

模 $2^n - 2^k - 1$ 加法器高效 VLSI 设计与实现

马 上,叶燕龙,胡剑浩

(电子科技大学 通信抗干扰技术国家级重点实验室,四川 成都 610054)

摘 要: 模加法器是余数系统(Residue Number System, RNS)的基本运算单元, $2^n - 2^k - 1$ 形式的余数基易于构建大动态范围和具有优良复杂度平衡性的多通道余数基.基于前缀运算和进位修正算法提出了一类新的模 $2^n - 2^k - 1$ 加法通用算法及其 VLSI 实现结构.该算法消除了重复的进位信息计算,且可采用任意已有的前缀运算结构.与同类型模加法器的分析对比结果表明,提出的模 $2^n - 2^k - 1$ 加法器具有优良的“面积 $\times$ 时延”特性.

关键词: 余数系统;模加法器;并行前缀;进位修正;超大规模集成电路

中图分类号: TP338.6

文献标识码: A

文章编号: 1000-7180(2010)10-0001-07

Design and Implementation of High Efficient Modulo $2^n - 2^k - 1$ Adder VLSI for RNS

MA Shang, YE Yan-long, HU Jian-hao

(National Key Laboratory of Communication,
University of Electronic Science and Technology of China, Chengdu 610054, China)

Abstract: The design of high efficient modulo adders is important for DSP algorithms implemented by Residue Number System (RNS). Moduli sets of this form $2^n - 2^k - 1$ offer many advantages, such as larger dynamic range and excellent balance among the RNS channels. In this paper, a general algorithm and its VLSI implementation structure are proposed for the modulo $2^n - 2^k - 1$ adder. The proposed algorithm is based on the techniques of prefix operation and carry correction, which eliminates the re-computation of carries. And any existing prefix operation structure can be adopted in the proposed structure. Compared with the same modulo $2^n - 2^k - 1$ adders with different structures, the proposed modulo adder offers better “area * delay”.

Key words: RNS; modulo adders; parallel prefix; carry correctness; VLSI Circuits

1 引言

余数系统(Residue Number System, RNS)在乘法和加法运算中各运算通道间无进位传播,减小了芯片关键路径时延,可有效降低芯片面积和功耗,因此在具有大量乘加运算的数字信号处理(Digital Signal Processing, DSP)系统中得到了广泛关注和研究^[1-2].在RNS中,模加法器是其基本运算单元,优化的模加法器设计是余数系统的应用中的基本问题之一.在模加法器设计中,通常需要同时处理两个

加法运算,并根据其中一个加法的最高进位选择某一个作为最终运算结果^[3-5].

按照模加法的余数基类型,可分为通用模加法器和特殊基模加法器两类.在通用模加法器设计中,不可避免地会使用重复的计算单元.由于特殊形式模加法器具有更好的实现性能,因此得到了广泛而深入的研究.Patel等人提出了一种基于进位修正的模 $2^n - (2^{n-2} + 1)$ 加法器实现结构^[6],在其进位计算模块中仅计算 $A + B + T$ 的进位信息,然后根据 $A + B + T$ 的最终进位对其各比特的进位进行修正

得到模加运算所需要的进位信息,从而消除了重复的进位计算单元,但其进位计算结构固定。

具有 $2^n - 2^k - 1 (1 \leq k \leq n - 2)$ 形式的余数基在构建高性能、高动态范围和优良通道间平衡性的余数系统中具有重要应用价值。本文基于并行前缀和进位修正算法,提出了一类新的具有 $2^n - 2^k - 1 (1 \leq k \leq n - 2)$ 形式的模加法设计方法和其高效 VLSI 实现结构。

2 相关理论基础

2.1 模加法

对于整数 $A, B (A, B \in [0, p))$ 进行模 p 的加法运算定义为

$$C = \langle A + B \rangle_p = \begin{cases} A + B & A + B < p \\ A + B - p & A + B \geq p \end{cases} \quad (1)$$

式(1)的一种等效表示为

$$C = \begin{cases} A + B & A + B + T < 2^n \\ \langle A + B + T \rangle_{2^n} & A + B + T \geq 2^n \end{cases} \quad (2)$$

式中, $n = \lceil \log_2^p \rceil$, 即 n 为不小于 $\log_2^p$ 的最小整数。式(2)表明,若 $A + B + T$ 的进位为“1”,则模加法的结果为 $A + B + T$ 的低 n 比特,反之则为 $A + B$, 这是绝大多数模加法器设计所遵循的基本原则。

2.2 并行前缀加法

二进制加法器的进位传播路径长度是影响其性能的主要因素,并行前缀运算可在进位传播的路径长度和实现面积之间做到灵活折中以满足具体应用要求,它是加法器设计中广泛被采用的方法。基于前缀算法的二进制加法器可分为三个清晰的处理模块,即数据预处理、前缀运算和求和运算模块,如图1所示。

