Aritmetické a logické operace (násobení)

INP - cvičení 5

Zdeněk Vašíček, 2010
vasicek@fit.vutbr.cz

Násobení

- Operace násobení se v HW realizuje jako postupné přičítání vhodně posunutého násobence na základě znalosti jednotlivých bitů násobitele.

- Existuje řada přístupů (Riple Carry Array, Carry Save Array, Wallace tree, Booth Encoding), které se v zásadě liší
  - v ploše, kterou násobička na čipu zabírá a
  - v rychlosti (zpoždění násobičky).

Násobení 8b čísel bez znaménka v dvojkové soustavě

```
  01010010 (násobenec)          82
× 01111101 (násobitel)        × 125
----------                    -----
  01010010                      410
 000000000                     1640
 010100100                     8200
----------                    -----
 0101001000                   10250
 01010010000
 010100100000
 0101001000000
 01010010000000
----------------
0010100000001010  (součin)
```

Výsledný součin musí obsahovat 8+8 = 16 bitů

Násobení 8b čísel se znaménkem v dvojkové soustavě

- Naivní implementace – zapamatování znaménka a převod na kladná čísla (nepoužívá se).
- Lze použít předchozí algoritmus je však nutné rozšířit (na šířku součinu) / šířit znaménko.

```
  11111111101011110  (násobenec)
× 0000000001111101  (násobitel)
------------------
  11111111010010010
  00000000000000000
  11111101010010000
  11111010100100000
  11110101001000000
  11101010010000000
  11010100100000000
  00000000000000000
------------------
1101000100001010  (součin)
```

```
   -82
 × 125
------
   410
  1640
  8200
------
-10250
```

Optimalizace z pohledu zpoždění

- Násobení sestává z řízeného přičítání dílčích součinů (násobitele) k průběžnému výsledku. Pokud je cílem proces zrychlit, musíme
  - buď urychlit operaci přičtení dílčího součinu nebo
  - počet operací přičtení redukovat (použitím jiné reprezentace).

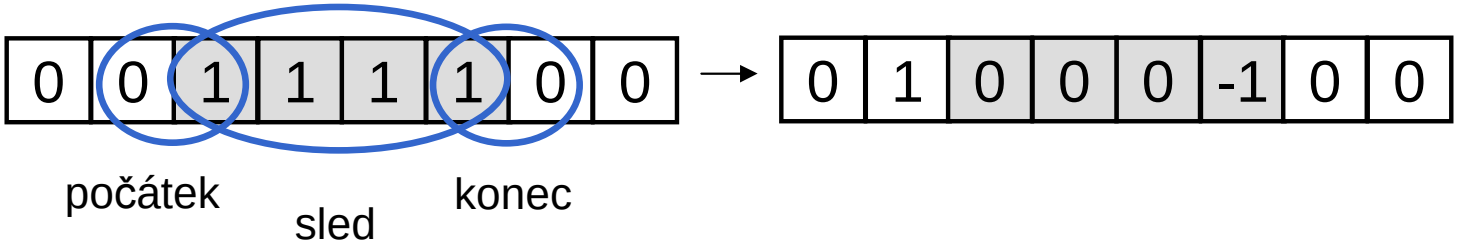
- Motivace: hodnotu $15_{10} = 1111_2$ lze reprezentovat také jako rozdíl

  $16 - 1 = 10000_2 - 00001_2$

- Při použití předchozího algoritmu se pro výpočet součinu 7 x 15 musí vyčíslit výraz $0111_2 + 01110_2 + 011100_2 + 0111000_2$ avšak při použití této reprezentace stačí vyčíslit výraz $01110000_2 - 0111_2$. Lze tedy ušetřit 50% operací.

