# Exercise1 Full adder Simulation

张旭东 12011923

## 1.Introduction

The purpose of this lab is to be familiar with the development process and do the simulation of the VHDL code of Full adder. Also, what we need to do is to design a test bench to simulate the VHDL code in behavior simulation,post synthesis simulation and post place and route simulation.

What's the most important in this lab is to observe whether the simulation result is consistent with those shown in lecture notes or not and whether the addition result of the full adder is consistent with common sense and explain the reasons behind them. What's more, it is considerable to distinguish the differences among the $3$ simulations and explain the reason for spikes' presence if them exist.

## 2.Lab result and analysis

### 2.1 Prelab exercise

Before beginning simulation, it is necessary to do theoretical analysis and derive results, which provides references for later simulation results. The digital circuit of full adder is as shown in `Fig.1`:

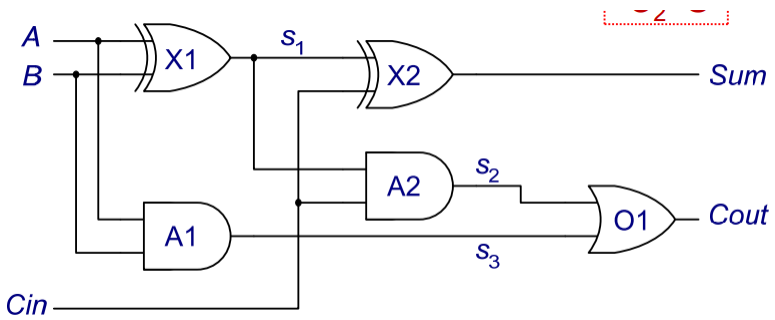


**Fig.1 Gate circuit diagram of full adder**

$A$ and $B$ are addends, $C_{in}$ is the low carry, $Sum$ is the result of addition, $C_{out}$ is the carry to the top and $S_1$, $S_2$ and $S_3$ are intermediate signal. The relations between them are:

$$
\begin{aligned}
s_1 &= A \bigoplus B \\
Sum &= s_1 \bigoplus C_{in} = (A \bigoplus B) \bigoplus C_{in} \\
s_3 &= AB \\
s_2 &= C_{in} s_1 = C_{in}(A \bigoplus B) \\
C_{out} &= s_2 + s_3 = (C_{in}(A \bigoplus B)) + (AB)
\end{aligned} \tag{1}
$$

According to the definition of full adder, the truth table of full adder is as shown in `Fig.2`.

| Input | | | Output | |
|---|---|---|---|---|
| A | B | Cin | Sum | Cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Fig.2 Truth table of full adder**

```
L1: s1 <= (A xor B) after gate_delay;
L2: s2 <= (Cin and s1) after gate_delay;
L3: s3 <= (A and B) after gate_delay;
L4: sum <= (s1 xor Cin) after gate_delay;
L5: cout <= (s2 or s3) after gate_delay;
```

**Fig.3 Inputs of full adder**

According to the truth table and `Formula 1`, the table and waveform of the result can be derived when the inputs are like `Fig.3` . The results are as shown in `Fig.4`

| | 0-10ns | 10-20ns | 20-30ns | 30-40ns | 40-50ns | 50-60ns | 60-70ns | 70-80ns |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Cin | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| s1 | u | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| s2 | u | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| s3 | u | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| Sum | u | u | 0 | 0 | 0 | 1 | 0 | 0 |
| Cout | u | u | 0 | 1 | 1 | 0 | 0 | 1 |

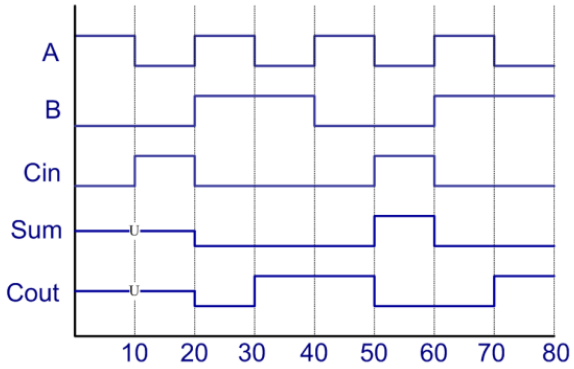


**Fig.4 derived table and waveform of full adder**

## 2.2 VHDL code and behavioral simulation

The first step to do behavioral simulation is to write the VHDL code for the full-adder according to `Formula 1`. The VHDL code is  shown as below:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
    Port ( A : in STD_LOGIC;
```

