摘要

随着集成电路规模的不断增大，集成工艺不断进步，对集成电路的设计方法提出了更高要求。而芯片规模日益增大，时钟频率不断提高，作为电路系统的时间参考，时钟信号在同步电路系统中占据着重要地位。时钟树是描述时钟信号的传播网络，它对于系统功能和性能都是至关重要的。

在集成电路后端设计中时钟树综合的主要任务是达到电路的静态时序要求。随着芯片设计的时钟频率越来越高，时钟结构越来越复杂，时钟树综合是深亚微米芯片后端设计中的一个重要环节。如何使时钟信号按照设定的时钟约束传输到芯片上各个寄存单元,如何在时钟树达到时序要求时尽量减少时钟网络上的缓冲器和倒相器数量以减小时钟网络的功耗开销和面积开销都是时钟树综合时需要考虑的问题。

本文课题的研究方向是基于 Garfield5 SoC（System-on-Chip）芯片设计，使用 Synopsys 公司的 Astro、PrimeTime 等后端设计工具探讨了在深亚微米后端设计流程中时钟树综合和优化技术。Astro 是 Synopsys 公司的集成电路后端设计工具，它集布局、时钟树综合和布线为一体。本文首先介绍了时钟树综合的概念、相关理论和影响时钟树性能的几个重要因素（时钟树源点、时钟周期、时钟树最大延迟和最小延迟、时钟偏差（clock skew）、传递时间（transition time）和缓冲器种类）。然后讨论了减小时钟偏差、调整时钟树延迟以及降低时钟树功耗的方法。并且结合实验室研发的 SoC 芯片 Garfield5，在 SMIC（中芯国际）0.18μmCMOS 工艺下，基于 Astro 物理设计流程，分析了不同设计方案对时钟树性能的影响。

Garfield5 的实验结果表明：结合 Astro 自动时钟树综合流程，采用功耗管理模块（PMC）布局优化、调整门控时钟连线权重和调整时钟源位置的方法后，系统主时钟（CLK5M）的时钟偏差控制在 0.158ns 以内，最长的时钟树延时路径调整到 1.75ns。Garfield5 芯片面积在 5mm×5mm 以内，最高频率达到 100MHz。

关键词： 时钟树综合；时钟偏差；时钟树延时；深亚微米

Abstract

The design methods of the integrated circuit meets more strict requirment according to the development of IC architecture and technology。As the IC architecture becomes more and more complicated and the frequency of clock keeps increasing,, the clock signal which is a universal time reference to synchronous digital circuit plays more and more important role in designs。Clock tree is a transmission network of clock signal, which is essential to the function and performance of system。

In the phase of the back-end design of the intergrated circuit, CTS (Clock Tree Synthesis) is performed to meet the static timing of the circuit。With the higher frequency and the more compicated architecture of the clock, CTS become more and more important in the DSM(Deep Sub-Micron) back-end design。How to make the clock signal transmit and arrive at the registers at the same time, and how to decrease the amount of the inserted buffers and inverters (large number of buffers and inverters increases the power and area consumption) in the clock distribution network are the main point that must be considered in CTS。

This thesis based on the design of Garfield5 SoC (System on Chip), using Astro and PrimeTime which are provided by Synopsys, discusses the flow of auto CTS and manual CTS optimization methods in the DSM back-end design process。Astro integrates the function of floorplan、placement、clock tree synthesis and routing。The concept of the CTS is introduced at first as well as the related theory and some important elements that can affect the performance of the clock tree (such as clock source、clock period、clock tree latency、clock skew、transition time and the type of buffers)。Then the approachs to decrease the clock skew, adjust the clock delay and reduce the power consumption in the clock tree are discussed。With the design of Garfield5, under the SMIC 0.18um technology process, based on the Astro physical design process, the results of different design methods are analyzed.

The experiment based on Garfield5 indicates: with the auto CTS flow using Astro, Power Management Control (PMC) module loaction optimization、adjusting the netweight of the wire and clock source loaction can improve the CTS performance。It maintians the clock skew of CLK5M within 0.18 ns and the longest clock delay path less than 2 ns。The area of the Garfield5 chip is under the limit of 5mm×5mm andt highest frequency attains 100MHz。

Key words: clock tree synthesis; clock skew; clock tree latency; deep sub-micron

东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：汪强 日 期：2006.4.15

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权东南大学研究生院办理。

研究生签名：汪强 导师签名：[illegible] 日 期：

第一章 绪论

1.1 集成电路的物理设计流程

集成电路经历了小规模集成（SSI）、中规模集成（MSI）、大规模集成(LSI)发展到目前的超大规模集成（VLSI）和特大规模集成（ULSI）阶段。同时，集成电路工艺也走向深亚微米，芯片特征尺寸的缩小、设计复杂性的提高、时钟速度的增快、电源电压的降低、布线层数的增加，使得深亚微米下的大规模集成电路的设计复杂度越来越高。而由此引出的一系列新的设计挑战。在集成电路的物理设计中，高速时钟树设计、信号完整性设计、以及低功耗设计等都对设计者提出了更高的要求。集成电路的物理设计就是把电路信息转化成 Foundry 厂可用的掩膜版图信息的过程，它包括数据准备、布局规划、布局、时钟树综合、布线及 DRC、LVS 等步骤。图 1-1 给出了物理设计的一般流程。

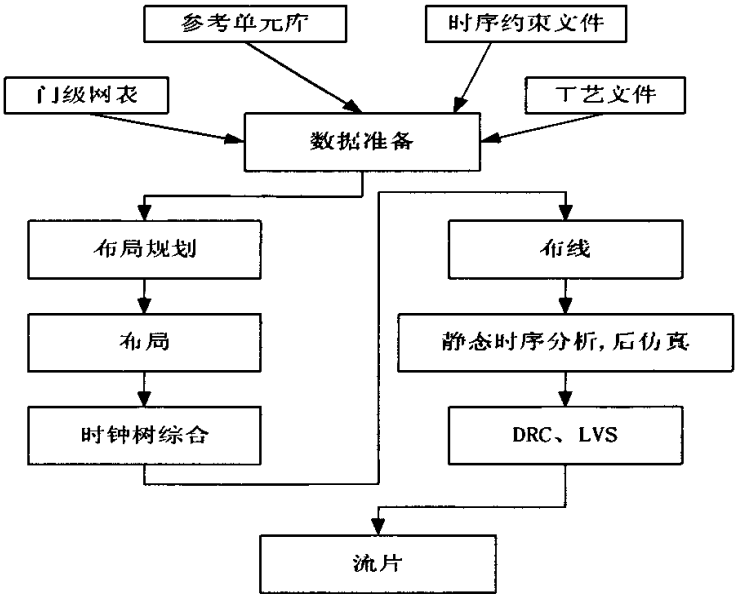

图 1-1 物理设计的一般流程

1.2 课题背景

随着集成电路技术的发展，芯片规模的增大，时钟频率的提高，时钟信号作为同步电路系统的统一时间参考，它对整个芯片系统的速度、功耗、面积等都有重大影响。因此全局时钟信号在芯片系统中占据着重要地位。集成电路的物理设计流程中时钟树综合是保证时钟信号满足设计要求的核心技术，它是影响整个电路系统能否达到时序设计要求的关键一环。对于单时钟域、低频时钟域或简单结构的时钟树设计，可以使用 EDA 工具(如 Synopsys 公司提供的后端设计工具 Astro)进行自动化的时钟树综合，一般都可以达到设计的时序要求。但在高性能的大规模系统芯片设计中，时钟信号有时需要驱动几千甚至上万个寄存器，它的负载很大，边线很长。要保证时钟信号按照时序要求到达各个寄存器，同时又要提高时钟信号传播的效率，并且确保时钟信号的波形满足设计要求，那

么一个高质量的时钟传播网络是必需的。在此类芯片设计中，时钟树综合首先要借助物理设计中的EDA工具在版图中插入大量的时钟缓冲器和倒相器来平衡时钟偏差和时钟延迟，设计者同时需要根据设计的具体要求，分析时钟树结构，在工具自动时钟树综合的基础上做细致的时钟树结构调整和优化工作。

总之，在高性能大规模的系统芯片中时钟频率高、时钟结构复杂、时钟同步要求严格，时钟树综合技术已成为芯片物理设计中的一个难点。

1.3 课题研究的主要内容和论文结构

本文课题研究的主要内容是结合Garfield5 SoC芯片，在Astro中对Garfield5芯片复杂的时钟结构（多时钟域技术、门控时钟技术等）进行分析和时钟树综合。它探讨了复杂时钟结构的时钟树综合流程和优化，探讨了影响时钟树性能的时钟偏差和时钟树延时减小的方法。

第一章介绍了集成电路物理设计流程以及时钟树综合在物理设计流程中的重要性。

第二章介绍了时钟树综合的概念与基本理论。对影响时钟树综合的几个主要因素，如时钟偏差、时钟树延时和时钟传递时间等进行了分析。对几种不同风格的时钟树结构类型进行了比较。

第三章针对Garfield5系统芯片的功耗管理模块，分析了Garfield5中的动态功耗控制策略、具体的时钟产生及时钟结构。

第四章是本文的重点，它结合Garfield5的设计，为达到芯片时序设计的要求，探讨了复杂时钟结构下的时钟树综合和优化方法。其中包括优化功耗管理模块的位置、设置门控时钟单元边线权重及优化时钟源点位置的方法。

第五章提出了总结和展望。

第二章 时钟树基本理论

在大规模集成电路中，大部分时序元件间的数据传输是由同步时钟信号控制的，时钟频率决定了数据处理和数据传输的速度，它是电路性能最主要的标志。当集成工艺进入深亚微米阶段，决定时钟频率的主要因素有两个[1]：同步时钟驱动的元件间的最大时钟偏差（clock skew）和组合逻辑电路的最长路径延时。时钟树结构的生成主要包含时钟网络拓补结构生成、实体嵌入及缓冲器布局等步骤。近年来时钟树结构基于时钟网络规则化的拓补结构，发展了多种不同风格的时钟树类型。常用的规则化拓补结构的时钟树类型有 H 树结构，它适用于规则的阵列电路。目前非规则阵列电路设计中常用的是一种基于 RC 匹配理论的时钟树类型——时钟缓冲器树结构。

2.1 基本概念

2.1.1 时钟与同步时序电路

现代系统芯片中大都采用同步时序电路，即采用周期性的同步信号或时钟。在同步时序电路中有公共的时钟信号，电路中各存储元件受它统一控制，只有在该时钟信号到来时存储元件的状态才能发生变化，从而使时序电路的输出发生变化，而且每来一个时钟信号，存储元件的状态和电路的输出状态才可能改变一次。如果时钟信号没有到来，输入信号的改变不能引起电路输出状态的变化。理想情况下，从时钟源分布到各个寄存器的时钟路径是完全均衡的，那么在系统中不同点处的时钟相位（相对于参照时间的时钟边沿的位置）也是完全相同的。但实际情况是，时钟信号到达芯片各个寄存器的时刻并不完全相同，这样就可能会限制系统的最高性能或者由于时钟锁存数据不对而造成芯片功能错误。图 2-1 表示一个同步时序电路的基本结构。

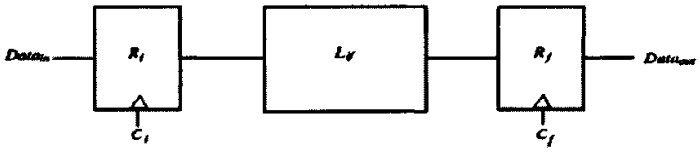

图 2-1 同步时序电路

在同步时序电路中，数据一般被锁存在具有双稳态的寄存器中，当寄存器的时钟信号到来时，数据信号开始离开当前寄存器向后传播，并且要在下一个时钟沿到来前到达下一级寄存器，以保证数据能被正确锁存。同步系统中的延迟元件可以分为以下三部分[2]：

1）存储元件；

2）逻辑元件；

3）时钟电路和时钟分布网络。

这三者之间的相互关系对于取得系统的高性能和高可靠性是十分关键的。一个时钟同步电路有功能逻辑元件和寄存器组成，这些寄存器都是全局时钟控制的。对于任意两个寄存器（Ri，Rj），他们相互之间的关系存在以下两种可能性：一是从 Ri 的输出端只经过组合逻辑无法到达 Rj 的输入端；另一是至少有一条组合逻辑的路径连接 Ri 的输出端和 Rj 的输入端。第一种情况下同一个时钟周期内，Ri 输出端信号的变化不会影响 Rj 输入端的信号，它们之间的时序不相关。而第二种情况下，Ri 输出端的信号会传播到 Rj 的输入端，即将 Ri 寄存的数据赋值给 Rj，此时（Ri，Rj）被称为时序上相邻的寄存器对，它们之间的数据传输也构成了局部数据路径。

图 2-2 所示的是同步时序电路中最基本的同步数据传输的路径形式。时钟信号 C_i 和 C_j 分别驱动时序上相邻的寄存器对 R_i 和 R_j，时钟信号 C_i 和 C_j 都是源于同一个时钟源。我们将时钟信号 C_i 从时钟源传播到第 i 个寄存器 R_i 时钟端的传播延迟记做 T_{Ci}，同理时钟信号 C_j 从时钟源传播到第 j 个寄存器 R_j 时钟端的传播延迟记做 T_{Cj}。当数据从 R_i 寄存器传播到 R_j 寄存器时，T_{Ci} 和 T_{Cj} 作为时间参考。时钟信号的传播延时 T_{Ci} 和 T_{Cj} 都是时钟源经过时钟网络传输到达各个寄存器的时钟端，理想时钟情况下，认为时钟信号是同时到达寄存器时钟端的，即 T_{Ci} 和 T_{Cj} 是相等的，它们之间没有时间差。

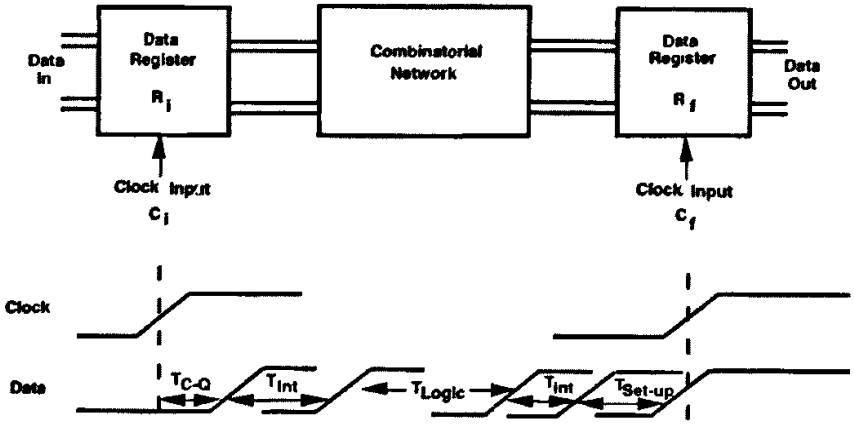

图 2-2 流水线数据通路的电路和时序参数

假设已知下列时序电路的时序参数：

- 寄存器的污染延时或称最小延时（$T_{C\text{-}Q\_cd}$）和最大传播延时（$T_{C\text{-}Q}$）。
- 寄存器的建立时间（T_{su}）和维持时间（T_{hold}）。
- 组合逻辑的污染延时或称最小延时（T_{logic_cd}）和最大延时（T_{logic}）。
- 数据信号穿过互联线所需要的时间 T_{int}。
- 时钟信号 C_i 和 C_j 的上升沿相对于全局参照时钟的位置（分别为 T_{Ci} 和 T_{Cj}）。

在理想情况下，T_{Ci} 和 T_{Cj} 相等。因此这一时序电路要求的最小的时钟周期仅取决于最坏情况的传播延时。周期必须足够长，以便在时钟的下一个上升沿到来之前数据能传播通过 Ri 寄存器和组合逻辑并在目标寄存器 Rf 处建立起来，以满足建立时间的要求。同时，目标寄存器的维持时间必须小于通过逻辑网络的最小传播延时，这两个约束[1]由下面的式子给出：

$$T > T_{C\text{-}Q} + T_{logic} + T_{int} + T_{su} \quad (2\text{-}1)$$

$$T_{hold} < T_{C\text{-}Q\_cd} + T_{logic\_cd} + T_{int} \quad (2\text{-}2)$$

但实际设计的时钟网络是偏离理想情况的。时钟信号既不是理想周期性的，也不是完全同步的，由于工艺和环境等因素的变化，时钟信号在空间和时间上都发生偏差，这都有可能导致电路性能的下降甚至电路功能错误。

2.1.2 时钟树基本概念

（1）时钟偏差（clock skew）

集成电路同步时序电路中一个时钟翻转的到达时间在空间上的差别称为时钟偏差。它是由时钟路径的静态不匹配以及时钟在负载上的差异造成的，引起时钟偏差的主要原因有以下几个方面[3]：

- 从时钟源到各个寄存器的连线长度不同。
- 时钟路径上连线的物理参数不同，例如连线电阻、电介常数、厚度、接触孔和通孔电阻、连线和边缘电容等。

- 时钟网络上插入的时钟缓冲器或倒相器的延迟不同。
- 时钟路径上器件参数的不同，例如 MOS 晶体管的阈值电压、沟道迁移率等，它们都会影响器件的延迟。

时钟偏差有局部时钟偏差（local skew）、全局时钟偏差（global skew）及有用时钟偏差（useful skew）三种类型：

1）局部时钟偏差

时钟信号到达两个时序相邻的寄存器所需的时钟延时的不同，称作局部时钟偏差，记做 T_{skewij}。如果时钟信号 C_i 和 C_j 是完全同步的，即两个信号是精确的同时到达，那么局部时钟偏差为零。也即是给定两个时序上相邻的寄存器 R_i 和 Rf，两个寄存器之间的局部时钟偏差定义为[4]：

$$T_{skewij} = T_{ci} - T_{cj} \tag{2-3}$$

其中 T_{Ci} 和 T_{Cj} 分别是时钟信号从时钟源传播到寄存器 R_i 和 R_j 的时钟延迟。局部时钟偏差定义中要求两个寄存器是时序上相邻的，即两个寄存器之间是构成一个数据通路的。对于由同一个时钟源驱动但时序上不相邻的两个寄存器，从时序分析的角度上看，它们之间的时钟偏差是没有意义的，因为它们间的时钟偏差不会对电路系统的性能产生影响。但是，从分析时钟传播网络的角度上，时钟信号分布是全局性的，因此引入全局时钟偏差作为一个全局性的时间参考是必要的。

2）全局时钟偏差

全局时钟偏差是指同步时钟网络中时钟源点到由它驱动的任意的寄存器间时钟信号到达的最大延时差，它不考虑这些寄存器之间是否存在数据传输关系[7]。图 2-3 分别给出了全局时钟偏差和局部时钟偏差的示意图，为简单起见，在图中一个时钟源信号驱动三个触发器。

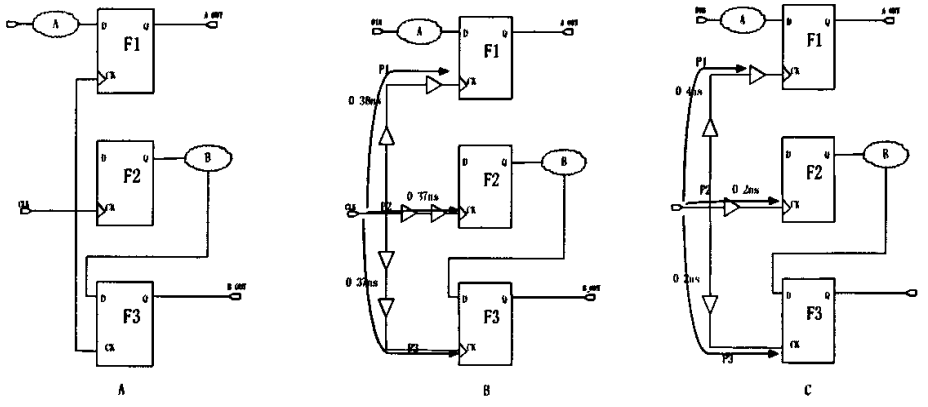

图 2-3 全局时钟偏差和局部时钟偏差

图 A 给出了时钟缓冲器插入之前的时钟网络，图 B 和图 C 分别是考虑全局时钟偏差和局部时钟偏差插入缓冲器后的时钟网络。在图 B 中，考虑到全局时钟偏差进行时钟树综合后，三条时钟路径之间的延时差值很小。它不考虑 F2 和 F3 之间的数据路径。而在图 C 中，触发器 F1 和 F2、F3 没有任何数据传输关系，因此，局部时钟偏差只计算 P2 和 P3 路经上延时的差值，在时钟树插入时也只考虑这两条路径上的时钟偏差。由于触发器 F1 和其它两个触发器没有任何数据传输关系，所以它只要求在一个周期之内将数据从它的 D 端传输到 Q 端，而根本不必考虑 P1 路径上的延时和其它路径上延时的差值。

理论上，在时钟树综合时，考虑局部时钟偏差更有意义。但是从设计方法和设计工具的实现上来看，考虑局部时钟偏差就要考虑数据路径，因此在自动工具做局部时钟偏差的时钟树综合时耗时更多且占用较多的内存资源。而设定系统级的全局时钟偏差，可以方便设计者给出简洁的时钟约束，也可以通过全局时钟偏差的约束对局部时钟偏差的取值范围进行限制。

3）有用时钟偏差

有用时钟偏差是为提高电路性能故意增加的从时钟源点到两寄存器之间的延时时间差异。在时序电路中，我们称限制电路系统性能的时序路径为关键时序路径。关键时序路径上的时钟偏差是一个重要的参考量。利用有用时钟偏差来放松时序约束，以提高电路系统性能是时钟树综合中一个有用的方法[10]。对于关键时序路径，可以设法使起点寄存器时钟信号早于终点寄存器时钟信号到达，这样可以从非关键路径上获取一些多余的时间给关键路径，其效果就是从关键路径延迟中减去时钟偏差的大小，也就是放宽了关键路径完成其功能的总时间要求，这样就减小了最小时钟周期的约束，也减小了各个局部数据路径的延迟差别。简而言之，可以通过有用时钟偏差来调整电路设计中时钟路径划分的不合理。

如图 2-4 所示，横向的椭圆代表组合逻辑电路的延迟，竖向的椭圆代表时钟路径上的延迟。R_2 至 R_3 之间是最长延时的数据路径，可以使 C_3 信号落后 C_2，产生负的时钟偏差来优化改局部数据路径上的时序。如果 C_1 信号与 C_3 同步，那么 R_1 至 R_2 路径上就是正的时钟偏差。我们假设寄存器的延迟为 2ns，为了使得 R_1 至 R_2 的路径和 R_2 至 R_3 具有相同的延迟，那么应该使 C_2 信号超前 C_3 信号 1.5ns，这样 R_1-R_2，R_2-R_3 总延迟均为 7.5ns。使用时钟偏差调度前后系统工作频率就得到提高，这说明合理的有用时钟偏差是对设计有益的。

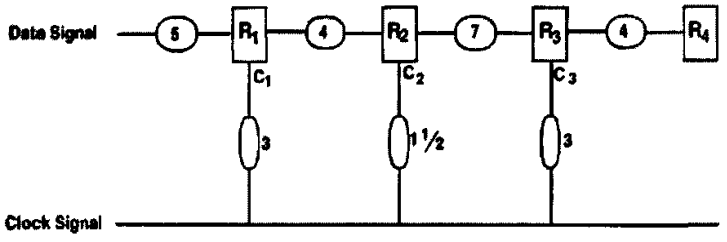

图 2-4　有用时钟偏差的应用示例

但是有用时钟偏差的值是有限制的，它和时钟周期有关，也和寄存器前一级路径的延迟有关。设计中一般都是通过几级串联的局部数据路径构成全局数据路径，其中有些寄存器（如图 2-4 所示的 R_2）是某个局部数据路径的起点寄存器，同时也是某个局部数据路径的终点寄存器，如果采用有用时钟偏差，该寄存器后的数据路径是负的时钟偏差，同时也给该寄存器前的路径带来正的时钟偏差，如果有用时钟偏差的值不合理就可能会带来新的系统性能瓶颈[6]。对于如图 2-4 的简单电路我们很容易得到有效的有用时钟偏差的值，但是对于芯片设计中的某一部分电路来说，有用时钟偏差的值是需要经过大量的复杂的线性方程求解得到的。一般它是用在关键时序模块中的一种提高电路性能的方法。有用时钟偏差主要受两方面的限制[5]：

