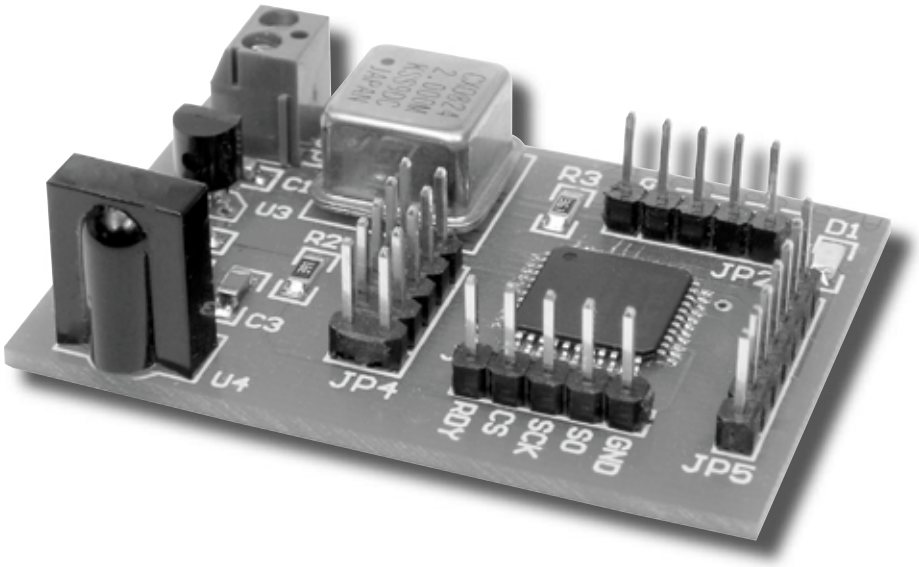
Dekoder RC5 z interfejsem SPI, opisany w języku Verilog, część 2

AVT-545

W artykule przedstawiono sprzętowy dekoder protokołu RC5 z interfejsem równoległym oraz szeregowym interfejsem kompatybilnym z SPI, opisany w języku Verilog i zrealizowany z wykorzystaniem układów CPLD firmy Xilinx.

Rekomendacje:

atykuł polecamy wszystkim zainteresowanym projektami realizowanymi na układach programowalnych. Oprócz tego, że ostatecznym rezultatem opisu jest kompletny projekt działającego dekodera RC5, to dodatkowo jest tu zaprezentowane projektowanie w języku Verilog.



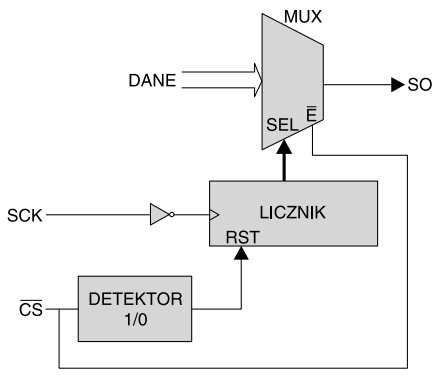
Szeregowy interfejs SPI

Budując sprzętowy dekoder RC5 z wykorzystaniem układów PLD, można wyposażyć go w szeregowy interfejs kompatybilny z SPI. Interfejs ten może być przydatny w przypadku współpracy dekodera z mikrokontrolerem (odciążając mikroprocesor z realizacji zadania dekodowania protokołu RC5) lub innymi urządzeniami peryferyjnymi.

