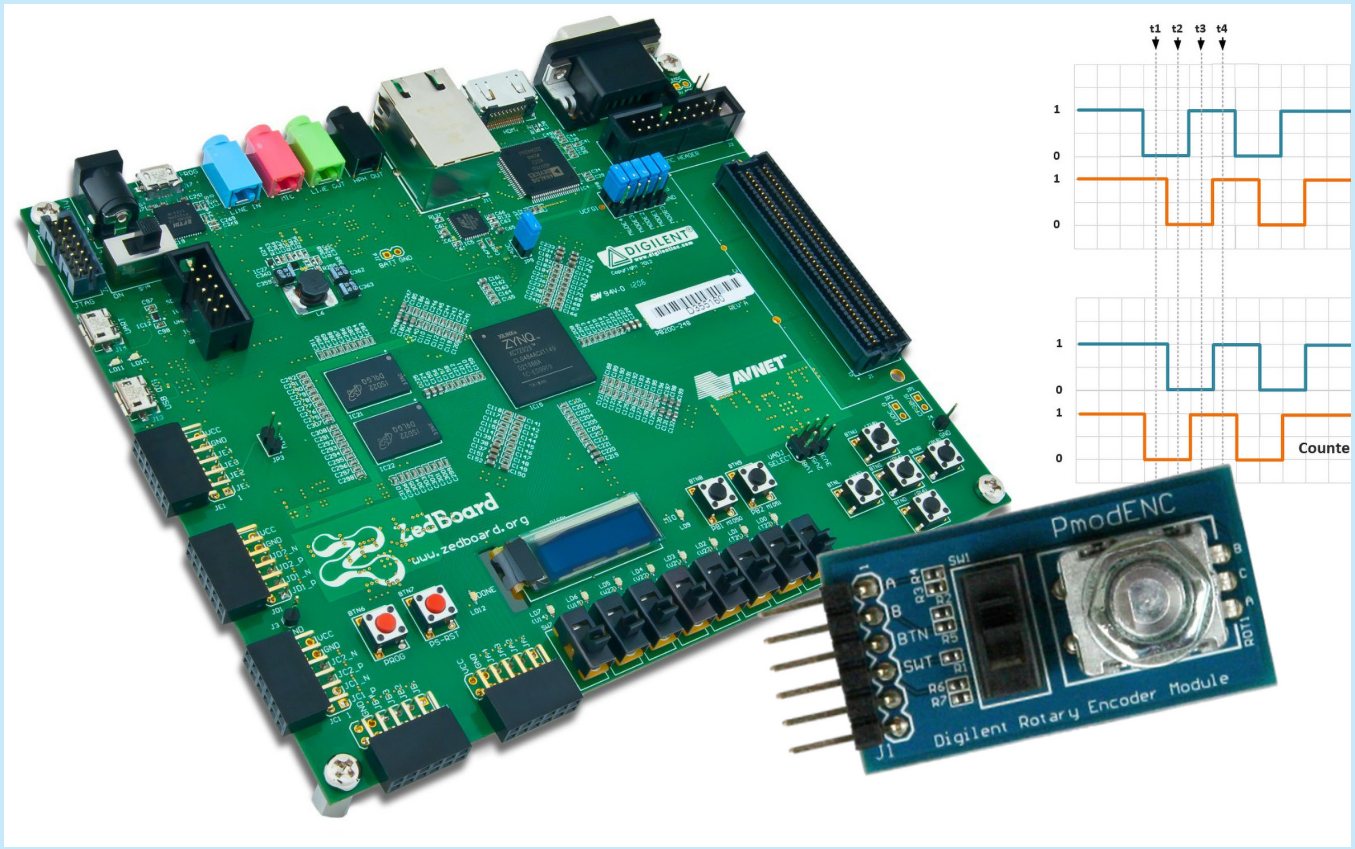


Jak używać układy SoC Xilinx Zynq-7000 z Linuksem – proste przykłady (6)

Interfejs Pmod i obsługa enkodera obrotowego na płytce Zedboard

Przyciski i przełączniki są najprostszymi elementami, za pomocą których można sterować urządzeniem. Czasem jednak przydają się bardziej zaawansowane elementy. Jednym z nich jest enkoder obrotowy. W tym artykule pokażę jak obsłużyć Przykładowy enkoder znajdujący się na płytce PmodENC firmy Digilent, przy okazji przedstawiając najprostszy interfejs obecny na wielu płytkach z układami FPGA – PMOD.

