APPLICATION OF FPGAs IN FLEXIBLE ANALOGUE ELECTRONIC IMAGES GENERATOR DESIGN

P. Kulla, Š. Slávik, J. Huska

Department of Radio and Electronics, FEI SUT, Ilkovičova 3, Bratislava, 812 19
tel.:+421 (0)7 602 91 896 (863), E-mail: kulla@elf.stuba.sk
www: http:\\www.elf.stuba.sk\~kulla\index.htm

Summary This paper focuses on usage of the FPGAs (Field Programmable Gate Arrays) Xilinx as a part of our more complex work dedicated to design of flexible analogue electronic images generator for application in TV measurement technique or/and TV service technique or/and education process. The FPGAs performs here the role of component colour R, G, B, synchronization and blanking signals source. These signals are next processed and amplified in other parts of the generator as NTSC/PAL source encoder and RF modulator. The main aim of this paper is to show the possibilities how with suitable development software use a FPGAs in analog TV technology.

1 DESCRIPTION OF ANALOGUE ELECTRONIC IMAGES GENERATOR

Analogue electronic images (AEI) generator is device which electronically generates in most cases testing images in appropriate TV standard system, e.g. vertical colour bars. Therein discussed generator works in E-NTSC/PAL systems and is capable to generate one, two, three or four color bar octets, as it can be seen on Fig.1, where one octet of bars is shown. Component colour R, G and B signals, which correlate with this bar octet, are then shown on Fig.2. As we can see, signals work only with two levels, therefore they can be generated using binary logic, e.g. based on FPGAs technology. Besides

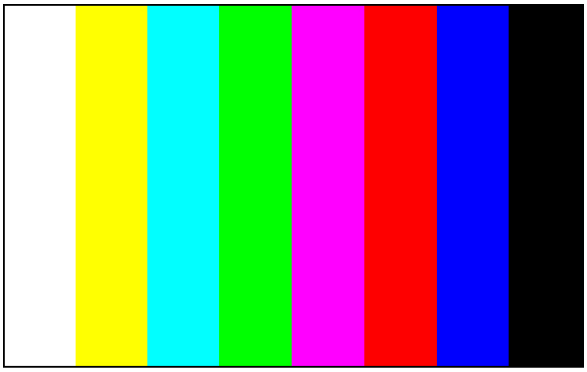

Fig.1 One octet of colour bars

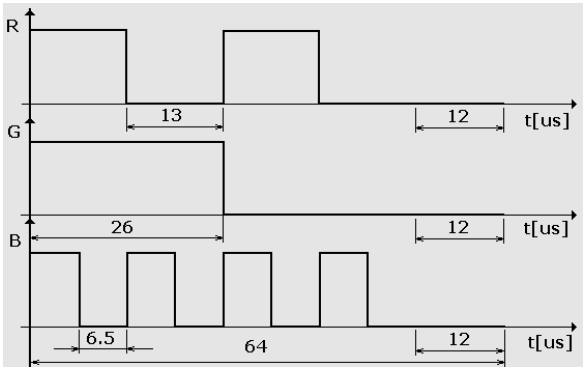

Fig.2 Component R, G and B signals of colour bar octet

component colour R, G, B signals, AEI generator generates also composite synchronization signal and composite blanking signal for TV raster synchronization and blanking, [1]. They can be seen on Fig.3. These signals, especially for PAL system, are described in norms CCIR – B, G (or D, K). Even if it does not have to seem too clear, these signals can also be generated as two valued (the different levels, shown on Fig.3, are obtained by different amplification and non - additive superposition outside of FPGA). Once we showed that

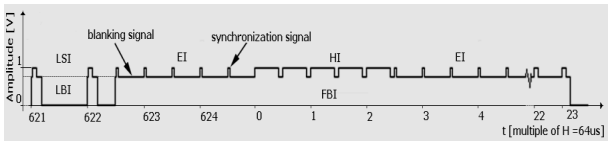

Fig.3 Illustration of blanking and synchronization signals

as colour signals R, G and B, so even synchronization and blanking signals may be generated in FPGA, we can now mark the position of this FPGA in the AEI generator. It is drawn on Fig.4.The FPGA device can be