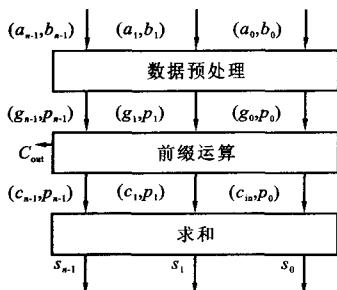


图1 基于前缀运算加法器结构及前缀运算

数据预处理模块根据加数 A, B 按位生成前缀运算所需的前缀运算对:

$$(g_i, p_i) = (a_i b_i, a_i \oplus b_i) \quad (3)$$

式中, g_i 和 p_i 分别表示第 i 位 ($i = 0, 1, \dots, n - 1$) 的进位生成和进位传播位, p_i 也被称为部分和信息, $g_i = a_i b_i$ (即 a_i 同 b_i 进行二进制“与”逻辑运算), $p_i = a_i \oplus b_i$ (“ $\oplus$ ”表示二进制“异或”逻辑运算)。

前缀运算模块用于生成求和模块所需的各比特进位信息,二进制加法前缀运算定义为

$$\begin{cases} (G_{i:i}^0, P_{i:i}^0) = (g_i, p_i) \\ (G_{i:k}^l, P_{i:k}^l) = (G_{i:j+1}^{l-1}, P_{i:j+1}^{l-1}) \cdot (G_{j:k}^{l-1}, P_{j:k}^{l-1}) \\ = (G_{i:j+1}^{l-1} + P_{i:j+1}^{l-1} G_{j:k}^{l-1}, P_{i:j+1}^{l-1} P_{j:k}^{l-1}) \end{cases} \quad (4)$$

式中, $i = 0, 1, \dots, n - 1, 0 \leq k \leq j \leq i, l = 1, 2, \dots, m, l$ 表示第 l 级运算, l 愈小则进位传播时延愈小;“ $\cdot$ ”表示前缀运算单元。第0级的前缀运算结果即为 (g_i, p_i) , $(g_i, p_i) = (G_{ii}^0, P_{ii}^0)$ 。 $(G_{i:k}^l, P_{i:k}^l)$ 表示在第 l 级计算得到的第 k 位到第 i 位前缀运算结果,也称为组前缀运算。实际上,当 $i = 0, 1, \dots, n - 1, 0 \leq k \leq j \leq i$ 且 $v \leq j$ 时,式也成立。

$$\begin{aligned} (G_{i:k}^l, P_{i:k}^l) &= (G_{i:v}^{l-1}, P_{i:v}^{l-1}) \cdot (G_{j:k}^{l-1}, P_{j:k}^{l-1}) \\ &= (G_{i:v}^{l-1} + P_{i:v}^{l-1} G_{j:k}^{l-1}, P_{i:v}^{l-1} P_{j:k}^{l-1}) \end{aligned} \quad (5)$$

由式(4)和式(5),可以构建多种不同结构的前缀运算结构,常见的二进制加法前缀结构有 Sklansky (SK)、Brent-Kung (BK)、Kogge-Stone (KS)、Han-Carlson (HC)、ELM 等,这些前缀结构通常也被称为前缀树。

经前缀运算后,即可得到每比特的进位信息 $c_i (i = 0, 1, 2, \dots, n)$, 它表示向第 i 位的进位:

$$\begin{cases} c_0 = c_{in} \\ c_i = G_{i-1:0}^m + P_{i-1:0}^m c_{in} & i = 1, 2, \dots, n \\ c_{out} = c_n \end{cases} \quad (6)$$

求和运算模块根据前缀运算模块所得到的进位信息 c_i 和预处理模块得到的部分和信息 p_i 求得最后的 s_i :

$$s_i = p_i \oplus c_i \quad i = 0, 1, \dots, n - 1 \quad (7)$$

2.3 单位门分析模型

单位门模型是 VLSI 电路设计中常用的分析模型之一。令单位门模型中的 1 个单位门所占用的面积和所引入的时延分别为 A_g 和 τ_g , 则非门和缓冲器

的面积和时延可忽略不计;简单的2输入门,如“与”门、“或”门、“与非”门和“或非”门等,其面积和时延分别为 A_g 和 τ_g ;而对于“异或”和“同或”这类复杂的2输入门的面积和时延则为 $2A_g$ 和 $2\tau_g$. 其他组合逻辑的面积为其所使用的单位门面积之和,时延为关键路径上单位门时延之和,例如1个二选一的MUX占用的面积为 $3A_g$,而时延为 $2\tau_g$.

