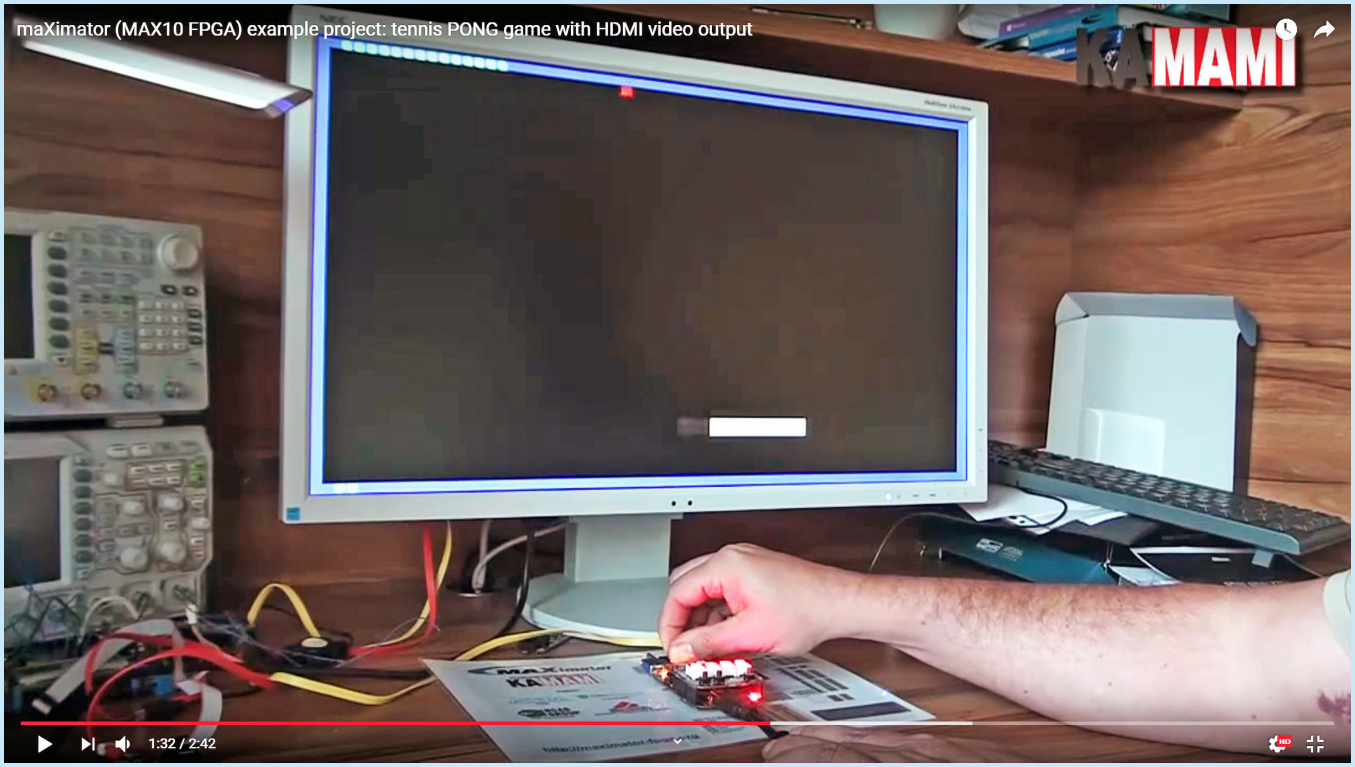


Jak to się robi w FPGA: gra PONG z wyjściem HDMI

Projektem, który opiszę w tym artykule jest implementacja wariacji gry Pong dla platformy maXimator z układem Altera (teraz już Intel) MAX10. Praca powstała niegdyś na potrzeby konkursu EP, którego celem było stworzenie projektu wykorzystującego licznik 74169. W ówczesnej odmianie gry pojawia się tylko jedna paletka i jest ona sterowana przez gracza, którą porusza wzdłuż dolnej krawędzi ekranu. Jeżeli uda mu się odbić nią piłkę i dotknie ona górnej krawędzi pola gry to gracz zyskuje punkt. W przypadku kiedy graczowi to zadanie nie uda się i piłka dotknie dolnej krawędzi ekranu punkt zdobywa komputer.

