# Lab3

## 实验目的

1. 实现一个组相连的cache
2. 用新的cache替换原有cache，使用快速排序和矩阵乘法

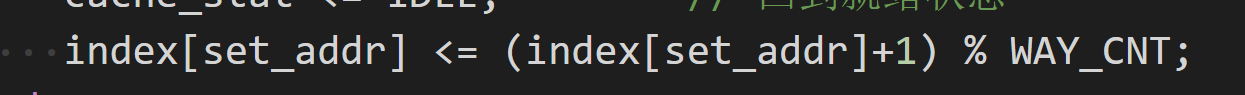
## 实验环境与工具

Windows环境下使用vivado仿真和综合工具

## 实验内容与过程

阶段一：实现LRU策略与FIFO策略的组相连

开始时，按照WAY\_CNT的顺序写入。FIFO只需记录每一组的第一路，以后每次都替换下一路，替换的路为下图所示

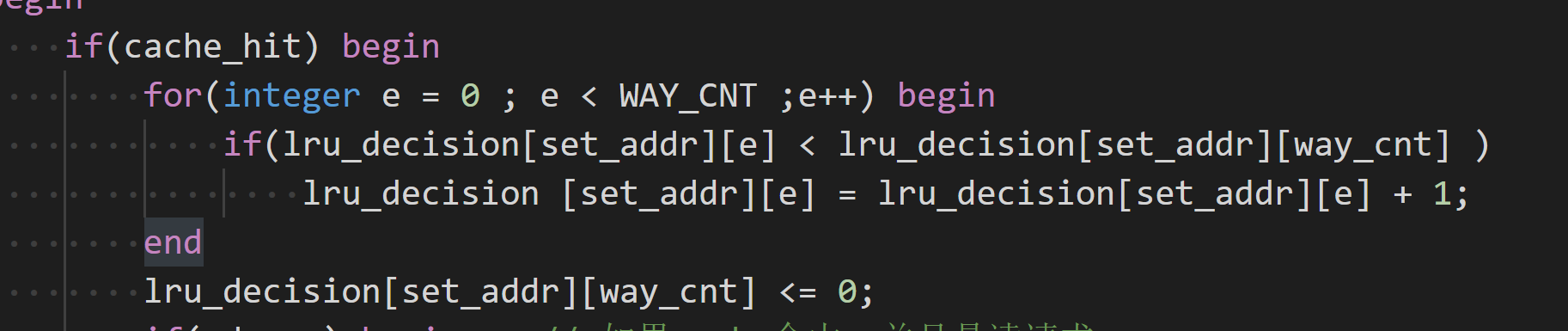


LRU需要用一个数组记录一组中每一路的距离上次使用的时间间隔，0为最近使用，WAY\_CNT-1表示最久未使用。每次命中就把对应路的使用间隔变为0，并把使用间隔低于其的路使用间隔都变大1时。每次替换使用间隔最大的路

