做了什么

Time　2025/9/18/19：45
Author 王昊

本周工作聚焦于PQC系列算法**乱数流**生成模块及**采样方法**的硬件实现。

1. 采样方法硬件实现

完成了包括**中心二项分布CBD，拒绝采样以及均匀采样模块**的硬件电路电路实现。

1.1. CBD（Centered Binomial Distribution）采样模块

设计思想就是流水线核多通道并行采样。

主要文件：

- `cbd_sampler.v`：顶层 CBD 采样器，负责并行管理多个采样通道、与外部控制逻辑交互。
- `cbd_lane.v`：单通道采样核心，实现 CBD 候选值生成、Bernoulli 判决以及拒绝采样。
- `bernoulli_sampler.v`：纯组合的 Bernoulli 判决单元，通过阈值比较生成随机符号。
- `rejection_filter.v`：拒绝采样单元，根据候选值幅度动态调整接受概率。
- `test/cbd_sampler_tb.v`：针对顶层模块的 testbench，驱动随机向量并记录硬件输出。
- `test/cbd_golden.py`：Python 黄金模型，用于生成激励向量、计算期望结果并验证硬件输出，同时给出基础统计检验数据。

顶层 `cbd_sampler`

端口	方向	宽度	描述
`clk`	输入	1	时钟信号
`reset`	输入	1	同步复位，高电平有效
`start`	输入	1	触发一次采样操作，在 `valid_in` 有效时加载随机向量
`valid_in`	输入	1	随机数据有效指示
`random_in`	输入	`RAND_WIDTH`	来自 SHAKE/PRNG 的随机字节流
`threshold`	输入	`BERN_WIDTH`	Bernoulli 采样阈值

端口	方向	宽度	描述
`sampled_vals`	输出	`LANES*CAND_BITS`	合并后的 CBD 采样结果，按通道低位打包
`accepted_flags`	输出	`LANES`	各通道拒绝采样接受标志
`done`	输出	1	所有通道输出有效时置位
采用200个测试向量进行输出特性检测效果如下。			

```
D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test>py -3 -u cbd_golden.py --vectors 200 --seed 2024 --generate
当前工作目录: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test
脚本所在目录: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test
输入文件绝对路径: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test\cbd_input.txt
开始生成 200 个向量，种子: 2024
输入文件路径: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test\cbd_input.txt
期望文件路径: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test\cbd_expected.txt
Generated 200 vectors and golden results using seed 2024.

D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test>py -3 -u  cbd_golden.py --verify
当前工作目录: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test
脚本所在目录: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test
输入文件绝对路径: D:\desktopnew\Vivado_Projects\0.Kyber\1.CBD\test\cbd_input.txt
Accepted sample distribution summary:
  value -3: observed     6 (1.022%), expected 1.562%
  value -2: observed    52 (8.859%), expected 9.375%
  value -1: observed   126 (21.465%), expected 23.438%
  value  0: observed   214 (36.457%), expected 31.250%
  value  1: observed   135 (22.998%), expected 23.438%
  value  2: observed    49 (8.348%), expected 9.375%
  value  3: observed     5 (0.852%), expected 1.562%
Chi-squared statistic: 9.937 with 6 degrees of freedom
Sample mean: 0.0000, variance: 1.3015
Hardware output matches golden model.
```

1.2. 拒绝采样（Rejection Sampling）模块

支持同时处理多个 lane 的候选值，流水线深度是2级

接口说明

信号	方向	位宽	描述
`clk`	输入	1	时钟信号
`rst_n`	输入	1	低有效异步复位
`random_valid`	输入	1	随机数据有效信号

信号	方向	位宽	描述
`random_in`	输入	128	外部随机字节流，可用于追踪调试
`q`	输入	16	`uniform mod q` 模式的阈值
`cand_bus`	输入	`LANES*CAND_BITS`	候选值并行输入
`urnd_bus`	输入	`LANES*CAND_BITS`	均匀随机数输入，用于 Bernoulli 判断
`threshold_bus`	输入	`LANES*CAND_BITS`	各 lane 的比较阈值
`mode_select`	输入	`LANES`	每个 lane 的模式选择位
`acc_bus`	输出	`LANES`	接受标志，表示对应 lane 是否接受样本
`sample_tdata`	输出	`LANES*CAND_BITS`	通过拒绝采样后的噪声样本，未被接受的 lane 填零
`sample_tvalid`	输出	1	样本有效指示，`CONST_TIME=1` 时等于 `random_valid` 的延迟版本

