电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2023091602012

姓 名 邓奕雷

（实验） 课程名称 EDA软件设计II

理论教师 孙静翎

实验教师 易黎

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：邓奕雷 学号：2023091602012 指导教师：易黎**

**实验地点：信软楼西303 实验时间：2025.05.08**

**一、实验名称：实验二 Hu调度算法实验**

**二、实验学时：4**

**三、实验目的：**

1. 深入理解 Hu 调度算法的原理和实现方法，掌握基于最长路径标签的资源受限调度策略。
2. 掌握对 BLIF 文件的解析方法，构建电路图结构并提取节点依赖关系。
3. 实现支持资源限制的调度算法，验证其对不同电路描述文件的正确性和通用性。
4. 提升对资源受限调度问题的分析能力，加深对 EDA 工具核心算法的理解。

**四、实验原理：**

### **Hu 调度算法核心思想**

Hu 调度算法适用于所有操作使用同一种资源的电路，核心是通过节点到终止节点的最长路径长度（标签）决定调度顺序。标签越大的节点优先调度，每次调度节点数不超过资源限制a。

### **算法步骤**

****标签计算****：从终止节点（输出节点）反向计算每个节点的标签，标签值为该节 点到终止节点的最长路径长度（路径上的节点数）。

****调度过程****：

初始化步骤l=1，收集所有前驱节点已调度或无前驱的 “就绪节点”。

从就绪节点中选择标签最大的≤a个节点，分配其调度步骤为l。

步骤l递增，重复直至所有节点调度完成。

### **数学描述**

|  |
| --- |
| Hu(G(V,E), a):  1. 计算所有节点标签label（最长路径长度）  2. l = 1  3. 循环：  U = {未调度且前驱全调度的节点}  S = U中label最大的≤a个节点  调度S中节点，设置其调度步骤为l  l += 1  直到所有节点调度完成 |

### **BLIF 文件解析**

与实验一类似，解析.inputs、.outputs、.names关键字，构建节点前驱 / 后继关系，并根据门类型（如与门、或门）确定执行周期（选做部分）。

**五、实验内容：**

### **基础要求**

假设电路中某种运算需要消耗的周期数为1，编程实现针对这种运算的Hu调度算法。要求程序对三个不同的blif文件进行解析，都能得到正确的输出。

资源限制参数a可配置（如 PPT 中示例a=3）。

### **选做要求**

假设有以下周期数消耗约定，编程实现针对其中一种运算的Hu调度算法。要求程序对三个不同的blif文件进行解析，都能得到正确的输出。



**六、实验器材（设备、元器件）：**

1. ****开发环境****：Visual Studio 2022
2. ****编程语言****：C++
3. ****测试文件****：三个不同结构的 BLIF 文件（如sample1.blif、sample2.blif、sample3.blif）

**七、实验步骤：**

根据实验原理，使用 C++ 语言实现 ASAP 和 ALAP 调度算法。代码主要包括以下几个部分：

* 1. **节点和图的定义：**

|  |
| --- |
| class Node {  public:  // 后继节点  vector<Node\*> next;  // 前驱节点  vector<Node\*> pre;  // 节点类型  Type type;  // 节点名称  string name;  // 调度开始周期，-1表示未调度  int startCycle;  // 调度执行时间  int executionTime;  // alap的节点开始时间（节点编号）  int alapStartCycle;  // 节点调度结束时间  int getEndCycle() {  return startCycle + executionTime;  }  Node(string nodeName, int execTime = 1) : name(nodeName), type(DEFAULT), executionTime(execTime) {  startCycle = -1;  }  Node(int execTime = 1) : type(DEFAULT), executionTime(execTime) {  startCycle = -1;  }  }; |
| class Graph {  public:  // 节点集  map<string, Node\*> nodes;  // 图的大小  int size;  // 清空调度状态  void clear() {  for (auto& pair : nodes) {  pair.second->startCycle = -1;  }  }  // 初始化图  Graph() : size(0) {  size = 0;  }  }; |

