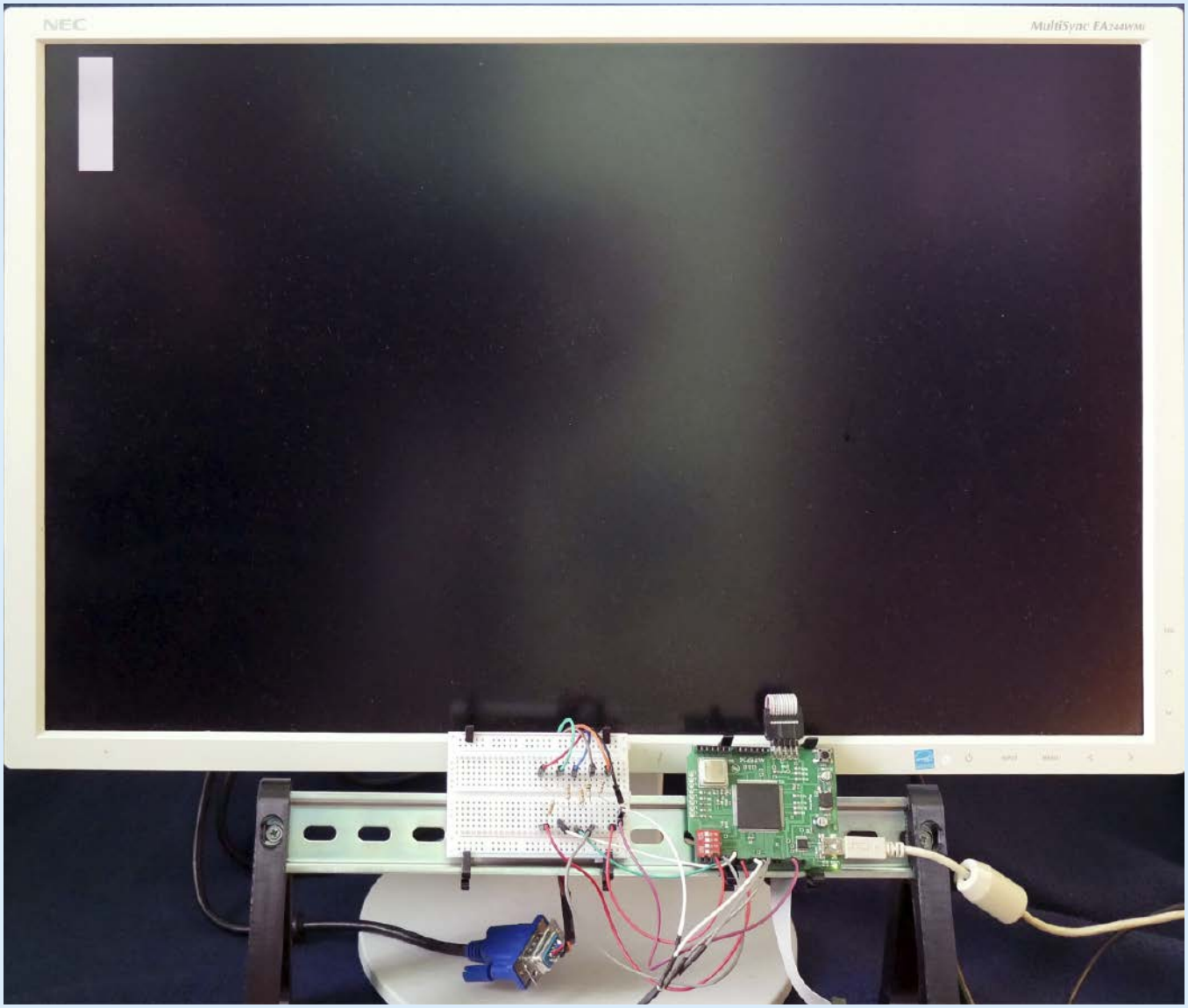


Eksperymenty z FPGA (19)