3 基于前缀运算和进位修正的模 $2^n - 2^k - 1$ 加法器

本文提出的模 $2^n - 2^k - 1$ 加法器结构如图2所示,该模加法器由以下几部分构成:

- (1) 数据预处理模块:完成并行前缀运算所需要的进位生成和进位传播信息对 (g_i, p_i) ;
- (2) 进位生成模块:完成 $A + B + T$ 的快速进位计算以获得 c_i^T 和 c_{out} ,其核心为一并行前缀运算模块;
- (3) 进位修正模块:根据 c_{out} 对 c_i^T 进行修正以得到完成模加法所需要的进位 c_i^{real} ;
- (4) 求和单元:同普通二进制加法器一样,根据各位的进位信息及部分和信息计算得到相应的和 s_i .

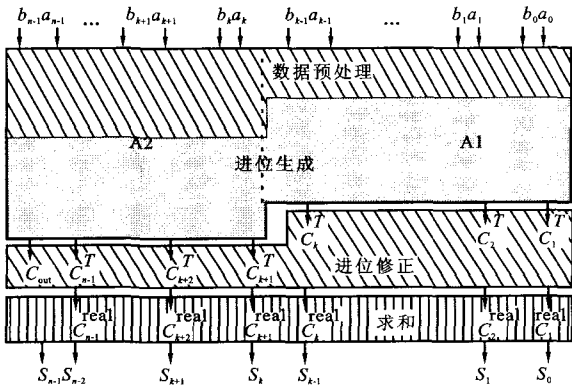


图2 模 $2^n - 2^k - 1$ 加法器结构

3.1 数据预处理

由于本文采用了对 $A + B + T$ 的进位信息 c_i^T 进位修正的方法获取模加运算所需的进位信息,因此数据预处理部仅需产生 $A + B + T$ 的进位生成和传播信息对 (g_i, p_i) ,因此数据预处理部分实际为3输入情况。

当 $p = 2^n - 2^k - 1$ 时, $T = 2^k + 1$, 其二进制表示为 $\underbrace{000 \cdots 001}_{(n-k)\text{bit}} \underbrace{00 \cdots 001}_{(k)\text{bit}}$. 可将 T 中的最低位“1”视

为默认最低进位为“1”,因此,当 $i = 0, 1, \dots, k-1$ 时:

$$\begin{cases} (g_0, p_0) = (a_0 + b_0, a_0 \oplus b_0) & i = 0 \\ (g_i, p_i) = (a_i b_i, a_i \oplus b_i) & i = 1, 2, \dots, k-1 \end{cases} \quad (8)$$

而当 $i = k, k+1, \dots, n-1$ 时,则可采用两级简化的进位保留加法器(Simple Carry Saved Adder, SCSA)将3输入变成2输入形式.即首先计算得到 a_i 加 b_i ($i = k, k+1, \dots, n-1$) 的进位保留形式 (g'_i, p'_i) :

$$(g'_i, p'_i) = (a_i b_i, a_i \oplus b_i) \quad i = k, k+1, \dots, n-1 \quad (9)$$

然后 (g'_i, p'_i) 作为第二级 SCSA 的输入,第二级 SCSA 可认为两个二进制数 $p'_{n-1} \cdots p'_{k+1} p'_k$ 和 $g'_{n-2} \cdots g'_{k+1} p'_k 1$ 进行进位保留相加, A_2 的数据预处理部分最终的输出为

$$\begin{cases} (g_k, p_k) = (g'_k, p'_k) & i = k \\ (g_i, p_i) = (p'_i g'_{i-1}, p'_i \oplus g'_{i-1}) & i = k+1, \dots, n-1 \end{cases} \quad (10)$$

至此,由式(8)~式(10)可得到前缀运算所需的必要信息.而 SCSA 的最高位输出 c_{SCSA} 则用于计算 $A + B + T$ 的最高进位 c_{out} .

$$c_{SCSA} = a_{n-1} b_{n-1} = g'_{n-1} \quad (11)$$

3.2 进位生成

进位生成模块根据数据预处理输出的 n 个 (g_i, p_i) 信息对快速生成各比特的进位 c_i^T ,它可以采用现有的任意一种 n 比特宽度的前缀算法实现,这里不再赘述。

此外,数据预处理部分 SCSA 的最高进位(如式(11)所示)并未参与前缀运算,需要将其同前缀树的最高进位 c_n^T 运算得到 $A + B + T$ 的进位输出 c_{out} . 考虑到在数据预处理部分, (g_0, p_0) 已默认最低进位输入为 $c_{in} = 1$ (式(8)),因此:

$$\begin{aligned} c_{out} &= c_{SCSA} + c_n^T \\ &= c_{SCSA} + G_{n-1:0} \\ &= c_{SCSA} + G_{n-1:l} + P_{n-1:l} G_{l-1:0} \\ &= c_{SCSA} + G_{n-1:l} + P_{n-1:l} c_l^T \end{aligned} \quad (12)$$

式中, $0 < l \leq n-1$. 根据式(4)和式(5)所示的前缀运算特性,一般来讲,总可以确定式(12)中适当

的 l 值以避免增加额外的面积和时延.

