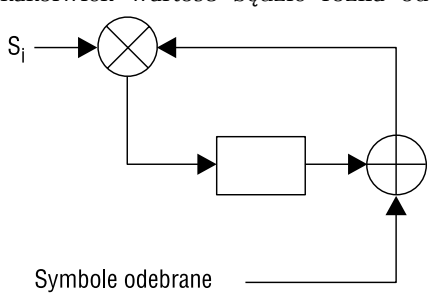
Kodowanie Reed–Solomona i VHDL, część 2

W EP 4/2005 opisano sposób zabezpieczenia w transmisji danych za pomocą dodatkowych słów kodowych CRC. Takie dodatkowe słowa dodane w nadajniku pozwalały odbiornikowi zweryfikować poprawność odebranych danych. Niestety, dodatkowe informacje zawarte w CRC nie pozwalają na korekcję błędów w przypadku ich wystąpienia. Tą niedogodność eliminuje, stosowane obecnie prawie zawsze w urządzeniach profesjonalnych w transmisji cyfrowej kodowanie Reed–Solomon'a (w skrócie RS). Istnieją również inne sposoby zabezpieczania danych w transmisji cyfrowej, np. kodowanie Trellisa. Kodowanie Reed–Solomona zostało opisane w 1960 roku w piśmie „Journal of the Society for Industrial and Applied Mathematics" przez Irvinga Reeda i Gusa Solomona.

Sprawdzanie poprawności – obliczanie syndromu

Syndrom **S** jest zbiorem składającym się z n–k symboli. Jeśli po stronie odbiornika wyliczymy elementy zbioru syndromu S i jeśli jakakolwiek wartość będzie różna od zera to oznacza, że transmisja danych została odebrana z błędami.



Rys. 4. Schemat układu obliczającego syndrom S

Jeśli przez r(x) oznaczymy wielomian stopnia n, którego współczynnikami będą odebrane symbole, to poszczególne elementy zbioru S wylicza się według następującego wzoru:

$S_i = r(\alpha^i)$, gdzie i=1, ..., n–k

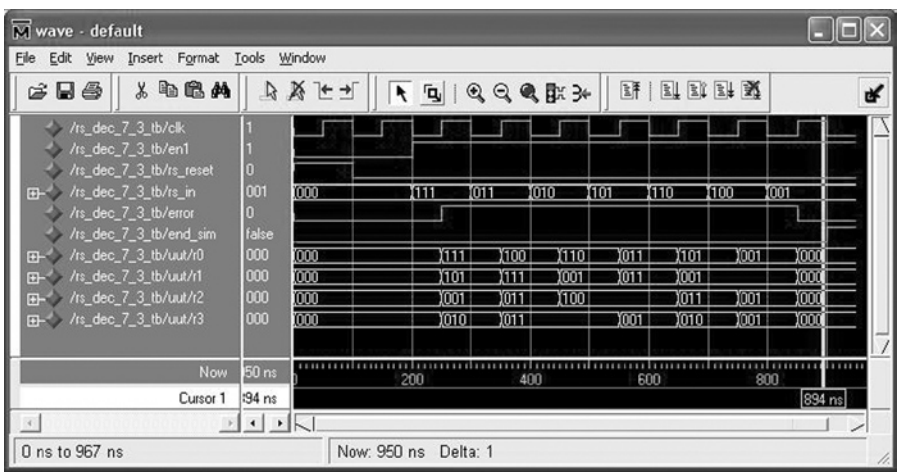
Na **rys. 4** przedstawiono sposób realizacji obliczenia syndromu z użyciem techniki cyfrowej. Implementację obliczenia syndromu S w języku VHDL dla naszego przykładu $GF(2^3)=1+x+x^3$ przedstawiono na **list. 3**.

Dekoder sprawdzający w odbiorniku poprawność danych działa w następujący sposób:

1. Podajemy na wejście RS_in wszystkie 7 odebranych symboli.
2. Jeśli po 7 taktach zegarowych na wyjściu Error pojawi się wartość 0 oznacza to, że transmisja została odebrana bezbłędnie.

