Level sensitive latch (D-Latch)

- The Verilog implementation of a D-latch is an always block that makes a *nonblocking assignment* ("\(<\leq\)") of d to q when the g input is nonzero.

- When g input is zero, then the *always* block does not make any assignment to q, causing the synthesis tool to infer a latch on the q output as the q output must retain its last known d value when g was nonzero.

- Nonblocking assignments ("\(<\leq\)") as opposed to blocking assignments ("\(=\)") should be used in always blocks that are used to synthesize sequential logic;

[from Thornton & Reese]
Edge-triggered storage element (D-FF)

- The @ symbol is used to specify an event control.
- Statements can be executed on changes in signal value or at a positive (posedge) or negative (negedge) transition of the signal.
- In general, edge-triggered storage elements are preferred to level-sensitive storage elements because of simpler timing requirements.
- The 1-bit edge-triggered storage elements provided by FPGA vendors are DFFs because of their simplicity and speed.

[Thornton & Reese]
DFF chains

- Each nonblocking assignment synthesizes to a single DFF whose input happens to be the output of another nonblocking assignment.
- The ordering of these nonblocking assignments within an always block does not matter

[Thornton & Reese]
DFF with asynchronous/synchronous inputs

(a) DFF with asynchronous Set/Reset

\[
\begin{array}{c|c|c|c|c|c|c}
\text{clk} & \text{d} & \text{r} & \text{s} & \text{clk} & \text{q} \\
1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 ? & d & 0 & \text{qold} \\
\end{array}
\]

always @(posedge clk or posedge r or negedge s)

begin
if (r) q <= 1'b0;
else if (!s) q <= 1'b1;
else q <= d;
end

(b) DFF with synchronous preset/clear

\[
\text{pre is low-true}
\]
\[
\text{clr is high-true}
\]
\[
\text{clr has priority over pre}
\]

always @(posedge clk)

begin
q <= d; //lowest priority
if (!pre) q <= 1'b1;
if (clr) q <= 0'b0; //highest priority
end

[Thornton & Reese]
Blocking vs. non-blocking statements

Zero-delay blocking assignments are so named because the assignment of the right-hand side (RHS) to the left-hand side (LHS) is completed without any intervening Verilog code allowed to execute, i.e., the assignment blocks the execution of the other Verilog code.

For nonblocking assignments within an always block, all RHS expressions are evaluated, and are only assigned to the LHS targets after the always block completes.
Avoid combinational loops

(a) A combinational loop

\[
\text{always @* begin} \\
\quad y = y + a; \\
\text{end}
\]

Output oscillates; period is dependent upon adder delay

(b) Sequential element in feedback path

\[
\text{always @(posedge clk) begin} \\
\quad y <= y + a; \\
\text{end}
\]

Output can only change on the active clock edge

[Thornton & Reese]
Guidelines to avoid frustration [Thornton & Reese]

- Use blocking assignments ("=") in always blocks that are meant to represent combinational logic. Use nonblocking assignments ("<=") in always blocks that are meant to represent sequential logic.
- Do not mix blocking and nonblocking assignments in the same always block.
- If an always block contains a significant amount of combinational logic that requires intermediate wires (and thus, intermediate assignments), then place this logic in a separate always block.
- If an always block for combinational logic contains complicated logic pathways due to if-else branching or other logic constructs, then assign every output a default value at the beginning of the block. This ensures that all outputs are assigned a value regardless of the path taken through the logic, avoiding inferred latches on outputs.
- Do not make assignments to the same output from multiple always blocks.
Loops in Verilog

```verilog
text
integer count;
integer y=1;
integer x=2;

initial
  for (count = 0; count < 128; count = count + 1)
    begin
      x <= x + y;
      y <= x;
    end

initial
  count = 0;
  while (count < 128)
    begin
    .
    .
      count = count +1;
    end

initial
  count = 0;
  repeat(128)
    begin
    .
    .
      count = count +1;
    end
```

Must contain a number or a signal value; only evaluated once at the beginning.
D2 example: A 1 second blinking light

module sec (input CLOCK_50, output reg [8:0] LEDG);

integer count=0;

initial LEDG[0]=1'b0;

always @(posedge CLOCK_50)
begin
    count=count+1;
    if(count == 50_000_000)
    begin
        count=0;
        if(LEDG[0]) LEDG[0]=1'b0;
        else LEDG[0]=1'b1;
    end
end
endmodule
Tasks

