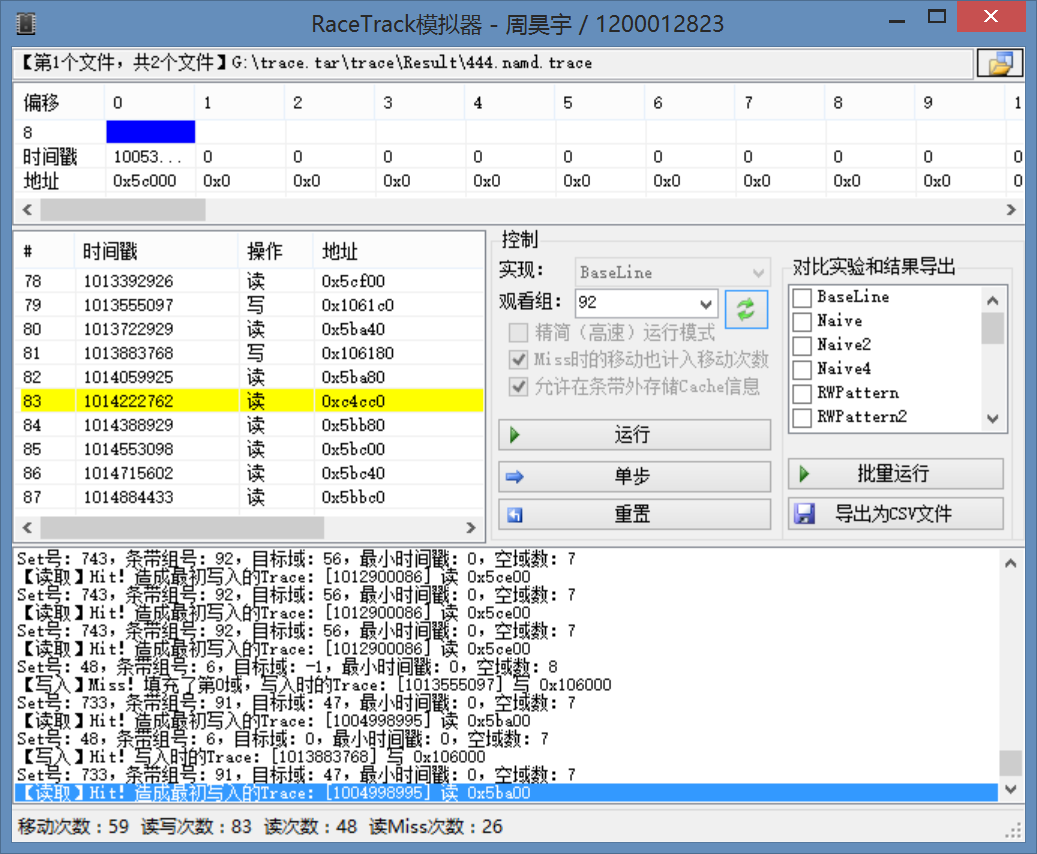
**计算机组织与体系结构Project3**

RaceTrack模拟器与实现策略探究报告

**周昊宇 / 1200012823**（单人组）

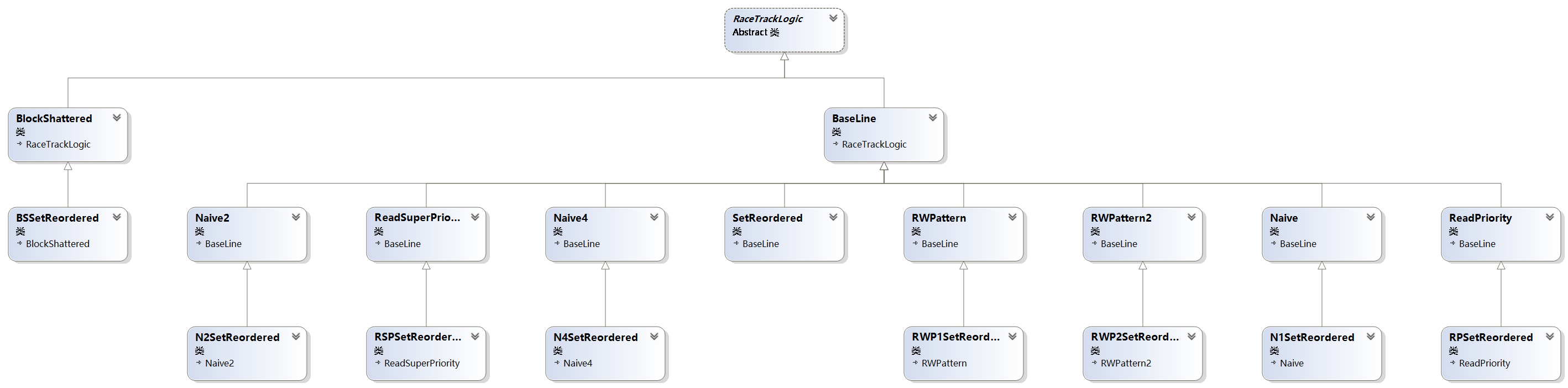
# 概要

本次Project我独立实现了一个能读取多个Trace并模拟RaceTrack行为的模拟器，采用C#编写并具有友好的图形界面，并且支持批量自动模拟、对比分析并导出结果。（因此建议使用Visual Studio 2012及以上版本打开源代码）



