รายการอ้างอิง

1. Hing-Yip Tong. A single chip micro-computer for A/V monitor and TV Receiver. IEEE Transactions on Consumer Electronics 36 (November 1990): 825-831.

2. กฤษณ์ อธิกุลวงศ์. การพัฒนาตัวประมวลผลคำบรรยายภาพไทย-อังกฤษแบบซ่อนได้. วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย, 2539.

3. Toshiba corporation. Toshiba original cmos 8-bit microcontroller: TLCS-870 series databook. 1st ed. (n.p.), 1995.

4. Hiroyasu Shindo, Tsutomu Nakazawa, Masaya Ohta, Kazuyuki Mitsuya, and Masaru Tonozuka. Microcontrollers for closed caption system. IEEE Transactions on Consumer Electronics 38 (August 1992): 268-273.

5. Tom S. Wei, Andre B. Walker, and Gary K. Pacey. A universal high performance television controller. IEEE Transactions on Consumer Electronics 36 (August 1990): 678-683.

6. เจน สงสมพันธุ์. ปฏิบัติการโทรทัศน์สี. พิมพ์ครั้งที่ 3. ปทุมธานี: สถาบันอิเล็กทรอนิกส์กรุงเทพรังสิต, 2540.

7. Electronic Industries Association. Recommended practice for line 21 data service. Washington, District of Columbia: Electronic Industries Association, 1994.

8. Federal Communications Commission. Report and order on GEN docket No. 91-1. United States of America: Federal Communications Commission, 1991.

9. สายัณห์ ธีรปัญญาวัฒน์ และ เอกชัย ลีลารัศมี. เครื่องแทรกคำบรรยายภาพแบบซ่อนได้ในระบบ PAL. บทความในหนังสือประกอบการประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 18 (พฤศจิกายน 2538): 542-547.