Boothův algoritmus

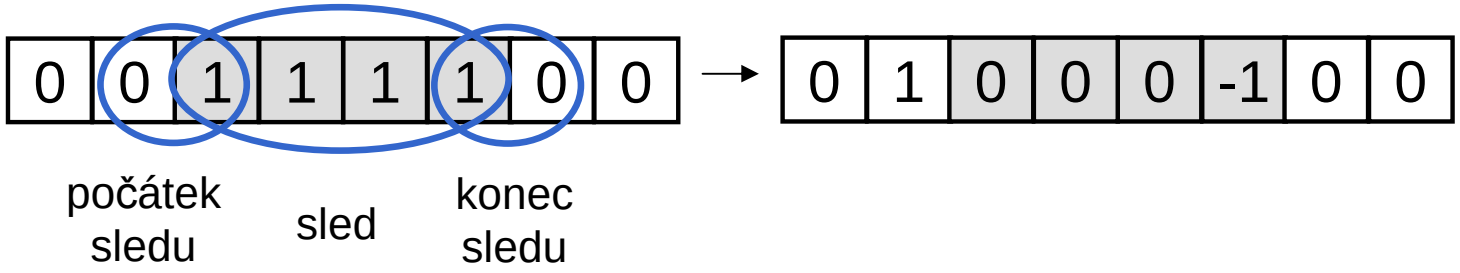
- Boothův algoritmus je algoritmus určený k redukci počtu operací, který využívá mimo operaci přičtení násobence také jeho odečtení.
- Princip: detekovat a nahradit řadu po sobě následujících jedniček (sled jedniček) v násobiteli a tím redukovat počet operací.



- Existuje řada variant, základní verze je Boothovo překódování radix 2 využívající znalosti bitu a jeho pravého souseda (je-li MSB vlevo).

Boothův algoritmus (radix 2) – tvorba kódu

- Cílem je detekovat sled jedniček pomocí znalostí hodnoty bitu a jeho souseda.



- Princip konstrukce kódu

hodnota překódovávaného bitu	hodnota sousedního bitu (bit vpravo)	kód	popis situace
0	0 / neexistuje	0	mimo sled
0	1	1	počátek sledu
1	1	0	sled jedniček
1	0 / neexistuje	-1	konec sledu

- Neefektivita: za sled je považována i osamocená jednička (010), která je nahrazena dvěma operacemi (1-10)

Násobení 8b čísel – Boothův algoritmus – radix 2

82 x 125 = 01010010 x 01111101

125:

0	1	1	1	1	1	0	1

→

1	0	0	0	0	-1	1	-1

```
      0 1 0 1 0 0 1 0     (82)
×   1 0 0 0 0-1 1-1     (125)
-----------------------
  1111111110101110     (-82)
  0000000010100100     (+82 << 1)
  1111111010111000     (-82 << 2)
  0010100100000000     (+82 << 7)
-----------------------
  0010100000001010
```

Počet operací se redukoval z 6 na 4

Boothův algoritmus (radix 4) – tvorba kódu

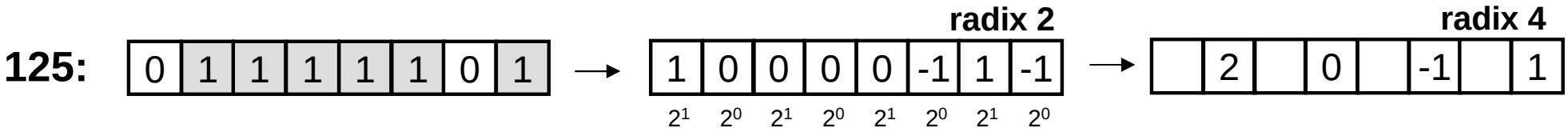
- Umožníme-li používat dvojnásobnou zápornou a kladnou hodnotu násobence, můžeme dále redukovat počet operací (vyrobit v HW požadované násobky není obtížné).
- **Kód vyššího řádu lze vytvořit z kódu radix 2 přiřazením vah 2^i jednotlivým pozicím.**

hodnota překódovávaných bitů	hodnota sousedního bitu (bit vpravo)	kód radix 2	kód radix 4
00	0	0 0 →	0
00	1	0 1 →	1
01	0	1-1 →	1
01	1	1 0 →	2
10	0	-1 0 →	-2
10	1	-1 1 →	-1
11	0	0-1 →	-1
11	1	0 0 →	0

