1. VHDL analysis: timing diagram

Shown below is the VHDL specification of a circuit. Analyze the VHDL code and complete the timing diagrams on the next page.

ENTITY Prob1 IS
  PORT ( Clock, Resetn, X, R : IN  STD_LOGIC;
         Q, Y, Z : OUT  STD_LOGIC );
END Prob1;

ARCHITECTURE Behavior OF Prob1 IS
  SIGNAL tsg : STD_LOGIC_Vector (1 DOWNTO 0);

BEGIN
  PROCESS ( tsg, R )
  BEGIN
    CASE tsg IS
      WHEN "01" =>
        IF R = '0' THEN
          Z <= '0';
        ELSE
          Z <= '1';
        END IF;
      WHEN OTHERS =>
        Y <= '1';
    END CASE;
  END PROCESS;

  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      tsg <= "01";
      Q <= '0';
    ELSIF ( Clock'EVENT AND Clock = '1') THEN
      CASE tsg IS
        WHEN "00" =>
          IF X = '0' THEN tsg <= "01";
          ELSE tsg <= "11";
          END IF;
        WHEN "01" =>
          IF R = '1' THEN tsg <= "00";
          ELSE tsg <= "11";
          Q <= '1';
        END IF;
        WHEN "11" =>
          tsg <= "01";
          Q <= 0;
        WHEN OTHERS =>
          tsg <= "00";
      END CASE;
    END IF;
  END PROCESS;
END Behavior;
1. Put answer here.

Complete the timing diagram for tsig, Q, Y, and Z. For maximum credit, show delays.
(12 pts)

Name:

- Clock
- Resetn
- R
- X
- tsg[1..0]

Specify tsg[1..0] in binary: 00, 01, 10, 11

Latch

through flipflop

0 1 1 0 0 hold hold hold hold hold hold hold hold hold hold
2. VHDL specification.

Create an 8-bit ALU using a behavioral architecture with the numeric_std package with the following operations:

<table>
<thead>
<tr>
<th>Sel</th>
<th>Output</th>
<th>Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>input1 + input2 (just like Lab 3)</td>
<td>'1' if input1 + input2 is bigger than the maximum number that can be written to output, '0' otherwise (just like Lab 3)</td>
</tr>
<tr>
<td>01</td>
<td>Swap the high-half bits of input1 with the low-half bits of input1, write this to output (just like Lab 3)</td>
<td>'0' if the result is “00000000”, '1' otherwise (unlike Lab 3)</td>
</tr>
<tr>
<td>10</td>
<td>One's complement of input1: Invert each bit</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>Two's complement of input1: One's complement plus 1</td>
<td>0</td>
</tr>
</tbody>
</table>

The entity specification is as follows:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_ns is
  port ( input1 : in std_logic_vector(7 downto 0);
         input2 : in std_logic_vector(7 downto 0);
         sel : in std_logic_vector(1 downto 0);
         output : out std_logic_vector(7 downto 0);
         overflow : out std_logic);
end alu_ns;
```

You are to complete the following ARCHITECTURE specification (below and on the next page) to implement this ALU.

ARCHITECTURE Behavior OF alu_ns IS -- answer for Problem 2
-- SIGNAL definitions

