

VERILOG

Deepjyoti Borah, Diwahar Jawahar

Outline

- ▶ 1. **Motivation**
- ▶ 2. Basic Syntax
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. Conditions and Loops in Verilog
- ▶ 5. Procedural Assignment
- ▶ 6. Timing controls
- ▶ 7. Combinatorial Logic in Verilog
- ▶ 8. Race conditions
- ▶ 9. Summary

What is Verilog ?

- ▶ Verilog is a hardware description language(HDL)
- ▶ Verilog is used to model digital circuits
- ▶ Verilog is a data flow language
- ▶ The way it differs from the programming language is by describing propagation time and signal strength

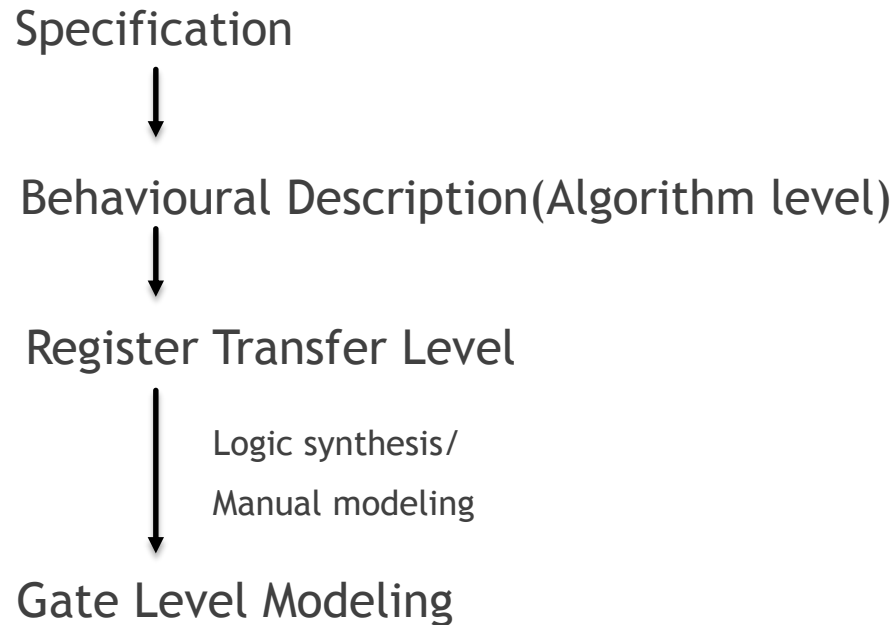
Verilog Timeline

- ▶ 1983/84-Developed by Prabhu Goel and Phil Moorby as Automated Integrated Design System
- ▶ 1985-Renamed as Gateway Design Automation
- ▶ 1990-Cadence Design Systems purchase it
- ▶ 1995-Released in public domain: IEEE 1364-1995 or Verilog-95
- ▶ 2001-Verilog 2001
- ▶ 2005-Verilog 2005

Why Verilog ..?

- ▶ Easy verification of circuits : replacement of breadboard and hand layout
- ▶ Concurrency of processes in hardware elements
- ▶ Logic synthesis
- ▶ Abstract level description of design without choosing specific fabrication technology
- ▶ Functional verification is early in the design process to meet requirements

How Verilog works



Approach:

Top-----→ Down

Bottom-----→Up

Mixed

Outline

- ▶ 1. Motivation
- ▶ 2. **Basic Syntax**
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. Conditions and Loops in Verilog
- ▶ 5. Procedural Assignment
- ▶ 6. Timing controls
- ▶ 7. Combinatorial Logic in Verilog
- ▶ 8. Race conditions
- ▶ 9. Summary

Modules

```
module test;  
reg clk, reset;  
wire [3:0] q;
```

```
ripple_carry_counter rcc(q, clk,  
reset);
```

```
initial begin
```

```
.....
```

```
.....
```

```
.....
```

```
end
```

```
endmodule
```

```
module tff(q, clk, reset);  
output reg q;  
input clk, reset;
```

```
always @(.....)  
begin
```

```
.....
```

```
.....
```

```
.....
```

```
end
```

```
endmodule
```

```
module ripple_carry_counter(q, clk,  
reset);  
output[3:0] q;  
input clk, reset;
```

```
tff tff0(q[0], ~clk, reset);
```

```
.....
```

```
.....
```

```
endmodule
```


A Verilog Module

Module Name

Port list, port declaration

Parameters

Declaration of Variables, wires etc.