Enkodery obrotowe to elementy mechaniczne pozwalające na ustawienie pewnych parametrów urządzenia (np. głośności). Od potencjometrów, czy innych analogowych elementów obrotowych różnią się tym, że nie zmieniają tak naprawdę bezpośrednio parametrów urządzenia, a tylko podają informację o swoim położeniu, lub tylko o samej, zwykle krokowej, zmianie położenia i kierunku, w którym zmiana nastąpiła. Na tej podstawie układ sterujący, np. mikrokontroler, czy układ FPGA może zmienić docelowy parametr. Enkodery są powszechnie używane jako np. regulatory głośności, czy wszelkich innych parametrów w radiach samochodowych i innych urządzeniach audio. Pozwalają też na ustawienie temperatury, czy timera w urządzeniach AGD.

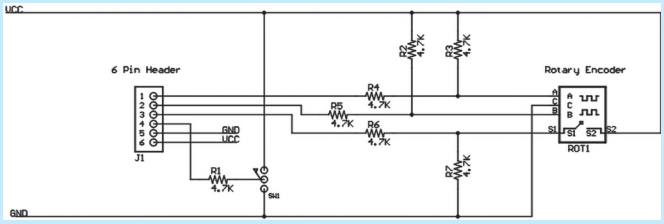
Istnieją różne rodzaje enkoderów obrotowych. Niektóre z nich pozwalają na przekazanie z różną precyzją swojego położenia, zaś najprostsze tylko na wysłanie informacji o obrocie o jeden krok w określonym kierunku. My będziemy się zajmować tym ostatnim.

Nasz obszar zainteresowań to elektronika, więc nie będę opisywał mechanicznej zasady działania enkodera. Opiszę tylko to, co niezbędne, aby móc go zastosować w układzie elektronicznym.

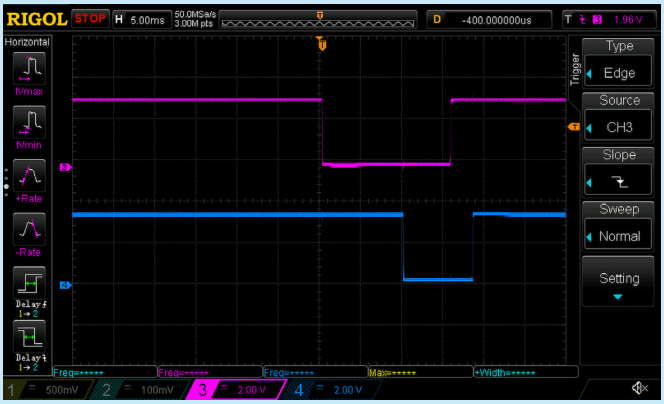
Jak już wspomniałem, będziemy się zajmować obsługą enkodera znajdującego się na płytce PmodENC firmy Digilent. Schemat elektryczny płytki znajduje się na **rysunku 1**. Widzimy na nim, że enkoder ma 3 wyprowadzenia, oznaczone jako A, B i C. Do wyprowadzeń

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

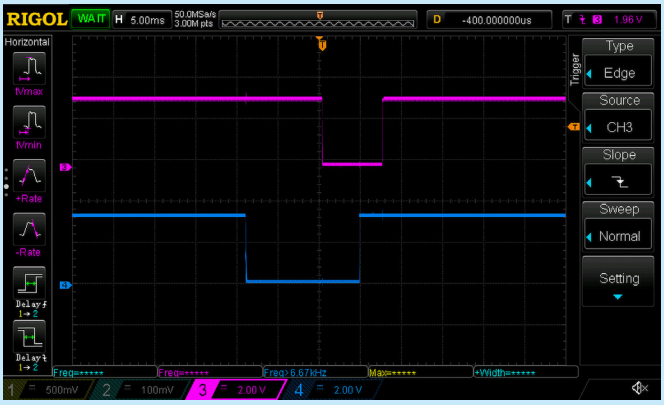
A i B podłączamy przez rezystor napięcie, a C jest podłączony do masy. Gdy gałka zostanie obrócona o kąt odpowiadający jednemu skokowi, to wyprowadzenia A i B zostają sekwencyjnie zwarte do C. Kolejność ich zwierania uzależniona jest od kierunku obrotu, a długość impulsów od szybkości obrotu. **Rysunki 2** i **3** przedstawiają oscylogramy, na których widać przebiegi napięć na wyprowadzeniach A i B, jakie można zaobserwować podczas obrotu gałki