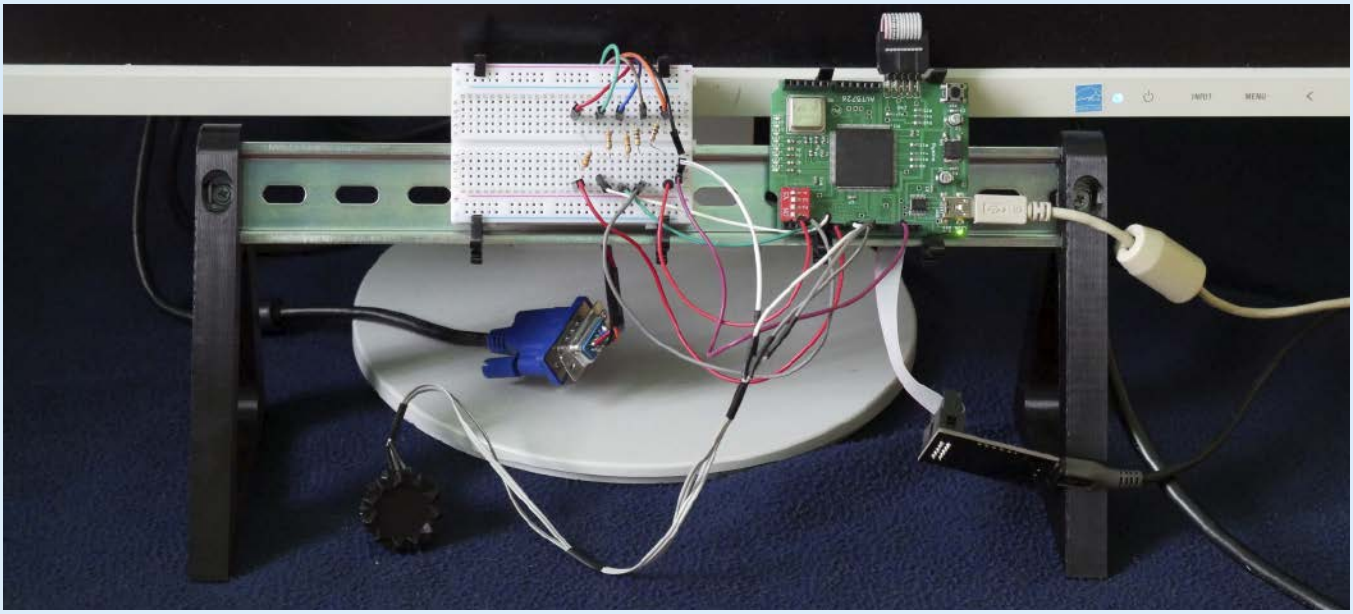
Coś dla graczy

Umiemy już obsługiwać interfejs VGA. Teraz spróbujemy użyć go do wyświetlenia gry PONG. Tak jak poprzednio przed przystąpieniem do wykonywania eksperymentów zachęcam do aktualizacji repozytorium z przykładami [1]. Można to szybko wykonać poprzez wywołanie polecenia `git pull`.

Niemal każdy spotkał się z grą Pong. Istnieje ona od niemal 50 lat. Jest to prosta symulacja tenisa stołowego. Poruszając paletką, staramy się odbić piłkę z nadzieją, że przeciwnikowi nie uda się do niej dotrzeć na czas.

Aby umożliwić sterowanie, potrzebujemy kontrolera dla gracza. Użyjemy w tym celu enkodera inkrementalnego z przyciskiem. Sposób jego podłączenia do płytki Rysino został pokazany na **rysunku 1**. Poza tym użyjemy także portu VGA, ale tę część już znamy z poprzednich odcinków.

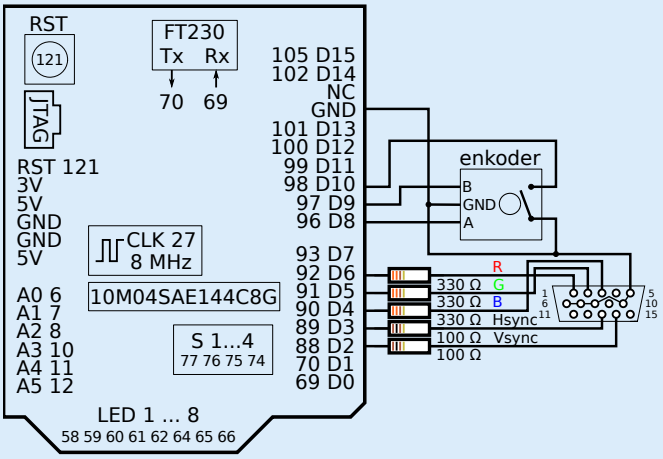
Połączone wszystkie elementy naszej „konsoli do gier" prezentuje **fotografia 1**. Enkodera nie dało się pewnie zamocować w płytce stykowej, więc dla podniesienia komfortu gry warto dolutować do niego przewody. Płytki zostały zamocowane na szynie DIN (TH35) za pomocą wydrukowanych zaczepów. Dzięki temu całość jest bardziej stabilna i można ją łatwiej przenosić z miejsca na miejsce. Mamy gotowy sprzęt, więc musimy przygotować oprogramowanie.



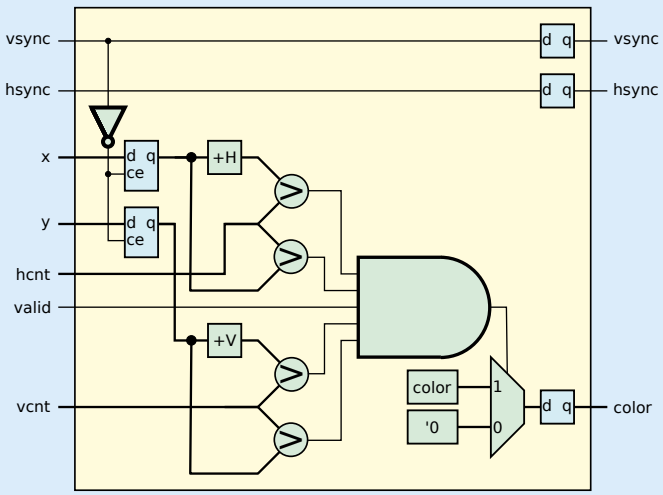
Fotografia 1. Połączone elementy zestawu testowego

Rysujemy prostokąt

Grafika w naszej grze będzie bardzo prosta. Potrzebna nam będzie tylko jedna figura geometryczna: prostokąt. Jak się okaże, jego narysowanie jest całkiem proste. Zaprojektujemy moduł, który będzie przyjmował współrzędne lewego górnego rogu figury i używał do zaświecenia odpowiednich pikseli. Schemat blokowy projektu prezentuje **rysunek 2**.



