湖南大學

数字电路与逻辑设计

课程实验报告

题　　目：　加法器、运算器的设计

学生姓名：　魏子铖

学生学号：　201726010308

专业班级：　软件 1703

完成时间：　2018.12.6

实验三 加法器、运算器的设计

班级 软件 1703 姓名 魏子铖 学号 201706010308

一、实验目的

1．用 VHDL 语言设计全加器；
2．利用设计的全加器组成四位并行加法器；
3．用逻辑图和 VHDL 语言设计四位串行加法器。
4．使用 VHDL 语言设计模型机运算器 ALU。

二、实验内容

1．熟悉 Quartus II 软件的基本操作，了解各种设计方法（原理图设计、文本设计、波形设计）
2．用逻辑图和 VHDL 语言设计一个全加器，最后仿真验证。
3．利用设计的全加器，用逻辑图和 VHDL 语言设计一个四位并行加法器，最后仿真验证。
4．用逻辑图和 VHDL 语言设计一个四位串行加法器，最后仿真验证。
5．使用 VHDL 语言设计模型机运算器 ALU，最后仿真验证。

三、实验方法

1. 实验方法

用 VHDL 语言设计一个全加器

1. 新建，编写源代码。

(1).选择保存项：【File】-【new project wizard】-【next】（设置文件路径+设置 project name 为 FullAdder_VHDL +设置 top-level design entity name 为 FullAdder_VHDL _VHDL）-【finish】

(2).新建：【file】-【new】-【VHDL File】-【OK】

2.写好源代码，保存文件（FullAdder_VHDL.vhd）

3.编译与调试。

确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译。编译结果有 4 个警告，文件编译成功。

4.波形仿真及验证。

(1).新建一个 vector waveform file，保存文件（FullAdder_VHDL.vwf）。。

(2).设置仿真模式【Assignments】-【Setting】-【Simulator Settings】-【Simulation mode】选择 Functional

(3).设置 end time（操作为：【edit】-【End Time】（设置为 8ns）-【ok】）。

(4).按照程序所述插入 X,Y,Z,C,S 三个节点（X,Y,Z 为输入节点，C,S 为输出节点）。

(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all；【list】）-【>>】-【ok】-【ok】）。

(5).设置 X,Y,Z 的输入波形…点击保存按钮保存。

（操作为：点击 name（如：Z）-右击-【value】-【clock】（如设置 period=2ns；offset=0ns），同理设置 name X,Y（如 period=8/4ns；offset=0ns），保存）。

(6).然后【Processing】-【Generate Functional Simulation Netlist】；

(7).然后【Processing】-【start simulation】，形成 name C，S 的输出图。

5.查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

用逻辑图设计一个全加器

1.新建，编写源代码。

(1).选择保存项:【File】-【new project wizard】-【next】(设置文件路径+设置 project name 为 FullAdder +设置 top-level design entity name 为 FullAdder）-【finish】

(2).新建:【file】-【new】-【Block Diagrame/Schematic File】-【OK】

2.画好原理图，保存文件（FullAdder.bdf）

3.编译与调试。

确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译。编译结果有 5 个警告，文件编译成功。

4.波形仿真及验证。

(1).新建一个 vector waveform file，保存文件（FullAdder_VHDL.vwf)。。

(2).设置仿真模式【Assignments】-【Setting】-【Simulator Settings】-【Simulation mode】选择 Functional

(3).设置 end time（操作为:【edit】-【End Time】(设置为 8ns）-【ok】)。

(4).按照程序所述插入 X,Y,Z,C,S 三个节点（X,Y,Z 为输入节点，C,S 为输出节点)。

(操作为：右击 -【insert】-【insert node or bus】-【node finder】(pins=all;【list】) -【>>】-【ok】-【ok】)。

(5).设置 X,Y,Z 的输入波形…点击保存按钮保存。

（操作为：点击 name（如：Z）-右击-【value】-【clock】(如设置 period=2ns; offset=0ns)，同理设置 name X,Y（如 period=8/4ns；offset=0ns)，保存)。

(6).然后【Processing】-【Generate Functional Simulation Netlist】;

(7).然后【Processing】-【start simulation】，形成 name C，S 的输出图。

5.查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

用 VHDL 语言，利用设计的全加器组成四位并行加法器

1.新建，编写源代码。

(1).选择保存项:【File】-【new project wizard】-【next】(设置文件路径+设置 project name 为 Adder_4_VHDL+设置 top-level design entity name 为 Adder_4_VHDL）-【finish】

