TRUE RANDOM NUMBER GENERATION USING PHYSICALLY UNCLONABLE FUNCTIONS

A Major Qualifying Project

Submitted to the Faculty

of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science in

Electrical and Computer Engineering

by

Berk Birand

Date: April 24, 2008

APPROVED:

Professor Berk Sunar, Project Advisor

Abstract

As embedded systems are becoming ubiquitous, the need for low-power circuits is increasing. An approach to reducing the complexity and power consumption of chips is to reuse components that are already present on the chip in alternative ways. Our design reuses the Physically Unclonable Function, mostly used for authentication, as a True Random Number Generator. With this approach, more secure authentication protocols that use randomness can be devised without adding too much complexity to the design.

Acknowledgments

I would like to thank my advisor, Prof Berk Sunar for his constant supervision throughout the project. He not only has introduced me to formal academic research, but has also paved the way for my Ph.D. I also would like to thank my lab partners, Ghaith, Erdinç, Kahraman and Deniz for accepting me in their close-knit circle, and for always being available to answer my questions.

Although I am the author of this report, I cannot take full credit for this work. I could not have achieved anything without the support and trust of my parents, Serda and Refik, and my brother, Burak.

Contents

1 Introduction

1.1 Problem Description

With the new advances in silicon production techniques, computers are entering every single part of our lives. Embedded systems contain chips that are so small that they can use the ambient electromagnetic radiation to generate their power.

Having so many small computers bring their own problems. They have very strict requirements, and since they do not have that many resources, their use is really hard. The amount of resources that can be allocated for them makes them especially prone to being actively hacked. Even on larger chips, space is always a premium.

For instance, chip space and power consumption are big limitations on the modern ultra-low-power devices. Since cryptographical circuits are computationally demanding, these smaller devices cannot use the bleeding-edge protocols.

1.2 Proposed Solution

It is always necessary to minimize the chip area and power consumption of embedded systems. One approach to increasing the efficiency of the chip is to reuse some components for several purposes. We propose the use of the PUF circuits as a random number generator.

The PUF circuit uses the chip's physical characteristics to identify the chip. They are frequently used for implementing authentication protocols. By using a feedback mechanism, our design exploits metastability in these circuits to build a true hardware random number generator. Since we are only adding a small circuit to the main PUF, our solution does not add a significant overhead in terms of power and area consumption.

1.3 Project Goals

The goal of this project is to design, implement and test the use of the PUF as a random number generator. The circuit has first been designed according to some requirements. The PUF circuit is implemented on a Xilinx XCVP30 FPGA prototyping board. Once the device is shown to operate properly, we built a feedback loop around it to make it function as a random number generator. Once the implementation was completed, the performance of the RNG was evaluated using statistical tests.

2 Literature Review

2.1 Random Number Generation

Random number generators (RNGs) are used in fields as varied as cryptography to music. Although the restrictions on the randomness of the numbers depends on the application, the RNGs can be classified into two categories: Pseudo-Random Number Generators (PRNGs) and True Random Number Generators (TRNGs).

2.1.1 Pseudo-Random Number Generators (PRNG)

PRNGs use a deterministic algorithm to generate a sequence of numbers from an initial value, called the *seed*. Given the same seed, the PRNG will always come up with the same sequence.

There are many implementations of PRNG functions. While most are more suited for being programmed into a processor, some others can be easily implemented on digital hardware. One commom digital implementation of a PRNG is the linear feedback shift register (LFSR) is widely used on chips as a PRNG. It is a shift register, where the bit that is shifted in with each state is a linear combination of the previous value (calculated by XORing several bits of the value). The register is initialized to a value, and once it is started, it keeps generating new numbers by continuously shifting in new bits.

The most critical factor when implementing a PRNG scheme is the source of the seed. For the system to work seamlessly like a real random number generator, the seed must be really random. On computer systems, common parameters such as date and time of the day, network activity or mouse movements can be made into a seed. This approach works well for applications that do not require much security (e.g., movement of a character in a game or generation of music).

PRNGs can be seeded from true-random number generators on certain occasions. These devices ar useful on systems where the TRNG works very

slowly, and cannot generate a throughput necessary for the application. It is only used initially to get a small initial value that is increased in length by the PRNG.

2.1.2 True Random Number Generators (TRNG)

True Random Number Generators (TRNGs) rely on a physical source of entropy to generate the bitstream. The circuit measures the entropy and converts it to bits. The means through which this measurement is made depends on the source of the randomness. An analog amplifier is used when thermal noise is used as the physical source. When radiation is used as a source, a Geiger counter can be employed. Other examples of entropy sources include avalanche noise of a Zener diode, atmospheric noise, jitter in an oscillator ring or traveling photons.

2.2 Physically Unclonable Functions

Physically Unclonable Functions (PUFs) use the physical properties of the chip on which the circuit is built to provide a secret. Using this approach, a secret key does not have to be stored on memory inside the chip [LLG+05]. The chip virtually cannot be duplicated; to do so, one would need to manufacture a different chip with the exact same characteristics as the original one.

In addition to being unique, PUFs also provide tamper resilience. If an attacker attempts to break the system by changing or monitoring the environmental conditions, the physical parameters will change, rendering the circuit unusuable.

The original PUF functions used optical patterns to provide the randomness [Rav01]. The version that we are using relies on the variations in propagation delays in the wires and gates [GCvDD03]. More information on this version of the PUF is available in Section 4.

The most important application of PUFs is in authentication. Before the system is deployed, the behavior of the PUF is recorded in a database. Its output when given a set of challenges is stored for future use. When the chip needs to be authenticated, a recorded challenge is sent, and the given response is compared to the one stored in the database. If the two match, the device is successfully authenticated.

2.3 Statistical Tests

By analyzing a large dataset, it is possible to understand the distribution of the numbers, and gauge whether they are suitable for use in real applications. Two of the most important test suites are Diehard [Die] and NIST [NIS]. For the purpose of this project, we used the NIST suite.

