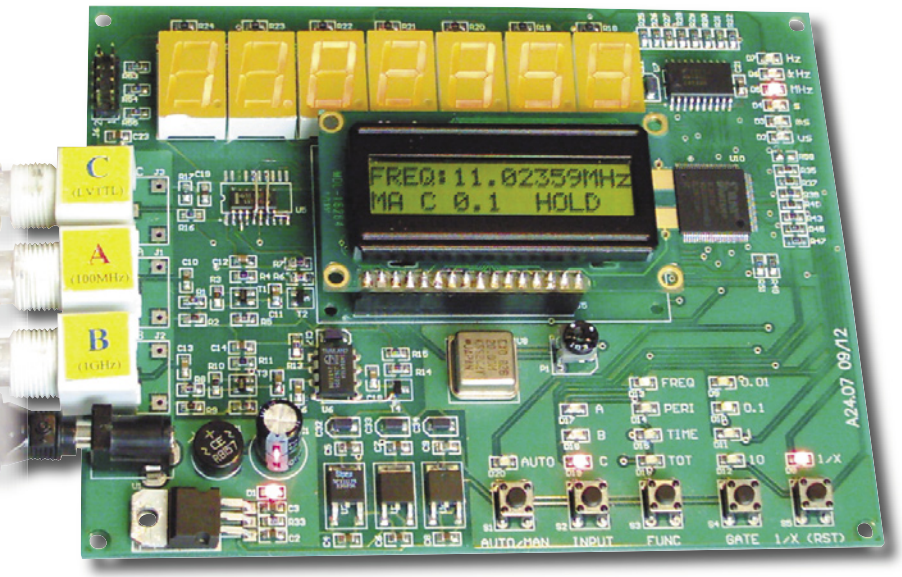
Uniwersalny miernik: częstotliwości, czasu, okresu na FPGA, część 3

AVT–5115

Jednym z przyrządów bardzo często wykorzystywanych w pracowni elektronika jest miernik częstotliwości. Tego typu przyrządy pomiarowe noszą czasem nazwę liczników uniwersalnych i oprócz pomiaru częstotliwości oferują również pomiar innych parametrów, takich jak okres, czas trwania impulsu, czy też całkowita liczba impulsów w pewnej grupie. W artykule przedstawiono opis budowy takiego licznika, do którego konstrukcji wykorzystano układy programowalne FPGA oraz język opisu sprzętu Verilog.



PODSTAWOWE PARAMETRY

- pomiar częstotliwości, okresu, czasu trwania impulsu (z funkcją stopera), całkowitej liczby impulsów,
- automatyczna zmiana zakresu,
- pomiary w trybie licznika konwencjonalnego i odwrotnego,
- podwójny wyświetlacz 7–cyfrowy, 7–segmentowy LED i/lub alfanumeryczny LCD 2x16 znaków,
- maksymalna rozdzielczość pomiaru częstotliwości: 10^{-6} Hz, czasu: 10 ns,
- trzy wejścia pomiarowe:
  - wejście A: zakres częstotliwości: 30 Hz…100 MHz, czułość S<75 mV (f_{in}=10 MHz), impedancja wejściowa Z_{in}>1,3 MΩ (f_{in}=1 kHz),
  - wejście B: zakres częstotliwości: 70 MHz...1 GHz, czułość (bez wtórnika T4) S≈10 mV, impedancja wejściowa Z_{in}=50 Ω,
  - wejście C: wejście cyfrowe LVTTL (z tolerancją 5 V), zakres częstotliwości od 0 do ok. 150 MHz
- zasilanie minimum 6 VAC lub 7,5 VDC

Opisany w poprzedniej części artykułu moduł dzielenia całkowitego jest wykorzystywany przez moduł obliczający odwrotność. Działanie tego ostatniego modułu opiera się na klasycznej metodzie dzielenia, takiej jak przy użyciu kartki papieru i długopisu.

