Literary Survey
True Random Number Generation in FPGAs

Adam Pfab
Computer Engineering 583

Random Numbers

Cryptographic systems require randomness to create strong encryption protection and unique identification. The strength of these protections resides in the non-predictability of moderate amounts of data. As the need for larger keys and more unique IDs grows, the need for random numbers also grows. This paper will summarize a survey of recent IEEE publications on the subject of True Random Number Generation in FPGAs.

Research

The publications presented in this survey were sourced from the IEEE website. The articles were refinement to the search terms “Random” and “FPGA”. They primarily comprise journals submitted from the years 2007-2010.

Problems in True Random Number Generation

Across the publications researched, two primary problems are identified in True Random Number Generation (TRNG). The first is a reliable source of randomness. The second is a proven method to generate entropy (the measure of "disorder" of a variable). A third subject was addressed in some papers, which was resource utilization.

The source of randomness was nearly uniformly identified as mining from analog sources. These analog sources are precisely the issue with FPGAs, which are digital in nature. By design, they are precise, uniform, and predicable.

Entropy was solved differently in each journal. Based upon the source of random data, different mechanisms were required to achieve a level of entropy desired. There was tight coupling between the two problems and their resolutions.

The third subject that some journals identified was that their particular solution provided and optimization from previous solutions (operating frequency, throughput, resources). While these statements were made, no paper explicitly addressed that the previous implementations were excessive or that previous implementations required redesign due to being too inefficient.

Sources of Randomness

Randomness is a compound issue. The primary issue is finding a source that provides variation. The second issue is that while a source may provide good variation, care must be taken to ensure that the source does not provide repeatable results.

Kwok and Lam [5] used a Xilinx Digital Clock Manager's (DCM) inherent jitter and coupled it with a high-frequency clock to generate an infinite stream of ones and zeroes. Specifically, they matched the jitter period of a slow clock with an accurate, high-frequency clock. For this configuration to work, a specific ratio of clock frequencies is required. The period of the fast clock needs to be the same order of magnitude to the jitter out of the DCM. The high-frequency clock was supplied to the D input of a D-Flip-Flop while the jitter-prone DCM output of a slower clock was supplied to the CLOCK input of that D-Flip-Flop.

This design is dependent upon two clock sources, both of which are external stimulus to the FPGA in their study (Xilinx Virtex II Pro). They took advantage of inherent limitations of the silicon itself and were able to generate data at the frequency of the slower clock rate (1 bit of data per clock).

While Kwok and Lam exploited a data source, they did not address the predictability of this source. Specifically, can the external sources or the environment (or both) be manipulated to result in a predictable output? Tsoi, Leung, and Leong [1] identify these, and more, limitations and propose an alternative.

Tsoi, Leung, and Leong proposed the use of a string of inverters that feed back on themselves to create a ring oscillation. Further, they propose, is that if a delay element can be incorporated and an additional XOR gate, that ring oscillation can be doubled, thereby halving the predictability on that ring. That ring oscillator feeds into the D input of a D-Flip-Flop, while a system clock feeds into the CLOCK input of that D-Flip-Flop. This design was implemented in 130 slices of a Xilinx XCV300-8 device and requires only one external source: a system clock.

They argue that this design is an improvement to Kwok and Lam due to the ability to generate data at the system clock frequency as opposed to the limitation of dependency of two clock ratios. This implementation also appears to resolve problems with respect to external manipulation. It removes the ability to adjust external stimulus and leaves only the predictability due to environment (temperature, power, process).

In contrast to these two clock-based sources, Cheung, Lee, Luk, and Villasenor [3] use pre-defined seeds. These seeds are entered into a Tausworthe Uniform Random Number Generator (URNG). This style of number generator inherently combines the seeds and shifts their outputs when not in reset. While it does not state that the shift registers are rotated at different frequencies, it would be assumed that by doing so with non-periodic clocks would result in better randomization.

This solution appears to solve a slightly different problem than the DCM solutions. The URNG may be excluded for use in random stream generation due to definable seeds and a predictable algorithm especially if the shift registers are (iso-)synchronous. Further, if the shift registers are not (iso-)synchronous, that implies three unique external clock sources which provide the weakness of manipulation.

Entropy Creation

While a random source is required for True Random Number Generation, it does not imply uniform distribution of ones and zeroes. Specifically, since TRNG is implemented digitally, the results are binary: ones and zeros. Additionally, it is statistically improbable that there is

precisely a 50% distribution. Therefore any given source is certainly biased towards generating ones or zeros. This then identifies a need to manipulate the data from the random source to provide the uniformity needed to provide the security that cryptography and identity rely on to complete their functions.

That said, Tsoi, Leung, and Leong attempt to prove that their ring oscillator provides sufficient distribution without further manipulation. They are careful, though, to point out that a specific implementation of the ring oscillator is required. To enforce good distribution, they have optimized the ring oscillator to take advantage of the Xilinx XCV300-8 architecture. This solution passed both NIST [7] and Diehard [8] random number testing. This was done using a very small amount of resources, which can be found in the table below.

Design	Period, ns	Slices (% XCV300)	BRAM (% XCV300)
PRNG	7.482	129 (4%)	4 (12%)

As you recall, Kwok and Lam used two clock sources and a D-Flip-Flop to generate a binary string of data at the frequency of the slower clock. They hypothesized three different randomizers to disassociate this stream. They chose to implement a parity filter in conjunction with a Strong fixing function (MD5). By experiment, they tested several combinations of different parity filter taps and MD5 compressions to determine an optimally-uniform output. Based upon testing a tap and compression pair was identified that passed NIST and DIEHARD test suites. This solution used a relatively small amount of resources, as can be seen in the table below

Module	DCM (% XC2VP20)	Slices (% XC2VP20)	BRAM (% XC2VP20)
TRNG	1 (12.5%)	8 (0%)	0 (0%)
MD5	0 (0%)	478 (5%)	1 (1%)

This solution requires the note that the result vectors are selected based upon test pass/fail criteria and not specifically pre-determined.

These two solutions apply probabilities to the random data source. The alternative that both Cheung, Lee, Luk, and Villasenor and Alimohammad, Fard, Cockburn, and Schlegel [4] is data permutation (via Gaussian Distribution).

Recall that both of these journals used pre-defined seeds and Tausworthe URNGs as their data source. They further generate random data by sourcing the URNG outputs into an address decoder. The address decoder is a complex circuit that comprises a barrel shifter, adders, and look-up tables similar to an SBOX. The output of the address decoder then feeds into a polynomial equation thus resulting in a random number. The table below identifies the resources needed to implement this entropy on a XILINX VIRTEX-4 XC4VLX100-12 FPGA.

component	slices	block-RAMs	DSP-blocks
	variable B_{x_1} - degree-1		
Tausworthe URNG	141	0	0
address decoding	268	0	0
polynomial evaluation	134	2	2
total	543	2	2

These random number generators were not tested against the NIST or Diehard test suites.

Similar to the previous Gaussian solution, the Mersenne Twister [6] utilizes data manipulation to generate a random distribution of ones and zeroes. The paper wasn't clear about what a Mersenne Twister actually does; instead it focuses on how to translate the design from software to hardware. The software algorithms were mapped to a Xilinx Virtex4 VFX100 FPGA with the following resources.

Slices	128
FFs	193
LUTs	213
BRAMs	4

Again, these random number generators were not tested against the NIST or Diehard test suite. Instead, the intent of the Mersenne Twister journal entry was to compare the efficiencies of a hardware implementation over a software implementation.

Conclusion

To create secure systems, random number generation must occur. Strong random number generation must be created using a good random data source and in conjunction with a mechanism to create an even distribution of data (entropy).

The journals studied for this paper included several random data sources as well as a strongly-coupled mechanism to further process that data. One common solution is to use truly random data sources with filtering as identified in [5] and [1]. Another common solution is to use less random seed data and strong permutations as identified in [3], [4], and [6].

References

[1] Tsoi, K.H.; Leung, K.H.; Leong, P.H.W.; , "High performance physical random number generator," Computers & Digital Techniques, IET , vol.1, no.4, pp.349-352, July 2007doi: 10.1049/iet-cdt:20050173

- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4271377&isnumber=4271368

[2] Sonnaillon, M.O.; Urteaga, R.; Bonetto, F.J.; , "Software PLL Based on Random Sampling," Instrumentation and Measurement, IEEE Transactions on , vol.59, no.10, pp.2621-2629, Oct. 2010doi: 10.1109/TIM.2009.2036459

- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5546971&isnumber=5571852

[3] Cheung, R.C.C.; Dong-U Lee; Luk, W.; Villasenor, J.D.; , "Hardware Generation of Arbitrary Random Number Distributions From Uniform Distributions Via the Inversion Method," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.15, no.8, pp.952-962, Aug. 2007doi: 10.1109/TVLSI.2007.900748

- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4276772&isnumber=4276770

[4] Alimohammad, A.; Fard, S.F.; Cockburn, B.F.; Schlegel, C.; , "A Compact and Accurate Gaussian Variate Generator," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.16, no.5, pp.517-527, May 2008doi: 10.1109/TVLSI.2008.917552

- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4476028&isnumber=4490005

[5] Kwok, S.H.M.; Lam, E.Y.; , "FPGA-based High-speed True Random Number Generator for Cryptographic Applications," TENCON 2006. 2006 IEEE Region 10 Conference , vol., no., pp.1-4, 14-17 Nov. 2006doi: 10.1109/TENCON.2006.344013

- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4142319&isnumber=4142121

[6] Xiang Tian; Benkrid, K.; , "Mersenne Twister Random Number Generation on FPGA, CPU and GPU," Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on , vol., no., pp.460-464, July 29 2009-Aug. 1 2009doi: 10.1109/AHS.2009.11

- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5325420&isnumber=5325404

[7] National Institute of Standards and Technology, agency of the U.S. Department of Commerce

- http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html

[8] The diehard tests are a battery of statistical tests for measuring the quality of a random number generator. They were developed by George Marsaglia over several years and first published in 1995 on a CD-ROM of random numbers

- http://www.stat.fsu.edu/pub/diehard/