第三章 快速资源估算模型

## 3.1 Verific及相关数据结构

### 3.1.1 Verific平台介绍

Verific Design Automation是Verific公司开发的一款用于构建SystemVerilog、VHDL和UPF的解析器平台，使其客户能够以低成本快速开发高级EDA产品。

Verific分析创建解析树并执行类型推断以解析标识符的含义。解析器/分析器模块支持整个系统Verilog IEEE 1800（包括Verilog IEEE 1364）、VHDL IEEE 1076和UPF IEEE 1801语言。生成的解析树/数据模型附带了大量的C++ API。

Verific的解析器平台被分发为C++源代码，并在所有32位和64位UNIX、Linux、MAC和Windows操作系统上进行编译。Verific的解析器平台目前在全球许多公司的生产和开发中使用，从EDA初创公司到《财富》500强半导体供应商。

我们将使用Verific，并基于Verific生成的语法树来完成后续的实验操作，由于Verific具有当前行业认可的Parse和Analyze功能，并基于C++平台实现开发，提供广泛的语法树API作为二次开发的基础，故我们选择在Verific的基础上实现可商业化使用的快速资源估算模型。首先需先对Verific所构造的基本数据结构进行相关了解及分析，从而选择构造合理的数据机构及API进行模型建模实现；Verific主要数据结构分为以下两个主要部分：基于RTL的语法树结构和基于Netlist的Database结构,下文将主要介绍资源估算模型主要使用的数据结构。

### 3.1.2 Verific-Parse-Tree数据结构

Verific在RTL级别Verilog VHDL的语法树Parser上有丰富的经验，通过对语法进行文本级别的流解析在内存中构建层次树（Hierarchy-Tree）结构用以表述Verilog VHDL语法信息，层次结构树（hier Tree）是设计层次结构的独立于语言的表示形式。设计的层次结构由Verilog语言中使用的模块实例化和生成语句定义。随后在本文档中，我们将此数据结构称为hier树或Parse-Tree。这个层次结构树结构是一个轻量级的静态详细层，位于分析的解析树之上，保存着关于各种设计对象的关键信息。树的每个节点都存储有关它所表示的设计对象的信息，以及该层次结构路径中特定于该设计对象的参数和泛型等常量的值。语法树结构主要由以下几种类结构：

①VeriTreeNode是所有Verilog树节点元素的基类。因此，它包含行文件信息和在所有解析树节点上的数据信息。其中VeriExpression、VeriIdDef、VeriStatement等中非常常用的派生类，用于对Verilog Design作进一步数据构造，下图为VeriTreeNode的继承示图：

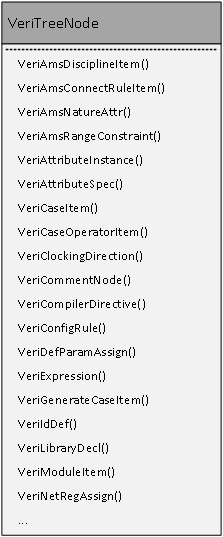


图3.1.2 图1

②VeriIdDef是标识符定义树节点，是标识符节点的基类。Verilog中声明的任何标识符都将成为VeriIdDef结构类。下图为VeriIdDef的类关系视图，其中VeriidDef的继承类VeriModuleID、VeriOperatorID、VeriInstID等为进一步对Verilog语言语法的描述类。

