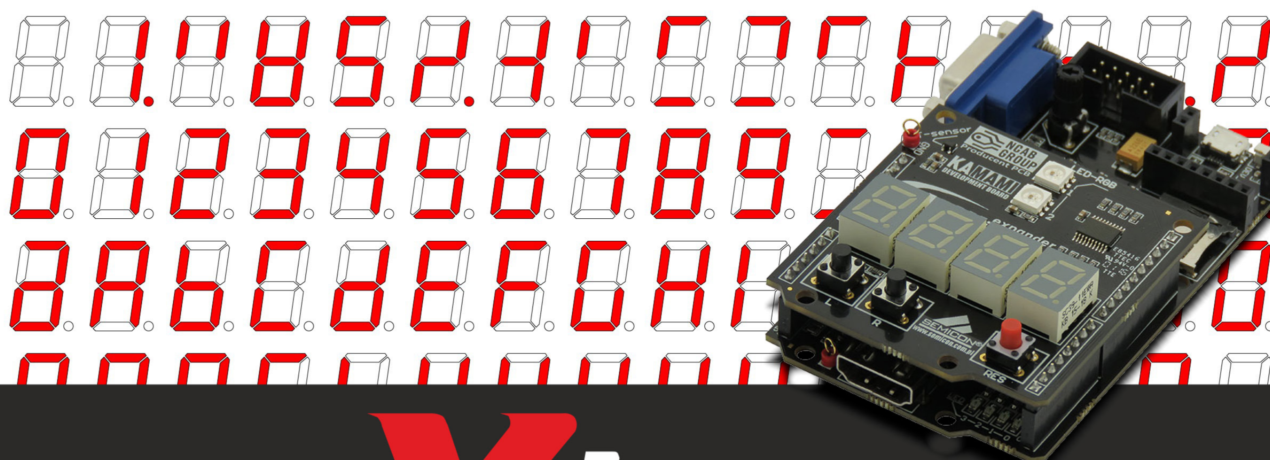


NIOS II na maXimatorze, czyli mikroprocesor w układzie FPGA (10)

Własne moduły w systemie – wyświetlacz 7-segmentowy i enkodery

Ostatnio osiągnęliśmy „kamień milowy" i dodaliśmy do naszego systemu pierwsze samodzielnie utworzone moduły. Teraz czas użyć tej wiedzy do zbudowania nieco bardziej skomplikowanych układów, które pozwolą na lepsze wykorzystanie zasobów sprzętowych. Konkretnie, zajmiemy się wykonaniem modułu sterownika wyświetlaczy 7-segmentowych oraz modułem do obsługi enkoderów inkrementalnych.

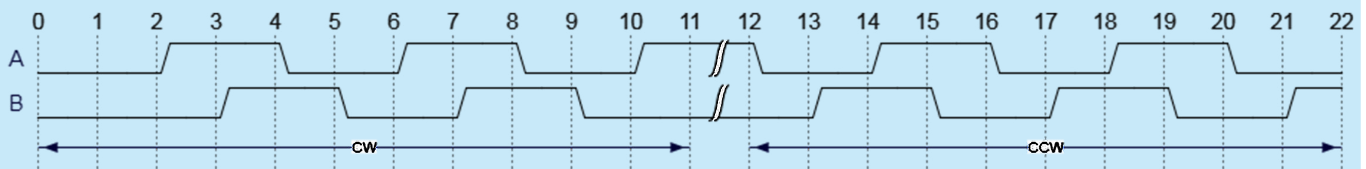
Tym razem nie mamy zbyt wiele do omówienia, ponieważ większość podstaw teoretycznych już wcześniej omówiliśmy! Wyświetlacz 7-segmentowy? Przecież już świetnie znamy jego obsługę! O enkoderach inkrementalnych zaraz się dowiemy, ale występujące w nich zjawisko drgań styków też już zwalczaliśmy podczas jednych z naszych pierwszych „spotkań". Tym razem te funkcje jednak zaimplementujemy nie w programie, ale w sprzęcie – to znaczy, że procesor nie będzie musiał przejmować się ich wykonaniem, gdyż zrobi to za niego dedykowany fragment układu zaimplementowanego w FPGA. Właśnie to jest największa przewaga takich systemów – zadania wymagające „dużej uwagi" jesteśmy w stanie realizować za pomocą, jak zaraz zobaczycie, nieskomplikowanych układów sprzętowych.