这个模块的测试先是基于python的goldenmodel生成测试向量。

```
D:\desktopnew\Vivado_Projects\0.Kyber\3.REJECT\test>py -3 reject_sampler_golden.py
Vector file generated. Run the Verilog simulation next.

D:\desktopnew\Vivado_Projects\0.Kyber\3.REJECT\test>
```

测试向量如下所示

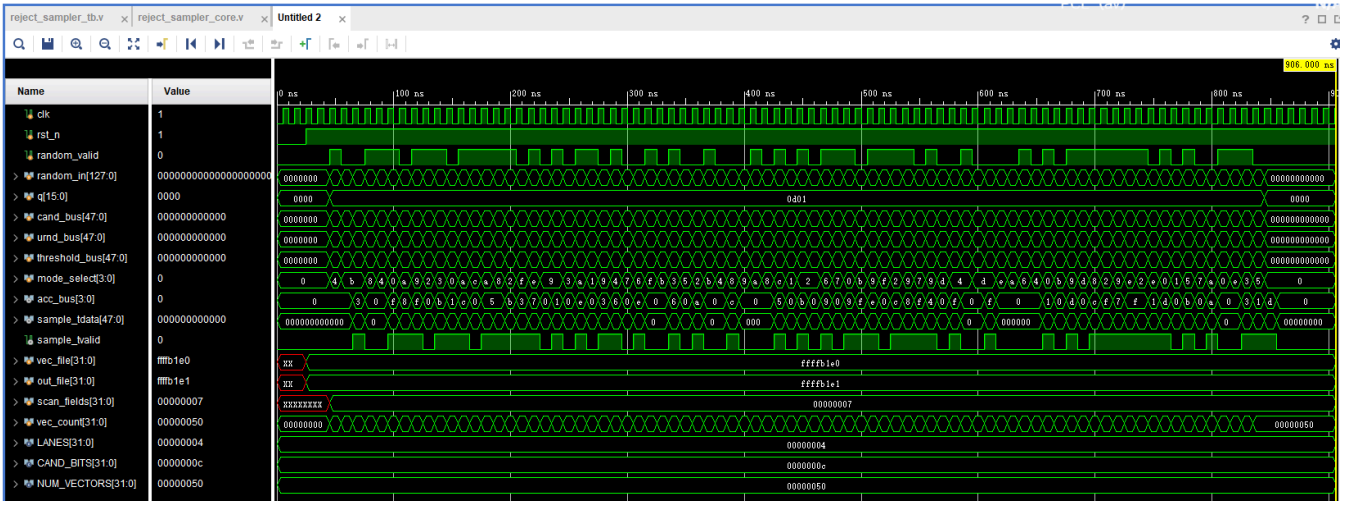
reject_vectors.txt

文件 编辑 查看 H1

```
1 0d01 d1f6689b85d0 b55ff27d887c a919e76fad4e 04 6972f683de11ee00366dadc088177abd
0 0d01 3f9ee8d54b2e a96c7da77472 da8a6a66fb0c 0b bffa07750a5d5bfe345986d33aa67f52
0 0d01 d79a3784809f 78d761a98395 67f45abb3edb 0b 55e3679f5656a72a9fa71a5963243c73
1 0d01 69fa9e43956f 02358f322218 b3578ae054d2 08 9a094dea760ee1c3b1ee4a0a3d41c6df
1 0d01 66ed46df3f6c 8e89ae9a4244 fd8338800458 04 51e8217b39260d7ad9a4e1e983e89a8a
1 0d01 3cb259c10175 6ba4031c8758 f9e78f58f2cf 00 f18f97a9cfb707cc1ab33b349e274b3f
0 0d01 003a125c1374 5d40ec8fd3ee 6a66b93521f7 0a 7431fb7dbb8b8a8545d01eb1504c2d68
1 0d01 d63e5ccce6f3 9f95216b4447 b695bd64dcbb 09 3912999cc5d8003800bf956dacb95ef2
1 0d01 fe2ede6da133 057aa0dcf199 73450b82eae1 02 568feb30dc7f4f27067930d52cc9c6f6
1 0d01 b9c52afd7813 a71fbdde2eeb 3d03e413151e 03 e4b4ae696f8532e6a8ddf0570debd1d2
0 0d01 a293c37638bf dcd3756cb466 519de3882349 00 682df1e792b9e160890dd63060ba4be5
1 0d01 8d89567e9587 cf1390e65b6b 74da2e903cbd 0a 4f00a56b92b1e4c477a80353c9effd6f
1 0d01 8121b8efc634 d3b8696b4c2d 44b91da1f44e 0c 84bc0a07a5dbc5263f5540974bf34d59
1 0d01 bacf5a5e1905 44720cc000fe 61ba5cd8d33e 0a 307b7edb854252c7d933db05abbec6bb
1 0d01 884f1c62955e 3353f9ec8237 321c5442b892 08 d12bc69323404cabe5b21f73a660d67a
1 0d01 d32ad542f372 0a478baaca83 88e475c398eb 02 8ace25a5bd9e3d016db48cadb3e51d95
0 0d01 81676e42d404 d73d8f3810cc 8dc4fcf35113 0f 94e0723dad5e7bb8e303a8d17b74a6e8
1 0d01 c405fd913101 ba3f8fb8ffdb 64d89421150f 0e bc34621c241f5b2201b7d2239f0a3326