W porównaniu z implementacją enkodera układ dekodera nie potrzebuje sygnału RS_switch. Dodatkowo na wyjściu jest tylko jedna linia o nazwie Error, która określa, czy w odebranej przez enkoder transmisji wystąpił błąd czy nie (odpowiednio wartość '1' określa wystąpienie błędu a '0' jego brak).

Linia *Error* jest wyjściem bramki OR, której wejściami są wyjścia rejestrów zliczających syndrom:



Rys. 5. Wyniki symulacji układu dekodera dla symboli kodu RS $GF(2^3) = 1 + x + x^3$ w programie Modelsim XE III Starter 6.0a

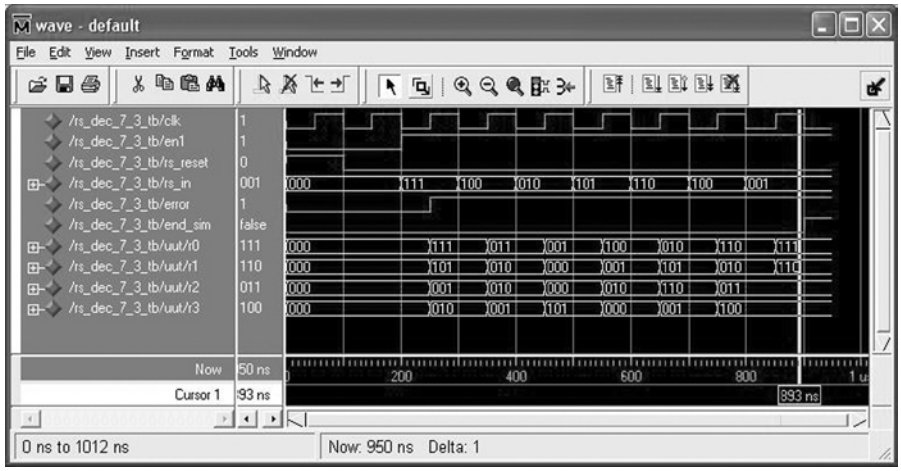
List. 3. Implementacja dekodera syndromu kodu RS (7, 3), dla GF(2^3) = 1 + x +x^3 w języku VHDL

```
--------------------------------------------------------------------------------
---- RS DECODER RS(7,3)
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity RS_dec_7_3 is
   port(
     CLK : in STD_LOGIC;
     EN1 : in STD_LOGIC;
     RS_Reset : in STD_LOGIC;
     RS_in : in STD_LOGIC_VECTOR (2 downto 0);
     Error : out STD_LOGIC);
end RS_dec_7_3;

architecture RS_dec_7_3_arch of RS_dec_7_3 is
  component rs_mnoz
    port(
      A : in std_logic_vector(2 downto 0);
      B : in std_logic_vector(2 downto 0);
      C : out std_logic_vector(2 downto 0));
  end component;
-- g(x) = a3 + a1x + a0x2 + a3x3 + x4
  constant S0: std_logic_vector(2 downto 0) := "001";
  constant S1: std_logic_vector(2 downto 0) := "010";
  constant S2: std_logic_vector(2 downto 0) := "100";
  constant S3: std_logic_vector(2 downto 0) := "011";
-- wynik sumowania
  signal SUM0 : std_logic_vector(2 downto 0);
  signal SUM1 : std_logic_vector(2 downto 0);
  signal SUM2 : std_logic_vector(2 downto 0);
  signal SUM3 : std_logic_vector(2 downto 0);
-- wynik mnozenia
  signal M0 : std_logic_vector(2 downto 0);
  signal M1 : std_logic_vector(2 downto 0);
  signal M2 : std_logic_vector(2 downto 0);
  signal M3 : std_logic_vector(2 downto 0);
-- wartosc rejestrow
  signal R0 : std_logic_vector(2 downto 0);
  signal R1 : std_logic_vector(2 downto 0);
  signal R2 : std_logic_vector(2 downto 0);
  signal R3 : std_logic_vector(2 downto 0);
begin
  SUM0 <= R0 xor RS_in;
  SUM1 <= R1 xor RS_in;
  SUM2 <= R2 xor RS_in;
  SUM3 <= R3 xor RS_in;
  RS_multi_0: RS_mnoz port map(SUM0, S0, M0);
  RS_multi_1: RS_mnoz port map(SUM1, S1, M1);
  RS_multi_2: RS_mnoz port map(SUM2, S2, M2);
  RS_multi_3: RS_mnoz port map(SUM3, S3, M3);
  Error <= R0(0) or R0(1) or R0(2) or R1(0) or R1(1) or R1(2) or
           R2(0) or R2(1) or R2(2) or R3(0) or R3(1) or R3(2);
  process (CLK, RS_Reset)
  begin
    if (RS_Reset = ,1') then
      R0 <= „000";
      R1 <= „000";
      R2 <= „000";
      R3 <= „000";
    elsif (CLK'event and CLK = ,1') then
      if (EN1 = ,1') then
        R0 <= M0;
        R1 <= M1;
        R2 <= M2;
        R3 <= M3;
      end if;
    end if;
  end process;
end RS_dec_7_3_arch;
```

```
Error <= R0(0) or R0(1) or R0(2) or
R1(0) or R1(1) or R1(2) or
         R2(0) or R2(1) or R2(2) or
R3(0) or R3(1) or R3(2);
```