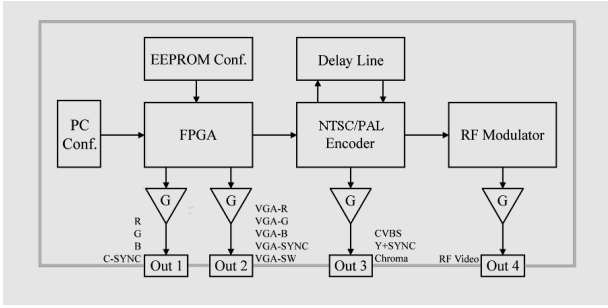

Fig.4 FPGA as a part of whole AEI generator

configured via plugged cable from PC or commonly after device power–up from EEPROM memory, which permanently holds configuration stream. Signals generated in FPGA are then driven out through ports Out 1 and Out 2 (these signals are in base frequency band), or processed in E-NTSC/PAL encoder and RF modulator. The block structure of whole logical design inside of the FPGA itself is shown on Fig.5. We can see therein one part which cares about synchronization and blanking signals, marked as Sync&Blank, and a second part marked as RGB, which takes care about component colour R, G and B signals. This division has a practical reason, because when we would like to create flexible type AEI generator, we would not need to change both these parts, but we could leave Sync&Blank (S&B) part without any change. The design is driven with 8 MHz clock signal from crystal controlled oscillator. External

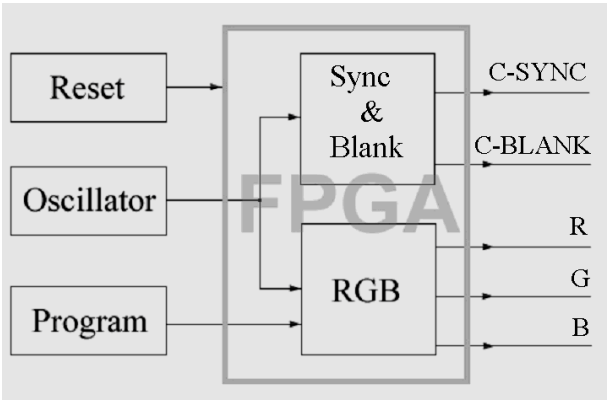

Fig.5 Block structure of logical design in FPGA

buttons Reset and Program allow entry and function configuration of full logical part of AEI generator.

2 GENERATION OF SYNCHRONIZING AND BLANKING SIGNALS

As already mentioned, both these signals, synchronization and blanking signal, are gathered in one logical part. In general for interleaved screening by CCIR – B(G) or D(K) the synthesis of complex blanking and synchronizing generator is joined with generational formula

$$2.\ f_H = N.\ f_v \qquad (1)$$

where

f_H is picture line repeating frequency,

f_V - field repeating frequency,

N - count of picture lines.

Further discussion about synthesis of complex blanking and synchronizing generator is presented e.g. in [1], [5], [7]. In relation to Eq.(1) and Fig.5 the full (composite) blanking signal C-BLANK generation expects from the reference clock signal PCLK generate number of additional signals with suitable output duty cycle and repeating frequency by Fig.6. Interrelationship between continuous horizontal blanking H-BLANK, vertical blanking V-BLANK and composite blanking C-BLANK signals can be defined then by following formula

$$C-BLANK = \overline{H-BLANK} \bullet \overline{V-BLANK} \qquad (2)$$

Likewise in relation to Eq.(1) and Fig.5 the full (composite) synchronizing signal C-SYNC generation expects from the reference clock signal PCLK generate number of another additional signals with suitable output duty cycle and repeating frequency by Fig.7. Interrelationship between continuous horizontal synchronizing H-SYNC, holding UI-CONT, equalizing VI-CONT, vertical synchronizing V-SYNC, vertical 7,5H impulses 7H5 and composite synchronizing C-SYNC signals in this case can be defined by following formula