Instantiation of lower end modules

Tasks & functions

Data flow statements:

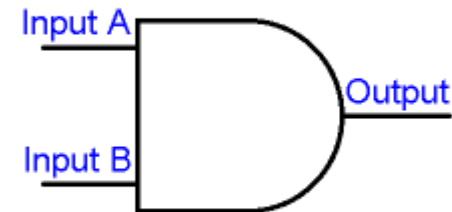
Always and initial blocks

(All behavioural statements)

End module

Ports

- ▶ Allow communication between a module and its environment.
- ▶ Three types of ports:
 - ▶ Input
 - ▶ Output
 - ▶ Inout



Data Types

- ▶ Register - Something that stores a value
- ▶ Nets- Connection between elements (commonly known as wire)
- ▶ Value set: 0, 1, x(unknown), z(high impedance)
- ▶ Reg unsigned variable
- ▶ Integer signed variable (32 bits)
- ▶ Time unsigned integer (64 bits)
- ▶ Real double precision floating point variable

Definition of Constants

- ▶ Little - Endian convention is widely used.
- ▶ `<width>`<base letter> <number>`
- ▶ Constants can be specified in decimal, hexadecimal, octal or binary format

E.g. `x = 4'd1`

Outline

- ▶ 1. Motivation
- ▶ 2. Basic Syntax
- ▶ 3. **Sequential and Parallel Blocks**
- ▶ 4. Conditions and Loops in Verilog
- ▶ 5. Procedural Assignment
- ▶ 6. Timing controls
- ▶ 7. Combinatorial Logic in Verilog
- ▶ 8. Race conditions
- ▶ 9. Summary

Sequential Blocks

reg a,b,c,d;

initial

begin

a = 1`b0;

b = 1`b1;

c = a;

d = b;

end

All statements within the block runs sequentially

a = 0

b = 1

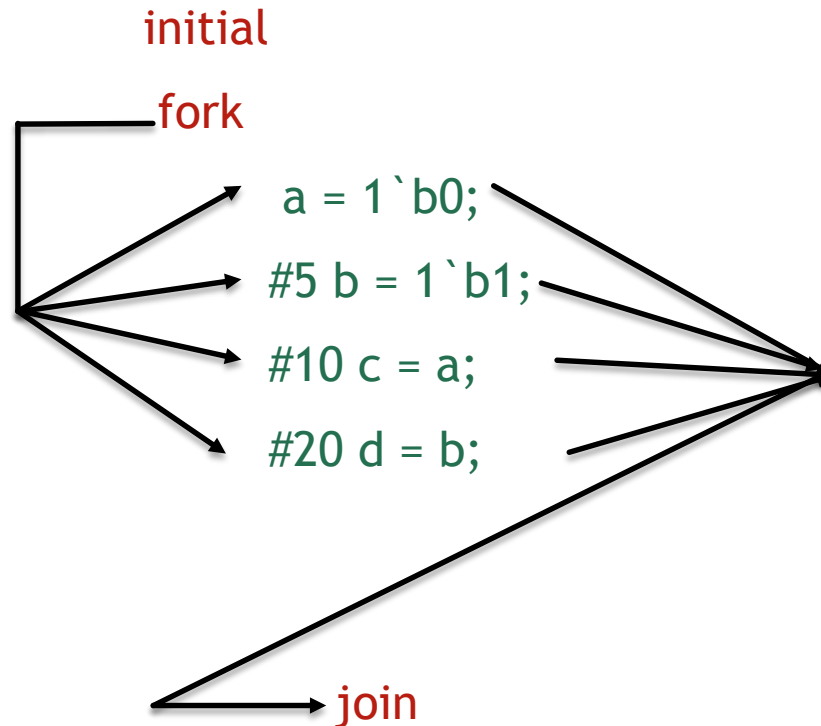
c = 0

d = 1

Parallel Blocks

All statements after fork runs in parallel

► Fork and Join



a = 0

After 5 cycle b = 1

After 10 cycle c = 0

After 20 cycle d = 1

What happens without delay??

