Solutions to Quiz 2

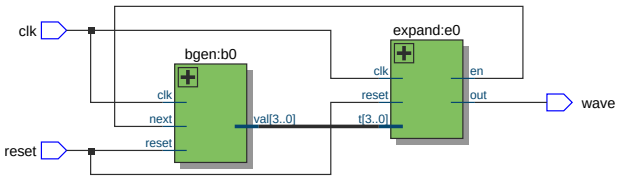
There was one version of Question 1 and two versions of Question 2. The values and the answers for all versions are given below.

Question 1

Given the following module declarations:

```
module bgen #(parameter b)
   ( input logic reset, clk, next,
     output logic [b-1:0] val ) ;

module expand #(parameter b)
   ( input logic reset, clk,
     input logic [b-1:0] t,
     output logic en,
     output logic out ) ;
```

Write a Verilog module named **tgen** that implements the following diagram:



Declare any additional signals required to implement the diagram, using any valid signal names. Use a value of 4 for the parameter **b** in each instantiation. You may use any of the conventions for associating signals and ports.

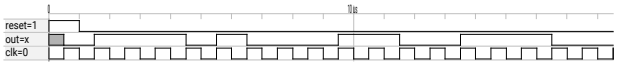
Answers

The following code generates the block diagram shown in the quiz. The **tgen** module shows different ways the **bgen** and **expand** modules could be instantiated.

```
// ELEX 2117 202510 Quiz 2 Question 1 - Example
// Ed.Casas 2025-2-22

// output next value in table 'tab' when 'next' asserted

module bgen #(parameter b)
   ( input logic reset, clk, next,
     output logic [b-1:0] val ) ;

   logic [b-1:0] tab [7] = '{ 3, 1, 1, 3, 2,   2, 3 } ;
   logic [3:0] i ;

   always_ff @(posedge clk) i
      <= reset ? '0 : next ? ( i == $size(tab)-1 ? '0 :
           ↪ i + 1'b1 ) : i ;

   assign val = tab[i] ;

endmodule

// set 'out' alternating low/high for duration 't'
// asserts 'en' to get next duration

module expand #(parameter b)
   ( input logic reset, clk,
     input logic [b-1:0] t,
     output logic en, out ) ;

   logic [b-1:0] i ;

   // count down the duration of each pulse
   always_ff @(posedge clk) i
      <= reset ? '0 : i == 0 ? t-1'b1 : i - 1'b1 ;

   // alternate +ve and -ve pulses
   always_ff @(posedge clk) out
      <= reset ? '0 : en ? !out : out ;

   // get next duration when start one
   assign en = i == 0 ;

endmodule

// example test waveform generator
// instantiates 'bgen' and 'expand'

module tgen
   ( input logic reset, clk,
     output logic wave ) ;

   localparam b = 4 ;

   logic en ;
   logic [b-1:0] t ;

   // equivalent instantiations:

   // bgen #(4) b0 ( reset, clk, e, t ) ;
   // bgen #(.b(4)) b0 ( .reset(reset), .clk(clk),
   //     .next(en), .val(t) ) ;
   // bgen #(.b(4)) b0 ( .reset, .clk,
   //     .next(en), .val(t) ) ;
   bgen #(.b(4)) b0 ( .next(en), .val(t), .* ) ;

   // expand #(4) e0 ( reset, clk, t, e, wave ) ;
   // expand #(.b(4)) e0 ( .reset(reset), .clk(clk),
   //     .t(t), .en(en), .out(wave) ) ;
   // expand #(.b(4)) e0 ( .reset, .clk,
   //     .t, .en, .out(wave) ) ;
   expand #(.b(4)) e0 ( .out(wave), .* ) ;

endmodule

module tgen_tb ;
   logic reset, clk ;
```

```
   logic wave ;

   tgen t0 (.*) ;

   initial begin
      $dumpfile("tgen_tb.vcd") ;
      $dumpvars(3) ;
      { reset, clk } = 2'b10 ;
      #1us reset = '0 ;
      #18us $finish ;
   end

   always #0.5us clk = ~clk ;

endmodule
```

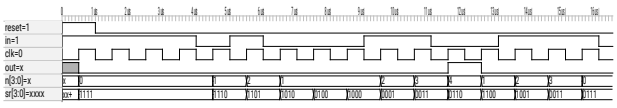
The (post-reset portion of) the **wave** output from **tgen** shown below is also the **in** waveform used in the next question.