10. Xilinx, Inc. The programmable logic data book 1998. (n.p.), 1997.

ภาคผนวก

ภาคผนวก ก

รายละเอียดของโปรแกรมภาษา VHDL ที่ใช้ออกแบบทีวีไมโครคอนโทรลเลอร์

```
--------------------------------------------------------------------------
-- 10-bit adder/subtractor
--------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity AD_SUB10 is
    port (A    : in std_logic_vector(9 downto 0);
          B    : in std_logic_vector(9 downto 0);
          Cin  : in std_logic;
          ADD  : in std_logic;
          S    : out std_logic_vector(9 downto 0));
end AD_SUB10;

-- behavioral implementation of the 8-bit adder
architecture BEHAVIORAL of AD_SUB10 is
begin
    p1: process(A, B, Cin, ADD)
    variable VSUM : std_logic_vector(9 downto 0);
    variable CARRY : std_logic;
    variable TMP_A : std_logic;
    begin
    CARRY := Cin;
    for i in 0 to 9 loop
        VSUM(i) := (A(i) xor B(i)) xor CARRY;        -- Output of Sum or Subtract

           if (ADD = '0') then                       -- Check operation command
               TMP_A := not(A(i));
           else
               TMP_A := A(i);
           end if;
        CARRY := (TMP_A and B(i)) or (CARRY and (TMP_A or B(i))); -- Carry logic
    end loop;
    S <= VSUM;
    end process p1;
end BEHAVIORAL;

-- ##########################################################################
--------------------------------------------------------------------------
-- ADD PULSE GENERATOR
-- generate pulse for PWM 14-bit
--------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity ADD_PLS is
port(DATA_IN    : in std_logic_vector(6 downto 0);  -- input 7-bit
     COUNT7HIGH : in std_logic_vector(6 downto 0); -- counter 7-bit high order
     COUNT7LOW  : in std_logic_vector(6 downto 0); -- counter 7-bit low order
     nPWM8BIT   : in std_logic; -- input from pwm 7-bit input
     nPWM14BIT  : out std_logic);   -- output pwm 14-bit
end ADD_PLS;

architecture RTL of ADD_PLS is
signal GENPULSE : std_logic;          -- GENERATE PULSE SIGNAL

begin

    process(DATA_IN,COUNT7HIGH,COUNT7LOW,nPWM8BIT)   -- develope from DATA TABLE TLCS-
870 Series
    begin
        if (COUNT7LOW = "1111111") then     -- last cycle of 127
            for i in 0 to 6 loop
                if (DATA_IN(i) = '1') then
                    if (COUNT7HIGH(6-i) = '1') then
                        GENPULSE <= '0';    -- GENPULSE
                    end if;
                end if;
            end loop;
        else                                -- if not last cycle, not GENPULSE
            GENPULSE <= '1';
        end if;
```

```
    end process;

    nPWM14BIT <= nPWM8BIT and GENPULSE;      -- OUTPUT

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 8-bit adder/subtractor
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity ADD_SUB8 is
    port (A    : in std_logic_vector(7 downto 0);
          B    : in std_logic_vector(7 downto 0);
          Cin  : in std_logic;
          ADD  : in std_logic;
          S    : out std_logic_vector(7 downto 0);
          Cout : out std_logic);
end ADD_SUB8;

-- behavioral implementation of the 8-bit adder
architecture BEHAVIORAL of ADD_SUB8 is
begin
    p1: process(A, B, Cin, ADD)
    variable VSUM : std_logic_vector(7 downto 0);
    variable CARRY : std_logic;
    variable TMP_A : std_logic;
    begin
    CARRY := Cin;
    for i in 0 to 7 loop
        VSUM(i) := (A(i) xor B(i)) xor CARRY;        -- Output of Sum or Subtract

            if (ADD = '0') then                      -- Check operation command
                TMP_A := not(A(i));
            else
                TMP_A := A(i);
            end if;
        CARRY := (TMP_A and B(i)) or (CARRY and (TMP_A or B(i)));  -- Carry logic
    end loop;

    S <= VSUM;
    Cout <= not(CARRY xor ADD);
    end process p1;
end BEHAVIORAL;
-- ##############################################################################
------------------------------------------------------------------------
-- 9-bit adder/subtractor
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity ADD_SUB9 is
    port (A    : in std_logic_vector(8 downto 0);
          B    : in std_logic_vector(8 downto 0);
          Cin  : in std_logic;
          ADD  : in std_logic;
          S    : out std_logic_vector(8 downto 0));
end ADD_SUB9;

-- behavioral implementation of the 9-bit adder
architecture BEHAVIORAL of ADD_SUB9 is
begin
    p1: process(A, B, Cin, ADD)
    variable VSUM : std_logic_vector(8 downto 0);
    variable CARRY : std_logic;
    variable TMP_A : std_logic;
    begin
    CARRY := Cin;
    for i in 0 to 8 loop
        VSUM(i) := (A(i) xor B(i)) xor CARRY;        -- Output of Sum or Subtract

            if (ADD = '0') then                      -- Check operation command
```

```
            TMP_A := not(A(i));
        else
            TMP_A := A(i);
        end if;
      CARRY := (TMP_A and B(i)) or (CARRY and (TMP_A or B(i))); -- Carry logic
  end loop;
  S <= VSUM;
  end process p1;
end BEHAVIORAL;

-- ##############################################################################
------------------------------------------------------------------------
-- Generate Address signal
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity ADDR_OSD is
port(COL      : in integer range 0 to 63;
     SUB_COL  : in integer range 0 to 15;
     SUB_ROW  : in integer range 0 to 15;
     THAI     : in std_logic;
     PAGE     : in std_logic;
     DISP_ROW_NO    : in integer range 0 to 3;
     FIELD    : in std_logic;
     CODE     : in std_logic_vector(7 downto 0);
     ADDR     : out std_logic_vector(15 downto 0));
end ADDR_OSD;

architecture RTL of ADDR_OSD is

signal SUB_COL_BIT  : std_logic_vector(3 downto 0);
signal COL_BIT      : std_logic_vector(5 downto 0);
signal DISP_ROW_NO_BIT  : std_logic_vector(1 downto 0);

       -- Integer to signed or unsigned vector
       function int2evec (l: integer; size: integer := 32) return std_logic_vector is
          variable result: std_logic_vector(size-1 downto 0);
          variable op: integer := 1;
          variable neg_fl: boolean := false;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          result := (size-1 downto 0 => '0');
          if op < 0 then
             op := op * (-1);
             neg_fl := true;
          end if;
          for i in 0 to SIZE-1 loop
             if (op mod 2) = 1 then
                result (i) := '1' ;
             end if ;
             op := op/2 ;
          end loop ;
          return result;
       end int2evec;

begin
        COL_BIT         <= int2evec(COL,6);
        SUB_COL_BIT     <= int2evec(SUB_COL,4);
        DISP_ROW_NO_BIT <= int2evec(DISP_ROW_NO,2);

        process(SUB_COL_BIT,COL_BIT,SUB_ROW,CODE,FIELD,DISP_ROW_NO_BIT,THAI,PAGE)
        begin

            if (SUB_COL_BIT(3 downto 1) = "110") then   -- read code at SUB_ROW 12-13
                ADDR(15 downto 12)  <= "1111";
                ADDR(11)            <= THAI;
                ADDR(10)            <= PAGE;
                ADDR(9 downto 8)    <= DISP_ROW_NO_BIT;
                ADDR(7 downto 2)    <= COL_BIT;
```

```
                if (SUB_ROW <= 6) then
                    ADDR(1 downto 0)    <= "00";
                elsif (SUB_ROW <= 12) then
                    ADDR(1 downto 0)    <= "01";
                else
                    ADDR(1 downto 0)    <= "10";
                end if;
            elsif (SUB_COL_BIT(3 downto 1) = "111") then     -- read font byte 1 at
SUB_ROW 14-15
                ADDR(15 downto 14)  <= "01";
                ADDR(13 downto 6)   <= CODE;
                ADDR(5 downto 2)    <= int2evec(SUB_ROW,4);
                ADDR(1)             <= FIELD;
                ADDR(0)             <= '0';
            elsif (SUB_COL_BIT(3 downto 1) = "011") then     -- read font byte 2 at
SUB_ROW 6-7
                ADDR(15 downto 14)  <= "01";
                ADDR(13 downto 6)   <= CODE;
                ADDR(5 downto 2)    <= int2evec(SUB_ROW,4);
                ADDR(1)             <= FIELD;
                ADDR(0)             <= '1';
            end if;

        end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- Arithmatic Logic Shift Unit : ALRU
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity ALRU is
port(   X   : in std_logic_vector(7 downto 0);      -- Left INPUT
        Y   : in std_logic_vector(7 downto 0);      -- Right INPUT
        SEL : in std_logic_vector(3 downto 0);      -- Select Operation Command
        Cin : in std_logic;                         -- Carry Flag input
        R   : out std_logic_vector(7 downto 0); -- Result OUTPUT
        FLAG: out std_logic_vector(1 downto 0));    -- Flag Register

        -- FLAG is (C,Z)
        -- Flag is C    Carry flag
        --         Z    Zero flag
end ALRU;

architecture RTL of ALRU is

signal S    : std_logic_vector(7 downto 0); -- ADSU8 Arithmatic Output
signal CO   : std_logic;                    -- ADSU8 output Carry flag
signal R_TMP: std_logic_vector(7 downto 0); -- ALRU  output temporary Result : ALRU
temporary output
signal C    : std_logic;                    -- Carry flag temp
signal C_ADSU8 : std_logic;

component ADD_SUB8                          -- add & subtrac unit with grouping signal
    port (A    : in std_logic_vector(7 downto 0);
          B    : in std_logic_vector(7 downto 0);
          Cin  : in std_logic;
          ADD  : in std_logic;
          S    : out std_logic_vector(7 downto 0);
          Cout : out std_logic);
end component;

begin
    u1 : ADD_SUB8 port map (A => X, B => Y, Cin => C_ADSU8, ADD => SEL(0), S => S,
Cout => CO);

    process(X,Y,SEL,S,CO,Cin)
```

```
begin
    case SEL(3 downto 2) is
        when "00" =>                -- Arithmetic unit
                                    -- 0000     A - B               SUB
                                    -- 0001     A + B               ADD
                                    -- 0010     A - B - Cin         SUC
                                    -- 0011     A + B + Cin         ADC
            case SEL(1 downto 0) is
                when "00" =>
                    C_ADSU8 <= '0';
                    C <= not(CO);
                when "01" =>
                    C_ADSU8 <= '0';
                    C <= CO;
                when "10" =>
                    C_ADSU8 <= Cin;
                    C <= CO;
                when "11" =>
                    C_ADSU8 <= Cin;
                    C <= CO;
                when others =>      -- Never go to this case
                    C_ADSU8 <= '0';
                    C <= '0';
            end case;
            R_TMP <= S;

        when "01" =>                -- Logic unit
                                    -- 0100     AND
                                    -- 0101     OR
                                    -- 0110     XOR
                                    -- 0111     MOVE    R <= Y

            case SEL(1 downto 0) is
                when "00" =>
                    R_TMP <= X and Y;
                when "01" =>
                    R_TMP <= X or Y;
                when "10" =>
                    R_TMP <= X xor Y;
                when "11" =>
                    R_TMP <= Y;
                when others =>  -- Never go to this case
                    R_TMP <= X;
            end case;

            C <= Cin;
            C_ADSU8 <= '0';

        when "10" =>                -- (Arithmatic) Shift unit
                                    -- 1000     ROL : Rotate left
                                    -- 1000     RLC : Rotate left with carry
                                    -- 1010     ROR : Rotate right
                                    -- 1011     RRC : Rotate right with carry

            case SEL(1 downto 0) is
                when "00" =>            -- ROL
                    R_TMP(7 downto 1) <= X(6 downto 0);
                    R_TMP(0)    <= X(7);
                    C <= Cin;
                when "01" =>            -- RLC
                    R_TMP(7 downto 1) <= X(6 downto 0);
                    R_TMP(0)    <= Cin;
                    C <= X(7);
                when "10" =>            -- ROR
                    R_TMP(7) <= X(0);
                    R_TMP(6 downto 0) <= X(7 downto 1);
                    C <= Cin;
                when "11" =>            -- RRC
                    R_TMP(7) <= Cin;
                    R_TMP(6 downto 0) <= X(7 downto 1);
                    C <= X(0);
                when others =>      -- Never go to this case
                    R_TMP <= "00000000";
```

```
                    C <= '0';
                end case;

                C_ADSU8 <= '0';

            when others =>          -- 11xx not use
                R_TMP <= "00000000";
                C_ADSU8 <= '0';
                C <= '0';
        end case;
    end process;

    process(R_TMP)              -- Check Zero flag
    begin
        if (R_TMP = "00000000") then
            FLAG(0) <= '1';
        else
            FLAG(0) <= '0';
        end if;
    end process;

    R <= R_TMP;
    FLAG(1) <= C;               -- Carry flag

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- CAPTION PREPROCESSOR
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity CAP_PRE is
port(CLK        : in std_logic;
     CAP_DATA   : in std_logic;
     FIELD      : in std_logic;
     H_SYNC     : in std_logic;
     V_SYNC     : in std_logic;

     CLK_DIV2   : in std_logic;
     nRESET     : in std_logic;
     STORE_DIR  : in std_logic;
     IO_ADDR       : in std_logic_vector(3 downto 0);
     DATA_IN    : in std_logic_vector(7 downto 0);
     DATA_OUT   : out std_logic_vector(7 downto 0);
     Z_CT       : out std_logic);
end CAP_PRE;

architecture NETLIST of CAP_PRE is

    component RST_DIV
    port(COUNT255   : in integer range 0 to 1023;
         ENABLE     : in std_logic;
         CAP_DATA   : in std_logic;
         CLK_EN     : out std_logic;
         RST_DIV24  : out std_logic);
    end component;

    component DIV24
    port (CLK       : in std_logic;
          RESET     : in std_logic;
          CLK_EN    : in std_logic;
          CLK_OUT   : out std_logic);
    end component;

    component SHIFT19B
    port (INPUT    : in std_logic;                      -- SERIAL INPUT
          SHIFT    : in std_logic;                      -- SHIFT when go to HIGH
          RESET    : in std_logic;                      -- RESET when data is read
          START_BIT : out std_logic_vector(2 downto 0);
```

```
          CHAR1    : out std_logic_vector(7 downto 0);   -- CHARACTOR No.1
          CHAR2    : out std_logic_vector(7 downto 0)); -- CHARACTOR No.2
    end component;

    component COUNT255
        port(nRESET : in std_logic; -- RESET
             CLK    : in std_logic; -- Clock
             DATA255: out integer range 0 to 1023);
    end component;

    component COUNT9B
        port(nRESET : in std_logic; -- RESET
             CLK    : in std_logic; -- Clock
             DATA9B: out integer range 0 to 511);
    end component;

    component CAP_REG
        port (CLK       : in std_logic;
              nRESET    : in std_logic;
              STORE_DIR : in std_logic;
              IO_ADDR   : in std_logic_vector(3 downto 0);
              DATA_IN   : in std_logic_vector(7 downto 0);
              DATA_OUT  : out std_logic_vector(7 downto 0);
              Z_CT      : out std_logic;
              RESET_CAP : out std_logic;
              NEW_DAT_FLAG  : in std_logic;
              CHAR1         : in std_logic_vector(7 downto 0);
              CHAR2         : in std_logic_vector(7 downto 0));
    end component;

signal ENABLE   : std_logic;
signal CLK_EN   : std_logic;
signal RST_DIV24: std_logic;
signal CAP_CLK  : std_logic;
signal START_BIT: std_logic_vector(2 downto 0);

signal LINE     : integer range 0 to 511;
signal H_COUNT  : integer range 0 to 255;

signal RESET_CAP    : std_logic;
signal NEW_DAT_FLAG : std_logic;
signal CHAR1        : std_logic_vector(7 downto 0);
signal CHAR2        : std_logic_vector(7 downto 0);


begin
    u1 : RST_DIV port map (H_COUNT, ENABLE, CAP_DATA, CLK_EN, RST_DIV24);
    u2 : DIV24 port map (CLK, RST_DIV24, CLK_EN, CAP_CLK);
    u3 : SHIFT19B port map (CAP_DATA, CAP_CLK, RESET_CAP, START_BIT, CHAR1, CHAR2);
    u4 : COUNT255 port map (H_SYNC, CLK, H_COUNT);
    u5 : COUNT9B port map (V_SYNC, H_SYNC, LINE);
    u6 : CAP_REG port map (CLK_DIV2, nRESET, STORE_DIR, IO_ADDR, DATA_IN, DATA_OUT,
Z_CT, RESET_CAP, NEW_DAT_FLAG, CHAR1, CHAR2);

    process(LINE,FIELD,START_BIT)
    begin
        if (LINE = 3) and (FIELD = '0') then            -- line 15+3 = 18 odd field