3.3 进位修正

为了便于后续的阐述,这里首先给出加法器在最低进位为“0”和“1”时各比特的进位之间的关系.

定理 1 令 $c_i (i = 0, 1, 2, \dots, n)$ 为 n 比特加法器各比特向其临近的高位输出的进位, c_{in} 为最低进位输入(即 $c_0 = c_{in}$), c_{out} 为最终的进位输出(即 $c_{out} = c_n$), 当 $c_{in} = 0$ 时, 各比特的进位为 c_i^0 ; 当 $c_{in} = 1$ 时 - 各比特的进位为 c_i^1 , 则: $c_{i+1}^0 = \overline{P_{i:0}} c_{i+1}^1 (i = 1, 1, 2, \dots, n - 1)$.

证明: 令加数 A, B 的二进制表示分别为 $a_{n-1}a_{n-2}\dots a_1a_0$ 和 $b_{n-1}b_{n-2}\dots b_1b_0$, 则 $p_i = a_i \oplus b_i$, $g_i = a_i b_i$, $P_{i:0} = p_i \dots p_1 p_0 = (a_i \oplus b_i) \dots (a_1 \oplus b_1)(a_0 \oplus b_0)$.

由并行前缀算法: $c_{i+1} = G_{i:0} + P_{i:0} c_{in}$, 则

$$\begin{cases} c_{i+1}^0 = G_{i:0} & c_{in} = 0 \\ c_{i+1}^1 = G_{i:0} + P_{i:0} & c_{in} = 1 \end{cases}$$

若 $P_{i:0} = 1$ 表明 $a_i \neq b_i$, 必然有 $g_i = a_i b_i = 0$, 则 $G_{i:0} = 0$, 因此 $c_{i+1} = c_{in}$, 即: $c_{i+1}^0 = 0, c_{i+1}^1 = 1$.

若 $P_{i:0} = 0$ 表明 c_{in} 不能被传播到 c_{i+1} , 因此 $c_{i+1} = G_{i:0}$, 与 c_{in} 无关, 即: $c_{i+1}^0 = c_{i+1}^1$.

因此: $c_{i+1}^0 = \overline{P_{i:0}} c_{i+1}^1$

证毕.

定理 1 表明了 c_i^0 可由 c_i^1 通过简单的逻辑运算得到. 对于模 $(2^n - 2^k - 1)$ 加法器, $T = 2^k + 1$, 其二进制表示为 $\underbrace{000\dots 001}_{(n-k)\text{bit}} \underbrace{00\dots 001}_{(k)\text{bit}}$. 可认为 $A + B + T$ 的运算过程被分解为两步, 首先计算 $S_A = A + B + \underbrace{000\dots 001}_{(n-k)\text{bit}}$, 然后计算 $S_B = S_A + \underbrace{000\dots 001}_{(n-k)\text{bit}} \underbrace{00\dots 000}_{(k)\text{bit}}$, 即可将 T 的二进制表示中的两个“1”视为图 2 中加法器 A1 和 A2 最低进位. 与此对应, 可通过两次修正得到 $A + B$ 的各比特进位. 第一次修正结果得到 $Y = X + \underbrace{000\dots 001}_{(n-k)\text{bit}} \underbrace{00\dots 000}_{(k)\text{bit}}$ 的进位信息, 第二次修正则可获得 $A + B$ 的进位信息, 而是否进行修正由 $A + B + T$ 的最高进位 c_{out} 决定. 为了不增加额外的时延, 设计中应进行特殊处理使得两次修正的时延不大于一次修正的时延.

(1) 第一次修正

由于 T 的二进制表示为 $\underbrace{000\dots 001}_{(n-k)\text{bit}} \underbrace{00\dots 001}_{(k)\text{bit}}$, 可

将 c_i^T 视为 $(A + B + T - 1) + c_{in} (c_{in} = 1)$ 的各比特进位输出, 因此可利用定理 1 对其进行修正得到 $A + B + T - 1$ 的进位输出 $c_i^{T-1} (i = 0, 1, 2, \dots, n - 2)$, 即:

$$c_{i+1}^{T-1} = \overline{P_{i:0}} c_i^T \quad (13)$$

由模加法器的基本理论, 对 $A + B + T$ 的进位进行修正的前提是 $c_{out} = 0$. 这里可以等效为一个二选一逻辑, c_{out} 作为选择器的控制端, 而 c_i^T 和 c_i^{T-1} 为数据输入端, 输出即为第一次修正的结果, 令其为 $c_{i+1}^{c_1} (i = 0, 1, 2, \dots, n - 2)$:

$$c_{i+1}^{c_1} = \overline{c_{out}} c_{i+1}^{T-1} + c_{out} c_{i+1}^T = c_{i+1}^T (c_{out} + \overline{P_{i:0}}) \quad (14)$$

