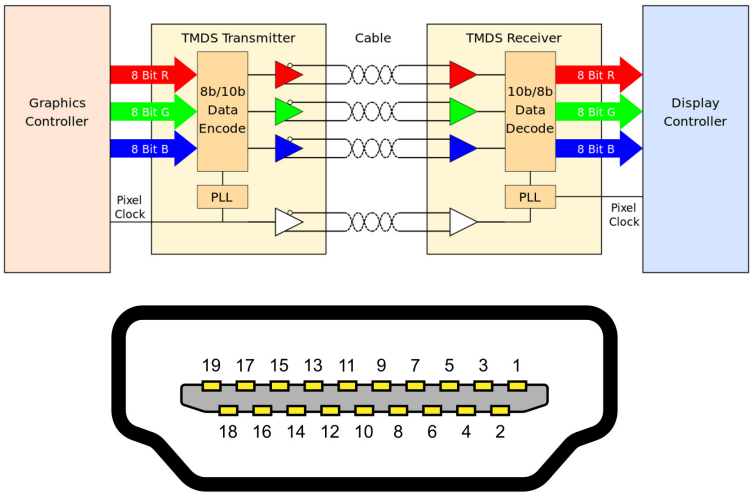


Autor artykułu składa serdeczne podziękowania Piotrowi Chodorowskiemu za zgodę na wykorzystanie kodu kodera TMDS oraz szkieletu układu generującego ramkę obrazu. Wkrótce na łamach EP ukaże się jego artykuł opisujący projekt gry PONG z wyświetlaniem obrazu za pomocą HDMI, w którym opisany zostanie bardziej szczegółowo koder TMDS. Co ciekawe, gra ta powstała bez wykorzystania systemu mikroprocesorowego – w oparciu o same układy logiczne – projekt gry można pobrać ze strony **https://www.maximator-fpga.org**.

NIOS II na maXimatorze, czyli mikroprocesor w układzie FPGA (14)

Interfejs HDMI, a więc koniec z sygnałami analogowymi

Zgodnie z obietnicą, podczas naszego przedostatniego spotkania zajmiemy się ujarzmieniem najbardziej skomplikowanego interfejsu, jakim dysponuje maXimator – mianowicie HDMI. Dzięki temu będziemy mogli generować obraz za pomocą interfejsu cyfrowego (i podłączyć sygnał do monitorów wyposażonych w złącza HDMI czy DVI). Rzecz jasna z rozdzielczością nie poszalejemy (z powodu ograniczonych zasobów i częstotliwości pracy), ale zdobędziemy bezcenne doświadczenie i nowe możliwości.

I tym razem zaczniemy naszą podróż do wyświetlenia tekstu na monitorze od przyjrzenia się interfejsowi, który wykorzystamy. Rozmieszczenie wyprowadzeń na tym złączu przedstawiono na **rysunku 1**, natomiast w **tabeli 1** opisano ich poszczególnych funkcje.

Transmisja różnicowa – o co chodzi?

Pierwszym zupełnie nowym pojęciem, które wymaga wyjaśnienia, jest pojęcie transmisji różnicowej. Taki rodzaj transmisji jest obecnie wykorzystywany praktycznie na każdym kroku – od interfejsu USB, poprzez interfejs SATA łączący np. dysk twardy z płytą główną, a na omawianym interfejsie HDMI skończywszy.

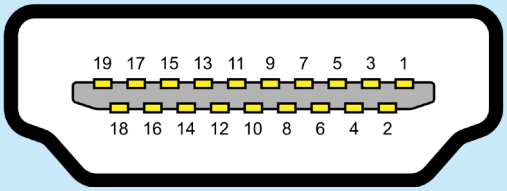
Transmisja różnicowa polega na zastosowaniu 2 przewodów do przekazania jednego sygnału w ten sposób, że badana jest, na przykład (tak jak w HDMI) różnica napięć (**rysunek 2**). W standardzie, którego będziemy używali, przyjęto, że jeśli napięcie na linii „plus" będzie występowało wyższe napięcie niż na linii „minus", to wtedy nadajemy logiczną jedynkę, a w przeciwnym wypadku – zero. Przewody te powinny być ze sobą skręcone i prowadzone dokładnie tą samą drogą. Dodatkowo, w wypadku HDMI każda taka para powinna mieć własny ekran.

