Solutions to Final Exam

Question 1

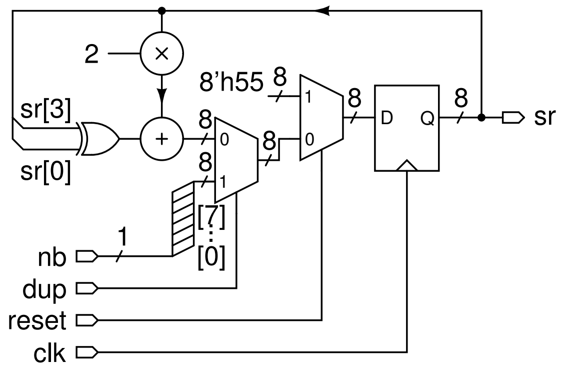
The two versions of this question differed only in bus width (16 or 32 bits). One of many possible solutions is:

```
module examq1 (
   input logic [15:0] max, min,
   input logic clk,
   output logic [15:0] y ) ;

   always_ff @(posedge clk)
      y += y < min ? 1 : y > max ? -1 : 0 ;

endmodule
```

Question 2

The two versions of this question differed only in the initialization values and the bits connected to the XOR gate.

A straightforward conversion of the code to a schematic would give:



The for-loop copies the 1-bit nb input to each bit of the 8-bit input of the multiplexer.

The optimized schematic generated by Quartus, shown in Figure 1, implements the multiply-and-add as a left-shift concatenated with the xor result.

Question 3

The two versions of this question differred only in how a and b were modified in each state.

The solution involves a state machine and two counters.

The solution below uses button as a clock to control the state transitions. A better solution would be to register the value of the button input and detect rising edges by comparing the current and previous values.

The question does not specify whether button is asynchronous with respect to clk. If button changed too quickly to provide an acceptable MTBF and debouncing logic didn't provide synchronization then a synchronizer would be needed.

```
module bcontrol(
   input logic button, clk,
   output logic [7:0] a, b ) ;

   logic [1:0] state ; // initial value assumed 0
   logic [7:0] a_next, b_next ;

   always_comb begin
      a_next = a ;
      b_next = b ;

`define v1
`ifdef v1      // first version of question
      case (state)
      0: a_next = a+1 ;
      1: b_next = b+1 ;
      2: begin
         a_next = '0 ;
         b_next = '0 ;
         end
      endcase
`else      // second version of question
      case (state)
      0: begin
         a_next = 8'hff ;
         b_next = 8'h00 ;
         end
      1: begin
         a_next = a-1 ;
         b_next = a+1 ;
         end
      endcase
`endif
   end

   always_ff @(posedge clk) begin
      a <= a_next ;
      b <= b_next ;
   end

   always_ff @(posedge button)
      state <= state == 2 ? 0 : state+1 ;

endmodule
```

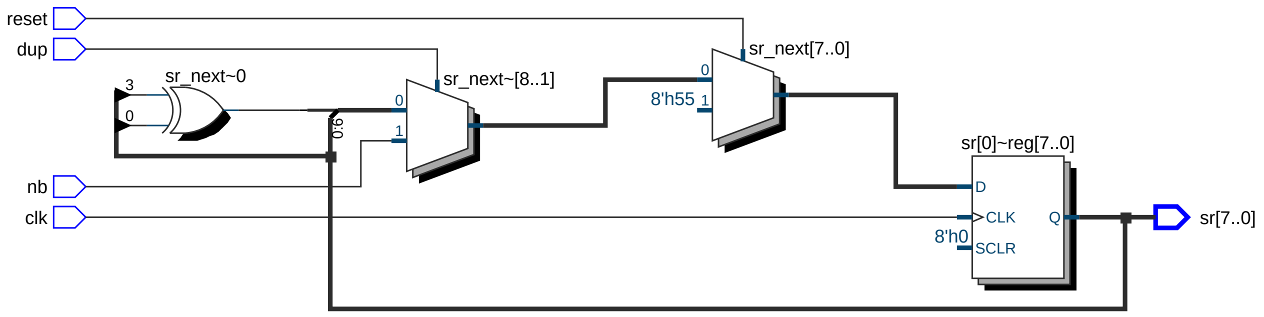


Figure 1: Quartus-generated solution for Question 2.

Question 4

The algorithm described by the C functions uses three variables (a, b, and done) which are implemented as three registers in RTL.

The control flow in the C function algorithm is a conditional loop controlled by done so the controller for the System Verilog implementation is a state machine with two states (done=0 or done=1). State transitions happen when `reset` is asserted (done→0) and when a==b (done→1).

The computation of the next values of a, b and done in each state is taken from the C code.

```
module gcd (
             input logic [31:0] a_in, b_in,
             output logic [31:0] a, b,
             output logic done,
             input logic clk, reset ) ;

   logic done_next ;
   logic [31:0] a_next, b_next ;

   always_comb begin
      if ( reset )
        begin
           done_next = '0 ;
           a_next = a_in ;
           b_next = b_in ;
        end
      else
        begin
           done_next = done ;
           a_next = a ;
           b_next = b ;
           if ( ! done )
             if ( a > b )
               a_next = a - b ;
             else if ( a < b )
               b_next = b - a ;
             else
               done_next = '1 ;
        end
   end

   always_ff @(posedge clk) begin
      done <= done_next ;
      a <= a_next ;
      b <= b_next ;
   end

endmodule
```

Question 5

The question asks for the MTBF for a two-flip-flop synchronizer without using the term "MTBF." This is given by:

$$\text{MTBF} = \frac{e^{T_s/C_2}}{C_1 f_d f_c}$$

where $C_1 = C_2 = 2$ ns, $f_d = 1$ kHz and $f_c = 50$ MHz are given in the question.

The slack time (T_s) is the amount by which a timing requirement is exceeded. For a two-flip-flop synchronizer this requirement is the setup time of the second flip-flop.

The setup slack time will be equal to the clock period (20 ns) minus any combinational logic delays between the flip-flops, minus the minimum setup time requirement (t_{SU}). A two-flop synchronizer connects the output of the first flip-flop directly to the input of the second flip-flop so the only combinational logic delay is the clock-to-output delay (t_{CO}) of the first flip-flop (5 ns). The setup time requirement of the second flip-flop was not given. Any reasonable assumption for t_{SU}, even zero, was marked correct. Thus $T_s = 20 - 5 - t_{\text{SU}} <= 15$ ns and

$$\text{MTBF} \leqslant \frac{e^{15/2}}{2 \times 10^{-9} \cdot 50 \times 10^{6} \cdot 1000} \approx 18\,\text{seconds}$$