上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

研究生课程实验报告

GRADUATE COURSE PROJECT REPORT

课　　程：高等数字电路设计

报告题目：Project2 Approximate Adder Design

学生姓名：赵拯基

学生学号：121039910125

任课教师：何卫锋 孙亚男

学院(系)：

开课学期：

目录

一、 项目目的

设计一款 32-bit 无符号的低功耗近似加法器，要求错误率上限。与 32-bit 无符号精确加法器对比，优化关键路径延迟和功耗。

二、 项目要求

2.1 项目设计要求

2.1.1 项目设计目标 1

设计一个 32-bit 无符号加法器（作为 baseline）

- 加法器结构可选择行波进位（RCA）、超前进位（CLA）等，也可根据文献选择其它结构；
- 搭建 SPICE 网表仿真测试，得到关键路径延迟和功耗等信息；
- 功耗的测量自行选取测试向量。

2.1.2 项目设计目标 2

设计一个 32-bit 无符号的低功耗近似计算加法器

- 自行查阅文献，选择合适（错误率不高于 10%）的近似加法器结构，搭建 SPICE 网表仿真测试；
- 优化选取结构的关键路径延迟和功耗；
- 与 baseline 进行功耗、延迟、漏电等各方面的比较，对近似加法器的优缺点进行讨论，保持相同的测试向量。

2.2 参数测量要求

- **关键路径延时**：输入信号与输出信号 50%翻转点间隔，指明关键路径；
- **动态功耗**：基于自行选取测试向量测量平均动态功耗；

- **泄漏功耗**：基于自行选取测试向量测量平均静态功耗；
- **面积估计**：$W \cdot L$或者粗略统计晶体管数量。

2.3 工艺以及仿真环境要求

- 工艺：7nm PTM LP（采用 7nm FinFET 工艺，栅极长度采用默认值 lg 为 11nm，宽度为最小宽度整数倍数，标准供电电压为 0.7V）
- 温度：25℃
- 输出负载：8 倍最小尺寸 inverter 门电容负载
- 输入信号上升下降时间：15ps

三、 前期准备

3.1 测试向量

通过 stimulus.vect 文件定义测试向量，在原有的基础上进行修改。

增加新的向量名称。cin 代表进位输入，输入类型，位宽 1bit；SUM 代表 32-bit 加法器的计算结果，包含进位输出，输出类型，位宽 33bits。

增加输出延时定义。odelay 表示输出延时，1ns。

增加阈值电压定义。vth 表示阈值电压，0.35v。当输出电压高于阈值电压 vth 时，视为逻辑 1；否则视为逻辑 0。

以上，增加输出采样设置，进行瞬态仿真时判断实际输出与测试向量中规定的输出是否一致。通过文件.err0 展示所有不一致的输出，通过此判断加法器逻辑的正确性以及评估错误率。详细内容参考附录的 stimulus.vect。

3.1.1 动态功耗测试向量

器件端口的逻辑状态发生不断的翻转，器件内部除泄漏电流外，还存在工作电流，且为了驱动容性负载还需要不断充放电。这种由于逻辑状态翻转而产生的功耗，称为动态功耗。

为了测量动态功耗，测试向量需要尽可能保持随机。通过 python 生成测试

向量，包括 2 个 32bits 加数 A、B 以及 1bit 进位输入 cin，并自动计算对应的 SUM。详细内容参考附录的 test_pattern.py，下图展示了生成的 10 个测试向量。

```
(base) PS D:\ADICD_lab> python -u "d:\ADICD_lab\proj_2\test_pattern.py"
71065160 C756EDE9 1 1385D3F4A
E8367C82 5D20C55C 1 1455741DF
7043295D 4DF0F4BE 0 0BE341E1B
81ACD240 485C9216 1 0CA096457
C61AC1A4 BE652A84 0 1847FEC28
4432E75A 19D598C3 0 05E08801D
E0B0166F 5A0A87CD 1 13ABA9E3D
FD6A89F2 0A11AAD9 0 1077C34CB
4001F199 41145E05 0 081164F9E
55091C23 0C69055E 1 061722182
```

图 1 生成测试向量

实际测试时，生成 25 个测试向量，瞬态仿真 26ns，测量供电的平均功耗。

3.1.2 静态功耗测试向量

当器件所有端口逻辑状态稳定而不发生变化，则端口无需充/放电，器件内部仅存在泄露电流，此时的功耗就是静态功耗。

输入端口即 2 个 32bits 加数 A、B 以及 1bit 进位输入 cin 保持全 0 或者全 1，分别测试静态功耗。

3.2 common elements

基本电路网表描述，位于 common_elements.net 文件，包括了最小尺寸的反相器、4 倍最小尺寸的反向器、8 倍最小尺寸的反相器、传输门、异或门（2 输入）、或门（2 输入、3 输入、4 输入、5 输入）、与门（2 输入、3 输入、4 输入、5 输入）。所有的逻辑门均采用相同大小的 PMOS 和 NMOS 即 nfin=1，不进行 resizing。

3.2 special elements

经过 resizing 或者针对结构进行优化后的逻辑门，位于 special_elements.net 文件，包括了基于传输门的异或门（2 输入）、resizing 后的逻辑门（2 输入或/与门、3 输入或/与门、4 输入或/与门）、门控逻辑门（2 输入门控或门、5 输入门控与门）。

四、 设计流程

4.1 精确加法器

4.1.1 具体实现方法

单纯的 32-bit 行波加法器（RCA）会在最终的进位输出产生极大输出延时；单纯的 32-bit 超前进位加法器（CLA）会产生极大的扇入扇出，造成电路性能的急剧下降。因此采用 RCA、CLA 共同构建 32-bit 加法器。构建 4-bit CLA，8 个 RCA 级联构建 32-bit 加法器，结构如下图所示。

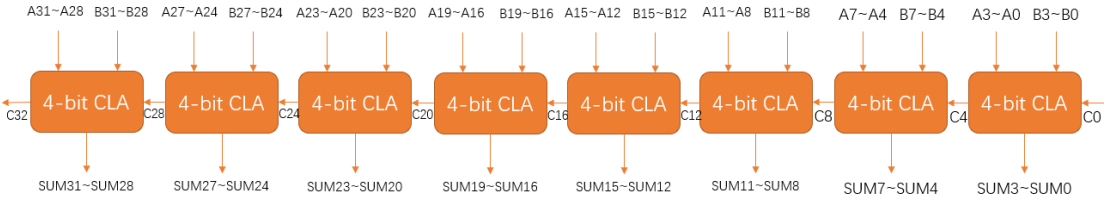

图 2 精确加法器结构

4-bit CLA 的进位链与和公式分别计算如图下。

$$\begin{cases} c_1 = G_0 + c_0 P_0 \\ c_2 = G_1 + c_1 P_1 = G_1 + (G_0 + c_0 P_0) P_1 = G_1 + G_0 P_1 + c_0 P_1 P_0 \\ c_3 = G_2 + c_2 P_2 = G_2 + (G_1 + G_0 P_1 + c_0 P_1 P_0) P_2 = G_2 + G_1 P_2 + G_0 P_2 P_1 + c_0 P_2 P_1 P_0 \\ c_4 = G_3 + c_3 P_3 = G_3 + (G_2 + G_1 P_2 + G_0 P_2 P_1 + c_0 P_2 P_1 P_0) P_3 = G_3 + G_2 P_3 + G_1 P_3 P_2 + G_0 P_3 P_2 P_1 + c_0 P_3 P_2 P_1 P_0 \end{cases}$$

图 3 CLA 进位链计算公式

$$\begin{cases} s_0 = P_0 \oplus c_0 \\ s_1 = P_1 \oplus c_1 \\ s_2 = P_2 \oplus c_2 \\ s_3 = P_3 \oplus c_3 \end{cases}$$

图 4 CLA 和计算公式

单个 4-bit CLA 的内部结构如下图所示，首先计算所有的$P_i = a_i \oplus b_i$和$G_i = a_i b_i$，依据所有的P_i和G_i计算和输出和进位输出。

