doi:10.11887/j.cn.202401021 http://journal.nudt.edu.cn

存储体编译和布局协同的片上缓存设计方法

刘必慰[1*],熊 琪[2],杨 茗[3],宋雨露[3]
(1. 国防科技大学 计算机学院,湖南 长沙 410073;2. 国防科技大学 理学院,湖南 长沙 410073;
3. 国防科技大学 军政基础教育学院,湖南 长沙 410073)

摘 要:为了提高片上缓存的速度、降低面积和功耗,提出了一种存储体编译和布局协同的片上缓存设计方法。该方法基于存储体在芯片上的不同空间位置预估该存储体的时序余量,分别采用拆分/合并、尺寸调整、阈值替换和长宽比变形等多种配置参数穷举组合进行存储体编译,根据时序余量选择最优的静态随机存取存储器存储体编译配置。将该方法与现有的物理设计步骤集成为一个完整的设计流程。实验结果表明,该方法能够降低约9.9%的功耗,同时缩短7.5%的关键路径延时。

关键词:片上缓存;静态随机存取存储器;协同设计;低功耗
中图分类号:TN492 **文献标志码**:A **文章编号**:1001-2486(2024)01-198-06

On-chip cache design method for cooperative memory compilation and layout

LIU Biwei[1*], *XIONG Qi*[2], *YANG Ming*[3], *SONG Yulu*[3]
(1. College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China;
2. College of Sciences, National University of Defense Technology, Changsha 410073, China;
3. College of Basic Education, National University of Defense Technology, Changsha 410073, China)

Abstract: In order to improve the speed of on-chip cache and reduce the area and power consumption, an on-chip cache design method based on cooperative memory compilation and layout was proposed. This method estimated the timing margin of memory array based on its different spatial positions on chip, and then performed memory compilation through exhaustive combination of various configuration parameters such as splitting/merging, size adjustment, threshold replacement and aspect ratio deformation. The best static random-access memory compilation configuration was selected according to the timing margin. This method was integrated with the existing physical design steps into a complete design flow. Experimental results show that this method can reduce the power consumption by about 9.9% and the critical path delay by 7.5%.

Keywords: on-chip cache; static random-access memory; collaborative design; low power consumption

现代微处理器和各种系统级芯片(system on chip,SoC)中都具有大容量片上缓存[1-2],它们一般由存储体编译器生成的静态随机存取存储器(static random-access memory, SRAM)存储体构成。这些单个存储体容量一般在1 Kbit~1 Mbit。多个这样的存储体,经过粘合逻辑合并和选择来形成各种大容量的Cache、Scratch Pad Memory、共享缓冲池等片上缓存结构。片上缓存的容量不断增大,其面积可达到全芯片面积的30%~45%或更高,功耗占比也随之提高,同时片上缓存往往也处于关键时序路径,决定了全芯片频率。因此进一步提高片上缓存的性能,并降低其功耗是提高芯片性能的关键。

现有的片上缓存的设计可划分为三个独立的阶段:①存储体编译与生成方面,有学者研究了SRAM存储体本身的定制优化设计[3-4],有的学者提出了高性能低功耗存储编译技术[5-6],也有学者在存储体集成上开展研究——Gupta等提出使用异构的存储体来优化片上存储子系统的总体面积[7],闫战磊等提出了超大容量存储体的定制优化技术[8];②在存储体布局方面,主要依赖于设计者人工进行,Cadence公司的mix placer工具[9]实现了自动存储体布局的功能,相比人工布局在性能和功耗上均有改进;③存储体相关粘合逻辑优化,通过现有的综合和布局布线工具实现。

