浙江大学

专业：xxxxxxxxxxxxxxxxxx

姓名：xxxxxxxxxxxxxxxxxx

学号：xxxxxxxxxxxxxxxxxx

日期：xxxxxxxxxxxxxxxxxx

地点：xxxxxxxxxxxxxxxxxx

实验报告

课程名称：电路与电子技术实验 I　　指导老师：xxxxxx　　成绩：___分

实验名称：四位串行二进制加法器　　实验类型：个人实验

姓名：xxxxxx　　学号：xxxxxxxxxxx　　同组学生姓名：无

目录：

电路与电子技术实验报告

实验9—四位串行二进制加法器

一、实验目的和要求

实验目的

1. 熟悉Quartus II软件的使用；
2. 掌握逻辑功能的VHDL语言描述和原理图描述的方法；
3. 进一步掌握四位串行二进制加法器的设计方法；
4. 掌握用仿真波形验证电路功能的方法。

实验要求

1. 请不要带食物进入实验室，更不允许在实验室用餐；
2. 请勿大声喧哗，不要随意走动，不要私自更换实验设备；
3. 请听从实验指导老师的安排，独立完成实验；
4. 实验完毕请关闭电源，万用电表用完后关闭电源归还，并摆放整齐；整理实验桌面，保持实验室整洁；
5. 请注意用电安全，包括人身安全和设备安全；
6. 文明实验。

二、实验内容和原理

实验内容

1. 用原理图方式描述4位串行全加器的功能；
2. 用VHDL语言描述1位二进制全加器的功能；
3. 通过波形仿真验证4位全加器的功能。

实验原理

本实验为软件实验，主要原理为**硬件描述语言**VHDL。

对于**4位串行进位二进制全加器**：

- 4位串行进位二进制全加器以1位全加器的设计为基础，将四个1位二进制全加器串接即可构成四位二进制全加器； 顶层采用原理图描述，底层采用VHDL语言描述，充分发挥原理图描述的直观性和HDL语言的灵活性。

对于**1位二进制全加器**：

其真值表为：

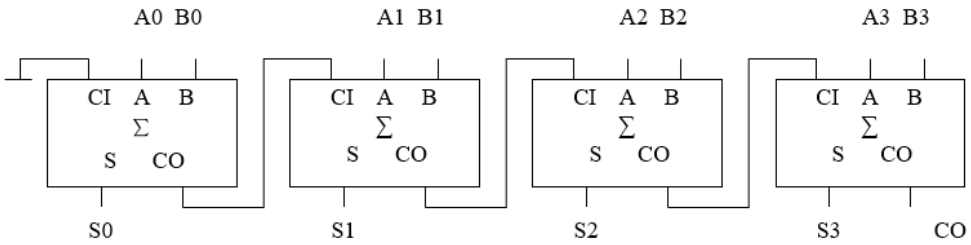
输入	**输出**
$A \quad B \quad C_i$	$S \quad C_0$
$0 \quad 0 \quad 0$	$0 \quad 0$
$0 \quad 0 \quad 1$	$1 \quad 0$
$0 \quad 1 \quad 0$	$1 \quad 0$
$0 \quad 1 \quad 1$	$0 \quad 1$
$1 \quad 0 \quad 0$	$1 \quad 0$
$1 \quad 0 \quad 1$	$0 \quad 1$
$1 \quad 1 \quad 0$	$0 \quad 1$
$1 \quad 1 \quad 1$	$1 \quad 1$

总结出函数式如下：

$$S = \overline{A}\,\overline{B}\,C_i + \overline{A}\,B\,\overline{C}_i + A\,\overline{B}\,\overline{C}_i + A\,B\,C_i = A \oplus^* B \oplus C$$

$$C_0 = \overline{A}\,B\,C_i + A\,\overline{B}\,C_i + A\,B\,\overline{C}_i + A\,B\,C_i = A\,B + B\,C_i + A\,C_i$$

有了1位二进制全加器以后，按照下图连线，即可得到4位串行进位二进制加法器。



4位串行进位二进制全加器连线图

上图中每一个方块都代表一个1位二进制全加器。

*：⊕代表异或（NAND）。

三、主要仪器设备

计算机，Windows 7系统，Quartus Prime 17.1软件；Altera制，FPGA类产品，编号MAX 10 10M50，10M50DAF484C7G开发板。