(2).新建:【file】-【new】-【VHDL File】-【OK】

2. 利用写好的 FullAdd_VHDL.vhd 文件,添加到顶层文件中作为元件。写好源代码,保存文件（Adder_4_VHDL.vhd）

3.编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译。编译结果有 5 个警告，文件编译成功。

4.波形仿真及验证。

(1).新建一个 vector waveform file，保存文件（Adder_4_VHDL.vwf)。

(2).设置仿真模式【Assignments】-【Setting】-【Simulator Settings】-【Simulation mode】选择 Functional
(3).设置 end time（操作为：【edit】-【End Time】（设置为 512ns）-【ok】）。
(4).按照程序所述插入 C0，A[3..0], B[3..0]，C4,S[3..0]（C0，A[3..0], B[3..0]为输入节点，C4,S[3..0]为输出节点）。
(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】）。
(5).设置 C0，A[3..0], B[3..0]的输入波形…点击保存按钮保存。
（操作为：点击 name（如：B[0]））-右击-【value】-【clock】（如设置 period=2ns; offset=0ns），同理设置 name B[1..3]A[0..3]（如 period=4/8/16/32/64/128/256ns; offset=0ns），name C0（设置为'0'）,保存）。
(6).然后【Processing】-【Generate Functional Simulation Netlist】;
(7).然后【Processing】-【start simulation】，形成 name C4 和 S[3..0]的输出图。

5.查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

用逻辑图，利用设计的全加器组成四位并行加法器
1.新建，编写源代码。
(1).选择保存项:【File】-【new project wizard】-【next】(设置文件路径+设置 project name 为 Adder_4_VHDL +设置 top-level design entity name 为 Adder_4_VHDL）-【finish】
(2).新建:【file】-【new】-【Block Diagrame/Schematic File】-【OK】
2.利用写好的 FullAdd.bdf 文件，保存为 FullAdd.bsf 文件，添加到顶层文件中作为元件。写好源代码，保存文件（Adder_4_VHDL.bdf）

3.编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译。编译结果有 5 个警告，文件编译成功。

4.波形仿真及验证。
(1).新建一个 vector waveform file，保存文件（Adder_4_VHDL.vwf）。
(2).设置仿真模式【Assignments】-【Setting】-【Simulator Settings】-【Simulation mode】选择 Functional
(3).设置 end time（操作为：【edit】-【End Time】（设置为 512ns）-【ok】）。
(4).按照程序所述插入 C0，A[3..0], B[3..0]，C4,S[3..0]（C0，A[3..0], B[3..0]为输入节点，C4,S[3..0]为输出节点）。
(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】）。
(5).设置 C0，A[3..0], B[3..0]的输入波形…点击保存按钮保存。
（操作为：点击 name（如：B[0]））-右击-【value】-【clock】（如设置 period=2ns; offset=0ns），同理设置 name B[1..3]A[0..3]（如 period=4/8/16/32/64/128/256ns; offset=0ns），name C0（设置为'0'）,保存）。
(6).然后【Processing】-【Generate Functional Simulation Netlist】;
(7).然后【Processing】-【start simulation】，形成 name C4 和 S[3..0]的输出图。

5.查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

用 VHDL 语言设计 4 位串行加法器

1.新建，编写源代码。

(1).选择保存项:【File】-【new project wizard】-【next】(设置文件路径+设置 project name 为 SerialAdder_4_VHDL+ 设置 top-level design entity name 为 SerialAdder_4_VHDL）-【finish】

(2).新建:【file】-【new】-【VHDL File】-【OK】

2.写好源代码，保存文件（SerialAdder_4_VHDL.vhd 和 FullAdder.vhd 和 SRG_4.vhd 和 Dffx.vhd）

编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译。编译结果有 5 个警告，文件编译成功。

4、波形仿真及验证。

(1).新建一个 vector waveform file,保存文件(SerialAdder_4_VHDL.vwf)

(2).设置仿真模式【Assignments】-【Setting】-【Simulator Settings】-【Simulation mode】选择 Functional

(3).设置 end time（操作为:【edit】-【End Time】(设置为 160us）-【ok】)。

(4).按照程序所述插入 Shift,Clock,Reset,SI 为输入节点，Q,O 为输出节点)。

(操作为：右击 -【insert】-【insert node or bus】-【node finder】(pins=all;【list】)-【>>】-【ok】-【ok】)。

(5).设置 Shift,Clock,Reset,SI 的输入波形…点击保存按钮保存。

（操作为：点击 name（如：Reset））-在波形图中选取合适的一段设定,值为 0，之后设置为 1，同理设置 name Shift(全部为 1)Clock(在 Reset 后连续 12 个周期)的各个节点，SI（可以任意设置），保存)。

(6).然后【Processing】-【Generate Functional Simulation Netlist】;

(7).然后【Processing】-【start simulation】，形成 name Q，O 的输出图。

5.查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

用逻辑图设计 4 位串行加法器

1.新建，编写源代码。

(1).选择保存项:【File】-【new project wizard】-【next】(设置文件路径+设置 project name 为 SerialAdder_4+设置 top-level design entity name 为 SerialAdder_4）-【finish】

(2).新建:【file】-【new】-【VHDL File】-【OK】

2. 利用写好的 FullAdd.bdf 文件，保存为 FullAdd.bsf 文件，添加到顶层文件中作为元件。同理，编写 SRG4_CLR（带重置功能的 SRG4 寄存器）保存为 SRG4_CLR.bsf 文件，添加到顶层文件中作为元件写好源代码，保存文件（SerialAdder_4.bdf）

