1 Chapter Outline

1. Introduction

In this chapter, Hardware Description Languages (HDLs) are introduced. The textbook will cover about both System Verilog and VHDL. These two languages are activated in similar principles, but have different syntax. Unlike well-known programming languages such as C or C++ language, these two languages are meant to describe the hardware and simulate it. The entire chapter will cover about describing materials in the previous chapters with System Verilog and VHDL.

Understanding about modules is the first step. Module is a description of what hardware produces. In modules, there are two types of modules, behavioral module, and structural module. Behavioral module describes what a module does, and structural module describes how it is composed with simple (lower level) modules.

After understanding the module, simulation and synthesis have to be understood. Simulation is a step for checking the design if the design affects the hardware to produce errors. Synthesis is changing a code to netlist.

2. Combinational Logic

This section reviews fundamental feature of combinational logic which is bitwise operator. Bitwise operator acts on single bit or multi-bit. Based on this fact, the combinational logic will be designed by HDLs, System Verilog and VHDL.

By using the knowledge obtained from the previous section, the fundamental element of combinational logic such as logic gates will be designed. Furthermore, the logic gates in combinational logic can be reduced by using conditional assignments and internal variables. For instance, multiplexer is described in few lines of System Verilog, and full adder is described with 5 logic gates from multiple logic gates which are more than 5.

As a result of precedence and numbers, the application of those two languages is various. Furthermore, it is able to describe X's and Z's, describing tristate buffer is possible. Moreover, these two languages support describing bit swizzling and delays.

3. Structural Modeling

The previous section was handling about behavioral modules. In this section, structural modules will be handled. Structural modeling is similar to building a structure with Lego blocks. To be specific, lower level of single bit combinational logics consists complicated multi-bit combinational logic. For instance, designing combinational logic such as 4:1 multiplexer can be held much easier by using structural modeling. Furthermore, by assigning lower 2:1 multiplexer and higher 2:1 multiplexer to the final 2:1 multiplexer, 4:1 multiplexer is designed.

To do the design mentioned above, designing 2:1 multiplexer has to be done before designing 4:1 multiplexer.

4. Sequential Logic

In this section, the sequential logic covered in the previous chapters will be designed by HDLs which are System Verilog and VHDL. To design the sequential logic, understanding the sequential logic is the most important part to start. If the understanding of sequential logic is sufficient, designing the sequential logic will be simple because the HDLs are describing the entire hardware's element and functionality.

The sequential logics handled in this section are registers, resettable registers, enabled registers, multiple registers, and latches. These sequential logics are described in both System Verilog and VHDL

5. More Combinational Logic

To design complex combinational logic much simpler, HDLs support blocking and nonblocking assignments by using always/process description. Blocking assignment is executed in the order where they appear in the code like a one code, and nonblocking assignment is executed concurrently. In always/process, there are some symbols which determine blocking assignment or nonblocking assignment. In case of System Verilog, = indicates blocking assignment and <= indicates nonblocking assignment, and in case of VHDL, := indicates blocking assignment and <= indicates nonblocking assignment. By using blocking and nonblocking assignment, full adder can be designed by using blocking assignment.

Using a case statement also can make the process of design much simpler. For instance, by using case statement including if statement in always/process statement, it can handle 'don't cares' case in the process of design.

Applicating these features above including blocking and nonblocking assignment, it is possible to design not only combinational logic but also sequential logic.

6. Finite State Machines

In this section, designing finite state machine with HDLs will be handled. As the actual FSM is divided into three parts which are the state register, the next state logic, and the output logic. Designing the FSM in HDLs will go through the three parts of the FSM.

In designing the FSM in both System Verilog and VHDL, blocking assignment and nonblocking assignment are used. By dividing the FSM into three parts, it is able to design much more complex FSM such as Moore FSM in HDLs.

7. Data Types

Data types are the elements which categorizes the variables when designing hardware in HDLs. Because of the different syntax in System Verilog and VHDL, there are different data types in both languages.

In case of System Verilog, System Verilog uses ‘logic’ data type. This data type is related to ‘reg’ because Verilog was previous version of System Verilog. ‘reg’ data type was making confusion with register so that ‘logic’ was appear to eliminate this confusion. Moreover, there are ‘always’ data type which enables to do blocking assignment, and ‘assign’ data type which enables to set the values of the variables.