NIST runs a series of tests on the given data. It slices the data into a number of bitstreams, and performs the tests on each bitstream individually. Two values are given for the results: a *p-value* and a *proportion*. The test suites starts with the hypothesis that the bitstream is random (called the null hypothesis). With each test that it performs, the software tries to prove that the null hypothesis is correct. The p-value is the probability that the null hypothesis is true for the specific test. The proportion value is the percentage of the bitstreams that passed the tests. When the p-value and proportion values are higher than a calculated threshold, the test is labeled as a pass, which indicates that bitstream is random as far as the test goes.

As an example, the frequency test checks the occurence of 0 and 1 in the stream to see if they behave like a real random number, which is to say there should be the approximately the same number of ones and zeros. The runs test checks if the continuous groups of ones and zeros (called a runs) behave like the output of an ideal TRNG.

3 Requirements

Before beginning the design of this project, we need to establish the requirements for our design.

- PUF Reusability

  The RNG design should use the PUF that is already present in the chip. The PUF will be used for some other job, such as authentication.

- Area Efficiency

  The implementation is geared towards low-power devices. It therefore needs to occupy only a limited amount of area on the chip. A typical guideline is to limit the use of the the cryptographic circuitry to less than 1000 gates.

- Tamper Resilience

  The system needs to be tamper resilient, and should therefore invalidate its output when its used. Using the PUF circuit as the source of the randomization makes sure that the circuit will indeed be tamper-proof. As explained in section above, performing an attack on the circuit modifies the circuit parameters..

- Low-power

  Keeping the power consumption at a minimum goes hand in hand with the low area efficiency. Smaller devices such as Wireless RFID devices, or wireless sensor networks do not have much available power. The circuit should work with minimal effort.

4 Design Overview

This section will summarize the design phase of the project by visiting each step of the process, starting from the PUF design.

4.1 Switch Block Chain

The switch-based PUF circuit relies on switching blocks to forward the pulse to the next step [GCvDD03]. The interface to these blocks have three inputs and two outputs. The first two inputs accept the pulses coming from the previous block. The third input takes in a one bit challenge. If the challenge bit is a zero, then the pulses are sent directly to the two outputs pins. If the challenge bit is a one, then the inputs are alternated and relayed to the outputs (input A goes to output B and vice versa). The challenge bit can thus control the shape of the path the two pulses take.

The switching blocks are implemented using multiplexers. Two 2-to-1 multiplexers are placed in each block, and are both connected to the same SELECT signal (see Figure 2).

We need to account for the internal optimizations in order to respect the chain-like structure of the design. Before placing the VHDL code on the FPGA, the Xilinx environment attempts to optimize the structure. It tends to fuse the many blocks into an equivalent structure that does not have a chain structure. Such optimizations must be disabled. We used VHDL constraints to tell the optimizer to place each of these codes into exactly one CLB, and not to place anything else in there. Although this approach does

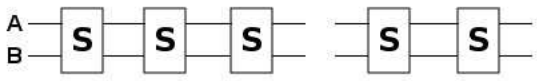


Figure 1: Switch chain

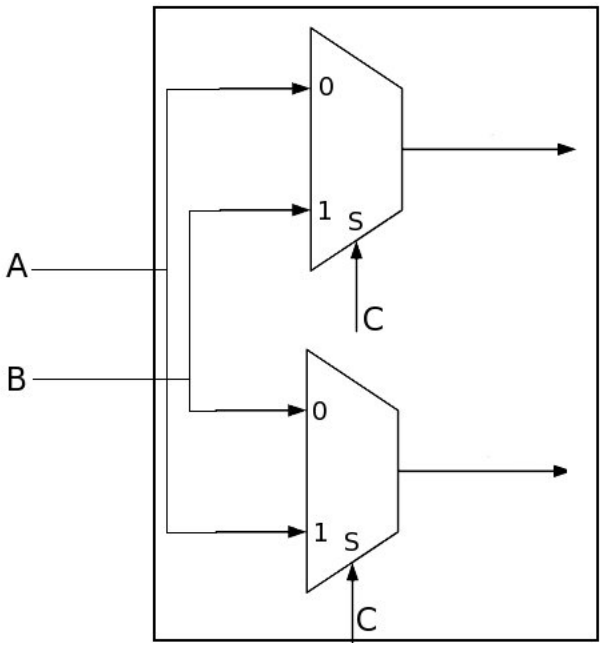


Figure 2: Inside of a switch block

not make the best use of the chip space on the FPGA, it is necessary to maintain the characteristics of the PUF circuit.

The PUFSwitch_Down and PUFSwitch_Up components are two 2-to-1 multiplexers. They are kept in their own entity files in order to make sure they are placed in slices of their own.

The content of the CLB blocks are shown in the Figures below. The content of the Look-Up Table (LUT) is also shown, as captured from the Xilinx FPGA Editor.

4.2 Arbiter and Metastability

In order to measure which of the pulses arrived at the destination, we will use a flip-lop as an arbiter. A flip-flop contained in the CLB block is connected to the last switch block as shown on Figure 3. One of the outputs of the switch block is connected to the Clock (CLK) pin, and the other output to

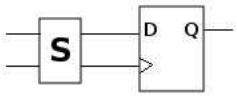


Figure 3: Last switch block and arbiter

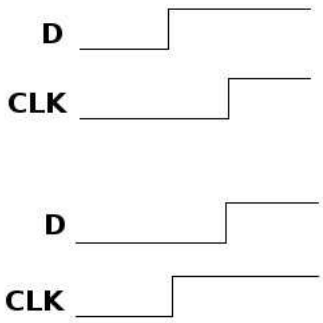


Figure 4: Timing of D and CLK signals

the Data (D) pin.

This setup of the flip-flop allows us to measure which of the two pulses arrived at the end of the chain first. If the clock signal reaches the FF first then a zero will be sampled (because the D signal is still zero), which will make the output to be a zero. On the other hand, if the D signal arrives first, then D will be equal to one when the clock pulse arrives, and the output will be one. As it can be seen from this discussion, the output is expected to be one if pulse A arrives there and zero if pulse B arrives there. A timing diagram of the arbiter is shown on Figure 4.

