ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Sprzętowa implementacja algorytmów

6. Generowanie liczb pseudolosowych przy użyciu rejestru LFSR.

Lider projektu: Politechnika Warszawska

Autor: Łukasz Mik

Akademia Nauk Stosowanych w Tarnowie

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

Declaration

This laboratory instruction has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

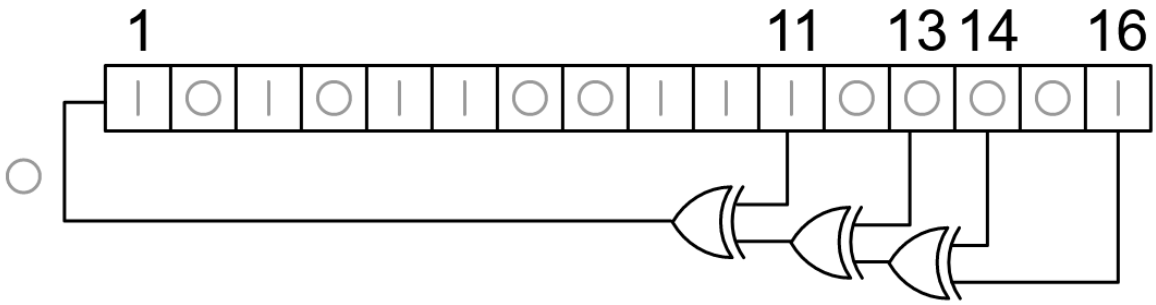
Copyright

Funding Disclaimer

This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

I. LSFR – podstawy teoretyczne.

LFSR to rejestr przesuwny z liniowym sprzężeniem zwrotnym, którego bit wejściowy jest funkcją liniową poprzedniego stanu. Do realizacji takiego sprzężenia zwrotnego najczęściej używa się bramki XOR, której wejścia są podłączone do wybranych wyjść rejestru natomiast wyjście bramki do jego wejścia. Początkowa wartość LFSR nazywana jest ziarnem, a ponieważ działanie rejestru jest deterministyczne, strumień wartości generowanych przez rejestr jest całkowicie zdeterminowany jego aktualnym (lub poprzednim) stanem. Ze względu na to, że rejestr LFSR ma skończoną liczbę stanów, po przejściu przez wszystkie z nich powraca do stanu początkowego i cykl powtarza się od nowa. Jednak LFSR z dobrze dobraną funkcją sprzężenia zwrotnego może wytworzyć sekwencję bitów, która wydaje się losowa i która ma bardzo długi cykl bez powtórzeń. Do typowych zastosowań liczników LFSR należą generatory: liczb pseudolosowych, szumu białego, bitowych sekwencji pseudolosowych. Implementacje sprzętowe i programowe LFSR są takie same. Przykład LFSR o długości 16 bitów pokazano na poniższym rysunku.



Bity wpływające na następny stan nazywane są odczepami. W tym przypadku odczepy są wyprowadzone z bitów o numerach 16,14,13 i 11. Najbardziej skrajny, prawy bit LFSR jest bitem wyjściowym. Na bitach z odczepów rejestru przesuwnego wykonywane są operacje XOR z bitem wyjściowym, a następnie sygnał zwrotny przekazywany jest do skrajnego lewego bitu. Dla rejestru o długości **n** bitów liczba wszystkich możliwych sekwencji bitów, a tym samym liczb pseudolosowych wynosi **2^n-1.** Układ odczepów sprzężenia zwrotnego w LFSR można wyrazić w arytmetyce pól skończonych jako wielomian mod 2. Oznacza to, że współczynniki wielomianu muszą wynosić 1 lub 0. Nazywa się to wielomianem sprzężenia zwrotnego lub odwrotnym wielomianem charakterystycznym. Na przykład, jeśli zaczepy znajdują się na 16, 14, 13 i 11 bitach (jak pokazano na rysunku), wielomian sprzężenia zwrotnego wynosi:

