

APPENDIX A

SAMPLE VHDL TEMPLATES

A.1 GENERAL VHDL CONSTRUCTS

A.1.1 Overall code structure

Listing A.1 Overall code structure

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

5 -- entity declaration
entity bin_counter is
    -- optional generic declaration
    generic(N: integer := 8);
    -- port declaration
10 port(
    clk, reset: in std_logic;           -- clock & reset
    load, en, syn_clr: in std_logic;    -- input control
    d: in std_logic_vector(N-1 downto 0); -- input data
    max_tick: out std_logic;            -- output status
15 q: out std_logic_vector(N-1 downto 0) -- output data
    );
end bin_counter;
```

```

-- architecture body
20 architecture demo_arch of bin_counter is
    -- constant declaration
    constant MAX: integer := (2**N-1);
    -- internal signal declaration
    signal r_reg: unsigned(N-1 downto 0);
25    signal r_next: unsigned(N-1 downto 0);
begin
    =====
    -- component instantiation
    =====
30    -- no instantiation in this code

    =====
    -- memory elements
    =====
35    -- register
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
40        elsif (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
    end process;

45    =====
    -- combinational circuits
    =====
    -- next-state logic
    r_next <= (others=>'0') when syn_clr='1' else
50        unsigned(d) when load='1' else
            r_reg + 1 when en='1' else
            r_reg;
    -- output logic
    q <= std_logic_vector(r_reg);
55    max_tick <= '1' when r_reg=MAX else '0';
end demo_arch;

```

A.1.2 Component instantiation

Listing A.2 Component instantiation template

```

library ieee;
use ieee.std_logic_1164.all;
entity counter_inst is
    port(
5        clk, reset: in std_logic;
        load16, en16, syn_clr16: in std_logic;
        d: in std_logic_vector(15 downto 0);
        max_tick8, max_tick16: out std_logic;

```

```

        q: out std_logic_vector(15 downto 0)
    );
end counter_inst;

architecture structure_arch of counter_inst is
begin
    -- instantiation of 16-bit counter, all ports used
    counter_16_unit: entity work.bin_counter(demo_arch)
        generic map(N=>16)
        port map(clk=>clk, reset=>reset,
            load=>load16, en=>en16, syn_clr=>syn_clr16,
            d=>d, max_tick=>max_tick16, q=>q);
    -- instantiation of free-running 8-bit counter
    -- with only the max_tick signal
    counter_8_unit: entity work.bin_counter
        port map(clk=>clk, reset=>reset,
            load=>'0', en=>'1', syn_clr=>'0',
            d=>"00000000", max_tick=>max_tick8, q=>open);
end structure_arch;

```

A.2 COMBINATIONAL CIRCUITS

A.2.1 Arithmetic operations

Listing A.3 Arithmetic operations

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity arith_demo is
    port(
        a, b: in std_logic_vector(7 downto 0);
        diff, inc: out std_logic_vector(7 downto 0)
    );
end arith_demo;

architecture arch of arith_demo is
    signal au, bu, diffu: unsigned(7 downto 0);
begin
    --=====
    -- convert inputs to unsigned/signed internally
    -- and then convert the result back
    --=====
    au <= unsigned(a);
    bu <= unsigned(b);
    diffu <= au - bu when (au > bu) else
        bu - au;
    diff <= std_logic_vector(diffu);

    --=====
    -- convert multiple times in a statement

```

```

=====
inc <= std_logic_vector(unsigned(a) + 1);
end arch;

```

A.2.2 Fixed-amount shift operations

Listing A.4 Fixed-amount shift operations

```

library ieee;
use ieee.std_logic_1164.all;
entity fixed_shift_demo is
  port(
5      a: in std_logic_vector(7 downto 0);
        sh1, sh2, sh3, rot, swap: out
          std_logic_vector(7 downto 0)
        );
end fixed_shift_demo;
10
architecture arch of fixed_shift_demo is
begin
  -- shift left 3 positions
  sh1 <= a(4 downto 0) & "000" ;
15  -- shift right 3 positions (logical shift)
  sh2 <= "000" & a(7 downto 3);
  -- shift right 3 positions and shifting in sign bit
  -- (arithmetic shift)
  sh3 <= a(7) & a(7) & a(7) & a(7 downto 3);
20  -- rotate right 3 positions
  rot <= a(2 downto 0) & a(7 downto 3);
  -- swap two nibbles
  swap <= a(3 downto 0) & a(7 downto 4);
end arch;