There is however a third case that may affect of the circuit works. If the two signals arrive there almost at the same time, then the output will be unpredictable. The data input to a flip-flop should be held constant for a certain time called 'setup delay' before the clock pulse occurs.

When the input does change within the setup window, the flip-flop enters

a metastable state. The output oscillates between 0 and 1, and keeps oscillating until it settles to one of them after an undefined amount of time. The likelihood for this state to persist decreases exponentially with time. The longer the flip-flop is in the metastable state the more likely it is to get out of it.

This behavior is usually avoided, as it may render a state machine to behave unexpectedly. For the purpose of our random number generator, we are intentionally looking for the set of challenges that will generate unpredictable output bits. These challenges create two paths in the switch chain that are so close to each other that the two pulses end up reaching the arbiter very close to each other, thus violating the setup time. Since the arbiter goes into the metastable state, the output starts oscillating and finally settles to a random value. The value that we obtaini thus is random, and is used to put the device in a feedback loop.

4.3 Feedback Loops

Our most important contribution to the PUF design is to add a feedback mechanism around the switch chain. We are feeding the sampled output back into a left-shift register whose parallel outputs are mapped to the challenge pins of the switch blocks (see Figure 5). As a result, every time a bit is sampled with the arbiter, a new challenge is obtained through the shift register. If the new bit is not random, the next state can be predicted from the previous one and the circuit behaves expectedly. When the new bit comes from the metastability of the flip-flop, the next state cannot be predicted from the previous one.

This setup allows the system to keep looping while generating new bits. With each random number generated, the system will deviate from the typical output of a pseudo-random number generator. After a few loops, the system enters a state that is completely unrelated to the initial state. After that point, the circuit behaves like a random number generator, and the

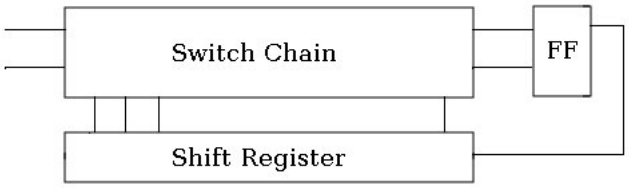


Figure 5: Feedback loop

subsequent bits generated can be used as part of a protocol.

4.4 State Machine

The PUF switch chain and arbiter are entirely asynchronous and do not require a state machine or a clock signal to operate. To use the shift register and to handle the serial communication with the computer, the circuit needs some sequential logic. This state machine sends the pulses to initiate the PUF's function, and collects the output of the flip-flop. It shifts the bit into the register, and sends the result to the computer through serial port. This sequence is repeated *ad infinitum* to keep generating bits.

State machine's are written in VHDL, but they can be more easily visualized through diagrams. One of the most popular ways of showing a state diagram is through an "Algorithmic State Machine" (ASM) diagram. On the graph, each state is described by a rectangular block, and each decision by an oval block. The arrows indicate the change in states, and the blocks contain the signals that are modified in that state. Although ASM diagrams make the design easier to implement, they are not too useful for communicating the overall picture of the state machine. For this purpose, we have created a state transition diagram, shown on appendix A.

4.5 Serial Port Communication

Our proposed design is geared for use in embedded systems. The generated bitstream can directly be interfaced through an on-chip bus, and can be made available to the microcontroller. During the development, we need to control it from a computer in order to initiate each cycle of the number generation, but also to collect the output for later analysis.

We have decided to use the serial port for this communication. This choice was mostly dictated by our development board, since serial communication is the only one natively supported (a parallel port extension was also available). The other advantage of this selection is the ease of use. We managed to find VHDL modules that we could integrate into our code.

We built the serial port connection using an UART module found on [Ope] . The module hides all the internal complexities of RS-232 communication, and makes it available through a simpler interface. The clock runs at half the speed of the built-in clock, 50 MHz. There are two registers for the receive and the transmit buffers, and two control signals (transmit-ready and received) for checking whether the buffers are ready for the next cycle.

4.6 PC Interface Perl Script

On the PC side, we wrote a Perl script for communicating with the board. The job of this script is to send a signal to the board to let it generate each random number, and return it to the computer. The development environment was a Windows XP machine, and we therefore had to get Cygwin [Cyg] to run Perl properly.

The Win32::SerialPort module was used for communicating with the board. It supports a very intuitive interface. We first configured the serial port connection parameters such as connection speed and parity bits. The two functions read and write are used to receive and transmit Perl characters strings over the connection.

4.7 Placement and Routing

For the PUF circuit to work effectively, the paths formed by the switch chain must have even delays. The race condition would otherwise not be possible; one of the paths would always be slower, introducing a bias to the output. On FPGAs, the design tool is responsible for the placement, that is, for deciding where to put the logic. Such decisions are usually made for effectively using the chip area. Once the blocks have been placed, the internal connections are connected during the synthesis step known as routing.

We needed to manually fine-tune the placement and routing phases in order to maintain even paths. The VHDL language has several constraining commands for selecting the precise location where each block will be placed. The blocks first need to be grouped together using relative coordinates such as "block A will be located above block B." The big group containing the multiplexer switches can then be placed using absolute positioning. We lined up the blocks vertically starting from the bottom-left corner of the FPGA.

When the blocks were placed with even distances between them, the routing done by the synthesis tool was adequate and did not need any tweaks.

To make sure that the two paths were of equal length, we ran a simple bias test which counts the number of zeros and ones in a random bitstream. If the paths are indeed equal, the number of ones and zeros in a large set of output data should be equal. In the tests we have performed, we have obtained very good bias results (49.82% and 50.18%), indicating that the paths are not biased.

4.8 Sensitivity

The design is very sensitive to the placement of the state machine circuitry. When side circuit was close, the outputs we got were not as predictable. However, when we isolated the PUF from the rest of the circuit, we got better results when trying to model the behavior. We can explain this by the

fact that the pulses are easily influenced with surrounding circuitry. If there are surrounding CLBs that are switching while the pulses are racing, the cross talk can affect the paths. Making sure that there are no other usable slices in the region reduces the error rate when the PUF is run by itself. For the purpose of the random number generator, we did not isolate the circuit so that we would get less predictable results.

