EE272B: Design Projects in VLSI Systems II
Mid-Quarter Review

Kylee Krzanich Sam Xu

May 1, 2021

Project Repository

Repository

```
https://github.com/krsandwich/EE272B
```

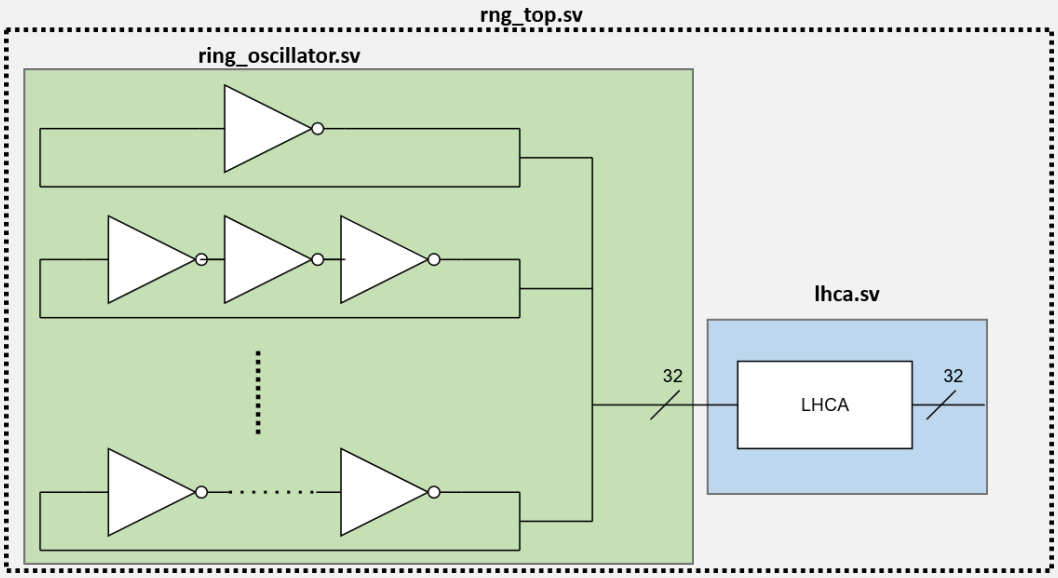
See commit hash: $d9598ef9da29977998abd91f4b384d73cec3cf41$
The repository contains a submodule which links to our chipyard repository containing our full-chip implementation of Chipyard's RocketChip with our accelerators. That repository can be found separately here:

```
https://github.com/inSam/chipyard
```

File Structure

RNG

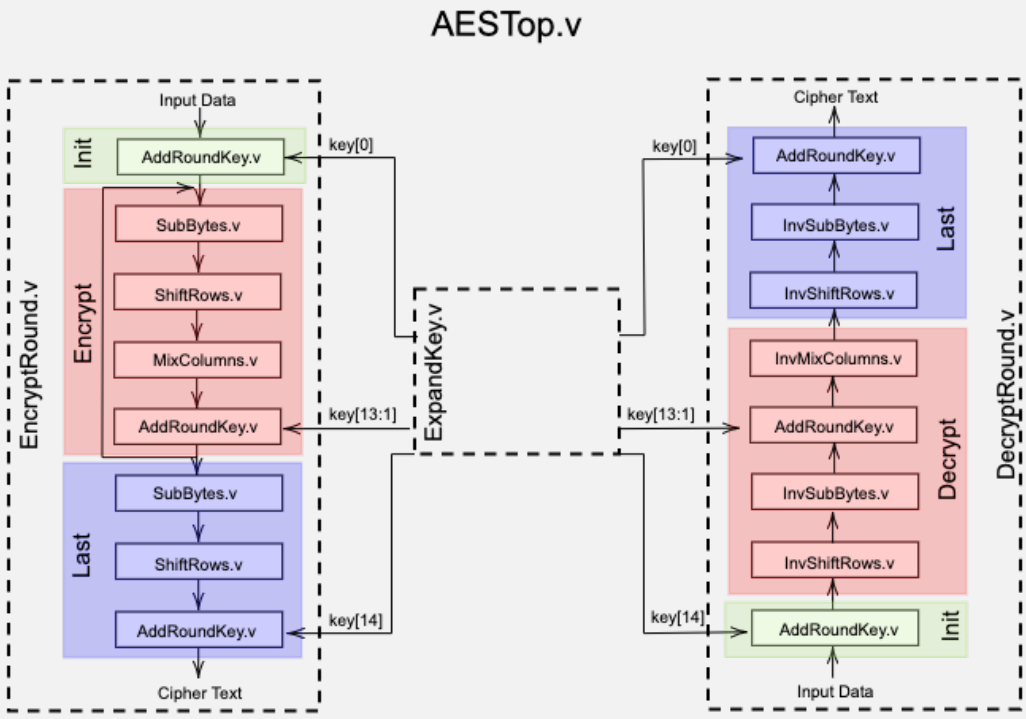
1. **rng_top.v**:This file contains all the modules which are depicted below.



