超图分割算法设计

一. 作业背景

随着集成电路设计的不断发展，硬件仿真逐渐成为了现代电路设计中不可或缺的重要环节。相比软件仿真，硬件仿真能够显著提升仿真速度，尤其在处理大规模电路设计时更具优势。然而，随着电路规模的快速增长，单个FPGA（现场可编程门阵列）的容量变得有限，无法满足高性能设计的需求。因此，为了应对这种挑战，需要使用多FPGA系统（MFS）来处理这些大规模的工程。在多FPGA系统中，系统分割算法的质量和速度成为了关键的挑战。一个优秀的分割算法能够在可控的时间内，生成对时序优化有利且易于FPGA内部布局和布线的分割结果。

二. 作业目标

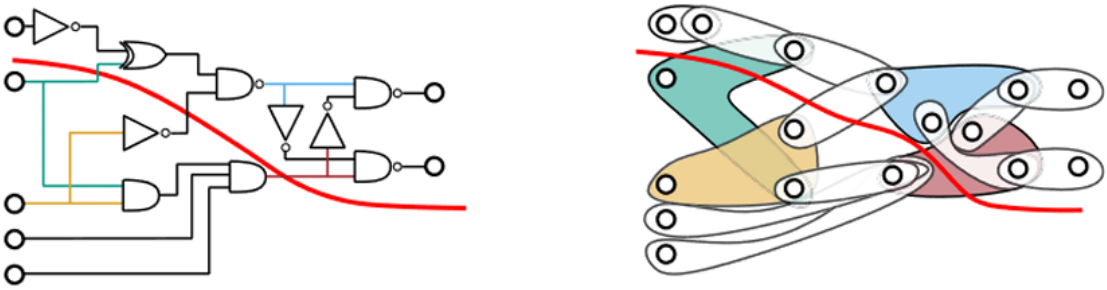
本次作业的目标是设计和实现一个高效的系统分割算法，在满足**FPGA资源约束**的前提下，优化指定的**目标函数**，最终输出合理的**划分结果**。

三. 作业流程

Step1：输入

输入总共四个文件：`design.info`，`design.topo`，`design.net`和`design.are`。其中`design.net`和`design.are`是对于电路网表的描述，`design.info`和`design.topo`是对于fpga的描述。

电路网表的描述



对于一个电路设计，我们首先需要将电路网表映射为**超图**。以上图为例，具体做法是：

- 将门单元表示为节点
- 将线网表示为超边（连接多个顶点的集和）

电路网表即可以表示为超图，方便后续利用超图划分算法进行划分。

本次作业的电路网表有如下特性：

- 电路网表中的每个门单元，会使用**FPGA资源**。共考虑八种资源：FF、LUT、BUFG、TBUF、DCM、 BRAM、DSP、PP。
- 线网表示成的超边是 **加权有向超边**。超边里的节点分为两类：**驱动节点**和**被驱动节点**。

在`design.are`和`design.net`中分别描述了这三个特性：

- `design.are`：描述每个节点使用的资源量。每行一个节点，后面8个数字分别表示8种资源使用量。

```
g1 9 0 2 0 0 0 0 0
g2 5 0 1 0 0 0 0 0
g3 4 0 0 0 0 0 0 0
g4 1 0 0 0 0 0 0 0
```

- `design.net`：描述每个超边的权重以及它包含的节点。每行一个超边，第一个节点为驱动节点，后面的数字表示超边权重，后续若干节点表示被驱动节点。

```
g4 9 g1
g4 10 g3
g2 3 g1 g4
```

FPGA的描述

本次作业的FPGA有如下特性：

- FPGA的**互联**：FPGA之间并不是两两都互联的，信号只能在直接互联的FPGA之间传递。
- FPGA的**资源**：共考虑八种资源：FF、LUT、BUFG、TBUF、DCM、 BRAM、DSP、PP。

在`design.info`和`design.topo`中分别描述了这三个特性：

- `design.info`：描述每个FPGA可用资源量。每行一个FPGA，后面8个数字分别表示8种资源的最大可用量。

```
FPGA1 10 0 2 0 2 0 2 4
FPGA2 10 2 2 0 2 0 2 4
```