Szeregowy interfejs SPI (*Serial Peripheral Interface*) do transmisji danych wykorzystuje trzy linie: CS – wybór danego urządzenia SPI (aktywny poziom niski), SCK – sygnał zegarowy, SO – wyjście danych. Za pośrednictwem interfejsu SPI dane przesyłane są w sposób synchroniczny. W pewnym uproszczeniu działanie interfejsu można opisać następująco: transmisję danych inicjuje układ nadrzędny ustawiając linię CS układu podrzędnego w stan niski. Układ podrzędny ustawia wówczas linię wyjściową S0 w stan odpowiadający pierwszemu transmitowanemu bitowi (bit najbardziej znaczący przesyłany jest jako pierwszy). Stan linii S0 może być próbkowany przez układ nadrzędny podczas opadającego zbocza sygnału zegarowego SCK. Układ podrzędny może zmieniać stan linii wyjściowej tylko wówczas, gdy sygnał SCK generowany przez układ nadrzędny znajduje się w stanie niskim. Wygenerowanie impulsu na linii SCK przez układ nadrzędny powoduje pojawienie się na linii S0 kolejnego transmitowanego bitu, itd.

Na **list. 3** przedstawiono kod w języku Verilog opisujący działanie interfejsu szeregowego SPI dla układu sprzętowego dekodera RC5.

Przedstawiona na list. 3 implementacja interfejsu szeregowego wykorzystuje rejestr przesuwny z wpisem równoległym. Jak wykazały doświadczenia, tego typu realizacja interfejsu zajmuje relatywnie dużo zasobów sprzętowych układu programowalnego (co jest istotne w przypadku implementacji na tanich ukła-



Rys. 7. Alternatywna realizacja interfejsu SPI

List. 3. Wirtualny komponent realizujący interfejs SPI dla układu podrzędnego

```
module spi_core(clk,data,cs,sck,s0);
input clk,cs,sck;
input [13:0] data;
output s0;

reg [15:0] sreg;
reg f1,f2;

always @(posedge clk)
begin
    if(~cs)
    begin
      //jeżeli linia CS znajduje się w stanie
niskim

      if(~f1) begin f1=1; f2=0; sreg=data;
end
      //równoległy wpis do rejestru przesuw-
nego sreg
      //wartości danych wejściowych - podczas
opadającego
      //zbocza sygnału CS

      if(~sck&&f2)
      begin
         sreg={sreg[12:0],sreg[13]};
        f2=0; //zeruj flagę f2
         //rejestr przesuwny sreg przesuwany
jest podczas gdy
         //SCK zmieni swój stan na niski
      end

      if (sck&&~f2) f2=1;
      //jeżeli SCK zmienił poziom na wysoki
- ustaw flagę f2

    end
    else f1=0;
    // jeśli CS=1, zeruj flagę f1

    end
  end

assign s0=sreg[13];
//SO to najstarszy bit rejestru przesuwnego
endmodule
```

dach programowalnych klasy CPLD, wyposażonych w niewielkie zasoby logiczne). Można jednak zastosować nieco prostszą konstrukcję interfejsu, tak jak zilustrowano to na **rys. 7**. Zamiast rejestru przesuwnego można wykorzystać licznik sprzężony z multiplekserem. Wystąpienie zbocza opadającego na linii CS spowoduje wyzerowanie licznika. Wyjścia licznika połączone są z wejściami wyboru multipleksera. Pojawienie się impulsów na wejściu zegarowym licznika spowoduje zwiększanie wartości licznika i odpowiadające tej wartości połączenie wyjścia multipleksera z jednym z wejść danych multipleksera. Efekt końcowy będzie identyczny jak z użyciem rejestru przesuwnego. Na **list. 4** przedstawiono kod w języku Verilog opisujący taką realizację interfejsu SPI.

Na **rys. 8** przedstawiono ilustrację sposobu działania interfejsu SPI zrealizowanego według opisu z list. 3. Liczba bitów interfejsu jest dostosowana do długości ramki protokołu RC5 i wynosi 14 bitów. Dlatego też w przypadku, gdy na linii SCK interfejsu wystąpi więcej niż 14 impulsów, wówczas dane pojawiają się na wyjściu „od początku" (czyli po najmniej znaczącym bicie kodu polecenia RC5 pojawia się pierwszy bit startu, itd.).

Działanie interfejsu SPI z list. **4** jest analogiczne, przy czym po 14 bitach ramki danych RC5, w przypadku dalszego taktowania linii SCK, pojawiając się dwa bity, których wartość zawsze jest równa 0 i dopiero wówczas kolejne bity ramki RC5 transmitowana jest od początku.