Rysunek 1. Schemat połączenia peryferiów do Rysino



Rysunek 2. Moduł wyświetlający prostokąt

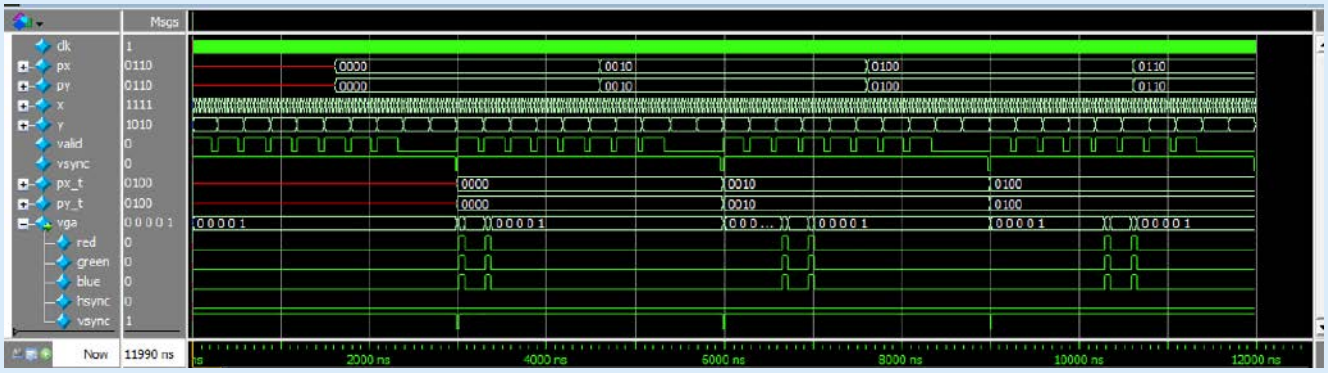
Współrzędne określające położenie są zapisywane w rejestrach, gdy aktywny jest sygnał synchronizacji pionowej (czyli w stanie niskim). Dzięki temu położenie figury nie zmieni się w środku

Listing 1. Struktura vga_t (15_VGA/vga_pkg.sv)

```
10 package vgaPkg;
11
12         typedef struct packed {
13         logic red;
14         logic green;
15         logic blue;
16         logic hsync;
17         logic vsync;
18         } vga_t;
19
20 endpackage : vgaPkg
```

Listing 2. Implementacja modułu rectangle (16_PONG/rectangle.sv)

```
10 module rectangle #(
11     parameter V = 20,
12     parameter H = 80,
13     parameter color = 3'b111,
14     parameter logX = 9,
15     parameter logY = 8
16 ) (
17     input wire clk,
18     input wire [logX-1:0]px,
19     input wire [logY-1:0]py,
20     input wire [logX-1:0]x,
21     input wire [logY-1:0]y,
22     input wire valid,
23     input wire vsync,
24     input wire hsync,
25     output vgaPkg::vga_t vga
26 );
27     logic x_in, y_in;
28     logic [logX-1:0]px_t;
29     logic [logY-1:0]py_t;
30
31     always_ff @(posedge clk)
32         if (vsync == 1'b0)
33             {px_t, py_t} <= {px, py};
34
35     assign x_in = x >= px_t && x < (px_t+H);
36     assign y_in = y >= py_t && y < (py_t+V);
37
38     always_ff @(posedge clk)
39         if (x_in && y_in && valid)
40             {vga.red, vga.green, vga.blue} <= color;
41         else
42             {vga.red, vga.green, vga.blue} <= '0;
43
44     always_ff @(posedge clk) begin
45         vga.hsync <= hsync;
46         vga.vsync <= vsync;
47     end
48
49 endmodule
```



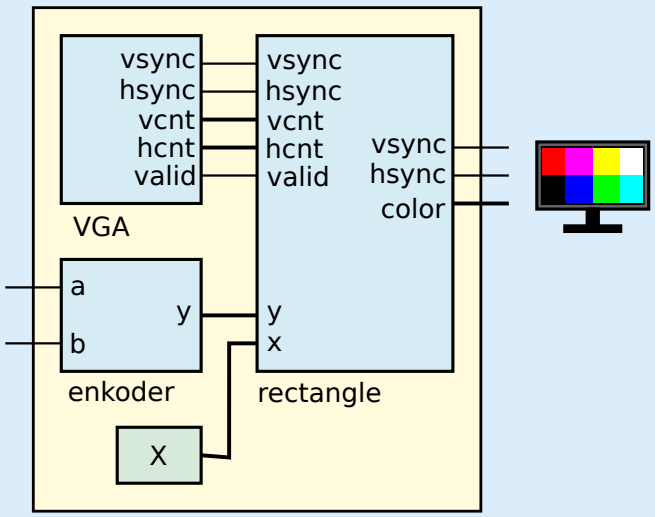
Rysunek 3. Symulacja wyświetlania prostokąta

