

    output [7:0] s;
    input [7:0]         a, b;

    wire   c1, c2, c3, c4, c5, c6;

    halfadder a0(a[0],b[0],s[0], c1);
    fulladder a1(a[1],b[1],c1,s[1],c2);
    fulladder a2(a[2],b[2],c2,s[2],c3);
    fulladder a3(a[3],b[3],c3,s[3],c4);
    fulladder a4(a[4],b[4],c4,s[4],c5);
    fulladder a5(a[5],b[5],c5,s[5],c6);
    fulladder a6(a[6],b[6],c6,s[6],c7);
    xor(s[7],a[7],b[7],c7);

endmodule

module adder_4_2_8bit(x,y,w,z,vs,vc);
        input [7:0] x,y,w,z;
        output [7:0] vs,vc;

        wire c1,c2,c3,c4,c5,c6,c7,s1;

        assign vc[0]=1'b0;
        adder_4_2_unit unit1(x[0],y[0],w[0],z[0],1'b0,vs[0],vc[1],c1);
        adder_4_2_unit unit2(x[1],y[1],w[1],z[1],c1,vs[1],vc[2],c2);
        adder_4_2_unit unit3(x[2],y[2],w[2],z[2],c2,vs[2],vc[3],c3);
        adder_4_2_unit unit4(x[3],y[3],w[3],z[3],c3,vs[3],vc[4],c4);
        adder_4_2_unit unit5(x[4],y[4],w[4],z[4],c4,vs[4],vc[5],c5);
```

```
        adder_4_2_unit unit6(x[5],y[5],w[5],z[5],c5,vs[5],vc[6],c6);
        adder_4_2_unit unit7(x[6],y[6],w[6],z[6],c6,vs[6],vc[7],c7);
        xor(s1,x[7],y[7],w[7]);
        xor(vs[7],s1,z[7],c7);

endmodule

module adder_5_2(x,y,v,w,z,vs,vc);
        input [7:0] x,y,v,w,z;
        output [7:0] vs,vc;

        wire c10,c11,c20,c21,c30,c31,c40,c41,c50,c51,c60,c61,c70,c71,s1,s2;

        assign vc[0]=1'b0;
        adder_5_2_unit unit1(x[0],y[0],v[0],w[0],z[0],1'b0,1'b0,vs[0],vc[1],c10,c11);
        adder_5_2_unit unit2(x[1],y[1],v[1],w[1],z[1],c10,c11,vs[1],vc[2],c20,c21);
        adder_5_2_unit unit3(x[2],y[2],v[2],w[2],z[2],c20,c21,vs[2],vc[3],c30,c31);
        adder_5_2_unit unit4(x[3],y[3],v[3],w[3],z[3],c30,c31,vs[3],vc[4],c40,c41);
        adder_5_2_unit unit5(x[4],y[4],v[4],w[4],z[4],c40,c41,vs[4],vc[5],c50,c51);
        adder_5_2_unit unit6(x[5],y[5],v[5],w[5],z[5],c50,c51,vs[5],vc[6],c60,c61);
        adder_5_2_unit unit7(x[6],y[6],v[6],w[6],z[6],c60,c61,vs[6],vc[7],c70,c71);
        xor(s1,v[7],w[7],z[7]);
        xor(s2,x[7],y[7],s1);
        xor(vs[7],s2,c70,c71);

endmodule

module extension_4_8(in,out);
        input [3:0] in;
        output [7:0] out;

        assign out={1'b0,1'b0,1'b0,1'b0,in};

endmodule

module adder_10_4bit_tree(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,s);
        input [3:0] a1,a2,a3,a4,a5,a6,a7,a8,a9,a10;
        output [7:0] s;
        wire [7:0] o1,o2,o3,o4,o5,o6,o7,o8,o9,o10;
        wire [7:0] vs1,vs2,vs;
        wire [7:0] vc1,vc2,vc;

        extension_4_8 ext1(a1,o1);
        extension_4_8 ext2(a2,o2);
```

```
        extension_4_8 ext3(a3,o3);
        extension_4_8 ext4(a4,o4);
        extension_4_8 ext5(a5,o5);
        extension_4_8 ext6(a6,o6);
        extension_4_8 ext7(a7,o7);
        extension_4_8 ext8(a8,o8);
        extension_4_8 ext9(a9,o9);
        extension_4_8 ext10(a10,o10);

        adder_5_2 level1_1(o1,o2,o3,o4,o5,vs1,vc1);
        adder_5_2 level1_2(o6,o7,o8,o9,o10,vs2,vc2);
        adder_4_2_8bit level2_1(vs1,vc1,vs2,vc2,vs,vc);

        adder8 CPA(vs,vc,s);
endmodule
```

Appendix Ⅱ Shell code

Synthesis shell code instance:

compile_dc.tcl

```
#/*****************************************************/
#/* Compile Script for Synopsys                       */
#/*                                                   */
#/* dc_shell-t -f compile_dc.tcl                      */
#/*                                                   */
#/* OSU FreePDK 45nm                                  */
#/*****************************************************/

#/* All verilog files, separated by spaces            */
set my_verilog_files [list adder_8_4bit_CPA.v]

#/* Top-level Module                                  */
set my_toplevel adder_8_4bit_CPA

#/* The name of the clock pin. If no clock-pin        */
#/* exists, pick anything                             */
set my_clock_pin clk

#/* Target frequency in MHz for optimization          */
set my_clk_freq_MHz 250

#/* Delay of input signals (Clock-to-Q, Package etc.)  */
set my_input_delay_ns 0.1

#/* Reserved time for output signals (Holdtime etc.)  */
set my_output_delay_ns 0.1


#/*****************************************************/
#/* No modifications needed below                     */
#/*****************************************************/
set OSU_FREEPDK [format "%s%s"    [getenv "PDK_DIR"] "/osu_soc/lib/files"]
set search_path [concat    $search_path $OSU_FREEPDK]
set alib_library_analysis_path $OSU_FREEPDK

set link_library [set target_library [concat    [list gscl45nm.db] [list dw_foundation.sldb]]]
set target_library "gscl45nm.db"
define_design_lib WORK -path ./WORK
```

