UCiSW 2 projekt – gra Pong

Marek Machliński (241308)

Daniel Król (241399)

Prowadzący: dr inż. Jacek Mazurkiewicz

Spis treści

Spis treści 1

Cel projektu 1

Opis komponentów 2

- Input Manager 2
- Player 4
- PowerUp 7
- CollisionManager 9
- RenderManager 17
- Ball 19
- GameLogic 21

Podsumowanie 24

Cel projektu

Celem projektu było wykonanie programu w VHDL realizującego grę Pong, który powinien zostać uruchomiony na zestawie Spartan3e Starter Kit. Gra oferuje możliwość rozgrywki dla dwóch graczy za pomocą jednej klawiatury. Sterowanie dla każdego z graczy odbywa się z użyciem dwóch klawiszy: W i S oraz strzałka w górę i strzałka w dół. Dodatkowo istnieje opcja resetu rozgrywki za pomocą klawisza R i pauzy za pomocą klawisza P. Każdy z graczy dysponuje paletką, którą może poruszać w górą lub w dół ekranu. Między graczami odbija się piłka, która jeśli przekroczy linię paletki któregoś z nich, to przeciwnik zdobywa punkt. Zadaniem graczy jest wprowadzanie paletki na kurs kolizyjny z piłką. Dodatkowo piłka może odbijać się od górnej i dolnej krawędzi ekranu. Po osiągnięciu maksymalnej liczby punktów jeden z graczy wygrywa. W grze pojawiają się także losowe ulepszenia, które gracze mogą zebrać, aby zmodyfikować rozgrywkę (np. powiększenie piłki, powiększenie paletki, przyspieszenie paletki itd.).

Opis komponentów

Każdy z komponentów został przetestowany przy użyciu symulacji behawioralnej. Nazwa każdego z obrazów poniżej nawiązuje do nazwy jego pliku testowego.

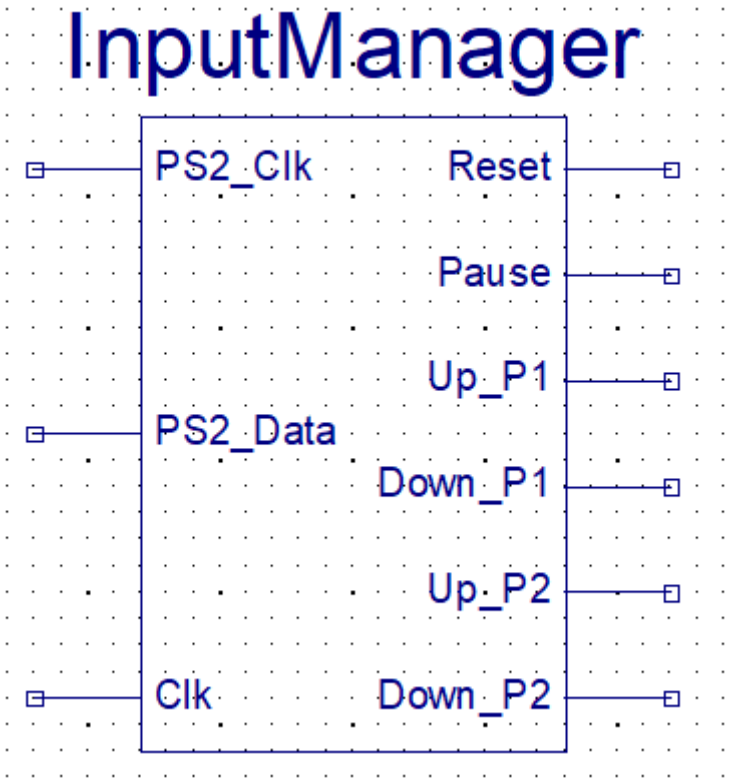
Input Manager

InputManager odpowiada za obsługę odczytanych klawiszy, które zostały wciśnięte. Korzysta on z modułu PS2_Kbd do odczytu kodów klawiszy z klawiatury. Jego logika polega na rozpoznaniu kodu wciśniętego klawisza za pomocą klauzuli case … when.

Moduł ten korzysta z danych dostarczanych przez moduł PS2_kbd w celu detekcji, który klawisz został naciśnięty. Tak odebrany sygnał naciśnięcia klawisza jest następnie propagowany do modułów odpowiedzialnych za obsługę zdarzeń następujących po naciśnięciu odpowiednich klawiszy. Sygnał resetu jest przekazywany do Player, aby zresetować punkty i do GameLogic, aby przywrócić wartości do stałych ustalanych na początku gry, a następnie wywołać reset w pozostałych modułach. Pauza aktywuje w RenderManager wyświetlenie litery P informującej o pauzie, w module Ball powoduje zatrzymanie poruszania się piłki, a w GameLogic zatrzymanie liczników do tworzenia ulepszeń.

Procesy w module:

```
begin
    if rising_edge(Clk) then
        Reset <= '0';
        Up_P1 <= '0';
        Down_P1 <= '0';
        Up_P2 <= '0';
        Down_P2 <= '0';
        if DO_Rdy = '1' then
            case DO is
                when X"2D" => -- Reset "R"
                    Reset <= not F0;
                    pause_int <= '1';
                when X"4D" => -- Pause "P"
                    if F0 = '1' then
                        pause_int <= not pause_int;
                    end if;
                when X"1D" => -- Up_P1 "W"
                    Up_P1 <= (not F0) and pause_int;
                when X"1B" => -- Down_P1 "S"
                    Down_P1 <= (not F0) and pause_int;
                when X"75" => -- UP_P2 "Up arrow"
                    UP_P2 <= (not F0) and E0 and pause_int;
                when X"72" => -- Down_P2 "Down arrow"
                    Down_P2 <= (not F0) and E0 and pause_int;
                when others =>
            end case;
        end if;
    end if;
end process;
```



Player

Instancja modułu Player jest osobna dla każdego z graczy. Posiada on proces odpowiedzialny za zwiększanie punktów, który może resetować liczbę punktów do zera w przypadku naciśnięcia resetu lub zwiększać liczbę punktów gracza w przypadku otrzymania sygnału zwiększenia punktów. Limitem punktów jest stała liczbowa. Drugi proces odpowiada za modyfikowanie prędkości poruszania się paletki gracza, która może zostać zwiększona do wartości maksymalnej lub zmniejszona do domyślnej (1 jednostka odległości na 1 jednostkę czasu). Analogicznie zachowuje się trzeci proces odpowiadający za modyfikację rozmiaru paletki gracza: może zostać powiększona do rozmiaru maksymalnego lub zmniejszona do rozmiaru minimalnego. Ostatni proces odpowiada za modyfikację pozycji paletki gracza, która jest obliczana na podstawie obecnej pozycji, docelowego kierunku poruszenia się i obecnej prędkości poruszania się paletki danego gracza.

Moduł ten ma swoje wejścia podłączone m. in. do modułu CollisionManager, od którego otrzymuje sygnały informujące o zmianie szybkości poruszania sią, rozmiaru i zdobytych punktach gracza. Od modułu InputManager sygnały związane z ruchami wykonywanymi przez gracza, a moduł gracza konsumuje je zmieniając pozycję danego gracza na podstawie tych sygnałów. Player produkuje trzy sygnały:
- Size, który wysyłany jest do CollisionManager, ponieważ warunki kolizji są zależne od rozmiaru paletki gracza, a także do RenderManager, ponieważ od rozmiaru gracza zależy to, jak będzie on wyświetlany

- Position, który także jest potrzebny do obliczania kolizji gracza i wyświetlania obrazu

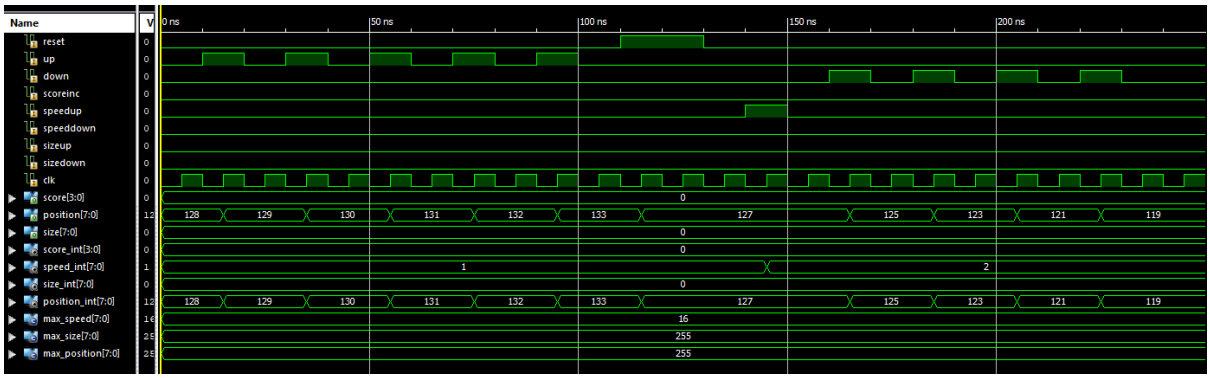
- Score, który jest podłączony na GameLogic w celu obliczania warunku końca gry i do RenderManager, aby wyświetlać aktualny wynik

