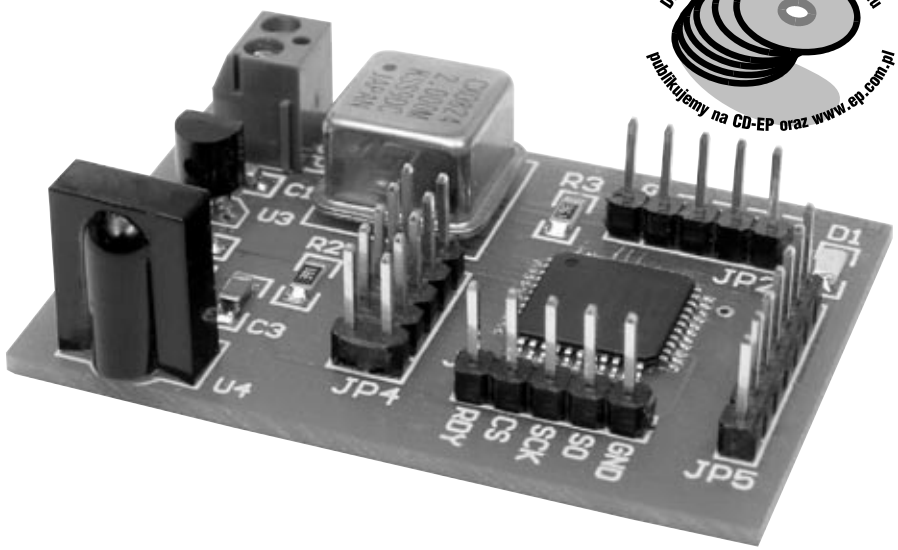
Dekoder RC5 z interfejsem SPI, opisany w języku Verilog, część 1

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:

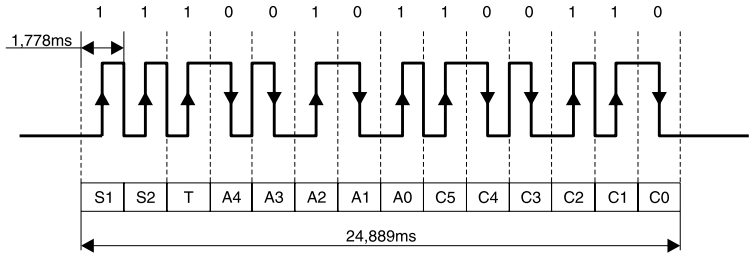
atykuł polecamy wszystkim zainteresowanym projektami realizowanymi na układach programowalnych. Oprócz tego, że ostatecznym rezultatem opisu jest kompletny projekt działającego dekodera RC5, to dodatkowo jest tu zaprezentowane projektowanie w języku Verilog.

Współcześnie, w większości przypadków, dekodowanie sygnałów zdalnego sterowania realizowane jest w sposób programowy z wykorzystaniem mikrokontrolerów. Alternatywnie, do dekodowania poszczególnych protokołów zdalnego sterowania można zastosować rozwiązania całkowicie sprzętowe. Dzięki użyciu układów programowalnych i komputerowych narzędzi syntezy logicznej jest to zadanie stosunkowo nieskomplikowane (w porównaniu z klasycznymi metodami projektowania i realizacji układów cyfrowych z wykorzystaniem „katalogowych" układów scalonych) i tanie w realizacji.

Obecnie są wytwarzane układy programowalne o coraz większej złożoności i zasobach logicznych przy jednoczesnym spadku ceny tych układów. Układy programowalne, szczególnie klasy FPGA (*Field Programmable Gate Array* – układy programowalne o strukturze komórkowej), swoimi możliwościami dorównują niejednokrotnie specjalizowanym układom scalonym ASIC (*Application Specific Integrated Circuit*), oferując przy tym możliwość prototypowania konkretnych realizacji i prowadzenia eksperymentów przy niemal minimalnych kosztach, oraz znacznie skracając czas wprowadzenia danego produktu na rynek.

Wraz z postępem w technologii wytwarzania układów programowalnych następuje również intensywny rozwój komputerowych narzędzi wspomagania projektowania CAD (*Computer Aided Design*) znanych jako systemy automatyzacji projektowania EDA (*Electronic Design Automation*). Narzędzia EDA umożliwiają realizację zadań syntezy (translacji specyfikacji projektu do jego implementacji) oraz symulację, w której specyfikacja lub szczegóły implementacji mogą być poddane analizie w celu sprawdzenia poprawności działania.

