CS150 Spring 2004
Problem Set #6 Solutions

Greg Gibeling

Note that this is a fully functional verilog description of the alarm clock in problem #1. We will post a student solution to problem #2 once the homework is graded. Please see the zip file for the actual verilog for this problem.

```
`timescale 1 ns/1 ns
`define half_cycle27 18.5

module Testbench;

        reg             Clock, Reset;
        reg             SetHour, SetMinute;
        reg             AlarmMode, AlarmOn;
        reg             Snooze;

        wire    [4:0]   DisplayHour;
        wire    [5:0]   DisplayMinute;
        wire            AlarmSounding;

        AlarmClock      CUT(    .Clock(         Clock),
                                .Reset(         Reset),
                                .SetHour(       SetHour),
                                .SetMinute(     SetMinute),
                                .AlarmMode(     AlarmMode),
                                .AlarmOn(       AlarmOn),
                                .Snooze(        Snooze),
                                .DisplayHour(   DisplayHour),
                                .DisplayMinute(DisplayMinute),
                                .AlarmSounding(AlarmSounding));

        always #(`half_cycle27) Clock = ~Clock;

        initial begin
                Clock =                         1'b0;
                Reset =                         1'b1;
                SetHour =                       1'b0;
                SetMinute =                     1'b0;
                AlarmMode =                     1'b0;
                AlarmOn =                       1'b0;
                Snooze =                        1'b0;

                #(`half_cycle27 * 2);
                Reset =                         1'b0;
                SetHour =                       1'b1;
                #(`half_cycle27 * 34);
                SetHour =                       1'b0;
                SetMinute =                     1'b1;
                #(`half_cycle27 * 208);
                SetMinute =                     1'b0;
                #(`half_cycle27 * 2);
                AlarmMode =                     1'b1;
                #(`half_cycle27 * 2);
                SetHour =                       1'b1;
                #(`half_cycle27 * 38);
                SetHour =                       1'b0;
                SetMinute =                     1'b1;
                #(`half_cycle27 * 2);
                SetMinute =                     1'b0;
                AlarmOn =                       1'b1;
                #(`half_cycle27 * 84);
                Snooze =                        1'b1;
                #(`half_cycle27 * 2);
```

```
                Snooze =                        1'b0;
        end
endmodule


module AlarmClock(      Clock,          // 27MHz Reference Clock
                        Reset,          // Power-up reset

                        SetHour,        // The set-hour button
                        SetMinute,      // The set-minute button

                        AlarmMode,      // 1 if the toggle is set to "alarm mode"
                        AlarmOn,        // 1 if the alarm is activated

                        Snooze,         // The snooze button

                        DisplayHour,    // The hour to display
                        DisplayMinute, // the minute to display
                        AlarmSounding  // 1 if the alarm is sounding
                );

        input           Clock, Reset;
        input           SetHour, SetMinute;
        input           AlarmMode, AlarmOn;
        input           Snooze;

        output  [4:0]   DisplayHour;
        output  [5:0]   DisplayMinute;
        output          AlarmSounding;

        wire            Minute, Hour;
        wire            NextClockMinute, NextClockHour;
        wire            NextAlarmMinute, NextAlarmHour;
        wire            SnoozeDone;

        wire    [4:0]   ClockHour, AlarmHour;
        wire    [5:0]   ClockMinute, AlarmMinute;

        wire            AlarmStart;
        reg             AlarmPending;

        assign  DisplayHour =           AlarmMode ? AlarmHour : ClockHour;
        assign  DisplayMinute =         AlarmMode ? AlarmMinute : ClockMinute;

        assign  NextClockMinute =       Minute | (~AlarmMode & SetMinute);
        assign  NextClockHour =         (Hour & Minute & ~SetMinute) | (~AlarmMode & SetHour);
        assign  NextAlarmMinute =       AlarmMode & SetMinute;
        assign  NextAlarmHour =         AlarmMode & SetHour;

        assign  AlarmStart =            AlarmOn & (AlarmHour == ClockHour) & (AlarmMinute ==
ClockMinute);
        assign  AlarmSounding =         (AlarmPending | AlarmStart) & SnoozeDone;

        always @ (posedge Clock) begin
                if (Reset | (Minute & AlarmSounding)) AlarmPending <= 1'b0;
                else if (AlarmStart) AlarmPending <= 1'b1;
        end

        Divide2Minute   D2M_Inst(       .Clock(         Clock),
                                        .Reset(         Reset),