- Rychlá kontrola - nejvyšší bit určuje znaménko (1 – záporné, 0 – kladné)

Násobení 8b čísel – Boothův algoritmus – radix 4

82 x 125 = 01010010 x 01111101

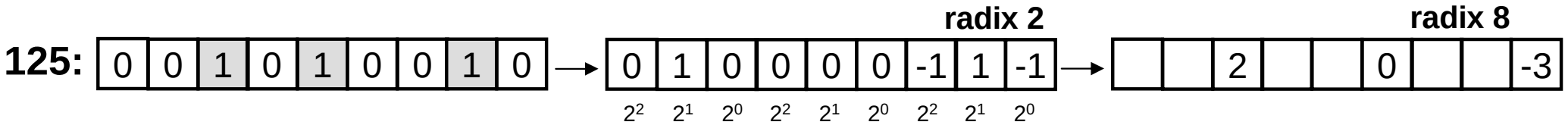



```
          0 1 0 1 0 0 1 0      (82)
×       2   0  -1   1        (125)
-------------------------------
  0000000001010010           (+82)
  1111111010111000           (-82 << 2)
  0010100100000000           (+164 << 6)
-------------------------------
  0010100000001010
```

- Pozor na důslednou práci se znaménkem a potřebný počet bitů pro zakódování dvojnásobku násobence. Jednotlivé dílčí součiny je zapotřebí posouvat o 2 bity vlevo!
- Počet operací se redukoval z 6 na 3

Násobení 8b čísel – Boothův algoritmus – radix 8

82 x 125 = 01010010 x 01111101

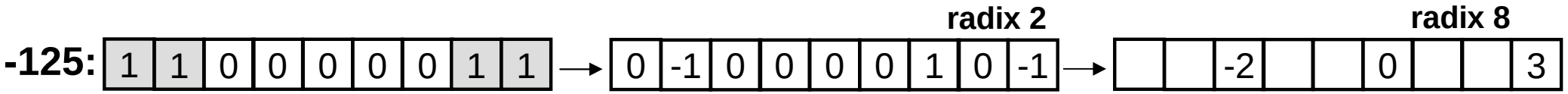



```
    0 0 1 0 1 0 0 1 0        (82)
×       2     0    -3       (125)
-------------------------------
  1111111100001010        (-246)
  0010100100000000        (+164 << 6)
-------------------------------
  0010100000001010
```

- Počet bitů musí být násobkem 3 (přechod na 9-bitová čísla)
- Počet operací se redukoval z 6 na 2.

Násobení 8b čísel – Boothův algoritmus – radix 8

-82 x -125 = 110101110 x 110000011




```
    1 1 0 1 0 1 1 1 0      (-82)
×     -2    0     3      (-125)
 ---------------------------------
  1111111100001010       (-246)
  0010100100000000       (+164 << 6)
 ---------------------------------
  0010100000001010
```

- Počet bitů musí být násobkem 3 (přechod na 9-bitová čísla)
- Počet operací se redukoval z 4 na 2 (viz počet jedničkových bitů násobitele).

Boothův algoritmus – obecný postup

Postup podle obecného Boothova algoritmu:

1. Zopakujeme nejvyšší bit skupiny (skupina je *k*-tice bitů, *k* dáno radixem)
2. K takto získané hodnotě přičteme hodnotu nejvyššího bitu skupiny vpravo od překódovávané skupiny
3. Výsledná hodnota je binárním vyjádřením relativní číslice

Překódujte číslo -121 na 9 bitech pomocí radix 8 ($2^k = 8$, $k = 3$)

-121 =

1 1 0	0 0 0	1 1 1
1 1 1 0	0 0 0 0	1 1 1 1
1 1 1 0	0 0 0 1	1 1 1 1
-2	1	-1

Ověření správnosti výsledku:

$-121 = -2 \cdot 26 + 1 \cdot 23 - 1 \cdot 20 = -2 \cdot 64 + 1 \cdot 8 - 1 \cdot 1$

Boothův algoritmus

- Q: Překódujte číslo -257 na 12 bitech do Boothova kódu radix 8.
- Q: Vynásobte čísla -81 x -78 s využitím Boothova algoritmu.
- Q: Kolik operací ušetříme při násobení čísel -81 x -78 na 8 bitech zavedením Boothova překódování radix 4?