4.9 Post-Processing by XORing

In order to further increase the result of the statistical tests that we ran, we performed some post-processing on the data. We managed to improve the results of the NIST tests dramatically by using an XORing circuit.

In the current scheme, a stream of bits is generated by the arbiter flip-flop. As explained in section 4.3, some of these bits can be random while others not. Ideally, we would have wanted to somehow guess which ones are random, and only send those to the output. Realistically, making such a decision is impractical, but we can achieve the same effect by using a digital trick.

XOR is a binary operation that evaluates to a '1' when the two input bits are different, and '0' when they are the same. Our post-processing scheme is built on XORing bits this way by groups of 8 bits. Out of this group, it suffices to have one random bit to give a result that is also random. Although this process increases the randomness results, it also decreases the throughput of the generator. It effectively slows it down eightfold, since it takes eight times the time to generate one bit.

We applied this method using a Perl script on the output files. Implementing it on the circuit would not add too much overhead either. By having one XOR gate and one flip-flop, the operation can be implemented serially. Each time a new bit is generated, it is XORed with the previous value of the flip-flop, and simultaneously saved for the next cycle.

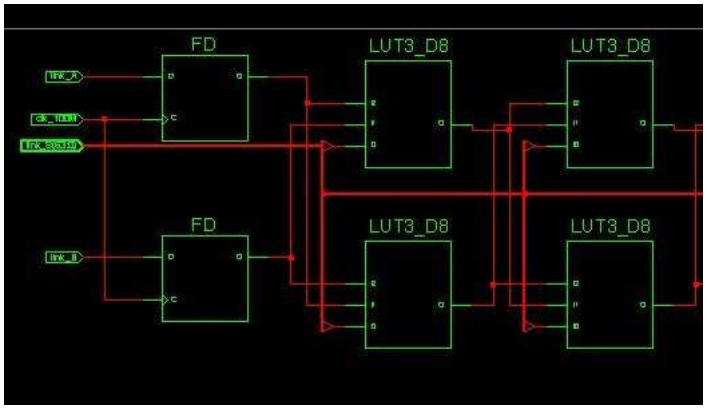


Figure 6: Beginning of switch chain as a Technology Schematic

5 Implementation/Results

We have implemented our design on a Xilinx University Program (XUP) Virtex-II Pro development system that has a XC2VP30 FPGA. This section will present the implementation of the system, and provide statistical tests for showing that the results are random.

5.1 RTL Diagrams

This section contains screenshots from the Xilinx ISE tool describing the various parts of the design. The configuration of the switch chain is shown on Figure 6. The two initial flip-flops are used for sending the pulses, and each Look-Up Table (LUT) implements half of the switch block. The outputs of each block is sent to the next level.

The arbiter can be observed on Figure 7. The entire switch chain is modeled using a single block on the left of the picture, and its two outputs, Q_a and Q_b are connected to the flip-flop.