Dane z interfejsu SPI mogą być odczytywane w dowolnej chwili z szybkością wyznaczoną przez układ nadrzędny (mikrokontroler). Jednak w przypadku interfejsu z list. 3 maksymalna częstotliwość impulsów na wejściu SCK musi być mniejsza niż częstotliwość taktowania na wejściu zegarowym (*clk*) interfejsu. W przypadku implementacji interfejsu według list. 4, częstotliwość impulsów na linii SCK może być znacznie większa niż częstotliwość taktowania na wejściu *clk* (licznik, będący głównym elementem interfejsu, taktowany jest bezpośrednio impulsami z linii SCK, podczas gdy pomocniczy sygnał taktujący *clk* wykorzystywany jest tylko do wykrywania opadającego zbocza sygnału na linii CS).

Na **list. 5** przedstawiono dodatkowo kod źródłowy funkcji, napisanej w języku C dla mikrokontrolera, dzięki której można odczytać dane z przedstawionego powyżej interfejsu SPI. Funkcja zwraca pełne 14 bitów ramki danych RC5.

Przykładowa aplikacja

Wykorzystując opisane powyżej wirtualne komponenty dekoderów RC5 oraz interfejsu SPI można zbudować rzeczywisty, sprzętowy dekoder protokołu RC5. Na **rys. 9** przedstawiono przykładowy schemat elektryczny takiego układu, a na **rys. 10** widok płytki drukowanej.

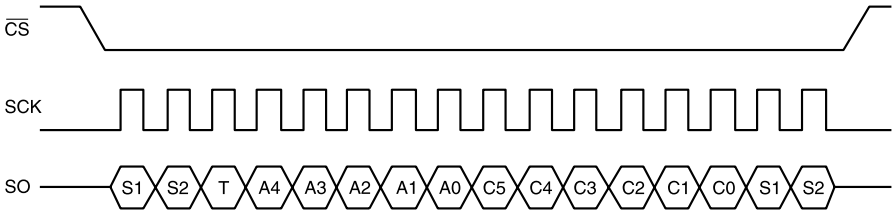
Jako układ programowalny wybrano tutaj Xilinx CPLD z rodziny XC9500XL typu XC9572XL w obudowie VQFP44. Układ zasilany jest napięciem 3,3 V, ale od strony wejść akceptuje napięcia o standardzie TTL. Jako stabilizator 3,3 V zastosowano typowy stabilizator uniwersalny typu LM317 z odpowiednio dobranymi wartościami rezystorów określającymi wartość napięcia wyjściowego. W układzie wytwarzania przebiegu taktującego pracuje typowy, czterokońcówkowy (w metalowej obudowie) zintegrowany generator zegara o częstotliwości 2 MHz (można zastosować generator o innej częstotliwości, ale wówczas należy odpowiednio zaprogramować dzielni-

List. 4. Alternatywna implementacja interfejsu SPI

```
module spi_core(data,clk,cs,sck,s0);
input cs,clk,sck;
input [13:0] data;
output s0;

wire [3:0] qcntr;

  cntr4 c1(.sck(sck),.clk(clk),.q(qcntr),.
cs(cs));
  mux m1(.data(data),.sel(qcntr),.out(s0));
  //definicja sposobu połączenia licznika i
multipleksera
  //(konkretyzacja modułów licznika i multi-
pleksera)

endmodule

//definicja modułu licznika
module cntr4(cs,clk,sck,q);
input sck,clk,cs;
output [3:0] q;
reg [3:0] q;

reg clrcnt,f;
wire rst;

always @(posedge clk)
begin
   if(~cs&&f)
   begin
     f=0; clrcnt=1;
   end
   else clrcnt=0;
   //sygnał zerowanie clrcnt dla licznika
ustawiany
   //w momencie wystąpienia zbocza opadają-
cego na CS

   if(cs&&~f) f=1;

end

assign rst=~clrcnt; //zmiana polaryzacji

always @(negedge sck or negedge rst)
begin
  if(~rst) q=4'd0;
  //asynchroniczne zerowanie licznika

  else
   q=q+1;
   //inkrementacja licznika

end

endmodule

//opis behawioralny modułu multipleksera
module mux(data,sel,out);
input [13:0] data;
input [3:0] sel;
output out;
reg out;

always @(sel or data)
begin
  case (sel)
   4'd0: out=data[13];
   4'd1: out=data[12];
   4'd2: out=data[11];
   4'd3: out=data[10];
   4'd4: out=data[9];
   4'd5: out=data[8];
   4'd6: out=data[7];
   4'd7: out=data[6];
   4'd8: out=data[5];
   4'd9: out=data[4];
   4'd10: out=data[3];
   4'd11: out=data[2];
   4'd12: out=data[1];
   4'd13: out=data[0];
   4'd14: out=0;
   4'd15: out=0;
  endcase

end

endmodule
```

