Verilog

- Gate and Switch Level Modeling -

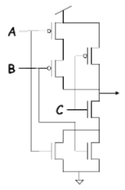
May 2014

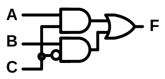
Ando KI

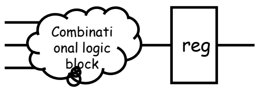
Contents

Levels of abstraction

- Structural levels
  - Switch level
    - Referring to the ability to describe the circuit as a netlist of transistor switches.
  - Gate level
    - Referring to the ability to describe the circuit as a netlist of primitive logic gates and functions.
- Functional levels
  - Register transfer level
    - Referring to the ability to describe the circuit as data assignment.
  - Behavioral level
    - Referring to the ability to describe the behavior of circuit using abstract constructs such as loops and processes.







Levels of modeling

- Analog-transistor level
  - Uses an electronic model of circuit elements such as resistor, capacitor, and inductor.
  - Allows analog values of voltages or currents to represent logic values on the interconnections.
- Switch level
  - Describes the interconnection of transmission gates which are abstractions of individual MOS and CMOS transistors.
- Logic level
  - Describes a digital circuit in terms of primitive logic functions such as AND, OR, NAND, and NOR.
  - Allows for the nets interconnecting the logic functions to carry 0, 1, x, and z.

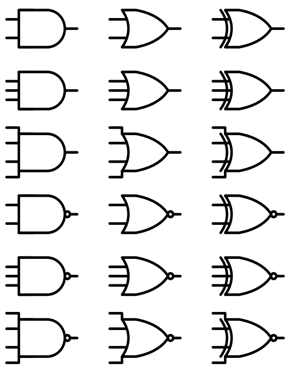
Built-in gates and switches

n_input gates	n_output gates	three-state gates	pull gates	MOS switches	bidirectional switches
and	buf	bufif0	pulldown	cmos	rtran
nand	not	bufif1	pullup	nmos	rtranif0
nor		notif0		pmos	rtranif1
or		notif1		rcmos	tran
xnor				rnmos	tranif0
xor				rpmos	tranif1

Variable input/output logic gates

- The gates (and, nand, nor, or, xor, xnor) shall have one output and one or more inputs.
  - The first terminal shall be output.

```
and Uand (out, in1, in2);
or  Uor   (out, in1, in2, in3);
```



and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

nand	0	1	x	z
0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

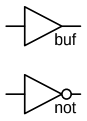
nor	0	1	x	z
0	1	0	x	x
1	0	0	0	0
x	x	0	x	x
z	x	0	x	x

xnor	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

Buffers

- Multiple output logic gates
  - buf: simply bypass
  - not: inverter

```
buf Ubuf (out1, in);
buf Ubuf (out1, out2, in);
```



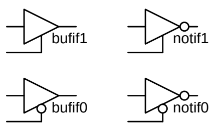
buf	
input	output
0	0
1	1
x	x
z	x

not	
input	output
0	1
1	0
x	x
z	x

Tri-state buffers

- Tri-state buffer gates
  - bufif0: buffer with active low control
  - bufif1: buffer with active high control
  - notif1: inverter buffer with active high control
  - notif0: inverter buffer with active low control

```
bufif1 Ubf1 (out, in, control);
```



bufif0	CONTROL				
		0	1	x	z
D	0	0	z	L	L
A	1	1	z	H	H
T	x	x	z	x	x
A	z	x	z	x	x

bufif1	CONTROL				
		0	1	x	z
D	0	z	0	L	L
A	1	z	1	H	H
T	x	z	x	x	x
A	z	z	x	x	x

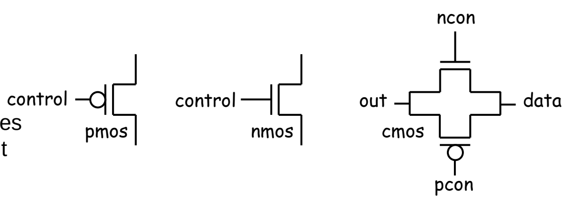
notif0	CONTROL				
		0	1	x	z
D	0	1	z	H	H
A	1	0	z	L	L
T	x	x	z	x	x
A	z	x	z	x	x