然而,现有的设计流程中这三个阶段是相互

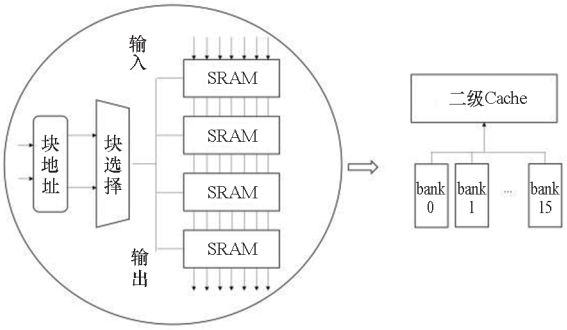
收稿日期:2021-11-01
基金项目:国家部委基金资助项目
*第一作者:刘必慰(1979—),男,江苏盐城人,研究员,博士,硕士生导师,E-mail:liubiwei04@nudt.edu.cn

割裂的。存储体编译时仅考虑存储体的功能,存储体布局时仅考虑存储体的几何尺寸,优化粘合逻辑时才考虑存储体的时序。这使得在存储体编译时不得不预留较大的时序余量,从而造成不必要的面积和功耗开销。

为了实现高性能、低功耗的片上缓存,本文提出了一种存储体编译和布局协同的设计方法,在存储体编译的同时就考虑该存储体的位置信息,从而准确地制定存储体编译的时序要求,编译出速度符合要求、功耗最优化的 SRAM 存储体。

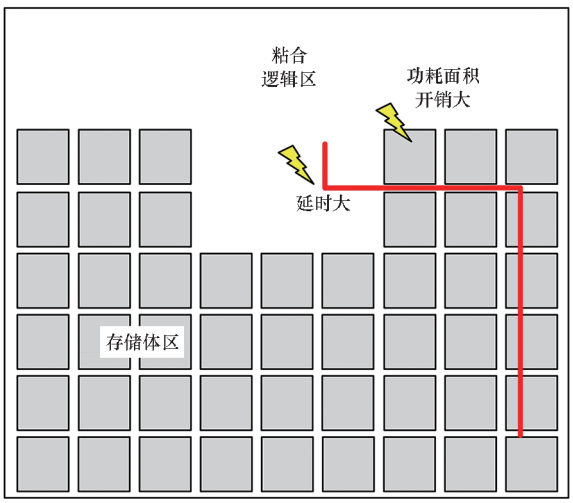
1　片上缓存构成与存储体编译空间

片上缓存的一般结构如图 1 所示,主体是 SRAM 存储体组成的存储阵列;存储阵列外是粘合逻辑——包括一系列的合并、选择、寄存操作,或者总线协议转换逻辑;最后输出到外部单元或总线上。



(a) 逻辑结构

(a) Logical structure



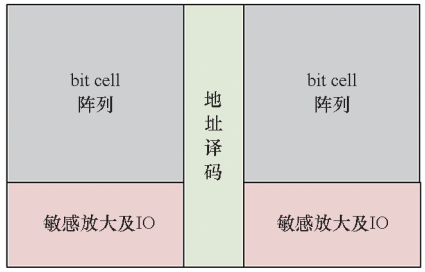
(b) 物理结构

(b) Physical structure

图 1　片上缓存的结构

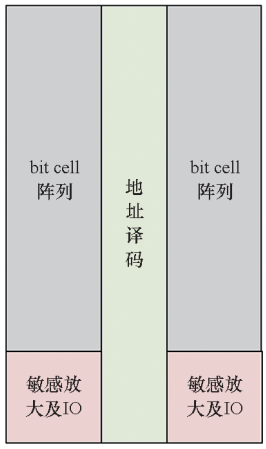
Fig. 1　On-chip cache structure

由于存储体具有较大的尺寸,不同位置的存储体与粘合逻辑的距离各不相同。例如图 1 中角部的存储体与粘合逻辑的距离较远,需要预留较大的时序余量,并采用速度快的存储体。然而中心区域的存储体与粘合逻辑的距离很近,如果和角部的存储体采用相同的编译配置则会导致功耗和面积的浪费。因此,对不同位置的存储体采用不同的编译配置能够减小面积、功耗并同时提升速度。一般来说存储体的编译配置包括阈值替换、尺寸调整、长宽比变形、拆分/合并四种,如图 2 所示。