RACE !!!

Outline

- ▶ 1. Motivation
- ▶ 2. Basic Syntax
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. **Conditions and Loops in Verilog**
- ▶ 5. Procedural Assignment
- ▶ 6. Timing controls
- ▶ 7. Combinatorial Logic in Verilog
- ▶ 8. Race conditions
- ▶ 9. Summary

Conditional statement

► If condition

Type1

if (<expression>) true statement ;

Type2

if (<expression>) true statement; **else** false statement;

Type 3

if(<expression1>) true statement1;
else if (expression2>) true statement2 ;
else if (<expression3>) true statement3;
else default_statement;

Multi way branching

- ▶ Keywords : case, endcase, default

```
case (expression)
    alternative1: statement1;
    alternative2: statement2;
    alternative3: statement3;
    ...
    ...
    default: default statement;
endcase
```

While loop

initial

begin

count = 0;

while (count < 128)

begin

\$display("Count = %d", count);

count = count +1;

end

end

For loop

initial

```
for ( count= 0; count <128; count = count+ 1)  
    $display (“count = %d”, count);
```

repeat

initial

begin

count = 0;

repeat(128)

begin

\$display("count = %d", count);

count = count + 1;

end

end

forever

```
initial
begin
    clock = 1`b0;
    forever #10 clock = ~clock;
end
```

- ▶ Use forever loop instead of always. (will be clarified in the next chapter)

Outline

- ▶ 1. Motivation
- ▶ 2. Basic Syntax
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. Conditions and Loops in Verilog
- ▶ **5. Procedural Assignment**
- ▶ 6. Timing controls
- ▶ 7. Combinatorial Logic in Verilog
- ▶ 8. Race conditions
- ▶ 9. Summary

Structured procedures

Initial

- ▶ Initial - process exactly once

```
initial
begin
    a = 1;
    #1;
    b=a;
end
```

Always

- ▶ Always - rescheduled many times

```
always @ (a or b )
begin
    if (a)
        c=b;
    else
        d= ~b;
end
```


Structured procedures

- ▶ Initial forever

initial forever

begin

clk = 0;

#1;

clk = 1;

#1;

end

- ▶ All behavioural statements can appear only within structured procedures
- ▶ Nesting between initial and always is not possible

Procedural Assignment

- ▶ Blocking Assignment (=)
 - ▶ Used for purpose of logic
 - ▶ Used for sequential execution of statements
- ▶ Non- Blocking Assignment (<=)
 - ▶ The simulator can schedule any statement non deterministically within a block
 - ▶ switch values without using temporary storage variables.

Blocking Assignment

▶ Example:

initial

begin

x=0;

y= 1;

z=0;

#15 count = count + 1;

count = 1;

end

▶ Values

▶ At time 0 : X=0, Y=1 and Z=0

▶ At time 15 :
initially count is assigned a junk value and then it changes to 1 in the next assignment

Non- Blocking Assignment

▶ Example

initial

begin

$x \leq 0;$

$y \leq 1;$

$z \leq 0;$

#15 $\text{count} \leq \text{count} + 1;$

$\text{count} \leq 1;$

end

▶ Values

▶ At time 0 : $x = 0; y = 1; z = 0; \text{count} = 1$

▶ At time 15 : $\text{count} \leq 2$

Uses of Non Blocking Assignment

- ▶ Values for flop1 and flop2 ..?

```
always @ (posedge clock)
```

```
    flop1 = flop2;
```

```
always @ (posedge clock)
```

```
    flop2 = flop1;
```

```
always @ (posedge clock)
```

```
begin
```

```
    flop <= flop2;
```

```
    flop2 <= flop1;
```

```
end
```

