VLSI Signal Process Final Report

A Fast Scheme for Image Size Change in the Compressed Domain

Rakesh Dugad, Student Member, IEEE, and Narendra Ahuja, Fellow IEEE

IEEE Transactions on Circuits and Systems for Video Technology, Vol. 11, no. 4, April 2001

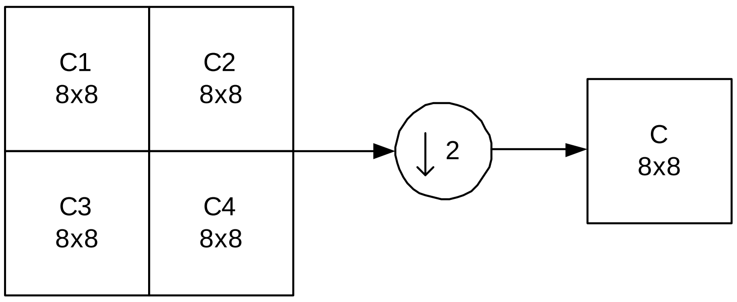
P89921014

張永義

I. Introduction:

改變 Image resize **在影像處理的領域是常常需要用到的運算，也有許多不同的演算法，而在** Video **的應用上如** MPEG-1, MPEG-2, motion JPEG **等等，其原始影像資料有經過壓縮處理，而** Image Resizing **後的資料亦需要經過壓縮，因此在資料壓縮的領域中直接做** Image Resizing **可以省略資料** Decompress **及** Compress **的步驟，本篇論文即是探討在資料壓縮的領域中做快速** Image Size Change **的方法。**

設 c1,c2,c3,c4 **代表** 4 **個彼此相鄰的** 8x8 **的** block**，每兩個** 8x8 **的** block **做平均可得** downsample **的** 8x8 block c.



而此 downsampling **的運算可由下列矩陣式表示：**

$$C = \sum_{i=1}^{4} h_i \, c_i \, g_i \qquad (1)$$

where the downsampling filters hi, gi are given by

$$h_2 = g_1^t = g_3^t = h_1 = \begin{bmatrix} U_{4x8} \\ O_{4x8} \end{bmatrix}$$

$$h_4 = g_2^t = g_4^t = h_3 = \begin{bmatrix} O_{4x8} \\ U_{4x8} \end{bmatrix} \qquad (2)$$

其中

$$U_{4x8} = \begin{bmatrix} .5 & .5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .5 & .5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .5 & .5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .5 & .5 \end{bmatrix}$$

O_{4x8} 是 4x8 的零矩陣。

若要在 DCT domain 直接執行此運算，則由(1)式得

$$DCT(c) = TcT^t = \sum_{i=1}^{4} DCT(h_i)DCT(c_i)DCT(g_i) \qquad (3)$$

其中 T 為 8x8 DCT operator matrix, $TT^t = T^tT = I$.

對 i=1 to 4, DCT(hi)和 DCT(gi)可以事先計算，所以給定 DCT(ci)即可借著矩陣乘法將 DCT(c)算出。但是此種方式並未降低運算量，因此作者提出另一種方式做 Compressed Domain Image Size Change.