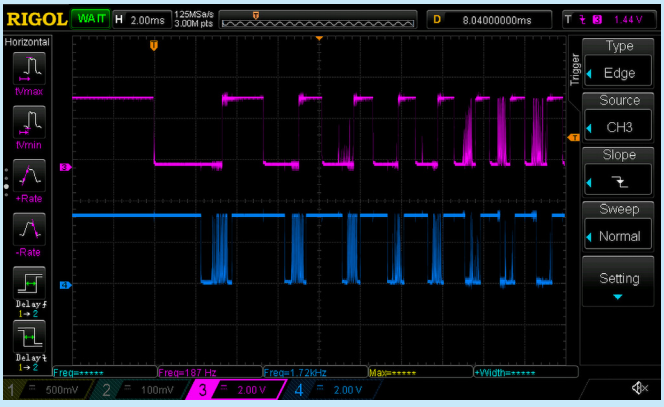
Rysunek 1. Schemat elektryczny modułu PmodENC



Rysunek 2. Przykładowe przebiegi na stykach enkodera (obrót w lewo)



Rysunek 3. Przykładowe przebiegi na stykach enkodera (obrót w prawo)



Rysunek 4. Drgania styków impulsatorów enkodera podczas szybkiego obracania osi

w lewo i w prawo. Istnieją różne sposoby wykrywania kierunku obrotu. Najprostszy jaki znam polega na wykrywaniu opadającego zbocza na jednym z wyprowadzeń i sprawdzaniu jaki jest w tym momencie stan na drugim wyprowadzeniu. Po wykryciu kierunku należy poczekać aż oba sygnały powrócą do stanu wysokiego i znów czekać na zbocze opadające.

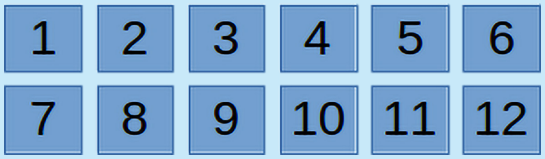
Enkodery są elementami mechanicznymi, więc podlegają takim samym prawom, jak np. przyciski. Przy ich obsłudze trzeba się zmierzyć z problemem tzw. drgań styków. Na rysunkach 2 i 3 obrót jest stosunkowo wolny i jak widać przebiegi są niezakłócone drganiami, jednak na **rysunku 4** widać już ten problem w dużym stopniu. Gdybyśmy mogli modyfikować układ, to warto by było dodać do niego kondensator, który wyeliminowałby drgania. W tym przypadku nie mamy jednak takiej możliwości, gdyż płytka jest już gotowa. Musimy poradzić sobie z problemem tak, jak w przypadku przycisków – wykonać debouncing. W tym celu po wykryciu zbocza na pierwszym wyprowadzeniu i poziomu napięcia, jaki jest na drugim wyprowadzeniu poczekamy odpowiednią ilość czasu z dalszym wykrywaniem poziomu napięć, tak aby przeczekać drgania. Z powodu dużej niestabilności sygnału na wyprowadzeniu B w późniejszym stadium, tzn. gdy na wyprowadzeniu A jest już stan wysoki, dobrze jest wykonać debouncing również po wykryciu stanu wysokiego na obu wyprowadzeniach. Empirycznie wyznaczyłem ten czas na 1 ms, ale polecam również poeksperymentować samemu, gdyż enkodery mogą się od siebie różnić.