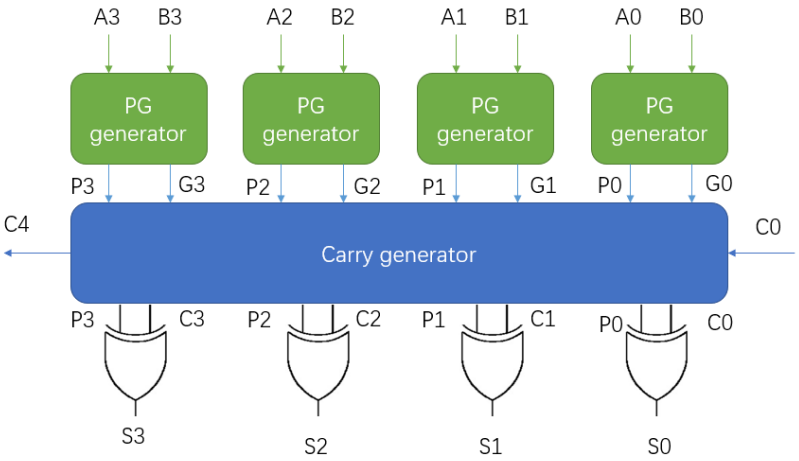

图 5 单个 4-bit CLA 结构

4.1.2 关键路径分析

对于单个 4-bit CLA 来说，从输入所有的 A、B 和 C0 到 CLA 的输出进位 C4 只需要 3 级门电路延迟，忽略不同多输入逻辑门的电路延迟差异，C1、C2、C3 同时生成。同时由于 S3=P3⊕C3，所以 4-bit CLA 关键路径为 4 级门延迟。

如果采用 4-bit CLA 级联构成 32-bit 加法器，则关键需要（3+7×2+1）=18 级门电路延迟。考虑到最终的进位输出，需要经过一个 5 输入的或门，产生的电路延迟可能大于 4 输入的或门加上一个 2 输入的异或门（搭建该加法器使用 common_elements.net 中的逻辑门）。

4.1.3 参数测量

RCA-CLA 精确加法器			
静态功耗	仿真时间 26ns	全 0 输入	2.911e-09W
		全 1 输入	5.281e-09W
平均静态功耗			**4.096e-09W**
动态功耗	仿真时间 26ns	第一组 25 个随机测试向量	1.019e-05W
		第二组 25 个随机测试向量	1.119e-05W
		第三组 25 个随机测试向量	1.117e-05W
平均动态功耗			**1.085e-05W**

关键路径	A0 和 B0 产生最终进位输出	2.592e-10s
	A0 和 B0 对 SUM31 产生影响	**2.648e-10s**
面积估算（统计 nfin 的数量）		220×8=1760

表 1 精确加法器参数测量结果

4.2 近似加法器

4.2.1 相关近似加法器设计方案以及选择

按照项目要求，首先满足错误率小于 10%。满足错误率要求的情况下尽可能低功耗设计，最终优化选取的电路结构降低关键路径延时。我们关注地是错误率（error rate ，ER）而不是错误距离（error distance ，ED），3 组测试向量总共 75 个，按照错误率 10%，总共至多可存在 8 个计算错误，不考虑单个错误计算结果与理论计算值之间的差异。

图 6 设计方案选择流程

4.2.1.1 低位或门加法器 LOA (Low-Part-OR Adder, LOA) [1]

这种加法器利用或门来计算低位和，得到的是近似值。同时利用精确加法器来计算高位和。为提高计算精度，低位最高两位通过与门产生进位信号并传递给高位精确部分。由于这种方法低位计算结果 为近似值，得到的结果错误率很高。并且低位的位宽越大，错误率越高。假设两个输入信号 IN1 和 IN2 位宽均为 n，高位位宽 h，低位位宽 l，输出为 OUT，那么 LOA 电路结构如下图所示。

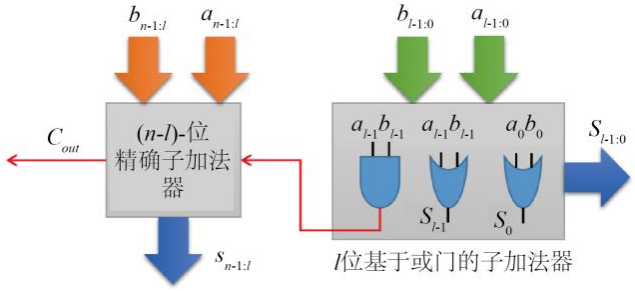

图 7 LOA 电路结构

4.2.1.2 容错加法器(Error-Tolerant Adder, ETA)

这种加法器同样是高位精确计算，低位近似处理。最初的 ETA 是利用修改后的逻辑异或门来计算低位和，用精确加法器计算高位和。这种算法同样错误率较高。后来提出改进方法，利用分块的思想，将整个电路结构分成若干个子加法器模块，这样可以将整条进位传输路径截断成较短路径，减少电路的延时与动态功耗。改进的 ETAII 由进位发生器、加数和发生器组成，低位的进位发生器产生的进位信号传递到相邻高位的加数和发生器。ETAII 的结构原理图如下图所示。

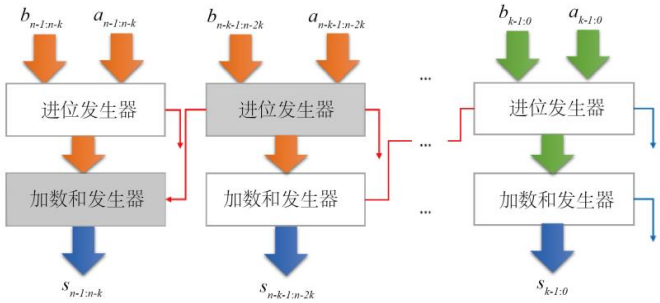

图 8 ETAII 电路结构图

由上图可看出，ETAII 电路内部存在进位预测，比单纯的分模块计算加数和更加准确，但同时电路也比较复杂，并且由于进位路径较长，延迟较大。它的延迟 O = log(2k)，其中 k 为每个子加法器的位宽。

4.2.1.3 精度可配置加法器(Accuracy Configurable Adder, ACA)[2]

这种加法器可以通过改变电路结构来配置精度，因此可以实现精度与性能和

功耗之间的平衡。以 16 位加法为例，ACA 的结构示意图如下所示。

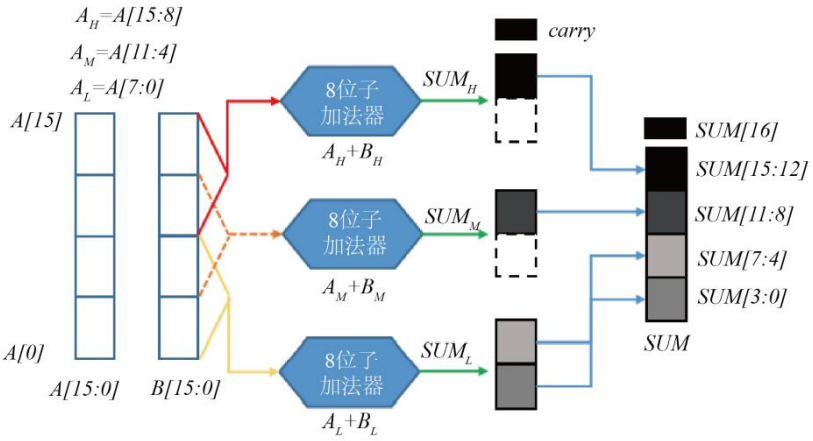

图 9 ACA 电路结构

根据上图，在 16 位 ACA 中，由三个 8 位子加法器模块分别产生部分求和的结果。子模块内部高四位计算结果，低四位计算进位输出。引入中间加法器(AM + BM)以提高精度，若没有中间加法器，当第八位进位为 1 时计算会出现错误。对于随机输入模式，错误率为 50.1%，而引入中间加法器后，随机输入模式的计算错误率降低到 5.5%。将 ACA 扩展至更一般的情况，当输入数据位宽为 N，子模块位宽为 2k 时，ACA 的实现框图如下图所示。

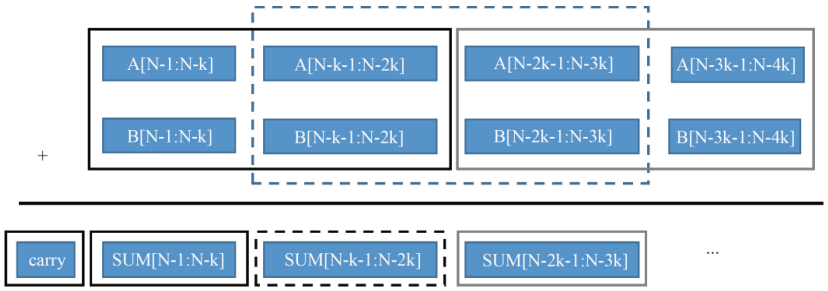

图 10 ACA 结构图

4.2.1.4 进位跳跃加法器(Carry-Skip Adder, CSA)[3]

CSA 也利用了模块化的思想，通过子模块来产生进位与部分和。此外，CSA 利用了一种进位跳跃机制，其原理图如下所示。根据原理图，CSA 的进位跳跃机制具体为：在计算第 i + 1 个子模块的进位输入信号时，如果第 i 个子模块的进位传递信号为 1，那么将第 i − 1 个子模块的进位输出作为该进位输入的结果，否则将第 i 个子模块的进位输出作为该进位输入的结果。通过配置额外的结构降低

