WP197 (v1.0) June 30, 2003

CipherStream Protocol—How CoolRunner-II CPLDs Protect FPGA IP

By: Jesse Jenkins

It doesn't usually take very long to create an FPGA design. Recently, however, a Xilinx competitor ran an ad declaring that while an FPGA can take up to a year to design, it can be cloned in only a second. Are FPGA designs really that insecure? While the ad seems absurdly hyperbolic, it *is* true that the bitstreams of some volatile FPGAs can be cloned. While it's unlikely that cloning could happen in "a second," fears about the insecurity of design efforts are valid ones. To alleviate these anxieties, this white paper will show you how to substantially secure the bitstream and the overall design of FPGAs using Xilinx CoolRunner™-II CPLDs. It will not particularly address Virtex™-II and its successors because they already employ Triple DES bitstream encryption, which is considered by many to be sufficiently strong encryption to deter IP theft. It will rather focus on volatile FPGAs in general, including both Xilinx FPGAs and competitors' FPGAs within its scope.

The Problem

Figure 1 shows a common way to download an FPGA from a standard EPROM, using a CPLD as the EPROM controller. The left hand side shows bitstream

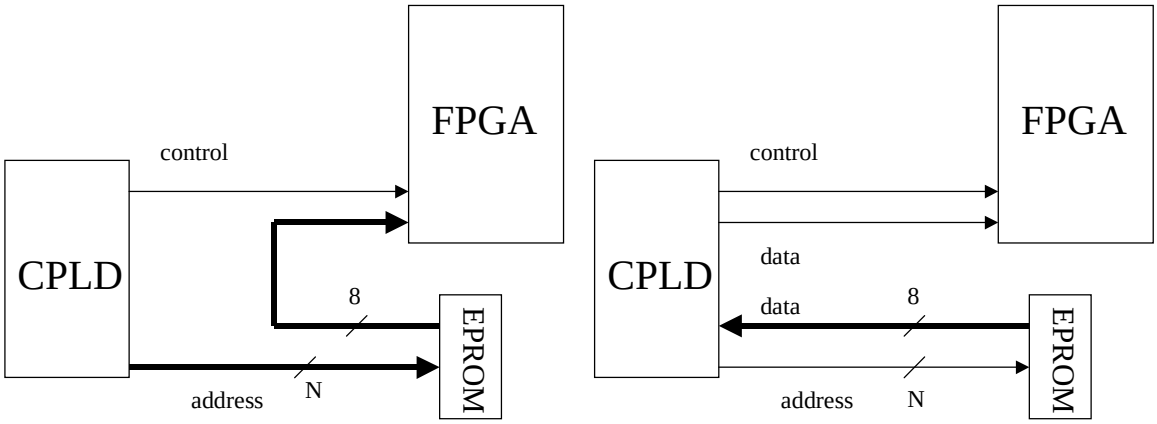

Figure 1: **Parallel Configuration (Left) and Serial Configuration (Right)**

delivery directly from the EPROM into the FPGA, with the CPLD managing the addressing and delivery of control strobes to the FPGA. The right hand version has the CPLD again managing the data, but in this situation it also serializes the data and may even drive configuration through the JTAG port. The "one second" clone would be simply copying the EPROM, which in reality can take a lot more than one second. So, the task is to make it harder to "clone." Note that the EPROM size and CPLD capacity will be chosen to provide sufficient capacity to satisfy the FPGA needs as well as other needs within the system. There may be multiple FPGAs chained for bitstream delivery, so Figure 1 is really a talking-point diagram and may be a substantially simpler configuration than many of today's systems actually use.

First Step — External Bitstream Encryption

To thwart cloners, we recommend encrypting the EPROM bitstream with software before programming the EPROM. Then, while the CoolRunner-II loads the FPGA, it decrypts the bitstream at the same time. Cloners can't copy the CPLD and the EPROM contents will be gibberish.

CoolRunner-II CPLDs are nonvolatile, and have many of the key features needed to support both encryption and decryption. The nonvolatility means they power up with internal bits that can be interpreted as "keys" or "passwords," or initialization vectors, depending on your point of view. Indeed, volatility is at the heart of the security issue, and it is possible—to a degree—to convey the nonvolatility of the CoolRunner-II parts to the volatile FPGA parts, so that it will take a lot of time and money to clone the FPGA design. For conceptual reasons, this white paper will consider a fairly simple encryption / decryption method, and point you to other literature for more elaborate methods that have been shown to provide greater security strength.

The example considered here is the synchronous stream cipher, which has both strengths and weaknesses. For simplicity, we analyze a simple Linear Feedback Shift Register (LFSR), which is known to be able to calculate a pseudo-random number within its structure. Figure 2 shows a simple LFSR. This is not specifically maximal length, but shows the basic structure.