Algorytm takiego dzielenia w postaci uproszczonej sieci działań ASM (*Algorithmic State Machine*) przedstawiono na **rys. 10**. W notacji ASM stany wewnętrzne układu sekwencyjnego (automatu) reprezentowane są za pomocą klatek operacyjnych (prostokątów), a warunki przejścia pomiędzy stanami opisują klatki decyzyjne (romby).

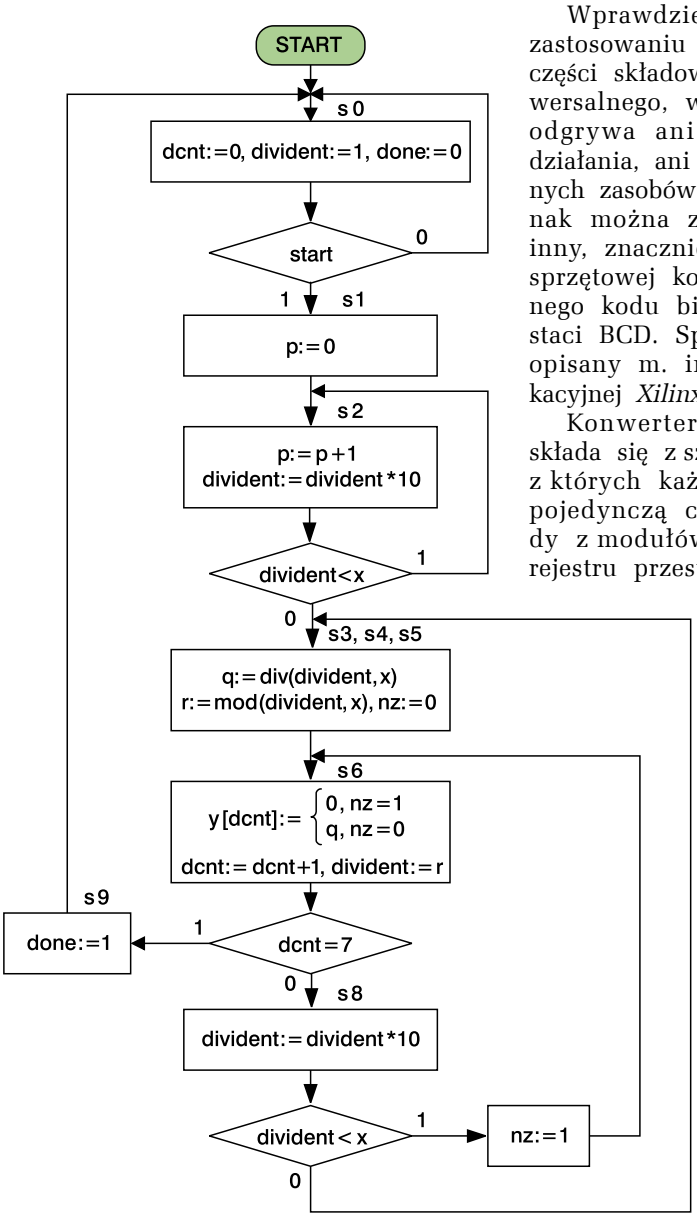
Algorytm rozpoczyna swoje działanie od wyznaczenia najmniejszej liczby naturalnej p spełniającej nierówność: $10^p \geq x$, gdzie x jest dzielnikiem – argumentem funkcji $f(x)=1/x$. Liczbę 10^p przyjmuje się początkowo jako dzielną (*divident*) i w kolejnych krokach wyznaczany jest całkowity iloraz q z dzielenia dzielnej i dzielnika x (operator *div*) oraz reszta r z tego dzielenia (operator *mod*), która jednocześnie staje się nową wartością dzielnej. Bieżący iloraz q stanowi pojedynczą cyfrę wyniku dzielenia $y=1/x$, zapisaną w kodzie BCD. W sieci działań z rys. 10 operację tę zapisano symbolicznie jako $y[dcnt]:=q$ (dla $nz=0$), gdzie $dcnt$ oznacza numer kolejnej cyfry wyniku. Dodatkowa zmienna nz przyjmuje wartość 1, gdy wartość dzielnej (czyli bieżącej reszty z dzielenia r) pomnożona przez 10 jest mniejsza od wartości dzielnika x. Sieć działań kończy swoją pracę powracając do stanu oczekiwania na sygnał rozpoczęcia kolejnych obliczeń, gdy wyliczona zostanie zadana liczba cyfr wyniku – w tym wypadku jest to 7 cyfr.