错误率，减少延时，但电路面积与功耗也因此而增加。它的关键路径延迟 O = log(2k)。

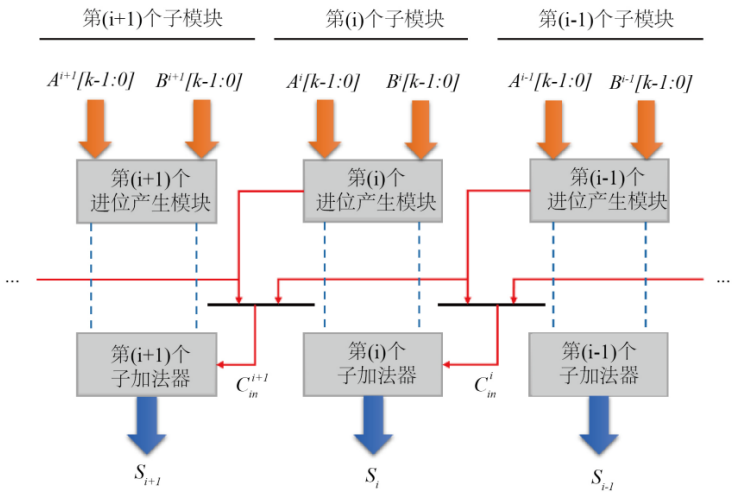

图 11 CSA 原理图

4.2.1.5 可配置超前进位加法器(Reconfigurable Approximate CLA, RAP-CLA)[4]

该加法器将进位链的输出分为两个部分，单个 RAP-CLA 的进位输出结构如下图所示。与精确计算进位输出相比，计算不精确的进位输出速度更快且功耗更低。基于这样的分割，实现精确和近似的操作的基础上构建可重构近似加法器实。因此，与传统的 CLA 相比，电路中增加了一个多路复用器，用于选择进位输出，控制信号为是否选择近似操作。为了降低功耗，当进位输出处于近似操作模式时，这部分使用 PMOS 进行电源门控。

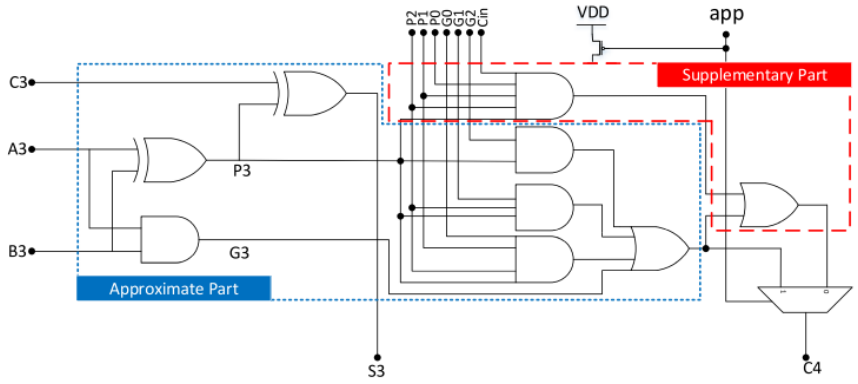

图 12 RAP-CLA 的进位输出电路

4.2.1.6 精度可配置的 radix-4 加法器(Accuracy-Configurable Radix-4 Adder, ACRA)[5]

其基础结构为传统的 2-bit RD4A，同时进行两位的运算操作，减少了传统加法器的传播延迟和功耗，如下图所示。

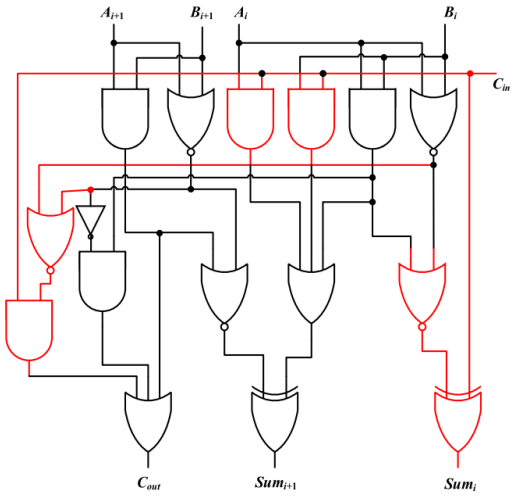

图 13 传统 2 位 Radix-4 加法器的架构

在此基础上进行修改，构建 ACRA，如下图所示。同样通过 PMOS 进行电源门控实现电源门控，近似计算时关闭部分逻辑电路减少功耗。下图蓝色部分的电路是为了降低近似输出时的错误距离。

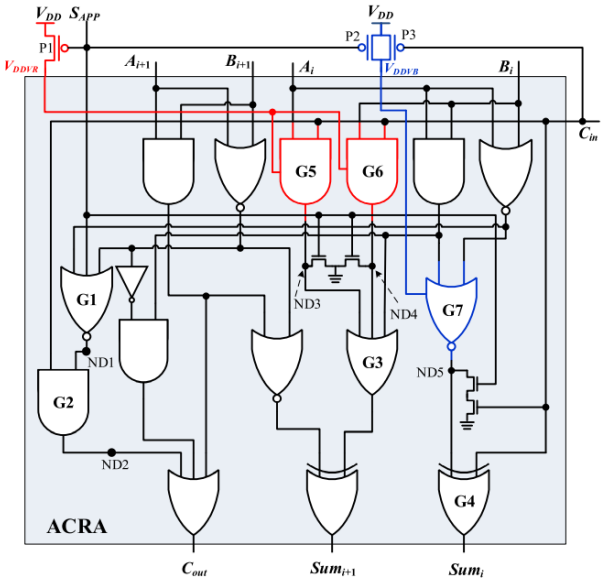

图 14 ACRA 电路结构

4.2.1.7 方案选择

LOA 由于完全利用逻辑或门进行低位运算，面积最小，功耗也最小，这跟它模块最少是相对应的。但由于其结构简单，未考虑精度的问题，它的错误率最高，即精度是最低的，首先被排除，但它的平均相对误差距离却很小。

剩下的近似加法器设计方案更复杂，通过更加复杂的电路结构降低错误率。按照相关参考文献提供的数据，ETAII、CSA、RAP-CLA、ACRA、ACA 均可以达到错误率要求。ACA 为了达到错误率要求，需要更高的重叠计算单元，面积开销太大；ACRA 单个单元仅支持 2-bit 加法，构建 32-bit 加法器需要 16 个级联，面积开销同样太大。

在 ETAII、CSA、RAP-CLA 中选择合适的方案。ETAII、CSA 只能实现固定精度误差的计算；RAP-CLA 可自由配置精度。

最终选择 RAP-CLA，与原有的 RCA-CLA 精确加法器采用相同的级联方式。

4.2.2 关键路径分析

构建的 RCA-CLA 近似加法器结构与原有的 RCA-CLA 精确加法器相似，结构如下图所示。4-bit 精确 CLA 与 4-bit 近似 CLA 级联，低比特位采用 5 个 4-bit 近似 CLA，高比特位采用 3 个 4-bit 精确 CLA，以达到错误率上限要求。

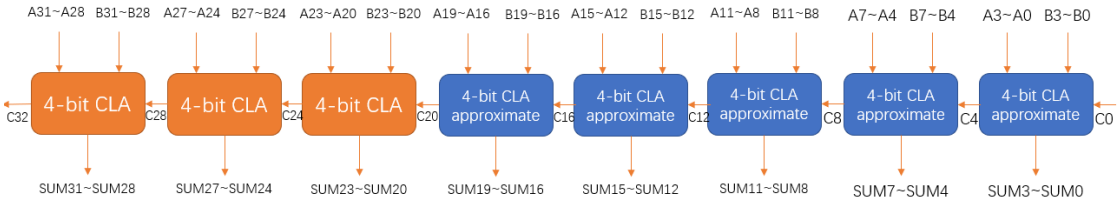

图 15 近似加法器结构

如前文所述，高比特位的精确 CLA 关键路径为 A20 和 B20 产生的进位传播，对最终的进位输出或者 SUM31 产生影响。经过 resizing 后，每一个 4-bit 精确 CLA 的进位输出所经过的 5 输入与门被拆分为 2 级电路，因此最终的进位输出或者 SUM31 输出将经过相同数量的逻辑门（逻辑门的种类不同，实际哪一个是关键路径需要经过测量）。

$$c_4 = G_3 + G_2P_3 + G_1P_3P_2 + G_0P_3P_2P_1 + c_0P_3P_2P_1P_0$$

近似计算时不考虑

图 16 近似加法器进位输出计算

考虑 4-bit 近似 CLA 的进位输出，进位输出公式如上图所示。通过公式可以观察到，上一级产生的进位输出并不会通过本级传播到下一级近似加法器。换句话说，近似加法器产生的进位输出仅能传播一级，影响下一级。

