ANSWERS - VHDL & Logic Synthesis, May 2000

Answer to question 1

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE WORK.all;

-- this package defines the necessary array types
PACKAGE crossover_pack IS
  TYPE controltype IS ARRAY (1 TO 8) OF std_logic_vector(1 to 3);
END PACKAGE crossover_pack;

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE WORK.crossover_pack.all;
```

a)

```
ENTITY crossover IS
PORT( x: IN std_logic_vector(0 TO 7);
      y: OUT std_logic_vector(0 TO 7);
      control: IN controltype;
      enable: IN std_logic
);
END crossover;
```

c) and b):

```
ARCHITECTURE xx OF crossover IS


  PROCEDURE switch_element( SIGNAL a: std_logic_vector;
                 s: IN INTEGER;
                 enable: IN std_logic;
                 SIGNAL i: IN std_logic;
                 SIGNAL o: OUT std_logic
                ) IS
  BEGIN
    IF enable='1' and a=conv_std_logic_vector( s, 3) THEN
      o <= i;
    ELSE o <= 'Z';
    END IF;
  END PROCEDURE switch_element;

BEGIN
  G1: FOR i IN 1 TO 8 GENERATE
    G2: FOR j IN 1 TO 8 GENERATE
      Switch_element( control(i), i, enable, x(i), y(j));
    END GENERATE;
  END GENERATE;
END xx;
```

Answer to question 2

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY register_file IS
  GENERIC(
    number_of_registers: NATURAL;
    word_size: NATURAL;
    tsetup, thold, tdelay: TIME
  );
  PORT(
    din: IN std_logic_vector( 1 TO word_size);
    dout: OUT std_logic_vector( 1 TO word_size);
    addr: IN INTEGER RANGE 0 to number_of_registers-1;
    cs, write_en, clk, reset: IN std_logic
  );
END register_file;

ARCHITECTURE test_timing OF register_file IS
  SUBTYPE addr_type IS INTEGER RANGE 0 TO number_of_registers-1;
  SUBTYPE word IS std_logic_vector( 1 TO word_size);
  TYPE ram_type IS ARRAY (addr_type) OF word;
  SIGNAL ram: ram_type;
  SIGNAL clk_del, wr_del: std_logic;
BEGIN

  wr_proc: PROCESS(clk)
  BEGIN
    IF reset = '1' THEN
      FOR i IN addr_type LOOP
        FOR j IN 1 to word_size LOOP ram(i)(j) <= '0';
        END LOOP;
      END LOOP;
    ELSIF clk'EVENT and clk='1' and clk'last_value='0' and write_en='1' THEN
      ram(addr) <= din;
    END IF;
  END PROCESS wr_proc;

  outblk: BLOCK (cs='1')
  BEGIN
    dout <= GUARDED ram(addr) AFTER tdelay;
  END BLOCK outblk;

  clk_del <= clk AFTER thold;
  wr_del <= write_en AFTER thold;

  check: PROCESS(clk_del)
  BEGIN
    IF (clk_del'EVENT and clk_del='1' and clk_del'LAST_VALUE='0') THEN
      IF not wr_del'STABLE(thold+tsetup) THEN
         REPORT "wr not stable near clock edge" SEVERITY warning;
      ELSIF wr_del='1' and not addr'STABLE(thold+tsetup) THEN
        REPORT "addr not stable during write cycle" SEVERITY warning;
      ELSIF wr_del='0' and not din'STABLE(thold+tsetup) THEN
        REPORT "din not stable during write cycle" SEVERITY warning;
      END IF;
    END IF;
  END PROCESS check;

END test_timing;
```

Answer to question 3

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE WORK.all;