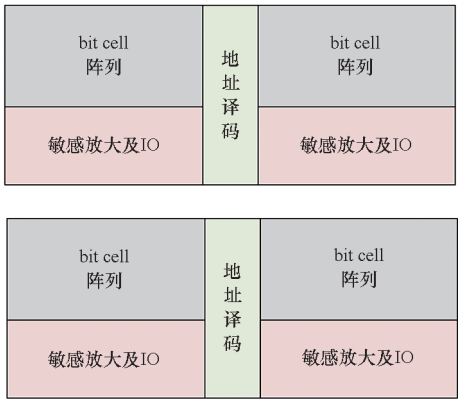
(a) 基本配置

(a) Basic configuration



(b) 改变列多选

(b) Change the MUX



(c) 拆分

(c) Splitting

图 2　存储体编译的多种配置

Fig. 2　Multiple configuration of memory compilation

1.1 阈值替换

集成电路中经常采用阈值替换的方式在速度和功耗之间权衡。低阈值的晶体管速度快但功耗高;高阈值的晶体管速度慢但功耗低,在标准单元里较为常见。阈值替换的优点是,其占用的面积相同,不需要调整芯片的布局规划,可以在物理设计的任意阶段替换插入。

存储体编译的基本配置如图 2(a)所示,SRAM 主要包含 bit cell 阵列、敏感放大及输入/输出(input/output, IO)、地址译码三个部分。阈值替换一般仅对地址译码以及 IO 等外围电路进行阈值替换。bit cell 的阈值替换会影响其噪声容限等特性,因此一般不进行阈值替换。

1.2 尺寸调整

尺寸调整也是速度和功耗之间权衡的一种常见方式。一般来说增大尺寸可以提高速度,但同时也会带来面积和功耗的增加。尺寸调整以 bit cell 为核心进行,根据性能要求增大 bit cell 的尺寸,然后根据 bit cell 的尺寸,调整设计译码器、IO、列多选器(multiplexer, MUX)等外围电路的尺寸。这一方法将会显著改变存储体面积,从而导致预布局规划(floorplan)的改变。

1.3 长宽比变形

存储体可以保持容量不变,但使长宽比发生改变。方法之一是改变存储体内的列多选,例如使得 bit cell 阵列的行数减半但列数加倍,再增加一个列多选来选择输出数据,如图 2(b)所示。另一种保持容量不变的方法是改变存储体的深度与宽度,例如深度减半而宽度加倍或反过来。也有以上两种方法的组合。

总的来说,这些方法能保持存储体的容量不变、总面积基本不变,但会明显改变字线和位线的长度,使得速度和功耗都有所变化。

1.4 拆分/合并

拆分有两种方式:一种是保持深度不变、拆分位宽,这种情况下,拆分后的小存储体可以保持和原大存储体的译码电路相同,仅仅将 bit cell 阵列宽度缩减一半;另一种是保持位宽不变、拆分深度,如图 2(c)所示,这种情况下,bit cell 阵列高度将缩减一半,地址缩减 1 位,译码电路随之缩减。拆分后的两个小存储体都保持原有数据位宽,使得数据线增加一倍,在外部需要加多选器在多个小存储体之间选择数据,这将对逻辑设计、布线都有影响。这两种方法都会缩短字线或位线,从而提高速度。但是拆分后,各个体都需要译码电路等外围电路,且需要增加存储体间的间距,因此面积和功耗会变大。反之可以将 2 个(或多个)存储体合并为 1 个,从而减小面积和功耗,但会导致性能降低。

2 协同编译方法

本文打破原有位置无关的同构 SRAM 存储体编译方法,提出了一种存储体位置驱动的存储体协同编译的流程。对远离粘合逻辑的存储体通过拆分、低阈值替换、增大尺寸等方式来加快速度;对于靠近粘合逻辑的存储体通过合并、高阈值替换、减小尺寸等方式来减小面积和功耗,达到时序、功耗和面积的均衡。其具体过程如图 3 所示。

