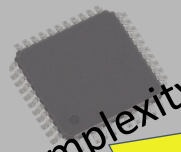
Digital Design with Verilog

Lecture 4: Digital Design with Verilog

Vacuum Tube

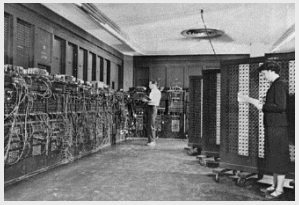
Discrete Transistor

Integrated Circuit



Clock Frequency Scaling, design complexity

Electronic Design Automation



ENIAC

Tx-0

Apollo Guidance Computer

Xerox Alto

Apple II

Intel 486DX

Intel Pentium

Intel Pentium 4

Transistor density increase according to Moore's law

*Images from different sources on internet

Evolution of Computer Aided Digital Design

- Small Scale Integration (SSI)
  - Less than 10 transistors
- Medium Scale Integration (MSI)
  - In between 10-100 transistors
- Large Scale Integration (LSI)
  - few thousands of transistors
- Very Large Scale Integration (VLSI)
  - More than 1,00,000 transistors
- Complexity increased over the time and Electronic Design Automation tools are necessary for the design of integrated circuits.
  - Front end tools
  - Back end tools

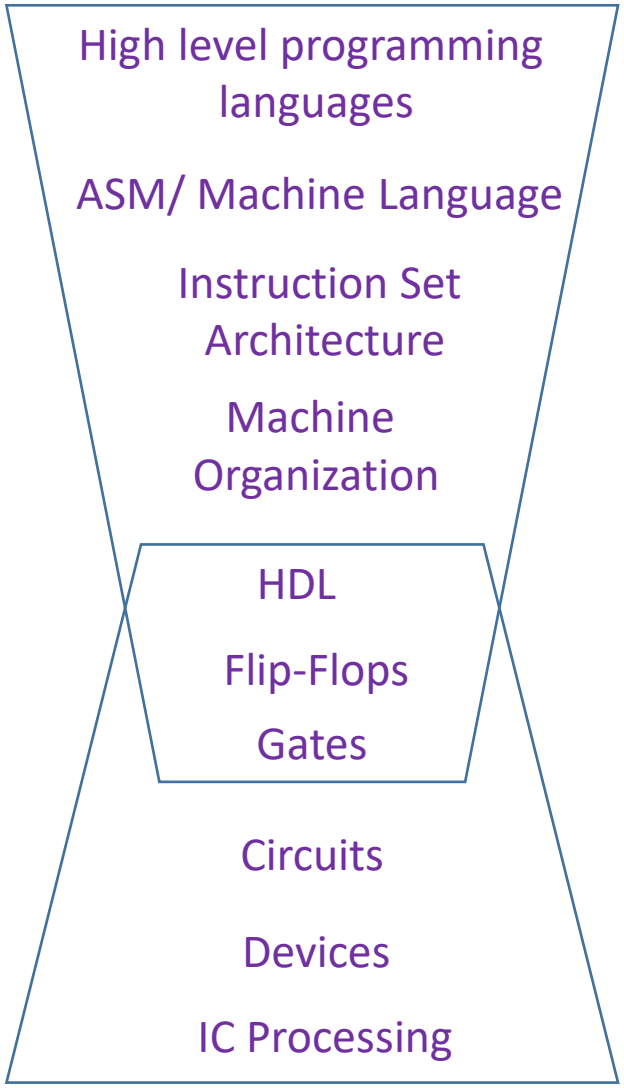
Emergence of HDLs

- Need for a standard language to describe the digital circuits
  - FORTRAN, Pascal, and C were being used to describe computer programs that were sequential in nature.

- Hardware Description Languages (HDLs) came into existence which can allow the designers to model the concurrency of processes found in the hardware.
  - Verilog HDL and VHDL

- Digital circuits are described at a “register transfer level (RTL)” by use of an HDL.
  - The designer has to specify how the data flows between registers and how the design process the data. Logic synthesis tool extracts the gates and their interconnections to implement the circuit.