```
set verilogout_show_unconnected_pins "true"
#set_ultra_optimization true
#set_ultra_optimization -force

analyze -f verilog $my_verilog_files

elaborate $my_toplevel

current_design $my_toplevel

link
uniquify

set my_period [expr 1000 / $my_clk_freq_MHz]

set find_clock [ find port [list $my_clock_pin] ]
if {    $find_clock != [list] } {
     set clk_name $my_clock_pin
     create_clock -period $my_period $clk_name
} else {
     set clk_name vclk
     create_clock -period $my_period -name $clk_name
}

set_driving_cell    -lib_cell INVX1    [all_inputs]
set_input_delay $my_input_delay_ns -clock $clk_name [remove_from_collection [all_inputs]
$my_clock_pin]
set_output_delay $my_output_delay_ns -clock $clk_name [all_outputs]

compile -ungroup_all -map_effort medium

compile -incremental_mapping -map_effort medium

check_design
#report_constraint -all_violators

set filename [format "%s%s"    $my_toplevel ".vh"]
write -f verilog -output $filename

set filename [format "%s%s"    $my_toplevel ".sdc"]
write_sdc $filename

#set filename [format "%s%s"    $my_toplevel ".db"]
#write -f db -hier -output $filename -xg_force_db
```

```
redirect timing.rep { report_timing }
redirect cell.rep { report_cell }
redirect power.rep { report_power }

quit
```

Place and route shell code:

encounter.conf

```
################################################
#                                              #
#    FirstEncounter Input configuration file   #
#                                              #
################################################
#
#

# Specify the name of your toplevel module
set my_toplevel    adder_8_4bit_CPA

################################################
# No changes required below
################################################

global env
set OSU_FREEPDK $env(PDK_DIR)/osu_soc

global rda_Input
set rda_Input(ui_netlist) $my_toplevel.vh
set rda_Input(ui_timingcon_file) $my_toplevel.sdc
set rda_Input(ui_topcell) $my_toplevel

set rda_Input(ui_netlisttype) {Verilog}
set rda_Input(ui_ilmlist) {}
set rda_Input(ui_settop) {1}
set rda_Input(ui_celllib) {}
set rda_Input(ui_iolib) {}
set rda_Input(ui_areaiolib) {}
set rda_Input(ui_blklib) {}
set rda_Input(ui_kboxlib) ""
set rda_Input(ui_timelib) "$OSU_FREEPDK/lib/files/gscl45nm.tlf"
set rda_Input(ui_smodDef) {}
set rda_Input(ui_smodData) {}
set rda_Input(ui_dpath) {}
```

```
set rda_Input(ui_tech_file) {}
set rda_Input(ui_io_file) ""
set rda_Input(ui_buf_footprint) {buf}
set rda_Input(ui_delay_footprint) {buf}
set rda_Input(ui_inv_footprint) {inv}
set rda_Input(ui_leffile) "$OSU_FREEPDK/lib/files/gscl45nm.lef"
set rda_Input(ui_core_cntl) {aspect}
set rda_Input(ui_aspect_ratio) {1.0}
set rda_Input(ui_core_util) {0.7}
set rda_Input(ui_core_height) {}
set rda_Input(ui_core_width) {}
set rda_Input(ui_core_to_left) {}
set rda_Input(ui_core_to_right) {}
set rda_Input(ui_core_to_top) {}
set rda_Input(ui_core_to_bottom) {}
set rda_Input(ui_max_io_height) {0}
set rda_Input(ui_row_height) {}
set rda_Input(ui_isHorTrackHalfPitch) {0}
set rda_Input(ui_isVerTrackHalfPitch) {1}
set rda_Input(ui_ioOri) {R0}
set rda_Input(ui_isOrigCenter) {0}
set rda_Input(ui_exc_net) {}
set rda_Input(ui_delay_limit) {1000}
set rda_Input(ui_net_delay) {1000.0ps}
set rda_Input(ui_net_load) {0.5pf}
set rda_Input(ui_in_tran_delay) {120.0ps}
set rda_Input(ui_captbl_file) {}
set rda_Input(ui_cap_scale) {1.0}
set rda_Input(ui_xcap_scale) {1.0}
set rda_Input(ui_res_scale) {1.0}
set rda_Input(ui_shr_scale) {1.0}
set rda_Input(ui_time_unit) {none}
set rda_Input(ui_cap_unit) {}
set rda_Input(ui_sigstormlib) {}
set rda_Input(ui_cdb_file) {}
set rda_Input(ui_echo_file) {}
set rda_Input(ui_qxtech_file) {}
set rda_Input(ui_qxlib_file) {}
set rda_Input(ui_qxconf_file) {}
set rda_Input(ui_pwrnet) {vdd}
set rda_Input(ui_gndnet) {gnd}
set rda_Input(flip_first) {1}
set rda_Input(double_back) {1}
set rda_Input(assign_buffer) {0}
```

```
set rda_Input(ui_pg_connections) [list \
                                        {PIN:vdd:} \
                                        {PIN:gnd:} \
                                                ]
set rda_Input(PIN:vdd:) {vdd}
set rda_Input(PIN:gnd:) {gnd}
```

encounter.tcl

```
####################################
# Run the design through Encounter
####################################

# Setup design and create floorplan
loadConfig ./encounter.conf
commitConfig

# Create Floorplan
floorplan -r 1.0 0.6 40.05 40.8 40.05 42

# Add supply rings around core
addRing -spacing_bottom 9.9 -width_left 9.9 -width_bottom 9.9 -width_top 9.9 -spacing_top 9.9
-layer_bottom metal1 -width_right 9.9 -around core -center 1 -layer_top metal1 -spacing_right
9.9 -spacing_left 9.9 -layer_right metal2 -layer_left metal2 -offset_top 9.9 -offset_bottom 9.9
-offset_left 9.9 -offset_right 9.9 -nets { gnd vdd }