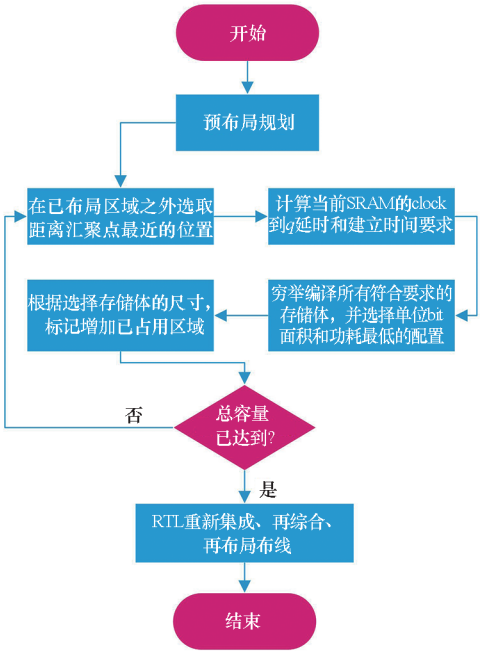


图 3 协同编译流程

Fig. 3 Cooperative compilation flow

设计包含粘合逻辑和存储体两部分。首先基于对粘合逻辑进行 floorplan,得到粘合逻辑的布局区域,在其中选取存储体的汇聚点,作为存储体距离计算的基准点——汇聚点一般为粘合逻辑区域的中心,或中心靠近端口的位置,同时在 floorplan 中将粘合逻辑区域设为已布局区域。

其次,在预布局规划中选择离汇聚点最近的未布局区域的位置作为第一个存储体布局位置。根据它相对于汇聚点的距离计算出对于布局在该位置存储体的时序要求。

然后,穷举编译可能的存储体,在包括深度、位宽、阈值、尺寸、长宽比等几个维度上穷举编译,在其中选择时序满足要求且单 bit 面积和单 bit 功

耗最小的一个。

最后,根据选出的存储体的尺寸,在 floorplan 中标记该区域为已布局区域,并记录已布局的总容量。继续选择下一个最近位置进行存储体编译,直到已布局存储体总容量达到预期。这样就得到了所有存储体的配置和布局位置。

经过以上流程,存储体可能有拆分和合并,原有的粘合逻辑可能需要微调。重新进行寄存器传输级(register transfer level,RTL)集成、综合,并对前序步骤生成的存储体位置进行布局布线,最终完成整个设计。

下面对位置相关的存储时序计算和穷举编译的过程再进一步详细说明。

2.1　位置相关时序约束确定

存储体协同编译过程如图 4 所示,如果已完成存储体 1 和存储体 2 的编译与布局,可以在未布局区域找到存储体 3 的布局位置,那么它的时序要求可分为两项:①粘合逻辑到存储体 3 的建立时间要求;②存储体 3 到粘合逻辑的建立时间要求。可表示为以下两个不等式:

$$t_{qm}+t_{gt}+t_{d}+t_{sr}+t_{mg}<t_{p} \tag{1}$$

$$t_{qr}+t_{gf}+t_{d}+t_{sm}+t_{mg}<t_{p} \tag{2}$$

其中:t_{qm}和 t_{sm}分别是存储体 clock 到 q 的延时和存储体的建立时间,二者是需要求解的目标;t_{gt}和 t_{gf}分别是粘合逻辑上组合逻辑的延时;t_{qr}和 t_{sr}分别是粘合逻辑上寄存器 clock 到 q 的延时和建立时间,它们可以从时序库中查表得到;t_{mg}是预留的一定余量,以抵消实际布线时少量的绕线和串扰的影响;t_{d} 是汇聚点到存储体距离造成的延时,可以采用与曼哈顿距离相关的延时模型进行计算[10];t_{p} 是电路工作的时钟周期。以上两个不等式变形即可得到存储体的时序要求:

$$t_{qm}<t_{p}-t_{gt}-t_{d}-t_{sr}-t_{mg} \tag{3}$$

$$t_{sm}<t_{p}-t_{qr}-t_{gf}-t_{d}-t_{mg} \tag{4}$$

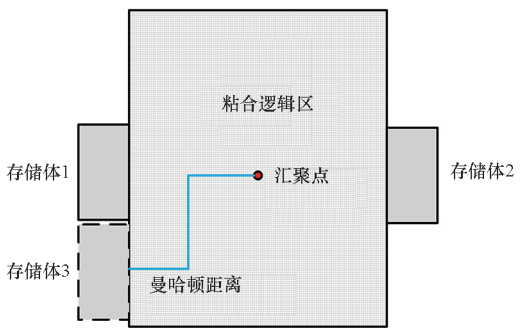


图 4　存储体协同编译过程

Fig. 4　Cooperative memory compilation process

还可以考虑使用有效时钟偏斜的设计简化约束,那样时钟树构建将更加复杂,但不影响本方法实现,因此不再做进一步的讨论。

2.2　存储体穷举编译

存储体穷举编译同时考虑深度、位宽、阈值、尺寸、长宽比等几个维度上可能的存储体实例生成,这是一个很大的搜索空间,为了在有限时间内完成搜索,需要对这些维度作出一些限制。

在宽度和深度方面,要求它们都必须是 2 的幂次,且总容量不超过 1 Mbit,这样便于 RTL 代码编写和物理实现,也不会跳过那些明显优化的配置;在阈值方面,考虑高阈值、普通阈值、低阈值三种选项;在尺寸方面,一般选择较小和较大的两种 bit cell 尺寸,更精细的尺寸选项在实际工程中也不会带来更多优化;在多选器方面,一般也为 2 的幂次,且最大不超过 16,因为超过 16 时存储器的速度会有较大程度降低。

在以上约束下,存储器的编译配置一般能缩减在 30 ~ 120 种之间,从而穷举编译能够在 3 ~ 4 个小时内完成。

3　实验与结果

针对第 2 节的方法,可以用一个实例来具体说明其过程并展示其效果。这是一个共有 48 Mbit存储容量的片上共享缓冲池,深度为 24 576,宽度为 256,具有 AXI 总线协议接口与片上其他部件互联。

图 5 是采用文献[5,8]中的传统方法进行的 SRAM 布局。该共享缓冲内部通过 96 个同构的数据存储体(data memory,DMem)。表 1 为不同

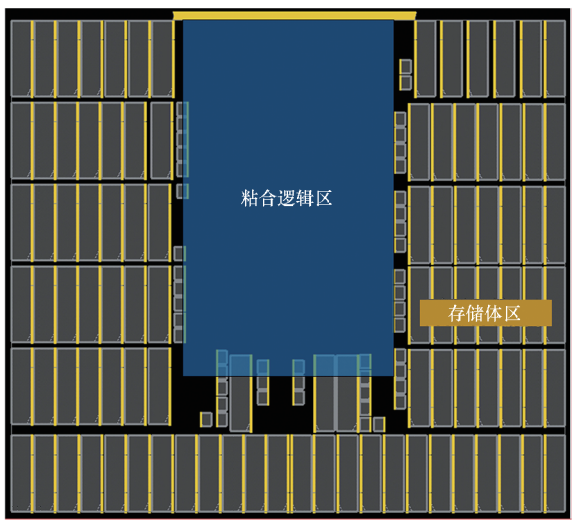


图 5　传统方法的 SRAM 布局

Fig. 5　The SRAM layout in traditional methodology

编译配置下存储体的指标，分别在最差低温（worst case low temperature，WCL）和最大漏流（max leakage，ML）两个端角下进行了分析。为了实现500 MHz的频率选择如表 1 中第一行的存储体配置方式，其容量为 4 096 × 128 bit，列多选为 4。其面积为 4.75 × 4.2 μm^2 = 19.95 μm^2，关键路径延时为 2.01 ns。

表 1 不同编译配置下存储体的指标

Tab. 1 The index of DMem under different compilation configurations

编译配置	WCL		面积/μm^2	ML	
	t_{qm}/ns	t_{sm}/ns		漏流功耗/mW	动态功耗/mW
DMem	0.80	0.19	176 × 650	3.71	2.35
DMemR	0.85	0.20	330 × 325	3.71	2.35
DMemRL	0.68	0.18	330 × 325	3.73	2.83
DMemX2	0.97	0.19	330 × 650	7.52	4.32
DMemX2H	1.25	0.29	330 × 650	7.44	3.54