II. Downsampling in the DCT domain:

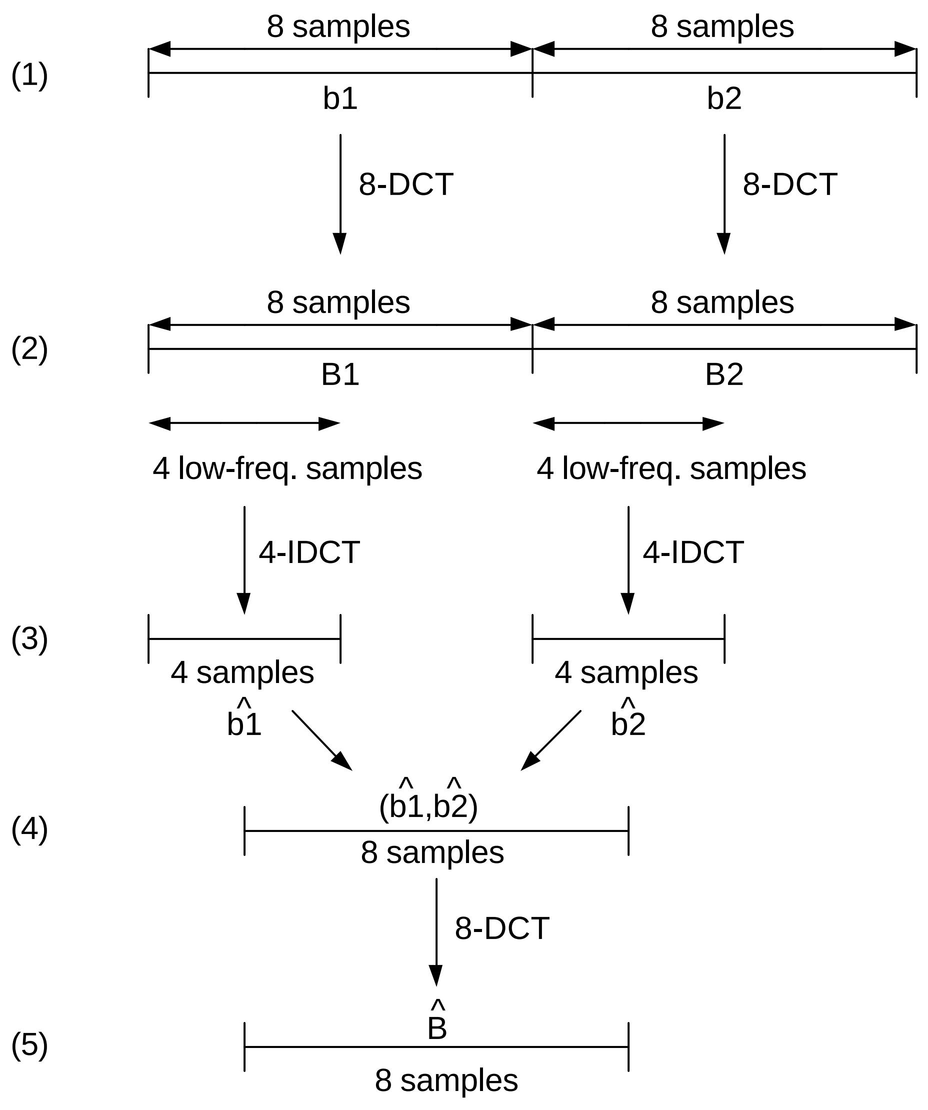


Fig.2

Fig.2 為作者提出的對 1-D signal 在 DCT domain 做 downsampling 的方法，首先對兩個連續的 8 sample blocks b1 和 b2 做 8-point DCT 得到 B1 和 B2，再各取其 4 個低頻係數做 inverse DCT，而其結果合併成 8 個取樣點，再做一次 8-point DCT 可以得到在 compressed domain downsampling 的結果(從 DCT domain 的 B1,B2 到 DCT domain 的 B)，此種做法在矩陣乘法上比(3)式的做法更有效率。

為將此 downsampling 的做法發展成一有效率的演算法，我們先導出 1-D 的方程式，再將其擴充為 2-D。b1,b2 為 spatial domain 的兩個連續的 8-pixel blocks 而 B1,B2 為其 8-point DCT，B1^,B2^是取 B1,B2 的 4 個 low-pass component 而 b1^及 b2^是對此 4 point 做 inverse DCT 的結果，B^為(b1^,b2^)的 DCT 結果，我們需要直接由 B1,B2 計算出 B^。