notif1	CONTROL				
		0	1	x	z
D	0	z	1	H	H
A	1	z	0	L	L
T	x	z	x	x	x
A	z	z	x	x	x

MOS switches

- MOS switches are <u>unidirectional channels</u> for data similar to the buff gates.
- Normal MOS
  - pmos
  - nmos
  - cmos
- Resistive MOS
  - It has significantly higher impedances between sources and drains when it conduct than pmos/nmos.
  - rpmos
  - rnmos
  - rcmos

```
pmos Up (out, data, control);
nmos Un (out, data, control);
cmos Uc (out, data, ncon, pcon);
```



pmos rpmos	CONTROL			
	0	1	x	z
D 0	0	z	L	L
A 1	1	z	H	H
T x	x	z	x	x
A z	z	z	z	z

nmos rnmos	CONTROL			
	0	1	x	z
D 0	z	0	L	L
A 1	z	1	H	H
T x	z	x	x	x
A z	z	z	z	z

Bidirectional pass switch

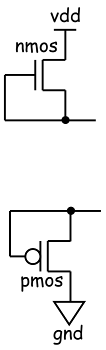
- Bidirectional pass switch
  - tran, rtan
  - tranif1, rtranif1
  - tranif0, rtranif0
- These pass switch shall <u>not delay signals propagating</u> through them.
- However, these can have <u>turn-on and turn-off delays</u>.

```
tran (inout1, inout2);
tranif1 (inout1, inout2, control);
```

Pullup and pulldown sources

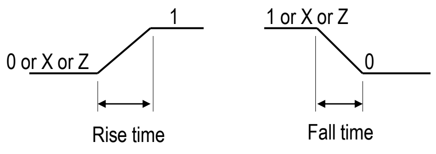
- A pullup source shall place a logic value 1 on the nets connected in its terminal list.
- A pulldown source shall place a logic value 0 on the nets connected in its terminal list.

```
pullup    Up1 (net1);
pulldonw Ud1 (net2);
```



Gate delay

- The gate delay specification can be zero, one, two, or three delays.
  - If there is no delay specification, there shall be no propagation delay through the gate.
    - 'and Uand(out, in1, in2);'
  - If there is one delay specification, it shall apply to both the rise and fall delay.
    - 'and #(3) Uand(out, in1, in2);'
  - If there are two delays, the first specifies rise delay and the second specifies fall delay.
    - 'and #(3, 5) Uand(out, in1, in2);'
  - If there are three delays, the last one is for turn-off delay.
    - Logic gates do not have the third delay.
    - Buffer gates can have this.
- #(1st delay[, 2nd delay[, 3rd delay]])
  - The first delay: rise delay
  - The second delay: fall delay
  - The third delay: turn-off delay
    - The transition to the high-impedances value.
- Each delay can be min:typ:max values
  - 'and #(1:2:3,2:1:3) U(y, x, z);'



Gate instantiation

```
GateType [strength] [delay] ListOfGateInstances;
```

```
and UA (out, in1, in2);
and UB (out1, in1, in2);
```

```
and UC (out1, in1, in2), UD (out2, in1, in2);
```

Note comma separation

```
and      (out3, in1, in2, in3), (out4, in1, in2);
```

Note there are no instance name.

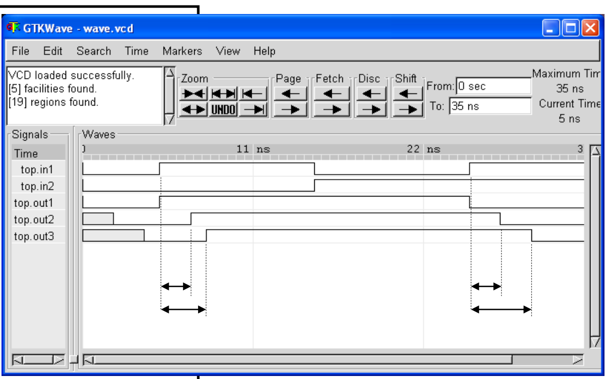
- ‘GateType’ is one of ‘and, or, …, buf, …’.
- ‘ListOfGateInstances’ is a comma-separated list which specifies the terminals of each gate and optionally names each instance.
- ‘delay’ will be #(r, f, t) form, where rise, fall, and transition (or turn-off).
- ‘strength’ specifies strength of driving, such as supply, strong, pull, and weak.