- `design.topo`：描述FPGA之间的连线。第一行给出两个FPGA之间存在连线（互连）。

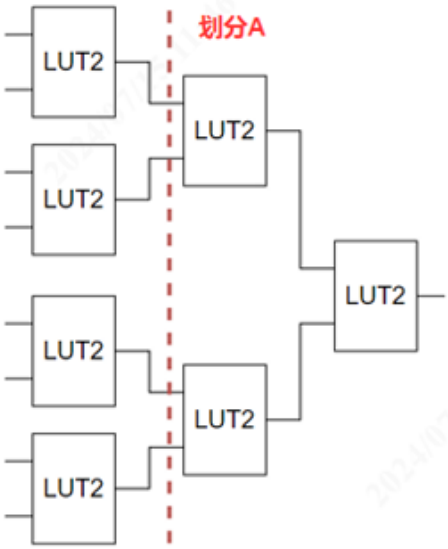
```
FPGA1 FPGA2
```

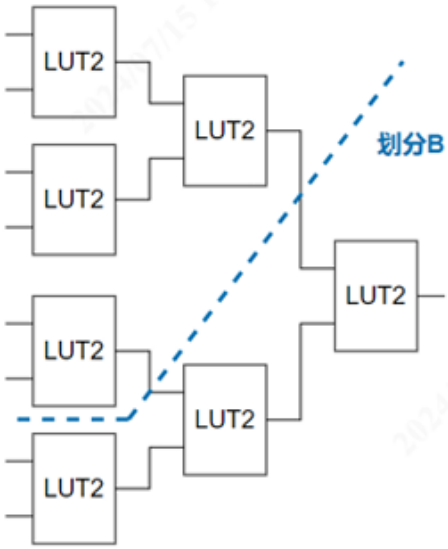
Step2：超图划分

超图划分的工作就是将电路网表的节点和超边合理地映射到FPGA资源上。接下来讲解本次作业的**划分要求**。

优化目标

本次作业的优化目标是**最小化总cut权重值**。由于信号在FPGA之间传播带来的延时要比FPGA内部高很多，所以减少切割数目是有意义的。以下图为例，划分A切割4个超边，划分B切割两个超边，理论上划分B比划分A延时更少。





在本次作业的优化目标公式如下：

$$f_{\text{connectivity}}(\Pi) = \sum_{e \in E_{\text{Cut}}(\Pi)} \omega(e) \cdot (\lambda(e) - 1)$$

上述公式计算了每条切割边e的权重 ω(e)和该边连接的额外fpga的数量 (λ(e)−1)的乘积。通过这种方式，公式试图最小化每个超边跨越的fpga的数量，从而优化布图，减少跨fpga的信号延时。

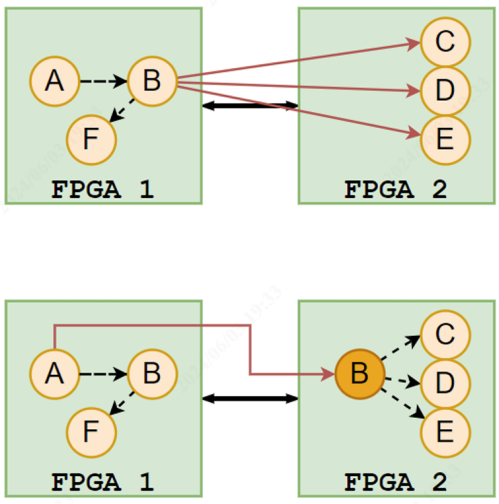
约束限制

本次作业只用考虑资源约束限制：

- **资源限制约束**：分割方案需要保证每一个FPGA上的每一种资源的使用量不超过上限。

拓展项

通过将电路中的逻辑单元复制到多个需要的FPGA上产生信号会带来更好的系统性能，其代价是资源使用量的增加。例如下图所示，此时存在三条边被切割。如果将节点B移动到FPGA 2中，又会带来两条新的边被切割。但如果将节点B复制一份到FPGA2中，则可以只有一条边被切割，从而以资源使用量增加为代价优化了系统性能。



逻辑复制功能当做拓展功能，能够进一步优化**总cut权重**。在约束满足情况下，本次作业不限制复制的节点数。

Step3：输出

输出文件是`design.fpga.out`，其中每一行表示一个FPGA，后续跟若干节点，表示被划分到这个FPGA的所有节点，被复制出的新节点用“*” 标识。

```
FPGA1: g2 g3 g4
FPGA2: g1 g4*     # 考虑逻辑复制的情况
```

四. 评测标准

- 每组需要提交可执行文件（命名为`partitioner`），如果使用了动态库，需要包含相关运行所需文件。
- 可执行文件需要满足如下输入输出要求：
  1. 使用-t参数指定测试数据路径，测试路径下包含四个输入文件。
  2. 使用-s参数指定输出文件`design.fpga.out`路径。

  ```
  ./build/bin/partitioner \
      -t ./data/case01 \
      -s ./design.fpga.out
  ```

  测试数据在项目文件夹`/data`下，命名为`case`加数字，共四个测试case。
- 单个case运行时间限定在一小时以内。

详细情况参考项目的`scripts`脚本和`README.md`说明。

五. 样例解读

输入输出

输入总共有四个文件，分别是：

- `design.info`：

  ```
  FPGA1 10 0 2 0 2 0 2 4
  FPGA2 10 2 2 0 2 0 2 4
  ```

- `design.are`：

  ```
  g1 9 0 2 0 0 0 0 0
  g2 5 0 1 0 0 0 0 0
  g3 4 0 0 0 0 0 0 0
  g4 1 0 0 0 0 0 0 0
  ```