编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译。编译结果有 25 个警告，文件编译成功。

4、波形仿真及验证。
(1).新建一个 vector waveform file,保存文件(SerialAdder_4.vwf)
(2).设置仿真模式【Assignments】-【Setting】-【Simulator Settings】-【Simulation mode】选择 Functional
(3).设置 end time（操作为：【edit】-【End Time】（设置为 150us）-【ok】）。
(4).按照程序所述插入 Shift,Clock,Reset,SI 为输入节点，Q,O 为输出节点）。
（操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】）。
(5).设置 Input[7..0]的输入波形…点击保存按钮保存。
（操作为：点击 name（如：Reset））-在波形图中选取合适的一段设定,值为 0，之后设置为 1，同理设置 name Shift(全部为 1)Clock(在 Reset 后连续 12 个周期)的各个节点，SI（可以任意设置），保存）。
(6).然后【Processing】-【Generate Functional Simulation Netlist】；
(7).然后【Processing】-【start simulation】，形成 name Q，O 的输出图。

5.查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

用 VHDL 语言设计模型机运算器 ALU
1.新建，编写源代码。
(1).选择保存项:【File】-【new project wizard】-【next】(设置文件路径+设置 project name 为 ALU+设置 top-level design entity name 为 ALU）-【finish】
(2).新建:【file】-【new】-【VHDL File】-【OK】
2.写好源代码，保存文件（ALU.vhd）

编译与调试。确定源代码文件为当前工程文件，点击【processing】-【start compilation】进行文件编译。编译结果有 5 个警告，文件编译成功。

4、波形仿真及验证。
(1).新建一个 vector waveform file,保存文件(ALU.vwf)
(2).设置仿真模式【Assignments】-【Setting】-【Simulator Settings】-【Simulation mode】选择 Functional
(3).设置 end time（操作为：【edit】-【End Time】（设置为 160us）-【ok】）。
(4).按照程序所述插入 Input[7..0]为输入节点，Output[1..0]为输出节点）。
(操作为：右击 -【insert】-【insert node or bus】-【node finder】（pins=all;【list】）-【>>】-【ok】-【ok】）。
(5).设置 Input 的输入波形（按照需要测试的数据进行设置，例如前四位一次设置为“1001” “0110”“1011”“0101”“1010”,后四位随机测试）,点击保存按钮保存。
(6).然后【Processing】-【Generate Functional Simulation Netlist】；
(7).然后【Processing】-【start simulation】，形成 name Output[1..0]的输出图。

5.查看 RTL Viewer:【Tools】-【netlist viewer】-【RTL viewer】。

四、实验过程

1、 a．用 VHDL 语言和逻辑图设计一个全加器

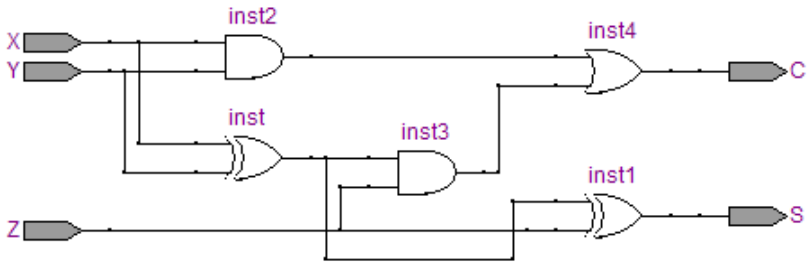
编译过程

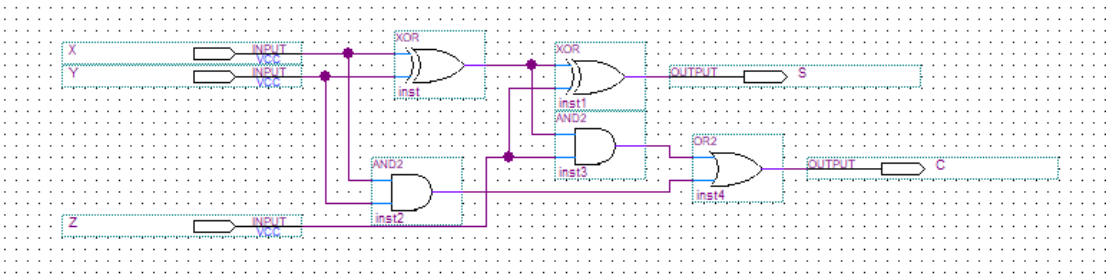
```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   entity FullAdd_VHDL is
4       port (x,y,z: in std_logic;
5             c,s: out std_logic);
6   end FullAdd_VHDL;
7
8   architecture rtl of FullAdd_VHDL is
9   begin
10      s <= (x xor y) xor z;
11      c <= (x and y) or ( z and (x or y));
12  end rtl;
```

a）源代码如图（VHDL 设计）

原理图如图

RTL 如图





b)编译、调试过程

编译通过，有 4 个警告;