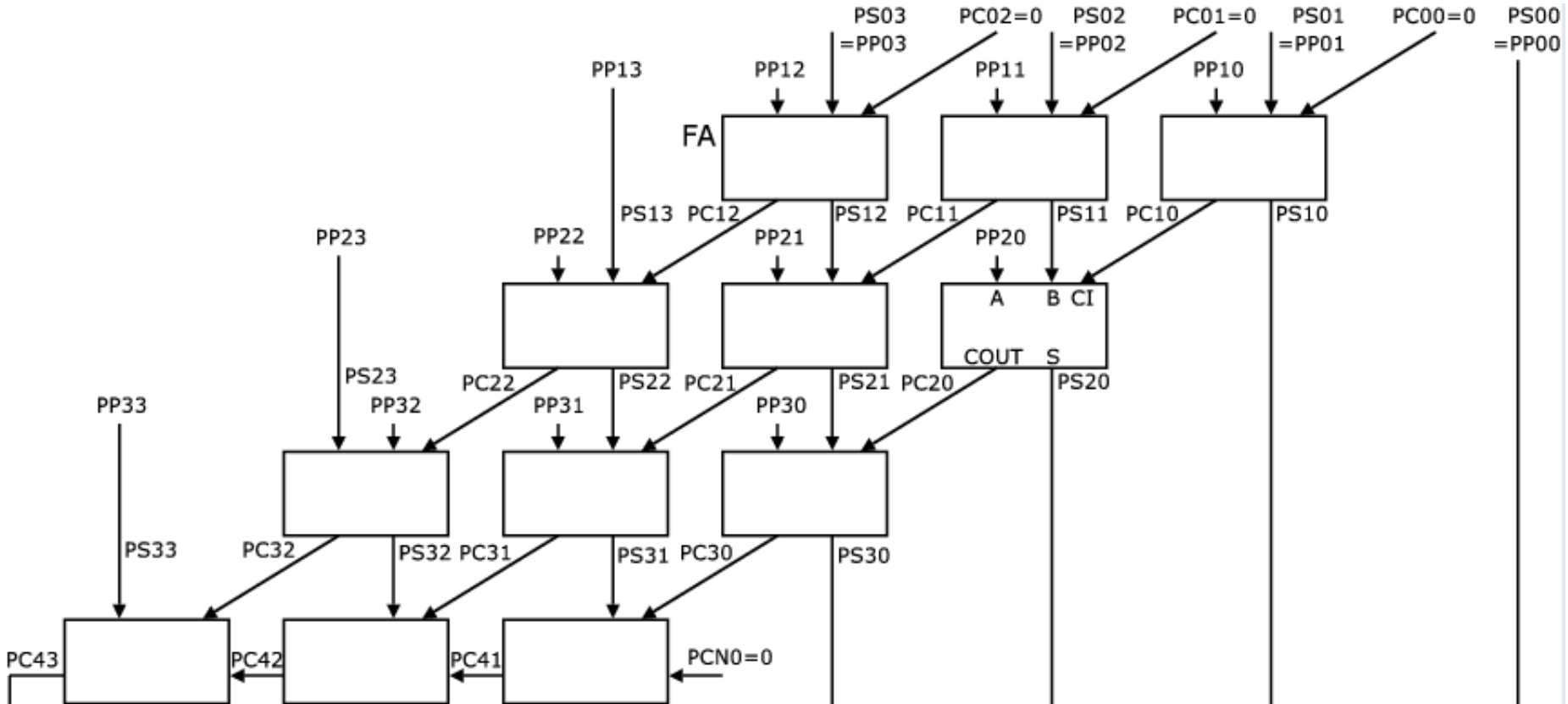
Násobička s uchováním přenosu ve VHDL

```
entity MULT is
  port (
    A, B : in  std_logic_vector(3 downto 0);
    PROD : out std_logic_vector(7 downto 0));
end MULT;

architecture RTL1 of MULT is
  constant N : integer := 4;
  type tarr is array(0 to N) of
               std_logic_vector(N-1 downto 0);
  signal PP, PC, PS : tar;

  component FA -- full adder
  port (
    A, B, CI : in  std_logic;
    S, COUT  : out std_logic);
  end component;

component ANDG -- and gate
  port (
    A, B : in  std_logic;
    C    : out std_logic);
  end component;
```