設 T(8x8)代表 8-point DCT operator matrix，T4(4x4) 代表 4-point DCT operator matrix。則可得下列方程式：

$$\hat{\mathrm{B}} = T\hat{\mathrm{b}} = T\begin{bmatrix} \hat{\mathrm{b}}1 \\ \hat{\mathrm{b}}2 \end{bmatrix} = [\,T_L \;\; T_R\,]\begin{bmatrix} T_4^t\,\hat{\mathrm{B}}_1 \\ T_4^t\,\hat{\mathrm{B}}_2 \end{bmatrix}$$

$$= T_L T_4^t \hat{\mathrm{B}}_1 + T_R T_4^t \hat{\mathrm{B}}_2 \qquad (4)$$

在此 TL 和 TR 是 8x4 的矩陣，分別代表 T 矩陣的前 4 行及後 4 行。

$$T = [\,T_L \mid T_R\,] = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{7\pi}{16} & -\cos\frac{7\pi}{16} & -\cos\frac{5\pi}{16} & -\cos\frac{3\pi}{16} & -\cos\frac{\pi}{16} \\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & -\cos\frac{3\pi}{8} & -\cos\frac{\pi}{8} & -\cos\frac{\pi}{8} & -\cos\frac{3\pi}{8} & \cos\frac{3\pi}{8} & \cos\frac{\pi}{8} \\ \vdots & & & & & & & \vdots \end{array}\right] \qquad (5)$$

$$T_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & -\cos\frac{3\pi}{8} & -\cos\frac{\pi}{8} \\ \cos\frac{2\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{2\pi}{8} \\ \cos\frac{3\pi}{8} & \cos\frac{9\pi}{8} & -\cos\frac{9\pi}{8} & -\cos\frac{3\pi}{8} \end{bmatrix} \qquad (6)$$

1. 對 k = 0,1,2,3,T_L 的(2k)列與 T4 的第 k 列相同，T_R 的(2k)列是 T_4 第 k 列的負數。
2. 因 orthogonal 的原因 8x4 矩陣 $T_L T_4^{\,t}$ 及 $T_R T_4^{\,t}$ 的第(2k)列，除了第 k 項外均為零，故矩陣約有 50%的零項。

3. T **矩陣和** T_4 **矩陣均有對稱項，可利用此對稱項化簡運算。**

$$T_L T_4^t(i,J) = (-1)^{i+j} T_R T_4^t(i,j) \quad \text{for } i = 0,1,...,7$$

$$T_L T_4^{\,t} = C + D \text{ and } T_R T_4^{\,t} = C - D$$

$$\hat{B} = (C+D)\hat{B}_1 + (C-D)\hat{B}_2 \qquad (7)$$

$$= C(\hat{B}_1 + \hat{B}_2) + D(\hat{B}_1 - \hat{B}_2) \qquad (8)$$

4. T 及 T_4 **為** DCT **矩陣，故可運用現有的** Fast DCT algorithms **來處理。**

Extension to 2-D:

設 b1,b2,b3,b4 **代表** 4 **個** 8x8 blocks **而** B1,B2,B3,B4 **代表其** DCT block，B1^,B2^,B3^,B4^**是** 4x4 **的矩陣，為** B1,B2,B3,B4 **的** low-pass **係數，而** b1^,b2^,b3^,b4^**為其** 4x4 inverse DCT。
定義

$$\hat{b} \overset{\text{def}}{=} \begin{bmatrix} \hat{b}_1 & \hat{b}_2 \\ \hat{b}_3 & \hat{b}_4 \end{bmatrix} \quad \text{and} \quad b \overset{\text{def}}{=} \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}$$

B^ = DCT(b^)，**由** B1,B2,B3,B4 **計算出** B^。

$$\hat{B} = T\hat{b}T^{\,t} \qquad (12)$$

$$= [\,T_L \;\; T_R\,] \begin{bmatrix} \hat{b}_1 & \hat{b}_2 \\ \hat{b}_3 & \hat{b}_4 \end{bmatrix} \begin{bmatrix} T_L^t \\ T_R^t \end{bmatrix} \qquad (13)$$

$$= [\,T_L \;\; T_R\,] \begin{bmatrix} T_4^t\hat{B}_1T_4 & T_4^t\hat{B}_2T_4 \\ T_4^t\hat{B}_3T_4 & T_4^t\hat{B}_4T_4 \end{bmatrix} \begin{bmatrix} T_L^t \\ T_R^t \end{bmatrix} \qquad (14)$$

$$= (T_LT_4^t)\hat{B}_1(T_LT_4^t)^t + (T_LT_4^t)\hat{B}_2(T_RT_4^t)^t + (T_RT_4^t)\hat{B}_3(T_LT_4^t)^t + (T_RT_4^t)\hat{B}_4(T_RT_4^t)^t \qquad (15)$$