Czterobitowe liczniki wykorzystane w projekcie pozwalają na zliczanie punktów od zera do piętnastu i dlatego po przekroczeniu tej wartości przez któregoś z graczy punkty są zliczane ponownie od zera. Jak widać zasady gry zostały znacznie uproszczone w stosunku do oryginału, ale celem przyświecającym przy tworzeniu tego układu była nauka generowania sygnału HDMI i reszta logiki została potraktowana tylko pobieżnie. Każdy z czytelników może pobrać projekt ze strony *maximator-fpga.org* (można go znaleźć w dziale *Examples*) i samodzielnie rozwinąć logikę gry do czego zachęcam.

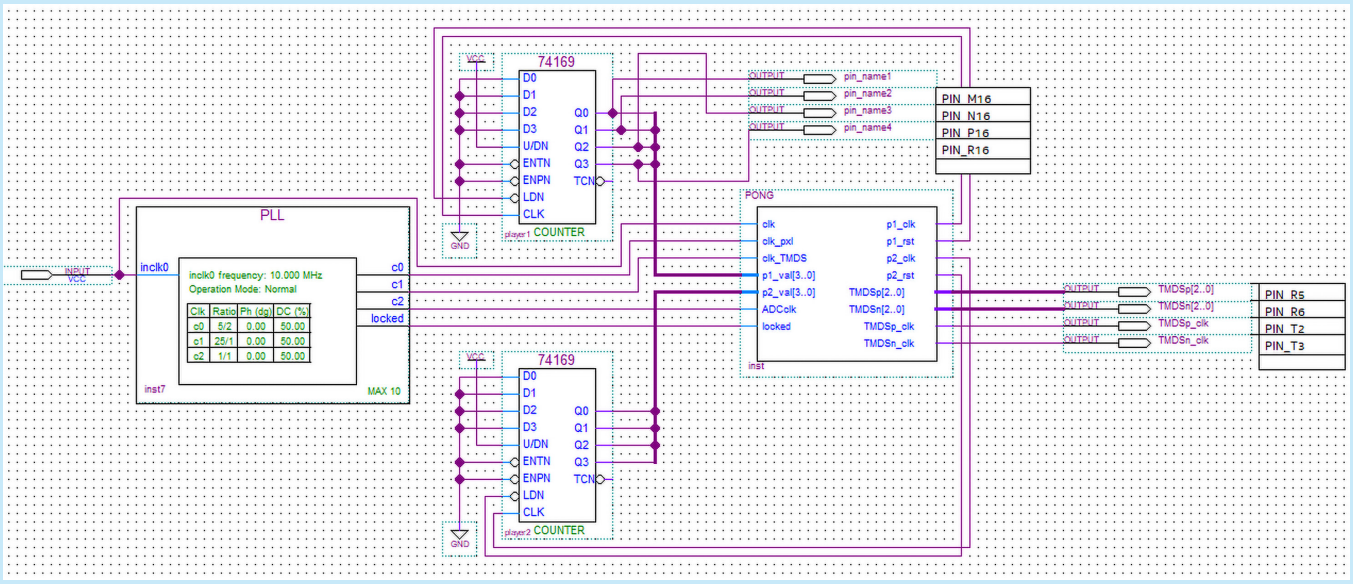
Top level

Najwyższa warstwa projektu (**rysunek 1**) została zaprojektowana w formie schematu połączeń. Cała główna część logiki układu jest wykonywana w jednostce PONG i składa się ona z dwóch zasadniczych części. Pierwsza z nich odpowiada za logikę gry i generowanie jej grafiki natomiast druga na podstawie danych z pierwszej części generuje sygnał HDMI. Całość kodu opisu sprzętu dla tego projektu została napisana w języku VHDL w środowisku Quartus Prime Lite Edition.