In case of VHDL, VHDL uses library from ‘IEEE.STD_LOGIC_1164” because “STD_LOGIC” is not built in VHDL. Unlike System Verilog, VHDL has Boolean datatypes which are ‘true(1)’ and ‘false(0)’. Moreover, VHDL has ‘INTEGER’ data type in order to represent positive and negative numbers. Furthermore, VHDL has special data types including ‘out’ and ‘buffer’.

8. Parameterized Modules

Parameterized modules allow to use variable bit width. This reduces the amount of HDLs to type. To be specific, by setting the value of n, it is possible to design the hardware with any amount of bit of input and output. The only one thing to consider is not to make the amount of bit extremely large. Parameterized modules make the design much simpler.

9. Testbenches

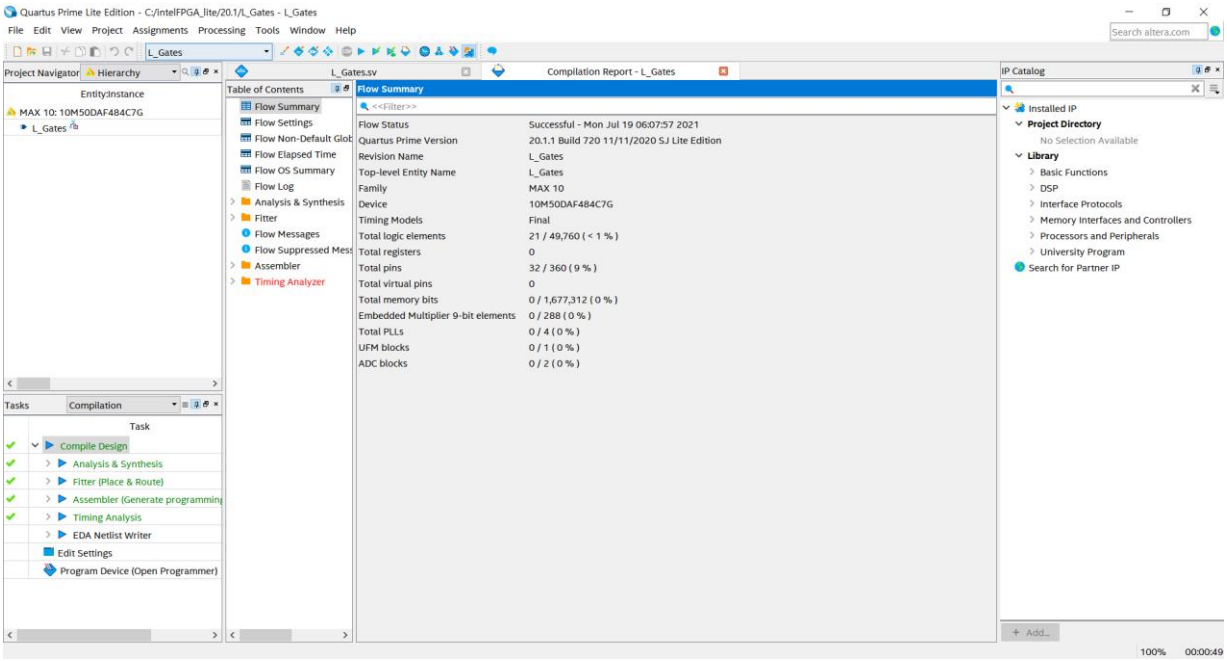
Testbench is a module which tests another module of HDLs. Unlike other modules, testbench is not synthesized. However, testbench is simulated in common of other modules. In the simulation, testbench checks test vectors, and then detects the error occurred. Moreover, by changing few lines of codes, testbench can compile more complex hardware design.