0 0d01 29c903194d89 6baecdf5a2a0 f9f9afa07770 09 a60138e24f12dce93effc29db65e3ef3
1 0d01 47b140a88805 4d4b0cb1185a a1c07cb2f1b3 09 0cd2daa187e11691743cc0c7ce826ad0
0 0d01 c9686df77330 442666d321b8 e7906cba044c 03 7340b9c09c46bb06daa5106357b17af8
1 0d01 67ae24d3d038 fd8d153b6413 712e7dfe4cae 0a 1df2c76fc2ecc33cce81fe28e5ec652b
1 0d01 f7da7823fb8f c01d078c6f36 7fe09739483c 01 4515645503fd4e06cd58896e03c1abfc
0 0d01 756534ab136d 4f726d8c179f 2a8b7d802153 09 cee532d803480980c8be67d440b59a33
1 0d01 48a1277c4e40 ebb018886b45 1dae09bbc7c0 04 75b5d6af1f4d72ccc769c9b096fe0c13
0 0d01 2910177562bf 6556b8eef52c 281d13f12765 07 3226e9ffbddc1de95c6d74c43e126e86
0 0d01 8d141e0a3f5b cc2df27a9d33 e3bc0cc5650c 06 aaa77388a1868f3c1e1b477653709b75
1 0d01 3ecfcbb1829a ebb47e404621 d1cee2cc435e 0f 3cd8d2f0862b43ec9d88d13ed791fabc
0 0d01 eb80ded3e71d 5359c716bbb9 dfa7067608c3 0b 4cec6b6ce1a3dd6d85bc1deda813e655
1 0d01 680f3f66e086 2abdaae02908 997d65f764a9 03 f8368bf88a5a6ae9154365f03048492c
0 0d01 50353d6fdfe3 4d6cc183f2ae aeda779f5a40 05 14363297734e02ec63835172c3533b28
0 0d01 56091fd4013a 47c376174685 3ca5fa2966a1 02 3b7e2efdfa8633b7ff6dec818ee55b0f
1 0d01 05f66cf44627 8f7c65b499c2 bdce4b27a0ea 0b beacd196498c416217c88de126233426
0 0d01 7e4ba472b53b 6e4b5355ccb8 5e525ac54435 04 9a40f9afaee5937096e271e7b283f1bc
0 0d01 6e5d7cae3fe7 6f28c31e8196 0436f7e4aa0f 08 842f78157b85cdc2c13356a540f000d9
0 0d01 e6e1331bebc1 c98df6a37d23 a0e585294200 09 046b5f5b3a999a3df15008607f4f1baa
1 0d01 46ebfcc0c5d1 b8c5d7c4d1f2 8dc79272d28f 0a 5b833de06bd63ad06f48f5c05f1d73a1
0 0d01 fb31508ec074 7b360a60c1bd 2670705cff24 08 98c49a89cdc2109c0d2e1088b428d048
1 0d01 1361e6c24938 24ad0c7e13c5 2abafe2051c8 0c a9209b5e146307a7a02721cfc7cd75ee
```