```
            B : in STD_LOGIC;
            Cin : in STD_LOGIC;
            Sum : out STD_LOGIC;
            Cout : out STD_LOGIC);
end entity full_adder;

architecture dataflow of full_adder is
    signal s1,s2,s3:std_logic ;
    constant gate_delay:time:=10ns;
begin
    L1:s1<=(A xor B) after gate_delay;
    L2:s2<=(Cin and s1) after gate_delay;
    L3:s3<=(A and B) after gate_delay;
    L4:Sum<=(s1 xor Cin) after gate_delay;
    L5:Cout<=(s2 or s3) after gate_delay;

end architecture dataflow;
```

The code is divided into two parts mainly. The first part is entity declaration and the second part is architectures.

The entity declaration defines an interface to a component and describes the input and output ports that can be seen from the outside. In the entity declaration part, three input ports are declared as $A$, $B$ and $C_{in}$, whose data types are `std_logic`. Two out ports are also declared as $Sum$ and $C_{out}$ , whose data types are `std_logic`.

The architecture defines the relationships between the inputs and outputs of a design entity, which provides an internal view of a component. It consists of a declaration section followed by a collection of concurrent statements and may be expressed in terms of behavior, data flow, or structure. In the architecture part, three intermediate signals, $s_1$, $s_2$, $s_3$ are declared, which are used to represent the outputs of internal gate circuit in order to assignment the signal of out ports. Besides, a constant data objects, `gate_delay` is declared to simulate the gate delay in the real circuits.

The second step to do behavioral simulation is write the testbench code for the full-adder. The VHDL code is shown as below:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_full_adder is
```

```
--  Port ( );
end tb_full_adder;

architecture Behavioral of tb_full_adder is
    component full_adder is
        port(A,B,Cin: in std_logic ;Sum,Cout: out std_logic );
    end component full_adder;
    signal A_tb,B_tb,Cin_tb: std_logic ;
    signal Sum_tb,Cout_tb:std_logic ;

begin
    UUT:full_adder port
map(A=>A_tb,B=>B_tb,Cin=>Cin_tb,Sum=>Sum_tb,Cout=>Cout_tb);
    A_tb<='1','0'after 10ns,'1'after 20ns,'0' after 30ns,'1'after 40ns,'0'
after 50ns,'1'after 60ns,'0' after 70ns;
    B_tb<='0','1'after 20ns,'0'after 40ns,'1'after 60ns;
    Cin_tb<='0','1'after 10ns,'0'after 20ns,'1'after 50ns,'0'after 60ns;


end Behavioral;
```

An very important point needed to be considered is that the top level entity of the test bench has no external ports, which means there are no input ports and out ports declared in the entity declaration. In the architecture part, the full adder component under test, the stimulus signals and the observed signals are declared and defined. What's more, stimulus values are generated according to the value of $A$, $B$ and $C_{in}$ in `Fig.4`.

The result of behavioral simulation are as shown in `Fig.5`.