Outline

- ▶ 1. Motivation
- ▶ 2. Basic Syntax
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. Conditions and Loops in Verilog
- ▶ 5. Procedural Assignment
- ▶ 6. **Timing controls**
- ▶ 7. Combinatorial Logic in Verilog
- ▶ 8. Race conditions
- ▶ 9. Summary

Delay based timing control

- ▶ Specifies the time duration between when a statement is encountered and when it is executed.
- ▶ Delay can be specified by

`#Number` `statement;`

`#identifier` `statement;`

`#(expression)` `statement;`

- ▶ Regular delay control
- ▶ Intra assignment delay control
- ▶ Zero delay control

Regular delay control

```
parameter latency = 20;
```

```
parameter delta = 2;
```

```
reg a, b, c, d;
```

```
initial
```

```
begin
```

```
    a = 0;
```

```
    #10 b = 1;
```

```
    #latency c = 0;
```

```
    #( latency + delta ) d = 1;
```

```
end
```


Intra assignment delay control

- ▶ Assign delay to the right of the assignment operator
- ▶ Alters the flow of activity in a different manner

initial

begin

$x = 0; z = 0;$

$y = \#5x + z;$

end

Zero delay control

- ▶ Statements are executed last within the same simulation time

```
initial
begin
    x=0;
    y=0;
end
```

```
initial
begin
    #0 x=1;
    #0 y=1;
end
```

Used to eliminate race condition

Event based timing control

- ▶ Event can be change in value of a register or net
- ▶ The @ symbol is used to specify an event control
- ▶ Statements can be executed at a positive or negative transition of a signal value

@ (clock) q = d;

@ (posedge clock) q = d;

@ (negedge clock) q = d;

q = @ (posedge clock) d;

Level sensitive timing control

- ▶ The ability to wait for a certain condition to be true
- ▶ The keyword **wait** is used for level sensitive constructs.

always

wait (count_enable) #20 count = count + 1;

Outline

- ▶ 1. Motivation
- ▶ 2. Basic Syntax
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. Conditions and Loops in Verilog
- ▶ 5. Procedural Assignment
- ▶ 6. Timing controls
- ▶ **7. Combinatorial Logic in Verilog**
- ▶ 8. Race conditions
- ▶ 9. Summary

Combinatorial logic in Verilog

- ▶ Combinatorial logic vs Sequential logic
- ▶ Blocking assignments which happen sequentially are used for combinational logic
- ▶ Non-Blocking assignments which happen in parallel are used for sequential logic
- ▶ Assign keyword for blocking statements

Outline

- ▶ 1. Motivation
- ▶ 2. Basic Syntax
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. Conditions and Loops in Verilog
- ▶ 5. Procedural Assignment
- ▶ 6. Timing controls
- ▶ 7. Combinatorial Logic in Verilog
- ▶ **8. Race conditions**
- ▶ 9. Summary

Race Conditions

- ▶ Verilog does not guarantee order of execution

```
initial
```

```
    a=0;
```

```
initial
```

```
    b=a;
```

```
initial
```

```
begin
```

```
    #1;
```

```
    $display("Value a=%a Value of b=%b",a,b);
```

```
end
```

Possible values for a and b ..?

Outline

- ▶ 1. Motivation
- ▶ 2. Basic Syntax
- ▶ 3. Sequential and Parallel Blocks
- ▶ 4. Conditions and Loops in Verilog
- ▶ 5. Procedural Assignment
- ▶ 6. Timing controls
- ▶ 7. Combinatorial Logic in Verilog
- ▶ 8. Race conditions
- ▶ **9. Summary**

Summary

- ▶ Complex logic simulation
- ▶ Easier to develop and debug circuits
- ▶ Easy to code and learn (constructs similar to programming languages)

So its now time for demo!