Odpowiedni kod w języku Verilog implementujący sieć działań z rys. 10 przestawiono na **list. 2**. Przyjęto, że szerokość słowa wejściowego x to 24 bity (dokładanie taka jak liczba bitów liczników 1 i 2 z rys. 7), a słowa wyjściowego y – 28 bitów (7 cyfr BCD). Odpowiednie nazwy symboliczne w opisie modułu z list. 2, włącznie z nazwami stanów automatu, korespondują bezpośrednio z nazwami zastosowanymi w sieci działań z rys. 10.

Moduł konwertera kodu

Klasyczny – programowy sposób konwersji kodu binarnego do postaci dziesiętnej opiera się na iteracyjnym dzieleniu przez 10 konwertowanej liczny i wyliczaniu reszty z tego dzielenia. Algorytm ten można byłoby zaimplementować również w sposób sprzętowy, zwłaszcza mając już gotowy moduł dzielenia całkowitego. Jednak jest to sposób nieoptymalny ze względu na stosunkowo dużą złożoność (ilość zasobów logicznych niezbędną do implementacji), jak również ze względu na szybkość konwersji (wymaganą liczbę taktów zegara).

W omawianym zastosowaniu konwertera, liczbę bloków logicznych niezbędną do implementacji można by zmniejszyć poprzez wprowadzenie współdzielenia zasobów modułu dzielenia całkowitego, który na przemian wykorzystywany byłby przez układ arytmetyczny i konwerter kodu. Z kolei zwiększenie szybkości konwersji możliwe byłoby do uzyskania poprzez wprowadzenie dedykowanego modułu dzielenia całkowitego przez 10, którego działanie oparte byłoby np. na rozwinięciu w szereg geometryczny liczby 1/10 – wówczas dzielenie odbywałoby się w jednym takcie zegara (chociaż byłoby to okupione pewnym wzrostem złożoności układu kombinacyjnego realizującego przesunięcia bitowe i dodawanie).



Rys. 10. Uproszczona sieć działań ASM układu obliczającego odwrotność

Wprawdzie w omawianym zastosowaniu konwertera jako części składowej licznika uniwersalnego, większej roli nie odgrywa ani szybkość jego działania, ani liczba zajmowanych zasobów logicznych, jednak można zastosować tutaj inny, znacznie lepszy sposób sprzętowej konwersji naturalnego kodu binarnego do postaci BCD. Sposób ten został opisany m. in. w nocie aplikacyjnej *Xilinx* – XAPP029.

Konwerter taki (**rys. 11**) składa się z szeregu modułów, z których każdy reprezentuje pojedynczą cyfrę BCD. Każdy z modułów jest rodzajem rejestru przesuwnego, przesuwającego bity w lewo. Informacja wejściowa (konwertowana liczba) wprowadzana jest szeregowo do pierwszego z modułów, reprezentującego najmniej znaczącą cyfrę BCD, począwszy od najstarszego bitu (MSB). Proces konwersji kończy się, gdy wszystkie bity konwertowanej liczby zostaną wprowadzone i wówczas uzyskana wartość w kodzie BCD jest dostępna na wyjściach kolejnych modułów w postaci równoległej.