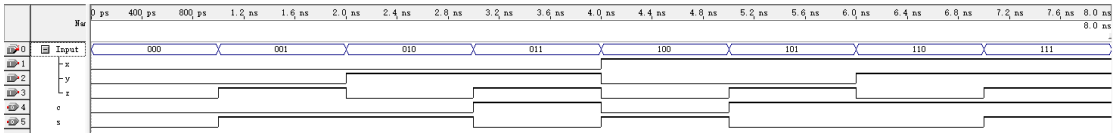
c)结果分析及结论

X,Y,Z 端口输入 3 个二进制数，相加后最后一位结果通过 S 输出，进位通过 C 输出。

波形仿真

a)波形仿真过程（详见实验步骤）

b)波形仿真波形图



c)结果分析及结论

0-1ns: X,Y,Z 输入 000，相加结果为 0，S 输出 0，C 输出 0，正确

1-2ns: X,Y,Z 输入 001，相加结果为 1，S 输出 1，C 输出 0，正确

2-3ns: X,Y,Z 输入 010，相加结果为 1，S 输出 1，C 输出 0，正确

3-4ns: X,Y,Z 输入 011，相加结果为 10，S 输出 0，C 输出 1，正确

4-5ns: X,Y,Z 输入 100，相加结果为 1，S 输出 1，C 输出 0，正确

5-6ns: X,Y,Z 输入 101，相加结果为 10，S 输出 0，C 输出 1，正确

6-7ns: X,Y,Z 输入 110，相加结果为 10，S 输出 0，C 输出 1，正确

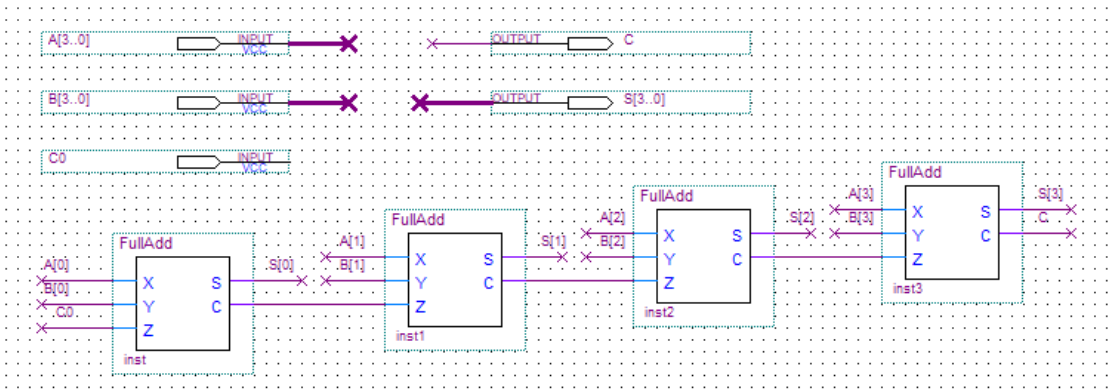
7-6ns: X,Y,Z 输入 111，相加结果为 11，S 输出 1，C 输出 1，正确