Listing 3. Generowanie wymuszenia dla symulacji (16_PONG/rectangle_tb.sv)

```
26      initial begin
27          for (frame = 0; frame < 4; frame++)
28              for (y = 0; y < 10; y++)
29                  for (x = 0; x < 15; x++)
30                      @(posedge clk);
31          $stop;
32      end
33
34      assign vsync = !(y == 9 && x == 14);
35      assign valid = (y < 8 && x < 12);
36
37      always_ff @(posedge clk)
38          if (x == 5 && y == 5) begin
39              px = 2*frame;
40              py = 2*frame;
41          end
```

ramki. Wymiary prostokąta są określone przez dwie stałe: `H` (pozioma) i `V` (pionowa). Cztery komparatory sprawdzają, czy obecnie przetwarzany piksel o współrzędnych `hcnt` i `vcnt` należy do prostokąta. Na tej podstawie multiplekser wybiera, czy na wyjście zostanie podany kolor czarny (wektor zerowy), czy zdefiniowany przez stałą `color`. Wyjście modułu zatrzaśnięte jest w przerzutnikach. Sygnały synchronizacyjne po prostu „przepływają" przez moduł. Dzięki temu mają takie samo opóźnienie, jak sygnał mówiący o kolorze.

Aby zmniejszyć liczbę sygnałów, które musimy podłączać, stworzymy strukturę `vga_t`, której kod został pokazany na **listingu 1**. Mamy w niej wszystkie pięć sygnałów złącza VGA. Możemy jej użyć nie tylko wewnątrz modułów, ale także jako wejścia i wyjścia. Jej użycie zostało pokazane m.in. na **listingu 2**, który prezentuje moduł `rectangle`. Stanowi w nim jedyne wyjście. Ale zacznijmy od początku. Definiujemy tu pięć parametrów. Pierwsze – `V` i `H`, to wymiary, a `color` to kolor prostokąta. Ostatnie dwa (`logX` i `logY`) to liczba bitów potrzebnych do reprezentacji współrzędnej piksela. Użyjemy



Rysunek 4. Połączenie generowania prostokąta z blokiem VGA

ich między innymi do zdefiniowania rozmiarów wejść: położenia figury oraz współrzędnych piksela. Poza tym dostajemy także sygnał `valid`, synchronizację oraz oczywiście zegar. Nie mamy tu resetu – nie jest on potrzebny, ponieważ poprawny stan ustali się najpóźniej po pierwszej ramce obrazu.

W liniach 31...34 widzimy zatrzaśnięcie położenia prostokąta wywołane przez sygnał synchronizacji pionowej (czyli raz na ramkę). Samo sprawdzenie, czy obecny piksel należy do prostokąta, zaczyna się w liniach 35...36. Testujemy osobno oś x i y, a wynik zapisujemy w tymczasowych zmiennych. Używamy słowa kluczowego `assign`, więc tworzymy tu logikę kombinacyjną. Korzystamy z nich w wierszu 39, gdzie utworzony jest multiplekser wybierający stan obecnego piksela. Na końcu umieszczamy jeszcze rejestry dla sygnałów synchronizacyjnych (44...47).

Aby sprawdzić nasz nowy moduł w symulacji, musimy przygotować testbench. Jego fragment odpowiedzialny za generowanie wymuszeń widzimy na **listingu 3**. Najpierw, w wierszach 26...32, ustalamy czas trwania symulacji na cztery ramki. Aby uprościć symulację ustalamy rozmiar ekranu na 15×10 pikseli. Nie jest to „prawdziwy" format, obsługiwany przez wyświetlacz. Za to jest na tyle mały, że wyniki będą łatwe do interpretacji. Sygnały synchronizacyjne powstają w liniach 34 i 35. Pozostaje nam jeszcze ustalenie położenia prostokąta, które będzie zmieniało się po każdej ramce (37...41).

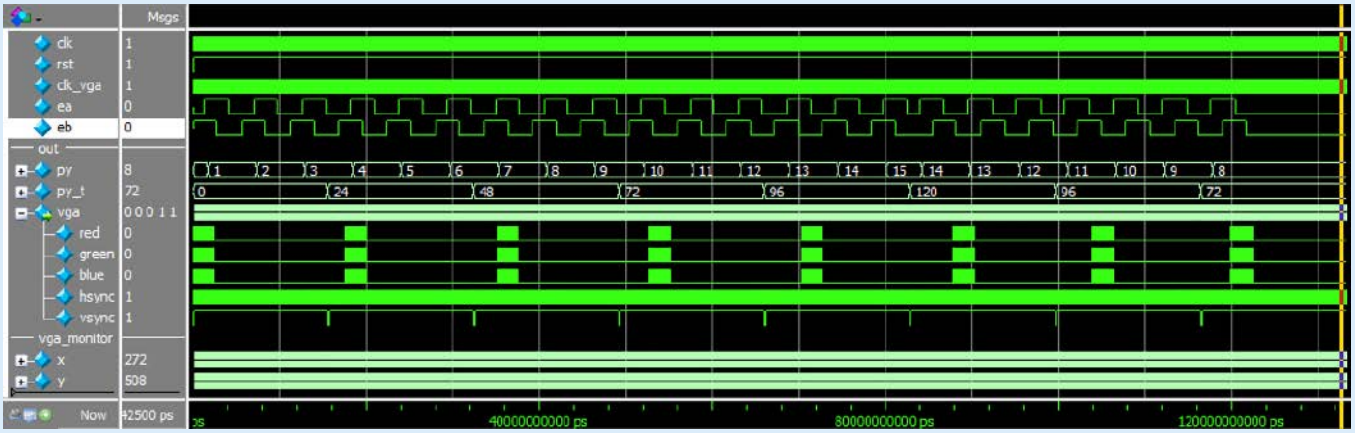
Uruchomiamy symulację – włączamy program ModelSim, przechodzimy do folderu 16_PONG i wykonujemy skrypt:

```
do rectangle.do
```

