Solutions

2003

E3.06 VHDL & Logic Synthesis

Examiners: T. J. W. Clarke & P. Y. K. Cheung

A C 5, I & 7.5

Solution to Question 1

This question carries 40% of total marks, and is compulsory. It examines the core material of the course as follows:

(a) - (c) test whether the student understands the use of signals, variables and control flow in synthesizable VHDL processes, and the ability to write simple hardware descriptions.

(e) & (f) test whether the student understands the operation of VHDL testbenches. g) tests whether the student understands how to describes FSMs in VHDL

(a) many implementations are possible - here is one:

```
ARCHITECTURE rtl of variable_count IS
BEGIN

P1: PROCESS
BEGIN
   WAIT UNTIL clk'EVENT AND clk='0'; -- negative edge
   IF    (rst = '1') OR
         ((sel = '1') AND count = conv_std_logic_vector(10,4)) OR
         ((sel = '0') AND count = conv_std_logic_vector(9,4))
   THEN
         count <= (OTHERS=>'0');
   ELSE
         count <= unsigned(count)+1;
   END IF;
END PROCESS P1;

END ARCHITECTURE rtl;
```

(b)

Input signals of a process occur on RHS of a concurrent assignment or in a conditional expression. In a well formed combinational process all input signals are in the sensitivity list. Output signals are those driven by the process (on LHS of concurrent assignments).

Conditions for correct synthesis:

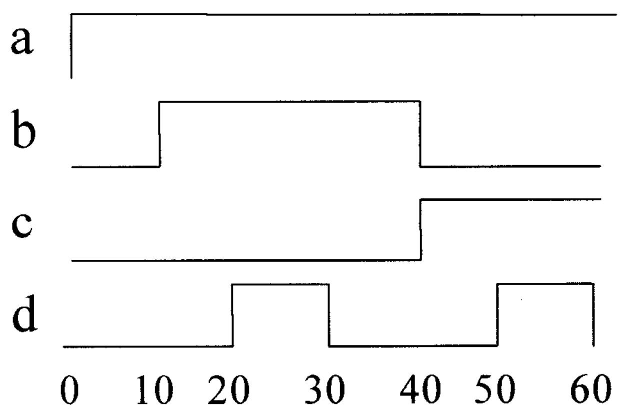
1) all inputs are in sensitivity list

2) all outputs are driven in all execution paths through the process body.

(c) Some care is needed to ensure that all + operations are 10 bit. In theory the addition could be broken into 2 9 bit additions followed by 1 10 bit addition, thus saving 2 full adder stages. This is not required for full marks.

```
ARCHITECTURE rtl OF adder IS
BEGIN
P1: PROCESS(a,b,c,d)
  VARIABLE x_v: std_logic_vector(9 DOWNTO 0);
BEGIN
  pos <= '0'; -- default;
  x_v := signed(a(7) & (a(7) & a)) + signed(b); -- 9 bit addition
  x_v := (signed(x_v) + signed(c)) + signed(d);
  x <= x_v;
  IF signed(x_v) > 0 THEN
       pos <= '1';
  END IF;
END PROCESS P1;
END ARCHITECTURE rtl;
```

(d) Times in ns. Note no glitch on *a* at 30ns.



(e)

An event has non-zero delta if the event is generated from a signal assignment with zero physical delay time. Delta delays can result in flip-flop input data changing before the flip-flop is clocked, where the clock is delayed by 2 delta or more, e.g. passes through two signal assignments:

```
clk1 <= clk;
clk2 <= clk1;

FF1: PROCESS
BEGIN
  WAIT UNTIL clk'EVENT AND clk='1';
  q <= d;
END PROCESS FF1;

FF2: PROCESS -- this process simulates incorrectly
BEGIN
  WAIT UNTIL clk2'EVENT AND clk2='1';
  q2 <= d2;
END PROCESS FF2;
```

(f)

Exhaustive testing would check correct output for every possible input bit pattern. In this case there are 4 8 bit inputs => 2**32 patterns. This would take a very long time to simulate. In this case a better strategy would be random testing (an independent random number for each input) for say 10000 tests together with explicit checking for all inputs at max or min values (16 cases in all).

(g)

```
ARCHITECTURE rtl OF fsm IS

TYPE state IS (s1, s2, s3);

SIGNAL ss, ss_next: state;
BEGIN

   P1: PROCESS(ss,x)
   BEGIN
     -- default values
     a <= '0'; b <= '1';
     ss_next <= ss;
     CASE ss IS

     WHEN s1 =>
       a <= '1';
        IF x='0' THEN ss_next <= s2; ELSE ss_next <= s3; END IF;

     WHEN s2 =>
       IF x='0' THEN ss_next <= s3; ELSE ss_next <= s1; END IF;

     WHEN s3 =>
       b <= '0';
       IF x='0' THEN ss_next <= s2; END IF;
     END CASE;
   END PROCESS P1;

   P2: PROCESS
   BEGIN
     WAIT UNTIL clk'EVENT AND clk='1';
     IF reset = '1' THEN
       ss <= s1;
     ELSE
       ss <= ss_next;
     END IF;
   END PROCESS P2;

END ARCHITECTURE rtl;
```

Solution to Question 2.

This question tests the student's ability to write synthesisable VHDL code. c) tests ability to design at RTL block level and d) tests ability to use structural VHDL.

(a)

```
FUNCTION sign_mag_greater(
        a: std_logic_vector(15 DOWNTO 0);
        b: std_logic_vector(15 DOWNTO 0)) RETURN BOOLEAN IS
BEGIN
  CASE std_logic_vector'( a(15), b(15)) IS
   WHEN "01" => RETURN TRUE;
   WHEN "10" => RETURN FALSE;
   WHEN "00" => RETURN unsigned(a) > unsigned(b);
   WHEN "11" => RETURN unsigned(a) < unsigned(b);
   WHEN OTHERS => RETURN FALSE;
  END CASE;
END FUNCTION sign_mag_greater;
```

(b)

```
ENTITY sorter_element IS
      PORT( clk: IN std_logic;
             mode: IN std_logic_vector( 1 DOWNTO 0);
            in_l, in_h: IN std_logic_vector( 15 DOWNTO 0);
            out_l, out_h: OUT std_logic_vector( 15 DOWNTO 0);
            unchanged: OUT std_logic);
END sorter_element;
```

```
ARCHITECTURE rtl OF sorter_element IS
      SIGNAL data_h, data_l, data_h_d, data_l_d:
         std_logic_vector(15 DOWNTO 0);
      SIGNAL unchanged_int: std_logic;
   BEGIN
         COMB: PROCESS(data_h,data_l, in_h, in_l, mode)
         BEGIN
         unchanged_int <= '0';
         data_h_d <= data_h;
         data_l_d <= data_l;
         CASE mode IS
         WHEN "00" =>
               IF sign_mag_greater(data_l,  data_h)
               THEN
                     data_h_d <= data_l;
                     data_l_d <= data_h;
                     unchanged_int <= '1';
               END IF;
         WHEN "10" =>
               data_l_d <= data_h;
               data_h <= in_h;
         WHEN "01" =>
               IF sign_mag_greater( data_h, in_h) THEN
                     data_h_d <= in_h;
                     unchanged_int <= '1';
               END IF;
               IF sign_mag_greater( in_l, data_l) THEN
                     data_l_d <= in_l;
                     unchanged_int  <= '1';
               END IF;
         WHEN OTHERS => data_h_d <= (OTHERS =>'-');
                        data_h_l <= (OTHERS =>'-');
                        unchanged_int <= '-';
         END CASE;
         END PROCESS COMB;

         FF: PROCESS
         BEGIN
         WAIT UNTIL clk'EVENT AND clk='1';
         data_h <= data_h_d;
         data_h <= data_l_d;
         END PROCESS FF;

         unchanged <= unchanged_int;
         out_h <= data_h;
         out_l <= data_l;
 END ARCHITECTURE rtl;
```

(c)

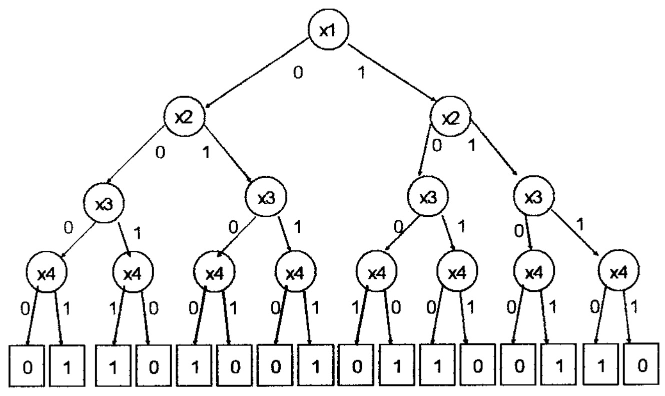
```
ARCHITECTURE struct OF sorter IS

  TYPE array64 IS ARRAY (0 TO 64) OF std_logic_vector(15 DOWNTO 0);
  SIGNAL z1, z2: array64;
  SIGNAL x: std_logic_vector(63 DOWNTO 0);
  CONSTANT kmax: std_logic_vector(15 DOWNTO 0) := (15=>'0',
OTHERS=>'1');
  CONSTANT kmin: std_logic_vector(15 DOWNTO 0) := (OTHERS=>'1');


BEGIN
  G1: FOR k IN 0 TO 63 GENERATE
       I1: ENTITY WORK.sorter_element PORT MAP(
            clk=>clk,
            mode=>m,
            in_h => z1(k),
            out_h => z2(k),
            in_l => z1(k+1),
            out_l => z2(k+1),
            unchanged => x(k));
  END GENERATE;


ANDPROC: PROCESS(x)
  VARIABLE y: std_logic := '1';
  BEGIN
       FOR k IN 0 TO 63 LOOP
            y := y AND x(k);
       END LOOP;
       a <= y;
  END PROCESS ANDPROC;

  MUXPROC: PROCESS(m,shift_in)
  BEGIN
     IF m = "10" THEN
       z1(0) <= shift_in;
     ELSE
       z1(0) <= kmax;
     END IF;
  END PROCESS MUXPROC;

  shift_out <= z1(64);
  z2(64) <= kmin;
  z2(0) <= shift_in;

END ARCHITECTURE struct;
```

Solution to Question 3

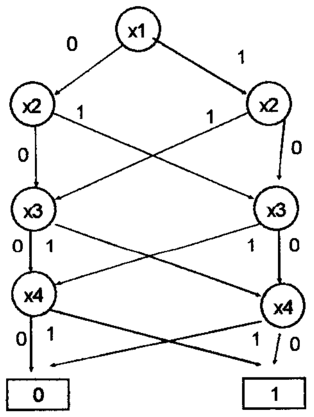
Part (a) tests whether the student can understand BDDs. Part (b) tests whether the student has mastered "many-to-1" circuit description in VHDL, and part (c) tests ability to construct complex structural VHDL descriptions.

(a) $x1 \oplus (x2 \oplus (x3 \oplus x4))$

OBDD



ROBDD



OBDD for *n* input **XOR** has 2^n-1 nodes, corresponding ROBDD has 1+2(n-1) nodes. Ratio is $2^n / (1+2(n-1)) >> 1$ (increases exponentially with *n*).

(b)

```
ARCHITECTURE rtl OF parity IS
BEGIN
   P1: PROCESS(y)
      VARIABLE zi: std_logic;
   BEGIN
      zi := '0';
      FOR j IN 0 TO n-1 LOOP
         zi := zi xor y(j);
      END LOOP;
      z <= zi;
   END PROCESS P1;
END ARCHITECTURE rtl;
```

(c)

```
ARCHITECTURE rtl_struct OF big_parity IS

   CONSTANT Unitnum: INTEGER := (M-1)/10+1;
   CONSTANT ExtraInputs: INTEGER := M - UnitNum*10;
   SIGNAL zi: std_logic_vector(UnitNum DOWNTO 0);

BEGIN

   G1: FOR j IN 0 TO UnitNum-1 GENERATE
      I1: ENTITY WORK.parity GENERIC MAP(10)
             PORT MAP(y(j*10+9 DOWNTO j*10),zi(j));
   END GENERATE;

   G2: IF ExtraInputs /= 0 GENERATE
   I2: ENTITY WORK.parity GENERIC MAP(ExtraInputs)
             PORT MAP(y( M-1 DOWNTO 10*UnitNum), zi(UnitNum));
   END GENERATE;

   G3: IF ExtraInputs /= 0 GENERATE
   I2: ENTITY WORK.parity GENERIC MAP(UnitNum+1)
             PORT MAP(zi(UnitNum DOWNTO 0), z);
   END GENERATE;

   G4: IF ExtraInputs = 0 GENERATE
   I2: ENTITY WORK.parity GENERIC MAP(UnitNum+1)
             PORT MAP(zi(UnitNum-1 DOWNTO 0), z);
   END GENERATE;


END ARCHITECTURE rtl_struct;
```

Solution to Question 4

Part (a) tests whether the student can use VHDL to write simple testbenches. Parts (b), (c) tests whether the student understands verification methodology.

(a)

```
ENTITY tbl IS
END tbl;



ARCHITECTURE behav OF tbl IS

TYPE ipr IS RECORD x1,x2: INTEGER; END RECORD;
TYPE ftype IS FILE OF ipr;
FILE fin: ftype OPEN read_mode IS "stimulus_file";

SIGNAL clk, start_i: std_logic := '0';
SIGNAL done_i: std_logic;
SIGNAL x1_i, x2_i: std_logic_vector(14 DOWNTO 0);
SIGNAL y1_i, y2_i: std_logic_vector(15 DOWNTO 0);

SIGNAL y1_int, y2_int: INTEGER;

BEGIN

   DUT: ENTITY WORK.testable PORT MAP( clk, start_i, done_i, x1_i,
   x2_i,
          y1_i, y2_i);

   clk <= NOT clk AFTER 10 ns;

   y1_int <= conv_integer(unsigned(y1_i));
   y2_int <= conv_integer(unsigned(y2_i));
```

```
  MAIN: PROCESS
     VARIABLE pair: ipr;
  BEGIN
     start_i <= '0';
     FOR I IN 1 TO 1000 LOOP
        WAIT UNTIL clk'EVENT AND clk='1';
     END LOOP;


     WHILE NOT endfile(fin) LOOP
        read( fin, pair);
        x1_i <= conv_std_logic_vector(pair.x1,15);
        x2_i <= conv_std_logic_vector(pair.x2, 15);
         start_i <= '1';
        WAIT UNTIL clk'EVENT AND clk='1';
        start_i <= '0';
        WAIT UNTIL clk'EVENT AND clk='1' AND done_i='1';

        ASSERT y1_int = fa_behave(pair.x1)+fb_behave(pair.x2)
        REPORT "bad y1 output when x1=" & INTEGER'IMAGE(pair.x1) &
         " and x2=" & INTEGER'IMAGE(pair.x2) SEVERITY error;

        ASSERT y2_int = fa_behave(pair.x1)-fb_behave(pair.x2)+2**14
        REPORT "bad y2 output when x1=" & INTEGER'IMAGE(pair.x1) &
         " and x2=" & INTEGER'IMAGE(pair.x2) SEVERITY error;

     END LOOP;
     REPORT "Tests finished" SEVERITY failure;
  END PROCESS MAIN;

END ARCHITECTURE behav;
```

b) There are 2**15 distinct inputs to fa, and the same number for fb. therefore 2**15+2**15-1=2**16 - 1tests are needed: fixed x1 and every x2, then fixed x2 and every value of x1, with the single duplicate test removed. This is much faster than exhaustive testing, and will check that all table values are correct with greater certainty than random testing. However it will not test that the additions to compute y1 & y2 are correct.

c) Need to test all top 8 bit combis of x1,x2, total of 256+256=512 tests. Also need to check the interpolation. Each input is defined by:

(x1high, x1low, x2high, x2low)

where high = top 8 bits, low = bottom 7 bits.

Various strategies are possible, e.g. the following 1024 tests:

(x, 0, 0, 0) (x=0 to 255)

(0, 0, x, 0) (x=0 to 255)

(rnd, x, rnd, rnd) (x=0 to 255)

(rnd, rnd, rnd, x) (x=0 to 255)

Where each rnd is a separate randomly generated number.