Aaron Mills
Literary Survey 2011
Reconfigurable Computing
CPRE 583

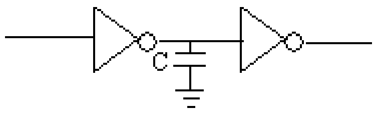
Literary Survey: Design of Physically Unclonable Functions using FPGAs

A common mechanism used in the process of securing computer systems is the cryptographic hash function. This is a so-called one-way function. A few properties of a good hash function follow.

- Given only an output and the hashing mechanism, the only way to determine the input should be to try every single possible input until you see that output.
- The size of the input space should be so large that guessing should take "a very long time".
- The size of the output space should be large enough that the chance of more than one input mapping to the same output should be very small.

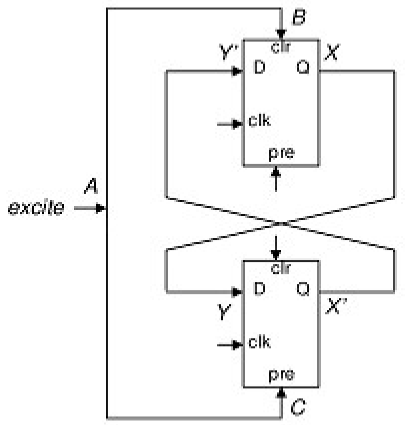
A Physically Unclonable Function (PUF) is effectively a hardware hashing function. It takes an input, or challenge, and provides an output, or response. However, there is an additional property involved: each PUF employing the same algorithm should provide a unique response to the same challenge. In other words, each PUF has a "unique signature" which makes it ideal for use in, for example, RFID cards for authenticating people [3]. In some sense this behavior is like the "salt" that is sometimes factored into cryptographic hash functions. Normally if a user uses the same password on two Linux machines, the stored hashes will be identical, since the same algorithm is used. This invites an attacker to utilize precomputed common input-output pairs, and just look up the hash to get the password. Adding a salt to the hash function breaks this technique since the salt causes the hashes on each machine to become unique.

A delay-based PUF obtains this "uniqueness" property by exploiting inconsistencies inherent in the fabrication of integrated circuits. Consider the very simple circuit segment below:



The capacitor C is inherent in the circuit design due to the parasitic capacitance in the layout of the inverter transistors, as well as the interconnects between them. However, when fabricated, each copy of the circuit above will have a slightly different value of C due to small variations in transistor sizing and configuration. The result is a slight difference in transmission delay. So if the PUF can create a race condition between two signals, for a given chip the same signal may always finish first. But between several chips, it is very hard to predict which signal will win. Back in the security realm, an attacker can attempt to clone the circuit or observe its behavior, but this will not help them "crack" somebody else's response set.

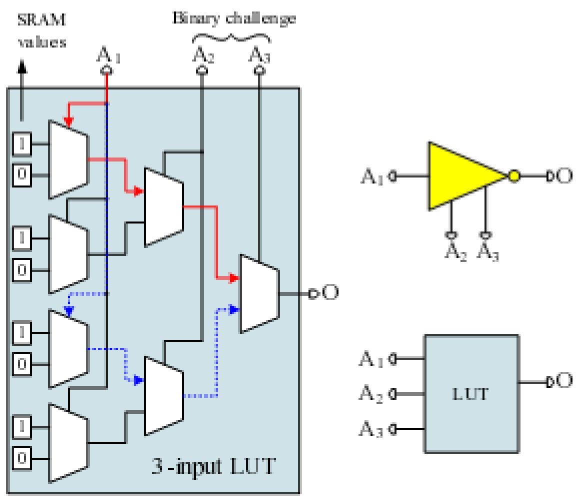
The aforementioned delay-based PUF is just one of a variety of known techniques. However, when using FPGAs, many of the more exotic techniques (such as the optical PUF) are ruled out as they require specialized hardware or fabrication processes. Assuming a delay-based PUF is to be employed, another constraint that designers face is that (compared to designing an ASIC) he or she has little control over how the circuit is physically routed. For example, a study [1] demonstrated that the following "Butterfly PUF" design is ineffective in an FPGA.