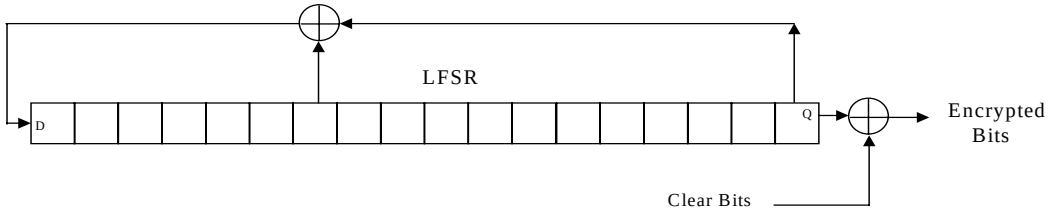

Figure 2: **Simple Linear Feedback Shift Register**

Most engineers have some exposure to LFSRs from logic design classes and possibly from communication coding theory. They are also frequently used to calculate checksums for data packets. LFSRs fall in the category of Linear Sequential Machines, which means they are constructed (typically) from D flip-flops and EX-OR gates. If initialized with a nonzero value, they will create a set of binary numbers within the flops, which will not repeat for a long clock sequence, depending on the location of the EX-OR gates and the number of D flip-flops. Identifying the feedback (aka "tap") sites to achieve the maximal length sequence is important. Xilinx has published a number of application notes regarding LFSR behavior, referenced at the end of this document.

One of the oldest ciphers (not particularly a stream cipher) is the One Time Pad, which appeared around World War I, and was subsequently shown by Claude Shannon to be a perfect cipher, with just a few caveats. Figure 3 shows a simplistic OTP.

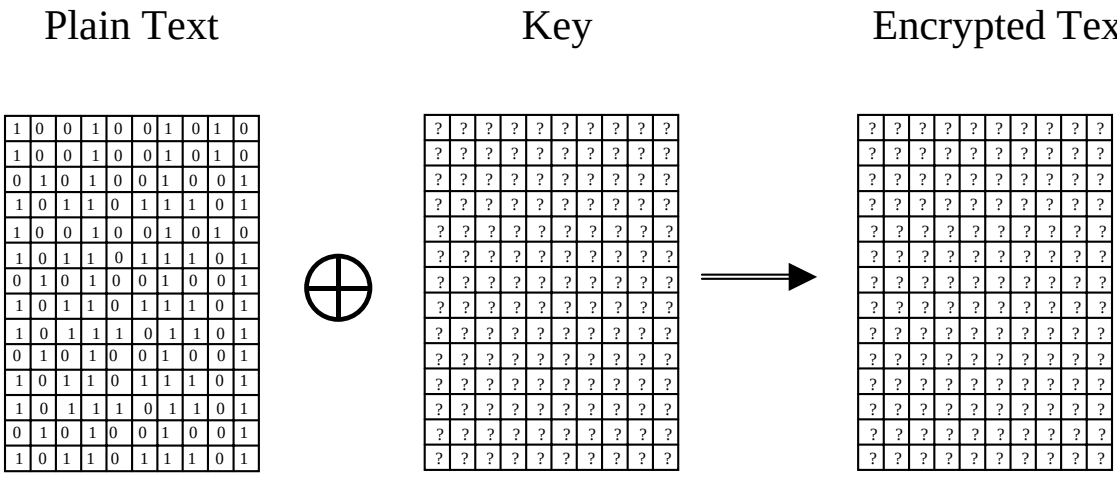

Figure 3: **Idea Behind a One Time Pad**

In Figure 3, we show the plain text (not encrypted) as a set of ones and zeroes. Then, we show the key as an array of question marks, as it is random and kept secret. When the plain text is EX-ORed with the corresponding key bits, the uncertainty of the result produces the array of question marks on the right. The requirements to be considered are the following:

1. The key length must match the message length
2. The key must be random

3. The key must not be re-used

Modern cryptographers have learned much, and tend to disregard the OTP as being particularly strong largely because the caveats are too tempting to violate. Long messages require long keys, which are a hassle in general. Obtaining a totally random key is also painful. Given the first two issues, there is a tendency to reuse keys. Nonetheless, for the sake of argument, let's suppose that we can fairly easily obtain a long random key, and we will only use it one time, with a given bitstream.

So, with that as a target, our first requirement is to create a random number that matches (or exceeds) the size of the target bitstream capacity. Table 1 summarizes some bitstream capacities for some Virtex I parts, as a reference. These values will help us select LFSR lengths so that we can create pseudo random numbers that are appropriately long and eliminate the risk of introducing repetition into what will become the key for our cipher. With this set of bitstreams as targets to secure, we will need to identify a set of LFSR structures that will have enough bits, and a way for us to create the state machines and know they won't repeat.