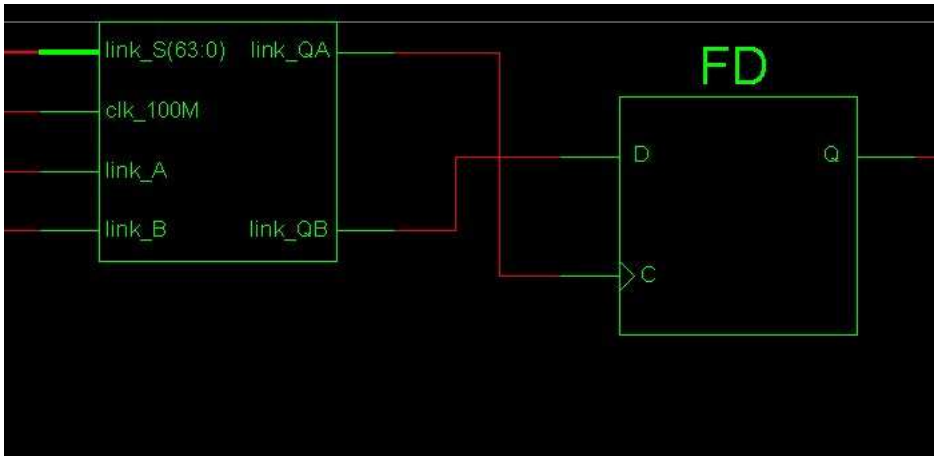


Figure 7: End of switch chain and arbiter

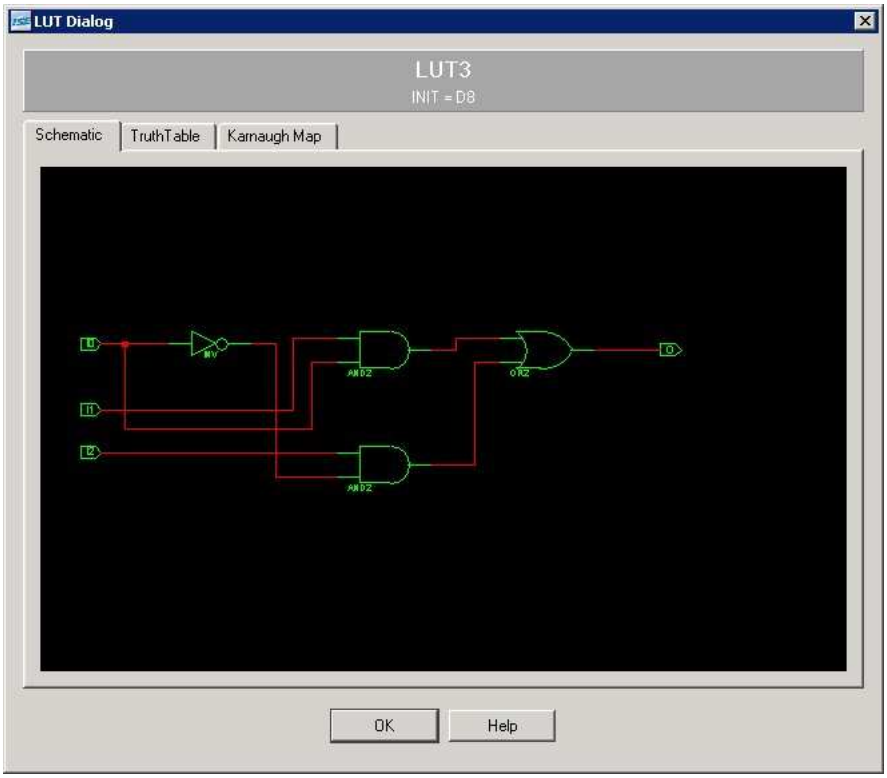


Figure 8: Schematic of the LUT multiplexer function

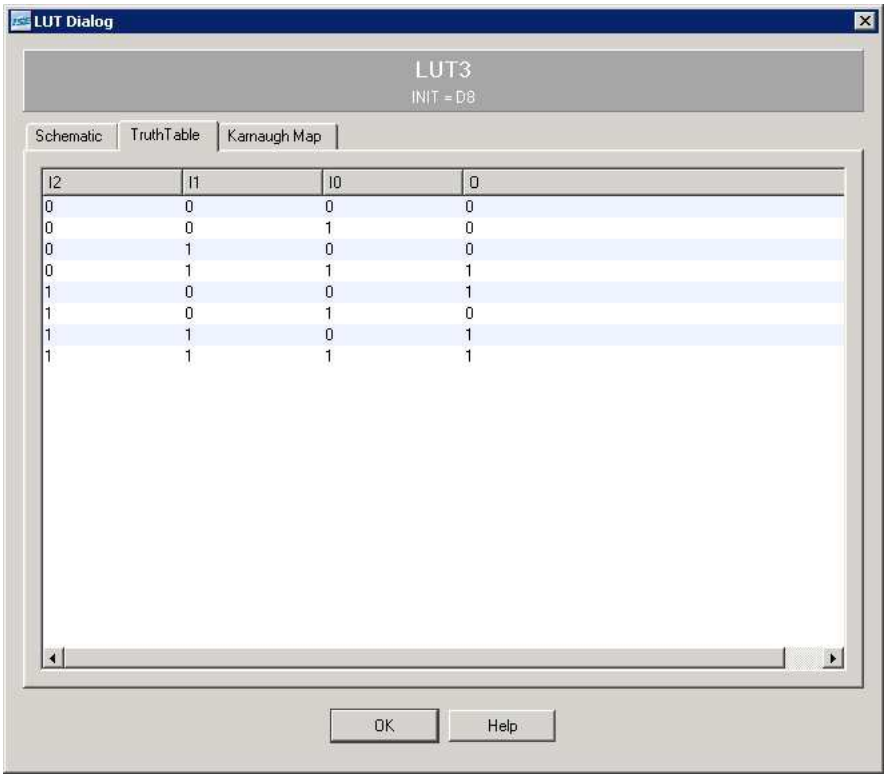


Figure 9: Truth Table for LUT switch function

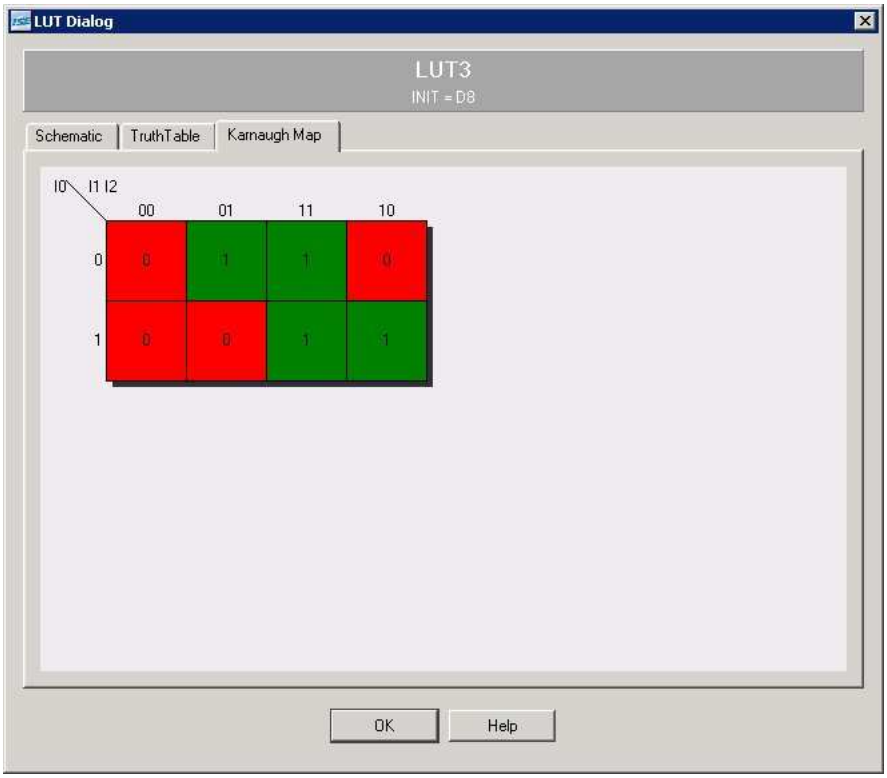


Figure 10: Karnaugh Map for the LUT switch function

5.2 FPGA Editor Views

The screenshots in this section help to explain the structure of the circuit as placed by the Xilinx tool. They have been obtained from the FPGA editor application that is a standard component of the toolkit.

Figure 11 shows the placement of the switch blocks in the FPGA CLBs. The CLBs containing the blocks are highlighted in red. It can be seen that they are placed vertically. An overall view of the zoomed out FPGA is shown on Figure 12. The chain occupies the center of the chip, and almost covers its entire length. Other circuitry, such as that of the state machine is interspread between the blocks.