* 1. **BLIF 文件解析：**

同实验一

* 1. **Hu算法实现：**

3.1 Hu算法

|  |
| --- |
| int Hu(Graph& g, int resource) {  // 每次循环是一个周期  int currentCycle = 0;  // 已调度节点的数量  int processCount = 0;  while (processCount < g.size) {  // 当前周期的可以调度节点集  vector<Node\*> unprocessedNodes = getUnprocessedNodes(g, currentCycle);  if (unprocessedNodes.empty()) {  currentCycle++;  continue;  }  // 获取当前周期正在调度的节点数  int processingCounts = getCurrentCycleProcessingCounts(g, currentCycle);  // 确定调度节点集  vector<Node\*> processingNodes = getProcessingNodes(unprocessedNodes, resource - processingCounts);  if (processingNodes.empty()) {  currentCycle++;  continue;  }  // 调度当前周期的节点  if (!processingNodes.empty()) {  for (Node\* node : processingNodes) {  node->startCycle = currentCycle;  processCount++;  }  }  currentCycle++;  }  return currentCycle - 1;  } |

3.2 确定节点集U：

|  |
| --- |
| vector<Node\*> getUnprocessedNodes(Graph& g, int currentCycle) {  // 可调度节点集U  vector<Node\*> unprocessedNodes;  for (auto& pair : g.nodes) {  Node\* currentNode = pair.second;  // 跳过已调度节点  if (currentNode->startCycle != -1) {  continue;  }  // 表示前驱节点是否已全部调度过  bool preProcessFinished = true;  int maxPreEndCycle = 0;  // 检查非输入节点的前驱  if (!currentNode->pre.empty()) {  for (Node\* n : currentNode->pre) {  // 存在前驱节点未调度  if (n->startCycle == -1) {  preProcessFinished = false;  break;  }  // 更新最大前驱结束周期  maxPreEndCycle = max(maxPreEndCycle, n->getEndCycle());  }  }  // 如果有前驱节点未调度，或当前周期小于最大前驱结束周期的节点  if (!preProcessFinished || currentCycle < maxPreEndCycle) {  continue;  }  unprocessedNodes.push\_back(currentNode);  }  return unprocessedNodes;  } |

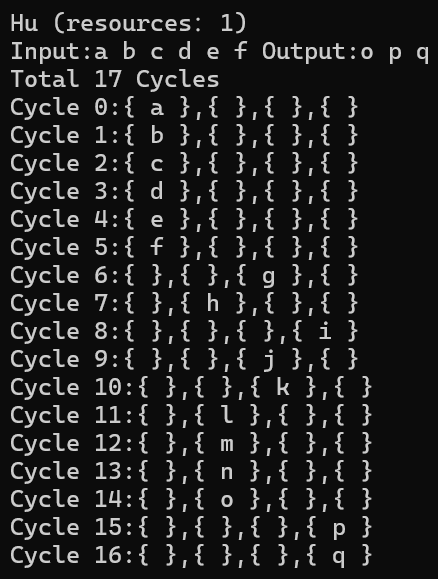
3.3 确定节点集S：

|  |
| --- |
| vector<Node\*> getProcessingNodes(vector<Node\*> unprocessedNode, int resource) {  // 最早调度节点集S  vector<Node\*> processingNodes;  while (!unprocessedNode.empty() && processingNodes.size() < resource) {  // 获取可调度节点集里的最早调度节点  Node\* minNode = unprocessedNode[0];  for (Node\* node : unprocessedNode) {  if (node->alapStartCycle < minNode->alapStartCycle) {  minNode = node;  }  }  processingNodes.push\_back(minNode);  // 使用标准库算法安全移除minNode  auto it = std::remove(unprocessedNode.begin(), unprocessedNode.end(), minNode);  unprocessedNode.erase(it, unprocessedNode.end());  }  return processingNodes;  } |

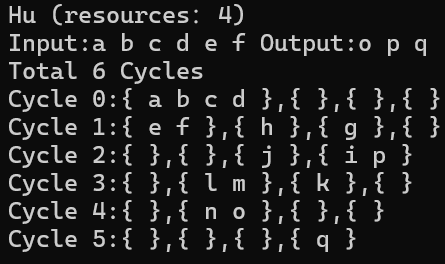
**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

