Mechanical Engineering and Technology 机械工程与技术, 2022, 11(5), 507-513
Published Online October 2022 in Hans. http://www.hanspub.org/journal/met
https://doi.org/10.12677/met.2022.115059

Hans汉斯

基于28 nm工艺的SOC芯片时钟树优化

侯　宇，王　爽

天津职业技术师范大学电子工程学院，天津

收稿日期：2022年9月12日；录用日期：2022年10月12日；发布日期：2022年10月21日

摘　要

针对SOC芯片设计中时钟树的综合效率和时序收敛问题，提出一种高效的时钟树综合方法，特别适用于现代先进深亚微米工艺中高度集成和高复杂度的设计。传统的时钟树综合方法，通过从下到上采用分步综合的方法进行了改进。该设计方法在基于台积电28 nm工艺的CPU芯片中成功进行了流片验证，结果表明，在实现传统设计功能并完成时序收敛的前提下，逐步去综合可以减少不必要的器件插入，减小芯片面积，降低整体功耗。

关键词

时序收敛，时钟树综合(CTS)，缓冲器，时钟偏移

SOC Chip Clock Tree Optimization Based on 28 nm Process

Yu Hou, Shuang Wang

School of Electronic Engineering, Tianjin University of Technology and Education, Tianjin

Received: Sep. 12th, 2022; accepted: Oct. 12th, 2022; published: Oct. 21st, 2022

Abstract

Aiming at the comprehensive efficiency and timing convergence of clock trees in SOC chip design, an efficient clock tree synthesis method is proposed, which is especially suitable for highly integrated and high complexity designs in modern advanced deep submicron processes. The traditional clock tree synthesis method has been improved by adopting a step-by-step synthesis method from bottom to top. The design method was successfully tested in the CPU chip based on TSMC's 28 nm process, and the results showed that under the premise of realizing the traditional design function and completing the timing closure, the gradual de-synthesis can reduce unnecessary device insertion, reduce the chip area, and reduce the overall power consumption.

文章引用: 侯宇, 王爽. 基于 28 nm 工艺的 SOC 芯片时钟树优化[J]. 机械工程与技术, 2022, 11(5): 507-513.
DOI: 10.12677/met.2022.115059

Keywords

Temporal Convergence, Clock Tree Synthesis (CTS), Buffer, Clock Skew

1. 简介

集成电路是基础性、先导性和战略性产业，其产品渗透着大到军工企业，小到日常生活用品，在信息技术领域中发挥着重要的作用。推动集成电路产业的发展成为了实现网络信息安全，实现核心技术自主可控，不可或缺的关键一步[1]。

时钟信号不仅仅是芯片设计当中整个数字电路中驱动负载最多、传送距离最大的信号翻转速率最高的信号，也是翻转速率最高的驱动负载最多、传送距离最长的信号[2]。在时钟网络设计过程中，工具先通过互联线的长度，利用 RC 计算出线上的延时算出时钟线上的延时，最后再在时钟线上通过插入具有不同时钟驱动能力的时钟缓冲剂与时钟反相器以达到平衡时钟定义点时钟源连接到各个寄存器时间单元上的时钟，抛开驱动器类型的不同，这一点将会尽可能地确保时钟的使能和信号能同步传输至每个时钟单元上的信号输入端，为保证数据有效传输奠定了基础[3]。这种根据芯片时钟网络的约束形成时钟树的过程称为时钟树综合[4]。随着工艺节点的提升，对于数字后端设计而言，时钟树综合已经成为整个后端设计的难点。由于芯片设计越来越复杂，后端工具在做时钟树综合时，表现出来的结果也不尽如人意。时钟树综合目标追求六小一多一平衡，分别为转换时间(slew)、时钟偏移(skew)、时钟源延迟(insertion delay)、面积(area)、功耗(power)和串扰(crosstalk)小，一多为时钟公共路径多，一平衡为波形平衡[5]。为达到理想的结果，工具的不智能性日益体现，人为干预 EDA 工具的时钟树综合已成为数字后端设计中常规的操作方式。本文设计采用 28 nm 工艺，芯片规模 120 多万门，主频时钟 800 MHZ，并对原有的时钟设计方案进行了优化，使其 skew、power 以及 timing violation 有了明显的改善。