Elementem, który napędza pracę całej logiki układu jest konfigurowalna PLL zawarta we wnętrzu układu FPGA. Na jej wejście jest podany sygnał 10 MHz z oscylatora kwarcowego znajdującego się na płytce maXimator. Na jej wyjściu z kolei zostały uzyskane trzy sygnały zegarowe. Pierwsze wyjście oznaczone jako c0 generuje sygnał 25 MHz wykorzystywany jako zegar TMDS. Drugie oznaczone jako c1 pozwala na uzyskanie zegara 250 MHz wykorzystywanego jako zegar dla danych pikseli. Wykorzystanie tych dwóch sygnałów zegarowych jest szerzej opisane w części dotyczącej interfejsu HDMI. Trzeci z sygnałów zegarowych (c2) to 10 MHz wykorzystywane przez blok ADC. Ostatecznie generowanie tego sygnału okazało się nie być niezbędne ze względu na to, że nie różni się w częstotliwości od sygnału wejściowego i blok ADC mógłby zostać taktowany bezpośrednio z rezonatora kwarcowego jednak taka konfiguracja pozwala na pewne eksperymenty z blokiem przetwornika. Praca logiki gry jest napędzana sygnałem zegarowym o częstotliwości 10 MHz wprost z rezonatora.

Kompletny projekt dla bezpłatnego środowiska Quartus Prime można pobrać ze strony ***http://bit.ly/2IXW29k***. Pod adresem ***http://bit.ly/2GN4SUr*** można obejrzeć film prtezentujący działanie gry opisanej w artykule.

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie **www.media.avt.pl**



Rysunek 1. Schemat logiczny gry PONG

Gra PONG

Układ logiki gry jest sterowany wejściem zegarowym i danymi z przetwornika ADC odczytującym napięcie z potencjometru umieszczonego na płytce ewaluacyjnej. W wewnętrznych rejestrach stale przechowywany jest obecny stan gry, który pozwala na generowanie grafiki. Aby umożliwić rozgrywkę na grywalnym poziomie prędkości piłki zegar został spowolniony 50000-krotnie, a zatem stan gry jest odświeżany 200 razy na sekundę. Piłka przy każdym odświeżeniu jest przesuwana o jeden piksel w górę lub w dół i o jeden piksel w prawo lub w lewo co przy rozdzielczości 640×480 okazało się odpowiednią prędkością. W pętli głównej gry jednocześnie są aktualizowane stany wszystkich obiektów w grze. Użycie określenie pętla główna w kontekście opisu układu synchronicznego może okazać się dyskusyjne, ale nie wchodząc w polemikę z samym sobą w dalszej części wykorzystam właśnie to wyrażenie.

Aby umożliwić określenie aktualnej pozycji paletki przy każdym odświeżeniu stanu gry zostaje odczytan pomiar z przetwornika ADC. Wartość ta jest 12-bitowa jednak taka duża dokładność zupełnie nie jest potrzebna dla tego projektu. Dodatkowo przy tak dokładnym pomiarze pojawiły się problemy z szumem nakładającym się na odczyt, a także mechanicznymi niedoskonałościami potencjometru w związku z czym paletka drgała nawet wtedy kiedy gracz nie dotykał pokrętła. Aby poradzić sobie z tą przypadłością do określania pozycji paletki zostały wykorzystane tylko 5 najstarszych bitów wartości odczytanej z przetwornika ADC. Taka rozdzielczość pozwala uzyskać zadowalającą dokładność przesuwu paletki i niweluje wszystkie wspomniane wcześniej problemy. Ze względu na możliwość poruszania paletką wyłącznie w jednej osi jej umiejscowienie jest przechowywane w jednym rejestrze: paddleX.

