Projektowanie PLD/FPGA z zestawem Lattice MachXO2 Pico Development Kit (3)

Pod koniec pierwszego kwartału 2011 roku Lattice wprowadził do produkcji seryjnej nową serię układów CPLD o nazwie MachXO2. Produkowane w procesie 65 nm, bazujące na technologii Flash układy MachXO2, w porównaniu ze swoim poprzednikiem – rodziną MachXO, charakteryzują się 3-krotnym zwiększeniem zasobów logicznych, 10-krotnym zwiększeniem pojemności wbudowanej pamięci RAM, więcej niż 100-krotnym zmniejszeniem poboru mocy (statyczny pobór mocy wynosi zaledwie 19 mikrowatów) oraz 30% spadkiem ceny. Oznacza to, że układy MachXO2, o zasobach logicznych sięgających 7000 tablic LUT, mogą swobodnie konkurować z „mniejszymi" układami FPGA rodzin Actel IGLOO, Altera Cyclone czy Xilinx Spartan. Wraz z nowymi układami Lattice udostępnił również zestaw ewaluacyjny MachXO2 Pico, umożliwiający praktyczne poznanie najważniejszych właściwości układów rodziny MachXO2. Celem niniejszego kursu jest zilustrowanie sposobu projektowania stosunkowo nieskomplikowanych systemów cyfrowych, opartych na nowoczesnych układach programowalnych, z wykorzystaniem zestawu MachXO2 Pico.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 15031, pass: 40nep417
• pierwsza i druga część kursu

Kontynuujemy kurs rozpoczęty w listopadowym numerze Elektroniki Praktycznej. W tej części wyjaśnimy działanie pojemnościowych pól dotykowych oraz zbudujemy miernik refleksu.

Przyciski pojemnościowe

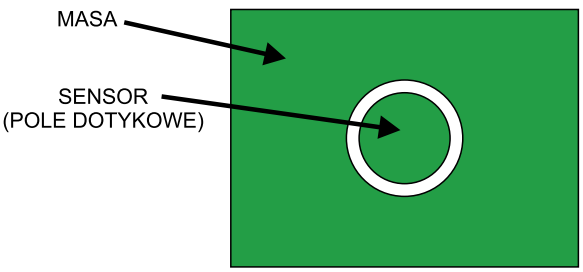
Płyta zestawu ewaluacyjnego MachXO2 Pico zawiera 4 pojemnościowe pola dotykowe – przyciski pojemnościowe. Podobnego typu przyciski są obecnie coraz częściej spotykane w sprzęcie audiowizualnym, szczególnie odbiornikach telewizyjnych i monitorach komputerowych. Stanowią one alternatywę dla klasycznych rozwiązań z przyciskami mechanicznymi. Główną zaletą przycisków pojemnościowych jest to, że w przeciwieństwie do przycisków mechanicznych, nie występuje zjawisko mechanicznego zużycia zestyków, niedokładnego kontaktu itp. Jednak praktyczne wykorzystanie przycisków pojemnościowych wiąże się z koniecznością stosowania dodatkowego układu elektronicznego determinującego moment zbliżenia palca użytkownika do pojemnościowego pola dotykowego. Pokażemy teraz jak taki układ skonstruować wykorzystując struktury programowalne CPLD/FPGA.

Istotą każdego systemu pojemnościowego jest oddziaływanie pola elektrycznego z układem przewodników. Typowym przykładem jest tutaj kondensator składający się z dwóch równoległych metalowych płytek. W takim kondensatorze linie sił pola elektrycznego skierowane są prostopadle do płytek, a zasadnicza część energii skoncentrowana jest pomiędzy tymi płytkami. Jednak w pobliżu brzegów okładek kondensatora następuje „wyciek" pola elektrycznego na zewnątrz. Linie sił pola elektrycznego tego wycieku nazywa się polem przybrzeżnym (*fringing field*). W konstrukcji przycisków pojemnościowych pole przybrzeżne odgrywa zasadniczą rolę i jest ono kierowane w aktywny obszar dotykowy, dostępny dla użytkownika.