Enkodery inkrementalne

Enkodery są urządzeniami elektromechanicznymi, które przetwarzają ruch na serię impulsów elektrycznych. W zależności od typu enkodera impulsy te mogą mieć różną konfigurację, liczbę kanałów, mogą wskazywać absolutne położenie enkodera (enkodery absolutne) albo zmianę tego położenia (enkodery inkrementalne). My zajmiemy się tym drugim typem enkodera, ponieważ są one łatwo dostępne i powszechnie używane.

Zwykle takie enkodery generują 2 sygnały, nazywane najczęściej A i B – generują, czyli z mechanicznego punktu widzenia powodują włączanie lub wyłączanie odpowiednich przełączników, najczęściej zwierających sygnały do masy – podobnie jak zwykłe przyciski, chociaż trzeba przyznać, że w bardziej wymagających zastosowaniach

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



Rysunek 1. Sygnały na wyjściach enkodera dla obrotów zgodnie (CW) i przeciwnie (CCW) do ruchu wskazówek zegara

czasem są używane enkodery optyczne bazujące na fototranzystorach. Sygnały te, w zależności od kierunku obrotów zmieniają swoją wzajemną fazę, jak można zobaczyć na **rysunku 1**.

Oczywiście, jest to sytuacja idealna, czyli taka, w której nie obserwujemy drgań styków (ale z nimi sobie potrafimy poradzić, więc nie musimy się tym teraz martwić). Merytorycznie jasno widzimy jak rozróżnić kierunek obrotów. Wystarczy patrzeć jaki stan w momencie danego zbocza na jednej linii ma druga linia – i tak na przykład w obrotach w prawo linia A ma stan wysoki w czasie zbocza narastającego B, zaś przy obrotach w drugim kierunku linia A ma wtedy stan niski. Nic prostszego! Jedyna drobna pułapka jaka na nas czeka to fakt, że różne enkodery generują różną ilość takich zboczy na jeden „krok". Czasem będzie to jedno zbocze na jednym kanale (np. chwile czasu 2-3 z rysunku), czasem dwa (2-4), a czasem więcej. Oczywiście jest też możliwość, że enkoder będzie nieco mniej standardowy i będzie generował na jednym kanale impulsy znacznie krótsze niż na drugim – takie rzeczy warto sprawdzić np. analizatorem stanów logicznych czy oscyloskopem.

Sterownik do wyświetlacza raz poproszę!

Na początek zacznijmy od utworzenia nowego pliku i zapisania go w podfolderze folderu projektu. Nasz moduł, oprócz portu magistrali Avalon-MM, za pomocą której będziemy nim sterować, musi posiadać dwa porty – do sterowania segmentami oraz załączaniem poszczególnych wyświetlaczy. Magistrala adresowa będzie 3 bitowa, gdyż utworzymy 4 rejestry zawierające dane dla każdego z wyświetlaczy oraz 2 do definiowania czasu wyłączenia i załączenia wyświetlaczy (**listing 1**). Sama procedura obsługi wyświetlaczy jest bardzo łatwa do wykonania (**listing 2**).

W mojej propozycji obsługi wyświetlacza występują 2 stany: *ACT* oraz *BLANK*. W pierwszym z nich (stanie aktywnym) układ zlicza cykle zegara aż do momentu osiągnięcia wartości zadanej poprzez wartość odpowiedniego rejestru. Wtedy następuje resetowanie licznika,