整体考虑精确加法器和近似加法器，关键路径为 A16 和 B16 产生的进位传播，对最终的进位输出或者 SUM31 产生影响。

4.2.3 参数测量

RCA-CLA 近似加法器			
静态功耗	仿真时间 26ns	全 0 输入	3.123e-09W
		全 1 输入	4.920e-09W
平均静态功耗			**4.022e-09W**
动态功耗	仿真时间 26ns	第一组 25 个随机测试向量（**错 2**）	8.462e-06W
		第二组 25 个随机测试向量（**错 1**）	9.447e-06W
		第三组 25 个随机测试向量（**错 4**）	8.903e-06W
平均动态功耗			**8.937-06W**
关键路径	A16 和 B16 产生最终进位输出		**1.727e-10s**
	A16 和 B16 对 SUM31 产生影响		1.706e-10s
面积估算（统计 nfin 的数量）			324×8=2592

表 2 近似加法器参数测量结果

五、 设计总结

经过 resizing 后，相同逻辑门的面积必定会变大。又加入个两路数据选择器以及门控电源电路，单个 RCA-CLA（324）的面积大小相较于原始的 CLA（220），面积增加了 47%。resizing 后，逻辑门的延时得到了优化，并且关键路径变短了，所以最终的关键路径延时降低了 35%。虽然面积（MOS 的尺寸）变大了，但由

于近似计算关闭了不需要的额外电路，动态功耗和静态功耗均有下降，其中动态功耗下降了 18%，静态功耗略微下降。

六、 参考文献

[1] Yao, T., Gao, D. and Fan, X. (2012) Three-Operand Floating-Point Adder. IEEE International Conference on Computer & Information Technology, Chengdu, 27-29 October 2012, 192-19

[2] Shafique, M., Ahmad, W., Hafiz, R., et al. (2015) A Low Latency Generic Accuracy Configurable Adder. 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, 7-11 June 2015, 1-6.

[3] Chirca, K., Schulte, M., Glossner, J., et al. (2004) A Static Low-Power, High-Performance 32-bit Carry Skip Adder. Euromicro Symposium on Digital System Design, Rennes, 31 August-3 September 2004, 615-619.

[4] O. Akbari, M. Kamal, A. Afzali-Kusha and M. Pedram, "RAP-CLA: A Reconfigurable Approximate Carry Look-Ahead Adder," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 65, no. 8, pp. 1089-1093, Aug. 2018, doi: 10.1109/TCSII.2016.2633307.

[5] K. -L. Tsai, Y. -J. Chang, C. -H. Wang and C. -T. Chiang, "Accuracy-Configurable Radix-4 Adder With a Dynamic Output Modification Scheme," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 8, pp. 3328-3336, Aug. 2021, doi: 10.1109/TCSI.2021.3085572.

七、 附录

7.1 stimulus.vect

```
; basic stimulus and instructions for test
radix 44444444 44444444 1 144444444
vname A[[31:0]] B[[31:0]] cin SUM[[32:0]]
io iiiiiiii iiiiiiii i ooooooooo

tunit ns
tdelay 0.5
```

odelay 1
slope 0.015
period 1

vih 0.7
vil 0
vth 0.35

; For dynamic power
;EBE3F35A F91E7AF4 0 1E5026E4E
;F8329902 D2F46076 0 1CB26F978
;AF46829C D00E34B7 0 17F54B753
;7BDCAB5E 6B0DD003 1 0E6EA7B62
;C24FD612 C661FA5B 0 188B1D06D
;5FF3EA77 7330432B 0 0D3242DA2
;19D35972 67A22010 0 081757982
;0FA6291E 5465C199 0 0640BEAB7
;952BD5B5 6C8A5732 1 101B62CE8
;F77E4B59 7B6F59CA 0 172EDA523
;E21294BB 750C57BD 1 1571EEC79
;B2C11297 64C31A83 0 117842D1A
;A0722A61 0B7C7A37 1 0ABEEA499
;26106C0B 019FD10F 0 027B03D1A
;F7230ACD 13FCB8EF 1 10B1FC3BD
;38582F46 B5B36D24 1 0EE0B9C6B
;9877C71F A1219F08 1 139996628
;389F8E74 26027E78 0 05EA20CEC
;10237886 818F7BFA 1 091B2F481
;CCD90539 D8A12FC6 0 1A57A34FF
;AC2CEF40 49726B5D 1 0F59F5A9E
;0BA3F911 5A17F24B 0 065BBEB5C
;4EFED883 04B017E1 0 053AEF064
;2370A89E 3A717E2E 1 05DE226CD
;351E5D4C 981A4E80 0 0CD38ABCC

;33B7E5CB 30177C4E 1 063CF621A
;87829249 E9C15771 0 17143E9BA
;9D552769 D5C08F35 0 17315B69E
;DDD25DA9 E99B67BF 0 1C76DC568
;11A5D60D 7EC691E9 0 0906C67F6
;A85F5DD9 3731BF4B 0 0DF911D24
;DDC297B1 C91CB095 0 1A6DF4846
;6EBF0B9E 528C0ED7 1 0C14B1A76

;043D10FC 22F0CBF2 1 0272DDCEF
;9F341F6C 7EE9BEA7 1 11E1DDE14
;8B3C9FCC 98F8C844 1 124356811
;E1BC764F 6CCD3B7B 0 14E89B1CA
;6E2CAA0C 2C6F515F 0 09A9BFB6B
;B1D5A5AB 433F7BB2 1 0F515215E
;D5BD9147 13A7505B 0 0E964E1A2
;317E38AC 6CCDA625 0 09E4BDED1
;E95D840D 76C0F6D3 0 1601E7AE0
;09F7935B BE97F422 1 0C88F877E
;9B6426F7 8E39ED2E 1 1299E1426
;03841515 234DC0A1 0 026D1D5B6
;CC46E52B 95F283DB 0 162396906
;027823D7 0A69BBC4 1 00CE1DF9C
;74239A66 254B8710 1 0996F2177
;D925F958 D5F7538F 0 1AF1D4CE7
;EE278E87 00018D85 0 0EE291C0C

;3B30272E 19001E60 0 05430458E
;6FCE42B3 0C057524 0 07BD3B7D7
;D1E6E92F 751A5FE1 1 147014911
;EFEC9204 28EC10BC 0 118D8A2C0
;0289B66E D17EB4B6 0 0D4086B24
;C2CF8D52 1A4B6680 1 0DD1AF3D3
;F96A98EC 113ACC85 0 10AA56571
;3B3864D9 3B3AD0D4 0 0767335AD
;4E016275 94273D1B 1 0E2289F91
;02F5C2CD 0A76BF08 0 00D6C81D5
;BF465FD8 4DB570AA 1 10CFBD083
;6281D085 8A86325D 1 0ED0802E3
;9EC5B668 907FF557 0 12F45ABBF
;31BA3401 A2011565 1 0D3BB4967
;DBEA33EA ACF0CC1E 1 188DB0009
;B3D4992C 945292DF 0 148272C0B
;5E418433 A901D822 0 107435C55
;CEC10B16 5F3F6E10 0 12E007926
;E5ACFF90 FFC0D9C8 0 1E56DD958
;29BFD5B7 02978184 1 02C57573C
;C7D37E30 5666C2A1 1 11E3A40D2
;3A3F2B83 FB60F406 1 135A01F8A
;AD8E51E9 4FE8FB97 0 0FD774D80
;05F27630 8C17A4D4 0 0920A1B04
;611CBD00 11A963BF 0 072C620BF

```
; For leakage power
;00000000 00000000 0 000000000
;FFFFFFFF FFFFFFFF 1 1FFFFFFFF

; You may add the input patterns for critical paths here:
; 针对精确全加器
;00000000 00000000 0 000000000
;00000001 7FFFFFFF 0 080000000

;00000000 00000000 0 000000000
;00000001 FFFFFFFF 0 100000000
; 针对近似加法器
;00000000 00000000 0 000000000
;00010000 7FFF0000 0 080000000

00000000 00000000 0 000000000
00010000 FFFF0000 0 100000000
```