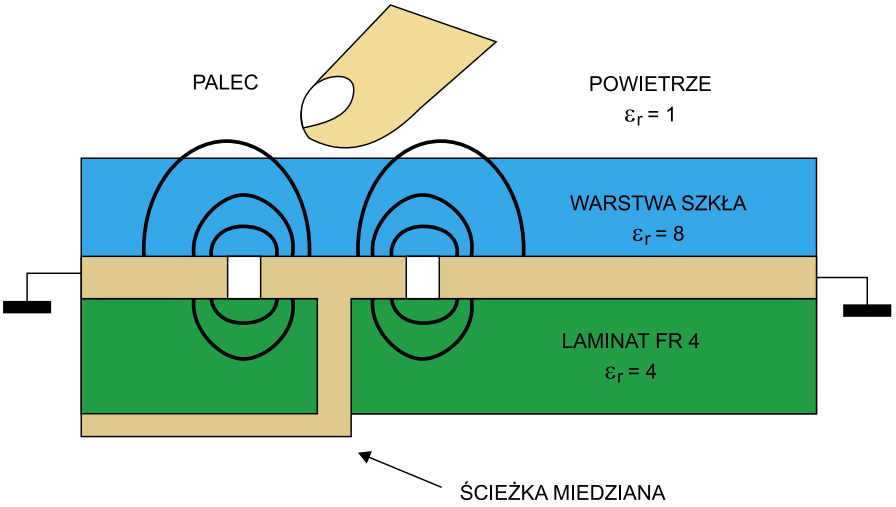
Typowo przyciski pojemnościowe mają kształt kół utworzonych z miedzi na górnej warstwie płytki drukowanej, o średnicy około 10 mm (średnica czubka palca dorosłej osoby), otoczonych płaszczyzną masy – rys. 27. Szerokość przerwy pomiędzy masą a aktywną częścią przycisku jest istotnym parametrem, który wymaga odpowiedniego doboru. Zazwyczaj wartość ta wynosi około 0.5 mm. Przekrój poprzeczny fragmentu płytki drukowanej z pojemnościowym polem dotykowym pokazano na rys. 28. Na rysunku tym czarnym kolorem zaznaczono również linie sił pola przybrzeżnego. Zbliżenie palca w okolice występowania pola przybrzeżnego powoduje pojawienie się powierzchni przewodzącej, oddziaływującej z tym polem (tkanki ludzkiego ciała wypełnione są przewodzącymi elektrolitami, na zewnątrz otoczonymi skórą – stratnym dielektrykiem). W wyniku czego pojemność całego układu nieznacznie wzrasta (zazwyczaj o kilka procent). Zmiana pojemności występuje podczas zbliżenia palca na pewną odległość od pola dotykowego. Czym ta odległość jest mniejsza, tym zmiana pojemności jest oczywiście większa. W praktyce można osiągnąć poprawne działanie przycisków pojemnościowych, w których pole dotykowe oddzielone jest nawet 10 mm warstwą szkła lub pleksiglasu (monitory komputerowe itp.). W przypadku zestawu MachXo2 Pico pole dotykowe od palca użytkownika oddziela jedynie cienka warstwa maski przeciwlutowniczej na płytce drukowanej.

Bardziej szczegółowe informacje na temat przycisków pojemnościowych można znaleźć np. w artykule „The art of capacitive sensing" autorstwa Marka Lee z Cypress Semiconductors Corp., dostępnym w internecie.

Zadaniem układu elektronicznego obsługującego przycisk pojemnościowy jest wy-



Rysunek 27. Widok z góry płytki PCB z polem dotykowym



Rysunek 28. Przekrój poprzeczny płytki z rysunku 25

krycie zmiany pojemności pola dotykowego podczas zbliżenia palca użytkownika. Dość oczywistym sposobem rozwiązanie tego problemu jest zbudowanie oscylatora, którego częstotliwość wytarzanego przebiegu zależy od pojemności pola dotykowego, a następnie dokonanie pomiaru częstotliwości i jej klasyfikacja (przycisk pojemnościowy naciśnięty lub zwolniony). Wykorzystując układy programowalne taki oscylator można zbudować praktycznie bez żadnych elementów zewnętrznych, a sam pomiar częstotliwości jest już zadaniem stosunkowo prostym.

