Arnav Goyal

Digital Design

Self Study

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 develop my understanding, and these references will be listed here.

This document should eventually serve as a standalone reference for learning or review of the subject matter. There is also a lot of organization within these documents, please refer to the table of contents within your PDF viewer for ease of navigation.

References

- Provided Course Materials from ECE 2277, ECE 3375
- Digital Design with an Introduction to the Verilog HDL - 5e - M. Mano, D. Ciletti
- Verilog Complete Tutorial - VLSI Point (YouTube Link)

The Basics

Hardware Description Languages

A **hardware description language (HDL)** is a specialized computer language used to describe the nature of digital electronic circuits.

- They include the notion of **time** [1] and **concurrency** [2]

In this handbook we will be concerning ourselves with the **Verilog** and **SystemVerilog** HDL. Both of which are used ubiquitously within the silicon industry for logic design and verification.

- SystemVerilog is a superset of Verilog[3]

Levels of Abstraction

Within this field of study we will often deal with the concept of **abstraction**. Verilog offers the description of things at three levels of abstraction.

- Gate-Level-Modeling[4]
- Dataflow Modeling[5]

[1] They include a notion of time through gate delays, and how long it takes for the signal to propagate through them

[2] Concurrency allows multiple things to happen at the same time, this is different to a regular programming language like python which is inherently not concurrent and is sequential in nature

[3] This is similar to the relationship between C and C++, we will also start by learning Verilog, and then diving into the features that SystemVerilog offers

[4] This is the lowest-level of abstraction and allows us to manually code each gate, and wire. Verilog already has the logic gates ready for use through basic syntax

[5] Also called Register Transfer Level Modeling, here we can talk about how data flows through our circuit, and it deals with continuous assignment

- Behavioural Modeling[6]

Here are some basic code examples of the three modeling styles ...

[6] This allows us to describe the operation of our circuits in english-like words, this is the highest level of abstraction

```
and G1(out, A, B);
or G2(out, A, B);
nand G3(out, A, B);
```

Code Snippet 1: Gate-Level Modeling

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

Code Snippet 2: RTL Modeling

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

Code Snippet 3: Behavioral Modeling

Modules & Entities

A **module** is the basic building block of Verilog, Modules are abstracted and interact with the external environment through their **ports**.

- Modules can be an element, or a collection of other (lower-level) modules
- Modules can be instantiated (but cannot be defined!) within other modules.
- A module that is not instantiated within any other module, is referred to as the **top-level module**

It is quite easy to declare a module as shown in Code Snippet 4, lets also learn by example though. Consider creating a 4×2 MUX from two 2×1 MUX blocks, which we have defined in Code Snippet 3. Lets assume that we encased that code in a module named `mux_2x1`.

```
1 module <moduleName> (
2 [port-list]
3 );
4 // Module specification goes here!
5 endmodule
```

Code Snippet 4: Module Declaration Syntax

```
module mux_4x1(
    input i0, i1, i2, i3, sel0, sel1,
    output out
);

// wires to route internal connections
wire outA, outB;

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

endmodule
```

Code Snippet 5: Creating a 4x2 MUX from two 2x1 MUX modules