1）时钟偏差的变动。电源电压的波动、工艺误差、工作温度和环境辐射等都可能使时钟延时发生变化，从而使得时钟偏差的值不能完全固定。这种不确定性就给有用时钟偏差带来难度。

2）全局路径上各个局部路径的延迟。如果整个全局路径都是关键路径，即每个局部数据路径都没有时间余量可利用，那么有用时钟偏差就无法奏效了。

（2）时钟延时（clock latency）

时钟信号从时钟源点传输到时序单元的时钟端口的时间称为时钟延时。在时钟树综合时，时钟偏差和时钟延时都是衡量时钟树性能的重要参数。一般为了要使时钟树的时钟偏差尽量在设计的目标要求以内，可能会适当的增大时钟延时的值以平衡时钟树。大的时钟延时会给时钟树平衡时钟偏差留有更多的空间余量。但是增大时钟延时也即是在时钟树综合中会引入更多的时钟缓冲器和倒相器，从而引起面积和功耗的增大。所以时钟偏差和时钟延时两个参量在时钟树综合中是需要权衡。针对不同的设计，取舍的程度不同。如果设计的时序要求比较容易达到，则可以加紧最长的时钟路径延时约束，以获得一个时钟延时小的时钟树，从而节省了面积和功耗的开销。反言之，如果是时序要求严格或者关键时序模块的时钟树综合，通常会优先考虑时钟偏差的要求。

（3）跳变时间（transition time）

时钟信号的跳变时间是指时钟信号在时序元件的时钟端处，电压从百分之十变化到百分之五十的上升或下降时间。由于时钟信号所负载的元件的充放电不是瞬间完成的，时钟信号从低电平转换到高电平（或从高电平转换到低电平）要经过一定的时间。时钟跳变时间受它所负载的元件驱动能力的影响：原件驱动能力越大，电容充放电时间越小，则时钟跳变延时也变小。

2.1.3 时钟布线方向与数据路径方向的关系

时钟布线方向和数据通过流水线的方向如图 2-5 所示：

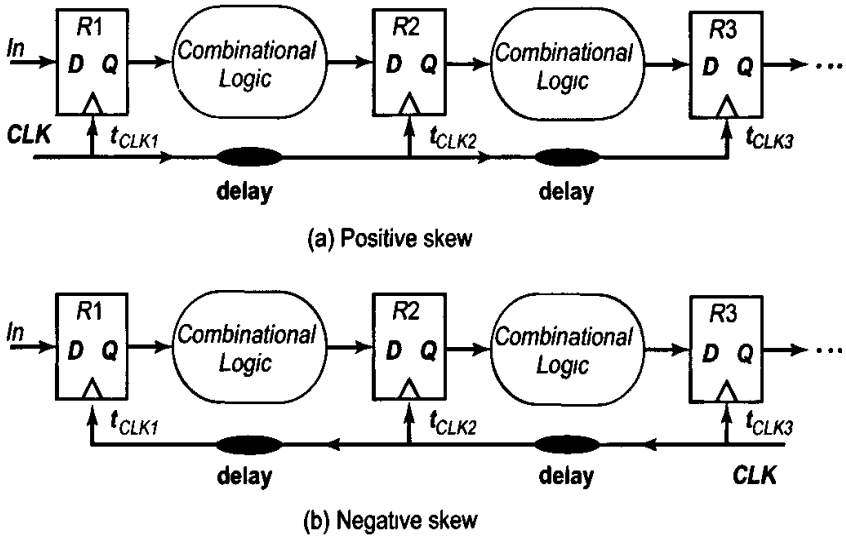

图 2-5 正时钟偏差（positive skew）和负时钟偏差（negative skew）

一般，定义时钟布线方向和数据通过流水线的方向一致为正时钟偏差，即 $T_{skew}>0$。如上图 2-6（a）所示，由 R1 在边沿点 1 处采样的一个新输入 In 将传播过组合逻辑并被 R2 在边沿点 4 处采样。在时钟偏差为正的情况下，那么信号由 R1 传播到 R2 的可用时间就增加了一个时钟偏差值 T_{skew}。组合逻辑的输出必须在 CLK2 上升沿（点 4）的一个建立时间之前有效。

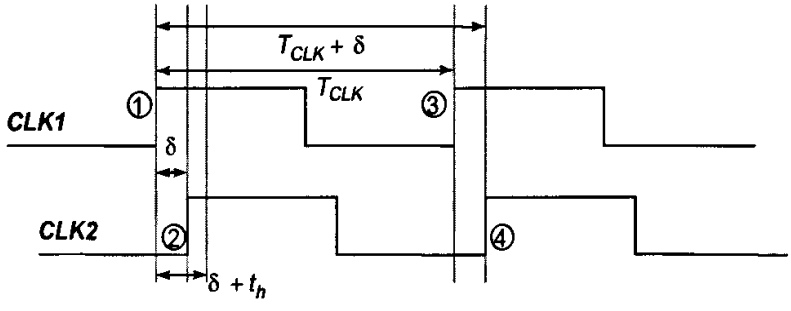

图 2-6 正时钟偏差情况下的时序图

在式（2-1）和（2-2）的基础上，就得到如下的时序约束公式如下[2]：

$$T + T_{skew} > T_{C\text{-}Q} + T_{logic} + T_{int} + T_{su} \quad (2\text{-}4)$$

$$T_{hold} + T_{skew} < T_{C\text{-}Q,cd} + T_{logic,\,cd} + T_{int} \quad (2\text{-}5)$$

由式（2-4）可以得知：正时钟偏差实际上具有改善电路性能的可能。也就是说电路可靠工作所要求的最小时钟周期随着正时钟偏差的增加而减小。但是由式（2-5）又可知，正时钟偏差会使电路

对竞争情况更加敏感，这将可能危及到整个时序系统的正确工作。这说明在满足维持时间的约束式（2-5）前提下，适当的增大正时钟偏差值有利于提高电路的性能，但是这一改进范围是有限的，因为较大的正时钟偏差值很快就会导致式（2-5）的约束违反。同理，将时钟布线与数据通过流水线的方向相反的情况定义为负时钟偏差，即 $T_{skew}<0$。负时钟偏差显著的提高了电路抗竞争能力，如果维持时间是0或者负值，那么竞争可以被消除，因为针对式（2-5）它是永远成立的。但是同时，负时钟偏差降低了可用于进行实际计算的时间，所以时钟周期必须要增大一个$|T_{skew}|$。简言之，负时钟偏差可以避免电路出错，但会降低电路性能。

随着工艺的发展，MOS器件都按等比例缩小，理想情况下，器件的大小和电压都按比例缩小，可以看作都乘以1/S，其中S>1。基于器件的延迟，例如 $T_{C\text{-}Q}$ 、$T_{set\text{-}up}$ 和 T_{logic} 都按1/S比例缩小。但与连线相关的延迟（比如 T_{skew}），它们仍然保持不变，考虑到边缘电容和电迁移，这些延迟反而增大了。因此，当器件尺寸按比例缩小时，系统可靠性不能忽略时钟偏差等带来的影响[18]，使得设计者采用特殊的方法来保证时钟偏差相关的约束得到满足，比如设计尺寸按比例缩小时，保持时钟分布网络连线宽度不缩小，或者更换连线材料等。

2.2 时钟树综合原理

时钟树也称为时钟网络分布，它是指时钟源和时钟所控制的寄存器时钟端之间的一系列的组合逻辑。它的逻辑结构是逐级增大增多的时钟缓冲器和时钟倒相器组成的树状结构，因此被形象的称为时钟树。时钟树综合就是根据设计和单元库的物理信息，在时钟路径上插入一个由时钟缓冲器和倒相器组成的树状结构，使得时钟信号严格按照设计要求到达芯片寄存器的时钟端口。也就是说时钟树综合把时钟信号从理想时钟转换为具有物理和电学特性的时钟信号。在时钟树综合过程中，由几个原则是必须要始终坚持[15]：

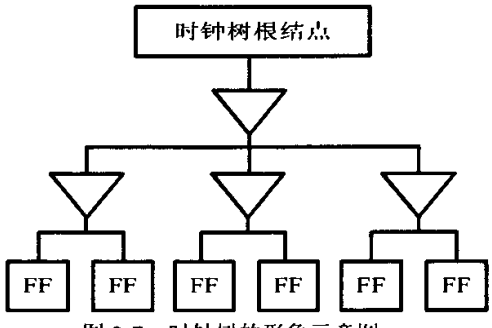

图2-7 时钟树的形象示意图

（1） 如果使用时钟倒相器做时钟信号驱动，一定要保证功能模块中的级数正确，不能把时钟信号极性弄错。

（2） 每个功能模块中时钟信号的最大上升下降跳变时间要明确和保证。

（3） 每个功能模块中的时钟偏差的最大值要明确和保证。

时钟树综合的参考因素除了时钟偏差以外，还有时钟信号的跳变时间和时钟延迟等，一个高质量的时钟树综合是使所有约束得到满足，但实际设计中各个因素之间总存在相互制约关系，使得时钟输综合要权衡的考虑总体性能。时钟树综合的过程，就是把时钟树上的逻辑单元分布在芯片上，并通过时钟连线将他们连接起来，完成时钟结构的物理版图实现。对于同步时序电路的时钟树综合技术，有很多种方法可以采用。将时钟信号从时钟输入端按照时序要求精确的传送到芯片上各个寄存器是非常困难的。而时钟树综合的主要任务就是消除时钟偏差，保证时钟信号同时到达各个功能单元，同时它还要考虑的问题包括：

（1） 时钟信号到各个时钟端口的上升/下降时间的约束；

（2） 时钟信号延时最小化；

（3） 时钟网络功耗最小化；

（4） 时钟网络面积最小化；

（5） 时钟网络可靠性要求，即由芯片加工过程导致的偏差最小化。

时钟树综合在整个后端物理设计流程中（见图 1-1）位于布局之后，信号的总体布线之前。它主要包含以下几个步骤：拓补生成、实体嵌入、缓冲器插布局及变线宽优化等。拓补生成和实体嵌入产生延时均衡的时钟树拓补结构和布线树，缓冲器插布局及变线宽优化主要作为后处理步骤对连线的延时和功耗等性能指标进行优化处理。

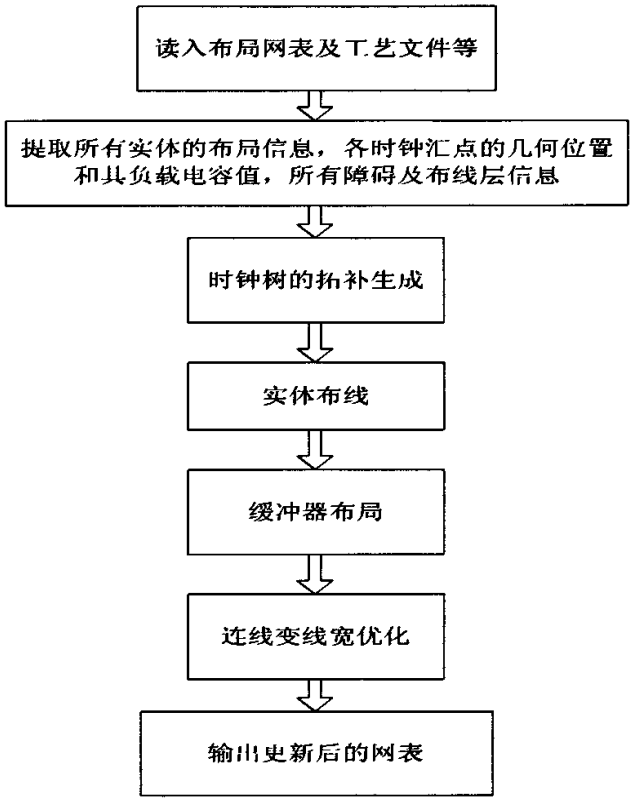

图 2-8 时钟树综合流程

（1） 读取有关布局网表和工艺信息

从网表和库文件中提取所有单元实体的布局信息，得到单元上时钟汇点的几何位置和负载电容信息及所有障碍的布局信息。另外，还需要从工艺文件中读取时钟网布线层数参数等。

（2） 时钟树的拓补生成

时钟树的拓补生成是将给定的各时钟端点按照一定的方式生成一棵树或网状的拓补结构，使得时钟偏差和总连线长度最小化。在拓补生成方法中，主要有两种方式：自顶向下和由底向上。前者从高层入手，可以总体把握要连线的各时钟端点的情况；后者从底层入手，优先考虑距离最近的时钟端点，产生的时钟树线长会比前一种方法明显的减小，但是当时钟端点数目很多的时候，它不容易控制端点之间的偏差约束。传统的时钟树拓补结构生成算法主要目标是平衡互连线长度。例如平均值与中值算法 MMM[24]（Method of Means and Medians）和递归式集合匹配 RGM[27]。但随着集成电路规模的增大和集成工艺的进步，时钟延时并不简单的是互连线长度的线性函数。现在有些改进的算法考虑了时钟延时平衡，例如零时钟偏斜的时钟树 ZST 算法，它是通过延长有较小时钟延时的

互连线来平衡时钟树的。但是零时钟偏斜的时钟树的主要问题是会造成时钟网络的许多交叠。在一层金属上的有最小延时的时钟树是设计者希望得到的，因为它避免了在时钟网络中使用通孔，并且使版图对于工艺变化的容忍度更强，同时避免了其它金属层次上高频干扰。基于零时钟偏斜的时钟树的算法，又发展了“平面时钟树”的研究，如 Max-Min 算法和平面 DME 算法等。

■ 线长平衡的划分法

MMM 算法是一种自顶向下的算法。首先以一条经过重心的垂直划分线将区域分割成两个子区域，并把重心和子区域的重心相连，然后经过子区域重心的水平划分线继续将子区域分割，该过程一直进行下去，划分线的方向是垂直和水平方向交替的，直到当前区域中只有两个时钟端点为止。如果两个子树等延时的连接点在某个不能布线的地方，比如恰好落在第三方硬核处，这种方案就只能产生时钟偏差非零的时钟树分布。在时钟汇点数目较小的情况下，该算法通常能得到很好的效果。但是，当时钟汇点较多时，布线的效果就会很不稳定。如图 2-9 所示：

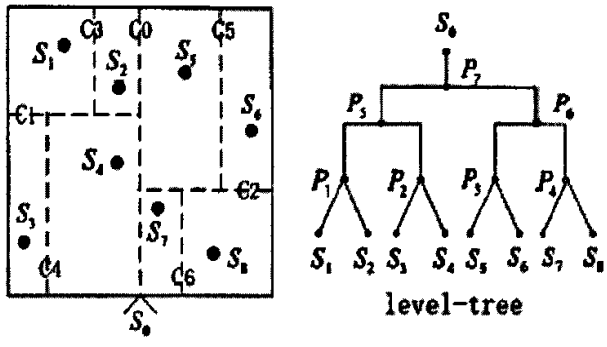

图 2-9 MMM 划分算法

RGM 则是一种由底向上的算法。它首先搜索所有给定的时钟端点之间的距离，找到距离最近的两个时钟端点结成一对，取其中心点为平衡点；并继续在剩下的所有时钟端点中寻找距离最近的时钟端点，连接成对，取其中点，反复进行直到完成所有时钟端点的连接。在此基础上，以所有的平衡点为新的时钟平衡点，进行匹配搜索，将距离最近的时钟端点连接成对，但是其平衡点位置取值为到底层时钟汇点等距离的点，类推到顶层，完成时钟拓补结构的生成。如图 2-10 所示：

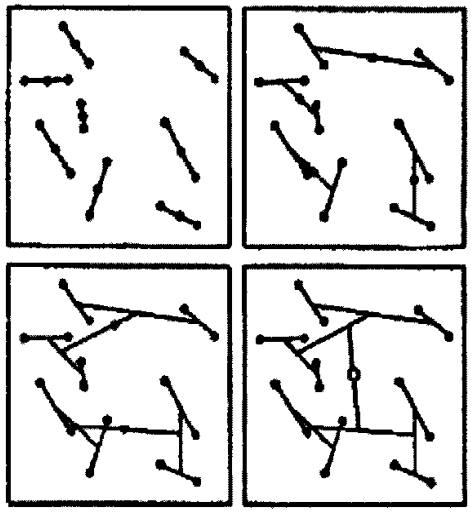

图 2-10 递归式几何匹配算法 RGM 示意图

■ 考虑延时平衡的算法

延时平衡时钟树将准确的用相同的时间将时钟从时钟源点传播到电路中各处寄存器的时钟端点。考虑延时平衡的时钟树也称为零时滞时钟树。它的代表性算法有 Tsay 的精确零时钟偏斜的时钟树拓补结构生成算法[27]。这种算法应用一定的延时模型来计算互连线的 RC 延时，并由此找到两端负载延时平衡点。

■ 平面化的时钟树算法

这类算法以 Max-Min 算法为代表。Max-Min 算法是：给定输入源的位置，根据“最大原则”和“最小原则”获得最短路径的平面布线。如图 2-11，首先找到距离时钟源点最远的时钟端点，并将它与时钟源点相连，它们的距离就是时钟树的最长路径，它们的连线就是时钟树的雏形。然后搜索没有连接到时钟树上的其它时钟端点 s_i，在已有时钟树的每一条边上都能找到一个点 b，使得从此点到 s_i 的路径与此点到时钟树已连接点的路径长度相等。则 b 为 s_i 的其中一个平衡点，d（b，s_i）就是点 b 的平衡距离。所谓最小规则就是当某个时钟端点连接到时钟树时，在所有平衡点中选择使得平衡点距离最小的点和时钟端点建立连线，所谓最大规则就是对于所有剩余的时钟端点，取最小平衡距离最大的那个时钟端点连接到时钟树上。这种算法可以保证时钟树的平面性，且由于最大规则，可以保证连线的平面化，由最小规则可以保证连线长度的最小化。

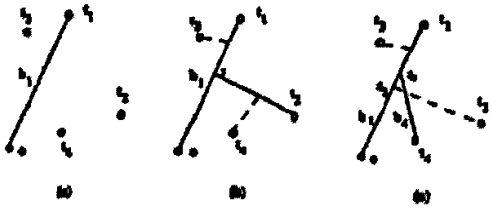

图 2-11 Max-Min 算法示意图

目前常用的时钟树拓补结构生成的算法仍然是线长平衡的划分法和考虑延时平衡的算法。它们的代表时钟树结构类型分别是 H 树结构和时钟缓冲器树结构。这将在下节中具体讨论。

（3） 实体嵌入

给定一个抽象的时钟树拓补结构，还需要对其进行实体嵌入（也称实体布线）。它将决定时钟连线的具体布线，由于时钟网络的特殊性，实体布线不仅要求走线短，还要达到最小时钟偏差约束。VLSI 在深亚微米阶段，用于估计互连线延迟的延迟模型已从简单的 RC 模型发展到复杂的高阶分量匹配模型。对实体布线来说，它需要高的延迟模型相对精度，也即保真度（fidenlity）。保真度是指在某一延迟模型下估算得到的最优解（近似最优解）接近于实际延迟模型下的最优解（近似最优解）。分别对线性模型，分布 RC 模型和分布 RCL 模型来进行互连线延迟的估计，将得到的结果与 SPICE 计算的结果进行比对，结果表明 Elmore 分布 RC 模型对连线延迟估计有较高的保真度[16]。

■ **集总式 RC 模型**

当连线的电阻比较小，并且电路的频率不太高时，可以只考虑连线的电容，这样可以把所有的分布电容集中为一个电容[2]，如图 2-12 所示。可以观察到，在这个模型中，连线是等电位的，连线本身没有任何延时，它对电路的影响仅仅是给驱动的门电路增加了容性负载。

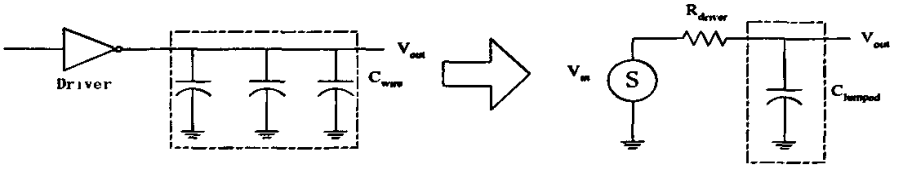

图 2-12 集总式 RC 模型

上图是将驱动器模拟成电压源和电阻，连线上所有的分布电容集总为一个电容 $C_{lumped}=L\times C_{wire}$，L 是连线的长度，$C_{wire}$ 是单位长度的电容，这样，就可以用一个偏微分方程把连线的寄生效应描述出来了。在器件的特征尺寸不是很小时，由于驱动器的电阻远大于连线的电阻，这种模型还是比较准确的，这种情况下，门的切换时间主导着信号穿过互连线的时间，由于忽略了连线电阻，连线上的任一点被认为同时收到信号。但是当集成工艺进入深亚微米阶段，随着器件的特征尺寸的下降，连线电阻不能被忽略，离驱动越远，连线上的延时就会更大。这种计算延时的方法是模型简单，工具处理速度快。但对于深亚微米高速集成电路设计不再适用。

■ **Elmore 模型**

Emlore 延时模型是当前设计中应用比较广泛的一种模型。在图 2-13 中，s 是源点，它提供信号，其它节点是汇点，它们被源点驱动。在这条路径上的电阻称作路径电阻 R_{ii}，比如在 s 和节点 4 间的路径电阻由此展开，定义共同路径上的电阻 R_{ik},它表示从源点到 k 节点和 i 节点共同的电阻，即

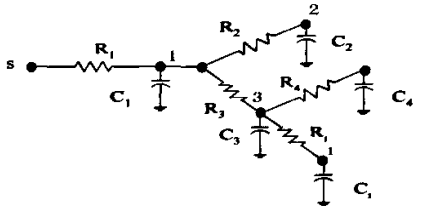

图 2-13 Emlore 延时模型

$$R_{44}=R_1+R_3+R_4 \tag{2-6}$$

$$R_{ik}=\sum Rf \implies (Rf\in[path(s\to i)\cap path(s\to k)]) \tag{2-7}$$

例如：在图 2-15 中 $R_{i4}=R_1+R_3$，而 $R_{i2}=R_1$.