Listing 7. Obsługa przycisku pojemnościowego

```
module cap_sens(input RST_IN,
       output LCD_COM0, LCD_COM1, LCD_COM2, LCD_COM3,
       output LCD_SEG0_D1, LCD_SEG1_D1,
       output LCD_SEG0_D2, LCD_SEG1_D2,
       output LCD_SEG0_D3, LCD_SEG1_D3,
       output LCD_SEG0_D4, LCD_SEG1_D4,
       inout  cap_btn1 );

`include "clk_rst_lcd.v"

// oscylator
    reg OscReg1;
    always @(posedge CLK)
   if(~RST) begin OscReg1 <= 0; end
   else OscReg1 <= ~cap_btn1;

   assign cap_btn1 = ~OscReg1?1'bz:OscReg1;
// koniec opisu oscylatora

// szerokość histerezy
 parameter Sensitivity = 10;

reg [6:0] FCnt1;
reg [6:0] FCnt1Smpl;
reg [7:0] SmplCntr;
wire SmplCntrCO = &SmplCntr;

// proces 1
always @(posedge CLK)
if(~RST) begin FCnt1<=0; end else
begin
 if(SmplCntrCO) FCnt1<=0;
 else
 if(OscReg1) FCnt1<=FCnt1+1;
end

// jeżeli touched1=1 - przycisk dotknięty
reg touched1;

//proces 2
always @(posedge CLK)
if(~RST)
begin
 touched1<=0;  SmplCntr<=0; FCnt1Smpl<=1;
end else
begin
 SmplCntr<=SmplCntr+1;
 if(SmplCntrCO)
 begin
   if (FCnt1Smpl<FCnt1-Sensitivity)
    begin touched1<=0; // palec oddalony
    FCnt1Smpl<=FCnt1; // zachowanie próbki
    end else
   if (FCnt1Smpl>FCnt1+Sensitivity)
    begin touched1<=1; // palec zbliżony
    FCnt1Smpl<=FCnt1; // zachowanie próbki
    end
 end
end

// gdy przycisk dotknięty, LCD wyświetla 8888
assign lcd1=touched1?127:0;
assign lcd2=touched1?127:0;
assign lcd3=touched1?127:0;
assign lcd4=touched1?127:0;

endmodule
```

Spójrzmy na kod pokazany na **listingu 7**. Zasadniczym elementem opisu jest realizacja oscylatora, którego częstotliwość zależy od wypadkowej pojemności dołączonej do dwukierunkowego portu (końcówki układu MachXO2) *cap_btn1*. Na płycie zestawu MachXO2 Pico do końcówki tej bezpośrednio podłączone jest pole dotykowe 1. Bardzo istotną rzeczą jest tutaj zapewnienie by na tej końcówce, w czasie gdy port jest skonfigurowany jako wejście lub gdy występuje na nim stan wysokiej impedancji, występował zawsze poziom niski (*pull down*). Można to osiągnąć przez odpowiedni zapis w pliku wymuszeń projektanta (*.lpf), np. dodając następującą linię kodu:

IOBUF PORT "cap_btn1" IO_TYPE=LVTTL PULLMODE=DOWN;

Na **rysunku 29** pokazano szczegółowe przebiegi czasowe występujące na zewnętrznej końcówce układu PLD do której podłączone jest pole dotykowe (*cap_btn1*) oraz zmiennej wewnętrznej *OscReg1*. Po przejściu sygnału zerującego *RST* w nieaktywny stan wysoki, zmienna *OscReg1* przyjmuje wartość 0, a tym samym port *cap_btn1* znajduje się w stanie wysokiej impedancji (przypisanie ciągłe w ostatniej linii sekcji kodu oscylatora). Ponieważ jednak port ten jest skonfigurowany jako *pull down* oznacza to, że na zewnętrznej końcówce układu PLD występuje poziom niski. Jednak wraz z każdym narastającym zboczem sygnału zegarowego zmienna *OscReg1* przyjmuje zanegowaną wartość odczytaną z portu *cap_btn1*, co oznacza, że podczas pierwszego zbocza zegara zmienna ta przyjmie wartość 1. Tym razem wspomniane przypisanie ciągłe w ostatniej linii sekcji kodu oscylatora z list. 6 sprawi, że na porcie *cap_btn1* pojawi się ten sam poziom, który zawiera zmienna *OscReg1*, czyli w tym przypadku poziom wysoki. Podczas kolejnego narastającego zbocza zegara zmienna *OscReg1* powróci do stanu niskiego (bo na *cap_btn1* jest poziom wysoki), a tym samym port *cap_btn1* przejdzie następnie w stan wysokiej impedancji. Poprzednio występujący na końcówce *cap_btn1* poziom wysoki spowodował naładowanie pojemności połączonej z tą końcówką (pojemności pola dotykowego). Przejście portu sterującego końcówką *cap_btn1* w stan wysokiej impedancji z jednoczesnym uaktywnieniem obwodu wymuszającego poziom niski, mającego charakter źródła prądowego typu ujście (*sink in*), spowoduje że na tej końcówce pojawi się przebieg liniowo opadający (rys. 29). W momencie gdy napięcie na tej końcówce spadnie poniżej dolnego poziomu progowego dla stanu niskiego (na rys. 29 oznaczonego jako Vt–) i jednocześnie wystąpi narastające zbocze zegara, zmienna *OscReg1* przejdzie w stan wysoki i cały proces będzie się powtarzał. Warto tu również dodać, że bufory wejściowe skojarzone z końcówkami układów MachXO2 zachowują się podobnie jak wejścia z przerzutnikiem Schmitta – wykazują histerezę, której szerokość również można konfigurować poprzez odpowiednie zapisy w pliku wymuszeń (możliwe są dwie wartości: histereza mała lub duża).