Table 1: **Some Virtex bitstream Lengths**

Device	# of Configuration Bits
XCV50	559,200
XCV100	781,216
XCV150	1,040,096
XCV200	1,335,840
XCV300	1,751,808
XCV400	2,546,048
XCV600	3,607,968
XCV800	4,715,616
XCV1000	6,127,744

LFSRs fall in two categories, internal feedback and external feedback. Both can create pseudorandom sequences of 2^n-1 patterns before repeating, if created with the correct primitive polynomial. Figure 4 shows the two types, and Table 2 gives the size (or degree) of the shift register and appropriate primitive polynomials.

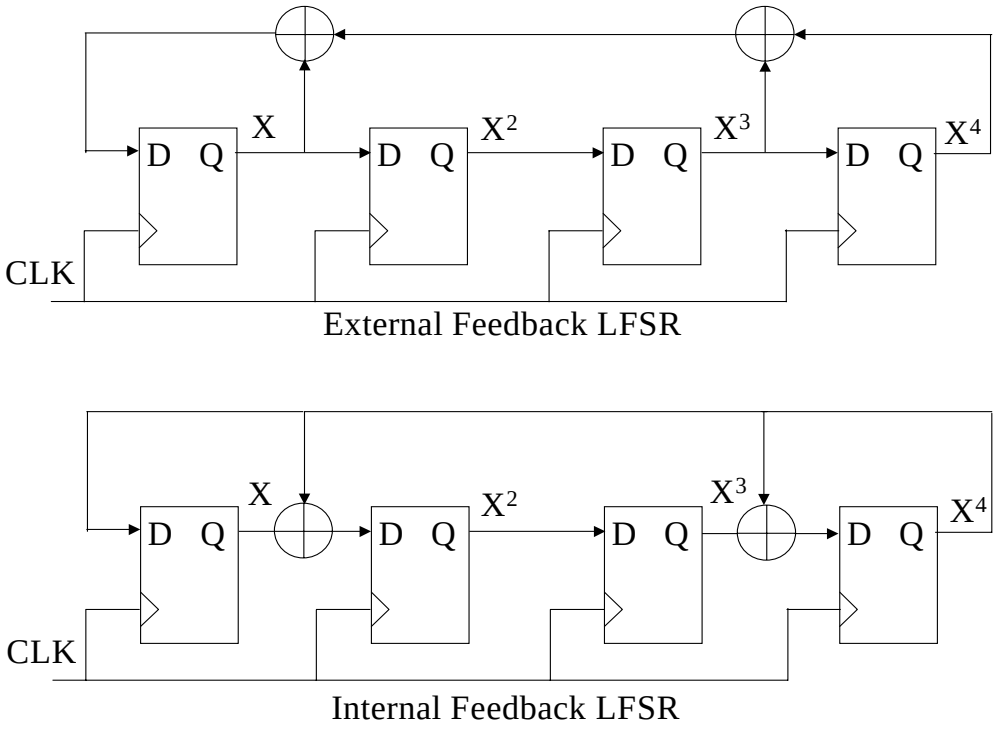

Figure 4: **External and Internal Feedback LFSRs for $P(X)=X^4+X^3+X+1$**

Note that in Figure 4, we do not show initialization circuitry. Typically either asynchronous set/reset circuits provide this, or it occurs from configuration bits within the CoolRunner-II CPLD during power-on initialization.

Table 2: **Some Primitive Polynomials and their Degree**

Degree(n)	Polynomial
2,3,4,6,7,15,22	$X^n + x + 1$
5,11,21,29	$Xn + X^2 + 1$
8,19	$X^n + X^6 + X^5 + X + 1$
9	$X^n + X^4 + 1$
10,17,20,25,28	$X^n + X^3 + 1$
12	$X^n + X^7 + X^4 + X^3 + 1$
13,24	$X^n + X^4 + X^3 + X + 1$
14	$X^n + X^{12} + X^{11} + X + 1$
16	$X^n + X^5 + X^3 + X^2 + 1$

Table 2: **Some Primitive Polynomials and their Degree *(Continued)***

18	$X^n + X^7 + 1$
23	$X^n + X^5 + 1$
26,27	$X^n + X^8 + X^7 + X + 1$
30	$X^n + X^{16} + X^{15} + X + 1$

The formula for the maximal length of the LFSR sequence given that it is derived from a primitive polynomial is simply $2^n - 1$, where "n" is the degree of the primitive polynomial, and we assume that it is initialized to a nonzero value. That being the case, here is how it can be used. Table 1 says that the bitstream for an XCV150 is 1,040,096 bits long. That means we need a primitive polynomial that is at least that long to create the pseudo-random number. A thousand (decimal) is a little less than 1024, which takes 10 bits to represent (0-1023). A million will take a little less than 20 bits to represent (1,048,576). That means the polynomial which has 20 bits will create a pseudo-random bitstream that is greater than 1,040,096 bits. Hence, choose n=20 and the corresponding polynomial would be: $X^{20} + X^3 + 1$. Figure 5 shows a corresponding LFSR. Of particular note is that it only takes a single EX-OR gate. In general, LFSRs that require minimal logic circuitry are attractive.