2. **testbench**:
   (a) **rng_tb.sv**: This is the top level testbench that generates $> 10,000$ random numbers
   (b) **test_rng.py**: This script calls the testbench and runs a series of statistical tests to check the randomness and uniformity of the random numbers.

AES

1. **rtl/AESTopFinal.v**:This file contains all the final modules which are described below.



2. **testbench/**:
   (a) **AESTopTb.sv**: This is the top level test bench for the entire top level. The rest are for testing individual modules.
   (b) **EncryptTb.sv**
   (c) **DecryptTb.sv**
   (d) **ExpandKeyTb.sv**
   (e) **AddRoundKeyTb.sv**
   (f) **MixColsTb.sv**
   (g) **ShiftRowsTb.sv**

Running RTL Tests

RNG

```
    python3 test_rng.py

```

The python script will both run the simulation and check run the statistical tests on its outputs. If the random numbers pass the statistical tests, which includes Z-Statistics, Kolmogrov-Smirnov, and Chi-Squared.

AES

```
cd AES/design
make check
```

The call to `make` will print the vcs and gcc compilation output to `compile_tb.log` and `compile_c.log` respectively. Then, it will output the simulation and if the encrypt input matches the decrypt output then it will print PASS. Next it will check the output of encrypt with a c code AES test. If you would like to change the keys or input data at any point, please change the `key.mems` and `inputs.mems` files. If you would like to run individual tests such as `ExpandKeyTb.sv`, run the following

```
cd AES/design
vcs -full64 -sverilog -timescale=1ns/1ps -debug_access+pp testbench/[INSERT NAME].sv
    rtl/AESTopFinal.v
./simv
```

If the modules are working correctly, you should just see PASS printed out.

Running Full-Chip Simulation with Chipyard

Our accelerator modules fully incorporated into chipyard can be found in the following repository:

```
https://github.com/inSam/chipyard
```

We have functioning full-chip simulation on chipyard with the following custom C-Tests found in **tests/** directory:

1. aes.c
2. rng.c

To run our full-chip simulation of chipyard with our accelerators, simply run the following in **sims/vcs**:

```
make run-binary CONFIG=CryptoConfig BINARY=../../tests/[TEST].riscv
```

Design Metrics

For our performance metric, we compared the performance of C code calling our accelerator to the stdlib random number generator **rand()**. The stdlib random number generator uses a simple version of a linear-feedback-shift-register, however, without any sources of entropy, it only generates pseudo-random numbers.
The stdlib tests can be found at **tests/rng_noa.c, tests/empty.c**. Using the number of simulation cycles as a estimate, we see that our accelerator reduces the number of cycles spent for random number generation by 75%. Therefore, our accelerator has a significant performance benefit on top of being a true-random-number-generator.
More specifically, $25715ns \rightarrow 6524ns$ in order to generate and print 10 random numbers using our module.
The performance benefit was likely because our design is able to generate a new random number every cycle.

For our AES module, to perform a single input-key pair encrypt and decrypt, we were able to achieve this in just $566ns$ using our accelerator compared to $13018ns$ using a native c-implementation. However, the performance of our AES module may have to be lowered as we work on timing of this module. Currently our area estimates are approximately $1882828.2\mu m^2$ which we hope to continue to improve.