            ENABLE          <= '1';
            NEW_DAT_FLAG    <= '0';
        elsif (START_BIT = "100") then
            ENABLE          <= '0';
            NEW_DAT_FLAG    <= '1';
        else
            ENABLE          <= '0';
            NEW_DAT_FLAG    <= '0';
        end if;
    end process;

end NETLIST;
-- ##############################################################################
------------------------------------------------------------------------
-- caption register
------------------------------------------------------------------------
```

```
library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity CAP_REG is
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR   : in std_logic_vector(3 downto 0);
          DATA_IN   : in std_logic_vector(7 downto 0);
          DATA_OUT  : out std_logic_vector(7 downto 0);
          Z_CT      : out std_logic;
          RESET_CAP : out std_logic;
          NEW_DAT_FLAG  : in std_logic;
          CHAR1         : in std_logic_vector(7 downto 0);
          CHAR2         : in std_logic_vector(7 downto 0));
end CAP_REG;

architecture RTL of CAP_REG is
begin

    process(CLK,nRESET,IO_ADDR,STORE_DIR)
    begin
        if (nRESET = '0') then
            DATA_OUT    <= "00000000";
            Z_CT        <= '1';
            RESET_CAP   <= '0';
        elsif (CLK = '0') and (CLK'event) then
            if (STORE_DIR = '1') then                   -- STORE DATA
                if (IO_ADDR = "0000") then
                    RESET_CAP   <= DATA_IN(7);
                end if;

                Z_CT        <= '1';
            else                                        -- LOAD DATA
                if (IO_ADDR = "0001") then
                    DATA_OUT    <= CHAR1;
                    Z_CT        <= '0';
                elsif (IO_ADDR = "0010") then
                    DATA_OUT    <= CHAR2;
                    Z_CT        <= '0';
                elsif (IO_ADDR = "0100") then
                    DATA_OUT    <= "0000000"&NEW_DAT_FLAG;
                    Z_CT        <= '0';
                else
                    Z_CT        <= '1';
                end if;
            end if;

        end if;

    end process;
end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- row comparator x 4
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity CMP_R_PK is
port(   ROW_REG0    : in integer range 0 to 15;
        ROW_REG1    : in integer range 0 to 15;
        ROW_REG2    : in integer range 0 to 15;
        ROW_REG3    : in integer range 0 to 15;
        ROW         : in integer range 0 to 15;
        DISP_ROW    : out integer range 0 to 3;
        HALT        : out std_logic);
end CMP_R_PK;
```

```
architecture RTL of CMP_R_PK is

signal CORRECT_GR   : std_logic_vector(3 downto 0);


component CMP_ROW
port(   ROW_REG     : in integer range 0 to 15;
        ROW         : in integer range 0 to 15;
        CORRECT     : out std_logic);
end component;

component GN_DSP_R
port(CORRECT_GR : in std_logic_vector(3 downto 0);
     DISP_ROW   : out integer range 0 to 3;
     HALT       : out std_logic);
end component;


begin

    u1 : CMP_ROW port map (ROW_REG0, ROW, CORRECT_GR(0));
    u2 : CMP_ROW port map (ROW_REG1, ROW, CORRECT_GR(1));
    u3 : CMP_ROW port map (ROW_REG2, ROW, CORRECT_GR(2));
    u4 : CMP_ROW port map (ROW_REG3, ROW, CORRECT_GR(3));
    u5 : GN_DSP_R port map (CORRECT_GR, DISP_ROW, HALT);


end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- row comparator
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity CMP_ROW is
port(   ROW_REG     : in integer range 0 to 15;
        ROW         : in integer range 0 to 15;
        CORRECT     : out std_logic);
end CMP_ROW;


architecture RTL of CMP_ROW is
begin
    process(ROW_REG,ROW)
    begin

        if (ROW_REG /= ROW) or (ROW = 2) then       -- disable when ROW = 2
            CORRECT <= '0';
        else                                        -- compare ROW
            CORRECT <= '1';
        end if;


    end process;
end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 7-bit comparator in PWM
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity CMP7B is
port(CLK        : in std_logic;
     COUNT7     : in std_logic_vector(6 downto 0);      -- counter 7-bit
     DATA_IN    : in std_logic_vector(6 downto 0);      -- input value
     nPWM_OUT   : out std_logic);
end CMP7B;

architecture RTL of CMP7B is
```

```
begin

    process(COUNT7,DATA_IN)
    begin               -- if complete cycle, input value not equ 0 RESET it
        if (COUNT7 = "0000000") and (DATA_IN /= "0000000") then
            nPWM_OUT <= '0';
        else
            if (COUNT7 < DATA_IN) then
                nPWM_OUT <= '0';                -- < threshold
            else
                nPWM_OUT <= '1';                -- > threshold
            end if;
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 7-bit comparator in REMOTE CONTROL PREPROCESSOR
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity CMPS7B is
port(DAT7B_IN    : in std_logic_vector(1 downto 0);  -- 2 msb of dat7b
     LOGIC_OUT   : out std_logic);
end CMPS7B;

architecture RTL of CMPS7B is
begin

    process(DAT7B_IN)
    begin
        if (DAT7B_IN = "01") then       -- range 20h to 3Fh  (logic low)
            LOGIC_OUT   <= '0';
        elsif (DAT7B_IN = "11") then    -- range 60h to 7Fh  (logic high)
            LOGIC_OUT   <= '1';
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 10-bit counter
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity COUNT10B is
    port(nRESET : in std_logic; -- RESET
         CLK    : in std_logic; -- Clock
         DATA10B: out integer range 0 to 1023);
end COUNT10B;

architecture RTL of COUNT10B is
signal COUNT : integer range 0 to 1023;

begin
    process(CLK,nRESET)
    begin
        if (nRESET = '0') then
            COUNT  <= 0;
        elsif (CLK = '1') and (CLK'event) then
            if (COUNT = 1023) then
                COUNT <= 0;
            else
                COUNT <= COUNT + 1;
```

```
            end if;
        end if;
    end process;

    DATA10B <= COUNT;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 14-bit counter
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity COUNT14B is
port(CLK*       : in std_logic;
     nRESET     : in std_logic;
     COUNT7HIGH : out std_logic_vector(6 downto 0);
     COUNT7LOW  : out std_logic_vector(6 downto 0));
end COUNT14B;

architecture RTL of COUNT14B is
signal COUNT14BIT : std_logic_vector(13 downto 0);


-- FUNCTION INCREMENT 14-BIT COUNTER ***********************************

    function increment(val : std_logic_vector) return std_logic_vector
    is
        -- normalize the indexing
        alias input : std_logic_vector(val'length downto 1) is val;
        variable result : std_logic_vector(input'range) := input;
        variable carry : std_logic := '1';
    begin
        for i in input'low to input'high loop
            result(i) := input(i) xor carry;
            carry := input(i) and carry;
            exit when carry = '0';
        end loop;
        return result;
    end increment;
-- END FUNCTION ********************************************************

begin
    process(CLK,nRESET)
    begin
        if (nRESET = '0') then              -- RESET counter
            COUNT14BIT <= "00000000000000";
        elsif (CLK = '1') and (CLK'event) then
            COUNT14BIT <= increment(COUNT14BIT);    -- count up when go high
        end if;
    end process;

    COUNT7HIGH  <= COUNT14BIT(13 downto 7);         -- separate signal
    COUNT7LOW   <= COUNT14BIT(6 downto 0);

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 1-bit adder/subtractor
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity COUNT1B is
port(CLK        : in std_logic;
     nRESET     : in std_logic;
     OUT1BIT    : out std_logic);
end COUNT1B;
```

```
architecture RTL of COUNT1B is
signal COUNT : integer range 0 to 1;

begin
    process(CLK,nRESET)
    begin
        if (nRESET = '0') then
            COUNT  <= 0;
        elsif (CLK = '1') and (CLK'event) then
            if (COUNT = 1) then
                COUNT <= 0;
            else
                COUNT <= COUNT + 1;
            end if;
        end if;
    end process;

    process(COUNT)       -- transform integer to bit
    begin
        if (COUNT = 1) then
            OUT1BIT <= '1';
        else
            OUT1BIT <= '0';
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 8-bit counter (0-255)
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity COUNT255 is
    port(nRESET : in std_logic; -- RESET
         CLK    : in std_logic; -- Clock
         DATA255: out integer range 0 to 255);
end COUNT255;

architecture RTL of COUNT255 is
signal COUNT : integer range 0 to 255;

begin
    process(CLK,nRESET)
    begin
        if (nRESET = '0') then
            COUNT  <= 0;
        elsif (CLK = '1') and (CLK'event) then
            if (COUNT = 255) then
                COUNT <= 255;
            else
                COUNT <= COUNT + 1;
            end if;
        end if;
    end process;

    DATA255 <= COUNT;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 7-bit counter (no return to 0)
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;
```

```
entity COUNT7BN is
port(CLK        : in std_logic;
     nRESET     : in std_logic;
     COUNT      : out std_logic_vector(6 downto 0));
end COUNT7BN;

architecture RTL of COUNT7BN is
signal COUNT_TMP : integer range 0 to 127;
signal COUNT_SIG : integer range 0 to 127;

        -- Integer to signed or unsigned vector
       function int2evec (l: integer; size: integer := 32) return std_logic_vector is
          variable result: std_logic_vector(size-1 downto 0);
          variable op: integer := l;
          variable neg_fl: boolean := false;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          result := (size-1 downto 0 => '0');
          if op < 0 then
             op := op * (-1);
             neg_fl := true;
          end if;
          for i in 0 to SIZE-1 loop
             if (op mod 2) = 1 then
                result (i) := '1' ;
             end if ;
             op := op/2 ;
          end loop ;
          return result;
       end int2evec;

begin

    process(CLK,nRESET)
    begin
        if (nRESET = '0') then
            COUNT_TMP <= 0;
        elsif (CLK = '1') and (CLK'event) then
            if (COUNT_TMP = 127) then
                COUNT_TMP <= 127;
            else
                COUNT_TMP <= COUNT_TMP + 1;
            end if;

        end if;
    end process;

    process(nRESET,COUNT_TMP)
    begin
        if (nRESET = '1') then
            COUNT_SIG <= COUNT_TMP;
        else
            COUNT_SIG <= COUNT_SIG;
        end if;
    end process;

    COUNT <= int2evec(COUNT_SIG,7);

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 9-bit adder/subtractor
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity COUNT9B is
    port(nRESET : in std_logic; -- RESET
         CLK    : in std_logic; -- Clock
```

```
        DATA9B: out integer range 0 to 511);
end COUNT9B;

architecture RTL of COUNT9B is
signal COUNT : integer range 0 to 511;

begin
    process(CLK,nRESET)
    begin
        if (nRESET = '0') then
            COUNT  <= 0;
        elsif (CLK = '1') and (CLK'event) then
            if (COUNT = 511) then
                COUNT <= 0;
            else
                COUNT <= COUNT + 1;
            end if;
        end if;
    end process;

    DATA9B  <= COUNT;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- cpu
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity CPU is
port(   CLK         : in std_logic;
        nRESET      : in std_logic;
        HALT        : in std_logic;
        nCE_ROM     : out std_logic;            -- ROM Chip Enable
        nCS_RAM     : out std_logic;            -- RAM Chip Select (+Output Enable)
        nRD         : out std_logic;            -- READ
        nWR         : out std_logic;            -- WRITE
        DATA_IN     : in std_logic_vector(7 downto 0);
        DATA_OUT    : out std_logic_vector(7 downto 0);
        Z_CT        : out std_logic;
        STORE_DIR       : out std_logic;                    -- load or store direction
        IO_ADDR           : out std_logic_vector(3 downto 0); -- internal address for
memory map i/o
        ADDR        : out std_logic_vector(14 downto 0);
        OSD_STATE   : out std_logic;
        OE_ADC      : out std_logic);
end CPU;

architecture STATE_MACHINE of CPU is

    type T_STATE is (OSD, PLACE_AR1, LOD_IR_L, INC_PC1, ARITH, LOD_IR_H, PLACE_AR2,
INC_PC2,
                     JUMP2, LOD_STO, STORE, LOAD, JUMP1);
    signal N_STATE : T_STATE;

component ALRU
port(   X   : in std_logic_vector(7 downto 0);      -- Left INPUT
        Y   : in std_logic_vector(7 downto 0);      -- Right INPUT
        SEL : in std_logic_vector(3 downto 0);      -- Select Operation Command
        Cin : in std_logic;                         -- Carry Flag input
        R   : out std_logic_vector(7 downto 0);     -- Result OUTPUT
        FLAG: out std_logic_vector(1 downto 0));    -- Flag Register

        -- FLAG is (C,Z)
        -- Flag is C    Carry flag
        --         Z    Zero flag
end component;

-- ALRU signal
signal TMP1         : std_logic_vector(7 downto 0); -- TMP1 Register
signal TMP2         : std_logic_vector(7 downto 0); -- TMP2 Register
```

```
signal SEL          : std_logic_vector(3 downto 0); -- Select Signal
signal C_FLAG       : std_logic;                    -- Carry in input to ALRU
signal R            : std_logic_vector(7 downto 0); -- Result Register

-- Register
signal AR           : std_logic_vector(15 downto 0);    -- Address Register
signal PC           : std_logic_vector(15 downto 0);    -- Program counter
signal IR           : std_logic_vector(15 downto 0);    -- Instruction Register
signal A            : std_logic_vector(7 downto 0); -- Register A
signal B            : std_logic_vector(7 downto 0); -- Register B
signal DPH             : std_logic_vector(7 downto 0); -- High Byte of Data PoinTer
Register (DPTR)
signal DPL             : std_logic_vector(7 downto 0); -- Low Byte of Data PoinTer
Register (DPTR)

signal C            : std_logic;        -- Carry Flag
signal Z            : std_logic;        -- Zero Flag

signal FLAG         : std_logic_vector(1 downto 0); -- temporary flag output signal
signal nRD_SIG      : std_logic;                    -- temporary READ signal

begin

    u1 : ALRU port map (TMP1, TMP2, SEL, C_FLAG, R, FLAG);
    ADDR(14 downto 0)   <= AR(14 downto 0);
    nCS_RAM <= not(AR(15));                 -- Select RAM if AR(15) high for data range
8000h-FFFFh
    nCE_ROM <= AR(15) or nRD_SIG;       -- READ operation and Select ROM
    nRD     <= nRD_SIG;                 --
    STORE_DIR   <= IR(12);

    process(nRESET, CLK)
    begin
        if (nRESET = '0') then
            N_STATE     <= OSD;     -- RESET all registers
            PC          <= "0000000000000000";
            IR          <= "0000000000000000";
            A           <= "00000000";
            B           <= "00000000";
            DPH         <= "00000000";
            DPL         <= "00000000";
            IO_ADDR <= "1111";
            C           <= '0';
            Z           <= '0';
            SEL         <= "0001";
            C_FLAG      <= '0';
            TMP1        <= "00000000";
            TMP2        <= "00000000";
            AR          <= "0000000000000000";
            nWR         <= '1';
            nRD_SIG     <= '1';
            DATA_OUT    <= "00000000";
            Z_CT        <= '1';
            OSD_STATE   <= '0';
            OE_ADC      <= '0';

        elsif (CLK'event) and (CLK = '1') then
            case N_STATE is
                when OSD =>
                    IO_ADDR <= "1111";

                    if (HALT = '0') then
                        N_STATE <= PLACE_AR1;
                    else
                        N_STATE <= OSD;
                    end if;

                    Z_CT        <= '1';
                    OSD_STATE   <= '1';
                    OE_ADC      <= '0';

                when PLACE_AR1 =>
                    AR      <= PC;                  -- Place Address Register
```

```
                nWR     <= '1';                 -- prepare for read ROM
                nRD_SIG <= '0';
                TMP1    <= PC(7 downto 0);
                TMP2    <= "00000001";
                SEL     <= "0001";
                C_FLAG  <= '0';
                Z_CT    <= '1';
