

---

电子科技大学示范性微电子学院

# 逻辑综合报告

(2024.11 - 2024.12)

课程名称 IC 综合实验 2

实验名称 IC 综合实验 2

指导老师 王忆文

学生姓名 李卓霖，况明远，邓锦琪，徐子涵

学生学号 2022340101012、2022340102004、  
2022020915006、2022340104032

组 号 11

---

# 电子科技大学

# 实验报告

实验地点：清水河校区国际创新中心 B523

实验时间：2024.11-2024.12

## 报告目录

一、实验室名称：清水河校区国际菁蓉创新中心 B523

二、实验项目名称：IC 综合实验 2

三、实验学时：100 学时

四、实验原理：请附页

五、实验目的：请附页

六、实验内容：请附页

七、实验器材（设备、元器件）：请附页

八、实验步骤：请附页

九、实验数据及结果分析：请附页

十、实验结论：请附页

十一、总结及心得体会：请附页

十二、对本实验过程及方法、手段的改进建议：请附页

实验报告成绩：\_\_\_\_\_

---

附页：

## 四、实验原理：

### 4.1. 逻辑综合的概念

逻辑综合是把 RTL 描述转换为实际电路的过程；Design Compiler 是 Synopsys 公司用于综合的核心工具，它可以方便地将 HDL 语言电路转换为基于工艺库的门级网表。

### 4.2. 逻辑综合的步骤

概括地说，*synthesis = translation + logic optimization + gate mapping*.

**1. Translation**：翻译。主要把描述 RTL 级的 HDL 语言，在约束下转换成 DC 内部的统一用门级描述的电路（Generic Boolean Gates），这是 DC 自己的库表现，通常指的是在逻辑综合过程中生成的基础逻辑门（如与门、或门、非门等）的表示。这些门不是特定于某一种工艺或库，而是通用的、抽象的逻辑门，用于在未指定具体实施细节时进行中间的逻辑优化和处理。以 GTECH 或者没有映射的 ddc 形式展现。

**2. Logic optimization**：逻辑优化，就是把统一用门级描述的电路进行优化，简单地说，那就是“把路径调整一下，门给改一下”等。具体而言，逻辑优化的核心目标是通过调整电路的路径、结构和逻辑门的调整，提高电路的性能，减少延迟、功耗，同时可能还要尽量减少面积。这里列举一些逻辑优化的主要方面：

- **路径优化**：通过重新排列逻辑门和信号路径，减少信号在电路中传播的时间。这可能包括在逻辑图中重新连接门，或使用更快的门来替代较慢的门。
- **门级修改**：替换逻辑门类型或合并门以实现更高效的实现。例如，如果某个组合逻辑可以用较少的门实现，利用逻辑化简的手段（德摩根定律等），优化工具会识别并进行这种替换。

- 
- **冗余消除**：识别并消除可以被删除而不改变电路功能的冗余逻辑。比如，一些输入的组合可能导致某些逻辑门总是处于某个固定状态，这些门可以被省略。
  - **共享与重用**：当多个条件可以实现重复的逻辑功能时，优化工具可以尝试共享这些逻辑运算以节省资源。
  - **实现目标**：根据设计目标（例如最低功耗、最高速度等）调整优化策略，有时这可能涉及在延迟与功耗、面积等参数之间的权衡。。

**3. Gate mapping**：门级映射，把优化了的统一门级描述根据 DC 用厂商的工艺库把电路给映射出来，得到一个.ddc 文件。这个.ddc 文件可以包含许多丰富的信息，比如映射的门电路信息与网表、.v 格式的网表、延时信息（sdf）、工作约束(sdc)等信息。（注意到 .ddc 不能用文本编辑器打开）。.ddc 这个包含的网表文件是实际意义的网表文件，而.v 这个形式的网表其实是用来做后仿真的文件。

### 4.3. DC 的启动方式

概括地说，DC 的启动方式有 3 种，下面一一介绍。

GUI：图形化界面，大规模设计不可能如此进行。

启动方式：`$design_vision`

Dc\_shell：DC 以命令行的形式启动。

启动方式：`$dc_Shell`

Batch mode: 批处理模式，也就是，前面那两种方式只是把 DC 启动起来，还没有真正地工作（即编译工程），前面两个方式需要通过 `source` 命令把脚本写进去以后，DC 读取才真正工作。而这种批处理模式是，在启动的同时，告诉 DC 执行哪些脚本,例如：

```
$dc_shell -topo -f run.tcl | tee -i run.log
```

这句命令的意思是：使用拓扑模式启动 DC，启动的同时执行 `run.tcl` 脚本文件，并且把启动过程中显示在终端的信息记录到 `run.log` 中。`| tee -i` 就是写进信息的管道命令，讲 `dc_shell -topo -f run.tcl` 执行后显示的信息（输出结果），