B^ = (C+D)B1^(C+D) ^{t} + (C+D)B2^(C-D) ^{t} +
    (C-D)B3^(C+D) ^{t} + (C-D)B4^(C-D) ^{t}
=[(C+D)B1^+(C-D)B3^](C+D) ^{t} +[(C+D)B2^+(C-D)B4^](C-D) ^{t}
=[C(B1^+B3^)+D(B1^-B3^)](C+D) ^{t} +
  [C(B2^+B4^)+D(B2^-B4^)](C-D) ^{t} (20)

Where

$$X = C(B1\hat{}+B3\hat{}) + D(B1\hat{}-B3\hat{}) \qquad (21)$$

$$Y = C(B2\hat{}+B4\hat{}) + D(B2\hat{}-B4\hat{}) \qquad (22)$$

比較(20)~(22)**與**(8)**式可知，**B^**可以用** 1-D **的方式來計算。**

III. Upsampling

在許多的應用場合中，需要 Upsampling **的應用，而上述** Image Downsampled **的做法亦可用來做** Image Upsampling。

將 B^**做** IDCT**得一** 8x8 **的** image block b^，**再將其分成** 4 **個** 4x4 **的** sub-blocks，b1^, b2^, b3^, b4^，**其中**

$$\hat{b} = \begin{bmatrix} \hat{b}1 & \hat{b}2 \\ \hat{b}3 & \hat{b}4 \end{bmatrix}$$

計算 4x4 DCT，B1^ = DCT(b1^)，**再將** B1^**視為** Upsample Image **的低頻部分，其他部分補零而的到一** 8x8 DCT，

$$\tilde{B}_1 = \begin{bmatrix} \hat{B}_1 & 0 \\ 0 & 0 \end{bmatrix}$$

再做 IDCT **即可得到一** 8x8 **的** image block，**用同樣方式計算** B2^, B3^, B4^，**可以達到** Upsample **的效果。**

我們可以利用(15)**式得到** B1^, B2^, B3^, B4^,

$$I_{8x8} = T^t\ T = \begin{bmatrix} T_L^t \\ T_R^t \end{bmatrix} \begin{bmatrix} T_L & T_R \end{bmatrix} = \begin{bmatrix} T_L^t T_L & T_L^t T_R \\ T_R^t T_L & T_R^t T_R \end{bmatrix} \qquad (23)$$

因此

$$T_L T_L^t = I_{4x4} = T_R T_R^t; \qquad T_L T_R^t = 0_{4x4} = T_R T_L^t \qquad (24)$$

從這些方程式可得

$$I_{4x4} = T_4(T_L^t T_L)T_4^t = (T_L T_4^t)^t (T_L T_4^t) \qquad (25)$$

同理可得

$$I_{4x4} = (T_L T_4^t)^t (T_L T_4^t);\quad 0_{4x4} = (T_L T_4^t)^t (T_R T_4^t)$$
$$0_{4x4} = (T_R T_4^t)^t (T_L T_4^t) \qquad (26)$$

將(25),(26)**代入**(15)**得**

$$\hat{B}_1 = (T_L T_4^t)^t \hat{B} (T_L T_4^t);\quad \hat{B}_2 = (T_L T_4^t)^t \hat{B} (T_R T_4^t) \qquad (27)$$
$$\hat{B}_3 = (T_R T_4^t)^t \hat{B} (T_L T_4^t);\quad \hat{B}_4 = (T_R T_4^t)^t \hat{B} (T_R T_4^t) \qquad (28)$$

而

$$\tilde{B}_1 = \begin{bmatrix} \hat{B}_1 & 0 \\ 0 & 0 \end{bmatrix}$$ **其它項亦同理可得。**

在 Secton II **中，**$T_L T_4{}^t = C + D$ **和** $T_R T_4{}^t = C - D$**，亦可應用於**(27)(28)**，計算出** $C^t BC$, $C^t BD$, $D^t BC$, $D^t BD$ **以求得** B1^ etc.