The FPGA editor also allows us to zoom into the slice to show the routing of the signals inside the chip. Figure 13 shows how this routing is performed for a switching block containing two multiplexers. It can be seen that both of the Look-Up Tables (LUTs) are used for the switching function, but the flip-flops are bypassed. The output signals are then routed to the next block.

The same close-up picture is shown for the arbiter on Figure 14. The two inputs are directly connected to one of the flip-flops found in the slice, so they can be sampled at their output.

Figures 8 to 10 show the content of the LUTs using various methods.

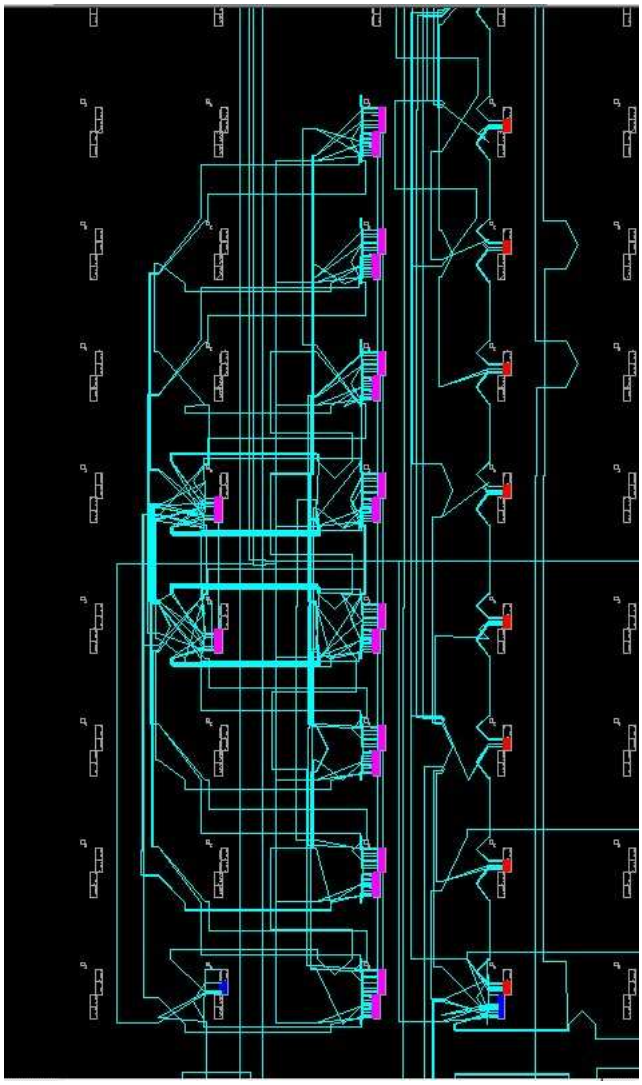


Figure 11: View of the switch chain (block in red)