```vhdl
SIGNAL tempIN1, tempIN2, tempOUT : unsigned(8 downto 0);
CONSTANT zero : std_logic_vector(7 downto 0) := X"00";
SIGNAL swapPOV : std_logic;
```
2. (continued)

BEGIN – You cannot use a PROCESS block

WITH sel SELECT – You must use this statement to define the (temp) ALU output.

\[
\text{tempout} <= \text{tempinp}1 + \text{tempinp}2 \text{ when "00",}
\]
\[
\text{ROTATE_LEFT (tempinp1, 4) when "01",}
\]
\[
\text{NOT (tempinp) when "10",}
\]
\[
\text{NOT (tempinp)+1 when "11",}
\]
\[
\text{zero when OTHERS;}
\]

-- You must use a conditional assignment to define the overflow signal.
-- You can use simply assignment statements for other signals.

\[
\text{tempinp1} <= \text{unsigned ('0' & input1);}
\]
\[
\text{tempinp2} <= \text{unsigned ('0' & input2);}
\]
\[
\text{swapov} <= '0' \text{ WHEN tempout = zero ELSE '1';}
\]
\[
\text{overflow} <=
\]
\[
\text{tempout(8) when sel = "00" ELSE swapov when sel = "01" ELSE '0';}
\]
\[
\text{output} <= \text{std_logic_vector(tempout(7 downto 0));}
\]

END Behavior;
The following VHDL code is used for Problem 3 and Problem 4:

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY Test1P3_4 IS
  PORT ( Clock, Resetn : IN STD_LOGIC;
         Z1, Z2, Z3 : OUT STD_LOGIC;
         Q, QQA, QQB, QQc : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END Test1P3_4;

ARCHITECTURE Behavior OF Test1P3_4 IS
  SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN

  PROCESS ( Clock, Resetn )
  BEGIN
    Q <= Count;

    IF (Count = "0000") THEN -- This IF statement is used for Problem 4(a)
      Z1 <= '1';
    ELSE
      Z1 <= '0';
    END IF;

    IF Resetn = '0' THEN
      Count <= "0000";
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      FOR i IN 0 TO 3 LOOP
        Count <= Count + 1;
        Z2 <= Count(0);
      END LOOP;
    END IF;

    IF (Count = "0000") THEN -- This IF statement is used for Problem 4(b)
      Z3 <= '1';
    ELSE
      Z3 <= '0';
    END IF;

    ELSE
      QQA <= Count;
    END IF;
    QQB <= Count;
    END PROCESS;
    QQc <= Count;
  END Behavior;
3. VHDL analysis – Timing diagram.

(a) Based on the code shown on Page 5, complete the following timing diagram for the values for Count (in binary): (5 pts)

(b) Assume the given Count values, complete the following timing diagram based on the code shown on Page 8 (i.e., don’t use the code to figure out the Count values, they are given to you for this part of the problem). Give all values in binary. (18 pts)
4. VHDL Analysis and re-code

(a) Given the IF statement at the top of the code shown on Page 5 (for Z1), convert it into a conditional assignment statement. (4 pts)

IMPORTANT: If necessary, you need to tell me what else you need to do to maintain the behavior of the original code (e.g., location of the new code, add additional code, etc.).

```vhdl
Z <= '1' WHEN COUNT = "0000" 
  ELSE '0';

has to be placed outside of the PROCESS block
```

(b) Given the IF statement in the middle of the code shown on Page 5 (for Z3), convert it into a SELECT assignment statement. (6 pts)

IMPORTANT: If necessary, you need to tell me what else you need to do to maintain the behavior of the original code (e.g., location of the new code, add additional code, etc.).

```vhdl
The SELECT assignment statement cannot be placed inside a PROCESS block.
However, the original code creates a flip-flop. We need a separate PROCESS statement to do so.

WITH COUNT SELECT
  TEMPZ <= '1' WHEN "0000" -- equivalent to
            '0' WHEN OTHERS; -- logic of original
            '1' IF statement

PROCESS (clock)
  IF (clock'EVENT AND clock = '1')
    THEN Z3 <= TEMPZ;
  END IF;
END PROCESS;
```
5. VGA Lab

![Vertical and Horizontal Synchronization Diagram]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>31.77 µs</td>
<td>3.77 µs</td>
<td>1.89 µs</td>
<td>25.17 µs</td>
<td>0.94 µs</td>
</tr>
</tbody>
</table>

Figure 1. Horizontal Refresh Cycle.

Given above is the horizontal refresh cycle for the VGA lab. Briefly define the following:

(a) **A**: Total time to output 1 row of pixels with blanking intervals
(b) **B**: Width of HORIZ_SYNC pulse to start a new row
(c) **C**: Part of the blanking interval
(d) **D**: Time to output 640 pixels of a row
(e) **E**: Part of the blanking interval

6. VHDL test bench.

Given the following entity definition for a generic adder:

```
entity adder is
  generic (WIDTH : positive := 8);
  port map (input1, input2 : in std_logic_vector (WIDTH-1 downto 0);
    carry_in : in std_logic;
    sum : out std_logic_vector (WIDTH-1 downto 0);
    carry_out: out std_logic);
end adder;
```

Modify the test bench on the next page:

- To test the "adder" entity as a 16-bit adder.
- Make the testbench code general; i.e., can be used to test for any number of bits by changing only the value of NBITS.
6. (continued). This is a copy of the adder_tb that you used in the labs, except:
   - I made changes to it (everything that is **bolded**).
   - You need to make all necessary changes to make it work with the generic adder from
     the previous page (**put changes right this page**)

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder_tb is
  end adder_tb;

architecture TB of adder_tb is

  constant NBITS : positive := 16;

  signal input1, input2, sum : std_logic_vector(1 downto 0);
  signal carry_in, carry_out : std_logic;

begin
  -- TB

  UUT : entity work.adder
      port map (       GENERIC MAP
      input1 => A,       (WIDTH => NBITS)
      input2 => B,       end map (WIDTH
      carry_in => C,       NBITS - 1)
      sum => D,        (2**NBITS) - 1
      carry_out => E);       end process


  -- function definitions here

  process
  begin
    for i in 0 to 15 loop
      for j in 0 to 15 loop
        for k in 0 to 1 loop
          A <= std_logic_vector(to_unsigned(i, 4));
          B <= std_logic_vector(to_unsigned(j, 4));
          C <= std_logic(to_unsigned(k, 1)(0));
          wait for 40 ns;
          D <= sum ;
          assert sum = sum_test(i, j, k) report "Sum incorrect";
          assert carry_out = carry_test(i, j, k) report "Carry incorrect";
        end loop;
      end loop;
    end loop;
    report "SIMULATION FINISHED!";
    wait;
  end process;
end

function X (constant a, b, c : integer) return std_logic is
begin
  if (a + b + c) mod 2 = 1 then return '1';
  else return '0';
end if;
end X;

function Y (constant a, b, c : integer) return std_logic is
begin
  return std_logic_vector(to_unsigned((a + b + c) mod 10, NBITS));
end Y;

function Z (constant a, b, c : integer) return std_logic_vector is
begin
  return std_logic_vector(to_unsigned((a + b + c) mod 16, NBITS));
end Z;
ENTITY __entity_name IS
  PORT(_input_name, __input_name
       _input_vector_name
       _bidir_name, __bidir_name
       _output_name, __output_name
  END __entity_name;

ARCHITECTURE a OF __entity_name IS
  SIGNAL __signal_name : STD_LOGIC;
  SIGNAL __signal_name : STD_LOGIC;
BEGIN
  -- Process Statement
  -- Concurrent Signal Assignment
  -- Conditional Signal Assignment
  -- Selected Signal Assignment
  -- Component Instantiation Statement
END a;

SIGNAL __signal_name : __type_name;

__instance_name: __component_name GENERIC MAP (__component_par => __connect_par)
PORT MAP (__component_port => __connect_port,
          __component_port => __connect_port);

WITH __expression SELECT
  __signal <= __expression WHEN __constant_value,
          __expression WHEN __constant_value,
          __expression WHEN __constant_value,
          __expression WHEN __constant_value;

  __signal <= __expression WHEN __boolean_expression ELSE
          __expression WHEN __boolean_expression ELSE
          __expression;

IF __expression THEN
  __statement;
  __statement;
ELSIF __expression THEN
  __statement;
  __statement;
ELSE
  __statement;
  __statement;
END IF;

WAIT UNTIL __expression;

CASE __expression IS
  WHEN __constant_value =>
    __statement;
    __statement;
  WHEN __constant_value =>
    __statement;
    __statement;
  WHEN OTHERS =>
    __statement;
    __statement;
END CASE;

Name __________________________

<optional_label>:
  FOR <loop_id> IN <range> LOOP
    -- Sequential Statement(s)
  END LOOP;

<generate_label>:
  FOR <loop_id> IN <range> GENERATE
    -- Concurrent Statement(s)
  END GENERATE;
1.3 OVERLOADED OPERATORS

<table>
<thead>
<tr>
<th>Description</th>
<th>Left</th>