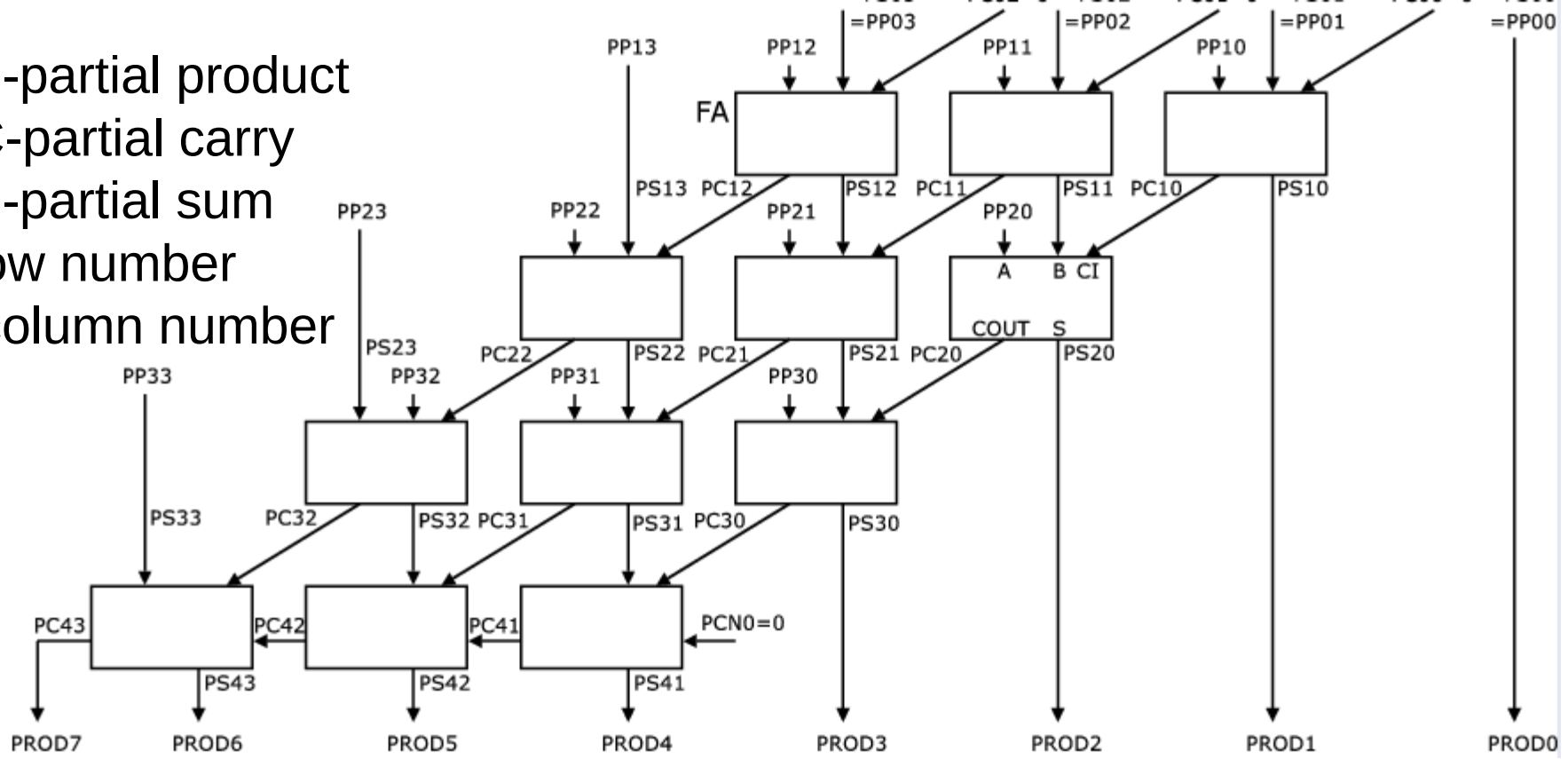
PP-partial product
PC-partial carry
PS-partial sum
j-row number
k-column number

```
begin
  pgen : for j in 0 to N-1 generate
    pgen1 : for k in 0 to N-1 generate
      and0 : ANDG port map (
        A => A(k), B => B(j), C => PP(j)(k));
    end generate;
    PC(0)(j) <= '0';
  end generate;
  PS(0) <= PP(0);
  PROD(0) <= PP(0)(0);  -- nultý bit výsledku
```