# Place standard cells
amoebaPlace

# Route power nets
sroute -noBlockPins -noPadRings

# Perform trial route and get initial timing results
trialroute
buildTimingGraph
setCteReport
report_timing -nworst    10 -net > timing.rep.1.placed

# Run in-place optimization
# to fix setup problems
setOptMode -mediumEffort -fixDRC -addPortAsNeeded
initECO ./ipo1.txt
fixSetupViolation
endECO
buildTimingGraph
```

```
setCteReport
report_timing -nworst    10 -net > timing.rep.2.ipo1

# Run Clock Tree Synthesis
createClockTreeSpec -output encounter.cts -bufFootprint buf -invFootprint inv
specifyClockTree -clkfile encounter.cts
ckSynthesis -rguide cts.rguide -report report.ctsrpt -macromodel report.ctsmdl
-fix_added_buffers

# Output Results of CTS
trialRoute -highEffort -guide cts.rguide
extractRC
reportClockTree -postRoute -localSkew -report skew.post_troute_local.ctsrpt
reportClockTree -postRoute -report report.post_troute.ctsrpt

# Run Post-CTS Timing analysis
setAnalysisMode -setup -async -skew -autoDetectClockTree
buildTimingGraph
setCteReport
report_timing -nworst    10 -net > timing.rep.3.cts

# Perform post-CTS IPO
setOptMode -highEffort -fixDrc -addPortAsNeeded -incrTrialRoute -restruct -topomap
initECO ipo2.txt
setExtractRCMode -default -assumeMetFill
extractRC
fixSetupViolation -guide cts.rguide

# Fix all remaining violations
setExtractRCMode -detail -assumeMetFill
extractRC
if {[isDRVClean -maxTran -maxCap -maxFanout] != 1} {
fixDRCViolation -maxTran -maxCap -maxFanout
}

endECO
cleanupECO

# Run Post IPO-2 timing analysis
buildTimingGraph
setCteReport
report_timing -nworst    10 -net > timing.rep.4.ipo2

# Add filler cells
```

```
addFiller -cell FILL -prefix FILL -fillBoundary

# Connect all new cells to VDD/GND
globalNetConnect vdd -type tiehi
globalNetConnect vdd -type pgpin -pin vdd -override

globalNetConnect gnd -type tielo
globalNetConnect gnd -type pgpin -pin gnd -override

# Run global Routing
globalDetailRoute

# Get final timing results
setExtractRCMode -detail -noReduce
extractRC
buildTimingGraph
setCteReport
report_timing -nworst    10 -net > timing.rep.5.final

# Output GDSII
streamOut final.gds2 -mapFile gds2_encounter.map -stripes 1 -units 1000 -mode ALL
saveNetlist -excludeLeafCell final.v

# Output DSPF RC Data
rcout -spf final.dspf

# Run DRC and Connection checks
verifyGeometry
verifyConnectivity -type all

# Get Gate report
reportGateCount -limit 0 -outfile gate.final

# Get power report
report_power -outfile power.final

puts "****************************************"
puts "* Encounter script finished                 *"
puts "*                                                  *"
puts "* Results:                                       *"
puts "* --------                                      *"
puts "* Layout:    final.gds2                      *"
puts "* Netlist: final.v                           *"
puts "* Timing:    timing.rep.5.final            *"
```

```
puts "* Area:        gate.final                          *"
puts "* Power:       power.final                          *"
puts "*                                                   *"
puts "* Type 'win' to get the Main Window    *"
puts "* or type 'exit' to quit                        *"
puts "*                                                   *"
puts "*************************************"
```

Appendix Ⅲ Full report

8-operand 4-bit adder:

CPA linear model

timing.rep:

```
  clock vclk (rise edge)                    4.00       4.00
  clock network delay (ideal)               0.00       4.00
  output external delay                    -0.10       3.90
  data required time                                   3.90
  ---------------------------------------------------------
  data required time                                   3.90
  data arrival time                                   -1.45
  ---------------------------------------------------------
  slack (MET)                                          2.45
```

power.rep:

```
  Cell Internal Power  =  79.3716 uW   (63%)
  Net Switching Power  =  47.1732 uW   (37%)
                          ---------
Total Dynamic Power    = 126.5448 uW  (100%)

Cell Leakage Power     =   4.1145 uW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000            0.0000  (   0.00%)
register           0.0000           0.0000            0.0000            0.0000  (   0.00%)
sequential         0.0000           0.0000            0.0000            0.0000  (   0.00%)
combinational  7.9372e-02        4.7173e-02        4.1145e+03            0.1307  ( 100.00%)
--------------------------------------------------------------------------------------------------
Total          7.9372e-02 mW     4.7173e-02 mW     4.1145e+03 nW         0.1307 mW
```

cell.rep:

```
U66                            OR2X1           gscl45nm        2.346500
U67                            AND2X1          gscl45nm        2.346500
U68                            AND2X1          gscl45nm        2.346500
U69                            OR2X1           gscl45nm        2.346500
U70                            AND2X1          gscl45nm        2.346500
U71                            AND2X1          gscl45nm        2.346500
U72                            OR2X1           gscl45nm        2.346500
U73                            AND2X1          gscl45nm        2.346500
U74                            AND2X1          gscl45nm        2.346500
U75                            OR2X1           gscl45nm        2.346500
U76                            AND2X1          gscl45nm        2.346500
U77                            AND2X1          gscl45nm        2.346500
U78                            OR2X1           gscl45nm        2.346500
U79                            AND2X1          gscl45nm        2.346500
U80                            AND2X1          gscl45nm        2.346500
U81                            AND2X1          gscl45nm        2.346500
U82                            XOR2X1          gscl45nm        4.693000
U83                            XOR2X1          gscl45nm        4.693000
U84                            XOR2X1          gscl45nm        4.693000
U85                            XOR2X1          gscl45nm        4.693000
------------------------------------------------------------------------------
Total 145 cells                                                490.418483
```

timing.rep.5.final:

```
#####################################################################
#  Generated by:      Cadence Encounter 10.13-s292_1
#  OS:                Linux x86_64(Host ID saturn.ece.iit.edu)
#  Generated on:      Sat Dec  8 06:49:07 2012
#  Design:            adder_8_4bit_CPA
#  Command:           report_timing -nworst  10 -net > timing.rep.5.final
#####################################################################
Path 1: MET Late External Delay Assertion
Endpoint:   s[6]  (^) checked with  leading edge of 'vclk'
Beginpoint: a1[1] (^) triggered by  leading edge of 'vclk'
Other End Arrival Time          0.000
- External Delay                0.100
+ Phase Shift                   4.000
= Required Time                 3.900
- Arrival Time                  1.703
= Slack Time                    2.197
```

power.final:

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:      0.2126      54.49%
Total Switching Power:     0.1734      44.45%
Total Leakage Power:     0.004115      1.055%
Total Power:               0.3902
-----------------------------------------------------------------------------------------
```

gate.final:

```
Gate area 2.8158 um^2
Level 0 Module adder_8_4bit_CPA                        Gates=       177 Cells=     145 Area=     498.4 um^2
```

CSA linear model

timing.rep:

```
  clock vclk (rise edge)                    4.00       4.00
  clock network delay (ideal)               0.00       4.00
  output external delay                    -0.10       3.90
  data required time                                   3.90
  ----------------------------------------------------------
  data required time                                   3.90
  data arrival time                                   -1.53
  ----------------------------------------------------------
  slack (MET)                                          2.37
```

power.rep:

```
  Cell Internal Power  =  79.0421 uW   (63%)
  Net Switching Power  =  46.7893 uW   (37%)
                          ---------
Total Dynamic Power    = 125.8314 uW  (100%)

Cell Leakage Power     =   3.8791 uW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000           0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000           0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000           0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000           0.0000  (   0.00%)
register           0.0000           0.0000            0.0000           0.0000  (   0.00%)
sequential         0.0000           0.0000            0.0000           0.0000  (   0.00%)
combinational  7.9042e-02       4.6789e-02        3.8791e+03           0.1297  ( 100.00%)
--------------------------------------------------------------------------------------------------
Total          7.9042e-02 mW     4.6789e-02 mW     3.8791e+03 nW        0.1297 mW
```

cell.rep:

```
U66                     AND2X1          gscl45nm        2.346500
U67                     OR2X1           gscl45nm        2.346500
U68                     AND2X1          gscl45nm        2.346500
U69                     AND2X1          gscl45nm        2.346500
U70                     OR2X1           gscl45nm        2.346500
U71                     AND2X1          gscl45nm        2.346500
U72                     AND2X1          gscl45nm        2.346500
U73                     OR2X1           gscl45nm        2.346500
U74                     AND2X1          gscl45nm        2.346500
U75                     AND2X1          gscl45nm        2.346500
U76                     OR2X1           gscl45nm        2.346500
U77                     AND2X1          gscl45nm        2.346500
U78                     AND2X1          gscl45nm        2.346500
U79                     OR2X1           gscl45nm        2.346500
U80                     AND2X1          gscl45nm        2.346500
U81                     AND2X1          gscl45nm        2.346500
------------------------------------------------------------------------------
Total 140 cells                                         471.177185
```

timing.rep.5.final:

```
#####################################################################
#  Generated by:      Cadence Encounter 10.13-s292_1
#  OS:                Linux x86_64(Host ID saturn.ece.iit.edu)
#  Generated on:      Sat Dec  8 07:06:40 2012
#  Design:            adder_8_4bit_linear
#  Command:           report_timing -nworst  10 -net > timing.rep.5.final
#####################################################################
Path 1: MET Late External Delay Assertion
Endpoint:   s[6]  (v) checked with  leading edge of 'vclk'
Beginpoint: a1[1] (^) triggered by  leading edge of 'vclk'
Other End Arrival Time          0.000
- External Delay                0.100
+ Phase Shift                   4.000
= Required Time                 3.900
- Arrival Time                  1.794
= Slack Time                    2.106
```

power.final:

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:      0.1925      54.76%
Total Switching Power:     0.1551      44.14%
Total Leakage Power:     0.003879      1.104%
Total Power:               0.3515
-----------------------------------------------------------------------------------------
```

gate.final:

```
Gate area 2.8158 um^2
Level 0 Module adder_8_4bit_linear                  Gates=       171 Cells=     140 Area=     482.0 um^2
```

[4:2] tree structure

timing.rep:

```
  clock vclk (rise edge)                                  4.00       4.00
  clock network delay (ideal)                             0.00       4.00
  output external delay                                  -0.10       3.90
  data required time                                                 3.90
  ------------------------------------------------------------------------
  data required time                                                 3.90
  data arrival time                                                 -1.09
  ------------------------------------------------------------------------
  slack (MET)                                                        2.81
```

power.rep:

```
  Cell Internal Power  =  64.9142 uW   (64%)
  Net Switching Power  =  36.9632 uW   (36%)
                         ---------
Total Dynamic Power    = 101.8774 uW  (100%)

Cell Leakage Power     =   3.6244 uW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.0000            0.0000            0.0000  (   0.00%)
register           0.0000            0.0000            0.0000            0.0000  (   0.00%)
sequential         0.0000            0.0000            0.0000            0.0000  (   0.00%)
combinational  6.4914e-02        3.6963e-02        3.6244e+03            0.1055  ( 100.00%)
--------------------------------------------------------------------------------------------------
Total          6.4914e-02 mW     3.6963e-02 mW     3.6244e+03 nW         0.1055 mW
```

cell.rep:

```
level2_1/unit2/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit2/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit3/FA1/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit3/FA1/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit3/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit3/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA1/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA1/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA1/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA1/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
-------------------------------------------------------------------------------
Total 133 cells                                           438.795485
```

timing.rep.5.final:

```
#####################################################################
#  Generated by:      Cadence Encounter 10.13-s292_1
#  OS:                Linux x86_64(Host ID saturn.ece.iit.edu)
#  Generated on:      Sat Dec  8 07:14:11 2012
#  Design:            adder_8_4bit_tree
#  Command:           report_timing -nworst  10 -net > timing.rep.5.final
#####################################################################
Path 1: MET Late External Delay Assertion
Endpoint:   s[6]  (^) checked with  leading edge of 'vclk'
Beginpoint: a1[2] (^) triggered by  leading edge of 'vclk'
Other End Arrival Time          0.000
- External Delay                0.100
+ Phase Shift                   4.000
= Required Time                 3.900
- Arrival Time                  1.286
= Slack Time                    2.614
```

power.final:

```
Total Power
-------------------------------------------------------------------------------------------------
Total Internal Power:      0.1078      55.41%
Total Switching Power:    0.08312      42.72%
Total Leakage Power:     0.003624      1.863%
Total Power:               0.1946
-------------------------------------------------------------------------------------------------
```

gate.final:

```
Gate area 2.8158 um^2
Level 0 Module adder_8_4bit_tree                      Gates=       160 Cells=     133 Area=     450.5 um^2
```

10-operand4-bit adder:

[5:2] tree structure

timing.rep:

```
  clock vclk (rise edge)                                   4.00       4.00
  clock network delay (ideal)                              0.00       4.00
  output external delay                                   -0.10       3.90
  data required time                                                  3.90
  --------------------------------------------------------------------------
  data required time                                                  3.90
  data arrival time                                                  -1.20
  --------------------------------------------------------------------------
  slack (MET)                                                         2.70
```

power.rep:

```
  Cell Internal Power  =  85.9996 uW   (63%)
  Net Switching Power  =  50.2964 uW   (37%)
                          ---------
Total Dynamic Power    = 136.2960 uW  (100%)

Cell Leakage Power     =   4.7534 uW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000           0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000           0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000           0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000           0.0000  (   0.00%)
register           0.0000           0.0000            0.0000           0.0000  (   0.00%)
sequential         0.0000           0.0000            0.0000           0.0000  (   0.00%)
combinational  8.6000e-02        5.0296e-02        4.7534e+03           0.1410  ( 100.00%)
--------------------------------------------------------------------------------------------------
Total          8.6000e-02 mW     5.0296e-02 mW     4.7534e+03 nW        0.1410 mW
```

cell.rep:

```
level2_1/unit3/FA1/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit3/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit3/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA1/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA1/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit4/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA1/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA1/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit5/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit6/FA2/HA1/U2 XOR2X1          gscl45nm        4.693000
level2_1/unit6/FA2/HA2/U2 XOR2X1          gscl45nm        4.693000
--------------------------------------------------------------------------------
Total 174 cells                                          574.892480
```

timing.rep.5.final:

```
#####################################################################
#  Generated by:      Cadence Encounter 10.13-s292_1
#  OS:                Linux x86_64(Host ID saturn.ece.iit.edu)
#  Generated on:      Sat Dec  8 09:10:25 2012
#  Design:            adder_10_4bit_tree
#  Command:           report_timing -nworst  10 -net > timing.rep.5.final
#####################################################################
Path 1: MET Late External Delay Assertion
Endpoint:   s[6]  (^) checked with  leading edge of 'vclk'
Beginpoint: a3[1] (^) triggered by  leading edge of 'vclk'
Other End Arrival Time          0.000
- External Delay                0.100
+ Phase Shift                   4.000
= Required Time                 3.900
- Arrival Time                  1.446
= Slack Time                    2.454
```

power.final:

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:      0.1448      54.17%
Total Switching Power:     0.1178      44.05%
Total Leakage Power:     0.004753      1.778%
Total Power:               0.2673
-----------------------------------------------------------------------------------------
```

gate.final:

```
Gate area 2.8158 um^2
Level 0 Module adder_10_4bit_tree                    Gates=       209 Cells=     174 Area=     589.9 um^2
```

Appendix Ⅳ Screenshot

8-operand 4-bit adder:

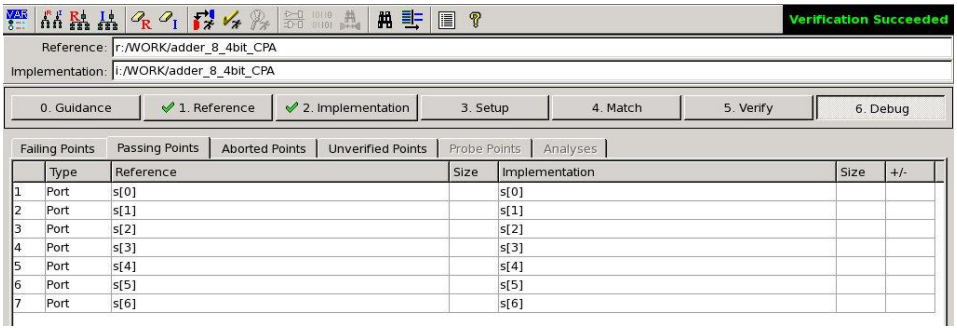
CPA linear model

Simulation result

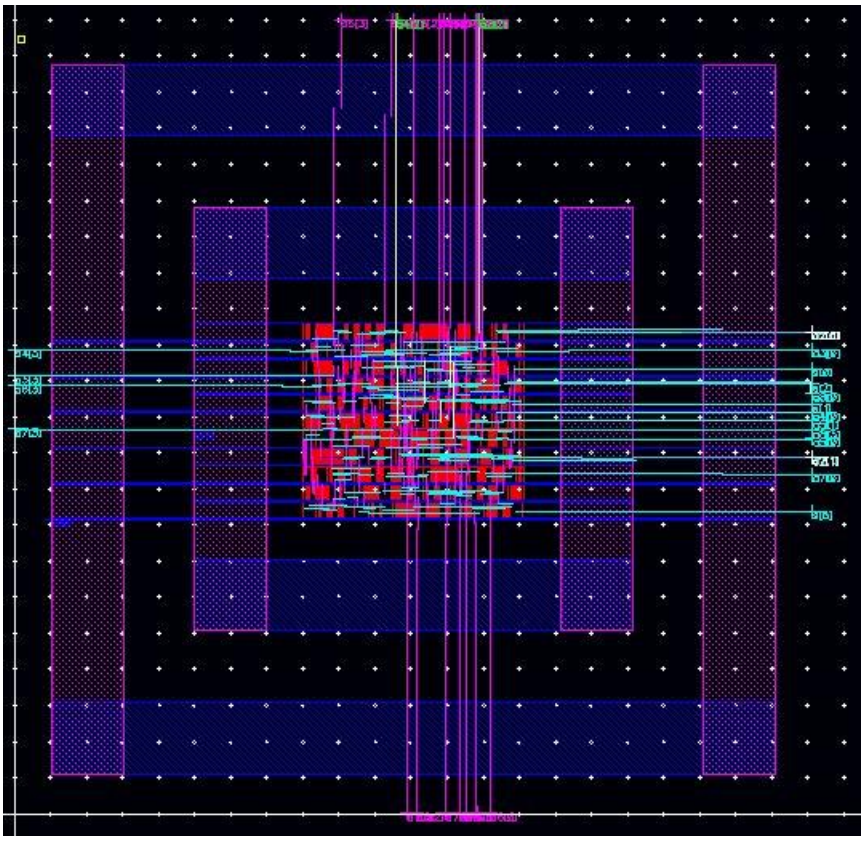
```
Compiling source file "adder_8_4bit_test.v"
Compiling source file "adder_8_4bit_CPA.v"
Highest level modules:
adder_8_4bit_test

At Time:                 100 a1=15 a2=15 a3=15 a4=15 a5=15 a6=15 a7=15 a8=15 Sum
=120
At Time:                 200 a1= 5 a2= 5 a3= 5 a4= 5 a5= 5 a6= 5 a7= 5 a8= 5 Sum
= 40
At Time:                 300 a1=10 a2=10 a3=10 a4=10 a5=10 a6=10 a7=10 a8=10 Sum
= 80
At Time:                 400 a1=15 a2=14 a3=13 a4=12 a5=11 a6=10 a7= 9 a8= 8 Sum
= 92
At Time:                 500 a1= 7 a2= 6 a3= 5 a4= 4 a5= 3 a6= 2 a7= 1 a8= 0 Sum
= 28
At Time:                 600 a1=11 a2=11 a3=11 a4=11 a5=11 a6=11 a7=11 a8=11 Sum
= 88
```

wave form

Equivalence checking result



Final circuit

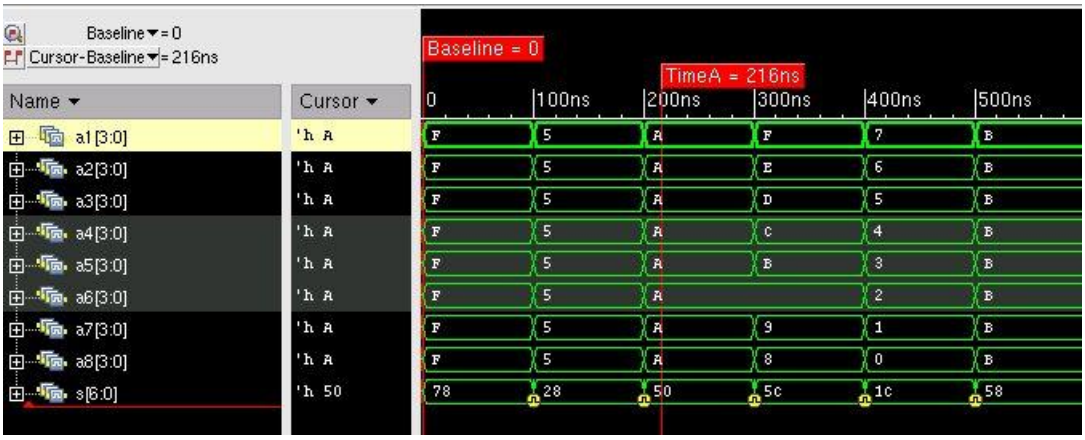


CSA linear model

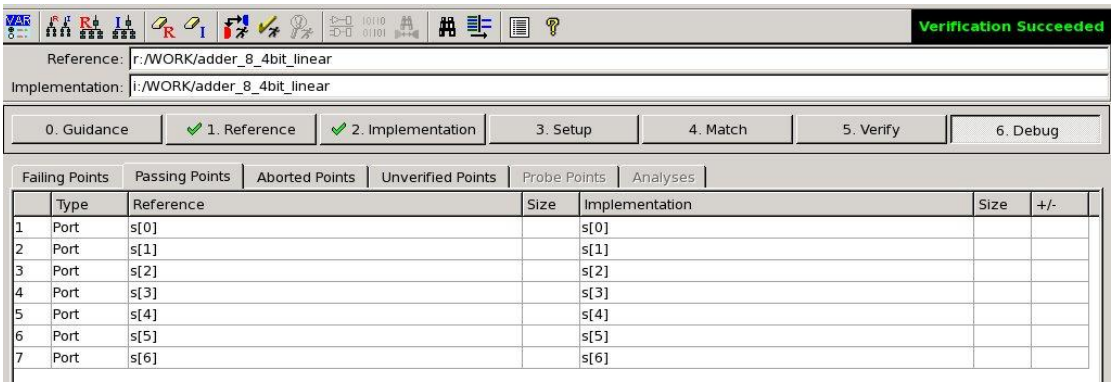
Simulation result

```
Compiling source file "adder_8_4bit_test.v"
Compiling source file "adder_8_4bit_linear.v"
Highest level modules:
adder_8_4bit_test

At Time:                 100 a1=15 a2=15 a3=15 a4=15 a5=15 a6=15 a7=15 a8=15 Sum
=120
At Time:                 200 a1= 5 a2= 5 a3= 5 a4= 5 a5= 5 a6= 5 a7= 5 a8= 5 Sum
= 40
At Time:                 300 a1=10 a2=10 a3=10 a4=10 a5=10 a6=10 a7=10 a8=10 Sum
= 80
At Time:                 400 a1=15 a2=14 a3=13 a4=12 a5=11 a6=10 a7= 9 a8= 8 Sum
= 92
At Time:                 500 a1= 7 a2= 6 a3= 5 a4= 4 a5= 3 a6= 2 a7= 1 a8= 0 Sum
= 28
At Time:                 600 a1=11 a2=11 a3=11 a4=11 a5=11 a6=11 a7=11 a8=11 Sum
= 88
```