Działanie pojedynczego modułu z rys. 11 opiera się na przesuwaniu w lewą stronę wszystkich bitów modułu wraz z każdym taktem zegara, przy czym wartość bitu z szeregowego wejścia MODIN umieszczana jest na najmniej znaczącej pozycji w słowie modułu. Jeżeli wartość tego słowa jest większa lub równa 5, wówczas w kolejnym takcie zegara bity nie są przesuwane, lecz następuje korekta wartości słowa zgodnie ze schematem: 5 zostaje zastąpione przez 0, 6 przez 2, 7 – 4, 8 – 6, 9 – 8 (wartość najmniej znaczącego bitu równa jest wartości wejścia MODIN). Jednocześnie w takim przypadku wyjście modułu MODOUT przyjmuje wartość 1.

Implementację takiego algorytmu konwersji w postaci modułu zawierającego opis behawioralny w języku Verilog przedstawiono na **list. 3**. Przy okazji praktycznie zilustrowano tutaj takie elementy języka Verilog, jak parametryzacja modułu (poprzez zmianę wartości kilku parametrów możliwy jest do uzyskania opis dla prawie dowolnej szerokości słowa wejściowego), zastosowanie funkcji, które są jednym ze sposobów opisu układu kombinacyjnego, wykorzystanie pętli oraz różnicę w działaniu przypisań proceduralnych: blokującego (operator „=” – przypisanie „natychmiastowe”) i nieblokującego (operator „<=” – przypisanie następuję podczas aktywnego zbocza zegara) wewnątrz bloku sekwencyjnego *always*.

Rozpoczęcie konwersji przez moduł wymaga ustawienia przez jeden takt zegara wejścia *start*. Konwersja trwa dokładnie *N* taktów zegara, gdzie *N* jest szerokością słowa wejściowego. Zakończenie konwersji sygnalizowane jest ustawieniem wyjścia *done*.

Synchronizator

Moduł synchronizatora wykorzystywany jest podczas pomiarów okresu i czasu trwania impulsu wejściowego. Jego zadaniem jest wysterowanie bramki G3 (rys. 7) w odpowiedzi na sygnał żądania wykonania pomiaru, pochodzący z układu sterującego. Bramka powinna być wówczas otwarta od

List. 2. Opis HDL modułu obliczającego odwrotność

```
module div1x(input clk,start,reset,
        input [23:0] x,
        output reg [27:0] y,
        output reg [3:0] p
              );

reg [3:0] state;
reg [27:0] divident;
wire [27:0] q,r;
wire comp_result;
reg done;
reg div_start;
wire div_done;
reg [2:0] dcnt;
reg nz;

reg [27:0] temp1,temp2;
wire [27:0] mult10result;

//nadanie wartości symbolicznym nazwom stanów automatu
parameter s0=0,s1=1,s2=2,s3=3,s4=4,s5=5,s6=6,s7=7,s8=8,s9=9;

//konkretyzacja modułu dzielenie całkowietego
divN5 m3(.clk(clk),.start(div_start),.divident(divident),
         .divisor({4'b0000,x}),.quotient(q),.remainder(r),.done(div_done));

//realizacja mnożenia x10 zawartości rejestru ,divident'
always @(divident)
begin temp1=divident<<3; temp2=divident<<1; end
assign mult10result=temp1+temp2;

//porównanie wyniku mnożenia z wartością x
assign comp_result=(mult10result<{4'b0000,x});

//realizacja automatu
always @(posedge clk)
begin
  if(~reset) state<=0;
  else
   case (state)
    s0: begin
     //nadanie wartości poczatkowych
          divident<=28'd1; done<=0; div_start<=0; dcnt<=3'd0; nz<=0;
     //oczekiwanie na uaktywnienie zewnętrznego sygnału ,start'
          if(start) state<=s1;
        end
    s1: begin p<=4'd0; state<=s2; end
    s2: begin
          //obliczanie najmniejszego p takiego, że 10^p>=x
          p<=p+1; divident<=mult10result;
          if(comp_result) state<=s2;
     else state<=s3;
        end
    s3: begin //aktywowanie modułu dzielenia całkowitego
      div_start<=1; state<=s4;
       end
    s4: begin
     div_start<=0; nz<=0; state<=s5;
     end
    s5: begin
     //oczekiwanie na zakończenie operacji dzielenia
     //jeżeli dzielenie całkowite wykonane przejście do stanu s6
     if(div_done) state<=s6;
     end
    s6: begin
      //aktualizacja rejestru wyniku (y - rejestr przesuwny)
      //q zawiera wynik dzielenia całkowitego: divident/x, r - resztę z tego dzielenia
      if (nz) y<={y[23:0],4'b0000};
      else begin y<={y[23:0],q[3:0]}; divident<=r; end
      dcnt<=dcnt+1; //zwiększenie licznika cyfr
      state<=s7;
     end
    s7: begin
     //jeżeli wyliczono już 7 cyfr - stop
     if(dcnt==3'd7) state<=s9;
     else state<=s8;
     end
    s8: begin
       //wykonanie mnożenia x10 uzyskanej reszty z dzielenia i zapamiętanie jej jako
     //dzielnej dla następnego dzielenia
     divident<=mult10result;
     //jeżeli wynik mnożenia mniejszy od x to bieżąca cyfra ilorazu częściowego to 0
     if(comp_result) begin nz<=1; state<=s6; end
     else state<=s3;
     end
    s9: begin
          done<=1; state<=s0;
     end
   endcase
end
endmodule
```