Dlaczego jednak stosuje się taki rodzaj transmisji? Po pierwsze, dzięki temu, oba przewody biegną tą samą drogą i są wystawione na działanie takich samych zaburzeń. Jeśli badamy tylko różnicę

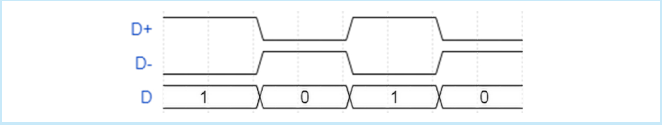
Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

napięć między tymi liniami, to te zaburzenia będą znacznie zminimalizowane (**rysunek 3**).

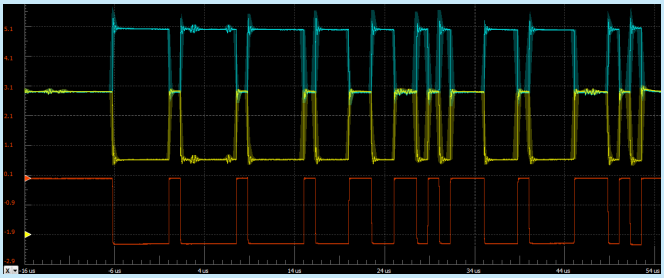
Drugi powód stosowania takiej transmisji to minimalizacja przesłuchów między kanałami powodowanych zjawiskami indukcyjnymi. Przypominając sobie lekcje fizyki każdy przewodnik, w którym płynie prąd: wytwarza wokół siebie pole magnetyczne, zaś zmienne pole magnetyczne powoduje z kolei indukcję siły elektromotorycznej w innym przewodniku, a więc dwa przewody ułożone obok siebie działają jak transformator. Podczas prowadzenia transmisji za pomocą prostych połączeń zmiany sygnału na jednym kanale będą zatem generowały zaburzenia na pozostałych. Jeśli zaś skorzystamy z pary różnicowej, to w dwóch sąsiadujących ze sobą przewodach prądy będą płynęły w przeciwnych kierunkach i pole magnetyczne pochodzące od takiej pary będzie zminimalizowane.



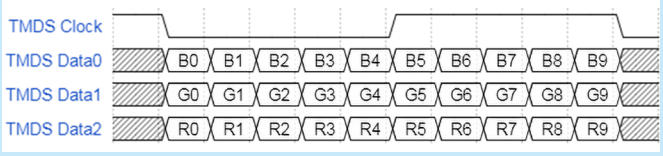
Rysunek 1. Układ wyprowadzeń złącza HDMI od strony gniazda (żeńskiej). Źródło: Wikipedia, użytkownik: Mobius



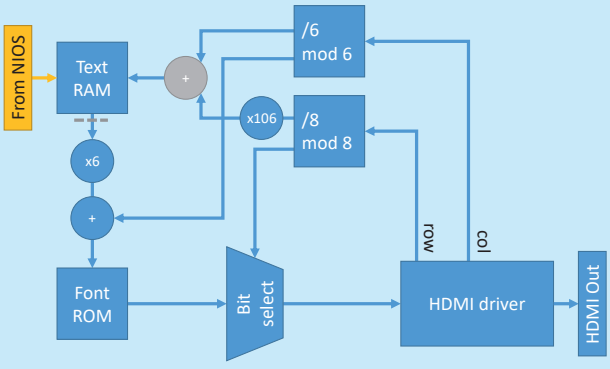
Rysunek 2. Zasada transmisji za pomocą pary różnicowej



Rysunek 3. Transmisja różnicowa na przykładzie magistrali CAN – widać, że zakłócenia na liniach transmisyjnych (żółta i niebieska) nie są obserwowane w sygnale różnicowym (czerwony)



Rysunek 4. Sygnał zegarowy oraz sygnały danych w transmisji TMDS



Rysunek 5. Schemat blokowy modułu wyświetlacza tekstowego z interfejsem HDMI

Oczywiście to nie wszystkie zalety takiej transmisji, ale myślę, że te w wystarczający sposób wyjaśniają, dlaczego to tak ważne zagadnienie.

Transmisja TMDS

Dane w interfejsie HDMI przesyłane są dla każdego koloru osobnym kanałem. Głębia koloru w standardzie HDMI 1.0 to 24 bity, czyli po 8 bitów na jeden kolor. W kanale koloru niebieskiego dodatkowo są kodowane 2 sygnały synchronizacyjne – synchronizacja pionowa i pozioma. Można więc powiedzieć, że HDMI to nic innego, jak przesłanie danych identycznych jak za pomocą interfejsu VGA tyle, że w postaci cyfrowej. Nawet sposób opisu zależności czasowych jest taki sam, jak dla VGA.