Specyfikacja projektu (opis projektowanego układu) do celów syntezy, przy użyciu narzędzi EDA, może być zrealizowana na kilka sposobów: za pomocą schematu, z wykorzystaniem przebiegów czasowych lub diagramu stanów oraz z zastosowaniem opisu tekstowego w jednym z języków opisu sprzętu – HDL (*Hardware Description Language*). Do języków HDL będących obecnie w powszechnym użyciu zalicza się VHDL i Verilog. Czasami stosowany jest też język CUPL (np. firma Altium integruje ten język z systemem Protel), AHDL (język wykorzystywany przez firmę Altera), oraz ABEL (np. kompilator tego języka jest standardowym wyposażeniem udostępnianego bezpłatnie pa-



Rys. 1. Ramka danych protokołu RC5

kietu WebPack ISE firmy Xilinx).

Wraz z wprowadzeniem języków opisu sprzętu oraz układów programowalnych pojawiły się nowe możliwości aplikacyjne. Jeden z kierunków takich aplikacji określa koncepcja produktu wirtualnego – systemu, funkcji lub układu scalonego, które nie istnieją w rzeczywistości materialnej, ale mogą być potencjalnie w każdej chwili zrealizowane. Takie układy wirtualne mogą być zarówno gotowymi produktami o zmiennych funkcjach i właściwościach, jak również elastycznymi „klockami", z których można konstruować większe systemy. Idea rdzeni projektowych szybko zyskała uznanie producentów i użytkowników. Funkcjonuje obecnie duży rynek wirtualnych komponentów specjalizowanych, tzw. rynek własności intelektualnej IP (*Intellectual Property*) zawartej w układach zaprojektowanych w postaci kodów źródłowych języków HDL. Rynkowa oferta rdzeni IP obejmuje oprócz standardowych układów cyfrowych takich jak sumatory, multipleksery, itp., również zaawansowane układy przetwarzania sygnałów cyfrowych takie, jak: filtry, kodery, dekodery, układy kryptograficzne, komutatory i rutery do sieci komputerowych, itp.

Dekodowanie sygnałów RC5

W protokole RC5 do kodowania przesyłanych bitów stosowany jest kod Manchester. Czas trwania bitu jest stały i wynosi 1,778 ms. Bit o wartości logicznej 1 kodowany jest jako zmiana poziomu sygnału z niskiego na wysoki (zbocze narastające) w połowie czasu trwania bitu. Bit o wartości logicznej 0 kodowany jest jako zmiana poziomu sygnału z wysokiego na niski (zbocze opadające) również w połowie czasu trwania bitu. Inaczej można też powiedzieć, że jedynka logiczna kodowana jest jako kolejno: przerwa o czasie trwania równym połowie czasu trwania bitu, czyli 889 µs i impulsu o identycznym czasie trwania. Podobnie zero logiczne kodowane jest jako kolejno: impuls o długości 889 µs (połowa czasu trwania bitu) oraz przerwa o takim samym czasie trwania.

