Tento článek je původním rukopisem textu publikovaného v časopise DPS Elektronika A-Z:
J. Šťastný. Implementace čítačů v číslicových systémech 2, DPS Plošné spoje od A do Z, no 4, pp. 11-14, 2011.
Bez souhlasu autora tohoto materiálu a redakce časopisu DPS a uvedení zdroje není povolena jakákoli další publikace, přetištění nebo distribuce tohoto materiálu nebo jeho části. Další podmínky použití jsou uvedeny na internetové stránce http://minimizedlogic.sweb.cz/.

Implementace čítačů v číslicových systémech 2

Jakub Šťastný
ASICentrum, s.r.o.
FPGA Laboratoř, Katedra teorie obvodů FEL ČVUT Praha

1 Úvod

V předchozím článku byly shrnuty základní vlastnosti čítačů, implementace a výhody a nevýhody binárního a Johnsonova čítače. V tomto příspěvku se budeme dále zabývat synchronní implementací čítače v Grayově kódu, kódu 1 z N a LFSR čítače. Poslední část, připravovaná do příštího čísla, bude potom věnována implementaci asynchronního čítače (*ripple counteru*) a shrnutí parametrů předvedených konstrukcí.

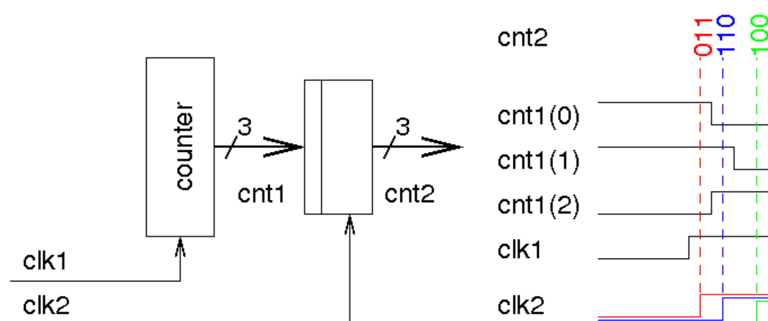
V celém textu označujeme počet registrů udržujících stav čítače jako N, počet stavů jako N_s. Jako f_{clk_max} označujeme maximální dosažitelnou pracovní frekvenci čítače, $T_{clk_min}=1/f_{clk_max}$ je pak minimální perioda hodinového cyklu.

Zkratkou MHVS budeme označovat maximální počet současně se měnících bitů na sběrnici na výstupu čítače – maximální Hammingovu vzdálenost dvou sousedních stavů čítače.

2 Grayův čítač

Binární čítač nemůžeme užít, potřebujeme-li vzorkovat jeho výstup hodinovým signálem asynchronním k hodinovému signálu, ze kterého běží čítač. V takovém případě by způsobilo vážné problémy to, že u binárního čítače se mohou dva sousední stavy lišit v podstatě v libovolném počtu bitů, viz obrázek 1. V levé části obrázku je zjednodušené schéma celé obvodové konfigurace. Binární čítač je řízen hodinami *clk1*, vzorkování je prováděno náběžnou hranou hodin *clk2.* Přitom hodiny *clk2* jsou plně asynchronní k hodinám *clk1*. Vidíme, že při naznačeném přechodu výstupu čítače mezi hodnotami 011 a 100 můžeme navzorkovat hodnoty 011, 110 i 100. Skutečné chování přitom závisí na konkrétních vzájemných časových posunech mezi signály *cnt1(0)*, *cnt1(1)* a *cnt1(2)*.

Abychom se podobným problémům vyhnuli, je nezbytné zajistit, aby se na výstupu čítače měnil vždy jen jeden bit. Právě tuto vlastnost splňuje Grayův čítač, speciální konstrukce stavové sekvence u něj zajišťuje MHVS 1. Tato vlastnost navíc umožňuje (za dodržení dalších dodatečných podmínek) navrhnout případný následný dekodér (blok *decoder* v obrázku 1 z prvního dílu seriálu [1]) tak, aby na jeho výstupech nebyly žádné statické ani dynamické hazardy. Stejně jako u binárního čítače – a oproti Johnsonovu čítači – je i zde výhodou minimální počet registrů nutných pro implementaci čítače procházejícího N_s stavy, $N=ceil(log_2(N_s))$. RTL schéma Grayova čítače lze nalézt v obrázku v příkladu 1, spolu s RTL VHDL implementací, stavová sekvence je potom v obrázku 2. Z důvodu úspory místa vynecháváme konstrukci ENTITY definující porty a generické parametry čítače. Použitá konstrukce je totožná s tou, kterou lze nalézt v příkladu 2 v předchozím dílu [1].



