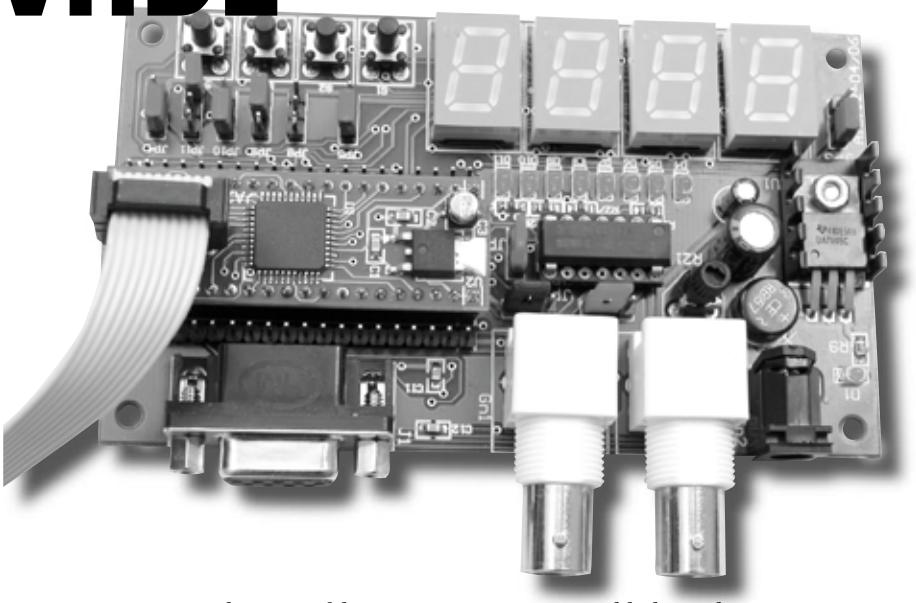
CRC w VHDL



Zabezpieczanie danych za pomocą sum kontrolnych CRC (Cyclic Redundancy Codes) nie jest zbyt popularne w projektach amatorskich, ale jest rzeczą obowiązkową na rynku profesjonalnym. CRC od strony „mikrokontrolerowej" przedstawiliśmy w serii artykułów począwszy od EP1/2003. Teraz przedstawimy implementację CRC w układach PLD.

Rekomendacje: *opisywany układ stanowi znakomitą ilustrację wykorzystania układów PLD oraz posługiwania się językiem opisu sprzętu VHDL, a dodatkowo pokazuje sposób implementacji jednego z ważnych algorytmów zabezpieczających informacje. Będzie zatem interesujący dla wszystkich czytelników zainteresowanych nowoczesną techniką cyfrową.*

Programowa realizacja obliczania CRC nie zawsze jest możliwa do zastosowania w praktycznej aplikacji, głównie ze względu na czas niezbędny do wykonania obliczeń. Znaczne przyspieszenie wykonywania obliczeń można uzyskać realizując je sprzętowo, a do tego celu idealnie nadają się układy PLD. Przykład pokazany w artykule (suma kontrolna wykorzystywana w sieciach ATM oraz interfejsie SMbus, określana mianem HEC lub sCRC8) jest tylko jednym z wielu możliwych algorytmów wyliczania sumy kontrolnej. Jej wielomian generujący ma postać: x^8+x^2+x+1, a zapis szesnastkowy: 0x07.

Czytelnikom zainteresowanym poznaniem tajników CRC gorąco polecam cykl artykułów opracowanych przez Jarosława Dolińskiego, którego pierwszą część opublikowaliśmy w EP1/2003. W tym artykule skupimy się przede wszystkim na omówieniu sprzętowej implementacji algorytmów obliczania CRC.