Przy każdym kroku głównej pętli zostaje określona nowa pozycja piłki i zapisana jako współrzędne X i Y w rejestrach ballX oraz ballY. Celem uproszczenia logiki gry piłka może poruszać się jedynie pod kątem 45 stopni. Jej kierunek jest zapisany na dwóch bitach dzięki czemu wiadomo w jakie miejsce powinna się przesunąć przy kolejnej iteracji. Dodatkowo cyklicznie są sprawdzane jej ewentualne kolizje z paletką oraz ramkami pola gry. W przypadku odbicia od paletki lub bocznej krawędzi pola gry zostaje zmieniony kierunek, w którym będzie podążać piłka. Jeżeli zetknęła ona się z dolną lub górną krawędzią ramki zostaje wygenerowany impuls dla licznika punktów pozwalający na zaktualizowanie ich ilości dla odpowiedniego gracza (w przypadku dolnej krawędzi punkt zdobywa gracz komputerowy, w przypadku górnej krawędzi punkt zostaje doliczony graczowi).

Na tym etapie są już zaktualizowane wartości pozycji piłki oraz pozycja paletki, a z liczników mogą zostać odczytane wartości punktów więc posiadamy już wszystkie informacje potrzebne do wygenerowania grafiki. Ze względów na prostotę całego projektu nie było potrzebne przygotowywanie bufora ramki obrazu, a wartości koloru wszystkich pikseli na ekranie są obliczane w sposób kombinacyjny, który można podzielić na dwa etapy. W pierwszym etapie zostaje określone jaki obiekt należy wyświetlić na konkretnym pikselu. W tym celu wykorzystując dwa liczniki modułu generatora sygnału HDMI przechowywujące numer obecnie wyświetlanego piksela (counterX oraz counterY) możliwe było wygenerowanie odpowiednich sygnałów. Dla przykładu ramka obszaru gry jest generowana kodem:

```
border <= '1' when counterX < 10 OR counterX >= 629
OR counterY < 10 OR counterY >= 469 else '0';
```

czyli na wyjściu border pojawia się logiczna jedynka kiedy licznik znajduje się w przedziale 0-9 lub 630-639 dla osi X oraz 0-9 lub 470-479 dla osi Y.

W bardzo podobny sposób uzyskiwany jest obraz piłki o wielkości 10×10 pikseli:

```
ball <= '1' when counterX >= ballX AND counterX <
ballX + 10 AND counterY >= ballY AND counterY <
ballY + 10 else '0';
```

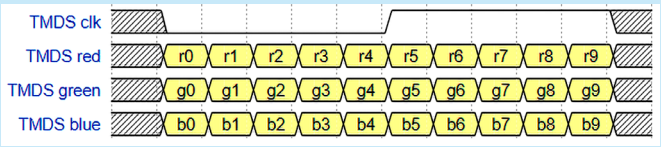
oraz obraz paletki o wielkości 100×20 pikseli:

```
paddle <= '1' when counterX >= paddleX AND counterX
< paddleX + 100 AND counterY >= 400 AND counterY <
420 else '0';
```

Kod robi się odrobinę bardziej złożony, ale z całą pewnością nie bardziej skomplikowany, w przypadku liczników punktów. Jako, że każdy punkt jest wyświetlany za pomocą osobnego kwadracika, a liczone są punkty dwóch graczy to linii podobnych do przedstawionych wcześniej musiało się znaleźć już 30. Kod ten mógłby zostać z powodzeniem zastąpiony wyrażeniem *generate*, co sugeruję przetestować osobom chcących sprawdzić swoje umiejętności opisu w języku VHDL.