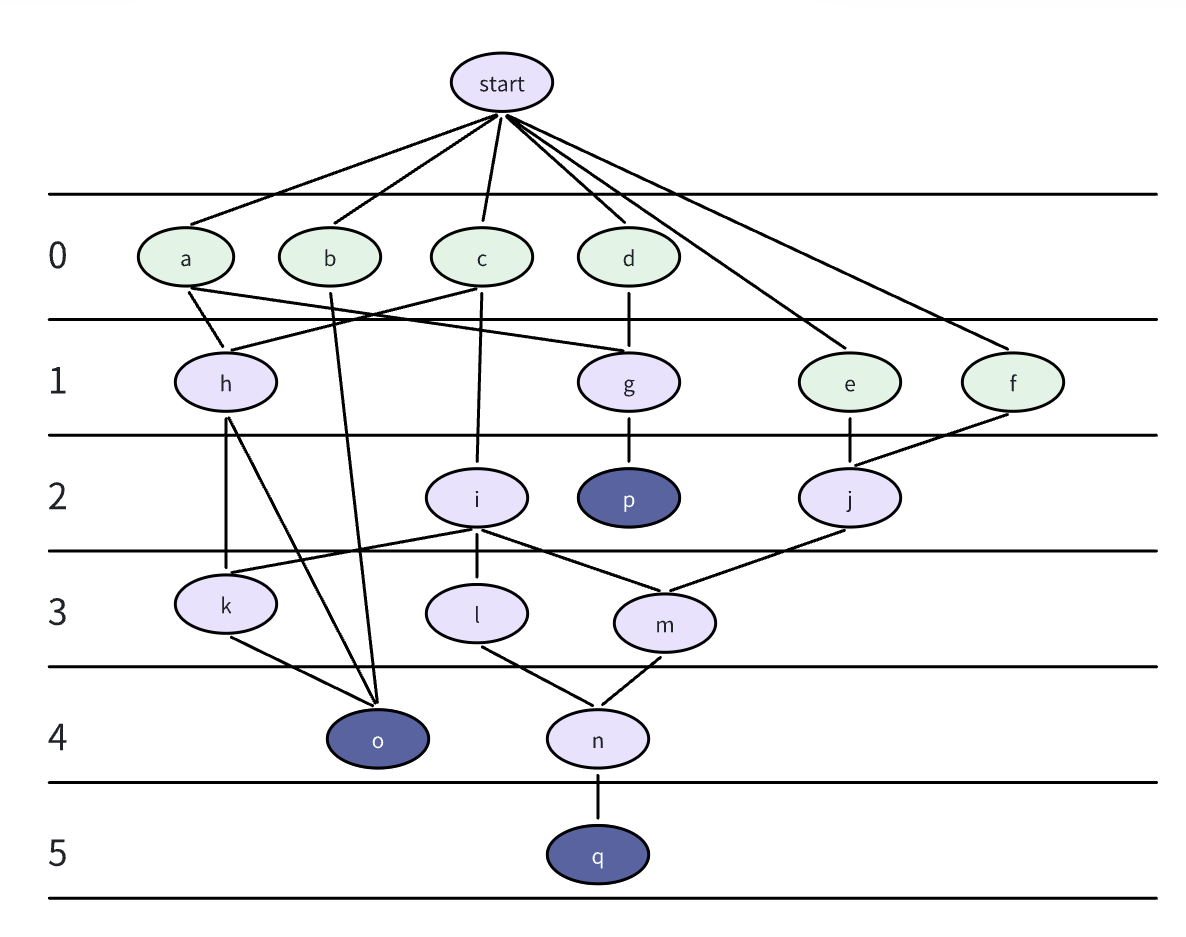
* 1. **结果输出：**
     1. 对于周期消耗为1的执行结果

1. 资源数为1：

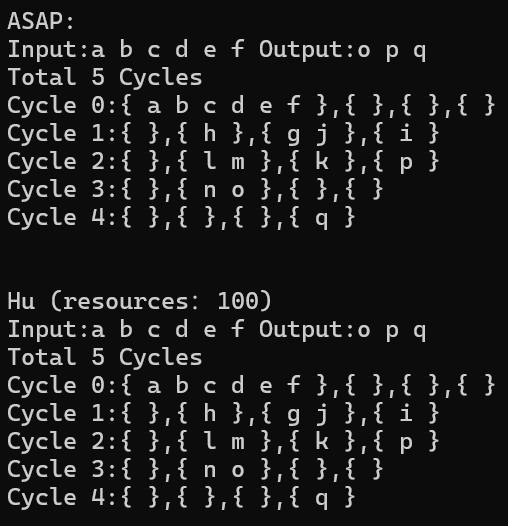


② 资源数为4：





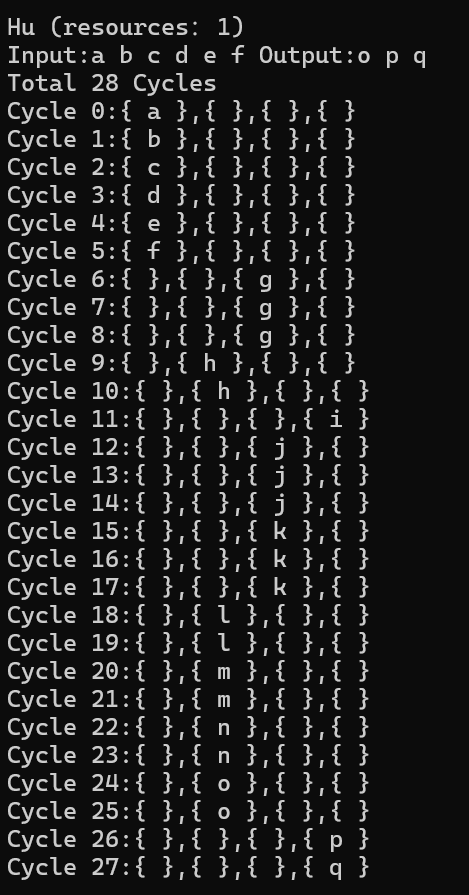
③ 资源数无限制时，退化为ASAP算法



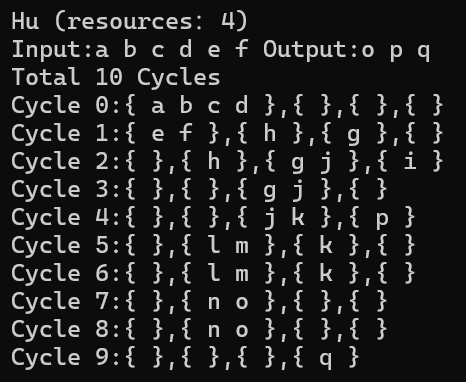
* + 1. **选做题：对于不同周期数消耗的执行结果**

以AND：2，OR：3，NOT：1为例

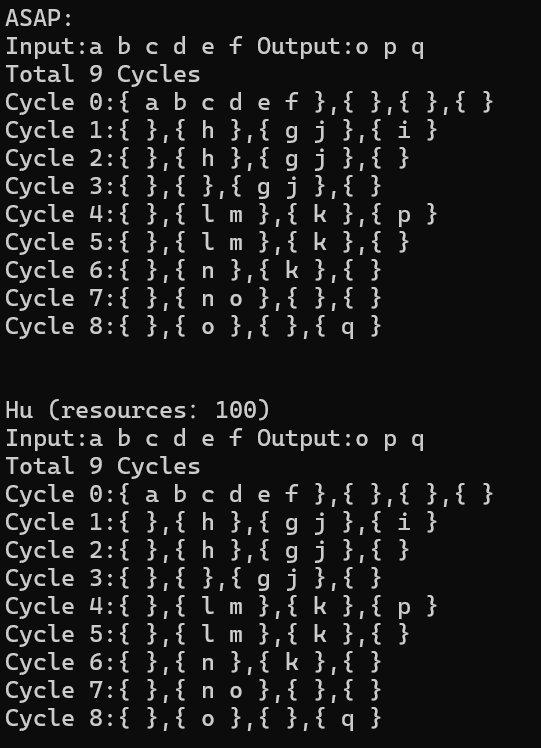
1. 资源数为1：



② 资源数为4：



③ 资源数无限制时，退化为ASAP算法



**九、总结及心得体会：**

### **1. 实验收获**

* 掌握了资源受限调度的核心思想，Hu 算法通过标签优先策略有效平衡了调度顺序和资源利用。
* 理解了 BLIF 文件解析的扩展性，通过调整节点执行时间和标签定义，可支持不同运算周期的场景。
* 提升了图论算法在 EDA 中的应用能力，如反向遍历计算最长路径，优先队列优化调度选择。

### **2. 遇到的问题及解决**

* ****多终止节点标签计算****：初始未考虑多输出情况，导致标签错误。通过收集所有输出节点并计算各自最长路径，取最大值解决。
* ****就绪节点重复筛选****：早期实现中未缓存就绪状态，导致效率低下。优化为每次仅检查未调度节点的前驱状态，提升性能。

### **3. 改进方向**

* ****性能优化****：使用优先队列存储就绪节点，避免每次排序，减少时间复杂度。
* ****错误处理****：增加 BLIF 文件格式校验，如环路检测（Hu 算法假设无环电路），提升鲁棒性。
* ****可视化展示****：参考实验一结果输出格式，增加调度步骤的图形化展示，便于调试和验证。

1. **对本实验过程及方法、手段的改进建议：**
   1. 增强鲁棒性：

添加环路检测，避免无限循环。

基于真值表内容，精确判断门类型。

* 1. 扩展功能：

支持多周期门延时，提升算法实用性。

增加图形化展示，直观呈现调度结果。

**报告评分：**

**指导教师签字：**