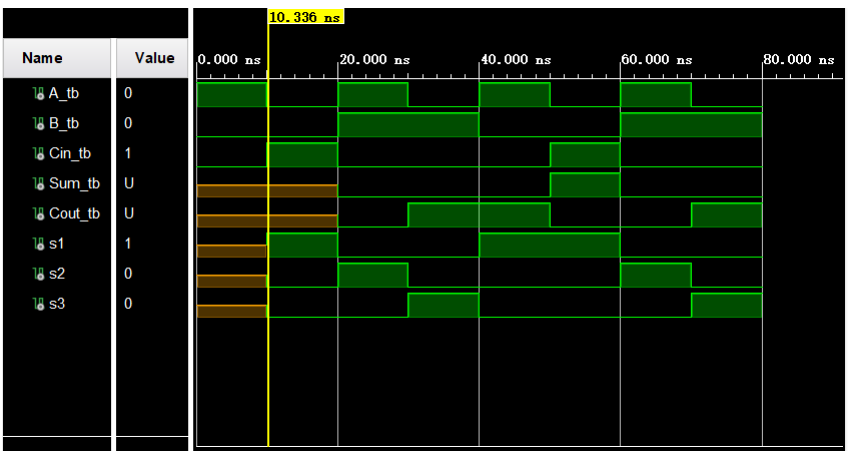

**Fig.5 Result of behavioral simulation**

It is obvious that the result is consistent with the derived table in `Fig.4` and those shown in lecture notes. The yellow lines means the value of that signal is unknown in that time. The values of $s_1$, $s_2$ and $s_3$ in the first $10ns$ are unknown and the values of $Sum$ and $C_{out}$ in the first $20ns$ are unknown, which is consistent with our expectation. According to `Fig.3`, $s_1$ and $s_3$ get value of $A \bigoplus B$ and $AB$ after `gate_delay`. And $s_2$ get value of $C_{in}s_1$ after `gate_delay`. In the first `gate_delay`, $C_{in}$ is 0 and $s_1$ is $U$, which don't affect the value of $C_{in}s_1$ is 0. So, the value of $s_2$ in the second $10ns$ is 0. The values of $Sum$ and $C_{out}$ come form $s_1$, $s_2$ and $s_3$, which means they have to go through an unknown period twice as long.

However, the result isn't consistent with common sense. The behavioral simulation can be used to check the syntax errors in the code and the correctness of the code behavior, not including the delay information except the delay in the signal assignment statement. So according to this point, the behavioral simulation holds the $10ns$ delay in the signal assignment statement and all of the results are derived from the $10ns$.

## 2.3 Post-Synthesis

The post-synthesis converts the circuit logic described by language into the interconnection relation with and-gate, or-gate, non-gate, flip-flop and other basic logic units. Synthesizable means that our code can be translated into a gate level circuit, and nonsynthesizable means that the code can't be translated into its corresponding gate level circuit. An very important point is that the post-synthesis will ignore the delay in the signal assignment statement. That means the delay between signals is regarded as 0 in theory.

The post- synthesis functional simulation implies that only the logic function described by VHDL code are tested and simulated to indicate whether the logic circuit meet the ideal case, which does not involve any hardware features. The result isn't consistent with those shown in lecture notes while it is consistent with common sense. The reason is that the gate_delay assigned in statement is ignored. That's why it worked out that way.

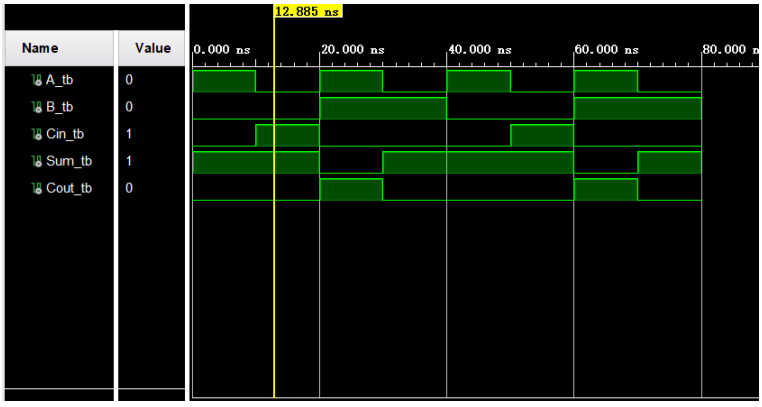

**Fig.6 Result of Post-Synthesis functional simulation**

The result of post-synthesis timing simulation is as shown in `Fig.7`. It is obviously that there exists a delta delay in the beginning, which is different from behavioral simulation and post-synthesis functional simulation. The value of delta delay is 2.576ns. Compared with the result of post-synthesis functional simulation, the result of post-synthesis timing simulation shifts that to the right by delta delay. The reason is that although multiple signal assignment statements are executed concurrently in simulated time and are referred to as concurrent signal assignment statements, a small delay——delta delay will be automatically set in order to make a logical sequence of signals transmission. The red lines in the beginning show the point. The post-synthesis timing simulation treats the delays of all gate circuits as the same. However, as we all know, different kinds of gate circuits have different delays in reality. But it is helpful for us to know the actual circuit condition.

