Digital Design & The Verilog HDL

Self Studies

Arnav Goyal

Winter 2023

CONTENTS

PREFACE

The purpose of this document is to act as a comprehensive note for my understanding on the subject matter. I may also use references aside from the lecture material to further organize my understanding, and these references will be listed under this portion.

In general this document follows the format of highlighting keywords in green. I can also introduce a *Definition* or a *Theorem*. There may also be various other things like code blocks which include `keywords` or `"strings"`. Remarks (similar to markdown style quotes). Or highlighted boxes. I might use these to organize things further if I deem necessary.

REFERENCES

- Provided Lecture Notes for ECE 2277 (Digital Logic Systems)
- Digital Design With An Introduction to the Verilog HDL - M. Mano, D. Ciletti
- Verilog Complete Tutorial - VLSI Point (YouTube Link)

Part I

DIGITAL DESIGN

Part II

THE VERILOG HDL

1 ~ THE BASICS

1.1 ~ HARDWARE DESCRIPTION LANGUAGES

Hardware Description Languages (HDLs) are a specialized computer language that are used to describe the structure and behavior of electronic circuits, they also include the notion of time-delays present in real digital circuits, and they also support multiple things happening at the same time - concurrency. This is different from programming languages which are line-by-line (sequential) in nature

This document goes over the Verilog HDL which is ubiquitous in the industry, and uses the `.v` file extension. Another key thing is that HDLs are used *after* designing the hardware, remember that they are NOT programming languages.

1.2 ~ LEVELS OF ABSTRACTION

Within this field of study we commonly encounter the need to use predesigned elements such as registers and multiplexers within more complex designs. When this arises we add a block for the multiplexer, its inputs, and its outputs, and treat it as a block without worrying about how it works. This is called abstraction.

There are three main levels of abstraction provided by Verilog, and we can code things at each level.

- Gate Level
- Dataflow Level (Register Transfer Level)
- Behavioural Level

allows us to connect hardware gate-by-gate, and wire-by-wire. Verilog has the basic gates ready for use through basic syntax. It is the lowest level of abstraction

```
1 and G1(out1, A, B);
2 or G2(out2, A, B);
3 nand G3(out3, A, B);
```

Snippet 1.1: Basic Gate Level Modeling

allows us to talk about how the data flows in a circuit and it deals with the concept of continuous assignment.

```
1 assign out1 = x & y;
2 assign out2 = x | y;
3 assign out3 = ~y;
```

Snippet 1.2: Basic RTL Modeling

allows us to describe the behavior of circuits in (almost) plain English, this is the highest level of abstraction and uses keywords such as `always` and `initial`. It is also usually the fastest way to write simple things such as conditional statements.

```
always @(sel, I0, I1):
begin
    if (sel)
        out = I1;
    else
        out = I0;
end
```

Snippet 1.3: Basic Behavioral Modeling

1.3 ~ MODULES & ENTITIES

A module is the basic building block of Verilog and it can be an element or collection of other lower-level modules. They provide their functionality to higher-level blocks through means of their ports, but they hide the internal implementation. This is essentially the concept of abstraction we discussed earlier, but this also allows for procedural based design.

```
module <moduleName> (
    [port-list]
);
// Module functionality goes here
endmodule
```

Snippet 1.4: Module Declaration Syntax

Aside from declaring modules, we can also create entities by instantiating modules. To learn by example we can use the code for a 2x1 MUX in Snippet 1.3 to create 4x1 MUX.

```
// Create the 2x1 MUX module
module mux_2x1 (
    input i0, i1, sel
    output out
);
    always @(i0, i1, sel)
    begin
        if (sel)
            out = i1;
        else
            out = i0;
    end
endmodule

// Create the 4x1 MUX module
module mux_4x1(
    input i0, i1, i2, i3, sel0, sel1,
    output out
);

    wire outA, outB;

    2x1mux A (i0,i1,sel0, outA);
    2x1mux B (i2,i3,sel0, outB);
    2x1mux C (outA, outB, sel1, out);

endmodule
```

Snippet 1.5: Module Instantition Example (4x1 MUX)