Example - $x=ax+y$

```
module test();
```

```
initial
```

```
begin
```

```
$dumpfile("myfirsttask1.vcd");
```

```
    $dumpvars(0,test);
```

```
end
```

```
reg [3:0] x;
```

```
reg [3:0] a;
```

```
reg [3:0] y;
```

```
initial begin
```

```
    x = 4'd1;
```

```
    a = 4'd1;
```

```
    y = 4'd2;
```

```
$display( "x= ax+y");
```

```
$monitor("%g x = %b a = %b y = %b",  
$time, x, a, y);
```

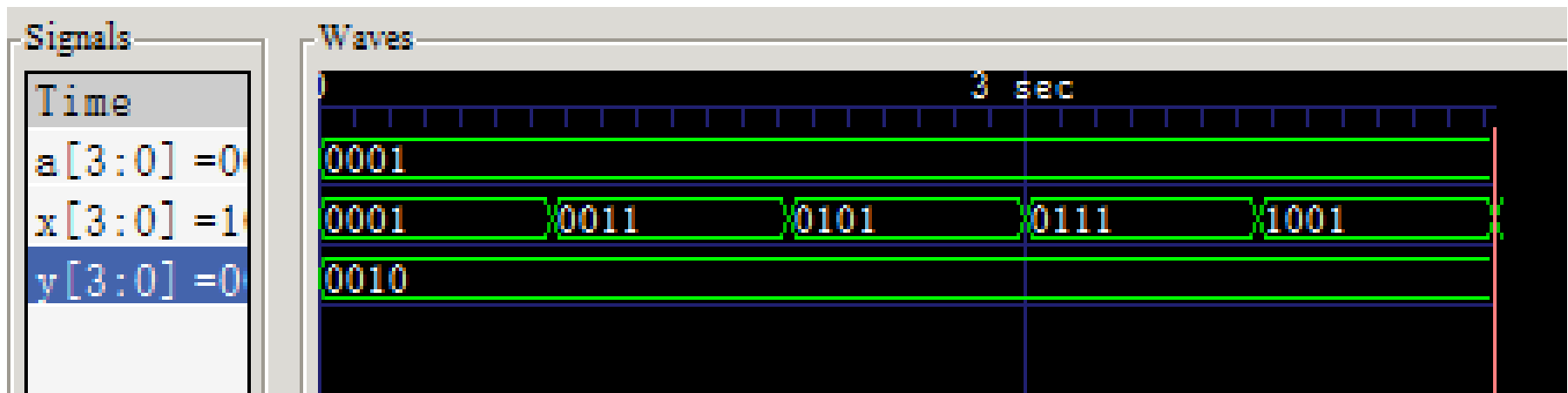
```
#5 $finish;
```

```
end
```

```
always
```

```
#1 x = x*a +y;
```

```
endmodule
```