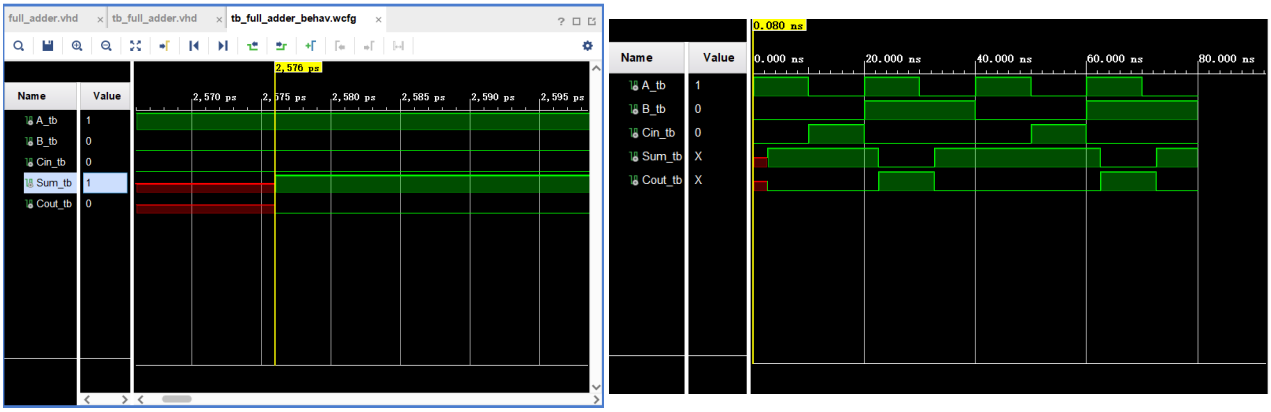


**Fig.7 Result of Post-Synthesis timing simulation**

## 2.4 Post-Implementation

The gate level netlist generated after synthesis only represents the virtual connection relationship between gate circuits, but does not specify the location of each gate circuits and the length of the connection. The basis of FPGA repeatable programming is the huge amount of configurable logic block (CLB), rich wiring resources and other resources. The purpose of post-implementation is to do the place and route. The process of place is the process of "placing" each gate in the gate level netlist into the CLB. This process is a mapping process. Routing is the process of connecting CLBS together according to logical relation by using rich wiring resources in FPGA.

The result of post-implementation functional simulation is as shown in `Fig8`. The result isn't consistent with those shown in lecture notes while it is consistent with common sense, which is basically the same as the result of post-synthesis functional simulation. That means the design meets our requirement perfectly.

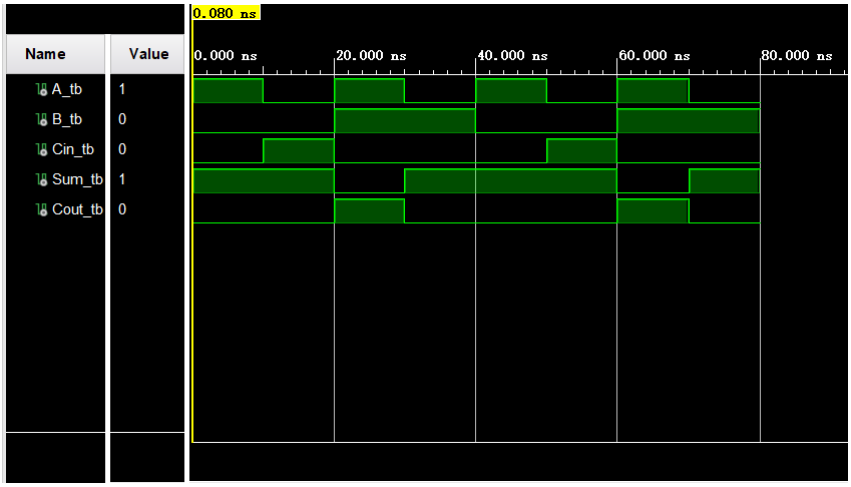


**Fig.8 Result of Post-Implementation functional simulation**

The result of post-implementation timing simulation is as shown in `Fig.9`. Obviously, it isn't the same as that of post-synthesis timing simulation and there are many spikes in the waveform of $Sum$ and $C_{out}$. What's more, the delays in the beginning of $Sum$ and $C_{out}$ are different. That is, the time delays of different kinds of gate circuits are different in post-implementation timing simulation, which isn't the same as post-synthesis timing simulation. What's more, the number of gate circuits $Sum$ and $C_{out}$ go through are different, which leads to the circuit signals arrive at them in different order.