7.2 common_elements.net

```
* size=1 inverter
.subckt inv_size_1 in out vdd vss
	Xpfet out in vdd vdd pfet l=lg nfin=1
	Xnfet out in vss vss nfet l=lg nfin=1
.ends inv_size_1

* size=2 inverter
.subckt inv_size_2 in out vdd vss
	Xpfet out in vdd vdd pfet l=lg nfin=2
	Xnfet out in vss vss nfet l=lg nfin=2
.ends inv_size_2

* size=4 inverter
.subckt inv_size_4 in out vdd vss
	Xpfet out in vdd vdd pfet l=lg nfin=4
	Xnfet out in vss vss nfet l=lg nfin=4
.ends inv_size_4

* size=8 inverter
.subckt inv_size_8 in out vdd vss
	Xpfet out in vdd vdd pfet l=lg nfin=8
	Xnfet out in vss vss nfet l=lg nfin=8
.ends inv_size_8
```

```
* customize size inverter
.subckt inv in out vdd vss p_size=1 n_size=1
	Xpfet out in vdd vdd pfet l=lg nfin=p_size
	Xnfet out in vss vss nfet l=lg nfin=n_size
.ends inv

* transmission gate
.subckt tg in out enable enable_n vdd vss
	Xpfet out enable_n in vdd pfet l=lg nfin=1
	Xnfet out enable   in vss nfet l=lg nfin=1
.ends tg

* 2-input xor gate
.subckt xor in_1 in_2 out vdd vss
	* Two-input NOR gate
	Xpfet_1 node_1 in_1 vdd     vdd pfet l=lg nfin=1
	Xpfet_2 s      in_2 node_1 vdd pfet l=lg nfin=1
	Xnfet_1 s      in_1 vss     vss nfet l=lg nfin=1
	Xnfet_2 s      in_2 vss     vss nfet l=lg nfin=1
	* F = !(AB+S)
	Xpfet_3 node_2 in_1 vdd     vdd pfet l=lg nfin=1
	Xpfet_4 node_2 in_2 vdd     vdd pfet l=lg nfin=1
	Xpfet_5 out    s    node_2 vdd pfet l=lg nfin=1
	Xnfet_3 out    in_1 node_3 vss nfet l=lg nfin=1
	Xnfet_4 node_3 in_2 vss    vss nfet l=lg nfin=1
	Xnfet_5 out    s    vss     vss nfet l=lg nfin=1
.ends xor

* 2-input or gate
.subckt or2 in_1 in_2 out vdd vss
	* Two-input NOR gate
	Xpfet_1 node_1 in_1 vdd     vdd pfet l=lg nfin=1
	Xpfet_2 s      in_2 node_1 vdd pfet l=lg nfin=1
	Xnfet_1 s      in_1 vss     vss nfet l=lg nfin=1
	Xnfet_2 s      in_2 vss     vss nfet l=lg nfin=1

	Xinv_size_1 s   out   vdd     vss inv_size_1
.ends or2

* 3-input or gate
.subckt or3 in_1 in_2 in_3 out vdd vss
	* Three-input NOR gate
	Xpfet_1 node_1 in_1 vdd     vdd pfet l=lg nfin=1
	Xpfet_2 node_2 in_2 node_1 vdd pfet l=lg nfin=1
```

```
    Xpfet_3 s       in_3 node_2 vdd pfet l=lg nfin=1
    Xnfet_1 s       in_1 vss    vss nfet l=lg nfin=1
    Xnfet_2 s       in_2 vss    vss nfet l=lg nfin=1
    Xnfet_3 s       in_3 vss    vss nfet l=lg nfin=1

    Xinv_size_1 s out vdd vss inv_size_1
.ends or3

* 4-input or gate
.subckt or4 in_1 in_2 in_3 in_4 out vdd vss
    * Four-input NOR gate
    Xpfet_1 node_1 in_1 vdd     vdd pfet l=lg nfin=1
    Xpfet_2 node_2 in_2 node_1 vdd pfet l=lg nfin=1
    Xpfet_3 node_3 in_3 node_2 vdd pfet l=lg nfin=1
    Xpfet_4 s       in_4 node_3 vdd pfet l=lg nfin=1
    Xnfet_1 s       in_1 vss    vss nfet l=lg nfin=1
    Xnfet_2 s       in_2 vss    vss nfet l=lg nfin=1
    Xnfet_3 s       in_3 vss    vss nfet l=lg nfin=1
    Xnfet_4 s       in_4 vss    vss nfet l=lg nfin=1

    Xinv_size_1 s out vdd vss inv_size_1
.ends or4

* 5-input or gate
.subckt or5 in_1 in_2 in_3 in_4 in_5 out vdd vss
    * Four-input NOR gate
    Xpfet_1 node_1 in_1 vdd     vdd pfet l=lg nfin=1
    Xpfet_2 node_2 in_2 node_1 vdd pfet l=lg nfin=1
    Xpfet_3 node_3 in_3 node_2 vdd pfet l=lg nfin=1
    Xpfet_4 node_4 in_4 node_3 vdd pfet l=lg nfin=1
    Xpfet_5 s       in_5 node_4 vdd pfet l=lg nfin=1
    Xnfet_1 s       in_1 vss    vss nfet l=lg nfin=1
    Xnfet_2 s       in_2 vss    vss nfet l=lg nfin=1
    Xnfet_3 s       in_3 vss    vss nfet l=lg nfin=1
    Xnfet_4 s       in_4 vss    vss nfet l=lg nfin=1
    Xnfet_5 s       in_5 vss    vss nfet l=lg nfin=1

    Xinv_size_1 s out vdd vss inv_size_1
.ends or5

* 2-input and gate
.subckt and2 in_1 in_2 out vdd vss
    * Two-input AND gate
    Xpfet_1 s       in_1 vdd     vdd pfet l=lg nfin=1
```

```
    Xpfet_2 s          in_2 vdd       vdd pfet l=lg nfin=1
    Xnfet_1 node_1 in_1 vss       vss nfet l=lg nfin=1
    Xnfet_2 s          in_2 node_1 vss nfet l=lg nfin=1

    Xinv_size_1 s    out    vdd       vss inv_size_1
.ends and2

* 3-input and gate
.subckt and3 in_1 in_2 in_3 out vdd vss
    * Three-input AND gate
    Xpfet_1 s          in_1 vdd       vdd pfet l=lg nfin=1
    Xpfet_2 s          in_2 vdd       vdd pfet l=lg nfin=1
    Xpfet_3 s          in_3 vdd       vdd pfet l=lg nfin=1
    Xnfet_1 node_1 in_1 vss       vss nfet l=lg nfin=1
    Xnfet_2 node_2 in_2 node_1 vss nfet l=lg nfin=1
    Xnfet_3 s          in_3 node_2 vss nfet l=lg nfin=1

    Xinv_size_1 s    out    vdd       vss inv_size_1
.ends and3

* 4-input and gate
.subckt and4 in_1 in_2 in_3 in_4 out vdd vss
    * Three-input AND gate
    Xpfet_1 s          in_1 vdd       vdd pfet l=lg nfin=1
    Xpfet_2 s          in_2 vdd       vdd pfet l=lg nfin=1
    Xpfet_3 s          in_3 vdd       vdd pfet l=lg nfin=1
    Xpfet_4 s          in_4 vdd       vdd pfet l=lg nfin=1
    Xnfet_1 node_1 in_1 vss       vss nfet l=lg nfin=1
    Xnfet_2 node_2 in_2 node_1 vss nfet l=lg nfin=1
    Xnfet_3 node_3 in_3 node_2 vss nfet l=lg nfin=1
    Xnfet_4 s          in_4 node_3 vss nfet l=lg nfin=1

    Xinv_size_1 s    out    vdd       vss inv_size_1
.ends and4

* 5-input and gate
.subckt and5 in_1 in_2 in_3 in_4 in_5 out vdd vss
    * Three-input AND gate
    Xpfet_1 s          in_1 vdd       vdd pfet l=lg nfin=1
    Xpfet_2 s          in_2 vdd       vdd pfet l=lg nfin=1
    Xpfet_3 s          in_3 vdd       vdd pfet l=lg nfin=1
    Xpfet_4 s          in_4 vdd       vdd pfet l=lg nfin=1
    Xpfet_5 s          in_5 vdd       vdd pfet l=lg nfin=1
    Xnfet_1 node_1 in_1 vss       vss nfet l=lg nfin=1
```

```
    Xnfet_2 node_2 in_2 node_1 vss nfet l=lg nfin=1
    Xnfet_3 node_3 in_3 node_2 vss nfet l=lg nfin=1
    Xnfet_4 node_4 in_4 node_3 vss nfet l=lg nfin=1
    Xnfet_5 s      in_5 node_4 vss nfet l=lg nfin=1

    Xinv_size_1 s   out   vdd     vss inv_size_1
.ends and5
```