$$x^{16} + x^{14} + x^{13} + x^{11} + 1$$

„Jedynka” w wielomianie nie odpowiada odczepowi - odpowiada wejściu do pierwszego bitu (tj. x^0, co odpowiada 1). Potęgi wyrażeń **x** reprezentują wybrane bity, licząc od lewej strony. Pierwszy i ostatni bit są zawsze podłączone odpowiednio jako odczep wejściowy i wyjściowy. LFSR może osiągnąć maksymalną długość wtedy i tylko wtedy, gdy odpowiedni wielomian sprzężenia zwrotnego jest wielomianem pierwotnym.
Przykładowy kod implementacji 16-bitowego LFSR w języku C pokazano na poniższym listingu.

```
#include <stdint.h>
unsigned lfsr_fib(void)
{
  uint16_t start_state = 0xACE1u;  //Any nonzero start state will work
  uint16_t bit;    //Must be 16-bit to allow bit<<15 later in the code
  unsigned period = 0;

  do
  {   // taps: 16 14 13 11;
      //feedback polynomial: x^16 + x^14 + x^13 + x^11 + 1
     bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)) & 1u;
     lfsr = (lfsr >> 1) | (bit << 15);
     ++period;
  }
  while (lfsr != start_state);

  return period;
}
```

Liczniki LFSR mają prostszą logikę sprzężenia zwrotnego niż naturalne liczniki binarne lub liczniki z kodem Graya i dlatego mogą działać z wyższymi częstotliwościami zegara. Należy jednak upewnić się, że LFSR nigdy nie wejdzie w stan zerowy, na przykład ustawiając go przy uruchomieniu na dowolny inny stan w sekwencji. Tabela wielomianów pierwotnych (Xilinx XAPP 052) pokazuje jak LFSR można ułożyć w postaci Fibonacciego lub Galois, aby uzyskać maksymalne okresy generowania sekwencji bez powtórzeń. Można uzyskać dowolny inny okres dodając do LFSR, który ma dłuższy okres, pewną logikę, która skraca sekwencję, pomijając niektóre stany.

Strumienie wyjściowe LFSR są deterministyczne. Jeśli obecny stan i pozycje bramek XOR w LFSR są znane, można przewidzieć następny stan. Nie jest to możliwe w przypadku prawdziwie losowych zdarzeń. W przypadku LFSR o maksymalnej długości znacznie łatwiej jest obliczyć następny stan, ponieważ istnieje ich tylko ograniczona liczba dla każdej długości.

II. Generowanie liczb pseudolosowych przy użyciu LFSR.

Jak już napisano wcześniej, rejestr o długości **n** może generować ciąg pseudolosowy o maksymalnej długości $\mathbf{2^n-1}$. Kody binarne na wyjściu tak skonstruowanego licznika po przetworzeniu C/A generują szum o rozkładzie równomiernym.

Podczas zajęć w pierwszej kolejności zaimplementujemy generator liczb pseudolosowych, zbudowany z 4-bitowego rejestru przesuwnego i wielomianu bardzo niskiego rzędu. Taki generator będzie w stanie wygenerować 15 różnych sekwencji bitów (liczb). Gdy rejestr generatora osiągnie 15 stan to sekwencja się powtarza. Należy zauważyć, że LFSR jest jednobitowym generatorem losowym – ciąg binarny jest pobierany z jego wyjścia. Jeśli chcemy sprawdzać stany na każdym z odczepów to wtedy każdy odczep trzeba wyprowadzić w postaci wektora bitów. W języku VHDL można to zrobić ustawiając port w trybie **inout**. Dla 4 bitowego rejestru wielomian pierwotny ma postać: $\mathbf{x^3 + x^2 + 1}$

Praktyczna implementacja generatora liczb pseudolosowych w języku VHDL

KROK 1: Tworzymy projekt o nazwie **lfsr4bit** w programie ISE Webpack. Należy pamiętać o odpowiednich parametrach docelowego układu FPGA

KROK 2: Wewnątrz projektu tworzymy nowy plik źródłowy VHDL o nazwie **lfsr4bit**. W części nagłówkowej tego pliku importujemy niezbędne biblioteki oraz definiujemy porty we/wy jednostki projektowej.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lfsr4bit is
  port (
        rst : in std_logic;
        clk : in  std_logic;
        rand : out std_logic_vector(3 downto 0)  -- LFSR
  );
end entity;
```

KROK 3: Następnie w opisie architektury dodajemy proces, w którym będzie realizowany rejestr przesuwny w liniowym sprzężeniem zwrotnym.