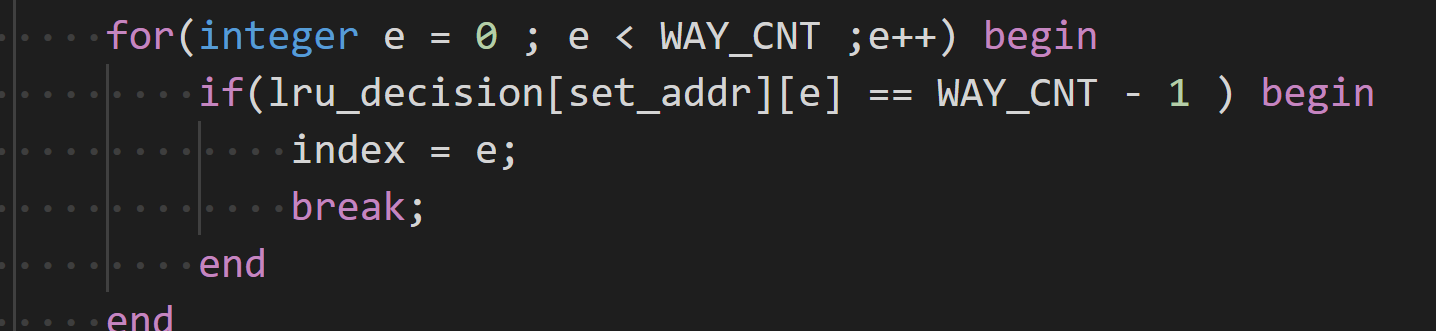
代码如下

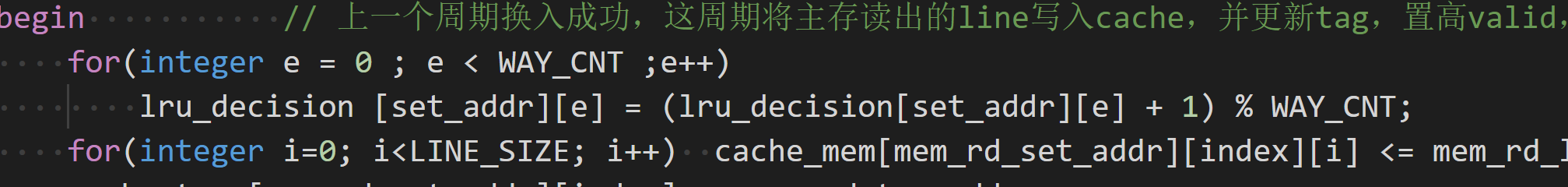
way\_cnt为命中的路

Cache命中后，使用间隔低于命中路的间隔加一，命中路间隔为0

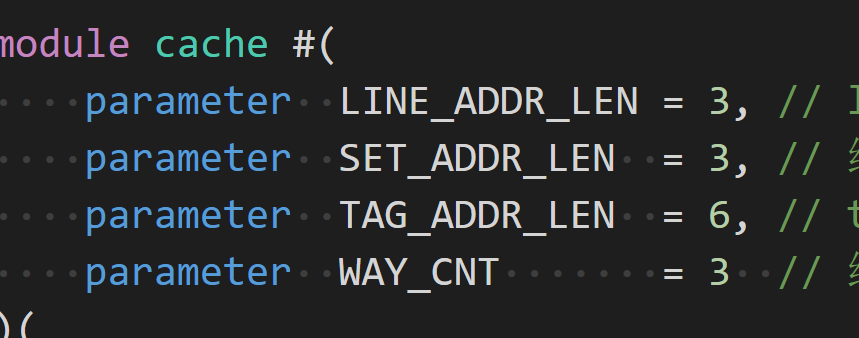


Cache miss后，找到使用间隔最大的路，记为index用于换入换出。最后把所有的使用间隔都加1，由于取余，原本间隔最大的正好变为0。

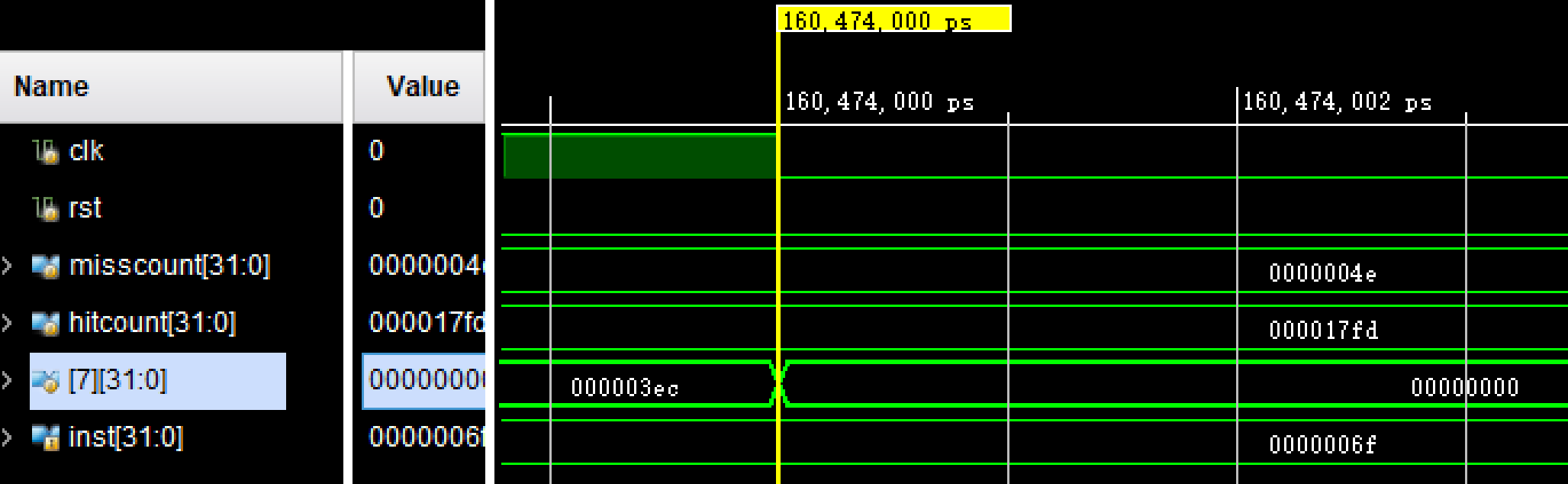




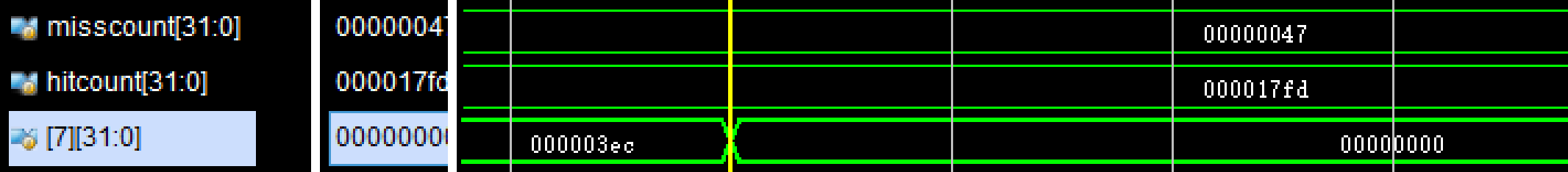
在默认设定中



FIFO快排缺失率为



LRU快排缺失率为



可以看到两种策略的缺失率几乎没有差别。

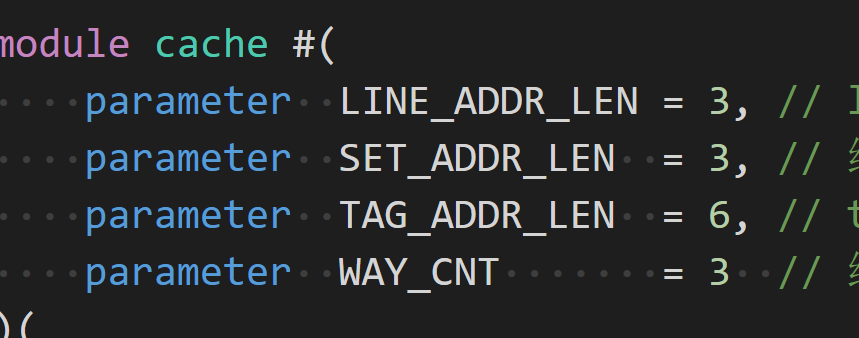
由于原本的数据中cache里有8\*8\*3=192个word，而快速排序中只有255个元素，所以在默认数据时，FIFO与LRU的差距并不明显。

