design.v

testbench.sv

Programming ‘Verilog’ {

[For Beginners Workshop]

“Verilog is a Hardware Description Language.”

-Doulos

}

Ahmad Reza Mirzaei

Table Of ‘Contents’ {

}

01 {

[What is Verilog?]

It’s a Hardware Description Language.

}

design.v

testbench.sv

Introduction {

Textual format for describing

- Circuits
- Systems

Verification

- Functionality
- Timing
- Power
- Utility

}

design.v

testbench.sv

```
1  FPGA < /1 > {
2
3
4      Field-Programmable Gate Array
5
6  }
7
8
9  ASIC < /2 > {
10
11
12     Application-Specific Integrated Circuit
13
14 }
```

Ahmad Reza Mirzaei

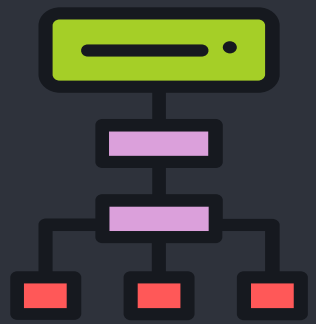
Abstraction Levels {

Behavioral level

Dataflow level

Gate level

Switch level



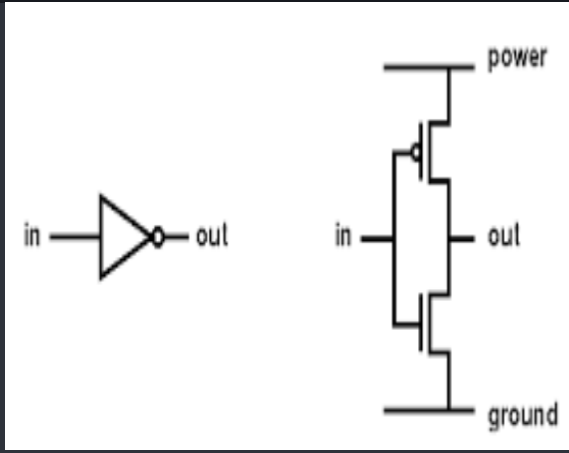
}

Switch Level {

Transistor level modeling

It is rarely used by designers these days as the complexity of circuits have required them to move to higher levels of abstractions.

}



Gate Level {

implemented in terms of logic gates and interconnections between these gates

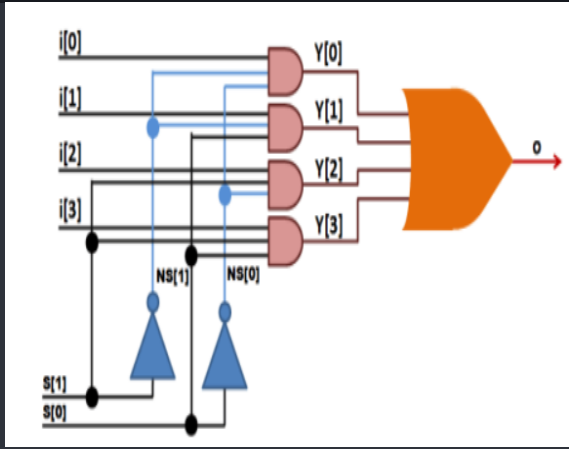
resembles a schematic drawing with components connected with signals

A change in the value of any input signal of a component activates the component

closer to the physical implementation

Since logic gate is most popular component, Verilog has a predefined set of logic gates known as ***primitives***

}



design.v

testbench.sv

Dataflow Level {

Designed by specifying the data flow

Realize how data flows between hardware registers

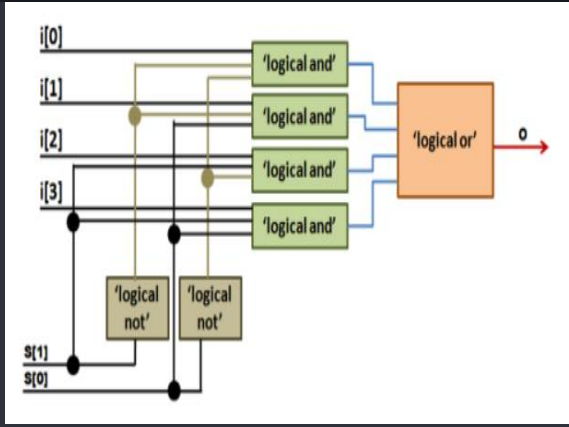
Realize how the data is processed in the design

Similar to logical equations

Made up of input signals and ***assigned*** to outputs

}

Can be quite easily translated into a structure and then implemented



design.v

testbench.sv

Behavioral Level {

Highest level of abstraction

Desired design algorithm

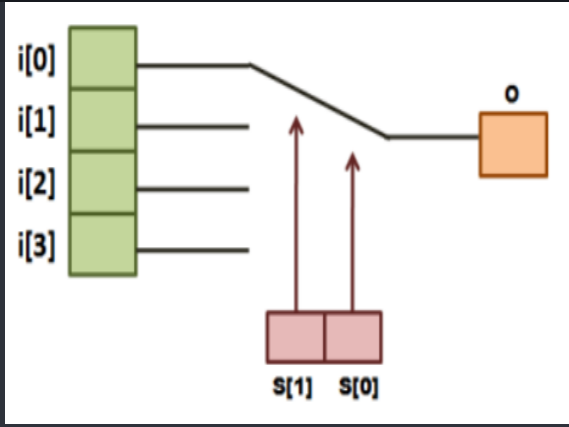
No concern for the hardware implementation details

Circuit in terms of its expected behavior

Closest to a natural language description of the circuit functionality

Most difficult to synthesize

}



design.v

testbench.sv

Synthesis-Based < /1 > {

You can synthesis it and even program it on FPGA.

}

Simulation-Based < /2 > {

You can just use it for simulation because it’s not synthesizable.

}

02 {

[How to write Verilog?]

The answer is “Really hard”

}

Ahmad Reza Mirzaei

design.v

testbench.sv

```
1  module name ([ports]);
2
3
4      A block of Verilog code that implements a certain
       functionality
5
6      Can be embedded within other modules
7
8      Higher level module can communicate with its lower
       level modules using their input and output ports
9
10     Ports declared in the ports cannot be redeclared
11     within the body of module
12
13 endmodule
14
```

design.v

testbench.sv

```
1  module name ([ports]);
2
3      Are a set of signals that act as inputs and outputs to
4      a particular module
5
6      Are the primary way of communicating with module
7
8  🖇️ Types of Ports
9      Input : receive values from outside
10     Output : send values to the outside
11     Inout : either send or receive values
12
13 endmodule
14
```

Module instantiation {

Modules can be instantiated within other modules and ports of these instances can be connected with other signals inside the parent module

We can connect ports in two manners:

By ordered list : connect ports by order of ports that declared in module definition

By name : connect ports using their name

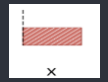
}

Data types {

Almost all data types can only have one of the four different values as given below except for ***real*** and ***event*** data types

0 : logic zero or false condition 

1 : logic one or true condition 

x : unknown logic value (can be zero or one) 

z : high-impedance state 

}

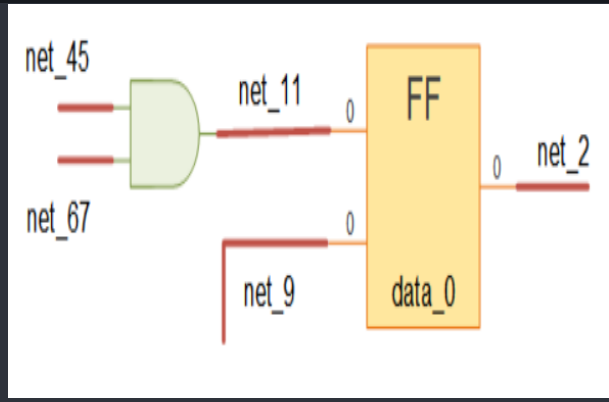
Nets {

Are used to connect between hardware entities like logic gates and hence do not store any value on its own

The most popular and widely used net in digital designs is of type ***wire***

The ***wire*** is similar to electrical wire that is used to connect two components on a breadboard

}



design.v

testbench.sv

Variables {

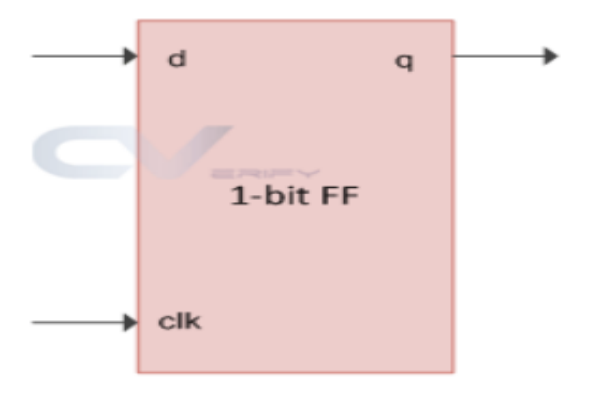
Is an abstraction of a data storage element and can hold values

A flip-flop is a good example of a storage element

Data type ***reg*** can be used to model hardware registers

Note that a ***reg*** need not always represent a flip-flop

}



design.v

testbench.sv

```
assign net = other_net or expression of different_nets
```

After equal sign you can place a signal name which can be either a single signal or a concatenation of different signal nets or even an expression of different signals.

Verilog Operators {

There'll always be some form of calculation required in digital systems

Let's look at some of operators in Verilog that would enable synthesis tools realize appropriate hardware elements

}

Arithmetic Operators {

a + b	a plus b
a - b	a minus b
a * b	a multiplied by b
a / b	a divided by b
a % b	a modulo b
a ** b	a to the power of b

}

If the second operator of division or modulo is zero, then the result will be x

If either operand of the power operator is real, then the result will also be real.

The result will be 1 if the second operand of a power operator is 0

Relational Operators {

```
a < b    a less than b

a > b    a greater than b

a <= b   a less than or equal to b

a >= b   a greater than or equal to b
```

}

An expression with relational operator will result in a 1 if the expression is evaluated to be true, and 0 if it is false

If either of the operands is x or z, then the result will be x

design.v

testbench.sv

Equality Operators {

```
a === b  a equal to b, including x and z

a !== b  a not equal to b, including x and z

a == b   a equal to b, result can be unknown

a != b   a not equal to b, result can be unknown
```

}

The result is 1 if true, and 0 if false

If either of the operands of == or != is x or z, then the result will be x

The result of === and !== always have a known value

design.v

testbench.sv

Logical Operators {

```
a && b   evaluates to true if a and b are true

a || b   evaluates to true if a or b are true

!a       converts non-zero value to zero and vice versa


 If either of operands is x then the result will be x
```

}

Bitwise Operators {

a & b

a | b

&	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

|	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

The operator will combine a bit in one operand with its corresponding bit in the other operand to calculate a single bit result

}

design.v testbench.sv

Shift Operators {

```
a << b   shifts a to left by b bits (zero padding)
a >> b   shifts a to right by b bits (zero padding)
a <<< b  shift a to left by b bits (zero padding, same as <<)
a >>> b  shift a to right by b bits (sign padding)
```

}

Concatenation {

Multi-bit Verilog wires and variables can be clubbed together to form a bigger multi-net wire or variable using ***concatenation*** operators { and } separated by commas

Concatenation is also allowed to have expression and sized constants as operands in addition to wires and variables

}

design.v

testbench.sv

Replication {

When the same expression has to be repeated for a number of times, a replication constant is used which need to be a non-negative number and cannot be x, z or any variable

this constant number is also enclosed within bracers along with the original concatenation operator and indicates the total number of times the expression will be repeated