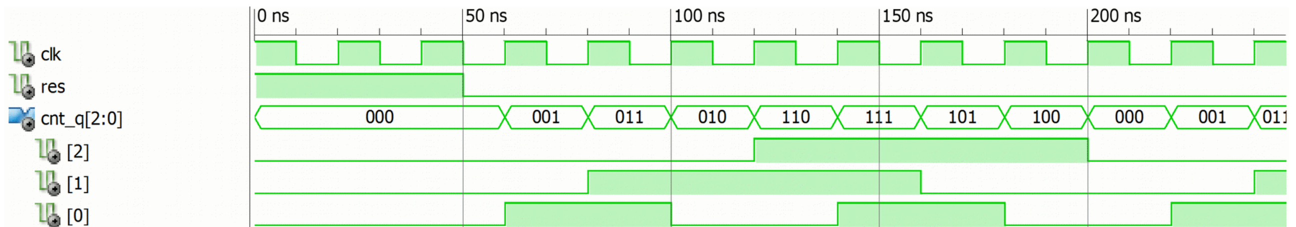
Obrázek 1: Vzorkování výstupu binárního čítače asynchronním signálem. Soubor sync_1.png.

Zřejmou nevýhodou Grayova čítače je větší plocha zabraná kombinační logikou pro generování následujícího stavu čítače; s tím souvisí i větší zpoždění v kritické cestě. Dlouhou kritickou cestu částečně odstraňuje řešení na obrázku 3 [2], ovšem za cenu většího množství registrů potřebných pro implementaci. A jako u binárního čítače, je i u Grayova

čítače změna počtu stavů pomocí ECO úpravy obtížná.

Použití Grayova kódu dále přináší omezení na délku stavové sekvence čítače; ta musí vždy obsahovat sudý počet stavů. V každém textu zabývajícím se kódy naleznete popis konstrukce Grayova kódu pro počet stavů $N_s=2^l$; nicméně je možné zkonstruovat Grayův kód pro obecný sudý počet stavů, viz [3].

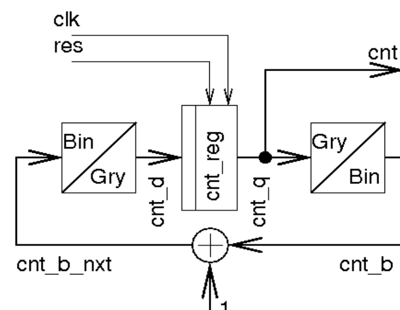
Všimněte si, že v čítači je použita binární sčítačka obklopená převodníky z a do Grayova kódu. Stavový registr nicméně obsahuje hodnotu v Grayově kódu. Výstup Grayova čítače musí být řízen přímo z registru, aby se předešlo zákmitům na výstupech jež by mohly být posléze navzorkovány jako legitimní hodnoty, více viz [4], kapitola 11.



Obrázek 2: Příklad běhu čítače, sekvence stavů je 000, 001, 011, 010, 110, 111, 101, 100. Soubor cnt_gray.png.

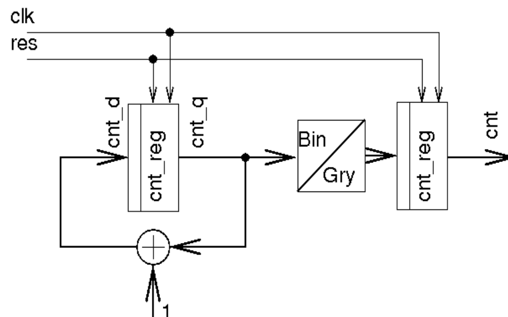
```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ARCHITECTURE rtl_gray OF cnt IS
  SIGNAL cnt_d     : std_logic_vector (N-1 DOWNTO 0);
  SIGNAL cnt_q     : std_logic_vector (N-1 DOWNTO 0);
  SIGNAL cnt_b     : std_logic_vector (N-1 DOWNTO 0);
  SIGNAL cnt_b_nxt : std_logic_vector (N-1 DOWNTO 0);

  COMPONENT gray2bin IS
    GENERIC (
           N   : natural := 4
        );
    PORT (
       gray : IN  std_logic_vector (N-1 DOWNTO 0);
       bin  : OUT std_logic_vector (N-1 DOWNTO 0)
    );
  END COMPONENT gray2bin;

  COMPONENT bin2gray IS
    GENERIC (
           N   : natural := 4
        );
    PORT (
       bin  : IN  std_logic_vector (N-1 DOWNTO 0);
       gray : OUT std_logic_vector (N-1 DOWNTO 0)
    );
  END COMPONENT bin2gray;

BEGIN
  reg_cnt : PROCESS (clk, res)
  BEGIN
      IF res='1' THEN
        cnt_q <= (OTHERS => '0');
      ELSIF clk'EVENT AND clk='1' THEN
        cnt_q <= cnt_d;
      END IF;
  END PROCESS reg_cnt;

  i_gray2bin:gray2bin
    GENERIC MAP (N => N)
    PORT    MAP (
       gray => cnt_q,
       bin  => cnt_b
    );

  cnt_b_nxt <= std_logic_vector(unsigned(cnt_b)+1);

  i_bin2gray:bin2gray
    GENERIC MAP (N => N)
    PORT    MAP (
       bin  => cnt_b_nxt,
       gray => cnt_d
    );
  cnt_out <= cnt_q;
END ARCHITECTURE rtl_gray;
```