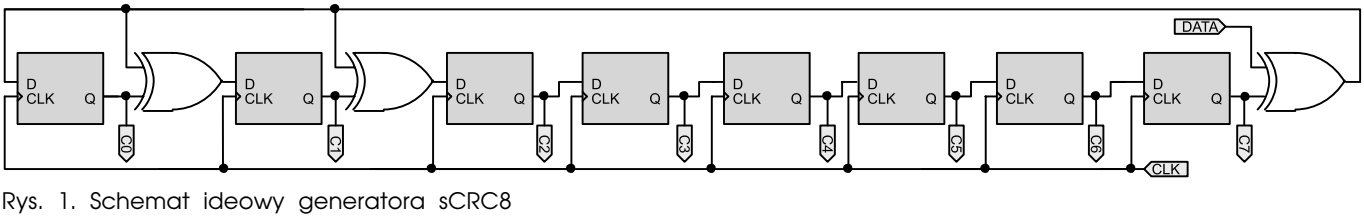
Jak przełożyć wielomian na sprzęt?

Najpoważniejszym zadaniem podczas realizacji projektu będzie przekonwertowanie wielomianu do postaci „pudełka" o zadanej przez użytkownika liczbie wejść i wyjściu, na którym pojawia się suma kontrolna po jej obliczeniu. Na przykładzie algorytmu sCRC8 pokażemy jak dokonać konwersji.

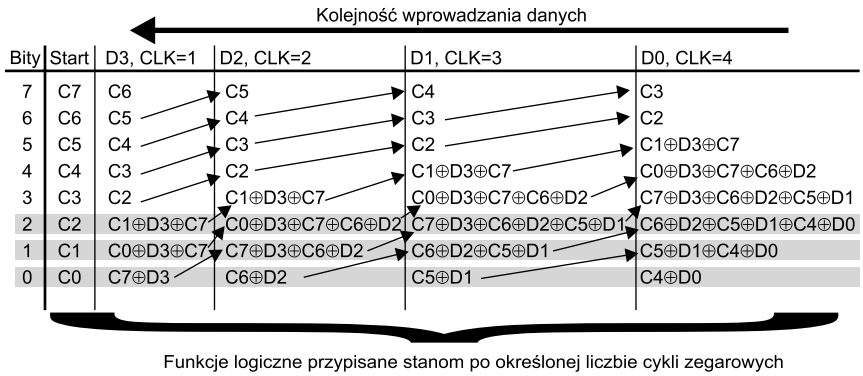
Podstawą będzie dla nas schemat blokowy szeregowego generatora sumy kontrolnej o konfiguracji odpowiadającej wielomianowi. Dla wielomianu x^8+x^2+x+1 schemat generatora pokazano na **rys. 1**.

Pierwszym, wydaje się także najprostszym w realizacji, sposobem opisu będzie wierne oddanie budowy układu pokazanego na schemacie, za pomocą równań logicznych w dowolnym języku HDL. Rozwiązanie rzeczywiście skuteczne, ale pod warunkiem, że generator CRC będzie zasilany danymi w postaci szeregowej (bardzo istotne: najstarszy bit danych jest podawany na wejście DATA jako pierwszy!). Jeśli chcemy jednak wykorzystać możliwości oferowane przez układy PLD lepszym wyjściem będzie badanie CRC dla słów o większej długości. Co wtedy?

Otóż znając wielomian generujący można odnaleźć równania logiczne przypisane każdemu bitowi CRC. Najprościej, choć jest to zadanie wymagające dokładności, można je utworzyć stosując metodę budowania równań *krok-po-kroku*. Polega ona na analizie stanów poszczególnych stopni generatora sumy kontrolnej po każdym cyklu zegarowym. Jeżeli liczba analizowanych



Rys. 1. Schemat ideowy generatora sCRC8




```
CRC(7) := C(3);
CRC(6) := C(2);
CRC(5) := D(3) xor C(1) xor C(7);
CRC(4) := D(3) xor D(2) xor C(0) xor C(6) xor C(7);
CRC(3) := D(3) xor D(2) xor D(1) xor C(5) xor C(6) xor C(7);
CRC(2) := D(2) xor D(1) xor D(0) xor C(4) xor C(5) xor C(6);
CRC(1) := D(1) xor D(0) xor C(4) xor C(5);
CRC(0) := D(0) xor C(4);
```