List. 5. Funkcja odczytu danych z interfejsu SPI, napisana w języku C dla mikrokontrolera

```
unsigned int spi_rx(void)
{
  char j, b;
  unsigned int w;
  w=0;

  CS=0;
  SCK=0;
  for (j=0;j<14;j++)
  {   SCK=1;
      w<<=1; b=SO; w|=b;
      SCK=0;
  }
  CS=1;
  return w;
}
```



Rys. 8. Ilustracja działania interfejsu SPI

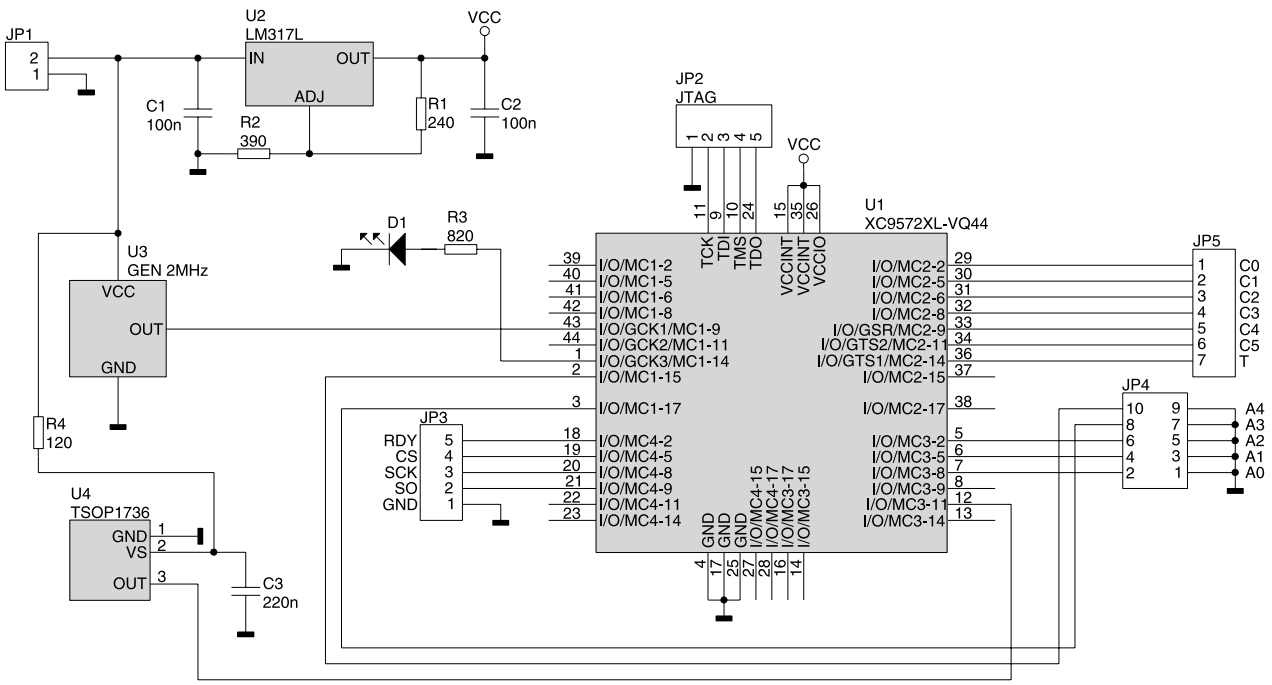
ki częstotliwości dla układu dekodera RC5, który musi być taktowany z częstotliwością 1 MHz). Do przetwarzania sygnałów podczerwieni, nadawanych przez pilota zdalnego sterowania, wykorzystano zintegrowany odbiornik podczerwieni firmy Vishay (TSOP1736). Na **list. 6** przedstawiono kod źródłowy w języku Verilog modułu implementującego dekoder RC5 i interfejs SPI.

