Sterownik VGA w układzie FPGA

Zastosowanie monitora VGA w aplikacji zrealizowanej w układzie FPGA nie jest trudne. Przekonamy się o tym po lekturze artykułu, w którym zaprezentowano projekt układu, opisanego w językach VHDL i Verilog, umożliwiającego wyświetlenie jakiegoś obiektu na ekranie monitora. Dobry wzorzec dla uczących się VHDL'a czy Verilog'a.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 10460, pass: 0646g3n0
• listingi do artukułu

Jako przykład implementacji w układach programowalnych prostego sterownika VGA przedstawiamy projekt układu, którego zadaniem jest wyświetlanie na ekranie monitora poruszającego się i odbijającego od czterech krawędzi ekranu prostego obiektu zwanego dalej kulką. Kulka w rzeczywistości jest kwadratem o boku o zadanej liczbie pikseli. Kulka podczas poruszania się po ekranie dodatkowo zmienia cyklicznie swój rozmiar, sprawiając wrażenie pulsowania i przy każdym odbiciu od górnej krawędzi ekranu zmienia również swój kolor na jeden z trzech kolorów podstawowych (czerwony, niebieski, zielony). Obraz wyświetlany jest w standardowej rozdzielczości VGA (640×480) z częstotliwością odświeżania równą 60 Hz.

W projekcie wykorzystano fragmenty kodów źródłowych w języku VHDL analogicznego projektu autorstwa Douga Hodsona, opublikowanego na stronie *www.retromicro.com*. Układ opisany w udostępnionym tam projekcie umożliwia wyświetlanie obrazu kulki o stałej wielkości, poruszającej się po ekranie jedynie w osi pionowej.

Opisywany w artykule układ składa się z dwóch modułów (komponentów): generatora sygnałów synchronizacji oraz modułu odpowiedzialnego za animację ruchu kulki na ekranie. W opisie projektu w języku VHDL występuje jeszcze trzeci nadrzędny moduł, który definiuje jedynie strukturę połączenia wspomnianych wcześniej komponentów.

Generator sygnałów synchronizacji

Zadaniem tego modułu jest wytworzenie sygnałów synchronizacji pionowej i poziomej oraz udostępnienie bieżących zawartości liczników określających współrzędne położenia wybieranego w danej chwili punktu (plamki) na ekranie monitora. Sposób działania tego modułu jest zdeterminowany strukturą ramki w standardzie VGA. Ramka sygnału VGA była już wielokrotnie opisywana na łamach Elektroniki Praktycznej (np. w EP 4/2007 przy okazji projektu testera monitorów VGA), dlatego nie będzie omawiana.

Na **listingu 1** przedstawiono kod w języku VHDL opisujący generator sygnałów synchronizacji. Oprócz wyjść sygnałów synchronizacji poziomej (*horiz_sync_out*) i pionowej (*vert_sync_out*) oraz stanu licznika kolumn – punktów w linii (*pixel_column*) i licznika wierszy – linii (*pixel_row*), moduł ma również wejścia sygnałów kolorów podstawowych (*red*, *green*, *blue*) oraz wyjścia tych sygnałów (*red_out*, *green_out*, *blue_out*). Na wyjściach sygnałów kolorów podstawowych pojawia się ten sam sygnał co na analogicznych wejściach wtedy, gdy liczniki kolumn i wierszy wskazują na aktywny obszar ekranu (sygnały *video_on_h* oraz *video_on_v* mają poziom wysoki). W czasie trwania przednich i tylnych przedziałów wyrównawczych, jak również podczas trwania samych impulsów synchronizacji, wyjścia kolorów podstawowych są wyzerowane (obraz jest wygaszany).

Moduł animacji ruchu kulki

Moduł jest głównym elementem projektu realizującym wszystkie efekty związane z ruchem kulki na ekranie monitora. Do modułu musi być dostarczony z generatora sygnałów synchronizacji impuls synchronizacji pionowej oraz stanu liczników wierszy i kolumn. Z kolei wyjścia kolorów podstawowych *red*, *green*, *blue* tego modułu powinny być połączone z analogicznymi wejściami modułu generatora sygnałów synchronizacji. Kod opisujący moduł animacji ruchu kulki przestawiono na **listingu 2**.