那么，假设现在这个电路网络中的每一个节点都对地放电，并且源点 s 在 t=0 时刻发生变化，那么在节点 i 的 Elmore 延时可由以下方程式表示：

$$\tau_{Di}=\sum_{k=1}^{N} C_kR_{ik} \tag{2-8}$$

Elmore 模型与集总式的 RC 模型相比，它可以计算出每条路径上单个延时，因此计算更加精确。Elmore 延时可以表示为互连集合参数的简单代数函数，对于每一个互连网络可以建模成 π 型电路，每一条边都使用 π 型电阻电容表示，比如对于 2-2 图中节点 1、3 间的电路延时模型可表示如图 2-14 所示，那么 1、3 节点间的延时可表示成：

$$\tau=\frac{1}{2}R_3C_3 \tag{2-9}$$

这样可以对电路网络中的任意两节点进行延时估计。Elmore 模型只计算互连线脉冲响应的一阶分量，忽略了一阶以上的高阶分量。并且只计算电路中的电阻和电容，没有考虑到电感的影响，在对延时估计精度要求不太高的情况下使用 Elmore 模型近似估算非常有效。它的优点是计算简单且有很好的保真特性。对于实体布线来说，Elmore 模型估计下的一个最优和近似最优解非常接近实际延迟下的最优解。也就是说 Elmore 模型下的时钟偏差与实际由 SPICE 模拟得到的时钟偏差有紧密关联。

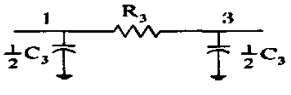

图 2-14 节点 1、3 间的 π 型电路

Astro 中时钟树综合采用的延时模型就是 Elmore 模型。但当时钟频率越来越高而工艺尺寸越来越小的时候，Elmore 模型的结果就很不准确。因为它没有考虑线间耦合，参数频变等非均匀性因素。

■ **AWE 模型**

在延时的计算中，为了得到更为精确的估算值，可以考虑利用高阶分量进行延时估算。但是，高阶分量的计算既费时又费力，计算的复杂度制约着计算分量的阶数，而精度的提高也并不明显，因而一味的提高计算阶数并不是明智之举。AWE（Asymptotic Waveform Evaluation）法即渐进波形估值法是在此基础上发展起来的一种降阶方法。AWE 法的设计思想是采用降阶逼近方法，将互连线作为一个（非）线性的、多端口的宏模型，与现有的电路模拟方法相结合，进行互连线网延时特性模拟。AWE 法是一种估算线性系统响应的一般方法，是用 Pade 逼近将系统脉冲响应用有理多项式表示，从而求出时域响应。它包括计算系统传输函数 Maclaurin 展开式的前 2n 项,然后根据传输函数展开式的截断式构造出一个降阶模型，最后计算降阶多项式截断模型的极点和留数，求出降阶系统的时域响应。基于 AWE 法，后人又做了许多改进，如 Arnoldi 算法[3]。

AWE 模型可以匹配任意阶参数，从而可以得到极为精确的数值解，在高阶匹配算法中其数值解和 SPICE 结果相差无几。并且 AWE 模型适用于各种物理模型，使用广泛，对深亚微米互连线的一些新特性做了考虑。AWE 模型也是 Astro 计算互连线延时一种模型，Astro 使用 AWE 进行延计算时，不但考虑了电阻、电容参数，而且考虑了电感参数。

图 2-15 是 Astro 使用 Elmore 模型和 AWE 模型进行互连线延时计算的对比。我们可以从图中观察到，AWE 模型考虑了各种互连寄生效应，并且 AWE 模型进行高阶计算，所以对互连延时的计算非常准确，由于利用 AWE 模型计算延时时间长且占用内存大，所以 Synopsys 推荐在布线后使用 AWE 模型进行互连线延时计算。

本质上，精确时序建模最基本的困难是无法反映有源器件对延迟的影响，例如模型中无法准确反映衬底负载和波形对时钟缓冲器延迟的影响。因此，目前自动时钟布线研究的重点还是在设法减小连线长度和通孔及交叉数目，以及如何有效控制时钟偏差。

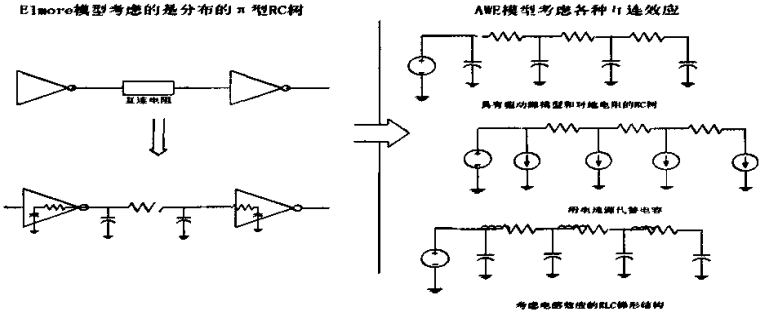

图 2-15 延时计算时采用的 Elmore 模型和 AWE 模型

（4） 缓冲器布局

一般缓冲器布局作为时钟拓补结构生成的后处理步骤。由于缓冲器对负载电容的去耦合效应，在时钟网络中适当的插入缓冲器，可以有效的优化连线延时。特别是时对于大规模的时钟网络，可以显著的改善时钟延时从而降低功耗。缓冲器布局作为后处理步骤也有一些不足之处：

- 在拓补生成及布线过程中未将缓冲器及实际布线环境的影响考虑进去，导致了布线在一定程度上的盲目性，使得时钟树的延时增加。
- 每次时钟布线中层次式插入缓冲器后，都需要修改原布局网表（时钟汇点的位置可能发生偏移），分阶段的修改时钟网表有时会破坏已生成的时钟网。随着插入缓冲器的数量的增多，计算效率也会受到影响。
- 层次式插入缓冲器后，通常需要加入额外的连线来平衡缓冲器的负载电容和延迟，可能造成总电容和功耗的增加。

目前已有一些算法可将时钟拓补结构生成，实体嵌入和缓冲器布局同时进行考虑。但还没有成熟稳定的时钟树自动综合工具采用这些算法来实现。

（4） 变线宽优化

变线宽优化也是一种优化延时的后处理步骤。通过选择合适的连线进行优化取得连线延时的减小和平衡。根据其延时估算的布线模型，调整时钟网互连的宽度从而改变线网分支上的电学参数（电阻电容及电感等），同时还可以增加时钟网络的可靠性，降低延时和时钟偏差的灵敏度。深亚微米的设计中，必须要考虑连线的分布特性。总连线电容的最小化不再意味这互连线延迟的最小化。当互连线的电阻和电容对集成电路的性能起决定作用的时候，互连线变线宽优化是很重要的。目前时钟树综合工具大都是基于平衡均匀线宽的连线来达到时钟布线目标的。等宽的时钟布线虽然设计复杂的程度低，但是它有以下一些缺点：

- 由于是递归的在两棵子树底根节点之间底连线上寻求合并点，但这种方案只适用于构造二叉树结构。
- 递归底平衡方案较难推广到更准确和复杂底延迟模型。
- 时钟偏差与时钟布线的紧密耦合使得布线缺少灵活性。

对于加载多级缓冲器的时钟树来说，由于缓冲器的取耦合作用使得各缓冲器驱动的子树负载电容较小，因此进行延迟优化是各子树所需的最大线宽也相应的减小。对于缓冲器所驱动的各子树来说，从根节点到其子孙节点之间的连线宽度一般呈单调递减趋势，靠近根节点处的连线尺寸应较宽以较少延迟和电流密度，而靠近叶子节点的连线尺寸应较小以有利于减小电容负载效应和降低功耗。

（5） 修改网表

时钟网络中插入缓冲器后，缓冲器作为一种单元，需要加入到原布局网表中。同时，插入的缓冲器可能改变了某些模块的布局位置，因此需要修改网表来反映当前的布局情况。工具会自动完成这一项的工作。但当插入的缓冲器数目增多的时候，修改网表耗用的时间增加。

2.3 时钟树结构类型

最早的一种拓补结构是 H 树结构。H 树是基于上述的 MMM 算法而实现的，它适用于特殊情况的布线方式，要求所有的时钟汇点以一种对称的形式排列。它一般只适用于规则的阵列电路结构。而实际电路设计中大部分的电路都是不规则的，所以实际电路设计最常用的时钟树结构是时钟缓冲器树状结构，也称为匹配的 RC 树。它采用一定的延时模型计算连线延时，使得传送时钟信号至子功能块的互连线具有相同的延时，也即上述的考虑延时平衡的算法。另外还有一种网格结构。

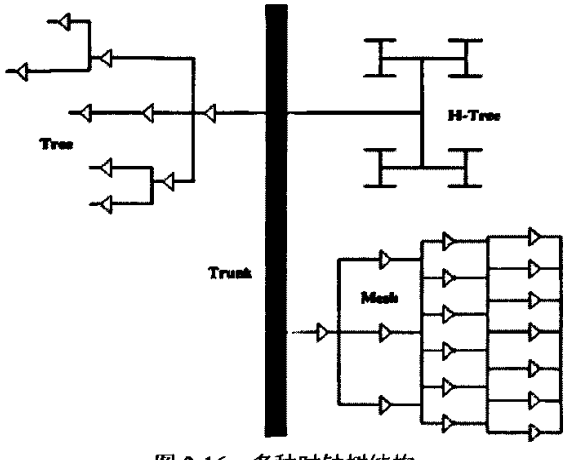

图 2-16 多种时钟树结构

2.2.1 H 树结构

时钟树分布的初始设计目的是为了保证时钟信号同时到达寄存器时钟端，即零时钟偏差的概念。假设把时钟源连到芯片的中心点，且时钟源到每个寄存器的连线和缓冲器完全一样，也即是时钟信号分布到每一个叶节点的路径绝对均衡，那么就得到了零时钟偏差的时钟树。H 树结构就是基于这个想法产生的。第一级的时钟驱动器连接到第一级 H 树的中心，时钟信号从中心送往 H 树的四角，这四个时钟信号再作为下一级 H 树的中心向四方角传播，依此类推，经过多级逐渐减小的 H 树，时钟信号到达各个寄存器的时钟端。这样，每条时钟分支路径上的延迟都保证是基本相同的，小的差异源于工艺的不均匀，各条时钟路径上的工艺参数不一样（比如连线阻抗不同）就会造成延迟有差异[8]。此时时钟偏差的大小取决于连线和缓冲器的物理大小、半导体工艺的均匀性。为了在 H 树结构正常传播信号，避免信号变形，经过证明 H 树上任一级别连线阻抗必须是前一级阻抗的二倍[12]。通常采用缩小连线宽度的方法来实现，因此 H 树的连线宽度是随着时钟传播路径逐级缩小的。

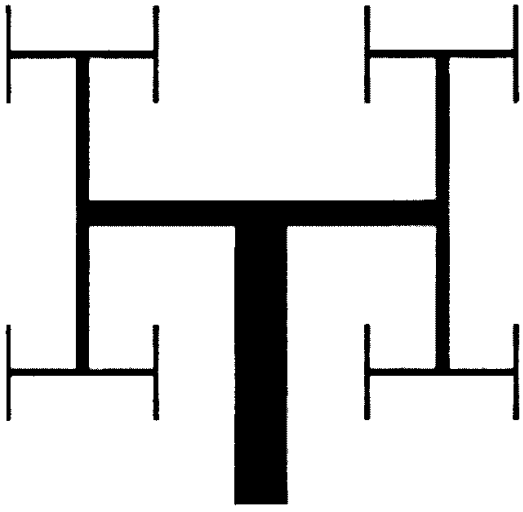

图 2-17 H-树结构示意图

平面 H 树给超大规模集成电路的设计方法和物理实现都带来了新的要求。具体来说，对于 H 树的时钟分布，时钟连线必须同时采用水平和垂直两个方向的布线。如果对于两层金属的 CMOS 工艺，这种曼哈顿结构就带来了布线上的困难，时钟连线不得不使用多个高阻值的通孔。这也是现代半导体工艺金属层次越来越多的一个重要原因。H 树结构和时钟缓冲器树结构相比，连线长度要长的多，因此电容也就要大的多，这样时钟树功耗和时钟延迟就会很大。这就造成了高速电路系统中时钟偏差和时钟树延迟之间的合理权衡的问题。

所以类似于 H 树这样的对称结构虽然能很好的控制时钟偏差，但是与此同时也引入了很大时钟延迟，选用对称结构的时钟分布时一定要考虑时钟延迟的影响。由于时钟偏差只影响到时序上相邻的寄存器，因此对称结构带来的好处是有限的[11]。对于结构十分不规则的设计，实现 H 树这样的对称时钟分布非常困难。结构不规则的设计最好采用时钟缓冲器的树状结构，既能减小时钟延迟，也能利用有用时钟偏差来优化系统性能，也易于消除不规则的版图结构对时钟带来的影响。因此非对称的时钟缓冲器树状结构更为常用，但 H 树的对称结构时钟分布在一些设计中也会用作局部的时钟分布方案。

2.2.2 时钟缓冲器树结构

在实际设计中的不规则电路，典型的时钟分布结构还是树状分布结构[13]。

如果时钟源的连线电阻大小和缓冲器的输出电阻相近，就可以只使用一个时钟缓冲器来驱动整个时钟树。这种方案比较适合整个时钟树结构都是金属分布的（即时钟路径上没有时钟有源器件所组成的组合逻辑等），连线的负载易于平衡。当只采用一个时钟缓冲器来驱动整个时钟网络时，首要的问题是确保缓冲器能够提供足够的驱动电流。只要缓冲器的输出电阻远大于连线电阻，输出电流足够大，就能获得良好的时钟波形，满足时序要求。但对于深亚微米超大规模电路，缓冲器的输出电阻远大于连线电阻的条件很难满足，更为常用的方法是在时钟分布网络中散布时钟缓冲器[11]，如图 2-18 所示：

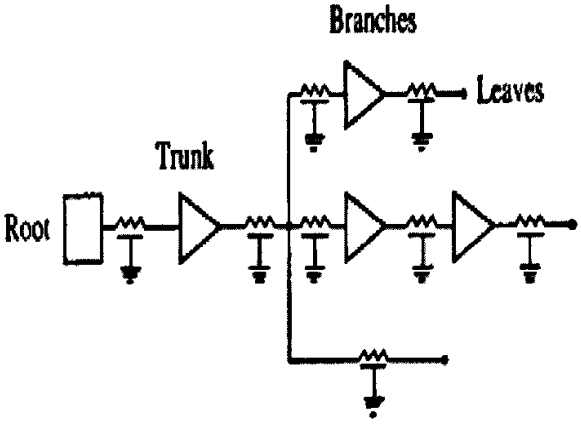

图 2-18　典型的非对称时钟树结构

这种结构虽然会增大芯片面积，但它能很好的控制时钟延迟，较为精确的保证时钟波形。因为这些缓冲器一方面作为中继元件，保证时钟信号在传播过程中不衰减；另一方面，时钟缓冲器将时钟连线的阻抗切割开来，起到隔离时钟连线阻抗的作用。一个时钟缓冲器的树状结构，时钟源和寄存器之间的缓冲器级别的多少取决于时钟汇点总的负载大小，包括寄存器和连线负载。对于一个较为平衡的时钟树来说，时钟偏差主要来自缓冲器，因为与连线等无源器件相比，缓冲器等有源器件更容易受工艺和环境参数的影响，延迟特性发生波动。一个缓冲器所能负载的缓冲器数目取决于自身所能提供的电流和所负载的连线和缓冲器电容，最后一级缓冲器的负载是寄存器时钟端输入电容和连线电容。但在实际的情况下，时钟树上各个时钟分支路径上由于连线阻抗和电容负载的不稳定，造成了时钟偏差的值难于控制。为了提高系统性能和可靠性，需要进行时钟偏差的调整，也即针对不同的时钟路径，采用合理的正时钟偏差或负时钟偏差值调整时序，从而减小各条路径之间的差异。如前所述，H 树结构是在整个芯片上进行的，也即是它是在打平了的全芯片上做时钟树的分布。它的一个显著的局限性在于它不支持整个芯片按照功能层次划分为小模块，即不支持层次化的设计方法。在芯片中某些模块的时序约束特别严格的情况下（比如系统芯片的处理器核），它就无法对局部时钟路径进行优化。

而典型的时钟缓冲器树结构中，是可以采用有效的时钟补偿方法来优化关键的局部电路的。如图 2-18 所示，不同的分支有不同数目的时钟缓冲器，他们驱动不同的 RC 阻抗。当时钟缓冲器的位置使得缓冲器的输出电阻和它所负载的阻抗值相当，那么这就可以将时钟路径上的 RC 互联网络简化为电容性阻抗的精确建模，一般情况下，仍然将时钟缓冲器树看作是 RC 阻抗分布网络。如果全局时钟分布网络的连线电阻相对较小，那么使用在芯片级别中央放置一个缓冲电路就可以满足整个系统时钟同步的要求。但是，在大多数超大规模集成电路系统中，时钟连线电阻、通孔电阻和庞大

的负载电容，形成了一个极大的阻抗。因此，即使在芯片中央放置时钟产生和分布电路，也需要特殊的技术来补偿连线和寄存器的波动带来的时钟偏差。各个时钟分支的差异源于不同线长和不同叶结点负载。常用的补偿技术有：

（1） 使用无源 RC 延迟元件和改变晶体管宽长比；

（2） 为了尽量减少各个时钟路径之间的差异，时钟缓冲器的放置应该使高电阻连线负载低电容，使低电阻连线负载高电容，各个逻辑块之间的时钟差异可以使用参数可配置的时钟缓冲器来补偿。

在时钟缓冲器树结构中使用这些时钟补偿技术的优点是有利于控制整个芯片的时钟偏差以及减小了时钟源到寄存器的时钟延迟。这是因为时钟树上的阻抗被分割开了，各功能块内部的连线电阻相对于模块内缓冲器的输出电阻要小的多，而模块之间的长时钟连线的负载很小，这样 RC 常数减小了，时钟延时就明显减小了。使用模块内补偿技术的另外一个好处是对于规模很大的芯片，可以把整个芯片划分成利于时钟树综合的多个功能模块，使工作人员可以并行的进行时钟树综合，而同时又能保证时钟分布网络的最优化。时钟树补偿技术依赖于器件和连线阻抗的参数调节。如果能对阻抗精确的估计，那么参数化的时钟缓冲器就能支持精确的时钟偏差调整。但是，它的局限性性十分明显：因为晶体管的电导对工艺、电源电压和环境参数等十分敏感，如果时钟延迟主要受晶体管的影响，那么时钟树补偿技术就很难达到预期的效果。

2.2.3 网格结构

网格结构一般用在时钟网络的最后一级。它把时钟直接分布到时钟控制元件的端口上。它的每一个格点上都可以获得时钟。网格结构的原理与时钟缓冲器树结构的匹配 RC 原理不同。它们的主要差别在于网格结构的最后一级驱动器到每一个负载的延时并不匹配。但是只要网格的尺寸尽量的小，那么它的绝对延时也就减小到最小，从而得到平衡的时钟树结构。

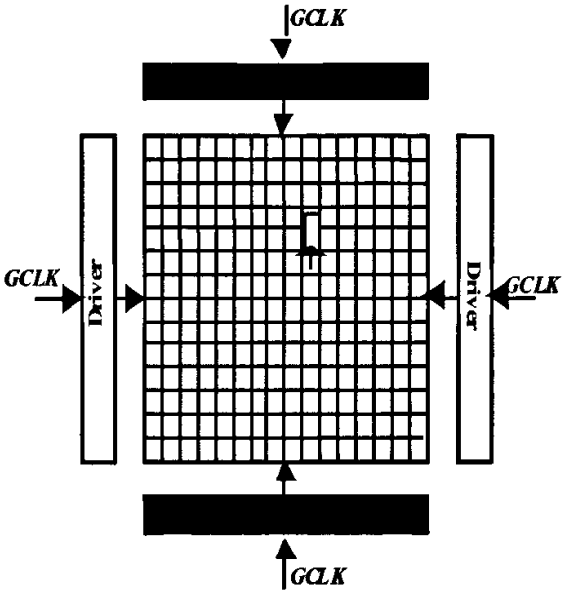

图 2-19 网格结构示意图

它的优点是允许在设计的后期进行时钟树的改动，因为在芯片上的各个点都很容易得到时钟。但网格结构的“冗余”连线很多，它的功耗损失相对比较大。

2.4 本章小结

本章首先介绍了同步时序电路概念及时钟树的时钟偏差和时钟树延时概念。然后重点介绍了时钟树综合的主要步骤：时钟树的拓补结构生成、实体嵌入及缓冲器布局。最后介绍了几种不同风格的时钟树结构，并在此基础上对典型的H树结构和时钟缓冲器树结构做了简单的介绍和优缺点对比。

第三章 Garfield5 时钟管理策略

动态时钟管理技术是目前很多系统芯片中应用到的时钟功耗的管理方案。在同步电路系统中有很大一部分功耗是消耗在时钟树上的，这是由两个方面的原因引起的[20]：

（1） 时钟连线很长，而且负载很大。

（2） 时钟连线的信号翻转很频繁。

而随着工艺尺寸的不断缩小，设计越来越复杂，工作频率越来越快，时钟分布网络上的功耗比重也越来越大。因此，对于系统电路来说，一个高质量的时钟树不仅要有好的性能，同时也要尽量减小时钟网络上的功耗开销。

3.1 Garfield5 基本架构

Garfield5 是由东南大学国家专用集成电路系统工程技术研究中心设计系统芯片，它面向工业控制、多媒体应用及其它消费类电子。它具有低功耗低成本的优点。它采用 SMIC0.18um 标准 CMOS 的工艺设计，内嵌 16/32 位 RISC 微处理器内核和 16 位定点高性能 DSP。它支持信号处理类应用，支持 WINCE/LINUX/ASIX 等嵌入式操作系统和支持 JTAG ICE 标准调试手段。Garfield 双核平台采用 AMBA 2.0 作为其骨干总线，所有外设都以 AMBA 接口集成。Garfield 双核平台还集成了 SRAM/SDRAM/NAND FLASH/NOR FALSH 控制器，LCDC（液晶控制器）、MMC（multimedia card）、MMA（multimedia accelerator）、PMC（power management controller）和数字锁相环（DPLL）等模块。它的基本架构如图 3-1 所示。

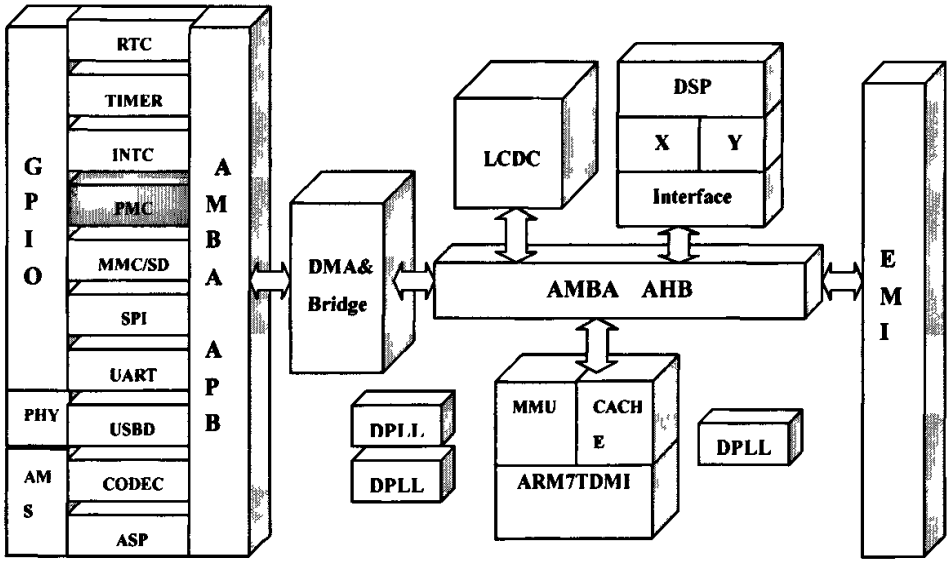

图 3-1 Garfield5 系统芯片的基本架构示意图

3.2 时钟网络与功耗的关联

如前所述，时钟树综合中引入的大量时钟缓冲器和倒相器是现代处理器和 SoC 动态功耗增加的重要原因。时钟网络的功耗在整个芯片的功耗中所占的比重很大。实际的芯片设计中，例如在 DEC Alpha 21164 微处理器芯片，它的总功耗为 50w，其中时钟分布网络的功耗达 20W，占到了 40%[6]。同样的在 Motorola MCORE 的 RISC 微处理器中，时钟分布网络消耗了总功耗的 36%[15]。因此时钟树的设计不但对于性能来说十分重要，对于功耗优化也至关重要。在时钟树综合中可采用的低功耗方案有[9]：

（1） 采用多电源电压降低功耗。时钟网络的输入电压通过低电压缓冲器降低电压，从而使得低电源电压在电路中传播。所以将缓冲器插入到时钟树中，可以保证电路正常工作的速度和电平转换速度。但插入缓冲器和电源转换元件带来新的功耗，需要和减少的功耗之间进行平衡。

（2） 采用减小时钟信号摆幅的办法。这种电路需要设计减少时钟信号摆幅的驱动器，也需要设计专门的接收电路。低摆幅方案的功耗优化效果略差，为了达到同样的速度，低摆幅方案往往需要更多的中继单元。

（3） 时钟门控技术。时钟门控技术是指系统电路中，部分电路空闲时可以将其时钟信号可关闭，从而减小时钟功耗的开销。电路空闲的状态是指，在一个或多个时钟周期内，该部分电路不作任何有意义的计算工作。

时钟门控可以很明显的减少时钟连线和电路中的信号翻转，它被看作在逻辑结构级最有效的动态功耗优化方案。只要能计算出时钟的空闲条件，就可以在时钟树结构中插入门控逻辑单元。基于行为综合的方法来确定设计中各个模块的时钟空闲条件，给定设计的预布局描述，可以通过模块的时间进度表来提取各个模块的激活和空闲时间。其结果是一串 0 或者 1 的集合，分别代表各个模块是激活还是空闲。把具有相同信号活动性的模块集中到一个子树，这样时钟门控单元就可以尽量的接近时钟根结点。这种算法所得出的各个子树被翻译成布局约束，要求同一子树中的模块都尽量靠近。这样时钟就可以使用 H 树布线方法来完成时钟树分布。目前有些 EDA 综合工具也具有一定的在不同设计抽象层次实现自动时钟门控的能力。

但是，时钟门控对时钟功耗也会有反面的影响[9]。为了减少时钟门控单元带来的功耗和面积的增加，很多个触发器应该尽量共用门控单元。但是如果共享同一时钟控制单元的触发器散布在芯片的各处，由于每个时钟域都是单独布线的，这样就会增加很多的连线。这样的话，每个时钟门控单元的负载会很大，尽管信号翻转活动减少了，但功耗还是增加了。因此，插入时钟门控和构造时钟树不是两个独立的过程，需要把它们结合起来。带锁存器的门控时钟结构如图 3-2 所示：