Wykryte zdarzenie obrotu należy przekazać dalej, na przykład do systemu operacyjnego. Można to zrobić na wiele sposobów, ale moim zdaniem najprościej będzie generować impulsy na GPIO, emulując w ten sposób przyciski. Na wyjściu bloku są dwa wyprowadzenia, na jednym z nich pojawią się impulsy z obrotu w jedną stronę, a na drugim – w drugą. Trzeba więc będzie przypisać w DTS odpowiednie kody przycisków na klawiaturze do odpowiednich wyprowadzeń GPIO, tak jak to zostało zrobione w przykładzie z GPIO.

Podczas obrotu zdarzenia mogą następować bardzo szybko, a możliwości wykrywania ich przez procesor poprzez GPIO są ograniczone. W związku z tym trzeba będzie zapisywać zdarzenia do kolejki, aby nie zgubić żadnego z nich. Aby pewnie wykryć wszystkie impulsy dobrze jest zastosować sterownik gpio-keys-polled i odpowiednio dobrać ich długość i częstotliwość sprawdzania. Zasada działania tego sterownika polega na sprawdzaniu stanu pinu GPIO z określoną częstotliwością. Nie jest więc konieczne wykorzystanie systemu przerwań.

Płytkę przyłączymy do złącza PMOD. Numeracja pinów tego złącza jest widoczna na **rysunku 5**. Piny 1…4 i 7…10 są ogólnego przeznaczenia, piny 5 i 11 podłączone są do masy, zaś 6 i 12 do zasilania. Dokładną specyfikację interfejsu PMOD można znaleźć na stronie firmy Digilent (***http://bit.ly/2HvWgVM***).

Naszym celem jest wykorzystanie do wykrywania obrotu enkodera struktury FPGA, tak więc trzeba do niej podłączyć enkoder. Do pinów FPGA podłączone są 4 z 5 dostępnych złącz, oznaczone jako JA, JB, JC i JD. JE podłączone jest do pinów EMIO i nie może być w tym przypadku wykorzystane. Zgodnie ze specyfikacją, złącze PMOD ma tylko jeden rząd, ale często, są dwurzędowe, jak w przypadku złącz obecnych na płytce Zedboard. Wybór rzędu, do którego podłączamy płytkę z enkoderem nie ma znaczenia, ale trzeba podjąć tę decyzję przed syntezą i wpisać do pliku contraints odpowiednie ustawienia. Ja w projekcie podłączyłem do górnego rzędu złącza JA,



Rysunek 5. Numeracja pinów złącza Pmod

Listing 1. Blok do obsługi enkodera

```
-- Encoder.vhd
-- Author: Stanisław Aleksiński

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity encoder is
    Generic(
        DEBOUNCE_CYCLES: integer:= 20000;
        PULSE_LENGTH_CYCLES: integer:= 2000000
    );
    Port (
           clk_i     : in std_logic;
           reset_n_i : in std_logic;
           enc_a_i   : in std_logic;
           enc_b_i   : in std_logic;
           out_l_o   : out std_logic;
           out_r_o   : out std_logic
           );
end encoder;

architecture Behavioral of encoder is

type main_state_t is (IDLE, DETECTED, DEBOUNCE, WAITING);
signal main_state : main_state_t;
signal state_after_debounce : main_state_t;

type output_state_t is (IDLE, PULSE_START, PULSE, WAIT_AFTER_PULSE);
signal output_state : output_state_t;

-- direction detected by main state
signal detected_direction : std_logic; -- 0 -> left; 1 -> right

-- current output direction to be pulsed by output state.
-- set by detections_counter_process
signal output_direction : std_logic := '0';

signal fifo_not_empty : std_logic := '0';

signal enc_a_sync_s: std_logic_vector(1 downto 0);
signal enc_b_sync_s: std_logic_vector(1 downto 0);

begin
    out_l_o <= '1' when output_state = PULSE and output_direction = '0' else '0';
    out_r_o <= '1' when output_state = PULSE and output_direction = '1'else '0';

    synchronization_process : process(clk_i)
    begin
    if rising_edge(clk_i) then

            -- passing input to 1st flip-flop
            enc_a_sync_s(0) <= enc_a_i;
            enc_b_sync_s(0) <= enc_b_i;

            -- passing input to 2nd flip-flop - this is synchronized input (with 2 clock cycles delay)
            enc_a_sync_s(1) <= enc_a_sync_s(0);
            enc_b_sync_s(1) <= enc_b_sync_s(0);
    end if;
    end process;

    main_process : process(clk_i)
        variable counter : integer := 0;
    begin
        if rising_edge(clk_i) then
            if (reset_n_i = '0') then
                main_state <= IDLE;
                counter := 0;
            else
                case main_state is
                    when IDLE =>
                        main_state <= IDLE;
                        if enc_a_sync_s(1) = '0' then
                            if enc_b_sync_s(1) = '1' then
                                detected_direction <= '0';
                            else
                                detected_direction <= '1';
                            end if;
                            main_state <= DETECTED;
                        end if;
                    when DETECTED =>
                        state_after_debounce <= WAITING;
                        main_state <= DEBOUNCE;
                    when DEBOUNCE =>
                        main_state <= DEBOUNCE;
                        counter := counter + 1;
                        if counter = DEBOUNCE_CYCLES then
                            counter := 0;
                            main_state <= state_after_debounce;
                        end if;
                    when WAITING =>
                        main_state <= WAITING;
                        if enc_a_sync_s(1) = '1' and enc_b_sync_s(1) = '1' then
                            state_after_debounce <= IDLE;
                            main_state <= DEBOUNCE;
                        end if;
                end case;
            end if;
        end if;
    end process;

    detections_counter_process: process(clk_i)
        variable pulses_counter : integer := 0;
        variable pulses : std_logic_vector(31 downto 0) := (others => '0');
    begin
        if rising_edge(clk_i) then
            if (reset_n_i = '0') then
                fifo_not_empty <= '0';
                pulses_counter := 0;
```