行 1，列 1　6,560 个字符　纯文本　100%　Windows (CR

然后基于tb中的readmem函数实现自动化对比




```
D:\desktopnew\Vivado_Projects\0.Kyber\3.REJECT\test>py -3 -u reject_sampler_golden.py --verify
Hardware outputs match the golden model.

D:\desktopnew\Vivado_Projects\0.Kyber\3.REJECT\test>
```

1.3. 均匀采样（Uniform Sampling）模块

通过 Barrett 除法进行模运算加速，并在流水线结构中实现8路并行处理

接口说明

端口	方向	宽度	说明
`clk`	输入	1	模块时钟输入
`rst`	输入	1	异步复位，低电平释放
`valid_in`	输入	1	输入有效标志
`random_in`	输入	128	PRNG 的随机输入数据，8 个 16 bit 候选值
`q`	输入	16	模运算参数，目标范围 `[0, q)`
`valid_out`	输出	1	输出有效标志，延迟 3 个周期
`sampled_vals`	输出	128	并行输出的均匀采样结果
`accept_mask`	输出	8	每条通道的接受标志，`1` 表示候选值通过拒绝采样

```
INFO: [XSIM 43-4323] No Change in HDL. Linking previously generated obj files to create kernel
INFO: [USF-XSim-69] 'elaborate' step finished in '1' seconds
Vivado Simulator 2018.3
Time resolution is 1 ps
Test completed successfully
$finish called at time : 785 ns : File "D:/desktopnew/Vivado_Projects/0.Kyber/4.UNIFORM/test/uniform_sampler_tb.v" Line 89
```