Example of gate delay (1/2)

DIY

- ‘codes/verilog/gate_switch/gate_delay’

```
`timescale 1ns/1ns
module top;
  reg  in1, in2;
  wire out1, out2, out3;
  xor            UxorA (out1, in1, in2);
  xor  #(2)      UxorB (out2, in1, in2);
  xor  #(3, 4) UxorC (out3, in1, in2);
  initial begin
        in1 = 0; in2 = 0; #5;
        in1 = 1; in2 = 0; #10;
        in1 = 0; in2 = 1; #10;
        in1 = 1; in2 = 1; #10;
        in1 = 0; in2 = 0; #5;
        $finish;
  end
  initial begin
     $dumpfile("wave.vcd");
     $dumpvars(1, in1, in2, out1, out2, out3);
  end
endmodule
```



```
VCD loaded successfully.
[5] facilities found.
[19] regions found.
```

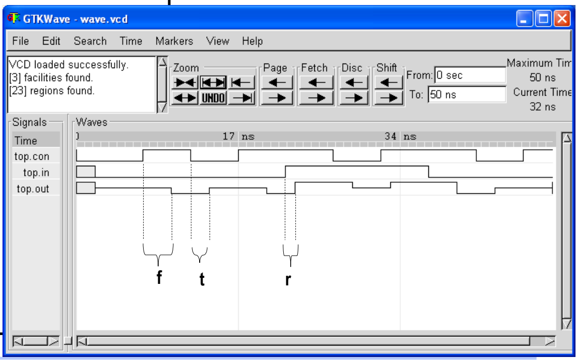
From: 0 sec
To: 35 ns
Maximum Tim
35 ns
Current Time
5 ns

Example of gate delay (2/2)

‘codes/verilog/gate_switch/buf_delay’

```
`timescale 1ns/1ns
module top;
  reg  in, con;
  wire out;
  bufif1 #(1, 3, 2) Ub (out, in, con); // r, f, t
  initial begin
        con = 0; #2;
        in  = 0; #5;  con = 1; #5; con = 0; #5; con = 1; #5;
        in  = 1; #5;  con = 0; #5; con = 1; #5;
        in  = 0; #5;  con = 0; #5; con = 1; #5;
        $finish;
  end
  initial begin
     $dumpfile("wave.vcd");
     $dumpvars(1, in, out, con);
  end
endmodule
```

DIY



```
VCD loaded successfully.
[3] facilities found.
[23] regions found.
```

From: 0 sec
To: 50 ns
Maximum Time 50 ns
Current Time 32 ns

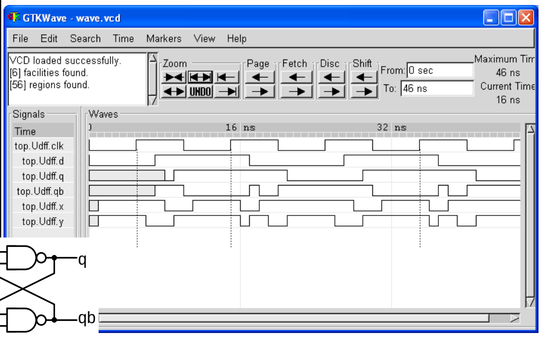
Example of gate-level model

‘codes/verilog/gate_switch/dff_gate’

```
`timescale 1ns/1ns
module top;
  reg d, clk;
  wire q, qb;
  initial begin
     clk = 0; forever #5 clk = ~clk;
  end
  initial begin
     d = 0; #7;  d = 1; #10;
     d = 0; #10; d = 1; #10;
     d = 0; #10;
     $finish;
  end
  dff Udff (q, qb, d, clk);
endmodule
module dff (q, qb, d, clk);
  output q, qb;
  input  d, clk;
  wire   x, y;
  nand #1 U1 (q, x, qb);  nand #1 U2 (qb, q, y);
  nand #1 U3 (x, d, clk); nand #1 U4 (y, x, clk);
  initial begin
    $dumpfile("wave.vcd");
    $dumpvars(1, clk, d, q, qb);
    $dumpvars(1, x, y);
  end
endmodule
```

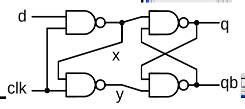
•Guess what will happen if the NAND gates have no delay.