```

A.2.3 Routing with concurrent statements

Listing A.5 Routing with concurrent statements

```

library ieee;
use ieee.std_logic_1164.all;
entity decoder1 is
  port(
5      a: in std_logic_vector(1 downto 0);
        en: in std_logic;
        y1, y2: out std_logic_vector(3 downto 0)
        );
end decoder1;
10
architecture concurrent_arch of decoder1 is
  signal s: std_logic_vector(2 downto 0);
begin
  =====

```

```

15  -- conditional signal assignment statement
    =====
    y1 <= "0000" when (en='0') else
        "0001" when (a="00") else
        "0010" when (a="01") else
20    "0100" when (a="10") else
        "1000";    -- a="11"

    =====
    -- selected signal assignment statement
    =====
25    s <= en & a;
    with s select
        y2 <= "0000" when "000"|"001"|"010"|"011",
            "0001" when "100",
30        "0010" when "101",
            "0100" when "110",
            "1000" when others;    -- s="111"
end concurrent_arch;

```

A.2.4 Routing with if and case statements

Listing A.6 If and case statement templates

```

library ieee;
use ieee.std_logic_1164.all;
entity decoder2 is
    port(
5      a: in std_logic_vector(1 downto 0);
        en: in std_logic;
        y1, y2: out std_logic_vector(3 downto 0)
    );
end decoder2;

10 architecture seq_arch of decoder2 is
    signal s: std_logic_vector(2 downto 0);
begin
    =====
15    -- if statement
    =====
    process(en, a)
    begin
        if (en='0') then
20            y1 <= "0000";
            elsif (a="00") then
                y1 <= "0001";
            elsif (a="01") then
                y1 <= "0010";
25            elsif (a="10") then
                y1 <= "0100";
            else
                y1 <= "1000";

```

```

        end if;
30    end process;

    -----
    -- case statement
    -----
35    s <= en & a;
    process(s)
    begin
        case s is
            when "000"|"001"|"010"|"011" =>
40                y2 <= "0001";
            when "100" =>
                y2 <= "0001";
            when "101" =>
                y2 <= "0010";
45            when "110" =>
                y2 <= "0100";
            when others =>
                y2 <= "1000";
        end case;
50    end process;
    end seq_arch;

```

A.2.5 Combinational circuit using process

Listing A.7 Combinational circuit using process

```

library ieee;
use ieee.std_logic_1164.all;
entity comb_proc is
    port(
5        a, b: in std_logic_vector(1 downto 0);
        data_in: std_logic_vector(7 downto 0);
        xa_out, xb_out: out std_logic_vector(7 downto 0);
        ya_out, yb_out: out std_logic_vector(7 downto 0)
    );
10 end comb_proc;

    architecture arch of comb_proc is
    begin
        -----
15    -- without default output signal assignment
        -----
        -- must include else branch
        -- output signal must be assigned in all branches
        process(a,b,data_in)
20    begin
            if a > b then
                xa_out <= data_in;
                xb_out <= (others=>'0');
            elsif a < b then

```

```

25      xa_out <= (others=>'0');
      xb_out <= data_in;
      else -- a=b
          xa_out <= (others=>'0');
          xb_out <= (others=>'0');
30      end if;
    end process;
    =====
    -- with default output signal assignment
    =====
35    process(a,b,data_in)
    begin
        ya_out <= (others=>'0');
        yb_out <= (others=>'0');
        if a > b then
40            ya_out <= data_in;
        elsif a < b then
            yb_out <= data_in;
        end if;
    end process;
45 end arch;

```

A.3 MEMORY COMPONENTS

A.3.1 Register template

Listing A.8 Register template

```

library ieee;
use ieee.std_logic_1164.all;
entity reg_template is
    port(
5      clk, reset: in std_logic;
        en: in std_logic;
        q1_next, q2_next, q3_next: in
            std_logic_vector(7 downto 0);
        q1_reg, q2_reg, q3_reg: out
10       std_logic_vector(7 downto 0)
    );
end reg_template;

architecture arch of reg_template is
15 begin
    =====
    -- register without reset
    =====
    process(clk)
20    begin
        if (clk'event and clk='1') then
            q1_reg <= q1_next;
        end if;
    end process;
end arch;