$$C-SYNC = F_1 + F_2 + F_3 \qquad (3)$$

where auxiliary signals $\mathbf{F_i}$ (i=1,2,3) have next forms

$$F_1 = \overline{H-SYNC + 7H5} \qquad (4)$$

$$F_2 = 7H5 \bullet \overline{V-SYNC} \bullet VI-CONT \qquad (5)$$

$$F_3 = V-SYNC \bullet \overline{UI-CONT} \qquad (6)$$

2.1 Software implementation of S&B generator

Software implementation of any digital circuit, in the future hardware implemented in the FPGA's inside structure, is coupled into possibilities of its development software system, [3], [4], [6]. The modern approach to software implementation of complex blanking and synchronizing generator is usage New Symbol Wizard generator with connection a VHDL editor for its inside structure editing. By this equipments created inside architecture of complex synchronizing and blanking generator, edited in Verilog Hardware Design Language and debugged for Xilinx FPGA silicon XC3020E, is shown in Fig.8. From this architecture one can see in which parallel processes are generated all of additional signals and how in correlation to Eq.(1) ÷ Eq.(6) are

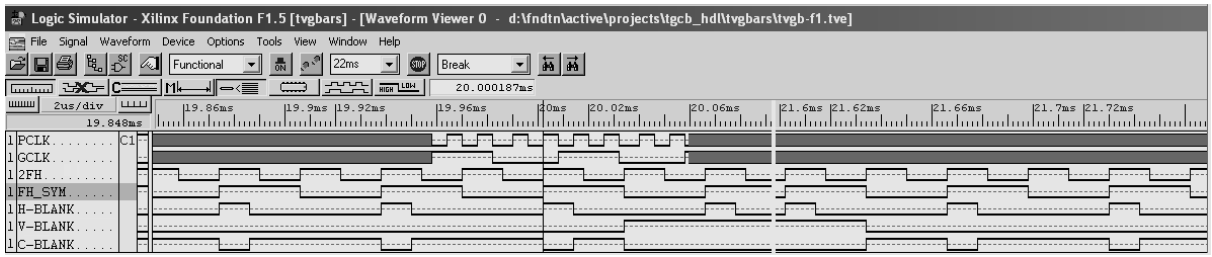

Fig.6 Timing diagram of inside and output C-BLANK signals of "blanking" part of S&B generator