流入到 run.log 文件中。这样子是为了在 DC 启动失败的时候，通过查看启动信息，进而排除错误。

#### 4.4. DC 综合的流程

详细地说，DC 综合需要的文件与完成综合后生成的文件如图 1 所示。

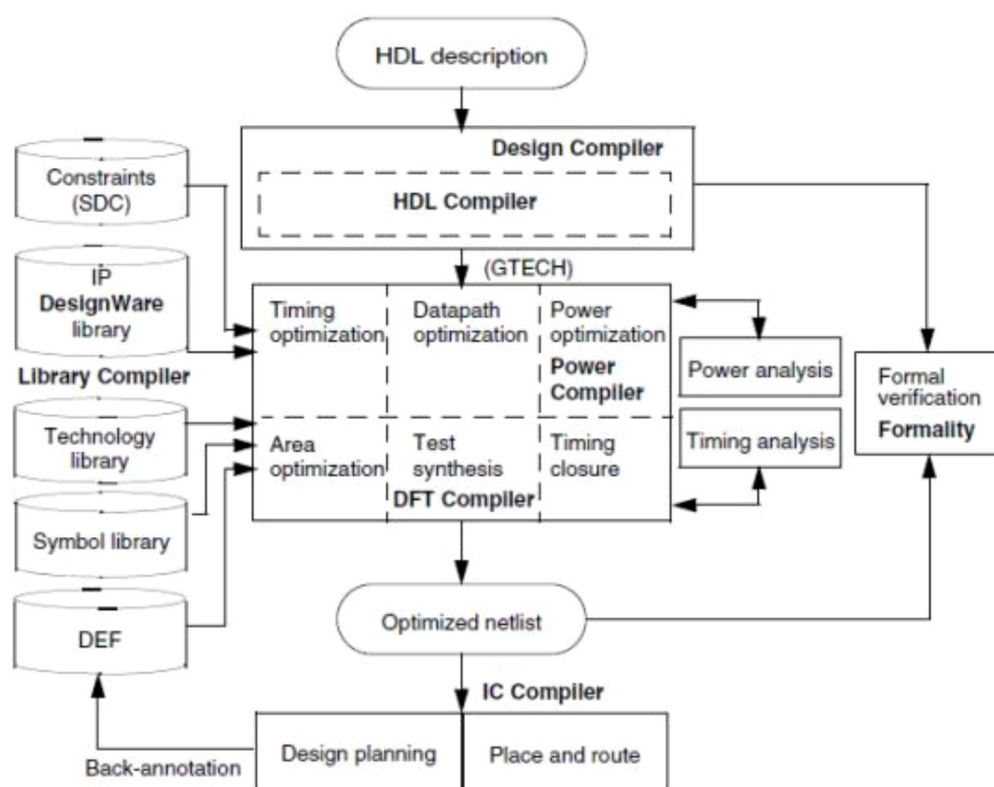


图 1 DC 综合的流程

具体到本实验，我们相比上图做了一定简化，仅从实验实际需要 DC 的角度出发，得到下图（图 2）。

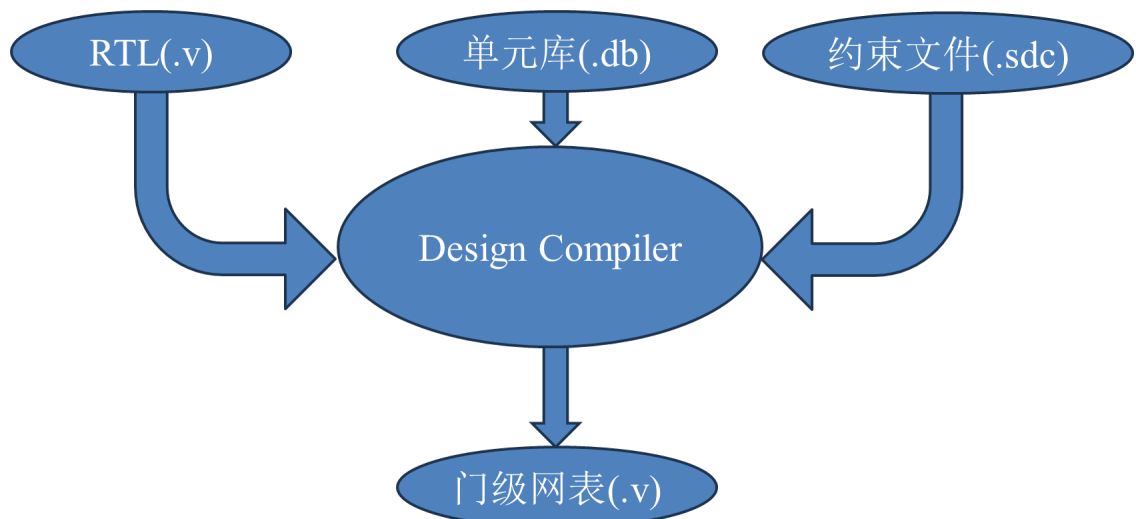


图 2 简化后的 DC 综合流程

#### 4.5. 形式验证

在数字设计流程中，形式验证（Formality）是一个重要的步骤，用于确保经过综合的设计与其原始 RTL（Register Transfer Level）描述之间的一致性。形式验证的核心目标是确认综合后的硬件实现（通常是门级网表）和 RTL 设计在功能上是等价的，即它们在相同输入下的输出是相同的。下面介绍一下一致性验证（Formality）的原理：

1. 等价性检查：

形式验证工具（如 Synopsys 的 Formality）比较综合后的门级网表（netlist）与原始 RTL 描述。在这个过程中，工具会自动识别信号和逻辑操作的对应关系，以确定两者是否在功能上等价。

2. 状态空间分析：

工具通常会通过建立一个状态空间模型，分析输入状态与输出结果之间的关系。它会检查是否存在输入组合使得两个设计在同一时间点上的输出不一致。

3. 符号化方法：

一些形式验证工具采用符号化的方法，通过抽象化输入、输出和状态，减少需要检查的空间，从而提高验证效率。这些符号方法帮助处理大型和复杂的设计并寻找可能的不一致。

4. 图形表示：

---

工具内部通常会将 RTL 和门级网表转换为图形表示（例如有向图），这使得它们能够在结构上进行比较，查找潜在的不一致点。

现在再介绍一下形式验证的主要流程：

1. 匹配（Match）：

对 RTL 描述与门级实现进行静态检查，确保信号名称和层次结构相匹配。这是验证过程的第一步，以确保两者的结构一致。

2. 验证（Verify）：

在匹配无误后，进入验证阶段，工具会执行更深入的检查，确保功能上相等。这涉及到比较信号的逻辑输出，并确保在所有可能输入下，两者产生相同的输出。

3. 调试（Debug）：

如果在验证阶段发现问题，工具会提供详细的调试信息，以帮助工程师定位和理解不一致的原因。这可能包括使用波形查看器可视化两者输出的差异，或提供失败的输入组合示例。

## 五、实验目的：

### DC 逻辑综合实验目的：

1. **深入理解综合过程：**通过软件进行逻辑综合，让我们从 EDA 课程中重视 RTL 的心态、孤立的看待 RTL 和电路的心态转变为理解从 RTL (Register Transfer Level) 设计到门级 (Gate Level) 表示的转化过程，学习如何将高层次的设计描述转换为满足时序、功耗和面积等设计约束的网表。

2. **优化设计参数：**掌握如何设置和调整综合工具中的各种参数，以实现设计目标，包括优化时序（如时钟频率）（本实验 DC 综合频率为 300 M）、功耗以及芯片面积（本实验要求是 DC 综合后面积小于 24 万。这一过程有助于提升设计的性能和效率。