W procesie nazwanym *RGB_Dsiplay*, jest wyliczana wartość sygnału *Ball_on*, którego poziom wysoki, przy danych wartościach liczników kolumn i wierszy oznacza, że aktualnie wybierany (wyświetlany na ekranie) punkt nie jest punktem tła lecz należy do obszaru zajmowanego przez kulkę. Sygnał ten jest wyliczany na podstawie koniunkcji stanów odpowiednich relacji (narzędzia syntezy wywnioskują z tego opisu zespół komparatorów) uwzględniających bieżące położenie kulki (środka kwadratu – współrzędne *Ball_X_pos* oraz *Ball_Y_pos*), rozmiar kulki (bok kwadratu - sygnał *Size*) oraz stan liczników wierszy i kolumn (współrzędne aktualnie wybieranego punktu).

Kolejny proces (*Move_Ball*) aktywowany podczas każdego narastającego zbocza sygnału synchronizacji pionowej (*vert_sync_out*) odpowiada za sterowanie ruchem kulki. W kolejnych instrukcjach warunkowych sprawdzane są współrzędne położenia środka kulki – oddzielnie współrzędna pozioma i pionowa oraz uwzględniany jest rozmiar kulki. Jeżeli kulka osiągnęła którąś z krawędzi ekranu nadawana jest odpowiednia wartość zmiennym *Ball_X_motion* oraz *Ball_Y_motion*. Wartości tych sygnałów następnie służą do wyliczenia następnego (podczas trwania następnej ramki obrazu) położenia kulki – następnych pionowych i poziomych współrzędnych położenia środka kulki. Na przykład, jeżeli kulka poruszając się w górę ekranu znalazła się na jego górnej krawędzi (czyli współrzędna pionowa położenia kulki *Ball_Y_pos* jest równa rozmiarowi kuli *Size* – lewy górny róg ekranu ma współrzędne [0,0]), wówczas sygnałowi *Ball_Y_motion* nadawana jest wartość pewnej stałej, która definiuje szybkość poruszania się kulki po ekranie (a dokładniej szybkość zmian położenia w osi pionowej). W przedstawionym kodzie na list. 2, wartość tej stałej otrzymywana jest poprzez wywołanie funkcji *conv_std_logic_vector* (w języku VHDL wielkość liter jest nieistotna), która jedynie dokonuje konwersji liczby całkowitej (pierwszy argument funkcji) o zadanej liczbie bitów (drugi argument funkcji) do typu standardowego *std_logic_vector*. Zamiast wywoływania tej funkcji można również wpisać stałą wartość w kodzie dziesiętnym. W przypadku, gdy kulka osiągnęła dolną krawędź ekranu, zmiennej *Ball_Y_motion* nadawana jest wartość tej

Listing 1. Kod opisujący moduł generatora sygnałów synchronizacji

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity VGA_SYNC is
 port(clock_25Mhz, red, green, blue        : in     std_logic;
    red_out, green_out, blue_out, horiz_sync_out, vert_sync_out: out std_logic;
    pixel_row, pixel_column: out std_logic_vector(9 downto 0));
end VGA_SYNC;

architecture sync_gen of VGA_SYNC is
        signal horiz_sync, vert_sync : std_logic;
        signal video_on, video_on_v, video_on_h : std_logic;
        signal h_count, v_count :std_logic_vector(9 downto 0);

begin

-- video_on przyjmuje poziom wysoki, gdy ma być wyświetlony obraz
video_on <= video_on_H and video_on_V;

pixel_row <= v_count; pixel_column <= h_count;

process
begin
        wait until (clock_25Mhz'EVENT) and (clock_25Mhz='1');

-- h_count zlicza piksele poziomo (640 + dodatkowe dla sygnału synchronizazji)
        if h_count = 799 then h_count <= „0000000000";
        else h_count <= h_count + 1; end if;

-- v_count zlicza wiersze pikseli (480 + dodatkowe dla sygnału synchronizacji)
        if (v_count >= 524) and (h_count >= 699) then v_count <= „0000000000";
        elsif h_count = 699 then v_count <= v_count + 1; end if;