Design Abstraction Levels

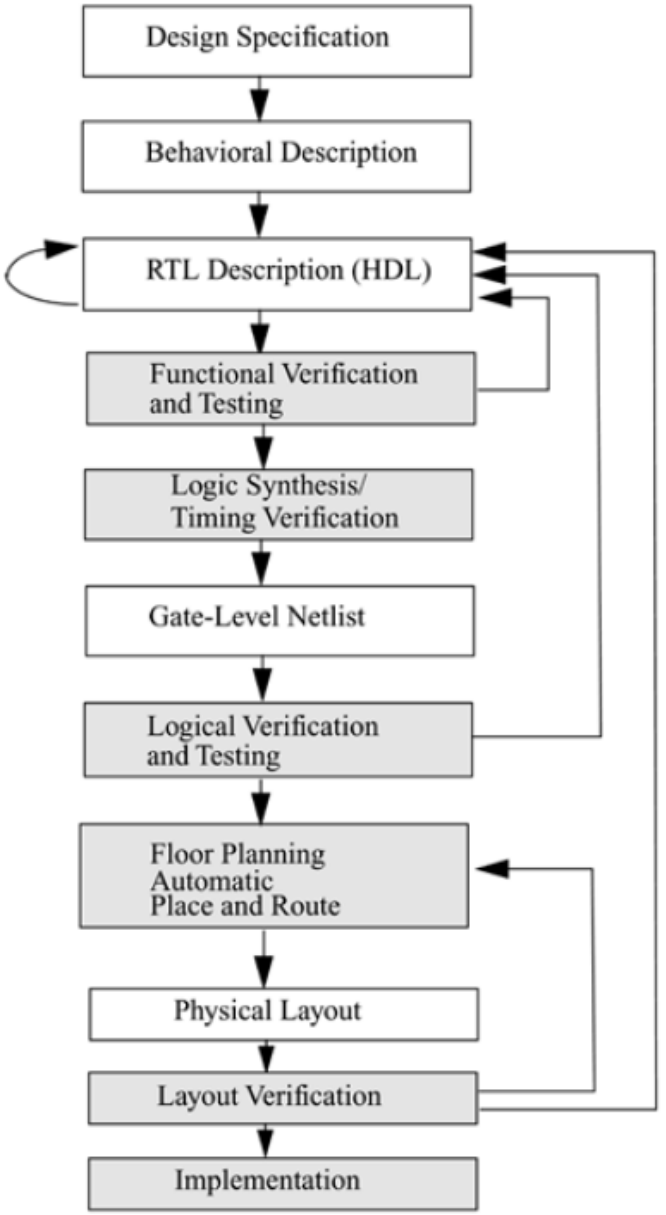


Design Abstraction Levels

What is Verilog HDL?

- A high level language that can model, represent, and simulate the digital circuits.
  - Hardware concurrency
  - Parallel activity flow
  - Different levels of abstraction to be mixed in the same model
- Design examples using Verilog HDL
  - Intel Pentium, ARM7, and many more.
- VHDL represents another high level language for digital system design.
- In this course, we study Verilog HDL
  - Reason: Most widely used in the industry and easy to learn, syntax is similar to C

Typical Design Flow of ICs



Note: Shaded blocks denote the processes in the design flow whereas unshaded blocks show the level of design representation.