Dane są przesyłane w technologii TMDS (*Transition-Minimized Differential Signaling* – **rysunek 4**). Po pierwsze, zapewnia ona

Tabela 1. Przypisania sygnałów interfejsu HDMI do styków złącza w maXimatorze

Pin	Nazwa	Pełniona funkcja
1	TMDS Data2+	Transmisja różnicowa kanału koloru czerwonego oraz ekran dla tej pary różnicowej
2	TDMS Data2 S	
3	TMDS Data2–	
4	TMDS Data1+	Transmisja różnicowa kanału koloru zielonego oraz ekran dla tej pary różnicowej
5	TDMS Data1 S	
6	TMDS Data1–	
7	TMDS Data0+	Transmisja różnicowa kanału koloru niebieskiego wraz z zakodowanymi sygnałami synchronizacyjnymi oraz ekran dla tej pary różnicowej
8	TDMS Data0 S	
9	TMDS Data0–	
10	TMDS Clock+	Różnicowy sygnał zegarowy wraz z ekranem
11	TMDS Clock S	
12	TMDS Clock–	
13	CEC	Consumer Electronics Control – kanał do komunikacji dwukierunkowej pomiędzy urządzeniami służący do ich sterowania (np. włącz, wyłącz)
14	RES	Sygnał zarezerwowany (wg specyfikacji HDMI 1.0)
15	SCL	Interfejs I²C służący m.in. do identyfikacji podpiętego monitora (podobnie jak w VGA, z wykorzystaniem protokołu DDC)
16	SDA	
17	GND	Masa
18	+5V	Zasilanie 5 V, min. 55 mA, które podaje źródło sygnału HDMI
19	HPDET	Sygnał wykrywania podpięcia urządzenia – podpięcie urządzenia powinno zewrzeć go do 5 V

Tabela 2. Parametry IP core użytych w projekcie

Nazwa modułu	Typ modułu	Kluczowe parametry
charRAMHDMI	RAM: 2-PORT	one read port and one write port 6360 7-bit words separate read and write clocks no register at read output port
fontROMHDMI	ROM: 1-PORT	768 8-bit words no register at output port Mem Init: FONT.mif
multiplier6HDMI	LPM_MULT	dataa: 7 bit datab: 3 bit, value = 6
multiplier106HDMI	LPM_MULT	dataa: 9 bit datab: 8 bit, value = 106
dividerHDMI	LPM_DIVIDE	numerator: 12 bit denominator: 3 bit

odpowiednie kodowanie danych 8-bitowych za pomocą 10 bitów, dzięki któremu parametry przebiegu są tak dopasowane, aby zminimalizować zaburzenia (na przykład poprzez odpowiednią dystrybucję bitów, aby wyeliminować długie ciągi zer lub jedynek). Po drugie, za pomocą tegoż kodowania jest możliwe przesyłanie wspomnianych sygnałów synchronizacyjnych, sygnału braku obrazu (dla obszarów „niewyświetlanych"), a także dodatkowych danych, na przykład dźwięku. Te 10 bitów jest transmitowanych dla każdego cyklu sygnału taktującego – piksele są przesyłane szeregowo. Sygnał zegarowy (taktujący przesyłanie pikseli) także jest podawany różnicowo i ma częstotliwość równą częstotliwości nadawania pikseli, jest zatem 10-krotnie wolniejszy niż częstotliwość nadawania poszczególnych bitów w kodowaniu TMDS.

Sam sposób kodowania jest dosyć złożony, a osoby zainteresowane mogą rozpracować go na postawie dokumentacji. Ja zaś zdecydowałem się nie odkrywać koła na nowo i w tej materii wykorzystałem kod Piotra Chodorowskiego.

Oprócz powyższej wiedzy przyda nam się także przygotowany wcześniej moduł interfejsu VGA (opisany w EP 1/2019) – poza zamianą interfejsu przekazywania danych do monitora… cała logika generowania znaków pozostaje dokładnie taka sama! A więc do dzieła!

Implementacja HDMI

Warto, abyśmy zaczęli od sensownych i możliwych do zrealizowania założeń projektowych. Przede wszystkim, biorąc pod uwagę, że dane musimy wysyłać z układu FPGA z częstotliwością 10-krotnie większą niż częstotliwość pikseli, musimy rozsądnie ograniczyć tę częstotliwość.