采用协同编译优化的方法得到如图 6 所示的存储体的配置和布局，选取的汇聚点是粘合逻辑中间靠上的区域。其中包含四种不同存储体来实现这一片上缓存，每种存储体的具体参数如表 1 所示。DMemX2H 是深度相比 DMem 增大 2 倍且进行了高阈值电压（high voltage threshold，HVT）单元替换的存储体，它布局在最靠近中心标准单元的区域，在 ML 端角下该存储体的单位动态功耗相比原存储体有 24.7% 的减少，单位漏流功耗与基准设计基本相当。DMemX2 是深度相比 DMem 增大 2 倍的存储体，它布局在 DMemX2H 的外围，它的单位动态功耗相比原 DMem 也有 8.1% 的降低。DMemR 容量与 DMem 相同，只是深度增加了一倍、宽度降低了一半，可以减少存储体端口的布线拥塞，同时其功耗也有所降低，DMemR 放置在 DMemX2 的更外围，用于跨越更远的距离。DMemRL 是在 DMemR 的基础上进一步做了低阈值电压（low voltage threshold，LVT）单元替换，仅用在左下和右下两个角上，用于提高这两个关键位置的速度。

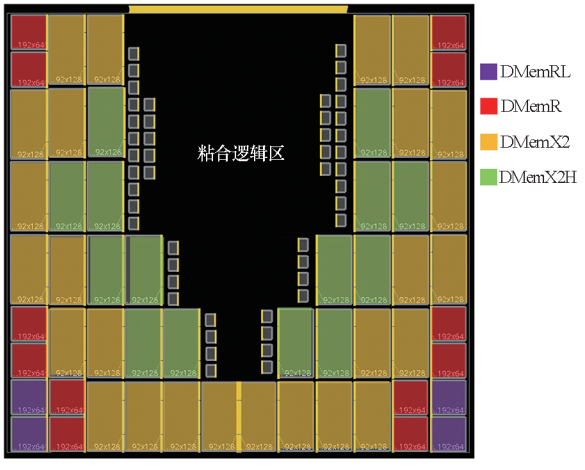


图 6 协同设计的 SRAM 布局

Fig. 6 The SRAM layout in co-operative methodology

在尝试了增大尺寸的配置后发现这种配置单位功耗过大，所以在所有位置都没有选用。

基准设计和优化设计的一些关键指标的对比如表 2 和表 3 所示。在面积方面优化设计减小了 8.6%，原因是采用了很多 DmemX2 容量的存储体，消除了存储体之间的缝隙，从而使全芯片宽度有所缩减。在关键路径延时上优化设计缩短了 7.5%，这是因为在角部采用了 LVT 的存储体。在实例数方面，在设计的各个阶段优化设计都要明显少于基准设计，这是因为采用了 DMemX2 以及 DMemR 存储体使得多选单元都在存储体内部，减少了外部所需要的多选单元及布线，因此单元数减少了 9.0%。类似的原因，布线线长也减少了 8.9%。

表 2 设计指标对比

Tab. 2 Design index comparison

设计方法	面积/mm^2	总线长/mm	关键路径延时
基准设计	4.75 × 4.2	63.0	2.01 ns@ WCL
优化设计	4.34 × 4.2	57.4	1.86 ns@ WCL

表 3 设计实例数对比

Tab. 3 Design instance count comparison

设计方法	初始实例数	place 后实例数	route 后实例数	holdfix 后实例数
基准设计	46 万	90 万	128 万	144 万
优化设计	46 万	88 万	122 万	131 万

最后，对比基准设计和优化设计的功耗情况。功耗分析都按 500 MHz 的频率进行，数据路径设定为 0.2 的翻转率，时钟路径设定为 2 的翻转率，时钟门控系数为 0.6。分别在 WCL 和 ML 两个端角下进行了分析，具体结果如表 4 所示。优化设计 WCL 下总功耗降低了 9.0%，漏流功耗与基准设计基本相当。优化设计 ML 下总功耗降低了 9.9%，漏流功耗降低了 5.8%。