DIY



```
VCD loaded successfully.
[6] facilities found.
[56] regions found.
```

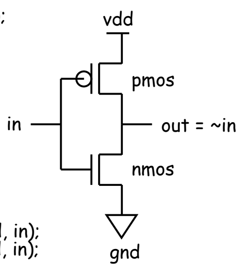
From: 0 sec
To: 46 ns
Maximum Time 46 ns
Current Time 16 ns



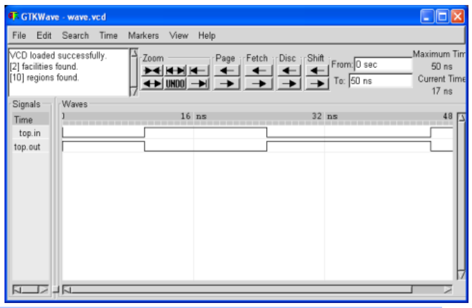
Example of switching-level model

- ‘codes/verilog/gate_switch/inverter_switch’

```
`timescale 1ns/1ns

module top;
   reg  in;
   wire out;
   inverter Uinv (out, in);

   initial begin
         in = 0; #10;
         in = 1; #15;
         in = 0; #20;
         in = 1; #5
         $finish;
   end
   initial begin
      $dumpfile("wave.vcd");
      $dumpvars(1, out, in);
   end
endmodule

module inverter (out, in);
       output out;
       input  in;
       //-------------------
       supply1 vdd;
       supply0 gnd;
       //-------------------
       pmos Upmos (out, vdd, in);
       nmos Unmos (out, gnd, in);
endmodule
```



DIY

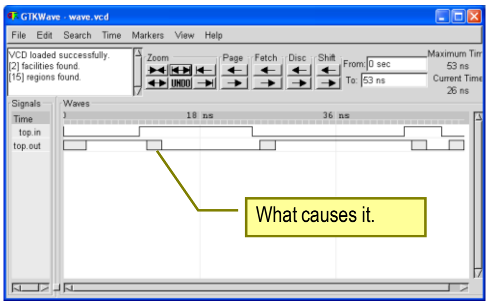


VCD loaded successfully.
[2] facilities found.
[10] regions found.

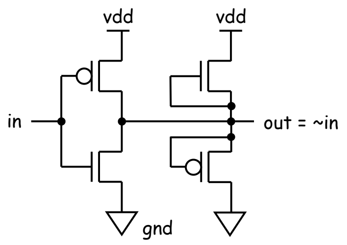
Example of pullup/pulldown (1/2)

- ‘codes/verilog/gate_switch/inverter_pull’

```
`timescale 1ns/1ns

module top;
  … …
endmodule

module inverter (out, in);
      output out;
      input  in;
      //-------------------
      supply1 vdd;
      supply0 gnd;
      //-------------------
      pmos #(1) Upmos (out, vdd, in);
      nmos #(3) Unmos (out, gnd, in);
      //-------------------
      pullup    pu (out);
      pulldown pd (out);
endmodule
```



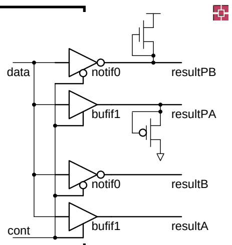
VCD loaded successfully.
[2] facilities found.
[15] regions found.



DIY

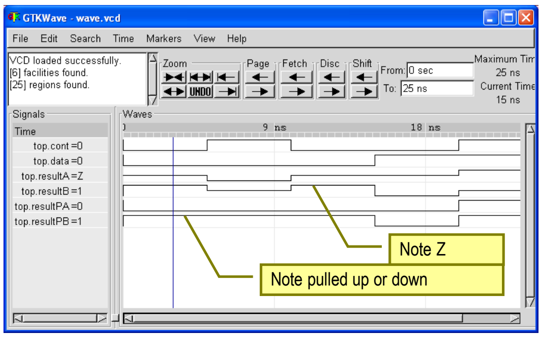
Example of pullup/pulldown (2/2)

```
`timescale 1ns/1ns
module top;
  reg  data, cont;
  tri  resultA, resultB;
  tri  resultPA, resultPB;
  // without pull-up/down
  bufif1 Ua  (resultA, data, cont);
  notif0 Ub  (resultB, data, cont);
  // with pull-up/down
  bufif1 Upa (resultPA, data, cont);
  notif0 Upb (resultPB, data, cont);
  pulldown   (resultPA);
  pullup     (resultPB);
  initial begin
        data = 0;
        cont = 0;
     #5  cont = 1;
     #5  cont = 0;
     #5  data = 1;
     #5  cont = 1;
     #5  cont = 0;
     #5  $finish;
  end
  initial begin
     $dumpfile("wave.vcd");
     $dumpvars(1, data, cont, resultA, resultB);
     $dumpvars(1, resultPA, resultPB);
  end
endmodule
```