3. **熟悉综合工具的使用：**通过实际操作 DC 工具，增强对专业软件的操控能力，理解其工作原理和常见的设置与选项，提高工作效率。

4. **实现功能验证与时序分析：**在综合过程中学习如何进行功能验证和初步的时序分析（report\_timing），确保设计在满足规范的同时，不出现逻辑错误，以便在后续的布局布线阶段减少修改的次数。

---

### FORMALITY 一致验证实验目的：

1. **理解一致性验证的重要性：**通过 FORMALITY 一致验证，我们能够理解在设计过程中验证 RTL 和门级实现之间的一致性的重要性，以确保设计的功能正确性，避免在综合和物理实现过程中引入错误。尤其是在我们这次高频（300 M）的设计中，DC 过于激进的综合策略可能改变代码本身映射到电路的含义，务必需要一致性验证。
2. **掌握形式验证技术：**学习如何使用 FORMALITY 等工具进行形式验证，理解其背后的数学原理，并掌握如何设置验证环境、选择合适的验证策略，以便在产品开发中保证设计的可靠性。
3. **提升错误检测与修复能力：**通过一致性验证实验，学生将培养出检测和修复设计中逻辑错误的能力，能够快速定位问题并采取适当的修复措施。
4. **与时序分析相结合：**结合 DC 逻辑综合中的时序分析，理解动态时序检查与形式验证的关系，从而提升全局视野，确保设计在时序和逻辑层面上一致。
5. **培养系统思维和工程实践能力：**通过一致验证的训练，增强系统思维能力，能够在完整的 IC 设计流程中，识别潜在风险，并采取有效的预防措施，从而提升整体设计质量。通过上述实验目的的实现，我们可以在数字 IC 设计的各个环节提升自身的能力，不仅包括技术操作，还有设计思维和团队合作能力，为将来的工作奠定坚实的基础。

## 六、实验内容：

本次 IC2 实验课程（数字方向）的设计内容为一个已知结构的 AI 加速器的 ASIC 全流程设计并通过评估和流片。已知结构的 AI 加速器具有 8bit 输入和 8bit 输出，可以工作在至少 50Mhz 的时钟下，能够完成给定卷积神经网络的运算。总体的面积要求为在不带 I/O 的情况下通过 DC 综合得到的网表面积为，版图的面积最大限制为 900um \* 900um，可以使用的最大的引脚数量为 28 个（含 4 到 6 个电源引脚）。

给定的卷积核的结构包含卷积层、池化层、全连接层各一层，共计 3 层。各层的详细介绍如下。

卷积层：在  $11 \times 11$  的矩阵数据外添加 Padding 后，对对应的矩阵进行卷积操作，其中，卷积核大小为  $3 \times 3$ ，步长为 2，共有 3 个卷积核；

池化层：对输出的数据进行最大池化，其中池化核大小为  $2 \times 2$ ，步长为 2；



---

全连接层：全连接层对应的权重矩阵大小为  $3 \times 3 \times 3$ ，与池化输出一一对应，需要完成一次乘加操作。

芯片的输入数据为  $1 \times 1$  的矩阵，是位宽为 8bit 的有符号数补码，按照卷积核、全连接权重、100 组  $11 \times 11$  的矩阵的数据依次输入。

芯片的输出数据为  $1 \times 1$  的矩阵，位宽为 8bit，输出的为全连接的结果通过量化所得到的内容。

由于在数据处理过程中得到的乘积数据为 16bit，加和后需要用 20bit（全连接 21bit）的数据记录才能保证数据不发生溢出，而后续进行池化、全连接和输出所需要的数据均为 8bit，因此需要一种量化规则来进行 20bit 或 21bit 的到 8bit 的量化处理。给定的卷积核的量化规则为：在保证符号正确的情况下拓展到 20 位来表示，然后将低 8 位抹除，剩余的 12 位若超过 int8 的表示范围，则取 int8 范围的最大值，反之，取这 12 位的低 8 位。由于数据均由补码表示，因此所得的结果只需要将最高位符号位向前扩展即可得到扩展数据。

## 七、实验器材（设备、元器件）：

实验器材（设备与软件）：

1. 计算机终端

2. 服务器终端

3. IC 设计工业软件：

DC (Design Compiler): 逻辑综合软件，用于将 RTL 代码转化为门级网表。

Formality：进行一致性验证。

## 八、实验步骤：

### 8.1. 步骤概述

图 3 给出了利用 DC 进行逻辑综合时操作的基本步骤，我们大体上根据这个图运行 DC。

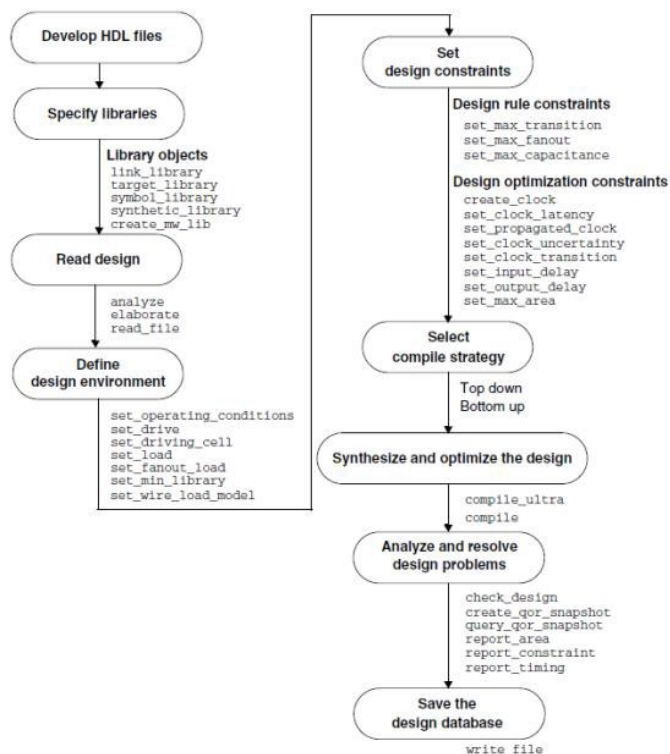


图 3 DC 进行逻辑综合时的操作步骤

## 8.2. 启动环境设置

我们按照前面的基本流程使用 DC 进行设置。在启动 DC 之前，首先要配置 DC 的启动环境，也就是那些库的设定。配置 DC 的启动环境主要是 .synopsys\_dc.setup。 .synopsys\_dc.setup 这个文件就是 DC 的配置文件，它配置了 DC 启动过程中要执行哪些命令、干哪些事。其中， search\_path 、 target\_library...等，这些是 DC 内部的变量名称，用来告诉 DC 做那些事的方法。

