Verilog notes

Tommy Bui

01-29-2023

Contents

1 Verilog Tutorial

Reference to Chipverify

1.1 Lore

In the early days of integrated circuits, engineers had to physically draw transistors & their netlist on paper. As circuits became more complex and larger in scale, this process eventually tedious. Languages such as VHDL & Verilog were developed to simply the process of describing the functionality of IC and let tools convert the behavior into hardware using combinational & sequential logic.

1.2 How is Verilog useful

Verilog creates a level of abstraction that hides the details of its implementation & technology.

E.g. the design of a D flip-flop requires the knowledge of the transistor layout in order to achieve an edge-triggered FF, rise/setup time, fall/clk-Q times to latch value onto flop, etc.

2 Introduction to Verilog

Source

A digital element such as a FF can be represented using combinational gates such as NAND or NOR gates. The functionality of a FF is determined based on the layout of such gates. How the gates have to be connected is usually determined using K-maps

Below is an schemetic of a Data flip-flop & its corresponding truth table. The output, q is asserted only when rstn & d are both set.

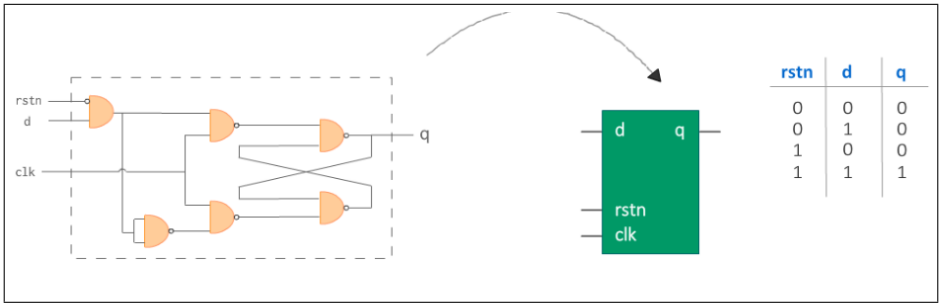

Figure 1: D flip flop

2.1 What is a hardware schematic?

A hardware schematic is a daigram that shows how the combinational gates should be connected to implemented a particular behaviour in hardware. From figure 1, a set of NAND gates are connected in order to create a D flip flop.

2.2 What is a Hardware Description Language?

It's easier to describe how a block of logic should behave & let software tools convert that behavior into an actual hardware scchematic. The langugage that describes the hardware functinality is classified as a Hardware Description Language.

2.3 Sections of Verilog Code

All behavior code should be described within the keywords module & endmodule.

2.3.1 Verilog section template

- Module definition & port list declaration
- List of input & output ports
- Declaration of Verilog data types
- Module instantiations
- Behavioural code

```
1  module [design_name] ( [port_list] );
2
3      [list_of_input_ports]
4      [list_of_output_ports]
5
6      [declaration_of_other_signals]
7
8      [other_module_instantiations_if_required]
9
10     [behavioral_code_for_this_module]
11 endmodule
```

Figure 2: Verilog example Template

3 Data Types

3.1 Verilog Syntax

Source (Lexical item. In lexicography [citation needed], a lexical item is a single word, a part of a word, or a chain of words (catena) that forms the basic elements of a language's lexicon (vocabulary)).

Lexical conventions in Verilog are similar to C in the sense that it contains a sense of tokens. A lexical token may consist of one or more characters and tokens can be comments, keywords, numbers, strings or white space. However, all lines are terminated by a semi-colon.

Verilog is case-sensitive, so variables, var_a & var_A differ.
Two types of comments:

- Single-line comment uses two forward slashes (e.g. //). Single line comments can be nested in a multiple line comment.
- Multiple-line comment starts with /* and ends with */ and cannot be nested

3.1.1 White space

White space refers to spaces, tabs, newlines, & formfeeds, and is usually ignored by Verilog except when it separates tokens. Be sure to take advantage of this when making readable code.

3.1.2 Operators

There are three types of operators: unary, binary, & ternary or conditional.

- Unary operators shall appear to the left of their operand
- Binary operators shall appear between their operands
- Conditional operators have 2 separate operates that separate 3 operands

```
1  x = ~y;              // ~ is a unary operator, and y is the operand
2  x = y | z;           // | is a binary operator, where y and z are its operands
3  x = (y > 5) ? w : z;  // ?: is a ternary operator, and the expression (y>5), w and z are its operands
```

Figure 3: Example of unary, binary, and ternary/conditional operators

If the expression (y¿5) is true, then variable x will get the value w, else y=z.