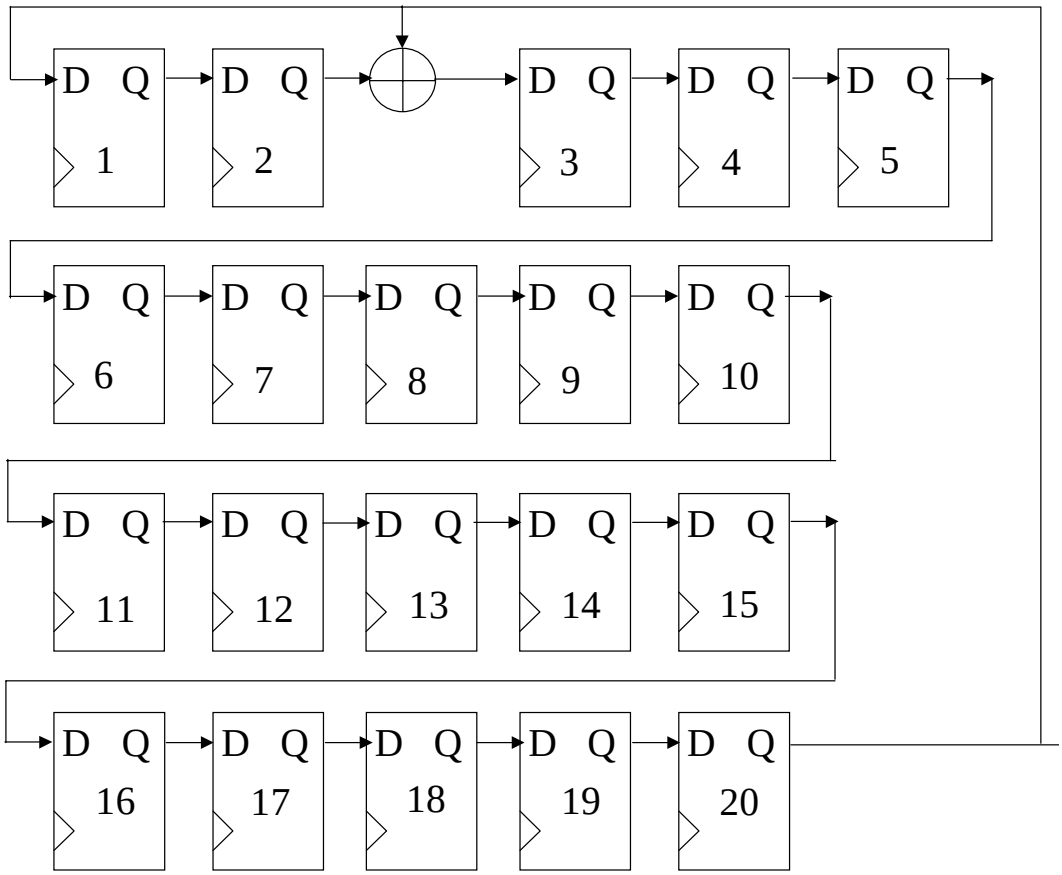

Figure 5: **Internal Feedback LFSR for Primitive Polynomial $X^{20} + X^3 + 1$ (clock and initialization signals not shown)**

The circuit in Figure 5 would take up the flip-flops for 20 macrocells, and would thus fit easily into the smallest CoolRunner-II CPLD. In CoolRunner-II CPLDs, each macrocell is comprised of a flip-flop and a set of logic driving it that can create a Sum of Products logic structure. Also contained within the macrocell is an EX-OR gate. The architecture is clustered into 16-macrocell function blocks, so this function would partially consume 1.25 FB. That leaves much logic available to form other functions, as needed.

Attacking the Bitstream Encryption

Now that we have constructed a proposal for using an LFSR to decrypt the bitstream, let's consider how it may be attacked by a "cloner." Just to reiterate, we are assuming that an FPGA design has been created and a corresponding bitstream produced. The bitstream was subsequently run through a software routine (typically), and has EX-ORed successive bits with an internal model of an appropriate LFSR. This resulted in an encrypted bitstream that is loaded into an EPROM.

It would be the developer's job to ensure that the bitstream calculation was done correctly, as was the creation of the appropriate file formatted for downloading (checksum calculations, etc.) When the EPROM contents are accessed by the CPLD for

subsequent delivery into the FPGA, they are EX-ORed with a hardware version of the LFSR, which produces clear bits for delivery to the FPGA.

How might this be attacked? There are several ways, but there are two that are most typical. First, an attacker could create a circuit capable of capturing the clear bits coming out of the CPLD. Figure 6 shows a simplified circuit for doing that.