一般 .synopsys\_dc.setup 文件有三个：

- 1 一个在 synopsys 的安装目录下，这个文件最好不要动；
- 2 一个在用户目录下，这个文件没事也不要动；
- 3 还有一个当前工作目录下，也就是启动 DC 的目录下（没有就需要自己创建），这个理应是要我们自己写的，不过我们贴心的助教帮助我们完成了一个大概。

所谓配置 DC 的启动环境，就是在启动的目录创建 .synopsys\_dc.setup 并且修改它。

---

这里简单介绍部分助教给的启动文件介绍：

1. **set**：自定义的一些变量，方便定义系统变量的时候，不用那么麻烦；
2. 指定搜索路径 (**Search\_path**)：当读入了一些工艺库时，忘了设置它的路径，那么 **DC** 就会根据这个变量设置的路径去寻找库。**DC** 可以根据这些路径进行寻找相关的库文件；或者，当需要多个.v 文件时，通过这个变量告诉 **DC** 那些.v 文件可能的路径，让 **DC** 根据.v 文件的名称去找.v 文件。
3. 指定综合库 (**synthetic\_library**)：这个库一般是 **synopsys** 的库：  
**DesignWare library** 和标准单元库，这里指定的就是库的名字了。  
**DesignWare library** 这个库是 **synopsys** 的 IP 库：当使用到 **synopsys** 公司的 IP 核的时候比如使用了该公司的乘法器 IP，那么就要定义这个综合库；此外，当需要用到这个库的一些比较高端的 IP 核的时候，是需要相关的证书的。这一点我深有体会：在我自己的电脑虚拟机上，  
**DC 不支持调用乘法器 IP 核，最终逻辑综合只能在实验室进行。**
4. 指定目标工艺库 (**target\_library**)：DC 将 RTL 级的 HDL 描述映射到实际的门级电路时所需要的标准单元库。这个库是半导体制造商提供的工艺库。附注：启动 **DC** 之后，可以通过 `printvar target_library` 查看工艺库名称。
5. 指定链接库 (**link\_library**)：**link\_library** 是 **target\_library** 一样的单元库或者是已经综合到 Netlist 的底层模块设计（比如 IP 核）。作用是：用于分辨电路中逻辑门和子模块的功能，然后用实际的库单元或者子模块代替它们；在由上而下的综合工程中，上一层的设计调用底层已经综合的模块时，将从 **link\_library** 中寻找并且链接起来，因此当读入的文件是门级网表（比如用到了 IP 核的网表）时，需要把 **link library** 设成指向生成该门级网表的目标库，否则 **DC** 因不知道网表中门单元电路的功能而报错。附注：**如果需要将已有的设计从工艺 A 转到工艺 B 时，可以将当前的单元综合库 A 设为 link\_library，而将单元综合库 B 设为 target\_library，重新映射一下就可以了。**
6. 路径面前加\*号表示开辟一块单独的内存空间给 **DC** 自己使用，然后先

---

搜寻内存中已有的库，然后再搜寻变量 `link_library` 指定的其他库。

DC 搜寻的库为 `search_path` 指定的目录（比如说之前读入设计时读入了库 a，库 a 存到内存里；这时 DC 在进行综合的时候，发现缺少某个东西，于是就先从库 a 里面找，找不到时就会从列表里面的变量路径中找）。

### 8.3. Design Ware 库

DesignWare 是 Synopsys 提供的知识产权(Intellectual Property, 简称 IP) 库。IP 库分成可综合 IP 库(synthesizable IP, SIP),验证 IP 库(Verification IP, VIP)和生产厂家库(foundry libraries)。IP 库中包含了各种不同类型的器件。这些器件可以用来设计和验证 ASIC, SoC 和 FPGA。库中有如下的器件:

- 积木块(Building Block)IP(数据通路、数据完整性、DSP 和测试电路等等)
- AMBA 总线构造(Bus Fabric)、外围设备(Peripherals)和相应的验证 IP
- 内存包(Memory portfolio)(内存控制器、内存 BIST 和内存模型等等)
- 通用总线和标准 I/O 接口(PCI Express,PCI-X,PCI 和 USB)的验证模型
- 由工业界最主要的明星 IP 供应商提供的微处理器和 DSP 核心
- 生产厂家库(Foundry Libraries)
- 板级验证 IP<Board verification IP)
- 微控制器(Microcontrollers, 如 8051 和 6811)

这其中，所有的 IP 都是事先验证过的、可重复使用的、参数化的、可综合的，并且不受工艺的约束。

这里列举网络上搜索到的常见的 DesignWare foundation 库单元，如图 4.

单元名称	说 明
DW01_absval	求绝对值
DW01_add	加法器
DW01_addsub	加/减法器
DW_addsub_dx	具有舍入、饱和功能的加/减法器
DW01_ash	算术移位器
DW_bin2gray	将二进制码转换为格雷码的单元
DW01_bsh	桶形移位器
DW01_cmp2	比较器
DW01_cmp6	比较器
DW_cmp_dx	双工(duplex)比较器
DW_centr_gray	格雷码计数器
DW01_csa	进位保存加法器
DW01_dec	递减器
DW_div	除法器

DW_div_pipe	流水线除法器
DW_gray2bin	将格雷码转换为二进制码的单元
DW01_inc	完成递增功能
DW01_incdec	完成递增/递减功能
DW_inc_gray	格雷码递增器

DW02_mac	乘加单元
DW_minmax	从几个输入中找出最小、最大值的单元
DW02_mult	乘法器
DW02_multp	部分积乘法器

DW02_mult_2_stage	2 阶流水线乘法器
DW02_mult_3_stage	3 阶流水线乘法器
DW02_mult_4_stage	4 阶流水线乘法器
DW02_mult_5_stage	5 阶流水线乘法器
DW02_mult_6_stage	6 阶流水线乘法器
DW_mult_dx	双工(duplex)乘法器
DW_mult_pipe Stallable	流水线乘法器
DW02_prod_sum	对一系列乘积求和