(2) 第二次修正

令第二次修正后得到的进位信息为 c_i^{real} . 采用与第一次修正相同的处理方法, 若 $c_{out} = 0$, 则 c_i^{real} 为 $A + B + T - 1 - 2^k$ (即 $A + B$) 的进位信息; 否则 c_i^{real} 为 $A + B + T$ 的进位信息, 即 c_i^{real} 为模加法器所需要的实际进位信息. 在第二次修正中同样利用定理 1 的结论, 但其主要目的是消除 T_{A2} 中的最低位“1”的影响.

由于 $T = 2^k + 1$, 当 $i = 1, 2, \dots, k$ 时, T_{A2} 中的“1”不再影响 $c_i^{c_1}$, 因此当 $i = 1, 2, \dots, k$ 时: $c_i^{\text{real}} = c_i^{c_1}$; 当 $i = k + 1, \dots, n - 1$ 时, 应对 $c_i^{c_1}$ 修正以消除 T_{A2} 所带来的影响. 由数据预处理部分的分析可知, 此时图 2 中加法器 A2 为带进位输入的两个加数 $p'_{n-1}\dots p'_{k+1}p'_k$ 和 $g'_{n-2}\dots g'_{k+1}g'_k 1$ 相加运算, 而 A2 的进位输入为图 2 中加法器 A1 经修正后的进位输出 $c_{i+1}^{c_1}$. 注意到 $g'_{n-2}\dots g'_{k+1}g'_k 1$ 的最低位为“1”, 因此可认为 A2 完成了最低进位输入为“1”, 加数为 $p'_{n-1}\dots p'_{k+1}p'_k$ 和 $g'_{n-2}\dots g'_{k+1}g'_k c_{i+1}^{c_1}$ 的加法运算. 即可以再次利用定理 1 对加法器的进位信息 $c_i^{c_1} (i = k + 1, \dots, n - 1)$ 修正以得到 $A + B$ 的进位, 再结合 c_{out} 则可得到模加法所需的各比特最终进位信息 c_i^{real} .

注意, 由于第二次修正是在考虑加法器 A2 的最低进位为常数“1”的情况下进行的, 因此修正过程中将使用到的进位传播信息均应由 $p'_{n-1}\dots p'_{k+1}p'_k$ 和 $g'_{n-2}\dots g'_{k+1}g'_k c_{i+1}^{c_1}$ 计算得到. 但两种加法运算中的加数仅最低位不同, 加数 $g'_{n-2}\dots g'_{k+1}g'_k 1$ 的最低位为“1”, 加数

$g'_{n-2} \cdots g'_{k+1} g'_{k-1} c_k^c$ 的最低位为 c_k^c , 即两种情况下的进位传播信息仅最低位不同. 则

$$\begin{cases} p''_k = p'_k \oplus c_{k-1}^c = c_{k-1}^c \oplus \overline{p_k} & i = k \\ p''_i = p_i & i = k+1, \dots, n-1 \end{cases} \quad (15)$$

则 $p''_{i:k} = P_{i:k+1} p''_k$, 当 $i = k+1, k+2, \dots, n-2$ 时, 由定理 1 和式(14), 第二次修正后的进位信息为

$$\begin{aligned} c_{i+1}^{\text{real}} &= (\overline{P_{i:k}} + c_{\text{out}}) c_{i+1}^{c_1} \\ &= (\overline{P_{i:k+1} p''_k} + c_{\text{out}}) c_{i+1}^{c_1} \\ &= c_{i+1}^T (c_{\text{out}} + \overline{P_{i:0}}) (\overline{P_{i:k+1} p''_k} + c_{\text{out}}) \end{aligned} \quad (16)$$

将式(14)、式(15)代入式(16), 则

$$c_{i+1}^{\text{real}} = c_{i+1}^T (c_{\text{out}} + \overline{P_{i:k+1}} + \overline{P_{k-1:0}} (p_k \oplus c_k^T)) \quad (17)$$

当 $i = k$ 时, $P''_{i:k} = P''_{k:k} = p''_k$, 同理可得

$$c_{i+1}^{\text{real}} = c_{i+1}^T (c_{\text{out}} + \overline{P_{k-1:0}} (p_k \oplus c_k^T)) \quad (18)$$