2. 实验方法手段

本文使用的时钟树综合工具是 Cadence 公司的 SoC Innovus，其内部集成了 CTS 引擎。Innovus 的时钟树综合在布局(placement)结束之后执行，有手动模式和自动模式两种。手动模式下，可以人为选择时钟树布线所用的金属层、插入驱动单元的总量及其每个金属层所采用驱动单元的种类。自动模式下，工具会根据时钟树约束文件(SPEC)自动选择使用的布线层数和驱动单元数量[6]。本文采用的是手动分步模式，流程如图 1 所示。Placement (布局)完成后对版图进行优化(preCTS)，这一阶段的优化是修复存在的一些 setup 时序违例(时钟信号有效沿触发寄存器之前，检查数据信号是否到达寄存器的据端并在一段时间内保持稳定，setup 时序违例也即建立时间违例)和设计规则违例(DRC)；然后根据设计要求生成时钟树约束文件，并通过合理的设定约束参数，实现时钟树综合。工具先统计出由每个时钟根节点至每个时钟一叶节点之间的最大延时，接着插入驱动器，在同时满足所有其他约束条件要求的情况下尽量地降低至每个时钟叶节点之间的最大延时，要求使它们的之间的最大延时偏差值不应大于约束文件中所定义的最大偏差 Max skew [7]。

时钟树综合过程当中工具还会调用布局引擎将新插入的驱动器放置正确，此时布线工作还没有进行，

工具会根据标准单元的位置去估算延迟[8]。为了确保设定的时钟树参数不受影响，而后根据综合结果在进行时钟树结构以及约束文件的优化(时钟树优化)比如说一些做树的驱动器(clock invter; clock buffer)的大小设置不合适；在做时钟树综合时 clock skew 约束的太紧导致时钟树长度太长等。由于 placement 阶段的优化时钟还是理想的，此时对于时钟门控单元(clock gate)摆放会有不合理的情况，当时钟树综合以后，时钟网络的延时(clock network latency)是真实有效的，此时根据具体路径在对相应的时钟单元进行布局有利于得到最优的时钟树。设计当中会优先对时钟树进行布线。时钟布线完成以后同时对建立(setup)时间和保持(hold)时间分析和优化[9]。