Rys. 2. Sposób konwersji generatora szeregowego na równoległy

cykli zegarowych będzie równa liczbie rejestrów (ogniw) wykorzystanych w generatorze, to nastąpiła konwersja generatora szeregowego na równoległy, w przypadku przyjęcia mniejszej liczby cykli zegarowych możemy utworzyć generator szeregowo-równoległy. Przykład takiej częściowej konwersji pokazano na **rys. 2**. Uzyskano generator dla wielomianu x^8+x^2+x+1 z równoległym wejściem czterobitowym. Równania logiczne odpowiadające poszczególnym bitom pokazano poniżej tablicy konwersji.

List. 1. Listing pakietu *PCKG* zawierającego funkcję *nextCRC*

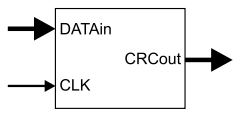
```
library IEEE;
use IEEE.std_logic_1164.all;

package PCKG is
function nextCRC
    ( data:  std_logic_vector(7 downto 0);
      prevCRC:   std_logic_vector(7 downto
0) )
    return std_logic_vector;

end PCKG;

package body PCKG is
  function nextCRC
    ( data:  std_logic_vector(7 downto 0);
      prevCRC:   std_logic_vector(7 downto
0) )
    return std_logic_vector is

    variable D: std_logic_vector(7 downto 0);
    variable C: std_logic_vector(7 downto 0);
    variable newCRC: std_logic_vector(7 do-
wnto 0);

  begin
    D := Data;
    C := prevCRC;
    newCRC(0) := D(7) xor D(6) xor D(0) xor
C(0) xor C(6) xor C(7);
    newCRC(1) := D(6) xor D(1) xor D(0) xor
C(0) xor C(1) xor C(6);
    newCRC(2) := D(6) xor D(2) xor D(1) xor
D(0) xor C(0) xor C(1) xor
                 C(2) xor C(6);
    newCRC(3) := D(7) xor D(3) xor D(2) xor
D(1) xor C(1) xor C(2) xor
                 C(3) xor C(7);
    newCRC(4) := D(4) xor D(3) xor D(2) xor
C(2) xor C(3) xor C(4);
    newCRC(5) := D(5) xor D(4) xor D(3) xor
C(3) xor C(4) xor C(5);
    newCRC(6) := D(6) xor D(5) xor D(4) xor
C(4) xor C(5) xor C(6);
    newCRC(7) := D(7) xor D(6) xor D(5) xor
C(5) xor C(6) xor C(7);
    return newCRC;
   end nextCRC;

end PCKG;
```

Nie ma jak funkcja

We wszystkich projektach dotychczas prezentowanych w EP wszelkie zadania były realizowane w plikach zawierających opisy kompletnych modułów funkcjonalnych. Teraz pokażemy inne, bardziej uniwersalne podejście - skorzystamy z funkcji.

W języku VHDL funkcja jest rodzajem podprogramu, który można wywołać z wieloma zmiennymi wejściowymi, ale zwracany jest tylko jeden wynik. Funkcja może być definiowana w ciele opisu HDL (do tego celu służy deklaracja *function*), ale w prezentowanym przykładzie posunęliśmy się krok dalej: funkcja została zadeklarowana w osobnym pakiecie, który można wykorzystać do przechowywania także innych własnych funkcji i procedur. Opis HDL pakietu zawierającego funkcję *nextCRC* obliczającą nową wartość CRC pokazano na **list. 1**. Pakiet nosi nazwę *PCKG*, a korzystanie z niego będzie możliwe po jawnym zadeklarowaniu go za pomocą dyrektywy *use work.pckg.all;* w głównym pliku projektu, jak to pokazano na **list. 2**. Zazwyczaj nie jest konieczne podawanie ścieżki dostępu do biblioteki użytkownika (znajdującej się domyślnie w katalogu *WORK*, czyli bieżącym projektu) za pomocą dyrektywy *library WORK;*.