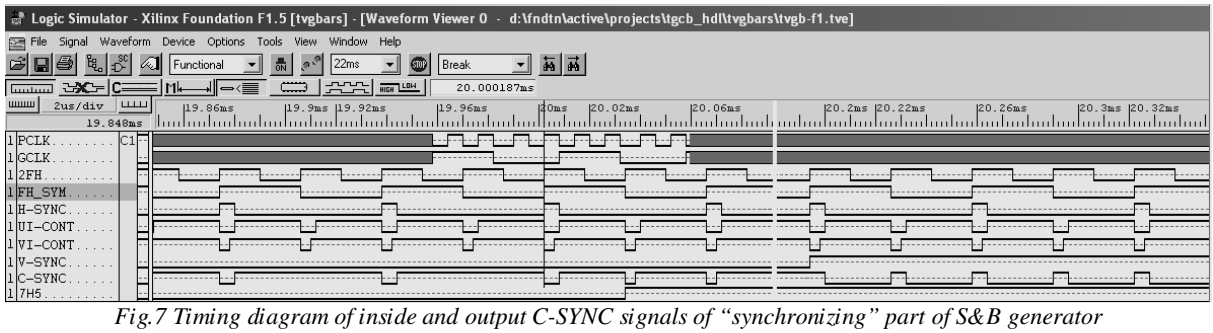

Fig.7 Timing diagram of inside and output C-SYNC signals of "synchronizing" part of S&B generator

```
-- S&B generator , July 2005,  created by SS and PK
library IEEE;
use IEEE.std_logic_1164.all;
library SYNOPSYS;
use SYNOPSYS.attributes.all;
entity TVSYNC_FULL_1 is
   port (  PCLK: in STD_LOGIC;
           V_SYNC: inout STD_LOGIC;
           V_BLANK: inout STD_LOGIC;
           H_SYNC: inout STD_LOGIC;
           H_BLANK: inout STD_LOGIC;
           C_SYNC: inout STD_LOGIC;
           C_BLANK: inout STD_LOGIC;
           GCLK: inout STD_LOGIC;
           UI_cont: inout STD_LOGIC;
           VI_cont: inout STD_LOGIC;
           FH2: inout STD_LOGIC;
           FH_sym: inout STD_LOGIC  );
end TVSYNC_FULL_1;
architecture TVSYNC_FULL_1_arch of TVSYNC_FULL_1 is
signal v_sync_a: STD_LOGIC;   --  pom. sig. na vytvorenie PSI
signal ENP, ENG: STD_LOGIC;
signal countP, countG, width_rsi, width_ui, width_vi: INTEGER
range 0 to 255;       -- counter
signal count_rzi: INTEGER range 0 to 511;       -- counter
signal count_v: INTEGER range 0 to 1023;       -- counter
begin
 make_GCLK: process(PCLK, ENG, GCLK)
  begin
  ENG <= '0';
  if ENG = '0' then
    ENG <= '1';
    GCLK<= '1';
    countG <= 0;
  elsif PCLK'event and PCLK = '0' then
    countG <= countG + 1;
    if countG = 1 then
      GCLK <= not GCLK;
      countG <= 0;
    end if;
  end if;
 end process make_GCLK;
 make_FH2: process(PCLK, ENP)
 begin
  if ENP = '0' then
    countP <= 0;
    ENP <= '1';
    FH2 <= '1';
  elsif PCLK'event and PCLK = '1' then
    countP <= countP + 1;
    if countP = 127 then
      FH2 <= not(FH2);
      countP <= 0;
    end if;
  end if;
 end process make_FH2;
 make_FH: process(FH2, FH_sym)
 begin
  if FH2'event and FH2 = '1' then
    FH_sym  <= not(FH_sym );
  end if;
 end process make_FH;
make_RZI_CONT: process(FH_sym, PCLK)
 begin
  if FH_sym = '0' then
    count_rzi <= 0;
    H_BLANK <= '0';
  elsif PCLK'event and PCLK = '0' then
    H_BLANK <= '1';
    count_rzi <= count_rzi + 1;
    if count_rzi >= 100 then
       H_BLANK <= '0';
    end if;
  end if;
 end process make_RZI_CONT;
```

```
make_RSI_CONT: process(FH_sym, PCLK, H_sync)
begin
 if FH_sym ='0' then
     H_SYNC <= '0';
     width_rsi <= 0;
 elsif  PCLK'event  and PCLK='1' then
    width_rsi <= width_rsi + 1;
    if ((width_rsi >= 12) and (width_rsi <= 48)) then
      H_SYNC <= '1';
    else
       H_SYNC <= '0';
    end if;
 end if;
end process make_RSI_CONT;
make_UI_CONT: process(PCLK, FH2)
begin
 if FH2 = '0' then
    UI_cont <= '1';
    width_ui <= 0;
 elsif PCLK'event and PCLK = '1' then
    width_ui <= width_ui + 1;
    if ((width_ui >= 12) and (width_ui <= 48)) then
       UI_cont <= '0';
    else
       UI_cont <= '1';
    end if;
 end if;
 end process make_UI_CONT;
make_VI_CONT: process(PCLK, FH2)
begin
 if FH2 = '0' then
    VI_cont <= '1';
    width_vi <= 0;
 elsif PCLK'event and PCLK = '1' then
    width_vi <= width_vi + 1;
    if ((width_vi >= 12) and (width_vi <= 30)) then
       VI_cont <= '0';
    else
       VI_cont <= '1';
    end if;
 end if;
end process make_VI_CONT;
make_V_SYNC_supply: process(FH2)
begin
 if FH2'event and FH2 = '1' then
    if count_v = 0 then
      V_BLANK <= '1';
      v_sync_a <= '1';
    end if;
    count_v <= count_v + 1;
    if count_v = 5 then
      V_SYNC <= '1';
    end if;
    if count_v = 10 then
      V_SYNC <= '0';
    end if;
    if count_v = 15 then
      v_sync_a <= '0';
    end if;
    if count_v = 50 then
      V_BLANK <= '0';
    end if;
    if count_v = 624 then
      count_v <= 0;
    end if;
 end if;
end process make_V_SYNC_supply;

 C_SYNC  <= not((not(v_sync_a) and H_SYNC) or
(not(VI_cont) and v_sync_a and not(V_SYNC)) or (UI_cont
and v_sync_a and V_SYNC));

 C_BLANK <= not(V_BLANK or H_BLANK);

end TVSYNC_FULL_1_arch;
```