Ja zdecydowałem się na wybór trybu 640×480 pikseli z częstotliwością odświeżania 60 Hz, co wraz z elementami synchronizacyjnymi daje nam częstotliwość pikseli 25,175 MHz. Tu jednak napotkamy drobny problem – taka częstotliwość okaże się niemożliwa do uzyskania na wyjściu pętli PLL przy 10 MHz częstotliwości wejściowej. Na szczęście jak pokazuje doświadczenie monitory radzą sobie z nieco zaniżoną częstotliwością i spokojnie możemy użyć częstotliwości 25 oraz 250 MHz, odpowiednio, dla zegara pikseli i „bitowego". Przy tych parametrach otrzymamy możliwość wyświetlenia 60 wierszy po 106 znaków (przy znaku o rozmiarach 6×8 pikseli).

Nasz moduł wyświetlania (**rysunek 5**) tworzymy na początku w 100% identycznie jak ten od VGA z tą różnicą, że musimy zwracać uwagę na inny niż poprzednio rozmiar obszaru wyświetlania, co wiąże się z tym, że rezygnujemy z dzielenia współrzędnych na 2, zmieniamy mnożenie z 85 na 106, a także zwiększamy szerokości bitowe niektórych sygnałów. Co oczywiste, zastosujemy też pamięć RAM o większej pojemności, dokładnie 6360 słów 7-bitowych. Rzecz jasna, zdefiniujemy też inne sygnały wejściowe i wyjściowe (**listing 1**). Następnie do projektu dodajmy enkoder TMDS zaprojektowany przez Piotra Chodorowskiego (**listing 2**).

Moduł ten ma wejście sygnału zegarowego powodującego zatrzaskiwanie danych na zboczu narastającym, wejście *vd_en*, które informuje, czy aktualnie nadajmy elementy obrazu, czy też jesteśmy poza obszarem wyświetlania, wejście *ctrl*, na którym podajemy kolejno sygnały synchronizacji pionowej i poziomej oraz oczywiście wejście 8-bitowych danych koloru i 10-bitowe wyjście udostępniające odpowiednie słowo kodowe.

Zostało jeszcze utworzenie głównego pliku sterownika, który będzie generował odpowiednią adresację pikseli oraz nadawał dane za pomocą interfejsów różnicowych (**listing 3**). W **tabeli 2** umieszczono zebrane w jednym miejscu parametry wykorzystanych *IP Core* w projektowanych przez nas modułach. Dodanie sufiksu HDMI jest w 100% celowe w tych nazwach – pozwoli to na uniknięcie ewentualnych problemów w sytuacji, gdyby ktoś chciał kiedyś użyć tych modułów jednocześnie.

Gdy mamy już wszystkie elementy naszej układanki kompletne, czas na uruchomienie *Platform Designera* i utworzenie interesującego nas modułu. Mam nadzieję, że tej procedury nie muszę już powtarzać. Ostatecznie powinniśmy skonfigurować wszystko identycznie, jak w wypadku interfejsu VGA z tą różnicą, że będziemy mieli 2 sygnały zegarowe i interfejs VGA z odpowiednimi sygnałami na wyjściu. Ostatecznie okno ustawień interfejsów modułu powinno wyglądać tak, jak na **rysunku 6**.

Jedną uwagą, którą chciałbym jeszcze przekazać, jest informacja na temat ustawienia *Clock rate*. Wpisanie tam wartości 0 powoduje,

Listing 1. Porty wejściowe/wyjściowe głównego modułu HDMI

```
ENTITY DisplayHDMI IS
    PORT (
        --avalon Memory-Mapped slave
        clk          : IN STD_LOGIC;
        reset_n      : IN STD_LOGIC;
        address      : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
        byteenable   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        read           : IN STD_LOGIC;
        readdata       : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        write          : IN STD_LOGIC;
        writedata    : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        --HDMI clocks
        HDMI_CLK25       : IN STD_LOGIC;
        HDMI_CLK250      : IN STD_LOGIC;
        --HDMI exported interface
        HDMI_TMDS_CLKp   : OUT STD_LOGIC;
        HDMI_TMDS_CLKn   : OUT STD_LOGIC;
        HDMI_TMDSp       : OUT STD_LOGIC_VECTOR(2 downto 0);
        HDMI_TMDSn       : OUT STD_LOGIC_VECTOR(2 downto 0)
    );
END DisplayHDMI;
```