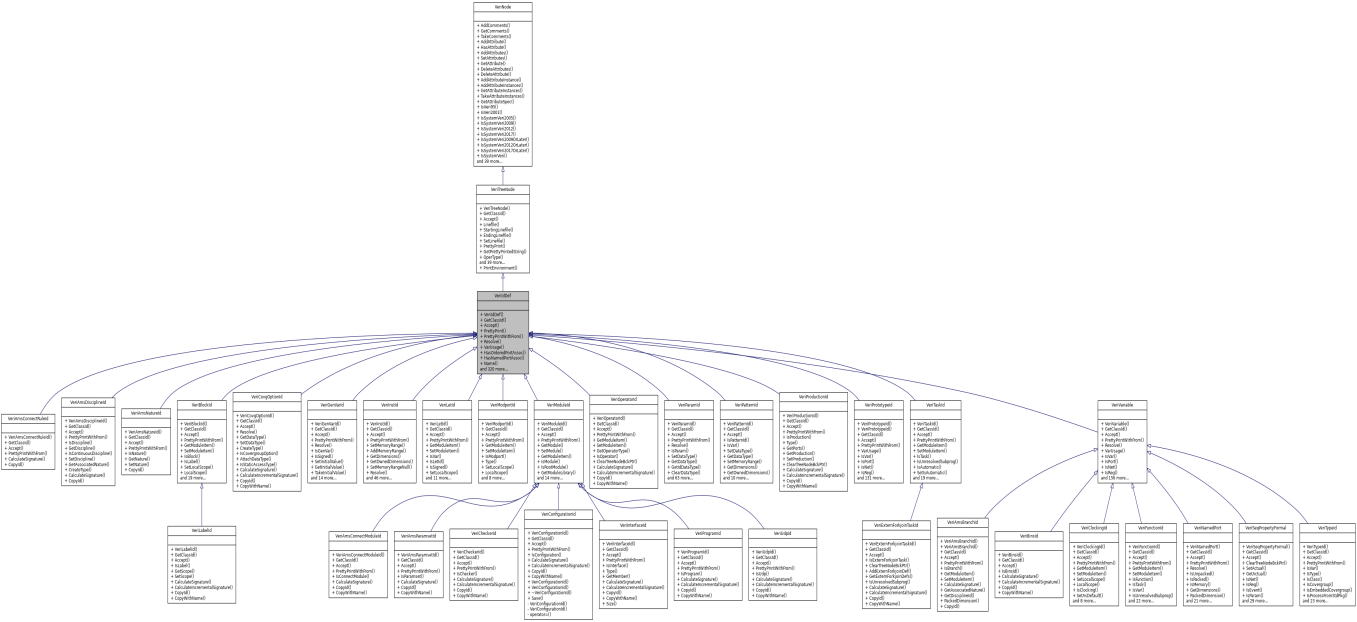


图3.1.2 图2

③VeriExpression是所有Verilog表达式解析树节点的基类。其继承类用于描述包括常量、名称引用、运算符以及出现在实例化端口关联列表和模块端口列表中的端口关联信息等。下图为VeriExpression的类关系视图。

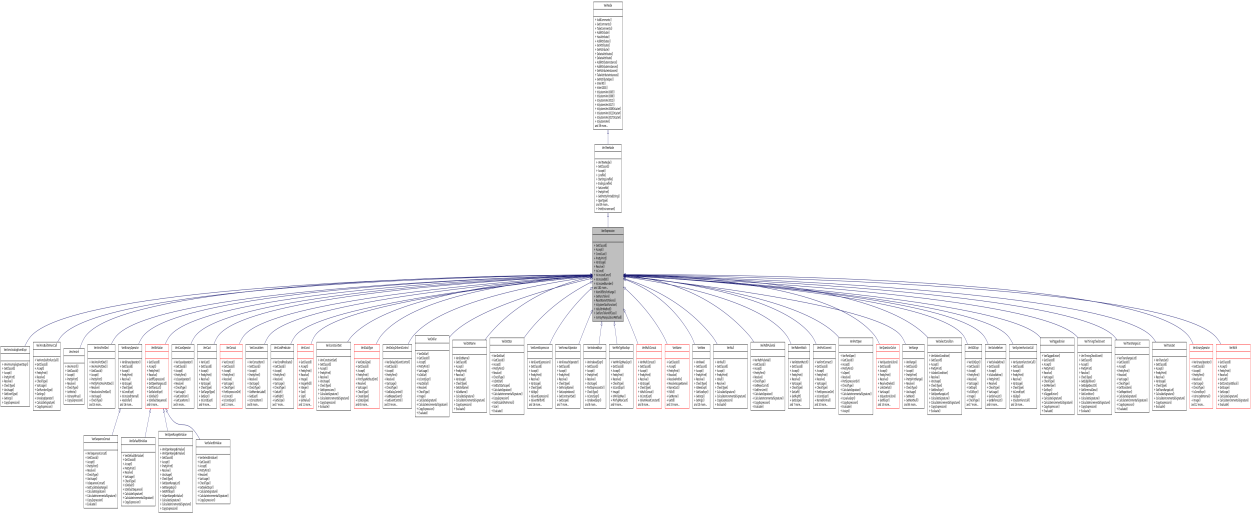


图3.2.1图3

④VeriStatement是所有Verilog statement 节点的基类，包括所有Always语句块等语法

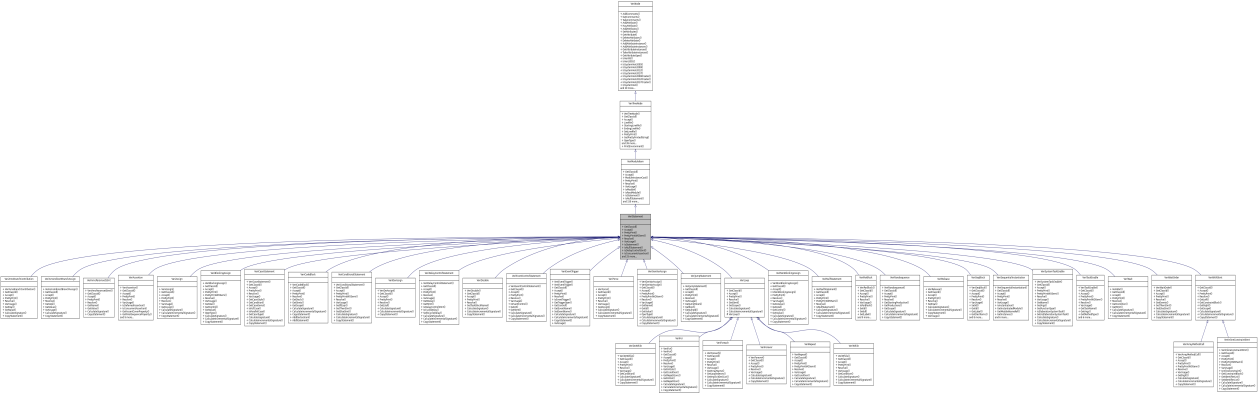


图3.2.1图4

⑤VeriModule位于任何Verilog解析树的顶部，是模块节点的基类。它以解析树的形式表示一个完整的Verilog模块。VeriModules存储在静态哈希表中，该哈希表表示已分析的所有Verilog模块。