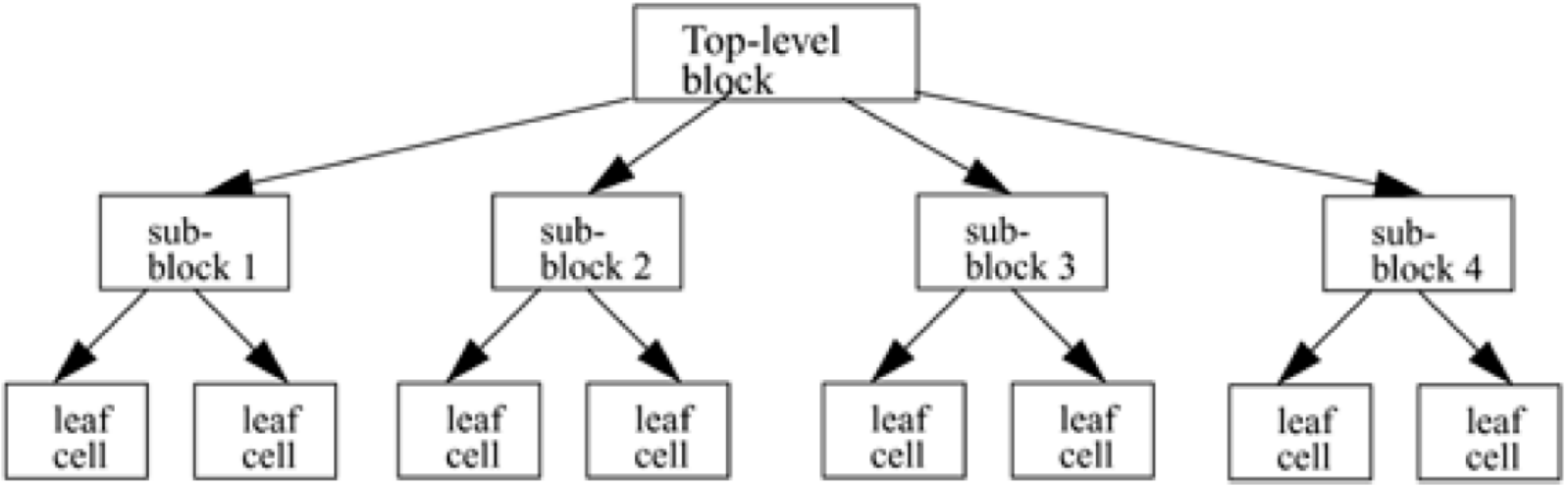
Importance of HDLs

- Designs can be described at a very abstract level using an HDL without considering the fabrication details
  - Logic synthesis tools create the netlist which is dependent on the technology.
- Functional verification can be done in the early design cycle.
  - Reduced design cycles
- Designing with HDL is like writing computer programs with comments which can provide a concise representation of the design, compared to a gate-level schematics.