7.3 special_elements.net

```
* 基于传输门的异或门
.subckt xor_tg in_1 in_2 out vdd vss
    Xinv_size_1 in_2   in_2_n   vdd   vss   inv_size_1

    Xpfet_inv out in_1 in_2    vdd pfet l=lg nfin=1
    Xnfet_inv out in_1 in_2_n vss nfet l=lg nfin=1

    Xpfet_tg out in_2 in_1    vdd pfet l=lg nfin=1
    Xnfet_tg out in_2_n in_1 vss nfet l=lg nfin=1
.ends xor_tg

* 2 路选择器
.subckt mux2to1 in_0 in_1 sel out vdd vss

    Xinv sel sel_n vdd vss inv p_size=3 n_size=2

    Xtg_1 in_0 out sel_n sel    vdd vss tg
    Xtg_2 in_1 out sel    sel_n vdd vss tg
.ends mux2to1

* 2-input or gate resizing
.subckt or2_r in_1 in_2 out vdd vss
    * Two-input NOR gate
    Xpfet_1 node_1 in_1 vdd     vdd pfet l=lg nfin=2
    Xpfet_2 s      in_2 node_1 vdd pfet l=lg nfin=2
    Xnfet_1 s      in_1 vss     vss nfet l=lg nfin=1
    Xnfet_2 s      in_2 vss     vss nfet l=lg nfin=1

    Xinv_size_1 s   out   vdd     vss inv_size_1
.ends or2_r

* 2-input or gate resizing&gating
.subckt or2_rg in_1 in_2 out enbale_n vdd vss
    Xpfet_g1 vdd_g1 enbale_n vdd     vdd pfet l=lg nfin=3
```

```
	* Two-input NOR gate
	Xpfet_1	node_1 in_1	vdd_g1 vdd pfet l=lg nfin=2
	Xpfet_2	s	in_2	node_1 vdd pfet l=lg nfin=2
	Xnfet_1	s	in_1	vss	vss nfet l=lg nfin=1
	Xnfet_2	s	in_2	vss	vss nfet l=lg nfin=1

	Xpfet	out	s	vdd_g1 vdd pfet l=lg nfin=1
	Xnfet	out	s	vss	vss nfet l=lg nfin=1
.ends or2_rg

* 3-input or gate resizing
.subckt or3_r in_1 in_2 in_3 out vdd vss
	* Three-input NOR gate
	Xpfet_1 node_1 in_1 vdd	vdd pfet l=lg nfin=3
	Xpfet_2 node_2 in_2 node_1 vdd pfet l=lg nfin=3
	Xpfet_3 s	in_3 node_2 vdd pfet l=lg nfin=3
	Xnfet_1 s	in_1 vss	vss nfet l=lg nfin=1
	Xnfet_2 s	in_2 vss	vss nfet l=lg nfin=1
	Xnfet_3 s	in_3 vss	vss nfet l=lg nfin=1

	Xinv_size_1 s out vdd vss inv_size_1
.ends or3_r

* 4-input or gate resizing
.subckt or4_r in_1 in_2 in_3 in_4 out vdd vss
	* Four-input NOR gate
	Xpfet_1 node_1 in_1 vdd	vdd pfet l=lg nfin=4
	Xpfet_2 node_2 in_2 node_1 vdd pfet l=lg nfin=4
	Xpfet_3 node_3 in_3 node_2 vdd pfet l=lg nfin=4
	Xpfet_4 s	in_4 node_3 vdd pfet l=lg nfin=4
	Xnfet_1 s	in_1 vss	vss nfet l=lg nfin=1
	Xnfet_2 s	in_2 vss	vss nfet l=lg nfin=1
	Xnfet_3 s	in_3 vss	vss nfet l=lg nfin=1
	Xnfet_4 s	in_4 vss	vss nfet l=lg nfin=1

	Xinv_size_1 s out vdd vss inv_size_1
.ends or4_r

* 2-input and gate resizing
.subckt and2_r in_1 in_2 out vdd vss
	* Two-input AND gate
	Xpfet_1 s	in_1 vdd	vdd pfet l=lg nfin=1
	Xpfet_2 s	in_2 vdd	vdd pfet l=lg nfin=1
	Xnfet_1 node_1 in_1 vss	vss nfet l=lg nfin=2
```

```
    Xnfet_2 s          in_2 node_1 vss nfet l=lg nfin=2

    Xinv_size_1 s   out   vdd      vss inv_size_1
.ends and2_r

* 3-input and gate resizing
.subckt and3_r in_1 in_2 in_3 out vdd vss
    * Three-input AND gate
    Xpfet_1 s          in_1 vdd      vdd pfet l=lg nfin=1
    Xpfet_2 s          in_2 vdd      vdd pfet l=lg nfin=1
    Xpfet_3 s          in_3 vdd      vdd pfet l=lg nfin=1
    Xnfet_1 node_1 in_1 vss       vss nfet l=lg nfin=3
    Xnfet_2 node_2 in_2 node_1 vss nfet l=lg nfin=3
    Xnfet_3 s          in_3 node_2 vss nfet l=lg nfin=3

    Xinv_size_1 s   out   vdd      vss inv_size_1
.ends and3_r

* 4-input and gate resizing
.subckt and4_r in_1 in_2 in_3 in_4 out vdd vss
    * Three-input AND gate
    Xpfet_1 s          in_1 vdd      vdd pfet l=lg nfin=1
    Xpfet_2 s          in_2 vdd      vdd pfet l=lg nfin=1
    Xpfet_3 s          in_3 vdd      vdd pfet l=lg nfin=1
    Xpfet_4 s          in_4 vdd      vdd pfet l=lg nfin=1
    Xnfet_1 node_1 in_1 vss       vss nfet l=lg nfin=4
    Xnfet_2 node_2 in_2 node_1 vss nfet l=lg nfin=4
    Xnfet_3 node_3 in_3 node_2 vss nfet l=lg nfin=4
    Xnfet_4 s          in_4 node_3 vss nfet l=lg nfin=4

    Xinv_size_1 s   out   vdd      vss inv_size_1
.ends and4_r

* 5-input and gate resizing&gating
.subckt and5_rg in_1 in_2 in_3 in_4 in_5 out enbale_n vdd vss
    * 门控
    Xpfet_g1 vdd_g1 enbale_n vdd   vdd pfet l=lg nfin=10
    * 两级电路
    * 一级电路的三输入与门
    Xpfet_11 s1          in_1 vdd_g1   vdd pfet l=lg nfin=1
    Xpfet_12 s1          in_2 vdd_g1   vdd pfet l=lg nfin=1
    Xpfet_13 s1          in_3 vdd_g1   vdd pfet l=lg nfin=1
    Xnfet_11 node_11   in_1 vss       vss nfet l=lg nfin=3
    Xnfet_12 node_12   in_2 node_11 vss nfet l=lg nfin=3
```

```
    Xnfet_13 s1          in_3 node_12 vss nfet l=lg nfin=3

    Xpfet1     out1      s1           vdd_g1 vdd pfet l=lg nfin=1
    Xnfet1     out1      s1           vss    vss nfet l=lg nfin=1
    *  一级电路的二输入与门
    Xpfet_21 s2          in_4 vdd_g1  vdd pfet l=lg nfin=1
    Xpfet_22 s2          in_5 vdd_g1  vdd pfet l=lg nfin=1
    Xnfet_21 node_21 in_4 vss         vss nfet l=lg nfin=2
    Xnfet_22 s2          in_5 node_21 vss nfet l=lg nfin=2

    Xpfet2     out2      s2           vdd_g1 vdd pfet l=lg nfin=1
    Xnfet2     out2      s2           vss    vss nfet l=lg nfin=1

    *  二级电路的二输入与门
    Xpfet_31 s3          out1 vdd_g1  vdd pfet l=lg nfin=1
    Xpfet_32 s3          out2 vdd_g1  vdd pfet l=lg nfin=1
    Xnfet_31 node_31 out2 vss         vss nfet l=lg nfin=2
    Xnfet_32 s3          out1 node_31 vss nfet l=lg nfin=2

    Xpfet3     out       s3           vdd_g1 vdd pfet l=lg nfin=1
    Xnfet3     out       s3           vss    vss nfet l=lg nfin=1
.ends and5_rg
```