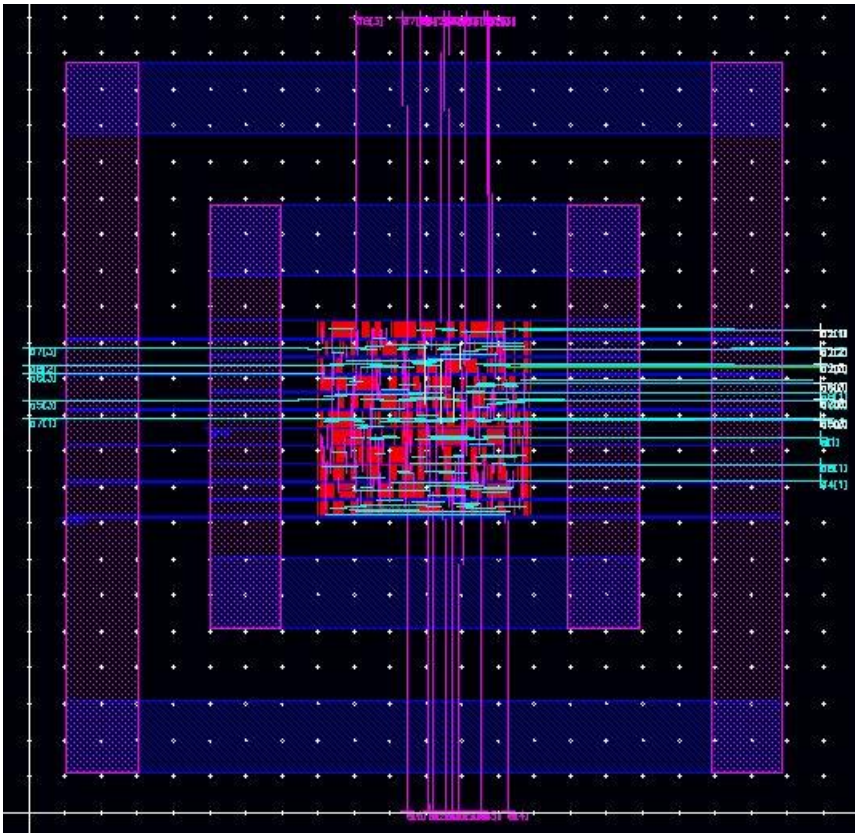
wave form



Equivalence checking result



Final circuit



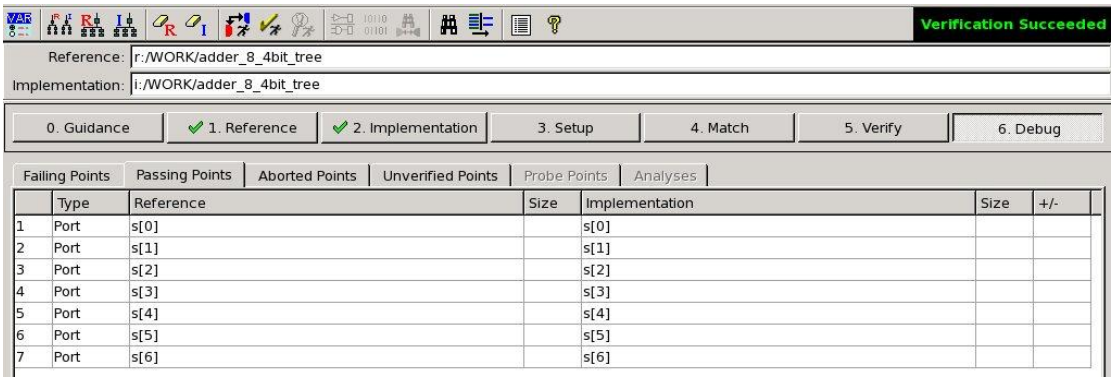
[4:2] tree structure

Simulation result

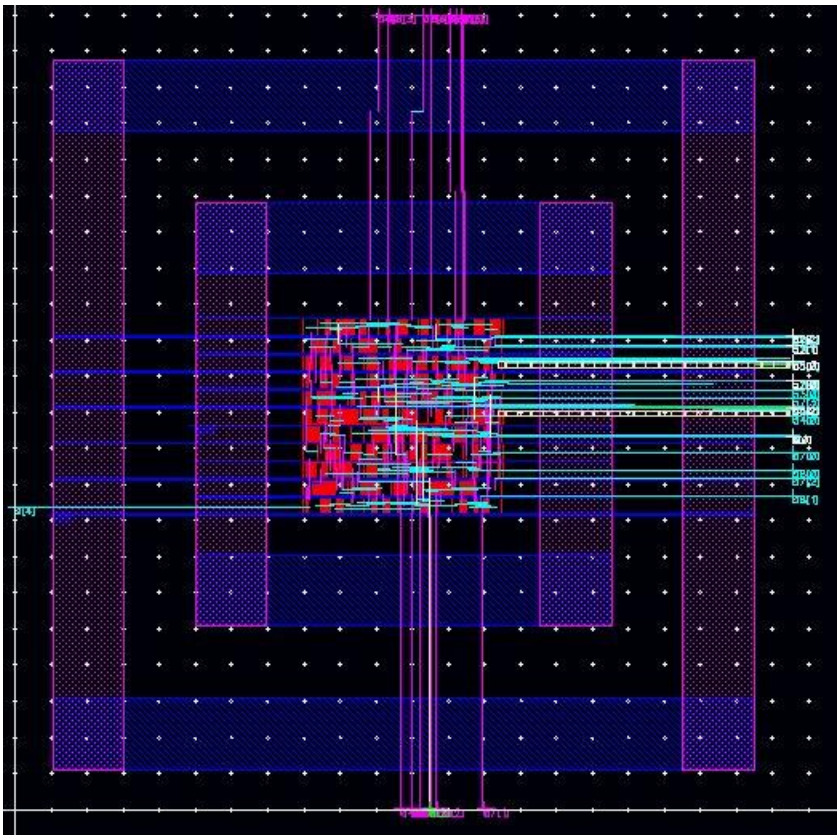
```
Compiling source file "adder_8_4bit_test.v"
Compiling source file "adder_8_4bit_tree.v"
Highest level modules:
adder_8_4bit_test

At Time:                100 a1=15 a2=15 a3=15 a4=15 a5=15 a6=15 a7=15 a8=15 Sum
=120
At Time:                200 a1= 5 a2= 5 a3= 5 a4= 5 a5= 5 a6= 5 a7= 5 a8= 5 Sum
= 40
At Time:                300 a1=10 a2=10 a3=10 a4=10 a5=10 a6=10 a7=10 a8=10 Sum
= 80
At Time:                400 a1=15 a2=14 a3=13 a4=12 a5=11 a6=10 a7= 9 a8= 8 Sum
= 92
At Time:                500 a1= 7 a2= 6 a3= 5 a4= 4 a5= 3 a6= 2 a7= 1 a8= 0 Sum
= 28
At Time:                600 a1=11 a2=11 a3=11 a4=11 a5=11 a6=11 a7=11 a8=11 Sum
= 88
```

wave form

Equivalence checking result



Final circuit