Po zaimportowaniu do środowiska EDA dla układów programowalnych kodów źródłowych z list. 5 – modułu nadrzędnego w hierarchicznej strukturze projektu – oraz jednego z modułów dekodera RC5 (których kody źródłowe zamieszczono na list. 1 i 2) i modułu interfejsu SPI (list. 3 lub 4), i jeszcze przypisując portom wejścia/wyjścia modułu nadrzędnego odpowiednie numery końcówek układu PLD – kompletny projekt można poddać kompilacji. Jako środowisko EDA wykorzystano tutaj pakiet Xilinx ISE w wersji 5.2i. W wyniku procesu kompilacji (implementacji) otrzymuje się plik w standardowym, dla układów CPLD, formacie JEDEC, który służy do zaprogramowania nieulotnej pamięci konfiguracji w danym układzie PLD. Zastosowany układ PLD posiada możliwość programowania w systemie – ISP (*In-System Programming*), poprzez intefejs JTAG, którego odpowiednie linie wyprowadzono na złącze JP2 płytki drukowanej. W celu zaprogramowania układu można wykorzystać uniwersalny programator ISP opisany w Elektronice Praktycznej 1/2004 lub też prosty programator, którego schemat można znaleźć w dokumentacji firmy Xilinx (programator ten był opisany również w Elektronice Praktycznej 4/2001).

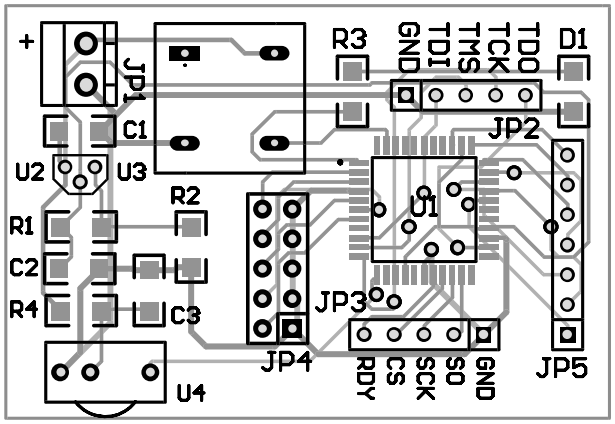
Po zaprogramowaniu układu PLD sprzętowy dekoder RC5 jest gotowy do pracy. Poprawne odebranie przez dekoder ramki danych protokołu RC5 sygnalizowane jest krótkim mignięciem diody LED. Jednocześnie na wyjściu RDY (złącze JP3) pojawia się w tym momencie stan wysoki. Na złączu JP5 występują kolejne bity (C0...C5) kodu polece-