令 $z_1 = \overline{P_{k-1:0}} (p_k \oplus c_k^T)$,

$z_2 = \overline{P_{i:k+1}} + \overline{P_{k-1:0}} (p_k \oplus c_k^T)$,

由式(14)、式(17)和式(18), 可得到由 c_{out} 控制的修正后模加运算所需的各比特进位信息:

$$c_{i+1}^{\text{real}} = \begin{cases} c_{i+1}^T (c_{\text{out}} + \overline{P_{i:0}}) & i = 0, 1, \dots, k-1 \\ c_{i+1}^T (c_{\text{out}} + \overline{z_1}) & i = k \\ c_{i+1}^T (c_{\text{out}} + \overline{z_2}) & i = k+1, \dots, n-2 \end{cases} \quad (19)$$

当 $i = 1, 2, \dots, k$ 时, c_i^{real} 的计算时延为 $2\tau_g$, 刚好与一个前缀运算单元时延相同. 由图 2 可知, 在绝大多数的前缀结构下, 本文所提出的模加法器特殊的数据预处理使得在前缀运算中 c_k^T 较 c_i^T ($i = k+1, \dots, n-1$) 至少会提前 1 级前缀运算单元出现 (即 $2\tau_g$), 不会出现 c_k^T 与 c_i^T 同时出现的情况. 利用该时延差, 经特殊设计则可保证在 $i = k+1, \dots, n-1$ 时 c_i^{real} 的计算时延不超过 $2\tau_g$, 即在进位修正中不引入额外时延. 这里采用的办法是将 c_k^T 作为选择信号, 预先计算得到 z_1 和 z_2 中的进位传播信息, 因此最多引入一个二选一电路带来的 $2\tau_g$ 单位门时延, 即与 $i = 1, 2, \dots, k$ 时的 c_i^{real} 计算时延相同, 不会引入额外时延. 若在前缀运算中 c_k^T 较 c_i^T ($i = k+1, \dots, n-1$) 提前 2 级或 2 级以上前缀运算单元的处理时延, 由于在产生 c_k^T 后 z_1 和 z_2 最多需经 1 个“异或”、1 个“与”门和 1 个“或”门的时延 (即 $4\tau_g$) 后计算得

到, 此时可不进行预处理, 仍不会引入额外时延.

3.4 求和运算

求和运算模块同一般的基于前缀运算的二进制加法器基本相同, 但 c_i^{real} 是在考虑了 c_{out} 控制下的修正结果, 即 $c_{\text{out}} = 0$ 时 c_i^{real} 为 $A + B$ 的进位信息, 反之则为 $A + B + T$ 的进位信息, 因此在求和时也应使用对应的 $A + B$ 或 $A + B + T$ 的部分和信息. 令 p_i^0 和 p_i^1 ($i = 0, 1, \dots, n-1$) 分别表示 $A + B$ 和 $A + B + T$ 的部分和信息, 在数据预处理模块计算得到的是 p_i^1 , 即 $p_i^1 = p_i$. 同时由数据预处理单元可知, 仅当 $i = 0, k$ 时 $p_i^0 \neq p_i^1$. 由以上分析可得

$$\begin{cases} p_0^0 = \overline{p_0}, p_0^1 = p_0 & i = 0 \\ p_k^0 = \overline{p_k}, p_k^1 = p_k & i = k \\ p_i^0 = p_i^1 = p_i & i = 1, \dots, k-1, \\ & k+1, \dots, n-1 \end{cases} \quad (20)$$

因此:

$$s_0 = \overline{c_{\text{out}}} p_0^0 + c_{\text{out}} p_0^1 = \overline{c_{\text{out}}} p_0 + c_{\text{out}} p_0 = c_{\text{out}} \oplus p_0 \quad (21)$$

$$s_k = c_k^{\text{real}} \oplus (\overline{c_{\text{out}}} p_k^0 + c_{\text{out}} p_k^1) = c_k^{\text{real}} \oplus c_{\text{out}} \oplus p_k \quad (22)$$

$$s_i = c_i^{\text{real}} \oplus p_i \quad (23)$$

在式(22)中, $c_{\text{out}} \oplus \overline{p_k}$ 可与 c_k^{real} 同时计算得到, 因此不会引入额外时延.