momentu gdy sygnał mierzony przyjmie wartość 0, do momentu, gdy w sygnale mierzonym wystąpi jeden pełny impuls i wraz z opadającym zboczem tego sygnału bramka G3 zostanie zamknięta.

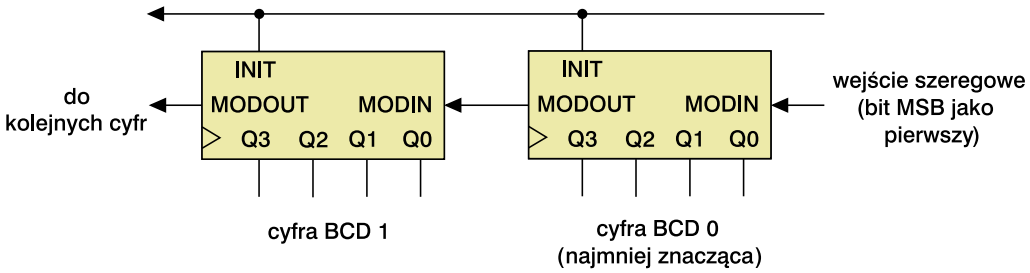
Takie sformułowanie sposobu działania synchronizatora wymusza jego budowę jako układu asynchronicznego. Gdyby synchronizator został zrealizowany jako układ synchroniczny, wówczas reagowałby na zmianę sygnału wejściowego i sygnału uzbrajającego (sygnału żądania wykonania pomiaru) jedynie w pewnych dyskretnych momentach czasu wyznaczonych sygnałem zegarowym. Ze względu na brak korelacji czasowej (synchronizacji) pomiędzy sygnałem mierzonym i sygnałem zegarowym, taki sposób działania układu byłby źródłem błędów pomiarowych o charakterze losowym, których wartość zależałaby od różnicy faz sygnałów wejściowego, mierzonego i uzbrajającego – w danej chwili czasu, a także od czasu wyjścia przerzutnika ze stanu metastabilnego, mogącego powstać na skutek niezachowania czasu ustalania danych tego przerzutnika.

Generalnie układy asynchroniczne posiadają kilka zalet w porównaniu z układami synchronicznymi. Są to m.in. większa szybkość działania (wydajność systemu asynchronicznego określana jest przez element o średnim czasie wykonania operacji, podczas gdy dla systemów synchronicznych jest to element najwolniejszy), brak problemów z dystrybucją sygnału zegarowego (*clock skew*, *clock jitter*), mniejszy pobór mocy (moc tracona jest tylko w elementach, które są w danej chwili „aktywne"), mniejsza emisja zakłóceń elektromagnetycznych oraz łatwość połączeń pomiędzy systemami taktowanymi zegarami o różnych częstotliwościach.