                N_STATE <= INC_PC1;
                OSD_STATE   <= '0';

            when INC_PC1 =>
                PC(7 downto 0) <= R;
                TMP1    <= PC(15 downto 8);
                TMP2    <= "0000000"&FLAG(1);
                Z_CT        <= '1';
                C_FLAG  <= '0';
                N_STATE <= LOD_IR_L;

            when LOD_IR_L =>
                PC(15 downto 8) <= R;                       -- Save PC that
incremented
                IR(7 downto 0)  <= DATA_IN;
                Z_CT        <= '1';
                N_STATE <= PLACE_AR2;

            when PLACE_AR2 =>
                AR      <= PC;                  -- Place high byte address of IR
                Z_CT        <= '1';
                TMP1    <= PC(7 downto 0);
                TMP2    <= "00000001";
                C_FLAG  <= '0';
                N_STATE <= INC_PC2;

            when INC_PC2    =>
                PC(7 downto 0) <= R;
                Z_CT        <= '1';
                TMP1    <= PC(15 downto 8);
                TMP2    <= "0000000"&FLAG(1);
                C_FLAG  <= '0';
                N_STATE <= LOD_IR_H;

            when LOD_IR_H =>
                IR(15 downto 8) <= DATA_IN;
                Z_CT        <= '1';

                case DATA_IN(7 downto 6) is
                    when "00" | "01" =>             -- Arithmatic Reg & Constant

                        PC(15 downto 8) <= R;

                        if (DATA_IN(5 downto 4) = "11") then

                            SEL     <= "0111";              -- move operation R <=
TMP2

                            if (DATA_IN(2) = '0') then      -- PCL
                                TMP2    <= PC(7 downto 0);
                            else                            -- PCH
                                TMP2    <= R;
                            end if;
                        else
                            C_FLAG <= C;

                            SEL     <= DATA_IN(5 downto 2);     -- ALRU operation
                                                            --  LOAD  TMP1  &  TMP2
register
                            if (DATA_IN(6) = '0') then              -- Reg & Const
(Source is 9 downto 8)
                                TMP2 <= IR(7 downto 0);
                            else                            -- Reg & Reg   (Source
is 5 downto 4)
                                case IR(5 downto 4) is          -- SOURCE
                                    when "00" =>
```

```
                                        TMP2 <= A;
                                    when "01" =>
                                        TMP2 <= B;
                                    when "10" =>
                                        TMP2 <= DPL;
                                    when "11" =>
                                        TMP2 <= DPH;
                                    when others =>                  -- NEVER GO TO
THIS CASE
                                        TMP2 <= "00000000";
                                end case;
                            end if;

                            case DATA_IN(1 downto 0) is             -- DESTINATION
                                when "00" =>
                                    TMP1 <= A;
                                when "01" =>
                                    TMP1 <= B;
                                when "10" =>
                                    TMP1 <= DPL;
                                when "11" =>
                                    TMP1 <= DPH;
                                when others =>                  -- NEVER GO TO THIS
CASE
                                    TMP1 <= "00000000";
                            end case;

                        end if;

                        N_STATE <= ARITH;
                    when "10" =>                -- Load/Store
                        PC(15 downto 8) <= R;
                        N_STATE <= LOD_STO;

                    when "11" =>                -- Jump
                        if (DATA_IN(5) = '1') then      -- Abs Jump     111
                            PC  <= DPH&DPL;
                            N_STATE <= OSD;
                        else                            -- Rel Jump         110
                            PC(15 downto 8) <= R;
                                                        --  Conditional  Jump
0xx
                            if ((DATA_IN(4) = '0') and ((DATA_IN(3 downto 2) =
"00" and C = '0') or (DATA_IN(3 downto 2) = "01" and C = '1')
                                                    or   (DATA_IN(3 downto 2) =
"10" and Z = '0') or (DATA_IN(3 downto 2) = "11" and Z = '1')))
                                or  (DATA_IN(4) = '1') then -- Unconditional Jump
                                    TMP1    <= PC(7 downto 0);
                                    TMP2    <= IR(7 downto 0);
                                    SEL     <= "000"&DATA_IN(0);
                                    C_FLAG  <= '0';
                                    N_STATE <= JUMP1;

                            else
                                N_STATE <= OSD;

                            end if;
                        end if;
                    when others =>
                        N_STATE <= OSD;

                end case;

            when ARITH =>
                SEL     <= IR(13 downto 10);        -- ALRU operation
                Z_CT        <= '1';

                case IR(9 downto 8) is
                    when "00" =>
                        A <= R;
                    when "01" =>
                        B <= R;
                    when "10" =>
```

```
            DPL <= R;
        when "11" =>
            DPH <= R;
        when others =>                  -- NEVER GO TO THIS CASE
            A <= "00000000";
    end case;

    C <= FLAG(1);
    Z <= FLAG(0);
    N_STATE <= OSD;

when JUMP1 =>
    PC(7 downto 0)  <= R;
    Z_CT        <= '1';
    TMP1    <= PC(15 downto 8);
    TMP2    <= "0000000"&FLAG(1);
    C_FLAG  <= '0';
    N_STATE         <= JUMP2;

when JUMP2 =>
    PC(15 downto 8) <= R;
    Z_CT        <= '1';
    N_STATE         <= OSD;

when LOD_STO =>                         -- Place Address to Load or Store
    if (IR(11) = '0') then                      -- indirect mem or map i/o
        AR  <= DPH&DPL;
    else                                        -- direct mem
        AR  <= "10000000"&IR(7 downto 0);
    end if;
    Z_CT        <= '1';

    case IR(12 downto 10) is
        when "000" =>                   -- load a OR load b
            nWR     <= '1';
            nRD_SIG <= '0';
            OE_ADC  <= '0';
            N_STATE <= LOAD;
        when "001" =>                   -- load MAP I/O
            nWR     <= '1';
            nRD_SIG <= '1';
            OE_ADC  <= '0';

            IO_ADDR <= IR(3 downto 0);

            N_STATE <= LOAD;
        when "010" =>                   -- load from addr 80xxh
            nWR     <= '1';
            nRD_SIG <= '0';
            OE_ADC  <= '0';
            N_STATE <= LOAD;
        when "011" =>                   -- load from A/D
            nWR     <= '1';
            nRD_SIG <= '1';
            OE_ADC  <= '1';
            N_STATE <= LOAD;
        when "100" =>                   -- store a OR store b
            nWR     <= '0';
            nRD_SIG <= '1';
            OE_ADC  <= '0';
            N_STATE <= STORE;
        when "101" =>                   -- store MAP I/O
            nWR     <= '1';
            nRD_SIG <= '1';
            OE_ADC  <= '0';
            N_STATE <= STORE;
        when "110" =>                   -- store to addr 80xxh
            nWR     <= '0';
            nRD_SIG <= '1';
            OE_ADC  <= '0';
            N_STATE <= STORE;
        when others =>
            nWR     <= '1';
```

```
                        nRD_SIG <= '1';
                        OE_ADC  <= '0';
                        N_STATE <= STORE;
                    end case;

                when STORE =>
                    Z_CT        <= '0';

                    if (IR(10) = '1') then
                        IO_ADDR <= IR(3 downto 0);
                    end if;

                    if (IR(8) = '0') then
                        DATA_OUT    <= A;
                    else
                        DATA_OUT    <= B;
                    end if;

                    N_STATE <= OSD;

                when LOAD =>
                    Z_CT        <= '1';
                    IO_ADDR <= "1111";

                    if (IR(8) = '0') then
                        A   <= DATA_IN;
                    else
                        B   <= DATA_IN;
                    end if;

                    N_STATE <= OSD;

                when others =>
                    N_STATE <= OSD;

            end case;

        end if;

    end process;

end STATE_MACHINE;
-- ##############################################################################
------------------------------------------------------------------------
-- data buffer
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity DATA_BUF is
port(   CLK         : in std_logic;
        SUB_COL     : in integer range 0 to 15;
        DATA_IN     : in std_logic_vector(7 downto 0);
        ATTR_DATA       :   out   std_logic_vector(4   downto   0);   --   F,I,R,G,B
(Flash,Italic,Red,Green,Blue)
        CODE_DATA   : out std_logic_vector(7 downto 0); -- Font Code or Attribute Code
        FONT_DATA     : out std_logic_vector(7 downto 0));      -- Font Data (Half
Character)
end DATA_BUF;

architecture RTL of DATA_BUF is

signal ATTR_TMP : std_logic_vector(4 downto 0);
signal FONT_TMP : std_logic_vector(7 downto 0);
signal SUB_COL_BIT   : std_logic_vector(3 downto 0);
signal HALF_SUB_COL  : integer range 0 to 7;
signal COME : std_logic;

        -- Integer to signed or unsigned vector
        function int2evec (i: integer; size: integer := 32) return std_logic_vector is
```

```
        variable result: std_logic_vector(size-1 downto 0);
        variable op: integer := 1;
        variable neg_fl: boolean := false;
         attribute synthesis_return of result:variable is "FEED_THROUGH" ;
      begin
        result := (size-1 downto 0 => '0');
        if op < 0 then
          op := op * (-1);
          neg_fl := true;
        end if;
        for i in 0 to SIZE-1 loop
          if (op mod 2) = 1 then
            result (i) := '1' ;
          end if ;
          op := op/2 ;
        end loop ;
        return result;
      end int2evec;

    -- function evec2int (l: std_logic_vector)                    return natural;
    -- Translate unsigned vector into integer representation
    ---
      function evec2int (l: std_logic_vector)
          return natural is
        variable result: natural := 0;
         attribute synthesis_return of result:variable is "FEED_THROUGH" ;
      begin
        for tl in l'range loop
          result := result * 2;
          if (l(tl) = '1' or l(tl) = 'H') then
            result := result + 1;
          end if;
        end loop;
        return result;
      end evec2int;

begin

    SUB_COL_BIT <= int2evec(SUB_COL,4);
    HALF_SUB_COL    <= evec2int(SUB_COL_BIT(2 downto 0));

    process(CLK,SUB_COL,DATA_IN)
    begin
        if (CLK = '0') and (CLK'event) then

            if (SUB_COL = 13) then
                if (DATA_IN(7 downto 5) = "000") then       -- control code 00 - 1F
                    ATTR_TMP <= DATA_IN(4 downto 0);
                    COME<= '0';
                else
                    COME<= '1';
                end if;
                                                            -- char data code
                CODE_DATA <= DATA_IN;
            elsif (SUB_COL = 7) or (SUB_COL = 15) then      -- font byte 2 and byte 1
                FONT_TMP <= DATA_IN and COME&COME&COME&COME&COME&COME&COME&COME;
            end if;

        end if;
    end process;

    process(HALF_SUB_COL, SUB_COL_BIT, FONT_TMP, ATTR_TMP)
    begin
        if (HALF_SUB_COL <= 6) then
            FONT_DATA   <= FONT_TMP;
        end if;

        if (SUB_COL_BIT = "0101") then      -- 5
            ATTR_DATA   <= ATTR_TMP;
        end if;
    end process;

end RTL;
```

```
-- ##############################################################################
--------------------------------------------------------------------------------
-- clock divider
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity DIV_CLK is
port(CLK        : in std_logic;
     nRESET     : in std_logic;
     DIV2       : out std_logic;
     DIV4       : out std_logic;
     DIV8       : out std_logic;
     DIV16      : out std_logic;
     DIV24      : out std_logic;
     DIV32      : out std_logic;
     DIV64      : out std_logic;
     DIV128     : out std_logic;
     DIV256     : out std_logic);
end DIV_CLK;

architecture NETLIST of DIV_CLK is

signal DIV2_TMP : std_logic;
signal DIV4_TMP : std_logic;
signal DIV8_TMP : std_logic;
signal DIV16_TMP: std_logic;
signal DIV24_TMP: std_logic;
signal DIV32_TMP: std_logic;
signal DIV64_TMP : std_logic;
signal DIV128_TMP: std_logic;
signal DIV256_TMP: std_logic;

component COUNT1B
port(CLK        : in std_logic;
     nRESET         : in std_logic;
     OUT1BIT    : out std_logic);
end component;

component DIV3
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          CLK_OUT   : out std_logic);
end component;

begin

    u1 : COUNT1B port map (CLK,nRESET,DIV2_TMP);
    u2 : COUNT1B port map (DIV2_TMP,nRESET,DIV4_TMP);
    u3 : COUNT1B port map (DIV4_TMP,nRESET,DIV8_TMP);
    u4 : COUNT1B port map (DIV8_TMP,nRESET,DIV16_TMP);
    u5 : COUNT1B port map (DIV16_TMP,nRESET,DIV32_TMP);
    u6 : COUNT1B port map (DIV32_TMP,nRESET,DIV64_TMP);
    u7 : COUNT1B port map (DIV64_TMP,nRESET,DIV128_TMP);
    u8 : COUNT1B port map (DIV128_TMP,nRESET,DIV256_TMP);
    u9 : DIV3 port map (DIV8_TMP,nRESET,DIV24_TMP);

    DIV2    <= DIV2_TMP;
    DIV4    <= DIV4_TMP;
    DIV8    <= DIV8_TMP;
    DIV16   <= DIV16_TMP;
    DIV24   <= DIV24_TMP;
    DIV32   <= DIV32_TMP;
    DIV64   <= DIV64_TMP;
    DIV128  <= DIV128_TMP;
    DIV256  <= DIV256_TMP;

end NETLIST;
-- ##############################################################################
--------------------------------------------------------------------------------
```

```
-- divide frequency by 24
--------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;

entity DIV24 is
    port (CLK       : in std_logic;
          RESET     : in std_logic;
          CLK_EN    : in std_logic;
          CLK_OUT   : out std_logic);
end DIV24;

architecture RTL of DIV24 is
    constant MAX_COUNT      : integer := 11;

    signal CLK_OUT_SIGNAL : std_logic;              --TEMPORARY CLK/24 SIGNAL
    signal I              : integer range 0 to MAX_COUNT;       --INDEX COUNTER

    signal COUNT    : integer range 0 to 63;

begin
    process(CLK,RESET)
    begin
        if (RESET = '1') then           -- RESET COUNTER if RESET go to HIGH
                I <= MAX_COUNT;
                CLK_OUT_SIGNAL <= '0';
                COUNT          <= 0;
        elsif (CLK = '1') and (CLK'event) then
            if (I = 0) then                     -- TOGGLE when half period
                I <= MAX_COUNT;

                if (COUNT = 38) then
                    COUNT   <= 38;
                    CLK_OUT_SIGNAL <= '0';
                else
                    COUNT          <= COUNT + 1;
                    CLK_OUT_SIGNAL <= not(CLK_OUT_SIGNAL);
                end if;

            else                                        -- COUNT UP
                I <= I - 1;
            end if;
        end if;
    end process;

    CLK_OUT <= CLK_OUT_SIGNAL and CLK_EN;
end RTL;
-- ##############################################################################
--------------------------------------------------------------------------
-- divide frequency by 3
--------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;

entity DIV3 is
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          CLK_OUT   : out std_logic);
end DIV3;

architecture RTL of DIV3 is
    constant MAX_COUNT      : integer := 2;

    signal I              : integer range 0 to MAX_COUNT;

begin
    process(CLK,nRESET)

    begin
        if (nRESET = '0') then
                I <= MAX_COUNT;
```

```
        elsif (CLK = '1') and (CLK'event) then
            if (I = 0) then
                I <= MAX_COUNT;
            else
                I <= I - 1;
            end if;
        end if;
    end process;

    process(I)
    begin
        if (I = MAX_COUNT) then
            CLK_OUT <= '1';