- ‘codes/verilog/gate_switch/buf_pull’

DIY


GTKWave - wave.vcd

```
VCD loaded successfully.
[6] facilities found.
[25] regions found.
```

From: 0 sec To: 25 ns Maximum Time 25 ns Current Time 15 ns

Signals: top.cont =0, top.data =0, top.resultA =Z, top.resultB =1, top.resultPA =0, top.resultPB =1

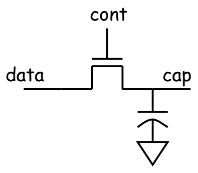
File Edit Search Time Markers View Help
Zoom Page Fetch Disc Shift
UNDO
Waves
9 ns
18 ns
Note Z
Note pulled up or down

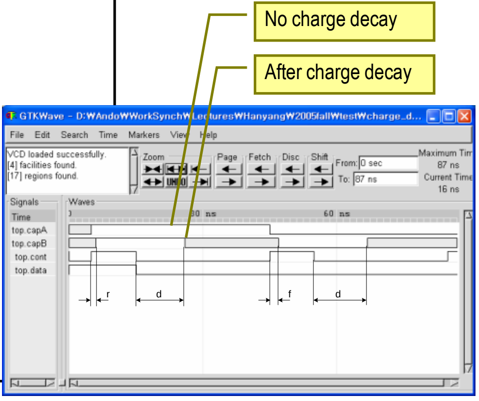
Charge decay of trireg net

- ‘trireg #(rise, fall, charge_decay) instance;’
  - No decay without charge_decay.
  - After charge_decay time, the trireg net makes a transition from 1 or 0 to X.
- ‘codes/verilog/gate_switch/charge_decay’

DIY

```
`timescale 1ns/1ns
module top;
  reg data, cont;
  trireg        capA;
  trireg #(1,2,10) capB;
  nmos UA(capA, data, cont);
  nmos UB(capB, data, cont);
  initial begin
    data = 1; cont = 0; #5; cont = 1; #10;
    data = 0; cont = 0; #30;  cont = 1; #10
            cont = 0; #30; cont = 1; #5
    #30 $finish;
  end
  initial begin
     $dumpfile("wave.vcd");
     $dumpvars(1, data, cont, capA, capB);
  end
endmodule
```




GTKWave - D:WAndoWWorkSynchWLecturesWHanyangW2005fallWtestWcharge_d...

```
VCD loaded successfully.
[4] facilities found.
[17] regions found.
```

From: 0 sec To: 87 ns Maximum Time 87 ns Current Time 16 ns

UDP

- User-defined primitive
  - Independent module
  - The same level as module definition
    - It can be appear anywhere in the source, either before or after it is instantiated inside a module.
    - It shall not be appear in module definition, i.e., between 'module' and 'endmodule'.
  - Instances of new UDP's can be used in exactly the same manner as the gate primitives.
  - Each UDP has exactly one output.
  - Value of UDP output can be 0, 1, or X.
    - Z is not supported.
    - Z in input will be X in the UDP.
- Types of UDP
  - Combinational UDP
    - Uses the value of its inputs to determine the next value of its output.
  - Sequential UDP
    - Uses the value of its inputs and the current value of its output to determine the value of its output.