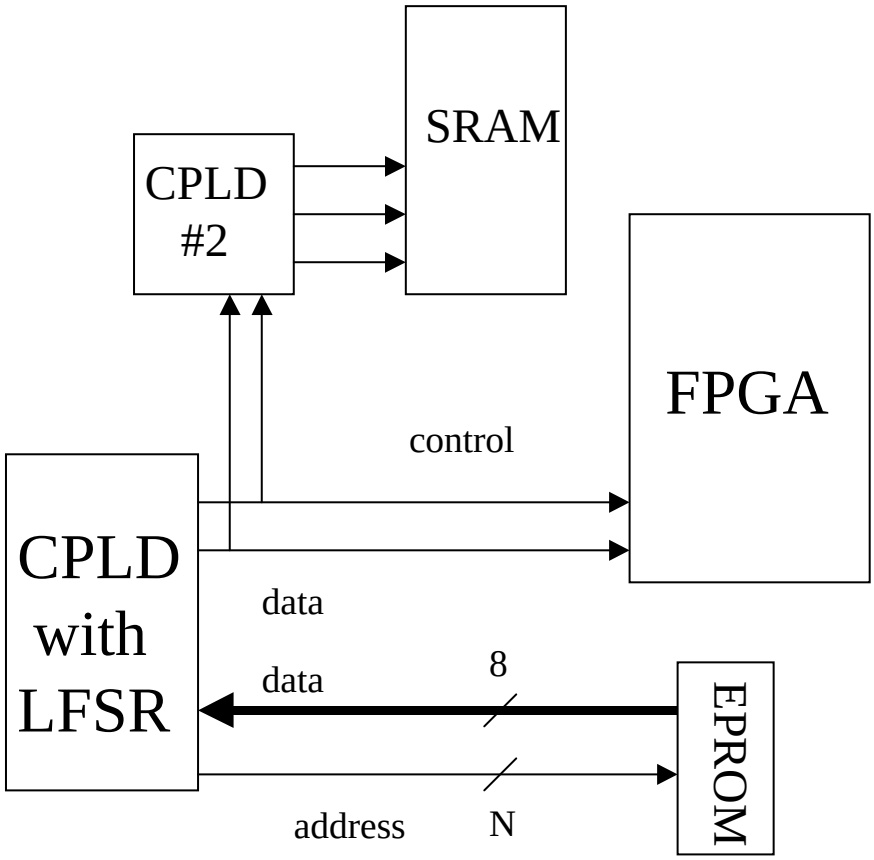

Figure 6: **Rough Circuit to Capture bitstream "On the Fly"**

In Figure 6, the CPLD with LFSR delivers its clear bits (serial or parallel) to the FPGA, while CPLD #2 contains a circuit that copies the bits as they pass by into a shadow SRAM. Details for CPLD #2 and the SRAM depth will depend on the bitstream being captured, but it will still require some serious training for the attacker to capture all the bits, in the right format, protocol, etc. Being off by a single shift out of a million plus bits won't work. Any noise, clock jitter, or any one of a number of other practical issues makes this a lot tougher than it looks. Then, the data would need to be formatted and driven into the FPGA to assure it works. It has its merit, but definitely takes more than a "second to clone."

The other general way to attack this would be to recognize in general the way the CPLD is working and to subvert it. If an EPROM with all zeroes is inserted into the EPROM position (remove chip and insert blank), then an interesting fact about the EX-OR function is discovered: A EX-OR 0 = A. That's right. A blank EPROM would divulge the pseudo-random number being created by the CPLD's LFSR. Simple LFSRs have had too much history! Being linear state machines, they have an impulse response. If you know that, you can convolve (logical convolution) the impulse

response with an input stream and predict the results. This has led to ways to identify the primitive polynomial very systematically, which means it can be cracked by a set of experiments.

There are other kinds of attacks. There are correlation attacks, linear syndrome attacks and a wide assortment of others. Luckily, there are also a bunch of circuits that have been developed for building stream ciphers that are much harder to crack. Those are described in an appendix, later.

Second Step — Simple Steganography

Steganography is basically hiding one message inside another. In our situation, we have identified a weakness of our LFSR—the blank EPROM attack. Although we will be recommending a more elaborate stream cipher solution, it won't hurt to add in protection against the blank EPROM.

Again, the idea is simple. The CPLD won't use up all its logic doing the stream cipher, so it makes sense to take out some insurance and add more circuitry to offer a change of response if being attacked. We could simply scan through the EPROM and if all addresses have zeroes, inhibit the delivery of the bitstream through the stream cipher. An attacker would then observe that we ran through all the addresses and didn't deliver anything to the output. That would work. Alternately, we could scan the EPROM and deliver a different but also pseudo-random pattern (say, from a different internal LFSR polynomial) to the FPGA. Doing so might even destroy the FPGA. This may or may not be a good thing.