2 HDL Examples

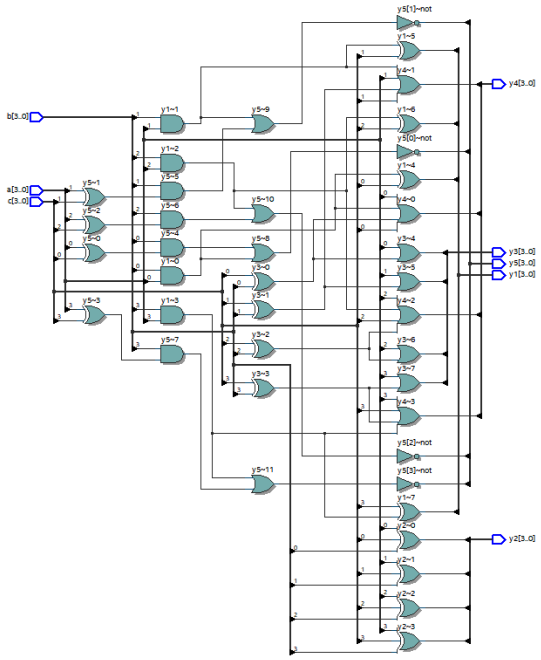
1. Example #1 (Example 4.3 variation)

```
1  module L_Gates(input logic[3:0] a, b, c,
2                 output logic[3:0] y1, y2,
3                                   y3, y4, y5);
4
5
6   assign y1 = (a & b) ^ c;
7   assign y2 = a ^ b ^ c;
8   assign y3 = a | (b ^ c) ;
9   assign y4 = (a & b) | (b ^ c) | (a | c);
10  assign y5 = ~((a & b) | (b & (a ^ c))) ;
11
12
13  endmodule
14
```

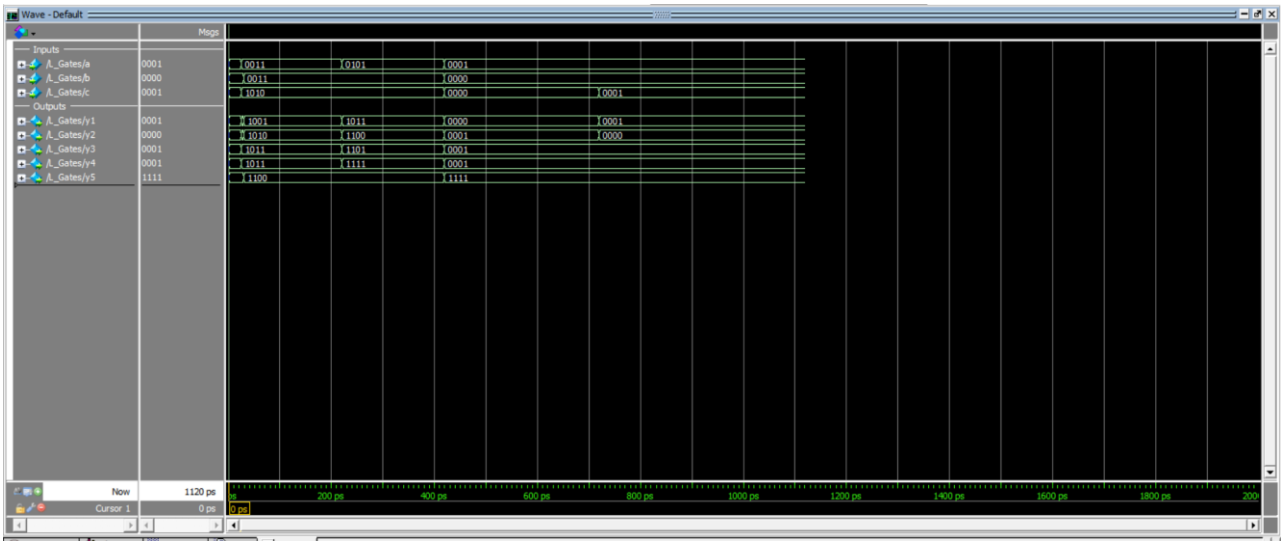
(Verilog Code)



(Synthesized image)



(RTL Viewer)

