EVALUATION SUMMARY:
VIA C3 NEHEMIAH RANDOM NUMBER GENERATOR

Cryptography Research performed an evaluation of the hardware-based random number generator (RNG) in the VIA Technologies C3 Nehemiah core. This document summarizes findings from the technical report prepared for VIA Technologies.

Background

Randomness is required for a variety of computational, statistical, and security-related applications. In particular, cryptographic applications rely on randomness to generate keys, produce unpredictable challenge values, create padding bytes, and derive other security-critical parameters. Almost all security protocols rely on sources of randomness, and random number generator flaws are a common security problem.

Most computers are completely deterministic in operation, and therefore lack convenient sources of randomness. As a result, developers of security software rely on software-based pseudo-random number generators (PRNGs). These generators collect "seed" data from unpredictable system activity, including keystrokes, mouse activity, and process timings. Although PRNG output is not truly random, properly-seeded PRNGs can generate sufficiently unpredictable output for most applications. Seeding, however, is generally a slow process involving user involvement, creating testing challenges and difficulties in automated environments.

The VIA Technologies C3 Nehemiah core incorporates a hardware-based random number generator (RNG). This feature can produce high-entropy output at fast bit rates. This document summarizes results from our testing of the RNG.

RNG Design

The RNG contains a raw bit generator, digital post-processing circuitry, and an interface accessible by a special-purpose instruction. This section contains a brief description of the Nehemiah RNG source based on operational modes recommended by Cryptography Research.

Raw bit generator. A collection of freewheeling oscillators serves as an entropy source. A slow freewheeling oscillator (~30 MHz) is utilized to sample the output of a fast freewheeling oscillator (~600 MHz). The jitter of the slow oscillator is further increased by adjusting the oscillator's bias voltage with output from two additional fast oscillators. Thermal noise, manufacturing variations, temperature, software settings, and local electrical conditions are expected to contribute entropy to the sampled output.

Digital post-processing . Pairs of bits in the sampled

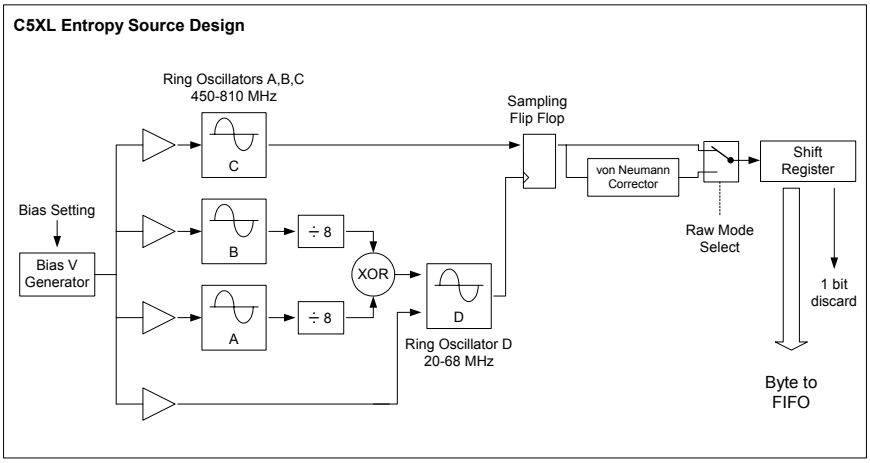

Figure 1: Nehemiah entropy source design.

CRYPTOGRAPHY RESEARCH

output are passed through a von Neumann whitener, which reduces (but does not completely remove) single-bit biases from the sampled data.[1] Whitened bits are then accumulated in a FIFO containing four 8-byte buffers (32 bytes total). To save power, the raw source oscillators are automatically shut down when the FIFO is full.

Data requests. User code may retrieve the contents of one 8-byte buffer at a time via the extended x86 XSTORE instruction. Each XSTORE operation either consumes 8 bytes (one buffer), or returns no data if no full buffer is available. The XSTORE instruction also copies the RNG configuration registers to user space, allowing applications to verify the exact RNG state at the time of the XTORE call.

Additional modes. The RNG contains advanced configuration settings that may be used by researchers and advanced users.[2] Ring 0 (operating system) code can adjust the oscillator bias voltage, and can optionally disactivate the von Neumann corrector for testing purposes. Other modes include a string filter and a 1-of-N bit discarder. The configuration settings are returned with each data request so that applications can ensure that the configuration is acceptable.

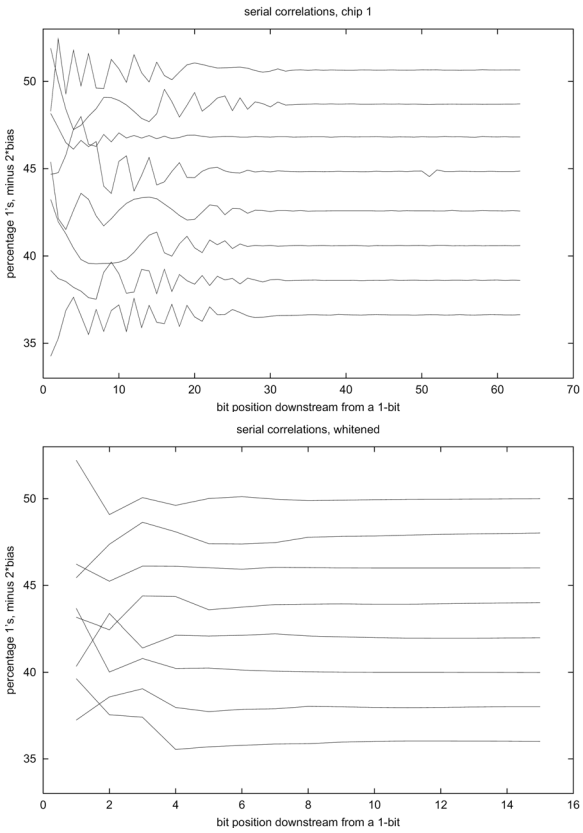

Figure 2: Serial correlations diminish more quickly when the von Neumann whitener is enabled