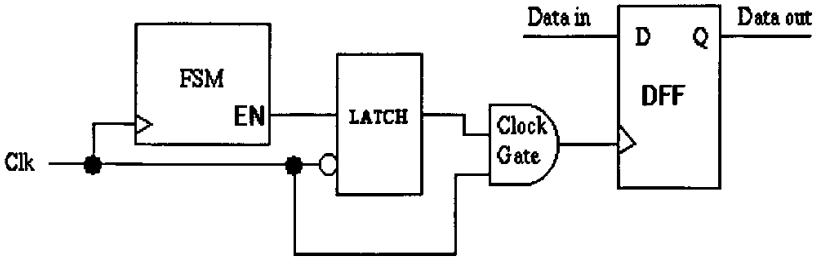

图 3-2 带锁存器的门控时钟结构图

3.3 Garfield5 功耗管理模块

Garfield5 的功耗管理模块是基于动态功耗管理的原理来实现的。动态功耗管理适用于以下两种情况：在一些应用中电路可能需要很高的工作频率，而在其它的一些应用中却可以大大降低工作频率；在一些应用中可能需要用到所有的硬件资源，但是在其它的一些应用中可能只需要用到电路中一部分硬件资源。动态管理 SoC 系统的功耗方案是以不牺牲系统的性能为前提，时钟频率是影响动态功耗的重要因素[21]。它的工作频率越高，功耗也就越大。但在很多时候，所有的模块并不是工作在同一时钟频率，或者同一个模块在不同的时段可以工作在不同的时钟频率。这些就是动态的配置 SoC 系统的时钟频率的前提。动态的管理 SoC 内部模块的时钟源供给，则是根据不同的应用，管理 SoC 内部的硬件资源。简而言之，就是进行内部模块的开和关操作。关闭单个模块，可以通过对每个模块设置一个使能位，然后对这个使能位编程做到关闭或打开那个模块。但这种做法不是最佳的，原因有二：其一，每个模块的接口部分必须是始终打开的，否则，MCU 无法随时对它的内部寄存器进行编程；其二，通过模块使能位只是关闭了它的功能操作，而并没有把它模块内的时钟树关闭掉，也就是说它里面的时钟树依然处于激活状态，而时钟树所造成的功耗占单个模块功耗的很大一部分。其实大多数模块都是同步系统，系统的所有操作都是在时钟信号的节拍下进行的[29]，关闭时钟源就能同时达到关闭模块和降低功耗的目的。

PMC 模块就是根据 Garfield5 的实际应用情况动态管理系统的时钟，不仅动态的配置系统的时钟频率，还可以动态的管理内部模块的时钟源供给。PMC 模块共包括 3 个子模块：控制状态转换和时钟源选择的 pm_ctrl 模块、与 APB 总线相连的 interface 模块、控制重启和唤醒机制的 reset 模块。它具有以下一些特性：

- 时钟源采用双晶振（32.768k/24.576MHz），其中 32.768KHZ 给 RTC 和 RESET 模块，用于日历计数；24.576MHZ 作为芯片的主时钟源。
- 采用三个 DPLL，USBD 和 CODEC 分别使用专用的 DPLL 进行时钟的产生以及控制；另一个 DPLL 用于产生芯片其它模块的高速时钟。
- 提供四种低功耗模式：slow、normal、idle 和 sleep。
  - SLOW 模式不打开 DPLL，芯片工作频率为 24.576MHZ/n，其中 n=1/2/4/6/8，其中缺省为 24.576MHZ/8；
  - NORMAL 工作在 DPLL 的时钟之下，此时 DPLL 的输入参考时钟为 24.576MHZ/6 或者 24.576MHZ/8；
  - 可以从 NORMAL 和 SLOW 进入 IDLE，IDLE 时只关闭 CORE 时钟；
  - SLEEP 关闭所有时钟，SLEEP 时的功耗<150uW，支持 SDRAM 的数据自刷新；
- 系统中模块工作时钟的集中控制策略，系统中的模块可以通过 PMC 来关闭和打开。
- 具有完整 reset 解决方案，提供 3 种 reset 选择：Hardware Reset/Software Reset/Watchdog Reset。

3.3.1 Pm_ctrl 子模块

Pm_ctrl 模块是 PMC 的核心模块，它掌控着系统工作模式的转换和各模块时钟源的选择以及打开与关断。下面分别讨论它的工作模式的转换和时钟源的控制情况：

（1） 工作模式

功耗管理模块中的工作模式状态机完成各种模式之间的切换和送出 PLL 的控制信号。为了降低 Garfield 的整体功耗，PMC 开发了低功耗管理机制的四种工作模式：Slow、Normal、Idle 和 Sleep。下面结合图 3-3 所示的工作模式流程图来说明它的工作机制。

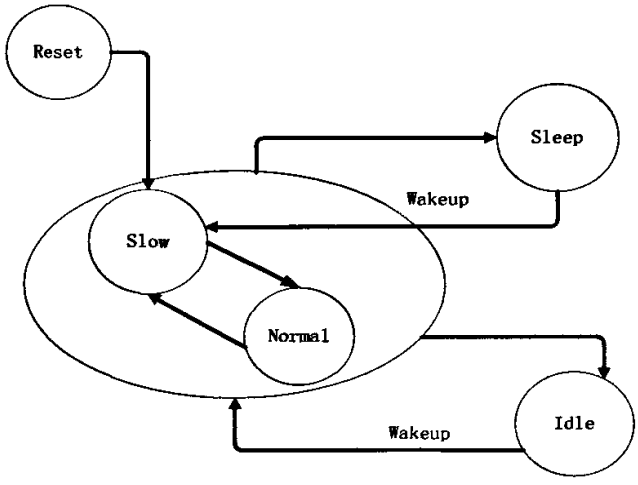

图 3-3 工作模式的流程图

Slow 模式

当系统复位以后或当系统关掉 PLL 不需要高速时钟运行时，系统进入到 Slow 模式。在 Slow 模式下，系统中的 MCU 核和所有的模块的时钟源都来自晶振。如果这个时候系统觉得有必要关掉某些模块，那么就可以通过配置功耗管理模块内部的寄存器，把相应模块的时钟源使能位关掉就可以了。在 Slow 模式下，对 PLL 的任何配置将被认为是误触发事件，要尽量避免。

Normal 模式

如果在某些应用中需要高速时钟，那么就应该切换到 Normal 模式。在 Normal 模式下，系统中的 MCU 核和所有模块的时钟源都来自 PLL。当然在这种模式下也可以根据系统的应用关掉某些模块。如果系统需要调整时钟的频率，可以通过动态配置 PLL 来实现。但是在动态配置 PLL 过程中，要注意这样一个问题：因为 PLL 有一个时钟锁定的时间，在这段时间内，它输出的时钟波形是不规则的，此时不能使用它作为芯片的时钟源。为了保证系统的正常运行，可以暂时把系统的时钟源切换到晶振时钟，待 PLL 的时钟输出稳定以后再把系统的时钟源切换到 PLL 时钟。硬件需要提供一个功能：只有在 Normal 模式下，重新配置 PLL 才认为是有效请求；其余情况下都认为是误触发事件。

Idle 模式

如果 MCU 核在当前状态下已经处理完所有任务，在很长一段时间内都将处于空闲状态，那么系统应该进入到 Idle 模式。在 Idle 模式下，只会关闭 MCU 核的时钟源，而所有的模块都保持原状。但在这种模式下，不可动态配置 PLL，以得到不同的时钟频率；也不可以动态地管理各模块的时钟源，因为这个时候 MCU 核已经休眠了，没办法对功耗管理模块内部的寄存器进行配置。无论前一个状态是 Slow 模式还是 Normal 模式，系统都可以进入到 Idle 模式；而当系统退出 Idle 模式时，它应该退回到前一个工作模式。当系统重新需要 MCU 核进行事务处理时，可以通过一个唤醒信号让系统退回到 Slow 模式或 Normal 模式。

Sleep 模式

如果整个系统都已经处理完所有的事务，并且在很长的一段时间内都将处于空闲状态，那么系统应该进入到 Sleep 模式。在 Sleep 模式下，关闭 MCU 核和所有模块的时钟源。虽然可以从 Slow 模式或 Normal 模式切换到 Sleep 模式，但是当它退出 Sleep 模式时，系统只能回到 Slow 模式。为了进一步降低整个芯片的功耗，在 Sleep 模式时会同时关闭 PLL，所以在它退出时只能回到 Slow 模式，然后根据当前的应用决定有没有再切换到 Normal 模式的必要。当系统需要再次进行事务处理时，可以通过一个唤醒信号唤醒整个 SoC 芯片系统。

（2） 时钟源控制

PMC 中的时钟源控制由一些多路选择器和一些门控时钟电路组成，多路选择器主要完成各种时钟源之间的选择，而门控时钟电路则完成 A720T 核和各模块时钟源的打开和关闭功能。图 3-4 就是功耗管理模块中时钟源控制框图。图 3-3 的时钟源控制的示意图中可知，在功耗管理模块中三个 PLL：一个是主 PLL（MPLL），它提供 Garfield5 中除 USB 模块和 CODEC 模块以外的所有模块的时钟源；另两个是次 PLL（UPLL 和 CPLL），它们只分别对 USB 和 CODEC 提供时钟源，图中只给出了 UPLL 的示例。选择器完成晶振时钟和 PLL 时钟的选择，被选中的时钟（out_clockSource）同时送到各个模块中，然后根据各个模块的需要门控的送出时钟源。

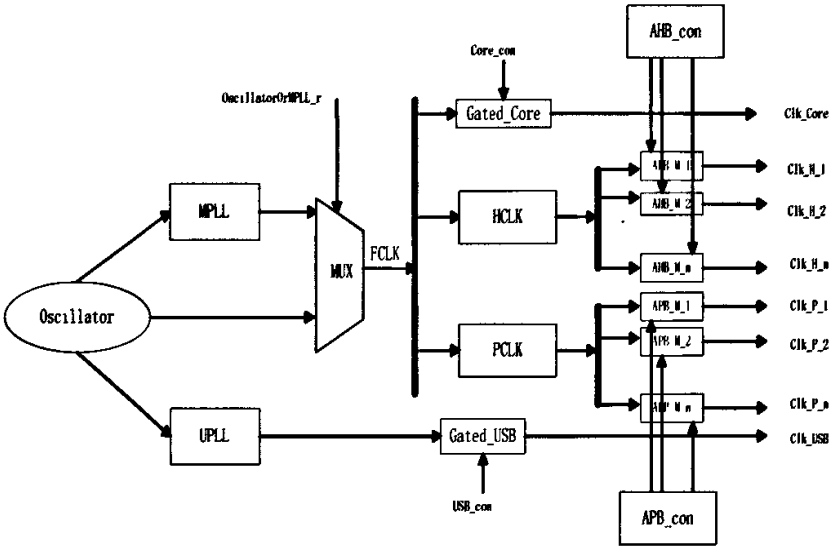

图 3-4 时钟源控制的示意图

PMC 中是采用选择器来实现时钟频率的转换。在以下三种情况需要用到多路选择器：

1) 由 Slow 模式切换到 Normal 模式；
2) 在 Normal 模式下重新配置 PLL；
3) 由 Normal 模式切换到 Slow 模式。

通过选择器来配置系统的时钟频率，虽然可以很方便的控制好整个芯片的功耗，但同时也带来了一些负面影响。因为多路选择器和门控时钟电路（如与非门单元）是最有可能产生毛刺的，而毛刺对同步数字系统的影响是致命的。它会导致同步的失败、数据的丢失、寄存器进入亚稳态，更为严重的是，使整个同步系统的功能失败。毛刺的产生是因为那些输入信号的时序匹配出现了问题，没有按照既定的顺序出现。因此在 RTL 设计时要保证做到时序的匹配，以降低毛刺产生的可能性。Garfield5 中是通过在的多路选择器和门控时钟电路之前加入一个锁存器来消除毛刺。

PMC 中的一个二选一的 MUX 控制时钟源选择电路如图 3-6。它的控制信号是 OscillatorOrMPLL，两个时钟源是 MPLL_output 和 clk_Oscillator，输出时钟是 out_ClockSource。当 OscillatorOrMPLL 为“1”时，MUX 选中 clk_Oscillator；当 OscillatorOrMPLL 为“0”时，MUX 选中 MPLL_output。当 MUX 的控制信号和时钟源下降沿同时到达时，输出时钟会产生毛刺。为此，通过增加使能信号 OscillatorEnable 和 MPLLEnable 来消除 MUX 选通时钟源带来的毛刺，在 MUX 选择其中任何一个时钟信号之前，通过使能信号把原来的时钟源先关断，这时输出低电平。当 MUX 选通另一方时，另一方的使能信号仍然是低，时钟输出还是为低，最后把该方的时钟使能信号打开，输出选中的时钟源，消除 MUX 选通开关带来的毛刺。但如果只用一个与门简单的将使能信号和时钟信号相与，使能信号还是会带来毛刺（如图 3-5）。这时有必要引入低电平选通的锁存器来过滤使能信号的毛刺，保证时钟的质量，这与前面提到的门控时钟技术相似。电路结构如图 3-6 所示。

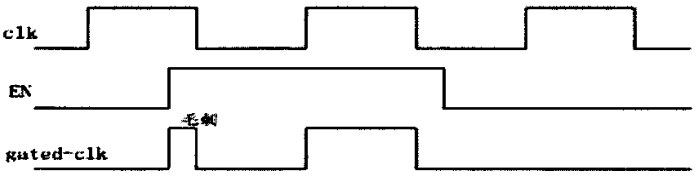

图 3-5 EN 信号引入的时钟毛刺示意图

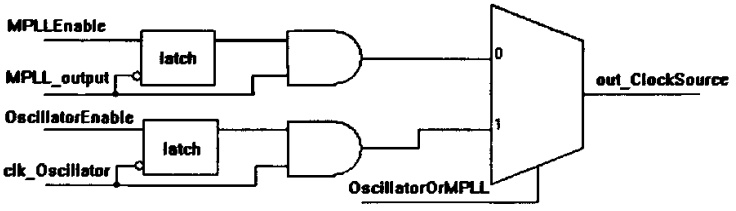

图 3-6 消除时钟毛刺的时钟源选择电路

3.3.2 Interface 子模块

Interface 子模块是 PMC 模块与总线接口的模块，这里我们只讨论与时钟相关联的部分。它经过门控时钟单元和选择器输出各个模块的控制时钟。它的典型选择电路如图 3-7 所示。经过如下结构的电路，它分别产生各个功能模块的控制时钟。这里需要说明的是，在测试状态下，各模块选择的是慢速晶振时钟。

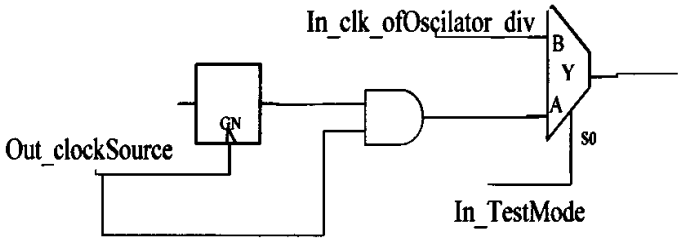

图 3-7 interface 模块中输出时钟的典型电路

3.3.3 Reset 子模块

Reset 模块主要是用来产生整个系统工作需要的 reset 信号，处理 WakeUp 信号，对外接的不规则的 reset、WakeUp 等唤醒信号进行规则化，避免误触发。它共有 3 种不同形式的 reset 信号：reset 按键触发的 Hardreset、从 RTC 模块来的 Watchdog reset 和通过配置寄存器产生的 Softreset，Hardreset 对包括 RTC 和 reset 模块本身的所有模块复位，其他两种 reset 机制则复位除 RTC 和 reset 模块本身的其他所有模块。从 Idle 和 Sleep 模式中恢复起来的机制分别说明如下：

- 从 Idle 中恢复出来的机制：
  - 内部所有模块的中断信号。系统回到进入 Idle 前的状态（Slow 或 Normal 模式）。

- 外部 in_WakeUp 信号（为了避免误触发，要求 in_WakeUp 信号有效时间大于 1s）。将经过 PMC 整形的 WakeUp 信号作为 INTC fiq 中断的一个中断源，系统回到进入 Idle 前的状态（Slow 或 Normal 模式）。
- 内部的 Alarm 信号。系统回到进入 Idle 前的状态（Slow 或 Normal 模式）。

■ 从 Sleep 中恢复出来的机制：

- 外部的 in_WakeUp 信号。将经过 PMC 整形的 WakeUp 信号作为 INTC fiq 中断的一个中断源，打开控制晶振时钟源的三态 pad，系统回到 Slow 状态。
- 内部的 Alarm 信号。打开控制晶振时钟源的三态 pad，系统回到 Slow 状态。

后端设计中工具会对 reset 等高扇出信号进行自动修复（PPO1 即 post placement optimization 1）。如果将时钟偏差和最大时钟延时路径的概念推广开来，reset 信号网络也有到达各终点元件的时间偏差和 reset 信号传输的延时路径。当设计对 reset 信号传播网络有严格时序要求时，可以采用 astHFCTS 命令进行 reset 信号传播网络的综合。Garfield5 中 reset 信号传播网络是在 PPO1 中自动修复的。

3.4 Garfield5 时钟结构

图 3-8 是 Garfield5 总体时钟结构示意图，它描述了 Garfield5 中的输入时钟，内部产生时钟和输出时钟的情况。图中所示的 TCK、CLK32、CLK5M、SYSCLK、USBCLK、CODECCLK 和 SDCLK 都要在时钟树综合时进行时钟源点的定义。

具体的时钟产生电路如图 3-9 所示，PMC 中主要的三个内部控制时钟为：经由触发器 Clk_ofOscillator_slow_reg 分频得到的时钟 clk_ofOscillator_slow；经过触发器 Clk_ofOscillator_div_reg 分频得到的时钟 clk_ofOscillator_div；MPLL 模块输出时钟 out_clk_MPLL。对于这些多工作模式的复杂时钟结构：首先提取出整个时钟树上所有可能的分支并选择需要进行时钟树综合的分支。CLK5M 上所有可能的分支有 clk_ofOscillator_slow、clk_ofOscillator_div、out_clockSource、USBCLK 及 CODECCLK。而需要进行独立综合的分支有 out_clockSource、USBCLK 及 CODECCLK。

其次，选择时钟树综合的主结构。由于 EDA 软件只能识别一种工作模式下的时钟结构来进行时钟树综合，选择对时钟树平衡和电路性能影响最大情况下的控制时钟作为主时钟结构。举例来说，如果某部分时钟在工作模式下和其他部分是异步工作的，但在测试模式下，它们处在扫描链中，和其他部分是同步的，我们就选择测试模式作为主时钟结构。相反情况下，如果某部分时钟在工作模式下和其他时钟是同步工作的，而且这部分时钟中还包含一些时序关键路径，我们就选择工作模式作为主时钟结构。由此可以看出工作模式并不一定优先于测试模式，选择依据是对时钟树平衡和系统性能的影响。保证选择的时钟结构边界是唯一的，不存在重叠和遗漏，然后借助 EDA 软件进行时钟树综合。

Garfield5 中选择的时钟树综合的主结构是 CLK5M。对于不包含在主时钟结构的局部，以及为了简化时钟结构而切割出来的局部时钟，如果该部分时钟和其他部分异步工作，那么可以作为一个独立的时钟完成时钟树综合与分布（如 USBCLK 及 CODECCLK）。如果该部分时钟和其他时钟还需要同步（SYSCLK），就需要在主时钟综合之前对这部分时钟进行预综合，作为主时钟树中的一棵子树，然后按照 4.2 节中的方法将 SYSCLK 的时钟源点设置为 CLK5M 的同步点后，再进行主时钟 CLK5M 的时钟树综合。

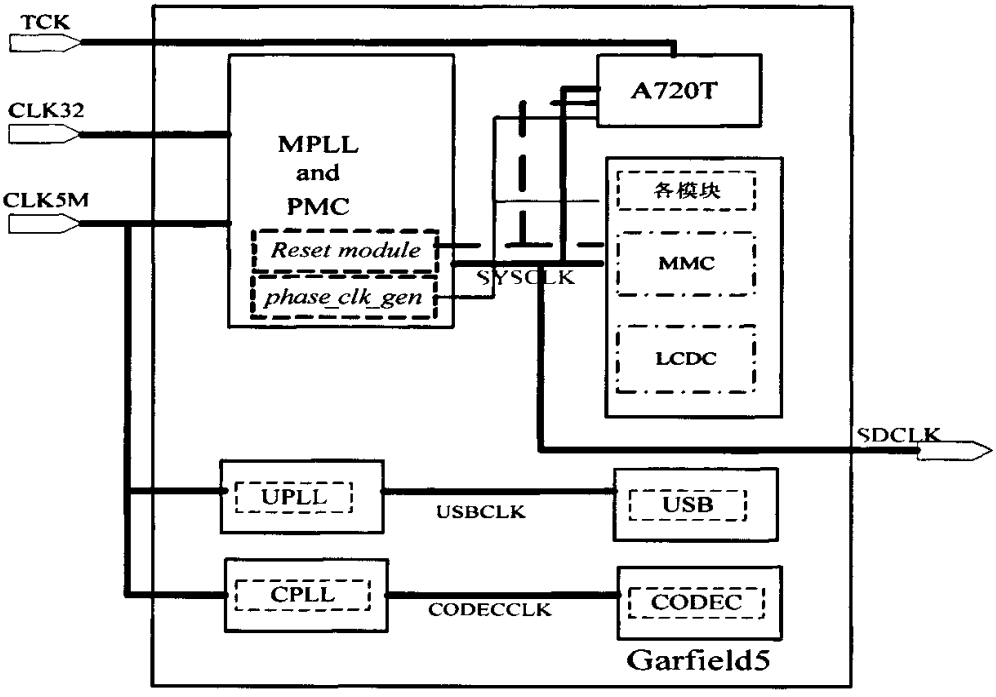

图 3-8　Garfield5 总体时钟结构示意图

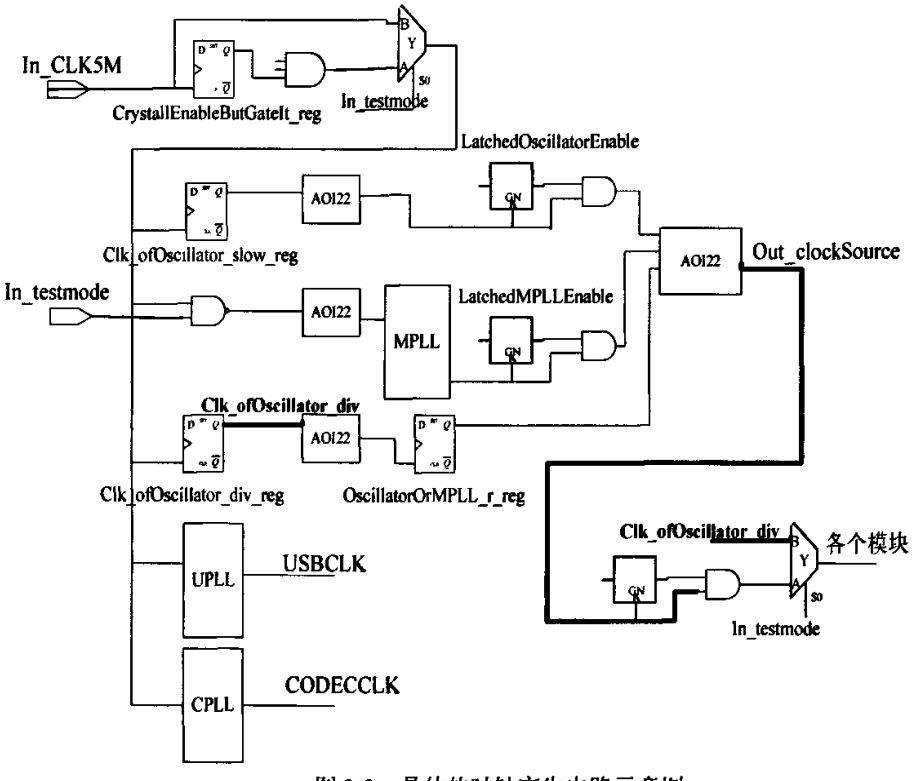

图 3-9　具体的时钟产生电路示意图

3.5 本章小结

本章主要介绍了 Garfield5 的总体架构。基于时钟树分布技术中引入的大量时钟缓冲器和倒相器是 SoC 动态功耗增加的重要原因，讨论了在时钟树综合中可采用的低功耗方案——门控时钟技术。具体结合 Garfield5 的功耗管理模块介绍了 Garfield5 的时钟结构和时钟树综合选择主时钟的方案。