**【界面截图】**

除此之外，我还实现了BaseLine，以及在其基础上通过修改读写端口布局、Set布局、Set编址等因子而组合派生出的共18种实现策略。在经过大量实验模拟后，我得出了一种相对效率较高的策略，其效率比起BaseLine有了较大改善（请见“实验结论”）。



**【各种RaceTrack实现的类图】**

**类图：ClassDiagram1.cd**

# 各种实现的说明

（具体端口位置可以参看RaceTrackLogic.cs的690行以后）（代码具有完全注释，可以放心观看）

## BaseLine

按照Project3的要求加以实现。注意这是在所给要求下所实现的性能最佳的版本，所有读写端口的移动方式都是就近原则。

## Naive

将BaseLine的4个读写端口增加为6个读写端口。

## Naive2

将Naive的端口平移2单位。

## Naive4

将Naive的端口平移4单位。

## RWPattern

采用单独的读端口和写端口，按照一读一写分布。

## RWPattern2

将写端口的方向调整为反方向，使得读写端口的探头在一起。

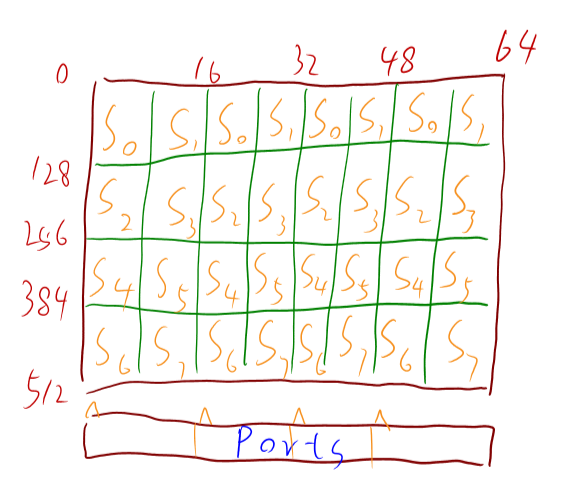
## ReadPriority

均匀放置四个写端口，其他位置全是读端口。

## ReadSuperPriority

在正中央放置一个写端口，其他位置全是读端口。

## SetReordered

对Set进行重新编址，使得第一个条带组放置第0、128、256、……号Set。

## BlockShattered

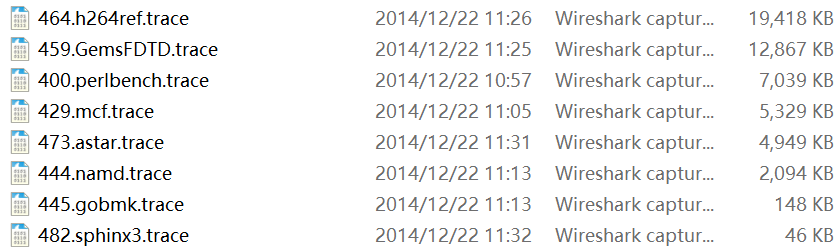
对Set进行重新划分如右图，即将原来的一个Block拆分成四部分，放在相差16的四个位置。读写时，四个探头同时读写对应的128个条带上的4个域。

## XX\_SetReordered

将名为XX的实现进行Set重新编址，如SetReordered一般。共9个。

# 实验过程和结果

## 实验过程

限于电脑性能，我选用了8个大小最小的Trace进行实验并综合结果。

操作方法可参考README.md。

## 实验结果

最后得出的实验结果已经附在报告文件夹下，下方是图表副本（越低越好）。所有数值都表示Shift次数，完整的实验结果中还有Miss次数和读写次数，不过由于所有实现方式的结果都一样，所以就省略了。

## 结果分析

1. 因为读操作多就大量放置读端口的策略是不合适的。
   1. 结论来源：ReadPriority系列的表现都很糟糕。
   2. 现象分析：原因可能是因为造成大量Shift的本来就是写操作，写操作是瓶颈。
2. 直接增加读写端口的密度十分有效。
   1. 结论来源：Naive系列的表现都很理想。
   2. 现象分析：显然……
3. 直接改变Set的编址十分有效。
   1. 结论来源：SetReordered系列的表现都比原来有很大提升。
   2. 现象分析：由于相邻的Set可能会由于局部性频繁访问，因此要避免相邻Set处在同一个组中以免不断Shift。
4. 改变Set的Block划分，将一个Block打散成四块存储，这种策略也是不合适的。
   1. 结论来源：BlockShattered系列的表现都很糟糕。
   2. 现象分析：可能由于条带无法针对任一探头进行就近Shift（每个区域的条带必须要让对应的探头读写，否则无法读写完全整个Block），所以这种策略束缚了Shift的灵活性。
5. 将探头分离成一读一写虽然有性能提升，但是不如Naive那种策略理想。
   1. 结论来源：RWPattern系列表现比BaseLine好，但是不如Naive系列。
   2. 现象分析：可能是因为如果发生Miss后再读，读写探头不在同一位置会导致多余的Shift。

# 实验结论

## 最佳策略

从实验Shift的平均值来看，Naive4\_SetReordered是最佳的RaceTrack Cache实现策略，也就是使用6个读写端口，并且将探头放置在4、16、28、40、52、64的位置，然后对Set进行重新编址，使得第一个条带组有原第0、128、256、……号Set。

**性能提升：56.16%**

## 经验收获

控制变量法确定局部最优，然后在最优的策略上再进行改进是一种探究策略的好方法。

程序越早重构越好……

重构时你不会后悔选择了一个强类型语言。