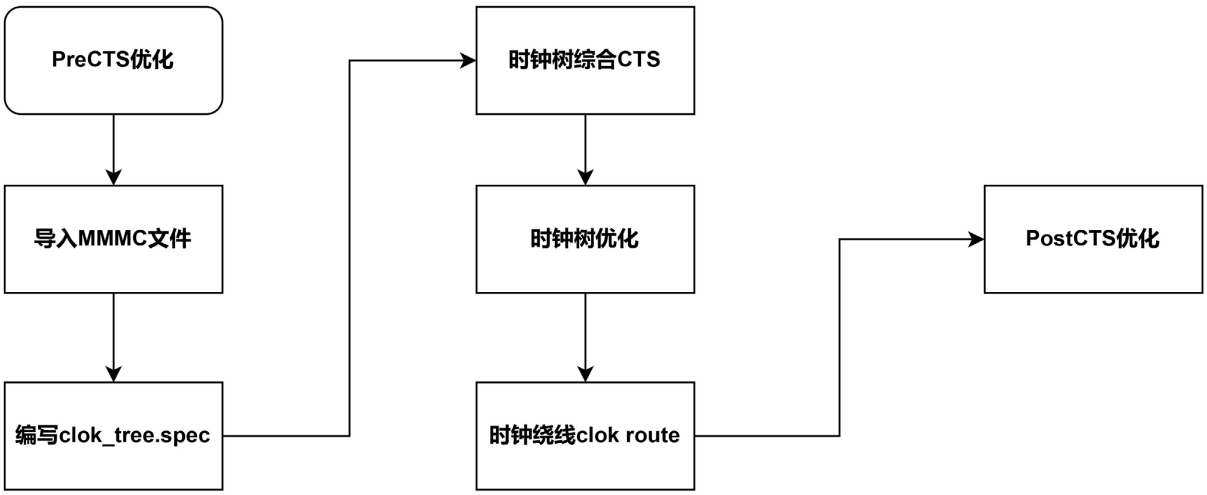

Figure 1. Clock tree synthesis process
图 1. 时钟树综合流程

2.1. 编写 clock_tree.spec 文件

此文件定义了时钟树综合时的约束，工具对哪些路径做 tree，哪些 sink 点进行 balance、skew、transition。此外定义全部为命令形式，在执行 ccopt_design 时如果没有读入 spec 文件，工具会自己产生一个，但是如果想做出好的 clock_tree PPA，必须要根据时钟结构去自己写相应的 spec 文件，特别是对于复杂的时钟结构，比如本设计中 clock_occ 结构，拥有大量的 generated clock 时，更需要根据时钟结构来自己定义 spec，从而去约束工具，得到自己想要的理想的时钟树。图 2 是 spec 中关键设置，由于关于 IP 的时钟是需要单独做树的不和其他时钟 balance 的所以与 IP 相关的时钟用命令[[all_fanout -from base-band_top/reset_clk_gen/U18/A -endpoints_only]] 得到所有关于 IP 的时钟 Pin，设置 Ignore，不与其他 skewgroup 做 balance，单独做树。

2.2. 时钟树综合

本文的时钟树综合与流程图有些许变化，是将时钟树综合分成两步但是都搭载 Innovus 时钟树引擎，是通过分析数据流走向以及时钟定义，将同源时钟中不需要做平的时钟优先做树，执行时钟树综合，分析结果后，在进行常规的时钟树综合。

一般工具默认的做法是所有同源时钟都会做 balance，所有非同源时钟之间都不会做 balance；通过看报告分析具体路径，发现本设计中时钟 GCLKB 是 CLKA 的子时钟，但是 GCLKB 和 CLKA 之间没有数据交互，二者没有必要做平，此时可以先做 GCLKB，做完后把 GCLKB mark 住并把 GCLKB 的 source 设成 exclude pin 再做 CLKA。相关脚本如下：

```
# These pins are ignore skew pins
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U18/A ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U18/A no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U36/A ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U36/A no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U46/A ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U46/A no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U63/A0 ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U63/A0 no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U63/B0 ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U63/B0 no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U63/B1 ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U63/B1 no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U65/A ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U65/A no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/U80/A1 ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/U80/A1 no_sdc_clock
set_ccopt_property sink_type -pin baseband_top/reset_clk_gen/pll_clk_in_buf/I0/A ignore
set_ccopt_property sink_type_reasons -pin baseband_top/reset_clk_gen/pll_clk_in_buf/I0/A no_sdc_clock

# Input pins determined constant across all timing configs.
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_clk_gate_tic_latched_global_timer_reg/latch/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_to_mcu_asyn_fifo_ctrl_clk_gate_ch_num_reg/latch/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_to_mcu_asyn_fifo_ctrl_clk_gate_cr_to_mcu_fifo_wrdata_reg/latch/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_to_mcu_asyn_fifo_ctrl_clk_gate_fifo_to_cr_addr_reg/latch/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_to_mcu_asyn_fifo_ctrl_clk_gate_fifo_wr_state_reg/latch/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_to_mcu_asyn_fifo_ctrl_clk_gate_tic_ch_num_reg/latch/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_to_mcu_asyn_fifo_ctrl_clk_gate_tic_end_counter_reg/latch/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk0_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk10_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk11_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk12_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk13_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk14_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk15_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk1_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk2_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk3_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk4_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk5_gate_I0/SE 0
set_ccopt_property case_analysis -pin baseband_top/bds_track_top/b1i_channel_top_cr_top_wrapper_cr_top_bypass_clk6_gate_I0/SE 0
```