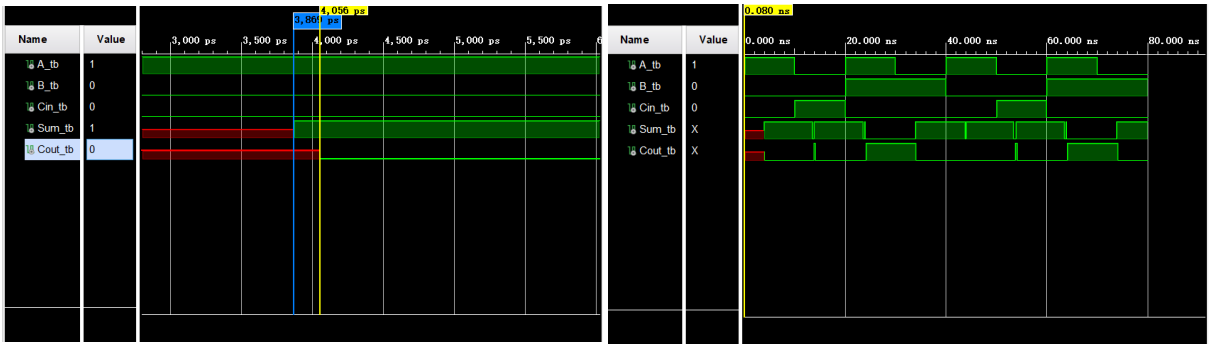


**Fig.9 Result of Post-Implementation timing simulation**

Enlarge the result to analyze the spikes.

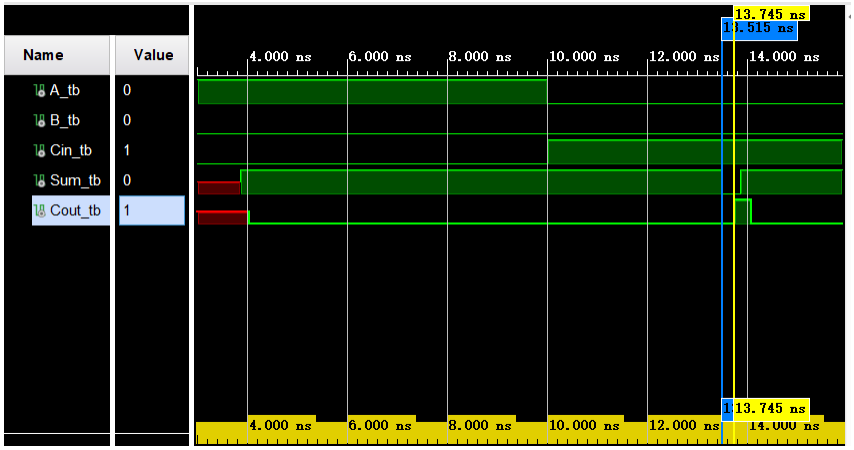

**Fig.10 Result of Post-Implementation timing simulation**

For the spike of $C_{out}$ waveform near 14ns, one possible reason for it changes from 0 to 1 and quickly recovers to 0 is that after 10ns, the values of $A$ and $C_{in}$ have changed. Before changing, the value of $s_1$ is 1. Once the value of $C_{in}$ changes from 0 to 1 and the value of $s_1$ is still 1 at this time, the expression of $s_2 = C_{in}s_1$ gives $s_2$ the value of 1 and the value of $s_3$ is still 0. Then, the expression of $C_{out} = s_2 + s_3$ gives $C_{out}$ the value of 1. After $s_1$ finishes the expression of $s_1 = A \bigoplus B$ ,$s_1$ changes to value of 0 and $s_2$ changes to value of 0. Then $C_{out}$ has the value of 0 again.

## 3.Conclusion

The main part of this report is the process of digital design of full adder using VIVADO and analysis of differences between the 3 simulation results, including behavioral simulation, post-synthesis simulation and post-implementation simulation. The importance of the first experiment is to let us have clear understanding of the differences between the 3 simulation.

There are some important points that need to be stressed again:

1. The behavioral simulation can be used to check the syntax errors in the code and the correctness of the code behavior, not including the delay information except the delay in the signal assignment statement. The delay assigned in codes will be considered by it.
2. The post-synthesis converts the circuit logic described by language into the interconnection relation with and-gate, or-gate, non-gate, flip-flop and other basic logic units and ignore the delay in the signal assignment statement. The only delay considered by post-synthesis timing simulation is the delta delay, which is automatically set in order to make a logical sequence of signals transmission.
3. Post-implementation considers the place and route. In post-implementation timing simulation, the time delays of different kinds of gate circuits are different. Although the result of post-implementation timing simulation are close to the simulation of real device operation, there are still some differences between them because the time delay of one kind of gate circuits in simulation of VIVADO is different to that of the same kind of gate circuit in actual hardware.

The above points need to be considered and analyzed in the later experiments.