Ostatnią częścią generowania grafiki gry jest określenie kolorów jej elementów. W tym momencie należy wygenerować składowe RGB obecnie przetwarzanego piksela. Obraz generowany w tym projekcie wykorzystuje 24 bity na zakodowanie koloru czyli po 8 bitów na każdą jego składową. Wykorzystywane w tym celu są trzy linie:

```
red <= (paddle OR ball) & (paddle OR ball) & border
& border & border & border & „11”;
green <= (paddle OR points1 OR points2) & (paddle
OR points1 OR points2) & border & border & border &
border & „11”;
```



Rysunek 2. Sposób kodowania piksela obrazu w HDMI

```
blue <= (paddle OR border) & (paddle OR points1 OR
points2) & border & border & border & border & „11”;
```

Taki zapis pozwolił w prosty sposób ustawiać bity kolorów dla odpowiednich elementów. Takie podejście jednak sprawdzi się tylko w przypadku niewielkich projektów jak na przykład ten.

W tym miejscu kończy się część, która obsługuje logikę gry. Cyklicznie wykonywana pętla pozwala na określenie stanu gry w danym momencie, a znany stan pozwala na określenie położenia i koloru każdego z pikseli. Mając te wszystkie informacje można przejść do kolejnej części jaką jest generowanie sygnału HDMI.

Interfejs HDMI

Aby uzyskać obraz na ekranie przekazując dane interfejsem HDMI należy wygenerować trzy sygnały TMDS, z których każdy zawiera dane dla osobnej składowej koloru (RGB) oraz sygnał zegara TMDS (**rysunek 2**). Interfejs TMDS (*Transition Minimized Differential Signaling*) koduje ośmiobitową wartość koloru piksela na dziesięciu bitach. Kodowanie to ma na celu zmniejszenie ilości tranzycji sygnałów oraz wyrównanie ilości jedynek i zer na liniach danych. Pozwala to na poprawienie własności elektromagnetycznych transmitowanego sygnału i umożliwia wykorzystanie połączeń kablowych gorszej jakości bez wpływu na jakość transmisji, a co za tym idzie obniżenie cen przewodów. Czynność, która z pozorów może wyglądać na skomplikowaną w rzeczywistości wcale taka nie jest o czym za chwilę każdy się przekona. W naszym przypadku zajmiemy się generowaniem obrazu o rozdzielczości 640×480 pikseli oraz częstotliwości odświeżania 60 Hz. Jako że przez interfejs HDMI mogą być przekazywane także inne informacje oprócz obrazu (np. informacje synchronizacji poziomej i pionowej lub dźwięk) w praktyce do odbiornika zostaje wysłana większa ramka (w naszym przypadku będzie to 800×525 pikseli – **rysunek 3**), a w miejscach gdzie nie jest przesyłany obraz zostają umieszczone dodatkowe informacje. Chcąc obliczyć częstotliwość zegara TMDS należy pomnożyć ilość pikseli przez częstotliwość odświeżania tj. 800×525×60 Hz = 25200000 Hz. Łatwy do uzyskania sygnał 25 MHz jest wystarczająco bliski potrzebnej częstotliwości i nie powinien przysporzyć trudności w odbiorze większości monitorom dlatego właśnie nim się posłużymy. Jako, że dane pojedynczych kolorów składowych pikseli są 10 bitowe należy je wysyłać z częstotliwością 250 MHz.

Sercem naszego transmittera HDMI są kodery TMDS, które przetwarzają ośmiobitowe wartości koloru piksela na dziesięciobitowe zakodowane słowa. W specyfikacji HDMI można odnaleźć dokładny opis algorytmu kodowania więc najlepszym pomysłem jest zaczerpnięcie wiedzy z tego właśnie źródła. Dla osób mniej cierpliwych poniżej znajduje się bardzo uproszczony opis tej czynności nie tłumaczący w sposób wyczerpujący każdej operacji, ale pozwalający na szybką implementację działającego kodera.