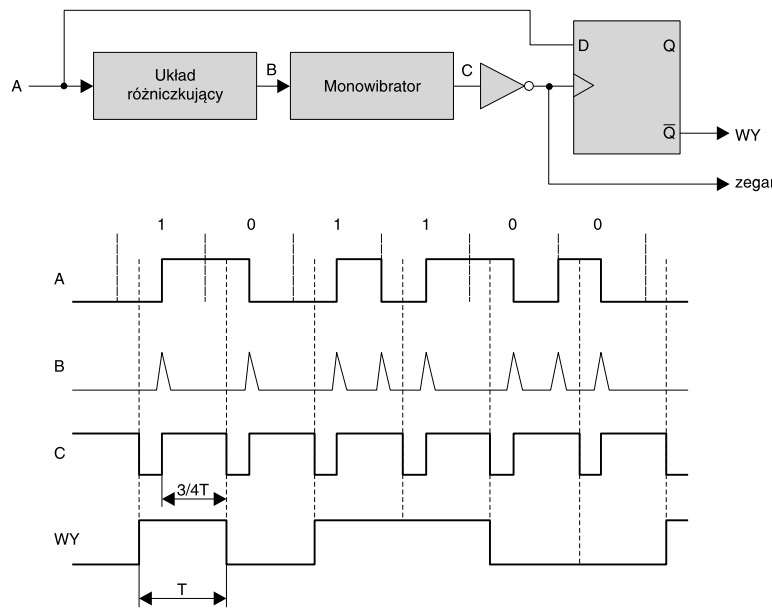
Na **rys. 1** przedstawiono przykładową ramkę danych protokołu RC5. Pierwsze dwa przesyłane bity (S1 i S2) to bity startowe, których wartość logiczna zawsze równa jest 1. Kolejnym, trzecim przesyłanym bitem jest bit T informujący o przytrzymaniu klawisza w pilocie zdalnego sterowania. Bit ten zmienia swoją wartość na przeciwną za każdym razem, kiedy naciskany jest klawisz w pilocie (wysyłane jest polecenie). W przypadku, kiedy klawisz w pilocie przytrzymany zostanie dłużej, kolejno wysyłane są identyczne ramki danych w odpowiednich odstępach czasowych – bit T nie jest wówczas cyklicznie negowany i ma cały czas stałą wartość. Kolejne przesyłane bity A4...A0 to bity adresu urządzenia RC5. Ostatnie 6 bitów ramki danych C0...C5 stanowią bity kodu polecenia.

Ponieważ protokół RC5 wykorzystuje kodowanie Manchester, więc do jego dekodowania można zastosować metody identyczne jak dla tego kodu. Dekodowanie kodu Manchester realizowane jest bądź z użyciem układów czasowych, bądź automatu synchronicznego.

Na **rys. 2** przestawiono ideę dekodowania protokołu RC5 z wykorzystaniem układu czasowego. Układ różniczkujący wytwarza impulsy przy każdej zmianie sygnału wejściowego. Impulsy te wyzwalają monowibrator (generator monostabilny), który generuje impulsy o szerokości równej 0,75 czasu trwania bitu w protokole RC5 (ok. 1,3 ms). Jeżeli następny impuls wyzwalający (następne zbocze) pojawi się wówczas, gdy monowibrator nie zakończył jeszcze generowania impulsu, to powtórne wyzwolenie monowibratora nie nastąpi. Zanegowane impulsy monowibratora stanowią sygnał zegarowy taktujący przerzutnik D, na którego wyjściu !Q występują wartości zdekodowanych bitów.

Na **rys. 3** przedstawiono graf automatu (inaczej graf przejść i wyjść dla modelu automatu Mealy'ego), wykorzystywanego do dekodowania protokołu RC5. Etykiety opisujące łuki oznaczają odpowiednio rodzaj zdarzenia, oraz po przecinku wartość logiczną zdekodowanego bitu. Zdarzenia opisane są następującymi symbolami:

sp – krótki impuls,



Rys. 2. Ilustracja idei dekodowania protokołu RC5 z wykorzystaniem układu czasowego

Tab. 1. Opis oznaczeń stosowanych na rys. 3

Zdarzenie	Ideał	Min.	Max.
sp	889 µs	444 µs	1333 µs
sg	889 µs	444 µs	1333 µs
lp	1778 µs	1334 µs	2222 µs
lg	1778 µs	1334 µs	2222 µs

sg – krótka przerwa,
lp – długi impuls,
lg – długa przerwa.

Zdarzenia te sklasyfikowane są w sposób podany w **tab. 1.**

Automat rozpoczyna swoje działanie od stanu S0, przy czym wejście do tego stanu powodowane jest pojawieniem się zbocza narastającego dekodowanego sygnału i jednocześnie oznacza, że odebrany został pierwszy bit o wartości logicznej 1. Następnie dekodowanych jest pozostałych 13 bitów (odebranie kolejno 14 bitów stanowi warunek zatrzymania automatu). Każde inne występujące zdarzenie, nie opisane grafem z rys. 3, traktowane jest jako błąd, powodujący przerwanie jego działania.

Sprzętowy dekoder RC5 wykorzystujący układ czasowy

Na **rys. 4** przedstawiono szczegółowy schemat implementacji dekodera protokołu RC5 wykorzystującego układ czasowy.