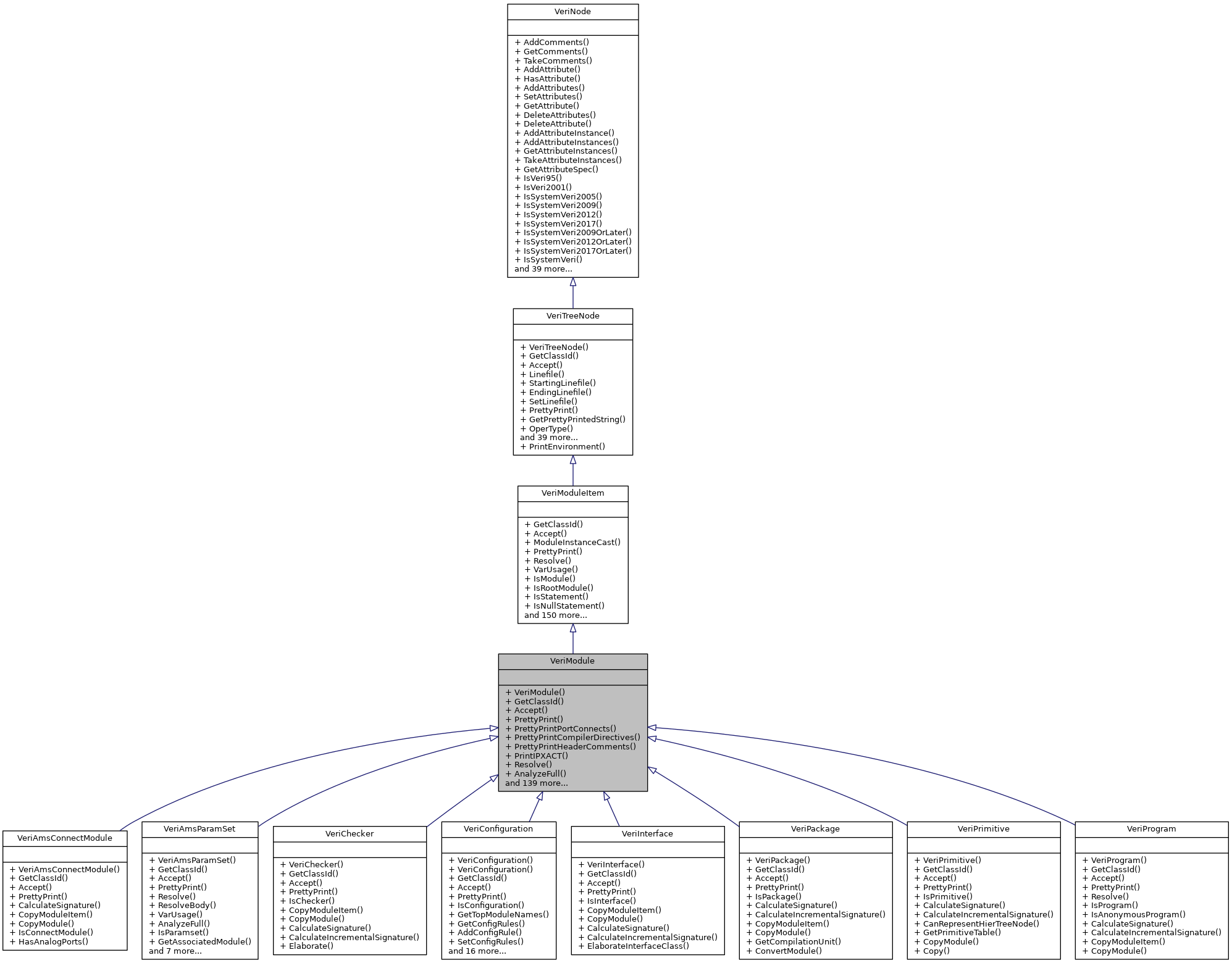


图3.2.1图5

上述5种数据结构为Verific中最基本的基类，所有语法树的语法表述及存储均可由其中一种进行表述，在本文中所有RTL代码的语法将会使用上述类进行语法树处理及构造，从而获取正确的设计层级及内容；

### 3.1.3 Netlist数据结构

由于传统技术映射是在网表级的基础上通过器件库使用图划分算法，将网表节点映射至对应的器件，所以为构建基于传统方法的快速资源估算模型，也需要使用Verific的Netlist数据结构来实现，Verific Netlist Database数据结构如下：