Hierarchical Modeling Concepts

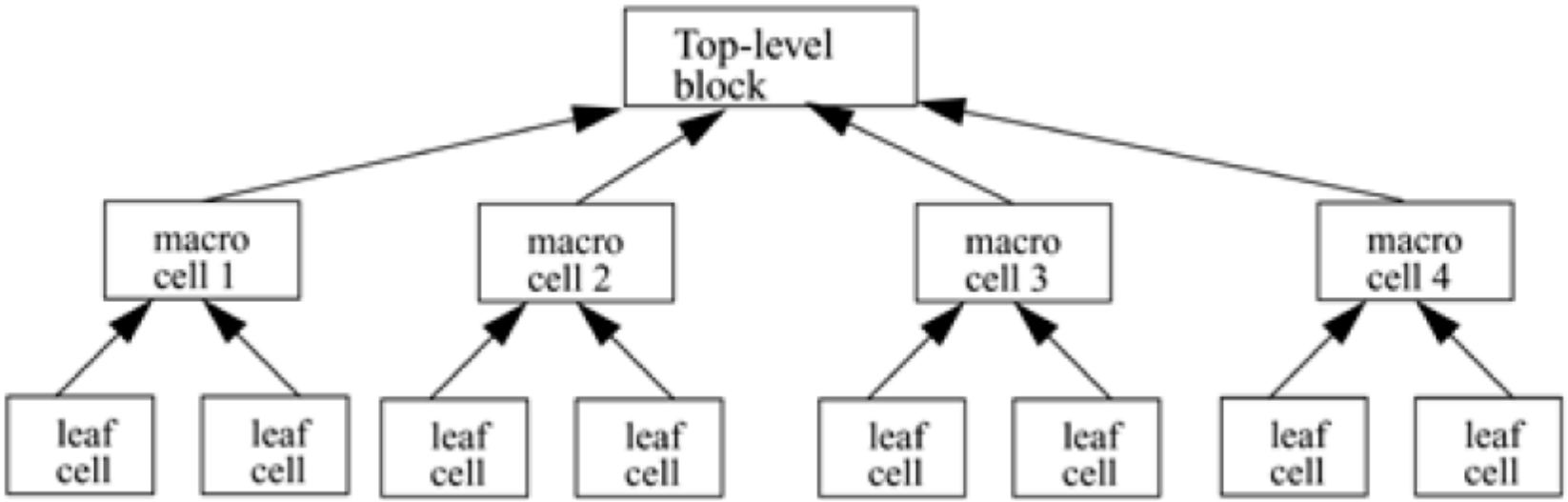
Design Methodologies

- Top-down approach



Design Methodologies

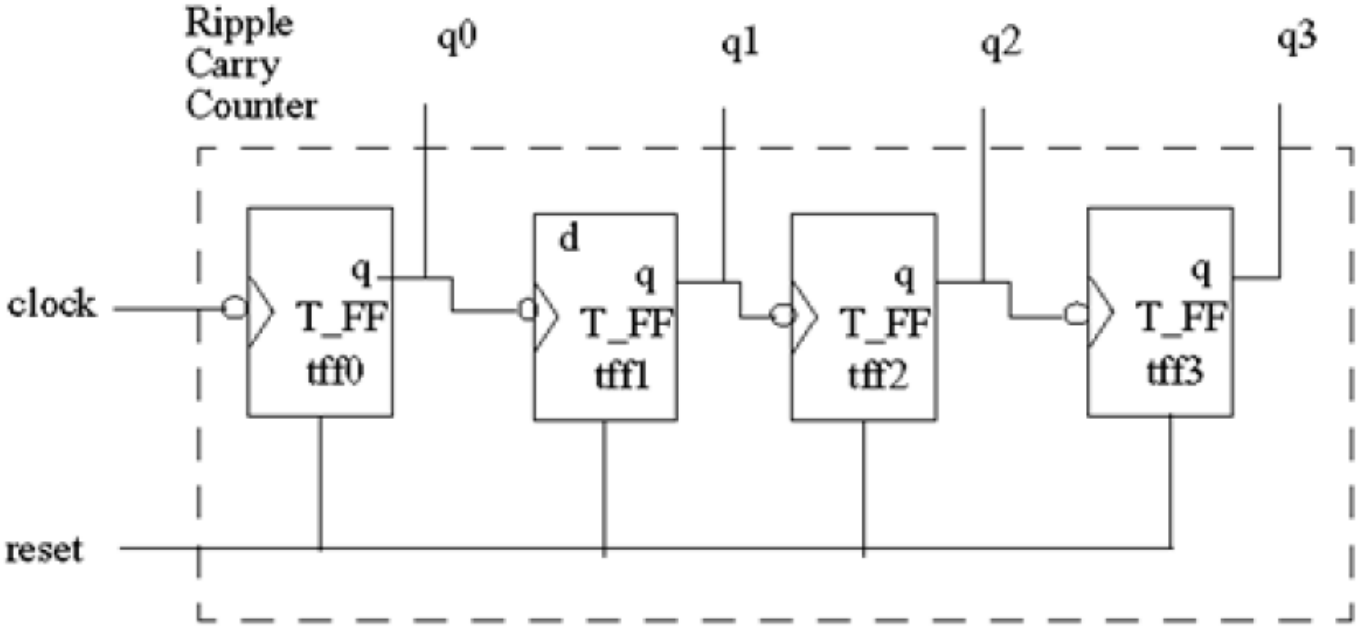
- Bottom-Up approach

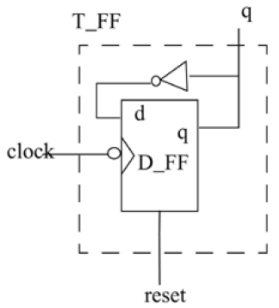


Typically, a combination of top-down and bottom-up flow is used.

