Documentation

Student: Shangkun LI
Student ID: 20307130125
Department of Physics

Repository: https://github.com/ShangkunLi/Logic_Synthesis_phyLS.git

本Project主要通过调用phyLS逻辑综合框架进行逻辑综合。

目录

1 项目说明

1.1 功能说明

本项目利用phyLS提供的逻辑综合工具代码框架，完善其中的balance和rewrite功能。然后利用该工具对10个benchmarks进行逻辑综合，并记录结果。

1.2 文件结构

```
Logic_Synthesis_phyLS   # 项目文件夹
├── phyLS   # 子项目文件夹（phyLS文件夹）
│   ├── benchmarks  # 测试用例
│   ├── build   # 编译文件夹
│   ├── CMakeLists.txt  # CMake文件，用于生成Makefile
│   ├── docs    # Documentation
│   ├── lib # 依赖的library
│   ├── LICENSE
│   ├── README.md   # 说明文档
│   └── src # 源文件
└── README.md   # 说明文档（本文）
```

2 代码说明

本项目主要基于已有的phyLS架构实现了balance和rewrite的功能。该部分可分为两部分构成：

1. 介绍balance算法的实现思路
2. 介绍rewrite算法的实现思路

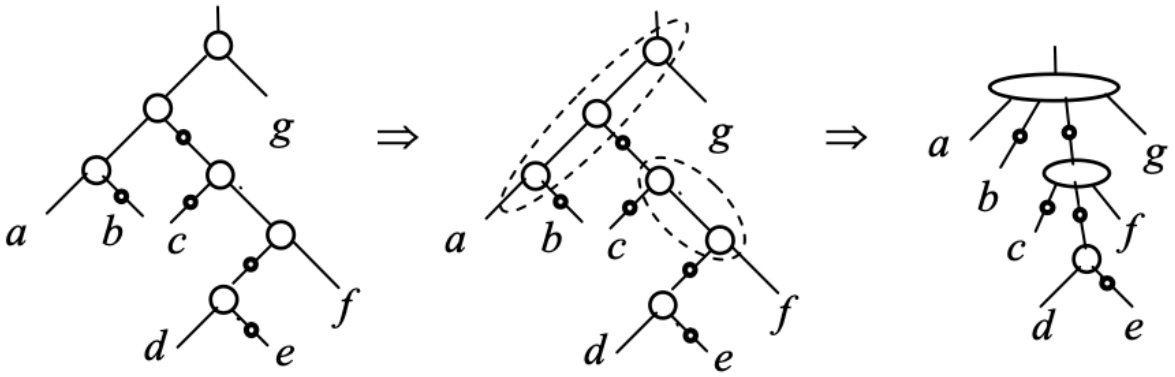
2.1 Balance算法实现思路

在逻辑综合优化中，我们常常使用AIG来表示逻辑电路，从而简化优化流程的目的。所谓AIG，是指And-Invertor Graph， 是由二输入与门和反相器组合成的简单逻辑单元。

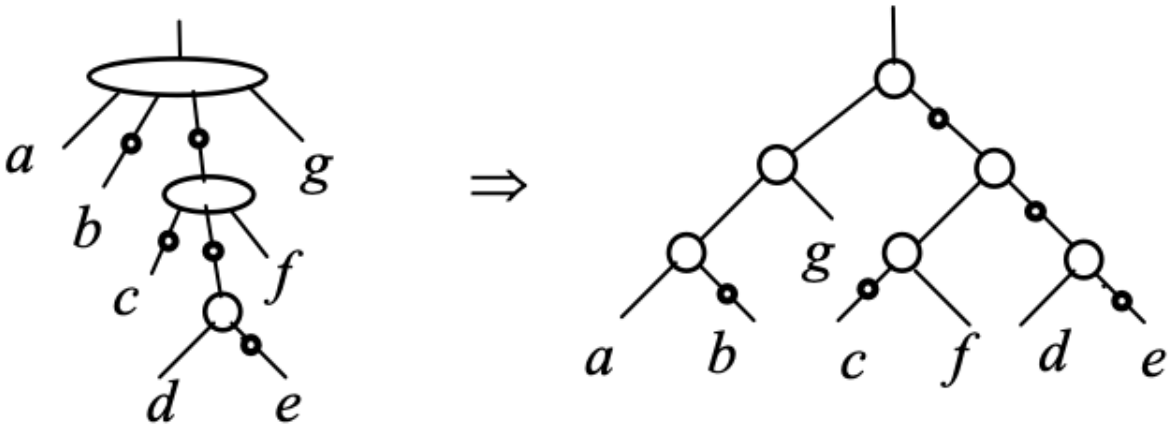
1. AND-Balance

   Balance按照拓扑顺序应用，并选择多输入与门的最小延迟树分解。
   利用AND-Balance算法时，balancing分两步，covering和tree-balancing。