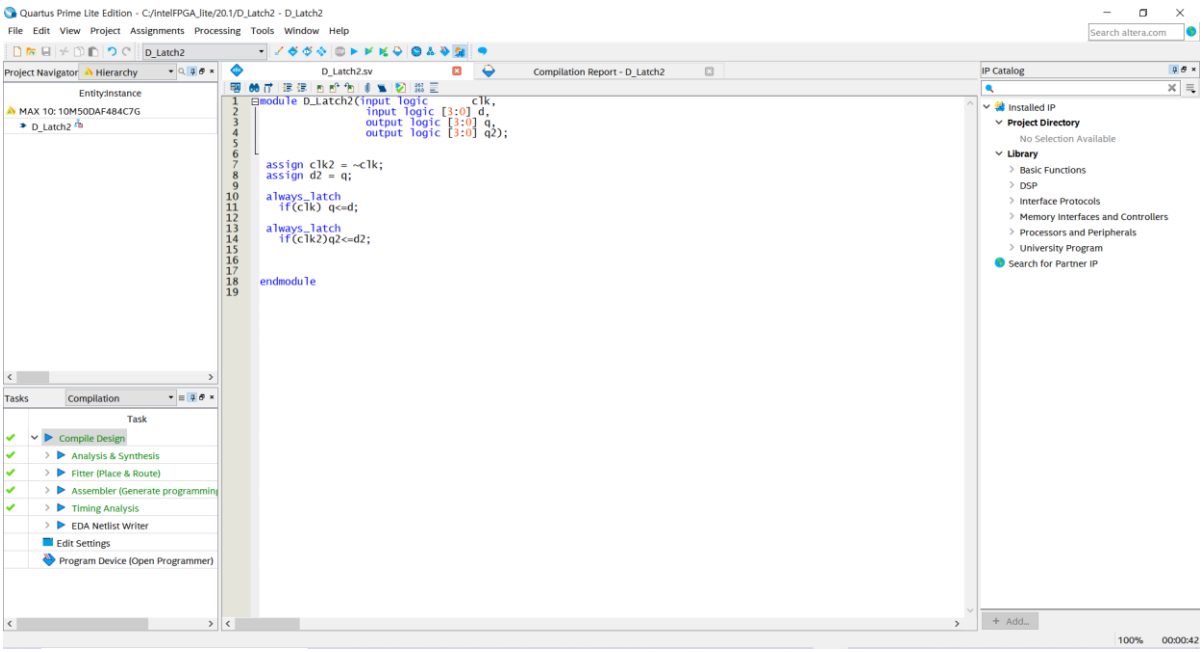


(Simulated Image)

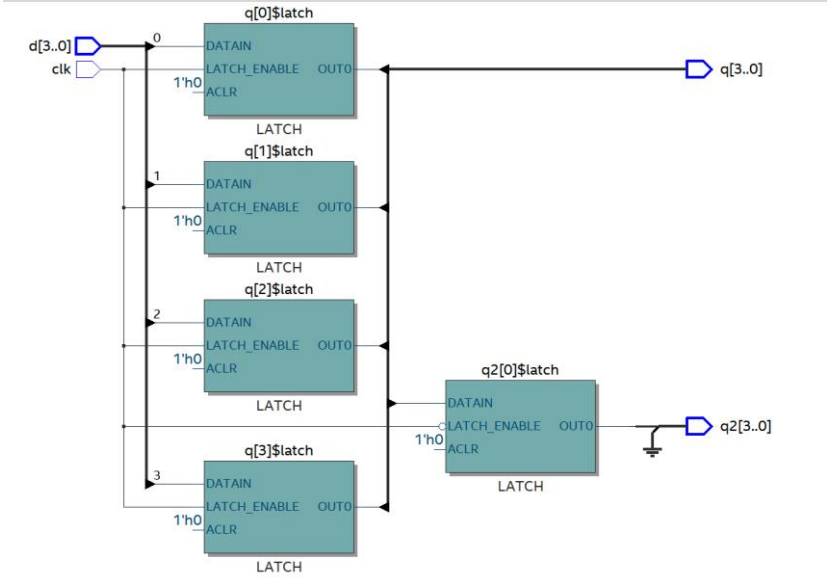
2. Example #2 (Example 4.21 variation)

```
1  module D_Latch2(input logic       clk,
2                  input logic [3:0] d,
3                  output logic [3:0] q,
4                  output logic [3:0] q2);
5
6
7    assign clk2 = ~clk;
8    assign d2 = q;
9
10   always_latch
11     if(clk) q<=d;
12
13   always_latch
14     if(clk2)q2<=d2;
15
16
17
18 endmodule
19
```