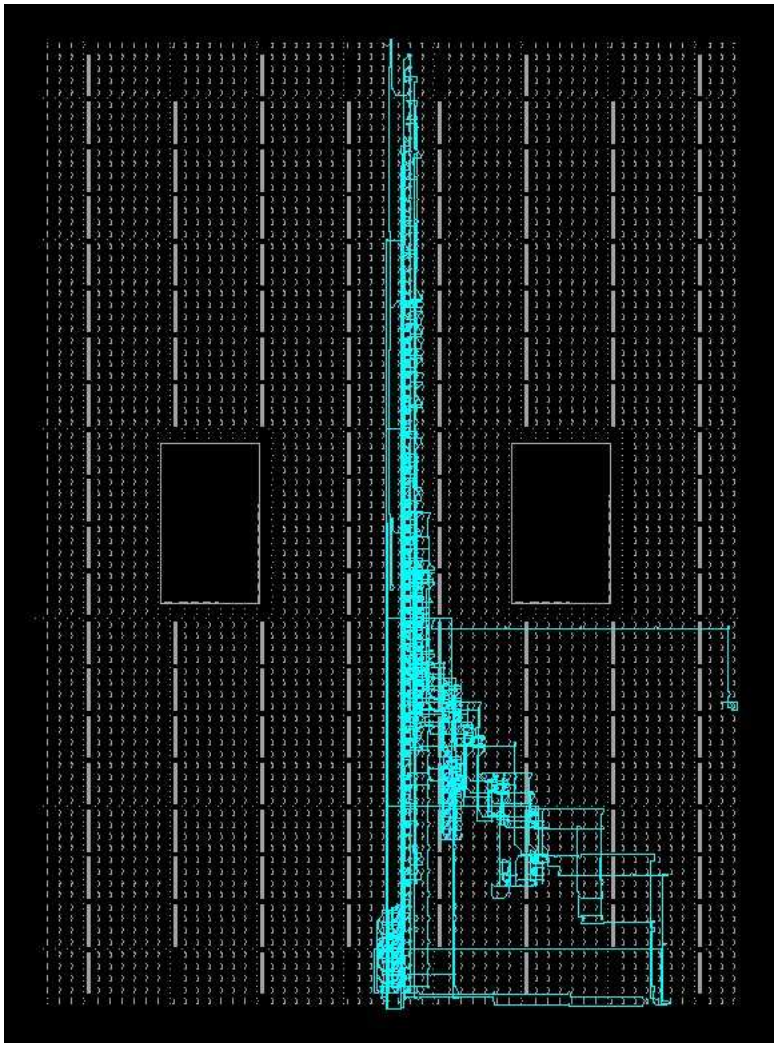


Figure 12: Overall FPGA chip

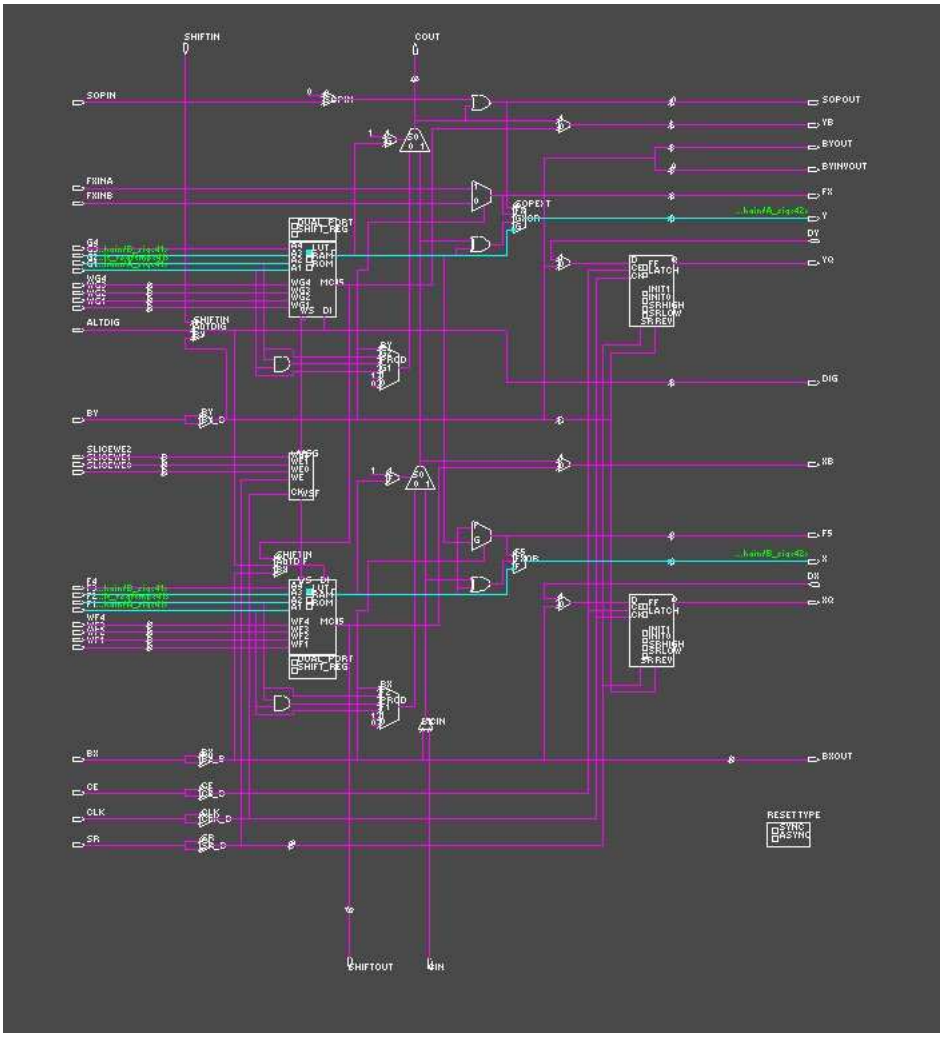


Figure 13: Content of slice containing switching block

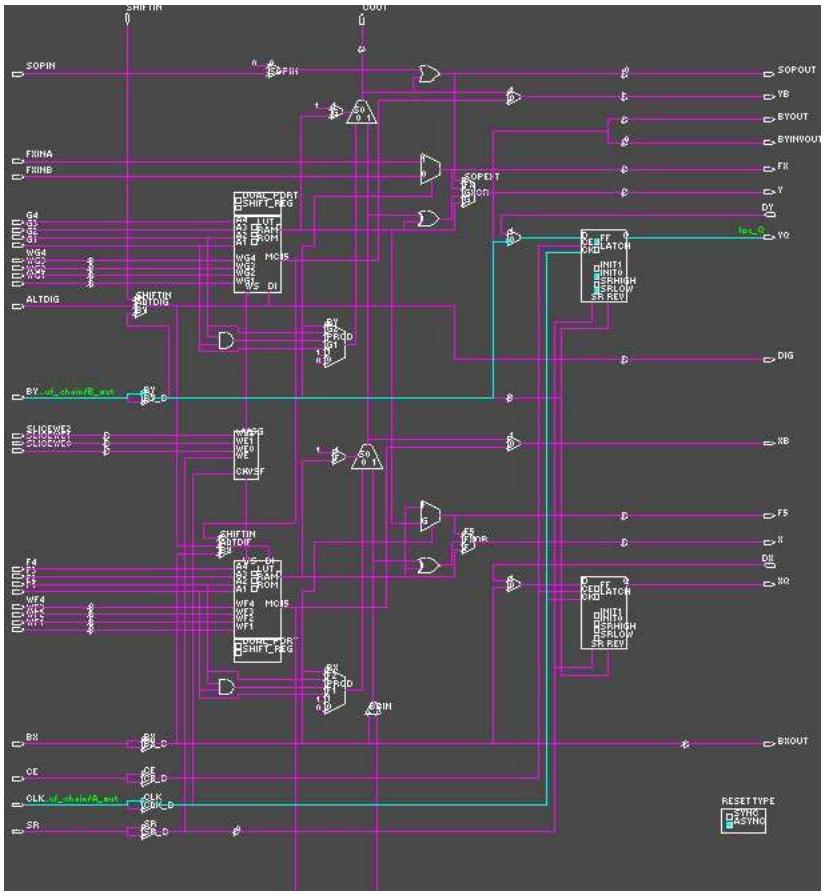


Figure 14: Content of slice containing arbiter

5.3 NIST Test Results

The output of the NIST test suite for our final design is shown on figure 15. The output shows that the random number generator passed almost all the tests. The failed tests are marked with an asterix. On this test run, only four of the tests failed to prove the null hypothesis.