ENTITY ram_testbench IS
END ram_testbench;
```

b) (with a) inside)

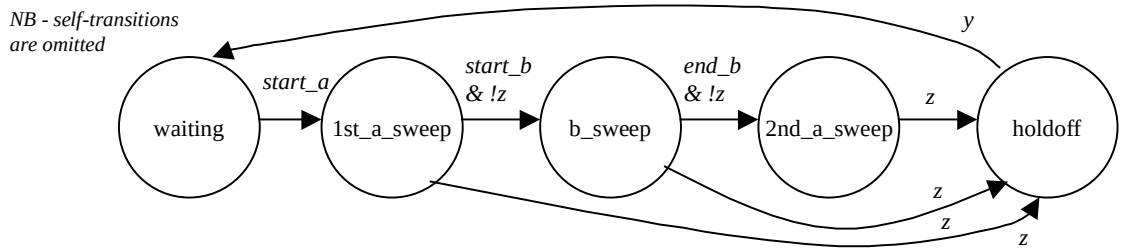
```
ARCHITECTURE examq OF ram_testbench IS

SIGNAL din: std_logic_vector( 1 TO 8);
SIGNAL dout: std_logic_vector( 1 TO 8);
SIGNAL addr: INTEGER RANGE 0 to 9;
SIGNAL cs, write_en, clk, reset: std_logic;

FUNCTION ram_test_data( i: INTEGER)
           RETURN std_logic_vector IS
VARIABLE rand: INTEGER;
BEGIN
  rand := 1;
  For j IN 1 TO i LOOP
    rand := (rand*101+55) mod 100001;
  END LOOP;
  RETURN conv_std_logic_vector( rand, 8);
END;
BEGIN
R1: ENTITY register_file
    GENERIC MAP( 10, 8, 10 ns, 5 ns, 20 ns )
    PORT MAP( din, dout, addr, cs, write_en, clk, reset );

test_data: PROCESS
  BEGIN

    clk <= '0';
    write_en <= '1';
    WAIT FOR 20 ns;

    FOR i IN 0 TO 9 LOOP
      addr <= i;
      din <= ram_test_data(i);
      WAIT FOR 20 ns;
      clk <= '1';
      WAIT FOR 20 ns;
      clk <= '0';

      -- read all ram locations
      FOR j IN 0 TO 9 LOOP
        addr <= j;
        WAIT FOR 21 ns;
        IF j <= i THEN
          ASSERT dout = ram_test_data(j)
          REPORT "Bad data from ram"
          SEVERITY warning;
        ELSE
          ASSERT dout = conv_std_logic_vector(0, 8)
          REPORT "Bad zero locations from ram"
          SEVERITY warning;
        END IF;
      END LOOP;
      --continue to write next location
    END LOOP;
```

```
    cs <= '1';

    WAIT;
  END PROCESS test_data;


  reset<= '1', '0' AFTER 10 ns;