Listing 1. cd.

```
                pulses := (others => '0');
            else
                if main_state = DETECTED then
                    pulses(pulses_counter) := detected_direction;
                    pulses_counter := pulses_counter + 1;
                end if;
                if output_state = PULSE_START then
                    output_direction <= pulses(0);
                    pulses := '0' & pulses(31 downto 1); --shift right
                    pulses_counter := pulses_counter - 1;
                end if;
                if pulses_counter > 0 then
                    fifo_not_empty <= '1';
                else
                    fifo_not_empty <= '0';
                end if;
            end if;
        end if;
    end process;

    output_process : process(clk_i)
        variable counter : integer := 0;
    begin
        if rising_edge(clk_i) then
            if (reset_n_i = '0') then
                output_state <= IDLE;
                counter := 0;
            else
                case output_state is
                    when IDLE =>
                        output_state <= IDLE;
                        if fifo_not_empty = '1' then
                            output_state <= PULSE_START;
                        end if;
                    when PULSE_START =>
                        output_state <= PULSE;
                    when PULSE =>
                        output_state <= PULSE;
                        counter := counter + 1;
                        if counter = PULSE_LENGTH_CYCLES then
                            counter := 0;
                            output_state <= WAIT_AFTER_PULSE;
                        end if;
                    when WAIT_AFTER_PULSE =>
                        output_state <= WAIT_AFTER_PULSE;
                        counter := counter + 1;
                        if counter = PULSE_LENGTH_CYCLES then
                            counter := 0;
                            output_state <= IDLE;
                        end if;
                end case;
            end if;
        end if;
    end process;

end Behavioral;
```

Listing 2. Wpis do pliku DTS

```
gpio-keys-polled-enc {
    compatible = “gpio-keys-polled”;
    #address-cells = <1>;
    #size-cells = <0>;
    poll-interval = <10>;
    btnenc {
        label = “btnenc”;
        gpios = <&gpio0 88 0>;
        linux,code = <28>; // enter
        debounce-interval = <50>;
    };
    encl {
        label = “encl”;
        gpios = <&gpio0 86 0>;
        linux,code = <105>; // left
        debounce-interval = <0>;
    };
    encr {
        label = “encr”;
        gpios = <&gpio0 87 0>;
        linux,code = <106>; // right
        debounce-interval = <0>;
    };
};
```

czyli pinów 1 – 4. Jeśli ktoś chce podłączyć ją inaczej, będzie musiał zmodyfikować plik *constraints*. Nasz enkoder wyposażony jest też w przycisk, a na płytce obecny jest też dodatkowy przełącznik. Oba zostaną podłączone do kolejnych pinów GPIO.