Vivado Tcl端Simulation测试通过。

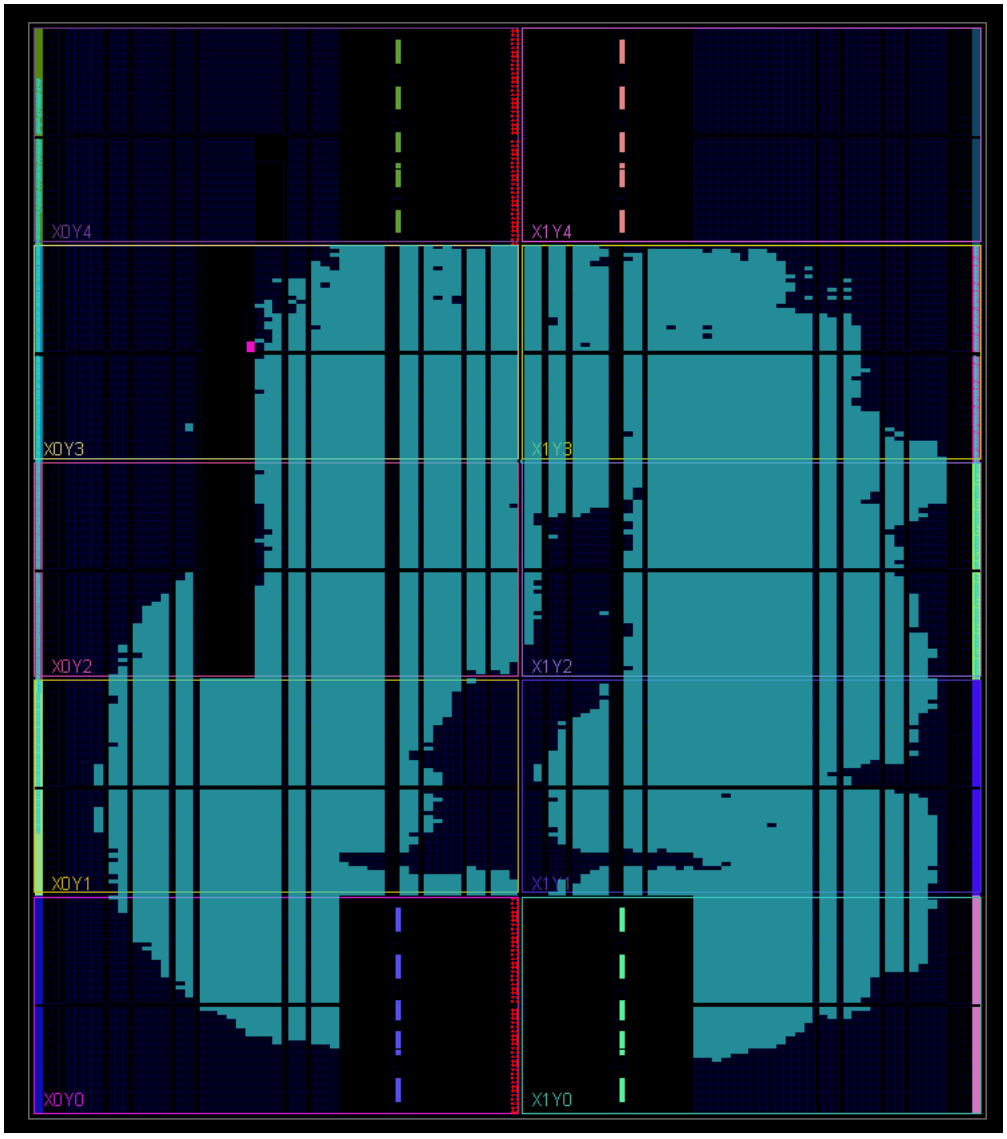
2. 乱数流生成

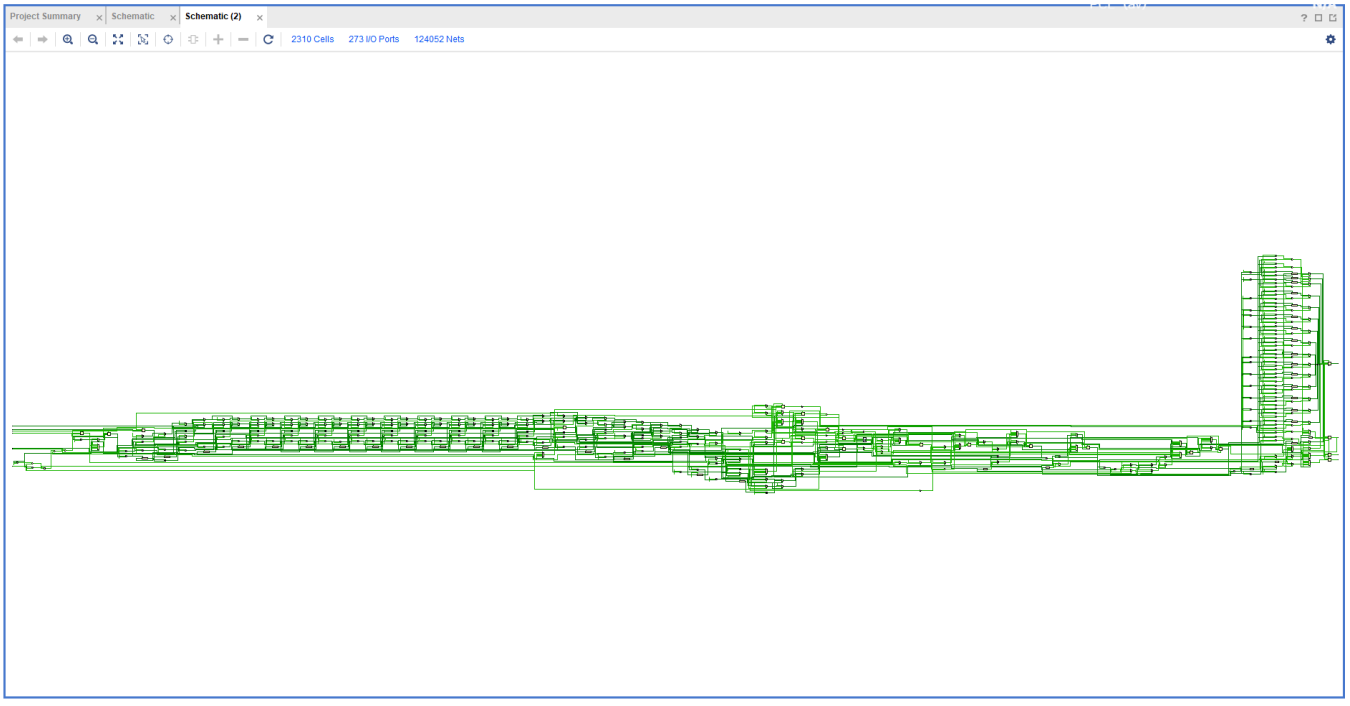
完成了**Keccak核**，支持SHAKE-128与SHAKE-256两种模式。采用了**24级流水线**设计