```

```

end process;

25
--=====
-- register with asynchronous reset
--=====

process(clk,reset)
30 begin
    if (reset='1') then
        q2_reg <=(others=>'0');
        elsif (clk'event and clk='1') then
            q2_reg <= q2_next;
35     end if;
end process;

--=====
-- register with enable and asynchronous reset
--=====
40
process(clk,reset)
begin
    if (reset='1') then
        q3_reg <=(others=>'0');
45     elsif (clk'event and clk='1') then
        if (en='1') then
            q3_reg <= q3_next;
            end if;
        end if;
50     end process;
end arch;

```

A.3.2 Register file

Listing A.9 Register file

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity reg_file is
5   generic(
        B: integer:=8; -- number of bits
        W: integer:=2 -- number of address bits
    );
    port(
10     clk, reset: in std_logic;
        wr_en: in std_logic;
        w_addr, r_addr: in std_logic_vector (W-1 downto 0);
        w_data: in std_logic_vector (B-1 downto 0);
        r_data: out std_logic_vector (B-1 downto 0)
15    );
end reg_file;

architecture arch of reg_file is
    type reg_file_type is array (2**W-1 downto 0) of

```



```

20      std_logic_vector(B-1 downto 0);
      signal array_reg: reg_file_type;
begin
      process(clk,reset)
      begin
25          if (reset='1') then
              array_reg <= (others=>(others=>'0'));
          elsif (clk'event and clk='1') then
              if wr_en='1' then
                  array_reg(to_integer(unsigned(w_addr))) <= w_data;
30              end if;
          end if;
      end process;
      -- read port
      r_data <= array_reg(to_integer(unsigned(r_addr)));
35 end arch;

```

A.4 REGULAR SEQUENTIAL CIRCUITS

Listing A.10 Sequential circuit template

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity bin_counter is
5   generic(N: integer := 8);
   port(
       clk, reset: in std_logic;
       load, en, syn_clr: in std_logic;
       d: in std_logic_vector(N-1 downto 0);
10      max_tick: out std_logic;
       q: out std_logic_vector(N-1 downto 0)
   );
end bin_counter;

15 architecture demo_arch of bin_counter is
   constant MAX: integer := (2**N-1);
   signal r_reg: unsigned(N-1 downto 0);
   signal r_next: unsigned(N-1 downto 0);

20 begin
   --=====
   -- register
   --=====
   process(clk,reset)
25   begin
       if (reset='1') then
           r_reg <= (others=>'0');
       elsif (clk'event and clk='1') then
           r_reg <= r_next;
30       end if;

```

```

    end process;
    =====
    -- next-state logic
    =====
35    r_next <= (others=>'0') when syn_clr='1' else
        unsigned(d)   when load='1' else
        r_reg + 1      when en='1'  else
        r_reg;
    =====
40    -- output logic
    =====
    q <= std_logic_vector(r_reg);
    max_tick <= '1' when r_reg=MAX else '0';
end demo_arch;