Dekoder składa się z:

- detektora zboczy wykrywającego zmiany poziomu sygnału z 1 na 0 oraz z 0 na 1,
- 3–bitowego licznika zliczającego liczbę pojawiających się zboczy sygnału wejściowego w momencie, gdy odmierzany jest czas przez licznik 11–bitowy,
- 11–bitowego licznika z wpisem równoległym liczącego wstecz, służącego do odmierzania czasu 3/4 T (T – czas trwania bitu w protokole RC5),
- detektora (komparatora) wykrywającego stan osiągnięcia przez licznik wartości 0,
- 14–bitowego rejestru przesuwnego zapamiętującego w odpowiednich chwilach czasu (wyznaczonych przez układ sterowania i licznik 11–bitowy) stan wejścia dekodera (wyjścia odbiornika podczerwieni),
- układu sterowania.

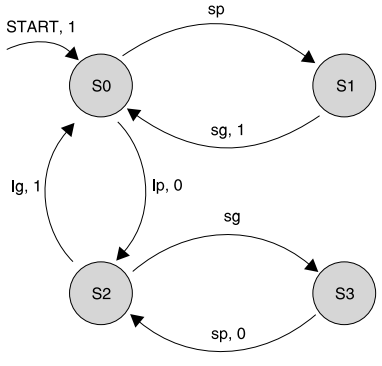
Opcjonalnie dekoder może być wyposażony w interfejs kompatybilny z SPI, służący do szeregowej wymiany danych np. z mikrokontrolerem.

Układ sterowania realizuje następujące funkcje:

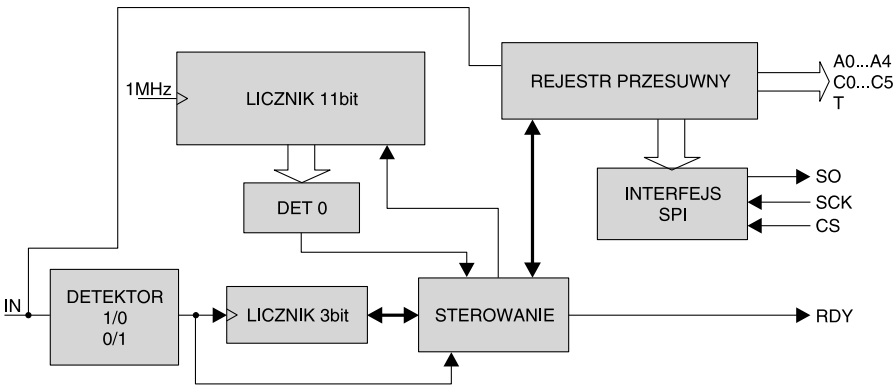
- Na podstawie informacji z detektora zboczy steruje pracą licznika 11–bitowego: w momencie wystąpienia zbocza, do tego licznika ładowana jest wartość stałej czasowej odpowiadająca 3/4 T. Jeżeli pojawi się kolejne zbocze sygnału wejściowego w momencie, gdy licznik nie zakończy jeszcze odmierzania czasu 3/4 T od momentu wystąpienia poprzedniego zbocza (licznik nie osiągnie jeszcze stanu 0), wówczas powtórne załadowanie stałej czasowej do licznika nie nastąpi.
- Sprawdza liczbę występujących zboczy sygnału wejściowego (za pośrednictwem licznika 3–bitowego), w czasie gdy licznik 11–bitowy odmierza czas 3/4 T. Wystąpienie w tym czasie więcej niż jednego zbocza sygnału wejściowego układ sterowania interpretuje jako błąd (dekodowany protokół nie jest protokołem RC5) i wznawia proces dekodowania od początku (zerowana jest zawartość rejestru przesuwnego).
- Na podstawie informacji z detektora osiągnięcia przez licznik 11–bitowy wartości 0, kieruje pracą rejestru przesuwnego wyznaczając moment próbkowania stanu sygnału wejściowego.
- Jeżeli od momentu ostatniego zakończenia odmierzania czasu 3/4 T upłynął czas dłuższy niż 3/4 T układ sterowania interpretuje taką sytuację jako błąd (normalnie w protokole RC5 następne zbocze powinno pojawić się po czasie 1/4 T) i proces dekodowania rozpoczynany jest od początku. W celu wyznaczenia momentu przekroczenia czasu oczekiwania na zbocze układ sterujący odpowiednio steruje pracą licznika 11–bitowego: licznik automatycznie ładowany jest wartością stałej czasowej 3/4 T, w momencie gdy osiągnie stan 0 (czyli, gdy ukończy odmierzanie poprzedniej wartości 3/4 T).
- Określa moment zakończenia dekodowania ramki protokołu RC5 ustawiając na pewien czas (3/4 T) wyjście RDY. Aby określić moment zakończenia dekodowania (odebranie wszystkich 14-tu bitów) układ sterowania analizuje stan najstarszego bitu (najbardziej znaczącego) wyjścia w rejestrze przesuwnym. Jeżeli bit ten ma wartość 1, wówczas oznacza to zakończenie procesu dekodowania.

