华南理工大学 广州国际校区
GUANGZHOU INTERNATIONAL CAMPUS, SCUT

Experimental Report

Curriculum:	Digital System Design(Experiment)
Student Name:	Wang Shiyi/王实依
Student No.:	202230055267
Student Major:	Artificial Intelligence/人工智能

未来技术学院
FUTURE TECH

Experiment 1: Digital system design with VHDL

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

I. Experiment Procedure

1. Experiment Purpose

- Familiarize with the basic structure of VHDL programming.
- Learn how to compile VHDL code in quartus II and define components.
- Learn how to test and verify the function of your design in Quartus II.

2. Experiment Tools

- Quartus II 13.1(64-bit)
- Modelsim SE-64 10.4

3. Experiment Tasks

1. Design a **1-bit adder** following the instructions on manual.
2. Instantiate the 1-bit adder as a **component** in the package.
3. Design a **4-bit adder** using the 1-bit adder component.
4. Create a **symbol file** of this 4-bit-adder and check it
5. Perform the **waveform simulation** Using Quartus II and Modelsim.
6. Run the simulation and record the results of Modelsim.

4. Detailed Procedure

4.1. Design a 1-bit adder

As is known to all, the hardware encapsulation of a 1-bit full adder should be like:

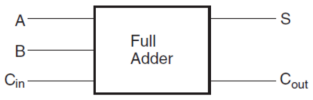


Fig 1: 1-bit full adder

And the logic structure of output ports should be:

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A\&B)|(A\&C_{in})|(B\&C_{in})$$

Therefore, we need to design a VHDL code according to the above design. Before writing the code, we need to create a Quartus II Project in order to compile the code.

For the specific project creation process, follow the wizard of the software. Some of the most important parameters are set as follows:

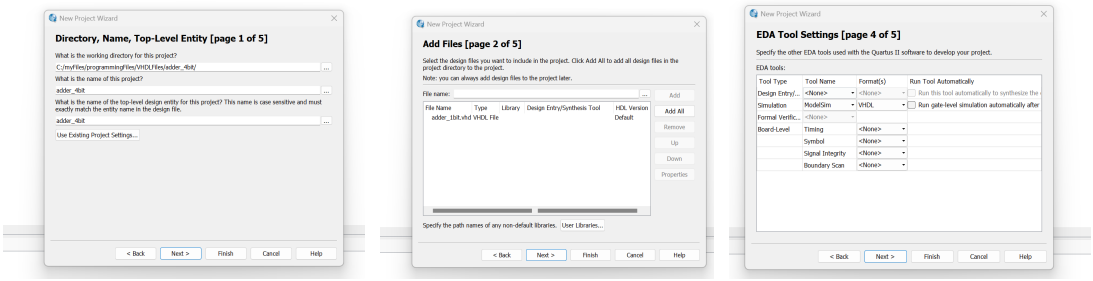


Fig 2: Important settings

Then, in order to test the function of the 1-bit adder, we set the *Top-level entity* as ”adder_1bit”:

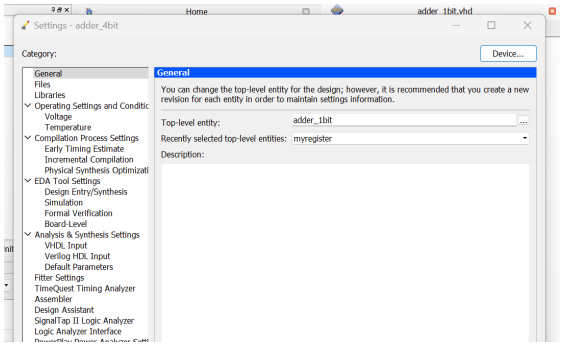


Fig 3: Top-level entity

Then design the VHDL code as:

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

entity adder_1bit is port(
    a,b,cin: in std_logic;
    s,cout: out std_logic
);
end entity adder_1bit;