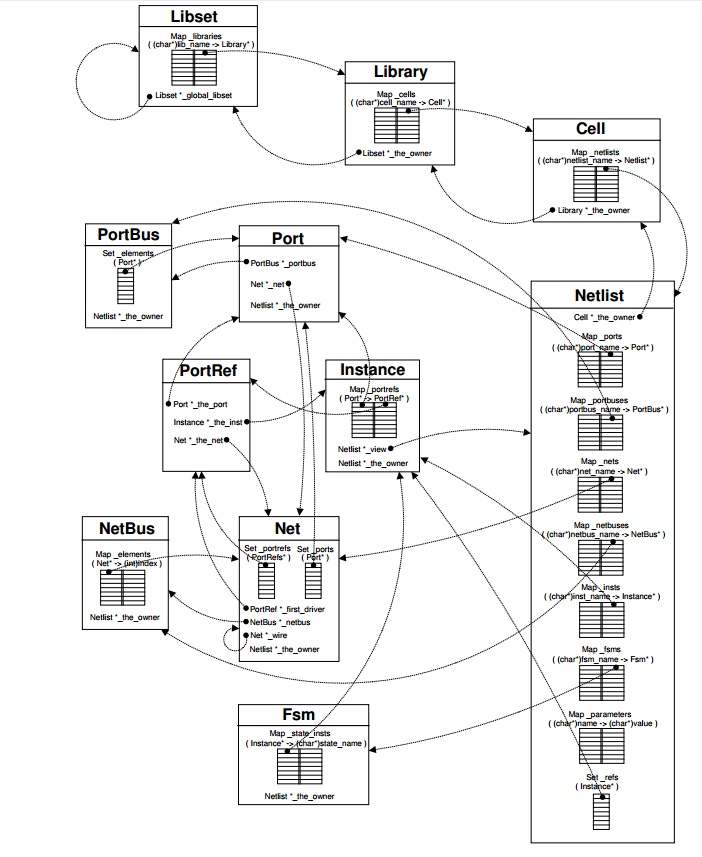


图3.1.3 图1

①Libset是一个单例对象（只实例化一次的对象），它包含（char\*）lib\_name到Library\*映射的映射。Libset没有所有者，因此被视为全局对象；

②Library是Libset拥有的对象，它包含所有Module模块命名到cell指针映射的映射。库对应于它的VHDL或Verilog等价物。例如，在VHDL中，将用户实体/体系结构对分析到一个名为“user”的库中，然后详细描述解析树，那么将创建一个名为“user”的库对象并将其添加到全局库集中。

③Cell是库所拥有的对象，它包含Netlist\_name到netlist\*映射的映射。当处理Verilog时，单元是包含单个网络列表的模块。就VHDL而言，单元是一个可以有多种体系结构（网络表）的实体。

④网络列表是数据库层次结构中的关键组件。它是一个由单元拥有的对象，包含其他网络列表实例化形式的功能（通过称为网络、端口和端口引用的对象连接在一起）.

它还包含一组自身正在实例化的引用（从其他网络列表中）。

如图所示，基本数据结构为LibSet用以存储所有的Library库，Library库为设计网表存储基本数据结构，Cell结构对应为VHDL代码中Module模块，每个Cell尤其仅有一个对应的Netlist结构，Netlist结构用以存储该Module所有互联信息及实例化信息，如：Port、Net、Instance，包括Net位宽、方向等；

基于Verific的上述数据结构以及相关的API，使得我们可以在后续基于C++语言实现传统快速资源估算模型提供基本数据基础；

## 3.2模型构建

### 3.2.1 模型流程分析

基于第二章节所述逻辑综合及资源映射原理，我们对模型的构建做出进一步的分析，模型输入为RTL级设计文件即Verilog VHDL文件，模型输出为该设计所需FPGA资源数目，其中对于面向大型RTL设计Partition中所需的最重要的关键的资源为FF及LUT，文件流如下图所示。



基于Technology Mapping的原理进一步分析，为实现基本的资源估算，则需要将输入的原始设计从RTL级代码转换至Netlist级别才能实现进一步的Technology Mapping，但是由于获取最直接的网表解构就需要使用综合器对RTL设计进行综合，通过前面对综合原理的分析，可知道综合主要分为以下几步，即RTL设计细化，Flatten，综合优化，综合约束，技术映射等几个步骤，但由于需要面向大型FPGA设计，综合器耗时太长，所以我们选择通过使用Verific来对设计进行静态细化及初步细化，使RTL级设计在内存中转变为伪网表，通过在内存中对伪网表进行Flatten操作，将伪网表所有的层级结构Ungroup至顶层，从而实现与基本综合器相对应的操作的快速版本，对Flatten至顶层的设计依据TechMaping原理使用已经获得的器件库进行器件匹配，基于本节分析及初步设计结果，着手使用Verific对传统资源估算进行进一步分析及模型构建，具体流程如图所示，关键操作及相关原理将在下文着重分析。



图3.2.1：流程图

### 3.2.2建模实现

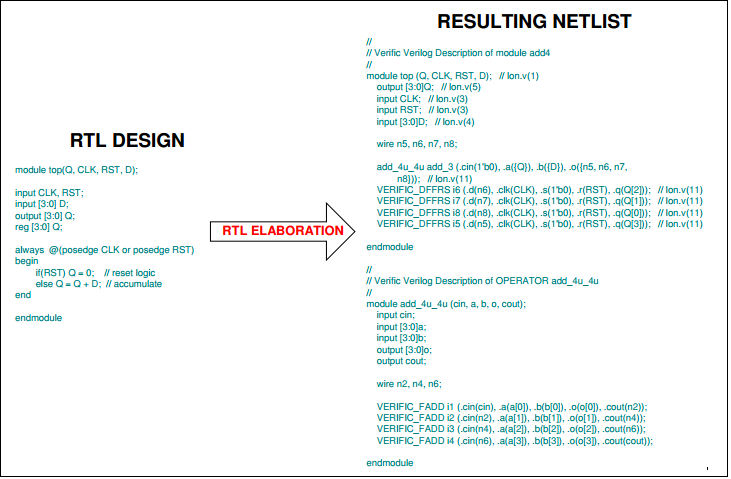
我们以上一节中对快速资源估算模型分析的流程进行进一步的分析，同时进行实现：

1. 通过将原始设计使用Verific的Parser接口进行设计读入，由于需要商业使用，需要考虑多种用户输入情况，首先考虑常见的通过Filelist进行读入，并支持对于文件正确性校验及报错，并支持输入文件包括.sv\.v\.vh\.svh等多种文件输入，同时支持通过路径进行设计输入，并对设计进行正确性校验，如图所示