-- Synchronizacja zmian wybranych sygnałów z sygnałem zegarowym
                red_out <= red and video_on;
                green_out <= green and video_on;
                blue_out <= blue and video_on;
                horiz_sync_out <= horiz_sync;
                vert_sync_out <= vert_sync;

end process;

process (h_count)
begin
-- Generowanie sygnału synchronizacji poziomej z użyciem h_count
--  horiz_sync  ------------------------------------_____--------
--  h_count     0                                  640   659      755    799

        if (h_count <= 755) and (h_count >= 659) then horiz_sync <= ,0';
        else     horiz_sync <= ,1'; end if;

  -- Generowanie sygnału określającego obszar aktywny ekranu (H)
        if (h_count <= 639) then video_on_h <= ,1';
        else video_on_h <= ,0';   end if;
end process;

process (v_count)
begin
-- Generowanie sygnału synchronizacji pionowej z użyciem v_count
--  vert_sync  ----------------------------------------_____------------
--  v_count    0                                    480   493-494          524

  if (v_count <= 494) and (v_count >= 493) then vert_sync <= ,0';
  else vert_sync <= ,1'; end if;

  -- Generowanie sygnału określającego obszar aktywny ekranu (V)
        if (v_count <= 479) then video_on_v <= ,1';
        else video_on_v <= ,0';   end if;
end process;
end sync_gen;
```

samej stałej co poprzednio lecz z zanegowanym każdym z jej bitów. Nowa składowa pionowa współrzędnej położenia kulki obliczana jest poprzez zsumowanie jej bieżącej wartości z wartością *Ball_Y_motion*. Ponieważ zgodnie z arytmetyką dwójkową dla n-bitowych liczb A i B, A – B = A + B' +1 („prim" oznacza negację), gdy kulka osiągnęła dolną krawędź ekranu (*Ball_Y_pos + Size = 479*), to od jej pionowej współrzędnej jest odejmowana stała wartość (+1 można tutaj zaniedbać) i kulka porusza się w górę ekranu. Analogicznie oblicza się położenie współrzędnej poziomej kulki.

Dodatkowo, wraz z instrukcjami warunkowymi związanymi ze sprawdzaniem pionowego położenia kulki, zaimplementowano rejestr przesuwający, który przesuwa o jeden bit w lewo trzybitowy sygnał *RGB* podczas każdego odbicia kulki od górnej krawędzi ekranu. Sygnał ten odpowiada za kolor wyświetlanej kulki (czerwony, zielony i niebieski). Jest to uwarunkowane zastosowaniem dla sygnału RGB rejestru przesuwającego pracującego w trybie licznika pierścieniowego („krążąca jedynka"). Takie użycie rejestru przesuwającego wymaga, aby po włączeniu zasilania (załadowaniu pliku konfigurującego do układu FPGA) w rejestrze została ustawiona jedynka na wybranej pozycji bitu. Domyślnie po załadowaniu pliku konfigurującego wszystkie rejestry (przerzutniki) są zerowane. W przypadku wykorzystania środowiska projektowego *Xilinx ISE* i narzędzia syntezy XST, w celu nadania wartości początkowej dla wybranych przerzutników, można posłużyć się atrybutem *INIT* – tak jak pokazano to na list. 2 (tuż przed instrukcją *begin* opisu architektury).

Ostatnia sekcja kodu w procesie *Move_Ball* odpowiada za modulację rozmiaru wyświetlanego obiektu. Rozmiar kulki oscyluje pomiędzy dwiema wartościami granicznymi (2 i 24 piksele – stałe, argumenty funkcji *conv_std_logic_vector*) i zmienia się o jeden punkt wraz z każdym narastającym zboczem impulsu synchronizacji pionowej.

Implementacja

Układ projektu można zrealizować wykorzystując np. zestaw ewaluacyjny ZL9PLD wraz z modułem dipPLD ZL10PLD z układem XC3S200. Ponieważ w zestawie dostępny jest generator sygnału zegarowego o częstotliwości 3,6864 MHz, a generator impulsów synchronizacji wymaga dostarczenia sygnału zegarowego o częstotliwości ok. 25 MHz (dokładnie 25,125 MHz – ale częstotliwość ta nie jest krytyczna, niewielkie odchyłki od tej częstotliwości mogą spowodować jedynie pewne zmiany rozmiarów obrazu na ekranie monitora i ułamkowe zmiany częstotliwości odświeżania), dlatego istnieje potrzeba odpowiedniego powielenia częstotliwości generatora z modułu dipPLD, albo wymiany tego generatora. W pierwszym przypadku z pomocą przychodzi blok syntezera częstotliwości (DCM) wbudowany w układ XC3S200. Syntezer można zaprogramować tak, aby jego częstotliwość wyjściowa była iloczynem częstotliwości wejściowej i pewnej liczby wymiernej, której wielkość określana jest poprzez nadanie wartości odpowiednim atrybutom. Taki sposób zastosowano w omawianym projekcie. Plik o nazwie *chipIO.vhd*, którego fragment pokazano na **list. 3** zawiera również, oprócz definicji struktury połączenia modułu generatora impulsów synchronizacji i modułu animacji ruchu kulki, opis implementacji syntezera DCM wraz definicją odpowiednich atrybutów. Dzięki temu uzyskuje się częstotliwość sygnału zegarowego bardzo bliską właściwej. Dokładnie jest to 24,88 MHz – ze względu na ograniczenia zakresu odpowiednich współczynników, przy danej częstotliwości wejściowej, nie jest możliwe uzyskanie częstotliwości wyjściowej bliższej nominalnej 25,125 MHz.

Jak to zrobić w Verilogu

Dla tych, którzy preferują język opisu sprzętu Verilog (do grona tych osób zalicza się również autor niniejszego artykułu), podajemy także wersję projektu opisaną w tym właśnie języku.

Verilog jest językiem opisu sprzętu, który powstał nieco później niż VHDL. Począt-

kowo służył jedynie do symulacji układów cyfrowych i dopiero później został zaadaptowany do celów syntezy logicznej. Składnia języka Verilog oparta jest na szeroko rozpowszechnionym języku C, podczas gdy składnia VHDL bazuje na obecnie rzadko używanym języku ADA. Język VHDL ma zaawansowane konstrukcje i abstrakcyjne typy danych, które nie występują w Verilogu. Dzięki czemu VHDL znacznie bardziej nadaje się do modelowania behawioralnego. Jednak tylko stosunkowo niewielki podzbiór instrukcji VHDL akceptowany jest przez narzędzia syntezy. Język Verilog z kolei uważany jest za znacznie bliższy sprzętowi niż VHDL. Przyglądając się kodom w języku VHDL i Verilog opisującym ten sam projekt, można odnieść wrażenie, że zazwyczaj kod w języku Verilog jest krótszy, bardziej zwarty i przejrzystszy w porównaniu do kodu w języku VHDL.

Na **listingach 4...6** (do pobrania z serwera FTP) przedstawiono kolejno opisy w języku Verilog modułu generatora impulsów synchronizacji, modułu animacji ruchu kulki oraz modułu nadrzędnego, definiującego połączenia dwóch pierwszych wymienionych modułów.

Podstawową jednostką projektową w języku Verilog jest moduł (*module*), który zawiera zarówno opis interfejsu jak również opis wnętrza modułu (jego funkcji logicznych). Nie ma tu podziału tak jak w języku VHDL, na jednostkę projektową – interfejs (*entity*) oraz architekturę (*architecture*). W Verilogu, podobnie jak w języku C, rozróżnine są duże i małe litery alfabetu.

W Verilogu występują dwa podstawowe typy danych: *wire* (i pochodne typu *wand, wor, tri*) oraz *reg*. Typ danych *wire* nie przechowuje swojej wartości (tak jak zmienna) lecz reprezentuje fizyczne połączenia w modelowanym układzie i zwykle stosowany jest w kontekście opisu strukturalnego. Danej typu *wire* można przypisać wynik tzw. przypisania ciągłego (*continuous assignment* – instrukcja *assign*) lub sygnał wyjściowy innego modułu lub bramki logicznej. Typ *reg* reprezentuje zmienne w języku Verilog.

W opisie układów kombinacyjnych w przedstawionym projekcie zastosowano dwie zasadnicze techniki kodowania: wykorzystanie przypisania do sygnału (*signal assignment*) w VHDL (np.: *pixel_row <= v_coint;*) i odpowiadającemu mu przypisania ciągłe (*continuous assignment*) w Verilogu (np.: *assign pixel_row = v_count;*) oraz wykorzystanie sekcji proceduralnej *always* w Verilogu i instrukcji *process* w VHDL, wraz z odpowiednio sformułowanymi listami wrażliwości (*sensitivity list*). Lista wrażliwości bloku *always* (Verilog) i *process* (VHDL) musi zawierać nazwy wszystkich sygnałów, które są wejściami opisywanego układu kombinacyjnego (wyjście układu kombinacyjnego jest funkcją bieżącego stanu wejść tego układu). Dla symulatora kod zawarty w bloku *always* oraz instrukcji *process* wykonywany jest wówczas, gdy nastąpi zmiana wartości sygnału znajdującego się na liście wrażliwości. Drugim istotnym warunkiem opisu układu kombinacyjnego jest to, aby tak organizować przypisania wartości do sygnału reprezentującego wyjście układu kombinacyjnego, aby w każdej ścieżce procesu (bloku *always*) wartość tego wyjścia była określona (w układach kombinacyjnych nic nie może być pamiętane z poprzedniego stanu). Niespełnienie tego warunku spowoduje, że narzędzia syntezy wywnioskują z takiego opisu, dla danego wyjścia, przerzutnik typu zatrzask.