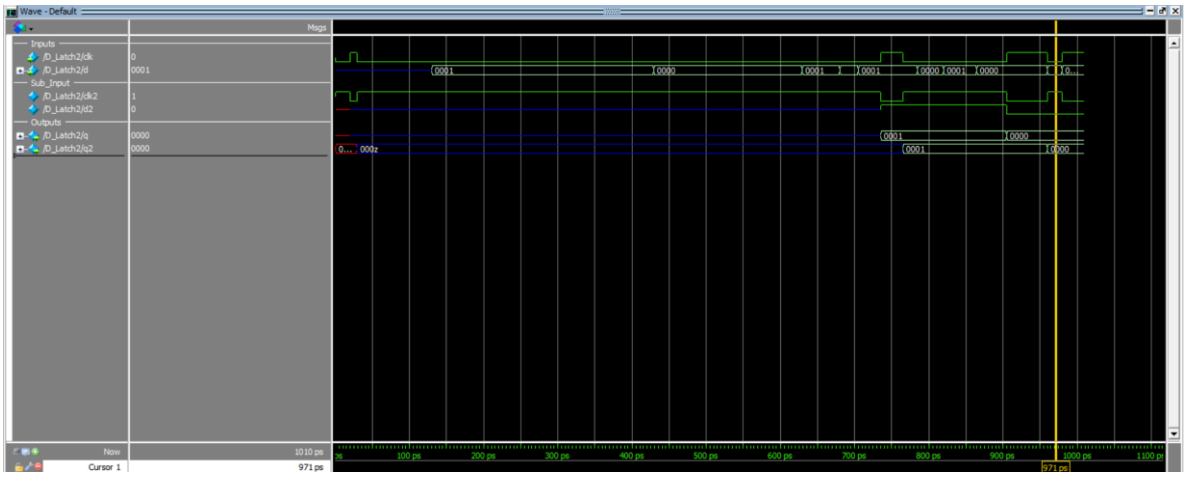
(Verilog Code)



(Synthesized Image)



(RTL Viewer)



(Simulated Image)

3 Glossary

1. Syntax

   Noun:

   1) That branch of modern logic that studies the various kinds of signs that occur in a system and the possible arrangements of those signs, complete abstraction being made of the meaning of the signs.
   2) The outcome of such a study when directed upon a specified language.
   3) A system or orderly arrangement.
   4) The grammatical rules and structural patterns governing the ordered use of appropriate words and symbols for issuing commands, writing code, etc., in a particular software application or programming language.

2. Module

   Noun:

   1) Part of a program that performs a distinct function
   2) An interchangeable, plug-in hardware unit

3. Endian

   Adjective:

   Denoting or relating to two systems of ordering data, in which the most significant unit is put either first or last.

4. Synthesize

   Verb:

   1) to form (a material or abstract entity) by combining parts or elements (opposed to analyze)
   2) to combine (constituent elements) into a single or unified entity

5. Precedence

   Noun:

   1) The right to precede in order, rank, or importance, priority
   2) The order to be observed in ceremonies by persons of different ranks, as by diplomatic protocol.

4 Interview Question

Question 4.3 What does the following SystemVerilog statement do?

```
result = | (data[15:0] & 16'hC820);
```

The following System Verilog statement adds the product(intersection) of 16 bit of ‘data’ and 1100100000100000 to the ‘result’.

5 Reflection

The content in this chapter was relatively easier to understand than other chapters. Although the HDLs are description language of the hardware, it was similar to my past experience of my programming language and the format was also similar to the other famous programming languages including python and C.

I think that HDLs are the tool to enlarge the aspect of the hardware and verify the theories in the past chapter or the future chapter. Thus, in my opinion, the most important thing after this chapter is understanding and using the concepts and the logic of the hardware. HDLs are the tools to construct the hardware and verify the logic of the hardware.

6 Questions

1. Is there any relation between HDLs and Assembly language?
2. Are there any limits in HDLs?
3. Which has more usage? Is it System Verilog or VHDL?