3.1.3 Number Format

```
16              // Number 16 in decimal
0x10            // Number 16 in hexadecimal
10000           // Number 16 in binary
20              // Number 16 in octal
```

Figure 4: number formats

By default, Verilog simulators treat numbers as decimals. In order to represent them in different radixes, there are certain rules which must be used.

Sized

Sized numbers are represented as show below, where size is written only in decimal to specify the number of bits in the number:

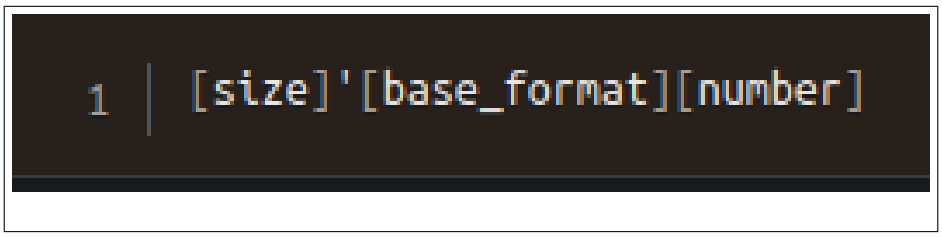

Figure 5: Template for sized numbers

- base_format can be decimal('d or 'D), hex('h or 'H), or octal('o or 'O) & specifies the base of the number
- number can be specified as any valid digit with respect to the specified based format

```
3'b010;      // size is 3, base format is binary ('b), and the number is 010 (indicates val
ue 2 in binary)
3'd2;        // size is 3, base format is decimal ('d) and the number is 2 (specified in de
cimals)
8'h70;       // size is 8, base format is hexadecimal ('h) and the number is 0x70 (in hex)
to represent decimal 112
9'h1FA;      // size is 9, base format is hexadecimal ('h) and the number is 0x1FA (in hex)
to represent decimal 506

4'hA = 4'd10 = 4'b1010 = 4'o12 // Decimal 10 can be represented in any of the four formats
8'd234 = 8'D234                  // Legal to use either lower case or upper case for base
format
32'hFACE_47B2;                   // Underscore (_) can be used to separate 16 bit numbers
for readability
```

Figure 6: ToDo

3.1.4 Unsized

Numbers without a size format specification have a default number of bits depending on the simulator & machine.

3.1.5 Negative

Negative numbers are specified with the - sign before the size of a number; It is illegal to have the - between base format & number.

```
1 integer a = 5423;       // base format is not specified, a gets a decimal value of 5423
2 integer a = 'h1AD7;     // size is not specified, because a is int (32 bits) value stored in a = 32'h0000_1AD7
```

Figure 7: ToDo

3.1.6 Strings

A sequence of characters enclosed in double quotes are strings. It cannot be split into multiple lines & every chracter in the string takes 1B to be stored:

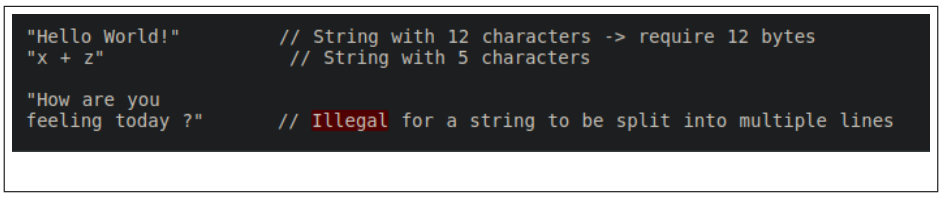

Figure 8: ToDo

3.1.7 Identifiers

Identifiers are names of variables; Can be made of alphanumeric characters and symbols. They cannot start with a digit nor a $:

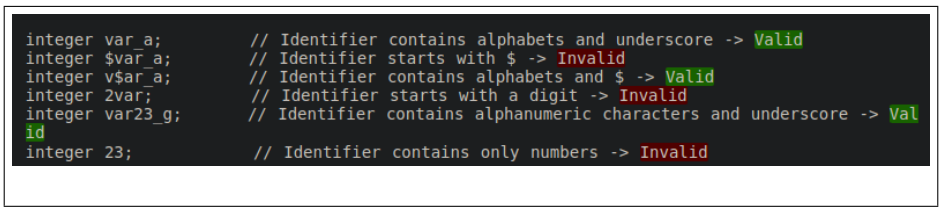

Figure 9: ToDo

3.1.8 Keywords

Keywords are special identifiers reserved to be define the language constructs & are lower case. A list of important keywords is as follows:

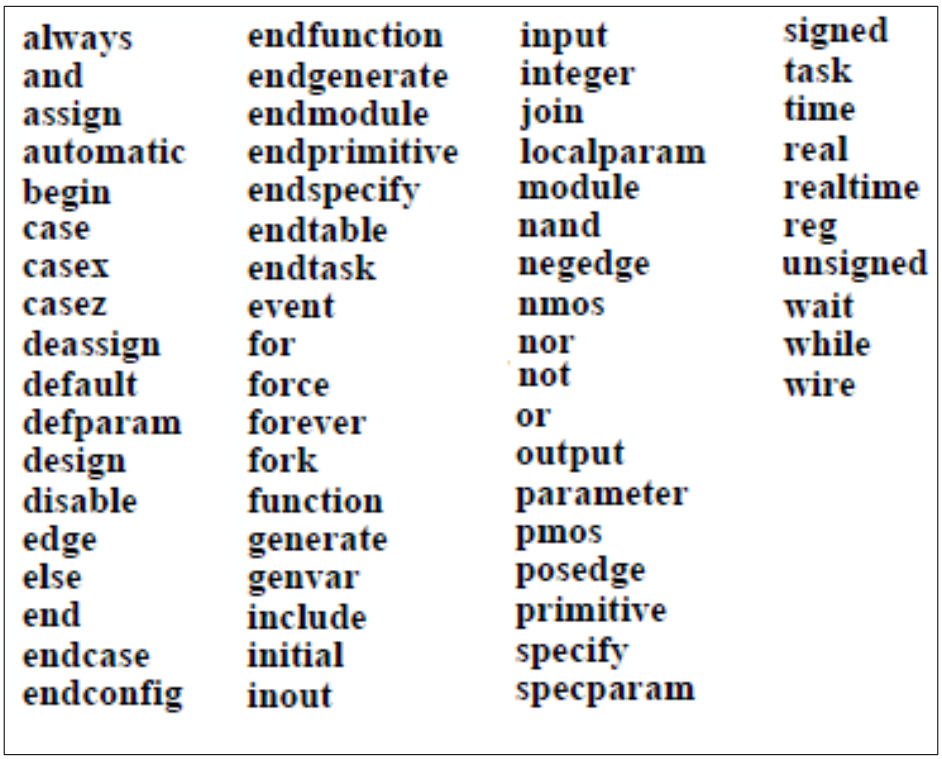

Figure 10: ToDo

3.1.9 Verilog Revisions

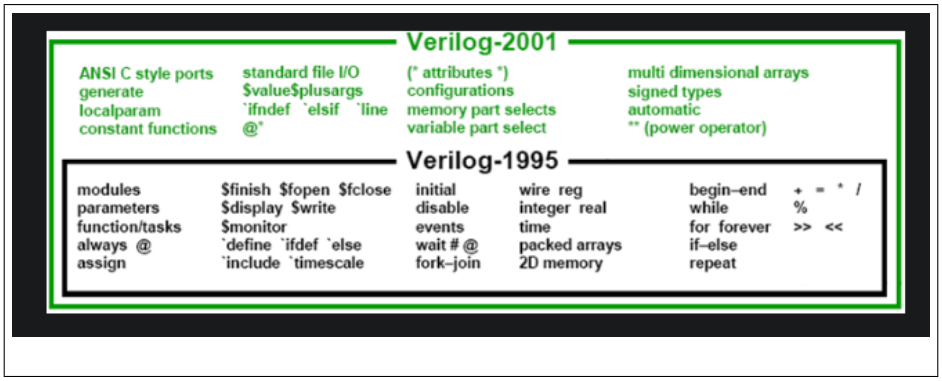

Figure 11: ToDo

3.2 Verilog Data Types

Source

Data-types in Verilog is meant to represent data storage elements like bits in a flip-flop & transmission elements like wires that connect between logic gates & sequential structures.

3.2.1 What values do variables hold?

Almost all data-types can have one of the four different values as given below except for real & event data types.