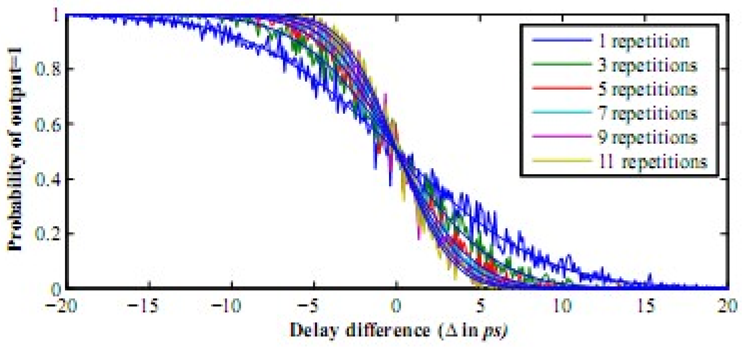
One latch is put into preset mode, while the other is put into clear mode by an excitation signal (ie. one bit of the challenge vector). This creates a transient unstable state with a final state that depends solely on process variation in the interconnects. However, it was discovered that the variation in the signal paths chosen by the synthesis tools has a far greater effect on the outcome than the random process variations. In other words, the static configuration created by synthesis tools is not "symmetric" enough to create an ideal race condition. Furthermore, the standard "Arbiter PUF", which sets up two identical gate delay paths and monitors which of the two "wins the race", suffers from the same problem. This study emphasizes the fact that an understanding of how a PUF design maps into the FPGA is essential to producing a good PUF.

One technique for overcoming this issue is to create a "tunable" circuit that compensates for routing mismatches, as is done in [2]. A high-resolution (pico-second) programmable delay logic

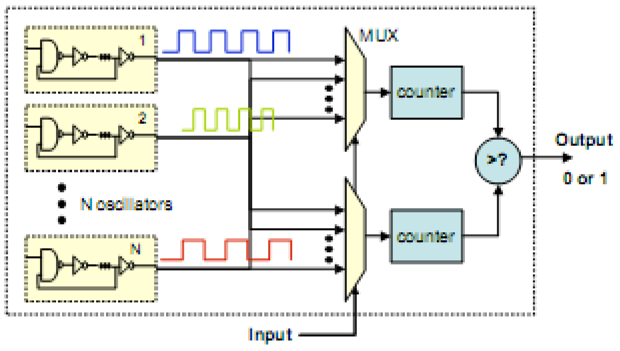
(PDL) is created which allows the paths in an Arbiter PUF to be compensated for routing variations. A standard FPGA LUT is employed as shown to create a programmable delay:



Logically, the device is a simple inverter for A1, with A2 and A3 acting as don't-cares. However, the latter two signals control the signal path, and thus the signal delay. Of course, a drawback of this method is that a tuning phase is required to ensure that all paths have similar delays. The experimenters applied a fixed challenge, and adjusted the path delays until, for any given response bit, the probability of seeing a 1 was estimated to be equal to seeing a 0. Presumably, as long as the same circuit is placed on the same type of chip consuming the same set of gates, the tuning will remain valid. One of their charts in particular demonstratively   the theory that, as long as the difference in path delays is very close to zero (and, in this case, the challenge is repeated a number of times with a majority-rule vote), the value of a given response bit will be 1 or 0 with probability 0.5.



A second technique is to design circuits that do not depend on perfect routing symmetry. An example that has probably seen the most success so far is the "Ring Oscillator" PUF. This circuit is described in [3].



Essentially it involves an array of inverter-based ring oscillators that will each oscillate with a slightly different frequency due to process variation. Each oscillation edge increments a counter, and the counts are compared at some point to determine the winner. It is the counter that effectively amplifies the difference in frequency and cancels out any routing differences between oscillators. Drawbacks of this circuit are that it is relatively power hungry, and that it requires a lot of silicon real-estate in order to produce a useful number of response bits.