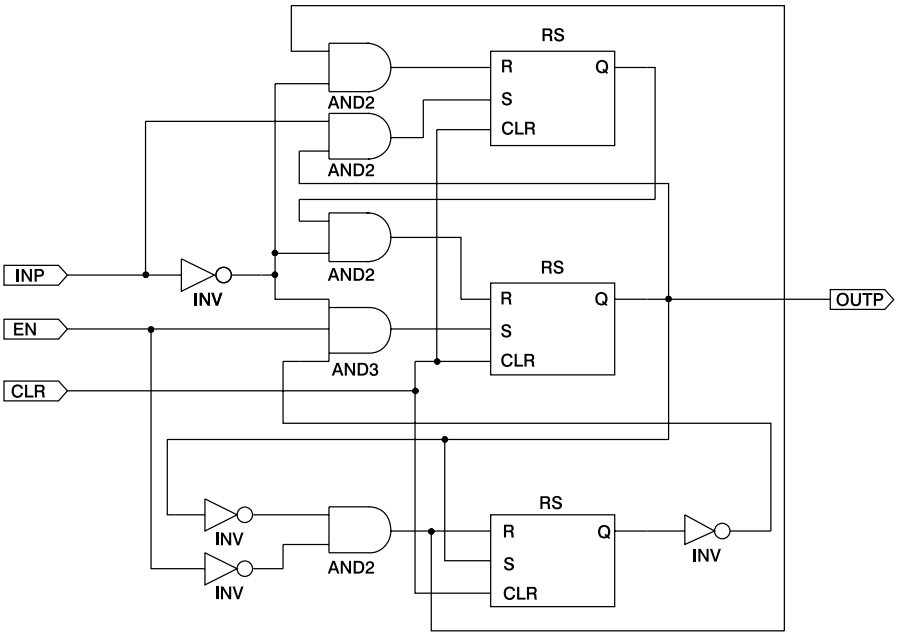
Istotnym problemem utrudniającym stosowanie układów asynchronicznych na szeroką skalę jest brak efektywnych metodologii projektowania i niedostatek narzędzi wspomagających projektowanie tego

typu układów. Projektując układy asynchroniczne należy wziąć pod uwagę szereg czynników, które nie są istotne w układach synchronicznych, takich jak występowanie hazardów, wyścigów, ograniczenia związane z przyjętym modelem czasowym, mechanizmy komunikacji pomiędzy poszczególnymi stopniami układu, itp. W związku z tym dość często zdarza się, że systemy asynchroniczne są bardziej złożone od funkcjonalnie zbliżonych systemów synchronicznych. Dlatego też znakomita większość obecnie projektowanych i eksploatowanych systemów jest oparta na układach synchronicznych.

Dodatkową trudnością w realizacji układów asynchronicznych w typowych, komercyjnych strukturach FPGA jest fakt, że zarówno sama architektura tych układów (np. nieprzewidywalność wyników implementacji w strukturze fizycznej), jak i narzędzia wspomagające proces projektowania (np. brak zapewnienia bezhazardowej realizacji układów kombinacyjnych, ograniczony wpływ projektanta na proces rozmieszczania i planowania połączeń), przeznaczone są do projektowania układów typowo synchronicznych. Choć możliwa jest specyfikacja układu asynchronicznego (np. za pomocą schematu lub w języku opisu sprzętu), to jednak jego implementacja może zawierać hazardy i wyścigi. W literaturze można jednak spotkać opisy realizacji, w komercyjnych strukturach FPGA wybranych układów asynchronicznych lub ich zasadniczych komponentów, takich jak układy koordynujące typu rendezvous (np. C–element Millera) czy arbiter. Jednak ich implementacja najczęściej wiąże się z koniecznością „ręcznej" ingerencji projektanta w proces rozmieszczania i planowania połączeń (*placement and routing*). Realizowane w ten sposób układy asynchroniczne należą prawie wyłącznie do klasy układów, które w kontekstowym tłumaczeniu można nazwać układami ze współpracą lokalną (*self–timed* – klasa układów asynchronicznych, która praktycznie nie jest opisywana w polskiej literaturze). Czasami spotyka się też realizację układów asynchronicznych z samotaktowaniem (*self–clocked*).