0	Represents logic zero or false condition
1	Refresents logic one or a true condition
x	Represents an unknown logic value (could be 0 or 1, metastability)
z	Represents high impedance state

The following image shows how these values are represented in timing diagrams & simulation waveforms. Most simulators use this convention where red stands for X and orange in the middle stands for high-impedance or Z.
:w

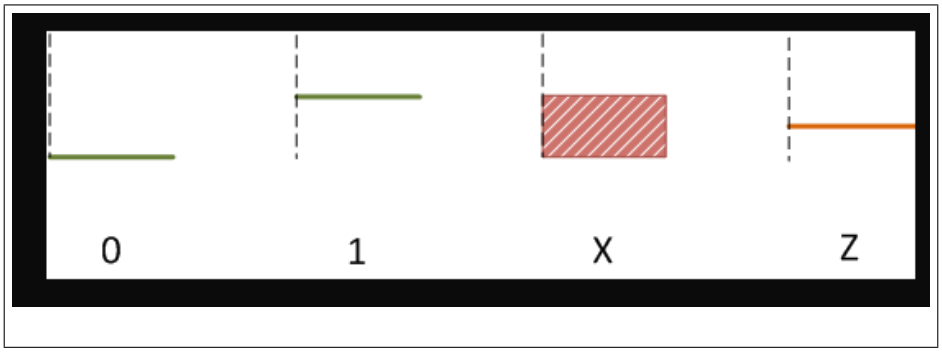

Figure 12: ToDo

3.2.2 What does the verilog value-set imply?

Since Verilog is essentially used to describe hardware elements like flip-flops & combinational logic like NAND & NOR, it has to model the value system found in hardware. A logic one would represent the voltage supply V_{dd}.

X or x means that the value is unknown at the time and could either be 1 or 0. This is an issue known as metastability. This differs from the boolean logic X, where it signifies don't care.

As with any incomplete eletric circuit, the wire that is not connected to anything will have high-impedance at that node which is represented by Z or z.

3.2.3 Nets & Variables

Nets & variables are the two main groups of data types which represent different hardware structures & differ in the way they are assigned & retain values.

Nets

Nets are used to connect between hardware entities like logic gates & don't store any values on their own. In the follow image, a net_11 connects between the output of the AND gate and the first input of the flip_floped called data_0. Similarly, the inputs of the AND gate are connected to nets, net_45 & net_67.

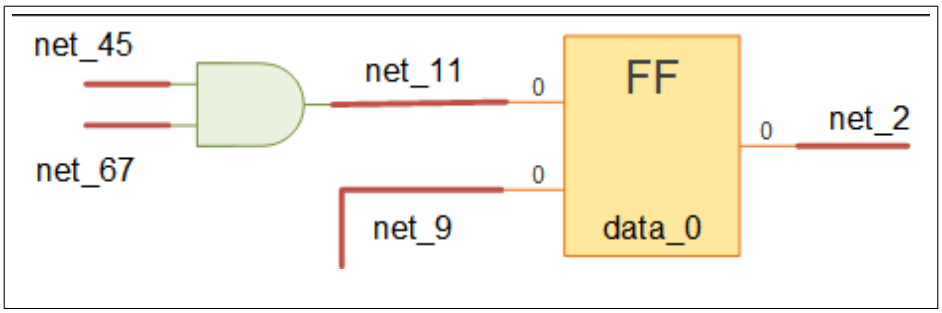

Figure 13: ToDo

There are different types of nets with various characterisitcs, the most commonly used net in digital design is of type wire. Wire is a Verilog data-type used to connect elements & to connect nets that are driven by a single gate or continuous assignment.

When there is a requirement for multiple nets, they can be bunched together to form a single wire. In the code below, we can have a 4-bit wire that sends 4 separate values on each one of these wires. A group of such entities is consider a vector.

$$wire[3:0]n0; //4-bitwire; exampleofavector$$

It is illegal to redeclare a variable name already declared by a net, parameter, or variable.

Variables

A variable is an abstraction of a data storage element & can hold values e.g. A flipflop.

Verilog datatype, reg can be used to model hardware registers since it can hold values between assignments. Note that a reg does not always represent a flipflop as it can also represent combinational logic.

3.2.4 Other DataTypes

Integer

Properties of datatypes in SV (hopefully same in Verilog)

An integer is a general purpose variable of 32-bits wide that can be used for other purposes while modeling hardware & stores integer values: