Digital Design with Verilog

Verilog - Synthesis

Lecture 18: Logic Synthesis with Verilog HDL – Part 1

Learning Objectives

- Mapping of few Verilog HDL types and constants to hardware
  - Reg data type
  - Constants
  - Parameters
- Value Holders for Hardware Modeling
  - Wire
  - Flip-Flop
  - Latch

Register Data Type

- The different kinds of register types that are supported for synthesis are:

  reg integer

- A reg declaration explicitly specifies the size, that is, the corresponding number of bits of the variable in hardware.

```
reg [1:25] Cpt;   // 25-bit variable
reg Bxr;          // 1-bit variable
```

- When no size is explicitly specified in a reg declaration, the default is one bit.

Register Data Type

- For an integer type, the maximum size is 32 bits and the number is assumed to be in 2's complement form.

- Optionally a synthesis system may perform data flow analysis of the model to determine the maximum size of an integer variable. For example

```
wire [1:5] Brq, Rbu;
integer Arb;
...
Arb= Brq + Rbu;
```

Register Data Type

- Size of *Arb* is determined to be 6 bits.
  - An adder of size 6 is sufficient.
  - The leftmost bit is the carry bit.

- The register types: time and real, are not supported for synthesis.

Constants

- There are three kinds of constants in Verilog HDL: integer, real and string.

- Real and string constants are not supported for synthesis.

- An integer constant can be written in either of the following two forms.
  - Simple decimal
  - Base format

Constants

- If an integer is written in the base format form, then the integer is treated as an unsigned number.

- If a size is explicitly specified for the integer, then the specified size is the number of bits used for the integer; if not, 32bits is used for the size.

```
30      Signed number, 32 bits
-2      Signed number, 32 bits in 2 's complement
2'b10   Size of 2 bits
6'd-4   6-bit unsigned number (-4 is represented in
            2's complement using 6 bits)
'd-10   32-bit unsigned number ( -10 is represented in
            2's complement using 32 bits)
```

Parameters

- A parameter is a named constant. Since no size is allowed to be specified for a parameter, the size of the parameter is the same as the size of the constant itself.

```
parameter RED = -1 , GREEN = 2 ;
parameter READY= 2'b01, BUSY= 2'b11, EXIT= 2'b10;
```

- *RED* and *GREEN* are two 32-bit signed constants. *READY, BUSY* and *EXIT* are three parameters of size 2 bits each.

Value Holders for Hardware Modeling

- The basic value holders in hardware are:
  - Wire
  - Flip-flop (an edge-triggered storage element)
  - Latch (a level-sensitive storage element)

- A variable in Verilog HDL can either be of the net data type or the register data type.

- For synthesis, a variable of net type maps to a wire in hardware and a variable of the register type maps either to a wire or a storage element (flip-flop or latch) depending on the context under which the variable is assigned a value

Value Holders for Hardware Modeling

- In Verilog HDL, a register variable retains its value through the entire simulation run, thus may infer memory.
  - However, this is too general for synthesis.
- Here is an example of a variable that is used as a temporary and therefore need not be a candidate for a storage element.

Value Holders for Hardware Modeling

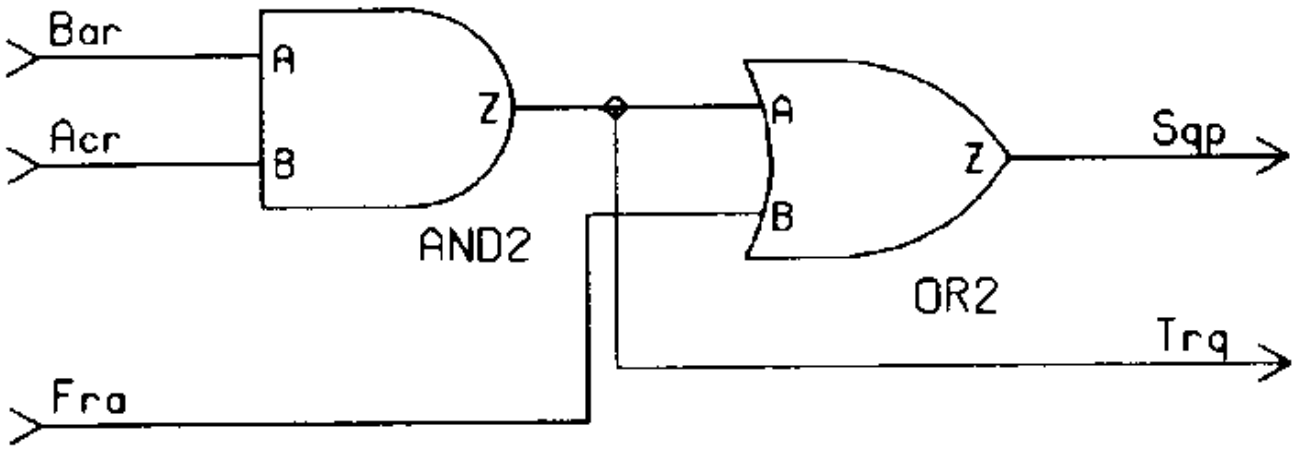
```
wire Acr, Bar, Fra;  //A wire is a net type.
reg Trq, Sqp;  //A reg is a register type.
...
always @ (Bar or Acr or Fra)
begin
Trq = Bar & Acr;
Sqp = Trq | Fra;
end
```

- Variable *Trq* is assigned in the first statement and then used in the right hand- side expression of the second statement.

- Verilog HDL semantics indicate that *Trq* retains its value through the entire simulation run.

Value Holders for Hardware Modeling

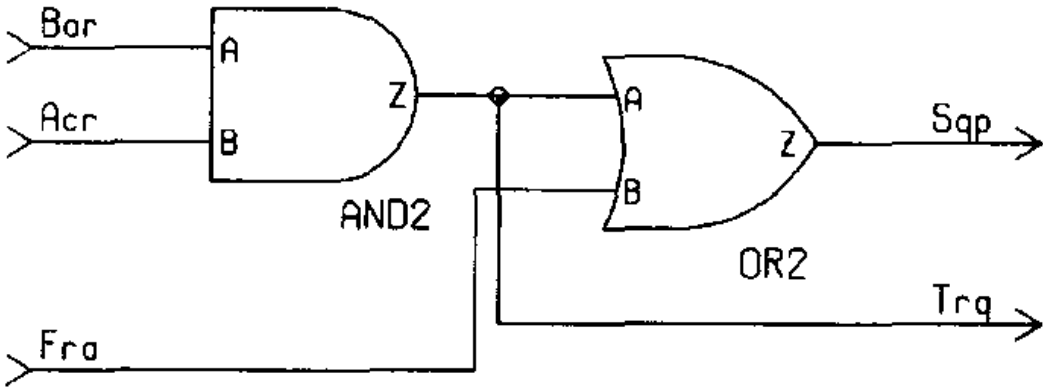
- However, it is not necessary to store the value of *Trq* as a storage element in hardware, since it is assigned and used immediately.



Variable *Trq* is a wire.

Value Holders for Hardware Modeling

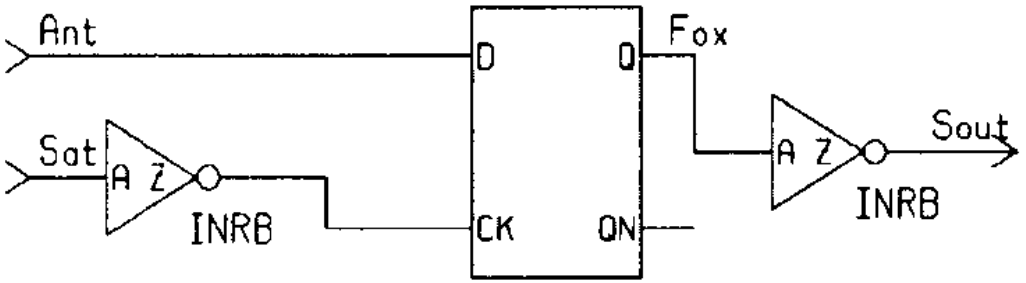
```
wire Acr, Bar, Fra;
reg Trq, Sqp;
...
always @ (Bar or Acr or Fra)
begin
    Sqp = Trq | Fra;
    Trq = Bar & Acr;
end
```



No latch for variable *Trq*.

Value Holders for Hardware Modeling

```
wire Sat, Ant;
reg Fox, Sout;
...
always @ (Sat or Ant)
begin
if (!Sat)
    Fox = Ant;
Sout = ! Fox;
end
```



FD1S1A

Variable *Fox* is a latch

Verilog Constructs to Gates

Objectives

- Synthesis of Verilog Constructs
  - –Procedural Assignment statements
- Blocking Vs Non Blocking
- Assignment restrictions for synthesis

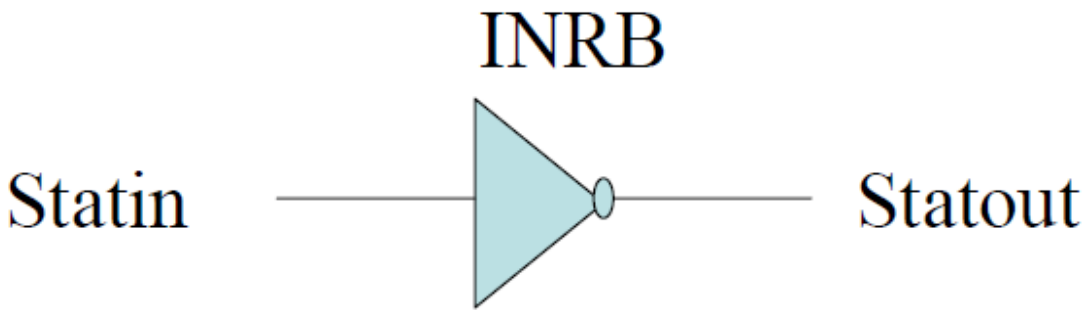
Continuous Assignment

- This represents in hardware, logic that is derived from the expression on the right-hand-side of the assignment statement driving the net that appears on the left-hand-side of the assignment statement.

- The target of a continuous assign statement is always a net driven by combinational logic.

```
module Continuous (Statin, StatOut);
        input Statin;
        output Statout;
        assign Statout= ~Statin;
endmodule
```

The Synthesized circuit



Delays in continuous assignments are ignored by the synthesis tool.

assign #2 EffectiveAB= DriverA| DriverB;

The #2 delay is ignored

Procedural Assignment Statement

- Two types
  - –Blocking
  - –Non-blocking
- In hardware, logic that is derived from the expression on the right-hand-side of the assignment statement drives the variable that appears on the left-hand-side of the assignment.

Procedural Assignments

- Procedural assignments can appear within an ‘initial’ or ‘always’ block; however, an ‘initial’ block is not supported for synthesis.

- **We consider procedural assignments only inside a ‘always’ block.**

Blocking procedural assignment

```
module Blocking (Preset, Count);
    input [0:2] Preset;
    output [3:0] Count;

    reg[3:0] Count;

    always @(Preset) //Adder with 1 and Preset as
    Count = Preset + 1; //Inputs
endmodule
```

- A 3-bit adder is costly –so we optimize.

The Optimization

pre0	pre1	pre2	Cou3	Cou2	Cou1	Cou0
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	0	1	1
0	1	1	0	1	0	0
1	0	0	0	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	1	1
1	1	1	1	0	0	0

The equations

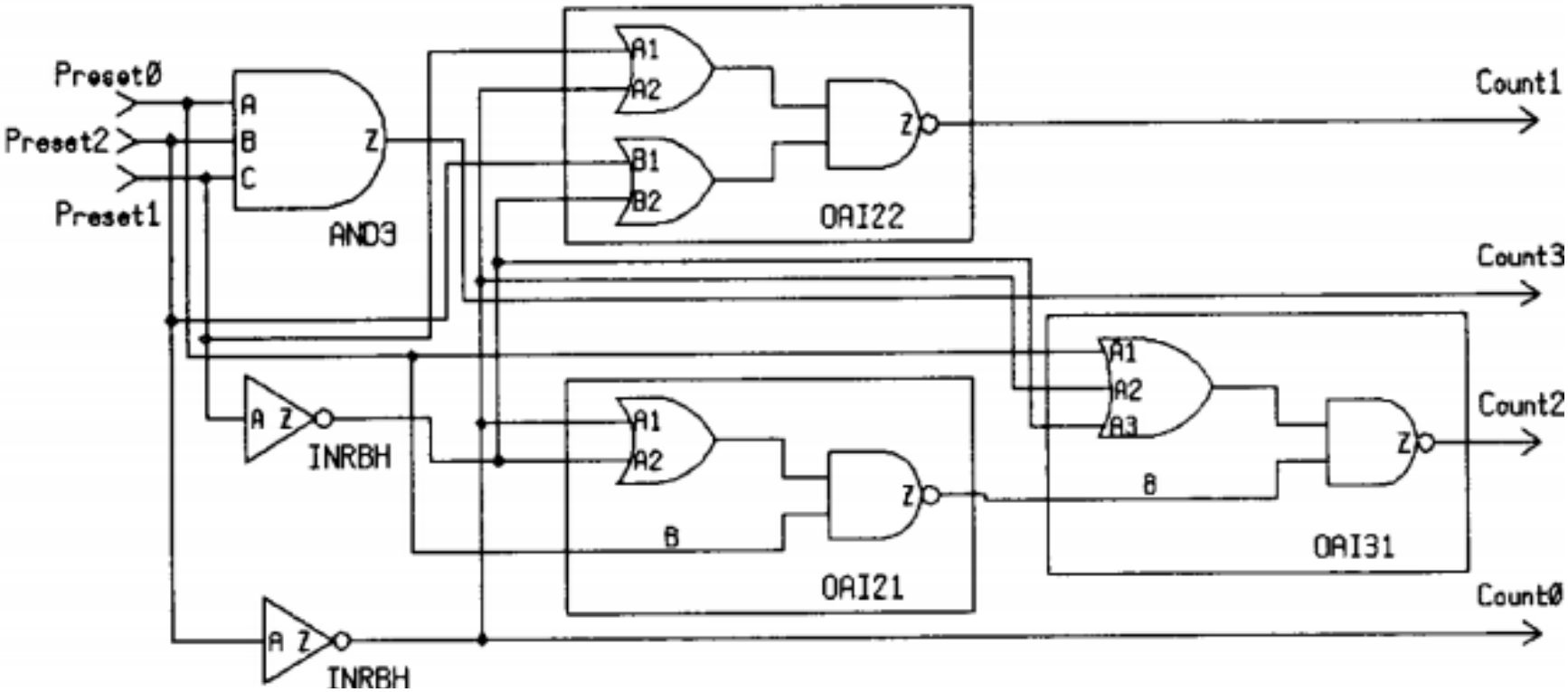
```
Count0 = !Preset2;
Count1 = (!Preset1 & Preset2) | (Preset1 & !Preset2)
Count2 = Preset0 & (!Preset1 | !Preset2) |
         (!Preset0 & Preset1 &Preset 2)
Count3 = Preset0 & Preset1 & Preset2;
```

- We use Demorgan’slaw

$$\overline{A}\,\overline{B} + \overline{C}\,\overline{D} = \overline{(A+B)\,(C+D)}$$

- OR –AND –INVERT construct
- See Slide for the circuit

Implemented Circuit



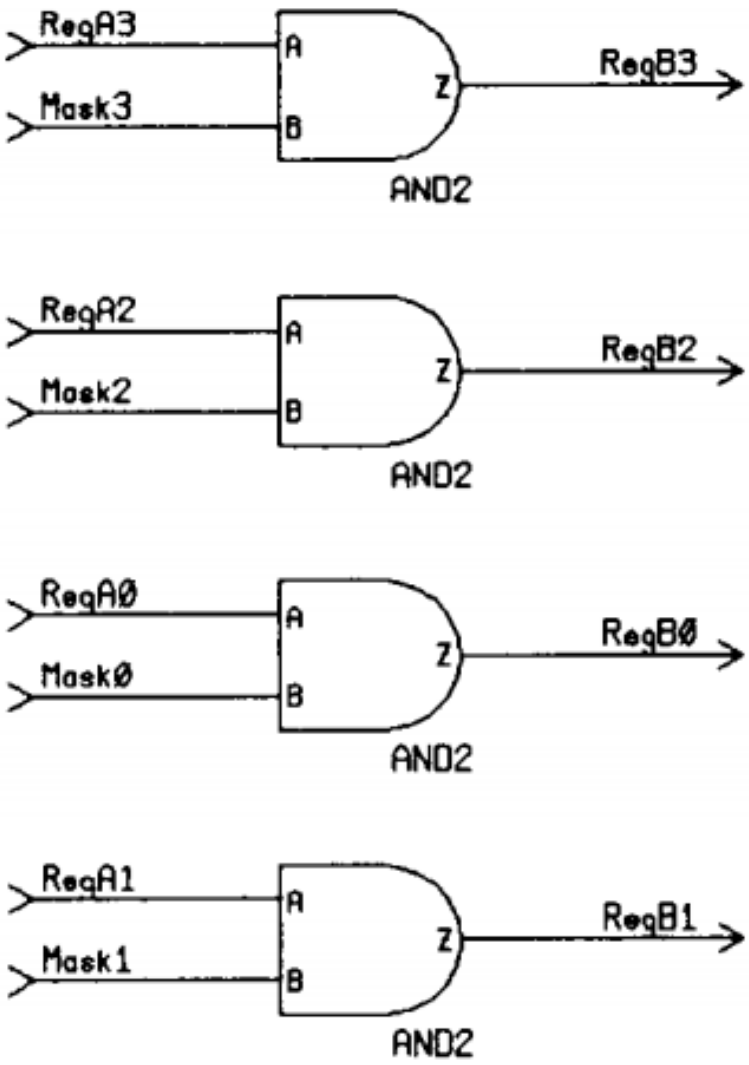
Non-Blocking Procedural Assignment

```
module NonBlocking(RegA, Mask, RegB);
  input [3:0] RegA, Mask;
  output [3:0] RegB;

  reg[3:0] RegB;
  always @(RegA or Mask)
  RegB<= RegA& Mask;

endmodule
```

The Circuit


ReqA3
Mask3
A
B
Z
ReqB3
AND2
ReqA2
Mask2
A
B
Z
ReqB2
AND2
ReqAØ
MaskØ
A
B
Z
ReqBØ
AND2
ReqA1
Mask1
A
B
Z
ReqB1
AND2

The Synthesis part

- Blocking or Non-blocking does not change the resultant combinational logic.

- Tips: Use Blocking assignments for modeling combinational logic and non-blocking assignments for modeling sequential logic.

- Why? –wait for some more time ☺))

Target of Assignment

- Target of a procedural assignment is synthesized to a wire, a flip-flop, or a latch, depending on the context.
- If the assignment is under the control of an edge of a clock, it infers a flip-flop.

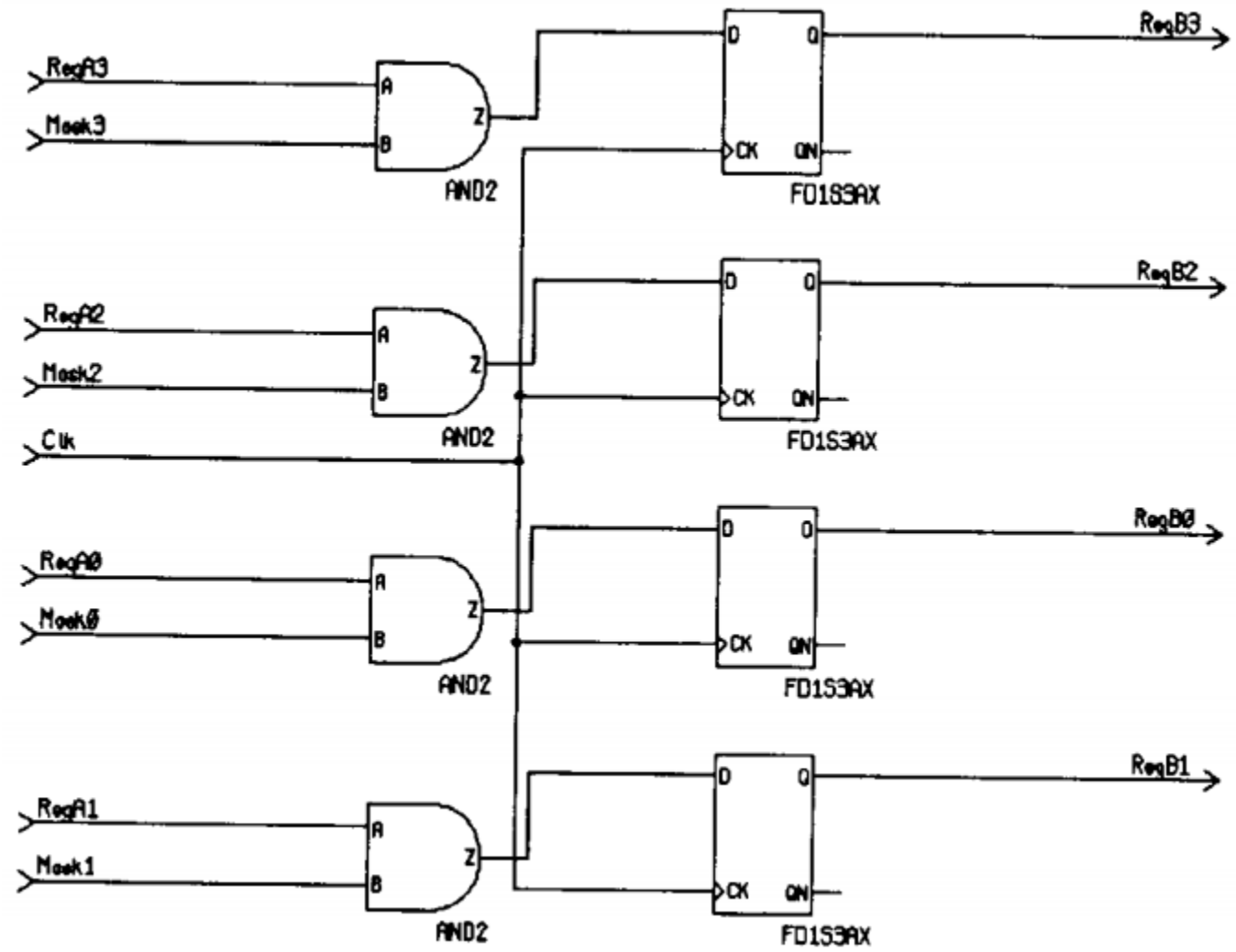
```
module Target(Clk, RegA, RegB, Mask);
  input Clk;
  input [3:0] RegA, Mask;
  output [3:0] RegB;

   reg[3:0] RegB;

   always @(posedgeClk)
   RegB<= RegA& Mask;

endmodule
```

The Circuit


RegA3
Mask3
AND2
RegB3
FD1S3AX
RegA2
Mask2
Clk
AND2
RegB2
FD1S3AX
RegA0
Mask0
AND2
RegB0
FD1S3AX
RegA1
Mask1
AND2
RegB1
FD1S3AX

Assignment Restrictions

- Any kind of delay control is ignored. This can basically lead to a mismatch between the Verilog and Synthesized modules.

```
#5 RegB<= RegA& Mask;
RegB= #2 RegA& Mask;
```

- Rule: A target cannot be assigned using a blocking assignment and a non-blocking assignment.

Explaining the Rule.

- If a target is assigned using a blocking (or a non-blocking) assignment, then the same target can only be assigned again using a blocking (or a non-blocking) assignment.

```
Count = Preset + 1;
Count <= Mask; //This is illegal since count is
//previously assigned using a bcloking assignment
```

References

- Chapter 1 & 2, Verilog HDL Synthesis by J. Bhaskar

- Disclaimer: I don’t own all the slides, these slides are copied and adopted from various public resources available on the internet.

Thank you