        else
            CLK_OUT <= '0';
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- frequency generator (from OSC4)
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity GEN_OSC is
port(   nRESET  : in std_logic;
        F8M     : out std_logic;
        F500K   : out std_logic;
        F16K    : out std_logic;
        F490    : out std_logic;
        F245    : out std_logic;
        F123    : out std_logic;
        F41     : out std_logic;
        F15     : out std_logic;
        F7      : out std_logic;
        F4      : out std_logic;
        F2      : out std_logic;
        F1      : out std_logic;
        F05     : out std_logic);
end GEN_OSC;

architecture NETLIST of GEN_OSC is

signal F8M_TMP  : std_logic;
signal F500K_TMP: std_logic;
signal F16K_TMP : std_logic;
signal F490_TMP : std_logic;
signal F15_TMP  : std_logic;
signal F245_TMP : std_logic;
signal F123_TMP : std_logic;
signal F41_TMP  : std_logic;
signal F7_TMP   : std_logic;
signal F4_TMP   : std_logic;
signal F2_TMP   : std_logic;
signal F1_TMP   : std_logic;
signal F05_TMP  : std_logic;

component OSC4
port (F8M   : out std_logic;
      F500K : out std_logic;
      F16K  : out std_logic;
      F490  : out std_logic;
      F15   : out std_logic);
end component;

component COUNT1B
port(CLK        : in std_logic;
     nRESET       : in std_logic;
```

```
     OUT1BIT    : out std_logic);
end component;

component DIV3
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          CLK_OUT   : out std_logic);
end component;

begin
    u1 : OSC4 port map (F8M_TMP, F500K_TMP, F16K_TMP, F490_TMP, F15_TMP);
    u2 : COUNT1B port map (F490_TMP, nRESET, F245_TMP);
    u3 : COUNT1B port map (F245_TMP, nRESET, F123_TMP);
    u4 : DIV3 port map (F123_TMP, nRESET, F41_TMP);

    u5 : COUNT1B port map (F15_TMP, nRESET, F7_TMP);
    u6 : COUNT1B port map (F7_TMP, nRESET, F4_TMP);
    u7 : COUNT1B port map (F4_TMP, nRESET, F2_TMP);
    u8 : COUNT1B port map (F2_TMP, nRESET, F1_TMP);
    u9 : COUNT1B port map (F1_TMP, nRESET, F05_TMP);

    F8M     <= F8M_TMP;
    F500K   <= F500K_TMP;
    F16K    <= F16K_TMP;
    F490    <= F490_TMP;
    F15     <= F15_TMP;
    F245    <= F245_TMP;
    F123    <= F123_TMP;
    F41     <= F41_TMP;

    F7      <= F7_TMP;
    F4      <= F4_TMP;
    F2      <= F2_TMP;
    F1      <= F1_TMP;
    F05     <= F05_TMP;

end NETLIST;
-- ##############################################################################
------------------------------------------------------------------------
-- Display ROW number signal generator
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity GN_DSP_R is
port(CORRECT_GR : in std_logic_vector(3 downto 0);
     DISP_ROW   : out integer range 0 to 3;
     HALT       : out std_logic);
end GN_DSP_R;

architecture RTL of GN_DSP_R is
begin

    process(CORRECT_GR)
    begin
        case CORRECT_GR is
            when "0001" =>          -- Row register no. x is equ
                DISP_ROW <= 0;
                HALT     <= '1';
            when "0010" =>
                DISP_ROW <= 1;
                HALT     <= '1';
            when "0100" =>
                DISP_ROW <= 2;
                HALT     <= '1';
            when "1000" =>
                DISP_ROW <= 3;
                HALT     <= '1';
            when others =>
                DISP_ROW <= 0;
                HALT     <= '0';
        end case;
    end process;
```

```
end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- HALT2 signal Generator for windowing screen
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity GN_HALT2 is
port(   LINE9B      : in std_logic_vector(8 downto 0);
        HALT        : in std_logic;
        HALT2       : out std_logic);
end GN_HALT2;

architecture RTL of GN_HALT2 is

signal LINE9B_INT   : integer range 0 to 511;

    -- function evec2int (l: std_logic_vector)                    return natural;
    -- Translate unsigned vector into integer representation
    ---
       function evec2int (l: std_logic_vector)
             return natural is
          variable result: natural := 0;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          for tl in l'range loop
             result := result * 2;
             if (l(tl) = '1' or l(tl) = 'H') then
                result := result + 1;
             end if;
          end loop;
          return result;
       end evec2int;

begin
    LINE9B_INT  <= evec2int(LINE9B);

    process(LINE9B_INT,HALT)
    begin
        if (LINE9B_INT < 32) or (LINE9B_INT > 271) then
            HALT2   <= '0';
        else
            HALT2   <= HALT;
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- Horizontal counter
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity H_COUNTR is
    port(H_SYNC : in std_logic; -- RESET Horizontal counter
         CLK    : in std_logic; -- Clock for Horizontal counter
         PLUS   : in integer range 0 to 7;
         COL    : out integer range 0 to 63;
         SUB_COL: out integer range 0 to 15);
end H_COUNTR;

architecture RTL of H_COUNTR is

signal DATA10B      : integer range 0 to 1023;
```

```
signal DATA10B_BIT  : std_logic_vector(9 downto 0);
signal GND          : std_logic;
signal PLUS_BIT     : std_logic_vector(9 downto 0);
signal VCC          : std_logic;
signal DATA10B_OUT  : std_logic_vector(9 downto 0);
signal Cout         : std_logic;

    -- function evec2int (l: std_logic_vector)                    return natural;
    -- Translate unsigned vector into integer representation
    ---
       function evec2int (l: std_logic_vector)
             return natural is
          variable result: natural := 0;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          for t1 in l'range loop
             result := result * 2;
             if (l(t1) = '1' or l(t1) = 'H') then
                result := result + 1;
             end if;
          end loop;
          return result;
       end evec2int;

        -- Integer to signed or unsigned vector
       function int2evec (l: integer; size: integer := 32) return std_logic_vector is
          variable result: std_logic_vector(size-1 downto 0);
          variable op: integer := 1;
          variable neg_fl: boolean := false;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          result := (size-1 downto 0 => '0');
          if op < 0 then
             op := op * (-1);
             neg_fl := true;
          end if;
          for i in 0 to SIZE-1 loop
             if (op mod 2) = 1 then
                result (i) := '1' ;
             end if ;
             op := op/2 ;
          end loop ;
          return result;
       end int2evec;

component COUNT10B
    port(nRESET : in std_logic; -- RESET
         CLK    : in std_logic; -- Clock
         DATA10B: out integer range 0 to 1023);
end component;

component AD_SUB10
    port (A    : in std_logic_vector(9 downto 0);
          B    : in std_logic_vector(9 downto 0);
          Cin  : in std_logic;
          ADD  : in std_logic;
          S    : out std_logic_vector(9 downto 0));
end component;

begin
    VCC         <= '1';
    GND         <= '0';

    DATA10B_BIT <= int2evec(DATA10B,10);
    PLUS_BIT    <= int2evec(PLUS,10);

    u1 : COUNT10B port map (H_SYNC, CLK, DATA10B);
    u2 : AD_SUB10 port map (DATA10B_BIT, PLUS_BIT, GND, VCC, DATA10B_OUT);

    COL     <= evec2int(DATA10B_OUT(9 downto 4));
    SUB_COL <= evec2int(DATA10B_OUT(3 downto 0));

end RTL;
```

```
-- ##############################################################################
--------------------------------------------------------------------------------
-- I/O port
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity IO_PORT is
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR   : in std_logic_vector(3 downto 0);
          DATA_IN   : in std_logic_vector(7 downto 0);
          DATA_OUT  : out std_logic_vector(7 downto 0);
          Z_CT      : out std_logic;
          P0_IN     : in std_logic_vector(2 downto 0);
          P1_OUT    : out std_logic_vector(7 downto 0);
          P2_OUT    : out std_logic_vector(3 downto 0));
end IO_PORT;

architecture RTL of IO_PORT is

begin

    process(CLK,nRESET,IO_ADDR,STORE_DIR)
    begin
        if (nRESET = '0') then
            P1_OUT  <= "00000000";
            P2_OUT  <= "0000";
            DATA_OUT <= "00000000";
            Z_CT     <= '1';

        elsif (CLK = '0') and (CLK'event) then
            if (STORE_DIR = '0') then                -- LOAD DATA

                if (IO_ADDR = "0000") then
                    DATA_OUT <= "00000"&P0_IN;
                    Z_CT     <= '0';
                else
                    Z_CT    <= '1';
                end if;
            else                                     -- STORE DATA

                if (IO_ADDR = "0001") then
                    P1_OUT  <= DATA_IN(7 downto 0);
                elsif (IO_ADDR = "0010") then
                    P2_OUT  <= DATA_IN(3 downto 0);
                end if;

                Z_CT <= '1';

            end if;
        end if;
    end process;
end RTL;
-- ##############################################################################
--------------------------------------------------------------------------------
-- IR remote control preprocessor
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity IR_PRE is
port(   CLK     : in std_logic;
        nIR     : in std_logic;
        CLK_DIV2    : in std_logic;
        nRESET      : in std_logic;
```

```
        STORE_DIR   : in std_logic;
        IO_ADDR     : in std_logic_vector(3 downto 0);
        DATA_IN     : in std_logic_vector(7 downto 0);
        DATA_OUT    : out std_logic_vector(7 downto 0);
        Z_CT        : out std_logic);
end IR_PRE;

architecture NETLIST of IR_PRE is

signal COUNT    : std_logic_vector(6 downto 0);
signal COUNT2BH : std_logic_vector(1 downto 0);
signal LOGIC_OUT : std_logic;

component COUNT7BN
port(CLK        : in std_logic;
     nRESET     : in std_logic;
     COUNT      : out std_logic_vector(6 downto 0));
end component;

component CMPS7B
port(DAT7B_IN   : in std_logic_vector(1 downto 0);
     LOGIC_OUT  : out std_logic);
end component;

component SHIFT12B
port(DATA_IN    : in std_logic;
     SHIFT      : in std_logic;
     GATE       : in std_logic;
     IR_CODE    : out std_logic_vector(5 downto 0));
end component;

component IR_REG
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR   : in std_logic_vector(3 downto 0);
          DATA_IN   : in std_logic_vector(7 downto 0);
          DATA_OUT  : out std_logic_vector(7 downto 0);
          Z_CT      : out std_logic;
          IR_CODE   : in std_logic_vector(5 downto 0);
          RM_CT     : out std_logic);
end component;

signal nGATE    : std_logic;
signal GATE     : std_logic;
signal IR_CODE  : std_logic_vector(5 downto 0);

begin
    COUNT2BH    <= COUNT(6 downto 5);
    GATE        <= not(nGATE);

    u1 : COUNT7BN port map (CLK, nIR, COUNT);
    u2 : CMPS7B port map (COUNT2BH, LOGIC_OUT);
    u3 : SHIFT12B port map (LOGIC_OUT, nIR, GATE, IR_CODE);
    u4 : IR_REG port map (CLK_DIV2, nRESET, STORE_DIR, IO_ADDR,
                          DATA_IN, DATA_OUT, Z_CT, IR_CODE, nGATE);

end NETLIST;
-- ##############################################################################
------------------------------------------------------------------------
-- IR remote control register
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity IR_REG is
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR   : in std_logic_vector(3 downto 0);
```

```
          DATA_IN    : in std_logic_vector(7 downto 0);
          DATA_OUT   : out std_logic_vector(7 downto 0);
          Z_CT       : out std_logic;
          IR_CODE    : in std_logic_vector(5 downto 0);
          RM_CT      : out std_logic);

end IR_REG;

architecture RTL of IR_REG is
begin

    process(CLK,nRESET,IO_ADDR,STORE_DIR)
    begin
        if (nRESET = '0') then
            DATA_OUT    <= "00000000";
            Z_CT        <= '1';
            RM_CT       <= '0';

        elsif (CLK = '0') and (CLK'event) then
            if (IO_ADDR = "0011") then
                if (STORE_DIR = '1') then           -- STORE DATA
                    RM_CT   <= DATA_IN(7);
                    Z_CT    <= '1';
                else                                -- LOAD DATA
                    DATA_OUT    <= "00"&IR_CODE;
                    Z_CT    <= '0';
                end if;
            else
                Z_CT    <= '1';
            end if;
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- italic controller
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity ITAL_CT is
port(ITALIC     : in std_logic;
     SUB_ROW    : in integer range 0 to 15;
     PLUS       : out integer range 0 to 7);
end ITAL_CT;

architecture RTL of ITAL_CT is

begin
    process(ITALIC,SUB_ROW)
    begin
        if (ITALIC = '0') then          -- normal font
            PLUS <= 4;
        elsif (SUB_ROW <= 1) then       -- shift right
            PLUS <= 0;
        elsif (SUB_ROW <= 3) then
            PLUS <= 1;
        elsif (SUB_ROW <= 5) then
            PLUS <= 2;
        elsif (SUB_ROW <= 7) then
            PLUS <= 3;
        elsif (SUB_ROW <= 9) then       -- normal
            PLUS <= 4;
        elsif (SUB_ROW <= 11) then      -- shift left
            PLUS <= 5;
        elsif (SUB_ROW <= 13) then
            PLUS <= 6;
        else
            PLUS <= 7;
```

```
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- microcontroller unit
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity MCU is
port(   CLK         : in std_logic;
        nRESET      : in std_logic;

        nCE_ROM     : out std_logic;            -- ROM Chip Enable
        nCS_RAM     : out std_logic;            -- RAM Chip Select (+Output Enable)
        nRD         : out std_logic;            -- READ
        nWR         : out std_logic;            -- WRITE

        DATA_BUS    : inout std_logic_vector(7 downto 0);       -- 8-bit DATA bus

        ADDR        : out std_logic_vector(15 downto 0);        -- 16-bit ADDRESS bus
        OE_ADC      : out std_logic;

        P0          : in std_logic_vector(2 downto 0);
        P1          : out std_logic_vector(7 downto 0);
        P2          : out std_logic_vector(3 downto 0);

        H_SYNC      : in std_logic;
        V_SYNC      : in std_logic;
        nFIELD_IN   : in std_logic;

        R           : out std_logic;
        G           : out std_logic;
        B           : out std_logic;

        nIR         : in std_logic;
        ADC_CLK     : out std_logic;

        PWM0        : out std_logic;
        PWM1        : out std_logic;
        PWM2        : out std_logic;
        PWM3        : out std_logic;
        PWM4        : out std_logic;

        CAPTION_IN  : in std_logic);

    attribute buffer_sig : string;
    attribute buffer_sig of CLK     : signal is "BUFGP";
    attribute buffer_sig of H_SYNC  : signal is "BUFGP";
    attribute buffer_sig of nIR     : signal is "BUFGS";

                        -- assign pin number
    type string_array is
        array (natural range<>, natural range <>) of character;

    attribute pin_number : string;
    attribute array_pin_number : string_array;
    attribute array_pin_number of DATA_BUS      : signal is
("P56","P58","P59","P61","P65","P67","P69","P71");
    attribute array_pin_number of ADDR          : signal is
("P10","P9","P8","P7","P6","P5","P4","P3","P84",

"P83","P82","P81","P80","P79","P78","P77");
    attribute pin_number of CLK             : signal is "P13";
    attribute pin_number of nRESET          : signal is "P72";
    attribute pin_number of nCE_ROM         : signal is "P62";
    attribute pin_number of nCS_RAM         : signal is "P70";
    attribute pin_number of nRD             : signal is "P68";
    attribute pin_number of nWR             : signal is "P66";
```

```
    attribute pin_number of OE_ADC          : signal is "P60";

    attribute array_pin_number of P0        : signal is ("P44","P45","P46");
    attribute   array_pin_number   of   P1                          :   signal   is
("P14","P15","P16","P17","P18","P19","P20","P23");
    attribute array_pin_number of P2        : signal is ("P47","P48","P49","P50");

    attribute pin_number of H_SYNC          : signal is "P35";
    attribute pin_number of V_SYNC          : signal is "P36";
    attribute pin_number of nFIELD_IN       : signal is "P40";

    attribute pin_number of R               : signal is "P37";
    attribute pin_number of G               : signal is "P38";