Pełny algorytm prezentuje **rysunek 4**. W trakcie kodowania dwukrotnie należy zliczyć ilość jedynek w słowie więc na tę okazję została napisana dedykowana funkcja, która znacznie upraszcza kod kodera:

```
function count_ones(s : STD_LOGIC_VECTOR) return
INTEGER is
    variable ones : NATURAL := 0;
begin
    for i in s’range loop
        if s(i) = ‘1’ then ones := ones + 1;
        end if;
    end loop;
    return ones;
end function count_ones;
```

W pierwszym kroku należy policzyć liczbę jedynek w słowie wejściowym data_in i zapisać ją do sygnału ones_din:

```
ones_din <= count_ones(data_in);
```

Jeżeli liczba ta jest większa od czterech lub równa czterem, a jednocześnie bit na pierwszej pozycji jest równy zero to należy wykonać odpowiednią operację XOR na bitach słowa wejściowego. Jeżeli warunki te nie zostały spełnione należy wykonać operacje XNOR. W sygnale xored została oznaczona konieczność wykonania operacji XOR:

```
xored <= ‘1’ when (ones_din > 4) OR ((ones_din = 4)
AND (data_in(0) = ‘0’)) else ‘0’;
```

Na tym etapie możemy już wykonać odpowiednią operację i dołożyć do ośmiu bitów danych dziewiąty bit symbolizujący, która operacja została wykonane. Jako że XNOR jest negacją wyniku XORa to dla uproszczenia kodu w przypadku konieczności wykonania operacji XNOR odpowiednie bity zostają XORowane z jedynką. Poniższa linia umożliwiła wykonanie tego kroku:

```
qm <= NOT(xored) & (qm(6 downto 0) XOR data_in(7
downto 1) XOR (6 downto 0 => xored)) & data_in(0);
```

W drugiej części algorytmu należy zbadać ilość jedynek i zer i wykonać odpowiednie działania w celu zachowania równowagi poziomów w sygnale. Na początku przesyłania każdej poziomej linii różnica zostaje wyzerowana:

```
disparity <= 0;
```

Na początku należy ustalić czy mamy do czynienia z przewagą, jednej z wartości. Od ilości jedynek jest odjęte 4 tak aby znak wartości ones_qm: wskazywał, z którym poziomem bitów wartości jest więcej:

```
ones_qm <= count_ones(qm) - 4;
balance_sgn <= ‘1’ when (ones_qm >= 0 AND disparity
>= 0) OR (ones_qm < 0 AND disparity < 0) else ‘0’;
```

Znając rezultat możemy określić czy należy zanegować bity danych:

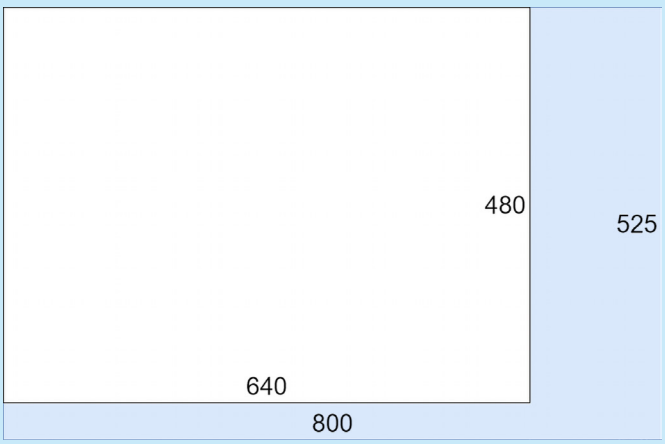
```
inv_qm <= NOT(qm(8)) when ones_qm = 0 OR disparity
= 0 else balance_sgn;
```

Teraz pozostaje nam już tylko stworzyć słowo wyjściowe dołączając do niego informację o negacji danych na pozycji najstarszego bitu i wykonaniu ewentualnej negacji:

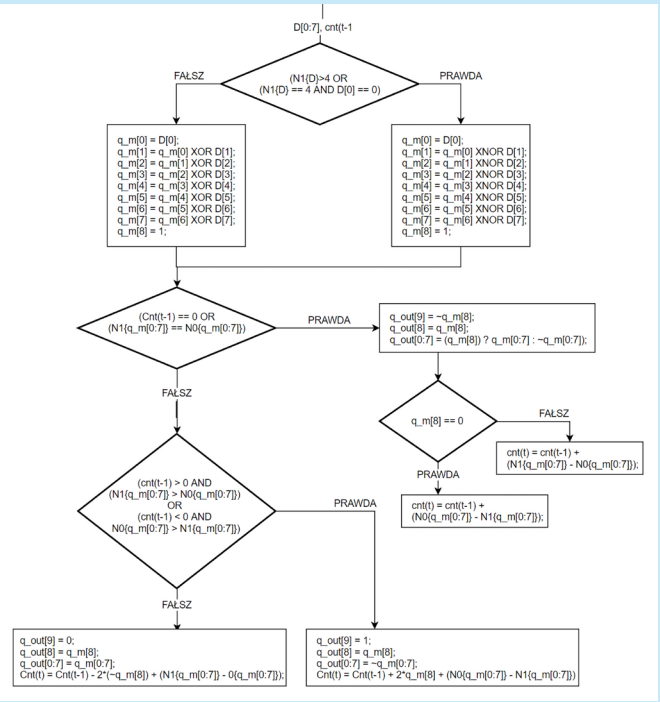
```
TMDS_data <= inv_qm & qm(8) & (qm(7 downto 0) XOR (7
downto 0 => inv_qm));
```

I na koniec trzeba zaktualizować wartości sygnałów zliczających nierówność przesyłanych zer i jedynek:

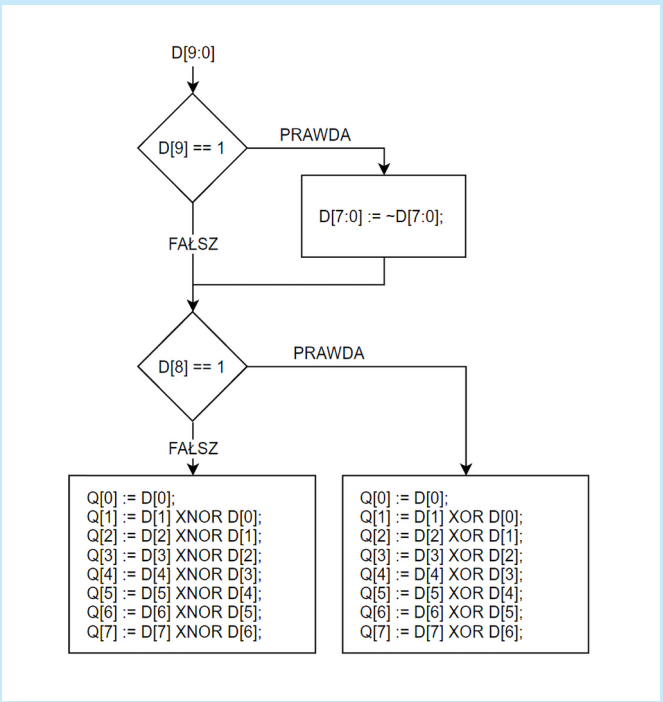
```
qm_var <= ‘0’ when (ones_qm = 0) OR (disparity = 0)
else ‘1’;
sgn <= 1 when (qm(8) XOR NOT(balance_sgn)) AND qm_
var else 0;
balance_acci <= ones_qm - sgn;
```