Listing 2. Porty enkodera TMDS

```
entity TMDS is port(
    clk      : in  STD_LOGIC;  -- pixel clock
    vd_en    : in  STD_LOGIC;  -- display area indication (1=display, 0=control data)
    ctrl     : in  STD_LOGIC_VECTOR(1 downto 0);  -- sync (in blue channel) vSync & hSync
    data_in  : in  STD_LOGIC_VECTOR(7 downto 0);  -- 8-bit color data in
    data_out : out STD_LOGIC_VECTOR(9 downto 0)   -- 10-bit encoded data out
);
```

Listing 3. Nadawanie sygnałów w interfejsie HDMI

```
-- rejestr przesuwny nadający dane TMDS z częstotliwością 250MHz
TMDS_Shift: process(HDMI_CLKOUT)
begin
    if falling_edge(HDMI_CLKOUT) then
        if TMDS_Reg_counter = 9 then
            TMDS_load <= '1';
            TMDS_Reg_counter <= 0;
        else
            TMDS_load <= '0';
            TMDS_Reg_counter <= TMDS_Reg_counter + 1;
        end if;

        if TMDS_load = '1' then
            TMDS_R_Reg <= TMDS_R;
            TMDS_G_Reg <= TMDS_G;
            TMDS_B_Reg <= TMDS_B;
        else
            TMDS_R_Reg <= '0' & TMDS_R_Reg(9 downto 1);
            TMDS_G_Reg <= '0' & TMDS_G_Reg(9 downto 1);
            TMDS_B_Reg <= '0' & TMDS_B_Reg(9 downto 1);
        end if;

    end if;
end process;

-- ręczne generowanie sygnałów różnicowych
-- sygnał zegarowy odwrócony w fazie, gdyż sygnały nadajemy na zboczu opadającym,
-- a nie narastającym oryginalnego zegara
HDMI_TMDS_CLKp  <=  not HDMI_CLKPX;
HDMI_TMDS_CLKn  <=      HDMI_CLKPX;
HDMI_TMDSp(0)   <=      TMDS_B_Reg(0);
HDMI_TMDSn(0)   <=  not TMDS_B_Reg(0);
HDMI_TMDSp(1)   <=      TMDS_G_Reg(0);
HDMI_TMDSn(1)   <=  not TMDS_G_Reg(0);
HDMI_TMDSp(2)   <=      TMDS_R_Reg(0);
HDMI_TMDSn(2)   <=  not TMDS_R_Reg(0);
```

że *Platform Designer* nie zwraca uwagi na to, jaki sygnał zegarowy podepniemy. Wpisanie tu innej niż zero wartości (wyrażonej w Hz) powoduje, że oprogramowanie będzie weryfikować, czy podłączony sygnał ma taką częstotliwość, jak wpisana przez nas w tym okienku. Dla naszego układu moglibyśmy zatem wpisać dla odpowiednich wejść sygnału zegarowego pożądane częstotliwości 25 i 250 MHz, wyrażone w Hz. Po tym zapisujemy moduł i… prawie gotowe!

Ostatnie połączenia

Dodajmy nasz moduł do podstawowego systemu (**rysunek 7**). Zabraknie nam oczywiście odpowiednich sygnałów zegarowych. Aby je wygenerować, otwieramy właściwości naszej pętli PLL i na wyjściach *c1* oraz *c2* generujemy częstotliwości 25 i 250 MHz. Dodatkowo, zmniejszamy rozmiar pamięci RAM naszego procesora na 16 384 bajty (inaczej pamięci modułu HDMI nie zmieszczą się w układzie – alternatywnie możemy i tu podzielić adresy pikseli i mieć 4-krotnie mniejszą liczbę znaków o większych pikselach, ale za to zachować więcej pamięci dla procesora). Następnie możemy standardowo dołączyć nowy komponent, wyeksportować interfejs HDMI i wykonać czynności zmierzające do przypisania pinów układu FPGA. Zanim jednak klikniemy na *Analysis&Synthesis*, musimy w *Assignments* → *Device* → *Device and Pin Options…* → *Configuration* ustawić *Configuration mode* na *Single Uncompressed Image with Memory Initialization*. Inaczej nie będzie możliwe zainicjalizowanie pamięci ROM znaków.

Teraz pozostało tylko odpowiednie przypisanie pinów i ustawienie standardu wszystkich wyjść na *3.3-V LVTTL* (**rysunek 8**). Po tym kroku możemy dokonać syntezy naszego układu, a po zaprogramowaniu go podpiąć monitor HDMI (lub DVI za pomocą odpowiedniego kabla).