7.4 accurate_adder.net

```
* 32 bits  精确加法器(CLA + RCA)
* 4 bits CLA
.subckt cla_4 a0 a1 a2 a3 b0 b1 b2 b3 s0 s1 s2 s3 c0 c4 vdd vss
    * P & G
    Xxor_p0    a0 b0 p0 vdd vss xor
    Xxor_p1    a1 b1 p1 vdd vss xor
    Xxor_p2    a2 b2 p2 vdd vss xor
    Xxor_p3    a3 b3 p3 vdd vss xor
    Xand2_g0 a0 b0 g0 vdd vss and2
    Xand2_g1 a1 b1 g1 vdd vss and2
    Xand2_g2 a2 b2 g2 vdd vss and2
    Xand2_g3 a3 b3 g3 vdd vss and2

    * c1 = g0 + c0p0
    Xand2_c1 c0      p0    c1_t vdd vss and2
    Xor2_c1    c1_t g0    c1      vdd vss or2

    * c2 = g1 + g0p1 + c0p1p0
    Xand3_c2 c0    p1    p0    c2_t1    vdd vss and3
```

```
    Xand2_c2 g0   p1   c2_t2   vdd vss and2
    Xor3_c2   g1 c2_t2 c2_t1 c2 vdd vss or3

    * c3 = g2 + g1p2 + g0p2p1 + c0p2p1p0
    Xand4_c3 c0   p2   p1   p0   c3_t1 vdd vss and4
    Xand3_c3 g0   p2   p1   c3_t2 vdd vss and3
    Xand2_c3 g1   p2   c3_t3 vdd vss and2
    Xor4_c3   g2 c3_t3 c3_t2 c3_t1 c3 vdd vss or4

    * c4 = g3 + g2p3 + g1p3p2 + g0p3p2p1 + c0p3p2p1p0
    Xand5_c4 c0   p3   p2   p1   p0   c4_t1 vdd vss and5
    Xand4_c4 g0   p3   p2   p1   c4_t2 vdd vss and4
    Xand3_c4 g1   p3   p2   c4_t3 vdd vss and3
    Xand2_c4 g2   p3   c4_t4 vdd vss and2
    Xor5_c4   g3 c4_t4 c4_t3 c4_t2 c4_t1 c4 vdd vss or5

    * s0 = p0 xor c0
    Xxor_s0   p0 c0 s0 vdd vss xor
    * s1 = p1 xor c1
    Xxor_s1   p1 c1 s1 vdd vss xor
    * s2 = p2 xor c2
    Xxor_s2   p2 c2 s2 vdd vss xor
    * s3 = p3 xor c3
    Xxor_s3   p3 c3 s3 vdd vss xor
.ends cla_4

* 4 bits CLA * 8 => 32 bits adder
.subckt adder_32_base
+ a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18 a19 a20 a21 a22
a23 a24 a25 a26 a27 a28 a29 a30 a31
+ b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 b17 b18 b19 b20 b21
b22 b23 b24 b25 b26 b27 b28 b29 b30 b31
+ s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23
s24 s25 s26 s27 s28 s29 s30 s31 cout
+ cin vdd vss
    Xcla_4_1 a0   a1   a2   a3   b0   b1   b2   b3   s0   s1   s2   s3   cin
cout_i1 vdd vss cla_4
    Xcla_4_2 a4   a5   a6   a7   b4   b5   b6   b7   s4   s5   s6   s7   cout_i1
cout_i2 vdd vss cla_4
    Xcla_4_3 a8   a9   a10 a11 b8   b9   b10 b11 s8   s9   s10 s11 cout_i2 cout_i3
vdd vss cla_4
    Xcla_4_4 a12 a13 a14 a15 b12 b13 b14 b15 s12 s13 s14 s15 cout_i3 cout_i4 vdd
vss cla_4
    Xcla_4_5 a16 a17 a18 a19 b16 b17 b18 b19 s16 s17 s18 s19 cout_i4 cout_i5 vdd
```

```
vss cla_4
    Xcla_4_6 a20 a21 a22 a23 b20 b21 b22 b23 s20 s21 s22 s23 cout_i5 cout_i6 vdd
vss cla_4
    Xcla_4_7 a24 a25 a26 a27 b24 b25 b26 b27 s24 s25 s26 s27 cout_i6 cout_i7 vdd
vss cla_4
    Xcla_4_8 a28 a29 a30 a31 b28 b29 b30 b31 s28 s29 s30 s31 cout_i7 cout      vdd
vss cla_4
.ends adder_32_base
```

7.5 approx_adder.net

```
* 4 bits CLA (approximate)
.subckt cla_4_app a0 a1 a2 a3 b0 b1 b2 b3 s0 s1 s2 s3 c0 c4 app vdd vss
    * P & G
    Xxor_tg_p0   a0 b0 p0 vdd vss xor_tg
    Xxor_tg_p1   a1 b1 p1 vdd vss xor_tg
    Xxor_tg_p2   a2 b2 p2 vdd vss xor_tg
    Xxor_tg_p3   a3 b3 p3 vdd vss xor_tg
    Xand2_r_g0 a0 b0 g0 vdd vss and2_r
    Xand2_r_g1 a1 b1 g1 vdd vss and2_r
    Xand2_r_g2 a2 b2 g2 vdd vss and2_r
    Xand2_r_g3 a3 b3 g3 vdd vss and2_r

    * c1 = g0 + c0p0
    Xand2_r_c1 p0     c0     c1_t vdd vss and2_r
    Xor2_r_c1   g0 c1_t c1 vdd vss or2_r

    * c2 = g1 + g0p1 + c0p1p0
    Xand3_r_c2 p1   p0   c0   c2_t1   vdd vss and3_r
    Xand2_r_c2 p1   g0 c2_t2   vdd vss and2_r
    Xor3_r_c2   g1 c2_t2 c2_t1   c2 vdd vss or3_r

    * c3 = g2 + g1p2 + g0p2p1 + c0p2p1p0
    Xand4_r_c3 p2   p1   p0   c0   c3_t1 vdd vss and4_r
    Xand3_r_c3 p2   p1   g0   c3_t2 vdd vss and3_r
    Xand2_r_c3 p2   g1   c3_t3 vdd vss and2_r
    Xor4_r_c3   g2 c3_t3 c3_t2 c3_t1 c3 vdd vss or4_r

    * (approximate) c4 = g3 + g2p3 + g1p3p2 + g0p3p2p1 + c0p3p2p1p0
    Xand5_rg_c4 p3   p2   p1   p0   c0   c4_t1 app vdd vss and5_rg
    Xand4_r_c4   p3   p2   p1   g0   c4_t2 vdd vss and4_r
    Xand3_r_c4   p3   p2   g1   c4_t3 vdd vss and3_r
    Xand2_r_c4   p3   g2   c4_t4 vdd vss and2_r
```

```
    Xor4_r_c4      g3 c4_t4 c4_t3 c4_t2 c4_1 vdd vss or4_r

    Xor2_rg_c4 c4_1 c4_t1 c4_0 app vdd vss or2_rg

    Xmux2to1 c4_0 c4_1 app c4 vdd vss mux2to1


    * s0 = p0 xor c0
    Xxor_tg_s0    p0 c0 s0 vdd vss xor_tg
    * s1 = p1 xor c1
    Xxor_tg_s1    p1 c1 s1 vdd vss xor_tg
    * s2 = p2 xor c2
    Xxor_tg_s2    p2 c2 s2 vdd vss xor_tg
    * s3 = p3 xor c3
    Xxor_tg_s3    p3 c3 s3 vdd vss xor_tg
.ends cla_4_app

* 4 bits CLA * 8 => 32 bits adder (approximate)
.subckt adder_32_app
+ a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a18 a19 a20 a21 a22
a23 a24 a25 a26 a27 a28 a29 a30 a31
+ b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15 b16 b17 b18 b19 b20 b21
b22 b23 b24 b25 b26 b27 b28 b29 b30 b31
+ s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23
s24 s25 s26 s27 s28 s29 s30 s31 cout
+ app1 app2 app3 app4 app5 app6 app7 app8
+ cin vdd vss
    Xcla_4_app_1 a0 a1 a2 a3 b0 b1 b2 b3 s0 s1 s2 s3 cin
cout_i1 app1 vdd vss cla_4_app
    Xcla_4_app_2 a4 a5 a6 a7 b4 b5 b6 b7 s4 s5 s6 s7 cout_i1
cout_i2 app2 vdd vss cla_4_app
    Xcla_4_app_3 a8 a9 a10 a11 b8 b9 b10 b11 s8 s9 s10 s11 cout_i2
cout_i3 app3 vdd vss cla_4_app
    Xcla_4_app_4 a12 a13 a14 a15 b12 b13 b14 b15 s12 s13 s14 s15 cout_i3 cout_i4
app4 vdd vss cla_4_app
    Xcla_4_app_5 a16 a17 a18 a19 b16 b17 b18 b19 s16 s17 s18 s19 cout_i4 cout_i5
app5 vdd vss cla_4_app
    Xcla_4_app_6 a20 a21 a22 a23 b20 b21 b22 b23 s20 s21 s22 s23 cout_i5 cout_i6
app6 vdd vss cla_4_app
    Xcla_4_app_7 a24 a25 a26 a27 b24 b25 b26 b27 s24 s25 s26 s27 cout_i6 cout_i7
app7 vdd vss cla_4_app
    Xcla_4_app_8 a28 a29 a30 a31 b28 b29 b30 b31 s28 s29 s30 s31 cout_i7 cout
app8 vdd vss cla_4_app
.ends adder_32_app
```