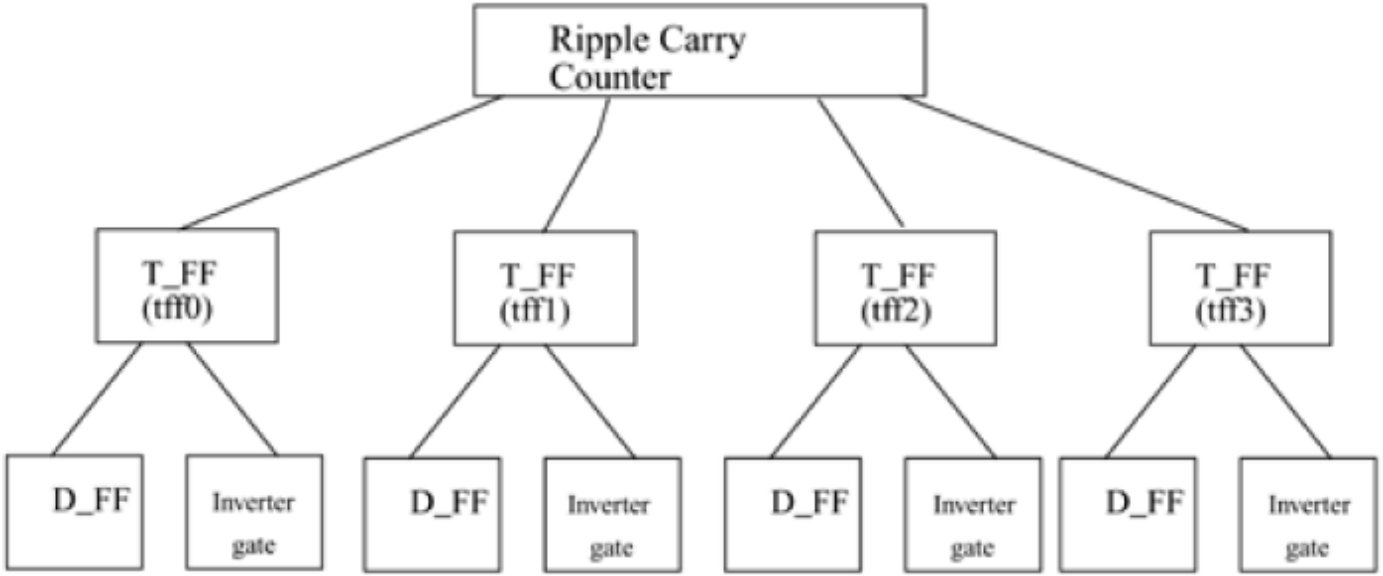
Example:

- 4-bit Ripple Carry Counter





Ripple Carry Counter: Design Hierarchy



Modules

- Module is the basic building block in Verilog.
  - Element or collection of lower-level design blocks
  - Communicates to the higher level blocks through its port interface (inputs and outputs)
  - Hides the internal implementation details
    - Allows the designer to modify the module internals without affecting the rest of the design.

```
module <module_name> (<module_terminal_list>);
...
<module internals>
...
...
endmodule
```

```
module T_FF (q, clock, reset);
.
.
<functionality of T-flipflop>
.
.
endmodule
```

Modules (Cont'd)

- Internals of the module can be defined at four levels of abstraction
  - Behavioral or algorithmic level
    - Similar to C program
    - Highest level of abstraction
  - Dataflow level
    - Designed according to the flow of data between registers and how the data is processed in the design
  - Gate level
    - Similar to describing a design in terms of a gate level logic diagram
  - Switch level
    - Implemented in terms of switches, storage nodes, and the interconnections between them.
    - Lowest level of abstraction
- Verilog allows to mix and match all four levels of abstractions in a design.