   其中covering将子集之间没有反相器且没有外部扇出的两输入与门组合在一起形成一个多输入与门：



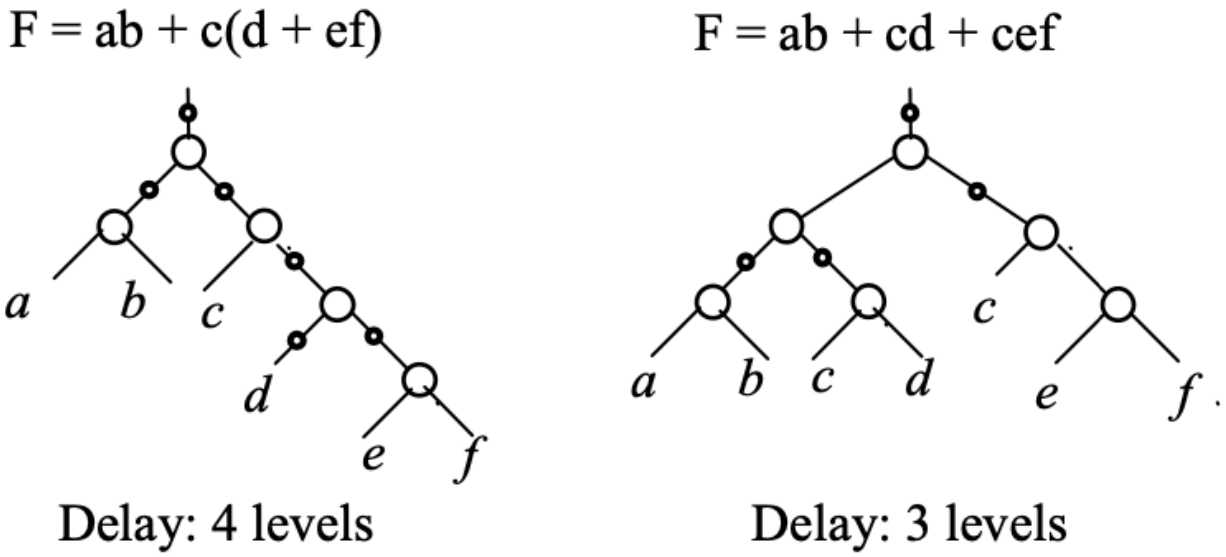
   Tree-balancing将covering得到的多输入与门分解为两输入与门，试图减少AIG的深度：



2. SOP-Balance for Small AIG：

   在规模较小的AIG图（输入变量数<10）中，我们可以根据真值表计算出逻辑函数，并将逻辑函数用SOP（Sum of Products）表示出来。随后对SOP表示形式的AIG图应用AND-Balance算法，就可以实现balance。

   如下图所示，我们可以将逻辑电路从4层减少到3层.



3. SOP-Balance for Large AIG:

   实际的AIG往往规模很大，这就不能用SOP的方式实现balancing。

   但我们可以通过将整个AIG划分为多个区域，对于每个区域使用SOP-Balance算法的方式实现balancing。这也是本Project中使用的方法。

   本Project中通过`foreach_co()`函数对每个Node实现遍历，并通过`balance_rec()`对每个Node的对应的割集进行SOP-Balance。

在phyLS中的算法调用代码请见`phyLS/src/core/balance.hpp`

2.2 Rewrite算法实现思路

Rewrite过程中是指通过执行AIG的重写过程，实现AIG节点数的减少和逻辑层级的减少。

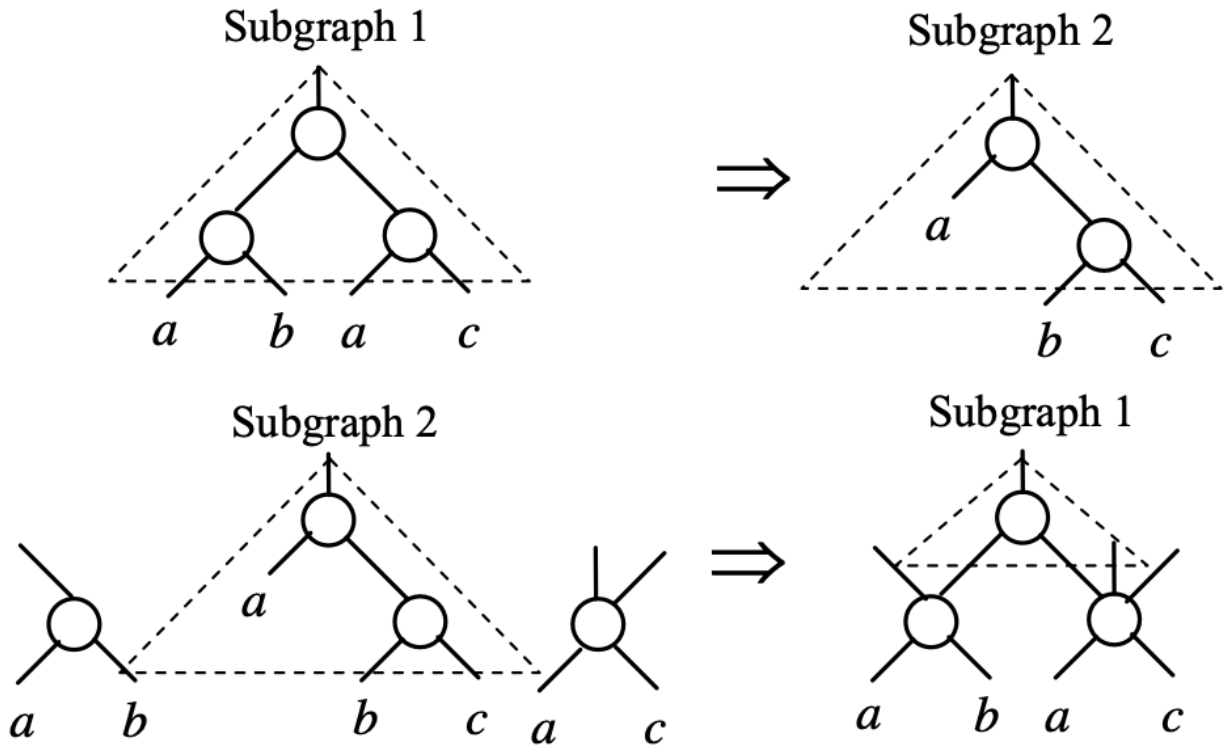
具体过程中，我们首先对每一个割集的子图进行遍历，每一个子图都隶属于某一个NPN等价类，将该子图的等价类记录下来。