    attribute pin_number of B               : signal is "P39";

    attribute pin_number of nIR             : signal is "P29";

    attribute pin_number of ADC_CLK         : signal is "P51";

    attribute pin_number of PWM0            : signal is "P24";
    attribute pin_number of PWM1            : signal is "P25";
    attribute pin_number of PWM2            : signal is "P26";
    attribute pin_number of PWM3            : signal is "P27";
    attribute pin_number of PWM4            : signal is "P28";

    attribute pin_number of CAPTION_IN      : signal is "P57";
end MCU;

architecture MASK of MCU is
    attribute buffer_sig of CLK_DIV2    : signal is "BUFGS";
    attribute buffer_sig of CLK_DIV24   : signal is "BUFGS";
    attribute buffer_sig of CLK_DIV256  : signal is "BUFGS";

component CPU
port(   CLK         : in std_logic;
        nRESET      : in std_logic;
        HALT        : in std_logic;
        nCE_ROM     : out std_logic;            -- ROM Chip Enable
        nCS_RAM     : out std_logic;            -- RAM Chip Select (+Output Enable)
        nRD         : out std_logic;            -- READ
        nWR         : out std_logic;            -- WRITE
        DATA_IN     : in std_logic_vector(7 downto 0);
        DATA_OUT    : out std_logic_vector(7 downto 0);
        Z_CT        : out std_logic;
        STORE_DIR       : out std_logic;                                -- load or store
direction
        IO_ADDR          : out std_logic_vector(3 downto 0); -- internal address for
memory map i/o
        ADDR        : out std_logic_vector(14 downto 0);
        OSD_STATE   : out std_logic;
        OE_ADC      : out std_logic);
end component;

component DIV_CLK
port(CLK        : in std_logic;
     nRESET     : in std_logic;
     DIV2       : out std_logic;
     DIV4       : out std_logic;
     DIV8       : out std_logic;
     DIV16      : out std_logic;
     DIV24      : out std_logic;
     DIV32      : out std_logic;
     DIV64      : out std_logic;
     DIV128     : out std_logic;
     DIV256     : out std_logic);
end component;

component IO_PORT
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR   : in std_logic_vector(3 downto 0);
          DATA_IN   : in std_logic_vector(7 downto 0);
```

```
          DATA_OUT   : out std_logic_vector(7 downto 0);
          Z_CT       : out std_logic;
          P0_IN      : in std_logic_vector(2 downto 0);
          P1_OUT     : out std_logic_vector(7 downto 0);
          P2_OUT     : out std_logic_vector(3 downto 0));
end component;

component GEN_OSC
port(   nRESET  : in std_logic;
        F8M     : out std_logic;
        F500K   : out std_logic;
        F16K    : out std_logic;
        F490    : out std_logic;
        F245    : out std_logic;
        F123    : out std_logic;
        F41     : out std_logic;
        F15     : out std_logic;
        F7      : out std_logic;
        F4      : out std_logic;
        F2      : out std_logic;
        F1      : out std_logic;
        F05     : out std_logic);
end component;

component OSD
port(   H_SYNC  : in std_logic;
        V_SYNC  : in std_logic;
        CLK     : in std_logic;
        CLK_DIV2: in std_logic;
        F41     : in std_logic;
        F1      : in std_logic;
        FIELD   : in std_logic;
        DATA_IN : in std_logic_vector(7 downto 0);
        ADDR_OUT: out std_logic_vector(14 downto 0);
        nCE_ROM : out std_logic;
        nCS_RAM : out std_logic;
        R       : out std_logic;
        G       : out std_logic;
        B       : out std_logic;
        HALT_OUT: out std_logic;
        STORE_DIR   : in std_logic;
        IO_ADDR     : in std_logic_vector(3 downto 0);
        nRESET      : in std_logic);
end component;

component IR_PRE
port(   CLK     : in std_logic;
        nIR     : in std_logic;
        CLK_DIV2    : in std_logic;
        nRESET      : in std_logic;
        STORE_DIR   : in std_logic;
        IO_ADDR     : in std_logic_vector(3 downto 0);
        DATA_IN     : in std_logic_vector(7 downto 0);
        DATA_OUT    : out std_logic_vector(7 downto 0);
        Z_CT        : out std_logic);
end component;

component PWMx5
port(CLK            : in std_logic;
     nRESET         : in std_logic;
     CLK_DIV2       : in std_logic;
     STORE_DIR  : in std_logic;
     IO_ADDR    : in std_logic_vector(3 downto 0);
     DATA_IN    : in std_logic_vector(7 downto 0);
     nPWM_OUT0,nPWM_OUT1,nPWM_OUT2,nPWM_OUT3,nPWM_OUT4  : out std_logic);
end component;

component CAP_PRE
port(CLK        : in std_logic;
     CAP_DATA   : in std_logic;
     FIELD      : in std_logic;
     H_SYNC     : in std_logic;
     V_SYNC     : in std_logic;
```

```
    CLK_DIV2    : in std_logic;
    nRESET      : in std_logic;
    STORE_DIR   : in std_logic;
    IO_ADDR        : in std_logic_vector(3 downto 0);
    DATA_IN     : in std_logic_vector(7 downto 0);
    DATA_OUT    : out std_logic_vector(7 downto 0);
    Z_CT        : out std_logic);
end component;

signal CLK_DIV2 : std_logic;
signal HALT     : std_logic;
signal nCE_ROM_CPU  : std_logic;
signal nCS_RAM_CPU  : std_logic;
signal nRD_CPU      : std_logic;
signal nWR_CPU      : std_logic;
signal DATA_OUT_CPU : std_logic_vector(7 downto 0);
signal Z_CT_CPU     : std_logic;
signal STORE_DIR    : std_logic;
signal IO_ADDR      : std_logic_vector(3 downto 0);
signal ADDR_CPU     : std_logic_vector(14 downto 0);
signal OSD_STATE    : std_logic;

signal INT_DATA_BUS : std_logic_vector(7 downto 0);

signal DATA_OUT_IO_PORT : std_logic_vector(7 downto 0);
signal Z_CT_IO_PORT : std_logic;

signal F41      : std_logic;
signal F1       : std_logic;

signal FIELD    : std_logic;
signal ADDR_OSD : std_logic_vector(14 downto 0);
signal nCE_ROM_OSD  : std_logic;
signal nCS_RAM_OSD  : std_logic;
signal HALT_OUT : std_logic;

signal CLK_DIV256   : std_logic;
signal DATA_OUT_IR_PRE  : std_logic_vector(7 downto 0);
signal Z_CT_IR_PRE  : std_logic;

signal CLK_DIV16    : std_logic;
signal CLK_DIV24    : std_logic;

signal nPWM0    : std_logic;
signal nPWM1    : std_logic;
signal nPWM2    : std_logic;
signal nPWM3    : std_logic;
signal nPWM4    : std_logic;

signal DATA_OUT_CAPTION : std_logic_vector(7 downto 0);
signal Z_CT_CAPTION     : std_logic;

begin

    u1 : CPU port map (CLK_DIV2, nRESET, HALT, nCE_ROM_CPU, nCS_RAM_CPU,
                       nRD_CPU, nWR_CPU, DATA_BUS, DATA_OUT_CPU, Z_CT_CPU,
                       STORE_DIR, IO_ADDR, ADDR_CPU, OSD_STATE, OE_ADC);
    u2 : DIV_CLK port map (CLK, nRESET, DIV2 => CLK_DIV2, DIV16 => CLK_DIV16,
                           DIV24 => CLK_DIV24, DIV256 => CLK_DIV256);
    u3 : IO_PORT port map (CLK_DIV2, nRESET, STORE_DIR, IO_ADDR, DATA_BUS,
                           DATA_OUT_IO_PORT, Z_CT_IO_PORT, P0, P1, P2);
    u4 : GEN_OSC port map (nRESET, F41 => F41, F1 => F1);
    u5 : OSD port map (H_SYNC, V_SYNC, CLK, CLK_DIV2, F41, F1, FIELD,
                       INT_DATA_BUS, ADDR_OSD, nCE_ROM_OSD, nCS_RAM_OSD,
                       R, G, B, HALT_OUT, STORE_DIR, IO_ADDR, nRESET);
    u6 : IR_PRE port map (CLK_DIV256, nIR, CLK_DIV2, nRESET, STORE_DIR,
                          IO_ADDR, INT_DATA_BUS, DATA_OUT_IR_PRE, Z_CT_IR_PRE);
    u7 : PWMx5 port map (CLK_DIV24, nRESET, CLK_DIV2, STORE_DIR, IO_ADDR,
                         INT_DATA_BUS, nPWM0, nPWM1, nPWM2, nPWM3, nPWM4);
    u8 : CAP_PRE port map (CLK, CAPTION_IN, FIELD, H_SYNC, V_SYNC, CLK_DIV2,
                           nRESET, STORE_DIR, IO_ADDR, INT_DATA_BUS,
                           DATA_OUT_CAPTION, Z_CT_CAPTION);
```

```
    ADDR(15)     <= '0';
    DATA_BUS     <= INT_DATA_BUS;
    INT_DATA_BUS<= DATA_BUS;
    HALT     <= HALT_OUT;
    FIELD    <= not(nFIELD_IN);
    ADC_CLK <= CLK_DIV16;
    PWM0     <= not(nPWM0);
    PWM1     <= not(nPWM1);
    PWM2     <= not(nPWM2);
    PWM3     <= not(nPWM3);
    PWM4     <= not(nPWM4);

    process(Z_CT_CPU,DATA_OUT_CPU)
    begin
        if (Z_CT_CPU = '1') then
            INT_DATA_BUS <= "ZZZZZZZZ";
        else
            INT_DATA_BUS <= DATA_OUT_CPU;
        end if;
    end process;

    process(Z_CT_IO_PORT,DATA_OUT_IO_PORT)
    begin
        if (Z_CT_IO_PORT = '1') then
            INT_DATA_BUS <= "ZZZZZZZZ";
        else
            INT_DATA_BUS <= DATA_OUT_IO_PORT;
        end if;
    end process;

    process(Z_CT_IR_PRE,DATA_OUT_IR_PRE)
    begin
        if (Z_CT_IR_PRE = '1') then
            INT_DATA_BUS <= "ZZZZZZZZ";
        else
            INT_DATA_BUS <= DATA_OUT_IR_PRE;
        end if;
    end process;

    process(Z_CT_CAPTION,DATA_OUT_CAPTION)
    begin
        if (Z_CT_CAPTION = '1') then
            INT_DATA_BUS <= "ZZZZZZZZ";
        else
            INT_DATA_BUS <= DATA_OUT_CAPTION;
        end if;
    end process;

    process(OSD_STATE, ADDR_CPU, nCE_ROM_CPU, nCS_RAM_CPU, nRD_CPU, nWR_CPU, ADDR_OSD,
nCE_ROM_OSD, nCS_RAM_OSD)
    begin
        if (OSD_STATE = '0') then
            ADDR(14 downto 0)    <= ADDR_CPU;
            nCE_ROM <= nCE_ROM_CPU;
            nCS_RAM <= nCS_RAM_CPU;
            nRD     <= nRD_CPU;
            nWR     <= nWR_CPU;

        else
            ADDR(14 downto 0)    <= ADDR_OSD;
            nCE_ROM <= nCE_ROM_OSD;
            nCS_RAM <= nCS_RAM_OSD;
            nRD     <= '0';
            nWR     <= '1';

        end if;
    end process;

end MASK;
-- ##############################################################################
------------------------------------------------------------------------
-- OSD
------------------------------------------------------------------------
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity OSD is
port(   H_SYNC  : in std_logic;
        V_SYNC  : in std_logic;
        CLK     : in std_logic;
        CLK_DIV2: in std_logic;
        F41     : in std_logic;
        F1      : in std_logic;
        FIELD   : in std_logic;
        DATA_IN : in std_logic_vector(7 downto 0);
        ADDR_OUT: out std_logic_vector(14 downto 0);
        nCE_ROM : out std_logic;
        nCS_RAM : out std_logic;
        R       : out std_logic;
        G       : out std_logic;
        B       : out std_logic;
        HALT_OUT: out std_logic;
        STORE_DIR   : in std_logic;
        IO_ADDR     : in std_logic_vector(3 downto 0);
        nRESET      : in std_logic);
end OSD;

architecture NETLIST of OSD is

signal PLUS         :  integer range 0 to 7;
signal PLUS_V       :  integer range 0 to 15;
signal ROW          :  integer range 0 to 31;
signal SUB_ROW      :  integer range 0 to 15;
signal COL          :  integer range 0 to 63;
signal SUB_COL      :  integer range 0 to 15;
signal HALT         :  std_logic;
signal HALT2        :  std_logic;
signal DISP_ROW_NO  :  integer range 0 to 3;
signal CODE         :  std_logic_vector(7 downto 0);
signal ATTR         :  std_logic_vector(4 downto 0);    -- F,I,R,G,B
signal FONT         :  std_logic_vector(7 downto 0);
signal ADDR         :  std_logic_vector(15 downto 0);
signal ENABLE_SIG   :  std_logic;
signal ITALIC       :  std_logic;
signal DOT          :  std_logic;
signal FLASH        :  std_logic;
signal COLOR        :  std_logic_vector(2 downto 0);       -- R,G,B
signal THAI     : std_logic;
signal PAGE     : std_logic;
signal SCROLL   : std_logic;
signal SEL      : std_logic;
signal ROW_REG0 : integer range 0 to 15;
signal ROW_REG1 : integer range 0 to 15;
signal ROW_REG2 : integer range 0 to 15;
signal ROW_REG3 : integer range 0 to 15;
signal BACKGROUND   : std_logic;
signal SW           : std_logic;
signal ENABLE       : std_logic;

component OSDTIMER
port(   H_SYNC      : in std_logic;     -- V_CLK (Vertical Clock)
        V_SYNC      : in std_logic;     -- RESET VERTICAL COUNTER
        CLK         : in std_logic;
        CLK_DIV2    : in std_logic;
        SEL         : in std_logic;
        PLUS        : in integer range 0 to 7;
        PLUS_V      : in integer range 0 to 15;
        HALT        : in std_logic;
        HALT2       : out std_logic;
        ROW         : out integer range 0 to 31;
        SUB_ROW     : out integer range 0 to 15;
        COL         : out integer range 0 to 63;
        SUB_COL     : out integer range 0 to 15);
end component;
```

```
component ITAL_CT
port(ITALIC     : in std_logic;
     SUB_ROW    : in integer range 0 to 15;
     PLUS       : out integer range 0 to 7);
end component;

component ADDR_OSD
port(COL     : in integer range 0 to 63;
     SUB_COL : in integer range 0 to 15;
     SUB_ROW : in integer range 0 to 15;
     THAI    : in std_logic;
     PAGE    : in std_logic;
     DISP_ROW_NO    : in integer range 0 to 3;
     FIELD   : in std_logic;
     CODE    : in std_logic_vector(7 downto 0);
     ADDR    : out std_logic_vector(15 downto 0));
end component;

component DATA_BUF
port(   CLK          : in std_logic;
        SUB_COL      : in integer range 0 to 15;
        DATA_IN      : in std_logic_vector(7 downto 0);
        ATTR_DATA    : out std_logic_vector(4 downto 0); -- F,I,R,G,B
        CODE_DATA    : out std_logic_vector(7 downto 0);
        FONT_DATA    : out std_logic_vector(7 downto 0));
end component;

component DOT_SEL
port(   CLK          : in std_logic;
        SUB_COL      : in integer range 0 to 15;
        ENABLE       : in std_logic;
        FONT_DATA    : in std_logic_vector(7 downto 0);
        DOT          : out std_logic);
end component;

component SCRL_CT
 port(SCROLL         : in std_logic;
     CLK         : in std_logic;
     PLUS_V      : out integer range 0 to 15);
end component;

component OSDLOGIC
port(   DOT      : in std_logic;
        F05      : in std_logic;
        BACKGROUND  : in std_logic;
        ENABLE   : in std_logic;
        FLASH    : in std_logic;
        COLOR    : in std_logic_vector(2 downto 0);      -- R,G,B
        R        : out std_logic;
        G        : out std_logic;
        B        : out std_logic;
        SW       : out std_logic);
end component;

component CMP_R_PK
port(   ROW_REG0     : in integer range 0 to 15;
        ROW_REG1     : in integer range 0 to 15;
        ROW_REG2     : in integer range 0 to 15;
        ROW_REG3     : in integer range 0 to 15;
        ROW          : in integer range 0 to 15;
        DISP_ROW     : out integer range 0 to 3;
        HALT         : out std_logic);
end component;

component OSD_REG
    port (CLK        : in std_logic;
          nRESET     : in std_logic;
          STORE_DIR  : in std_logic;
          IO_ADDR    : in std_logic_vector(3 downto 0);
          DATA_IN    : in std_logic_vector(7 downto 0);
          THAI       : out std_logic;
          PAGE       : out std_logic;
          SCROLL     : out std_logic;
```

```
          SEL       : out std_logic;
          ROW_REG0  : out integer range 0 to 15;
          ROW_REG1  : out integer range 0 to 15;
          ROW_REG2  : out integer range 0 to 15;
          ROW_REG3  : out integer range 0 to 15);
end component;

begin

    u1 : OSDTIMER port map (H_SYNC, V_SYNC, CLK, CLK_DIV2, SEL, PLUS,
                            PLUS_V, HALT, HALT2, ROW, SUB_ROW, COL, SUB_COL);
    u2 : ITAL_CT port map (ITALIC, SUB_ROW, PLUS);
    u3 : ADDR_OSD port map (COL, SUB_COL, SUB_ROW, THAI, PAGE, DISP_ROW_NO,
                            FIELD, CODE, ADDR);
    u4 : DATA_BUF port map (CLK, SUB_COL, DATA_IN, ATTR, CODE, FONT);
    u5 : DOT_SEL port map (CLK, SUB_COL, ENABLE_SIG, FONT, DOT);
    u6 : SCRL_CT port map (SCROLL, F41, PLUS_V);
    u7 : OSDLOGIC port map (DOT, F1, BACKGROUND, ENABLE_SIG, FLASH, COLOR,
                            R, G, B, SW);
    u8 : CMP_R_PK port map (ROW_REG0, ROW_REG1, ROW_REG2, ROW_REG3, ROW,
                            DISP_ROW_NO, HALT);
    u9 : OSD_REG port map (CLK_DIV2, nRESET, STORE_DIR, IO_ADDR, DATA_IN,
                           THAI, PAGE, SCROLL, SEL, ROW_REG0, ROW_REG1,
                           ROW_REG2, ROW_REG3);

    ENABLE_SIG  <= HALT2 and ENABLE;
    HALT_OUT    <= ENABLE_SIG;
    nCE_ROM <= ADDR(15);
    nCS_RAM <= not(ADDR(15));
    ADDR_OUT    <= ADDR(14 downto 0);
    FLASH       <= ATTR(4);
    ITALIC      <= ATTR(3);
    COLOR       <= ATTR(2 downto 0);
    BACKGROUND  <= '0';
    ENABLE      <= '1';

end NETLIST;
-- ##############################################################################
------------------------------------------------------------------------
-- OSD register
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity OSD_REG is
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR   : in std_logic_vector(3 downto 0);
          DATA_IN   : in std_logic_vector(7 downto 0);
          THAI      : out std_logic;
          PAGE      : out std_logic;