Procesy w module:

```
process(Clk, Reset, ScoreInc)
    begin
        if rising_edge(Clk) then
            if Reset = '1' then
                score_int <= X"0";
            else
                if ScoreInc = '1' and score_int < X"9" then
                    score_int <= score_int + 1;
                end if;
            end if;
        end if;
    end process;

    process(Clk, Reset, SpeedUp, SpeedDown)
    begin
        if rising_edge(Clk) then
            if Reset = '1' then
                speed_int <= X"001";
            else
                if SpeedUp = '1' and speed_int < max_speed then
                    speed_int <= speed_int + 1;
                end if;
                if SpeedDown = '1' and speed_int > X"001" then
                    speed_int <= speed_int - 1;
                end if;
            end if;
        end if;
    end process;
```

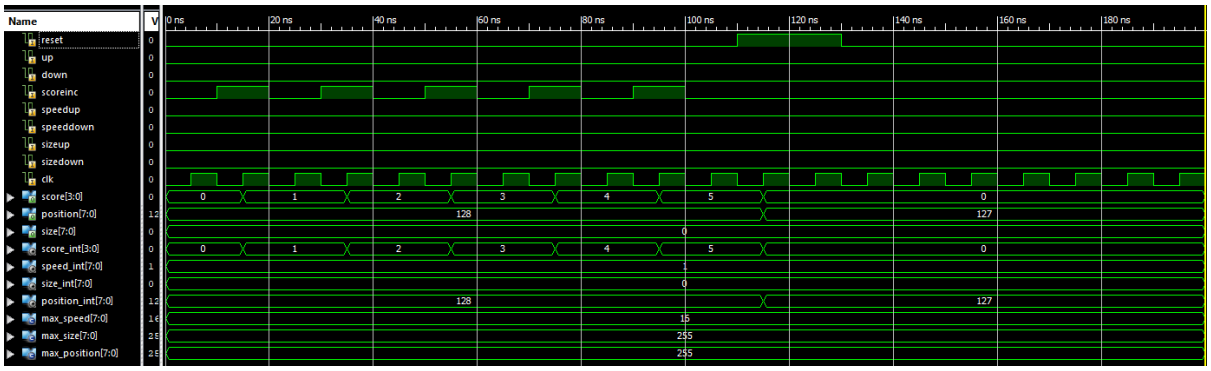
```
    process(Clk, Reset, SizeUp, SizeDown)
    begin
        if rising_edge(Clk) then
            if Reset = '1' then
                size_int <= X"000";
            else
                if SizeUp = '1' and size_int < max_size then
                    size_int <= size_int + 1;
                end if;
                if SizeDown = '1' and size_int > X"000" then
                    size_int <= size_int - 1;
                end if;
            end if;
        end if;
    end process;

    process(Clk, Reset, Up, Down)
    begin
        if rising_edge(Clk) then
            if Reset = '1' then
                position_int <= X"0F0";
            else
                if Up = '1' and position_int + size_int <= max_position -
speed_int then
                    position_int <= position_int + speed_int;
                end if;
                if Down = '1' and position_int >= speed_int + size_int then
                    position_int <= position_int - speed_int;
                end if;
            end if;
        end if;
    end process;
```

Testowanie modułu polegało na obserwacji zmian wartości pozycji gracza zależnie od aktualnej prędkości poruszania się, pozycji i otrzymanego docelowego kierunku poruszania się. Paletka gracza powinna poruszać się od aktualnej pozycji gracza o wartość szybkości poruszania się w stronę zdefiniowaną przez sygnał kierunku poruszania. Dodatkowo w momencie odebrania sygnału przyspieszenia gracz powinien zacząć poruszać się ze zwiększoną szybkością:



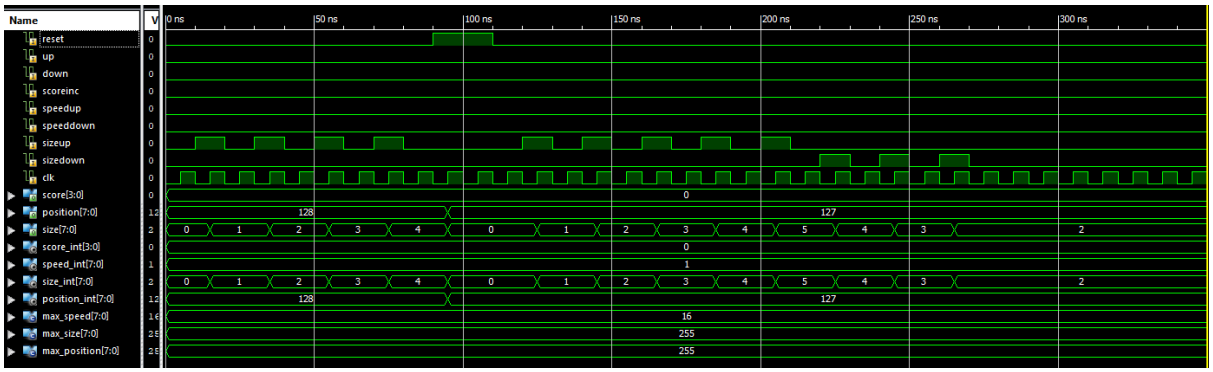
1 Player_shouldMoveUpAndDown

Moduł gracza powinien reagować na sygnał resetu poprzez zresetowanie aktualnej wartości punktów gracza i przesunięciu go do pozycji domyślnej:



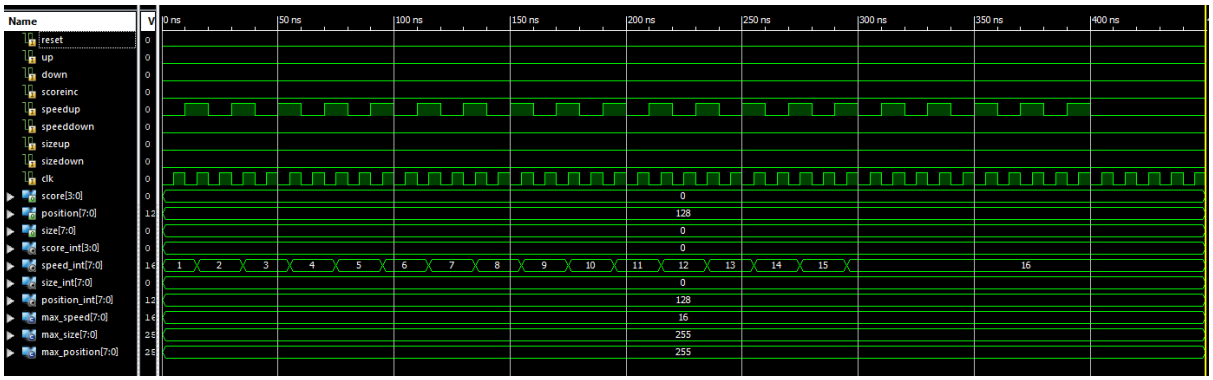
2 Player_shouldResetScore

Gracz powinien również reagować na sygnały zwiększania i zmniejszania się rozmiaru jego paletki. Dla resetu wielkość paletki powinna być przywracana do domyślnej:



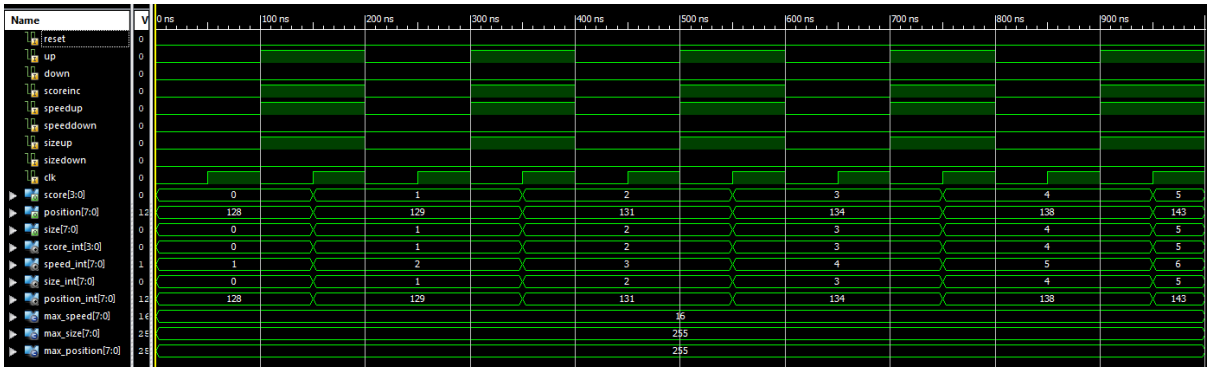
3 Player_shouldSizeDown

Gracz powinien reagować na zmiany jego szybkości poruszania się uwzględniając maksymalną wartość prędkości, po osiągnięciu której nie zostaje ona zwiększana:

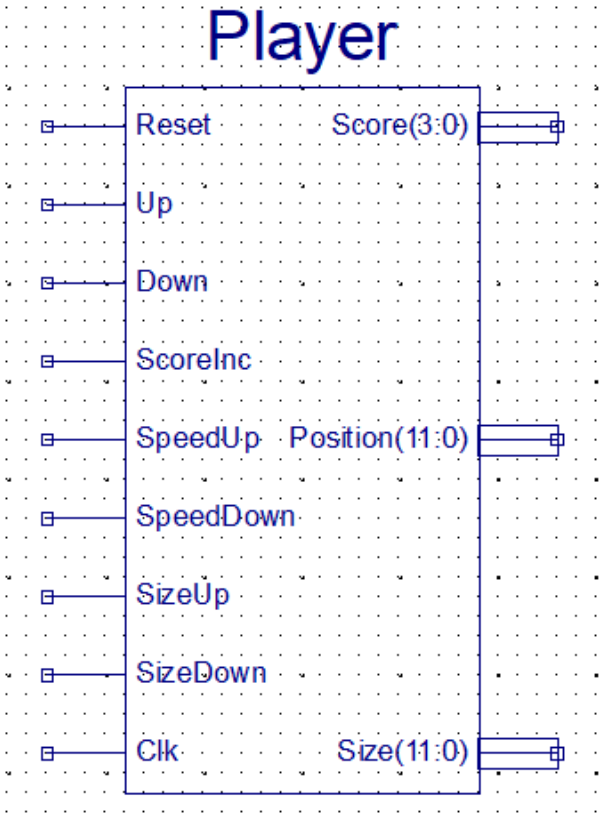


4 Player_shouldSpeedUpToMaxSpeed

Ostatecznie gracz powinien reagować na wszystkie podane wcześniej sygnały tak samo w przypadku gdy występują one osobne jak i w momencie, gdy występują jednocześnie. Test obrazujący zachowanie tego modułu pod wpływem wszystkich sygnałów go dotyczących:



5 Player_behavior



PowerUp

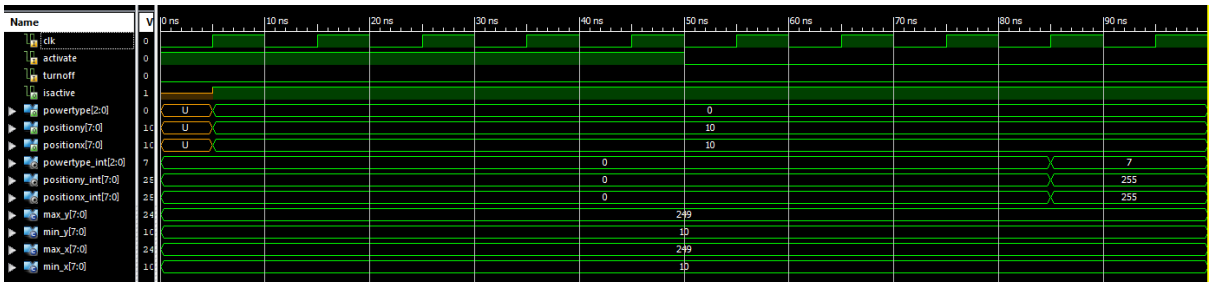
Ten moduł odpowiada za zarządzanie stanem ulepszeń rozmieszczonych na mapie. Ulepszenia rozmieszczane są w losowych punktach, a w przypadku wylosowaniu pozycji dla ulepszenia poza obszarem gry pozycja ta zostaje przesunięta do krawędzi, poza którą ta pozycja wykracza. Dane ulepszenie może być aktywowane lub dezaktywowane. Moduł ten posiada submoduł RandGen, którego instancje odpowiadają kolejno za generowanie losowego typu ulepszenia (typ ulepszenia określa, czy ulepszeniem będzie np. powiększenie paletki czy powiększenie piłki), a także za generowanie losowej wartości pozycji ulepszenia osobno w płaszczyźnie X i Y.

Moduł ten przyjmuje sygnał Activate od modułu GameLogic, który oznacza stworzenie nowego ulepszenia na planszy. Od CollisionManager przyjmuje sygnał odpowiadający za wyłączenie efektu ulepszenia. PowerUp produkuje sygnał isActive sygnalizujący, czy dane ulepszenie jest w tym momencie aktywne, co jest potrzebne w CollisionManager, aby odpowiednio rozpatrywać kolizje z powstałymi ulepszeniami i z RenderManager, aby wyświetlać odpowiednie symbole dla pojawiających się ulepszeń. PowerUpType mówi o rodzaju (efekcie) ulepszenia, więc jest potrzebny w CollisionManager, aby określić, z którym ulepszeniem zaszła ewentualna kolizja, a także w RenderManager, aby określić, oznaczenie którego z ulepszeń powinno być wyświeltane. Dodatkowo sygnały zawierające koordynaty w osi X i Y są wysyłane do CollisionManager, aby móc określić, czy powinna zajść kolizja z ulepszeniem znajdującym się w danym miejscu, a także do RenderManager, aby wyświetlać symbole ulepszeń w odpowiednich pozycjach.

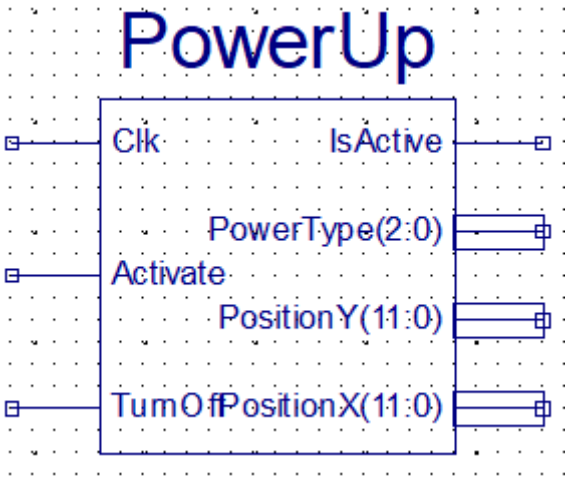
Procesy w module:

```
process(Clk, Activate, TurnOff)
    begin
        if rising_edge(Clk) then
            if TurnOff = '1' then
                IsActive <= '0';
            elsif Activate = '1' then
                PowerType <= powerType_int;
                if positionY_int < min_y then
                    PositionY <= min_y;
                elsif positionY_int > max_y then
                    PositionY <= max_y;
                else
                    PositionY <= positionY_int;
                end if;
                if positionX_int < min_x then
                    PositionX <= min_x;
                elsif positionX_int > max_x then
                    PositionX <= max_x;
                else
                    PositionX <= positionX_int;
                end if;
                IsActive <= '1';
            end if;
        end if;
    end process;
```

Przykładowa generacja losowego ulepszenia:



6 PowerUp_shouldGenerateRandomPowerUp



CollisionManager

Jest to moduł odpowiadający za sprawdzanie kolizji podczas poruszania się piłki. Na początku ustalany jest rozmiar piłki na podstawie jej pozycji i aktualnego rozmiaru. Następnie sprawdzane są warunki matematyczne związane z pozycją paletki gracza (także z uwzględnieniem ulepszeń), pozycji piłki i wektora poruszania się piłki. Na podstawie tego najpierw wykrywana jest kolizja z prawą ścianą, a następnie sprawdzanie, czy w tym miejscu ściany znajdowała się paletka gracza. Jeśli nie było tam paletki, to punkt zyskuje gracz po lewej stronie. Analogiczne warunki są sprawdzane dla kolizji z lewą ścianą. Ostatnimi opcjami jest sprawdzanie kolizji z górną i dolną krawędzią ekranu, jednak wtedy ustalany jest jedynie kolejny wektor odbicia piłki bez sprawdzania warunków dotyczących punktacji. Drugi z procesów tego modułu odpowiada za sprawdzanie kolizji z ulepszeniami. Odbywa się to na podstawie obecnej pozycji piłki, obecnej szybkości poruszania się piłki, docelowego miejsca poruszenia się piłki i pozycji ulepszenia. W przypadku kolizji z ulepszeniem sprawdzany jest jego typ, a na podstawie tego typu aktywowane jest konkretne ulepszenie. Taka kolizja możliwa jest w czterech przypadkach, dlatego za sprawdzanie każdej z możliwych pozycji piłki względem ulepszenia odpowiedzialna jest osobna wartość w klauzuli case … when.

Moduł ten przyjmuje sygnały związane z kierunkiem poruszania się, prędkością, i pozycją piłki, aby określać jej kolizję z innymi elementami. Pozycje i rozmiary gracza potrzebne są podczas obliczania kolizji piłki z paletką. Informacje o ulepszeniu, czyli jego typ, pozycja na mapie i status aktywności potrzebne są do określenia czy nastąpiła kolizja z tym obiektem i aktywacją którego z ulepszeń powinno to poskutkować. Na wyjściu CollisionManager znajduje się sygnał informujący moduł Ball, w którym momencie powinien został ustawiony nowy wektor ruchu piłki, a także wartość samego nowego wektora poruszania się piłki. Pozostałe sygnały wyjściowe dotyczą rodzaju podniesionego

ulepszenia i zależnie od docelowego obiektu na który dane ulepszenie ma wpływ, są wysyłane do Player lub Ball, gdzie dopiero są przetwarzane na konkretne efekty względem zasad gry.