W celu implementacji sprzętowej tak zdefiniowanego dekodera RC5, z zastosowaniem narzędzi EDA i języków HDL najlepiej posłużyć się opisem mieszanym: strukturalno – behawioralnym. Strukturę dekodera określa sposób połączenia poszczególnych bloków funkcjonalnych, podczas gdy działanie tych bloków można opisać behawioralnie.

Podstawową jednostką projektową w języku Verilog jest tzw. moduł. Może więc on zawierać opis behawioralny działania poszczególnych bloków z rys. 4. Jednak w przedstawionym na **list. 1** kodzie specyfikującym działanie dekodera



Rys. 3. Graf automatu dekodowania protokołu RC5



Rys. 4. Schemat blokowy dekodera RC5 z wykorzystaniem układu czasowego

List. 1. Dekoder kodu RC5 opisany w języku Verilog, wykorzystujący układ czasowy

```
module rc5decoder(clk,in,rdy,rc5code);
                        //deklaracja portów we/wy
input clk,in;
output rdy;
output [13:0] rc5code;
                        //deklaracja zmiennych
reg [13:0] rc5code;
reg rc5ready,in1,load,start,to,onair;
reg [13:0] rc5temp;
wire [10:0] qcntr;
reg [1:0] ecnt;
 cntr c1(.clk(clk),.load(load),.d(11'd1333),.q(qcntr));
                        //konkretyzacja (utworzenie instancji) modułu
licznika
                        //(licznik zdefiniowany jest w oddzielnym module)
                        //poniżej opis behawioralny działania dekodera
   always @(posedge clk)
   begin
         in1<=in;       //przypisanie nie blokujące
                        //zapamiętanie stanu wejścia w poprzednim takcie
zegara
        if((in1&~in)||(~in1&in))
      begin
                        //jeżeli zbocze 0/1 lub 1/0 sygnału wejściowego
                        //(realizacja detektora zboczy)
       if(~start)
       begin
                        //jeżeli nie rozpoczęto jeszcze dekodowania -
                        //ustaw odpowiednie wartości początkowe
       start=1'b1; onair=0;
       rc5temp=14'd1; ecnt=0;
       end

       else ecnt=ecnt+1;
                        //inkrementuj wartość licznika błędu
         if(~onair)
       begin
                        //jeżeli wystąpiło zbocze na wejściu
                        // i nie trwa odmierzania czasu 3/4T
         load=1'b1;     //załaduj licznik stałą 3/4T
         onair=1'b1;
       end

       to=0;
      end

      else
      load=0;
        if(ecnt==2'b11)
      begin
                        //jeżeli zbyt dużo zboczy - rozpocznij dekodowa-
nie od początku
       start=0; to=0;
      end

      if(qcntr==11'd0)
      begin
                        //licznik osiągnął stan 0

      onair=0; ecnt=0;
      if(~to)
      begin
        to=1'b1;
                        //jeżeli to=1, licznik odmierza czas oczekiwania
na następne zbocze
        if(start)
                rc5temp={rc5temp[12:0],in};
                        //realizacja rejestru przesuwnego
      end

      else
      begin
                        //jeżeli przekroczono czas oczekiwania na zbocze
        start=0; to=0; rc5ready=0;
      end

      end

      if(rc5temp[13]==1'b1)
      begin
                        //jeżeli odebrano już 14 bitów - koniec dekodowa-
nia

       rc5code=rc5temp; rc5ready=1; start=0;
      end

   end

   assign rdy=rc5ready;

endmodule
                        //definicja działania licznika 11-bitowego
module cntr(clk,load,d,q);
input clk,load;
input [10:0] d;
output [10:0] q;
reg [10:0] tmp;
always @(posedge clk)
begin
 if(load) tmp=d;
 else
 if(tmp==11'd0) tmp=d;
 else tmp=tmp-1;
                        //jeżeli licznik osiągnął 0 - ładuj automatycznie
stałą
                        //czasową d, w przeciwnym przypadku dekrementuj
zawartość licznika
end

assign q=tmp;

endmodule
```