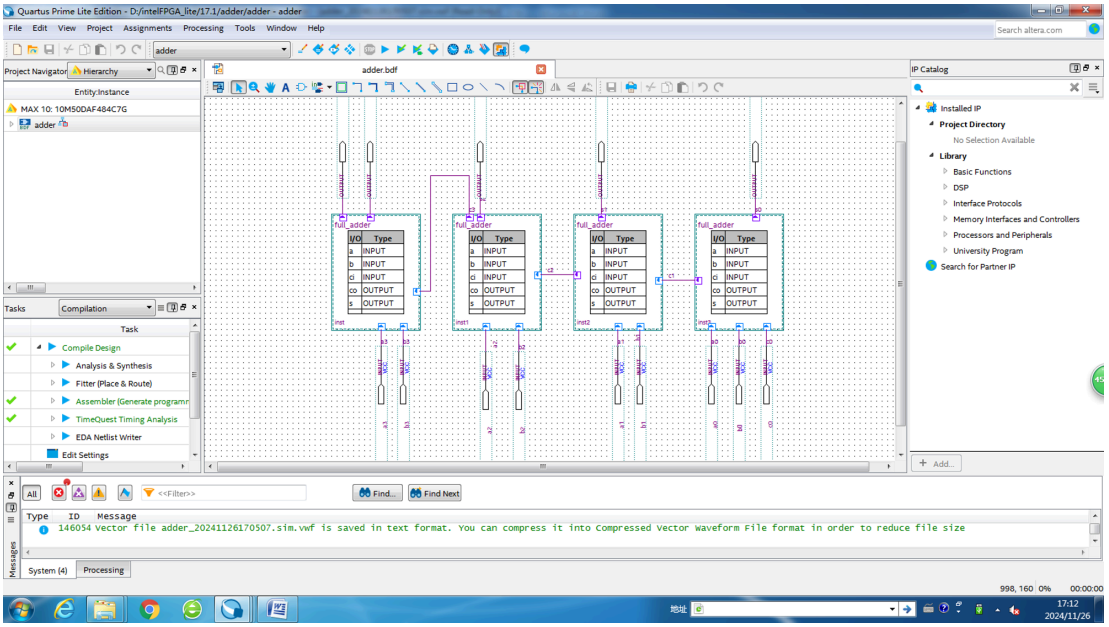
四、操作方法和实验步骤

首先实现上文的连线图接线，然后用VHDL实现1位二进制全加器：

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

--  Entity Declaration

ENTITY full_adder IS

  -- {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!

  PORT (
    a : IN STD_LOGIC;
    b : IN STD_LOGIC;
    ci : IN STD_LOGIC;
```

```
    co : OUT STD_LOGIC;
    s : OUT STD_LOGIC
  );

  -- {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!

END full_adder;