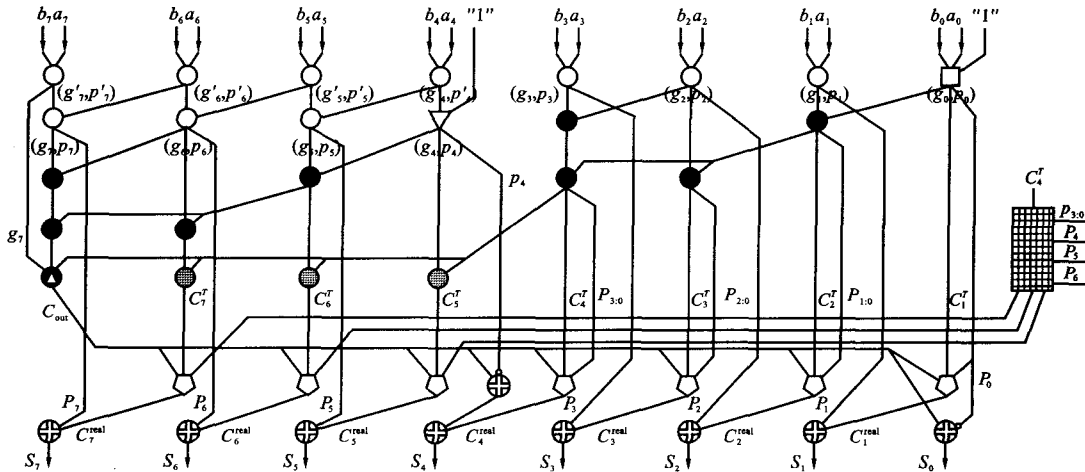
3.5 设计实例

基于本文算法的模 $2^8 - 2^4 - 1$ 加法器 VLSI 实现结构如图 3(a) 所示, 图 3(b) 为各处理模块功能. 其中“○”、“▽”和“□”用于前缀运算的数据预处理, 分别对应式(8)、式(9)和式(10)的运算. “●”表示前缀运算单元, 本例中采用了 Sklansky 前缀树, 共 11 个前缀处理单元, 它对应式(4)的运算. 前缀树的树枝末端的前缀运算单元的所生成的进位信息即为进位修正所需的进位信息 c_i^T . “⊙”完成式(12)的计算, 主要目的是避免在前缀运算结束后再进行 c_{SCSA} 和 c_n^T 的“或”运算, 消除额外的一级单位门时延. 对式(12)中的 l 适当选择可使“⊙”仅引入 1 个“或”门时延, 例如本实例中选择 $l = 4$. “◇”用于进位修正, 本例中共 7 个进位修正单元, 它们完成式(19)的运算. “■”用于计算式(19)中的 $P_{i:0}$, z_1 和 z_2 , 使得它们至少同进位信息 c_i^T 同时出现, 不引入额外的时延. 在最坏情况下, 需要根据 p_i ($i = 0, 1, \dots, n-1$)

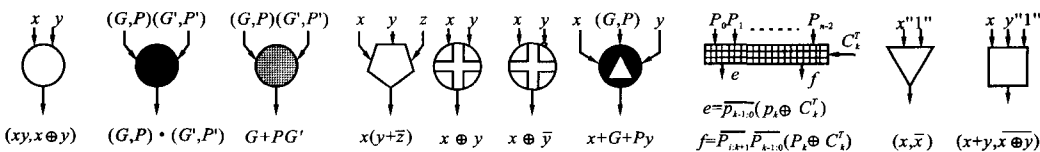
2) ——计算 z_1 和 z_2 所需的组进位传播信息,实际实现中若前缀结构中已具有这些信息,则可节约相应的处理单元面积.例如本实例中,前缀运算中已经计算得到了 $P_{i:0}(i = 0, 1, \cdots, 3)$. “ $\oplus$ ”用于最后的求和预算,它实际为二进制“异或”操作,完成式(21)、式(22)和式(23)的运算.其中“ $\oplus$ ”表示将对

应的输入“取反”,在进位修正的同时完成式(22)中的 $c_{out} \oplus p_k$ 运算,从而使得最后的求和运算仅为简单的异或运算,不会引入额外的时延.

本实例根据本文所提出算法针对模 $2^8 - 2^4 - 1$ 加法器给出了详细设计和说明,前缀运算选用了 Sklansky 树.



(a) 模 $2^8 - 2^4 - 1$ 加法器 VLSI 实现结构



(b) 各处理模块功能

图 3 模 $2^8 - 2^4 - 1$ 加法器

4 性能分析与比较

表 1 给出了基于单位门模型的面积和时延性能比较,选择这几种模加法器进行比较的原因是它们的模加运算类型或采用的算法基本类似.可表明本文所设计模加法器较普通二进制加法器之间的差别.

图 4(a)给出了 Patel 模 $2^n - 2^{n-2} - 1$ 加法器、ELMMA 和当 $k = n - 2$ 时本文所提出算法同基于

Sklansky 前缀树的普通二进制加法器的“面积 $\times$ 时延”对比结果,图 4(b)则给出了在实现相同模加法(模 $2^n - 2^{n-2} - 1$)时,本文算法较 Patel 和 ELMMA 所节约的“面积 $\times$ 时延”百分比,可见随着 n 的增大,本文所提出的算法较 Patel 模加法器的节约了 30% 左右,而较 ELMMA 则节约了 45% 左右;当大于 3 时,较 Patel 等人的模 $2^n - 2^{n-2} - 1$ 加法器节约的“面积 $\times$ 时延”百分比至少在 10% 以上,较 ELMMA 则至少在 20% 以上.