Fig.8 Full architecture of Sync&Blank generator written in VHDL of software system FND Exp F1.5

created final output signals H-BLANK and C-BLANK.

3 GENERATION OF COMPONENT COLOUR R, G, B SIGNALS

Component colour signals R, G and B generation is maintained by Fig.5 in block marked RGB. These signals are derived, e.g. for alternate number of colour bar octets, from clock signal (8 MHz) using appropriate divider signed on Fig.9 as block U1 (DIV X). For example to create one colour bar octet we must divide clock signal 104 times, because as seen on Fig.2, one colour bar takes 6,5 µs, clock period is 0,125 µs, then 6,5 : 0,125 = 52, therefore we need to divide the clock signal 104 times to obtain pulse with duration 6,5 µs. Thus we create continuous pulse signal with pulse width 6,5 µs. The start of dividing is driven by falling edge of H–BLANK signal, what is horizontal blanking signal from block Sync&Blank, and which means start of new TV line. For creation of one octet of colour bars on TV screen we need to create only 4 pulses with width 6,5 µs (Fig.2). This is maintained in block marked on Fig.9 as U2, which is multi input AND gate and which detects number Y, in this case number 4 (rising edge of fifth pulse), and which resets D flip–flop and other parts of schema. Thus we created needed B signal. Next two needed G and R signals are derived using division by two. Letters X and Y in blocks U1 and U2 mean that we do not need to use only numbers 104 for divider U1 and 4 for detector U2 as described, but when we need to generate other count of colour bar octets, we can use other constants. Table 1 shows these constants for case of generating 1, 2, 3 or 4 octets of vertical colour bars.

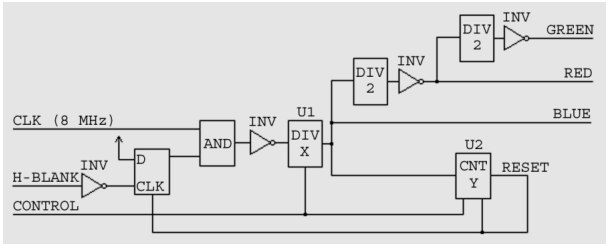

Fig.9 Logical layout of the RGB generator

Tab.1 X,Y constants specification

Count of color bar octets	X – value of U1 block	Y – value of U2 block
1	104	4
2	52	8
3	34	12
4	26	16