Listing 1. Definicje rejestrów danych wyświetlaczy
```
entity SEG7 is
    port (
        --avalon memory-mapped slave
        clk         : in std_logic;
        reset_n     : in std_logic;
        address     : in std_logic_vector(2 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);
        --display exported interface
        display     : out std_logic_vector(3 downto 0);
        segment     : out std_logic_vector(7 downto 0)
    );
end SEG7;
-- (...)
```

wygaszenie wyświetlacza, ustawienie kolejnego wyświetlacza jako aktywnego i przejście do stanu *BLANK*. W tym stanie (stanie wygaszenia) na początku także odmierzany jest czas, po czym ustawiana jest kombinacja z rejestru odpowiadającego danemu wyświetlaczowi, załączany jest tenże wyświetlacz. Także tu resetujemy licznik i przechodzimy do stanu, od którego zaczynaliśmy nasza „wycieczkę". Fakt, że nie musimy ręcznie ustawiać numeru aktywnego wyświetlacza na zero po dojściu do końca wiąże się z tym, że rejestr go przechowujący jest tak zdefiniowany, że sam się przepełnia i ustawia na 0 po wartości 3.

Czas trwania każdego ze stanów ma znaczenie dla działania wyświetlacza. Czas stanu wygaszenia daje czas wyświetlaczowi na wyłączenie się (m. in. tranzystorom, które go sterują), tak aby nie pojawiały się „duszki", zaś czas stanu aktywnego definiuje czas przez jaki każdy z wyświetlaczy jest załączony. Suma tych czasów definiuje częstotliwość odświeżania wyświetlacza. Ciekawą możliwością w takim systemie jest możliwość regulacji jasności wyświetlaczy – wystarczy, przy zachowaniu stałego okresu (oraz dbając aby nie pojawiały się „duszki") zmieniać stosunek obu wspomnianych parametrów (tak jak działa współczynnik wypełnienia w *PWM*).

Listing 2. Procedura obsługi wyświetlaczy
```
-- (...)
-- jeśli opadające zbocze zegara
elsif falling_edge(clk) then
    -- w zależności od stanu
    case state is
    -- oczekiwanie na koniec czasu wygaszenia wszystkich wyświetlaczy
    when BLANK =>
        -- jeśli czas nie minął to liczymy dalej
        if counter < blankTime then
            counter <= counter + 1;
        -- jeśli czas minął to zapalamy wyświetlacz, resetujemy licznik i przechodzimy do kolejnego stanu
        else
            counter <= (others => '0');
            segment <= segments(actDisplay);
            display(actDisplay) <= '1';
            state <= ACT;
        end if;
        -- oczekiwanie na koniec czasu zapalenia wyświetlacza
        when ACT =>
        -- jeśli czas nie minął to liczymy dalej
            if counter < activeTime then
                counter <= counter + 1;
                -- jeśli czas minął to gasimy wyświetlacze, resetujemy licznik
                -- zwiększamy numer wyświetlacza i przechodzimy do poprzedniego stanu
            else
                counter <= (others => '0');
                segment <= (others => '0');
                display <= (others => '0');
                actDisplay <= actDisplay + 1;
                state <= BLANK;
            end if;
            -- w niezdefiniowanych przypadkach powrót do stanu BLANK
            when others =>
                state <= BLANK;
    end case;
-- (...)
```

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

Oprócz oczywiście powyższego implementujemy także zapis wszelkich koniecznych rejestrów podobnie jak ich odczyt – dokładnie tak jak w module *PWM*. Następnie wystarczy analogicznie w *Platform Designer* dodać nowy moduł, wskazać właśnie przygotowany plik, ustawić odpowiednie parametry interfejsów (szczególnie interfejsu *Avalon-MM*), dodać interfejsy *Conduit* dla sygnałów sterujących wyświetlaczami i… gotowe!

Moduł dodajemy do naszego systemu, następnie wykonujemy standardowe kroki zmierzające do kompilacji, po drodze nie zapominając o przypisaniu odpowiednich wyprowadzeń w *Pin Planner* i wybraniu dla nich adekwatnego standardu napięciowego.