Example: Levels of abstraction-Behavioural

```
module mux2_to_1 (out, s, i0, i1);
    // Port declarations from the I/O diagram
    output out;
    input i0, i1;
    input s;
    reg out;

    always @(s or i0 or i1)
        case (s)
            1'd0 : out = i0;
            1'd1 : out = i1;
            default: $display("Invalid control signals");
        endcase
endmodule
```

Example: Levels of abstraction-Data flow

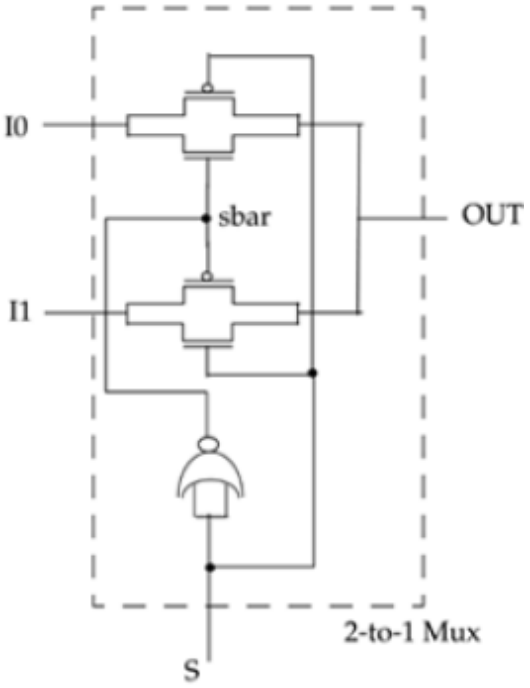
```
module mux2_to_1 (out, s, i0, i1);
    // Port declarations
    output out;
    input i0, i1;
    input s;

    assign out = s ? i1: i0;
endmodule
```

Example: Levels of abstraction-Gate Level

```
module mux2_to_1 (out, s, i0, i1);
    output out;
    input i0, i1;
    input s;

     wire y0, y1, y2, y3;// Internal wire declarations
    // Gate instantiations
    // Create sn signal.

    not (sn, s);

    // 3-input and gates instantiated
    and (y0, i0, sn);
    and (y1, i1, s);

    //2-input or gate instantiated
    or (out, y0, y1);

endmodule
```

Example: Levels of abstraction-Switch Level

```
module mux2_to_1 (out, s, i0, i1);

    output out;
    input s, i0, i1;

    //internal wire
    wire sbar; //complement of s

    //equivalent to a not gate
    my_nor nt(sbar, s, s);

    //instantiate cmos switches
    cmos (out, i0, sbar, s);
    cmos (out, i1, s, sbar);

endmodule
```



Instances

- Module is just a template from which actual objects are created.
  - Specify how the module will work, and its interface; modules must be instantiated for use in the design.
- Unique object for each invocation of module
- Each object has its own---name, Variables, parameters, and I/O interface
- *Instantiation* is nothing but the process of creating objects from a module template, and the objects are called “instances”.

Example: Instantiation

```
// Define the top-level module called ripple carry
// counter. It instantiates 4 T-flipflops.
```

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

    output [3:0] q;
    input clk, reset;

    // Four instances of the module T_FF are created.
    // Each has a unique name.
    // Each instance is passed a set of signals.
    // Notice, that each instance is a copy of the module T_FF.