architecture arch_adder_1bit of adder_1bit is
begin
    s <= a xor b xor cin;
    cout <= (a and b)or(a and cin)or(b and cin);
end architecture arch_adder_1bit;
```

Press ”Start Compilation” button to see whether the code works fine.

4.2. Instantiate the 1-bit adder as a component

Modify the settings, set the *Top-level entity* as ”adder_4bit”:

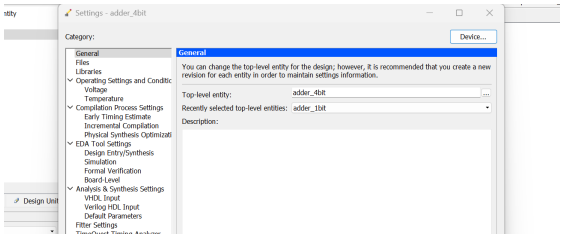


Fig 4: New Top-level entity

Then, create a new VHDL file and add it to the Project. In the new file, write the following code to create an EMPTY entity with a component named **adder_1bit**:

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

entity adder_4bit is port(
    a,b: in std_logic_vector(3 downto 0);
    cin: in std_logic;
    cout: out std_logic;
    s: out std_logic_vector(3 downto 0)
);
end entity adder_4bit;

architecture arch_adder_4bit of adder_4bit is
    component adder_1bit is port(
        a,b,cin: in std_logic;
        s,cout: out std_logic
    );
    end component adder_1bit;
    signal c:std_logic_vector(2 downto 0) := (others => '0');
begin
-- leave space to see the compilation result
end architecture arch_adder_4bit;
```

Compile the project to see if it works fine.

4.3. Design a 4-bit adder using the 1-bit adder component

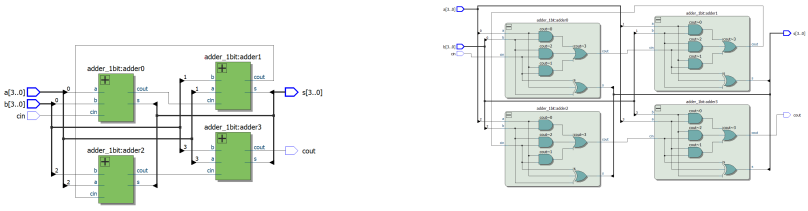
Complete the code above as follows:

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

entity adder_4bit is port(
    a,b: in std_logic_vector(3 downto 0);
    cin: in std_logic;
    cout: out std_logic;
    s: out std_logic_vector(3 downto 0)
);
end entity adder_4bit;

architecture arch_adder_4bit of adder_4bit is
    component adder_1bit is port(
        a,b,cin: in std_logic;
        s,cout: out std_logic
    );
```

```
    end component adder_1bit;
    signal c:std_logic_vector(2 downto 0) := (others => '0');
begin
    adder0: adder_1bit port map(a(0), b(0), cin, s(0),c(0));
    adder1: adder_1bit port map(a(1), b(1), c(0), s(1),c(1));
    adder2: adder_1bit port map(a(2), b(2), c(1), s(2),c(2));
    adder3: adder_1bit port map(a(3), b(3), c(2), s(3),cout);
end architecture arch_adder_4bit;
```

Start the compilation to see it works fine.

To examine the design, we can check the RTL Viewer:



(a) RTL View(Coarse) (b) RTL View(Detailed)

Fig 5: RTL Views

4.4. Create a symbol file of this 4-bit-adder and check it

Select ”File-Create/Update-Create Symbol File for Current File”:

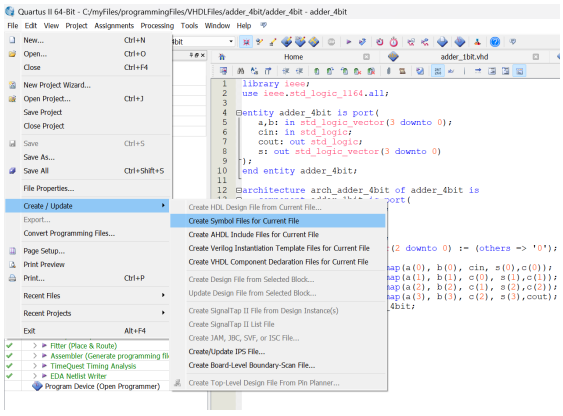


Fig 6: Create Symbol File

To check the ”novel” symbol file, we create a new ”Block Diagram/Schematic File”:

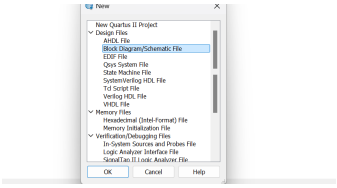
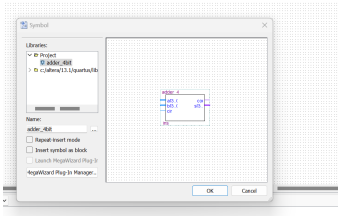
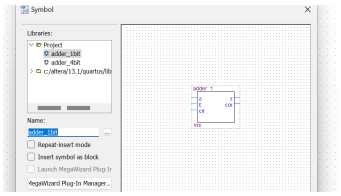


Fig 7: New ”Block Diagram/Schematic File”

In the new file, we tried to insert a new symbol we’ve just updated. In the same way, we can also update adder_1bit to the symbol library:



(a) Symbol of adder_4bit



(b) Symbol of adder_1bit

Fig 8: Updated Symbols

4.5. Perform the waveform simulation Using Quartus II and Modelsim, run the simulation and record the results of Modelsim

Modify the settings as:

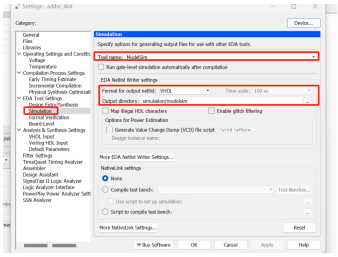


Fig 9: Simulation settings

Select ”Processing-Start-Start Test Bench Template Writer” to generate template file(.vht file).

Start Modelsim Program. Create a new library named ”work1”:

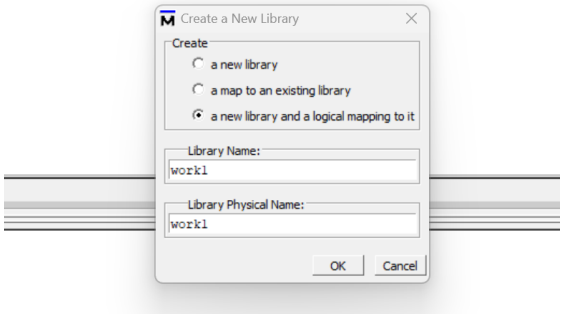


Fig 10: work1

Create a new project and add all the code files and the vht file to the project.

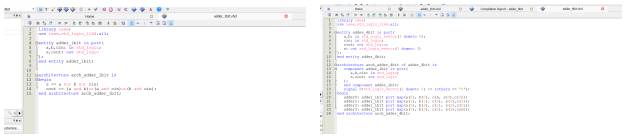
Modify the template file to run the test program:

```
-- omitted

init : PROCESS
    -- variable declarations
    BEGIN
        -- code that executes only once
        a <= "0000";
        b <= "0000";
        cin <= '0';
        for k in 0 to 1 loop
            for j in 0 to 3 loop
                for i in 0 to 3 loop
                    wait for 10ns;
                    a <= a+1;
                end loop;
                b <= b+1;
            end loop;
            cin <= not cin;
        end loop;
    WAIT;
END PROCESS init;

-- omitted
```

Then compile the whole project, run the simulation to see the result.

II. Experimental record

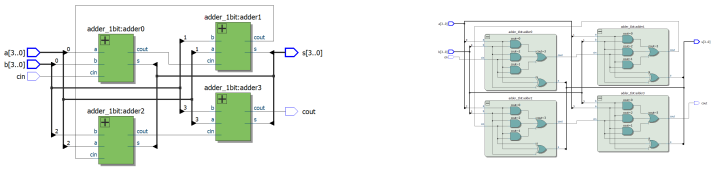
1. Design code of the adders



(a) adder_1bit (b) adder_4bit

Fig 11: Design code of the adders

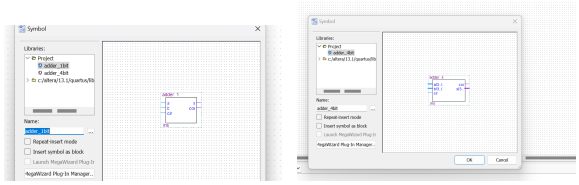
2. RTL View of the hardware



(a) RTL View 1 (b) RTL View 2

Fig 12: Design code of the adders

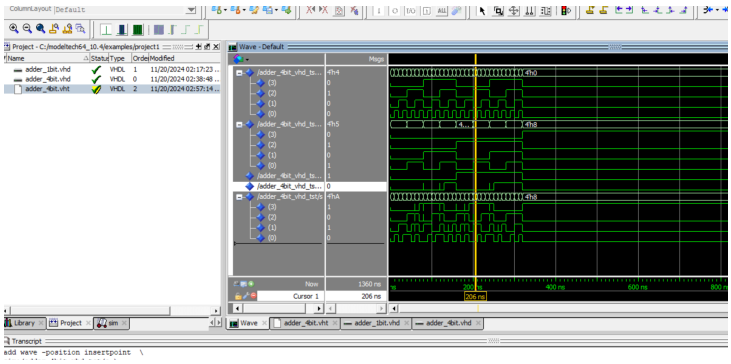
3. Symbols of adders



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

Fig 13: Symbols of adders

4. Waveform of adder_4bit



(a) Waveform 1

(b) Waveform 2

III. Analysis and discussion

According to the waveform, the hardware acted the function of an adder, adding the hex integer from the 2 input ports up, and output a hex integer from port ”s”. Simultaneously, the ”cout” port presents the carry from the addition correctly.

Due to **hazard, interruption, delay of signals**, the waveform shows some glitches. But it still works correctly overall.

IV. Other Attachments

Nothing else.