Rysunek 3. Pole gry na ekranie monitora



Rysunek 4. Algorytm kodowania TDMS



Rysunek 5. Algorytm dekodowania TDMS

Opis symboli na rysunkach 4 i 5:	
D	Ośmiobitowe dane wejściowe enkodera
Cnt	Rejestr używany do przechowywania nierówności ilości wysłanych jedynek i zer. Dodatnia wartość reprezentuje nadmiar jedynek, negatywna wartość oznacza nadmiar zer. Wyrażenie cnt{t-1} oznacza wartość wyliczoną w poprzednim kroku, a wyrażenie cnt{t} reprezentuje wartość dla obecnie kodowanych danych
q_m	Wartość pośrednia kodowanych danych
q_out	Dziesięciobitowa wartość wyjściowa zakodowanego słowa
N1{x}	Operator zwracający liczbę jedynek w słowie 'x'
N0{x}	Operator zwracający liczbę zer w słowie 'x'

```
balance_accn <= disparity - balance_acci when inv_
qm = '1' else disparity + balance_acci;
disparity <= balance_accn;
```

Co 10 taktów zegara 250 MHz zostaje do koderów wczytana ośmiobitowa wartość koloru piksela, jest ona kodowana, a następnie odczytana i zapisana do rejestrów przesuwnych, które co jeden takt ustawiają na najmłodszej pozycji kolejne bity do wysłania.

```
TMDS_ShiftR <= '0' & TMDS_ShiftR(9 downto 1);
TMDS_ShiftG <= '0' & TMDS_ShiftG(9 downto 1);
TMDS_ShiftB <= '0' & TMDS_ShiftB(9 downto 1);
```

Dodatkowo na kanale przesyłającym dane niebieskich składowych barw w obszarze pikseli niewidocznych na ekranie zostają przesyłane specjalne słowa kodowe pozwalające na synchronizację pionową i poziomą obrazu. **Tabela 1** przedstawia te słowa. Ich dobór nie jest przypadkowy – składają się z możliwie wielu przejść sygnału z zera na jeden i na odwrót. Umożliwia to łatwą synchronizację zegarów w nadajniku i odbiorniku. Podczas nadawania danych pikseli w kolumnach od 656 do 751 zostaje ustawiony bit C0, natomiast w wierszach od 490 do 491 zostaje ustawiony bit C1.

W interfejsie HDMI dane są przesyłane różnicowo i dlatego na wyjściu należy generować otrzymane wartości i ich zanegowane odpowiedniki:

```
TMDSp(2) <= TMDS_ShiftR(0);
TMDSn(2) <= NOT(TMDS_ShiftR(0));
TMDSp(1) <= TMDS_ShiftG(0);
TMDSn(1) <= NOT(TMDS_ShiftG(0));
TMDSp(0) <= TMDS_ShiftB(0);
TMDSn(0) <= NOT(TMDS_ShiftB(0));
TMDSp_clk <= clk_pxl;
TMDSn_clk <= NOT(clk_pxl);
```

Bit kontrolny		Słowo kodowe
C0	C1	9...0
0	0	1101010100
0	1	1101010101
1	0	0010101011
1	1	0010101010

Korzystając z okazji warto wspomnieć że dekodowanie sygnału TMDS jest znacznie prostsze niż kodowanie. Wystarczy wykonać dwie operacje. Sprawdzić najstarszy bit i w przypadku gdy jest on aktywny zanegować 8 bitów danych, a następnie sprawdzić drugi najstarszy bit i w zależności od jego stanu wykonać operację XOR lub XNOR na bitach danych. Algorytm ten przedstawiono na **rysunku 5**.

Zakończenie

Interfejs HDMI do niedawna był spotykany tylko w droższych modułach ewaluacyjnych lecz za sprawą Maximatora stał się dostępny dla znacznie szerszego grona odbiorców, a przede wszystkim również dla hobbystów. Mimo przeświadczenia wielu początkujących projektantów układów cyfrowych do skomplikowania tego standardu w rzeczywistości wcale taki nie jest. Projekt opisany w tym artykule jest stosunkowo prostym układem pokazującym w praktyczny sposób jak generować, a następnie przesyłać grafikę za pomocą złącza HDMI. Ze względu na małą objętość kodu nie potrzeba wiele czasu żeby zrozumieć istotę jego działania i można rozpocząć samodzielne rozwijanie własnych projektów w obsługę tego interfejsu.

Piotr Chodorowski