DW_prod_sum_pipe	对一系列乘积求和，用流水线方式实现
DW01_satrnd	进行舍入操作
DW_shifter	移位器(算术移位和桶形移位)
DW_square	求平方运算
DW_squarep	求平方运算，结果用部分积表示
DW_sqrt	求平方根
DW_sqrt_pipe	用流水线方式求平方根
DW01_sub	减法器
DW02_sum	矢量加法器
DW02_tree	Wallace Tree 压缩单元(Wallace Tree 乘法器的构件)

DW02_cos	进行三角函数余弦运算
DW02_sin	进行三角函数正弦运算
DW02_sincos	进行三角函数正/余弦运算
DW_div_seq	时序除法器
DW_mult_seq	乘法器，采用时序方式
DW_sqrt_seq	平方根单元，采用时序方式

DW_add_fp Floating Point Adder	浮点数加法单元
DW_cmp_fp	浮点数比较器
DW_div_fp	浮点数除法器
DWflt2i_fp	完成浮点数到整数的转换
DW_i2flt_fp	完成整数到浮点数的转换
DW_mult_fp	浮点数乘法器

图 4 常见的 DesignWare foundation 库单元

这些库单元如此好用，我们怎么用呢，很简单！

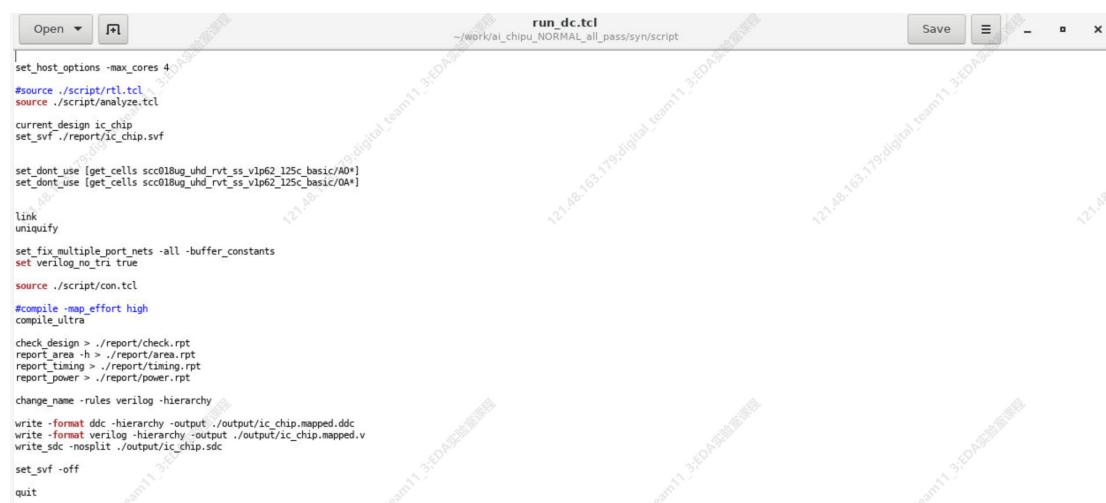
可以用运算符推论法(Operator Inferencing)或功能推论法(Functional Inferencing)。运算符推论法是直接在设计中使用“+、-、\*、>、一和<”等的运算符。功能推论法是在设计中例化(instantiate) DesignWare 中某种算术单元，例如直接指定用库中的 DWF\_mult\_tc, DWF\_div\_uns 和 DWF\_sqrt\_tc 单元。

由于 DesignWare 库中的所有器件都是事先验证过的，使用该 IP 库我们可以设计得更快，设计的质量更高，增加设计的生产力和设计的可重复使用性，减少设计的风险和技术风险。对于每个运算符，一般地说 DesignWare 库中会有多个结构(算法)来完成该运算。这样就允许 DC 在优化过程中评估速度/面积的折衷，选择最好的实现结果。对于一个给定的功能，如果有多个

DesignWare 的电路可以实现它，Design Compiler 将会选择能最好满足设计约束的电路。此外使用 DesignWare 中的 DW Foundation 库是需要许可证的 (license)，DW Foundation 库提供了更好的设计质量(Quality of Result)。

## 8.4. Run\_dc.tcl

完整的脚本如下，这部分笔者没有想要说明的点。



```
set_host_options -max_cores 4
#source ./script/rtl.tcl
source ./script/analyze.tcl

current_design ic_chip
set_svf ./report/ic_chip.svf

set_dont_use [get_cells scc018ug_uhd_rvt_ss_vlp62_125c_basic/A0*]
set_dont_use [get_cells scc018ug_uhd_rvt_ss_vlp62_125c_basic/OA*]

link
uniquify

set_fix_multiple_port_nets -all -buffer_constants
set_verilog_no_tri true

source ./script/con.tcl

#compile -map_effort high
compile_ultra

check_design > ./report/check.rpt
report_area -h > ./report/area.rpt
report_timing > ./report/timing.rpt
report_power > ./report/power.rpt

change_name -rules verilog -hierarchy

write -format ddc -hierarchy -output ./output/ic_chip.mapped.ddc
write -format verilog -hierarchy -output ./output/ic_chip.mapped.v
write_sdc -nosplit ./output/ic_chip.sdc

set_svf -off

quit
```

图 5 run\_dc.tcl

## 8.5. con.tcl

这里我们做了一些修改。

首先是把 clk\_uncertainty 从一个具体的值改为了时钟周期的比例，这么做是为了考虑我们设计 300 M 频率的高速设计，也更符合实际情况（形象地说，总不能时钟周期 3 ns，skew 也 3 ns 吧！）。