- Tasks are declared with the keywords task and endtask.
- Tasks can have input, inout, and output arguments to pass values (different than in modules).
- Tasks can have local variables but cannot have wires.
- Tasks can only contain behavioral statements.
- Tasks do not contain always or initial statements but are called from always blocks, initial blocks, or other tasks and functions.
- Can operate directly on reg variables defined in the module.

```verbatim
module ...
... always
  begin
    BOP (AB_AND, AB_OR, AB_XOR, A, B);
    BOP (CD_AND, CD_OR, CD_XOR, C, D);
  end

task BOP;
  output [15:0] ab_and, ab_or, ab_xor;
  input [15:0] a, b;
  begin
    ab_and = a & b;
    ab_or = a | b;
    ab_xor = a ^ b;
  end
endtask
... endmodule
```
Example: clock display on DE2

- Last lecture we had a simple example of a 1 second blinking LED
- Let's generalize it to a clock display
Task to display digits

task digit2sev(input integer digit, output [6:0] disp);
begin
    if (digit == 0) disp = 7'b1000000;
    else if (digit == 1) disp = 7'b1111001;
    else if (digit == 2) disp = 7'b0100100;
    else if (digit == 3) disp = 7'b0110000;
    else if (digit == 4) disp = 7'b0011001;
    else if (digit == 5) disp = 7'b0010010;
    else if (digit == 6) disp = 7'b0000011;
    else if (digit == 7) disp = 7'b1111000;
    else if (digit == 8) disp = 7'b0000000;
    else if (digit == 9) disp = 7'b0011000;
end
endtask
module clock(CLOCK_50, HEX0, HEX1, HEX2, HEX3);
output reg [6:0] HEX0, HEX1, HEX2, HEX3;
input CLOCK_50; integer count=0;
reg [3:0] d1=0, d2=0, d3=0, d4=0;

always @(posedge CLOCK_50)
begin
  count=count+1;
  if (count == 50_000_000)
  begin
    count=0; d1 = d1+1;
    if(d1 == 10)
    begin
      d1 = 0; d2 = d2+1;
      if(d2 == 6)
      begin
        d2 = 0; d3 = d3 + 1;
        if(d3 == 10)
        begin
          d3 = 0; d4 = d4 + 1;
          if(d4 == 6) d4 = 0;
        end
      end
      end
      digit2sev(d1, HEX0); digit2sev(d2, HEX1);
      digit2sev(d3, HEX2); digit2sev(d4, HEX3);
    end
  end
end

endmodule
Resource utilization is 152 LEs
Second code

```verilog
module clock(CLOCK_50, HEX0, HEX1, HEX2, HEX3);
input CLOCK_50;
output reg [6:0] HEX0, HEX1, HEX2, HEX3;
integer count=0;
reg [15:0] ticks=16'd0;
reg [5:0] seconds=6'd0, minutes=6'd0;

initial display_time(seconds, minutes);

always @(posedge CLOCK_50)
begin
    count = count+1;
    if (count == 50_000_000)
        begin
            count=0;
            ticks = ticks + 1;
            seconds = ticks % 60;
            minutes = ticks / 60;
            display_time (seconds, minutes);
        end
end
end
```
task display_time(input [5:0] s, input [5:0] m);
begin
    digit2sev(s%10, HEX0);
    digit2sev(s/10, HEX1);
    digit2sev(m%10, HEX2);
    digit2sev(m/10, HEX3);
end
d-endtask

task digit2sev(input integer digit, output [6:0] disp);
begin
    if (digit == 0) disp = 7'b1000000;
    else if (digit == 1) disp = 7'b1111001;
    else if (digit == 2) disp = 7'b0100100;
    else if(digit == 3) disp = 7'b0110000;
    else if(digit == 4) disp = 7'b0011001;
    else if(digit == 5) disp = 7'b0010010;
    else if(digit == 6) disp = 7'b0000011;
    else if(digit == 7) disp = 7'b1111000;
    else if(digit == 8) disp = 7'b0000000;
    else if(digit == 9) disp = 7'b0011000;
end
endtask
endmodule
Resource utilization for 2\textsuperscript{nd} code

- Circuit consumes 611 LEs (2\% of the chip logic resources).
- You have to be careful! Changing ticks, seconds and minutes to integer increases area to become 2500 LEs (8\% of the utilization)
Lab 3: BrainTuner for DE2!

- You are prompted with 20 “random” questions to answer
- The board should tell you instantly whether you are right or wrong, and at the end, the board should give you a final score and time at the end
- The questions should be challenging

Time in seconds

1st number

2nd number

result

A +

S -
P *

Two push buttons for right / wrong