

Synthesizable Verilog Code Examples

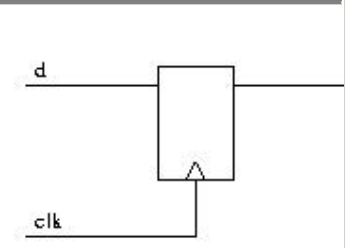
D Type Flip Flops:

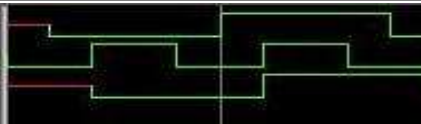
Two things to note about inferring flip flops:

- Non blocking signal assignment (`<=`) should always be used
- The sensitivity list must have the keyword `posedge` or `negedge`. (also for resets)

D-type flip flop

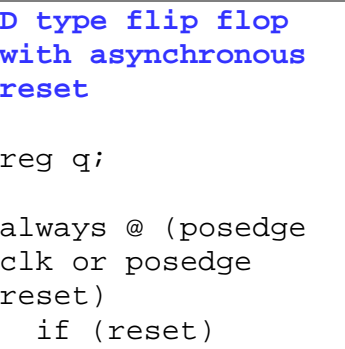
```
reg q;  
  
always @ (posedge  
clk)  
  q <= d;
```

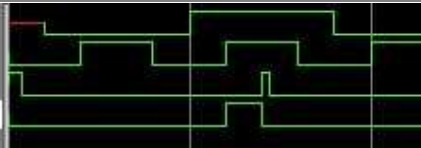


/test/d	0	
/test/clk	1	
/test/q	St0	

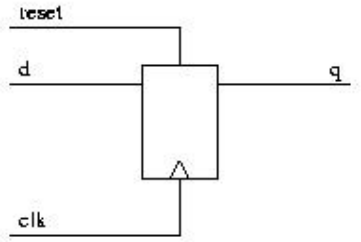
**D type flip flop
with asynchronous
reset**

```
reg q;  
  
always @ (posedge  
clk or posedge  
reset)  
  if (reset)  
    q <= 1'b0;
```



/test/d	0	
/test/clk	1	
/test/reset	0	
/test/q	St0	

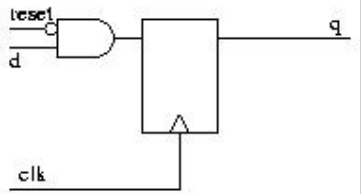
```
else
  q <= d;
```

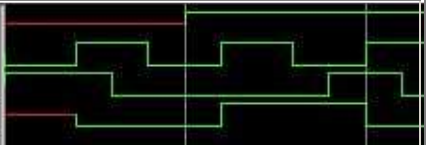


**D type flip flop
with synchronous
reset**

```
reg q;

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

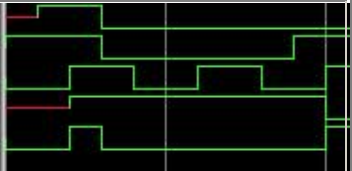


/test/d	1	
/test/clk	1	
/test/reset	0	
/test/q	St0	

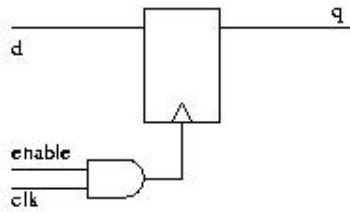
**D type flip flop
with gated clock**

```
reg q;
wire gtd_clk =
enable && clk;

always @ (posedge
```

/test/u1/d	St0	
/test/u1/enable	St1	
/test/u1/clk	St1	
/test/u1/q	0	
/test/u1/gated_clk	St1	

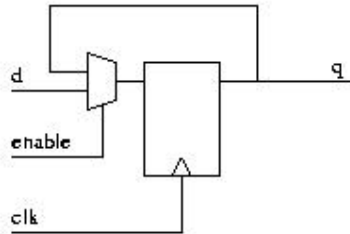
```
gtd_clk)
  q <= d;
```



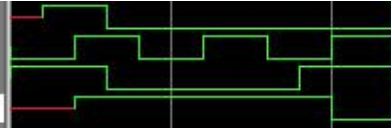
Data enabled D type
flip flop