}

design.v

testbench.sv

```
always @ (event);


    Is one of the procedural blocks in Verilog

    Statements inside an always block are executed sequentially

    The always block is executed at some particular event

    the event is defined by sensitivity list


end
```

design.v

testbench.sv

```
always @ (event);

end
```

Is the expression that defines when the always block should be executed

Is specified after the @ operator within parentheses ( )

May contain either one or a group of signals whose value change will execute the always block

design.v

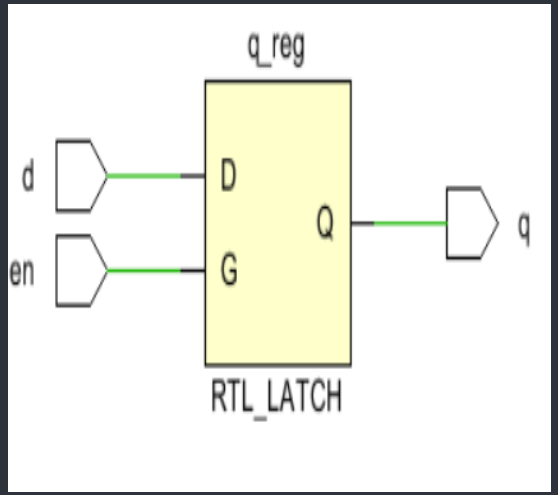
testbench.sv

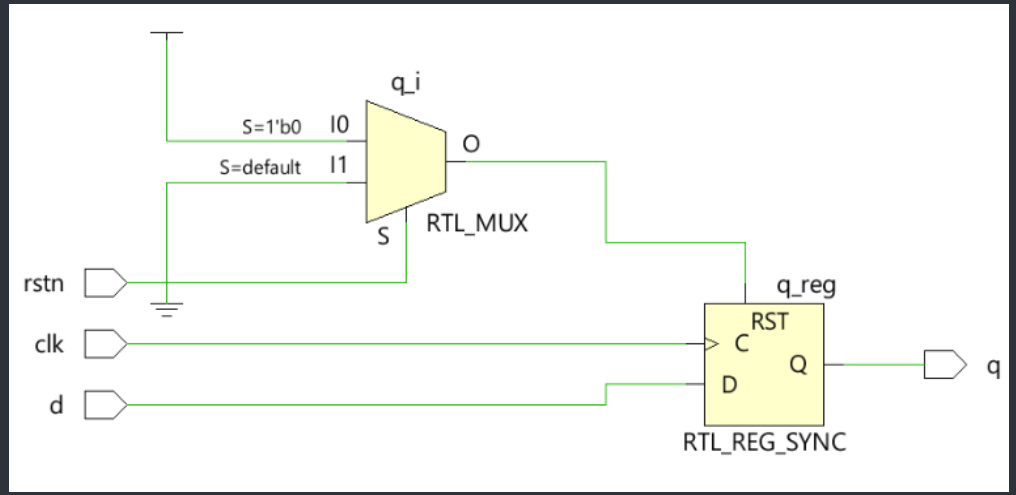
```
if ([expression]) begin

    This conditional statement is used to make a decision on whether
    the statement within if block should be executed or not

    If the expression evaluates to be false (0, x, z) the statement
    inside if block will not be executed

end else begin

    If there is an else statement and expression is false then
    statements within the else block will be executed

end
```

design.v

testbench.sv

Let’s Design {





}

design.v

testbench.sv

```
for (initial; condition; step) begin



end
```

Is the most widely used loop in software

Is primarily used to replicate hardware logic in Verilog

design.v

testbench.sv

```
genvar i;
generate
        for (initial; condition; step) begin



    Allows to multiply module instances or perform conditional
    instantiation of any module



        end
endgenerate
```

design.v

testbench.sv

```
Thanks; {

    'Do you have any questions?'

        ahmadrezamirzaei38@gmail.com

        ahmadrmirzaei

        ahmadrmirzaei

        ahmadrmirzaei

}
```

Ahmad Reza Mirzaei