```

A.5 FSM

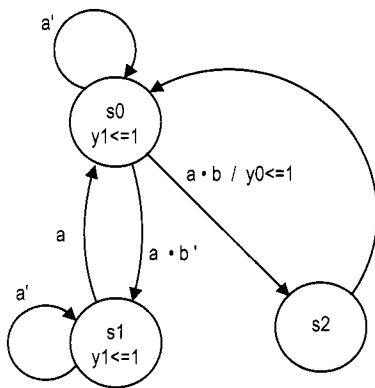
Listing A.11 FSM template

```

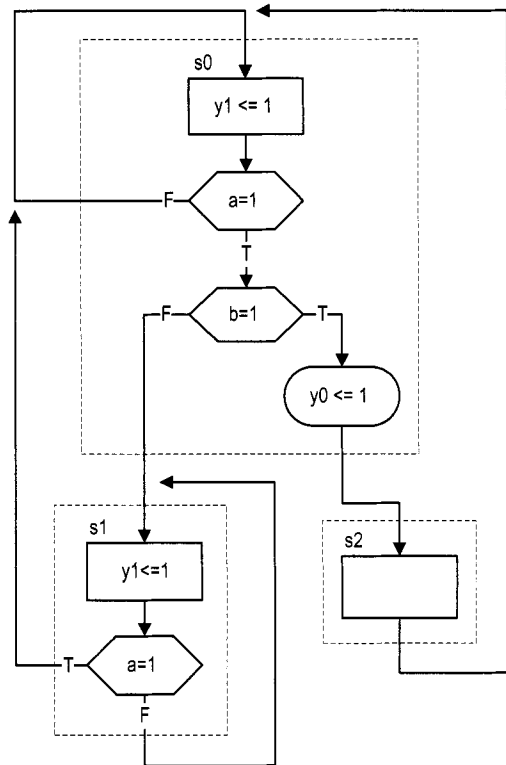
-- code for the FSM in Figure A.1
library ieee;
use ieee.std_logic_1164.all;
entity fsm_eg is
5   port(
        clk, reset: in std_logic;
        a, b: in std_logic;
        y0, y1: out std_logic
    );
10  end fsm_eg;

architecture two_seg_arch of fsm_eg is
    type eg_state_type is (s0, s1, s2);
    signal state_reg, state_next: eg_state_type;
15  begin
    =====
    -- state register
    =====
    process(clk, reset)
20    begin
        if (reset='1') then
            state_reg <= s0;
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
25        end if;
    end process;
    =====
    -- next-state/output logic
    =====
30    process(state_reg, a, b)
    begin
        state_next <= state_reg; -- default back to same state
    end process;

```



(a) State diagram



(b) ASM chart

Figure A.1 State diagram and ASM chart of an FSM template.

```

y0 <= '0';    -- default 0
y1 <= '0';    -- default 0
35  case state_reg is
      when s0 =>
          if a='1' then
              if b='1' then
                  state_next <= s2;
40                  y0 <= '1';
              else
                  state_next <= s1;
              end if;
              -- no else branch, use default
45          end if;
      when s1 =>
          y1 <= '1';
          if (a='1') then
              state_next <= s0;
50          -- no else branch, use default
          end if;
      when s2 =>
          state_next <= s0;
      end case;
55  end process;
end two_seg_arch;

```

A.6 FSM D

Listing A.12 FSM D template

```

-- code for the FSM D shown in Figure A.2
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
5  entity fib is
      port(
          clk, reset: in std_logic;
          start: in std_logic;
          i: in std_logic_vector(4 downto 0);
10         ready, done_tick: out std_logic;
          f: out std_logic_vector(19 downto 0)
      );
end fib;

15  architecture arch of fib is
      type state_type is (idle,op,done);
      signal state_reg, state_next: state_type;
      signal t0_reg, t0_next, t1_reg, t1_next:
          unsigned(19 downto 0);
20  signal n_reg, n_next: unsigned(4 downto 0);
begin
    =====