W wyniku opisanego wyżej procesu, zmienna *OscReg1* będzie oscylowała pomiędzy wartościami 0 i 1 z częstotliwością zależną od pojemności dołączonej do końcówki *cap_btn1*. Kolejna sekcja kodu na list. 7 służy do pomiaru częstotliwości tych oscylacji, a dokładniej – określany jest moment, gdy ta częstotliwość spada (zbliżenie palca do pola dotykowego). Pomiar częstotliwości oscylacji odbywa się poprzez zliczanie impulsów

Listing 8. Opis 16-bitowego rejestru LFSR typu Galois

```
module LFSR16(input CLK, RST,
              output [15:0] Q);

(*init="1101101010001101"*) reg [15:0] sr_reg;

wire x1=sr_reg[14]^sr_reg[0];
wire x2=sr_reg[13]^sr_reg[0];
wire x3=sr_reg[11]^sr_reg[0];

always @(posedge CLK)
if(~RST) begin sr_reg<=705; end
else sr_reg<={sr_reg[0],sr_reg[15],x1,x2,
              sr_reg[12],x3,sr_reg[10:1]};

assign Q=sr_reg;

endmodule
```

OscReg1 przez określony czas. Czas ten wyznaczony jest przez zapełnienie 8-bitowego licznika *SmplCntr* (wszystkie bity ustawione w stan wysoki, sygnał *SmplCntCO* – w stanie wysokim). Zliczaniem tych impulsów zajmuje się proces 1 z instrukcją *always* na list. 7. Liczba impulsów zawarta jest w zmiennej *FCnt1*.

Drugi proces z instrukcją *always*, podczas gdy sygnał *SmplCntCO* znajduje się w stanie wysokim, sprawdza czy bieżąca liczba zliczeń oscylacji zmiennej *OscReg1*, pomniejszona o pewną wartość (parametr *Sensitivity*) mającą interpretację szerokości histerezy, jest większa niż poprzednio zapamiętana liczba zliczeń. Jeżeli tak oznacza to, zwiększenie częstotliwości oscylacji, a tym samym oddalenie palca użytkownika od pola dotykowego. Jednocześnie zapamiętywana jest bieżąca liczba zliczeń. Analogicznie, zbliżenie palca do pola dotykowego spowoduje, że bieżąca liczba zliczeń, powiększona o pewną wartość (*Sensitivity*), będzie mniejsza od poprzednio występującej. Prawidłowa reakcja na zbliżanie i oddalenie palca od pola dotykowego wymaga odpowiedniego dobrania parametru *Sensitivity*. W przypadku płytki zestawu MachXO2 Pico, dobrą wartością tego parametru jest wartość około 10.

Działanie projektu z list. 7, zaimplementowanego na płycie zestawu MachXO2 Pico, sprowadza się do wyświetlanie samych ósemek na wyświetlaczu LCD, w momencie gdy wykryty zostanie fakt zbliżenia palca do pola dotykowego. Oddalenie palce powoduje wygaszenie wyświetlacza.