Figure 2. Spec key settings
图 2. Spec 关键设置

create_clock -name CLKA -period 12.5 -waveform{0, 6.25} [get_ports{clka}]

create_generated_clock -name GCLKB -source[get_ports{clka}] -divide_by 2

-add -master_clock[get_clocks{CLKA}] [get_pins ***/Q]

clock_opt -no_clock_route -only_cts -clock_trees [get_clocks GCLKB]

mark_clock_tree -clock_trees [get_clocks GCLKB] -clock_synthesized

set_clock_tree_exceptions -dont_touch_subtrees [get_pins ***/Q]

clock_opt -no_clock_route -only_cts -clock_trees [get_clocks CLKA]

将以上脚本添加到时钟树综合流程中并将自己编写的 spec 文件中除 CLKA GCLKB 之外的时钟都不去处理，运行第一步时钟树综合。完成后在把已经做完的 CLKA 和 GCLKB 相关的设置去掉进行常规的时钟树综合，进而完成整个时钟树综合，得到理想的时钟树结果。

为了得到时序违例可控，drc 可控的结果，下列设置都是关于时钟线设置，首先要设置参数，trunk net 采取两倍线宽和三倍间距的中高层走线，并且使用 shielding。Leaf net 设置为两倍线宽中层走线的方式。让后设置 library cell，考虑降低功耗，clock tree 优先使用 inverter。然后设置 transition、skew 等的值。最后得出一个 500 p 以内的时钟 skew 以及 DRV 数量在 100 条以内的结果。相关脚本如下：

-cts_use_inverters true \

-ccopt_modify_clock_latency false \

-io_opt secondary \

-cts_target_skew 0.2 \

-cts_target_slew 0.1 \

-route_top_non_default_rule_cts_2w2s \

-route_top_shieding_net VSS \

```
-route_top_bottom_perferred
##########cts_seting########
set_clock_tree_options \
-layer_list {M6 M5 M4 M3} \
-target_skew 0 \
-max_transition 0.1 \
-max_capactiance 0.1 \
-max_fanout 32 \
-routing_rule 2w3s_rule \
-use_default_routing_for_sinks 1 \
-buffer_relocation true \
-insert_boundary_cell false \
-logic_level_balance false \
-ocv_clustering true \
-ocv_path_sharing true \
-use_leaf_routing_rule_for_sinks 1 \
-routing_rule_for_sinks_CLK_leaf \
-layer_list_for_sinks {M2 M3 M4 M5 M6}
```

2.3. 时钟树综合

在时钟树 CTS 综合以后我们基本得到了整体时钟树的长度，我们一般只需要分析那些 skew 大于 50 p 的路径，时序违例大于 100 p 以上的需要我们重点分析时钟树结构。因为这种大的 setup 违例，基本上是个别时钟树没有做好，需要单独做树而不能和别的时钟一起 balance。图 3 为测试模式下的 occ 电路结构图。从图中可以看到测试电路通过 clk 输出到 soc Design 中，at speed test 的目的是测试芯片在其工作频率下是否能正常工作，在设计中 test 信号由 pll 晶振共同构成 occ 电路输入到设计中，在本设计中可以简化为 D1、D2、D3 是同一时钟源经过 occ 分频以及 mux 电路简化而来，工具再做时钟树综合时默认同源时钟是要做 balance 的但是这一组 sink 点就没有做 balance，D2 时钟树比 D1 长了 160 p 并且 D2 到相关寄存器的距离并不远，D3 比 D2 时钟树长了 500 p 并且 D3 相关的寄存器虽然多但是物理距离上是用不掉 400 皮秒延时的。所以这就导致 setup 很差，hold 没有窗口去修，在和前端确认相关时钟数据流走向后，确认为工具在这部分没优化好，需要将 D2 和 D3 相关的寄存器添加 group 放到各自附近，添加脚本约束进行时钟树优化。相关脚本命令如下：

set_clock_tree_exceptions -float_pins D2/CK -float_pin_max_delay_rise 0.15 -float_pin_max_delay_fall 0.15

set_clock_tree_exceptions -float_pins D3/CK -float_pin_max_delay_rise 0.65 -float_pin_max_delay_fall 0.65。