Uzyskane wyniki prezentuje **rysunek 3**. W pierwszej linii widzimy sygnał zegarowy. Dalej znajdują się pozostałe wejścia. Na samym końcu widzimy wyjście. W jednej linii jest wypisana cała struktura (opisana jako *vga*). Możemy ją rozwinąć i zobaczyć każdy z sygnałów składowych osobno. Zachęcam Czytelnika do przeanalizowana wygenerowanych przebiegów i rozstrzygnięcia, czy moduł zachowuje się zgodnie z oczekiwaniami.

Spróbujmy teraz połączyć nasz nowy moduł z generatorem sygnału VGA. Na początek naszym celem będzie jedynie wyświetlenie prostokąta symbolizującego paletkę gracza oraz zmiana jego pozycji za pomocą enkodera. Schemat modułu został pokazany na **rysunku 4**. Sama integracja modułu VGA z modułem *rectangle* jest prosta: łączymy wyjścia z wejściami. Natomiast sygnały wygenerowane przez blok *rectangle* trafią bezpośrednio na wyjścia układu FPGA. Pozostaje nam jeszcze obsługa enkodera. Nie będziemy się jednak w nią zagłębiać, ponieważ jest prawie identyczna jak ta, którą przygotowaliśmy w trzecim odcinku naszego cyklu. Jednak tym razem do licznika dodane zostało nasycenie. W grze spowoduje to, że dolna i górna krawędź ekranu nie są połączone.

Implementacja tego projektu została pokazana na **listingu 4**. Główną zmianą w stosunku do poprzednich eksperymentów z VGA jest użycie struktury jako wyjścia w module topowym. Jednak Quartus radzi sobie z tym bez żadnego problemu. Możemy przypisać każdy element struktury do poszczególnych wyjść układu FPGA. Sama



Rysunek 5. Symulacja wyświetlania paletki

zawartość modułu jest bardzo prosta: znajdziemy tu połączone razem instancje znanych nam już modułów. Aby zwiększyć prędkość przewijania, wartość uzyskana z enkodera została pomnożona razy 8. Zostało to zrealizowane w linii 57 za pomocą dołączenia trzech zerowych bitów na początku wektora.

Do przetestowania modułu `rectangle_top` użyjemy monitora portu VGA przygotowanego w poprzednim odcinku. Sposób jego połączenia prezentuje **listing 5**. Poza nim testbench zawiera także generowanie sygnałów imitujących ruch enkodera. Symulację uruchamiamy poleceniem:

```
do rectangle_top.do
```

Jej wykonanie zajmie dość dużo czasu (do kilkudziesięciu minut). Zawiera ona ponad osiem pełnych ramek. W sprzęcie będzie to trwało nieco ponad jedną dziesiątą sekundy. Wygenerowane przebiegi prezentuje **rysunek 5**, a uzyskane obrazy **rysunek 6**. Analizując je, należy wziąć pod uwagę to, że pierwsza zapisana ramka nie jest poprawnie zsynchronizowana. Dopiero kolejne odzwierciedlają efekt uzyskany na monitorze.

Wróćmy do przebiegów czasowych. Pierwsze dwa wiersze to zegar generowany przez rezonator kwarcowy oraz reset. Projekt jest taktowany zegarem `clk_vga` uzyskanym z pętli PLL. Sygnały `ea` i `eb` to wymuszenia imitujące działanie enkodera. Aktualną wartość zadaną przez enkoder widzimy w wierszu `py`. Natomiast `py_t` jest to położenie zatrzaskiwane co ramkę w module `rectangle`. Na końcu znajdziemy wyjściowy sygnał `vga`.

Listing 4. Generowanie wymuszenia dla symulacji (16_PONG/rectangle_top.sv)

```
10 module rectangle_top #(
11     parameter H = 800,
12     parameter V = 525,
13     parameter H_BIT = $clog2(H),
14     parameter V_BIT = $clog2(V)
15 ) (
16     input wire clk,
17     input wire rst,
18     input wire a,
19     input wire b,
20     output vgaPkg::vga_t vga
21 );

33     encoder #(
34         .MAX((480-RECTANGLE_V)/8+1)
35     ) enc (
36         .clk(clk_vga),
37         .rst(rst),
38         .a(a),
39         .b(b),
40         .x(py));
41
42     vga #(.H(H), .V(V)) vga_inst (
43         .clk(clk_vga),
44         .rst(rst),
45         .hsync(hsync_vga),
46         .vsync(vsync_vga),
47         .valid(valid_vga),
48         .hcnt(hcnt_vga),
49         .vcnt(vcnt_vga));
50
51     rectangle #(
52         .V(RECTANGLE_V), .H(RECTANGLE_H),
53         .logX(H_BIT), .logY(V_BIT)
54     ) rect (
55         .clk(clk_vga),
56         .px(10),
57         .py({py, 3'd0}),
58         .x(hcnt_vga),
59         .y(vcnt_vga),
60         .valid(valid_vga),
61         .vsync(vsync_vga),
62         .hsync(hsync_vga),
63         .vga(vga));
```