Implementacja układów mnożących i sumujących jest zrealizowana dokładnie tak samo jak w przypadku implementacji układu enkodera.



Rys. 6. Wyniki symulacji układu dekodera dla symboli kodu RS GF(2^3) = 1 + x +x^3 w programie Modelsim XE III Starter 6.0a

Na **list. 4** przedstawiono testbench, którego użyto do sprawdzenia poprawności wygenerowanego kody dekodera. W testbenchu wprowadzono na wejście układu dekodera RS_in dokładnie takie same dane jakie uzyskano w czasie symulacji enkodera. Na **rys. 5** i **6** przedstawiono wyniki symulacji odpowiednio przy transmisji bezbłędnej i z błędem. Jak widać dokładnie po wprowadzeniu siedmiu symboli do testowanego układu dekodera sygnał wyjściowy Error przyjął wartość '0', co oznacza brak błędów w transmisji.

Dla weryfikacji poprawności układu dekodera zmodyfikowano jedną linię w kodzie testbencha, tak, aby zasymulować wystąpienie błędu. W symulacji jako drugi symbol w kolejności zamiast wartości „011", wprowadzono „100".

Dekoder – korekcja błędów

Część dekodująca kodu RS jest najbardziej skomplikowana. Opiera się ona o specjalne operacje na macierzach równań. Dekodowanie można podzielić na kilka procesów:

- znalezienie miejsc wystąpienia błędów;
- znalezienie wartości tych błędów;
- poprawienie błędów.

Jeżeli chodzi o zajętość miejsca w układzie FPGA to o ile RS enkoder, w przypadku rozszerzonych pól Galois GF(2^8), czyli dla bajtowych słów kodowych, może zająć nie całe 200 slice'ów układu z rodziny Spartan 2 lub 3 to dekoder może zająć już ponad 2000 slice'ów.

Powstało wiele implementacji dekoderów RS w językach opisów sprzętu i są one dostępne w postaci IP core'ów, dostępnych za odpowiednią opłatą.

Zainteresowanych odsyłam również do gotowych rozwiązań w C/C++, które obecne są w Internecie, np *http://www.eccpage.com/* lub *http://itpp.sourceforge.net/latest/*. Osoby zajmujące się opisywaniem urządzeń w języku VHDL nie powinny mieć problemów z przejściem z implementacji w C/C++ na VHDL.