Listing 3. Funkcja zaświecająca segmenty

```
void setDigit(uint8_t digit, uint8_t dp, uint8_t pos)
{
    if(pos < 4)
    {
        // dla wartosci spoza zakresu gasimy wszystkie segmenty
        if(digit > 16) IOWR_32DIRECT(SEG7_0_BASE, pos * 4, 0|(dp!=0?(1<<SEG_DP):0));
        // dla pozostalych wartosci wyswietamy odpowiednia liczbe
        else IOWR_32DIRECT(SEG7_0_BASE, pos * 4, digits[digit] | (dp!=0?(1<<SEG_DP):0));
    }
}
```

Listing 4. Eliminowanie drgań styków

```
-- Project:         maximator-Tutorial-10
-- File:                debounce.vhd
-- Version:         1.0 (03.08.2018)
-- Author:          Piotr Rzeszut (http://piotr94.net21.pl)
-- Description: Signal debouncing

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity DEBOUNCE is
    port (
        -- zegar
        clk             : in std_logic;
        -- wejście
        signalIn            : in std_logic;
        -- wyjście
        signalOut       : out std_logic;
        -- wejście do ustawiania czasu eliminacji drgań styków
        debounceTime    : in std_logic_vector(31 downto 0)

    );
end DEBOUNCE;

architecture basic of DEBOUNCE is
    -- rejestr licznika
    signal counter  : std_logic_vector(31 downto 0);
    -- rejestry do zatrzaskiwania poprzedniego stanu wejścia
    signal p1       : std_logic;
    signal p2       : std_logic;
begin
    -- układ ma być wrażliwy na zmiany na poniżej wymienionych liniach
    process(clk) is
    begin
        -- jeśli opadające zbocze zegara
        if rising_edge(clk) then
            -- zatrzaksujemy poprzedni stan w p2
            p2 <= p1;
            -- a aktualny w p1
            p1 <= signalIn;
            -- jeśli było zbocze (zmiana)
            if (p1 xor p2) = '1' then
                -- reset licznika
                counter <= (others => '0');
            elsif counter < debounceTime then
                counter <= counter + 1;
            else
                signalOut <= p2;
            end if;
        end if;
    end process;
end basic;
```

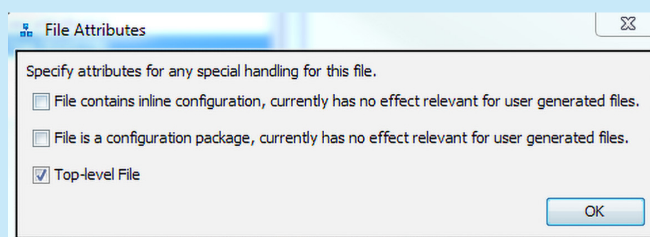
Oprogramowanie…

…mamy właściwie już gotowe! Wystarczy, że weźmiemy na „warsztat" oprogramowanie, które przygotowaliśmy wcześniej, gdy wykorzystywaliśmy timer. Oczywiście musimy usunąć fragmenty odpowiedzialne za odświeżanie i bezpośredni dostęp do wyprowadzeń, a potem zmodyfikować funkcje ustawiające kombinację segmentów (**listing 3**).

W zasadzie jedyną zmianą, jakiej dokonaliśmy jest… zastąpienie zapisu kombinacji segmentów do tablicy zapisem do naszego modułu. Dodatkowo musimy ustawić w naszym module czasy wygaszenia i zapalenia wyświetlaczy:

```
IOWR_32DIRECT(SEG7_0_BASE, 16, 100-1);
IOWR_32DIRECT(SEG7_0_BASE, 20, 2000-1);
```

Pierwszy z nich ustawiamy na 100 (zapisujemy wartość 99, ponieważ podział następuje przez podaną tu wartość zwiększoną o 1 – oczywiście dla naszego zastosowania tak kosmetyczna różnica nie ma znaczenia, ale będąc tu dokładnym staram się zwrócić uwagę na te kwestię – w innym momencie może być ona kluczowa), czyli czas wygaszenia wyświetlaczy wyniesie 1/50 MHz * 100 = 2 μs. To czas, który potrzebny jest na wyłączenie się tranzystorów i innych elementów sterujących. Z kolei przez 40 μs każdy z wyświetlaczy będzie zapalony. I to wszystko! resztą zajmie się działający niezależnie od procesora moduł!