List. 6. Kompletny sprzętowy dekoder RC5 z interfejsem SPI

```
module top(clock,in,cs,sck,s0,rdy,ld,C,A
,T);
//definicja portów we - wy
input clock,in,cs,sck;
output ld,s0,rdy,T;
output [4:0] A; // adres RC5
output [5:0] C; // kod polecenia

wire clk;
reg rc5ready;
wire [13:0] rc5code;

  clock_divider cd1(.clock(clock),.
clk(clk));
  //konkretyzacja modułu dzielnika często-
tliwości
  //sygnał clk ma teraz częstotliwość 1MHz

  rc5decoder dec(.clk(clk),.in(in),.rdy(r-
dy),.rc5code(rc5code));
  //konkretyzacja opisanego wcześniej modu-
łu dekodera

  assign ld=rdy; //dioda LED połączona z
sygnałem rdy

  assign C={rc5code[5],rc5code[4],rc5code-
[3],rc5code[2],
            rc5code[1],rc5code[0]};
  //przypisanie do odpowiednich wyjść kodu
polecenia RC5

  assign A={rc5code[10],rc5code[9],rc5code-
[8],rc5code[7],rc5code[6]};
  //przypisanie adresu urządzenia RC5

  assign T=rc5code[11];
  //bit informujący o przytrzymaniu klawi-
sza w pilocie

  spi_core spi1(.clk(clk),.cs(cs),.
sck(sck),.s0(s0),.data(rc5code));
  //konkretyzacja modułu interfejsu SPI

endmodule

//moduł dzielnika częstotliwości
module clock_divider(clock,clk);
input clock;
output clk;

reg q;

always @(posedge clock)
begin
  q=~q;
end

 assign clk=q;

endmodule
```



Rys. 9. Schemat ideowy układu sprzętowego dekodera RC5



Rys. 10. Schemat montażowy płytki drukowanej dekodera

nia RC5 oraz bit T (informujący o przytrzymaniu klawisza w pilocie), a na złączu JP4 dostępne są bity adresu urządzenia (A0...A4). Zdekodowane bity ramki danych RC5 można również odebrać poprzez wbudowany interfejs SPI (linie CS, SCK, SO złącza JP3).

Sprzętowy dekoder RC5 można również tak skonfigurować, by odbierał tylko ramkę danych z określonym adresem urządzenia RC5 (podobnie jak robił to wycofany już z produkcji układ SAA3049 firmy Philips). W tym celu wystarczy nieco zmienić kod w module nadrzędnym – tak jak to pokazano na **list. 7**.

W tym przypadku adres urządzenia RC5 odebrany z ramki danych porównywany jest z adresem odczytanym z zewnętrznych końcówek układu dekodera (stan linii A0...A4 złącza JP3). Jeżeli oba adresy są zgodne, wówczas ustawiane jest wyjście RDY (złącze JP3) i na krótką chwilę zapala się dioda LED. Na wyjściach C0...C5 oraz T dostępne są wówczas odpowiednie dane. Dane odczytane za pomocą interfejsu SPI dostępne są niezależnie od tego, czy oba adresy urządzeń RC5 są zgodne, czy też nie.

Na kolejnych rysunkach przedstawiono przebiegi czasowe w wybranych punktach sprzętowego dekodera RC5 otrzymane za pomocą analizatora stanów logicznych. Na **rys. 11** pokazano przebiegi czasowe dla układu dekodera opisanego kodem z list. 1. Pierwszy kanał analizatora ilustruje stan sygnału wejściowego dekodera (wyjście zintegrowanego odbiornika podczerwieni) w momencie nadawania ramki danych RC5 z kodem polecenia równym 01h oraz adresem urządzenia 16h. Drugi kanał przedstawia zachowanie sygnału *onair* (list. 1 – sygnał wyprowadzono na zewnętrzne końcówki układu PLD). Sygnał ten odpowiada sygnałowi z wyjścia monowibratora z rys. 3. Jego opadające zbocze wyznacza moment próbkowania sygnału wejściowego przez rejestr przesuwny (czyli stan dekodowanego bitu). Trzeci kanał analizatora ilustruje przebieg sygnału na wejściu *rdy*. Poziom wysoki na tym wyjściu oznacza pomyślne zakończenie dekodowania ramki danych RC5.