第二是我们大幅提高时钟频率了，这里给出的版本是 250 M 的，最终我们的设计在 DC 综合时达到了 300 M。



```
set Tclk 4
set Ttarget 5
set unc_perc 0.02

set_timing_remove_clock_reconvergence_pessimism true
set_critical_range [expr $Tclk * 0.1] $current_design
set_ideal_ports {clk_io rstn_io}
set_ideal_network clk_io
set_ideal_network rstn_io

create_clock -period $Tclk [get_ports clk_io]
set_input_delay [expr 0.25*$Ttarget] -clock clk_io [remove_from_collection [all_inputs] $ideal_ports]
#set_input_delay [expr 0.25*$T] -clock clk_io [remove_from_collection [all_inputs] $ideal_ports]

set_output_delay [expr 0.25*$Ttarget] [all_outputs]
#set_output_delay [expr 0.25*$T] [all_outputs]

set_clock_uncertainty -setup [expr $Tclk * $unc_perc] [get_clocks clk_io]
set_clock_uncertainty -hold [expr $Tclk * [expr $unc_perc * 0.5]] [get_clocks clk_io]
set_max_fanout 100 $current_design
set_input_transition 2.0 [remove_from_collection [all_inputs] $ideal_ports]
set_load 5 [all_outputs]
```

图 6 con.tcl

## 九、实验数据及结果分析：

### 9.1. 时序表现

如图所示，DC 综合时频率达到 300 M (3.33 ns).

Path Group: clk_io		
Path Type: max		
Point	Incr	Path
-----		
clock clk_io (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
u_CNN_TOP/CNN_CORE_CTL_inst/u_CNN_CORE0/mult_A_reg_reg[7][5]/CK (DQUHDV2)	0.00 #	0.00 r
u_CNN_TOP/CNN_CORE_CTL_inst/u_CNN_CORE0/mult_A_reg_reg[7][5]/Q (DQUHDV2)	0.46	0.46 f
U3823/ZN (INUHDV2)	0.14	0.60 r
U7423/ZN (XNOR2UHDV0P4)	0.33	0.93 r
U4874/ZN (NAND2UHDV0P4)	0.14	1.06 f
U4872/ZN (NAND2UHDV0P4)	0.20	1.26 r
U3587/Z (AND2UHDV0P7)	0.23	1.50 r
U7432/Z (CLKXOR2UHDV0P7)	0.15	1.65 r
U7433/Z (CLKXOR2UHDV0P7)	0.32	1.98 r
U6142/ZN (NOR2UHDV0P7)	0.09	2.06 f
U3187/ZN (INUHDV0P4)	0.11	2.17 r
U3381/ZN (NAND2UHDV1)	0.09	2.26 f
U6141/ZN (INUHDV2)	0.08	2.34 r
U5836/ZN (CLKNOR2UHDV4)	0.07	2.41 f
U3364/ZN (NAND2UHDV1)	0.10	2.51 r
U7489/ZN (NAND3UHDV2)	0.10	2.61 f
U3354/ZN (NAND2UHDV2)	0.12	2.73 r
U3793/ZN (CLKNAND2UHDV0P7)	0.11	2.84 f
U6170/ZN (NAND2UHDV1)	0.13	2.97 r
U6169/Z (XOR2UHDV2)	0.11	3.08 r
u_CNN_TOP/CNN_CORE_CTL_inst/u_CNN_CORE0/product_reg_reg[7][11]/D (DQUHDV3)	0.00	3.08 r
data arrival time		3.08
clock clk_io (rise edge)	3.33	3.33
clock network delay (ideal)	0.00	3.33
clock reconvergence pessimism	0.00	3.33
clock uncertainty	-0.07	3.26
u_CNN_TOP/CNN_CORE_CTL_inst/u_CNN_CORE0/product_reg_reg[7][11]/CK (DQUHDV3)	0.00	3.26 r
library setup time	-0.19	3.08
data required time		3.08
-----		
data required time		3.08
data arrival time		-3.08
-----		
slack (MET)		0.00

1

图 7 时序表现

### 9.2. 面积表现

可以看到，即便在超高频下，我们设计的面积仅 17 万。



```

*****
Report : area
Design : ic_chip
Version: 0-2018.06-SP1
Date   : Fri Dec 27 09:24:22 2024
*****

Information: Updating design information... (UID-85)
Warning: Design 'ic_chip' contains 1 high-fanout nets. A fanout number of 1000 will be used for delay calculations involving these nets. (TIM-134)
Library(s) Used:

    scc018ug_uhd_rvt_ss_vlp62_125c_basic (File: /public/digital_LIB/lib/std/SCC018UG_UHD_RVT_V0.4a/liberty/1.8v/scc018ug_uhd_rvt_ss_vlp62_125c_basic.db)
    SP018WP_Vlpl_max (File: /public/digital_LIB/lib/io/SP018WP_Vlpl/syn/3p3v/SP018WP_Vlpl_max.db)

Number of ports:      107
Number of nets:       12522
Number of cells:      12334
Number of combinational cells: 10870
Number of sequential cells: 1442
Number of macros/black boxes: 21
Number of buf/inv:    1686
Number of references: 120

Combinational area:   105836.238424
Buf/Inv area:         10297.997105
Noncombinational area: 63579.265381
Macro/Black Box area: 223440.000000
Net Interconnect area: undefined (No wire load specified)

Total cell area:      392855.503805
Total area:           undefined

Hierarchical area distribution
-----

```

Hierarchical cell	Global cell area		Local cell area			Design
	Absolute Total	Percent Total	Combi-national	Noncombi-national	Black-boxes	
ic_chip	392855.5038	100.0	78665.9339	46445.4152	223440.0000	ic_chip
u_CNN_TOP/DATA_inst	44304.1547	11.3	27170.3046	17133.8502	0.0000	DATA
Total			105836.2384	63579.2654	223440.0000	

```

1

```

图 8 面积表现

### 9.3. 功耗表现

最后查看功耗，如图 9 所示，总动态功耗为 22.56 mW.

```

*****
Report : power
        -analysis_effort low
Design : ic_chip
Version: 0-2018.06-SP1
Date   : Fri Dec 27 09:24:23 2024
*****

Library(s) Used:

    scc018ug_uhd_rvt_ss_vlp62_125c_basic (File: /public/digital_LIB/lib/std/SCC018UG_UHD_RVT_V0.4a/liberty/1.8v/scc018ug_uhd_rvt_ss_vlp62_125c_basic.db)
    SP018WP_Vlpl_max (File: /public/digital_LIB/lib/io/SP018WP_Vlpl/syn/3p3v/SP018WP_Vlpl_max.db)

Operating Conditions: SP018WP_Vlpl_max   Library: SP018WP_Vlpl_max
Wire Load Model Mode: top

Global Operating Voltage = 1.62
Power-specific unit information :
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = ns
Dynamic Power Units = 1mW (derived from V,C,T units)
Leakage Power Units = 1uW

Cell Internal Power = 22.4076 mW (99%)
Net Switching Power = 150.6621 uW (1%)
-----
Total Dynamic Power = 22.5582 mW (100%)
Cell Leakage Power = 4.5272 uW

