逻辑综合 project

一、框架软件概述

phyLS 工具是一个基于 EPFL 逻辑综合库和开源逻辑综合工具 ABC 的逻辑综合工具。

优化对象

phyLS 工具可以针对不同规模的组合逻辑电路进行优化。这些电路可以是现代 FPGA 电路中的常规电路，亦或是国际上许多通用的基准电路，也可以是以新兴纳米逻辑为基础的电路，例如以多数逻辑为基础的电路等等。当然，我们需要将电路转化为一定的表达形式，例如：AIG[1]文件、bench 文件、verilog 文件等，才能够被工具所识别和优化。同时，在工具的内部也可以自行输入真值表，并进行优化、转换、输出等操作。因此，我们的工具具有良好的兼容性。

优化目标

“功耗、延迟、面积”一直以来都是衡量电路性能优劣的三个重要标准。因此，phyLS 工具从电路的逻辑深度和面积出发，最终分别来实现电路延迟和面积的优化。功耗的降低从一定程度上也受其他两个指标的影响，所以逻辑优化尽管无法直接优化功耗，但也从一定程度上影响了电路的功耗。

二、基本软件运行说明

系统要求

成功运行 phyLS 工具需要配置以下编译环境：

- Clang 6.0.1
- GCC 7.3.0
- GCC 8.2.0

安装方法

安装需要以命令行的形式实现，安装代码如下：

```
1    git clone -b aig --recursive https://github.com/panhongyang0/phyLS.git
```

[1] 全称为 And Inverter Graphs，即基于与门和反相器的电路图的电路图。

```
2    cd phyLS
3    mkdir build && cd build
4    cmake ..
5    make
6    ./bin/phyLS
```

用户也可以指定编译器的路径来进行编译，只需在 cmake 命令加上后缀，代码如下：
cmake -DCMAKE_CXX_COMPILER=/path/to/c++-compiler ..

三、基本操作命令

I/O 命令

本节将展示 phyLS 中不同的输入输出命令

read_aiger	读取一个 aig 文件
write_aiger	写入一个 aig 文件
read_verilog	读取一个 verilog 文件
write_verilog	写入一个 verilog 文件
read_bench	读取一个 bench 文件
write_bench	写入一个 bench 文件
write_dot	写入一个 dot 文件

所有的输入输出命令只需要输入或者输出符合格式要求并且没有语法错误的相应文件即可，若报错，可先检查文件的正确性

一般命令

这一小节介绍的是 phyLS 中通用的操作命令

alias	别名命令，可将单个或若干个命令自定义成一个命令并可重新命名
current	转换当前的数据类型，详细转换参考命令 current -h
help	显示 phyLS 中所有可用的命令
print	打印当前的数据结构，包括图的类型，基本输入输出
ps	打印数据统计，包括图的类型、输入输出、结点数目和逻辑级数
set	设置环境变量
show	显示相关电路的图形
store	存储当前数据结构，并显示结点数

quit	退出

映射命令

techmap	将大电路 AIG 映射到 ASIC netlist

将大电路切割成小部分处理，类似于动态规划思想

四、需要实现的功能

优化命令

balance	优化电路层级
rewrite	优化电路节点数量

（1） balance: 请添加算法在 /phyLS/src/core/balance.hpp，相关算法请参考 phyLS/lib/mockturtle/include/mockturtle/algorithms/aig_balancing.hpp，可以参考论文 Mishchenko A, Brayton R, Jang S, et al. Delay optimization using SOP balancing[C]//2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2011: 375-382.

图 1 为 balance 算法的伪代码，主要包括以下几个主要步骤：

1、 collect leaves of the AND tree
2、 recur over the leaves
3、 sort by decreasing level
4、 mark TFI cone of n
5、 generate the AND tree
6、 replace if new
7、 remember the substitution and the new node as already balanced

```
subject_graph performSopBalancing (
  subject_graph S,   // S is an And-Inverter Graph
  int K,             // K is the cut size
  int C )            // C is the number of cuts at each node
{
  for each node n in S, in a topological order {
    compute C structural K-input cuts of n;
    for each cut {
      compute truth table;
      compute irredundant SOP;
      perform delay-optimal balancing of the SOP;
      if ( the cut has smaller AIG level than the best cut )
        save the cut as the best cut;
    }
    if ( root node AIG level is reduced using the best cut )
      update AIG structure;
  }
  return S;
}
```

图 1：balance 算法伪代码

（2） rewrite: 请添加算法在 phyLS/src/core/rewrite.hpp，相关算法请参考 phyLS/lib/mockturtle/include/mockturtle/algorithms/cut_rewriting.hpp，可以参考论文

Mishchenko A, Chatterjee S, Brayton R. DAG-aware AIG rewriting a fresh look at combinational logic synthesis[C]//Proceedings of the 43rd annual Design Automation Conference. 2006: 532-535.

图 2 为 rewrite 算法的伪代码，主要包括以下几个主要步骤：

1、 initial node map

2、 enumerate cuts

3、 for cost estimation we use reference counters initialized by the fanout size

4、 original cost

5、 optimize

6、 update cost

```
Rewriting( network AIG, hash table PrecomputedStructures, bool UseZeroCost )
{
    for each node N in the AIG in the topological order {
        for each 4-input cut C of node N computed using cut enumeration {
            F = Boolean function of N in terms of the leaves of C
            PossibleStructures = HashTableLookup( PrecomputedStructures, F );
            // find the best logic structure for rewriting
            BestS = NULL; BestGain = -1;
            for each structure S in PossibleStructures {
                NodesSaved = DereferenceNode( AIG, N );
                NodesAdded = ReferenceNode( AIG, S );
                Gain = NodesSaved – NodesAdded;
                Dereference( AIG, S );  Reference( AIG, N );
                if ( Gain > 0 || (Gain = 0 && UseZeroCost) )
                    if (  BestS = NULL ||  BestGain < Gain )
                        BestS = S; BestGain = Gain;
            }
            if ( BestS == NULL ) continue;
            // use the best logic structure to update the netlist
            NodesSaved = DereferenceNode( AIG, N );
            NodesAdded = ReferenceNode( AIG, S );
            assert( BestGain = NodesSaved – NodesAdded );
        }
    }
}
```

图 2：rewrite 算法伪代码

（3） 库函数，基本包括：

1、phyLS/lib/mockturtle/include/mockturtle/networks/aig.hpp

该文件包括了对 aig network 进行一些基本操作，如 aig.make_signal(n)表示创建名字为“n”的一个信号，aig.is_ci(n)表示判断节点“n”是否为输入，aig.foreach_fanin(){}/foreach_pi(){}/foreach_po(){}表示对每个 aig 节点、输入、输出进行遍历，其中{}中可以添加操作，等等。

2、mark_tfi(n)表示创建节点 n 的 TFI

3、cut_enumeration(aig)表示枚举 aig network 的所有 cut，以便 rewrite 进行操作，详细内容请参考 phyLS/lib/mockturtle/include/mockturtle/algorithms/cut_enumeration.hpp。

五、测试示例

（1） 在终端中运行 phyLS，并读取电路，benchmark 保存在 phyLS/benchmarks/中。

- ./bin/phyLS
- read_aiger ../benchmarks/adder.aig
- ps -a

（2） 进行两步逻辑优化

- balance

- rewrite

（3） 经过一系列优化后，最后一步是对电路进行映射

- read_genlib ../src/mcnc.genlib
- techmap