          SCROLL    : out std_logic;
          SEL       : out std_logic;
          ROW_REG0  : out integer range 0 to 15;
          ROW_REG1  : out integer range 0 to 15;
          ROW_REG2  : out integer range 0 to 15;
          ROW_REG3  : out integer range 0 to 15);
end OSD_REG;

architecture RTL of OSD_REG is

    -- function evec2int (l: std_ulogic_vector)                    return natural;
    -- Translate unsigned vector into integer representation
    ---
       function evec2int (l: std_logic_vector)
             return natural is
          variable result: natural := 0;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
```

```
            for tl in l'range loop
               result := result * 2;
               if (l(tl) = '1' or l(tl) = 'H') then
                  result := result + 1;
               end if;
            end loop;
            return result;
         end evec2int;

begin

    process(CLK,nRESET,IO_ADDR,STORE_DIR)
    begin
        if (nRESET = '0') then
            ROW_REG0    <= 2;
            ROW_REG1    <= 2;
            ROW_REG2    <= 2;
            ROW_REG3    <= 2;
            THAI        <= '0';
            PAGE        <= '0';
            SCROLL      <= '0';
            SEL         <= '0';

        elsif (CLK = '0') and (CLK'event) then
            if (STORE_DIR = '1') then           -- STORE DATA
                if (IO_ADDR = "0100") then
                    ROW_REG0 <= evec2int(DATA_IN(3 downto 0));
                elsif (IO_ADDR = "0101") then
                    ROW_REG1 <= evec2int(DATA_IN(3 downto 0));
                elsif (IO_ADDR = "0110") then
                    ROW_REG2 <= evec2int(DATA_IN(3 downto 0));
                elsif (IO_ADDR = "0111") then
                    ROW_REG3 <= evec2int(DATA_IN(3 downto 0));
                elsif (IO_ADDR = "1000") then
                    THAI    <= DATA_IN(3);
                    PAGE    <= DATA_IN(2);
                    SCROLL  <= DATA_IN(1);
                    SEL     <= DATA_IN(0);
                end if;
            end if;

        end if;

    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- OSD output controller
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity OSDLOGIC is
port(   DOT     : in std_logic;
        F05     : in std_logic;
        BACKGROUND  : in std_logic;
        ENABLE  : in std_logic;
        FLASH   : in std_logic;
        COLOR   : in std_logic_vector(2 downto 0);      -- R,G,B
        R       : out std_logic;
        G       : out std_logic;
        B       : out std_logic;
        SW      : out std_logic);
end OSDLOGIC;

architecture RTL of OSDLOGIC is

signal DOT_SIG  : std_logic;
```

```
begin

    DOT_SIG <= DOT and (not(FLASH) or F05);

    R       <= COLOR(2) and DOT_SIG;
    G       <= COLOR(1) and DOT_SIG;
    B       <= COLOR(0) and DOT_SIG;

    process(ENABLE,BACKGROUND,DOT_SIG)
    begin
        if (ENABLE = '0') then
            SW <= '0';
        elsif (BACKGROUND = '0') then
            SW <= DOT_SIG;
        else
            SW <= '1';
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- OSD timer
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity OSDTIMER is
port(   H_SYNC      : in std_logic;     -- V_CLK (Vertical Clock)
        V_SYNC      : in std_logic;     -- RESET VERTICAL COUNTER
        CLK         : in std_logic;
        CLK_DIV2    : in std_logic;
        SEL         : in std_logic;
        PLUS        : in integer range 0 to 7;
        PLUS_V      : in integer range 0 to 15;
        HALT        : in std_logic;
        HALT2       : out std_logic;
        ROW         : out integer range 0 to 31;
        SUB_ROW     : out integer range 0 to 15;
        COL         : out integer range 0 to 63;
        SUB_COL     : out integer range 0 to 15);
end OSDTIMER;

architecture NETLIST of OSDTIMER is

signal H_CLK    : std_logic;
signal V_CLK    : std_logic;

component H_COUNTR
    port(H_SYNC : in std_logic; -- RESET Horizontal counter
         CLK    : in std_logic; -- Clock for Horizontal counter
         PLUS   : in integer range 0 to 7;
         COL    : out integer range 0 to 63;
         SUB_COL: out integer range 0 to 15);
end component;

component V_COUNTR
    port(V_SYNC : in std_logic;     -- RESET VERTICAL COUNTER
         H_SYNC : in std_logic;     -- V_CLK (Vertical Clock)
         PLUS_V : in integer range 0 to 15;
         HALT   : in std_logic;
         HALT2  : out std_logic;
         ROW    : out integer range 0 to 31;
         SUB_ROW: out integer range 0 to 15);
end component;

component SEL_CLK
port(   CLK     : in std_logic;
        CLK_DIV2: in std_logic;
        H_SYNC  : in std_logic;
```

```
        V_SYNC  : in std_logic;
        SEL     : in std_logic;
        H_CLK   : out std_logic;
        V_CLK   : out std_logic);
end component;

begin

    u1 : H_COUNTR port map (H_SYNC, H_CLK, PLUS, COL, SUB_COL);
    u2 : V_COUNTR port map (V_SYNC, V_CLK, PLUS_V, HALT, HALT2, ROW, SUB_ROW);
    u3 : SEL_CLK port map (CLK, CLK_DIV2, H_SYNC, V_SYNC,  SEL, H_CLK, V_CLK);

end NETLIST;
-- ##############################################################################
------------------------------------------------------------------------
-- 10-bit adder/subtractor
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity PWM_REG is
    port (CLK       : in std_logic;
          nRESET    : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR   : in std_logic_vector(3 downto 0);
          DATA_IN   : in std_logic_vector(7 downto 0);
          PWM0L     : out std_logic_vector(6 downto 0);
          PWM0H     : out std_logic_vector(6 downto 0);
          PWM1      : out std_logic_vector(5 downto 0);
          PWM2      : out std_logic_vector(5 downto 0);
          PWM3      : out std_logic_vector(5 downto 0);
          PWM4      : out std_logic_vector(5 downto 0));
end PWM_REG;

architecture RTL of PWM_REG is

begin

    process(CLK,nRESET,IO_ADDR,STORE_DIR)
    begin
        if (nRESET = '0') then
            PWM0L   <= "0000000";
            PWM0H   <= "0000000";
            PWM1    <= "000000";
            PWM2    <= "000000";
            PWM3    <= "000000";
            PWM4    <= "000000";

        elsif (CLK = '0') and (CLK'event) then
            if (STORE_DIR = '1') then           -- STORE DATA
                if (IO_ADDR = "1001") then
                    PWM1    <= DATA_IN(5 downto 0);
                elsif (IO_ADDR = "1010") then
                    PWM2    <= DATA_IN(5 downto 0);
                elsif (IO_ADDR = "1011") then
                    PWM3    <= DATA_IN(5 downto 0);
                elsif (IO_ADDR = "1100") then
                    PWM4    <= DATA_IN(5 downto 0);
                elsif (IO_ADDR = "1101") then
                    PWM0L   <= DATA_IN(6 downto 0);
                elsif (IO_ADDR = "1110") then
                    PWM0H   <= DATA_IN(6 downto 0);
                end if;
            end if;

        end if;

    end process;

end RTL;
```

```
-- ##############################################################################
--------------------------------------------------------------------------------
-- PWM x 5 unit
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity PWMx5 is
port(CLK            : in std_logic;
     nRESET         : in std_logic;
     CLK_DIV2       : in std_logic;
     STORE_DIR  : in std_logic;
     IO_ADDR    : in std_logic_vector(3 downto 0);
     DATA_IN    : in std_logic_vector(7 downto 0);
     nPWM_OUT0,nPWM_OUT1,nPWM_OUT2,nPWM_OUT3,nPWM_OUT4  : out std_logic);
end PWMx5;

architecture NETLIST of PWMx5 is

    component COUNT14B
    port(CLK         : in std_logic;
         nRESET      : in std_logic;
         COUNT7HIGH : out std_logic_vector(6 downto 0);
         COUNT7LOW  : out std_logic_vector(6 downto 0));
    end component;

    component CMP7B
    port(CLK         : in std_logic;
         COUNT7      : in std_logic_vector(6 downto 0);
         DATA_IN     : in std_logic_vector(6 downto 0);
         nPWM_OUT    : out std_logic);
    end component;

    component ADD_PLS
    port(DATA_IN     : in std_logic_vector(6 downto 0);  -- input 7-bit
         COUNT7HIGH : in std_logic_vector(6 downto 0); -- counter 7-bit high order
         COUNT7LOW  : in std_logic_vector(6 downto 0); -- counter 7-bit low order
         nPWM8BIT    : in std_logic; -- input from pwm 7-bit input
         nPWM14BIT  : out std_logic);   -- output pwm 14-bit
    end component;

    component PWM_REG
    port (CLK        : in std_logic;
          nRESET     : in std_logic;
          STORE_DIR : in std_logic;
          IO_ADDR    : in std_logic_vector(3 downto 0);
          DATA_IN    : in std_logic_vector(7 downto 0);
          PWM0L      : out std_logic_vector(6 downto 0);
          PWM0H      : out std_logic_vector(6 downto 0);
          PWM1       : out std_logic_vector(5 downto 0);
          PWM2       : out std_logic_vector(5 downto 0);
          PWM3       : out std_logic_vector(5 downto 0);
          PWM4       : out std_logic_vector(5 downto 0));
    end component;

signal COUNT7_H   : std_logic_vector(6 downto 0);
signal COUNT7     : std_logic_vector(6 downto 0);
signal nPWM0_TEMP : std_logic;

signal PWM0L     : std_logic_vector(6 downto 0);
signal PWM0H     : std_logic_vector(6 downto 0);
signal PWM1      : std_logic_vector(6 downto 0);
signal PWM2      : std_logic_vector(6 downto 0);
signal PWM3      : std_logic_vector(6 downto 0);
signal PWM4      : std_logic_vector(6 downto 0);

signal PWM1_TMP : std_logic_vector(5 downto 0);
signal PWM2_TMP : std_logic_vector(5 downto 0);
signal PWM3_TMP : std_logic_vector(5 downto 0);
signal PWM4_TMP : std_logic_vector(5 downto 0);

begin
```

```
    u1 : CMP7B      port map (CLK,COUNT7,PWM0H,nPWM0_TEMP);
    u2 : CMP7B      port map (CLK,COUNT7,PWM1,nPWM_OUT1);
    u3 : CMP7B      port map (CLK,COUNT7,PWM2,nPWM_OUT2);
    u4 : CMP7B      port map (CLK,COUNT7,PWM3,nPWM_OUT3);
    u5 : CMP7B      port map (CLK,COUNT7,PWM4,nPWM_OUT4);
    u6 : COUNT14B port map (CLK,nRESET,COUNT7_H,COUNT7);
    u7 : ADD_PLS port map (PWM0L,COUNT7_H,COUNT7,nPWM0_TEMP,nPWM_OUT0);
    u8 : PWM_REG port map (CLK_DIV2, nRESET, STORE_DIR, IO_ADDR, DATA_IN,
                  PWM0L, PWM0H, PWM1_TMP, PWM2_TMP, PWM3_TMP, PWM4_TMP);

    PWM1    <= PWM1_TMP&'0';
    PWM2    <= PWM2_TMP&'0';
    PWM3    <= PWM3_TMP&'0';
    PWM4    <= PWM4_TMP&'0';

end NETLIST;
-- ##############################################################################
------------------------------------------------------------------------
-- reset caption clock divider
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity RST_DIV is
port(COUNT255   : in integer range 0 to 255;        -- input TIME BASE
     ENABLE     : in std_logic;                     -- Odd Field & Line 22
     CAP_DATA   : in std_logic;                     -- CAPTION input
     CLK_EN     : out std_logic;                        -- enable CAPTION CLK
     RST_DIV24  : out std_logic);                       -- RESET DIV24 signal
end RST_DIV;

architecture RTL of RST_DIV is
begin
    process(COUNT255,CAP_DATA,ENABLE)
    begin
        if (ENABLE = '1') then                      -- Odd Field & Line 22
            if (COUNT255 < 156) then                    -- CLOCK RUN-IN period
                RST_DIV24  <= CAP_DATA;
                CLK_EN  <= '0';
            else                                    -- after CAPTION DATA period
                RST_DIV24  <= '0';
                CLK_EN <= '1';
            end if;
        else                                        -- NOT (odd & line 22)
            RST_DIV24 <= '1';
            CLK_EN <= '0';
        end if;
    end process;
end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- Scroll controller
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity SCRL_CT is
port(SCROLL     : in std_logic;
     CLK        : in std_logic;
     PLUS_V     : out integer range 0 to 15);
end SCRL_CT;

architecture RTL of SCRL_CT is
signal COUNT    : integer range 0 to 15;

begin

    process(SCROLL,CLK)
    begin
```

```
        if (SCROLL = '0') then                          -- normal display disable counter
            COUNT   <= 0;
        elsif (CLK = '1') and (CLK'event) then          -- count when scrolling
            if (COUNT = 15) then
                COUNT   <= 15;
            else
                COUNT   <= COUNT + 1;
            end if;
        end if;
    end process;

    PLUS_V  <= COUNT;
end RTL;
-- ##############################################################################
-------------------------------------------------------------------------
-- select frequency or half frequency
-------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity SEL_CLK is
port(   CLK     : in std_logic;
        CLK_DIV2: in std_logic;
        H_SYNC  : in std_logic;
        V_SYNC  : in std_logic;
        SEL     : in std_logic;
        H_CLK   : out std_logic;
        V_CLK   : out std_logic);
end SEL_CLK;

architecture RTL of SEL_CLK is

signal H_SYNC_DIV2 : std_logic;

component COUNT1B
port(CLK        : in std_logic;
     nRESET     : in std_logic;
     OUT1BIT    : out std_logic);
end component;

begin
    u1 : COUNT1B port map (H_SYNC, V_SYNC, H_SYNC_DIV2);

    process(SEL,CLK,H_SYNC,CLK_DIV2,H_SYNC_DIV2)
    begin
        if (SEL = '0') then
            H_CLK   <= CLK;
            V_CLK   <= H_SYNC;
        else
            H_CLK   <= CLK_DIV2;
            V_CLK   <= H_SYNC_DIV2;
        end if;
    end process;

end RTL;
-- ##############################################################################
-------------------------------------------------------------------------
-- 12-bit shift register & custom code checker
-------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity SHIFT12B is
port(DATA_IN    : in std_logic;
     SHIFT      : in std_logic;
     GATE       : in std_logic;
     IR_CODE    : out std_logic_vector(5 downto 0));
```

```
end SHIFT12B;

architecture RTL of SHIFT12B is
signal DATA_TMP : std_logic_vector(11 downto 0);
begin
    process(DATA_IN,SHIFT,GATE)
    begin
        if (GATE = '0') then
            DATA_TMP <= "000000000000";
        elsif (SHIFT = '0') and (SHIFT'event) then
            DATA_TMP(10 downto 0) <= DATA_TMP(11 downto 1);
            DATA_TMP(11) <= DATA_IN;
        end if;
    end process;

    process(DATA_TMP)
    begin
        if (DATA_TMP(5 downto 0) = "110101") then       -- DISTAR custom code
            IR_CODE <= DATA_TMP(11 downto 6);
        else
            IR_CODE <= "000000";
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- 19-bit Shift register
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;

entity SHIFT19B is
    port (INPUT   : in std_logic;                          -- SERIAL INPUT
          SHIFT   : in std_logic;                          -- SHIFT when go to HIGH
          RESET   : in std_logic;                          -- RESET when data is read
          START_BIT : out std_logic_vector(2 downto 0);
          CHAR1   : out std_logic_vector(7 downto 0);    -- CHARACTOR No.1
          CHAR2   : out std_logic_vector(7 downto 0)); -- CHARACTOR No.2
end SHIFT19B;

architecture RTL of SHIFT19B is
signal INT_DATA : std_logic_vector(18 downto 0);

begin
    process(INPUT,SHIFT,RESET)                             -- SHIFT PROCESS
    begin
        if (RESET = '1') then
            INT_DATA    <= "0000000000000000000";
        elsif (SHIFT = '1' and SHIFT'event) then
            INT_DATA(17 downto 0) <= INT_DATA(18 downto 1);
            INT_DATA(18)          <= INPUT;
        end if;
    end process;

    START_BIT           <= INT_DATA(2 downto 0);

    CHAR1(7)             <= (INT_DATA(10) xor INT_DATA(9) xor INT_DATA(8) xor INT_DATA
(7)) xor
                           (INT_DATA(6) xor INT_DATA(5) xor INT_DATA(4) xor INT_DATA
(3));
                                                       -- PARITY BIT OF CHARACTOR 1
                                                       -- if correct it is HIGH
    CHAR1(6 downto 0)   <= INT_DATA(9 downto 3);
                                                       -- CHAR 1 DATA (MSB to LSB)
    CHAR2(7)                 <= (INT_DATA(18) xor INT_DATA(17) xor INT_DATA(16) xor
INT_DATA(15)) xor
                           (INT_DATA(14)   xor   INT_DATA(13)   xor   INT_DATA(12)   xor
INT_DATA(11));
                                                       -- PARITY BIT OF CHARACTOR 2
                                                       -- if correct it is HIGH
    CHAR2(6 downto 0)   <= INT_DATA(17 downto 11);
```