IV. ASIC **實作**(DCT parts, **使用** VHDL **硬體描述語言**)

```
--
-- dct.vhd
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity dct is
   port( yuv_in: in std_logic_vector(7 downto 0);
         iSelLum: in std_logic_vector(1 downto 0);
         clk,RESET,START: in std_logic;
         DCToutput: out std_logic_vector(10 downto 0);
-- sram interface
         DCToeb: out std_logic;
```

```
          DCTdbus: inout std_logic_vector(15 downto 0);
          DCTweb: out std_logic;
          DCTabus: out std_logic_vector(5 downto 0)
      );
   end;

   architecture arc of dct is
      signal Yorder, Xlifo, Ylifo: std_logic_vector(15 downto 0);
      signal YInput: std_logic_vector(15 downto 0);
      signal a, c, f, b, d, e, g: std_logic_vector(21 downto 0);
      signal X0, X1, X2, X3, X4, X5, X6, X7: std_logic_vector(21 downto 0);
      signal Yout, Xlifo_out, Ylifo_out: std_logic_vector(21 downto 0);
      signal counter1, counter2: std_logic_vector(6 downto 0);
   --------------------------------------------------------------------------
-
      signal ilifo0,ilifo1,ilifo2,ilifo3: std_logic_vector(15 downto 0);
      signal olifo0,olifo1,olifo2,olifo3: std_logic_vector(21 downto 0);
      signal XYadd,XYsub: std_logic_vector(16 downto 0);
      signal XYadd1,XYadd2,XYadd3,XYadd4,XYadd5,XYadd7: std_logic_vector(21
downto 0);
      signal XYadd8,XYadd9,XYadd10,XYadd14: std_logic_vector(21 downto 0);
      signal XYsub1,XYsub2,XYsub3,XYsub4,XYsub5,XYsub7: std_logic_vector(21
downto 0);
      signal XYsub8,XYsub9,XYsub11,XYsub12,XYsub13:  std_logic_vector(21 downto
0);
      signal atmp,ctmp,ftmp,btmp,dtmp,etmp,gtmp:  std_logic_vector(21 downto
0);
      signal detmp,bdtmp,actmp,atmp2,dtmp2,dtmp3: std_logic_vector(21 downto
0);
      signal atmp1,ctmp1,ftmp1,btmp1,dtmp1,etmp1,gtmp1: std_logic_vector(21
downto 0);
      signal atemp,ctemp,ftemp,btemp,dtemp,etemp,gtemp: std_logic_vector(21
downto 0);
      signal neg_a, neg_b, neg_c, neg_d, neg_e, neg_f, neg_g: std_logic_vector(21
downto 0);
      signal X1_mux, X2_mux, X3_mux,X4_mux: std_logic_vector(21 downto 0);
      signal X5_mux, X6_mux, X7_mux: std_logic_vector(21 downto 0);
      signal add0_out, add1_out, add2_out, add3_out: std_logic_vector(21 downto
0);
      signal add4_out, add5_out, add6_out, add7_out: std_logic_vector(21 downto
0);
      signal accu0_reg, accu1_reg, accu2_reg, accu3_reg: std_logic_vector(21
downto 0);
      signal accu4_reg, accu5_reg, accu6_reg, accu7_reg: std_logic_vector(21
downto 0);
      signal Xin_wire, Yin_wire: std_logic_vector(21 downto 10);
      signal tcounter,counter: std_logic_vector(6 downto 0);
      signal taddress, address_tmp: std_logic_vector(5 downto 0);
      signal tmp1,RowInput: std_logic_vector(15 downto 0);
      signal img_da,img_data: std_logic_vector(7 downto 0);
      signal yuv_in_b7,temp1,RW,OEb,nWA,Web: std_logic;
      signal DCTout: std_logic_vector(10 downto 0);
   begin

      DCToeb<= OEb;

   process(clk)
   begin
      if clk='1' and clk'event then
         img_data<= yuv_in(7 downto 0);
      end if;
   end process;

   mux_in:
   process(img_data,YInput,counter1)
   begin
      if counter1(2)='1' then
         Xlifo<= img_data(7) & img_data(7) & img_data & "000000";
         Yorder<= YInput;
      else
         Xlifo<= YInput;
         Yorder<= img_data(7) & img_data(7) & img_data & "000000";
      end if;
   end process;

      Ylifo<= ilifo3;
```

```
lifo_in:
process(clk)
begin
  if clk='1' and clk'event then
    if RESET='0' then
      ilifo0<= (others=> '0');
      ilifo1<= (others=> '0');
      ilifo2<= (others=> '0');
      ilifo3<= (others=> '0');
    elsif START='1' then