2.4. 时钟绕线和 postCTS 优化

当时钟树优化完成后，就要对时钟单元进行绕线了，本设计中时钟频率是 1 GHz，频率越高对布线要求也就越高。如果时钟信号按照标准单元绕线层次进行绕线，随着频率增高，信号线之间串扰也会随之增大，所以一般将时钟路径的布线宽度和间距设置的比默认值大一些(图 3 中两倍线宽三倍线距，中高

层走线)，这样不仅可以解决频率增加引起的串扰问题，还可以减小时钟路径上功耗和延迟[10]。时钟布线规则定义了使用的金属层、布线的间距以及宽度。时钟绕线完成后，真实的延时(latency)就已经确定了，驱动器的内部延时，以及整条路径的 latency 是一个真实可分析的状态，此时为了得到干净的时钟树结构，postCTS 优化重点也放在了时钟路径上的 setup 时序违例，主要目的是处理冗余的驱动器(驱动能力太小或者太大的 buffer、invter)。

表 1 分别是两种时钟树综合方案的时序报告比对，从表中可以看出在总的时间裕度(TNS)和违例路径数方面都有了很大的改善。相比较传统 CTS，改进后的 CTS 的 setup 最大时序违例(WNS)小了 100 多皮秒，其违例路径数也是控制在了 100 以内，而且 TNS 减小了 82%。同时 setup 时序很好，那么留给 hold 的余量也会很足，hold 时间违例通过在 Endpoint 前插缓存器就可以使时序得到满足，TNS 和违例路劲数的减少意味着插入的缓存器减少，这样就可以减小芯片的面积和功耗[11]。

尽管都采用了手动 CTS 的方法，但是要想得到理想的时钟树结构以及时树长度，就必须对时钟树结构有一个深刻的认知，这样才会得到最优的时序，本文采用的分析方法可以有效的减少时钟偏移，促进时序收敛。这大大减轻了后续 setup 违例修复的工作量，减少迭代次数，缩短了整个设计周期。同时，时钟性能变好了，在优化时插入的缓存器数量大大减少，这有利于减小整个芯片的功耗。分别对两种方案的时序进行优化和修复，传统 CTS 的时序结果需要优化 8 次才才能达到流片的要求，而使用改进后的 CTS 仅仅优化了了 4 次，大大减少了芯片设计周期[12]。