****

**2、对设计进行Elaborate操作，Elaborate操作介绍：**

通过对上一步骤获得的完整输入设计的Parse-Tree进行Elaborate，从而获得对应设计的由Verific GTec Cell构成的伪Netlist的设计文件，Elaborate后的设计语法依旧满足Verilog HDL的语法信息，RTL Elaborate会使用门级的网表来表示之的RTL级代码。Elaborate会将RTL级代码进行逻辑展开，获得具有符合电路逻辑的门级视图网络和门级逻辑电路的VerilogHDL表述，该步骤为FPGA设计流程中综合的前置步骤，为下一步进行优化的技术映射做好基础工作，具体细节可参照图上所示；**具体如图：**

****

**3、对设计进行Flatten操作，将设计进行展平操作**

在步骤2的基础上，我们将对设计进行Flatten操作，Flatten操作顾名思义是将现有包含层级关系的层级设计进行展平操作，也可以称为ungroup操作，参考下图样例可以更好的说明Flatten操作对于设计的操作，**如图所示：**

****

****

在FPGA设计综合流程中Flatten操作会将一些层次树底层的Module、逻辑提升至顶层，将可以合并的逻辑进行合并，同时将一些冗余的互联Net进行优化，为进步一的综合优化和器件映射做好准备，因为对于FPGA或者数字集成电路而言，它是没有层次的概念的，所以Flatten操作可以更好的让寄存器级设计与物理器件之间建立更进一步的联系；

**4、读入基本器件库，器件库内容如下；**

由于我们中所设计的快速资源估算模型是面向Xilinx的FPGA的模型，所使用的基本器件库是Xilinx公司Virtex系列器件库，其中主要包含的为基本逻辑单元即CLB，具体为MUX、LUT1~6、各形式的触发器即FF、FDR等，一些IP器件由于知识产权等问题无法直接使用；以LUT、FF、MUX为例进行说明**：**

**5、进行匹配操作，即基本技术映射；**

对步骤4中完成Flatten后的网表文件进行TechnologyMapping，具体操作为：将Flatten后的Netlist网表文件依据设计逻辑对将步骤四中的器件库的器件进行匹配和覆盖，匹配后的网表将不存在基本的门级逻辑（布尔逻辑），而将会被器件库中的基本逻辑器件所替代，如图中所示：分别为MUX、D-FLIP-FLOP触发器、4-输入LUT器件库代码。

****

**图xx**

1. **统计映射结果展示**

下图为以上述思路构建的快速资源估算模型的初步估算结果，如图可以获得设计所使用的LUT1-6、BUFG、FF器件数量：



1. **模型估算结果分析**

为对初步建立的模型及输出结果进行进一步的分析，我们分别设计复杂度由小至大的多个openCore上的经典Case使用快速资源估算模型进行估算同时与赛灵思公司的EDA工具Vivado进行综合后输出的资源利用率报告进行比对，结果如下表所示：



## 3.3 模型优化

通过上图与Vivado综合后结果对比，可以发现，在无优化的情况下仅对设计进行细化及基本技术映射虽然可以得到基本的资源，且在速度上得到了有效的提升，但是存在较大的误差，通过分析赛灵思的综合器及TechMapping，我们可以知道在与FPGA板载器件进行基本逻辑单元及触发器映射时存在优化项，可以理解为约束项，如：面积约束、功耗约束、时序约束、逻辑门约束等多种约束；结合面向大型RTL设计分割流程的实际需求，增加以下优化方向：