Rysunek 2. Ustawianie pliku głównego w Component Editor

Czas zdekodować enkoder

Aby skutecznie uporać się z problemem obsługi enkodera najlepiej jeśli zadanie podzielimy na dwie części: eliminację drgań styków oraz właściwą obsługę enkodera.

Eliminowanie drgań styków Eliminowanie drgań styków odbywa się sprzętowo w oparciu o dokładnie takie same zasady, jak gdybyśmy robili to za pomocą programu (**listing 4**). Przy każdym zboczu sygnału zegarowego zatrzaskujemy w rejestrze *p1* aktualny stan wejścia, zaś w rejestrze *p2* poprzednią wartość rejestru *p1*. Jednocześnie sprawdzamy czy wartości tych rejestrów są różne (funkcja *xor*) – jeśli tak jest to wystąpiło zbocze i resetujemy licznik. Jeśli sygnał utrzymuje się stabilny to zwiększamy wartość licznika aż do momentu osiągnięcia ustalonego poprzez wejście *debounceTime*. Po osiągnięciu tej wartości na wyjście zatrzaskujemy sygnał z rejestru *p2*. Stan taki potrwa aż do wykrycia kolejnego zbocza. Rzecz jasna funkcja ta będzie realizowana w sprzęcie – poprzez odpowiedni „układ bramek i przerzutników".

Enkoder we własnej osobie Jeśli będziemy już posiadać pozbawione drgań styków sygnały enkodera możemy wykorzystać podobny mechanizm jak w module eliminacji drgań styków do wykrywania zbocz narastających i opadających na poszczególnych kanałach (risingA/B, fallingA/B). Oczywiście funkcjonalności te implementujemy w innym pliku (**listing 5**) i tu będzie trochę różnic w generowaniu modułu *Platform Designer*, które za moment omówimy. Następnie w zależności od stanu drugiej z linii w momencie wystąpienia zbocza zwiększamy lub zmniejszamy wartość licznika. Proste?

Dzięki temu w procesorze możemy odczytać 32-bitową liczbę reprezentującą ilość impulsów wygenerowanych przez enkoder.

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

W zależności od kierunku obrotów liczba ta będzie wzrastać lub maleć. Oczywiście musimy pamiętać, że enkoder na jeden „przeskok" może generować kilka impulsów. Dodatkowo, w module zaimplementujemy możliwość ustawiania stanu licznika z poziomu procesora, a także oczywiście możliwość definiowania czasu eliminacji drgań styków.

Po przygotowaniu takich dwóch plików (u mnie nazwane *debounce.vhd* oraz *encoder.vhd* i zapisane w folderze *Encoder*) rozpoczynamy procedurę dodawania nowego modułu *Platform Designer*. Tym razem w okienku dodawania plików dodajemy oba wspomniane wyżej pliki. Tym razem musimy wskazać, który z nich jest plikiem głównym. Aby to zrobić klikamy dwukrotnie w kolumnie *Attributes* obok pliku *encoder.vhd* i w okienku (**rysunek 2**), które wyskoczy wybieramy *Top-level File*. Po tym kopiujemy pliki do okienek symulacji i wykonujemy analizę, następnie uzupełniamy zakładkę sygnałów i interfejsów. I już możemy dodać moduł do naszego projektu. Ostatecznie, po uzupełnieniu połączeń, kluczowe fragmenty naszego projektu powinny wyglądać jak na **rysunku 3**.