    T_FF tff0(q[0],clk, reset);
    T_FF tff1(q[1],q[0], reset);
    T_FF tff2(q[2],q[1], reset);
    T_FF tff3(q[3],q[2], reset);
endmodule
```

Example: Instantiation

//Define the module T_FF. It instantiates a D-flipflop. We assumed that module D-flipflop is defined elsewhere in the design.

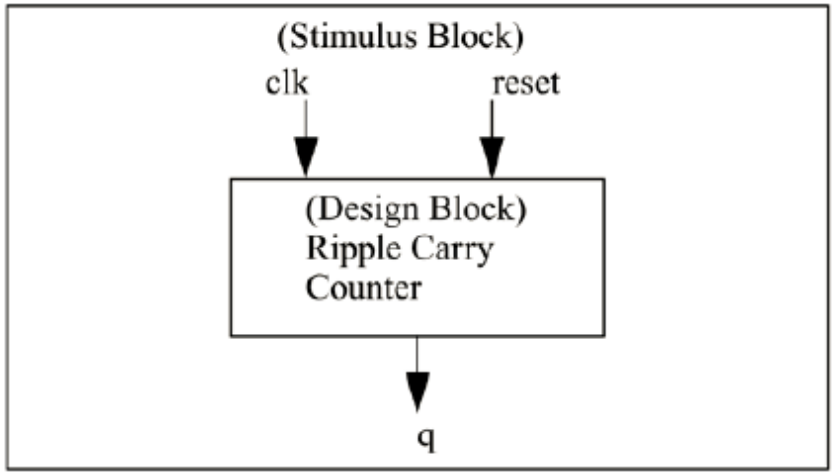
```
module T_FF(q, clk, reset);
    output q;
    input clk, reset;
    wire d;

    // Instantiate D_FF. Call it dff0.
    D_FF dff0(q, d, clk, reset);

    // not gate is a Verilog primitive.
    not n1(d, q);

endmodule
```

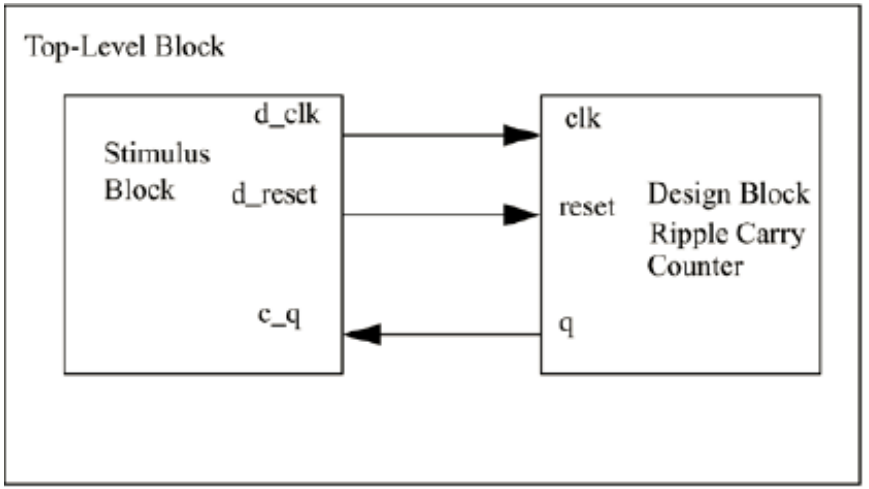
Example: D Flip-Flop

```
// module D_FF with synchronous reset
module D_FF(q, d, clk, reset);
    output q;
    input d, clk, reset;
    reg q;
    //Lots of new constructs. Ignore the functionality
    // of the constructs for now. Concentrate on how the
    // design block is built in a top-down fashion.