### 3.3.1 BottomToUP计算方法

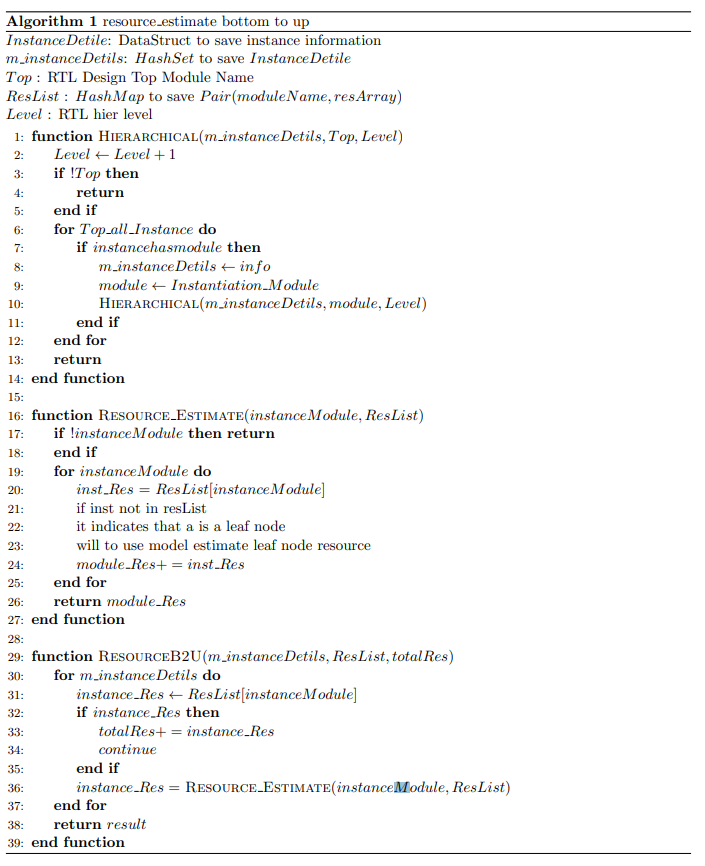
由于使用Verific进行Elaborate处理及Flatten后无法得到每个层级对应的资源数量，且对整个设计进行Elaborate及Flatten所花费的时间较高，同时由于该资源估算模型面向RTLPartition进行使用，所以需要每个层级每个Inst的资源用于确认RTLPartition时正确的分割边界，所以我们从Verilog HDL与FPGA片上资源映射关系角度出发，在不考虑优化的前提下，每个Module的资源是由其所例化的Instance对应的Module的资源与该Module本身的胶水逻辑所组成，如下图样例所示，红框标注分别为例化Instance与胶水逻辑；



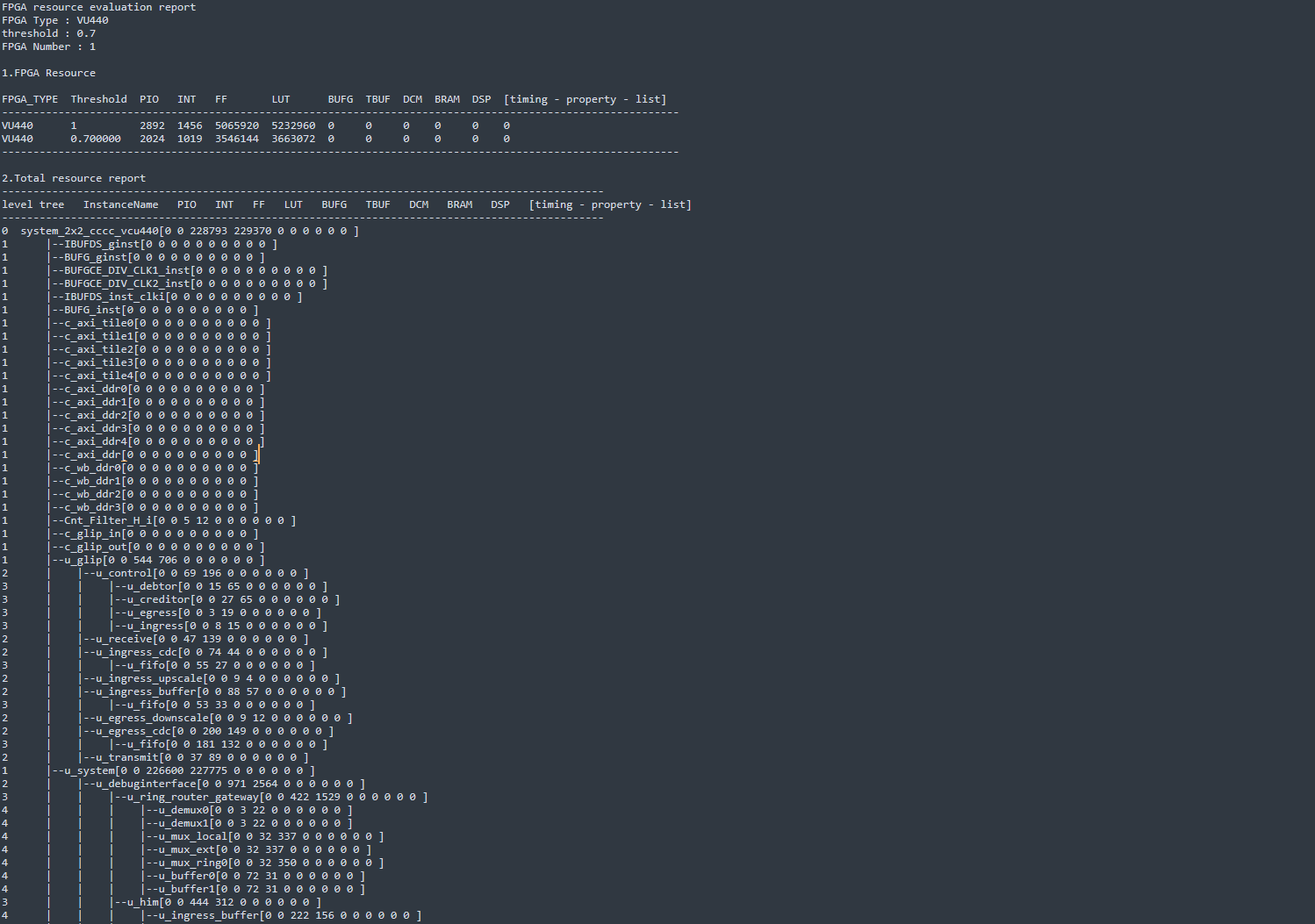
以上述分析为出发点，我们希望可以结合Verilog HDL设计建模方式，以该方式为参考，能够更加准确，更加快速的计算每个Module、Instance的资源，我们以RTL设计及计算资源所需数据为出发，设计以下数据机构用于存储每个节点的资源，如图所示：