Na **rys. 12** przedstawiono przebiegi czasowe w układzie sprzętowego dekodera RC5 wykorzystującego ideę automatu sekwencyjnego (kod z list. 2). Kanały pierwszy i ostatni analizatora stanów logicznych ilustrują zachowanie sygnałów wejściowego *in* oraz wyjściowego *rdy*. Na pozostałych kanałach widoczne są przebiegi sygnałów określające rodzaj zdarzenia dla automatu sekwencyjnego wykorzystywanego do dekodowania protokołu RC5 (sygnały: *lg* – długa przerwa, *sg* – krótka przerwa, *lp* – długi impuls, *sp* – krótki impuls).

Zbigniew Hajduk

WYKAZ ELEMENTÓW

Rezystory (SMD 1206)
R1: 240Ω
R2: 390Ω
R3: 820Ω
R4: 120Ω

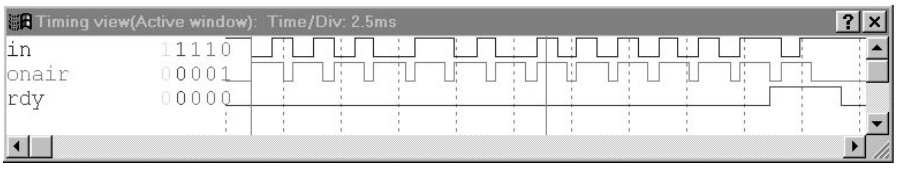
Kondensatory (SMD 1206)
C1, C2: 100nF
C3: 220nF

Półprzewodniki
U1: XC9572XL-VQ44
U2: LM317 (TO-92)
U3: generator 2MHz
U4: TSOP1736
D1: LED (SMD1206)

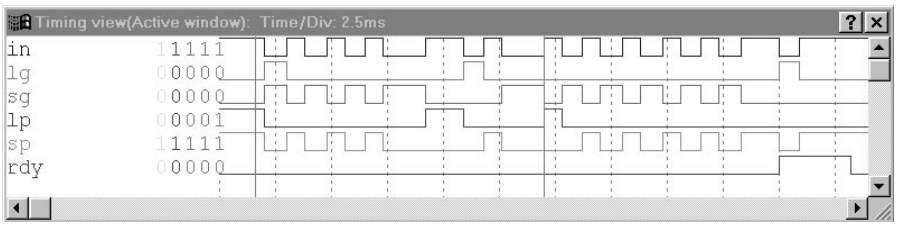
Inne
JP1: ARK-2/3,5mm
JP2, JP3, JP4: gold pin

List. 7. Kod sprzętowego dekodera RC5 akceptującego określony adres urządzenia RC5

```
module top(clock,in,cs,sck,s0,rdy,ld,C,A,T);
input clock,in,cs,sck;
output ld,s0,rdy,T;

input [4:0] A;
output [5:0] C;
wire [4:0] rc5addr;
reg [5:0] C;
wire clk;
reg T,ready;
wire rdy;
wire [13:0] rc5code;

  clock_divider cd1(.clock(clock),.clk(clk));
  rc5decoder dec(.clk(clk),.in(in),.rdy(r-
dy),.rc5code(rc5code));
  spi_core spi1(.clk(clk),.cs(cs),.sck(sck),.
s0(s0),.data(rc5code));

  assign rc5addr={rc5code[10],rc5code[9],
                  rc5code[8],rc5code[7],rc5c
ode[6]};
  //wydzielenie adresu urządzenia

  always @(posedge clk)
  begin
    if(rdy)
    begin
     if(rc5addr==A)
     begin
         //jeżeli zgadza się adres RC5 z adre-
sem ustawionym
         //na liniach zewnętrznych układu PLD

         C={rc5code[5],rc5code[4],rc5code[3],
rc5code[2],
             rc5code[1],rc5code[0]};
       T=rc5code[11];
       ready=1;
     end
    end
    else ready=0;

  end

  assign ld=ready;
endmodule
```



Rys. 11. Przebiegi czasowe w układzie dekodera RC5 zaimplementowanego według kodu z list. 1



Rys. 12. Przebiegi czasowe w układzie dekodera RC5 zaimplementowanego według kodu z list. 2