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY gray2bin IS
    GENERIC (N   : natural := 4);
    PORT (
       gray : IN  std_logic_vector (N-1
DOWNTO 0);
       bin  : OUT std_logic_vector (N-1
DOWNTO 0)
    );
END ENTITY gray2bin;

ARCHITECTURE rtl OF gray2bin IS
  SIGNAL bin_i : std_logic_vector (N-1
DOWNTO 0);
BEGIN
  bin_i(N-1) <= gray(N-1);
  g2b:FOR index IN N-2 DOWNTO 0 GENERATE
    bin_i(index) <= gray(index) XOR
bin_i(index+1);
  END GENERATE g2b;
  bin <= bin_i;
END ARCHITECTURE rtl;

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY bin2gray IS
    GENERIC (
    N   : natural := 4
 );
    PORT (
       bin  : IN  std_logic_vector (N-1
DOWNTO 0);
       gray : OUT std_logic_vector (N-1
DOWNTO 0)
    );
END ENTITY bin2gray;

ARCHITECTURE rtl OF bin2gray IS
BEGIN
  gray <= bin XOR ('0'&bin((N-1) DOWNTO
1));
END ARCHITECTURE rtl;
```



Soubor cnt_gray_sch.png.

Příklad 1: Grayův čítač - RTL kód a schéma.

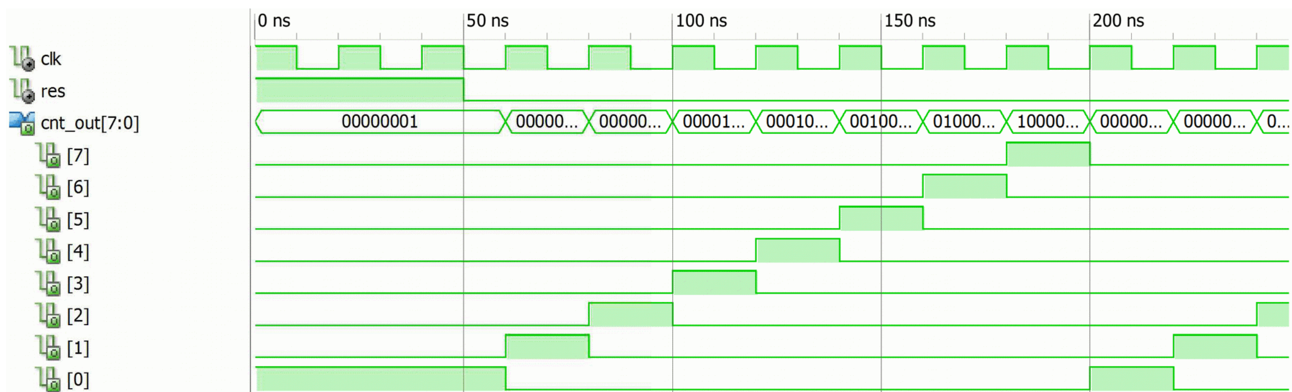


Obrázek 3: Implementace Grayova čítače s kratšími kritickými cestami. *Soubor cnt_gray2_sch.png*

3 Čítač v kódu 1 z N

Čítač v kódu „1 z N“ (*one-hot encoding*) je implementován jak je uvedeno v příkladu 2, v obrázku 4 je potom příklad stavové sekvence. Vidíme, že je každý stav zakódovaný jako binární řetězec složený ze samých nul jen s jednou jedničkou. Délka stavu čítače je rovná počtu stavů, potřebujeme tedy $N=N_s$ registrů. Mezi registry zde není žádná kombinační logická funkce, počáteční nastavení je zajištěno resetem obvodu a jednička pak „samovolně“ obíhá posuvným registrem.

Jednoznačnou nevýhodou čítače je velké množství registrů potřebných pro jeho implementaci; to může vést ke zvýšené spotřebě elektrické energie v hodinovém stromu čítače. Kódování „1 z N“ má ale i řadu výhod. Stejně jako u Johnsonova čítače je i zde velmi redukovaná kombinační logická funkce pro generování následujícího stavu, to umožňuje čítači pracovat na vyšší hodinové frekvenci, než v případě binárního, či Grayova čítače. I zde jsou omezeny hazardy na výstupu případného navazujícího kombinačního detektoru, potlačení nicméně – na rozdíl od Johnsonova kódování – není absolutní, MHVS je 2. Dále je zde jednoduše možné pomocí ECO úpravy vložit do čítače další stav.



Obrázek 4: Příklad běhu čítače, sekvence stavů:00000001, 00000010, 00000100, 00001000, 00010000, 00100000, 01000000, 10000000. *Soubor cnt_one_hot.png*