Teraz czas na sfinalizowanie kwestii związanej z przypisaniem pinów (rzecz jasna po wygenerowaniu systemu i innych krokach, które już do tego momentu powinny nam wejść w krew). Ale musimy pamiętać o jednej szalenie ważnej kwestii – o ile nasz enkoder będzie zwierać wyprowadzenia do masy to musimy w jakiś sposób zapewnić stan wysoki – czyli w skrócie użyć rezystorów podciągających do zasilania. O ile korzystamy z wyprowadzeń wyposażonych na naszej płytce w translatory napięć (piny cyfrowe) to musimy takie rezystory dodać (z podciąganiem do 5 V). Ale już jeśli zdecydujemy się na użycie pinów analogowych (nie wykorzystanych w nakładce maXimator expander – np. A4 i A5) możemy skorzystać z wewnętrznych rezystorów podciągających w które wyposażony jest nasz układ FPGA. Oczywiście są to rezystory o stosunkowo dużych wartościach i mogą nie sprawdzać się w pewnych aplikacjach, jednak w naszej sytuacji w zupełności wystarczą. Jak je dodać? Zacznijmy od kliknięcia prawym klawiszem myszy w dowolnym miejscu na nagłówku kolumn listy używanych wyprowadzeń w *Pin Planner*. Następnie wybierzmy *Customize Columns...* i w oknie które się pojawi znajdźmy w *Available columns* kolumnę *Weak Pull-Up Resistor* po czym kliknijmy na strzałkę w prawo (>) i *OK* (**rysunek 4**).

Teraz wśród kolumn powinna pojawić się właśnie wybrana (dodatkowo kolumny możemy tak przeciągać, aby ich kolejność była dla nas wygodna). Odszukujemy sygnały związane z enkoderem i klikamy dla nich dwukrotnie w nowododanej kolumnie po czym wybieramy *on*, podobnie dla drugiego wyprowadzenia związanego z enkoderem. Voilà! Ostatecznie definicje wyprowadzeń dla naszego projektu powinny wyglądać tak jak na **rysunku 5**.

Listing 5. Obsługa enkodera

```
-- (...)
    debounceA : component debounce
        port map (
            clk             =>clk,
            signalIn            =>encoderAB(0),
            signalOut       =>encoderA,
            debounceTime    =>debounceTime
        );

    debounceB : component debounce
        port map (
            clk             =>clk,
            signalIn            =>encoderAB(1),
            signalOut       =>encoderB,
            debounceTime    =>debounceTime
        );

    risingA <= (not p1A) and p2A;
    fallingA <= p1A and (not p2A);

    risingB <= (not p1B) and p2B;
    fallingB <= p1B and (not p2B);
    -- układ ma być wrażliwy na zmiany na poniżej wymienionych liniach
    process(clk, reset_n) is
    begin
        -- jeśli linia reset w stanie niskim
        if reset_n = '0' then
            debounceTime        <= (others => '0');
            counter             <= (others => '0');
        -- jeśli opadające zbocze zegara
        elsif falling_edge(clk) then
            -- zatrzaskujemy sygnały po eliminacji drgań styków celem wykrycia zbocz
            p2A <= p1A;
            p1A <= encoderA;

            p2B <= p1B;
            p1B <= encoderB;

            -- obsługa enkodera i zmiana wartości licznika
            if risingA = '1' then
                if p2B = '1' then
                    counter <= counter + 1;
                else
                    counter <= counter - 1;
                end if;
            elsif fallingA = '1' then
                if p2B = '1' then
                    counter <= counter - 1;
                else
                    counter <= counter + 1;
                end if;
            elsif risingB = '1' then
                if p2A = '1' then
                    counter <= counter - 1;
                else
                    counter <= counter + 1;
                end if;
            elsif fallingB = '1' then
                if p2A = '1' then
                    counter <= counter + 1;
                else
                    counter <= counter - 1;
                end if;
            end if;
-- (...)
```

Listing 6. Obsługa enkodera w programie głównym

```
int main()
{
    // ustawienia wyświetlacza - czasu wygaszenia i czasu aktywnego
    IOWR_32DIRECT(SEG7_0_BASE, 16, 100-1);
    IOWR_32DIRECT(SEG7_0_BASE, 20, 2000-1);
    // ustawienie czasu eliminacji drgań styków na 1 ms
    IOWR_32DIRECT(ENCODER_0_BASE, 4, 50000-1);
    // ustawienie wartości początkowej licznika enkodera
    IOWR_32DIRECT(ENCODER_0_BASE, 0, 0x1000);
    /* Event loop never exits. */
    while (1)
    {
        // przepisujemy zawartość licznika enkodera na wyświetlacze 7-segmentowe
        intDisplayHex(IORD_32DIRECT(ENCODER_0_BASE, 0));
    }
    return 0;
}
```

Parę linijek programu do sukcesu!