第四章　基于 Garfield5 时钟树综合及优化

时钟树综合在后端设计流程中是处于标准单元布局之后，信号布线之前。它的基本流程如图 4-1 所示。Garfield5 的时钟树综合采用 Synopsys 的后端设计工具 Astro。

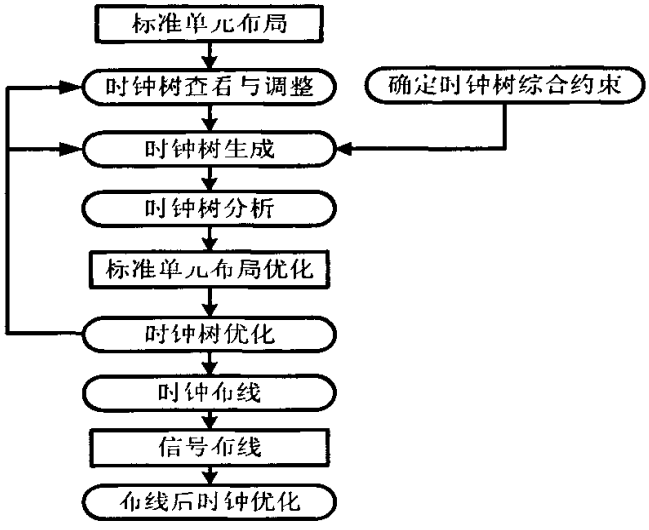

图 4-1　时钟树综合的基本流程

由第三章可知，Garfield5 的时钟结构包含异步时钟及门控时钟逻辑单元，并且它支持不同工作模式和可配置时钟结构。它是一个非常复杂的时钟结构体系。虽然 Astro 是 Synopsys 公司很成熟的后端版图设计软件，它可以根据设计者给出的时钟约束，自动生成时钟树。但是对于结构复杂的时钟系统，后端设计者需要在自动时钟树综合的基础上采用有效的时钟树综合方案和技术，才能得到质量比较高的时钟树。本章将结合 Garfield5 的时钟树综合，讨论在 Astro 中工具自动化时钟树综合的流程以及手动调整时钟树的一些优化技术。

4.1 时序约束文件准备

一个完整的时序约束文件包含：时钟的创建、时钟的约束、时序路径的约束、输入输出端口的约束等等。正确的时序约束文件才有可能使得芯片的后续设计工作得以完成。理论上，时序约束文件是前端设计人员在综合过程中加入时序约束而由综合工具（如 Design Compile）自动写出的。但实际设计过程中，时序约束文件是要根据实际的时钟树结构和时序要求不断的进行调整和修改，所以最终合理的时序约束文件是经过反复的实验不断的修正而得到的。

4.1.1 时钟的创建

描述时钟的几个要素是：

- 时钟源：时钟波形引入的端口或者引脚，一个时钟也可以有多个时钟源。
- 时钟周期：时钟波形重复的最小时间间隔。
- 波形：简单的波形有上升沿和下降沿。一般只要描述出这两个边沿就可以说明波形，即{上升沿位置、下降沿位置}。

(1) create_clock

在 Design Compiler 中进行门级网表综合时，时钟源点可以设定在层次化模块的端口（ports）处或具体的标准单元的引脚（pins）上。以下两条命令创建的 CLK5M 时钟是完全一致的。但要注意的是，为了使 Astro 准确的找到创建的时钟源点，时钟源点的位置定义应尽量选择在具体的标准单元的引脚上，因为在后端的版图设计中，它是把层次化去掉的，也就是扁平化的物理设计。因此层次化的模块端口不复存在，这样就会引起在后端设计中无法找到定义在模块端口处的时钟。所以在 Astro 中最好的创建时钟的方法就是将时钟源点都设置在具体的标准单元的输出引脚上。如下面第二种定义方法，Astro 就能明确的辨认各个创建的时钟。

```
create_clock -name "CLK5M" -period 9 -waveform {0 4.5} [get_ports {CLK5M}]
create_clock -name "CLK5M" -period 9 -waveform {0 4.5} [get_pins {Pad_in_CLK5M/XC}]
```

(2) create_generated_clock

Garfield5 中不仅仅只有外部的晶振时钟，还有内部生成的分频时钟。图 4-2 以二分频时钟为例，说明由时钟源分频得到的新时钟的创建方法。Create_generated_clock 是 Astro 用于创建生成时钟的命令。新的时钟频率可以直接在原时钟的频率上 divide /multipile 一个倍数。用 create_generated_clock 命令生成的时钟也称为衍生时钟，它在时钟树综合时可以看作一个新的独立的时钟源来对它控制的模块进行时钟树综合，但是本质上它仍然是原始时钟源的一个分支。平衡这个分支和原始时钟源点间的时钟偏差或时钟延时可以通过调整衍生时钟源点的位置或插入时钟缓冲器来完成。

```
create_generated_clock -name "SYSCLK" -source [get_pins {Pad_in_CLK5M/XC}]
                       -divide_by 1 [get_pins {U_garfieldbody/U_pm/U_PM_ctrl/U57/Y}]
```

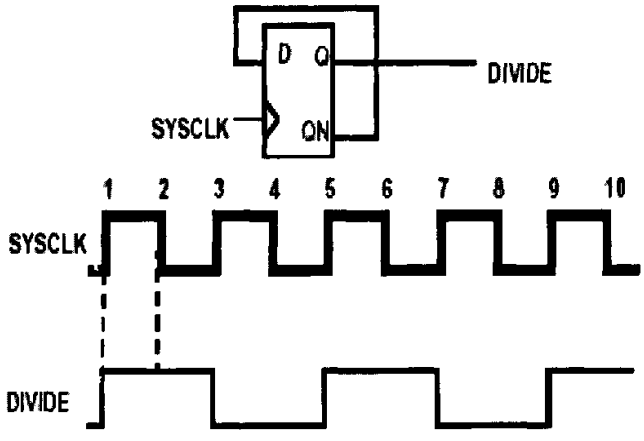

图 4-2 二分频时钟波形图

(3) 虚拟时钟（virtual clock）的创建

一个简单设计中所有可能的时序路径包含有四种类型：输入端口到寄存器；寄存器到寄存器；寄存器到输出端口；不经过寄存器，通过组合电路直接从输入端口到输出端口。虚拟时钟主要是对异步输入输出端口或纯组合逻辑进行约束。虚拟时钟在设计中没有时钟源点，它的定义只包含时钟周期和时钟名称。一般可以把异步的输入端口和输出端口绑定到虚拟时钟上去，从而设置输入端口和输出端口的延迟。这样可以避免在 Astro 或 PrimeTime 中做时序分析时报出一些没有时钟约束的端口的路径。设置一个虚拟时钟并将输入输出端口绑定到虚拟时钟上（假设所有的输入输出端口都是异步），可以用以下的命令实现：

```
create_generated_clock -name "VirCLK" -period 9 -waveform {0, 4.5}
set_input_delay 0.5 –clock VirCLK [all_inputs]
set_output_delay 0.5 –clock VirCLK [all_outputs]
```

4.1.2 时钟的约束

时钟树综合之前，必须要设定时钟树综合的目标（时钟偏差，时钟延时等）、一些特殊的时序路径约束（虚假路径，多周期路径等）和输入输出端口进行约束。需要说明的是，所有的时序约束都要经过前端设计人员的仔细确认才能加入到正式的时序文件约束中，因为前后端设计人员必须要保证通过工具所得到的时序信息是可以真实的反映实际电路时序信息的。

（1）set_clock_uncertainty 和 set_clock_latency

在时钟树综合以前，时钟信号都是理想的。理论上，逻辑综合时的时序约束应当尽量的严格，这样可以在后端实际进行时钟树综合时获得更多的空间和时间余量，从而使得时钟树综合的结果较为符合时序的要求。如果一开始的逻辑综合中时序约束就很宽松，经过后端布局和时钟树综合以及连线模型的精确化等步骤，它的时序可能就会达不到设计的要求。而在时钟树综合时可以在时钟的约束文件中设置一些参量对时钟进行合理的约束。如：

```
set_clock_uncertainty –clock SDCLK 0.2
set_clock_latency –clock SDCLK 2.0
```

它们的值都是根据实验和经验得到的相对合理的值。时钟不确定值(clock uncertainty)如第二章中所述它就是时钟树综合时预设的时钟偏差的目标值。它的值预设的太大可能会引起维持时间时序上的违规导致电路出现竞争冒险情况或者 Astro 根本就无法实现这样的时序要求。SDCLK 时钟延时的值也是要依据时序。这将在 SDCLK 时钟树综合时具体说明。

（2）set_false_path

Garfield5 的时钟结构中，除了系统主时钟 CLK5M 以外，还有两个片外输入时钟：TCK（A720T 模块中的时钟）和 CLK32（提供 RTC 计时时钟和 PMC 模块中 Reset 模块的时钟）。在时钟树综合时，它们是完全独立的时钟源。但是，在多时钟的电路设计中，不同的工作模式下的时钟频率可能不同，所以对于同一个标准单元，它在不同的情况下可能要被不同的时钟所控制。前端设计者一般会采用选择器（如 MUX）电路，通过改变控制端口的信号的取值来选择时钟通过的路径。在 Garfield5 中典型的情况就是在功能模式和测试模式下，选用不同的时钟频率来驱动电路。异步时钟之间的数据路径在时序上是没有要求的。在约束文件中，异步时钟可以设为时钟域之间的虚假路径（set_false_path）。虚假路径的命令如下所示：

```
set_false_path –from [get_clocks CLK5M] –to [get_clocks TCK]
set_false_path –from [get_clocks TCK] –to [get_clocks CLK5M]
```

同步时钟控制的某些时钟路径上也有一些时序路径是功能上不相关、不影响电路性能或者功能上根本不存在的路径。约束文件中必须要将这些路径罗列出来以避免时钟树综合过程中浪费资源和时间来对这些路径进行时钟树平衡和优化，甚至这些虚假路径上的时序违规会掩盖实际存在的关键时序路径，而使得电路的性能达不到要求。例如：

```
set_false_path -from U_garfieldbody/U_dmac/U_FIFO_ReadPointer_reg*/CK
               -to U_garfieldbody/U_dmac/U_AHB_SLAVE_DMACC4DestAddr_r_reg*/D
set_false_path -through U_garfieldbody/u_dsp_top/AMBE_core/XDB_cs*
               -to U_garfieldbody/u_A720TW/HRDATAM*,
```

（3）set_multicycle_path

Synopsys 公司提供的后端设计软件如 Design Compiler、Astro 和 Prime Time 等自动化工具都是缺省的人为所有的电路都是单时钟周期工作的。这意味着电路在一个时钟周期之内要将数据从一条路径的开始端传递到结束端。但在某些情况下，电路并不是在这样的方式下工作的。对电路中的某些数据路径来说它并不是单周期传递数据，所以必须对这些特殊情况下的时序路径做出约束说明。否则，时序分析将不能正确的反映电路真实的工作情况。一般可以使用多周期路径（multicycle path）来对这些路径进行说明，例如：

set_multicycle_path -setup 2 -from U_garfieldbody/u_dsp_top/AMBE_core/DAU/*aaTO2_reg*/CKN

set_multicycle_path -hold 1 -from U_garfieldbody/u_dsp_top/AMBE_core/DAU/*arM_o_reg*/CKN

（4）set_disable_timing

Garfield5 中采用了很多时钟选择器。经过时钟选择器选择的其中一个时钟作为与它相连的所有模块的控制时钟。那么，对于另外一个时钟通路就可以通过 set_disable_timing 命令来阻止它的时钟传递。这样就避免了在 Astro 或 PrimeTime 进行时序分析时，会报出根本不需要的时序路径上的时序信息，节约了设计时间。它的命令格式如下：

set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_Core

（5）set_input_delay 和 set_output_delay

约束文件中最好包含每一个输入端口和输出端口的约束，异步端口的约束也可以通过虚拟时钟来实现。Garfield5 中最主要的端口的约束是与 SDRAM 相关的控制信号和数据地址信号端口的约束。它们的输入输出的延时时间值都是经过 SDRAM 相关的说明文档进行计算的。例如：

set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA0}]

set_output_delay 3 -max -clock "SDCLK" [get_ports {out_ADDR14}]

set_output_delay -1.5 -min -clock "SDCLK" [get_ports {out_ADDR14}]

（6）set_load

同样对于输出端口也要有负载的约束。Garfield5 中的输出端口的负载值是经验值。

set_load -pin_load 20.6 [get_ports {out_ADDR14}]

set_load -min -pin_load 20.6 [get_ports {out_ADDR14}]

逻辑综合和时钟树综合的约束参数还有很多，以上只是列举了 Garfield5 中用到的一些约束。简而言之，时序约束文件是要根据具体的电路设计和性能要求而定的，它必须要保证真实的反映实际电路的时序情况，因为时序约束文件是后端物理设计中各个步骤的实现目标。

4.2 CTS 前时钟树结构分析及调整

在时钟树综合之前，需要查看时钟树结构，这样做的目的有两点：保证前端、后端所定义的时钟一致；调整时钟树结构，达到性能结果更优。图 4-3 是基于 Interactive CTS 选项察看的 SYSCLK 示意图。查看时钟树时需要特别注意的一些单元有：时钟选择器、门控时钟单元中的锁存器、宏单元模块和输出时钟管脚等。

结合第三章时钟结构图和图 4-3 的 SYSCLK 的时钟结构，U57 是时钟选择器（可选择时钟通路 clk_of_Oscillator_div, clk_ofOscillator_slow 及 in_MPLL_Fout），它的输出端口是与 pm_interface 模块里的各个门控时钟单元和二选一选择器的输入端口相连，所以选择 U57 的输出端口作为 SYSCLK 时钟源点来控制功能模式下各模块的时钟。查看时钟树结构除了帮助后端设计人员了解核对具体的时钟结构外，它还可以让设计者对时钟结构进行的调整。一般，在查看时钟树结构的同时，可以直接在 Astro 中设置 ignore pin 和 sync pin。时钟树综合中把时钟不再继续向下传播的 pin 定义为时钟树的结束点。时钟树的结束点有两种：

- sync pin，也称作时钟汇点。时钟汇点包含具有触发沿信息的时序单元的时钟端口和设计人员根据时序要求使用命令 ataDefineSyncPin 或 ataDefineSyncPort 所定义的同步时钟端口。在时钟树综合时，Astro 对时钟汇点的处理是既要满足最大时钟偏差和最小时钟延时的要求，又要满足最大的跳变时间、负载电容、最大扇出及最大缓冲器级别的要求。
- ignore pin。ignore pin 在时钟树综合时只考虑它的最大跳变时间、负载电容、最大扇出及最大缓冲器级别的目标。对它的时钟偏差和时钟延时是没有要求的。简言之，被定义了 ignore pin 的时钟端口在时钟树综合时是被忽略不进行同步时钟平衡的。对于以下情况的端口，Astro 会默认为是 ignore pin：

- 设计人员使用命令 dbDefineIgnorePins 定义的时钟端口
- 没有触发沿信息的时序门的时钟端
- 时序门电路的非时钟端（比如 set 或 reset 端）
- 输入端到输出端没有时序路径的逻辑门
- 输出端没有负载的逻辑门
- 任何使时钟树传输终止的端口（比如边界输出端口等）
- case analysis 的声明阻止时钟传输到时钟汇点

Ignore pin 的设置要谨慎，因为在时钟树综合时，如果 Astro 认为它是 ignore pin 那么在时钟树综合时就不会对由这个 ignore pin 衍生出的连线进行优化。而同时，这些连线又被标志成为时钟线，在处理一般的信号时序优化时也不会对时钟信号进行任何操作。因此，如果一个 ignore pin 的带载很多，那它上面的 DRC 违规就得不到修复。当然设计人员可以设置参数 axSetIntParam "acts" "high fanout net synthesis" 1 对这类 ignore pin 进行高扇出综合，这样时钟树综合的时候它就会考虑修复 ignore pin 上的 DRC 违规。

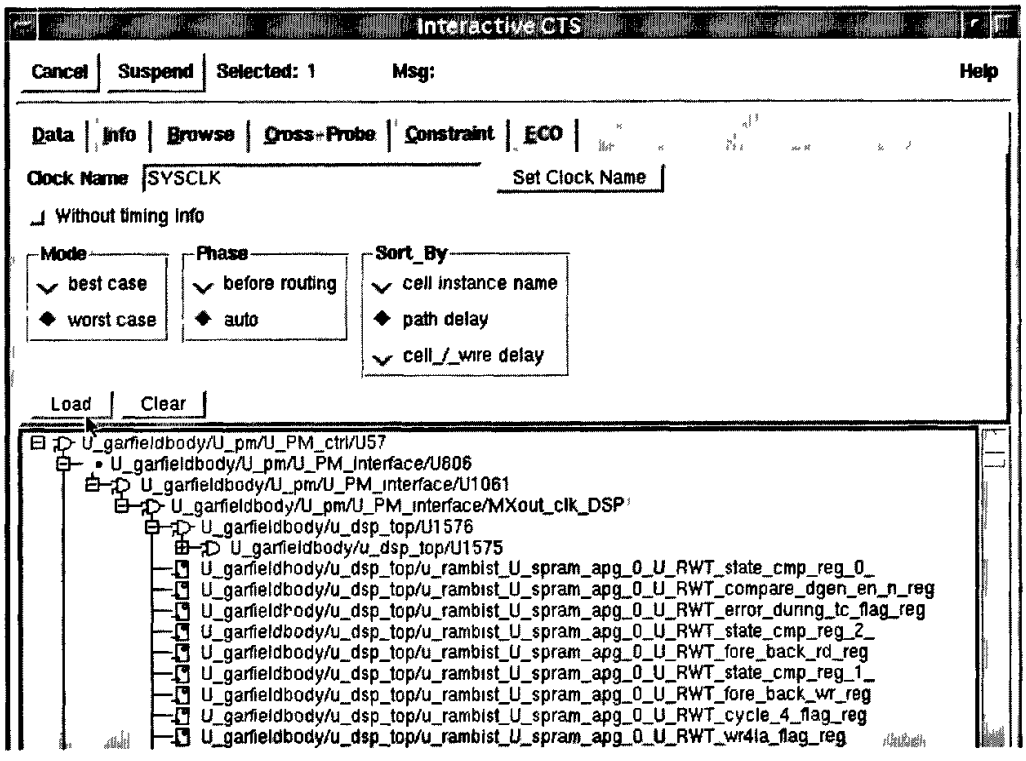

图 4-3 Astro 查看 SYSCLK 时钟结构

在调整时钟树结构时，针对对以下三类的时序结构设置了 ignore pin：

- pm_interface 模块里选择器的 A 端（与 clk_of_Oscillator_div 相连的端口）都设为 ignore pin。这是因为当电路在功能模式下（也即 set_case_analysis 设为 0,因为 mux 的 S0 端接的控制信号是 tesemode 的非），选择器选择的时钟通路是经由 B 端传输的时钟信号。一般，只要在 Astro 中设定了 set_case_analysis 的值，它会自动辨认时钟选择的路径，而不必重新设置时钟选择器上的 ignore pin。
- 门控时钟结构中的锁存器单元，它的 GN 端在高版本的 Astro 中也是默认的 ignore pin。但是，一般情况下，设计人员还是会将这些锁存器的 GN 端单独罗列出来重新定义为 ignore pin。这样除了防止可能漏设掉 ignore pin，同时也便于设计者确认核对时钟结构。

■　一些特殊的单元。它们在时序上不要求与其它寄存器同步。

简单的 ignore pin 设置的命令如下：

```
dbPurgeIgnorePin (geGetEditCell)  "*"
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/\
U_PM_interface/LatchedDSPEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/\
U_PM_interface/MXout_clk_DSP" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/ clk_OfOscillator_div_reg" '("RN")
```

在 Astro 中设置 ignore pin 的界面如下图所示：

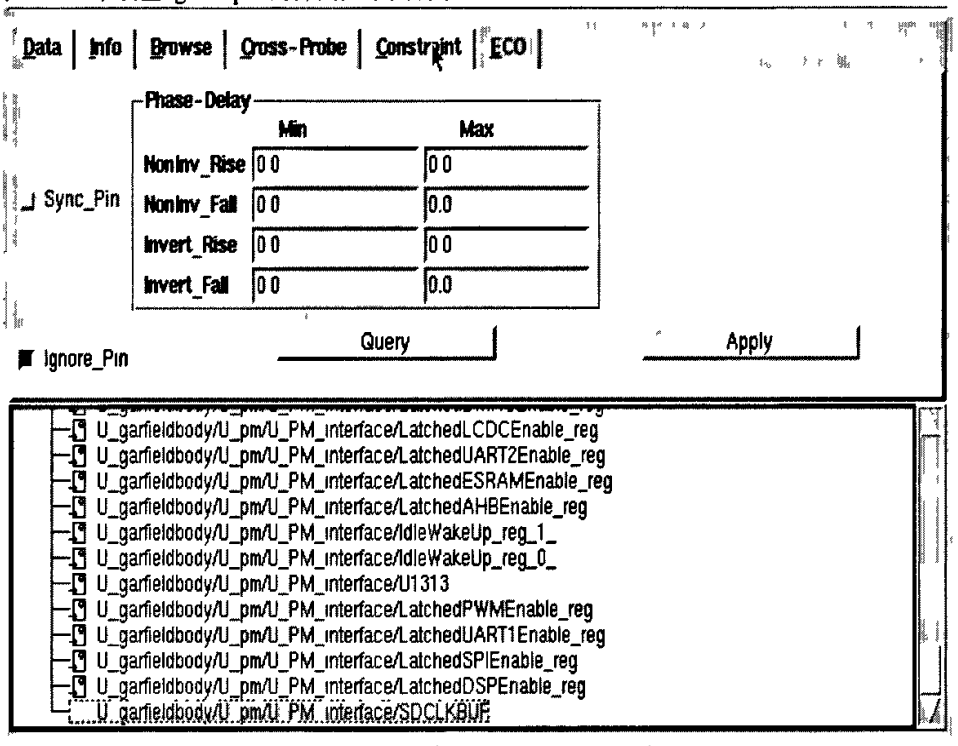

图 4-4　Astro 中设置 ignore pin 界面

而 sync pin 的定义在 Garfield5 设计中主要是应用在宏模块的时钟输入端口。因为在 Astro 中对于 sync pin，时钟树综合时不会去优化模块内部的时序，但是它会计算模块端口输入时钟到它内部寄存器单元间的时钟延时，然后根据时钟约束的要求在定义的 sync pin 之前插入时钟缓冲器或导向器来平衡时钟源点到模块内部寄存器和模块外的寄存器之间的时钟延时。简单的 sync pin 设置的命令如下：

```
dbPurgeSyncPin    (geGetEditCell)  "*"
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_ctrl/\
U_PM_ClkSrcSel/OscillatorOrMPLL_r_reg" '(("CK" "nonInvertRise" (0.00 0.00)))
```

Ignore pin 和 sync pin 的设置需要确保正确无误。如果将需要进行同步时钟约束的时钟树的结束点错误的设置为 ignore pin，可能会导致整个时钟树的不平衡和时序违规；而漏设 ingore pin 会浪费软件工具的资源去平衡不必要的时钟树的结束点，这样也可能导致降低对关键时序路径优化的效果。也可以在 Astro 中可以利用 dbDumpIgnorePin 导出所有定义了的 ignore pin（同理 dbDumpSyncPin 可导出所有的 sync pin），再次与前端设计人员进行核对确认。Garfield5 的完整的 sync pin 和 ignore pin 的设置脚本见附录 D 和附录 E。