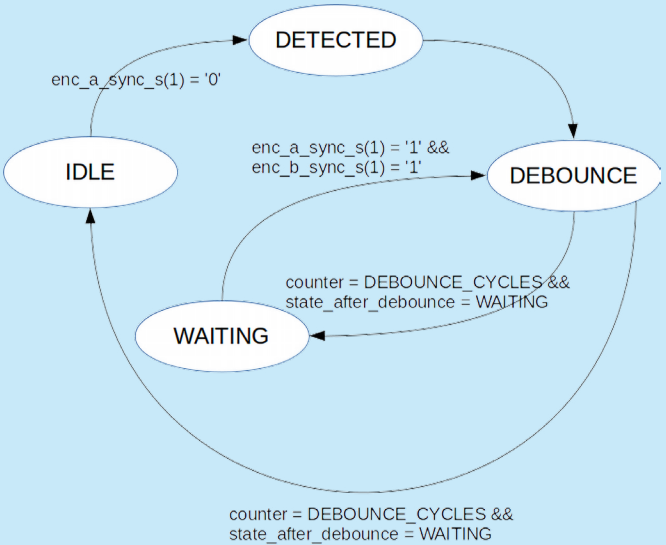
Kod VHDL

Na **listingu 1** pokazano kod VHDL bloku do obsługi enkodera. Są w nim obecne 4 procesy. Pierwszy z nich, nazwany *synchronization_process*, służy do synchronizacji sygnału wchodzącego do bloku. W dużym skrócie jest ona potrzebna, aby upewnić się, że sygnał pochodzący z enkodera pojawi się już w układzie synchronicznie ze zboczem zegara, tzn. nie w czasie trwania zbocza. W przeciwnym wypadku mogłoby wystąpić zjawisko metastabilności, które jest w układach cyfrowych bardzo niepożądane. Zainteresowanych odsyłam do szerszych opisów problemu.

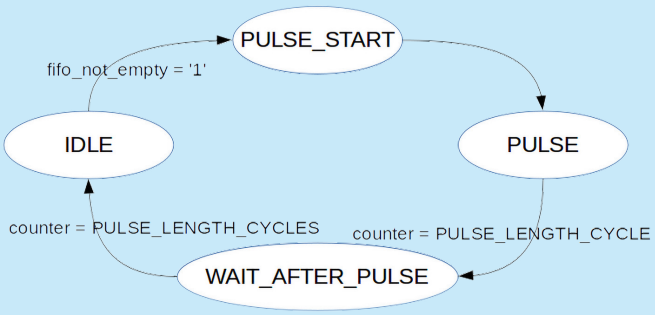
Kolejny proces o nazwie *main_process* to automat stanów odpowiadający za wykrywanie zdarzenia obrotu i jego kierunku. Gdy będąc w stanie IDLE wykryje obrót, przechodzi do stanu DETECTED, ustawiając jednocześnie sygnał *detected_direction* na '0' lub '1'. Przekazuje w ten sposób do kolejnego procesu informację o samym zdarzeniu, jak i o kierunku. W dalszych stanach tego procesu następuje debouncing i oczekiwanie na powrót do stanu wyjściowego, po którym ponownie następuje debouncing. Przejścia stanów tego automatu ilustruje **rysunek 6**.

Następny, wspomniany już, proces nazywa się *detections_counter_process*. Jego zadaniem jest zapisywanie do kolejki FIFO następujących zdarzeń i przekazywanie ich do ostatniego procesu – *output_process*. Ostatni proces to drugi automat stanów, którego zadaniem jest generowanie impulsów. Diagram jego przejść ilustruje **rysunek 7**. W stanie IDLE czeka, aż *detections_counter_process* zasygnalizuje (poprzez sygnał *fifo_not_empty*), że kolejka nie jest pusta i trzeba wygenerować impuls na jednym z wyjść bloku. Gdy automat przejdzie w stan PULSE_START, proces *detections_counter_process* dekrementuje licznik zdarzeń i usuwa najstarsze, poprzez przesunięcie bitów w rejestrze *pulses* będącym wspomnianą już kolejką FIFO. Stany PULSE i WAIT_AFTER_PULSE odpowiadają odpowiednio za wysłanie impulsu i odczekanie przed wysłaniem kolejnego.