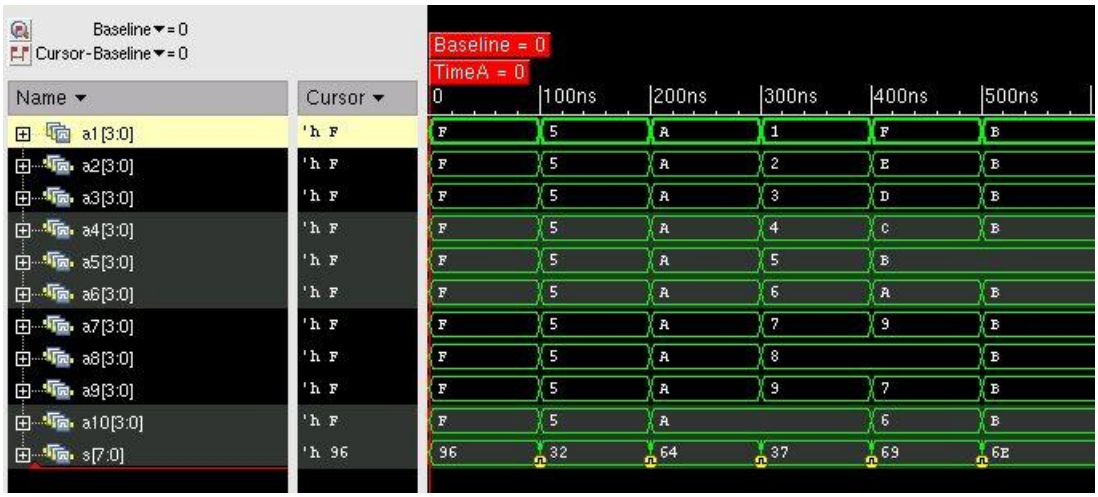
10-operand 4-bit adder:

[5:2] tree structure

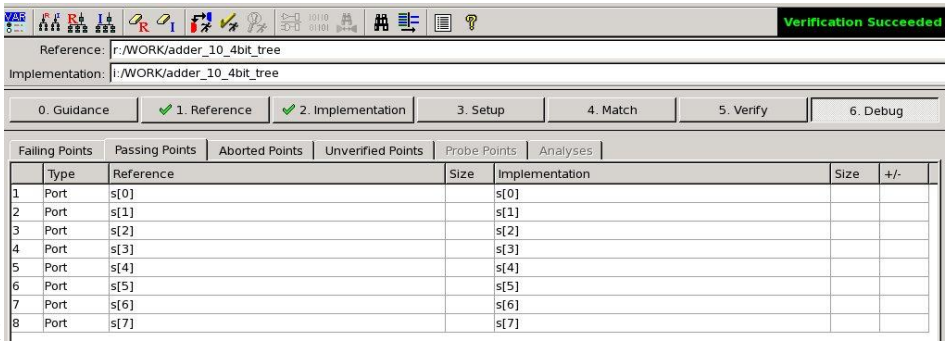
Simulation result

```
Compiling source file "adder_10_4bit_test.v"
Compiling source file "adder_10_4bit_tree.v"
Highest level modules:
adder_10_4bit_test

At Time:                 100 a1=15 a2=15 a3=15 a4=15 a5=15 a6=15 a7=15 a8=15 a9=
15 a10=15 Sum=150
At Time:                 200 a1= 5 a2= 5 a3= 5 a4= 5 a5= 5 a6= 5 a7= 5 a8= 5 a9=
 5 a10= 5 Sum= 50
At Time:                 300 a1=10 a2=10 a3=10 a4=10 a5=10 a6=10 a7=10 a8=10 a9=
10 a10=10 Sum=100
At Time:                 400 a1= 1 a2= 2 a3= 3 a4= 4 a5= 5 a6= 6 a7= 7 a8= 8 a9=
 9 a10=10 Sum= 55
At Time:                 500 a1=15 a2=14 a3=13 a4=12 a5=11 a6=10 a7= 9 a8= 8 a9=
 7 a10= 6 Sum=105
At Time:                 600 a1=11 a2=11 a3=11 a4=11 a5=11 a6=11 a7=11 a8=11 a9=
11 a10=11 Sum=110
```

wave form



Equivalence checking result



x

Final circuit