```
architecture Behavioral of lfsr4bit is

  signal lfsr       : std_logic_vector (3 downto 0); -- LFSR register
  signal feedback      : std_logic;                  -- LFSR feedback

begin

  feedback <= not(lfsr(3) xor lfsr(2));-- feedback by polynomial x^3+x^2+1

  lfsr_pr : process (clk)
   begin
    if (rising_edge(clk)) then
      if (rst = '0') then
        lfsr <= (others=>'0');
      else
        lfsr <= lfsr(2 downto 0) & feedback;
      end if;
    end if;
  end process lfsr_pr;
  rand <= lfsr;                          -- parallel output of LFSR
end architecture;
```

KROK 4: Teraz trzeba powiązać porty we/wy z deklaracji jednostki do pinów układów FPGA. W tym celu tworzymy plik UCF (opcja *Implementation Constraints File*)i dodajemy w nim kod wiążący porty z pinami układu.

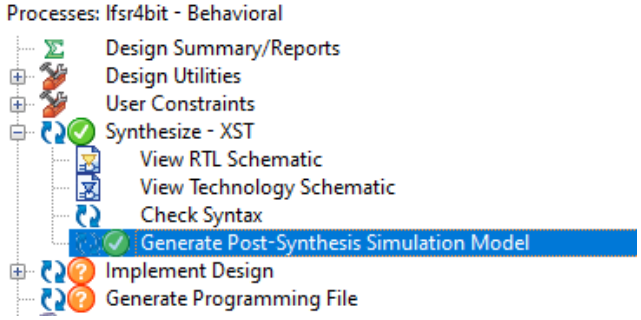
```
#clock
NET "clk"  LOC = P129 | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;
#reset
NET "rst"  LOC = P80  | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
#LFSR parallel output on 4 LEDs
NET "rand[0]"  LOC = P46 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "rand[1]"  LOC = P47 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "rand[2]"  LOC = P48 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "rand[3]"  LOC = P49 | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

Po uzupełnieniu plików źródłowych można przeprowadzić syntezę projektu, żeby sprawdzić czy nie ma jakichś błędów syntaktycznych bądź formalnych w języku VHDL. Narzędzie sprawdzi też, czy opis przygotowany przez nas opis jest syntezowalny do postaci cyfrowej.

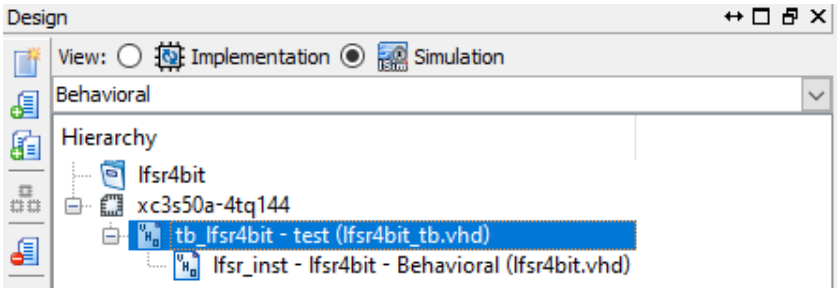
KROK 5: W kolejnym kroku dokonamy symulacji projektu. Tworzymy zatem nowy plik o nazwie **lfsr4bit_tb** wybierając przy tworzeniu opcję *VHDL Test Bench.* W oknie edycyjnym pliku z programem testowym wklej poniższy kod.

```vhdl
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.ALL;

entity tb_lfsr_v1 is
end entity;

architecture test of tb_lfsr_v1 is

  constant PERIOD   : time   := 83 ns;--frequency ~12MHz
  signal clk        : std_logic := '0';
  signal rst        : std_logic := '0';
  signal rand       : std_logic_vector(3 downto 0);
  signal endSim     : boolean   := false;

  component lfsr_v1 is
    port (
      rst      : in  std_logic;
      clk      : in  std_logic;
      rand     : out std_logic_vector(3 downto 0)
    );
  end component;

begin
  clk     <= not clk after PERIOD/2;
  rst    <= '1' after  PERIOD*2;
  endSim  <= true after PERIOD*60;

 -- End the simulation
 process
 begin
   if (endSim) then
     assert false
       report "End of simulation."
       severity failure;
   end if;
   wait until (clk = '1');
 end process;

 lfsr_inst : lfsr_v1
 port map (
   clk      => clk,
   rst      => rst,
   rand     => rand
 );
end architecture;
```