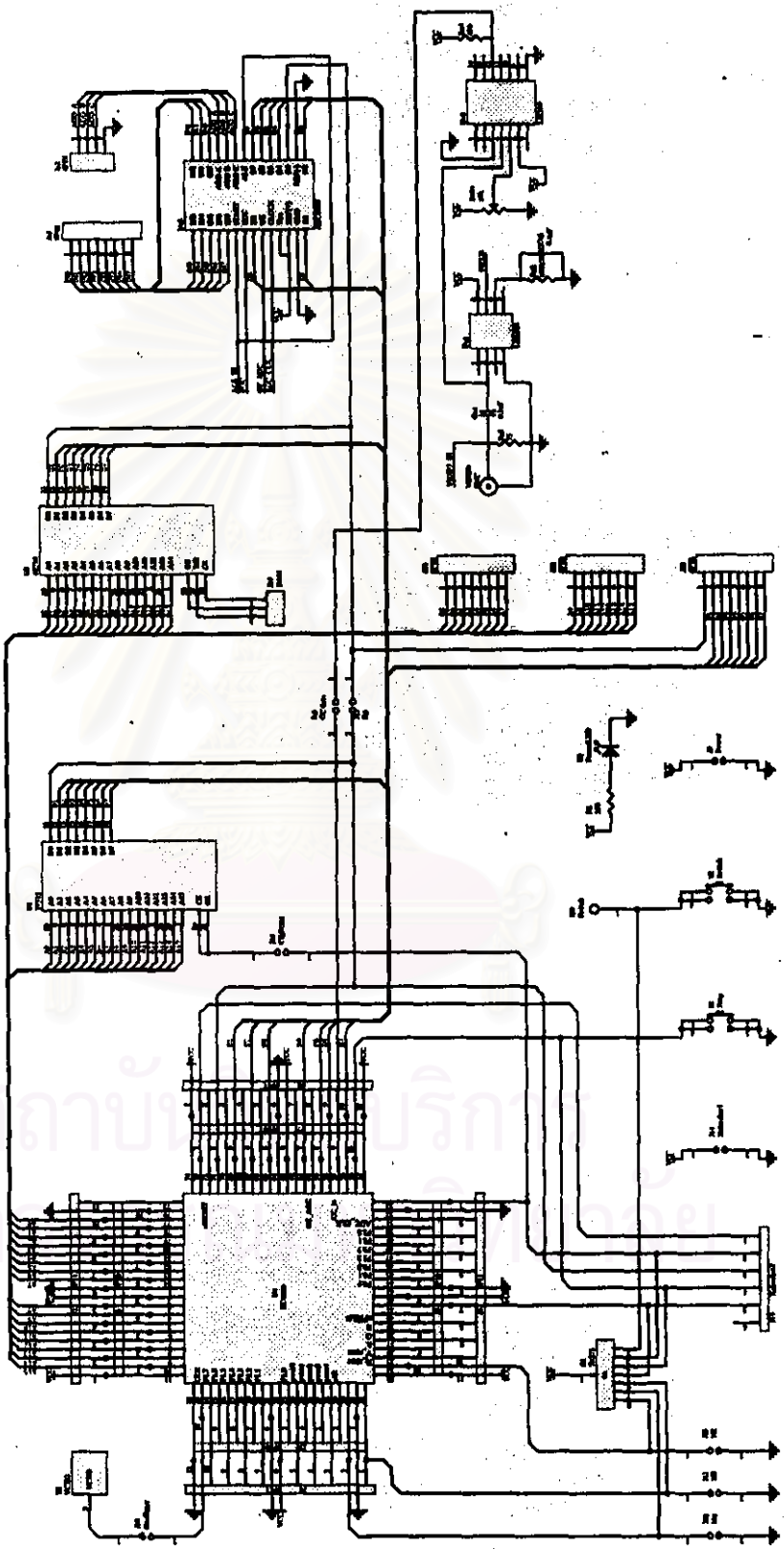
```
                                                        -- CHAR 2 DATA (MSB to LSB)
end RTL;
-- ##############################################################################
-------------------------------------------------------------------------
-- 10-bit adder/subtractor
-------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity DOT_SEL is
port(   CLK          : in std_logic;
        SUB_COL      : in integer range 0 to 15;
        ENABLE       : in std_logic;
        FONT_DATA    : in std_logic_vector(7 downto 0);
        DOT          : out std_logic);
end DOT_SEL;

architecture RTL of DOT_SEL is

signal SUB_COL_BIT   : std_logic_vector(3 downto 0);
signal HALF_SUB_COL  : integer range 0 to 7;

        -- Integer to signed or unsigned vector
       function int2evec (l: integer; size: integer := 32) return std_logic_vector is
          variable result: std_logic_vector(size-1 downto 0);
          variable op: integer := l;
          variable neg_fl: boolean := false;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          result := (size-1 downto 0 => '0');
          if op < 0 then
             op := op * (-1);
             neg_fl := true;
          end if;
          for i in 0 to SIZE-1 loop
             if (op mod 2) = 1 then
                result (i) := '1' ;
             end if ;
             op := op/2 ;
          end loop ;
          return result;
       end int2evec;

    -- function evec2int (l: std_logic_vector)                    return natural;
    -- Translate unsigned vector into integer representation
    ---
       function evec2int (l: std_logic_vector)
             return natural is
          variable result: natural := 0;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          for tl in l'range loop
             result := result * 2;
             if (l(tl) = '1' or l(tl) = 'H') then
                result := result + 1;
             end if;
          end loop;
          return result;
       end evec2int;

begin
    SUB_COL_BIT <= int2evec(SUB_COL,4);
    HALF_SUB_COL    <= evec2int(SUB_COL_BIT(2 downto 0));

    process(FONT_DATA,HALF_SUB_COL,ENABLE,CLK)      -- select bit no. to dot output
    begin
        if (ENABLE = '0') then
            DOT <= '0';
        elsif (CLK = '1') and (CLK'event) then
            DOT <= FONT_DATA(7 - HALF_SUB_COL);
```

```
        end if;
    end process;

end RTL;
-- ##############################################################################
------------------------------------------------------------------------
-- VERTICAL COUNTER
------------------------------------------------------------------------

Library IEEE;
use IEEE.Std_Logic_1164.all;
Library EXEMPLAR;
use exemplar.exemplar.all;

entity V_COUNTR is
    port(V_SYNC : in std_logic;      -- RESET VERTICAL COUNTER
         H_SYNC : in std_logic;      -- V_CLK (Vertical Clock)
         PLUS_V : in integer range 0 to 15;
         HALT   : in std_logic;
         HALT2  : out std_logic;
         ROW    : out integer range 0 to 31;
         SUB_ROW: out integer range 0 to 15);
end V_COUNTR;

architecture RTL of V_COUNTR is

signal DATA9B       : integer range 0 to 511;
signal DATA9B_BIT   : std_logic_vector(8 downto 0);
signal GND          : std_logic;
signal PLUS_V_BIT   : std_logic_vector(8 downto 0);
signal VCC          : std_logic;
signal DATA9B_OUT   : std_logic_vector(8 downto 0);
signal Cout         : std_logic;

    -- function evec2int (l: std_logic_vector)                    return natural;
    -- Translate unsigned vector into integer representation
    ---
       function evec2int (l: std_logic_vector)
             return natural is
          variable result: natural := 0;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          for tl in l'range loop
             result := result * 2;
             if (l(tl) = '1' or l(tl) = 'H') then
                result := result + 1;
             end if;
          end loop;
          return result;
       end evec2int;

        -- Integer to signed or unsigned vector
       function int2evec (l: integer; size: integer := 32) return std_logic_vector is
          variable result: std_logic_vector(size-1 downto 0);
          variable op: integer := l;
          variable neg_fl: boolean := false;
           attribute synthesis_return of result:variable is "FEED_THROUGH" ;
       begin
          result := (size-1 downto 0 => '0');
          if op < 0 then
             op := op * (-1);
             neg_fl := true;
          end if;
          for i in 0 to SIZE-1 loop
             if (op mod 2) = 1 then
                result (i) := '1' ;
             end if ;
             op := op/2 ;
          end loop ;
          return result;
       end int2evec;

component COUNT9B
```

```
    port(nRESET : in std_logic; -- RESET
         CLK    : in std_logic; -- Clock
         DATA9B: out integer range 0 to 511);
end component;

component ADD_SUB9      -- for plus
    port (A    : in std_logic_vector(8 downto 0);
          B    : in std_logic_vector(8 downto 0);
          Cin  : in std_logic;
          ADD  : in std_logic;
          S    : out std_logic_vector(8 downto 0));
end component;
component GN_HALT2
port(   LINE9B      : in std_logic_vector(8 downto 0);
        HALT        : in std_logic;
        HALT2       : out std_logic);
end component;

begin
    VCC         <= '1';
    GND         <= '0';

    DATA9B_BIT  <= int2evec(DATA9B,9);
    PLUS_V_BIT  <= int2evec(PLUS_V,9);

    u1 : COUNT9B port map (V_SYNC, H_SYNC, DATA9B);
    u2 : ADD_SUB9 port map (DATA9B_BIT, PLUS_V_BIT, GND, VCC, DATA9B_OUT);
    u3 : GN_HALT2 port map (DATA9B_BIT, HALT, HALT2);

    ROW     <= evec2int(DATA9B_OUT(8 downto 4));
    SUB_ROW <= evec2int(DATA9B_OUT(3 downto 0));

end RTL;
-- ##############################################################################
```

ภาคผนวก ข

รายละเอียดแผนภาพวงจรบอร์ดต้นแบบของทีวีไมโครคอนโทรลเลอร์



ประวัติผู้เขียน

นายอัษฎางค์ แทนสถิตย์ เกิดวันที่ 4 ธันวาคม พ.ศ. 2518 ที่กรุงเทพมหานคร สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า จากจุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2539 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมไฟฟ้า ที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2540 ในระหว่างการศึกษาระดับมหาบัณฑิตนี้ได้รับทุนการศึกษาจากสำนักงานพัฒนาวิทยาศาสตร์ และเทคโนโลยีแห่งชาติ (สวทช.)