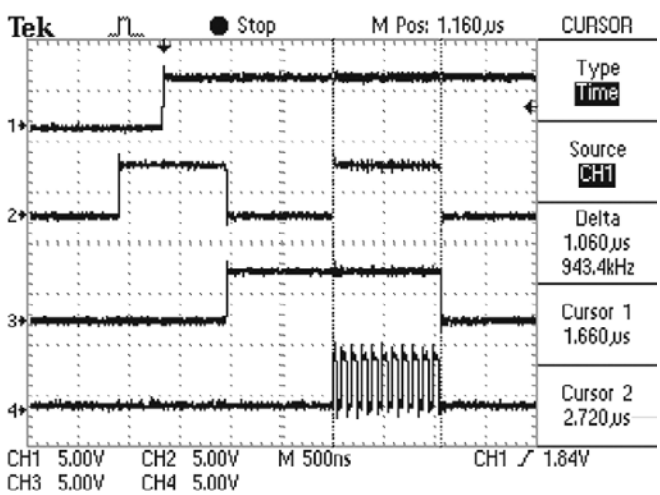
Rys. 11. Schemat blokowy konwertera kodu BIN –> BCD



Rys. 12. Schemat ideowy asynchronicznego układu synchronizatora

Niekiedy jednak, w pewnych szczególnych przypadkach, możliwa jest implementacja prostych układów asynchronicznych w strukturach FPGA bez szczególnych zabiegów ze strony projektanta i jego ingerencji w sposób działania narzędzi syntezy. Przykładem takiego układu jest właśnie opisywany tu synchronizator. Jako jedyny blok całego urządzenia został on wyspecyfikowany za pomocą edytora schematów w środowisku WebPack ISE. Schemat ideowy synchronizatora przedstawia rys. 12.

Synchronizator ma konstrukcję zbliżoną do klasycznej architektury układów asynchronicznych, dla której blok przejść jest podzielony na układ kombinacyjny i blok pamięci z asynchronicznymi, statycznymi przerzutnikami SR. W tego typu układach istotne jest aby realizacja funkcji wzbudzeń przerzutników SR nie zawierała hazardów oraz aby nie występowała symulta-