Procesy w module:

```
process(Clk, Reset, BallVector, BallPositionY, BallPositionX,
PlayerLeftPosition, PlayerRightPosition)
        variable player_margin_up : UNSIGNED(11 downto 0) := X"000";
        variable player_margin_down : UNSIGNED(11 downto 0) := X"000";
        variable ball_margin_up : UNSIGNED(11 downto 0) := X"000";
        variable ball_margin_down : UNSIGNED(11 downto 0) := X"000";
    begin
        if rising_edge(Clk) then
            PlayerLeftScore <= '0';
            PlayerRightScore <= '0';
            if Reset = '1' then
                ball_vector <= not BallVector;
                SetBallVector <= '1';
            else
                SetBallVector <= '0';
                ball_vector <= BallVector;
                ball_margin_up := BallPositionY + BallSize;
                ball_margin_down := BallPositionY - BallSize;
                if BallVector(0) = '1' then
                    -- Check if hits right wall (point)
                    if BallPositionX >= max_x - BallSpeed - BallSize then
                        player_margin_up := PlayerRightPosition +
PlayerRightSize + player_minimal_size;
                        player_margin_down := PlayerRightPosition -
PlayerRightSize - player_minimal_size;
                        -- Bounce from right player (if on vector)
                        if (ball_margin_down <= player_margin_up and
ball_margin_down >= player_margin_down) or -- Player bigger than ball
                            (ball_margin_up <= player_margin_up and
ball_margin_up >= player_margin_down) or     --/
                            (player_margin_down <= ball_margin_up and
player_margin_down >= ball_margin_down) or -- Ball bigger than player
                            (player_margin_up <= ball_margin_up and
player_margin_up >= ball_margin_down) then   --/
                            SetBallVector <= '1';
                            ball_vector(0) <= '0';
                        else
                            PlayerLeftScore <= '1';
                        end if;
                    end if;
                else
                    -- Check if hits left wall (point)
                    if BallPositionX <= BallSpeed + BallSize then
                        player_margin_up := PlayerLeftPosition +
PlayerLeftSize + player_minimal_size;
                        player_margin_down := PlayerLeftPosition -
PlayerLeftSize - player_minimal_size;
                        -- Bounce from left player (if on vector)
                        if (ball_margin_down <= player_margin_up and
ball_margin_down >= player_margin_down) or -- Player bigger than ball
                            (ball_margin_up <= player_margin_up and
ball_margin_up >= player_margin_down) or     --/
                            (player_margin_down <= ball_margin_up and
player_margin_down >= ball_margin_down) or -- Ball bigger than player
                            (player_margin_up <= ball_margin_up and
player_margin_up >= ball_margin_down) then   --/
```

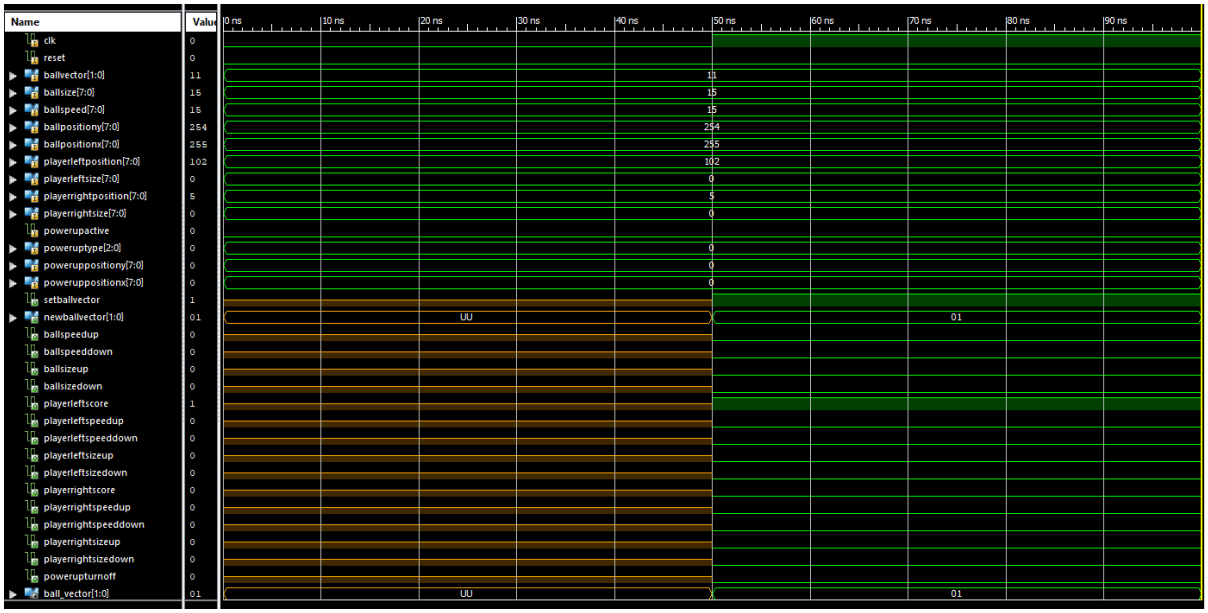
```
                            SetBallVector <= '1';
                            ball_vector(0) <= '1';
                        else
                            PlayerRightScore <= '1';
                        end if;
                    end if;
                end if;
                if BallVector(1) = '1' then
                    -- Check if bounces from top wall
                    if BallPositionY >= max_y - BallSpeed - BallSize then
                        SetBallVector <= '1';
                        ball_vector(1) <= '0';
                    end if;
                else
                    -- Check if bounces from down wall
                    if BallPositionY <= BallSpeed + BallSize then
                        SetBallVector <= '1';
                        ball_vector(1) <= '1';
                    end if;
                end if;
            end if;
        end if;
    end process;

    process(Clk, ball_vector, BallPositionY, BallPositionX, PowerUpActive)
        variable ball_next_y : UNSIGNED(11 downto 0) := X"000";
        variable ball_next_x : UNSIGNED(11 downto 0) := X"000";
    begin
        if rising_edge(Clk) then
            BallSpeedUp <= '0';
            BallSpeedDown <= '0';
            BallSizeUp <= '0';
            BallSizeDown <= '0';
            PlayerLeftSpeedUp <= '0';
            PlayerLeftSpeedDown <= '0';
            PlayerLeftSizeUp <= '0';
            PlayerLeftSizeDown <= '0';
            PlayerRightSpeedUp <= '0';
            PlayerRightSpeedDown <= '0';
            PlayerRightSizeUp <= '0';
            PlayerRightSizeDown <= '0';
            PowerUpTurnOff <= '0';
            -- Check power up
            if PowerUpActive = '1' then
                if ball_vector(0) = '1' then
                    ball_next_x := BallPositionX + BallSpeed;
                else
                    ball_next_x := BallPositionX - BallSpeed;
                end if;
                if ball_vector(1) = '1' then
                    ball_next_y := BallPositionY + BallSpeed;
                else
                    ball_next_y := BallPositionY - BallSpeed;
                end if;
                case ball_vector is
                    when "00" => -- Down left
                        if PowerUpPositionX >= ball_next_x and
PowerUpPositionX <= BallPositionX and
                            PowerUpPositionY >= ball_next_y and
PowerUpPositionY <= BallPositionY then
                            PowerUpTurnOff <= '1';
```

```
                                case PowerUpType is
                                when "000" =>
                                    BallSpeedUp <= '1';
                                when "001" =>
                                    BallSpeedDown <= '1';
                                when "010" =>
                                    BallSizeUp <= '1';
                                when "011" =>
                                    BallSizeDown <= '1';
                                when "100" =>
                                    if ball_vector(0) = '1' then
                                        PlayerLeftSpeedUp <= '1';
                                    else
                                        PlayerRightSpeedUp <= '1';
                                    end if;
                                when "101" =>
                                    if ball_vector(0) = '1' then
                                        PlayerLeftSpeedDown <= '1';
                                    else
                                        PlayerRightSpeedDown <= '1';
                                    end if;
                                when "110" =>
                                    if ball_vector(0) = '1' then
                                        PlayerLeftSizeUp <= '1';
                                    else
                                        PlayerRightSizeUp <= '1';
                                    end if;
                                when "111" =>
                                    if ball_vector(0) = '1' then
                                        PlayerLeftSizeDown <= '1';
                                    else
                                        PlayerRightSizeDown <= '1';
                                    end if;
                                when others =>
                            end case;
                            end if;
                        when "10" => -- Up left
                            if PowerUpPositionX >= ball_next_x and
PowerUpPositionX <= BallPositionX and
                                PowerUpPositionY >= BallPositionY and
PowerUpPositionY <= ball_next_y then
                                PowerUpTurnOff <= '1';
                                case PowerUpType is
                                when "000" =>
                                    BallSpeedUp <= '1';
                                when "001" =>
                                    BallSpeedDown <= '1';
                                when "010" =>
                                    BallSizeUp <= '1';
                                when "011" =>
                                    BallSizeDown <= '1';
                                when "100" =>
                                    if ball_vector(0) = '1' then
                                        PlayerLeftSpeedUp <= '1';
                                    else
                                        PlayerRightSpeedUp <= '1';
                                    end if;
                                when "101" =>
                                    if ball_vector(0) = '1' then
                                        PlayerLeftSpeedDown <= '1';
                                    else
```