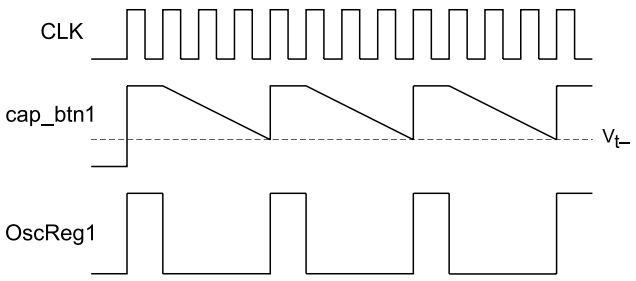
Projekt: miernik refleksu

Pokażemy teraz stosunkowo prosty projekt, którego zasadniczą funkcją jest pomiar czasu reakcji użytkownika. Na wyświetlaczu LCD będą w sposób pseudolosowy pojawiać się cyfry od 1 do 4, w również losowych odstępach czasu. Zadaniem użytkownika będzie jaknajszybsze dotknięcie przycisku pojemnościowego, odpowiadającego wyświetlanej cyfrze. Po określonej liczbie prób na wyświetlaczu pojawi się średni czas reakcji użytkownika, prezentowany z dokładnością do jednej setnej sekundy (10 ms).

Ponieważ założyliśmy, że cyfry będą pojawiać się w sposób losowy, do realizacji naszego projektu będziemy potrzebować generatora liczb losowych. Realizacja w układach programowalnych rzeczywistego generatora liczb losowych TRNG (*true random number generator*) jest oczywiście możliwa, jednak wiąże się z pewną komplikacją układu (temat generatorów TRNG implementowanych w strukturach FPGA jest przedmiotem badań i publikacji naukowych – generatory takie mają zastosowanie w układach kryptograficznych). Dużo prostsza jest realizacja generatora pseudolosowego, którego wyjście jest określone deterministycznie, chociaż rozkład liczb wyjściowych przypomina rozkład losowy. W przypadku naszego projektu nie ma to jednak żadnego znaczenia. Jedną z możliwości realizacji generatora liczb pseudolosowych jest wykorzystanie rejestru przesuwającego z liniowym sprzężeniem zwrotnym LFSR (*Linear Feedback Shift Register*). Rejestr taki charakteryzuje się tym, że jego bit wejściowy jest liniową funkcją poprzedniej zawartości rejestru. Realizacja takiego rejestru jest bardzo podobna do obliczania wielomianów CRC, przy czym w rejestrze LFSR inne jest źródło bitu wejściowego. W praktyce najczęściej spotka się rejestry LFSR typu Fibonacciego oraz Galois.

Na list. 8 pokazano kod modułu realizującego 16-bitowy rejestr LFSR typu Galois. Istotną rzeczą jest tutaj zainicjowanie rejestru wartością początkową (tzw. zarodkiem – *seed*) po zerowaniu lub włączeniu zasilania. W naszym przypadku, w czasie zerowania, do rejestru LFSR wpisywana jest wybrana arbitralnie wartość. Warto również dodać, że podobny moduł opisujący rejestr LFSR można również uzyskać wykorzystując aplikację *IPexpress* generatora wirtualnych komponentów pakietu *Lattice Diamond*, w sposób analogiczny jak zrobiliśmy to w przypadku bloku EFB (rys. 25).

Oprócz rejestru LFSR, drugim istotnym składnikiem naszego projektu jest moduł odmierzający zadany interwał czasowy. Moduł ten będzie wykorzystywany zarówno do celów pomiaru czasu reakcji użytkownika, jak również w momencie oczekiwania na wylosowanie i wyświetlenie określonej cyfry. Kod takiego modułu pokazano na list. 9, a opis jego portów przedstawia tab. 8. Dla osiągnięcia rozdzielczości pomiaru czasu równej 10ms, moduł powinien być taktowany zegarem o częstotliwości 12 MHz.



Rysunek 29. Przebiegi czasowe dla oscylatora z listing 6

Tabela 8. Opis portów modułu z listingu 8

Moduł: DELAY

Port	Kierunek	Liczba bitów	Opis
CLK12MHZ	wejście	1	Sygnał zegarowy o częstotliwości 12 MHz.
RST	wejście	1	Sygnał zerujący (aktywny poziom niski).
START	wejście	1	Poziom wysoki przez co najmniej jeden takt zegara inicjuje odmierzanie czasu.
DEL	wejście	10	Stała czasowa. Jedna jednostka jest równa 10 ms (dla zegara 12 MHz).
DONE	wyjście	1	Zakończenie odmierzania czasu jest sygnalizowane ustawieniem wyjścia w stan wysoki przez jeden takt zegara.