                                        .Minute(        Minute));

        MinuteCount     MC_Clock(       .Clock(         Clock),
                                        .Reset(         Reset),
                                        .Enable(        NextClockMinute),
                                        .Minute(        ClockMinute),
                                        .Rollover(      Hour));

        HourCount       HC_Clock(       .Clock(         Clock),
                                        .Reset(         Reset),
                                        .Enable(        NextClockHour),
```

```
                                    .Hour(          ClockHour),
                                    .Rollover(      ));

        MinuteCount     MC_Alarm(   .Clock(         Clock),
                                    .Reset(         Reset),
                                    .Enable(        NextAlarmMinute),
                                    .Minute(        AlarmMinute),
                                    .Rollover(      ));

        HourCount       HC_Alarm(   .Clock(         Clock),
                                    .Reset(         Reset),
                                    .Enable(        NextAlarmHour),
                                    .Hour(          AlarmHour),
                                    .Rollover(      ));

        SnoozeDelay     SD_Inst(    .Clock(         Clock),
                                    .Reset(         Reset),
                                    .Set(           Snooze & AlarmSounding),
                                    .Enable(        Minute),
                                    .Done(          SnoozeDone));
endmodule


module Divide2Minute(   Clock,          // Clock Signal
                        Reset,          // Power-up Reset
                        Minute          // Goes high for one cycle once per minute
                );

        input           Clock, Reset;
        output          Minute;

        reg     [30:0]  Count;

        // Note that we're assuming a 27MHz clock...
        // assign       Minute =                        (Count == 31'd1619999999);

        // Use a more "reasonable" number to allow simulation to run...
        assign  Minute =                                (Count == 31'd3);

        always @ (posedge Clock) begin
                if (Reset | Minute) Count <=            31'd0;
                else Count <=                           Count + 1;
        end
endmodule


module MinuteCount(     Clock,          // Clock Signal
                        Reset,          // Power-up Reset
                        Enable,         // Count a minute
                        Minute,         // The current minute
                        Rollover        // Goes high when we're about to roll over
                );

        input           Clock, Reset, Enable;
        output  [5:0]   Minute;
        output          Rollover;

        reg     [5:0]   Minute;

        assign  Rollover =                              (Minute == 6'd59);

        always @ (posedge Clock) begin
                if (Reset) Minute <=                    6'd0;
                else if (Enable) begin
                        if (Rollover) Minute <=                 6'd0;
                        else Minute <=                  Minute + 1;
                end
        end
endmodule
```

```
module HourCount(       Clock,          // Clock Signal
                        Reset,          // Power-up Reset
                        Enable,         // Count an hour
                        Hour,           // The current hour
                        Rollover        // Goes high when we're about to roll over
                );

        input           Clock, Reset, Enable;
        output  [4:0]   Hour;
        output          Rollover;

        reg     [4:0]   Hour;

        assign  Rollover =                              (Hour == 5'd23);

        always @ (posedge Clock) begin
                if (Reset) Hour <=                      5'd0;
                else if (Enable) begin
                        if (Rollover) Hour <=           5'd0;
                        else Hour <=                    Hour + 1;
                end
        end
endmodule


module SnoozeDelay(     Clock,          // Clock Signal
                        Reset,          // Power-up Reset
                        Set,            // Start the snooze delay
                        Enable,         // Count a minute
                        Done            // Goes high when the delay has run out
                );

        input           Clock, Reset, Set, Enable;
        output          Done;

        reg     [2:0]   Count;

        assign  Done =                                  (~|Count);

        always @ (posedge Clock) begin
                if (Reset) Count <=                     3'h0;
                else if (Set) Count <=                  3'h5;
                else if (Enable & ~Done) Count <=       Count - 1;
        end
endmodule
```