Kilka linijek oprogramowania

Teraz czas na uruchomienie *Eclipse* i wygenerowanie szkieletu oprogramowania, a następnie napisanie funkcji do obsługi wyświetlacza, które w zasadzie będą identyczne, jak te do obsługi VGA, z uwzględnieniem większej liczby znaków. Wszak znów całą skomplikowaną pracę wykonuje sprzęt, a my tylko podajemy mu w pamięci RAM odrobinę danych.

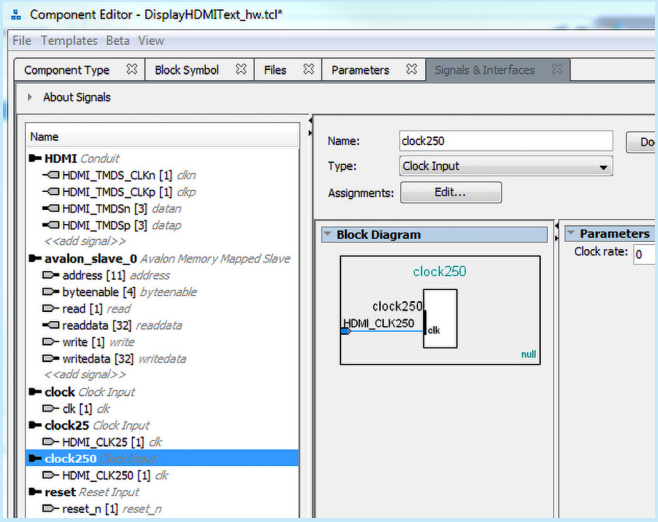
Jeśli wszystko zostało wykonane poprawnie, powinniśmy zobaczyć nasz tekst na ekranie. Rzecz jasna i tu można pokusić się o dodanie koloru kosztem pamięci RAM. Tym razem mamy większe pole do popisu, ponieważ HDMI udostępnia całą 24-bitową przestrzeń kolorów!

Podsumowanie

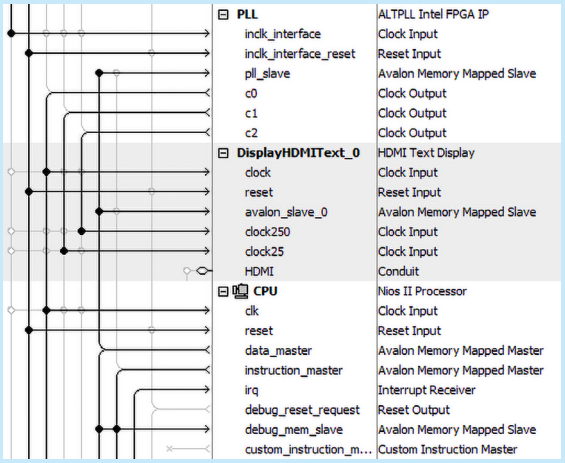
W ramach podsumowania nawiążę do naszego pierwszego spotkania – pamiętacie, jak mówiłem o zaletach FPGA w wypadku generowania obrazu na przykładzie interfejsu VGA? A teraz wyobraźcie sobie, jak potężny procesor musielibyśmy zaprzęgnąć do generowania sygnałów HDMI, gdyby nie układy FPGA czy specjalizowane układy scalone.

Jest to też zarazem przedostatnia część niniejszego kursu, zaś ostatnia projektowa. W czasie ostatniego wspólnego spotkania (ale mam nadzieję zaczynającego Wasze częste samodzielne spotkania z układami FPGA) postaram się uzupełnić informacje, na które zabrakło miejsca w dotychczasowych 14 częściach, m.in. wykorzystanie systemu wygenerowanego w *Platform Designer* w sposób inny niż *Top Level* (i podłączenie po drodze dowolnych układów) oraz symulacje, nieodzowne w czasie projektowania modułów VHDL. No i oczywiście podsumujemy cały kurs, liczący 15 spotkań. Do zobaczenia!

Piotr Rzeszut, AGH



Rysunek 6. Prawidłowo ustawione sygnały i interfejsy dla modułu HDMI



Rysunek 7. Prawidłowo podłączony do systemu moduł HDMI

hdmi_clkn	Output	PIN_T3
hdmi_clkp	Output	PIN_T2
hdmi_datan[2]	Output	PIN_R6
hdmi_datan[1]	Output	PIN_R3
hdmi_datan[0]	Output	PIN_T5
hdmi_datap[2]	Output	PIN_R5
hdmi_datap[1]	Output	PIN_R2
hdmi_datap[0]	Output	PIN_T4

Rysunek 8. Poprawne przypisanie pinów interfejsu HDMI