Listing 9. Moduł odmierzający zadany interwał czasowy

```
module DELAY(input CLK12MHZ,RST,START,
             input [9:0] DEL,
             output reg DONE);

reg [16:0] cnt;
reg [9:0] cnt2;
reg [1:0] ST;

always @(posedge CLK12MHZ)
if(~RST) begin ST<=0; cnt<=0; DONE<=0; end
else
 case(ST)
  0: begin cnt2<=0; DONE<=0; if(START) ST<=1; end
  1: begin cnt<=0;
      if(DEL==cnt2) begin DONE<=1; ST<=0; end else ST<=2;
     end
  2: begin if (cnt<120000) cnt<=cnt+1; else
           begin ST<=1; cnt2<=cnt2+1; end
     end
 endcase
 endmodule
```

Listing 10. Moduł główny projektu miernika refleksu

```
module cap_sens_demo(input RST_IN, CLK12MHZ,
        output LCD_COM0, LCD_COM1, LCD_COM2, LCD_COM3,
        output LCD_SEG0_D1, LCD_SEG1_D1,
        output LCD_SEG0_D2, LCD_SEG1_D2,
        output LCD_SEG0_D3, LCD_SEG1_D3,
        output LCD_SEG0_D4, LCD_SEG1_D4,
        inout cap_btn1, cap_btn2, cap_btn3, cap_btn4);

wire touched1, touched2, touched3, touched4;
reg [7:0] ld1,ld2,ld3,ld4;
wire [3:0] dgt1,dgt2,dgt3,dgt4;

`include „clk_rst_lcd.v"

reg sw_lcd;
assign lcd1=sw_lcd?lcd_encode(dgt1):ld1;
assign lcd2=sw_lcd?(lcd_encode(dgt2)|8'h80):ld2;
assign lcd3=sw_lcd?lcd_encode(dgt3):ld3;
assign lcd4=sw_lcd?lcd_encode(dgt4):ld4;

  // oscylator dla pojemnościowych pól dotykowych
    reg OscReg1, OscReg2, OscReg3, OscReg4;

   always @(posedge CLK)
   if(~RST) begin
      OscReg1 <= 0; OscReg2 <= 0;
      OscReg3 <= 0; OscReg4 <= 0;
   end else
   begin
      OscReg1 <= ~cap_btn1;
      OscReg2 <= ~cap_btn2;
      OscReg3 <= ~cap_btn3;
      OscReg4 <= ~cap_btn4;
   end

   assign cap_btn1 = ~OscReg1?1'bz:OscReg1;
   assign cap_btn2 = ~OscReg2?1'bz:OscReg2;
   assign cap_btn3 = ~OscReg3?1'bz:OscReg3;
   assign cap_btn4 = ~OscReg4?1'bz:OscReg4;
 //

// instancja modułu detekcji zmiany częstotliwości
oscylatorów
CAPSENSE cs1 (.CLK(CLK),.RECAL(RST),
  .OscReg1(OscReg1),.OscReg2(OscReg2),.OscReg3(OscReg3),
  .OscReg4(OscReg4),.touched1(touched1),.
touched2(touched2),
  .touched3(touched3),.touched4(touched4));

wire [15:0] RandValue;
// instancja rejstru LFSR
LFSR16 randomg(.CLK(CLK),.RST(RST),.Q(RandValue));

reg [3:0] ST;
reg [9:0] rv1,rv2;
reg go;
reg [1:0] rnd;
reg [3:0] tcnt;
wire tdone;

// synchronizacja z zegarem zewnętrznym (12MHz)
always @(posedge CLK12MHZ)
begin
 rv1<=RandValue[9:0]; rv2<={2'b00,rv1[7:0]};
end

reg [9:0] tc,rtime;
reg [15:0] srtime;
reg b2d_start;
wire b2d_done;

// automat sekwencyjny odpowiedzialny za główną
funkcjonalność
always @(posedge CLK12MHZ)
if(~RST) begin ST<=0;  end
else
 case (ST)
  0: begin sw_lcd<=0; ld1<=64; ld2<=64; ld3<=64; ld4<=64;
           tcnt<=0; b2d_start<=0; srtime<=0;
           if(touched1|touched2|touched3|touched4)  ST<=1;
     end
  1: begin
      ld1<=0; ld2<=0; ld3<=0; ld4<=0;
      if(~(touched1|touched2|touched3|touched4))
      begin tc<=rv2; go<=1; ST<=2; end
     end
  2: begin
     go<=0; if(tdone) begin ST<=3; rnd<=rv2[1:0]; end
     end
  3: begin
     if(rnd==0) ld1<=48;
     else if(rnd==1) ld2<=109;
     else if(rnd==2) ld3<=121;
     else if(rnd==3) ld4<=114;
     tc<=1; rtime<=0; tcnt<=tcnt+1; go<=1; ST<=4;
    end
  4: begin go<=0; if(tdone) ST<=5; end
  5: begin rtime<=rtime+1;
      if((rnd==0&touched1)|(rnd==1&touched2)|
         (rnd==2&touched3)|(rnd==3&touched4))
      begin srtime<=srtime+rtime;
         ST<=6; end else begin go<=1; ST<=4; end
     end
```

