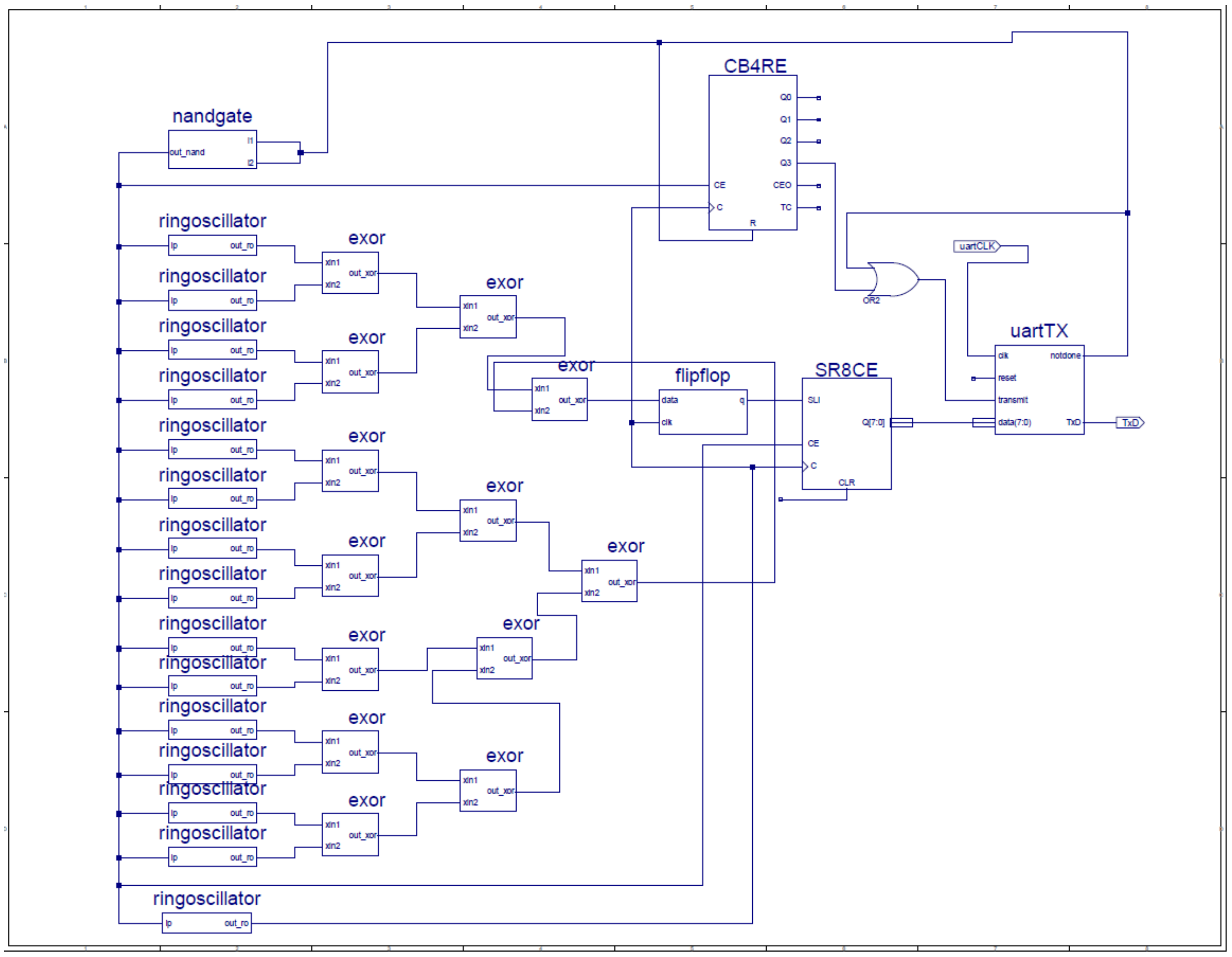
Introduction

The need for random number generators is very obvious in cryptographic protocols due to their unpredictability. Data has to be random and unpredictable for any cryptographic system to be secure. Physical Unclonable Functions (PUF) can be used for random number generation due to their one -way functionality and unclonable nature caused by process variation in IC fabrication. PUF's have inherent properties suitable for cryptographic key generation, chip authentication and random number generation.

In this project we were required to do FPGA-based PUF implementation. Implementing PUF on FPGAs involves significant challenges due to the limitations in knowledge of gate-level structure of the FPGA. Our Random number generator is based on the design in "Physical Unclonable Function and True Random Number Generator : a Compact and Scalable Implementation" as it provides drastic area reduction and scalability which are necessary in a FPGA implementation.

Theory of Operation



The input to the ring oscillator is a control signal to start and stop the oscillation. This signal is controlled by the UART signal which will pause the generation of random bits while it is transmitting the previously generated 8 bits. Once a character is finished being transmitted, the input to the ring oscillator will go high and begin feeding random bits into the XOR tree. The output of the XOR tree feeds to a flip flop which will be used to sample the random bitstream. The reason for choosing this design was for the scalability for testing different configurations. Additionally, it was a simple design and a good place to start experimenting.