b．利用设计的全加器，用 VHDL 语言和原理图设计四位并行加法器

编译过程

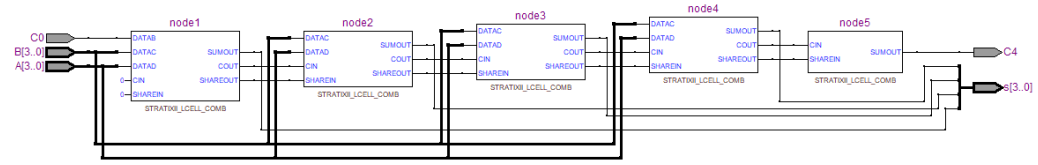
a）源代码如图（VHDL 设计）

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity Adder_4_VHDL is
6      port(A,B:in std_logic_vector(3 downto 0);
7           C0: in std_logic;
8           s: out std_logic_vector(3 downto 0);
9           C4: out std_logic);
10 end Adder_4_VHDL;
11
12 architecture behavioral of Adder_4_VHDL is
13 component FullAdd_VHDL
14     port(X,Y,Z : in std_logic;
15          c,s : out std_logic);
16 end component;
17 signal d,e,f:std_logic;
18 begin
19     a1:FullAdd_VHDL port map(a(0),b(0),c0,d,s(0));
20     a2:FullAdd_VHDL port map(a(1),b(1),d,e,s(1));
21     a3:FullAdd_VHDL port map(a(2),b(2),e,f,s(2));
22     a4:FullAdd_VHDL port map(a(3),b(3),f,c4,s(3));
23 end behavioral;
```



逻辑图如图

RTL 如图



b)编译、调试过程

编译通过，有 5 个警告;

c)结果分析及结论

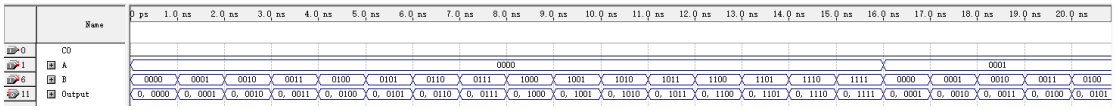
C0 提供输入但是固定输入为 0，其他 4 个全加器分别处理最低位到最高位，每一个全加器输出一位结果，最后一个全加器额外输出一个进位。

例如输入“0000”和“1111”，输出结果为‘0’和“1111”;

波形仿真

a)波形仿真过程（详见实验步骤）

b)波形仿真波形图



c)结果分析及结论

以 2ns 为一周期，每周期两个 4 位二进制数相加，得到得 5 位结果（包括最后一位进位和 4 位结果），均正确

c.用 VHDL 语言和原理图设计一个串行加法器

编译过程

a）源代码如图（VHDL 设计）

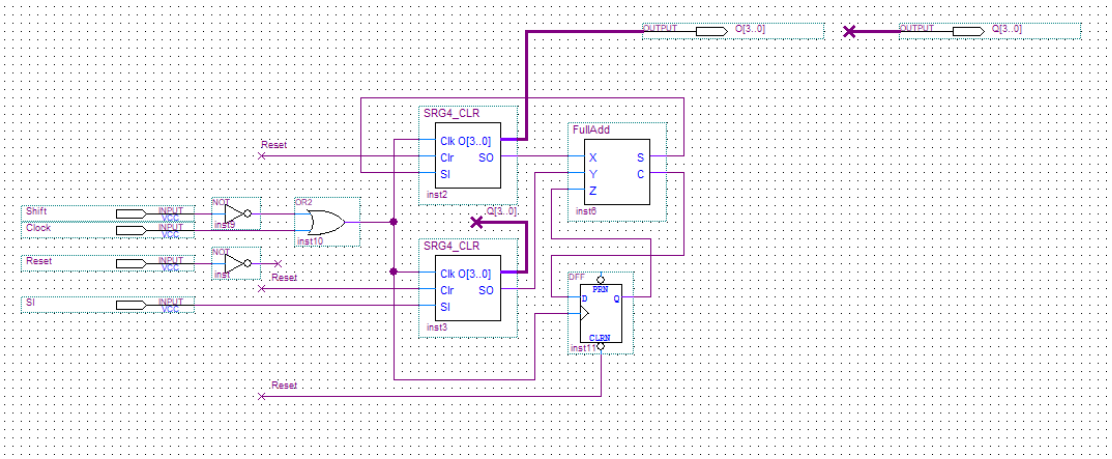
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity SerialAdder_4_VHDL is
    port(SI,Clock,Shift,Reset:in std_logic;
         Q,O: out std_logic_vector(3 downto 0));
end SerialAdder_4_VHDL;

architecture rtl of SerialAdder_4_VHDL is
    signal Clk,Clr,A,B,S,C,QD:std_logic;

    component SRG_4
    port(SI,Clk,Clr:in std_logic;
    S0:out std_logic;
    S1:out std_logic_vector(3 downto 0));
    end component SRG_4;

    component FullAdder
    port(x,y,z:in std_logic;
         s,c: out std_logic);
    end component FullAdder;

    component Dffx
    port(D,Clk,Clr:in std_logic;
         Q:out std_logic);
    end component Dffx;

    begin
    Clk<= (not shift or Clock);
    Clr<= (not Reset);
    SRG_A:SRG_4 port map(S,Clk,Clr,A,O);
    SRG_B:SRG_4 port map(SI,Clk,Clr,B,Q);
    Df:Dffx port map(C,Clk,Clr,QD);
    FullAdd:FullAdder port map(A,B,QD,S,C);
end rtl;
```

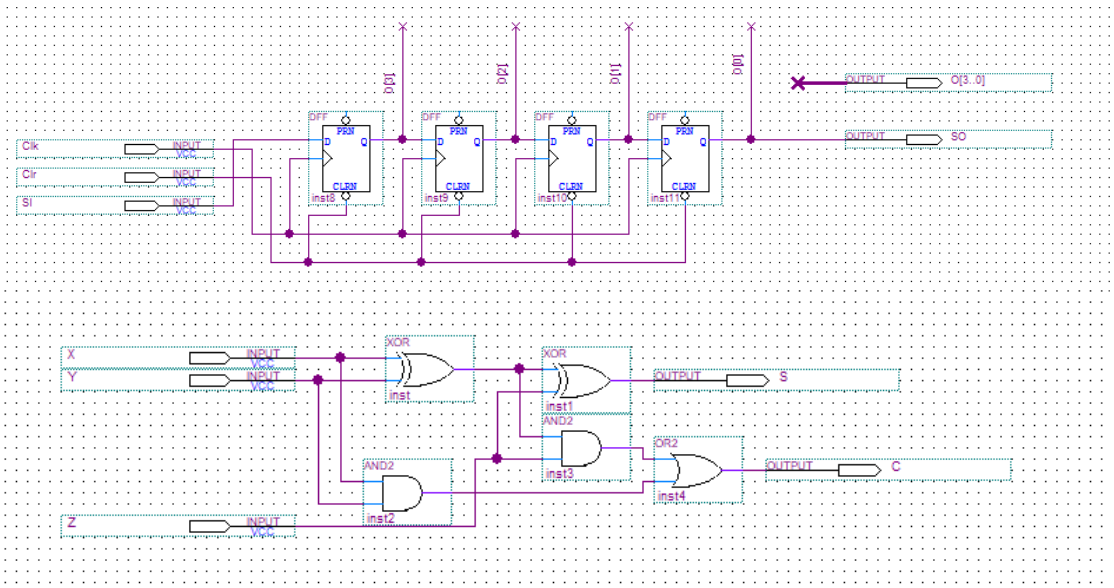
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity SRG_4 is
    port(SI,Clk,Clr:in std_logic;
         S0: out std_logic;
         S1:out std_logic_vector(3 downto 0));
end SRG_4;