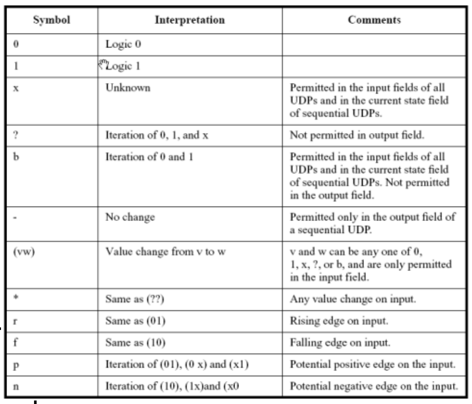
Form of UDP

- UDP has exactly one output port.
  - The output port must come first in the comma-separated list of ports.
- UDP has none or multiple input ports.
- Bidirectional port is not permitted.
- All port shall be scalar.
  - Vector ports are not permitted.
- Combinational UDP
  - Cannot has 'reg'.
- Sequential UDP
  - Can have 'reg' declaration for the output port.
  - Can have 'initial' block.
    - It specifies the value of the output port when simulation begins.

```
primitive name_of_UDP (oport [, iport1 [, iport2 [, …]]]);
output oport; [reg oport]
[input iport1; [input iport2; [input iport3; […]]]]
[initial oport = initial_value;]
table
 …
endtable
endprimitive
```

UDP table

- The state table of UDP defines the behavior of a UDP.



Symbol	Interpretation	Comments
0	Logic 0	
1	Logic 1	
x	Unknown	Permitted in the input fields of all UDPs and in the current state field of sequential UDPs.
?	Iteration of 0, 1, and x	Not permitted in output field.
b	Iteration of 0 and 1	Permitted in the input fields of all UDPs and in the current state field of sequential UDPs. Not permitted in the output field.
-	No change	Permitted only in the output field of a sequential UDP.
(vw)	Value change from v to w	v and w can be any one of 0, 1, x, ?, or b, and are only permitted in the input field.
*	Same as (??)	Any value change on input.
r	Same as (01)	Rising edge on input.
f	Same as (10)	Falling edge on input.
p	Iteration of (01), (0 x) and (x1)	Potential positive edge on the input.
n	Iteration of (10), (1x)and (x0	Potential negative edge on the input.

```
// UDP table for combinational UDP
// iport1  iport2  iport3 … : oport;
table
<UDP table symbol list for input ports> : <output port value> ;
endtable
```

```
// UDP table for sequential UDP
// iport1  iport2  iport3 … : current_oport : next_oprt;
table
<UDP table symbol list for input ports> : <current_oprt> : <next_oport> ;
endtable
```

Combinational UDP example

DIY

- 'codes/verilog/gate_switch/udp_mux'

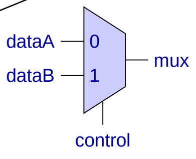
```
primitive
multiplexer (mux, control, dataA, dataB);
output mux;
input control, dataA, dataB;
table
// control dataA dataB mux
01 0 : 1 ;
01 1 : 1 ;
01 x : 1 ;
00 0 : 0 ;
00 1 : 0 ;
00 x : 0 ;
10 1 : 1 ;
11  1 : 1 ;
1x 1 : 1 ;
10 0 : 0 ;
11 0 : 0 ;
1x 0 : 0 ;
x0 0 : 0 ;
x1 1 : 1 ;
endtable
endprimitive
```

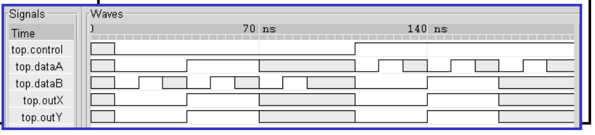
```
primitive
multiplexer (mux, control, dataA, dataB);
output mux;
input control, dataA, dataB;
table
// control dataA dataB mux
01?:1 ; // ? = 0 1 x
00?:0 ;
1?1:1 ;
1?0:0 ;
x00:0 ;
x11:1;
endtable
endprimitive
```

Output port comes first.

'?' is '0', '1', and 'x'.

'0xx' and '1xx' are not specified.
Thus this will make 'x' for its output when it happens.





Sequential UDP examples (1/2)

- ‘codes/verilog/gate_switch/udp_ff’

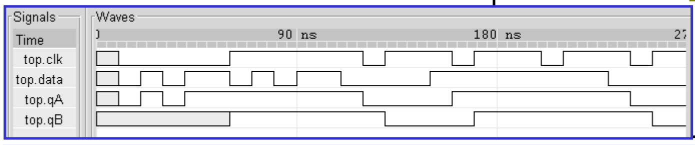
```
// level-sensitive sequential UDP
primitive latch (q, clock, data);
 output q; reg q;
 input clock, data;
 table// clock data qq+
 01 : ? :1 ;
 00 : ? :0 ;
 1? : ? :- ;// - = no change
 endtable
 endprimitive
```

Note ‘reg’.