所以在下面改变参数时我们使用矩阵乘法的结果进行比较

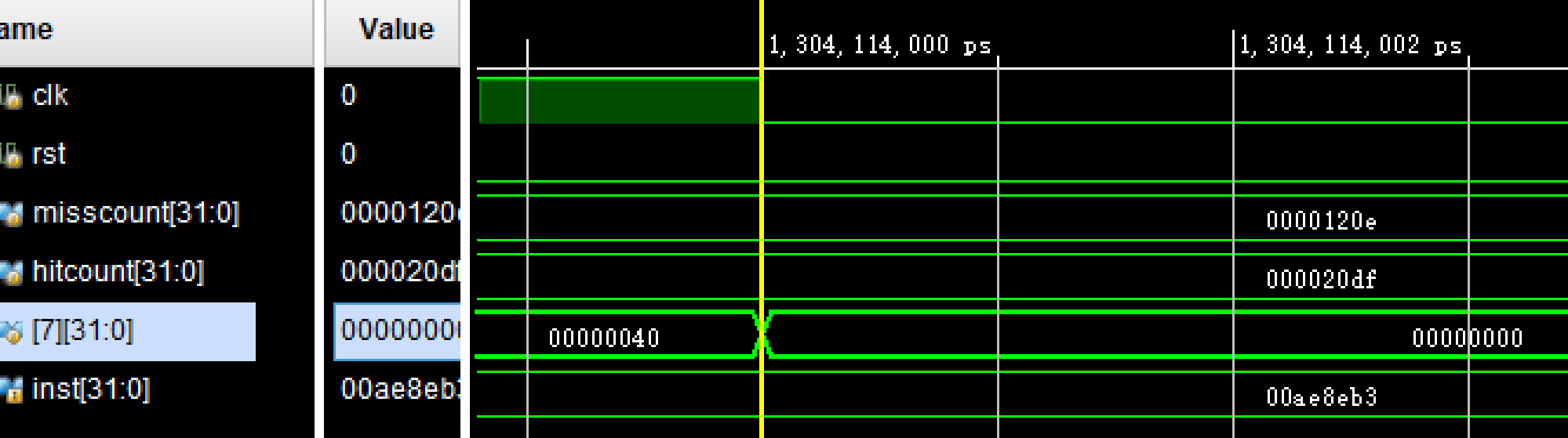
开始执行指令的时间都是12000ps

使用测试程序为默认的矩阵乘法

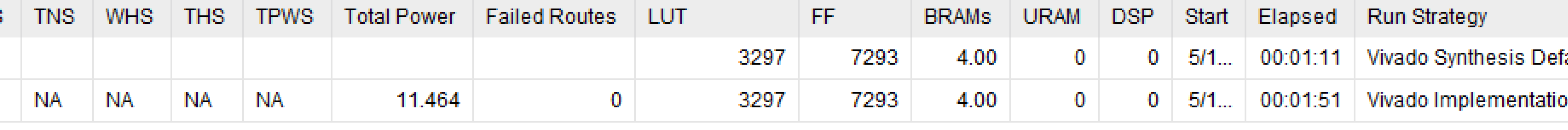
如下设置



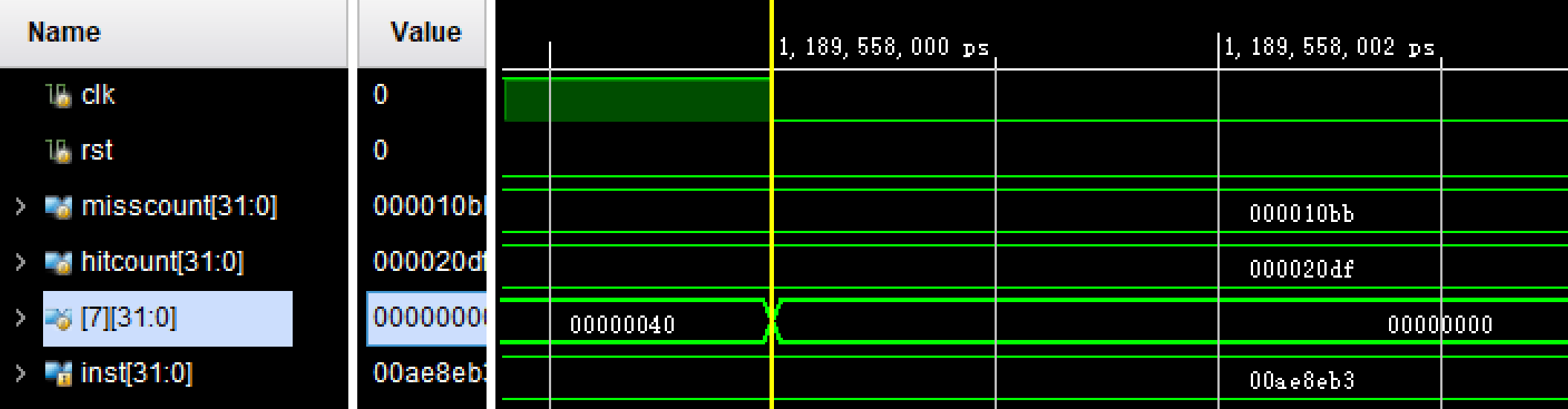
FIFO缺失率如下图

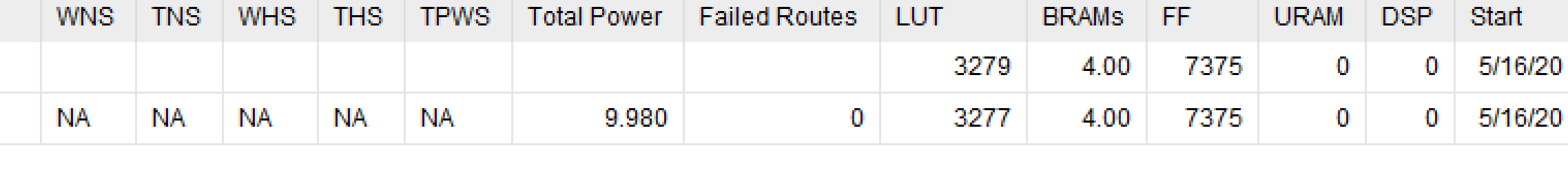


