Question 1

You are to design a module that accumulates statistics on an incoming data stream consisting of two individual bytes. The I/O to the module are as follows:

```verilog
input clock;          // Clock
input reset;          // synchronous reset - active low
input clear;          // Clears statistics when high (synchronous)
input [7:0] DataIn1;  // Input Data 1
input [7:0] DataIn2;  // Input Data 2

// all outputs are registered
output [7:0] EvenParity;  // # of data with Even parity
output [7:0] GreyCode;   // # of data with pattern 10101010 or 01010101
output overflow;        // =1 if any of the counters above overflow
```

Thus, an example of this module running might Behave....

```
clock    1 1 1 1 1 1 1 1 ...... 1 1 1 1 1 1 1 1
clear    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
DataIn1  02 AA AA  ...... 02
DataIn2  03 03 55  ...... 03
EvenParity 00 01 03 05  ...... FF 00
GreyCode  00 00 01 03  ......  
Overflow  0  ...... 0 1
```

Design, verify, synthesize (and reverify after synthesis) a module that meets these specifications. Please turn in the following:
• A drawing of your design.

• A fully commented Verilog listing of your design and test fixture. (Make sure you verify the functionality of overflow.)

• Final report timing summary (from Synopsys).

• Final area (from Synopsys) - use report_area;

• A plot of post-synthesis simulation results with worst case SDF timings. (As appropriate, for off-campus students.)

• The final schematic with your name on it.

[20 points]

Question 2
The purpose of this question is to give you practice in reading and interpreting Verilog. [10 points]

Sketch the logic being described in the following Verilog fragments (c is best captured as a Truth Table):

a.

wire [31:0] A, Abar, out;
assign Abar = ~A;
assign out = {A[31], A[31:1]};

b.

wire [6:0] lines;
wire [7:0] all;
wire [2:0] select;
tri shared;

assign all = {1’bz, lines};
assign shared = all[select[2:0]];

c.

always@A
begin
for (N = 0; N <=7; N = N+1)
  if (A==N) Y[N] = 1;
  else Y[N] = 0;
end
Question 3
The following code obtains the parity for the input A.

```verilog
reg [7:0] A;
wire parity;

assign parity = ^A;
```

Show how you would use a for loop in a procedural block to obtain the same function. Hand in your Verilog code. [5 points]

Question 4
Hand-synthesize the following code fragment using a multiplexor. ie. Draw a schematic showing that this would synthesize too if a 8:1 MUX were used. (Note: It is very unlikely that Synopsys would use an 8:1 mux for this design.)[5 points]

```verilog
input [2:0] X;
input Y, W;
reg E, Ereg;

always@(posedge clock)
  Ereg <= #1 E;

always@(X or Y or W)
  begin
    E = 1’b0;  \ default prevents latches
case(X)
  3'b10x : E = Y;
  3'b111 : E = Y & W;
  endcase
  end
```