--  Architecture Body

ARCHITECTURE full_adder_architecture OF full_adder IS
BEGIN
  s <= a XOR b XOR ci;
  co <= (a AND b) OR (b AND ci) OR (ci AND a);
END full_adder_architecture;
```

最终的效果图如下：



4位串行进位二进制全加器效果图

连接完毕后进行仿真测试，测试结果请参见实验结果与分析部分。

最后进行引脚约束，上板测试，观察结果是否正确。（可选）

最后对上面实现4位二进制全加器的基本步骤进行总结：

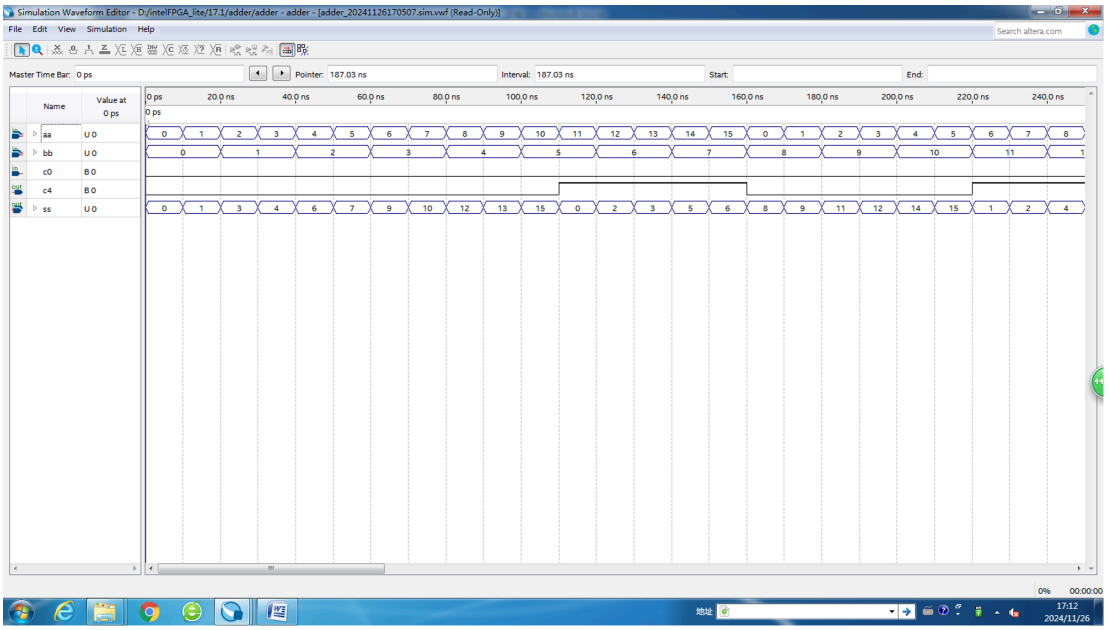
1. 创建工程；
2. 依照原理图连接4位串行进位二进制全加器的电路；
3. 使用VHDL实现1位二进制全加器；
4. 仿真验证；
5. 引脚约束，上板测试。（可选）

五、实验数据记录和处理

本实验没有实验数据，无需进行处理。

六、实验结果与分析

本实验实现了4位串行进位二进制全加器，并验证了其正确性，仿真波形简图及验证如下：



4位串行进位二进制全加器仿真波形简图

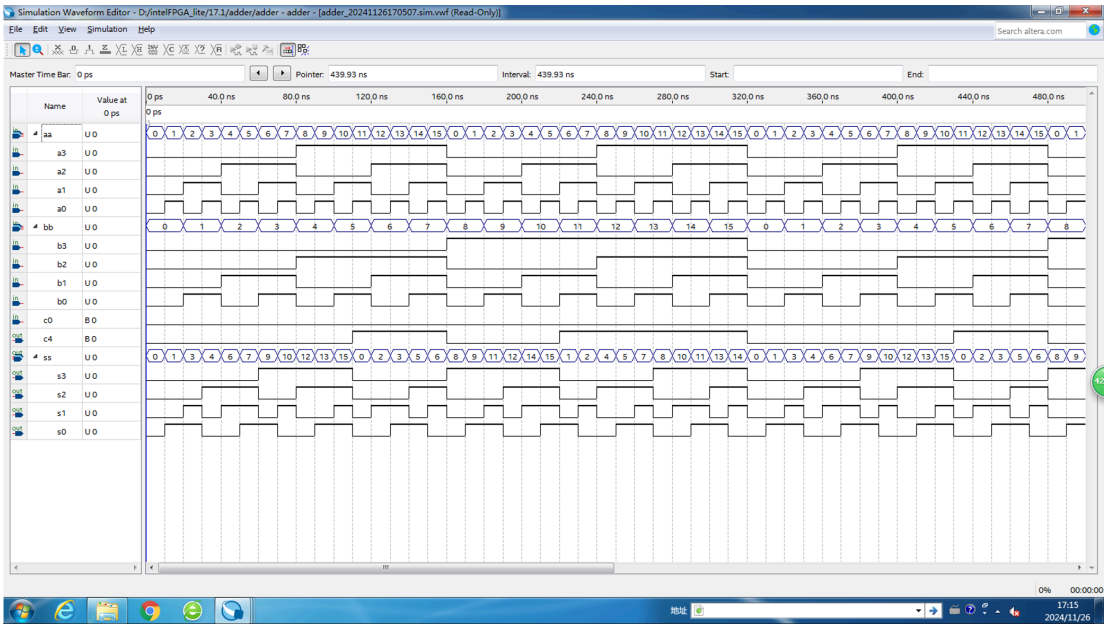
上图中的`aa`、`bb`分别为两个4位加数，`c0`为最低位的carry-in，在加法器中应当置0。`ss`为4位的相加结果输出，`c4`是最高位的carry-out，如其为`1`，则表示我们的计算发生了overflow（4位加法，结果`ss`的范围在**不大于15**的自然数，若超出该范围，如计算`8 + 9 (=17, > 15)`即发生overflow。发生overflow时，`ss`的结果不再具有意义）。

上面的波形图很清晰的显示了我们的加法器的计算结果：

- $0+0=0$　　$1+0=1$　　$2+1=3$　　$3+1=4$
- $4+2=6$　　$5+2=7$　　$6+3=9$　　$7+3=10$

- $8+4=12$　　$9+4=13$　　$10+5=15$　　$11+5=$overflow
- $12+6=$overflow　　$13+6=$overflow
- $14+7=$overflow　　$15+7=$overflow
- $0+8=8$　　$1+8=9$　　$2+9=11$　　$3+9=12$
- ……（更多请自行看图验证，限于篇幅，列举到此）

下面给出更加详细的仿真波形图：



4位串行进位二进制全加器仿真波形全图

容易知道，我们的计算结果均正确，4位串行进位二进制全加器实现正确。

七、讨论、心得

讨论

本次实验采用了**自顶向下**的方法，遵循着**模块化**的思路，用原理图与硬件描述语言VHDL相结合的方式实现了4位串行进位二进制全加器。

但是这种方法比较复杂，尤其是连线部分非常繁琐，对于更大工程，本方法具有局限性。

VHDL本身可以进行**模块化**描述。因此连线部分完全可以省去，使用全VHDL描述的方法实现4位串行进位二进制全加器如下：

首先实现1位二进制全加器：