```

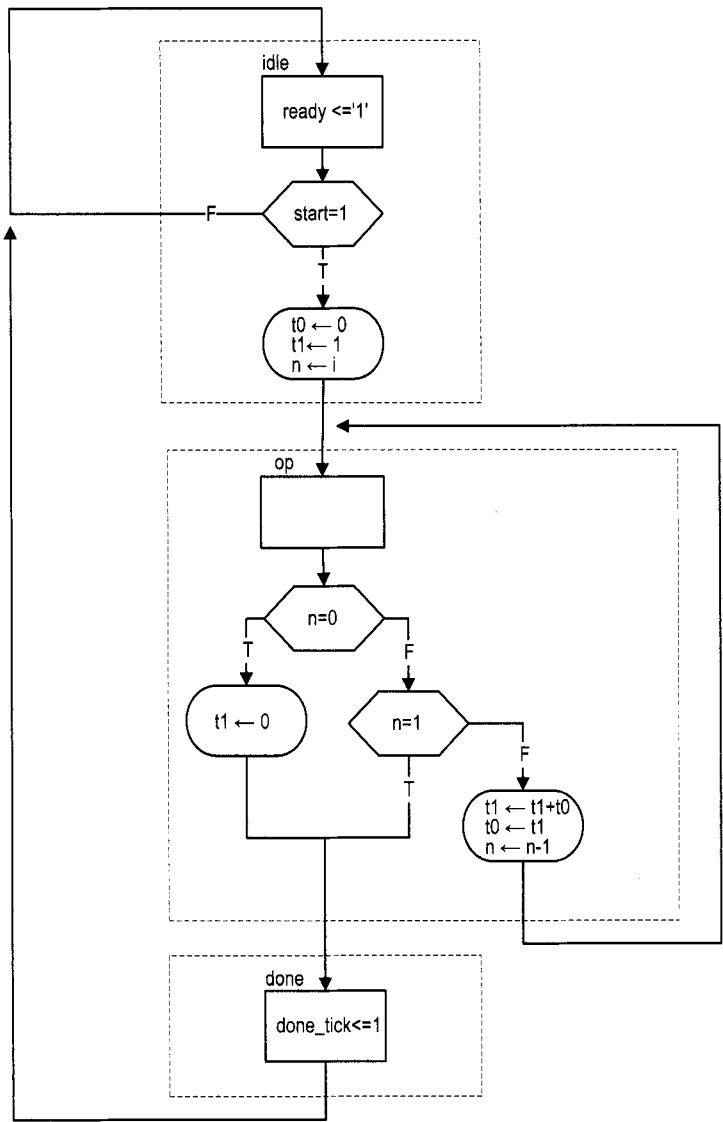


Figure A.2 ASMD chart of an FSMD template.

```

-- state and data registers
=====
25 process (clk, reset)
begin
    if reset='1' then
        state_reg <= idle;
        t0_reg <= (others=>'0');
        t1_reg <= (others=>'0');
30         n_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
            t0_reg <= t0_next;
            t1_reg <= t1_next;
35             n_reg <= n_next;
        end if;
    end process;
    =====
40 -- next-state logic and data path functional units
    =====
    process (state_reg, n_reg, t0_reg, t1_reg, start, i, n_next)
    begin
        ready <= '0';
        done_tick <= '0';
45         state_next <= state_reg; -- default back to same state
        t0_next <= t0_reg; -- default keep previous value
        t1_next <= t1_reg; -- default keep previous value
        n_next <= n_reg; -- default keep previous value
50         case state_reg is
            when idle =>
                ready <= '1';
                if start='1' then
                    t0_next <= (others=>'0');
                    t1_next <= (0=>'1', others=>'0');
55                     n_next <= unsigned(i);
                    state_next <= op;
                end if;
            when op =>
                if n_reg=0 then
60                     t1_next <= (others=>'0');
                    state_next <= done;
                elsif n_reg=1 then
                    state_next <= done;
                else
65                     t1_next <= t1_reg + t0_reg;
                    t0_next <= t1_reg;
                    n_next <= n_reg - 1;
                end if;
            when done =>
70                 done_tick <= '1';
                    state_next <= idle;
                end case;
        end process;
75 -- output

```

```

    f <= std_logic_vector(t1_reg);
end arch;

```

A.7 S3 BOARD CONSTRAINT FILE (S3.UCF)

```

#=====
#   Pin assignment for Xilinx
#   Spartan-3 Starter board
#=====

#=====
# clock and reset
#=====
NET "clk"      LOC = "T9" ;
NET "reset"    LOC = "L14";

#=====
# buttons & switches
#=====
# 4 push buttons
NET "btn<0>"    LOC = "M13";
NET "btn<1>"    LOC = "M14";
NET "btn<2>"    LOC = "L13";
NET "btn<3>"    LOC = "L14";  #btn<3> also used as reset

# 8 slide switches
NET "sw<0>"     LOC = "F12";
NET "sw<1>"     LOC = "G12";
NET "sw<2>"     LOC = "H14";
NET "sw<3>"     LOC = "H13";
NET "sw<4>"     LOC = "J14";
NET "sw<5>"     LOC = "J13";
NET "sw<6>"     LOC = "K14";
NET "sw<7>"     LOC = "K13";

#=====
# RS232
#=====
NET "rx"        LOC = "T13" | DRIVE=8 | SLEW=SLOW;
NET "tx"        LOC = "R13" | DRIVE=8 | SLEW=SLOW;

#=====
# 4-digit time-multiplexed 7-segment LED display
#=====
# digit enable
NET "an<0>"     LOC = "D14";
NET "an<1>"     LOC = "G14";
NET "an<2>"     LOC = "F14";
NET "an<3>"     LOC = "E13";

# 7-segment led segments