END examq;
```

Answer to Question 4

a)

FSM inputs: x, y, z, start_a, start_b, end_b
FSM outputs:
sample_now= not(holdoff or waiting) & x
en = not(waiting) & x
sel = b_sweep



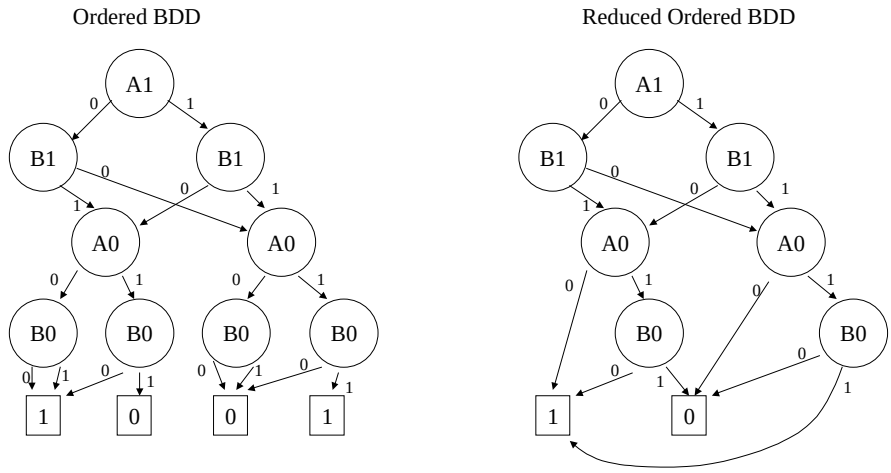
b)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;

ENTITY timebase IS
PORT(
  clk, start_a, start_b, end_b: IN std_logic;
  a, b, h: IN integer range 0 to 1023;
  x_address: OUT std_logic_vector(9 DOWNTO 0);
  sample_now: OUT std_logic
);
END timebase;

ARCHITECTURE xx OF timebase IS

SIGNAL fdivide, ratio, x: INTEGER RANGE 0 TO 1023;


TYPE fsmstate IS ( waiting, first_a_sweep, b_sweep, second_a_sweep, holdoff);


SIGNAL state: fsmstate;

SIGNAL count_this_cycle: BOOLEAN;

BEGIN

  fsm:PROCESS(clk, state)
  BEGIN
    CASE state IS
      WHEN waiting =>
        IF start_a='1'
        THEN state<=first_a_sweep;
        END IF;
      WHEN first_a_sweep =>
```

```
        IF start_b='1'
        THEN state <= b_sweep;
        END IF;
      WHEN b_sweep =>
        IF end_b = '1'
        THEN state<= second_a_sweep;
        END IF;
      WHEN second_a_sweep => null; -- do nothing
      WHEN holdoff =>
        IF h = x THEN
          x <= 0;
          state<= waiting;
        END IF;
    END CASE;

    IF count_this_cycle THEN
      IF x = 1023 THEN
        state <= holdoff;
        x <= 0;
      ELSE
        x <= x+1;
      END IF;
    END IF;

  END PROCESS fsm;

divide_by_n:PROCESS(clk)
  BEGIN
    IF clk'EVENT and clk='1' and clk'LAST_VALUE='0' THEN
      IF fdivide = ratio THEN
        fdivide <= 0;
        count_this_cycle <= TRUE;
      ELSE
        fdivide <= fdivide + 1;
        count_this_cycle <= FALSE;
      END IF;
    END IF;
  END PROCESS divide_by_n;


  select_ratio: PROCESS(a,b,state)
  BEGIN
    IF state=b_sweep THEN
      ratio <= b;
    ELSE
      ratio <= a;
    END IF;
  END PROCESS select_ratio;

  sample_now <=
    '1' WHEN count_this_cycle and not (state = holdoff) ELSE
    '0';

END xx;
```

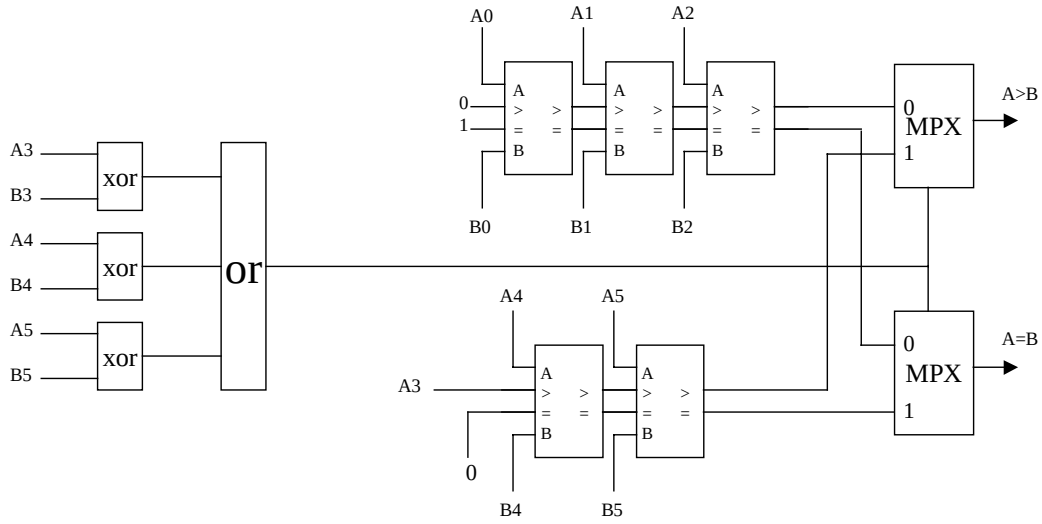
Answer to Question 5

a)



b)

For A>B and A=B to depend on X,Y we must have A3:5=B3:5. Use this condition as select input for two 2 input MPX, one for A>B, one for A=B. If outputs do not depend on Z,Y can generate outputs from three compare blocks, with "$>_i$," and "=" inputs to LS block don't care. Therefore for the LS block "=" must be 0, "$>_i$" must be A_I and the block can be omitted. Max propagation delay=8 units, from A0, B0.