通过以上述数据结构，用以实现Bottom to up的计算方式，通过树遍历算法及递归算法构造存储当前Parse\_tree对应的树结构及查找表，该查找表用以存储已被计算过的Module的资源，算法伪代码如图所示：



通过正向遍历构造结构，逆向计算资源的方式，实现快速层次资源估算，输入为RTL级Verilog HDL设计，输出为该设计每个层级所使用的资源数量及相对应的树结构图，若在输入时提供FPGA型号和资源，程序也会评估该设计所需要的FPGA的数量。如图所示：



图中为开源设计openrisc，经过该快速资源估算模型的Bottom-to-Up计算所获得资源数量与每个Hierarchy的资源数量；

### 3.3.2冗余逻辑优化及合并

在逻辑综合和技术映射时，综合开发环境会对电路规模、速度进行优化。综合的优化过程中会去掉常数输入、悬空输出等信号，还会针对源文件中未连接的输入或具有多个驱动的输出等问题报告错误或者警告。有时即使设计上有缺陷，工具也会按照既定的方式自行解决，但所生成的电路可能并不符合开发者的本意，所以调试时需要对浸膏信息特别留意。综合优化的过程还会对逻辑进行组合置换，因此需要注意的是，并非源文件中所有模块、寄存器、信号都会出现在综合结果的网表中。

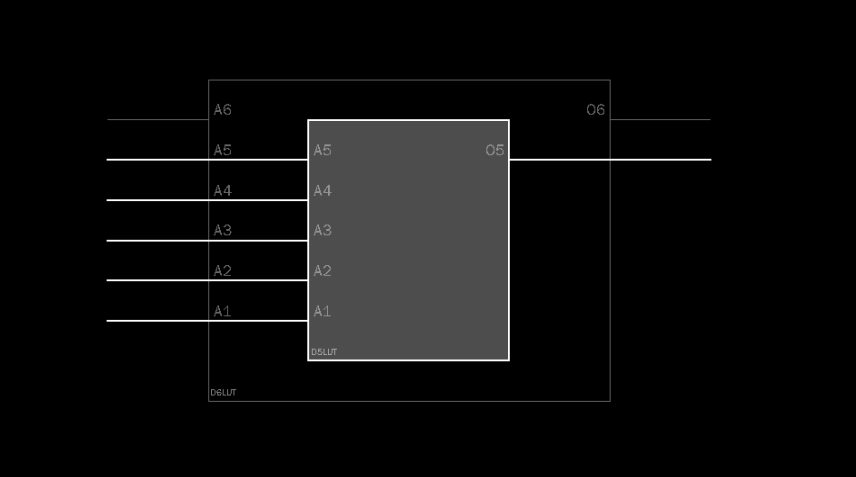
通过前述步骤得到的快速资源估算模型，虽然已经可以得到资源的结果，同时若使用bottom2up的算发可以得到每个层级的资源结果，但是我们通过第二章的综合和技术映射的原理可以得知，在实际的综合和技术映射过程中，是对设计进行了大量的优化和处理的操作，所以如果我们不对当前的快速资源估算模型进行优化和约束，那么该模型的结果是不可用，没有价值和意义的；

所以我们为使我们资源估算的精确度更加高，所以需要符合逻辑综合和优化的过程，通过分析逻辑综合流程，在逻辑综合的工程中会对冗余逻辑进行优化，比如一些复选器或单纯的数值传递等非有效逻辑进行合并。