如果某一子图可以被同一等价类中的某一个子图替换掉，则将该子图的引用数减1，直到该子图的引用数减为0，将该子图移除。同时，如果有新的子图加入，则需要考虑新加入子图的开销。移除旧子图获得的收益减去新加入子图的开销，就是总的收益。如此对所有子图进行遍历，将保留总收益最大的替换操作即可实现rewrite。

在phyLS中的算法调用代码请见`phyLS/src/core/rewrite.hpp`

其中关于AIG的子图库的选择有多种，我们在此提供了两种选择，并用Complete AIG Database实现本实验。

示例如图所示。



3 编译及结果

3.1 编译

从远程仓库下载源代码

```
git clone https://github.com/ShangkunLi/Logic_Synthesis_phyLS.git --recurse-submodules
```

创建编译目标文件夹

```
cd phyLS
mkdir build && cd build
```

进行编译

```
cmake ..
make
```

执行程序

```
./bin/phyLS
```

3.2 结果

对20个benchmarks分别进行测试，分别记录其优化前和进行balancing、rewriting优化后的相关结果。

在表格中用粗体标记出优化后有所改善的指标。

优化前/后	i/o	gates	level		#gates	area	delay
adder	256/129	1020/1020	255/255		639/639	1849/1849	204.9/204.9
arbiter	256/129	11839/11839	87/87		6678/6678	18773/18773	70.7/70.7
bar	135/128	**3336/3141**	12/12		**2319/2161**	**5911/5693**	10.2/10.9
cavlc	10/11	**693/687**	16/16		**485/483**	1212/1218	**14.3/14.1**
ctrl	7/26	**174/146**	10/10		**113/110**	**275/265**	**8.3/7.9**
dec	8/256	304/304	3/3		296/296	648/648	3.7/3.7
div	128/128	**57247/41933**	4372/4406		**55376/40950**	**127233/99321**	**3516.5/3442.3**
hyp	256/128	**214335/212833**	24801/24802		116061/116717	364855/365750	**16771.4/16723.3**
i2c	147/142	**1342/1262**	**20/16**	After tech mapping	**1010/986**	**2458/2346**	**14.6/13.2**
int2float	11/7	**260/229**	**16/15**		**158/154**	**429/393**	12.9/12.9
log2	32/32	**32060/29901**	**444/392**		14037/15629	46105/48845	**294.9/273.7**
max	512/130	2865/2865	**287/229**		**2474/2363**	**5650/5386**	**205.5/177.8**
mem_ctrl	1204/1231	**46836/46701**	114/114		**33361/33316**	**81657/81466**	87.8/87.8
multiplier	128/128	**27062/24760**	**274/264**		11676/11712	**40751/40296**	**209.2/208.8**
priority	128/8	**978/832**	**250/246**		**756/675**	**1736/1515**	**199.3/197.7**
router	60/30	**257/245**	**54/27**		198/213	527/547	**37.5/20.7**
sin	24/25	**5416/5188**	**225/183**		3387/3518	9462/9883	**149.4/133**
sqrt	128/64	**24618/18768**	5058/6048		**19211/13827**	**44523/35227**	**4235.8/4088.2**
square	64/128	**18484/17606**	**250/249**		**9704/9669**	**28288/27812**	**199.4/199.2**
voter	1001/1	**13758/10861**	**70/69**		**8325/5613**	**22807/16929**	**53.5/46.2**