Podsumowanie

Kodowanie Reed–Solomona jest bardzo interesującą alternatywą dla CRC. Pozwala zabezpieczyć przesyłane dane i po dodaniu kilku dodatkowych słów kodowych można

List. 4. Implementacja testbencha dekodera syndromu kodu RS (7, 3), dla GF(2^3) = 1 + x + x^3 w języku VHDL

```
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity RS_dec_7_3_tb is
end RS_dec_7_3_tb;
architecture TB_ARCHITECTURE of RS_dec_7_3_tb is
  component RS_dec_7_3
    port(
      CLK : in STD_LOGIC;
      EN1 : in STD_LOGIC;
      RS_Reset : in STD_LOGIC;
      RS_in : in STD_LOGIC_VECTOR (2 downto 0);
      Error : out STD_LOGIC);
  end component;
  signal CLK : STD_LOGIC;
  signal EN1 : STD_LOGIC;
  signal RS_Reset : STD_LOGIC;
  signal RS_in : STD_LOGIC_VECTOR (2 downto 0);
  signal Error : STD_LOGIC;
  signal END_SIM: BOOLEAN:=FALSE;
begin
  UUT : RS_dec_7_3
      port map ( CLK => CLK, EN1 => EN1, RS_Reset => RS_Reset,
                 RS_in => RS_in, Error => Error );
  STIMULUS: process
  begin
    EN1 <= '0';
    RS_Reset <= '1';
    RS_in <= "000";
    wait for 100 ns;
    RS_Reset <= '0';
    wait for 100 ns;
    EN1 <= '1';
    RS_in <= "111";
    wait for 100 ns;
    RS_in <= "011";
    wait for 100 ns;
    RS_in <= "010";
    wait for 100 ns;
    RS_in <= "101";
    wait for 100 ns;
    RS_in <= "110";
    wait for 100 ns;
    RS_in <= "100";
    wait for 100 ns;
    RS_in <= "001";
    wait for 100 ns;
    END_SIM <= TRUE;
    wait;
  end process;
  CLOCK_CLK : process
  begin
    if END_SIM = FALSE then
      CLK <= '0';
      wait for 50 ns; --0 ps
    else
      wait;
    end if;
    if END_SIM = FALSE then
      CLK <= '1';
      wait for 50 ns; --50 ns
    else
      wait;
    end if;
end process;
end TB_ARCHITECTURE;
configuration TESTBENCH_FOR_RS_dec_7_3 of RS_dec_7_3_tb is
  for TB_ARCHITECTURE
    for UUT : RS_dec_7_3
      use entity work.RS_dec_7_3(RS_dec_7_3_arch);
    end for;
  end for;
end TESTBENCH_FOR_RS_dec_7_3;
```

nie tylko stwierdzić czy transmisja odbyła się poprawnie, ale i poprawić część uszkodzonych słów.

Warto skorzystać z kodowanie Reed–Solomona w projektach wykorzystujących układy FPGA, które współpracują z w miarę szybkim mikroprocesorem. Wówczas kodowanie (enkoder) wraz z wyliczaniem syndromu można zrealizować w układzie FPGA, zaś część poprawiającą błędy (pełny dekoder) w mikroprocesorze. Oczywiście takie rozwiązanie będzie efektywne, jeśli w transmisji nie przewidujemy występowania wielu błędów.

Korzystając z niniejszego artykułu i przedstawionego przykładu można przygotować enkoder i dekoder (tylko syndrom) dla dowolnego kodu RS, np takiego, w którym wymagane byłoby użycie słów 8–bitowych.

Marcin Nowakowski

Zalecana literatura

[1] L.H. Charles Lee: „Error–Control Block Codes for Communications Engineers", Artech House, inc, 2000, Boston – London

[2] Bernard Sklar "Introduction to Reed–Solomon Codes": article is provided courtesy of Prentice Hall, 12.04.2002