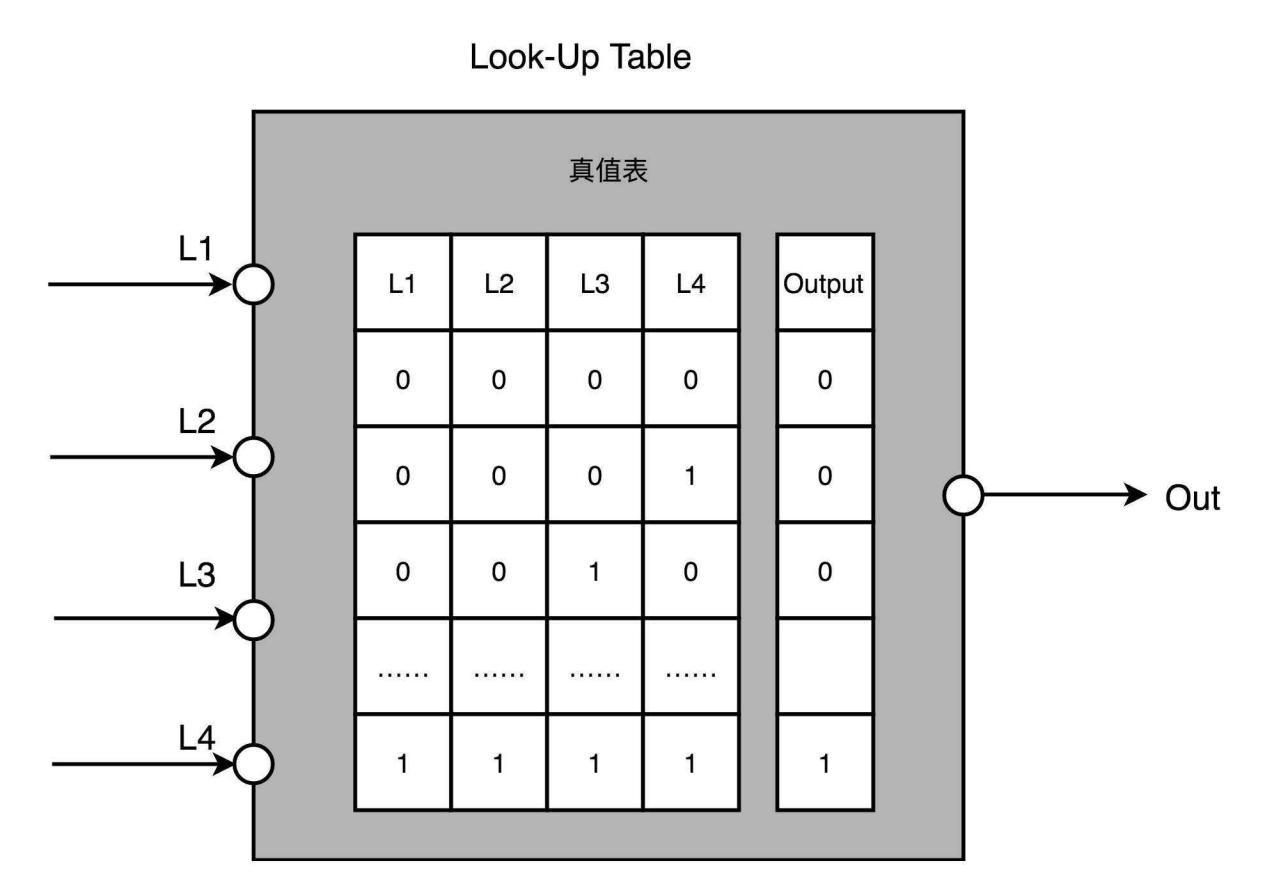
### 3.3.3最小面积优化方法即LUT合并

通过本文第二章中对逻辑综合过程中优化步骤的分析，我们知道在综合开发环境会对RTL代码进行很多优化，从而提高设计的性能、减少硬件资源消耗等，所以在快速资源估算模型中也需加入相关优化项，来对RTL设计进行进一步的优化，从而使得资源估算模型的结果更加的精确。

首先我们考虑资源共享，即在互斥条件下共享算术逻辑单元，主要有关系运算、加法运算、减法运算、除法运算等；



根据FPGA的结构，可以知道不同系列、不同世代、不同型号的FPGA在逻辑单元的组成上是有很大区别的，比如对于赛灵思的VU440，其基本逻辑单元即CLB单元是以LUT5为基础，同理也有以LUT1、LUT2、LUT3、LUT4、LUT6为基础，支持至最高为LUT8的FPGA，由于LUT1~8之间存在换算关系，LUT的本质就是**如图所示**的2进制查找表，故而2个LUT1可以合并组成为一个LUT2，同理以LUT5为Base的FPGA，可以使用多个LUT5来组成LUT6等高维度资源，从而有效的是缩小FPGA布线后使用的面积即器件使用的数量；



在资源共享时我们遵循以下原则：

1. 运算符和操作数均相同：一定可以贡献逻辑单元
2. 相同的运算符，并且仅有一个共同的操作数，那么通过使用一个多路开关可以实现共享逻辑单元
3. 相同运算符，完全不同的操作数，可以共享逻辑单元
4. 不同的运算符，相同的操作数，可以共享逻辑单元
5. 不同的运算符，仅有一个公共操作数，可以通过引入多路开关来实现共享逻辑单元
6. 不同运算符、不同操作数，可以通过引入两个多路开关，从而实现逻辑单元的共享

除去资源共享优化外，本文基于对RTL代码的分析，增加公共子表达式优化，即保存公共子表达式的值，在其后引用该表达式的地方可以直接赋值。

所以基于以上原理，在进行冗余逻辑优化和合并的基础上，可以增加最小面积优化方法，在资源估算时，以LUT4为base进行初步资源估算，在当前基础上，以LUT4之间的优化方法，我们可以以高维度LUT为Base的FPGA作为方向，将多个LUT在合理的范围内进行合并，合理的范围即在一个Module内的组合逻辑或连续赋值、阻塞赋值、非阻塞赋值等LUT合并至LUT6，从而实现以最小面积为目标的优化策略；

****

## 3.4本章小结

本章详细叙述了，在以第二章理论为基础的前提下，使用Verific构建面向RTLPartition的大型FPGA设计的快速资源估算模型，同时提供多种优化及约束方案，对模型进行约束及二次优化，使得对设计在FPGA上的资源估算精确度更好，同时提供多种额外功能，如用户指定资源输入、对应不同FPGA的使用数量评估等作为商业软件对用户的友好性功能；同时也为后续基于人工智能的FPGA资源估算模型做好基础工作。