Na **listingu 10** pokazano kod modułu głównego projektu. W części deklaracyjnej modułu, oprócz znanych już portów wejścia – wyjścia, związanych z wyświetlaczem LCD oraz pojemnościowymi polami dotykowymi, pojawił się też dodatkowy port zewnętrznego sygnału zegarowego *CLK12MHZ*. Końcówka układu MachXO2 związana z tym portem, połączona jest na płycie ewaluacyjnej zestawu MachXO2 Pico z jedną z końcówek rezonatora kwarcowego 12 MHz, który z kolei dołączony jest do układu FT2232H, obsługującego łącze USB. Stąd na wspomnianym porcie *CLK12MHZ* modułu głównego występuje przebieg zegarowy 12 MHz.

W dalszej części kodu modułu głównego następuje opis oscylatora, związanego z pojemnościowymi polami dotykowymi – analogicznie jak na list. 7. Cześć obsługi pojemnościowych pól dotykowych, odpowiedzialną za detekcję zmiany częstotliwości oscylatorów, przeniesiono do modułu o nazwie *CAPSENSE*, którego kod pokazano na **listingu 11**. Kod modułu detektora zmiany częstotliwości oscylatorów bazuje na opisie z list. 7, przy czym zamiast dwóch niezależnych procesów, tutaj zastosowano pojedynczy proces, który realizuje dokładnie te same czynności.

Zasadniczą część kodu z list. 10 stanowi opis automatu sekwencyjnego, który realizuje całą funkcjonalność projektu. Warto zwrócić uwagę, że zmienna *tcnt* odpowiada za liczbę prób, po której wyliczana jest średnia czasu reakcji użytkownika. Wartość tej zmiennej sprawdzana jest w stanie automatu o numerze 6. Ze względu na uproszczony sposób wyliczania średniej, liczba prób powinna być potęgą liczby 2. Wówczas do obliczenia średniej czasu reakcji, zmienną *srtime* – zawierającą sumę czasów reakcji podczas wszystkich prób, wystarczy przesunąć w prawo o odpowiednią liczbę bitów (dla 8 prób tak jak na list. 10, powinno nastąpić przesuniecie zmiennej *srtime* w prawo o 3 bity).

W ostatniej sekcji kodu z list. 10 występują jeszcze instancje modułu odmierzania interwału czasowego (list. 8) oraz modułu konwertera BIN –> BCD, którego kod pokazaliśmy na list. 6. W tym ostatnim przypadku odpowiednim parametrom modułu (tab. 7) nadaliśmy jawnie nowe wartości, tak aby wyjściowy kod BCD reprezentowany był za pomocą 4 cyfr.

Działanie całego projektu, zaimplementowanego na płycie zestawu MachXO2 Pico jest następujące. Po włączeniu zasilania lub naciśnięciu przycisku zerowania S1, na wyświetlaczu pojawi się ciąg znaków „----”. Dotknięcie dowolnego pola dotykowego spowoduje wygaszenie wyświetlacza i jednocześnie losowo odmierzany jest odcinek czasu (czas oczekiwania może wynosić nawet kilka sekund), po którym na wyświetlaczu pojawia się losowo wybrana cyfra od 1 do 4. Naciśniecie pola dotykowego odpowiadającego wyświetlonej cyfrze (dla cyfry 1 jest to pole 1 itd.), spowoduje wygaszenie wyświetlacza i jednocześnie rozpoczyna się odmierzanie nowego interwału czasowego o losowej wartości, po czym znowu wyświetlana jest losowa cyfra itd. Po ośmiu próbach zamiast kolejnej wylosowanej cyfry wyświetlacz pokazuje średni czas reakcji w sekundach, z dokładnością do dwóch cyfr po przecinku (10 ms). Dotkniecie w tym momencie dowolnego pola dotykowego uruchamia od początku całą procedurę losowania i pomiaru czasu.