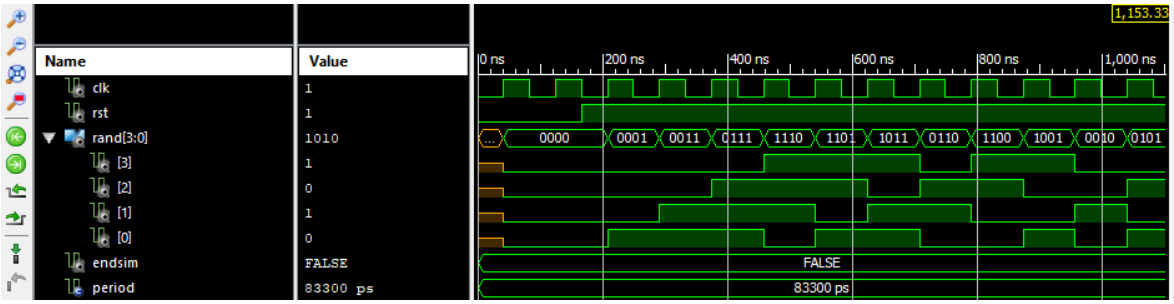
KROK 6: Generujemy behawioralny model symulacyjny (Post-Synthesis).



KROK 7: Zmieniamy w oknie projektu widok na symulacyjny, a z menu rozwijanego wybieramy opcję Behavioral.



KROK 8: W oknie procesów klikamy dwukrotnie na opcję *Simulate Behavioral Mode*, po czym otworzy się okno symulatora *iSim.* W poziomym pasku narzędziowym, w miejscu, gdzie podajemy czas zakończenia symulacji wpisujemy 5us. Dopasowujemy opcjami *Zoom* zawartość okna symulatora, żeby wyświetlić interesujący nas fragment przebiegów czasowych.



KROK 9: Żeby możliwe było śledzenie zmian stanów na wyjściu rejestru LFSR za pomocą diod LED konieczne jest zmniejszenie częstotliwości zegara taktującego rejestr. W tym celu wprowadzamy wewnątrz architektury dodatkowy sygnał o nazwie **clk_1Hz**, który będzie odpowiadał za generowanie sygnału taktującego o częstotliwości 1Hz. Po wprowadzeniu poprawek w kodzie opis architektury powinien przyjąć postać jak na przedstawionym niżej fragmencie kodu.

```
architecture Behavioral of lfsr4bit is
  signal lfsr          : std_logic_vector (3 downto 0); -- LFSR register
  signal feedback  : std_logic;                         -- LFSR feedback
  signal clk_1Hz     : std_logic;

begin

  feedback <= not(lfsr(3) xor lfsr(2));-- feedback by polynomial x^3+x^2+1
```

```
  process(clk)
       variable counter : integer:=0;
  begin
       if (rising_edge(clk)) then
      if (counter>6000000) then
        counter:=0;
       clk_1Hz <= not clk_1Hz;
      else
        counter:=counter+1;
      end if;
    end if;
  end process;

  lfsr_pr : process (clk_1Hz)
      begin
    if (rising_edge(clk_1Hz)) then
      if (rst = '0') then
        lfsr <= (others=>'0');
      else
        lfsr <= lfsr(2 downto 0) & feedback;
      end if;
    end if;
  end process lfsr_pr;
  rand <= lfsr;     -- parallel output of LFSR

end architecture;
```

KROK 10: Po skompilowaniu projektu i wygenerowaniu pliku konfiguracyjnego należy zaprogramować układ docelowy i sprawdzić poprawność działania generatora.

ZADANIE: Korzystając z przykładu omówionego na lekcji zaprojektuj generator liczb pseudolosowych z rejestrem LFSR o długości 8 bitów. Sprzężenia zwrotne ma być zrealizowane za pomocą wielomianu: $\mathbf{x^8 + x^6 + x^5 + x^4 + 1}$

References

- J. Majewski, P. Zbysiński – *Układy FPGA w przykładach.* Wydawnictwo BTC, Legionowo 2007.
- *User manual for Elbert V2 - Spartan 3A FPGA Development Board.* https://numato.com/docs/elbert-v2-spartan-3a-fpga-development-board/
- C. A. Chami – *Pseudo random generation Tutorial* - https://fpgaer.tech/?p=51
- Xilinx iSim User Guide - UG660 (v14.1) https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx14_1/plugin_ism.pdf