Przykładem drugiego z wymienionych sposobów opisu układu kombinacyjnego w omawianym projekcie jest fragment kodu w module animacji ruchu kulki, opisujący sygnał *Ball_on* (proces *RGB_Display* w opisie VHDL – list. 2 i analogiczny blok *always* w opisie Verilog – list. 5). Należy zwrócić uwagę na znaczenie poszczególnych operatorów w języku Verilog. Postać operatorów jak i ich znaczenie są tutaj niemal identyczne jak w języku C. W języku VHDL operator „&” jest operatorem sklejania (konkatenacji – ten sam operator w Verilogu ma postać „{...}”). W Verilogu „&”, podobnie jak w języku C, oznacza operator iloczynu bitowego. Użycie operatora sklejania w kontekście takim jak w procesie *RGB_Display* oznacza „poszerzenie” sygnału (wektora) *Ball_X_pos* do 11 bitów („doklejany” bit MSB jest wyzerowany). Taki zabieg nie jest konieczny, jeżeli zapew-

Listing 2. Kod opisujący moduł animacji ruchu kulki

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ball is
 port(red,green,blue: out std_logic;
      vert_sync_out: in std_logic;
      pixel_row, pixel_column: in std_logic_vector(9 downto 0));
end ball;

architecture behavior of ball is
        signal dir,shift_en, Ball_on     : std_logic;
        signal Ball_Y_motion             : std_logic_vector(9 downto 0);
        signal Ball_X_motion, Size       : std_logic_vector(9 downto 0);
        signal Ball_Y_pos, Ball_X_pos    : std_logic_vector(9 downto 0);
        signal RGB                       : std_logic_vector(2 downto 0);
   attribute INIT : string;
   attribute INIT of RGB : signal is „001”;
begin
        Red   <= RGB(0) when Ball_on = ,1’ else ,1’;
        Green <= RGB(1) when Ball_on = ,1’ else ,1’;
        Blue  <= RGB(2) when Ball_on = ,1’ else ,1’;

    -- Ball_on przyjmuje stan wysoki, gdy ma być wyświetlona “kulka”
        RGB_Display:
        process (Ball_X_pos, Ball_Y_pos, pixel_column, pixel_row, Size)
        begin
                if (,0’ & Ball_X_pos <= pixel_column + Size) and
                (Ball_X_pos + Size >= ,0’ & pixel_column) and
                (,0’ & Ball_Y_pos <= pixel_row + Size) and
                (Ball_Y_pos + Size >= ,0’ & pixel_row ) then Ball_on <= ,1’;
        else    Ball_on <= ,0’;  end if;
        end process RGB_Display;

        Move_Ball:
        process
        begin
      -- Realizuje przesunięcie kulki raz na każdy przedział sychr. V
        wait until vert_sync_out’event and vert_sync_out = ,1’;

        -- odbicie od góry i dołu ekranu
        if (Ball_Y_pos + Size) >= CONV_STD_LOGIC_VECTOR(479,10) then
          Ball_Y_motion <= not CONV_STD_LOGIC_VECTOR(2,10); shift_en <= ,0’;
        elsif Ball_Y_pos <= Size  then
          Ball_Y_motion <= CONV_STD_LOGIC_VECTOR(2,10);
          if shift_en = ‘0’ then -- realizacja rejestru przesuwnego
          RGB <= RGB(1 downto 0) & RGB(2); shift_en <= ,1’; end if;
        end if;

      -- wyliczenie następnego pionowego położenia “kulki”
        Ball_Y_pos <= Ball_Y_pos + Ball_Y_motion;

        -- odbicie od lewej i prawej krawędzi ekranu
        if (Ball_X_pos + Size) >= CONV_STD_LOGIC_VECTOR(639,10) then
          Ball_X_motion <= not CONV_STD_LOGIC_VECTOR(2,10);
        elsif Ball_X_pos <= Size then
            Ball_X_motion <= CONV_STD_LOGIC_VECTOR(2,10);
        end if;

      -- wyliczenie następnego poziomego położenia “kulki”
        Ball_X_pos <= Ball_X_pos + Ball_X_motion;

          -- zmiana rozmiaru “kulki”
          if Size >= CONV_STD_LOGIC_VECTOR(24,10) then dir <= ,0’; end if;
          if Size <= CONV_STD_LOGIC_VECTOR(2,10) then dir <= ,1’; end if;
          if dir = ,1’ then Size <= Size + 1;
          else Size <= Size - 1;   end if;

         end process Move_Ball;
end behavior;
```