Posługiwanie się pojemnościowymi polami dotykowymi, zwłaszcza w sytuacji gdy chcemy skutecznie dotknąć odpowiednie pole w jaknajkrótszym czasie, wymaga pewnej wprawy. Można tutaj również dojść do wniosku, że czułość przycisków pojemnościowych jest nieco zbyt mała. Czułość możemy jednak poprawić zwiększając np. częstotliwość zegarową procesu odpowiadającego za oscylator (list. 7, 10) oraz dobierając pozostałe parametry takie jak liczba bitów licznika *SmplCntr* odpowiedzialnego za czas zliczania oscylacji itp.

Zbigniew Hajduk
zhajduk@kia.prz.edu.pl

Listing 10. c.d.

```
  6: begin if(tcnt<8) ST<=1; else
     begin srtime<=srtime>>3; b2d_start<=1; ST<=7; end
    end
  7: begin b2d_start<=0; ST<=8; end
  8: begin if(b2d_done) begin sw_lcd<=1; ST<=9; end end
  9: if(~(touched1|touched2|touched3|touched4)) ST<=10;
  10: if(touched1|touched2|touched3|touched4)
         begin ST<=1; sw_lcd<=0; srtime<=0; tcnt<=0; end

 endcase

// instancja modułu odmierzania interwału czasowego (list. 8)
DELAY del(.CLK12MHZ(CLK12MHZ),.RST(RST),.START(go),
          .DEL(tc),.DONE(tdone));

// instancja modułu konwertera BIN -> BCD (list. 6)
bin2bcd #(.NO_BITS_IN(14),.NO_BCD_DIGITS(4),.BIT_CNT_WIDTH(4)) b2d
          (.clk(CLK12MHZ),.start(b2d_start),.done(b2d_done),
           .data_in(srtime[13:0]),.data_bcd({dgt1,dgt2,dgt3,dgt4}));

endmodule
```

Listing 11. Moduł detekcji zmiany częstotliwości oscylatorów

```
module CAPSENSE (input CLK, RECAL,
        input OscReg1, OscReg2, OscReg3, OscReg4,
        output reg touched1, touched2, touched3, touched4);

parameter Sensitivity = 10;

reg [6:0] FCnt1, FCnt2, FCnt3, FCnt4;
reg [6:0] FCnt1Smpl, FCnt2Smpl, FCnt3Smpl, FCnt4Smpl;
reg [7:0] SmplCntr;
wire SmplCntrCO = &SmplCntr;

always @(posedge CLK)
if(~RECAL)
begin
 touched1<=0; touched2<=0; touched3<=0; touched4<=0;
 FCnt1Smpl<=1; FCnt2Smpl<=1; FCnt3Smpl<=1; FCnt4Smpl<=1;
 FCnt1<=0; FCnt2<=0; FCnt3<=0; FCnt4<=0;  SmplCntr<=0;
end else
begin
 SmplCntr<=SmplCntr+1;
 if(SmplCntrCO)
 begin
   FCnt1<=0; FCnt2<=0; FCnt3<=0; FCnt4<=0;

   if (FCnt1Smpl<FCnt1-Sensitivity)
    begin touched1<=0; FCnt1Smpl<=FCnt1; end else
   if (FCnt1Smpl>FCnt1+Sensitivity)
    begin touched1<=1; FCnt1Smpl<=FCnt1;
    end

   if (FCnt2Smpl<FCnt2-Sensitivity)
    begin touched2<=0; FCnt2Smpl<=FCnt2; end else
    if (FCnt2Smpl>FCnt2+Sensitivity)
    begin touched2<=1; FCnt2Smpl<=FCnt2; end

   if (FCnt3Smpl<FCnt3-Sensitivity)
    begin touched3<=0; FCnt3Smpl<=FCnt3; end else
    if (FCnt3Smpl>FCnt3+Sensitivity)
    begin touched3<=1; FCnt3Smpl<=FCnt3; end

   if (FCnt4Smpl<FCnt4-Sensitivity)
    begin touched4<=0; FCnt4Smpl<=FCnt4; end else
    if (FCnt4Smpl>FCnt4+Sensitivity)
    begin touched4<=1; FCnt4Smpl<=FCnt4; end
 end else
 begin
  if(OscReg1) FCnt1<=FCnt1+1;
  if(OscReg2) FCnt2<=FCnt2+1;
  if(OscReg3) FCnt3<=FCnt3+1;
  if(OscReg4) FCnt4<=FCnt4+1;
 end
end

endmodule
```