Wywołanie funkcji z poziomu opisu jest nadzwyczaj proste, polega bowiem na przypisaniu wyniku działania funkcji wybranemu sygnałowi



Rys. 3. Symbol generatora CRC zbudowanego przez autora

(może to być magistrala wyjściowa lub sygnał „zagrzebany" zdefiniowany za pomocą dyrektywy *signal*):

jakis_sygnal <= nazwa_funkcji (parametr1, parametr3, parametr3,....);

Funkcja *nextCRC* oblicza CRC dla słów 8-bitowych. Ma ona dwa parametry wejściowe:

- *data* - 8-bitowe wejście kolejnego bajtu danych,
- *prevCRC* - 8-bitowe wejście danej o wartości dotychczas obliczonej CRC.

Należy pamiętać, że deklarowane typy sygnałów wejściowych funkcji są typu *std_logic_vector* i taki sam typ ma dana uzyskiwana na wyjściu funkcji.

W wyniku wykonania funkcji *nextCRC* otrzymujemy 8-bitowe słowo *newCRC*, które zawiera nową wartość CRC (w kolejnej rundzie obliczeń *newCRC* jest przypisywana zmiennej *prevCRC*).

Ponieważ układy „czysto" kombinacyjne słabo są przystosowane do pracy z dużymi częstotliwościami, dla zapewnienia bezpieczeństwa projektu zmienna *newCRC* jest zatrzaskiwana w 8-bitowym rejestrze D. Aktualizacja słowa wyjściowego odbywa się wraz z narastającym zboczem sygnału *CLK*.

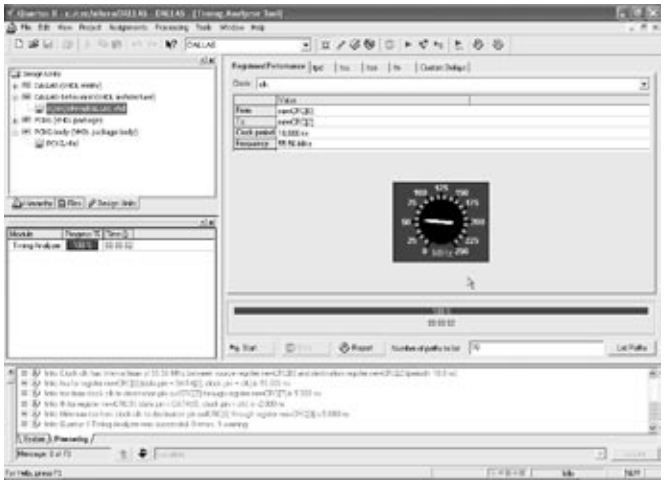
Możliwości *newCRC*

Na **rys. 3** pokazano symbol generatora CRC powstałego w wyniku kompilacji opisu z list. 2. W takiej konfiguracji projektant musi zadbać o sekwencyjne podawanie danych na wejście generatora i jego taktowanie. Należy pamiętać, że pełnię możliwości generatora CRC można uzyskać dla paczek 8 x 8-bitowych słów danych.

Kompletny projekt wymaga niewielkich zasobów logicznych, zajmuje bowiem 13/36 makrokomórek układu XC9536XL (i pochodnych), co przy obecnej cenie takiego układu (XC9536XL-10 kosztuje 6,5 zł brutto - *www.kamami.pl*) oznacza, że koszt generatora CRC o przepływności bliskiej 452 Mb/s wynosi 2,35 zł brutto. Nieco gorzej pod względem zajętości zasobów wypadają układy EPM7032S-10 firmy Altera, bowiem zajęte jest 20 z 32 makrokórek (co wynika z trudniejszej implementacji w tych układach funktorów ExOR), a maksymalna możliwa do uzyskania przepływność generatora wynosi 444,5 Mb/s (przy częstotliwości taktowania 55,56 MHz - **rys. 4**).