architecture behavior of SRG_4 is
signal s:std_logic_vector(3 downto 0);
component Dffx
port(D,Clk,Clr:in std_logic;
    Q:out std_logic);
end component dffx;

begin
    S0<=s(0);
    d1:Dffx port map(SI,Clk,Clr,s(3));
    d2:Dffx port map(s(3),Clk,Clr,s(2));
    d3:Dffx port map(s(2),Clk,Clr,s(1));
    d4:Dffx port map(s(1),Clk,Clr,s(0));
    S1<=s;
end behavior;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity FullAdder is
port(x,y,z:in std_logic;
     s,c: out std_logic);
end FullAdder;

architecture behavior of FullAdder is
    begin
        s<= x xor y xor z;
        c<= (x and y) or (z and(x xor y));
end behavior;
```

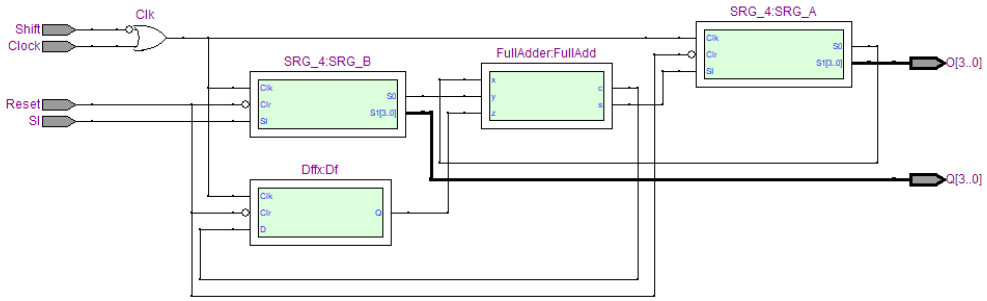
```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity Dffx is
    port(D,Clk:in std_logic;
    Clr:in std_logic;
    Q:out std_logic);
end Dffx;