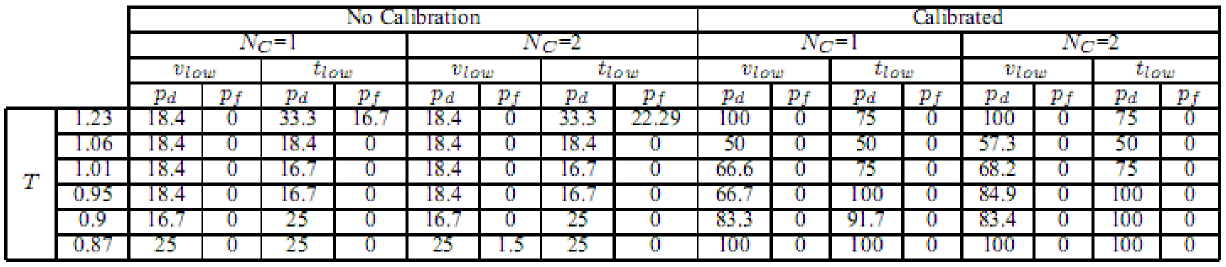
One last significant problem that is faced by all delay-based PUF designs is the effect of temperature on the physical properties of a circuit. For example, copper increases in resistance with temperature. Assuming an RC time-constant delay model, this also increases signal delay. This may not seem like a significant problem in, for example, environmentally-controlled datacenters, but certainly creates a challenge for devices meant to be portable.

In [3] the effect of temperature is partially canceled out due to the fact that it is not the absolute oscillator frequency that matters, but rather the relative frequency. Furthermore, if each oscillator frequency is configured sufficiently far away from the rest, the probability of two oscillators swapping levels in the frequency hierarchy is lowered. Coming from another perspective, the comparison pairs can be carefully chosen to ensure this property holds true.

In [2] the set of challenges is divided into "robustness" sets. Specifically, the set of challenges whose responses does not vary across a range of programmed delay values is considered highly robust, and experimental data shows that the error rates (ie probability of the same challenge producing different responses on the same chip) is much lower. The suggestion is that, if some particular challenges are avoided, the response can be more reliable.

A number of approaches use a "forgiveness" model to correct for errors as suggested briefly in [3]. In [4] a threshold is established in terms of the Hamming distance between the expected response and the actual response (ie in an authentication scenario). As long as the response falls within a particular set of values, the device is authenticated. Of course, the drawbacks are that the potential response set is reduced, and there is a small probability for a false negative during authentication.

Finally, in [5] an onboard calibration system is used. The clock pulse width is adjusted based on on-chip temperature and voltage readings which are available on most modern FGPAs. However, a device-specific constant has to be determined for each device in order to calibrate the relationship between the chip temperature and the pulse width adjustment. More realistically the author suggests determining the constant for a large set of devices, and then averaging them to create a universal constant. In the paper the authors suggest that even in this case, providing temperature and voltage compensation decreases the intra-chip response variation significantly.



		No Calibration								Calibrated							
		N_C=1				N_C=2				N_C=1				N_C=2			
		v_{low}		t_{low}		v_{low}		t_{low}		v_{low}		t_{low}		v_{low}		t_{low}	
		p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f	p_d	p_f
T	1.23	18.4	0	33.3	16.7	18.4	0	33.3	22.29	100	0	75	0	100	0	75	0
	1.06	18.4	0	18.4	0	18.4	0	18.4	0	50	0	50	0	57.3	0	50	0
	1.01	18.4	0	16.7	0	18.4	0	16.7	0	66.6	0	75	0	68.2	0	75	0
	0.95	18.4	0	16.7	0	18.4	0	16.7	0	66.7	0	100	0	84.9	0	100	0
	0.9	16.7	0	25	0	16.7	0	25	0	83.3	0	91.7	0	83.4	0	100	0
	0.87	25	0	25	0	25	1.5	25	0	100	0	100	0	100	0	100	0

Pd and Pf are the probability for detection (ie the device authenticates when it should) and false alarm (the device authenticates when it should not), respectively. vlow and tlow are low voltage and low temperature conditions, respectively. T is clock pulsewidth.

Resources

[1] Sergey Morozov, Abhranil Maiti, and Patrick Schaumont, An Analysis of Delay Based PUF Implementations on FPGA, ARC 2010
[2] Mehrdad Majzoobi, Farinaz Koushanfar, Srinivas Devadas, FPGA PUF using Programmable Delay Lines, WIFS 2010
[3] G. Edward Suh, Srinivas Devadas, Physical Unclonable Functions for Device Authentication and Secret Key Generation, DAC 2007
[4] Zdenek (Sid) Paral and Srinivas Devadas, Reliable and Efficient PUF-Based Key Generation Using Pattern Matching, HOST 2010
[5] Mehrdad Majzoobi, Farinaz Koushanfar, Time-Bounded Authentication of FPGAs