      if counter1(1 downto 0)=3 then
        ilifo3<= Yorder;
      else
        ilifo3<= ilifo2;
      end if;
      if counter1(1 downto 0)=2 then
        ilifo2<= Yorder;
      elsif counter1(1 downto 0)<2 then
        ilifo2<= ilifo1;
      end if;
      if counter1(1 downto 0)=1 then
        ilifo1<= Yorder;
      elsif counter1(1 downto 0)<1 then
        ilifo1<= ilifo0;
      end if;
      if counter1(1 downto 0)=0 then
        ilifo0<= Yorder;
      end if;
    end if;
  end if;
end process;
------------------------------------------------------------
  XYadd<= ( Ylifo(15) & Ylifo ) + ( Xlifo(15) & Xlifo );
  XYsub<= ( Ylifo(15) & Ylifo ) - ( Xlifo(15) & Xlifo );

ACF_BDEG:
process(clk)
```

```
    begin
       if clk='1' and clk'event then
          if RESET='0' then
             a<= (others=> '0');
             b<= (others=> '0');
             c<= (others=> '0');
             d<= (others=> '0');
             e<= (others=> '0');
             f<= (others=> '0');
             g<= (others=> '0');
          elsif START='1' then
             a<= atmp;
             b<= btmp;
             c<= ctmp;
             d<= dtmp;
             e<= etmp;
             f<= ftmp;
             g<= gtmp;
          end if;
       end if;
    end process;

       XYadd1<= ( XYadd(16) & XYadd & "0000" );
       XYadd2<= ( XYadd(16) & XYadd(16) & XYadd & "000" );
       XYadd3<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd & "00" );
       XYadd4<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd & '0' );
      XYadd5<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd );
       XYadd7<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) &
                  XYadd(16) & XYadd(16) & XYadd(16 downto 2) );
       XYadd8<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) &
                  XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16 downto 3) );
       XYadd9<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) &
                 XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16 downto
4) );
       XYadd10<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) &
                   XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) &
                   XYadd(16 downto 5) );
       XYadd14<= ( XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) &
                   XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) &
                   XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16) & XYadd(16 downto
9) );

       XYsub1<= ( XYsub(16) & XYsub & "0000" );
       XYsub2<= ( XYsub(16) & XYsub(16) & XYsub & "000" );
       XYsub3<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub & "00" );
       XYsub4<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub & '0' );
      XYsub5<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub );
       XYsub7<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                  XYsub(16) & XYsub(16) & XYsub(16 downto 2) );
       XYsub8<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                  XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16 downto 3) );
       XYsub9<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                 XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16 downto
4) );
       XYsub11<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                   XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                   XYsub(16) & XYsub(16 downto 6) );
       XYsub12<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                   XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                   XYsub(16) & XYsub(16) & XYsub(16 downto 7) );