这周的大部分时间花在了Keccak核上，其余采样方法是很好搞的。

因为流水线级数很深，导致优化很困难，而且资源大量占用。（基于Artix 7系列芯片）

Resource	Utilization	Available	Utilization %
LUT	69915	133800	52.25
FF	40387	267600	15.09
IO	273	285	95.79
BUFG	2	32	6.25





本来是想着做一个高速的Keccak核，现在看来加深流水不太行。

3. 论文阅读

最近在看Yuriy Polyakov，Sujoy Sinha Roy 以及Kristin E. Lauter等人的文章。但是有些多，而且搞这几个代码，尤其是Keccak搞太久了，论文进度比较慢，此处就不再列出。

4. 总结

之前说暑期工程实践，我就做PQC采样方法的硬件实现来了。现在看来采样只是一个很简单的，最不可能成为短板的一个东西。所以我觉得后面还是把所有采样方法集成且重构，然后开始做NTT PWM等是一个比较可取的方式。
以上几个硬件代码都是雏形，我只能确保其功能是正确的。关于方法上的创新型，PPA指标等都等待进一步提升，可能会觉得我搞这么多算法没啥用，但是我感觉首先得完成一个从无到有，然后再去慢慢优化的一个过程。毕竟去IEEE，Crypto上找硬件实现的论文，那种太高难度的方法也不是一两周就能复现出来的，所以先把简单的搞定，然后看看哪些地方可以继续优化。
我下一步计划就是先搭建一个简单的kyber框架出来，不涉及RISCV和ISA相关，先让这个框架能跑起来，测一测整个系统中的短木板到底是在哪里，然后继续改进。
当然这个过程中的全局规划还是要做的，控制逻辑也是要做的，舍弃了ISA后只能通过硬件调度完成整个kyber，这个过程也是相当耗费心力，不过目前我是先打算把组件搞定，比如采样方法，乱数流生成核心，NTT，INTT，PWM等。

遇到什么问题

- 几个语法问题，在google后已经解决。
- Keccak核的性能尚不能满足需求。
- NTT编程困难。尝试用 Cooley–Tukey 分层算法和Montgomery 模乘方案的NTT，还没编出来。这个有些难度，我在假期尝试编程，不久后转去采样方法了，因此弃置了一段时间。
- 在构造Kyber过程中，预测未来会遇到各个部分频率不同的问题，也就是工程中常说的木桶效应（短板效应）。为此我觉得一个比较好的方式是搭建整体框架，包括我现在实现的Keccak和采样，都是一个baseline版本。最后落到应用上的话肯定是优化后的。尤其是Keccak核，硬件编程中我其实用了挺多的软件编程思维，未来优化空间很大。
- 关于验证困难问题。已经解决，现在对构造goldenmodel很熟练了，进阶了一下VScode使用方式，以上几个硬件工程的代码，都可以在cmd中调用指令实现自动验证。

下周计划

- 离散高斯分布采样和恒定时间常数采样还没做，下周去完成。
- NTT/INTT 模块设计：继续Kyber 算法核心的 NTT 与 INTT 变换模块实现，计划采用 Radix-2 蝶形结构 + 流水线 + Cooley–Tukey 分层算法+Montgomery 模乘方案。
- 读论文

APPENDIX

电路工程文件详见附件Codes.zip