```
/* Name: Eugene Ngo
   Date: 3/7/2023
   Class: EE 371
   Lab 6: Parking Lot 3D Simulation
*/

// carIncrCounter takes in an increment and two reset inputs
// to update a stored logic variable that acts as a storage
// of the number of cars that have enterred on a given hour.
// This count is reset everytime the reset switch is flipped
// or if the hour increment button is hit, allowing for a new
// value to be enterred into the RAM storage with the addressIterator's
// address output.
`timescale 1 ps / 1 ps
module carIncrCounter (inc, buttonReset, clk, reset, out);
   // Inc logic is used to incremement the value of the
   // outputted car numbers until it reaches a maximum of 8
   // and then stopping.

   input logic inc, clk, reset, buttonReset;
   output logic [3:0]  out;

   // Sequential logic for counting up and counting down depending on the input.
   always_ff @(posedge clk) begin
      // if Sw[9] or Key[0] is pressed, reset.
      if (reset | buttonReset) begin
         out <= 4'b0000;
      end
      // Otherwise, just increment the value.
      else if (inc & out < 4'b1000) begin //increment when not at max
         out <= out + 4'b0001;
      end
      // If none of the above are met, hold onto the current value.
      else
         out <= out; // hold value otherwise
   end // always_ff

endmodule

// carIncrCounter_testbench tests all expected, unexpected, and edgecase behaviors
// of carIncr, ensuring it counts up to 8, and holds that value, unless the button
// or switch reset values are triggered. THis is used to store values into the RAM.
module carIncrCounter_testbench();
   logic inc, clk, reset, buttonReset;
   logic [3:0] out;
   logic CLOCK_50;

   carIncrCounter  dut (inc, buttonReset, CLOCK_50, reset, out);

   // Setting up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      CLOCK_50 <= 0;
      forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
   end // initial

   initial begin
      reset <= 1;                @(posedge CLOCK_50); // reset
      reset <= 0;                @(posedge CLOCK_50); // inc past max limit
      inc <= 0;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      inc <= 1;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      buttonReset <= 1;                @(posedge CLOCK_50); // reset
      buttonReset <= 0;                @(posedge CLOCK_50); // inc past max limit
      inc <= 0;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      inc <= 1;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      $stop;
   end
endmodule // counter_testbench
```