List. 2. Dekoder RC5 opisany w języku Verilog, wykorzystujący automat sekwencyjny

```
module rc5decoder_II(clk,in,rdy,rc5code);
input clk,in;
output rdy;
output [13:0] rc5code;
wire short_pulse,long_pulse,short_gap,long_gap;
reg[3:0] state;
wire time_out;
reg [13:0] rc5temp;
reg rc5ready;
reg [13:0] rc5code;
 timer t1(.clk(clk),.in(~in),
        .lp(long_pulse),.sp(short_pulse),
     .lg(long_gap),.sg(short_gap),.to(time_out));
                        //Konkretyzacja modułu timera (licznik wraz z kom-
paratorami)
                        //Poniżej realizacja automatu
  always@(posedge clk)
  begin
   if(time_out)
   begin
                        //jeżeli przekroczono czas oczekiwania na zbocze -
                        //automat wraca do stanu początkowego
      state=4'd0; rc5ready=0;
   end

   else
   begin
     case(state)
     4'd0:
      begin
       rc5temp=0;
       if(~in) state=4'd1;
        else state=4'd0;
                        //jeżeli zmiana poziomu wejścia na niski -
                        //przejdź do stanu 1, w przeciwnym przypadku pozo-
stań w 0
      end

   4'd1:
      begin
       rc5temp={rc5temp[12:0],1'b1};
                        //wpisanie jedynki do rejestru przesuwnego
                        //(pierwszy bit startu ramki RC5)
       state=4'd2;      //przejdź do stanu 2 czyli S0
      end

    4'd2:
      begin
                        //stan S0 automatu (rys. 3.16)
       if(short_pulse&&long_pulse) state=4'd0;
                        //jeżeli inne zdarzenie niż krótki impuls lub dłu-
gi impuls
                        //wróć do stanu 0
       else
       if(short_pulse&&!long_pulse) state=4'd3; //S1
                        //jeżeli krótki impuls przejdź do stanu 3 (S1)
       else
       if(long_pulse&&!short_pulse) state=4'd4;
                        // jeżeli długi impuls przejdź do stanu pośrednie-
go 4
       else state=4'd2;
                        //w przeciwnym przypadku pozostań w bieżącym sta-
nie 2
      end

    4'd3:
      begin
                        //stan S1
      if(short_gap&&long_gap) state=4'd0;
                        //jeżeli zdarzenie inne niż spodziewane - następny
stan 0
       else
       if(short_gap&&!long_gap) state=4'd5;
                        //jeżeli krótka przerwa wróć do stanu S0
       else state=4'd3;
      end

    4'd4:
      begin
                        //odebrano bit o wartości 0
       rc5temp={rc5temp[12:0],1'b0};
                        //wpisanie 0 do rejstru przesuwnego
       state=4'd6;      //przejdź do stanu S2
      end

    4'd5:
      begin
                        // odebrano bit o wartości 1
      rc5temp={rc5temp[12:0],1'b1};
                        //wpisanie 1 do rejestru przesuwnego
         state=4'd2;//przejdź do stanu S0
      end

    4'd6:
      begin
                        //stan S2
      if(short_gap&&long_gap) state=4'd0;
                        //jeżeli zdarzenie inne niż spodziewane - następny
stan 0
       else
       if(long_gap&&!short_gap) state=4'd5;
                        //jeżeli długa przerwa przejdź do stanu pośrednie-
go 5
       else
       if(short_gap&&!long_gap) state=4'd7;
                        //jeżeli krótka przerwa przejdź do stanu S3
       else state=4'd6;
      end
    4'd7:
      begin
                        //stan S3
       if(short_pulse&&long_pulse) state=4'd0;
```