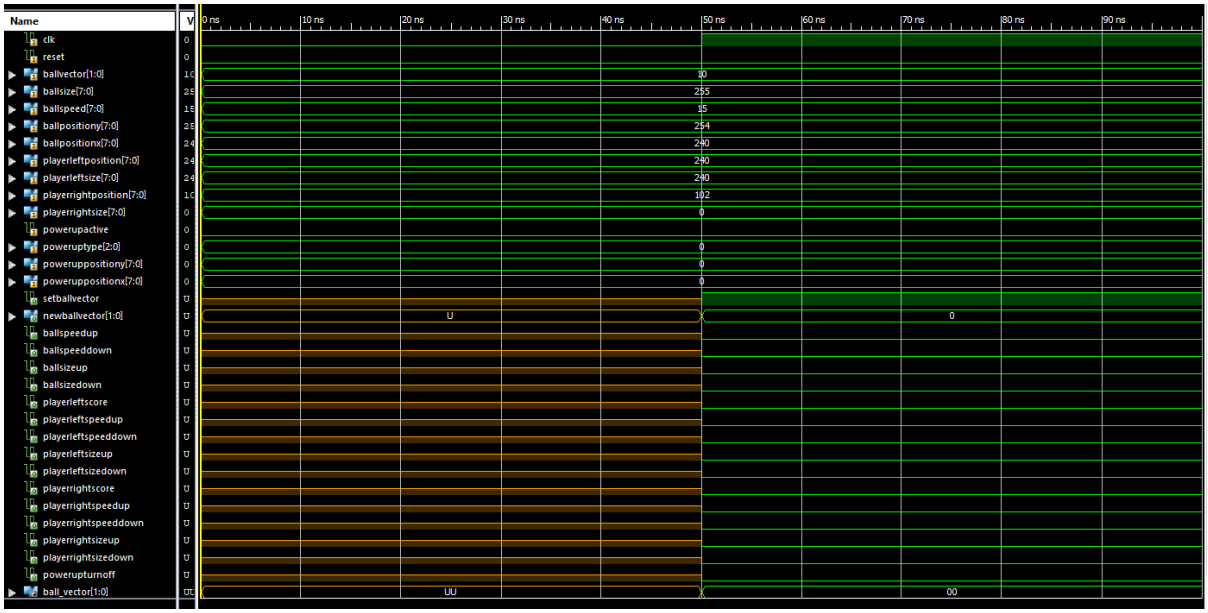
```
                                    PlayerRightSpeedDown <= '1';
                                end if;
                            when "110" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSizeUp <= '1';
                                else
                                    PlayerRightSizeUp <= '1';
                                end if;
                            when "111" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSizeDown <= '1';
                                else
                                    PlayerRightSizeDown <= '1';
                                end if;
                            when others =>
                        end case;
                        end if;
                    when "01" => -- Down right
                        if PowerUpPositionX >= BallPositionX and
PowerUpPositionX <= ball_next_x and
                            PowerUpPositionY >= ball_next_y and
PowerUpPositionY <= BallPositionY then
                            PowerUpTurnOff <= '1';
                            case PowerUpType is
                            when "000" =>
                                BallSpeedUp <= '1';
                            when "001" =>
                                BallSpeedDown <= '1';
                            when "010" =>
                                BallSizeUp <= '1';
                            when "011" =>
                                BallSizeDown <= '1';
                            when "100" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSpeedUp <= '1';
                                else
                                    PlayerRightSpeedUp <= '1';
                                end if;
                            when "101" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSpeedDown <= '1';
                                else
                                    PlayerRightSpeedDown <= '1';
                                end if;
                            when "110" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSizeUp <= '1';
                                else
                                    PlayerRightSizeUp <= '1';
                                end if;
                            when "111" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSizeDown <= '1';
                                else
                                    PlayerRightSizeDown <= '1';
                                end if;
                            when others =>
                        end case;
                        end if;
                    when "11" => -- Up right
```

```
                        if PowerUpPositionX >= BallPositionX and
PowerUpPositionX <= ball_next_x and
                            PowerUpPositionY >= BallPositionY and
PowerUpPositionY <= ball_next_y then
                            PowerUpTurnOff <= '1';
                            case PowerUpType is
                            when "000" =>
                                BallSpeedUp <= '1';
                            when "001" =>
                                BallSpeedDown <= '1';
                            when "010" =>
                                BallSizeUp <= '1';
                            when "011" =>
                                BallSizeDown <= '1';
                            when "100" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSpeedUp <= '1';
                                else
                                    PlayerRightSpeedUp <= '1';
                                end if;
                            when "101" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSpeedDown <= '1';
                                else
                                    PlayerRightSpeedDown <= '1';
                                end if;
                            when "110" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSizeUp <= '1';
                                else
                                    PlayerRightSizeUp <= '1';
                                end if;
                            when "111" =>
                                if ball_vector(0) = '1' then
                                    PlayerLeftSizeDown <= '1';
                                else
                                    PlayerRightSizeDown <= '1';
                                end if;
                            when others =>
                        end case;
                        end if;
                    when others =>
                end case;
            end if;
        end if;
    end process;
```

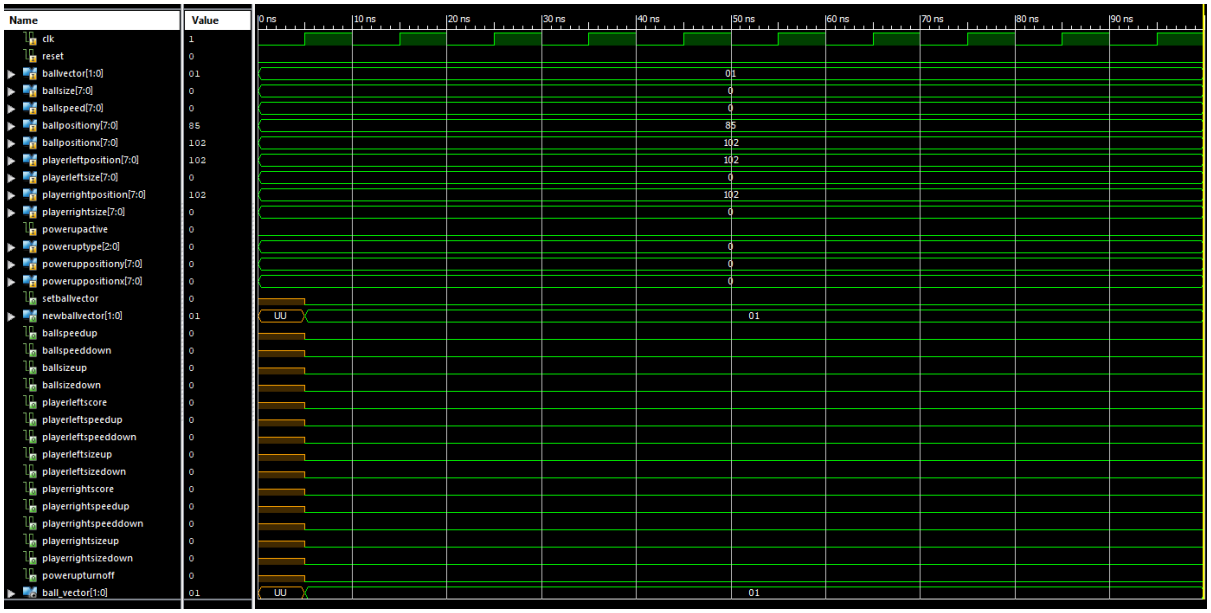
Testowanie tego modułu polegało na sprawdzeniu jaki wektor (czyli dwubitowy kierunek) odbicia piłki zostanie zwrócony dla konkretnych wartości pozycji paletki gracza, piłki i docelowego wektora poruszenia się. Dodatkowo w przypadku trafienia w jedną ze ścian graczy (gdy w tym momencie nie ma tam paletki) powinien być zdobywany punkt przez przeciwnika. Przykładowe wartości pozycji obiektów i reakcja na nie:



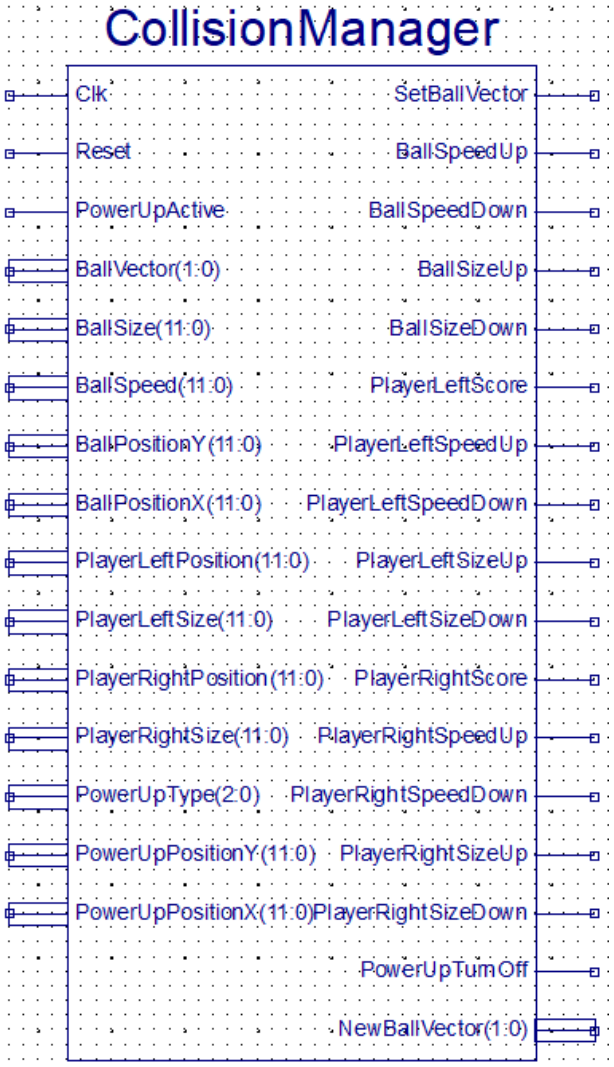
7 CollisionManager_shouldHitRightWall



8 CollisionManager_shouldBounceLeftWall



9 CollisionManager_behavior



RenderManager

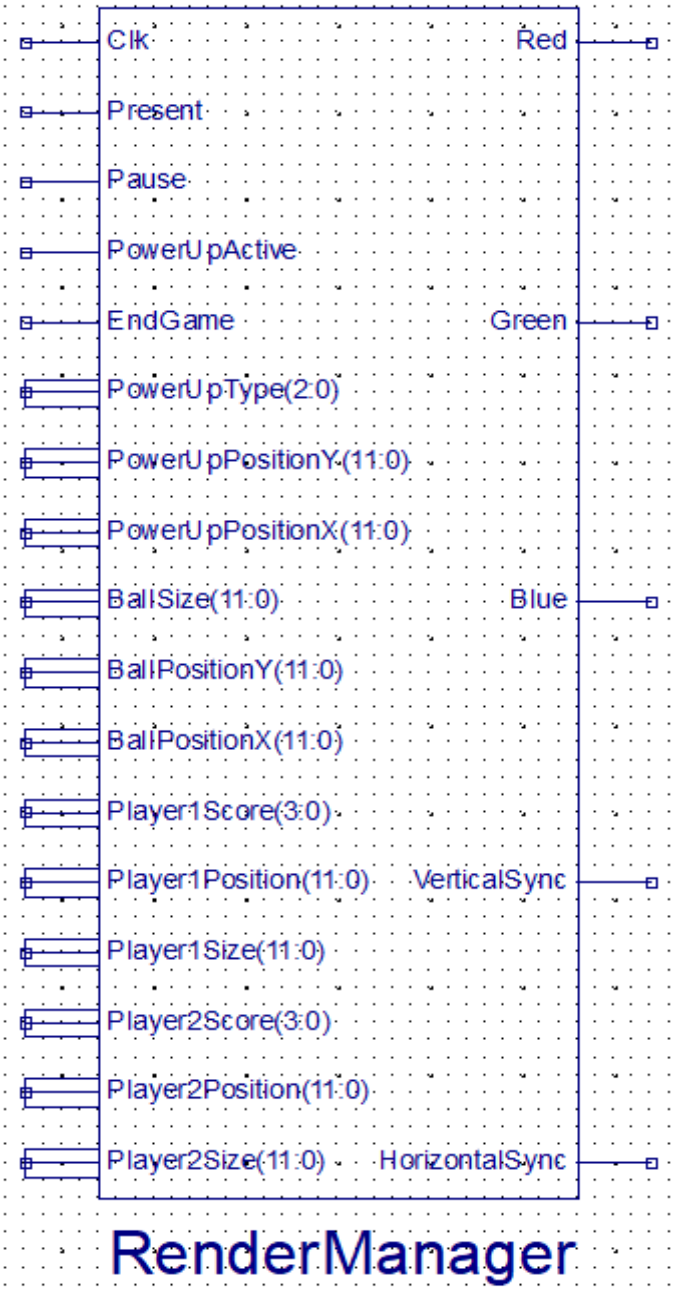
Moduł ten został zaimplementowany jako ostatni ze względu na to, że służy do konsumowania informacji na temat obiektów, które znajdują się w grze, a następnie wyświetlaniu ich w odpowiednich miejscach na ekranie. Moduł ten posiada proces odpowiadający za inicjalizację dwuwymiarowych wektorów, za pomocą których odbywa się pisanie do VGA. Jednak synteza takich buforów zajmowała kilkanaście godzin, z racji czego kod ten został wykomentowany. Główną częścią tego modułu jest proces odpowiadający za rozpoznawanie obiektów, jakie należy wyświetlić, a następnie tworzenie odpowiednich kształtów w wektorach pikseli w celu wysyłania ich i wizualizacji na ekranie. Odbywa się to za pomocą dwóch buforów, do których ładowane są wartości odpowiednie dla konkretnego kształtu do wyświetlenia (np. piłki czy paletki). Każdy z obiektów posiada indywidualny sposób odwzorowania na kształt do wyświetlenia.

Moduł ten przyjmuje sygnał Present informujący kiedy powinna zostać wyświetlona kolejna klatka obrazu. Pauza sygnalizuje zatrzymanie wyświetlania do czasy jej dezaktywacji. ActivePowerUp oznacza, czy dane ulepszenie powinno być wyświetlane ze względu na swój stan aktywności. Koniec gry sygnalizuje zakończenie wyświetlania obrazu. Trzy sygnały dotyczące PowerUp mówią, gdzie i jaki symbol ulepszenia powinien zostać wyświetlony na planszy. Trzy sygnały dotyczące piłki mówią o tym, gdzie i o jakiej wielkości wyświetlić piłkę na planszy. Kolejne sygnały wejściowe dają informację o tym, jakie wartości liczbowe punktów graczy, a także symbole ich paletek (zmienne zależnie od ich pozycji i rozmiaru) powinny zostać wyświetlone. Moduł ten produkuje wartości kolorów RGB w trzech sygnałach, które docelowo powinny zostać wysłane do złącza VGA w celu wyświetlenia. Sygnały VerticalSync i HorizontalSync są odpowiedzialne synchronizację wyświetlania w pionie i poziomie.

```
process(Clk) -- Render thread, frequency?
begin
    if rising_edge(Clk) then
        if vga_front_buffer = 0 then
            --Red <= vga_buffer0(y_out)(x_out)(2);
            --Green <= vga_buffer0(y_out)(x_out)(1);
            --Blue <= vga_buffer0(y_out)(x_out)(0);
        else
            --Red <= vga_buffer1(y_out)(x_out)(2);
            --Green <= vga_buffer1(y_out)(x_out)(1);
            --Blue <= vga_buffer1(y_out)(x_out)(0);
        end if;
        x_out <= x_out + 1;
        if x_out = 0 then
            y_out <= y_out + 1;
        end if;
    end if;
end process;

process(Clk, Present, y_out) -- Present thread
begin
    if rising_edge(Clk) and Present = '1' then
        if y_out = 0 then
            if vga_front_buffer = 0 then
                --vga_buffer0 <= (others => (others => VGA_Black));
            else
                --vga_buffer1 <= (others => (others => VGA_Black));
            end if;
            vga_front_buffer <= vga_front_buffer + 1;
        end if;
    end if;
end process;
```

```
    -- Write thread
    process(Clk, Player1Score, Player2Score, Player1Position, Player1Size,
Player2Position, Player2Size, PowerUpActive, PowerUpType, BallPositionY,
BallPositionX, BallSize, Pause, EndGame)
    begin
        if rising_edge(Clk) then
            -- Clear buffer
            if vga_front_buffer = 0 then
                --vga_buffer1 <= (others => (others => VGA_Black));
            else
                --vga_buffer0 <= (others => (others => VGA_Black));
            end if;
            -- Write scores (max 9)
            case Player1Score is
                when "0000" =>
                        if vga_front_buffer = 0 then
                            for i in 1 to 50 loop
                                --vga_buffer1(to_integer(250 +
i))(to_integer(50)) <= VGA_Green;
                                --vga_buffer1(to_integer(250 +
i))(to_integer(150)) <= VGA_Green;
                            end loop;
                            for i in 1 to 100 loop
                                --
vga_buffer1(to_integer(250))(to_integer(50 + i)) <= VGA_Green;
                                --
vga_buffer1(to_integer(250))(to_integer(50 + i)) <= VGA_Green;
                            end loop;
                        else
                            for i in 1 to 50 loop
                                --vga_buffer0(to_integer(250 +
i))(to_integer(50)) <= VGA_Green;
                                --vga_buffer0(to_integer(250 +
i))(to_integer(150)) <= VGA_Green;
                            end loop;
                            for i in 1 to 100 loop
                                --
vga_buffer0(to_integer(250))(to_integer(50 + i)) <= VGA_Green;
                                --
vga_buffer0(to_integer(300))(to_integer(50 + i)) <= VGA_Green;
                            end loop;
                        end if;
                when "0001" =>
```

Dalsza część modułu wygląda analogicznie (wyświetlanie kolejnych znaków bazując na weryfikacji występującego przypadku.



Ball

Moduł Ball posiada proces zarządzający prędkością poruszania się i rozmiarem piłki. Operuje on na sygnale Reset, w przypadku którego wystąpienia następuje zresetowanie tych dwóch wartości do wartości domyślnych. W przypadku aktywności któregoś z sygnału odpowiadającego za modyfikację prędkości czy rozmiaru piłki następuje obliczenie nowej wartości tej wielkości. Drugi proces odpowiada za zmianę pozycji piłki. W przypadku resetu zostaje ona ustawiona na wartość domyślną. W przypadku nastąpienia zmiany pozycji piłki jest ona obliczana na podstawie wektora kierunku przesunięcia piłki, który może przyjmować cztery wartości zależnie od kierunku.

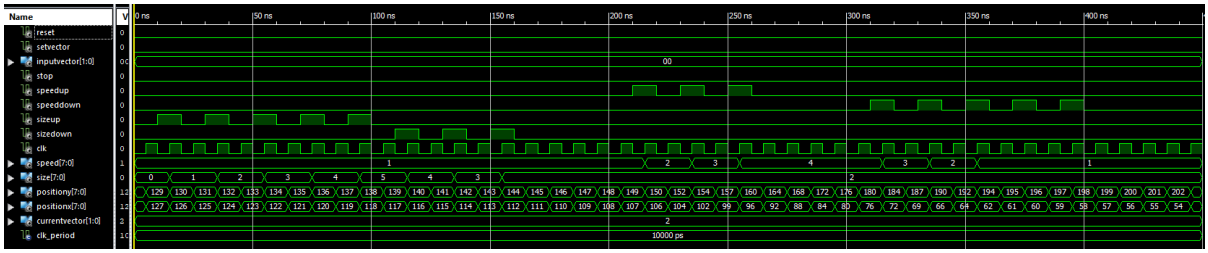
Moduł ten odbiera sygnał resetu, który informuje o konieczności przywrócenia piłki do pozycji początkowej. Sygnał SetVector mówi o tym, kiedy powinien zostać skonsumowany kolejny nowy wektor poruszania się piłki. Sygnał stopu mówi o pauzie w grze, podczas której piłka powinna przestać się poruszać. Pozostałe sygnały wejściowe odpowiadają zmianom parametrów piłki, czyli zmianą jej wielkości lub szybkości poruszania się. Wynikiem pracy tego modułu jest wartość prędkości piłki, która

jest wysyłana do CollisionManager w celu obliczeń związanych z kolizją. W tym samym celu wysyłane są sygnały informujące o rozmiarze i pozycji piłki, jednak to dodatkowo są wysyłane jeszcze do RenderManager w celu wyświetlania na ekranie. Obecny wektor poruszania się piłki jest wysyłany do CollisionManager, ponieważ jest tam potrzebna informacja o tym, w którą stronę piłka się porusza.