表 1 基于单位门模型的性能对比

	面积	时延
Patel, et al	$1.5n \log_2 n + 1.5n \log_2 (n - 1) + 2^{\log_2 (n - 1) - 1} + 11n - 2$	$2 \log_2 n + 5$
ELMMA	$(4n - 7) \log_2 (n - 1) + 11n + 4$	$2 \log_2 n + 7$
SKPPA($c_{in} = 0$)	$1.5n \log_2 n + 5n$	$2 \log_2 n + 3$
本文	$1.5n \log_2 n + 13n - 5k - 8$	$2 \log_2 n + 7$

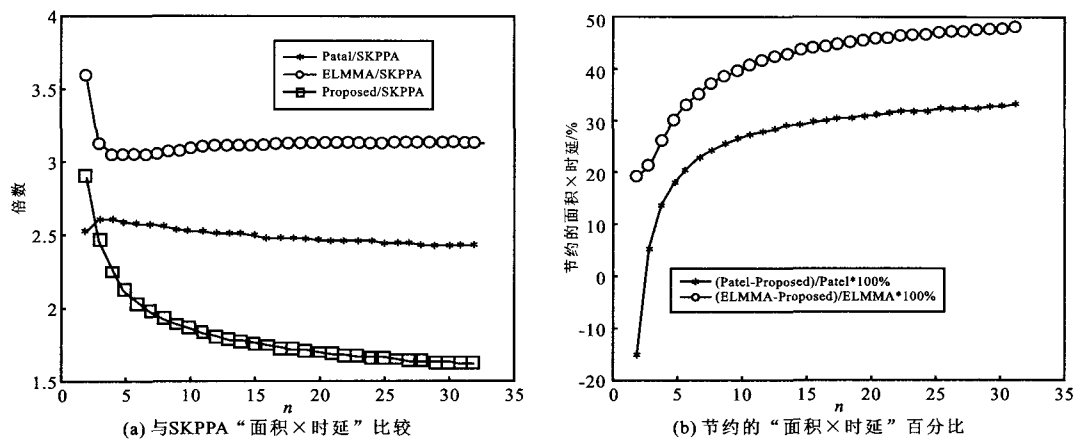


图4 基于单位门模型的“面积×时延”特性比较

5 结束语

本文提出了一种优化的基于进位修正和并行前缀算法的具有形式的模 $2^n - 2^k - 1$ 加法器通用实现算法及其 VLSI 实现结构,它由数据预处理、进位计算、进位修正和求和运算这三个模块构成.消除了用于进位计算的重复单元,该结构具有一个完整的前缀运算单元,在实现中可根据具体情况选择高效的并行前缀结构找到速度与面积的最佳平衡点.此外,具有 $2^n - 2^k - 1$ 形式的余数基易于构建大动态范围、优良通道间复杂度平衡性的多通道 RNS,而本文提出的算法涵盖了基于 $2^n - 2^k - 1$ 形式的余数基的各通道的基本单元的实现,其实现结构统一,简化了设计,从而为构建这一类新的余数基打下了良好的基础.有关时延、面积的分析 and 对比充分说明了本文所提出的算法具有优良的“面积×时延”特性,从而可实现高速低功耗的模加法单元.

参考文献:

- [1] Ma S, Hu J H, Zhang L, et al. An efficient RNS parity checker for moduli set $\{2^n - 1, 2^n + 1, 2^{2n} + 1\}$ and its ap-

plications[J]. Sci China Ser F - Inf Sci, 2008, 51(10): 1 - 10.

- [2] Liu Y, Lai E M K. Design and implementation of an RNS - based 2 - D DWT processor[J]. IEEE Trans on Consumer Electronics, 2004, 50(1): 376 - 385.
- [3] Hiasat A A. High - speed and reduced - area modular adder structures for RNS[J]. IEEE Trans on Computers, 2002, 51(1): 84 - 89.
- [4] Patel R A, Benaissa M, Powel N, et al. ELMMA: a new low power high - speed adder for RNS[C]//Proc. IEEE Workshop on Signal Processing Systems. Austin: IEEE, 2004: 95 - 100.
- [5] Patel R A, Benaissa M, Powell N, et al. Novel power - delay - area - efficient approach to generic modular addition [J]. IEEE Trans on Circuits and Systems I: Regular Papers, 2007, 54(6): 1279 - 1292.
- [6] Patel R A, Benaissa M, Boussakta S. Fast modulo $2^n - (2^{n-2} + 1)$ addition: a new class of adder for RNS[J]. IEEE Trans on Computers, 2007, 56(4): 572 - 576.

作者简介:

马 上 男, (1978 -), 博士研究生. 研究方向为 VLSI 设计和无线通信.