3.1 Software implementation of RGB generator

Likewise as in part 2.1, the modern approach to software implementation of RGB generator is usage New Symbol Wizard generator with connection a VHDL editor for its inside structure editing, too. By these equipments created inside architecture of full programmable RGB generator, edited in Verilog Hardware Design Language and debugged for Xilinx FPGA silicon XC3020E, is shown in Fig.10.

```
-- RGB generator, September 2005,  created by SS and PK
library IEEE;
use IEEE.std_logic_1164.all;
library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity TVGVIDEO11 is
   port (
      PCLK: in STD_LOGIC;
      H_BLANK: in STD_LOGIC;
      C_BLANK: in STD_LOGIC;
      V_SYNC: in STD_LOGIC;
      RED: inout STD_LOGIC;
      GREEN: inout STD_LOGIC;
      BLUE: inout STD_LOGIC;
      Q_TDDFF: inout STD_LOGIC;
      Q_PRG: inout STD_LOGIC;
      My_GSR: in STD_LOGIC;
      PRG: in STD_LOGIC
      );
end TVGVIDEO11;

architecture TVGVIDEO11_arch of TVGVIDEO11 is
      component STARTUP
            port (GSR: in std_logic);
      end component;

 signal count_r,CONST_R: integer range 0 to 127;     -- counter
& constant
 signal count_g,CONST_G: integer range 0 to 255;     -- counter
& constant
 signal count_b,CONST_B: integer range 0 to 63;      -- counter
& constant

 signal SEL: STD_LOGIC_VECTOR (1 downto 0);
 signal r,g,b,EN: STD_LOGIC;
 signal count_delay: integer range 0 to 63;

 begin
    U2: STARTUP  port map (GSR=>My_GSR);

flip_flop:process (PRG, Q_TDDFF)
   begin

      if (Q_TDDFF = '1') then
        Q_PRG <= '0';
      elsif (PRG'event and PRG = '1') then
        Q_PRG <= '1';
      end if;
 end process flip_flop;
  delay_increment:process (V_SYNC)
    begin
      if (V_SYNC'event and V_SYNC = '1') then
         count_delay <= count_delay + 1;
          if count_delay = 1 then
            Q_TDDFF <= '0';
          elsif (count_delay = 63) then
            count_delay <=0;
            Q_TDDFF <= '1';
          end if;
      end if;
  end process delay_increment;
prog:process (Q_PRG, EN)
    begin
      if EN = '0' then
        SEL(0) <= '0';
        SEL(1) <= '0';
        EN <= '1';
      elsif Q_PRG'event and Q_PRG='1' then
            SEL(0) <= not SEL(0);
            if (SEL(0)='1') then
            SEL(1) <= not SEL(1);
            end if;
      end if;
 end process prog;
```

Fig.10 Full architecture of RGB generator – 1/2

```
rgb:process (PCLK, H_BLANK, SEL, const_r, const_g,
const_b)
  begin
       if (SEL(0) or SEL(1))='0' then
          CONST_R <= 103;
          CONST_G <= 207;
          CONST_B <= 51;
       elsif (not SEL(0) or SEL(1))='0' then
          CONST_R<=51;
          CONST_G<=103;
          CONST_B<=25;
       elsif (SEL(0) or not SEL(1))='0' then
          CONST_R<=33;
          CONST_G<=67;
          CONST_B<=16;
       else
          CONST_R<=25;
          CONST_G<=51;
          CONST_B<=12;
       end if;

  if H_BLANK='1' then
    count_r <= 0;
    count_g <= 0;
    count_b <= 0;
    r  <= '1';
    g  <= '1';
    b  <= '1';
    elsif (PCLK'event and PCLK = '0') then
       count_r <= count_r +1;
       count_g <= count_g +1;
       count_b <= count_b +1;
            if (count_r = CONST_R) then
                 r <= not(r);
                 count_r <=0;
            end if;
            if (count_g = CONST_G) then
                 g <= not(g);
                 count_g <=0;
            end if;
            if (count_b = CONST_B) then
                b <= not(b);
                count_b <=0;
            end if;
   end if;
 end process rgb;
RGB_set:process (C_BLANK, r, RED, g,GREEN, b, BLUE)
 begin
   RED  <= not(r and C_BLANK);
    Green <= not(g and C_BLANK);
    BLUE  <= not(b and C_BLANK);
 end process  RGB_set;

end TVGVIDEO11_arch;
```

Fig.10 Full architecture of RGB generator – 2/2

4 FULL ANALOGUE ELECTRONIC IMAGES GENERATOR HARDWARE IMPLEMANTATION

In Fig.11 is shown the whole logic design, implemented in FPGA Xilinx silicon - XC4003E, where both Sync&Blank generator and RGB generator parts can be seen as two bigger boxes marked TVGVIDEO11 and TVSYNC_FULL_1. For hardware implementation needed configuration bit-stream for this device has 53 984 bits and determines needed capacity of configurable EEPROM by Fig.4 (MC28C64). Next Fig.12 shows inside structure of silicon - XC4003E after programming procedure in Epic Design Viewer/Editor of its development software system. At last in Fig.13 is shown the complete produced flexible analogue electronic images generator, where FPGA Xilinx can be seen in right top corner, [2].