```
reg q;
```

```
always @ (posedge clk)
  if (enable)
    q <= d;
```



/test/d	0
/test/clk	1
/test/enab	1
/test/q	St0

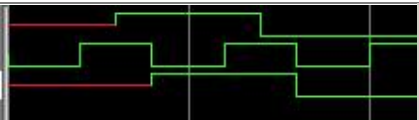


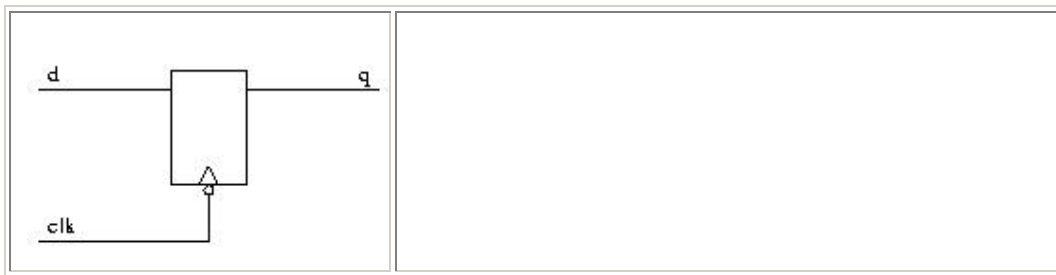
Negative edge
triggered D type
flip flop

```
reg q;
```

```
always @ (negedge
clk)
  q <= d;
```

/test/d	0
/test/clk	1
/test/q	St0





Latches

Latch

```

reg q;

always @ (q or
enable)
    if (enable)
        q = d;
        
```

●	/test/d	1	
●	/test/enable	1	
●	/test/q	St1	

Multiplexers

**Two input
multiplexer
(using if else)**

```

reg y;

always @ (a or
b or select)
    if (select)
        y = a;
    else
        y = b;
        
```

●	/muxtb/a	0	
●	/muxtb/b	1	
●	/muxtb/select	0	
●	/muxtb/y	St1	



Two input multiplexer (using ternary operator ?:)

```

wire t =
(select ? a :
b);

```

/muxtb/a	0			
/muxtb/b	1			
/muxtb/select	0			
/muxtb/t	St1			

Two input multiplexer (using case statement)

```

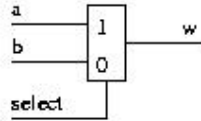
reg w;

// mux version
3
always @ (a or
b or select)
case
(select)
1'b1 : w
= a;
default :

```

/muxtb/a	0			
/muxtb/b	1			
/muxtb/select	0			
/muxtb/w	St1			

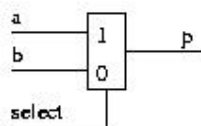
```
w = b;
endcase
```



Two input
multiplexer
(using default
assignment and
if)

```
reg p;

// mux version
4
always @ (a or
b or select)
begin
p = b;
if (select)
p = a;
end
```



/muxtb/a	0			
/muxtb/b	1			
/muxtb/select	0			
/muxtb/p	St1			

Three input priority
encoded mux
multiplexer (using
if else)

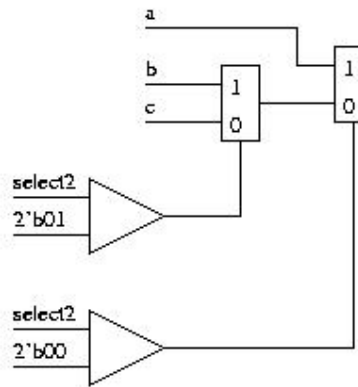
```
reg q;
```

/muxtb/a	1			
/muxtb/b	0			
/muxtb/c	0			
/muxtb/select2	00	00	01	11
/muxtb/q	St1			