Chyba już wszyscy spodziewamy się jak łatwe i bezproblemowe będzie obsłużenie enkodera z poziomu programu?

Zaraz po ustawieniach wyświetlacza ustawiamy czas eliminacji drgań styków na 1 ms (podzielnik 50000) oraz podajemy wartość początkową licznika. W pętli głównej jedynie przepisujemy wartość licznika napędzanego enkoderem na wyświetlacz. Po uruchomieniu całości powinniśmy móc zwiększać i zmniejszać zawartość licznika na wyświetlaczu za pomocą obrotów enkodera. Jeśli wartość przeskakuje np. o 2 przy każdym kroku oznacza to, że nasze „pokrętło" generuje 2 impulsy na 1 skok – musimy wtedy wprowadzić proste matematyczne dzielenie tej liczby. Czyż to wszystko nie stało się teraz dziecinnie proste?

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

Najlepsze to to, że procesor w ogóle nie martwi się o tak prozaiczne (a zarazem i złożone) rzeczy jak odświeżanie wyświetlaczy czy obsługę enkodera – oto zaleta układów FPGA! Zupełnie jakbyśmy nasz klasyczny mikrokontroler obudowali dodatkowymi układami scalonymi, z tą różnicą że tu wszystko wykonujemy bez dodatkowych kosztów i wewnątrz jednej czarnej skrzyneczki!

Podsumowanie

Tym razem nie obędzie się bez kilku zadań do przemyślenia i wykonania, jeśli będziecie mieli ochotę:

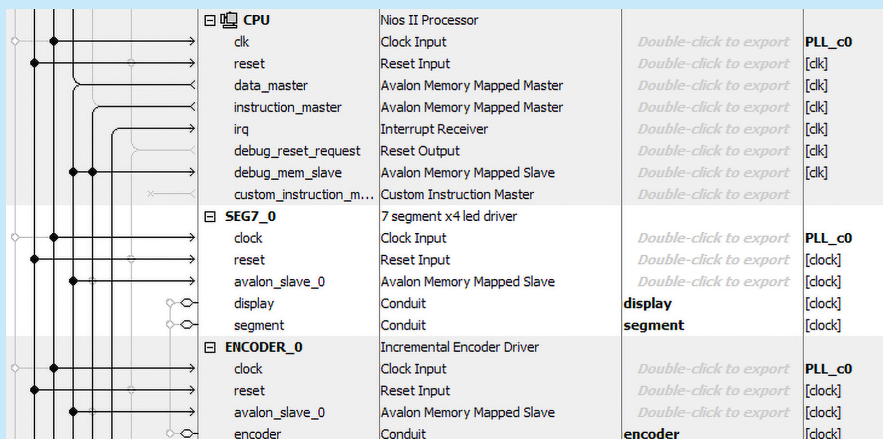
1. W układzie sterownika wyświetlaczy marnujemy rejestry – wykorzystujemy 32-bitowe rejestry do zapisania jedynie 8 bitów wzoru dla każdej pozycji. Spróbujcie tak zmodyfikować układ, aby wykorzystać jedynie jeden 32-bitowy rejestr i pełne możliwości linii *byteenable* oraz funkcje zapisu 8-bitowych danych ze strony procesora. Po tym powinniście móc zmniejszyć liczbę bitów adresu tego modułu!
2. W układzie sterownika do enkoderów użyliśmy dodatkowych rejestrów do wykrywania zboczy. Ale przecież moduł *debounce* wie kiedy występuje zbocze na wyjściu! Potrafilibyście tak zmodyfikować te oba moduły, aby to moduł eliminacji drgań styków był wyposażony w dodatkowe wyjścia informujące o zboczu narastającym i opadającym?
3. Napisać proste biblioteki oraz pliki z opisem rejestrów naszych modułów. W szczególności do sterownika wyświetlacza dopisać funkcję regulującą jasność.