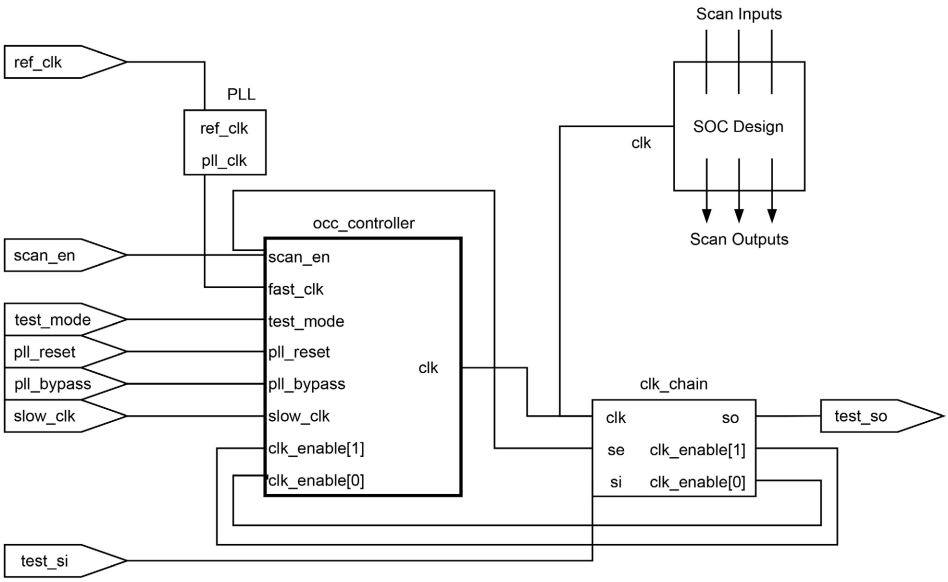

Figure 3. OCC simplifies test circuit diagrams
图 3. OCC 简化测试电路图

Table 1. Setup timing reports
表 1. Setup 时序报告

	Setup	all	reg2reg	reg2cgate	default
传统 CTS	WNS/ns	−0.174	−0.142	−0.032	0.067
	TNS/ns	−0.1081	−1.041	−0.040	0.000
	Violating Paths	240	200	40	0
	All path	2.75E+05	2.46e+05	7769	95816

Continued

	Setup	all	reg2reg	reg2cgate	default
改进后的 CTS	WNS/ns	−0.044	−0.041	−0.002	−0.001
	TNS/ns	−0.195	−0.189	−0.003	−0.003
	Violating Paths	81	74	2	5
	All path	2.75E+05	2.46e+05	7769	95816

Table 2. Setup timing reports
表 2. 两种时钟树综合使用的单元格数对比

	使用单元类型	用的单元个数
传统时钟树综合	CKND*BWP40P140	22094
手动分步时钟树综合	CKND*BWP40P140	15220

3. 结论

随着芯片集成规模越来越大，由十万级门电路演变为现在的百万级、千万级，不光集成度越来越高，为应对市场需求芯片的工作频率也越来越高，这就突出了芯片时序收敛的重要性。从本文分析的结果可以看出，与传统时钟树相比，分步手动模式下的时钟树综合迭代周期更短，时钟偏移比 Innovus 推荐的时钟树综合少了 100 多皮秒，由表 2 可知，在两种时钟树综合方式使用的单元类型一样的情况下，分步时钟树综合使用的单元数量少了 6874 个，节省了芯片面积，同时 skew、power 以及 timing violation 有了明显的改善。并且本文设计的芯片已经完成流片工作，所以此类方法是可以采纳并投入实践的。

参考文献

[1] 陈力颖, 汤勇, 吕英杰. 基于 28 nm 工艺数字芯片的时钟树设计[J]. 天津工业大学学报, 2019, 38(1): 76-82.

[2] Patel, N. (2013) A Novel Clock Distribution Technology-Multisource Clock Tree System (MCTS). *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, **2**, No. 6.

[3] 孙腾达. 手机基带芯片的低功耗设计[D]: [硕士学位论文]. 西安: 西安电子科技大学, 2013.

[4] 陈天宇. 基于 40 nm 工艺的低功耗 GPU 模块后端物理设计[D]: [硕士学位论文]. 武汉: 华中科技大学, 2014.

[5] 严伟, 范光宇, 朱兆伟, 等. 一种改进型 FBT 时钟树结构[J]. 微电子学, 2017(1): 92-95.

[6] 杨朱黎. 时钟偏移补偿技术的研究及应用[D]: [硕士学位论文]. 长沙: 国防科学技术大学, 2012.

[7] 李洋洋. 基于 28 nm 工艺的数字芯片静态时序分析及优化[D]: [硕士学位论文]. 西安: 西安电子科技大学, 2016.

[8] 祝雪菲, 张万荣, 万培元, 等. 一种有效实现 IC 时序收敛的方法[J]. 微电子学, 2015, 45(4): 474-478+483.

[9] 张弛. 基于时钟网络的低功耗物理设计方法研究与实现[D]: [硕士学位论文]. 北京: 国防科学技术大学, 2014.

[10] 柏璐. Flash 内存控制芯片布局优化设计[D]: [硕士学位论文]. 北京: 北京工业大学, 2010.

[11] 熊俊峰. YHFT-DX 芯片低功耗物理设计的研究与实现[D]: [硕士学位论文]. 北京: 国防科学技术大学, 2013.

[12] 贺京. 基于 65 nm 的低功耗设计与等价性验证[D]: [硕士学位论文]. 西安: 西安电子科技大学, 2013.