华南理工大学

South China University of Technology

VHDL Experimental Report

Curriculum:	Digital System Design (Experiment)
Student Name:	Liu Xingyan/刘兴琰
Student No.:	202264690069
Student Major:	Artificial Intelligence/人工智能
Examiner:	Wu Hancong/吴瀚聪

未来技术学院
FUTURE TECH

Name	刘兴琰	**Class**	22人工智能1班
Date	11月20日	**Score**	
Examiner	吴瀚聪		

1 Experiment Procedure

1.1 Experiment Purpose

- Learn the basics of VHDL programming.
- Practice writing, compiling, and debugging VHDL code in Quartus II.
- Test and verify designs through simulations in Quartus II and ModelSim.

1.2 Experiment Tools

- **Quartus II 13.1 (64-bit)**
- **ModelSim SE-64 10.4**

1.3 Experiment Tasks

1. Follow the instructions to create a basic 1-bit adder circuit.
2. Use the 1-bit adder as a reusable component in a package.
3. Combine multiple 1-bit adders to create a 4-bit adder.
4. Create a symbol file for the 4-bit adder using Quartus II.
5. Test the 4-bit adder by running a waveform simulation.
6. Use ModelSim to verify the design and document the outputs.

1.4 Detailed Procedure

1.4.1 Design a 1-bit Adder

The hardware encapsulation of a 1-bit full adder is shown below:

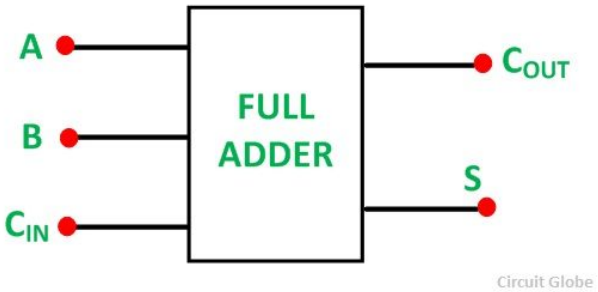


图 1: 1-bit Full Adder

These equations define the 1-bit full adder for implementation in hardware.

$$S = A \oplus B \oplus \text{Cin}$$

$$\text{Cout} = (A \wedge B) \vee (A \wedge \text{Cin}) \vee (B \wedge \text{Cin})$$

Expanded forms:

$$S = (A \vee B \vee \text{Cin}) \wedge \neg((A \wedge B) \vee (A \wedge \text{Cin}) \vee (B \wedge \text{Cin}))$$

$$\text{Cout} = (A \wedge B) \vee (\text{Cin} \wedge (A \vee B))$$

Write the following VHDL code after creating a Quartus II project:

Listing 1: VHDL Code for 1-bit Full Adder

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY fadd IS
PORT (a, b, ci: IN std_logic;
      co, sum: OUT std_logic
     );
```

```
END fadd;

architecture dataflow of fadd is
begin
    co <= (a and b) or (b and ci) or (a and ci);
    sum <= a xor b xor ci;
end dataflow;
```

Compile the code by pressing the ”Start Compilation” button to ensure functionality.

1.4.2 Instantiate the 1-bit Adder as a Component

The 1-bit adder is encapsulated as a reusable component within a package. The package `components` declares the component `fadd`, which defines the input ports `a`, `b`, and `ci`, and the output ports `co` and `sum`. This allows the 1-bit adder to be instantiated multiple times in hierarchical designs. The VHDL code for the component declaration is shown below:

Listing 2: VHDL Code for Component Declaration

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

package components is
    component fadd is
        port (
            a, b, ci: IN std_logic;
            co, sum: OUT std_logic
        );
    end component;
end components;
```

1.4.3 Design a 4-bit Adder Using the 1-bit Adder Component

Using the 1-bit adder component declared in the package, a 4-bit adder is designed by instantiating four instances of the 1-bit adder. The entity `fadd4` defines the inputs `a` and `b` as 4-bit vectors, the carry-in (`ci`), and the outputs `sum` (a 4-bit vector) and `co` (carry-out). The architecture `stru` of the 4-bit adder connects the four 1-bit adders sequentially, with the carry-out signal of each stage

passed to the carry-in signal of the next stage via an internal signal `ci_ns`. The complete VHDL code for the 4-bit adder is shown below:

Listing 3: VHDL Code for 4-bit Adder

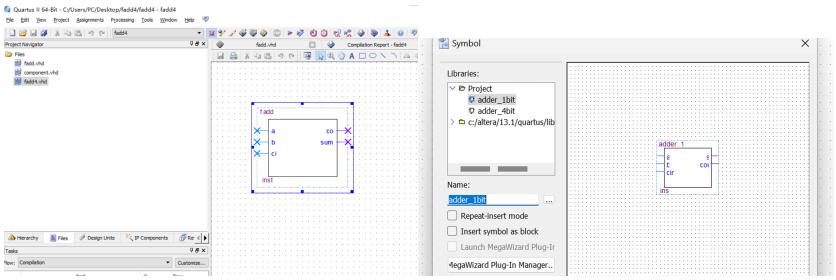
```
USE IEEE.std_logic_1164.ALL;
USE work.components.all;

entity fadd4 is
port (
    a, b: in std_logic_vector(3 downto 0);
    ci: in std_logic;
    co: out std_logic;
    sum: out std_logic_vector(3 downto 0)
);
end fadd4;

ARCHITECTURE stru OF fadd4 IS
SIGNAL ci_ns: std_logic_vector(2 downto 0);
BEGIN
    U0: fadd PORT MAP (
        a => a(0),
        b => b(0),
        ci => ci,
        co => ci_ns(0),
        sum => sum(0)
    );
    U1: fadd PORT MAP (
        a => a(1),
        b => b(1),
        ci => ci_ns(0),
        co => ci_ns(1),
        sum => sum(1)
    );
    U2: fadd PORT MAP (
        a => a(2),
        b => b(2),
        ci => ci_ns(1),
        co => ci_ns(2),
        sum => sum(2)
    );
    U3: fadd PORT MAP (
        a => a(3),
        b => b(3),
        ci => ci_ns(2),
        co => co,
        sum => sum(3)
    );
END stru;
```

1.4.4 Create a Symbol File of the 4-bit Adder and Check It

The symbols for the 4-bit adder (`adder_4bit`) and 1-bit adder (`adder_1bit`) were generated using Quartus II. These symbols allow for graphical representation and integration of the respective modules in schematic designs. The updated symbols are shown below:



(a) Symbol of `adder_4bit` (b) Symbol of `adder_1bit`

图 2: Symbols of `adder_4bit` and `adder_1bit`

1.4.5 Perform Waveform Simulation and Record Results

To validate the functionality of the 4-bit adder, a testbench was created in ModelSim to perform waveform simulation. The architecture `fadd4_arch` includes the declaration of internal signals for the inputs (`a`, `b`, `ci`), the outputs (`sum`, `co`), and connections between the master ports and the adder component.

The testbench initializes the inputs as follows:

- `a = "0000"`
- `b = "0000"`
- `ci = '0'`

The key testbench code snippet is shown below:

Listing 4: Testbench Code for 4-bit Adder Simulation

```
ARCHITECTURE fadd4_arch OF fadd4_vhd_tst IS
    -- Constants
    SIGNAL a : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
    SIGNAL b : STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL ci : STD_LOGIC;
    SIGNAL co : STD_LOGIC;
    SIGNAL sum : STD_LOGIC_VECTOR(3 DOWNTO 0);

BEGIN
    U1: fadd4 PORT MAP (
        a => a,
        b => b,
        ci => ci,
        co => co,
        sum => sum
    );

    init : PROCESS
    BEGIN
        a <= "0000";
        b <= "0000";
        ci <= '0';
        FOR k IN 0 TO 1 LOOP
            FOR j IN 0 TO 3 LOOP
                FOR i IN 0 TO 3 LOOP
                    WAIT FOR 10 ns;
                    a <= a + 1;
                END LOOP;
                b <= b + 1;
            END LOOP;
            ci <= NOT ci;
        END LOOP;
        WAIT;
    END PROCESS init;
END fadd4_arch;
```

A nested loop iterates through all possible combinations of the 4-bit inputs `a` and `b` with the carry-in signal toggled for each complete iteration. At each step:

- The inputs `a` and `b` are incremented sequentially.
- A delay of `10 ns` is added between transitions.
- The carry-in signal `ci` is toggled after each full sequence.