4.3 基于 Astro 自动时钟树综合

如前所述，Garfield5 系统芯片中的时钟结构是相当复杂和庞大的，在时钟树综合之前选择合理的时钟树综合的先后次序是非常重要的。在时钟树综合时，一种方案是将时序要求严格的最后执行时钟树综合。这是因为如果预先放置的单元和要插入的时钟缓冲器或倒相器发生重叠时，那些单元将被移走来放缓冲器或倒相器，致使时钟树插入越晚，被打扰的可能性就越小。另一种方案是时序要求严格的最先执行时钟树综合。这是因为在一开始进行的时钟树综合会有比较大的资源，它可以得到比较好的时钟树结构。而后执行的时钟树的综合可利用的空间资源就比较小。选择时钟综合的先后次序要结合实际的设计。Garfield5 中选择的方案第二种，即是先综合时序要求严格的 SYSCLK 的时钟树。这是考虑到 Garfield5 的时钟有：SYSCLK, CODECCLK, USBCLK, SDCLK, CLK5M, TCK, CLK32。除了 SYSCLK, CODECCLK,和 USBCLK 三个时钟是快速时钟，其它的时钟频率都很低。Astro 对低速时钟的平衡比较容易实现，并且它们都是异步的时钟域，在最后对它们进行时钟树综合并不会影响前面已经综合好的时钟树性能。Garfield5 在功能模式下各个模块的主时钟是 SYSCLK(除了 RTC 模块中的一些单元在功能模式下是由 CLK32 时钟控制)。所以 Garfield5 的时钟综合方案是首先综合 SYSCLK；然后进行 CODECCLK 和 USBCLK 的综合；再进行 CLK5M 的时钟树综合；最后是异步时钟(TCK, CLK32)时钟树综合和手工调整 SDCLK 的延迟。本章节主要讨论 SYSCLK 和 CLK5M 的时钟树（因为它们是系统主时钟，对电路的性能起决定性的作用）。自动化时钟树综合流程主要包括 CTS（clock tree sythesis），PPO2（post placement optimization 2）和 CTO（clock tree optimization）三个部分。它的具体步骤如下：

（1） 时序参数设置（timing setup）。

在每一次进行和时序相关的操作步骤之前都要进行时序参数设置。CTS 之前所有的时序检查使用的都是理想时钟，因此它的参数设置为选择 Ignore Propagated Clock 和 Enable Ideal Network Delay。互连线延时模型设置为 Elmore。电容模型设置为 TLU（Table Look-Up），它是一种电阻和电容的查找表模式，在这种模型下，定义了每层金属的方块电阻，并根据金属线间距和宽度给出了同层间电容和不同层间电容的查找表模式，Astro 预设是使用 TLU 模式。

（2） 时钟树选项设置（clock commom options）。

CTS 要对时钟树选项进行设置，这些选项包括环境（最好、最坏及典型）、时钟偏差类型（全局时钟偏差、局部时钟偏差及有用时钟偏差）、优化程度、时钟线定义、时钟 Buffer 及倒相器定义、时钟树结构和时钟树优化方式以及目标的设置。

- 环境是指时钟树综合时所使用工艺条件，在标准库单元中有 worst、best、typical 三种环境的说明。例如对于 SMIC 0.18um 工艺，三种不同的工作环境下相关参数如下表所示[22]：

表 4-1 SMIC 0.18um 不同环境下的工艺参数

环境	温度（T）	电压（V）	工艺系数（k）
Slow	125℃	1.62V	1.27
Typical	25℃	1.8V	1.00
Fast	0℃	1.98V	0.793

Astro 预设的时钟树综合环境是 worst。一般会把 worst 和 best 都选上，这样 Astro 在 CTS 时会给出一个折衷的值。太悲观和太乐观的环境参数的选择都可能对 CTS 产生不好的结果。

- 时钟偏差类型的选择要依据实际的设计而言。Garfield5 中选择的是全局时钟偏差类型进行时钟树综合。对于有些设计来说，如果时序要求很严格（如全定制的微处理器核）且设计中数据路径的时序违规不是很多，则可以通过有用时钟偏差来进行关键时序路径上的时钟偏差的调整。

- 在 CTS 过程中时钟缓冲器和倒相器的插入是对时钟偏差和时钟延时进行优化，这个优化算法的迭代次数 Astro 预设为 2。
- 时钟线定义的选项可以通过查看 SDC 文件中定义的时钟源点的信号线来确定。Astro 支持对几个时钟信号线同时进行综合。这里要注意的是几个时钟信号同时进行综合时，要考虑到它们综合的先后次序。但如果几个时钟都是异步且慢速的时钟，先后次序的意义并不大。
- 而考虑到面积及功耗的因素，可以对时钟树综合时要插入的时钟缓冲器和倒相器进行选择，Astro 预设的值是所有的时钟缓冲器和倒相器及延时单元。
- Garfield5 中采用了门控时钟，所以在选项里要使能门控时钟，使时钟能穿过门控单元。
- 时钟树综合目标值（时钟偏差和时钟延时）的设置有两种方式：一是导入 SDC 文件；另一种是在 clock commom option 的选项对话框中设置 target skew 和 target insert delay。SDC 文件中如果使用了命令 set_clock_uncertainty 和 set_clock_latency 来设置 CTS 的目标值，那么在 CTS 时，clock commom option 预设的值就是 SDC 文件中的设置的目标值。Astro 会努力的实现插入的时钟延时不低于目标值，时钟偏差不高于目标值。这里要特别说明的是：如果设计中有多个时钟，且每个时钟的目标值不尽相同，但对话框中时钟偏差和时钟延时的目标值都只有一个，这就会产生目标值设置的矛盾。要解决这个问题的办法有两种：一是通过设置参数 axSetIntParam "acts" "ignore set_clock_uncertainty" 0 改变工具的预设值，使之采用 SDC 文件中对各个时钟所设置的时钟偏差和时钟延时值。另一方法是对时钟逐个进行综合，每个时钟的约束分别定义在不同的 SDC 文件中，通过每次重新读入 SDC 文件来改变选项的预设值。显然第一种方法更实用。

下面是 clock commom option 的脚本：

```
astClockOptions
formDefault "Clock Common Options"
setFormField "Clock Common Options" "Best" "1"
setFormField "Clock Common Options" "Typical" "1"
setFormField "Clock Common Options" "Worst" "1"
setFormField "Clock Common Options" "Skew Type" "Global"
setFormField "Clock Common Options" "Skew Type" "Local" "0"
setFormField "Clock Common Options" "Skew Type" "Useful" "0"
setFormField "Clock Common Options" "Clock Nets" "U_garfieldbody/U_pm/ClockSource"
setFormField "Clock Common Options" "Target Skew" 0.2
setFormField "Clock Common Options" "Optimization Effort" "5"
setFormField "Clock Common Options" "Synthesis Effort" "5"
setFormField "Clock Common Options" "Buffers/Inverters"
                                    "CLKBUFX1,CLKBUFX2,CLKBUFX3,CLKBUFX4,
                                    CLKBUFX8,CLKBUFX12,CLKBUFX16,CLKBUFX20"
setFormField "Clock Common Options" "Buffer Sizing: LEQ Cells"
                                    "CLKBUFX1,CLKBUFX2,CLKBUFX3,CLKBUFX4,
                                    CLKBUFX8,CLKBUFX12,CLKBUFX16,CLKBUFX20"
setFormField "Clock Common Options" "Delay Cells"
                                     "CLKBUFX1,CLKBUFX2,CLKBUFX3,CLKBUFX4,
                                    CLKBUFX8,CLKBUFX12,CLKBUFX16,CLKBUFX20"
formOK "Clock Common Options"
formCancel "Clock Common Options"
```

（3） 时钟树综合（Clock Tree Synthesis）。

时钟树综合中选项都可以使用 Astro 预设的，它主要是借助工具算法自动完成的一项工作。至于它所包含的 clock tree optimization 的选项将在以下的章节中详细的说明。CTS 的综合脚本如下：

```
astCTS
formDefault "Clock Tree Synthesis"
setFormField "Clock Tree Synthesis" "Best" "1"
setFormField "Clock Tree Synthesis" "Worst" "1"
setFormField "Clock Tree Synthesis" "Typical" "1"
setFormField "Clock Tree Synthesis" "Skew Type" "Global"
setFormField "Clock Tree Synthesis" "Skew Type" "Local" "0"
setFormField "Clock Tree Synthesis" "Skew Type" "Useful" "0"
formOK "Clock Tree Synthesis"
```

CTS 过程中会引入大量时钟缓冲器和倒相器。它们虽然在一定程度上优化了时钟树结构，但是由于这些时钟缓冲器和倒相器的插入：拥塞（congestion）可能会增大；同时为了平衡时钟树结构，可能会牺牲掉一些非时钟控制单元的摆放位置，使得它们不是处在最佳的位置上；还有 CTS 也会带来最大跳变时间和最大电容等 DRC 违规。这些问题可以借助 PPO2 来进行优化。

（4） PPO2（post placement optimization 2）。

PPO2 的主要目的是修复维持时间的时序违规。在 PPO2 中 setup fixing 的选项一定要选上，如果忽略此项，那么修复维持时间的同时会破坏一部分建立时间的时序，这将使得前面的工作不能达到原来的效果，甚至要重新重复前面的工作。PPO2 另外一个主要的目的是修复 DRC 违规。另外，在 PPO2 时一般要选上 remove buffer（这一选项不是 Astro 中预设的），它是用来移除一些时钟缓冲器以减小拥塞。如果在设计中使用了扫描链，在做 PPO2 之前应该要将扫描链连接好。这样得到的维持时间的修复才是反映了实际电路情况的。

（5） CTO（clock tree optimization）。

所有时钟树结构都生成以后，可以通过 CTO 来优化时钟树，它在一定范围内可以使得时钟树性能有所提高，但幅度不大。Astro 中的 CTO 是工具自动计算完成的，它包含了下面这些优化方法：

- 移动时钟缓冲器和倒相器以及逻辑单元的位置。

如图 4-5 所示，当时钟缓冲器和倒相器的位置变化时，时钟路径上的 RC 阻抗分布也随之发生变化，从而影响时钟路径上的延迟和其他物理特性。

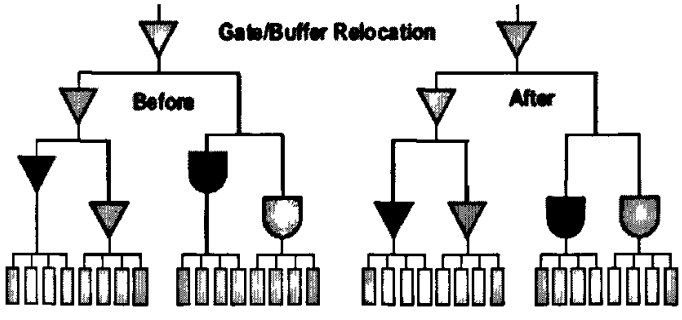

图 4-5 移动逻辑单元位置优化时钟树

- 调整寄存器的级别。

如图 4-6 所示，如果某些时钟路径分支的延时不是很好，可以通过改变这些路径上寄存器的级别来优化时钟树。

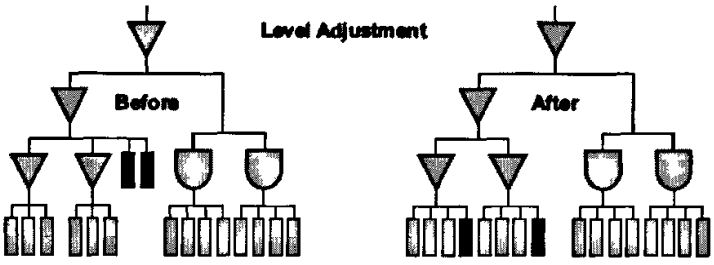

图 4-6 调整寄存器级别优化时钟树

■ 调整时钟树上时钟缓冲器、倒相器和门控单元的大小。

标准单元库中同一逻辑功能的单元具有不同的大小，不同大小的单元在物理特性和电学特性上都有很大的不同。表 4-2 是 SMIC0.18um 标准单元库中时钟缓冲器不同大小的单元之间的对比[22]。

表 4-2 不同大小的时钟缓冲器参数比较

	高度(um)	宽度(um)	输入电容(pF)	A-Y↑延时(ns)
CLKBUFXL	5.04	2.64	0.0021	0.0544
CLKBUFX1	5.04	2.64	0.0035	0.0632
CLKBUFX2	5.04	2.64	0.0031	0.0715
CLKBUFX4	5.04	3.30	0.0042	0.0782
CLKBUFX8	5.04	4.62	0.0079	0.0727
CLKBUFX12	5.04	10.56	0.0191	0.0730
CLKBUFX16	5.04	12.54	0.0229	0.0761
CLKBUFX20	5.04	15.84	0.0315	0.0713

由上表可以看出，时钟路径上的缓冲器大小不同就会带来时钟延迟和连线负载的不同。当某些时钟路径的驱动电流较小，导致时钟缓冲器或者倒相器后边的连线延迟过大，从而使相关时钟路径的延迟增大时，可以通过图 4-7 所示的方案，调整单元大小来优化相关时钟路径。

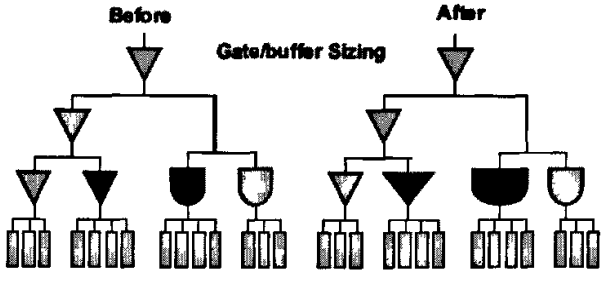

图 4-7 改变逻辑单元大小优化时钟树

■ 重新分配时钟缓冲器或倒相器所驱动的时序单元。

初始生成的时钟树结构中，寄存器在时钟树上的分配可能不合理，有可能逻辑关系和物理位置联系密切的寄存器没有分配到一起，如图 4-8 所示，可以对时钟树叶结点的寄存器进行重新分配，从而起到优化时钟树结果的作用。

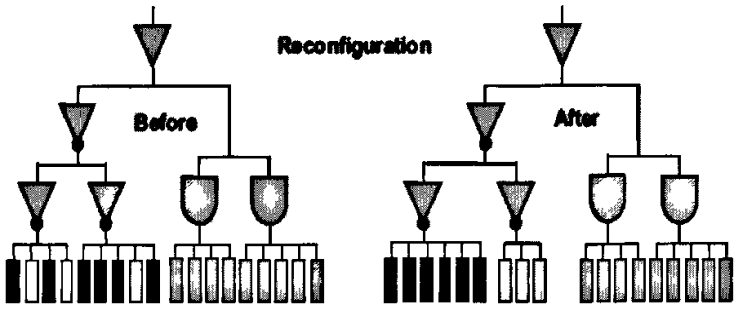

图 4-8 重新分配寄存器优化时钟树

■ 插入延迟单元给较少负载的连线。

设计中有些特别短的路径需要较大的延迟才能平衡，如果依靠缓冲器或者倒相器提供，那么需要很多的单元插入，会对布局布线资源造成很大的浪费，而且会有较大的功耗。制造厂商一般会提供专门的延迟单元，延迟单元的内部结构是一系列的缓冲器串接而成的。表 4-3 是 SMIC0.18um 标准单元库中延迟单元的延时和物理特性，通过和表 4-3 比较可以看出，一个延迟单元就可以提供比时钟缓冲器大的多延迟[22]。

表 4-3 SMIC0.18um 库中延迟单元的物理参数

	高度(um)	宽度(um)	输入电容(pF)	A-Y↑延时(ns)
DLY1X1	5.04	3.96	0.0019	0.1555
DLY2X1	5.04	3.96	0.0020	0.3018
DLY3X1	5.04	4.62	0.0020	0.4952
DLY4X1	5.04	4.62	0.0020	0.7190

通过比较可以看出，使用延迟单元具有延迟大，面积小，连线少的特点，根据这特点，当时钟树上某些时钟路径延迟小于其他时钟路径延迟时，如图 4-9 所示，给短的时钟路径插入延迟单元，能有效平衡时钟树，同时代理的物理影响最小。另外一种用到该优化方案的地方就是不同的时钟树间的平衡，最简单的办法就是在其中一个延时较短的时钟树根结点加延迟单元。

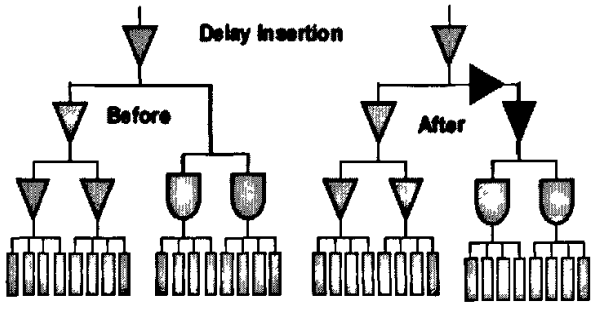

图 4-9 插入延迟单元优化时钟树

■ 增加冗余负载。

为了达到时钟平衡，可以在已生成的时钟树基础上增加冗余负载，使得时钟树更好的平衡。如图 4-10 所示。

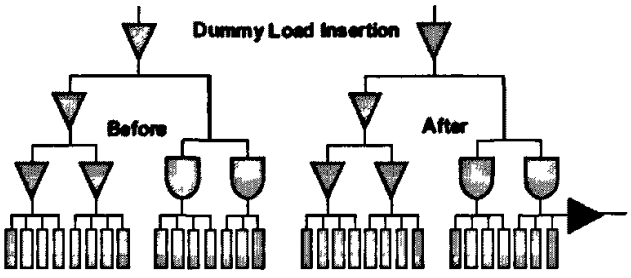

图 4-10 增加冗余负载优化时钟树

按照上述步骤对 SYSCLK 进行时钟树综合完成后，首先需要平衡 SDCLK。SDCLK 是时钟源点（U57 的输出端）直接输出到芯片输出管脚的。在 SYSCLK 综合过程中，SDCLK 这个分支是通过定义 ignore pin 断开的。平衡 SDCLK 时钟的方法就是在这条时钟路径上插入延时单元。插入的延时值是根据 SYSCLK 和 SDCLK 之间的最大时序违规值来设定的。由实验知，Garfield5 中 SDCLK 的延时需要插入 2.3ns 左右。如果在进行 SDCLK 时钟平衡之前，在约束文件中设置：

set_clock_latency 2 [get_clocks {SDCLK}]

SDCLK 综合时 Astro 会尽力达到这个设定的目标值。如果选择插入的延时单元是时钟缓冲器，那么需要插入几十个的时钟缓冲器，这是不必要的。一般需要插入的延时较大时，选择延迟单元更为适用，延迟单元所能提供的延迟时间远远大于时钟缓冲器。所以它在一定程度上减小了面积和功耗开销。如果 SDCLK 综合完以后，仍然有它与 SYSCLK 之间的时序违规，这个值现在是比较小的，大概在 0.8 左右，那么可以通过 Change netlist 选项来人为的调节时钟延时。它的命令如下：

```
astChangeNetlist
setFormField "Change Netlist" "Operation" "Insert Buffer"
setFormField "Change Netlist" "Insert PortInst Name"
"U_garfieldbody/U_pm/U_PM_interface/SDCLKBUF"
setFormField "Change Netlist" "Insert PortInst Name"
"U_garfieldbody/U_pm/U_PM_interface/SDCLKBUF/A"
setFormField "Change Netlist" "Insert Master Name" "CLKBUF8X"
formOK "Change Netlist"
```

表 4-4 和表 4-5 比较了 CTS，PPO2 和 CTO 每个步骤之后的各个时钟的时序情况。

表 4-4 Clock skew 比较

Clock skew	SYSCLK	CLK5M
CTS	0.287	0.258
PPO2	0.282	0.255
CTO	0.267	0.23

表 4-5 Clock latency（Max）比较

Clock lantency(Max)	SYSCLK	CLK5M
CTS	2.721	2.278
PPO2	2.641	2.324
CTO	2.56	2.081

4.4 时钟树优化技术

只依靠工具自动生成时钟树综合有时不能达到时序的要求，对此，必须要进行一些手工的优化方法对时钟树结构进行调整和优化。下面主要介绍通过优化 PMC 模块布局，调整时钟门控连线权重和调整时钟源点位置的方法对时钟树进行优化。在未进行时钟树综合之前，一个合理的布局是实现一个效果良好的时钟树结构的先决条件。Garfield5 系统芯片中包含五个 SRAM 和 A720T IP 硬核，另外还有 CODEC 和 ADC_PANEL 宏模块以及三个数字 PLL 模块，这些宏单元的面积占到芯片面积近乎五分之三。如果布局不合理，就会形成窄而长的通道，给时钟树平衡带来困难。如果在窄而长的通道放置有多个寄存器，那么需要在通道口创建新的时钟，对于通道内部的寄存器进行局部平衡，然后再和整个时钟树平衡。这样的做法是很复杂的，所以设计人员应在最初布局的时候就考虑到要尽量留出大块面积给标准单元进行布局。得到最佳的布局方案后，时钟树综合才能进行。

图 4-11 和图 4-12 是两种不同的宏单元布局的方案。图中标示的数字代表各宏模块：1 为 MPLL，2 为 CODEC，3 为 ADC_PANEL，4 为 DPLL，5 为 UPLL，6 为 A720T，7 为 DPRAM0，8 为 DPRAM1，9 为 XRAM0，10 为 XRAM1，11 为 XRAM2。

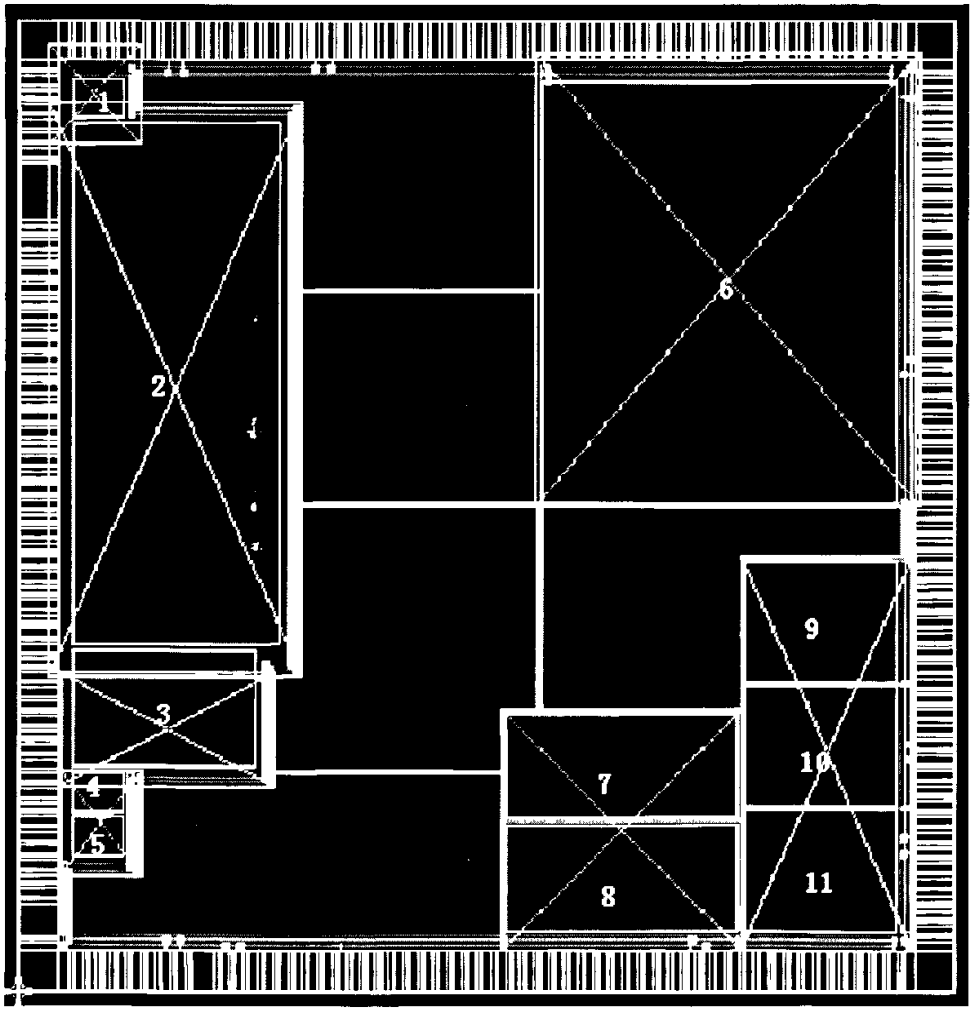

图 4-11 Garfield5 合理的宏单元布局

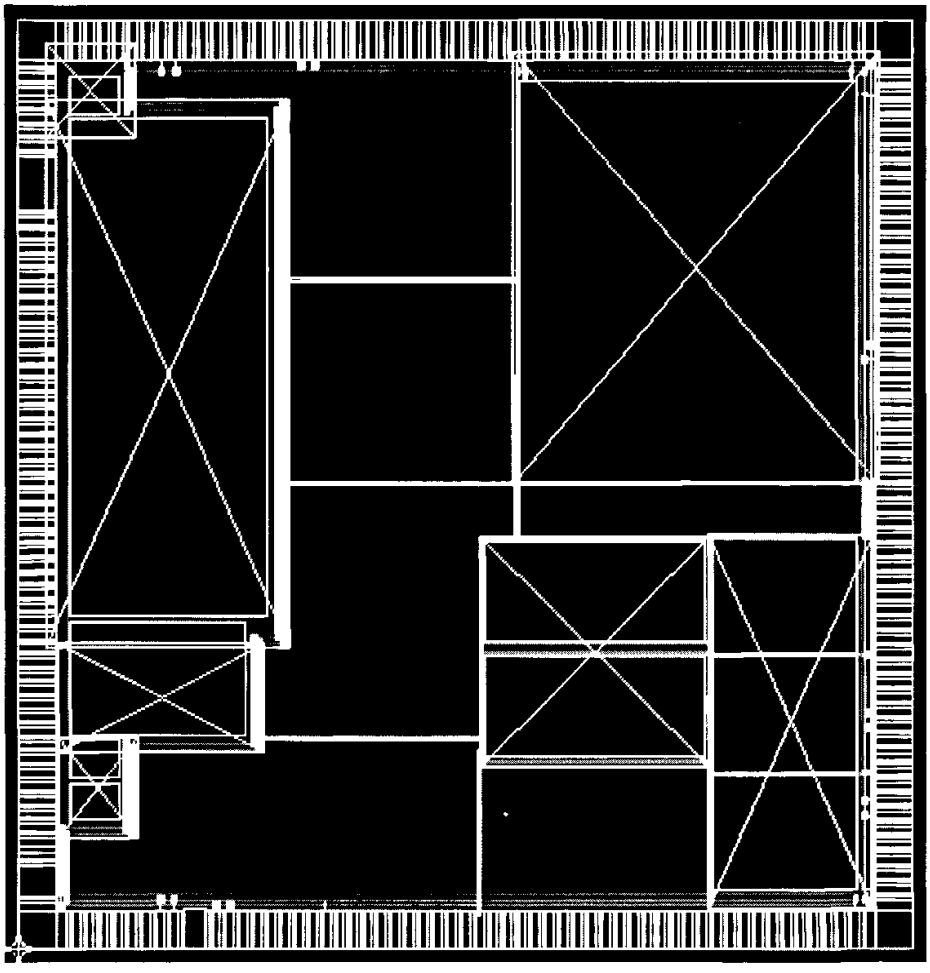

图 4-12　不合理的宏单元布局

4.4.1　优化 PMC 模块布局

对于 Garfield5 时钟树综合来说，最重要的一部分就是 PMC 时钟控制模块的摆放位置。如前面所分析的 Garfield5 时钟结构所述，PMC 模块中包含了时钟分频器，初级门控单元，时钟选择器等。这些都是要在时钟综合中需要特别关注的元件。可以通过在 Astro 中限定一个合适的区域用来放置 PMC 模块，这样就优化了 PMC 模块的位置。这块区域最好靠近 PLL 或者处于芯片中央。这样也就将时钟树的初始级的门控单元和时钟选择器限制在这个区域里了。且框定一块区域也可以减小 PMC 模块中各时序路径的长度，使得时钟树综合可以更好的平衡。在 Astro 中，可以将功耗管理模块里的逻辑单元定义为一个群(group)。 其命令格式如下：

```
define _cell (geGetEditCell)
axCreateGroup (geGetEditCell) "Clk_group"
axAddCellToGroup (geGetEditCell) "Clk_group" "gated_cell_1"
axAddCellToGroup (geGetEditCell) "Clk_group" "gated_cell_2"
.....
axAddCellToGroup (geGetEditCell) "Clk_group" "gated_cell_n"
```