Podczas tego spotkania wykonaliśmy kolejne 2 moduły sprzętowe do naszego systemu, które odciążają procesor i pokazują jak niewielkim wysiłkiem można osiągnąć bardzo przydatne układy. Przy okazji nauczyliśmy się, że dany moduł może składać się z wielu plików, a także jak wykorzystać rezystory podciągające wbudowane w układ FPGA z którym pracujemy.

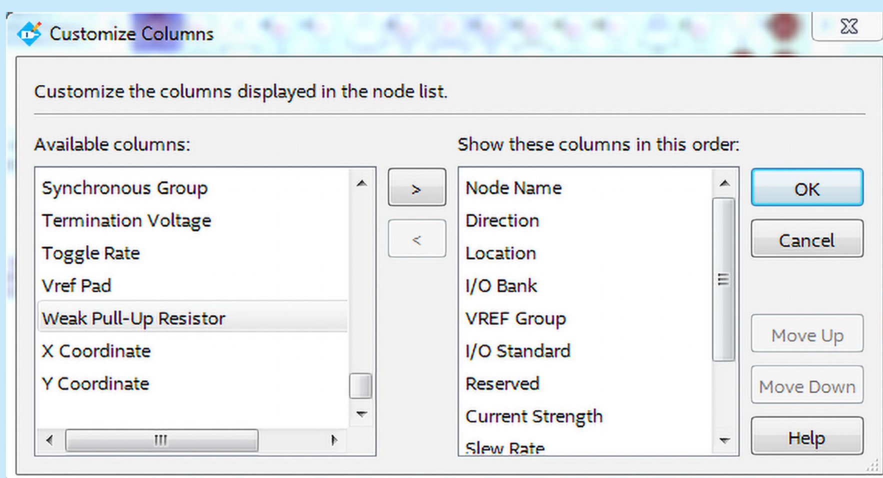
Następnym razem zajmiemy się obsługą ostatniego elementu naszego zestawu, którego jeszcze nie „dotknęliśmy" – diodami kolorowymi WS2812. Na fali tego rozwiązania pokusimy się też o sterowanie większej liczby takich diodek (umieszczonych na taśmie lub np. na module NeoPixel Shield.

Powodzenia z zadaniami i do następnego spotkania w barwach RGB!

Piotr Rzeszut, AGH



Rysunek 3. Kluczowe części systemu po dodaniu do niego modułów sterownika wyświetlaczy i enkodera



Rysunek 4. Dodawanie kolumny kontroli rezystorów podciągających w Pin Planner

Node Name	Direction	Location	eak Pull-Up Resis	I/O Standard
in clk_clk	Input	PIN_L3		3.3-V LVTTL
out display_display[3]	Output	PIN_D16		3.3-V LVTTL
out display_display[2]	Output	PIN_D15		3.3-V LVTTL
out display_display[1]	Output	PIN_E16		3.3-V LVTTL
out display_display[0]	Output	PIN_E15		3.3-V LVTTL
in encoder_encoder[1]	Input	PIN_E1	on	3.3-V LVTTL
in encoder_encoder[0]	Input	PIN_F1	on	3.3-V LVTTL
out led_export[3]	Output	PIN_R16		3.3-V LVTTL
out led_export[2]	Output	PIN_P16		3.3-V LVTTL
out led_export[1]	Output	PIN_N16		3.3-V LVTTL
out led_export[0]	Output	PIN_M16		3.3-V LVTTL
in reset_reset_n	Input	PIN_B10		3.3-V LVTTL
out segment_segment[7]	Output	PIN_F16		3.3-V LVTTL
out segment_segment[6]	Output	PIN_G16		3.3-V LVTTL
out segment_segment[5]	Output	PIN_G15		3.3-V LVTTL
out segment_segment[4]	Output	PIN_H16		3.3-V LVTTL
out segment_segment[3]	Output	PIN_H15		3.3-V LVTTL
out segment_segment[2]	Output	PIN_J16		3.3-V LVTTL
out segment_segment[1]	Output	PIN_J15		3.3-V LVTTL
out segment_segment[0]	Output	PIN_L16		3.3-V LVTTL

Rysunek 5. Ustawienia pinów dla naszego projektu

www.elektronikapraktyczna.pl

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