Listing 3. Kod opisujący moduł nadrzędny projektu

```
entity chipIO is
   port(
      pin_sysclk  : in std_logic;
      pin_vga_red     : out std_logic;
      pin_vga_green   : out std_logic;
      pin_vga_blue    : out std_logic;
      pin_vga_hsync_n : out std_logic;
      pin_vga_vsync_n : out std_logic
      );
end chipIO;
architecture arch of chipIO is
  signal sysClk   : std_logic;

  component VGA_SYNC
        port( clock_25Mhz : in std_logic;
    red, green, blue : in std_logic;
    red_out, green_out, blue_out : out std_logic;
    horiz_sync_out, vert_sync_out : out std_logic;
    pixel_row, pixel_column : out std_logic_vector(9 downto 0));
  end component;

  signal vga_pixel_row     : std_logic_vector(9 downto 0);
  signal vga_pixel_column  : std_logic_vector(9 downto 0);
  signal vga_vert_sync_out : std_logic;

  component ball
    port(   signal red,green,blue : OUT std_logic;
    signal vert_sync_out : in std_logic;
    signal pixel_row, pixel_column : in std_logic_vector(9 downto 0));
  end component;

  component DCM
     port(CLKFX: out std_logic;
                CLKIN: in std_logic);
  end component;

   attribute CLKFX_MULTIPLY: integer;
  attribute CLKFX_DIVIDE: integer;
  attribute CLK_FEEDBACK: string;
  attribute CLKIN_PERIOD: integer;
  attribute CLK_FEEDBACK of VGA_clock: label is „NONE”;
  attribute CLKFX_MULTIPLY of VGA_clock: label is 27;
  attribute CLKFX_DIVIDE of VGA_clock: label is    4;
  attribute CLKIN_PERIOD of VGA_clock: label is    275;

  signal ball_red   : std_logic;
  signal ball_green : std_logic;
  signal ball_blue  : std_logic;

begin
  vgadriver:
    VGA_SYNC
    port map(
    clock_25Mhz    => sysClk,
    red            => ball_red,
    green          => ball_green,
    blue           => ball_blue,
    red_out        => pin_vga_red,
    green_out      => pin_vga_green,
    blue_out       => pin_vga_blue,
    horiz_sync_out => pin_vga_hsync_n,
    vert_sync_out  => vga_vert_sync_out,
    pixel_row      => vga_pixel_row,
    pixel_column   => vga_pixel_column
    );

  pin_vga_vsync_n <= vga_vert_sync_out;

  balldriver:
   ball
    port map(red => ball_red,
                  green => ball_green,
             blue => ball_blue,
    vert_sync_out => vga_vert_sync_out,
    pixel_row => vga_pixel_row,
    pixel_column => vga_pixel_column);

  -- pin_sysclk=3.6864MHz, sysClk=24.88MHz
  VGA_clock : DCM
    port map(CLKIN => pin_sysclk,CLKFX => sysClk);
end arch;
```