然后在版图上划定一个物理区域，规定这个群里的逻辑单元只能放置在该区域中，可以设置这个区域的物理约束为：

axCreateRegion (geGetEditCell) "H" '(1800 2000) '(2300 2500)

表 4-6 是进行了 PMC 模块位置优化后的时序对照。可见 PMC 的位置摆放优化可以较大的改善电路时序。

表 4-6 PMC 模块位置优化前后的时序

	时钟延迟最大/最小值(ns)	时钟偏差(ns)	插入缓冲器数目
CLK5M_A	2.081/1.851	0.23	643
CLK5M_B	2.031/1.839	0.192	582

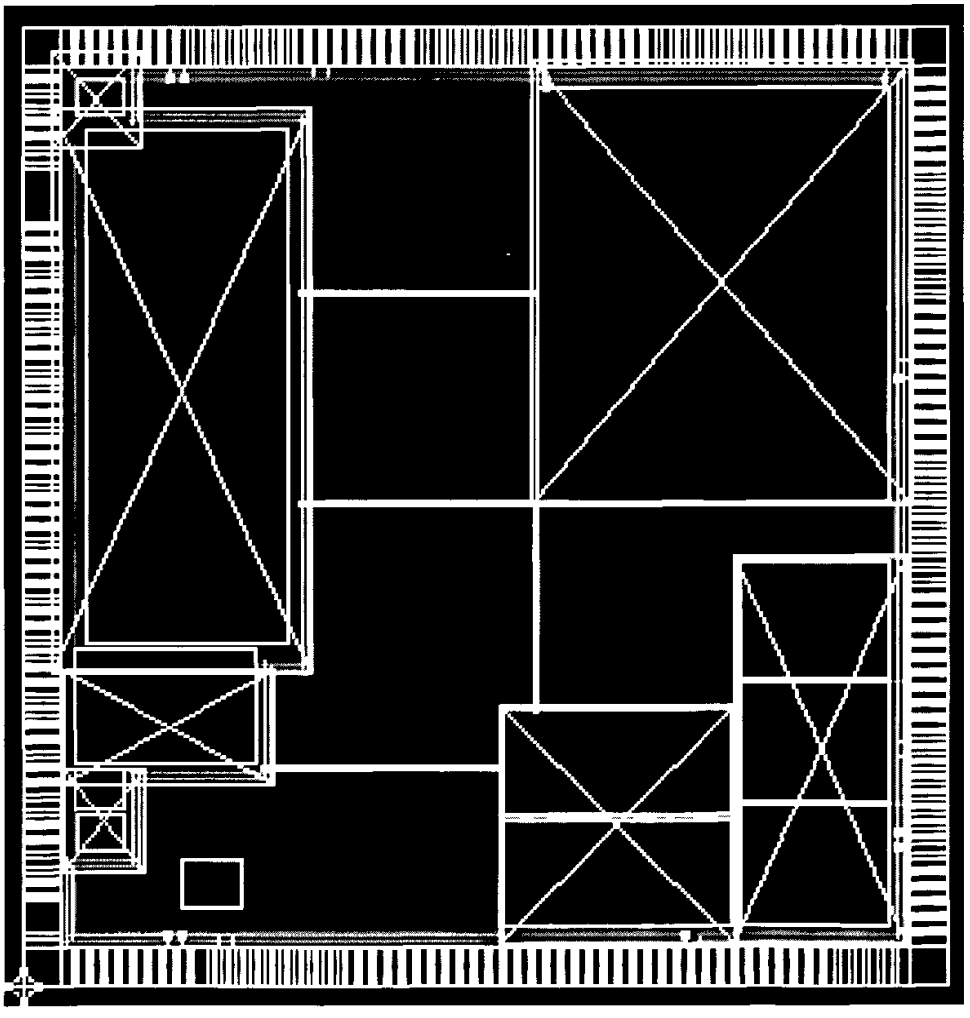

图 4-13 PMC 模块靠近边角处的摆放位置

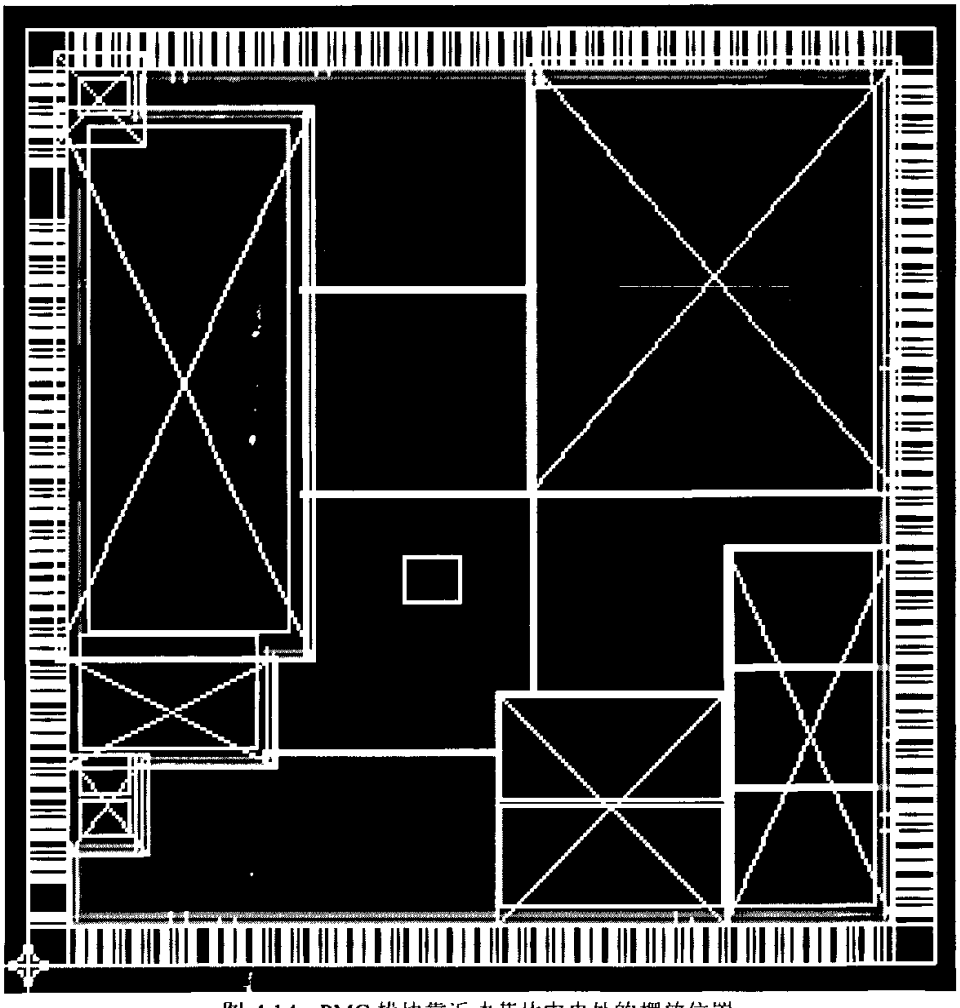

图 4-14　PMC 模块靠近边芯片中央处的摆放位置

4.4.2　调整门控时钟连线权重

在 PMC 模块中包含 A720T, AHB, LCDC, MMC, UART, DMAC, PWM, SPI, AC97, CODEC 和 ADC 模块的初级门控时钟单元。这些门控时钟单元位置摆放对后面的时钟树综合也有很重要影响。

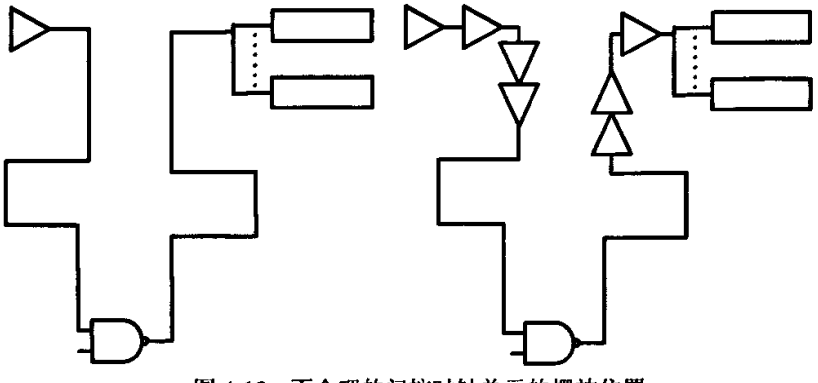

图 4-15　不合理的门控时钟单元的摆放位置

如果门控时钟单元的位置摆放不合理，会引入较大的时钟树延时，这样会导致进行时钟树综合时引入大量的时钟缓冲器和倒相器或者长的连线，导致芯片的面积和功耗的增大。同时较大的时钟延时也会影响电路的时序，造成时钟树上某些时钟门控单元的路径不满足时序设计的要求。如图 4-15 所示的门控时钟逻辑单元的位置远离寄存器和时钟源，依靠后端设计工具进行自动的时钟树综合时，结果就会插入大量的时钟缓冲器和长的时钟连线，造成时钟树延迟的增大。所以在布局和时钟树综合时都要尽量的使门控时钟单元的位置靠近时钟源点，使它的连线尽量的缩短。

连线权重是设置了标准单元布局时的连线的优先级，基于 Milkyway 数据库的工具在标准单元布局时将连接在高优先级连线上的标准单元尽力放在一起，使高优先级的连线尽可能的短[10]。这个方案的思路就是找到时钟树上和门控逻辑单元相连的连线，增大其权重，在设计布局和时钟树综合时 Astro 就会根据所设的连线权重值来努力的使该连线尽量的短，从而优化了门控时钟单元的位置，减小了时钟树延迟。其具体方法如下：

首先可以在 PrimeTime 中找到时钟树上连接门控逻辑单元的连线。其命令如下所示：

get_nets –of [get_clock_network –object –type gating_clock_pins]

然后在 Astro 中通过 TDF 文件来设置这些连线的权重。其在 TDF 文件中增加的内容如下：

netweight "gating_net" 255 255

其中 255 是连线权重的值，取值范围在 1～255 之间。在时钟树综合之前的标准单元布局中导入以上约束，可以优化时钟门控单元的布局。以 Garfield5 中的 CLK5M 为例，表 4-6 列出了两种情况下的时序比较，CLK5M_A 是不使用任何改变连线权重的方案，直接借助 EDA 工具综合，CLK5M_B 是使用增加连线权重方案来改善门控逻辑单元布局的结果。

表 4-6　加入连线权重的时序比较

	时钟延迟最大/最小值(ns)	时钟偏差(ns)	插入缓冲器数目
CLK5M_A	2.031/1.839	0.192	582
CLK5M_B	1.918/1.755	0.163	503

如果在系统芯片设计中，利用 Design Compiler 进行工具自动化的插入门控时钟单元，就会得到级数很多的门控时钟单元的路径。这在以前的 Garfield 系列芯片中实现过，其中 LCDC 模块的门控时钟达到七级之多。由于工具无法自动的对门控单元的位置进行优化，它使得时钟树综合时很难达到平衡。这样的情况下，通常会将级数多的时钟树切割成小的时钟树来处理，使它的门控时钟级数控制在五级以内。Garfield5 中没有自动插入门控时钟，这里就不作详述。

4.4.3　调整时钟源点的位置

在时钟源设定好以后，通过在 Astro 中查看时钟树的结构，以 CLK5M 为例。它的原始时钟源点是从芯片外部引脚直接引入的一个时钟管脚。这个时钟管脚处在芯片左上角，而同时经过时钟网络的传递，它的时钟汇点散布在芯片的各个位置，那么靠近时钟源位置的寄存器和远离时钟源的寄存器之间的时钟汇点就很难平衡。经过多次的实验证明时钟源点的位置对整个时钟树的影响非常重要，对于时钟树的延时和时钟偏差都有重要的影响。为了得到好的时钟树综合与分布结果，我们可以通过在时钟源后插入两级时钟缓冲器，将时钟源移至芯片的中央位置。但这首先需要修改网表，引入这两级时钟缓冲器。如果通过修改前端的门级网表文件的方法来做这个改动，就需要返回至到芯片布局的步骤，这样耗时很多，一般通过简单的 ECO（Engineering Change Order）的方法就可以进行插入缓冲器的改动。图 4-16 所示的是没有经过改动的时钟源点的位置，通过飞线（fly lines）可以看到，它所控制的时钟端口在芯片的各个角落。

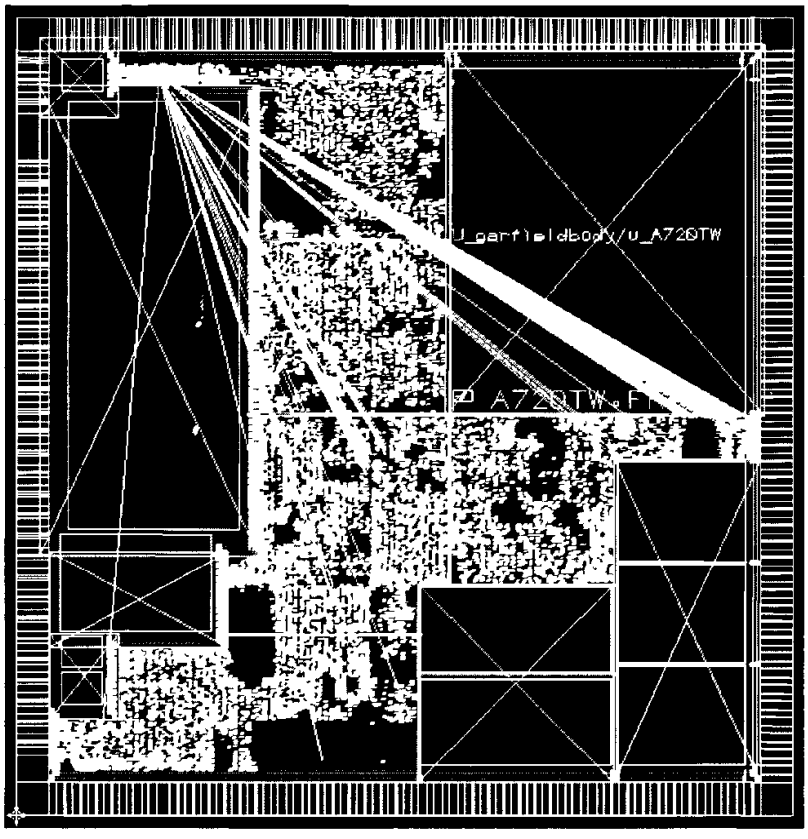

图 4-16　位于边角位置的时钟源点

下面的 Pad_in_CLK5M/XC 是未做改动之前的时钟源头，U_buf1 是 CLKBUFX4，U_buf2 是 CLKBUFX8，这样可以使驱动逐级增大。原时钟定义为：

```
create_clock –name "CLK5M" –period 9 –waveform {0 4.5} [get_pins {Pad_in_CLK5M/XC}]
```

对应的 ECO 改动文件如下[10]：

```
; Insert two clock buffers at the clock root CLK5M
+I CLKBUFX4 U_buf1
+I CLKBUFX8 U_buf2
+N New_net1
+N New_net2
-P XC PAD_IN_CLK5M Clk
+P XC PAD_IN_CLK5M New_net1
+P A U_buf1 New_net1
+P A U_buf2 New_net2
+P Y U_buf1 New_net1
+P Y U_buf2 Clk
```

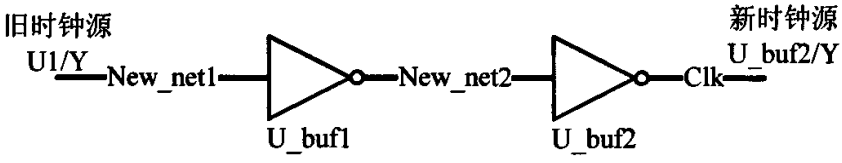

图 4-17　时钟源插入缓冲器的示意图

这样在逻辑网表上就完成了改动，然后我们将新增的两个时钟缓冲器在版图上放置，使得时钟源点处在芯片得中央的位置。如图4-17所示。为了提供足够的时钟驱动能力，最好是将前一级时钟缓冲器靠近时钟源放置，后一级时钟缓冲器放在较远的位置。

新的时钟源的创建命令是：

create_clock –name CLK5M –period 9 –waveform {0 4.5} [get_pins U_buf2/Y]

表4-7是做此调整前后产生的时钟树综合结果比较：

表4-7 时钟源位置调整前后的时序比较

	时钟偏差(ns)	最大时钟延迟(ns)	插入缓冲器数目
原CLK5M	0.163	1.918	503
调整后CLK5M	0.158	1.750	486

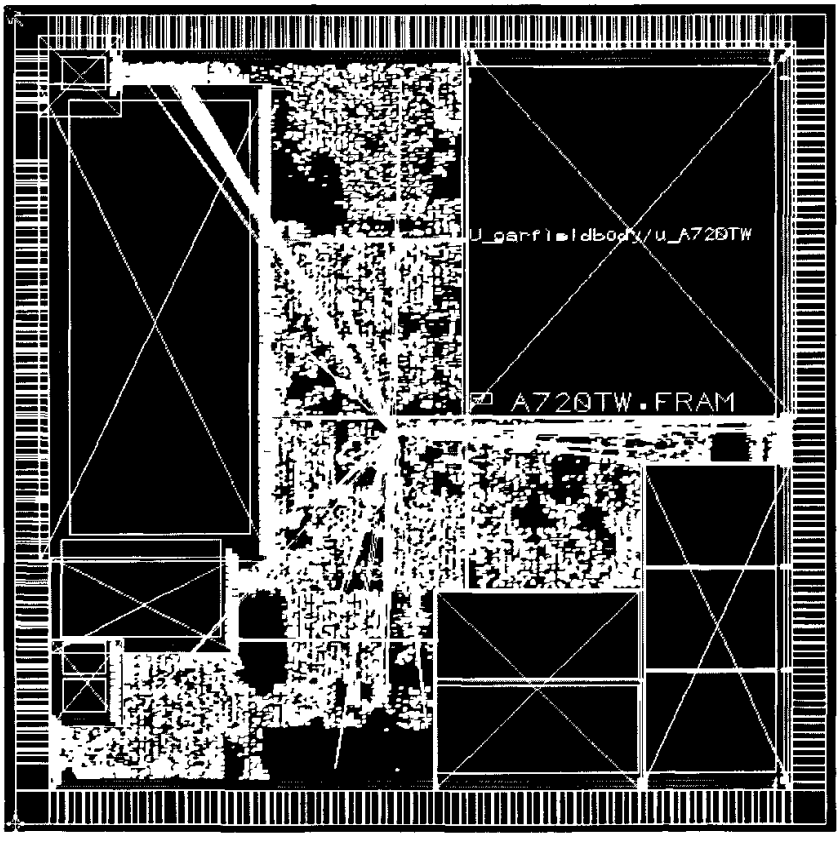

图4-18 时钟源处于芯片中央的位置

4.5 本章小结

本章详细介绍了 Garfield5 的时钟树综合实现流程，具体包括时钟树综合的数据准备，时钟树结构的查看与调整，时钟树自动生成以及时钟树优化方法（PMC 模块的位置优化、门控时钟单元连线权重设置及时钟源点位置调整）。并讨论了流程中各个步骤对时钟树综合的作用和影响。最后得到了满足设计要求的系统主时钟（CLK5M）时钟树的时钟偏差控制在 0.158ns 以内及最大时钟树延时调整到 1.8ns。

第五章 总结与展望

高性能的系统芯片中，时钟信号是统一的时间参考，时钟周期也是系统性能的一个基本指标。时钟树作为时钟信号的传播网络，对于系统功能和性能都具有重大影响，时钟树综合与分布成为后端物理设计的一个重点和难点。

本文结合东南大学工程中心 Garfield5 系统芯片的项目，介绍了时钟树综合的基本理论、功耗管理模块、时钟树综合自动实现流程和时钟树优化方法（优化 PMC 模块位置、门控时钟连线权重设置及时钟源点的位置优化）。

随着工艺尺寸的缩小，当制造工艺在 0.13um 以下时，系统芯片时钟的信号完整性问题会更为严峻。同时时钟偏差并不随着工艺等比例缩小而等比例缩小，且时钟偏差是和每一个局部路径相关的。因此，工艺特征尺寸等比例缩小时，时钟树的综合与分布变得十分复杂。时钟树综合与分布进一步工作和研究方向：