表 4　功耗对比

Tab. 4　Power consumption comparison

单位:W

设计方法	WCL		ML	
	总功耗	漏流功耗	总功耗	漏流功耗
基准设计	1.78	0.24	4.55	1.54
优化设计	1.62	0.23	4.10	1.45

本文的方法已应用于一款 SoC 芯片片上共享缓存的物理设计。SoC 芯片如图 7 所示,其中,片上共享缓存实测达到了预期的频率,全芯片功耗符合预期。

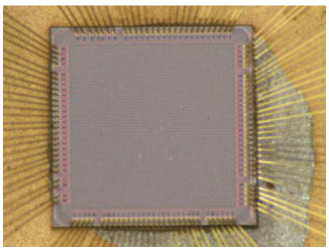


图 7　集成了优化片上缓存的 SoC 芯片

Fig. 7　SoC chip integrated optimized on-chip cache

4　结论

本文提出了一种存储体编译和布局协同的片上缓存设计方法。该方法基于存储体在芯片上的不同位置,分别采用拆分/合并、尺寸调整、阈值替换和长宽比变形等多种方法优选 SRAM 存储体编译实例,从而同时带来性能的提升、功耗和面积的减小。实验结果表明,该方法相比传统的设计方法能够降低约 9.9% 的功耗,同时缩短 7.5% 的关键路径延时。

参考文献(References)

[1]　刘胜, 卢凯, 郭阳, 等. 一种自主设计的面向 E 级高性能计算的异构融合加速器[J]. 计算机研究与发展, 2021, 58(6): 1234 – 1237.
LIU S, LU K, GUO Y, et al. A self-designed heterogeneous accelerator for exascale high performance computing[J]. Journal of Computer Research and Development, 2021, 58(6): 1234 – 1237. (in Chinese)

[2]　YANG C, CHEN S M, ZHANG J, et al. A novel DSP architecture for scientific computing and deep learning[J]. IEEE Access, 2019, 7: 36413 – 36425.

[3]　HANSRAJ, CHAUDHARY A, RANA A. Ultra low power SRAM cell for high speed applications using 90 nm CMOS technology [C]//Proceedings of the 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions), 2020.

[4]　POUSIA S, MANJITH R. Design of low power high speed SRAM architecture using SK-LCT technique [C]//Proceedings of International Conference on Current Trends towards Converging Technologies, 2018.

[5]　KAMINENI S, GUPTA S, CALHOUN B H. MemGen: an open-source framework for autonomous generation of memory macros[C]//Proceedings of IEEE Custom Integrated Circuits Conference, 2021.

[6]　GUTHAUS M R, STINE J E, ATAEI S, et al. OpenRAM: an open-source memory compiler[C]//Proceedings of 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016.

[7]　GUPTA P R, VISWESWARAN G S, NARANG G, et al. Heterogeneous memory assembly exploration using a floorplan and interconnect aware framework [C]//Proceedings of the 29th IEEE International System-on-Chip Conference, 2016.

[8]　闫战磊, 孙永节, 刘必慰. 超大容量片上存储器的定制优化设计[C]//第十七届计算机工程与工艺年会暨第三届微处理器技术论坛, 2013.
YAN Z L, SUN Y J, LIU B W. Customized and optimized design of large capacity on chip memory[C]//Proceedings of the 17th Annual Conference on Computer Engineering and technology, 2013. (in Chinese)

[9]　MCLELLAN P. Innovus mixed placer[EB/OL]. (2020 – 10 – 06)[2021 – 10 – 20]. https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/innovus-mixed-placer.

[10]　WANG X S, LIU W P, YU M Y. A distinctive O(mn) time algorithm for optimal buffer insertions[C]//Proceedings of the 16th International Symposium on Quality Electronic Design, 2015.

(编辑: 王颖娟, 罗茹馨)