The basic concept of Ring oscillator exploited in PUFs is that no two oscillators oscillate at same frequency due to process variation. Our design uses ring oscillator with three inverters and a latch. Each stage provides a phase shift of $\pi/3$ and remaining π is provided by the dc inversion. The Ring oscillator modules are controlled by the sampling clock of the flip- flop which is in turn fed to the XOR and finally to another Flip- flop to get random bit stream.

The remaining circuitry is for control and data transmitting purposes. When the element *flipflop* sees a negative clock edge, it will read the data bit. When the 8 bit shift register sees a positive edge, it will read the value from the flip flop. The counter is a 4-bit counter that when it hits 8, will signal the UART module to transmit the 8 bits from the shift register. We experimented a little with the signal to capture the bit from the XOR tree and decided to use an additional ring oscillator to trigger the clock for the shift register and counter. The transmit needs to be asserted the entire time the data is to be transmitted, so we added a control signal called notdone to hold the system in reset and allow time for the data to be transmitted.

The ring oscillator, exor, flipflop and nand gate were programmed in structural verilog as required by the project description. Once these modules were programmed, a schematic symbol was created to allow easier modifications and analysis of the circuit. The control circuitry was introduced by adding pre-made schematic symbols to the project.

After the project was compiled, we programmed it to the NEXYS 2 FPGA board by Digilent. We found that of none of our configurations did we exceed 1% utilization of the FPGA hardware. The program started the moment the programming was completed. Data was sent 8-bits at a time to a computer using the on-board UART/RS-232 converter. A program called RealTerm was used to read the data coming in at 115200 baud and write the raw binary data to files. Each file captured was 10.5MB to start and we moved to 10.9MB as this was closer to the limit allowed by our DIEHARD analysis program. Data was captured at about 84000 bits per second.

Randomness tests

The random number generator would be of no use if the bits were not truly random. For analyzing our data, we chose the DIEHARD suite of tests. We found a simple program consisting of 15 different tests from Florida State University. We found this through the Computer Security Resource Center subdomain of the NIST.gov website. Below is a table of the p-values reported by each of the tests.

	Configurati on 1	Configurati on 2	Configurati on 3	Configurati on 4	Configurati on 5	Configurati on 6
Test 1	0.945396	0.707231	0.678550	0.986255	0.499671	0.707231
Test 2	0.867284	0.005960	0.660993	0.280946	0.635438	0.351279
Test 3	0.750098	0.941036	0.549448	0.460903	0.411146	0.616631
Test 4	1.000000	1.000000	1.000000	1.000000	0.604330	0.999998
Test 5	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Test 6	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Test 7	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Test 8	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Test 9	0.999993	0.999981	0.898372	1.000000	1.000000	1.000000
Test 10	0.950756	0.357439	0.684720	0.989251	0.999998	0.503590
Test 11	0.927618	0.923622	0.755347	0.756841	0.039171	0.998805
Test 12	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Test 13	0.187307	0.623619	0.571687	0.249305	0.199229	0.015428
Test 14	0.582885	0.257433	0.027016	0.400894	0.973895	0.575681
Test 15	0.999914	0.999931	0.998771	0.083578	0.999847	1.000000

Configuration 1: 8 ring oscillators triggered on 50MHz/8 clock signal
Configuration 2: 8 ring oscillators triggered on additional ring oscillator
Configuration 3: Same as configuration 2, but is a restart test
Configuration 4: Same as configuration 2, but the ring oscillators are always running
Configuration 5: 14 ring oscillators triggered with a 15th ring oscillator
Configuration 6: 14 ring oscillators with 7 instead of 3 inverters and a 15th ring oscillator for trigger

Test 1: Birthday Spacings
Test 2: Overlapping Permutations
Test 3: Rank of 32x32 matrices
Test 4: Ranks of 6x8 matrices
Test 5: Monkey Tests on 20-bit Words
Test 6: Monkey Tests OPSO, OQSO, DNA
Test 7: Count the 1’s in a Stream of Bytes

Test 8: Count the 1’s in Specific Bytes
Test 9: Parking Lot Test
Test 10: Minimum Distance Test
Test 11: Random Spheres Test
Test 12: The Sqeeze Test
Test 13: Overlapping Sums Test
Test 14: Runs Test
Test 15: The Craps Test

According to the test suite each test is based on a calculated test statistic value , which inturn is a function of the data. For all the individual tests P-value is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested , given the kind of non-randomness assessed by the test. Accordingly, if a P-value is determined to be equal to 1 , then the sequence is concluded to have perfect randomness. A P-value of zero points to a completely non-random sequence. Our results have P-values in the desired range for each of the tests at different conditions. Hence, sequences generated can be considered to be random with a confidence of 99.9% .

Conclusion

Implementing a cryptographically secure random number generator on an FPGA is very feasible and produced good results. We found through a battery of tests, that the data produced was indeed random and thus safe to use for security applications. Most of the challenges we encountered with this project had to do with Xilinx as well as the associated simulation software, and for good reason. There are no initial states of the ring oscillator, thus the simulator did not know where to start and how to resolve the combinatorial loops. Overall, I consider this project to have been successful. All data has been recorded and is available in the supplementary files.