Modules

```
module test;
reg clk, reset;
wire [3:0] q;

ripple_carry_counter rcc(q, clk,
reset);

initial begin
$dumpfile("dump.vcd");
$dumpvars(0, test);
clk = 1'b0;
reset = 1'b1;
#20 reset = 1'b0;
#200 reset = 1'b1;
#30 reset = 1'b0;
#30;
$finish;
end

always #5 clk = ~clk;

endmodule
```

```
module tff(q, clk, reset);
output reg q;
input clk, reset;

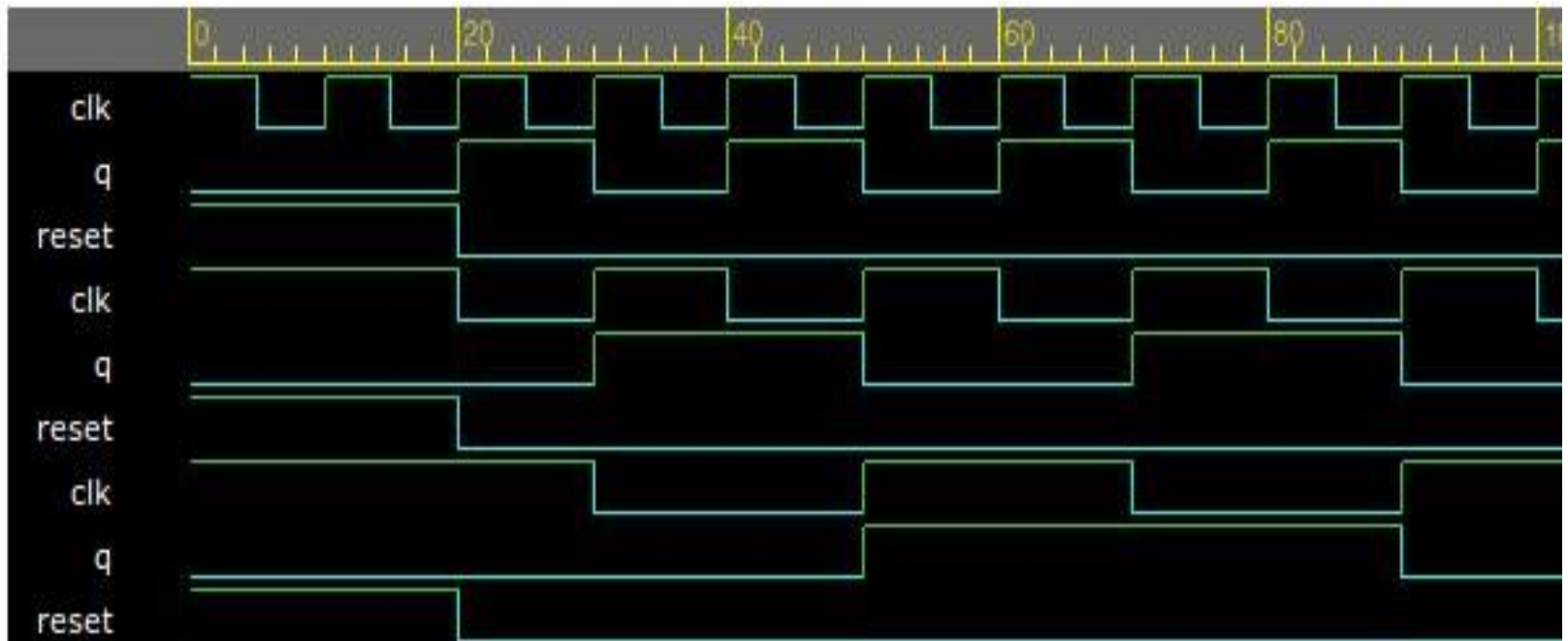
always@ (posedge reset or posedge clk)
begin
if(reset)begin
q<=1'b0;
end else begin
q<= ~q;
end
end

endmodule

module ripple_carry_counter(q, clk, reset);
output[3:0] q;
input clk, reset;

tff tff0(q[0], ~clk, reset);
tff tff1(q[1], ~q[0], reset);
tff tff2(q[2], ~q[1], reset);
tff tff3(q[3], ~q[2], reset);

endmodule
```



References

- ▶ Verilog Hardware Descriptive Language 5th edition, Donald Thomas, Philip Moorby, 2002, Kluwer Academic.
- ▶ Verilog HDL, A guide to digital design and synthesis, Samir Palnitkar, Sun Soft Press
- ▶ Verilog HDL Synthesis (A practical primer), J Bhasker, Star galaxy publishing
- ▶ Verilog Non Blocking assignment with Delays, Myths and Mysteries, Clifford E. Cummings, 2002, Sunburst Design Inc, 1-18, 44-47
- ▶ Graphics source Wikipedia.org
- ▶ <http://www.edaplayground.com/>

Credits

- ▶ Deepjyoti Borah
 - ▶ Need for Verilog
 - ▶ Sequential and Parallel blocks
 - ▶ Conditions and Loops in Verilog
 - ▶ DEMO - Ripple carry counter 4 bit
- ▶ Diwahar Jawahar
 - ▶ Behaviour modelling in Verilog
 - ▶ Event and delay based timing controls
 - ▶ Procedural assignment and Structured procedures
 - ▶ Combinatorial Logic in Verilog & DEMO

Questions ?

Thank you!