A third approach would be to scan the EPROM and look for a specific set of bits at particular locations. This means that we would read the EPROM, and when certain addresses are created, we would compare what we found in the EPROM to internal copies of the same bits within the CPLD. If all of the targeted address contents matched the internal CPLD copies, the next round of addressing would be collecting the real encrypted EPROM data, and delivering it to the FPGA. The particular addresses we choose would constitute our "steganographic" authentication message: "Yes, this is a correct EPROM, so go ahead and decrypt it and load the FPGA."

At this point, we have done something that is suspicious: we have introduced a complete scan of the EPROM before we start delivering the decrypted data. We could add confusion here by using an LFSR to pseudo-randomly produce EPROM addresses, and again look for contents of specific address locations. If we did the initial scan pseudo-randomly, then did a linear addressing when we read out the encrypted data, an attacker could identify when we switch from a scan phase to a decrypt phase. So, if we chose to do the scan for correct EPROM data, then subsequently did a pseudo-random access of the EPROM for the decrypting, it would be harder for an attacker to discover what is going on. It also means the encrypted bitstream would have to be loaded into the EPROM file in an interesting order that might take a while to get right. But, it has merit for thwarting attackers.

Attacking the Steganography

It is difficult to discover the steganography, but not impossible. As described above, there are two distinct reading phases. The first, where the CPLD simply looks through the EPROM for certain items, must be done completely. If any addresses are skipped, then that narrows the search for what is being looked at. Make sure the CPLD examines every address within the EPROM, and doesn't take any action differently until everything has been looked at. Short cycling the search can result in ultimately divulging the decision points, which can be a weakness with regard to Differential Power Analysis (DPA), and Tempest attacks.

DPA means that an attacker could inspect the power-supply pins of the CPLD, observe its behavior when scanning a blank EPROM, scan its behavior with a valid EPROM, and compare addresses and data when there are differences. If the attacker can discover which addresses are being scanned for, he can then do a brute force attack on those addresses to find which pass the steganography test. If all addresses are looked at before a conclusion is reached, this makes the decision for multiple data items occur only once, and always at the end. It is best, however, even given that, to still make sure the key inspected data is scattered through the middle of the EPROM address space.

There are also other subtle techniques along these lines, such as using neural nets. Tempest is basically doing a similar inspection of the electromagnetic radiation coming off the CPLD. All of these attacks have merit. Luckily, they are also very hard to accomplish and sort out. See **References**, page 11 at the end for additonal information.

The Third Step—Repartitioning

As commented earlier, frequently CPLDs are on the board doing different tasks than loading the FPGA. They are programmable logic devices with their own characteristics and abilities, among which is relatively fast pin to pin speeds and high F_{MAX} operation. That being the case, their nonvolatility can be used in collaboration with the volatile requirements of SRAM based FPGAs. Design partitioning is basically separating aspects of a given design and distributing it among the various logic components within the system being designed. The recommendation here is simple. Inspect the FPGA design and identify a section of it that will be critical for the overall operation, but can actually be placed in the CPLD instead of in the FPGA. This may mean redefining the FPGA functionality, but here is a hint: FPGAs are extremely good at high speed, regular arithmetic operations, and those aspects frequently are best held within the FPGA. However, the control function for those aspects might be good operations to move into the CPLD.

Conclusion

We have discussed general techniques to make it hard for cloners to obtain FPGA bitstreams. We discussed combining three techniques—cryptography, steganography and partitioning—to raise the bar for cloners. The circuitry for each is fairly small, especially in light of the additional security they will bring to your system.

It would be possible to gain extra security by adopting just cryptography, or just partitioning (although the use of all of the three techniques together will keep your system best protected). If a key element of the FPGA is within the CPLD, and its security bits are set, then it will require reverse engineering the CPLD to get the rest of the design—assuming, of course, that the function contained in the CPLD is not so obvious that it can be deduced by observation. CoolRunner-II CPLDs add substantially to the security picture, when properly used! Be sure and read more about the CoolRunner-II security in **Xilinx White Paper WP170**.

Appendix: Additional Steps for the Insecure and Paranoid

As mentioned earlier, LFSRs are considered to be cryptographically insecure when directly building stream ciphers. The references include several options which go beyond simple LFSR solutions and appear to have merit as viable stream ciphers. In particular, Reference 1 summarizes more than a dozen that combine LFSRs in interesting ways to increase the resistance to attack. Among them are:

- Bilateral Stop and Go generators
- Dynamic random sequence generator

- Gollman cascade
- Shrinking generator
- Self shrinking generator

Typically, these ciphers combine several LFSRs by sequentially stacking them, by combining several with multiplexers, by logically combining multiple LFSRs to direct the clocking of others, etc.

Going beyond cryptography, steganography and partitioning, there are other actions that can be taken. Using chip scale packages with balls that hug the board surface makes it difficult to probe signals. Connecting PCB traces on internal traces, without surfacing except right at the pin sites, also makes it difficult. Employing conformal chip coatings is still another way of keeping signals secure, and it is certain that even more techniques exist.