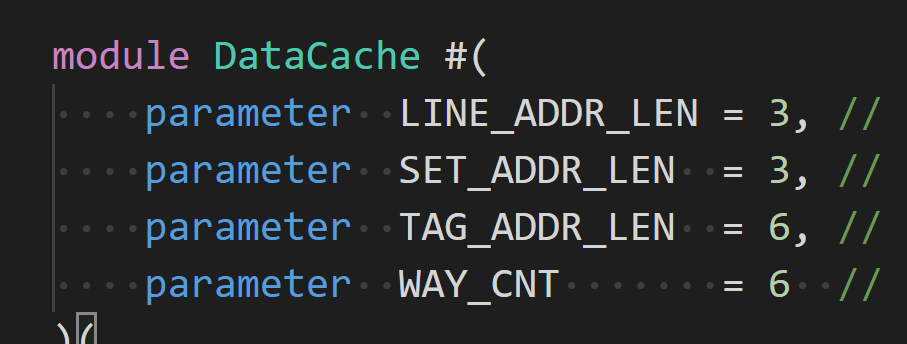
综合结果如下图



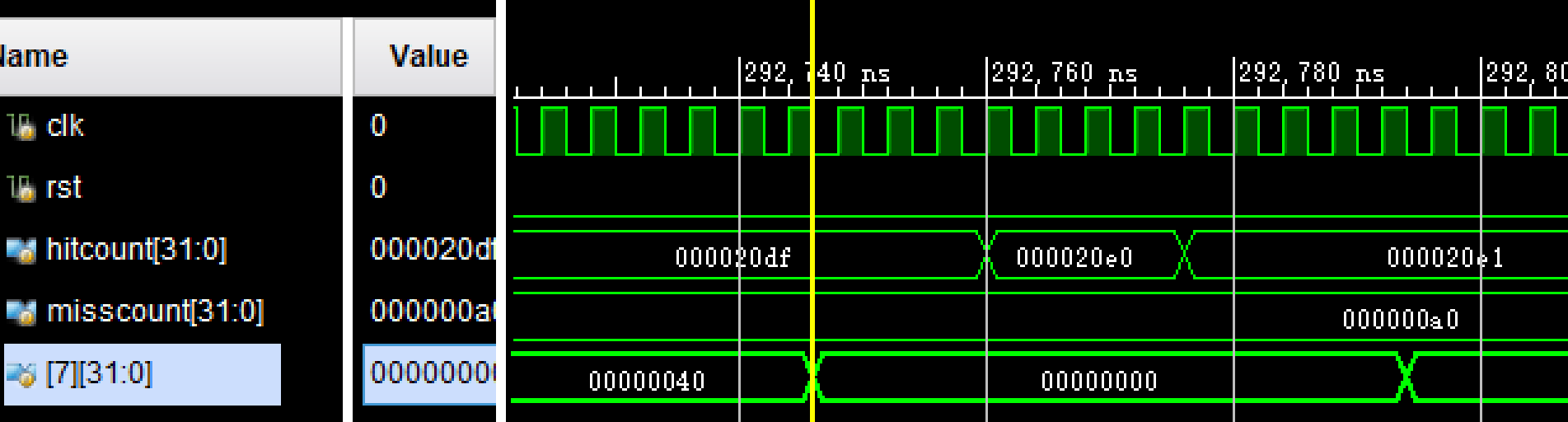
LRU缺失率如下

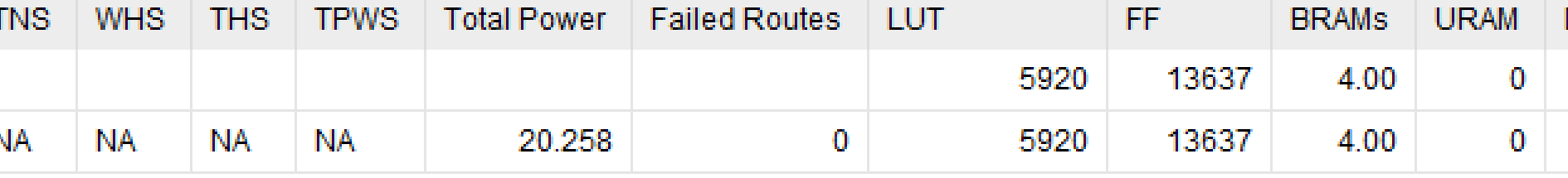




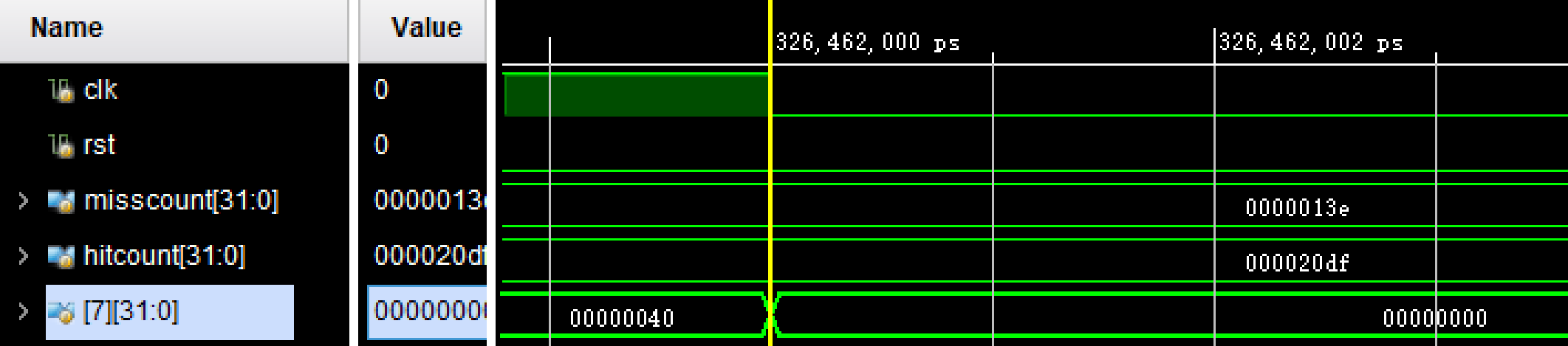


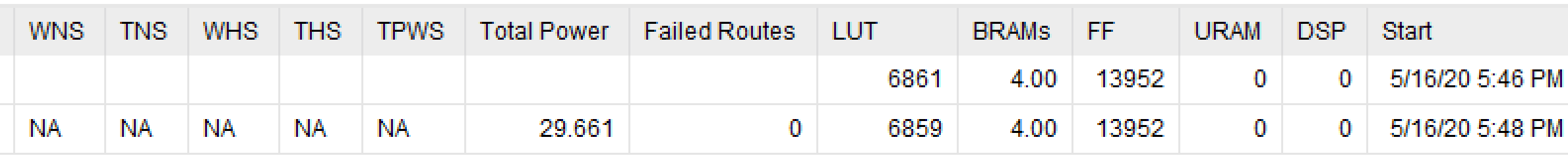
FIFO：

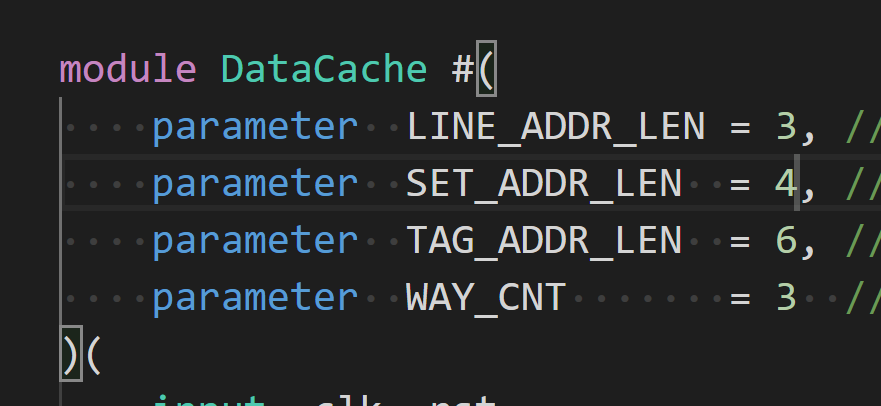