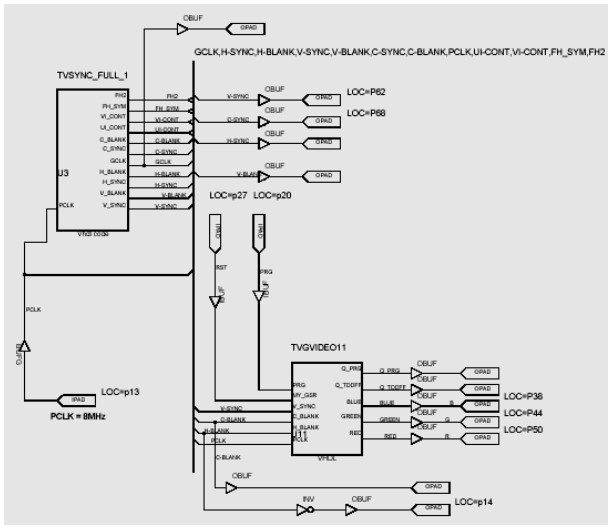

Fig.11 Whole logic design of Sync&Blank and RGB generator parts in schematic editor

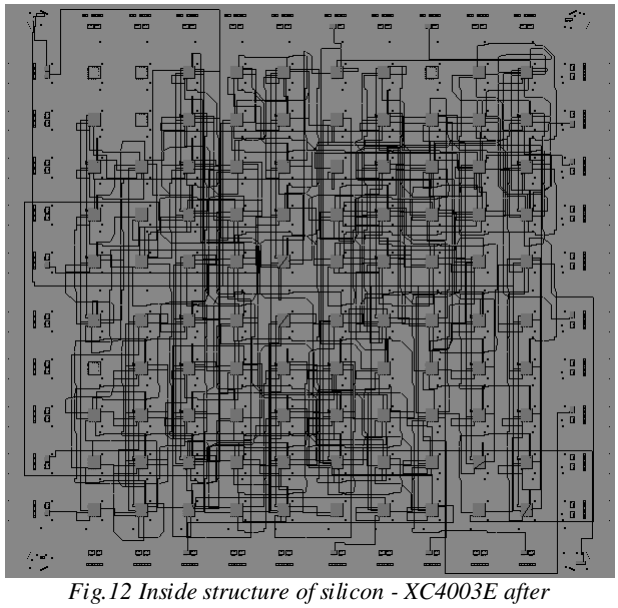

Fig.12 Inside structure of silicon - XC4003E after programming procedure in Epic Design Viewer/Editor

5 CONCLUSION

In contribution described and discussed main questions relative to complex design of flexible analogue electronic images generator show on possibilities how simply use FPGAs in design of its logical parts (see Fig.5). Obtained results suggest on their full exploitation within education of subject of Analogue and Digital Television in Master's program of Radio and Electronics or/and in TV measurement technique or/and TV service technique.

REFERENCES

[1] Ševčík,P., Kulla,P.: Televízna technika (príručka na cvičenia). Skriptum EF STU , ES STU, Bratislava, 2001,359 str., 4. vydanie

[2] Slávik,Š.: Generátor testovacích obrazov pre analógový systém FTV-PAL. Diplomová práca, KRE FEI STU, Bratislava 2005, 55 str.

[3] http://direct.xilinx.com/bvdocs/publications/4000.pdf

[4] http://direct.xilinx.com/bvdocs/publications/ds006.pdf

Fig.13 Complete produced PAL flexible analogue electronic images generator

[5] Kulla,P.: Studio Television Circuits and Equipment. Lectures and seminaries, Study text, Dept. of Radio & Electronics, Bratislava, 1997-2005, (in slovak)
[6] XILINX: The Programmable Logic Data Book, XILINX 1993
[7] Philips: Book IC02b, Philips 1992

Acknowledgement
This contribution is supported by the Slovakia Ministry of Education under VEGA Grant No. G-1/0144/03 and VTP Project No. 102/VTP/2000.