architecture behavior of Dffx is
    begin
        process(D,Clk,Clr)
        begin
            if (Clk'event and Clk='1')then
                Q <= D;
            end if;
            if (Clr = '1')then
                Q <= '0';
            end if;
        end process;
end behavior;
```

原理图如图:





RTL 如图



b)编译、调试过程

编译通过，有 5 个警告;

c)结果分析及结论

为了不出现未定义的情况，先进行一次重置，Reset 在一开始设 0，将所有的触发器触发成复位。然后连续 12 个时钟信号周期，同时 Shift 始终输入 1，保证时钟信号有效。

在前 4 个周期的上升沿，依次从低到高输入 4 位二进制数，一位每周期存入 B 寄存器中;

在第 5 到第 8 个周期的上升沿，B 寄存器的 4 位二进制数从低位到高位，一位每周期，通过全加器与'0'相加后，原值存入到 A 寄存器中，同时从低到高输入新的 4 位二进制数，一位每周期存入 B 寄存器中;

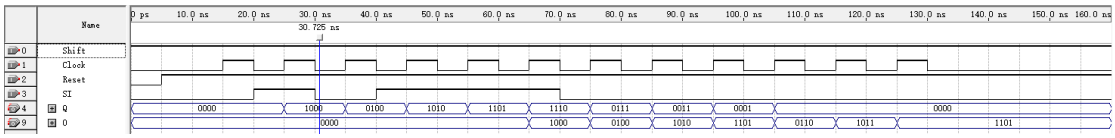
在第 9 到第 12 个周期的上升沿，每一个周期将 B 寄存器的一位二进制数输出，与 A 寄存器的一位二进制数以及进位结果（第一次为 0）相加，将不包括进位的结果重新存入 A 寄存器中，进位结果在下个周期与 A，B 寄存器输出的二进制数相加。第 12 个周期的结果就是两个 4 位二进制数除去最高位进位的相加结果。

为了方便调试和观察结果，我们对两个寄存器的 4 个触发器保存的值进行输出，输出端口为 O【3..0】和 Q【3..0】

O【3..0】和 Q【3..0】的结果应与上述原理逻辑一致。

<u>波形仿真</u>

a)波形仿真过程（详见实验步骤）
b)波形仿真波形图



c)结果分析及结论
Shift 始终设置为 1，保证时钟信号有效
一开始 Reset 设置为 0，对整个加法器进行重置。
在时钟信号的第一个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0000”，O 不变，为“0000”;
在时钟信号的第二个周期，SI 输入为 1，B 寄存器在上升沿保存该值，Q 结果为“1000”，O 不变，为“0000”;
在时钟信号的第三个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0100”，O 不变，为“0000”;
在时钟信号的第四个周期，SI 输入为 1，B 寄存器在上升沿保存该值，Q 结果为“1010”，O 不变，为“0000”;

在时钟信号的第五个周期，SI 输入为 1，B 寄存器在上升沿保存该值，Q 结果为“1101”，B 的输出一位在 A 的输出一位与全加器相加后，保存到 O 的最高位，O 变为“0000”;
在时钟信号的第六个周期，SI 输入为 1，B 寄存器在上升沿保存该值，Q 结果为“1110”，B 的输出一位在 A 的输出一位与全加器相加后，保存到 O 的最高位，O 变为“1000”;
在时钟信号的第七个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0111”，B 的输出一位在 A 的输出一位与全加器相加后，保存到 O 的最高位，O 变为“0100”;
在时钟信号的第八个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0011”，B 的输出一位在 A 的输出一位与全加器相加后，保存到 O 的最高位，O 变为“1010”;

在时钟信号的第九个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0001”，B 的输出一位在与全加器相加后，保存到 O 的最高位，O 变为“1101”;
在时钟信号的第十个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0000”，B 的输出一位在与全加器相加后，保存到 O 的最高位，O 变为“0110”;
在时钟信号的第十一个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0000”，B 的输出一位在与全加器相加后，保存到 O 的最高位，O 变为“1011”;
在时钟信号的第十二个周期，SI 输入为 0，B 寄存器在上升沿保存该值，Q 结果为“0000”，B 的输出一位在与全加器相加后，保存到 O 的最高位，O 变为“1101”;

以上结果符合预期，正确

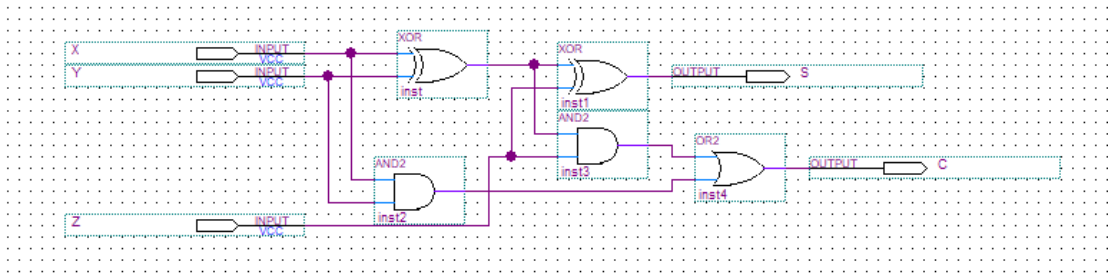
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ALU is
port(Input:in std_logic_vector(7 downto 0);
        Output:out std_logic_vector(1 downto 0));
end ALU;

architecture behaviour of ALU is
signal cmd : std_logic_vector(3 downto 0);
signal R1,R2,R3 : std_logic_vector(1 downto 0);
begin
    cmd <= Input(7 downto 4);
    R1 <= Input(3 downto 2);
    R2 <= Input(1 downto 0);

    R3 <= (R1+R2) when cmd = "1001" else
          (R1-R2) when cmd = "0110" else
          (R1 or R2) when cmd = "1011" else
          (not R1) when cmd = "0101" else
          (R1(0)&R1(1)) when (cmd = "1010" and R2 = "00") else
          (R1(0)&R1(1)) when (cmd = "1010" and R2 = "11") else
          "ZZ";

    Output <= R3;
end behaviour;
```