částečné součiny



PP-partial product
PC-partial carry
PS-partial sum
j-row number
k-column number



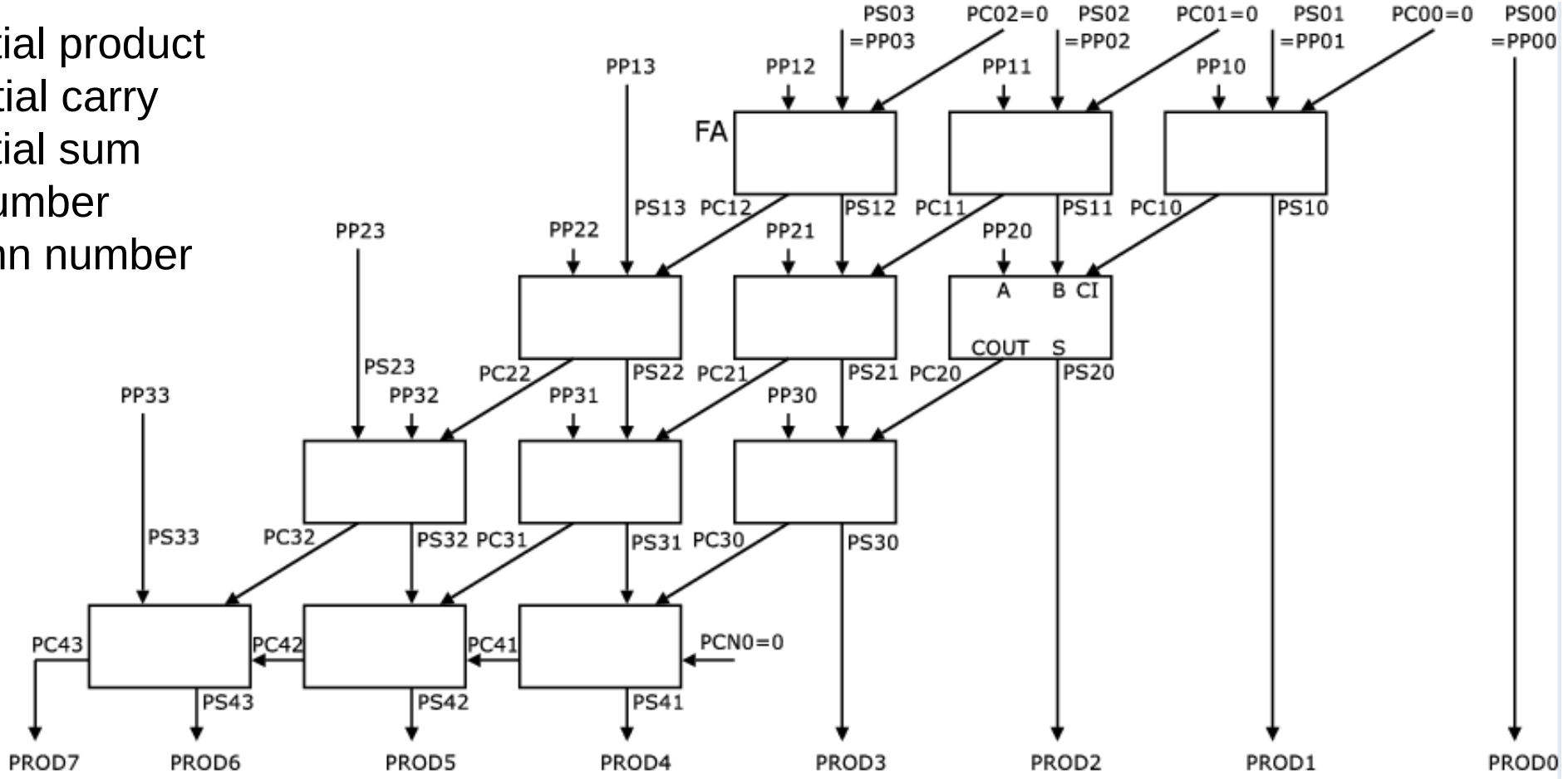
sčítačky řádků 1..N-1

```
addr : for j in 1 to N-1 generate
    addc : for k in 0 to N-2 generate
      fa0 : FA port map (
        A => PP(j)(k), B => PS(j-1)(k+1), CI => PC(j-1)(k),
        S => PS(j)(k), COUT => PC(j)(k));
    end generate;
    PROD(j)    <= PS(j)(0);
    PS(j)(N-1) <= PP(j)(N-1);
  end generate;
  PC(N)(0) <= '0';
```

```
  addlast : for k in 1 to N-1 generate
    fa1 : FA port map (
      A => PS(N-1)(k), B => PC(N-1)(k-1), CI => PC(N)(k-1),
      S => PS(N)(k), COUT => PC(N)(k));
  end generate;
  PROD(2*N-1) <= PC(N)(N-1);
  PROD(2*N-2 downto N) <= PS(N)(N-1 downto 1);
end RTL1;
```