nimy, że prawa strona operatora relacji nie przekroczy rozmiaru 10 bitów. W omawianym projekcie tak jest (maksymalna wartość *pixel_column + Size* to 824 – liczba, którą można bez problemu zapisać na 10 bitach), dlatego też w kodzie w języku Verilog, w tym przypadku zrezygnowano z użycia operatora sklejania.

Instrukcja *process* oraz blok *always* może służyć również do opisu układów sekwencyjnych. Rozważmy fragment kodu modułu generatora impulsów synchronizacji opisujący układ sekwencyjny (VHDL – list. 1, Verilog – list. 4). W języku VHDL mamy instrukcję *process* z pustą listą wrażliwości, ale na początku znajduje się instrukcja *wait*. Dalsze instrukcje procesu są wykonywane tylko wówczas, gdy warunek po instrukcji *wait* będzie prawdziwy – w tym przypadku, gdy wystąpi narastające zbocze sygnału zegara (*clock_25MHz*). Alternatywny opis w VHDL, dający w wyniku syntezy taki sam układ, składałby się z instrukcji *process* z listą wrażliwości zawierającą tylko sygnał zegarowy (ogólnie: niezależnie od języka HDL, dla opisu układów sekwencyjnych lista wrażliwości może zawierać jedynie sygnał zegarowy, zerujący lub ustawiający) i pierwszą instrukcją procesu w postaci: *if (clock_25Mhz'event) and (clock_25Mhz = '1') then*. W języku Verilog również można modelować układy sekwencyjne wykorzystując blok *always* z pustą listą wrażliwości. Wówczas wewnątrz bloku *always* stosuje się instrukcję (jedną lub więcej – kod pomiędzy tymi instrukcjami opisuje pojedynczy stan automatu) w postaci: *@(posedge clk*), której skutkiem jest zwieszenie wykonywania instrukcji bloku do momentu wystąpienia narastającego zbocza sygnału *clk* (podobnie jak *wait until (clk'event) and (clk = '1')* w VHDL). Taki sposób opisu w Verilogu określa się jako projektowanie z dokładnością cyklu (*cycle accurate design*). Jednak ten sposób nie jest wspierany przez niektóre narzędzia syntezy, w tym również przez XST *Xilinxa*. Dlatego też w projekcie zastosowano klasyczny sposób opisu układu sekwencyjnego z blokiem *always* w postaci: *always @(posedge clock_25Mhz) begin ... end*. W takim przypadku instrukcje bloku wykonywane są za każdym razem, gdy wystąpi narastające zbocze sygnału *clock_25Mhz*.

W języku Verilog warto również zwrócić uwagę na dwa rodzaje przypisań proceduralnych: blokujące (operator „=”) oraz nieblokujące (operator „<=”). Analogiczne typy przypisań nie występują w VHDL, a początkującym użytkownikom Veriloga sprawiają pewne kłopoty. W dużym skrócie przypisanie blokujące („=”) możemy traktować jako przypisanie natychmiastowe. Wartość zmiennej, której przypisano wartość z użyciem operatora „=”, jest znana już w następnej linijce kodu (następnej instrukcji). Przypisanie nieblokujące z operatorem „<=” używane jest w kontekście zdarzeń związanych ze zboczem narastającym lub opadającym wybranego sygnału (zazwyczaj zegara lub sygnału zerującego) i odnosi skutek dopiero wówczas gdy to zdarzenie wystąpi.

Podsumowanie

Tworzenie aplikacji z układami FPGA, w których do wizualizacji danych wykorzystuje się monitor VGA, jest stosunkowo nieskomplikowane. Omawiany projekt niewielkim nakładem pracy można uatrakcyjnić budując prostą grę zręcznościową na wzór znanej gry komputerowej „Arkanoid”.

Na przykładzie projektu pokazano również możliwość użycia dwóch języków opisu sprzętu: VHDL i Verilog. Przejście od opisu układu w jednym języku do opisu w drugim – w typowych przypadkach – nie jest skomplikowane, ale wymaga jednak pewnej wiedzy.

Zbigniew Hajduk
zhajduk@prz-rzeszow.pl