d．用 VHDL 语言和逻辑图设计一个全加器

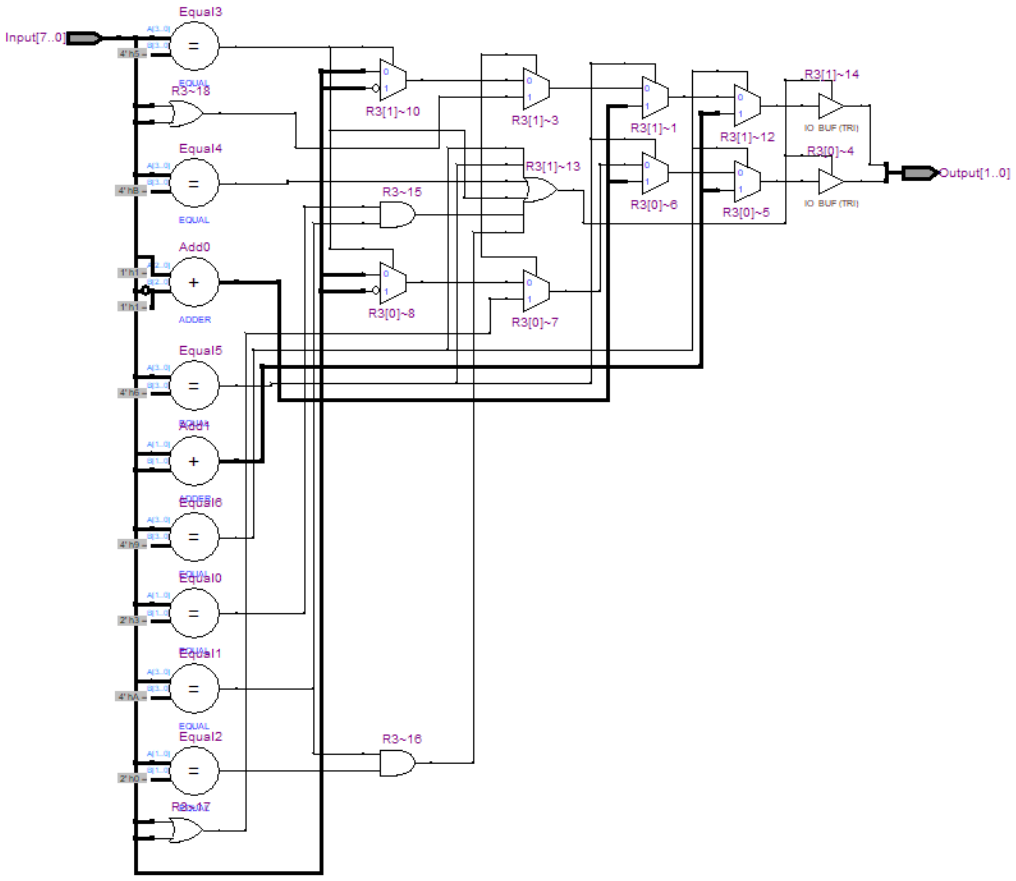
编译过程

a）源代码如图（VHDL 设计）

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ALU is
port(Input:in std_logic_vector(7 downto 0);
        Output:out std_logic_vector(1 downto 0));
end ALU;

architecture behaviour of ALU is
signal cmd : std_logic_vector(3 downto 0);
signal R1,R2,R3 : std_logic_vector(1 downto 0);
begin
    cmd <= Input(7 downto 4);
    R1 <= Input(3 downto 2);
    R2 <= Input(1 downto 0);

    R3 <= (R1+R2) when cmd = "1001" else
          (R1-R2) when cmd = "0110" else
          (R1 or R2) when cmd = "1011" else
          (not R1) when cmd = "0101" else
          (R1(0)&R1(1)) when (cmd = "1010" and R2 = "00") else
          (R1(0)&R1(1)) when (cmd = "1010" and R2 = "11") else
          "ZZ";

    Output <= R3;
end behaviour;
```



RTL 如图



b)编译、调试过程

编译通过，有 4 个警告;

c)结果分析及结论

当前四位为“1001”时，将后四位中的前两位与后两位相加，保存到 R1 中，输出;

当前四位为“0110”时，将后四位中的前两位与后两位相减，保存到 R1 中，输出;

当前四位为“1011”时，将后四位中的前两位与后两位做或运算，保存到 R1 中，输出;

当前四位为“0101”时，将后四位中的前两位做非运算，保存到 R1 中，输出;

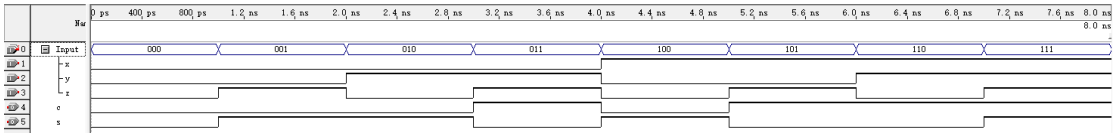
当前四位为“1010”，最后两位是“00”时，将后四位中的前两位做循环右移计算，保存到 R1 中，输出;

当前四位为“1001”时，最后两位是“11”时，将后四位中的前两位做循环左移计算，保存到 R1 中，输出;

波形仿真

a)波形仿真过程（详见实验步骤）

b)波形仿真波形图



c)结果分析及结论

当前四位为“1001”时，将后四位中的前两位与后两位相加，保存到 R1 中，输出；

当前四位为“0110”时，将后四位中的前两位与后两位相减，保存到 R1 中，输出；

当前四位为“1011”时，将后四位中的前两位与后两位做或运算，保存到 R1 中，输出；

当前四位为“0101”时，将后四位中的前两位做非运算，保存到 R1 中，输出；

当前四位为“1010”，最后两位是“00”时，将后四位中的前两位做循环右移计算，保存到 R1 中，输出；

当前四位为“1001”时，最后两位是“11”时，将后四位中的前两位做循环左移计算，保存到 R1 中，输出；

仿真中结果均符合预期，正确

五、实验结论

本次实验内容多，难度较大，完成这次实验是一次很大的挑战。在做预习报告之前，对实验内容的很多概念都不是很了解，比如锁存器，触发器，寄存器。在做实验之前先认真的看了书本，也用 quarters 自己实现了一遍锁存器和触发器，让我对于书本知识和实验内容有了一个深刻的认识。本次实验内容很多要用到 VHDL 语音，经常查资料和问同学，再加上自己的尝试，最后也掌握了不少 VHDL 的语法。本次实验难度虽大，但也收获很多。