```
-- 1位全加器模块
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY FullAdder IS
    PORT (
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        Cin : IN STD_LOGIC;
        Sum : OUT STD_LOGIC;
        Cout : OUT STD_LOGIC
    );
END FullAdder;

ARCHITECTURE Behavioral OF FullAdder IS
BEGIN
    PROCESS (A, B, Cin)
    BEGIN
      Sum <= A XOR B XOR Cin;
      Cout <= (A AND B) OR (B AND Cin) OR (Cin AND A);
    END PROCESS;
END Behavioral;
```

然后建立4位串行进位二进制全加器顶层模块，并在其中实例化前面已经建好的1位二进制全加器：

```
-- 4位串行进位二进制全加器顶层模块
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
```

```
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY SerialAdder4Bit IS
    PORT (
        A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Cin : IN STD_LOGIC;
        Sum : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        Cout : OUT STD_LOGIC
    );
END SerialAdder4Bit;

ARCHITECTURE Behavioral OF SerialAdder4Bit IS
    SIGNAL carry : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
    -- 实例化1位全加器
    FA0 : ENTITY work.FullAdder PORT MAP (
        A => A(0),
        B => B(0),
        Cin => Cin,
        Sum => Sum(0),
        Cout => carry(0)
        );

    FA1 : ENTITY work.FullAdder PORT MAP (
        A => A(1),
        B => B(1),
        Cin => carry(0),
        Sum => Sum(1),
        Cout => carry(1)
        );

    FA2 : ENTITY work.FullAdder PORT MAP (
        A => A(2),
        B => B(2),
        Cin => carry(1),
```

```
            Sum => Sum(2),
            Cout => carry(2)
            );

        FA3 : ENTITY work.FullAdder PORT MAP (
            A => A(3),
            B => B(3),
            Cin => carry(2),
            Sum => Sum(3),
            Cout => carry(3)
            );

        Cout <= carry(3);
    END Behavioral;
```

这样，我们就实现了4位串行进位二进制全加器。效果与上文中实现的相同，而且代码更加清晰简洁，易于维护，不需要考虑复杂的连线过程。

除了自顶向下的方法之外，由于4位二进制串行进位加法器的简单性，我们可以考虑直接实现（列真值表，分析逻辑函数），但分析过程十分繁琐，这里限于篇幅从略。必须指出的是，只有对于最基本，最简单的模块，我们才考虑直接实现。**自顶向下**仍然是最重要且最普适的实现方法。

心得

笔者已经具有熟练的Verilog基础，再进行本次实验的时候自然感觉十分简单。通过本次实验，笔者对比了Verilog和VHDL两种硬件描述语言的区别，接触并使用了新的软件工具Quartus Prime 17.1（区别于笔者目前在用的Vivado 2024.1）。

VHDL和Verilog对于硬件的描述思路是相同的，但VHDL稍显繁琐。对于Quartus Prime 17.1，笔者的感受是：总体上并不好用，有很多很复杂但实际上并无意义的操作——这对于初学者来讲并不容易上手并且非常容易犯错。工具的更新迭代是十分迅速的，希望我们的实验室能够紧跟时代，绽放活力！