Solutions to Final Exam

Question 1

For the following declarations in a System Verilog module:

```
logic a = '1 ;
logic [7:0] b  = 8'ha6 ;
logic [1:0][3:0] c  = '{4'b0000, 4'b1001 };
```

the simulation output below shows sizes and values of the following expressions:

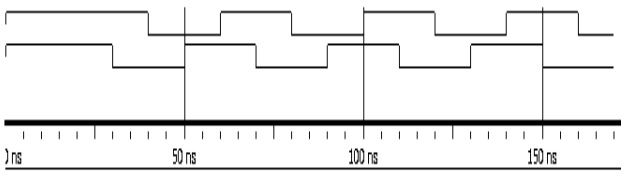
```
 b==?8'bxxxx0110: bits:  1 val: 0000 0001
signed'(b) >>> 1: bits:  8 val: 1101 0011
     {b[7:1],a}: bits:  8 val: 1010 0111
         b[7:4]: bits:  4 val: 0000 1010
         {2{c}}: bits: 16 val: 0000 1001
         b+1'b1: bits:  8 val: 1010 0111
      a ? b : c: bits:  8 val: 1010 0110
```

Question 2

The following code:

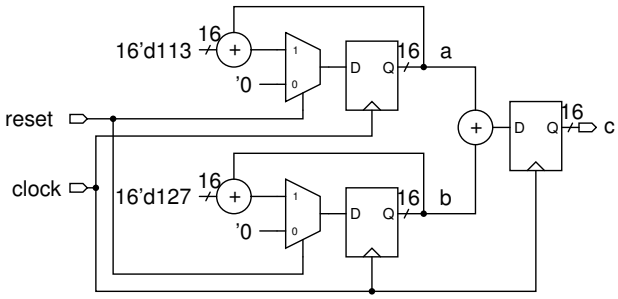
```
module examtb ( output logic ca, cb ) ;

  initial begin
     ca = '1 ;
     cb = '1 ;

     #30ns cb = '0 ;
     #10ns ca = '0 ;

     forever begin
        #10ns cb = ~cb ;
        #10ns ca = ~ca ;
     end

  end

endmodule
```

generates the required waveforms:



Question 3

The following schematic:



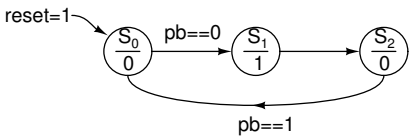
implements the following System Verilog module:

```
module mksum
  ( output logic [15:0] c,
    input logic reset, clk ) ;

   logic [15:0] a, a_next, b, b_next ;

   always@(posedge clk) begin
      a <= a_next ;
      b <= b_next ;
      c <= a_next + b_next ;
   end


   always_comb begin
      if ( reset ) begin
         a_next = '0 ;
         b_next = '0 ;
      end else begin
         a_next = a + 16'd113 ;
         b_next = b + 16'd127 ;
```

```
      end
   end

endmodule
```

Question 4

The following state transition diagram:



can be implemented by the following System Verilog module:

```
// ELEX 7660 201710 Final Exam Question 4

module pulsegen (
                 input logic pb,
                 output logic pulse,
                 input logic clk, reset ) ;

   typedef enum logic [1:0] { S0, S1, S2 }
                state_t ;
   state_t state, next_state ;

   logic pulse_next ;

   // your code goes here

   always_comb begin

      next_state = state ;

      if ( reset )
        next_state = S0 ;
      else
        case(state)
          S0: if ( pb == 0 )
            next_state = S1 ;
          S1: next_state = S2 ;
          S2: if ( pb == 1 )
            next_state = S0 ;
        endcase
   end

   always_comb begin
      if ( next_state == S1 )
        pulse_next = '1 ;
      else
        pulse_next = '0 ;
   end

   always_ff@(posedge clk) begin
      state <= next_state ;
      pulse <= pulse_next ;
   end

endmodule

module examq4 ;

   logic pb, pulse, clk, reset ;

   pulsegen p0 ( pb, pulse, clk, reset ) ;

   // clock and reset
   initial begin
      reset = '1 ;
      clk = '0 ; #500ns ;
      clk = '1 ; #500ns ;
      reset = '0 ;
      forever begin
         clk = '0 ; #500ns ;
         clk = '1 ; #500ns ;
      end
   end

   // pushbutton
   initial begin
      $dumpfile("examq4.vcd") ;
      $dumpvars ;
      pb = '1 ;
      #2us pb = '0 ;
      #2us pb = '1 ;
      #2us $finish ;
   end

endmodule
```

with the simulation results shown in Figure 1.

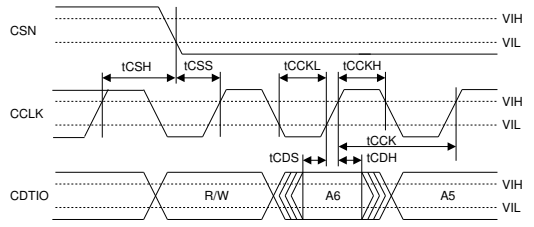
Question 5

The following timing specifications are found in the datasheet for an audio codec IC:

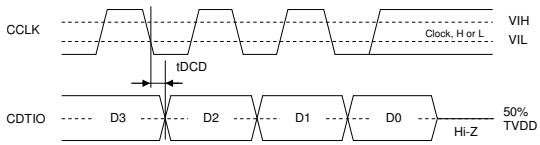
Control Interface Timing (3-wire Serial: AK4951EG)					
CCLK Period	tCCK	200	-	-	ns
CCLK Pulse Width Low	tCCKL	80	-	-	ns
Pulse Width High	tCCKH	80	-	-	ns
CDTIO Setup Time	tCDS	40	-	-	ns
CDTIO Hold Time	tCDH	40	-	-	ns
CSN "H" Time	tCSW	150	-	-	ns
CSN Edge to CCLK "↑"	tCSS	50	-	-	ns
CCLK "↑" to CSN Edge	tCSH	50	-	-	ns
CCLK "↓" to CDTIO (at Read Command)	tDCD	-	-	70	ns
CSN "↑" to CDTIO (Hi-Z) (at Read Command)	tCCZ	-	-	70	ns

Figure 1: Question 4 testbench results.

y The timing diagram for a read cycle is:



The timing diagram for a write cycle is:



The signal directions are:

Signal	Direction
CSN	input
CCLK	input
CDTIO	input (during write cycles)
CDTIO	output (during read cycles)

(a) Specifications whose time ends at an input (I) are requirements, those ending at an output (O) are "guaranteed responses."

specification	Measured to (I/O)	Requirement (R) or Guaranteed (G)
tCCK	CCLK (I)	R
tCDS	CCLK (I)	R
tCDH	CDTIO (I)	R
tDCD	CDTIO (O)	G

(b) The maximum clock (CCLK) *frequency* is given by the inverse of the minimum clock period (200ns) or 5 MHz.

(c) During a write cycle the data (CDTIO) must be valid tCDS (40 ns, CDTIO setup time) before the rising edge of the clock. If the clock period is 1 μs (1000 ns), then the maximum delay between the *previous* rising clock edge and CDTIO being valid is 1000-40=960 ns.

(d) The minimum delay between the rising edge of CCLK and CDTIO being invalid is tCDH which is 40 ns.