```
ARCHITECTURE rtl_one_hot OF cnt IS
  SIGNAL cnt_d : std_logic_vector (N-1 DOWNTO 0);
  SIGNAL cnt_q : std_logic_vector (N-1 DOWNTO 0);
BEGIN
  reg_cnt : PROCESS (clk, res)
  BEGIN
      IF res='1' THEN
        cnt_q <= (0=>'1', OTHERS => '0');
      ELSIF clk'EVENT AND clk='1' THEN
        cnt_q <= cnt_d;
      END IF;
  END PROCESS reg_cnt;
  cnt_d   <= cnt_q(N-2 DOWNTO 0)&cnt_q(N-1);
  cnt_out <= cnt_q;
END ARCHITECTURE rtl_one_hot;
```

clk res
S R R R R R R R
cnt_q

Soubor cnt_onehot_sch.png

Příklad 2: „1 z N“ čítač - RTL kód a schéma.

4 LFSR

LFSR čítač (*Linear Feedback Shift Register*) patří mezi poněkud exotičtější konstrukce, kterým se návrháři spíše vyhýbají (autor textu si pamatuje na několik vášnivých diskuzí o vhodnosti jeho použití i z vlastní praxe). LFSR čítače mají nicméně velké množství aplikací a jsou naprosto nepostradatelné v mnoha aplikacích počínaje kryptografií, přes vysílání v rozptýleném spektru až po obyčejné čítače.

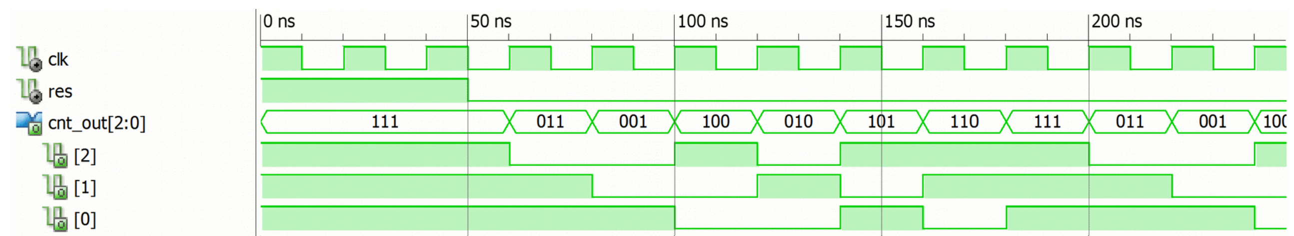
LFSR je v principu posuvný registr doplněný o jedno/několik hradel XOR ve zpětné vazbě sloužících pro generování sekvence stavů. Na rozdíl od binárního čítače LFSR o délce N bitů prochází jen $N_s=2^N-1$ stavy, jeden stav je vždy zakázaný (00....000 pro konstrukci užívající hradel XOR, případně 11...111 pro konstrukci s hradlem XNOR). Pohledem na schéma v příkladu 3 lze snadno zjistit, že ze zakázaného stavu (zde 000) se čítač nedostane, dojde k jeho zaseknutí (*lockup*). U LFSR čítačů je tak nezbytné zajistit pomocí resetu vhodné počáteční podmínky (zde tedy resetovat čítač do libovolného nenulového stavu, například 111).

Příklad 3 ukazuje VHDL kód a schéma LFSR čítače procházejícího sedmi stavy, sekvence stavů je přitom zachycena v obrázku 5. Je zřejmé, že za extrémní jednoduchost bloku platíme cenu v podobě zdánlivě chaotické stavové sekvence. Poznamenejme zde, že stavová sekvence má skutečně některé vlastnosti náhodného procesu a proto jsou LFSR čítače často užívány v aplikacích, kde je třeba používat pseudonáhodná čísla. O statistických vlastnostech je možné se dozvědět více v české knize [6]. Stejně jako u Johnsonova čítače, nebo čítače v kódu 1 z N je i zde kombinační logická funkce pro generování dalšího stavu velmi redukovaná. Čítač je tak relativně malý a může pracovat s vyšší f_{clk_max}.

Teorie vlastní konstrukce LFSR čítačů je poměrně komplexní, zájemce o detaily odkazujeme na texty [6,7,8], mnoho informací lze také získat prostým hledáním hesla „LFSR counter" na vyhledávači Google.

Méně zřejmou nevýhodou konstrukce LFSR čítače je o něco větší obtížnost návrhu obecného bloku čítače. Pro jeho správnou funkci je třeba správné konfigurace zpětných vazeb v posuvném registru (vyjádřené pomocí tzv. generujícího polynomu, více viz např. [6]), přitom konfigurace je pro každou délku čítače unikátní. Tabulka 1 (převzatá z [9]) obsahuje konfigurace LFSR čítačů pro různé šířky stavového registru. Podíváme-li se na zvýrazněný řádek a srovnáme-li ho se schématem v obrázku v příkladu 3, můžeme snadno nahlédnout na to, jak je čítač navržen.

Závěrem poznamenejme, že případná modifikace čítače pomocí ECO úpravy nebývá obtížná.



Obrázek 5: Příklad běhu čítače, sekvence stavů 111, 011, 001, 100, 010, 101, 110. Všimněte si, že stavů je jen sedm, ne osm. Soubor cnt_lfsr.png.