    always @(posedge reset or negedge clk)
        if (reset)
            q <= 1'b0;
        else
            q <= d;
endmodule
```

Components of a Simulation

- Functionality of the design has to be tested by applying stimulus and checking the results
  - Keep the stimulus and design block always separate.
- Verilog itself can be used to write the stimulus
  - Stimulus block is also often called as “test bench”
- Two styles of stimulus applications are possible

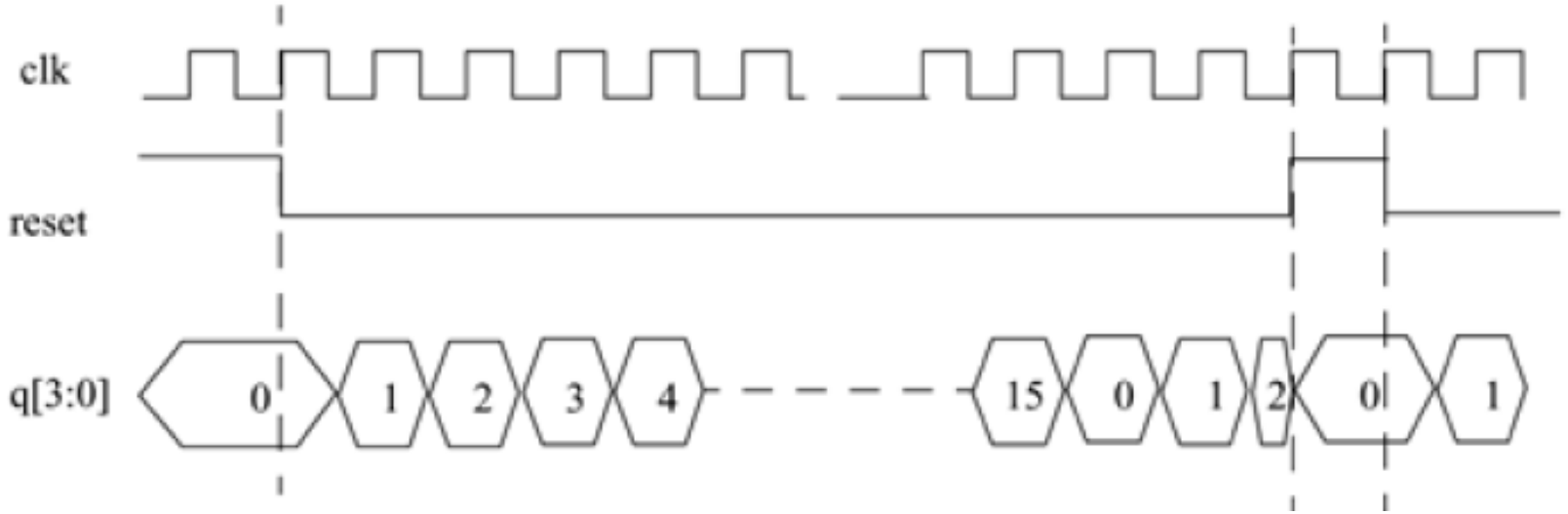




Example: Test bench

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

    // instantiate the design block
    ripple_carry_counter r1(q, clk, reset);
    // Control the clk signal that drives the design block.
    //Cycle time = 10

    initial
        clk = 1'b0; //set clk to 0
    always
        #5 clk = ~clk;//toggle clk every5 time units

    // Control the reset signal that drives the design block
    // reset is asserted from 0 to 20 and from 200 to 220.
    initial
    begin
        reset = 1'b1;
        #15 reset = 1'b0;
        #180 reset = 1'b1;
        #10 reset = 1'b0;
        #20 $finish; //terminate the simulation
    end
    // Monitor the outputs
    initial
    $monitor($time, " Output q = %d", q);
endmodule
```

Example: Counter Simulation Output



Example: Counter Simulation Output

```
  0 Output q =  0
 20 Output q =  1
 30 Output q =  2
 40 Output q =  3
 50 Output q =  4
 60 Output q =  5
 70 Output q =  6
 80 Output q =  7
 90 Output q =  8
100 Output q =  9
110 Output q = 10
120 Output q = 11
130 Output q = 12
140 Output q = 13
150 Output q = 14
160 Output q = 15
170 Output q =  0
180 Output q =  1
190 Output q =  2
195 Output q =  0
210 Output q =  1
220 Output q =  2
```

Demo

References

- Chapter 1 & 2 Verilog HDL by Samir Palnitkar, Second Edition.