7.6 adder_test.sp

```
adder_test

* 工艺库
.lib "../PTM-MG/modelfiles/models" ptm7lstp

* 原件库电路网表
.inc "./common_elements.net"
.inc "./special_elements.net"

* 负载电路
.inc "./load_circuit.net"

* 加法器电路网表
.inc "./accurate_adder.net"
.inc "./approx_adder.net"

* 激励输入
.vec "./stimulus.vect"

.option acct list post probe
.global VDD_DESIGN VDD_LOAD VSS_COM
.temp 25

.param SUPPLY = 0.7v
.param T          = 26ns

* Xadder_32_base
* + A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9] A[10] A[11] A[12] A[13] A[14] A[15]
* + A[16] A[17] A[18] A[19] A[20] A[21] A[22] A[23] A[24] A[25] A[26] A[27] A[28] A[29] A[30] A[31]
* + B[0] B[1] B[2] B[3] B[4] B[5] B[6] B[7] B[8] B[9] B[10] B[11] B[12] B[13] B[14] B[15]
* + B[16] B[17] B[18] B[19] B[20] B[21] B[22] B[23] B[24] B[25] B[26] B[27] B[28] B[29] B[30] B[31]
* + SUM[0] SUM[1] SUM[2] SUM[3] SUM[4] SUM[5] SUM[6] SUM[7] SUM[8] SUM[9] SUM[10] SUM[11] SUM[12] SUM[13] SUM[14] SUM[15]
* + SUM[16] SUM[17] SUM[18] SUM[19] SUM[20] SUM[21] SUM[22] SUM[23] SUM[24] SUM[25] SUM[26] SUM[27] SUM[28] SUM[29] SUM[30] SUM[31]
* + SUM[32]
```

```
* + cin VDD_DESIGN VSS_COM adder_32_base

Xadder_32_app
+ A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9] A[10] A[11] A[12] A[13] A[14] A[15]
+ A[16] A[17] A[18] A[19] A[20] A[21] A[22] A[23] A[24] A[25] A[26] A[27] A[28] A[29] A[30] A[31]
+ B[0] B[1] B[2] B[3] B[4] B[5] B[6] B[7] B[8] B[9] B[10] B[11] B[12] B[13] B[14] B[15]
+ B[16] B[17] B[18] B[19] B[20] B[21] B[22] B[23] B[24] B[25] B[26] B[27] B[28] B[29] B[30] B[31]
+ SUM[0] SUM[1] SUM[2] SUM[3] SUM[4] SUM[5] SUM[6] SUM[7] SUM[8] SUM[9] SUM[10] SUM[11] SUM[12] SUM[13] SUM[14] SUM[15]
+ SUM[16] SUM[17] SUM[18] SUM[19] SUM[20] SUM[21] SUM[22] SUM[23] SUM[24] SUM[25] SUM[26] SUM[27] SUM[28] SUM[29] SUM[30] SUM[31]
+ SUM[32]
+ VDD_DESIGN VDD_DESIGN VDD_DESIGN VDD_DESIGN VDD_DESIGN VSS_COM VSS_COM VSS_COM
+ cin VDD_DESIGN VSS_COM adder_32_app


* 外部供电
VVDD_DESIGN VDD_DESIGN 0 'SUPPLY'
VVDD_LOAD   VDD_LOAD   0 'SUPPLY'
VVSS_COM    VSS_COM    0 0


.tran 1ns 'T'

* 功耗测量
* .measure tran avg_p AVG p(VVDD_DESIGN) from=0ns to='T'

* 关键路径延迟测量
* 针对精确全加器
*.measure tran critical_t trig v(A[0]) val='supply/2' rise=1 targ v(SUM[31]) val='supply/2' rise=1
*.measure tran critical_t trig v(A[0]) val='supply/2' rise=1 targ v(SUM[32]) val='supply/2' rise=1
* 针对近似加法器
*.measure tran critical_t trig v(A[16]) val='supply/2' rise=1 targ v(SUM[31]) val='supply/2' rise=1
.measure tran critical_t trig v(A[16]) val='supply/2' rise=1 targ v(SUM[32]) val='supply/2' rise=1
```

* 输出设置
.probe tran v(cin)
.probe tran v(A[0])
.probe tran v(A[1])
.probe tran v(A[2])
.probe tran v(A[3])
.probe tran v(A[4])
.probe tran v(A[5])
.probe tran v(A[6])
.probe tran v(A[7])
.probe tran v(A[8])
.probe tran v(A[9])
.probe tran v(A[10])
.probe tran v(A[11])
.probe tran v(A[12])
.probe tran v(A[13])
.probe tran v(A[14])
.probe tran v(A[15])
.probe tran v(A[16])
.probe tran v(A[17])
.probe tran v(A[18])
.probe tran v(A[19])
.probe tran v(A[20])
.probe tran v(A[21])
.probe tran v(A[22])
.probe tran v(A[23])
.probe tran v(A[24])
.probe tran v(A[25])
.probe tran v(A[26])
.probe tran v(A[27])
.probe tran v(A[28])
.probe tran v(A[29])
.probe tran v(A[30])
.probe tran v(A[31])
.probe tran v(B[0])
.probe tran v(B[1])
.probe tran v(B[2])
.probe tran v(B[3])
.probe tran v(B[4])
.probe tran v(B[5])
.probe tran v(B[6])
.probe tran v(B[7])
.probe tran v(B[8])

```
.probe tran v(B[9])
.probe tran v(B[10])
.probe tran v(B[11])
.probe tran v(B[12])
.probe tran v(B[13])
.probe tran v(B[14])
.probe tran v(B[15])
.probe tran v(B[16])
.probe tran v(B[17])
.probe tran v(B[18])
.probe tran v(B[19])
.probe tran v(B[20])
.probe tran v(B[21])
.probe tran v(B[22])
.probe tran v(B[23])
.probe tran v(B[24])
.probe tran v(B[25])
.probe tran v(B[26])
.probe tran v(B[27])
.probe tran v(B[28])
.probe tran v(B[29])
.probe tran v(B[30])
.probe tran v(B[31])
.probe tran v(SUM[0])
.probe tran v(SUM[1])
.probe tran v(SUM[2])
.probe tran v(SUM[3])
.probe tran v(SUM[4])
.probe tran v(SUM[5])
.probe tran v(SUM[6])
.probe tran v(SUM[7])
.probe tran v(SUM[8])
.probe tran v(SUM[9])
.probe tran v(SUM[10])
.probe tran v(SUM[11])
.probe tran v(SUM[12])
.probe tran v(SUM[13])
.probe tran v(SUM[14])
.probe tran v(SUM[15])
.probe tran v(SUM[16])
.probe tran v(SUM[17])
.probe tran v(SUM[18])
.probe tran v(SUM[19])
.probe tran v(SUM[20])
```

```
.probe tran v(SUM[21])
.probe tran v(SUM[22])
.probe tran v(SUM[23])
.probe tran v(SUM[24])
.probe tran v(SUM[25])
.probe tran v(SUM[26])
.probe tran v(SUM[27])
.probe tran v(SUM[28])
.probe tran v(SUM[29])
.probe tran v(SUM[30])
.probe tran v(SUM[31])
.probe tran v(SUM[32])

.end
```

7.7 load_circuit.net

```
* 输出负载
Xinv_size_8_0   SUM[0]   out0   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_1   SUM[1]   out1   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_2   SUM[2]   out2   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_3   SUM[3]   out3   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_4   SUM[4]   out4   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_5   SUM[5]   out5   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_6   SUM[6]   out6   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_7   SUM[7]   out7   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_8   SUM[8]   out8   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_9   SUM[9]   out9   VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_10 SUM[10] out10 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_11 SUM[11] out11 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_12 SUM[12] out12 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_13 SUM[13] out13 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_14 SUM[14] out14 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_15 SUM[15] out15 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_16 SUM[16] out16 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_17 SUM[17] out17 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_18 SUM[18] out18 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_19 SUM[19] out19 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_20 SUM[20] out20 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_21 SUM[21] out21 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_22 SUM[22] out22 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_23 SUM[23] out23 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_24 SUM[24] out24 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_25 SUM[25] out25 VDD_LOAD VSS_COM inv_size_8
```

```
Xinv_size_8_26 SUM[26] out26 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_27 SUM[27] out27 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_28 SUM[28] out28 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_29 SUM[29] out29 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_30 SUM[30] out30 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_31 SUM[31] out31 VDD_LOAD VSS_COM inv_size_8
Xinv_size_8_32 SUM[32] out32 VDD_LOAD VSS_COM inv_size_8
```