References

Stream Ciphers

1. Bruce Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons, Inc., 1996, Chapters 16 and 17 (pp369 –428)
2. A.J. Menezes, P. van Oorschott, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press 1997, pp 191-222 (Chapter 6, "Stream Ciphers") (**http://www.cacr.math.uwaterloo.ca/hac/**)
3. "Stream Ciphers," M.J.B Robshaw, *RSA Laboratories Technical Report TR-701*, Version 2.0, July 25, 1995 (**ftp://ftp.rsasecurity.com/pub/pdfs/tr701.pdf**)
4. "Clock-Controlled Shift Registers: A Review," D. Gollman, W. Chambers, *IEEE Journal on selected Areas in Communications*, Vol. 7, Nol 4, May 1989
5. "A New Family of Stream Ciphers Based on Cascaded Small S-Boxes," Lin Gan, Stan Simmons and Stafford Tavares, *2001 Canadian Conference on Electrical and Computer Engineering*, Toronto, Ontario
6. "SOBER: A Stream Cipher based on Liner Feedback over GF(2^8)," Greg Rose (**http://www.qualcomm.com.au/Sober.html**)
7. "SNOW – a new stream cipher," Patrick Ekdahl, Thomas Johansson, *Proceedings of the first Nessie Workshop*, Nov. 13-14, 2000. (check the whole webpage at: **http://www.it.lth.se/cryptology/snow/snow10.pdf**)
8. "The Lili-128 Keystream Generator," E. Dawson, A. Clark, J. Golic, W. Millan, L. Penna, L. Simpson (**http://www.isrc.qut.edu.au/resource/lili/lili_nessie_workshop.pdf**)
9. "Pseudorandom Bit Generators in Stream-Cipher Cryptography," Kencheng Zeng, Chung-Huang Yang, Dah-Yea Wei and T.R.N. Rao, Computer, February 1991, pp 8 – 17
10. "A Low Cost, High Speed Encryption System and Method," G. Mayhew, I.E.E.E. 1994
11. "The Alternating Step (r,s) Generator," Sept. 2002, Ali Adel Kanso (**http://www.lsv.ens-cachan.fr/~goubault/SECI-02/Final/actes-seci02/pdf/004-Kanso.pdf**)
12. "Windmill pn-sequence generators," B.J.M. Smeets, W.G. Chambers, *I.E.E. Proceedings*, Vol. 136, Pt. E. No.5, September 1989, pp 401 – 404

13. "Clock-controlled shift registers in binary sequence generators," W. G. Chambers, *I.E.E. Proceedings*, Vol. 135, Pt. E, No. 1, January 1988, pp. 17 – 24

14. "Mutually clock-controlled cipher keystream generators," W.G. Chambers, S.J. Shepard, *Electronics Letters*, 5th July 1997, Vol. 33, No. 12, (I.E.E.E.), pp 1020 – 1021

LFSRs.

15. *Shift Register Sequences*, S. Golomb, Aegean Park Press, Laguna Hills, CA. 1982

16. "Pseudorandom bit generator based on dynamic feedback topology," R. Mita, G. Palumbo, S. Pennisi and M. Poli, *Electronics Letters*, 12th September 2002, vol. 38, No. 19, pp 1097 – 1098

17. "Maximal and Near-Maximal Shift Register Sequences: Efficient Event Counters and Easy Discrete Logarithms," Douglas W. Clark, Lih-Jyh Weng, *IEEE Transactions on Comuputers* 43,5 (May 1994, pp. 560-568

18. "The Theory of Autonomous Linear Sequential Networks," Bernard Elspas, *IRE Transactions on Circuit Theory*, 1959, CT-6, pp. 45-60

19. "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators," Peter Alfke, XAPP 052, July 7, 1996, Xilinx (**http://www.xilinx.com/xapp/xapp052.pdf**)

20. "LFSRs as Functional Blocks in Wireless Applications," (XAPP220), January 11, 2001,Stephen Lim and Andy Miller (**http://www.xilinx.com/xapp/xapp220.pdf**)

21. "PN Generators Using the SRL Macro," (XAPP211), January 9, 2001, Andy Miller and Michael Gulotta (**http://www.xilinx.com/xapp/xapp211.pdf**)

22. "Linear Feedback Shift Registers in Virtex Devices," (XAPP 210), January 9, 2001, Maria George and Peter Alfke (**http://www.xilinx.com/xapp/xapp210.pdf**)

23. "Gold Code Generators in Virtex Devices," (XAPP 217), January 10, 2001, Maria George, Mujtaba Hamid and Andy Miller (**http://www.xilinx.com/xapp/xapp217.pdf**)

24. "Generators for sequences with Near-Maximal Linear Equivalence," W.G. Chambers, D. Gollmann, *I.E.E. Proceedings*, Vol. 135, Pt. E, No. 1, January 1988, pp 67 – 69