```

```

NET "sseg<7>" LOC = "P16"; # decimal point
NET "sseg<6>" LOC = "E14"; # segment a
NET "sseg<5>" LOC = "G13"; # segment b
NET "sseg<4>" LOC = "N15"; # segment c
NET "sseg<3>" LOC = "P15"; # segment d
NET "sseg<2>" LOC = "R16"; # segment e
NET "sseg<1>" LOC = "F13"; # segment f
NET "sseg<0>" LOC = "N16"; # segment g

#=====
# 8 discrete LEDs
#=====
NET "led<0>" LOC = "K12";
NET "led<1>" LOC = "P14";
NET "led<2>" LOC = "L12";
NET "led<3>" LOC = "N14";
NET "led<4>" LOC = "P13";
NET "led<5>" LOC = "N12";
NET "led<6>" LOC = "P12";
NET "led<7>" LOC = "P11";

#=====
# VGA outputs
#=====
NET "rgb<2>" LOC = "R12" | DRIVE=8 | SLEW=FAST;
NET "rgb<1>" LOC = "T12" | DRIVE=8 | SLEW=FAST;
NET "rgb<0>" LOC = "R11" | DRIVE=8 | SLEW=FAST;
NET "vsync" LOC = "T10" | DRIVE=8 | SLEW=FAST;
NET "hsync" LOC = "R9" | DRIVE=8 | SLEW=FAST;

#=====
# PS2 port
#=====
NET "ps2c" LOC="M16" | DRIVE=8 | SLEW=SLOW;
NET "ps2d" LOC="M15" | DRIVE=8 | SLEW=SLOW;

#=====
# two SRAM chips
#=====
# shared 18-bit memory address
NET "ad<17>" LOC="L3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<16>" LOC="K5" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<15>" LOC="K3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<14>" LOC="J3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<13>" LOC="J4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<12>" LOC="H4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<11>" LOC="H3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<10>" LOC="G5" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<9>" LOC="E4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<8>" LOC="E3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<7>" LOC="F4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<6>" LOC="F3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "ad<5>" LOC="G4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;

```



```

100 NET "ad<4>"      LOC="L4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
    NET "ad<3>"      LOC="M3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
    NET "ad<2>"      LOC="M4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
    NET "ad<1>"      LOC="N3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
    NET "ad<0>"      LOC="L5" | IOSTANDARD = LVCMOS33 | SLEW=FAST;

```

```
# shared oe, we
```

```

NET "oe_n"      LOC="K4" | IOSTANDARD = LVCMOS33 | SLEW=FAST;
NET "we_n"      LOC="G3" | IOSTANDARD = LVCMOS33 | SLEW=FAST;

```

```
# sram chip 1 data, ce, ub, lb
```

```

NET "dio_a<15>" LOC="R1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<14>" LOC="P1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<13>" LOC="L2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<12>" LOC="J2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<11>" LOC="H1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<10>" LOC="F2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<9>"  LOC="P8" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<8>"  LOC="D3" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<7>"  LOC="B1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<6>"  LOC="C1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<5>"  LOC="C2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<4>"  LOC="R5" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<3>"  LOC="T5" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<2>"  LOC="R6" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<1>"  LOC="T8" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_a<0>"  LOC="N7" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "ce_a_n"    LOC="P7" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "ub_a_n"    LOC="T4" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "lb_a_n"    LOC="P6" | IOSTANDARD=LVCMOS33 | SLEW=FAST;

```

```
# sram chip 2 data, ce, ub, lb
```

```

NET "dio_b<15>" LOC="N1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<14>" LOC="M1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<13>" LOC="K2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<12>" LOC="C3" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<11>" LOC="F5" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<10>" LOC="G1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<9>"  LOC="E2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<8>"  LOC="D2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<7>"  LOC="D1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<6>"  LOC="E1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<5>"  LOC="G2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<4>"  LOC="J1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<3>"  LOC="K1" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<2>"  LOC="M2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<1>"  LOC="N2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "dio_b<0>"  LOC="P2" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "ce_b_n"    LOC="N5" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "ub_b_n"    LOC="R4" | IOSTANDARD=LVCMOS33 | SLEW=FAST;
NET "lb_b_n"    LOC="P5" | IOSTANDARD=LVCMOS33 | SLEW=FAST;

```

```
#=====
```

```
# Timing constraint of S3 50-MHz onboard oscillator
# name of the clock signal is clk
#=====
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 40 ns HIGH 50 %;
```