Sčítačky v posledním řádku

PP-partial product
PC-partial carry
PS-partial sum
j-row number
k-column number



Plocha a doba výpočtu

- Předpokládejme násobení N-bitů x M-bitů
- Potřebujeme (N-1)*M sčítaček (některé mohou být poloviční)
- Zpoždění (předpokládejme, že zpoždění 1 FA odpovídá zpoždění 2 hradel)
  - + 1 x hradlo AND (první částečný součin)
  - + M-1 řádků sčítaček
  - + N-1 sčítaček v posledním řádku

+ Celkem 2*(M-1 + N-1) + 1 [zpoždění hradla]

Boothovo překódování s radixem 4 ve VHDL

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

entity RECODER is
  port (
    DIN  : in  std_logic_vector(15 downto 0);
    BIT3 : in  std_logic_vector( 2 downto 0);
    DOUT : out std_logic_vector(16 downto 0));
end RECODER;

architecture RTL of RECODER is
  constant N : integer := 16;
  subtype bitn1 is std_logic_vector(N downto 0);

  function COMP2 (D : in bitn1) return bitn1 is
    variable Dn : bitn1;
  begin
    Dn := not D;
    return(Dn+1);
  end COMP2;
```

Funkce ve VHDL vracející dvojkový doplněk čísla D

Boothovo překódování s radixem 4 ve VHDL (2)

```
begin
  process (DIN, BIT3)
  begin
    case BIT3 is
      when "001" | "010" =>
        DOUT <= DIN(N-1) & DIN;
      when "101" | "110" =>
        DOUT <= COMP2(DIN(N-1) & DIN);
      when "100" =>
        DOUT <= COMP2(DIN & '0');
      when "011" =>
        DOUT <= DIN & '0';
      when others =>
        DOUT <= (others => '0');
      end case;
  end process;
end RTL;
```

překódovávaný blok			kód
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0