```

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	( % )	Attrs
io_pad	2.2079	4.4769e-03	0.5717	2.2130	( 9.81%)	
memory	0.0000	0.0000	0.0000	0.0000	( 0.00%)	
black_box	0.0000	0.0000	0.0000	0.0000	( 0.00%)	
clock_network	0.0000	0.0000	0.0000	0.0000	( 0.00%)	
register	20.0936	1.5082e-02	1.2352	20.1099	( 89.13%)	
sequential	0.0000	0.0000	0.0000	0.0000	( 0.00%)	
combinational	0.1060	0.1311	2.7203	0.2398	( 1.06%)	
Total	22.4075 mW	0.1507 mW	4.5272 uW	22.5627 mW		

```

1

```

图 9 功耗表现

## 9.4. 一致性验证

我们依次通过了 match, verify, 如图 10 、图 11 所示。

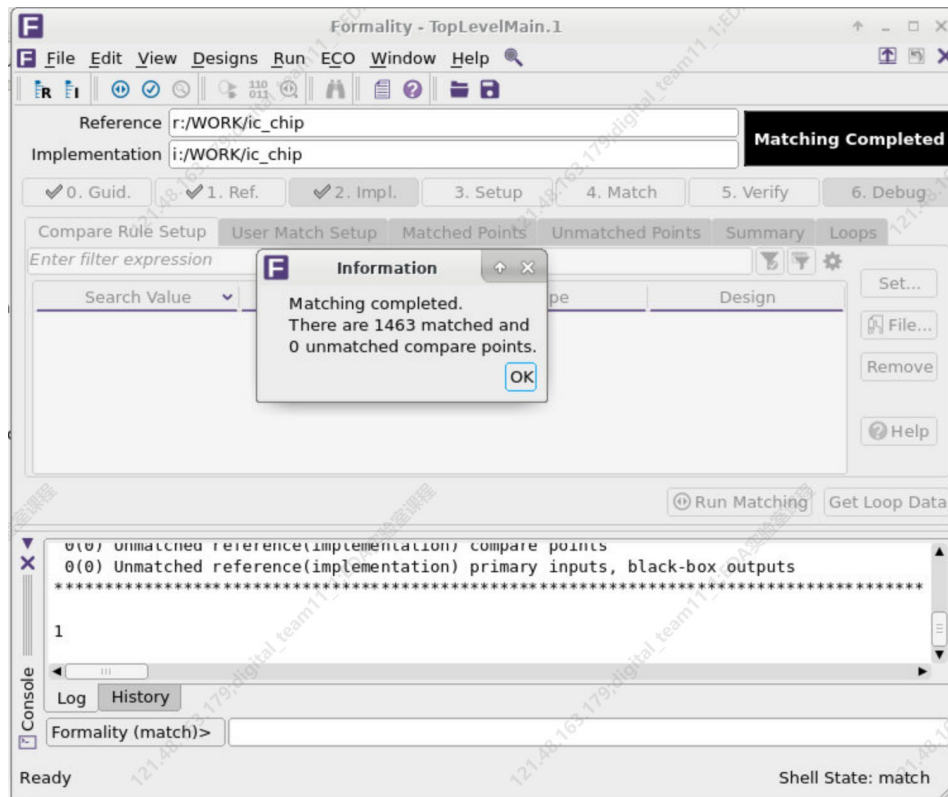


图 10 match

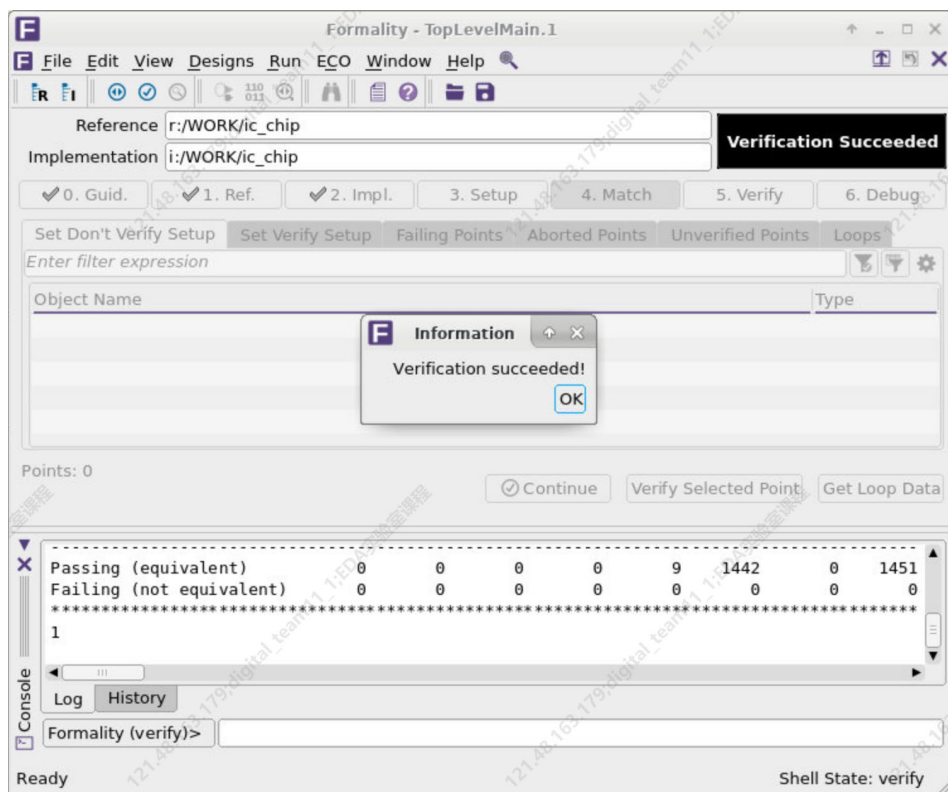


图 11 verify

这里呈现一个 debug 的过程，可以从图中看出，在图示单元，netlist 与 rtl 一致，如图 12 所示。

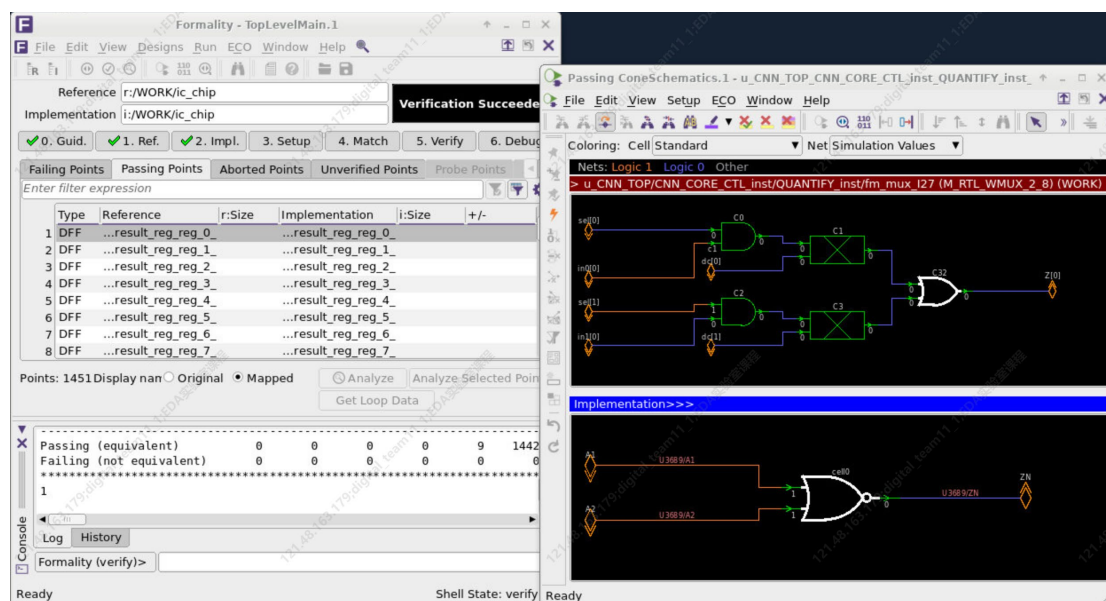


图 12 debug

## 十、实验结论：

通过对 DC 逻辑综合和 FORMALITY 一致性验证的实验，得出以下结论：

1. 逻辑综合的重要性：DC 逻辑综合是数字 IC 设计流程中的关键环节，能够有效地将 RTL 设计转换为门级网表。通过此过程，设计师能够确保设计在满足功能需求的同时，优化设计的时序、功耗和面积，为后续的物理实现奠定基础。

2. 综合工具的掌握：掌握 DC 工具的使用，使得我们能够更高效地进行综合，熟悉各种参数设置及其对设计结果的影响，这种技能在实际项目中具有重要的工程价值。

3.形式验证的必要性：FORMALITY 一致验证不仅验证了 RTL 和门级实现之间的逻辑一致性，也为设计提供了一种可靠的验证方法，避免了潜在的逻辑错误在后续阶段的引入，提高了整个设计流程的可靠性。