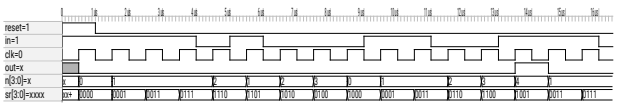
Question 2

Write a module named **detect** with one-bit inputs named **clk**, **reset**, and **in**, and a one-bit output named **out**.

out should be set to 1 when the sequence 0, 1, 1, 0 (or 1, 0, 0, 1) has appeared on **in** on consecutive rising edges of **clk**. When **reset** is asserted it should be assumed all previous and current values of **in** were/are 1 (or 0). **out** should be set to 1 at the rising edge of **clk** on the last value of the sequence and stay high for one **clk** period. For example:



or:



Any correct solution is acceptable. The timing diagram above shows two possible state variables that you could use. **n** counts the correct number of values seen; **sr** stores the most recent four input values.

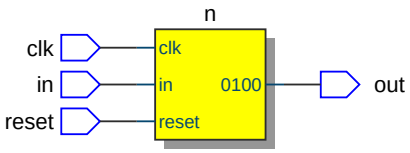
Answers

There were two versions of this question. The two answers are named **detect_hi** and **detect_low**. Each solution shows two possible design approaches, one using a counter and one using a shift register, although only *one* of these was required.

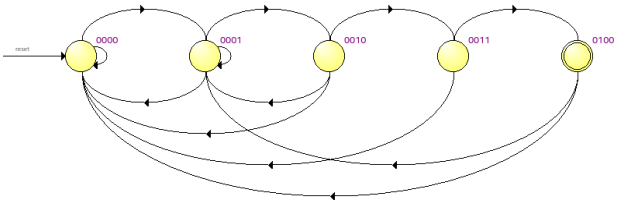
```
module detect_lo
   ( input logic clk, reset,
     input logic in,
     output logic out ) ;

   logic [3:0] n, sr ;

   // update number of values seen from sequence
   always_ff @(posedge clk) n
      <= reset ? 0 :
         n == 0 && in == 1 ? 1 :
         n == 1 && in == 0 ? 2 :
         n == 2 && in == 0 ? 3 :
         n == 3 && in == 1 ? 4 :
         in == 1 ? 1 :
         0 ;

   assign out = n == 4 ;

   // update input shift register
   always_ff @(posedge clk) sr
      <= reset ? '0 :
         { sr[2:0], in } ;

   logic out_ ;
   assign out_ = sr == 4'b1001 ;

endmodule

module detect_hi
   ( input logic clk, reset,
     input logic in,
     output logic out ) ;

   logic [3:0] n, sr ;

   // update number of values seen from sequence
   always_ff @(posedge clk) n
      <= reset ? 0 :
         n == 0 && in == 0 ? 1 :
         n == 1 && in == 1 ? 2 :
         n == 2 && in == 1 ? 3 :
         n == 3 && in == 0 ? 4 :
         in == 0 ? 1 :
         0 ;

   logic out_ ;
   assign out_ = n == 4 ;

   // update input shift register
   always_ff @(posedge clk) sr
      <= reset ? '1 :
         { sr[2:0], in } ;

   assign out = sr == 4'b0110 ;

endmodule

// synthesis translate off
`timescale 1us/100ns
```

```
module detect_tb ;

   logic clk, reset ;
   logic in ;
   logic outhi, outlo ;

   detect_hi dh0 (.out(outhi), .*) ;
   detect_lo dl0 (.out(outlo), .*) ;

   int t[] = { 3, 1, 1, 3, 2, 2, 3, 0 } ;

   initial begin
      $dumpfile("detect_tb.vcd") ;
      $dumpvars ;

      { reset, clk, in } = 3'b101 ;
      #1us reset = '0 ;

      for ( int i=0 ; t[i] ; i++ )
         #t[i] in = ~in ;

      #1us $finish ;
   end

   always #0.5us clk = ~clk ;

endmodule
// synthesis translate on
```

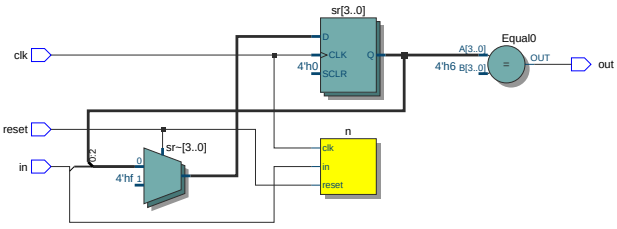
Running the testbench generates the waveforms shown above and the **`detect_lo`** version synthesizes to a single state machine:



with a state transition diagram of:



while the **`detect_hi`** version synthesizes to:



(for some reason the state machine code is also generated).