```
ARCHITECTURE rtl_lfsr OF cnt IS
  SIGNAL cnt_d   : std_logic_vector (N-1 DOWNTO 0);
  SIGNAL cnt_q   : std_logic_vector (N-1 DOWNTO 0);
  SIGNAL cnt_xor : std_logic;
BEGIN
  reg_cnt : PROCESS (clk, res)
  BEGIN
      IF res='1' THEN
        cnt_q <= (OTHERS => '1');
      ELSIF clk'EVENT AND clk='1' THEN
        cnt_q <= cnt_d;
      END IF;
  END PROCESS reg_cnt;
  cnt_xor <= cnt_q(1) XOR cnt_q(0);
  cnt_d   <= cnt_xor&cnt_q(N-1 DOWNTO 1);
  cnt_out <= cnt_q;
END ARCHITECTURE rtl_lfsr;
```

clk res S 2 S 1 S 0 cnt_q

soubor cnt_lfsr_sch.png

Příklad 3: LFSR čítač - RTL kód a schéma.

N	Registry pro zpětnou vazbu
1	0
2	1,0
3	**1,0**
4	1,0
5	2,0
6	1,0
7	1,0
8	6,5,1,0
9	4,0
10	3,0
11	2,0
12	7,4,3,0
13	4,3,1,0
14	12,11,1,0
15	1,0
16	5,3,2,0

Tabulka 1: Zpětné vazby v LFSR čítačích pro různé šířky stavového registru.

5 Literatura

[1] Jakub Šťastný. Implementace čítačů v číslicových systémech. DPS Plošné spoje od A do Z, XXXX, str. XXX-XXX.
[2] Clifford Cummings. Simulation and synthesis techniques for asynchronous FIFO designs. SNUG San Jose 2002.
[3] Clive Maxfield. Yet another Gray code conundrum, July 19, 2007
http://www.pldesignline.com/201002340;jsessionid=SEEHOQLT04WKZQE1GHPSKHWATMY32JVN?printableArticle=true
(kontrolováno 1.5.2011)
[4] Jakub Šťastný. FPGA Prakticky, BEN Praha 2011.
[5] Steve Golson. State machine design techniques for Verilog and VHDL, 1994.
[6] Jiří Adámek. Kódování. Sešit XXXI, matematika pro vysoké školy technické, SNTL 1989.
[7] Xilinx. Linear Feedback Shift Registers in Virtex Devices: XAPP 210.
[8] Xilinx. Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators: XAPP 052.
[9] Tom Balph. LFSR counters implement binary polynomial generators. EDN Design feature,
http://www.edn.com/archives/1998/052198/11df_06.htm (kontrolováno 1.5.2011)