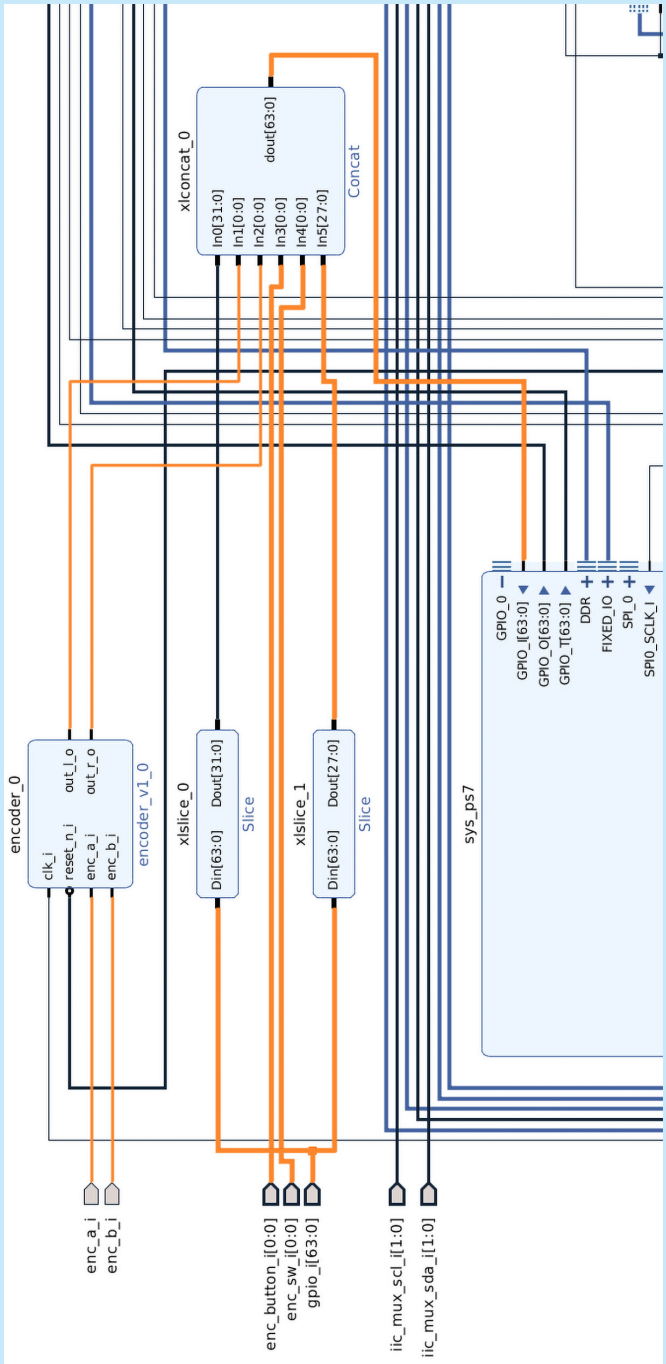
Kod z list. 1 znajduje się w pliku *enkoder.vhd*, zaś plik *contraints.xdc* zawiera ustawienia przypisania portów projektu do pinów FPGA W pliku *encoder_tb.vhd* znajduje się tzw. testbench, niezbędny w każdym projekcie tworzonym na układy FPGA. Wykonanie testu pozostawiam czytelnikowi.