The connections to the instantiated `fadd4` component ensure the correct mapping of inputs and outputs for simulation. After running the testbench, the waveform results are generated in ModelSim, verifying the correct behavior of the 4-bit adder.

2 Experimental Record

1. **Design Code of the Adders:**

   - (a) `adder_1bit`

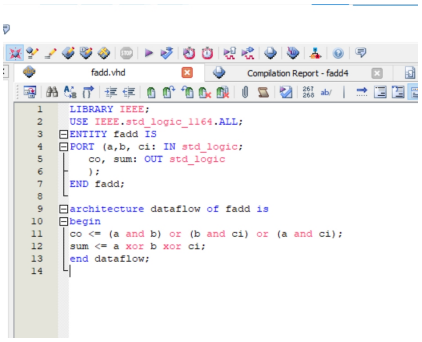


图 3: Design Code of `adder_1bit`

   - (b) `adder_4bit`

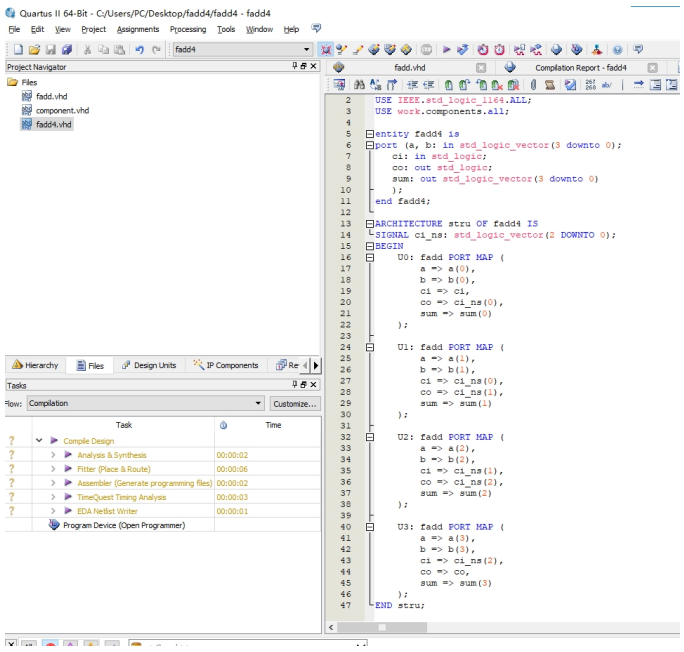


图 4: Design Code of `adder_4bit`

3 Analysis and Discussion

The simulation results confirm that the 4-bit adder functions correctly. It adds the inputs from ports `a` and `b` and outputs the sum through port `s`. The carry-out signal at port `cout` correctly indicates overflow.

Minor glitches caused by signal delays or hazards were observed but did not impact the adder's functionality. The design meets all operational requirements under simulation.

4 Other Attachments

- **Waveform of `adder_4bit`:**
  - (a) Waveform 1

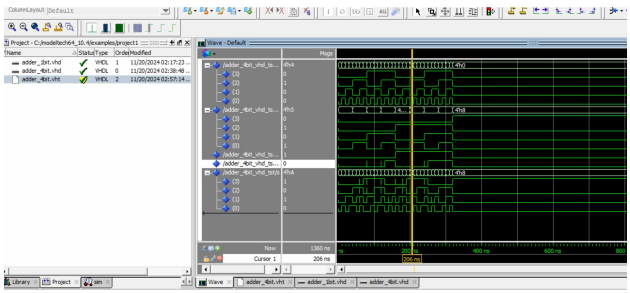


图 5: Waveform 1 of `adder_4bit`

  - (b) Waveform 2

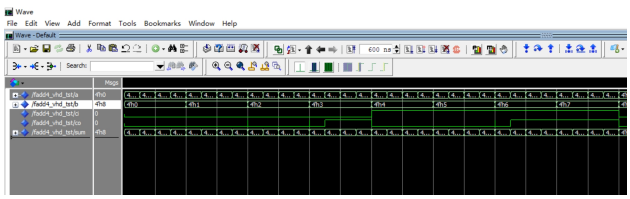


图 6: Waveform 2 of `adder_4bit`