Procesy w module:

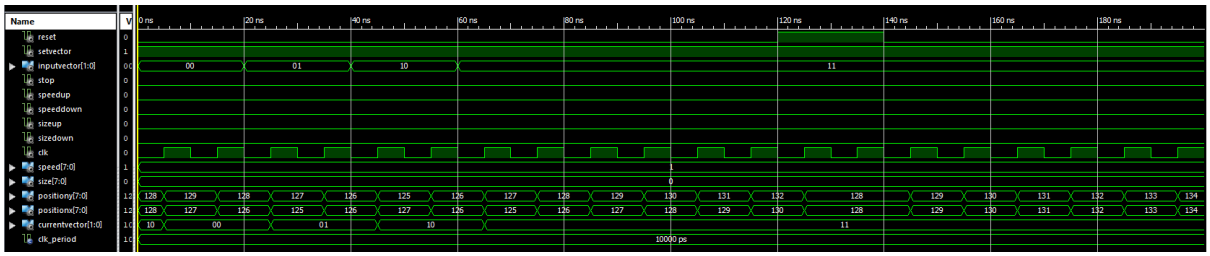
```
process(Clk, Reset, SpeedUp, SpeedDown, SizeUp, SizeDown)
    begin
        if rising_edge(Clk) then
            if Reset = '1' then
                speed_int <= X"001";
                size_int <= X"000";
            else
                if SpeedUp = '1' and speed_int < max_speed then
                    speed_int <= speed_int + 1;
                end if;
                if SpeedDown = '1' and speed_int > X"001" then
                    speed_int <= speed_int - 1;
                end if;
                if SizeUp = '1' and size_int < max_size then
                    size_int <= size_int + 1;
                end if;
                if SizeDown = '1' and size_int > X"000" then
                    size_int <= size_int - 1;
                end if;
            end if;
        end if;
    end process;

    process(Clk, Reset, SetVector)
    begin
        if rising_edge(Clk) then
            if Reset = '1' or SetVector = '1' then
                moveVector <= InputVector;
            end if;
            if Reset = '1' then
                positionY_int <= X"140";
                positionX_int <= X"140";
            elsif Stop = '0' then
                case moveVector is
                    when "00" =>
                        positionY_int <= positionY_int - speed_int;
                        positionX_int <= positionX_int - speed_int;
                    when "10" =>
                        positionY_int <= positionY_int + speed_int;
                        positionX_int <= positionX_int - speed_int;
                    when "01" =>
                        positionY_int <= positionY_int - speed_int;
                        positionX_int <= positionX_int + speed_int;
                    when "11" =>
                        positionY_int <= positionY_int + speed_int;
                        positionX_int <= positionX_int + speed_int;
                    when others =>
                        positionY_int <= X"140";
                        positionX_int <= X"140";
                end case;
            end if;
        end if;
    end process;
```

Zmienianie się wartości pozycji piłki jest zależne od jej aktualnej pozycji, rozmiaru i prędkości. Piłka posiadając ciągle ten sam wektor poruszania się może zmieniać swoją pozycję w różny sposób zależnie od wartości prędkości czy rozmiaru:

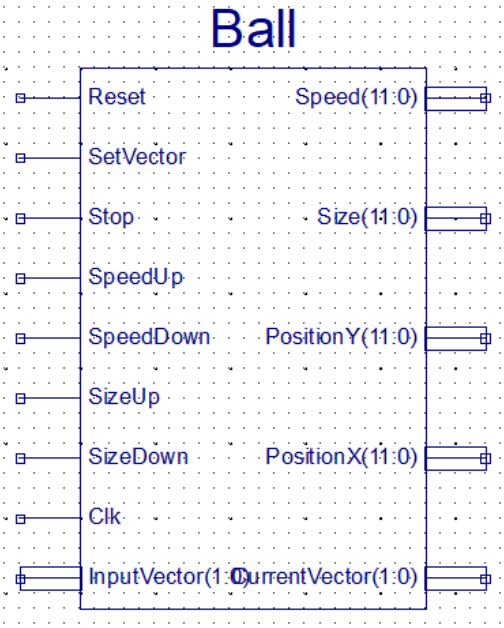


10 Ball_shouldChangeSpeedAndSize

W przypadku zmiany wartości wektora docelowego poruszenia się piłka powinna zaczynać poruszać się w kierunku zdefiniowanym przez aktualny wektor poruszania się:



11 Ball_shouldMoveByVector



GameLogic

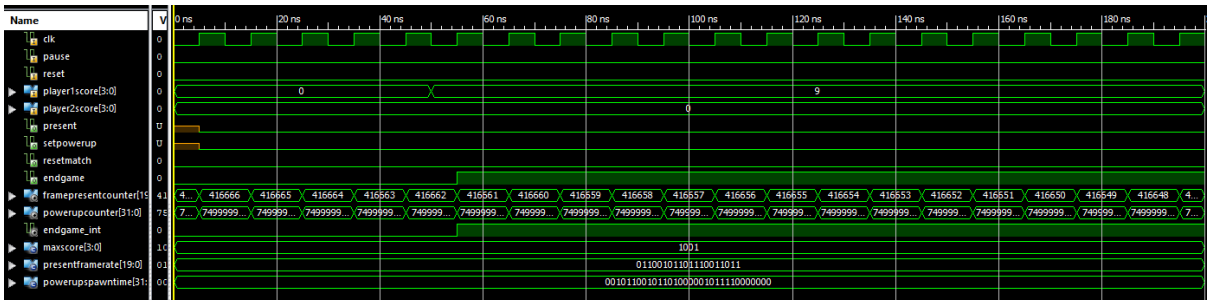
Ten moduł odpowiada za procesy związane z warunkami gry abstrahując od mechaniki jej działania. Pierwszy proces odpowiada za sprawdzanie warunku końca gry, którym jest zdobycie maksymalnej wartości punktów przez którego z graczy. Drugi proces odpowiada za ulepszenia podczas wystąpienia resetu lub pauzy w trakcie rozgrywki. Dla resetu licznik czasu stworzenia ulepszenia jest resetowany, dla pauzy jest zatrzymywany, a normalnie zmniejsza się o jedną jednostkę. Ostatni proces zarządza obliczaniem momentu, w którym powinna zostać wyświetlona kolejna klatka prezentująca obraz gry. Działa on jedynie na podstawie sygnału zegarowego.

Moduł ten otrzymuje sygnały pauzy i resetu, aby przywracać wartości liczbowe liczników tworzenia ulepszeń do wartości początkowych lub zatrzymywania ich zwiększania się. Dodatkowo otrzymuje on sygnały symbolizujące liczbę punktów graczy, aby móc decydować o zakończeniu rozgrywki. GameLogic wysyła sygnał oznaczający zakończenie gry do RenderManager, aby poinformował on o zakończeniu rozgrywki poprzez wyświetlenie napisu. Sygnał resetu rozgrywki jest wysyłany do CollisionManager, aby zresetować wartości szybkości poruszania, rozmiarów i pozycji gracza i piłki. Sygnał ustawienia ulepszenia jest wysyłany w momencie osiągnięcia odpowiedniej wartości przez licznik tworzenia ulepszeń, aby dane ulepszenie mogło zostać stworzone. Sygnał Present jest odpowiedzialny za wysyłanie informacji do RenderManager o tym, kiedy powinna zostać wyświeltona kolejna klatka obrazu.