       XYsub13<= ( XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                   XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16) &
                   XYsub(16) & XYsub(16) & XYsub(16) & XYsub(16 downto 8) );
    -----------------------------------------------------------
    process(clk)
    begin
      if clk='1' and clk'event then
       atemp<= XYadd2 + XYadd4;
       ctemp<= XYadd1 + XYadd10;
       ftemp<= XYadd3 + XYadd4;
       btemp<= XYsub1 + XYsub13;
       dtemp<= XYsub3 - XYsub12;
       detmp<= XYsub2 + XYsub5;
       gtemp<= XYsub4 + XYsub5;
      end if;
    end process;
```

```
process(clk)
begin
 if clk='1' and clk'event then
   actmp<= XYadd5 + XYadd7;
   ftmp1<= XYadd8 - XYadd14;
   bdtmp<= XYsub7 + XYsub9;
   etmp1<= XYsub11 - XYsub8;
   gtmp1<= XYsub8 - XYsub13;
 end if;
end process;

   atmp2<= actmp + XYadd9;
   dtmp3<= detmp + bdtmp;

   atmp<= atemp + atmp2;
   ftmp<= ftemp + ftmp1;
   dtmp<= dtemp + dtmp3;
   etmp<= detmp + etmp1;
   gtmp<= gtemp + gtmp1;
   ctmp<= ctemp - actmp;
   btmp<= btemp - bdtmp;
------------------------------------------------------------
   neg_a<= not a + 1;
   neg_b<= not b + 1;
   neg_c<= not c + 1;
   neg_d<= not d + 1;
   neg_e<= not e + 1;
   neg_f<= not f + 1;
   neg_g<= not g + 1;

accu:
process(counter2,a,b,c,d,e,f,g,neg_a,neg_b,neg_c,neg_d,neg_e,neg_f,neg_g)
begin
   case counter2(1 downto 0) is
      when "00" =>
         X2_mux<= neg_c;
         X4_mux<= a;
```

```
        X6_mux<= neg_f;
        X1_mux<= g;
        X3_mux<= neg_e;
        X5_mux<= d;
        X7_mux<= neg_b;
     when "01" =>
        X2_mux<= neg_f;
        X4_mux<= neg_a;
        X6_mux<= c;
        X1_mux<= e;
        X3_mux<= neg_b;
        X5_mux<= g;
        X7_mux<= d;
     when "10" =>
        X2_mux<= f;
        X4_mux<= neg_a;
        X6_mux<= neg_c;
        X1_mux<= d;
        X3_mux<= neg_g;
        X5_mux<= neg_b;
        X7_mux<= neg_e;
     when others =>
        X2_mux<= c;
        X4_mux<= a;
        X6_mux<= f;
        X1_mux<= b;
        X3_mux<= d;
        X5_mux<= e;
        X7_mux<= g;
   end case;
end process;
---------------------------------------------------------------

   add0_out<= a + accu0_reg;
   add2_out<= X2_mux + accu2_reg;
   add4_out<= X4_mux + accu4_reg;
   add6_out<= X6_mux + accu6_reg;
   add1_out<= X1_mux + accu1_reg;
   add3_out<= X3_mux + accu3_reg;
   add5_out<= X5_mux + accu5_reg;
   add7_out<= X7_mux + accu7_reg;

X0_7U:
process(clk)
begin
   if clk='1' and clk'event then
      if RESET='0' then
         accu0_reg<= (others=> '0');
         accu1_reg<= (others=> '0');
         accu2_reg<= (others=> '0');
         accu3_reg<= (others=> '0');
         accu4_reg<= (others=> '0');
         accu5_reg<= (others=> '0');
         accu6_reg<= (others=> '0');
         accu7_reg<= (others=> '0');
         X0<= (others=> '0');
         X1<= (others=> '0');
         X2<= (others=> '0');
         X3<= (others=> '0');
         X4<= (others=> '0');
         X5<= (others=> '0');
         X6<= (others=> '0');
         X7<= (others=> '0');
      elsif START='1' then
         case counter2(1 downto 0) is
            when "11" =>
               accu0_reg<= (others=> '0');
               accu1_reg<= (others=> '0');
               accu2_reg<= (others=> '0');
               accu3_reg<= (others=> '0');
               accu4_reg<= (others=> '0');
               accu5_reg<= (others=> '0');
               accu6_reg<= (others=> '0');
               accu7_reg<= (others=> '0');
               X0<= add0_out;
               X1<= add1_out;
```

```
             X2<= add2_out;
             X3<= add3_out;
             X4<= add4_out;