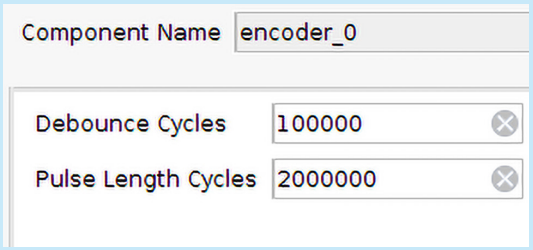
Rysunek 6. Przejścia stanów automatu obsługującego enkoder



Rysunek 7. Przejścia stanów automatu detections_counter_process



Rysunek 8. Schemat podłączeń automatów obsługujących enkoder w projekcie FPGA



Rysunek 9. Okno parametryzacji enkodera

Aby dodać blok do projektu referencyjnego, należy najpierw utworzyć nowy projekt np. o nazwie *Encoder*, dodać do niego źródła (plik *enkoder.vhd* i *encoder_tb.vhd*, BEZ pliku *contraints.xdc*) i spakować jako IPcore. Opis tego procesu znajduje się w poprzedniej części kursu. Po spakowaniu, dodajemy IPcore do block design'u. Wyprowadzenia A i B należy wyeksportować na zewnątrz, zaś L i R podłączyć do pinów GPIO. Dla ułatwienia warto sobie pomóc blokami slice (wyprowadza część szyny z sygnałami) i concat (łączy ze sobą szyny i pojedyncze sygnały). Trzeba też dodać porty sw i button i też podłączyć do GPIO. Po zakończeniu podłączania schemat powinien wyglądać jak na **rysunku 8**. Blok enkodera należy skonfigurować – należy ustawić liczbę cykli zegara, wyznaczającą czas trwania debouncingu i impulsu (**rysunek 9**). W tym przypadku zegar ma częstotliwość 100 MHz, więc jego okres wynosi 10 ns, w związku z czym czas debouncingu wynosi 1 ms (100000 cykli), a czas trwania impulsu to 20 ms (2000000 cykli).

Jak już wspomniałem, aby wykrywanie zdarzeń z poziomu Linuksa dobrze działało, warto użyć sterownika gpio-keys-polled. Domyślnie nie jest on wkompilowany w jądro, trzeba więc go dodać w opcjach konfiguracyjnych. Opis tego procesu pojawił się już w jednej z poprzednich części kursu. Sterownik można znaleźć w opcjach pod *Device Drivers* → *Input device support* → *Keyboards* → *Polled GPIO buttons*. Dokładniejszy opis tego sterownika i innych zagadnień związanych z GPIO na płytkach z Zynq można znaleźć na stronie Wiki Xilnx (***http://bit.ly/2I0c6rb***).

Aby sterownik zadziałał, trzeba też dodać odpowiedni wpis do DTS – zamieszczono go na **listingu 2**. Wpis dodajemy poniżej wpisu o przyciskach.

Po zbudowaniu Linuksa i skompilowaniu zmodyfikowanego DTS można przetestować enkoder. Do testów wykorzystamy pierwszą z uruchomionych już aplikacji, testującą przyciski, przełączniki i diody. Obrót enkodera będzie przyspieszał lub spowalniał częstotliwość migania diody.

Enkoder pojawi się w systemie plików jako jedno z urządzeń wejściowych, prawdopodobnie pod ścieżką */dev/input/event1*. Warto jednak to sprawdzić, metodą opisaną na wspomnianej już stronie Wiki Xilinx (komenda cat /dev/input/event1 | hexdump).

Podsumowanie

W tej części kursu udało się utworzyć kolejny dedykowany blok w strukturze FPGA. Dodaliśmy też nowy sterownik do Linuksa i wpis do DTS, dzięki któremu ten sterownik został zastosowany. Tym razem mogliśmy się zatrzymać też na kodzie VHDL i obsłudze podłączonego z zewnątrz do płytki sprzętu.

Po przetestowaniu enkodera można zauważyć, że ten sposób jego obsługi wprowadza ograniczenia w postaci opóźnienia reakcji programu, jak również okazjonalnie gubione są zdarzenia obrotu o jeden skok. Rozwiązaniem pierwszego problemu mogłoby być napisanie bloku tak, aby korzystał z systemu przerwań, zaś rozwiązaniem drugiego z nich mogłoby być zastosowanie analogowego filtra, lub jego cyfrowego odpowiednika, który być może pozwoliłby na wyeliminowanie drgań styków. Ten sposób obsługi enkodera z pewnością pozwoli już jednak na zaprojektowanie prostego interfejsu użytkownika w kolejnych projektach..

Stanisław Aleksiński

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