Działa?

Do weryfikacji poprawności działania konieczny okazał się dostępny na stronie *http://www.smbus.org/faq/crc8Applet.htm* kalkulator sumy kontrolnej. Czemu



Rys. 4. Widok okna analizatora czasowego programu Quartus II

Rys. 5. Widok okna symulatora ModelSIM z wynikami symulacji generatora sCRC8

akurat ten? Spośród wielu dostępnych na różnych stronach internetowych ten kalkulator działa poprawnie! Jego możliwości nie są co prawda zbyt duże (ograniczone do interesującego nas wielomianu), ale wyniki działania okazały się całkowicie wiarygodne w przeciwieństwie do szeregu innych kalkulatorów uniwersalnych.

Weryfikację funkcjonalną przeprowadzono na płytce ewaluacyjnej z układem XC9536XL. Projekt skompilowano za pomocą bezpłatnego pakietu WebPack ISE6.3.i, a symulację programową umożliwił (także bezpłatny) ModelSIM XE 5.8 (**rys. 5**).

Problematyczne okazało się zweryfikowanie funkcjonalne projektu za pomocą symulatora zaimplementowanego w środowisku Quartus II, ponieważ kompilator ignoruje przypisania inicjalizujące zmienne typu *signal newCRC : std_logic_vector(7 downto 0) := „00000000”;* dla układów MAX7KS. Drobny, aczkolwiek dokuczliwy, zabieg wprowadzenia sygnału zerującego i wstępnego zerowania rejestru *newCRC* zapewnił oczekiwany efekt.

Podsumowanie

Projekt prezentowany w artykule jest doskonałą ilustracją najważniejszej domeny układów PLD: sprzętowa realizacja algorytmów, które w wersji programowej zabierają wiele czasu. Duża prędkość liczenia CRC nie jest oczywiście na co dzień niezbędna, ale warto zdać sobie sprawę, że z odpowiednio szybkim obliczeniem CRC stosowanej w CANbus nie radzą sobie typowe mikrokontrolery, kłopotliwe jest także programowe obliczanie CRC przy „bezpiecznych” transmisjach via IrDA, a nawet RS232 (oczywiście dla większych prędkości).

Czytelnicy interesujący się językami opisu sprzętu mają przy okazji możliwość poznania sposobu tworzenia i korzystania z funkcji w języku VHDL, mogli się także zapoznać ze sposobem korzystania z zewnętrznych pakietów.

Piotr Zbysiński, EP
piotr.zbysinski@ep.com.pl

List. 2. Listing głównego pliku projektu generatora sumy kontrolnej sCRC8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.pckg.all;

entity DALLAS is port (
   DATA : in std_logic_vector(7 downto 0);
   clk : in std_logic;
   outCRC: out std_logic_vector(7 downto 0)
      );
end DALLAS;

architecture behavioral of DALLAS is

signal newCRC : std_logic_vector(7 downto 0) :=
„00000000”;

begin
process (clk, DATA, newCRC)
   begin
      if clk = ‚1’ and rising_edge(clk) then
         newCRC <= nextCRC (DATA, newCRC);
      else null;
   end if;
end process;

outCRC <= newCRC;

end behavioral;
```

Generator generatorów

Pod adresem:
http://www.easics.be/webtools/crctool
jest dostępny generator plików w językach VHDL i Verilog, umożliwiający utworzenie niemalże dowolnego generatora CRC.

Bezpłatne narzędzia

Projekt przedstawiony w artykule przygotowano i przesymulowano za pomocą bezpłatnych narzędzi udostępnianych przez producentów układów PLD (Altera i Xilinx): Quartus II 4.2 oraz WebPack ISE 6.3i. Są one dostępne na stronach internetowych: *www.altera.com* oraz *www.xilinx.com*.