List. 2. cd.

```
                        //jeżeli zdarzenie inne niż spodziewane - następny stan 0
       else
       if(short_pulse&&!long_pulse) state=4'd4;
                        //jeżeli krótki impuls idź do stanu pośredniego 4
       else state=4'd7;
      end

    endcase
     if(rc5temp[13]==1'b1)
     begin
                        //jeżeli bit MSB rejstru przesuwnego zapalony -
                        //dekodowanie zakończone
       rc5code=rc5temp;
                        //zapamiętaj odebrane bity w rejestrze zatrzaskowym
       rc5ready=1; state=4'd0;
                        //ustaw flage zakończenia dekodowania i przejdź do stanu 0
     end
   end
  end

  assign rdy=rc5ready;
                        //przepisanie flagi zakończenia dekodowania do portu wyjściowego
endmodule

                        //Poniżej kod modułu timera
module timer(clk,sp,lp,sg,lg,in,to);
input clk,in;
output sp,lp,sg,lg,to;
reg sp1,lp1,sg1,lg1,rst,in1,to;
wire [11:0] time1;
wire [4:0] tm;
 cntr c1(.clk(clk),.q(time1),.rst(rst));
                        //konkretyzacja modułu licznika
 assign tm={time1[11],time1[10],time1[9],time1[8]};
                        //zmienna tm zawiera 4 najbardziej znaczące bity licznika
 assign sp=sp1;
 assign lp=lp1;
 assign sg=sg1;
 assign lg=lg1;
                        //poniżej opis behawioralny działania układu licznik - komparatory
 always @(posedge clk)
 begin
     if(time1>12'd2300)
      to=1;
     else to=0;
                        //jeżeli oczekiwanie na zbocze dłużej niż 2.3ms - ustaw flagę to

   in1<=in;             //przypisanie nie blokujące
                        //(zapamiętanie stanu wejścia w poprzednim takcie zegara)

   if(in1&~in)          //zbocze opadające na wejściu
   begin
     rst=1;             //wyzeruj licznik (w następnym takcie)
  if(tm>4'd2&&tm<4'd5)//krótki impuls
  begin
    sp1=1; lp1=0;
    sg1=0; lg1=0;
  end

  else
  if(tm>4'd4&&tm<4'd8)//długi impuls
  begin
    sp1=0; lp1=1;
    sg1=0; lg1=0;
  end

  else
  begin
                        //jeżeli impuls spoza przedziału - błąd
     sp1=1; lp1=1;
     sg1=0; lg1=0;
  end
    end

    else
    if(~in1&in)         //zbocze narastające na wejściu
    begin
      rst=1;            //zeruj licznik
  if(tm>4'd2&&tm<4'd5)//krótka przerwa
  begin
    sg1=1; lg1=0;
    sp1=0; lp1=0;
  end
  else
  if(tm>4'd4&&tm<4'd8)//długa przerwa
  begin
    sg1=0; lg1=1;
    sp1=0; lp1=0;
  end
  else
  begin
      //jeżeli przerwa spoza przedziału - błąd
     sg1=1; lg1=1;
     sp1=0; lp1=0;
  end
    end
    else
  rst=0;
 end
endmodule

                        //Poniżej kod licznika
module cntr(clk,rst,q);
input clk,rst;
output [11:0] q;
reg [11:0] tmp;
 always @(posedge clk)
 begin
    if (rst) tmp=0;
                        //synchroniczne zerowanie
     else tmp=tmp+1;
                        //inkrementacja licznika
 end
   assign q=tmp;
endmodule
```

RC5 wygodniej było zintegrować bloki: detektora zboczy, licznika 3–bitowego, detektora osiągnięcia przez licznik 11–bitowy stanu zerowego, rejestru przesuwnego oraz układu sterowania – w module głównym dekodera. Osobny moduł zawiera jedynie definicję działania bloku licznika 11-bitowego.