<th>Operands</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>bnanu and</td>
<td>u1</td>
<td>u1, u1v</td>
<td>u1, u1v</td>
</tr>
<tr>
<td>bnanu-or</td>
<td>u1</td>
<td>u1, u1v</td>
<td>u1, u1v</td>
</tr>
<tr>
<td>bnanu-xor</td>
<td>u1</td>
<td>u1, u1v</td>
<td>u1, u1v</td>
</tr>
<tr>
<td>bnanu-not</td>
<td>u1</td>
<td>u1, u1v</td>
<td>u1, u1v</td>
</tr>
</tbody>
</table>

1.4 CONVERSION FUNCTIONS

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1l</td>
<td>b</td>
<td>TO_BITVECTOR(u1l, x1mp)</td>
</tr>
<tr>
<td>u1v</td>
<td>bv</td>
<td>TO_BITVECTOR(u1v, x1mp)</td>
</tr>
<tr>
<td>u2</td>
<td>u1</td>
<td>TO_BITVECTOR(u2, x1mp)</td>
</tr>
<tr>
<td>b</td>
<td>u1</td>
<td>TO_BITVECTOR(u1, x1mp)</td>
</tr>
<tr>
<td>b</td>
<td>u2</td>
<td>TO_BITVECTOR(u2, x1mp)</td>
</tr>
</tbody>
</table>

2. IEEE's STD_LOGIC

2.1 PREDEFINED TYPES

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIGNED</th>
<th>UNSIGNED</th>
<th>RESERVED</th>
<th>USER_ID</th>
<th>-bit宽范围</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
</tbody>
</table>

2.2 OVERLOADED OPERATORS

<table>
<thead>
<tr>
<th>Left</th>
<th>Op</th>
<th>Right</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1v</td>
<td>+</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>-</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>*</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>/</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>%</td>
<td>u1v</td>
<td>u1v</td>
</tr>
</tbody>
</table>

2.3 PREDEFINED FUNCTIONS

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIGNED</th>
<th>UNSIGNED</th>
<th>RESERVED</th>
<th>USER_ID</th>
<th>bit宽范围</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
</tbody>
</table>

3. IEEE's NUMERIC_BIT

3.1 PREDEFINED TYPES

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIGNED</th>
<th>UNSIGNED</th>
<th>RESERVED</th>
<th>USER_ID</th>
<th>-bit宽范围</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
</tbody>
</table>

3.2 OVERLOADED OPERATORS

<table>
<thead>
<tr>
<th>Left</th>
<th>Op</th>
<th>Right</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1v</td>
<td>+</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>-</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>*</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>/</td>
<td>u1v</td>
<td>u1v</td>
</tr>
<tr>
<td>u1v</td>
<td>%</td>
<td>u1v</td>
<td>u1v</td>
</tr>
</tbody>
</table>

3.3 PREDEFINED FUNCTIONS

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIGNED</th>
<th>UNSIGNED</th>
<th>RESERVED</th>
<th>USER_ID</th>
<th>bit宽范围</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
<tr>
<td>LIST</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>STRING</td>
<td>8</td>
</tr>
</tbody>
</table>

3.4 CONVERSION FUNCTIONS

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1v</td>
<td>bv</td>
<td>TO_BITVECTOR(u1v, x1mp)</td>
</tr>
<tr>
<td>u1v</td>
<td>u1v</td>
<td>TO_BITVECTOR(u1v, x1mp)</td>
</tr>
<tr>
<td>u1v</td>
<td>u2</td>
<td>TO_BITVECTOR(u2, x1mp)</td>
</tr>
<tr>
<td>u2</td>
<td>u1</td>
<td>TO_BITVECTOR(u2, x1mp)</td>
</tr>
<tr>
<td>u2</td>
<td>u2</td>
<td>TO_BITVECTOR(u2, x1mp)</td>
</tr>
<tr>
<td>b</td>
<td>u1</td>
<td>TO_BITVECTOR(u1, x1mp)</td>
</tr>
<tr>
<td>b</td>
<td>u2</td>
<td>TO_BITVECTOR(u2, x1mp)</td>
</tr>
<tr>
<td>b</td>
<td>u2</td>
<td>TO_BITVECTOR(u2, x1mp)</td>
</tr>
</tbody>
</table>

©1995-2000 Qualis Design Corporation

© 1995-2000 Qualis Design Corporation. Permission to reproduce and distribute this material is hereby granted.

See reverse side for additional information.
IMPORTANT: Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.

COVER SHEET:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (22 pts)</td>
<td></td>
</tr>
<tr>
<td>2 (20 pts)</td>
<td></td>
</tr>
<tr>
<td>3 (23 pts)</td>
<td></td>
</tr>
<tr>
<td>4 (10 pts)</td>
<td></td>
</tr>
<tr>
<td>5 (5 pts)</td>
<td></td>
</tr>
<tr>
<td>6 (20 pts)</td>
<td></td>
</tr>
</tbody>
</table>

Total

Re-Grade Information:

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________