25. "Generation of High-Speed Pseudo-Random Sequences Using Multiplex-Techniques," F. Sinnesbichler, A. Ebberg, A. Felder, R. Weigel, 1996 *IEEE MTT-S Digest*, pp 1351 – 1354

Stream Ciphers Attacks

26. "Cryptanalysis of Three Mutually Clock-Controlled Stop / Go Shift Registers," Jovan Dj. Golic, *IEEE Transactions on Information Theory*, Vol. 46, No. 3, May 2000, pp. 1081 – 1090

27. "On the Applicability of Distinguishing Attacks Against Stream Ciphers," Greg Rose, Philip Hawkes, Third Nessie Workshop

28. "A New Algorithmic Procedure to test m-Sequence Generating Connections of Stream Cipher's LFSRs," A.Ahmad, S. Al-Busaidi and M.J. Al-Mushrafi, IEEE Catalogue No. 01CH37239

29. "Derivation of the Feedback Taps of an LFSR from a Sequence Fragment," D.H. Green, *Electronics Letters*, 16th August 1990, Vol. 26, No. 17, pp 1352 – 1353.

30. "Searching for the Optimum Correlation Attack," Ross Anderson (**http://www.cl.cam.ac.uk/Research/Security/studies/st-alg.html**)

31. "Fast Correlation Attacks based on Turbo Code Techniques," Thomas Johansson, Frederick Johnsson (**http://citeseer.nj.nec.com/cache/papers/cs/23734/http:zSzzSzwww.it.lth.sezSzthomaszSzpaperszSzpaper080.pdf/johansson99fast.pdf**)

32. "Another Attack on A5/1," Patrick Ekdahl, Thomas Johansson, *I.E.E.E. Transactions on Information Theory*, Vol. 49, No. 1, January 2003, pp. 284 – 289

33. "A Fast Correlation Attack on Lili-128," Thomas Johansson, Frederick Johnson, (**http://citeseer.nj.nec.com/cache/papers/cs/23734/http:zSzzSzwww.it.lth.sezSzthomaszSzpaperszSzpaper140.pdf/a-fast-correlation-attack.pdf**)

Steganography

34. "Information Hiding – A Survery," Fabien A.P. Petitcolas, Ross J. Anderson and Markus G. Kuhn, *Proceedings of the I.E.E.E.* Vol. 87, No. 7, July 1999, pp 1062 –1078 (**http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/ieee99-infohiding.pdf**)

Randomness

35. "Statistical Testing of Random Number Generators," Juan Soto, *Proceedings of the 22nd National Information Systems Security Conference*, Crystal City, Virginia, October, 1999 (see also reference 1, 2 and 15)

36. "Guaranteeing the Diversity of Number Generators," A. Shamir, B. Tsaban, (**http://arxiv.org/PS_cache/cs/pdf/0112/0112014.pdf**)

Further Reading

Application Notes

http://www.xilinx.com/xapp/xapp375.pdf (Timing Model)
http://www.xilinx.com/xapp/xapp376.pdf (Logic Engine)
http://www.xilinx.com/xapp/xapp377.pdf (Low Power Design)
http://www.xilinx.com/xapp/xapp378.pdf (Advanced Features)
http://www.xilinx.com/xapp/xapp379.pdf (High Speed Design)
http://www.xilinx.com/xapp/xapp380.pdf (Cross Point Switch)
http://www.xilinx.com/xapp/xapp381.pdf (Demo Board)
http://www.xilinx.com/xapp/xapp382.pdf (I/O Characteristics)
http://www.xilinx.com/xapp/xapp383.pdf (Single Error Correction Double Error Detection)
http://www.xilinx.com/xapp/xapp384.pdf (DDR SDRAM Interface)
http://www.xilinx.com/xapp/xapp387.pdf (PicoBlaze Microcontroller)
http://www.xilinx.com/xapp/xapp388.pdf (On the Fly Reconfiguration)
http://www.xilinx.com/xapp/xapp389.pdf (Powering CoolRunner-II CPLDs)
http://www.xilinx.com/xapp/xapp393.pdf (8051 Microcontroller Interface)
http://www.xilinx.com/xapp/xapp394.pdf (Interfacing with Mobile SDRAM)

CoolRunner-II Data Sheets

http://direct.xilinx.com/bvdocs/publications/ds090.pdf (CoolRunner-II Family Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds091.pdf (XC2C32 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds092.pdf (XC2C64 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds093.pdf (XC2C128 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds094.pdf (XC2C256 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds095.pdf (XC2C384 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds096.pdf (XC2C512 Datasheet)

CoolRunner-II White Papers

http://www.xilinx.com/publications/products/cool2/wp_pdf/wp165.pdf (Chip Scale Packaging)

http://www.xilinx.com/publications/whitepapers/wp_pdf/wp170.pdf (Security)

http://www.xilinx.com/publications/whitepapers/wp_pdf/wp198.pdf (Cell Phone Handsets)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/30/03	1.0	Initial Xilinx release.