```

always @ (a or b or
c or select2)
    if (select2 ==
2'b00)
        q = a;
    else
        if (select2 ==
2'b01)
            q = b;
        else
            q = c;

```



Three input priority
encoded mux
multiplexer (using
case)

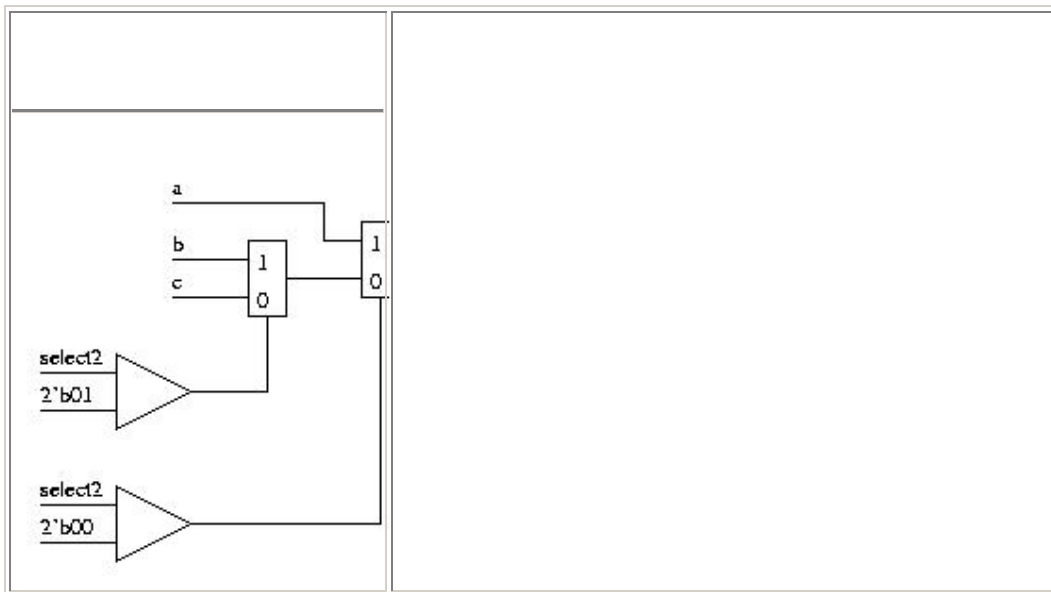
```

reg r;

// Priority encoded
mux, version 2
always @ (a or b or
c or select2)
    begin
        r = c;
        case (select2)
            2'b00: r = a;
            2'b01: r = b;
        endcase
    end

```

/muxtb/a	1				
/muxtb/b	0				
/muxtb/c	0				
/muxtb/select2	00	00	01	11	
/muxtb/r	St1				



Three input
multiplexer
with no
priority
(using case)

```
reg s;

always @ (a
or b or c or
select2)
begin
  case
  (select2) //
synopsys
parallel_cas
e
    2'b00:
s = a;
    2'b01:
s = b;

default: s =
c;
  endcase
end
```





<p>Comparator (using assign)</p> <pre> module comparator1 (a,b,c); input a; input b; output c; assign c = (a == b); endmodule </pre>	
--	--

<p>Comparator (using always)</p> <pre> module comparator2 (a,b,c); input a; input b; output c; reg c; always @ (a or b) if (a == b) c = 1'b1; else c = 1'b0; endmodule </pre>	
--	--

--	--

Finite State Machines

A full example of a state machine and associated test bench

The state machine

The following is a simple for state finite state machine.

The methodology for writing finite state machines is as follows:

1. Draw a state diagram. Label all conditions, label all output values. Label the state encoding.
2. Use parameter to encode the states as in the example.
3. Use two processes or always statements- one sequential and one combinational. See the example.
4. The state machine is normally resettable _ choose synchronous or asynchronous.
5. The combinational process normally has one big case statement in it. Put default values at the beginning.

There are a couple of neat things about the example. We are using parameters is the test bench and passing them to the state machine using parameter passing We are using tasks to control the flow of the testbench We are using hierarchical naming to access the state variable in the state machine from the test bench. Finally we are using test bench messages which allow us to monitor the current state from the simulation waveform viewer (assuming we change the bus radix of the 'message' to ascii.

```
// -----  
//  
// STATE MACHINE  
//
```

```

// -----
module state_machine(sm_in,sm_clock,reset,sm_out);

parameter idle   = 2'b00;
parameter read   = 2'b01;
parameter write  = 2'b11;
parameter wait   = 2'b10;

input sm_clock;
input reset;
input sm_in;
output sm_out;

reg [1:0] current_state, next_state;

always @ (posedge sm_clock)
begin
    if (reset == 1'b1)
        current_state <= 2'b00;
    else
        current_state <= next_state;
    end

always @ (current_state or sm_in)
begin
    // default values
    sm_out = 1'b1;
    next_state = current_state;
    case (current_state)
    idle:
        sm_out = 1'b0;
        if (sm_in)
            next_state = 2'b11;
    write:
        sm_out = 1'b0;
        if (sm_in == 1'b0)
            next_state = 2'b10;
    read:
        if (sm_in == 1'b1)
            next_state = 2'b01;
    wait:
        if (sm_in == 1'b1)
            next_state = 2'b00;
    endcase
end

endmodule

```