Answer to Question 6

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_signed.all;



ENTITY fp_adder IS
GENERIC(
  e_size: INTEGER := 8;
  m_size: INTEGER := 16
);
PORT(
  a, b: IN std_logic_vector( 0 TO e_size+m_size-1);
  c: OUT std_logic_vector( 0 TO e_size+m_size-1);
  clk: std_logic
);
END fp_adder;

ARCHITECTURE pipeline OF fp_adder IS
SUBTYPE floatw IS std_logic_vector( 0 TO e_size+m_size-1);
SUBTYPE mantissa IS std_logic_vector( e_size TO m_size-1);
SUBTYPE exponent IS std_logic_vector( 0 TO e_size-1);
SUBTYPE exponent_integer IS INTEGER RANGE 0 TO 2**e_size-1;
CONSTANT exp_mod: INTEGER := 2**e_size;

SIGNAL a_swap, b_swap, a_align, b_align: floatw;
SIGNAL add_exp: exponent;
SIGNAL add_mant: std_logic_vector( 0 TO m_size); -- 1 longer than mantissa

-- extract exponent field as positive integer
FUNCTION get_exp( x: floatw) RETURN exponent_integer IS
BEGIN
  RETURN conv_integer( "0" & x(0 TO e_size-1));
END FUNCTION get_exp;

-- extract mantissa as std_logic_vector (for signed ops)
FUNCTION get_mant( x: floatw) RETURN mantissa IS
BEGIN
  RETURN x(e_size TO m_size-1);
END FUNCTION get_mant;


-- could use a library arithmetic shift right - here is
-- a DIY one
FUNCTION my_shr( x: mantissa; sh: INTEGER) RETURN mantissa IS
  VARIABLE result: mantissa;
BEGIN
  FOR i IN mantissa'RANGE LOOP
    IF  i - mantissa'LOW >= sh THEN
      result(i):= x(i-sh);
    ELSE
      result(i) := x(x'LOW);
    END IF;
  END LOOP;
  RETURN result;
END FUNCTION my_shr;
```

```
-- could use library function - here is a simple 1 bit sign extend
FUNCTION sign_extend_1( x: mantissa)
  RETURN std_logic_vector IS
BEGIN
  RETURN x(x'LOW) & x;
END FUNCTION sign_extend_1;



BEGIN

  swap: PROCESS(clk,a,b)
  BEGIN
  -- note unsigned comparison
    IF clk'EVENT and clk='1' and clk'LAST_VALUE='0' THEN
      IF get_exp(a) > get_exp(b) THEN
        a_swap <= a; b_swap <= b;
      ELSE
        a_swap <= b; b_swap <= a;
      END IF;
    END IF;
  END PROCESS swap;

  align: PROCESS(clk,a_swap,b_swap)
  VARIABLE shift: INTEGER;
  BEGIN
  -- a operand has exp >= b operand exp
    IF clk'EVENT and clk='1' and clk'LAST_VALUE='0' THEN
      shift := get_exp(a)- get_exp(b);
      a_align <= a_swap;
      b_align <= a_swap(0 TO e_size-1) &
                 my_shr(get_mant(b_swap), shift);
    END IF;
  END PROCESS align;

  add: PROCESS(clk,a_align,b_align)
  BEGIN
    IF clk'EVENT and clk='1' and clk'LAST_VALUE='0' THEN
    add_mant <= sign_extend_1(get_mant(a_align))+
                sign_extend_1(get_mant(b_align));
    add_exp <= a_align(0 TO e_size-1);
    END IF;
  END PROCESS add;




END pipeline;
```