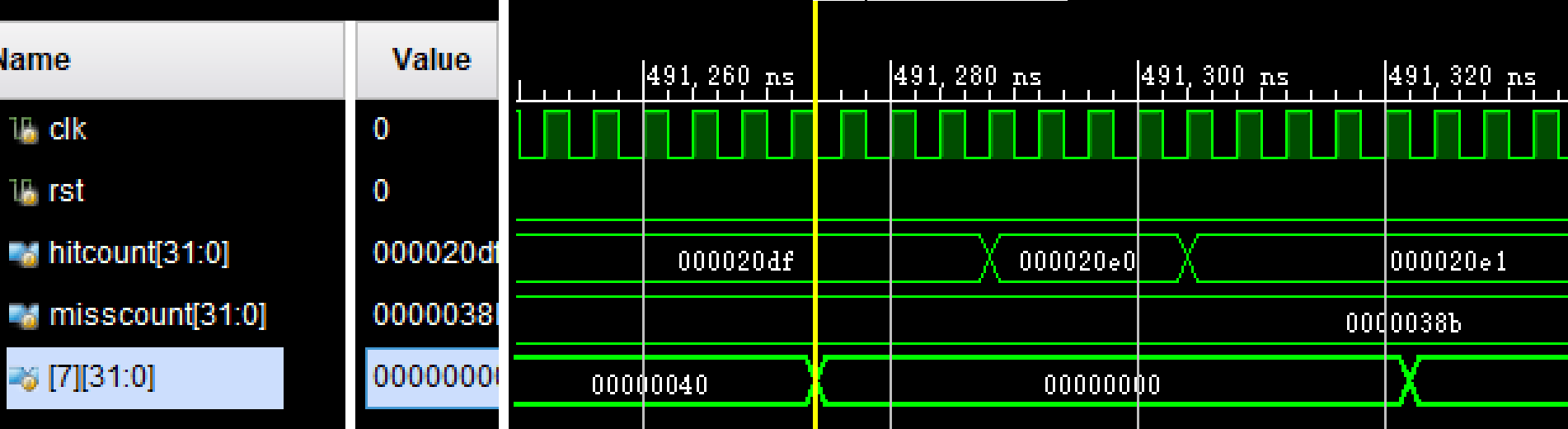
LRU：

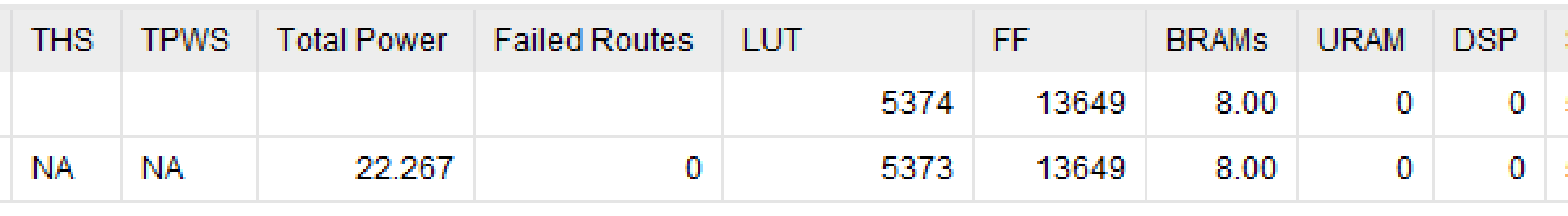




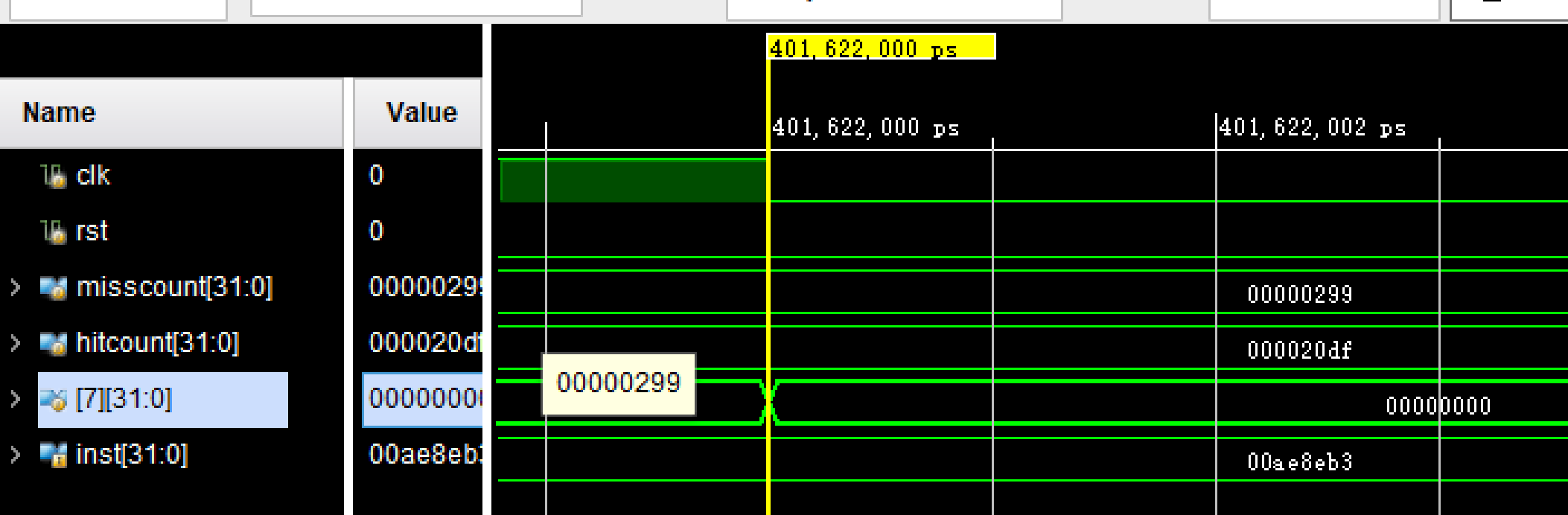


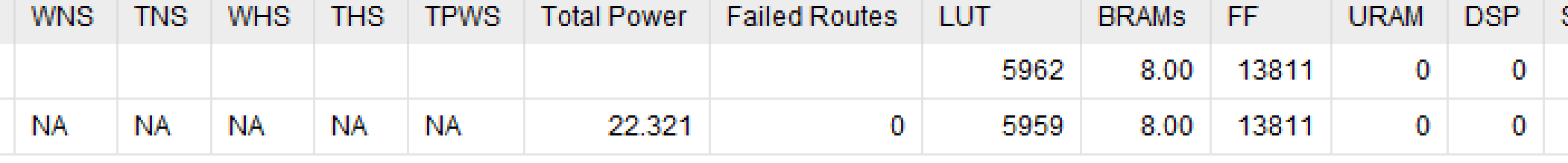
FIFO：

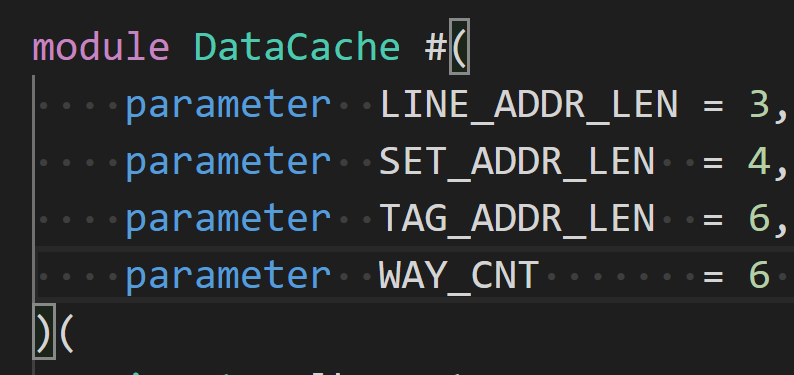




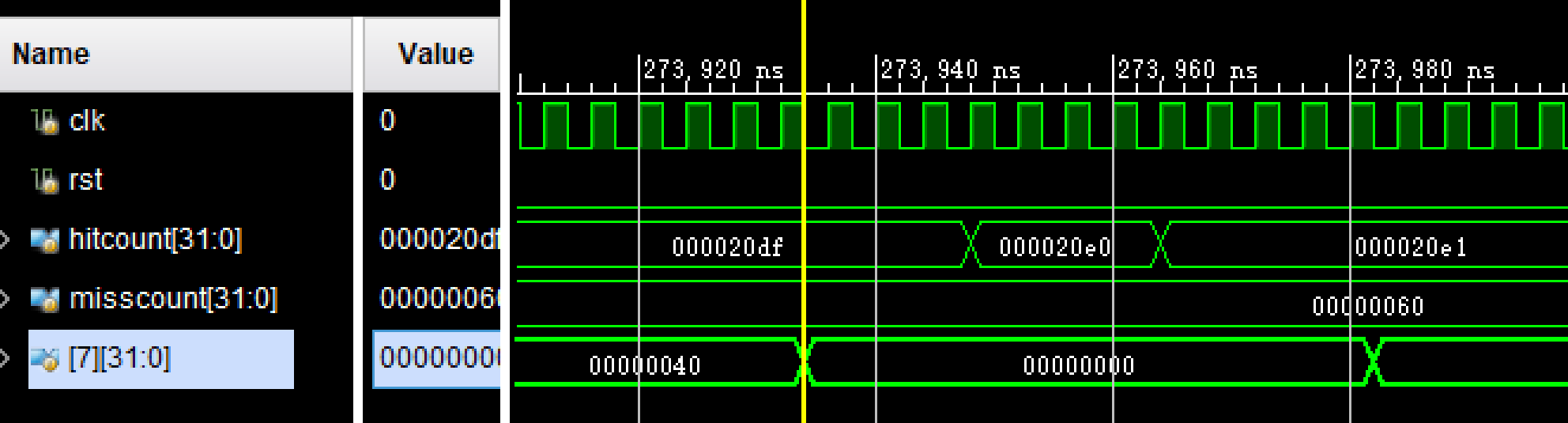
LRU:

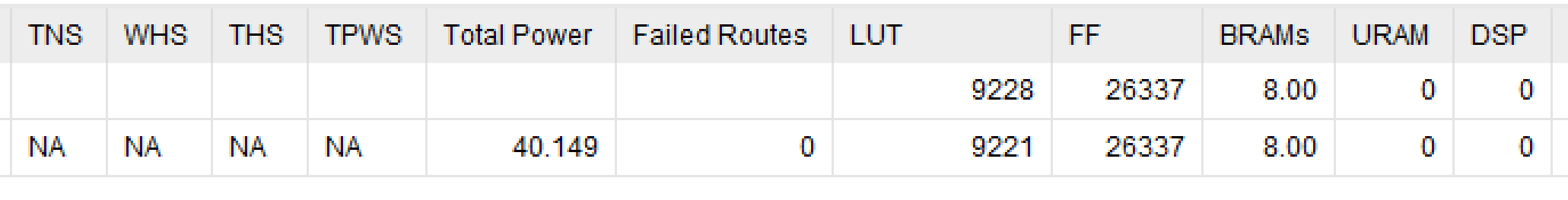




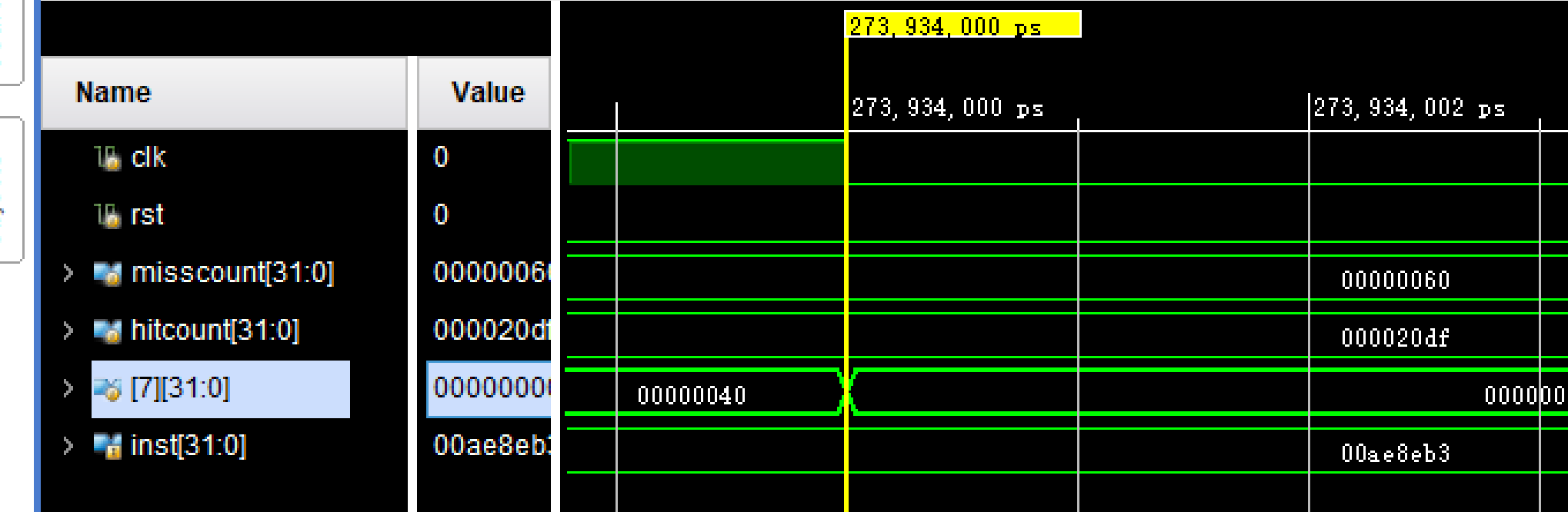


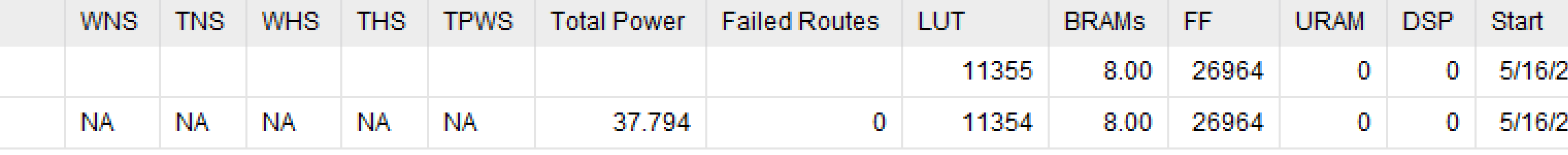
FIFO：

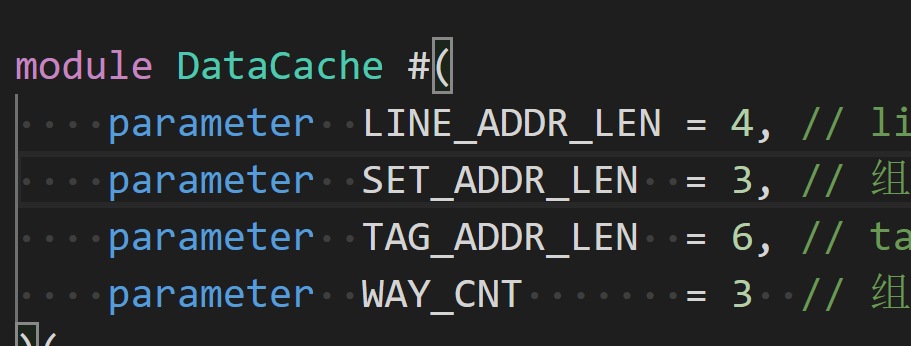




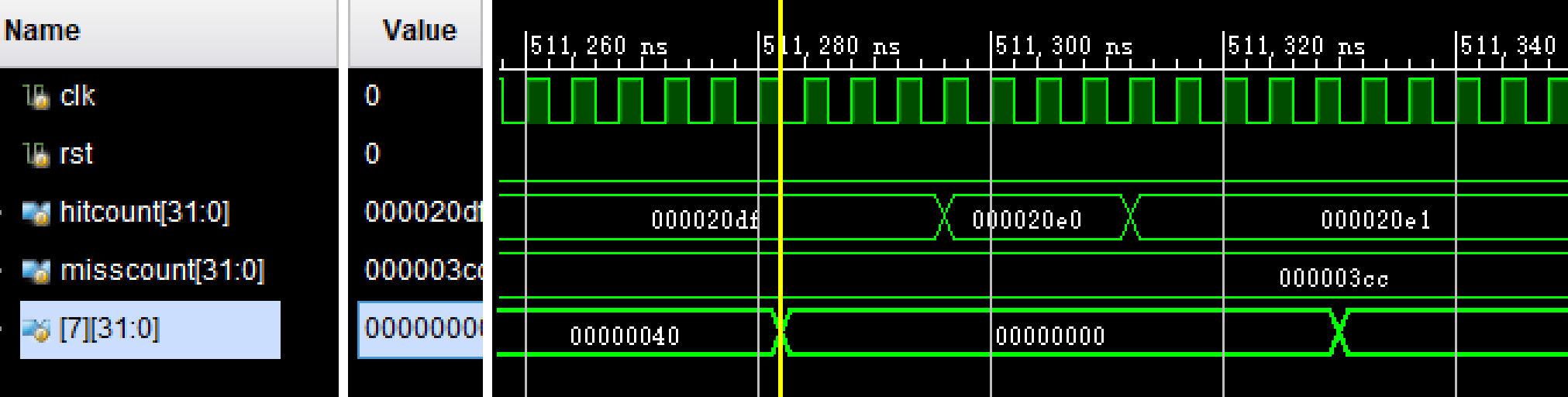
LRU：

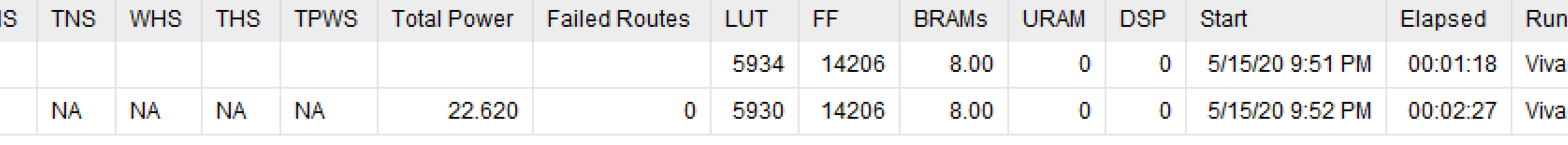




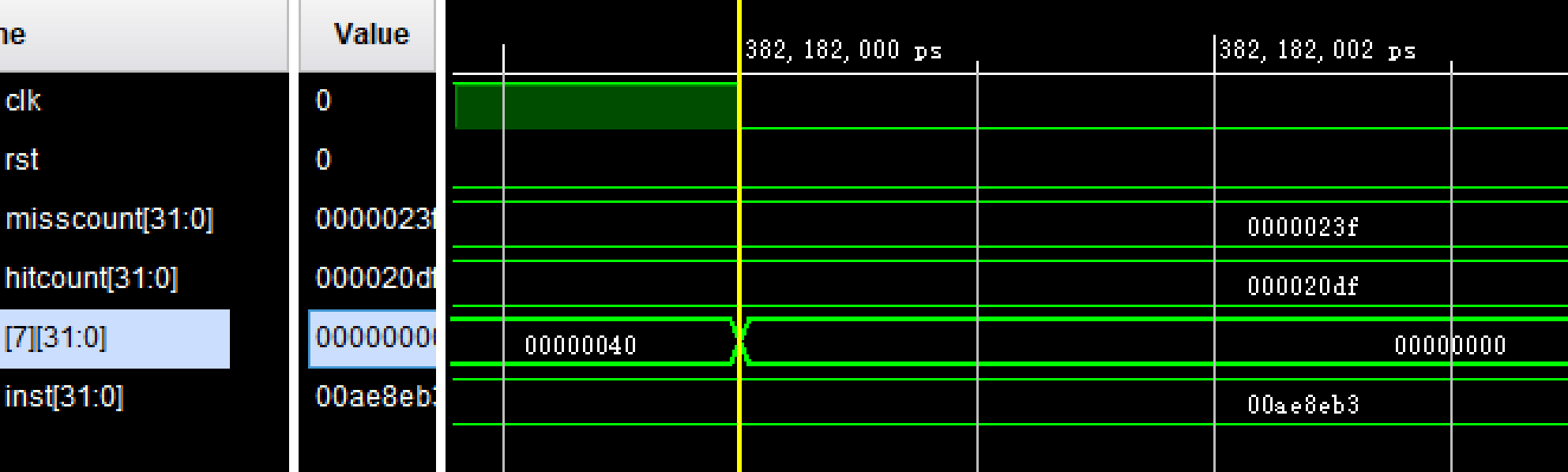


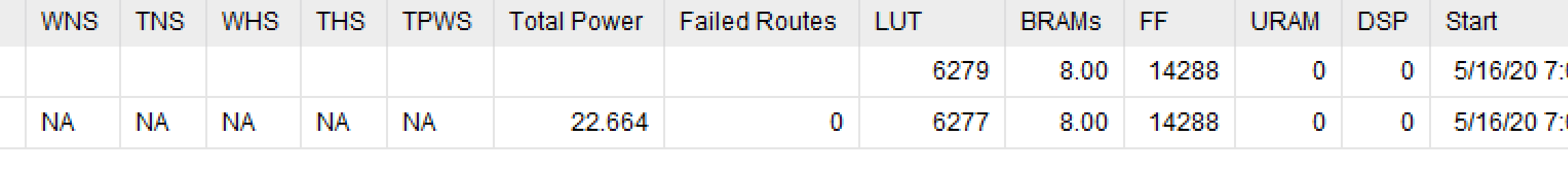
FIFO:

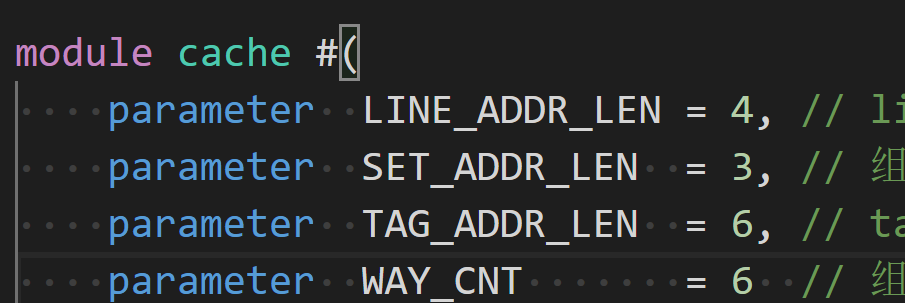




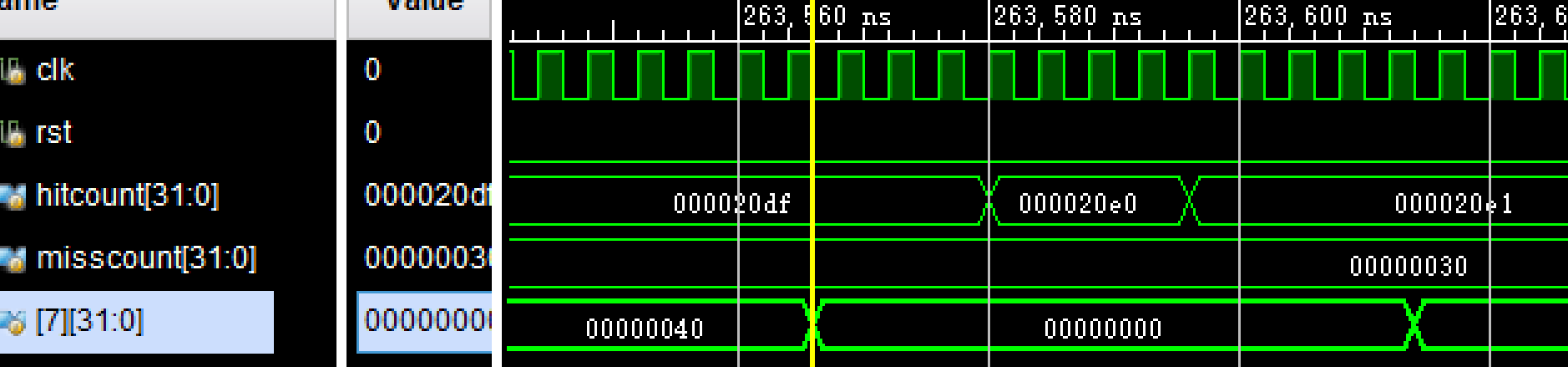
LRU:

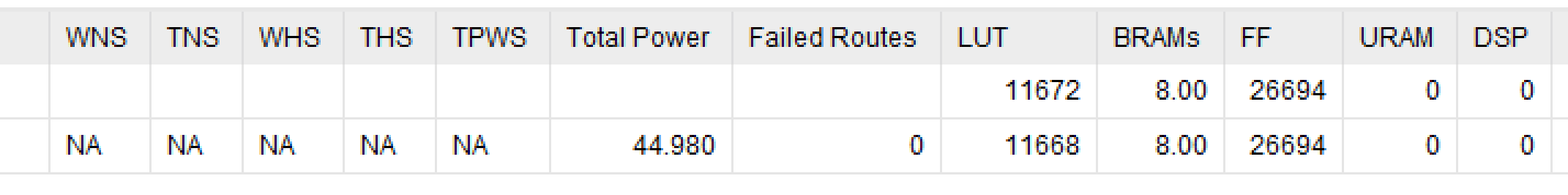




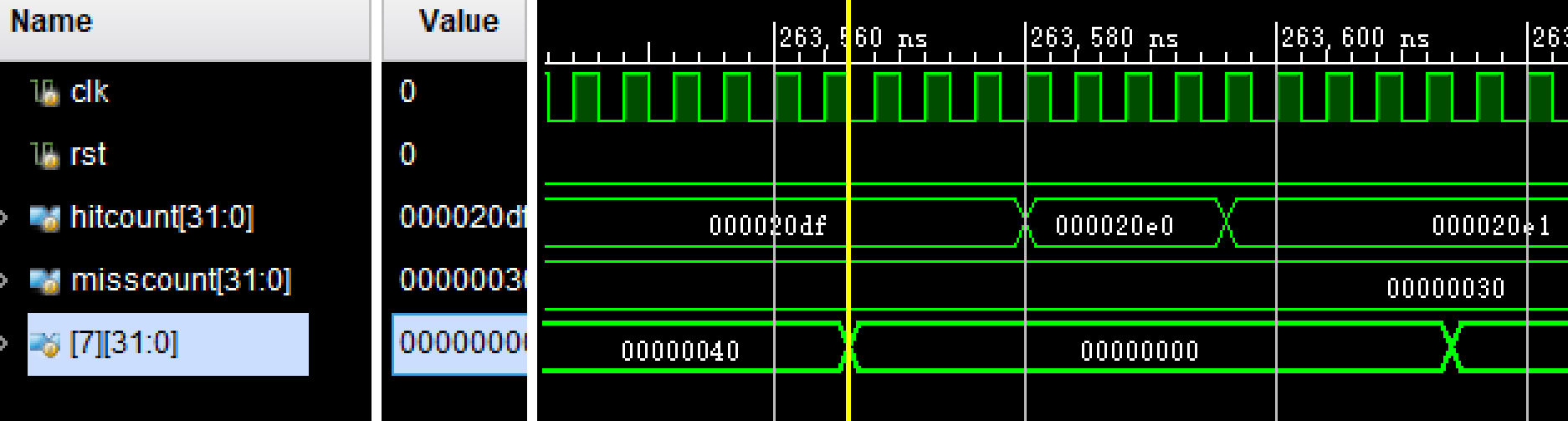


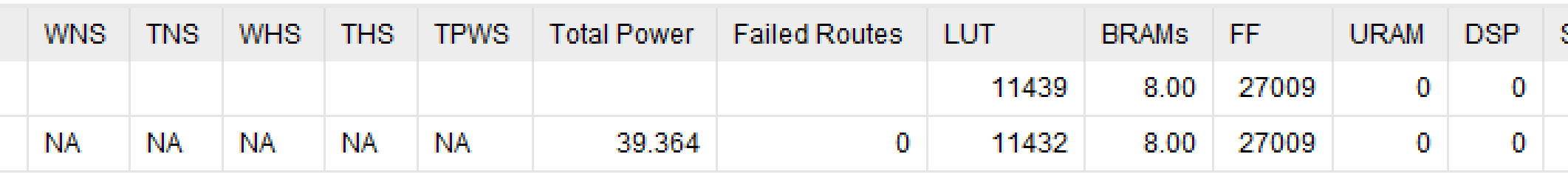
FIFO：

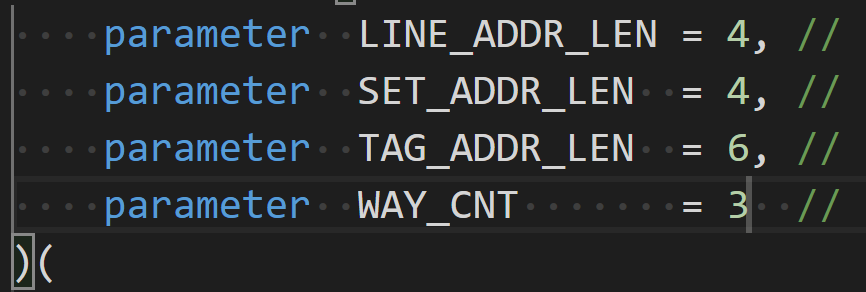




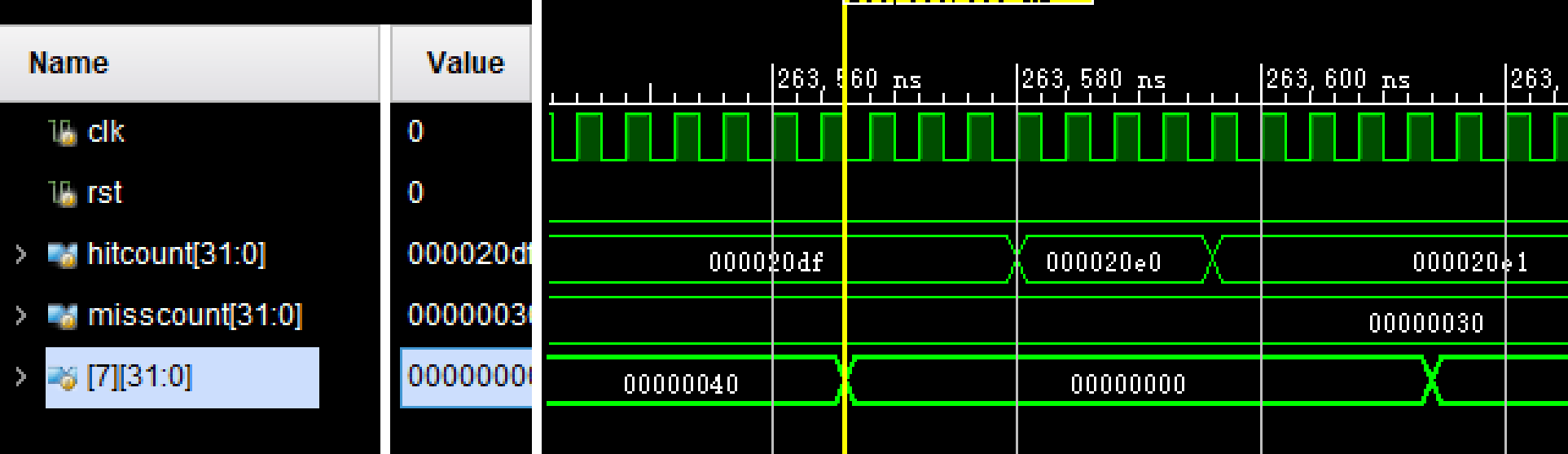
LRU：

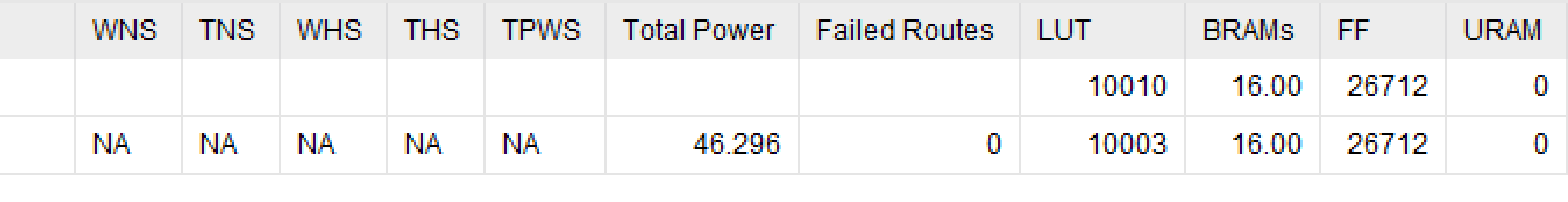




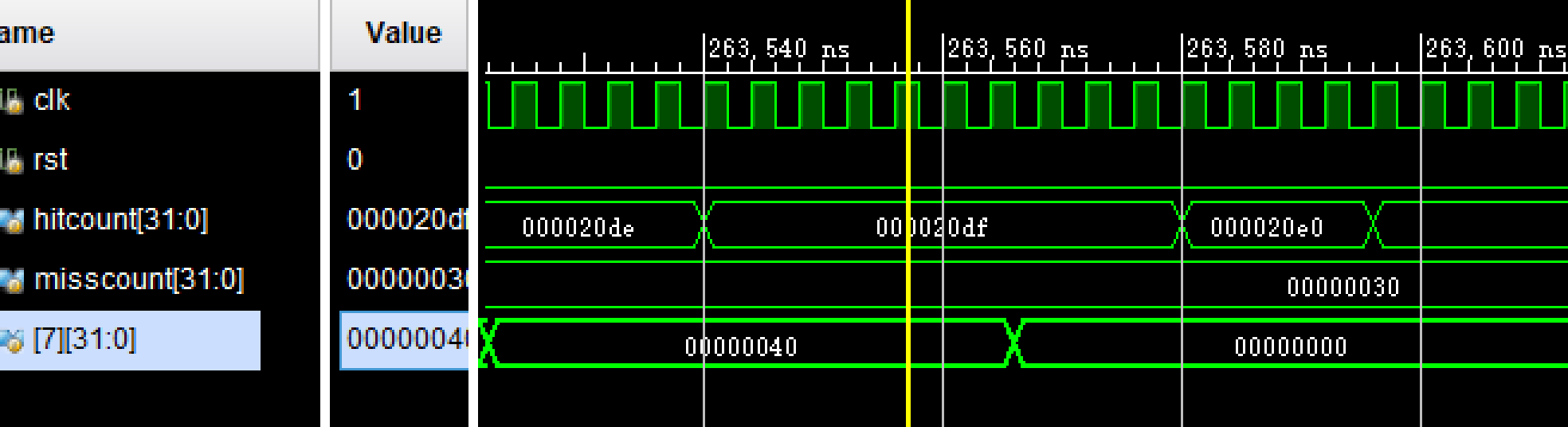


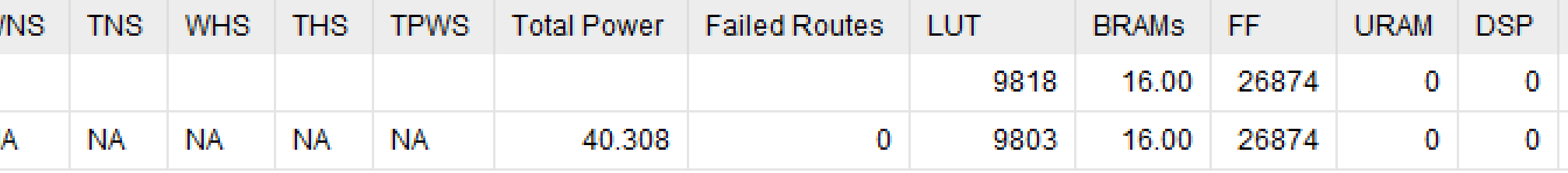
FIFO：

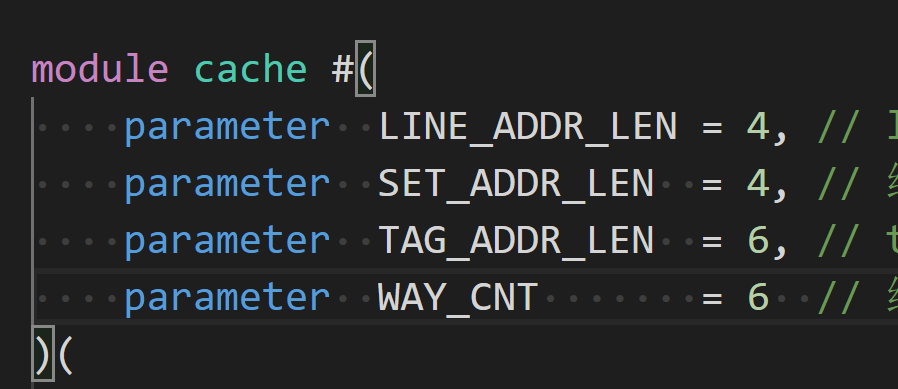




LRU：

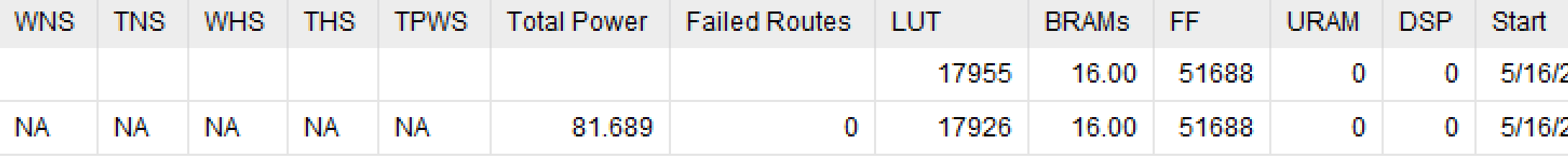




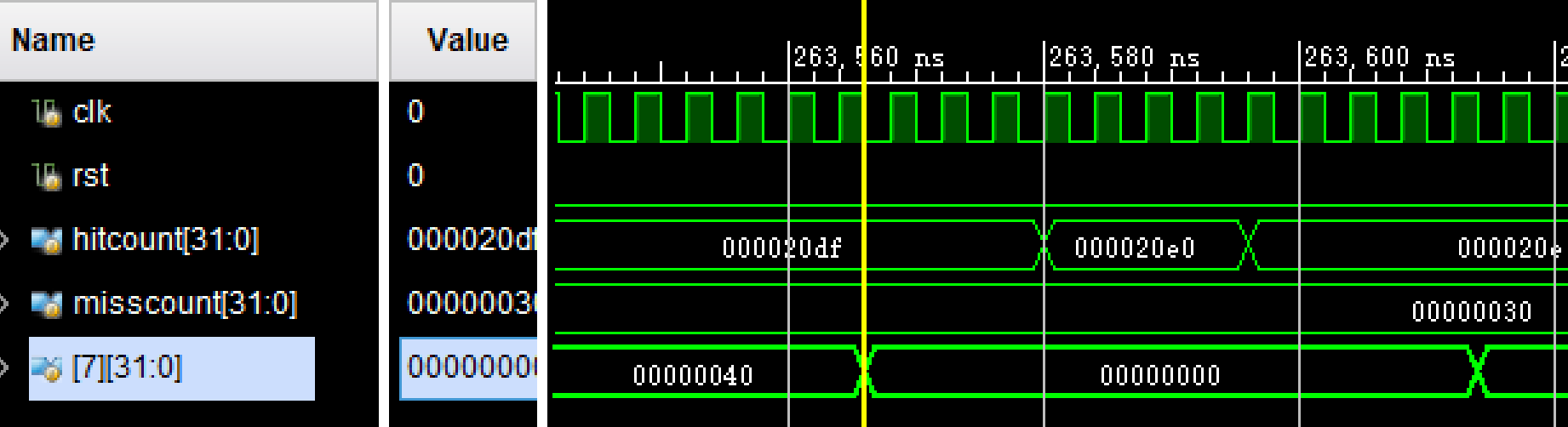


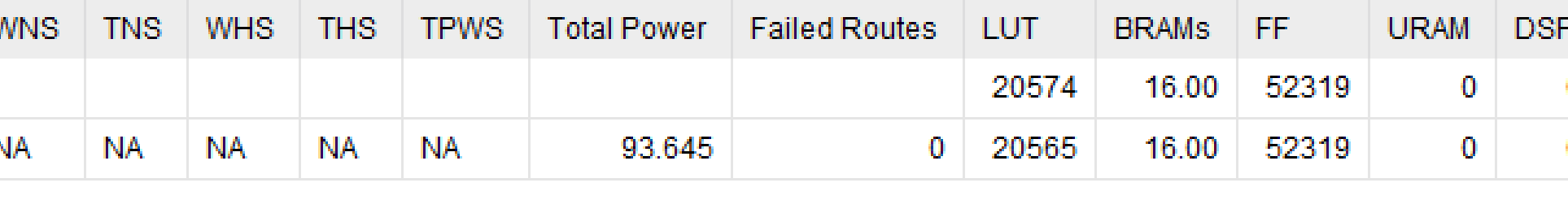
FIFO：





LRU:





我们设开始时间12000ps（较小忽略），结束时间为七号寄存器变为0的时间。时间单位为100000ps

电路面积用total power表示，以10000为单位1

以时间反比

将数据统计为表格得

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 替换策略 | LINE\_ADDR\_LEN | SET\_ADDR\_LEN | WAY\_CNT | 缺失率 | 运行时间 | 电路面积 | 效率与面积之比\*100 |
| FIFO | 3 | 3 | 3 | 54.9% | 13.0 | 1.14 | 6.75 |
| LRU | 3 | 3 | 3 | 50.9% | 11.9 | 1.00 | 8.40 |
| FIFO | 3 | 3 | 6 | 1.90% | 2.92 | 2.02 | 17.0 |
| LRU | 3 | 3 | 6 | 3.78% | 3.26 | 2.97 | 10.3 |
| FIFO | 3 | 4 | 3 | 10.8% | 4.91 | 2.23 | 9.13 |