- `design.net`：

```
g4 9 g1
g4 10 g3
g2 3 g1 g4
```

- `design.topo`：

```
FPGA1 FPGA2
```

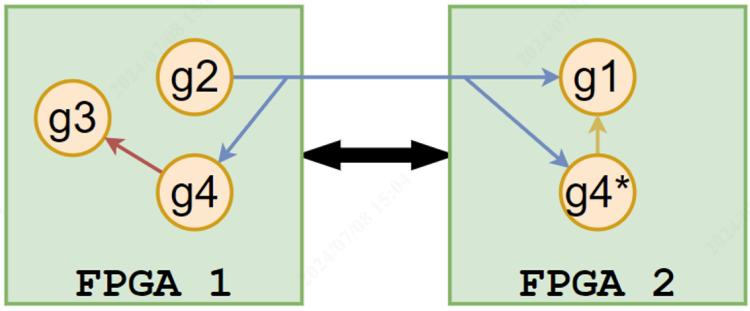
输出文件是：

- design.fpga.out：

```
FPGA1: g2 g3 g4
FPGA2: g1 g4*
```

输出方案解读

`design.fpga.out`的分割方案可视化如下图，三条超边用三种颜色。



- 约束条件：

  1. FPGA1 上资源总使用：10 0 1 0 0 0 0 0 FPGA2 上资源总使用：10 0 2 0 0 0 0 0 ，资源使用量均不超过可用资源。
  2. 每一个节点都出现在输出文件中。

  综上，满足了约束目标，该方案是合理的方案。

- 优化目标计算：

  只有一条超边被切割，该超边权重为3，跨越两个FPGA，所以总的被切割权重为`3*(2-1)=3`。

六. 补充材料

多级划分框架

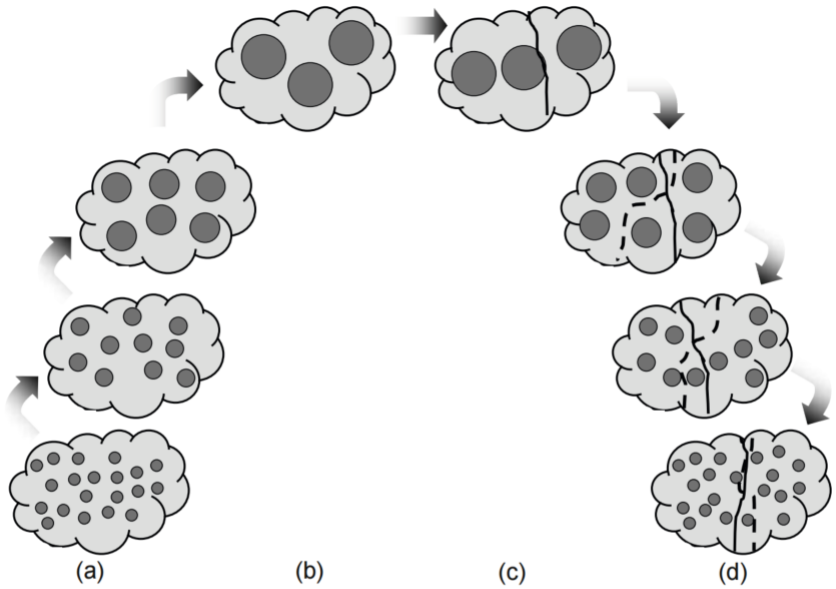


Fig. 2.6 An illustration of multilevel partitioning. (a) The original graph is first coarsened through several levels. (b) The graph after coarsening. (c) After coarsening, a heuristic partition is found of the most-coarsened graph. (d) That partition is then projected onto the next-coarsest graph (dotted line) and then refined (solid line). Projection and refinement continue until a partitioning solution for the original graph is found.

目前主流的超图分割算法大多采用多级划分框架，包含三个阶段：

1. **聚类/粗化**：核心思想是将超图中的节点和边通过一定的规则逐步“合并”或“缩小”，减少超图的规模，使得在更小的超图上进行划分时计算更加高效。粗化阶段主要考虑两个方面
    - **节点的匹配与合并策略**：一般使用启发式方法将具有高关联度的节点对合并成一个“超级节点”，重复进行合并步骤直到达到预设的粗化力度。
    - **聚类大小控制**：粗化力度过高，容易导致后续细化困难，粗化力度不够，容易导致初始划分算法耗时长效果差。
2. **初始划分**：在较小规模的粗化图上快速生成一个初始分割解。主要有**递归二分法**和**直接k-way法。**
3. **细化**：逐层还原超节点，使它们恢复为原始的多个节点。在每个还原步骤中，使用优化算法（**KL或FM算法**）对划分结果进行细化调整，以提高划分质量。

更详细内容，可以参考VLSI physical design: from graph Partitioning to timing closure一书的第二章。

开源项目

下面罗列可供参考的开源超图分割器：

1. metis 超图分割器：早期的超图分割器。
2. par-metis超图分割器：metis分割器的改良版，引入并行性。
3. kahypar 超图分割器：提供用户友好的自定义目标函数接口。
4. mt-kahypar 超图分割器：利用TBB多线程库重构的kahypar分割器，性能更优。
5. TritonPart 超图分割器：OpenRoad高级综合项目的子项目。相比kahypar，提供更多的约束条件，同时考虑时序优化。

参考资料

1. metis 论文
2. kahypar 论文
3. Triton Part 论文

4. VLSI physical design: from graph Partitioning to timing closure