             X5<= add5_out;
             X6<= add6_out;
             X7<= add7_out;
           when others=>
             accu0_reg<= add0_out;
             accu1_reg<= add1_out;
             accu2_reg<= add2_out;
             accu3_reg<= add3_out;
             accu4_reg<= add4_out;
             accu5_reg<= add5_out;
             accu6_reg<= add6_out;
             accu7_reg<= add7_out;
         end case;
       end if;
   end if;
end process;

------------------------------------------------------------
mux_middle:
process(X0,X1,X2,X3,X4,X5,X6,X7,counter2)
begin
   case counter2(1 downto 0) is
     when "00" =>
       Xlifo_out<= X0;
       Yout<= X7;
     when "01" =>
       Xlifo_out<= X1;
       Yout<= X6;
     when "10" =>
       Xlifo_out<= X2;
       Yout<= X5;
     when others =>
       Xlifo_out<= X3;
       Yout<= X4;
   end case;
end process;

   Ylifo_out<= olifo3;

lifo_out:
process(clk)
begin
   if clk='1' and clk'event then
      if RESET='0' then
         olifo0<= (others=> '0');
         olifo1<= (others=> '0');
         olifo2<= (others=> '0');
         olifo3<= (others=> '0');
      elsif START='1' then
         if counter2(1 downto 0)=3 then
            olifo3<= Yout;
         else
            olifo3<= olifo2;
         end if;
         if counter2(1 downto 0)=2 then
            olifo2<= Yout;
         elsif counter2(1 downto 0)<2 then
            olifo2<= olifo1;
         end if;
         if counter2(1 downto 0)=1 then
            olifo1<= Yout;
         elsif counter2(1 downto 0)<1 then
            olifo1<= olifo0;
         end if;
         if counter2(1 downto 0)=0 then
            olifo0<= Yout;
         end if;
      end if;
   end if;
end process;

   Xin_wire<= Xlifo_out(21 downto 10) + 1;
   Yin_wire<= Ylifo_out(21 downto 10) + 1;
```

```
mux_out:
process(Xin_wire,Yin_wire,counter2)
begin
   if counter2(2)='1' then
      DCTout<= Xin_wire(21 downto 11);
   else
      DCTout<= Yin_wire(21 downto 11);
   end if;
end process;

   DCToutput<= DCTout;

process(OEb,counter2,Ylifo_out,Xlifo_out)
begin
   if OEb='1' then
      if counter2(2)='1' then
         DCTdbus<= Ylifo_out(20 downto 5);
      else
         DCTdbus<= Xlifo_out(20 downto 5);
      end if;
   else
      DCTdbus<= (others=> 'Z');
   end if;
end process;

   counter<= counter1 + 1;
   taddress<= counter1(5 downto 0) when counter1(6)='1' else
             ( counter1(2 downto 0) & counter1(5 downto 3) );

AG:
process(clk)
begin
    if clk='1' and clk'event then
      if RESET='0' then
         counter1<= "0110111";     -- 55 original
         tcounter<= (others=> '0');
         counter2<= (others=> '0');
```

```vhdl
      elsif START='1' then
        counter1<= counter;
        tcounter<= counter1;
        counter2<= tcounter;
        address_tmp<= taddress;
        DCTabus<= address_tmp;
      end if;
   end if;
end process;

   OEb<= not RW;

process(clk)
begin
   if clk='1' and clk'event then
      nWA<= RW;
   end if;
end process;

   DCTweb<= nWA nand clk;

process(clk)
begin
   if clk='1' and clk'event then
      if RESET='0' then
        temp1<= '1';
        RW<= '1';
      else
        temp1<= START;
        RW<= START and not temp1;
      end if;
   end if;
end process;

process(clk)
begin
   if clk='1' and clk'event then
      if OEb='0' then
```

```
            RowInput<= DCTdbus;
        end if;
    end if;
end process;

MEM:
process(clk)
begin
    if clk='1' and clk'event then
        if RESET='0' then
            tmp1<= (others=> '0');
            YInput<= (others=> '0');
        elsif START='1' then
            tmp1<= RowInput;
            YInput<= tmp1;
        end if;
    end if;
end process;

end;
```