Another output file of the test is shown on figure 16. This file shows the frequency of ones and zeros in the output, for each of the bitstreams. It can be seen that there almost as many ones as zeros, which is an additional sanity check to show that the system behaves as expected.

```
---------------------------------------
  P-VALUE  PROPORTION  STATISTICAL TEST
---------------------------------------
  0.637119   1.0000    frequency
  0.213309   1.0000    block-frequency
  0.964295   1.0000    cumulative-sums
  0.834308   1.0000    cumulative-sums
  0.090936   1.0000    runs
  0.000000 * 1.0000    longest-run
  0.162606   1.0000    rank
  0.162606   1.0000    fft
  0.035174   0.9750    nonperiodic-templates
  0.213309   0.9750    overlapping-templates
  0.000000 * 1.0000    universal
  0.122325   0.9750    apen
  0.585209   0.9750    serial
  0.788728   0.9750    serial
  0.000000 * 1.0000    lempel-ziv
  0.739918   0.9250 *  linear-complexity
- - - - - - - - - - - - - - - - -- - - - - - - - - - - - -
The minimum pass rate for each statistical test with
the exception of the random excursion (variant) test
is approximately = 0.942804 for a sample size = 40
binary sequences.
```

Figure 15: Output of the NIST test suite

```
------------------------------------------------

FILE = run1_80_xor_by_8_40bs.dat   ALPHA = 0.0100
------------------------------------------------

BITSREAD = 20000 0s = 10073 1s = 9927
BITSREAD = 20000 0s = 9852 1s = 10148
BITSREAD = 20000 0s = 10001 1s = 9999
BITSREAD = 20000 0s = 10013 1s = 9987
BITSREAD = 20000 0s = 10132 1s = 9868
BITSREAD = 20000 0s = 10032 1s = 9968
BITSREAD = 20000 0s = 10064 1s = 9936
BITSREAD = 20000 0s = 9968 1s = 10032
BITSREAD = 20000 0s = 9889 1s = 10111
BITSREAD = 20000 0s = 9968 1s = 10032
BITSREAD = 20000 0s = 9878 1s = 10122
BITSREAD = 20000 0s = 10092 1s = 9908
BITSREAD = 20000 0s = 9875 1s = 10125
```

Figure 16: Bias output of the NIST test suite

6 Future Work

A suggested future work would be to test the design at different temperature levels to see how the circuit is affected. Normally variations in temperature should not affect the behavior of the circuit, since both paths are changing at the same time. However the randomness of the system may depend on the ambeint temperature.

The usefulness of the system can also be tested by using the circuit as part of a larger scheme, such as a cryptographic protocol. The bitstream can be interfaced by a larger state machine that can use the bits as part of a protocol.

7 Conclusion

We have shown that the Physically Unclonable Function made from switch-chains can alternatively be used as a hardware random number generator. Using the metastability in the arbiter, we built a prototype that is suitable for many applications, especially that of cryptographic authentication protocols. We have tested the output of the randomness, and have concluded that for most practical applications the system behaves unpredictably.

References

[Cyg] Cygwin Information and Installation. `www.cygwin.com/`. Accessed April 24, 2008.

[Die] Diehard Battery of Tests of Randomness. `http://www.stat.fsu.edu/pub/diehard/`. Accessed April 24, 2008.

[GCvDD02] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 148–160, New York, NY, USA, 2002. ACM.

[GCvDD03] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Delay-based Circuit Authentication and Applications. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 294–301, 2003.

[LDG+04] J. W. Lee, L. Daihyun, B. Gassend, G. E. Suh amd M. van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *Symposium of VLSI Circuits*, pages 176–179, 2004.

[LLG+05] Daihyun Lim, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srinivas Devadas. Extracting secret keys from integrated circuits. *IEEE Trans. VLSI Syst.*, 13(10):1200–1205, 2005.

[NIS] Nist Random Number Generation and Testing. `http://csrc.nist.gov/groups/ST/toolkit/rng/index.html`. Accessed April 24, 2008.

[Ope] OpenCores.org. `http://opencores.org/`. Accessed April 24, 2008.

[OSD04] Charles W. O'Donnell, G. Edward Suh, and Srinivas Devadas. Puf-based random number generation. Number 481, November 2004.

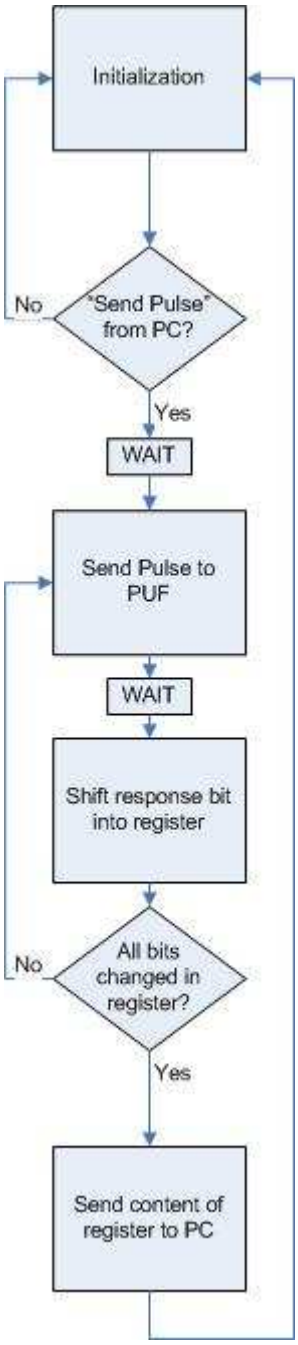
[Pos98] R. Posch. Protecting Devices by Active Coating. *Journal of Universal Computer Science*, 4(7):652–668, 1998.

[Rav01] Pappu Srinivasa Ravikanth. *Physical one-way functions.* PhD thesis, 2001. Chair-Stephen A. Benton.

[SMKT06] B. Skoric, S. Maubach, T. Kevenaar, and P. Tuyls. Information-theoretic Analysis of Coating PUFs. Cryptology ePrint Archive, Report 2006/101, 2006.

[TS06] P. Tuyls and B. Skoric. Secret Key Generation from Classical Physics: Physical Uncloneable Functions. In S. Mukherjee, E. Aarts, R. Roovers, F. Widdershoven, and M. Ouwerkerk, editors, *AmIware: Hardware Technology Drivers of Ambient Intelligence*, volume 5 of *Philips Research Book Series*. Springer-Verlag, Sep 2006.

A State Machine Transition Diagram


Initialization
"Send Pulse" from PC?
No
Yes
WAIT
Send Pulse to PUF
WAIT
Shift response bit into register
All bits changed in register?
No
Yes
Send content of register to PC