Testy w sprzęcie

Dla przetestowania efektów naszej pracy w sprzęcie uruchamiamy środowisko Quartus i ładujemy projekt *16_PONG/rectangle.qpf*. Posłuży nam on także do kolejnych testów, dlatego przed włączeniem budowy musimy ustawić moduł *rectangle_top* jako nadrzędną instancję projektu (*top level entity*). Teraz możemy rozpocząć budowę projektu. Gdy się zakończy, pozostanie nam już tylko zaprogramowanie płytki. Efekt został pokazany na fotografii otwierającej artykuł. Gdy poruszymy gałką enkodera, prostokąt powinien zmienić swoje położenie. Finalny efekt możemy także zobaczyć na filmie [2].

Piłka

Piłka będzie reprezentowana jako kwadrat. Jednym z celów postawionych przed naszym projektem jest rozdzielenie logiki gry od generowania obrazu. Tę drugą funkcję będzie pełnił moduł *rectangle*. Zastanowimy się teraz, w jaki sposób możemy generować ruch piłki tak, aby mógł on zająć się jej wyświetleniem.

Implementacja jest pokazana na **listingu 6**. Poza standardowym zegarem i resetem mamy tu jeszcze dwa wejścia. Pierwsze z nich, `game` mówi nam, czy obecnie trwa rozgrywka, a drugie `ce`, czy należy obliczyć kolejną klatkę animacji. Mamy także trzy wyjścia: pozycja piłki w dwóch osiach oraz `reflection`. Przyjmuje ono stan wysoki, gdy nastąpiło odbicie piłki od pionowych krawędzi ekranu. Sygnał ten przyda nam się przy rozstrzyganiu, czy gra powinna się zakończyć.

Listing 5. Generowanie wymuszenia dla symulacji (16_PONG/rectangle_top.sv)

```
55     rectangle_top dut (
56         .clk(clk),
57         .rst(rst),
58         .a(ea),
59         .b(eb),
60         .vga(vga));
61
62     vga_monitor #(
63         .NAME("rectangle")
64     ) monitor (
65         .clk(dut.clk_vga),
66         .rst(rst),
67         .c({vga.red, vga.green, vga.blue}),
68         .hsync(vga.hsync),
69         .vsync(vga.vsync));
```

Listing 6. Generowanie ruchu piłki (16_PONG/ball.sv)

```
10 module ball #(

23 ) (
24     input wire clk,
25     input wire rst,
26     input wire game,
27     input wire ce,
28     output logic reflection,
29     output logic [LOG_X-1:0] ball_x,
30     output logic [LOG_Y-1:0] ball_y
31 );

48     always_ff @(posedge clk)
49         if (!rst)
50             by_init <= BALL_Y_MIN;
51         else begin
52             if (by_init < BALL_Y_MAX-BALL_V)
53                 by_init <= by_init + 1'd1;
54             else
55                 by_init <= BALL_Y_MIN;
56         end

69       end else if (!game) begin

73             by_tmp <= by_init;

79       end else begin
80             if (ce) begin
81                 bx_tmp <= bx + ball_vx;
82                 by_tmp <= by + ball_vy;
83             end
84             if (ce1) begin
85                 if (bx_tmp < BALL_X_MIN) begin
86                     bx <= BALL_X_MIN;
87                     ball_vx <= -ball_vx;
88                     reflection_tmp <= '1;
89                 end else if (bx_tmp > BALL_X_
MAX - BALL_H) begin
90                     bx <= BALL_X_MAX - BALL_H;
91                     ball_vx <= -ball_vx;
92                     reflection_tmp <= '1;
93                 end else begin
94                     bx <= bx_tmp;
95                     reflection_tmp <= '0;
96                 end
```

Listing 7. Testy dla modułu ball (16_PONG/ball_tb.sv)

```
26     initial begin
27         ce <= 1'b0;
28         @(posedge clk);
29         forever begin
30             ce <= !ce;
31             @(posedge clk);
32         end
33     end
34
35     initial begin
36         game <= 1'b0;
37         repeat(10) @(posedge clk);
38         game <= 1'b1;
39         repeat(400) @(posedge clk);
40         game <= 1'b0;
41         repeat(40) @(posedge clk);
42         game <= 1'b1;
43         repeat(800) @(posedge clk);
44         $stop;
45     end
```

W wierszach 48...56 widzimy licznik, który cały czas jest inkrementowany z częstotliwością głównego zegara. Jest on używany do losowania początkowej pozycji piłki. Jak widzimy w linii 79, gdy nie jest prowadzona gra, jego zawartość jest kopiowania do aktualnego położenia piłki w pionie.

W czasie rozgrywki pozycja piłki w obu osiach jest inkrementowana o aktualną prędkość (80...82). W następnym takcie zegara następuje sprawdzenie, czy położenie mieści się w zadanym zakresie. Jeżeli nie następuje odbicie, czyli zmiana znaku prędkości dla danej współrzędnej. Jego przebieg dla osi poziomej widzimy w wierszach 85...96. Wersja dla osi pionowej jest analogiczna.