Procesy w module:

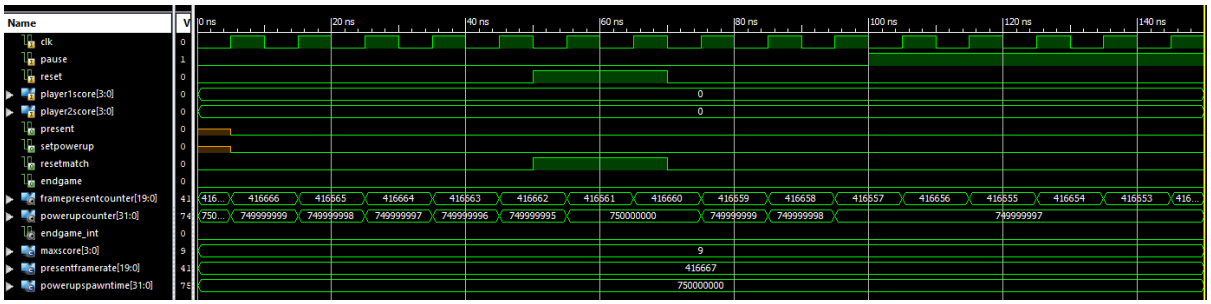
```
process (Clk, Reset, Player1Score, Player2Score)
    begin
        if rising_edge(Clk) then
            if Player1Score = maxScore or Player2Score = maxScore then
                endGame_int <= '1' and not Reset;
            else
                endGame_int <= '0';
            end if;
        end if;
    end process;

    process (Clk, Pause, Reset)
    begin
        if rising_edge(Clk) then
            if Reset = '1' then
                powerUpCounter <= powerUpSpawnTime;
                SetPowerUp <= '0';
            elsif Pause = '0' then
                if powerUpCounter = 0 then
                    powerUpCounter <= powerUpSpawnTime;
                    SetPowerUp <= '1';
                else
                    powerUpCounter <= powerUpCounter - 1;
                    SetPowerUp <= '0';
                end if;
            end if;
        end if;
    end process;

    process (Clk)
    begin
        if rising_edge(Clk) then
            if framePresentCounter = 0 then
                framePresentCounter <= presentFrameRate;
                Present <= '1';
            else
                framePresentCounter <= framePresentCounter - 1;
                Present <= '0';
            end if;
        end if;
    end process;
```

Moduł ten powinien reagować na zmianę wartości punktów graczy i kończyć grę w momencie osiągnięcia przez któregoś z nich maksymalnej liczby punktów:

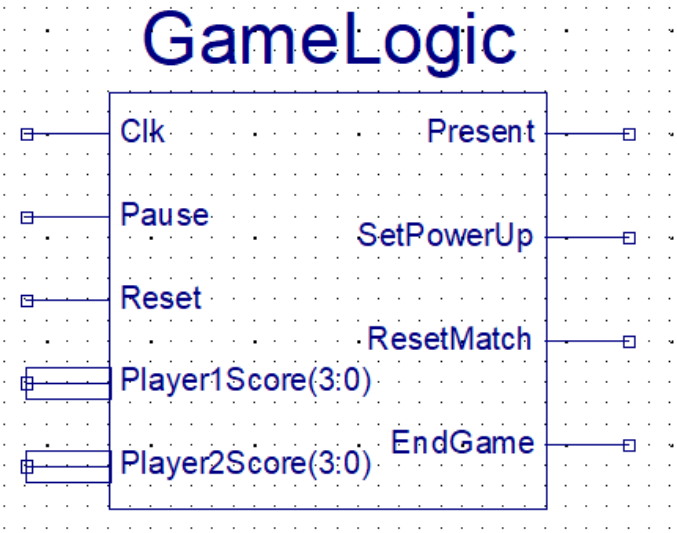


12 GameLogic_shouldEndGame

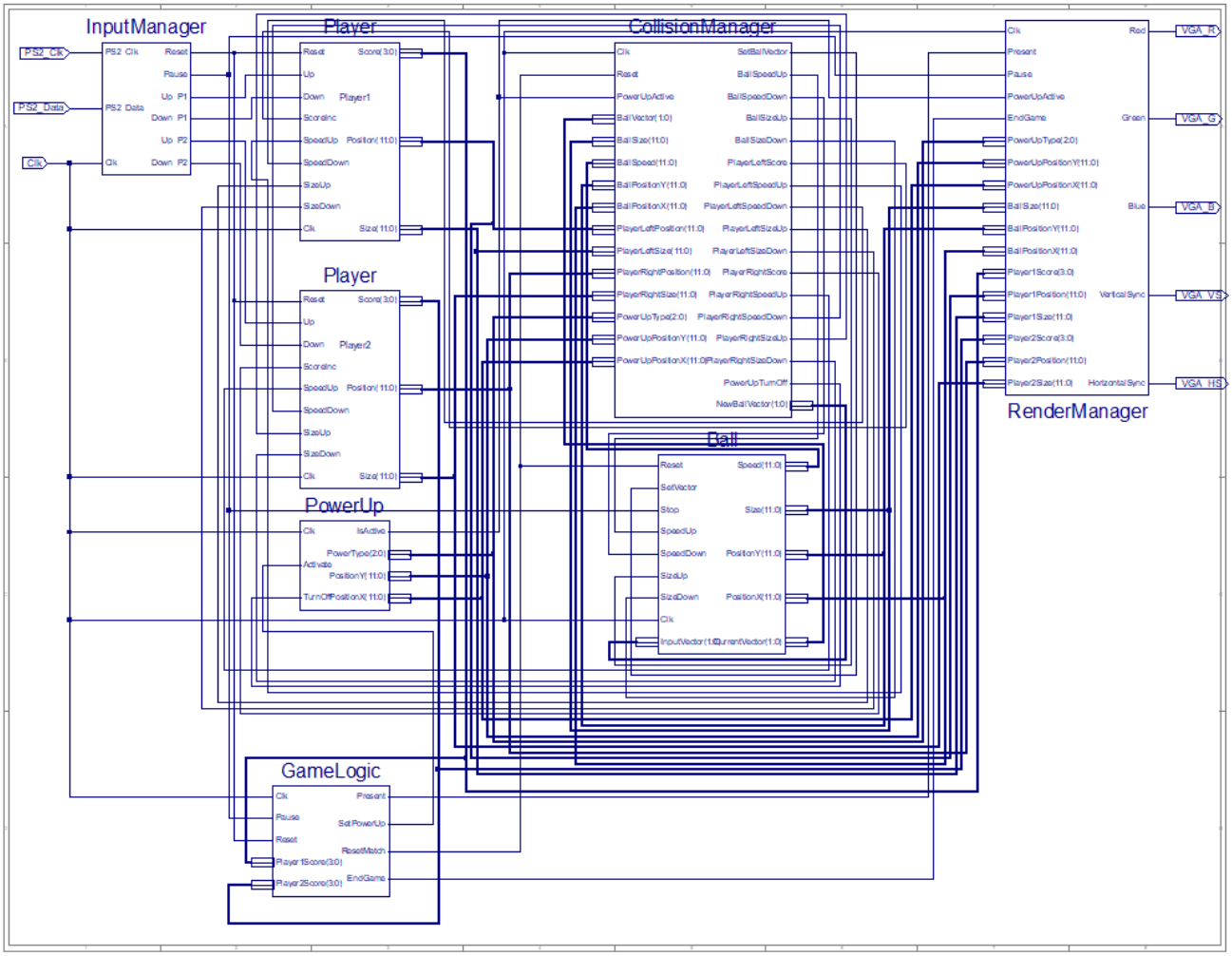
W przypadku resetu wartość licznika czasu do pojawienia się ulepszenia powinna zostać przywrócona do wartości domyślnej, a dla pauzy powinna zostać „zamrożona” wraz z licznikiem czasu wysyłania kolejnej klatki obrazu do ekranu podczas całego trwania pauzy:



13 GameLogic_shouldResetGame



Schemat całego projektu:



Podsumowanie

Z perspektywy użytkownika korzystającego z urządzenia z działającym projektem na ekranie powinny wyświetlać się następujące obiekty: paletki dwóch graczy po prawej i lewej stronie ekranu, piłka pomiędzy graczami, liczniki punktów dla każdego z graczy, a także litery symbolizujące różne rodzaje ulepszeń pojawiające się na mapie. Użytkownicy powinny móc poruszać swoimi paletkami w pionie za pomocą czterech klawiszy: strzałka w górę i w dół oraz klawisze W i S. Dodatkowo naciśnięcie R powinno resetować rozgrywkę, czyli przywracać stan początkowy gry, a naciśnięcie klawisza P powinno skutkować zamrożeniem rozgrywki do czasu ponownego naciśnięcia P. Po rozpoczęciu rozgrywki piłka powinna odbijać się od paletek graczy i ścian ekranu, a w przypadku wpadnięcia w ścianę za którymś z graczy punktacja wyświetlana dla przeciwnika powinna wyświetlić liczbę o 1 większą.

Obecnie wszystkie moduły oprócz RenderManager posiadają testy, na których zostało sprawdzone zachowanie poszczególnych komponentów. Odbyło się to za pomocą analizy wartości sygnałów podczas symulacji behawioralnej. Dla modułu RenderManager nie zostały stworzone testy, ponieważ służy on jedynie do wyświetlania obrazu na ekranie. W celach dodatkowych projektu znalazło się odtwarzanie dźwięków podczas kluczowych wydarzeń gry (np. zdobycie punktu, zebranie ulepszenia), jednak nie zostało ono zaimplementowane ze względu na brak dostępu do fizycznego sprzętu, na którym mogłoby to zostać przetestowane, a także ze względu na stosunkowo dużą

złożoność projektu składającego się z elementów, które nie zostały przetestowane fizycznie. Skutkiem tego był brak użycia wcześniej zaplanowanego modułu „WAVreader” do odtwarzania dźwięku. Poza tym pozostałe cele zostały zrealizowane. Cały projekt był tworzony z użyciem systemu kontroli wersji GIT, co dwukrotnie pozwoliło na łatwiejsze zidentyfikowanie problemu leżącego w ostatnich zmianach dodanych do kodu. Problematyczna okazała się synteza projektu, ponieważ w końcowej fazie implementacji zajmowała kilkanaście godzin, co wymusiło wykomentowanie linii RenderManagera powodujących znaczne wydłużenie syntezy (duży rozmiar bufora ekranowego). Pisanie testów równolegle z implementacją modułów okazało się przydatne ze względu na wczesne wykrywanie błędów, co prawdopodobnie zaoszczędziło kilku godzin poprawiania błędów dotyczących wielu modułów korzystających z wadliwego już modułu. Projekt nie został ostatecznie przeniesiony na zestaw Spartan3e ze względu na brak dostępu do laboratorium, w którym znajdowała się płytka.