- 复杂时钟树结构的综合与分布：现代系统芯片由于高性能和低功耗的要求，时钟结构会越来越复杂。时钟树的综合与分布之所以成为后端物理设计的难点，主要的原因在于目前没有 EDA 工具能有效的自动完成复杂时钟结构的综合与分布。
- 工艺敏感的时钟树综合与分布：时钟树上电阻、电容、电感都和工艺参数紧密相关，当工艺特征尺寸小于 90nm 以后，工艺参数不对称性的影响会极为严重，会给平衡时钟树和控制时钟偏差带来很大困难。如果能在时钟树综合与分布阶段考虑到工艺参数的影响，时钟树设计上采取有效的工艺补偿技术，会大大改善时钟树的质量。
- 系统设计级的时钟树综合：目前时钟树综合完全在物理版图阶段进行，很少涉及前端系统设计和结构设计，而实际上，高层次上的时钟规划对时钟树综合与分布结果的优化程度更大，效果更为显著，而且能降低后端物理设计阶段中时钟树综合与分布的难度。
- 微波级高频率的时钟树综合与分布：随着系统需求的不断增加，系统芯片的时钟频率会越来越快，时钟树综合与分布的约束会更加苛刻。而当频率到达微波量级，芯片上的寄生电感会对时钟连线的阻抗产生重大影响，从而需要新的时序延时模型和方法来进行时钟树综合与分布。
- 时钟树综合与分布结果的调试和测试。目前 IC 设计流程中，时钟树生成后，要经过信号布线、寄生参数提取及静态时序分析来评价时钟树综合与分布结果，然后根据分析结果确定优化和调整方案。当静态时序分析的结果不能达到设计的要求，就有可能重新进行时钟树的综合与分布，这个迭代过程是很耗时的，延长了芯片后端设计周期。目前 IC 设计流程中并没有针对时钟树平衡的状况进行专门的芯片产品测试，而时钟信号的传播质量是系统性能和功能的基石，在高性能芯片设计中，对时钟树进行产品测试也将成为迫切的需求。

致谢

本课题从选题、收集材料、制定方案和具体实施，至始至终都是在导师陆生礼老师、罗岚老师和杨军老师的悉心指导和大力帮助下完成的。在本人的整个硕士研究生生活时期，他们也给予我以学习、生活、等各个方面的极大关心与照顾。在此致以由衷的感谢。

在本课题的研究过程中，时龙兴老师、陆生礼老师都给予我了很大的帮助和研究方向上的指导，在此向他们表示衷心的谢意。

另外，我也要感谢 ASIC 工程中心的凌明老师和钟锐老师以及其它所有帮助我的老师，感谢他们在我的硕士研究生期间对我的谆谆教诲。

同时我要特别感谢罗岚老师和杨军老师，感谢他们在我做课题期间给予我的技术指导和协助，在此向他们表示由衷的感谢。

黄凯同学、王丽英同学、毕宗军同学等也都在我的课题研究过程中给予了我许多的帮助。在共同学习期间，他们对我的研究提出了很多有价值的参考意见，在此向他们表示感谢。

此外，我要感谢我的父母，感谢我的同学和朋友，他们给了我最大的精神支持。

附录

A: Pre-CTS timing setup 设置脚本

```
atTimingSetup
atCmdSetField "Ignore Interconnect" "0"
atCmdSetField "Enable Time Borrowing" "1"
atCmdSetField "Enable Preset/Clear Arcs" "1"
atCmdSetField "Enable Recovery/Removal Arcs" "1"
atCmdSetField "Enable Mixed Clock/Signal Edges" "1"
atCmdSetField "Enable Scan Enable" "1"
atCmdSetField "Enable Inter Clock" "1"
atCmdSetField "Enable Default Clock" "1"
atCmdSetField "Include Library Max Tran" "1"
atCmdSetField "Include Library Max Cap" "1"
atCmdSetField "Enable Multi-Clocks Per Reg" "1"
atCmdSetField "Enable CRPR" "0"
atCmdSetField "Enable Data Check" "1"
atCmdSetField "Ignore Clock Uncertainty" "0"
atCmdSetField "Ignore Propagated Clock" "0"
atCmdSetField "Set IO Clock Latency" "1"
atCmdSetField "Enable Ideal Network Delay" "0"
atCmdSetField "Enable Gated Clock Checks" "1"
atCmdSetField "Enable CrossTalk Effects" "0"
atCmdSetField "Set Useful Skew" "0"
atCmdSetField "Include Non Propagated Nets" "1"
atCmdSetField "Include SyncPort PhaseDelay" "1"
atCmdSetField "Enable Trace Mode" "0"
atCmdSetField "Include IO Path" "1"
atCmdSetEnvModel
```

B: Post-CTS timing setup 设置脚本

```
atTimingSetup
atCmdSetField "Ignore Interconnect" "0"
atCmdSetField "Enable Time Borrowing" "1"
atCmdSetField "Enable Preset/Clear Arcs" "1"
atCmdSetField "Enable Recovery/Removal Arcs" "1"
atCmdSetField "Enable Mixed Clock/Signal Edges" "1"
atCmdSetField "Enable Scan Enable" "1"
atCmdSetField "Enable Inter Clock" "1"
atCmdSetField "Enable Default Clock" "1"
atCmdSetField "Include Library Max Tran" "1"
atCmdSetField "Include Library Max Cap" "1"
atCmdSetField "Enable Multi-Clocks Per Reg" "1"
atCmdSetField "Enable CRPR" "0"
atCmdSetField "Enable Data Check" "1"
atCmdSetField "Ignore Clock Uncertainty" "0"
atCmdSetField "Ignore Propagated Clock" "0"
atCmdSetField "Set IO Clock Latency" "1"
atCmdSetField "Enable Ideal Network Delay" "0"
atCmdSetField "Enable Gated Clock Checks" "1"
atCmdSetField "Enable CrossTalk Effects" "0"
atCmdSetField "Set Useful Skew" "0"
atCmdSetField "Include Non Propagated Nets" "1"
atCmdSetField "Include SyncPort PhaseDelay" "1"
atCmdSetField "Enable Trace Mode" "0"
atCmdSetField "Include IO Path" "1"
atCmdSetEnvModel
```

C: 时序约束文件

```
create_clock -name "CLK5M" -period 9 -waveform {0 4.5} [get_pins {Pad_in_CLK5M/XC}]
create_clock -name "CLK32" -period 31250 -waveform {0 15625} [get_pins
{Pad_32768K_pad/XC}]
create_clock -name "TCK" -period 9 -waveform {0 4.5} [get_pins {Pad_in_TCK/C}]
create_clock -name "USBCLK" -period 20 -waveform {0 10} [get_pins
{U_garfieldbody/U_pm/U_PM_UPLL/MXout_clk_USBD/Y}]
create_clock -name "CODECCLK" -period 20 -waveform {0 10} [get_pins
{U_garfieldbody/U_pm/U_PM_CPLL/MXout_clk_CODEC_PLL/Y}]
create_clock -name "SYSCLK" -period 9 -waveform {0 4.5} [get_pins
{U_garfieldbody/U_pm/U_PM_ctrl/U57/Y}]
create_generated_clock -name "SDCLK" -source [get_pins
{U_garfieldbody/U_pm/U_PM_ctrl/U57/Y}] -divide_by 1 [get_pins
{U_garfieldbody/U_pm/U_PM_interface/SDCLKBUF/Y}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA31_PC0}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA30_PC1}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA29_PC2}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA28_PC3}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA27_PC4}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA26_PC5}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA25_PC6}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA24_PC7}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA23_PC8}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA22_PC9}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA21_PC10}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA20_PC11}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA19_PC12}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA18_PC13}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA17_PC14}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA16_PC15}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA0}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA1}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA2}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA3}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA4}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA5}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA6}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA7}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA8}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA9}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA10}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA11}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA12}]
```

```
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA13}]
set_input_delay 3 -clock "SDCLK" [get_ports {inout_DATA14}]
set_false_path -from [get_clocks CLK5M]    -to [get_clocks CLK32]
set_false_path -from [get_clocks CLK32]    -to [get_clocks CLK5M]
set_false_path -from [get_clocks CLK5M]    -to [get_clocks TCK]
set_false_path -from [get_clocks TCK]      -to [get_clocks CLK5M]
set_false_path -from [get_clocks CLK32]    -to [get_clocks TCK]
set_false_path -from [get_clocks TCK]      -to [get_clocks CLK32]
set_false_path -from [get_clocks CLK32]    -to [get_clocks SYSCLK]
set_false_path -from [get_clocks SYSCLK]   -to [get_clocks CLK32]
set_false_path -from [get_clocks USBCLK]   -to [get_clocks SYSCLK]
set_false_path -from [get_clocks SYSCLK]   -to [get_clocks USBCLK]
set_false_path -from [get_clocks CODECCLK] -to [get_clocks SYSCLK]
set_false_path -from [get_clocks SYSCLK]   -to [get_clocks CODECCLK]
set_false_path -from [get_ports in_Resetn]
set_false_path -from [get_ports in_WakeUp]
set_false_path -from [get_ports in_nTRST]
set_false_path -from [get_ports in_NANDBOOT]
set_false_path -from [get_ports in_BOOT32]
set_false_path -from [get_ports in_PageSize]
set_false_path -from [get_ports in_AddrCycle]
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_Core
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_AHB
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_APB
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_LCDC
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_MMC
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_ESRAM
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_DMAC
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_EMI
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_INTC
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_GPT
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_RTC
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_UART1
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_UART2
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_SPI
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_PWM
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_AC97
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_GPIO
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_PMC
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_DSP
set_disable_timing -from A -to Y U_garfieldbody/U_pm/U_PM_interface/MXout_clk_CODEC
.....
```

D: Ignore pin 设置脚本

```
dbPurgeIgnorePin (geGetEditCell)  "*"
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_DSP" '("A")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/MXout_clk_CODEC" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_PMC" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_GPIO" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_AC97" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_PWM" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_SPI" '("A")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/MXout_clk_UART2" '("A")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/MXout_clk_UART1" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_RTC" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_GPT" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_INTC" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_EMI" '("A")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/MXout_clk_DMAC" '("A")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/MXout_clk_ESRAM" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_MMC" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_LCDC" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_APB" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/MXout_clk_AHB" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_UPLL/MXout_clk_USBD" '("A")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_CPLL/MXout_clk_CODEC_PLL" '("A")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedDSPEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedADCEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedCODECEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedAC97Enable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedPWMEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedSPIEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedUART1Enable_reg" '("GN")
```

```
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedDMACEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedESRAMEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedMMCEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedLCDCEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedAHBEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedCoreEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_interface/LatchedUART2Enable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_CPLL/CPLL_LatchedClockEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/clk_OfOscillator_div_reg" '("RN")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U_PM_interface/SDCLKBUF" '("A")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U20" '("B")
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/U_pm/U20" '("A")
```

E: Sync pin 设置脚本

```
dbPurgeSyncPin (geGetEditCell) "*"
dbDefineIgnorePin (geGetEditCell) "U_garfieldbody/u_dsp_top/DSPCLKn_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_UPLL/UPLL_LatchedClockEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_ctrl/U_PM_ClkSrcSel/MXout_ClkSource" '("A""B")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_ctrl/U_PM_ClkSrcSel/LatchedMPLLEnable_reg" '("GN")
dbDefineIgnorePin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_ctrl/U_PM_ClkSrcSel/LatchedOscillatorEnable_reg" '("GN")
dbDefineSyncPin (geGetEditCell)
"U_garfieldbody/U_pm/U_PM_ctrl/U_PM_ClkSrcSel/OscillatorOrMPLL_r_reg" '(("CK"
"nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/U_ADC/U_ADC_PANEL"
'(("ADCCLK_pan" "nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/U_CODEC" '(("in_Pclk" "nonInvertRise" (0.00
0.00))("in_Cclk" "nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/u_dsp_top/uXRAM/uram0" '(("CLK"
"nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/u_dsp_top/uXRAM/uram1" '(("CLK"
"nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/u_dsp_top/uXRAM/uram2" '(("CLK"
"nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/u_dsp_top/uYDPRM/udp0" '(("CLKA"
"nonInvertRise" (0.00 0.00))("CLKB" "nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/u_dsp_top/uYDPRM/udp1" '(("CLKA"
"nonInvertRise" (0.00 0.00))("CLKB" "nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/u_A720TW" '(("HCLK" "nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/U_UPLL" '(("I_ref" "nonInvertRise" (0.00 0.00)))
dbDefineSyncPin (geGetEditCell) "U_garfieldbody/U_MPLL" '(("I_ref" "nonInvertRise" (0.00 0.00)))
```

F: CTS 综合脚本

```
astClockOptions
formDefault "Clock Common Options"
setFormField "Clock Common Options" "Best" "1"
setFormField "Clock Common Options" "Typical" "1"
setFormField "Clock Common Options" "Skew Type" "Global"
setFormField "Clock Common Options" "Clock Nets"
"CLK32,CLK5M,TCK,U_garfieldbody/clk_USB,U_garfieldbody/clk_CODEC_PLL,
U_garfieldbody/U_pm/ClockSource"
setFormField "Clock Common Options" "Target Skew" 0.2
setFormField "Clock Common Options" "Optimization Effort" "5"
setFormField "Clock Common Options" "Synthesis Effort" "5"
setFormField "Clock Common Options" "Buffers/Inverters"
"CLKBUFX1,CLKBUFX2,CLKBUFX3,CLKBUFX4,CLKBUFX8,CLKBUFX12,CLKBUFX16,CLKBU
FX20"
setFormField "Clock Common Options" "Buffer Sizing: LEQ Cells" "CLKBUFX1 CLKBUFX2
CLKBUFX3 CLKBUFX4 CLKBUFX8 CLKBUFX12 CLKBUFX16 CLKBUFX20"
setFormField "Clock Common Options" "Delay Cells" "CLKBUFX1 CLKBUFX2 CLKBUFX3
CLKBUFX4 CLKBUFX8 CLKBUFX12 CLKBUFX16 CLKBUFX20"
formOK "Clock Common Options"
astCTS
formDefault "Clock Tree Synthesis"
setFormField "Clock Tree Synthesis" "Best" "1"
setFormField "Clock Tree Synthesis" "Typical" "1"
setFormField "Clock Tree Synthesis" "Skew Type" "Global"
formOK "Clock Tree Synthesis"
```

G: Astro 自动时钟树综合时钟偏差报表

Clock: SYSCLK
Pin: U_garfieldbody/U_pm/U_PM_ctrl/U57/Y
Net: U_garfieldbody/U_pm/ClockSource
Operating Condition = worst
The clock global skew = 0.267
The longest path delay = 2.055
The shortest path delay = 1.788
The longest path delay end pin: U_garfieldbody/U_gpio/U_A_8/ext_data_s2_reg/CK
The shortest path delay end pin:U_garfieldbody/U_rtc/u_sync_32toP/out_YMD_r_P_reg_25_/CK
Clock: CODECCLK
Pin: U_garfieldbody/U_pm/U_PM_CPLL/MXout_clk_CODEC_PLL/Y
Net: U_garfieldbody/clk_CODEC_PLL
Operating Condition = worst
The clock global skew = 0.000
The longest path delay = 0.740
The shortest path delay = 0.740
The longest path delay end pin: U_garfieldbody/U_CODEC/in_Cclk
The shortest path delay end pin: U_garfieldbody/U_CODEC/in_Cclk
Clock: USBCLK
Pin: U_garfieldbody/U_pm/U_PM_UPLL/MXout_clk_USBD/Y
Net: U_garfieldbody/clk_USB
Operating Condition = worst
The clock global skew = 0.187
The longest path delay = 1.906
The shortest path delay = 1.719
Clock: TCK
Pin: Pad_in_TCK/C
Net: TCK
Operating Condition = worst
The clock global skew = 0.022
The longest path delay = 1.237
The shortest path delay = 1.215
The longest path delay end pin:
U_garfieldbody/u_dsp_top/u_rambist_U_dpram_apg_1_U_DGEN_data_reg_mir_reg_14_/CK
The shortest path delay end pin:
U_garfieldbody/u_dsp_top/u_rambist_U_spram_apg_1_U_DGEN_data_reg_mir_reg_2_/CK
Clock: CLK32
Pin: Pad_32768K_pad/XC
Net: CLK32
Operating Condition = worst
The clock global skew = 0.179
The longest path delay = 1.508

The shortest path delay = 1.330

The longest path delay end pin: U_garfieldbody/U_rtc/u_sync_Pto32/HMS_r_32_reg_24_/CK

The shortest path delay end pin:

U_garfieldbody/U_pm/U_PM_ctrl/U_PM_WakeUp/WakeUpQualifyCounter_reg_12_/CK

Clock: CLK5M

Pin: Pad_in_CLK5M/XC

Net: CLK5M

Operating Condition = worst

The clock global skew = 0.230

The longest path delay = 2.081

The shortest path delay = 1.851

The longest path delay end pin:

U_garfieldbody/U_pm/U_PM_CPLL/MXout_clk_CODEC_PLL/B

The shortest path delay end pin: U_garfieldbody/Sync_CrystallEnableButGateIt_reg/CKN

H: 优化后的时钟偏差报表

Clock: SYSCLK
Pin: U_garfieldbody/U_pm/U_PM_ctrl/U57/Y
Net: U_garfieldbody/U_pm/ClockSource
Operating Condition = worst
The clock global skew = 0.178
The longest path delay = 2.001
The shortest path delay = 1.833
The longest path delay end pin: U_garfieldbody/U_gpio/U_A_8/ext_data_s2_reg/CK
The shortest path delay end pin:U_garfieldbody/U_rtc/u_sync_32toP/out_YMD_r_P_reg_25_/CK
Clock: CODECCLK
Pin: U_garfieldbody/U_pm/U_PM_CPLL/MXout_clk_CODEC_PLL/Y
Net: U_garfieldbody/clk_CODEC_PLL
Operating Condition = worst
The clock global skew = 0.000
The longest path delay = 0.740
The shortest path delay = 0.740
The longest path delay end pin: U_garfieldbody/U_CODEC/in_Cclk
The shortest path delay end pin: U_garfieldbody/U_CODEC/in_Cclk
Clock: USBCLK
Pin: U_garfieldbody/U_pm/U_PM_UPLL/MXout_clk_USBD/Y
Net: U_garfieldbody/clk_USB
Operating Condition = worst
The clock global skew = 0.187
The longest path delay = 1.906
The shortest path delay = 1.719
Clock: TCK
Pin: Pad_in_TCK/C
Net: TCK
Operating Condition = worst
The clock global skew = 0.022
The longest path delay = 1.237
The shortest path delay = 1.215
The longest path delay end pin:
U_garfieldbody/u_dsp_top/u_rambist_U_dpram_apg_1_U_DGEN_data_reg_mir_reg_14_/CK
The shortest path delay end pin:
U_garfieldbody/u_dsp_top/u_rambist_U_spram_apg_1_U_DGEN_data_reg_mir_reg_2_/CK
Clock: CLK32
Pin: Pad_32768K_pad/XC
Net: CLK32
Operating Condition = worst
The clock global skew = 0.179
The longest path delay = 1.508

The shortest path delay = 1.330
The longest path delay end pin: U_garfieldbody/U_rtc/u_sync_Pto32/HMS_r_32_reg_24_/CK
The shortest path delay end pin:
U_garfieldbody/U_pm/U_PM_ctrl/U_PM_WakeUp/WakeUpQualifyCounter_reg_12_/CK
Clock: CLK5M
Pin: Pad_in_CLK5M/XC
Net: CLK5M
Operating Condition = worst
The clock global skew = 0.158
The longest path delay = 1.908
The shortest path delay = 1.750
The longest path delay end pin:
U_garfieldbody/U_pm/U_PM_CPLL/MXout_clk_CODEC_PLL/B
The shortest path delay end pin: U_garfieldbody/Sync_CrystallEnableButGateIt_reg/CKN

参考文献

【1】 Jan M.Rabaey,Anantha Chandrakasan and Borivoje Nikolie.数字集成电路-设计透视（第2版）. 北京：清华大学出版社，2004年.

【2】 E. G. Friedman and J. H. Mulligan Jr., "Pipelining and clocking of high performance synchronous digital systems," in VLSI Signal Processing Technology, M. A. Bayoumi and E. E. Swartzlander Jr., Eds. Norwell, MA: Kluwer, 1994, pp. 97–133.

【3】 D. Wann and M. Franklin, "A synchronous and clocked control structures for VLSI based interconnection networks," IEEE Trans. Comput., vol. C-32, pp. 284–293, Mar. 1983.

【4】 E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits", Proceedings of the IEEE Volume 89, Issue 5, May 2001 Page(s):665-692.

【5】 S. Dhar, M. Franklin, and D. Wann, "Reduction of clock delays in VLSI structures," in Proc. IEEE Int. Conf. Computer Design, Oct. 1984, pp. 778–783.

【6】 J. P. Fishburn, "Clock skew optimization," IEEE Trans. Comput., vol. 39, pp. 945–951, July 1990.

【7】 周凤亭.基于PC+ASTRO的深亚微米布局布线流程研究.东南大学.2005.

【8】 E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits", Proceedings of the IEEE Volume 89, Issue 5, May 2001 Page(s):665-692.

【9】 Monica Donno, Enrico Macii, Luca Mazzoni, "Power-Aware Clock Tree Planning", ISPD'04, April 18–21, ACM 1-58113-817, 2004.

【10】 Synopsys.Astro User Guide.Version V-2004.06

【11】 P. Ramanathan and K. G. Shin, "A clock distribution scheme for nonsymmetric VLSI circuits," in Proc. IEEE Int. Conf. Computer-Aided Design, Nov. 1989, pp. 398–401.

【12】 W. Khan and N. Sherwani, "Zero skew clock routing algorithm for high performance ASIC systems," in Proc. IEEE Int. Conf. ASICs, Sept. 1993, pp. 79–82.

【13】 D. A. Joy and M. J. Ciesielski, "Placement for clock period minimizationwith multiple wave propagation," in Proc. ACM/IEEE 28th Design Automation Conf., June 1991, pp. 640–643.

【14】 W. D. Grover, "A new method for clock distribution," IEEE Trans.Circuits Syst. I, vol. 41, pp. 149–160, Feb. 1994.

【15】 D. R. Gonzales, "Micro-RISC Architecture for the Wireless Market," IEEE Micro, Vol. 19, No. 4, pp. 30-37, July-August 1999.

【16】 李芝燕.高速性能的时钟布线算法研究.浙江大学.1999.

【17】 M. Edahiro, "A clock net reassignment algorithm using Voronoi diagrams," in Proc. IEEE Int. Conf. Computer-Aided Design, Nov. 1990, pp. 420–423.

【18】 D. G. Messerschmitt, "Synchronization in digital system design," IEEE J. Select. Areas Commun., vol. 8, pp. 1404–1419, Oct. 1990..

【19】 H. B. Bakoglu, J. T. Walker, and J. D. Meindl, "A symmetric clock distribution tree and.

optimized high-speed interconnections for reduced clock skew in ULSI and WSI circuits," in Proc. IEEE Int. Conf. Computer Design, Oct. 1986, pp. 118–122.

【20】 Q. Zhu and W. W. M. Dai, "High-speed clock network sizing optimization based on distributed RC and lossy RLC interconnect models," IEEE Trans. Computer-Aided Design, vol. 15, pp. 1106–1118, Sept. 1996.

【21】 A. Vittal and M. Marek-Sadowska, "Power optimal buffered clock tree design," in Proc. ACM/IEEE Design Automation Conf., June 1995, pp. 497–502.

【22】 王家正.Garfield系统级功耗分析及管理.东南大学.2005.

【23】 SMIC, "SMIC 0.18um Process 1.8-Volt SAGE-X Standard Cell Library DateBook", release 4.0.

【24】 M.A.B.Jackson. A Srinivasan and E.S.Kuh, "Clock routing for high performance ".ICs,Proc,ACM/IEEE Design Automation Conf.1990,pp573-579

【25】 Wayne Wolf, 《现代VLSI电路设计》(英文影印第三版), 科学出版社, 2003.

【26】 M. S. McGregor, P. B. Denyer, and A. F. Murray, "A single-phase clocking scheme for CMOS VLSI," in Proc. Stanford Conf. Advanced Research in VLSI, Mar. 1987, pp. 257–271.

【27】 K. A. Sakallah, T. N. Mudge, T. M. Burks, and E. S. Davidson, "Synchronization of pipelines," IEEE Trans. Computer-Aided Design, vol. 12, pp. 1132–1146, Aug. 1993.

【28】 W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "A unified algorithm for gate sizing and clock skew optimization to minimize sequential circuit area," in Proc. IEEE Int. Conf. Computer-Aided Design, Nov.1993, pp. 220–223.

【29】 金肖科.SoC IP硬核复用技术的研究[硕士学位论文].东南大学.2004

【30】 Synopsys.PrimeTime User Guide. Version V-2004.06

硕士期间发表论文

1．汪珺，罗岚. Garfield5 微处理器芯片的电源网络和面积优化. 电子器件. 2006,第二十九卷第 3 期