Generowanie wymuszeń na potrzeby testów pokazuje **listing 7**. W wierszach 26...32 generowany jest sygnał `ce`. Docelowo będzie on w stanie wysokim przez jeden cykl zegara na każdą ramkę obrazu. Na potrzeby symulacji zmienia się jednak w każdym takcie zegara. Drugą częścią jest wytworzenie sygnału game, które znajdziemy w wierszach 35...45. W różnych odstępach czasu następuje jego zmiana, co pozwoli nam zaobserwować, w jaki sposób moduł będzie na niego reagował.

Symulację uruchamiamy rozkazem:

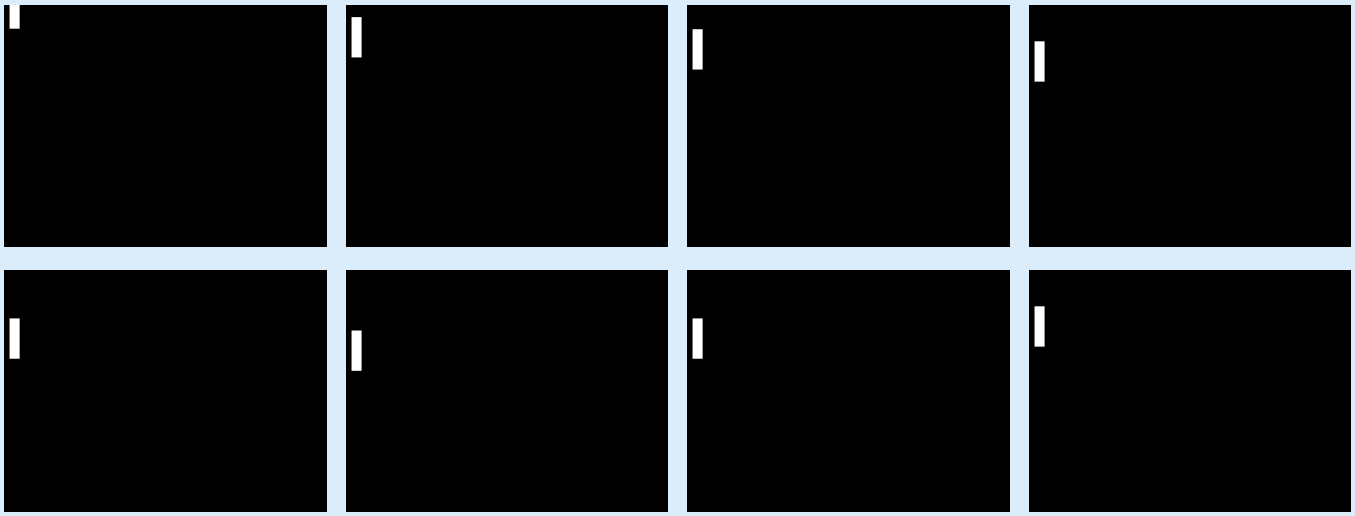
```
do ball.do
```

Wyniki prezentuje **rysunek 7**. Pierwsze dwa wiersze to zegar oraz reset. Dalej znajdziemy sygnał `game` oraz `ce`. Z wyjść najbardziej interesują nas `ball_x` oraz `ball_y`. Jest to aktualne położenie piłki. Wykresy pokazują je zarówno w formie tekstowej, jak i graficznej. Gdy gra jest wstrzymana (`game == 0`), oś pozioma jest ustawiana na środek ekranu, a na pionowej następuje „losowanie" pozycji. Widzimy także, że każde odbicie od pionowej krawędzi ekranu generuje stan wysoki na wyjściu `reflaction`.

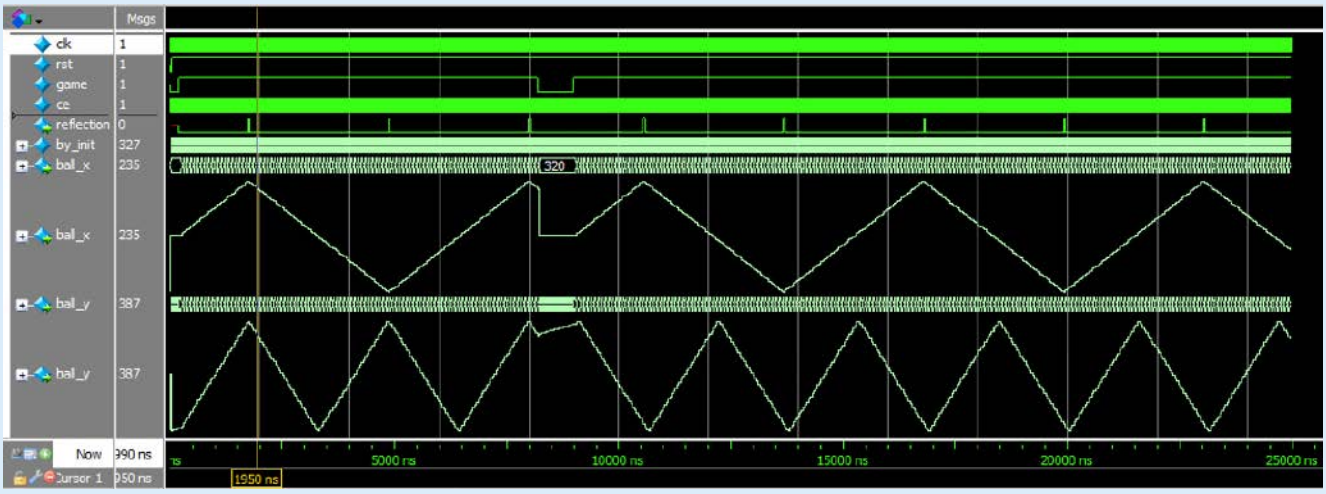
Spróbujmy teraz uruchomić samą piłkę w sprzęcie. W tym celu możemy wziąć poprzedni projekt i zastąpić piłką moduł enkodera. Uzyskany wynik pokazuje **rysunek 8**. Sygnał `ce` jest generowany na narastającym zboczu synchronizacji pionowej. Nowością jest sygnał `button` sterowany przez przycisk enkodera. Steruje on wejściem `game`.