| LRU | 3 | 4 | 3 | 7.90% | 4.01 | 2.23 | 11.2 |
| FIFO | 3 | 4 | 6 | 1.14% | 2.74 | 4.01 | 9.10 |
| LRU | 3 | 4 | 6 | 1.14% | 2.74 | 3.78 | 9.66 |
| FIFO | 4 | 3 | 3 | 11.5% | 5.11 | 2.26 | 8.66 |
| LRU | 4 | 3 | 3 | 6.83% | 3.82 | 2.27 | 11.5 |
| FIFO | 4 | 3 | 6 | 0.57% | 2.64 | 4.50 | 8.42 |
| LRU | 4 | 3 | 6 | 0.57% | 2.64 | 3.94 | 9.61 |
| FIFO | 4 | 4 | 3 | 0.57% | 2.64 | 4.63 | 8.18 |
| LRU | 4 | 4 | 3 | 0.57% | 2.64 | 4.03 | 9.40 |
| FIFO | 4 | 4 | 6 | 0.57% | 2.64 | 8.17 | 4.64 |
| LRU | 4 | 4 | 6 | 0.57% | 2.64 | 9.37 | 4.04 |

由表格数据容易发现：

1. 总体来说cache电路的面积和cache的容量成正比
2. 在cache容量没有接近数据规模的情况下，
3. 当cache较小时，缺失率很高，需要大量的换入换出
4. 通过变大LINE\_ADDR\_LEN，SET\_ADDR\_LEN，WAY\_CNT使cache容量翻倍，都可以使缺失率和运行时间显著减小，其中WAY\_CNT增大降低缺失率的效果最好。
5. 通过效率与面积之比，我们得出第三组数据整体功耗最好。
6. 当cache容量接接近或超过数据规模的时候，两种策略效果几乎一样，因为这个时候只需将所有数据换入cache即可，增大cache在缺失率与运行时间上提升并不显著。

## 实验总结

本次实验难点主要在LRU的实现上，要充分理解LRU的原理才能在硬件上较好的实现。

本次实验的一个易错点是记录缺失率的时候，命中次数与缺失次数都要在有读写请求的时候才能增加，否则会在cache空闲的时候不停自增。

通过本次实验，我初步掌握了FIFO与LRU两种替换策略以及cache参数变化时对性能的影响。

## 意见

无