W przedstawionym kodzie, celem zmniejszenia niezbędnych do implementacji zasobów sprzętowych układu programowalnego, zredukowano do 2 liczbę bitów licznika zliczającego liczbę pojawiających się zboczy sygnału wejściowego w czasie gdy odmierzany jest czas 3/4 T (co jednak wiąże się ze zmniejszeniem skuteczności wykrywania opisanych sytuacji, ale w praktyce okazało się zupełnie wystarczające).

Tak zdefiniowany wirtualny komponent dekodera RC5 może być postrzegany jako pewna „czarna skrzynka" z następującymi sygnałami wejściowymi i wyjściowymi:

- *clk* – zegarowy sygnał taktujący (musi być 1 MHz),
- *in* – wejście dekodera, które powinno być połączone z wyjściem odbiornika podczerwieni (aktywny poziom niski – nie trzeba więc negować sygnału z odbiornika),
- *rdy* – wyjście informujące o pomyślnym zakończeniu dekodowania protokołu (na tym wyjściu pojawia się impuls o czasie trwania ok. 3 ms)
- *rc5code* – 14-bitowe wyjście danych zdekodowanych (stan poszczególnych bitów w odebranej ramce danych protokołu RC5)

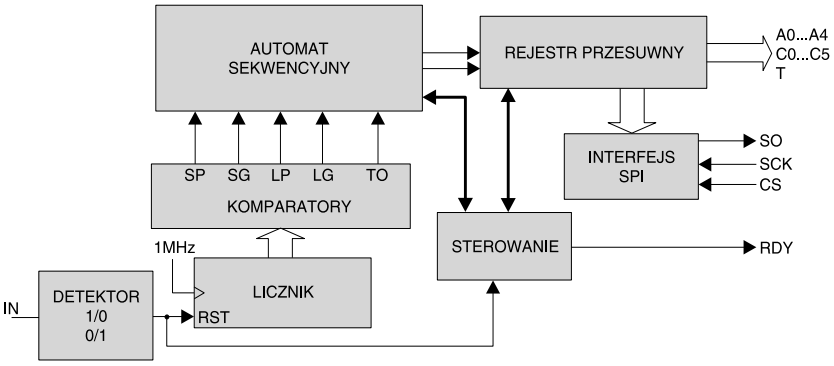
Sprzętowy dekoder RC5 wykorzystujący automat sekwencyjny

W przypadku implementacji sprzętowej dekodera z wykorzystaniem automatu sekwencyjnego odpowiedni schemat blokowy układu dekodera może wyglądać tak, jak pokazano na **rys. 5**. Ta wersja dekodera składa się z:

- detektora zboczy (1/0 lub 0/1) zerującego licznik (wystąpienie zbocza sygnału wejściowego powoduje wyzerowanie licznika),
- licznika 11-bitowego odmierzającego czas z rozdzielczością 1 μs (przy taktowaniu 1 MHz),
- zespołu komparatorów, który określa wystąpienie odpowiednich zdarzeń dla automatu sekwencyjnego (krótki impuls – SP, długi impuls – LP, krótka przerwa – SG, długa przerwa – LG, przekroczenie czasu

oczekiwania na zbocze – TO),

- automatu sekwencyjnego, działającego według grafu z rys. 3, będącego zasadniczą częścią dekodera,
- rejestru przesuwnego, zapamiętującego kolejne odebrane bity ramki protokołu RC5, wyznaczone przez automat sekwencyjny,
- prostego układu sterująco–nadzorującego, odpowiedzialnego m. in. za wyznaczania końca transmisji ramki danych RC5.



Rys. 5. Schemat blokowy dekodera RC5 z wykorzystaniem automatu sekwencyjnego

Kod źródłowy implementacji dekodera z wykorzystaniem automatu sekwencyjnego przedstawiono na **list. 2**.

Tak zaimplementowany wirtualny komponent dekodera RC5, z punktu widzenia sygnałów wejścia – wyjścia, zachowuje się identycznie jak dekoder opisany poprzednio. Wymaga jednak nieco więcej zasobów sprzętowych danego układu programowalnego (dla przykładu: w przypadku implementacji z zastosowaniem układu Xilinx XC9572XL i narzędzi XST zawartych w pakiecie Xilinx ISE 5.2i, dekoder ten zajmuje 63 makrokomórki, podczas gdy poprzednia wersja – 57, w obu przypadkach dla strategii silnej optymalizacji powierzchni).

Zbigniew Hajduk