Rys. 13. Fragment oscylogramu ilustrujący działanie synchronizatora

List. 3. Opis modułu konwertera kodu BIN->BCD

```
module bin2bcd(clk,start,data_bin,data_bcd,done);
parameter NO_BITS_IN=24;
parameter NO_BCD_DIGITS=7;
parameter BIT_CNT_WIDTH=5;

input clk,start,init;
output done;
input [NO_BITS_IN-1:0] data_bin;
output [4*NO_BCD_DIGITS-1:0] data_bcd;
reg [4*NO_BCD_DIGITS-1:0] data_bcd;
reg [4*NO_BCD_DIGITS-1:0] temp_bcd;
reg [BIT_CNT_WIDTH-1:0] bit_cnt;
reg busy;
reg [NO_BITS_IN-1:0] bin;

//blok kombinacyjny: definicja funkcji - opisu działania modułów
//reprezentujących kolejne cyfry BCD
function [4*NO_BCD_DIGITS-1:0] bcd;
input [4*NO_BCD_DIGITS-1:0] din;
input modin;
integer i;
reg modout;
reg [3:0] digit,digit2;
begin
 //modin - zewnętrzne wejście szeregowe
 modout=modin;
 for (i=0;i<NO_BCD_DIGITS;i=i+1) //pętla
 begin //digit2 reprezentuje bieżące 4 bity słowa modułu BCD
  digit2[0]=din[4*i];
  digit2[1]=din[4*i+1];
  digit2[2]=din[4*i+2];
  digit2[3]=din[4*i+3];
  case(digit2) //jeżeli słowo >=5 - korekta wartosci
    4'd5: digit={3'b000,modout};
    4'd6: digit={3'b001,modout};
    4'd7: digit={3'b010,modout};
    4'd8: digit={3'b011,modout};
    4'd9: digit={3'b100,modout};
   //jeżeli słowo <5 wówczas przesuniecie bitów w lewo
    default: digit={digit2[2:0],modout};
  endcase
  //modout przyjmuje wartość 1 gdy słowo >=5
  modout=(digit2>4'b0100)?1'b1:1'b0;
  //wstawienie wyliczonych bitów pojedynczej cyfry BCD w odpowiednie
  //miejsce całego słowa reprezentującego wszystkie cyfry BCD
  bcd[4*i]=digit[0];
  bcd[4*i+1]=digit[1];
  bcd[4*i+2]=digit[2];
  bcd[4*i+3]=digit[3];
 end
end
endfunction

//blok sekwencyjny
always @(posedge clk)
begin
 if(init) data_bcd<=0;
 else
 begin
   if(start&~busy)
   begin
   busy<=1'b1; bin<=data_bin; bit_cnt<=NO_BITS_IN-1; temp_bcd=0;
   end

   if(busy)
   begin
   //"natychmiastowe" przypisanie blokujące
    temp_bcd=bcd(temp_bcd,bin[NO_BITS_IN-1]);

    bin<={bin[NO_BITS_IN-2:0],bin[NO_BITS_IN-1]};
    if(bit_cnt!=0) bit_cnt<=bit_cnt-1;
    else begin
          busy<=1'b0;
         //dzięki poprzedniemu przypisaniu blokującemu do temp_bcd
         //wartość ta może być użyta do aktualizacji wyjścia data_bcd
         //w bieżącym takcie zegara
           data_bcd<=temp_bcd;
        end
   end
 end
end
assign done=~busy;
endmodule
```

niczna zmiana stanów na obu wejściach przerzutnika, jak również, aby oba wejścia przerzutnika nie znajdowały się jednocześnie w stanie aktywnym.

Z punktu widzenia implementacji w FPGA najtrudniejszy do spełnienia jest warunek pierwszy. Jednak w przypadku synchronizatora, funkcje wzbudzeń przerzutników są na tyle proste (maksymalnie 3–wejściowe), że do implementacji każdej z tych funkcji wystarcza jedna tablica LUT (*Look Up Table* – generator funkcji) wewnątrz wybranego układu FPGA. Implementacja dowolnej funkcji kombinacyjnej za pomocą pojedynczej tablicy LUT jest zawsze bezhazardowa. Spełnienie pozostałych warunków gwarantuje odpowiednia postać funkcji wzbudzeń.

W ten sposób zaimplementowany moduł synchronizatora (bez stosowania specjalnych wymuszeń projektanta) został poddany najpierw szczegółowym symulacjom czasowym (wykorzystującym model czasowy układu uzyskany po syntezie i implementacji w blokach logicznych FPGA) a następnie drobiazgowym testom w fizycznym układzie. Wszystkie eksperymenty potwierdzały prawidłowe działanie synchronizatora.

Na **rys. 13** przedstawiono rzeczywisty oscylogram ilustrujący działanie synchronizatora jako bloku miernika częstotliwości. Kanał pierwszy oscyloskopu ilustruje przebieg sygnału żądania wykonania pomiaru, kanał drugi to wejściowy przebieg mierzony, kanał trzeci – wyjście synchronizatora – sterowanie bramką G3, kanał 4 – impulsy na wyjściu bramki głównej (G1) miernika (sygnały mają amplitudę TTL, gdyż zostały pobrane ze złącza J5, które jest sterowane przez bufor U11 zasilany napięciem 5 V).

Zbigniew Hajduk
zhajduk@prz-rzeszow.pl