Fragment implementacji tego rozwiązania widoczny jest na **listingu 8**. Sposób połączenia modułów jest klasyczny. Piłka jest kwadratem o boku długości 20 pikseli. Wymiary te są potrzebne zarówno przy obliczaniu, symulacji jak i wyświetlaniu. Sygnał `ce` przyjmuje stan wysoki na zboczu narastającym synchronizacji pionowej (wiersz 51).

Ponieważ symulowanie topowych modułów zajmuje poro czasu, zrezygnowałem z niego w tym przypadku. Poprzestałem jedynie



Rysunek 6. Wyniki z monitora portu VGA

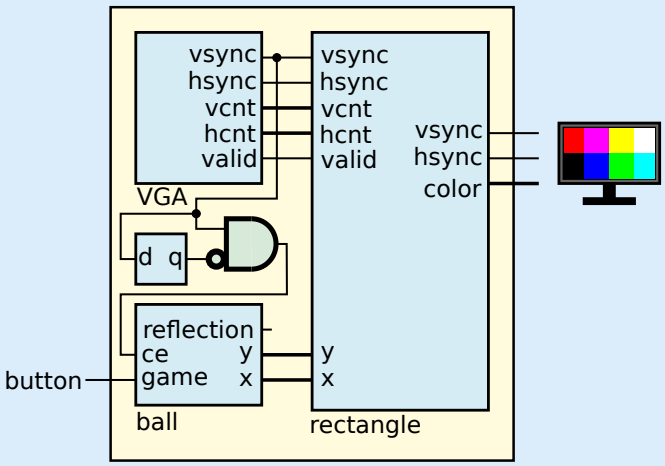


Rysunek 7. Symulacja ruchu piłki

Listing 8. Połączenie modułu piłki (16_PONG/ball_top.sv)

```
35      vga #(.H(H), .V(V)) vga_inst (
36          .clk(clk_vga),
37          .rst(rst),
38          .hsync(hsync_vga),
39          .vsync(vsync_vga),
40          .valid(valid_vga),
41          .hcnt(hcnt_vga),
42          .vcnt(vcnt_vga));
43
44      always_ff @(posedge clk_vga)
45          vsync_vga_r <= vsync_vga;
46
47      ball #(.BALL_H(20), .BALL_V(20)) ball_inst (
48          .clk(clk_vga),
49          .rst(rst),
50          .game(button),
51          .ce(vsync_vga & !vsync_vga_r),
52          .ball_x(ball_x),
53          .ball_y(ball_y));
54
55      rectangle #(
56          .V(20), .H(20),
57          .logX(H_BIT), .logY(V_BIT)
58      ) rect (
59          .clk(clk_vga),
60          .px(ball_x),
61          .py(ball_y),
62          .x(hcnt_vga),
63          .y(vcnt_vga),
64          .valid(valid_vga),
65          .vsync(vsync_vga),
66          .hsync(hsync_vga),
67          .vga(vga));
```

na teście pojedynczych modułów. Generowanie obrazu piłki sprawdzimy za pomocą projektu *rectangle.qpf*. Przed rozpoczęciem budowy musimy jednak zmienić topowy moduł na *ball_top.sv*. Po uruchomieniu po ekranie będzie sunąć piłka odbijająca się od jego brzegów. Natomiast gdy naciśniemy przycisk, zobaczymy proces losowania pozycji startowej. Piłka zostanie przeniesiona na środek ekranu i z dużą prędkością będzie poruszać się po linii pionowej. Gdy zwolnimy



Rysunek 8. Połączenie piłki do modułu vga

przyciski, rozpocznie swój normalny ruch od ostatniego położenia. Wynik możemy zobaczyć także na filmie [2].

Podsumowanie

W tym odcinku wyświetliliśmy na ekranie prostokąt. W pierwszej wersji jego położenie było zadawane za pomocą enkodera, a w drugim widzieliśmy na bieżąco generowaną animację ruchu piłki. W następnym odcinku przygotujemy algorytm sterujący położeniem przeciwnika oraz uruchomimy całą grę PONG.

Rafał Kozik
rafkozik@gmail.com

Materiały dodatkowe:
[1] Repozytorium http://bit.ly/33uYPxs
[2] Film demonstrujący działanie projektu https://bit.ly/3fZkHsP