5. 问题解决能力的提升：在实验过程中，针对设计中出现的时序问题、逻辑错误等，我们培养了快速分析和解决实际问题的能力，这对于未来面对复杂

---

工程挑战具有重要意义。

6. 团队协作与沟通能力：通过小组合作进行实验，我们小组在团队协作中加强了沟通技巧和协作能力，这对于 ASIC 设计这样一个高度互动的工程过程尤为重要。

## 十一、总结及心得体会：

在这次实验过程中我接触到数字 IC 设计流程中**逻辑综合部分的实践**，并额外拓展了一致性验证的学习。

我们为了逻辑综合的正常进行，从图形化界面开始逐步探索，最终理解了绝大多数脚本的含义，**恰当的修改与 ultra 的策略**是我们能够成为**全班最高频率**的保证（当然，我们后端也是非常努力才能保持这个频率）。

逻辑综合的重要性与收获我想我在“实验步骤”中已经体会很深深刻了，这里就不赘述了。我想谈谈关于“**形式验证**”的事情。其实，我们一开始并未直接想到形式验证这个方法，我们小组遇到了前方通过但是门仿不通过的难题，咨询老师与助教后，我们探索式的打开了 Formality 的图形化界面，在我们的共同努力下，我们不仅学会了大致的使用方法，而且通过 debug 功能找到了 RTL 代码中设计的错误，这部分花了我们小组一天半的时间，辛苦的同时也收获满满。

最后想感谢 Even 老师，他是我们小组的“全科战神”，想感谢天舒姐、雪淞哥和俊凯哥，你们是我们经常打扰的对象，这份报告中有很多你们帮助的身影，想感谢以及经常也到来的“后端高手”——皓宇哥，以及其他组内的研究生学长学姐。想在这里不恰当的呈现以下我们的成果，亦是表达我们小组的不胜感激，是你们的引导与帮助成就了非凡的 11 组！

表 1 参考设计与 11 组设计对照表

版本	参考设计	我们的100M版本	我们的250M版本	我们的 <b>294M</b> 版本
寄存器大小	13*13*8	4*11*8		
DC最高频率	50M	100M	250M	300M
DC后面积	240000	150177	157046	<b>169415</b>
后仿反标率	约为45%	70% & 100%		
第一个周期	-	<b>145</b>		
总周期	-	<b>12124</b>		

## 十二、对本实验过程及方法、手段的改进建议：

1. 希望可以有逻辑综合更具体脚本的讲解，目前我们的讲解是针对图形化界面的，可以参考一下我的 **8.1 - 8.3** 小节，希望能够抛砖引玉了！
2. 别的真没有了，IC 2 真的太好了!!!

## 十三、参考文献

书籍：《专用集成电路设计使用教程》、《数字 IC 系统设计》等；  
 来自 EETOP 的资料若干；  
 DC 手册。