‘?’ is ‘0’, ‘1’, and ‘x’.

```
// edge-sensitive sequential UDP
primitive d_edge_ff (q, clock, data);
output q;reg q;
input clock, data;
table
// clock dataqq+
(01) 0:?:0; // obtain output on rising edge of clock
(01) 1:?:1; // obtain output on rising edge of clock
(0?) 1:1:1; // obtain output on rising edge of clock
(0?) 0:0:0; // obtain output on rising edge of clock
(?0) ?:?:-; // ignore negative edge of clock
?  (??):?:-; // ignore data changes on steady clock
endtable
endprimitive
```

(01) Rising edge.
(10) Falling edge.
(??) Any value change.



DIY

Sequential UDP examples (2/2)

- ‘codes/verilog/gate_switch/udp_dff’

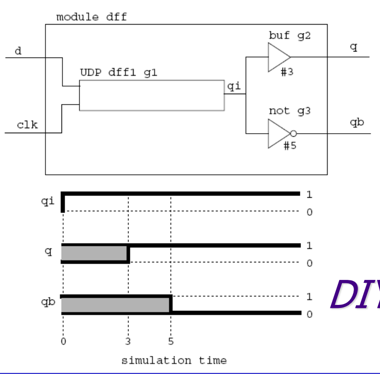
```
primitive dff1 (q, clk, d);
input clk, d;
output q;  reg q;
initial q = 1'b1;
table
// clkdqq+
r0:?:0;
r1:?:1;
f?:?:-;
?*:?:-;
endtable
endprimitive

module dff (q, qb, clk, d);
input clk, d;
output q, qb;
dff1   g1 (qi, clk, d);
buf #3 g2 (q, qi);
not #5 g3 (qb, qi);
endmodule
```

Note ‘initial’.

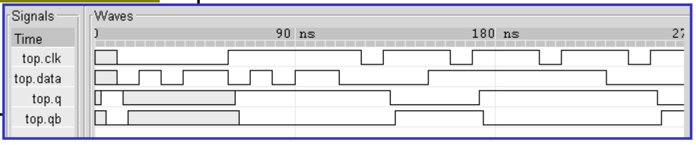
Stable signal of clock does not affect the output even input changes.

‘*’ is the same as (??), any value change.

See how to instantiate UDP.

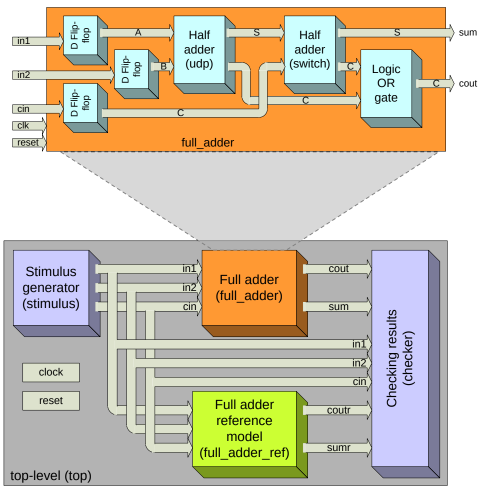


DIY



Project

- UDP half-adder
  - Make a half-adder in UDP.
    - Its module name is 'half_adder_udp'.
      - Thus, code 'half_adder_udp'.
  - Apply it to the following hierarchy.
    - 'full_adder', and
    - 'top'.
- Switch-level half-adder
  - Make a half-adder in switch-level.
    - Draw switch-level circuit using nMOS, pMOS, or cMOS.
    - Its module name is 'half_adder_switch'.
      - Thus code 'half_adder_switch.v'.
  - Apply it to the following hierarchy.
    - 'full_adder', and
    - 'top'.



Reading

- IEEE Std. 1364-2001, IEEE Standard Verilog Hardware Description Language. (Chapter 7. Gate and Switch Level Modeling; Chapter 8. User Defined Primitives)