Summary of Evaluation Results

Testing procedures. Multiple processors were tested in a range of environmental conditions, oscillator stop/start usage patterns, and conditions found in multi-application environments. The Nehemiah core allows direct access to the raw (unwhitened) bit generator output, permitting the design to be validated with higher assurance than many other RNG designs. Because the RNG may be used in multi-application environments, we included tests that assumed that the RNG was being shared by hostile processes running concurrently on the same processor.

Entropy. The raw bit generator (whitener disabled) was estimated to yield 0.78 to 0.99 bits of entropy per raw output bit. Operation with the whitener enabled is believed to provide 0.99 bits (typical) of entropy per output bit. When used at the recommended settings, it should be reasonable to assume that the RNG provides 0.75 bits of entropy per output bit.[3] Note that our recommendations have been chosen conservatively, as the extremely high performance of the RNG allows applications to gather extra data with minimal overhead.

Bitrate. The RNG generates output at significantly higher rates than most PC-based randomness resources. Raw bits are produced at rates of 30 to 50 Mbits/sec, and whitened bits were observed at rates of 4 to 9 Mbits/sec. Variations in output rates depend on the RNG configuration and the oscillator rates. PRNGs seeded with the Nehemiah RNG should be able to easily sustain output in excess of 2 Mbits of entropy per second, which should eliminate blocked PRNG reads in virtually all applications.

[1] A von Neumann whitener reduces single-bit biases by outputting either a single "whitened" bit or no bits for every 2 bits generated by the raw source. This whitening process can be disabled for testing purposes.

[2] A complete description of operational modes is available in the *VIA C5XL Processor Random Number Generator Application Note*, provided by VIA Technologies.

[3] Sensitive operations/applications (such as initial PRNG seeding) should use even more conservative values if there is no significant performance impact.

31:22	21:16	15	14	13	12:10	9:8	7	6	5	4:0
Reserved	String Filter Count	String Filter Failed	String Filter Enable	Raw bits enable	DC bias	Reserved	Reserved	RNG enable	Reserved	Current Byte Cout
	Read/ Write	Read/ Write	Read/ Write	Read/ Write	Read/ Write			Read/ Write		Read only

Figure 3: Nehemiah RNG Configuration Register (MSR 0x110B)

RNG Use

The RNG should be initialized at boot by code with privileged (ring 0) access to a Nehemiah specific Machine Specific Register (MSR). The RNG may be used by any user-level application after it has been initialized.[4]

As with any hardware entropy source, it is recommended that the Nehemiah RNG be used as a source of seed data for a cryptographically strong PRNG or hash function. In such a design, an entropy pool is periodically seeded with data from XSTORE calls. The entropy pool may be occasionally "stirred" with a mixing function, which combines entropy across RNG outputs. Finally, application requests for random numbers are served by an output function, which transforms the entropy pool to generate output data for use by applications.

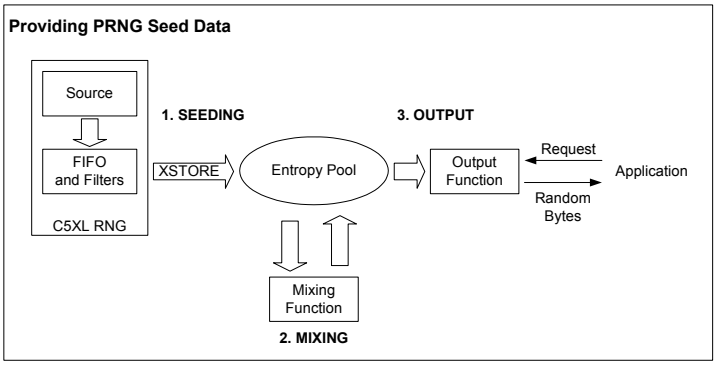

Figure 4: Seeding a PRNG with the Nehemiah RNG.

A well-designed PRNG serves as an entropy buffer in which the various rates of seeding and output, as well as the quality of the seed material, determine the quality of the overall output. In particular, the use of a cryptographically-based PRNG enables the per-bit entropy of the Nehemiah RNG to be improved and also provides an additional margin of safety to applications requiring randomness.

Examples of commonly-used PRNGs include the Linux/UNIX /dev/random resource and the Yarrow[5] PRNG. For OS randomness resources, the rate of contributed entropy can be estimated as 0.75 bits of entropy per XSTORE bit when the RNG is properly configured, although conservative developers can always use smaller estimates for this value.

Conclusion

Our analysis indicates that the Nehemiah core Random Number Generator is a suitable source of entropy for use in cryptographic applications. Because the RNG is part of the processor, it can be easily incorporated into software applications and functions well within a multi-application environment without requiring separate device drivers. Overall, we believe that the Nehemiah RNG meets the design objective of providing applications with a high-performance, high-quality, and easy-to-use random number generator. We hope that other vendors will follow VIA's lead and incorporate random sources into their processor designs.

The complete Cryptography Research report, "Evaluation of VIA C3 Random Number Generator," dated February 27, 2003 is available upon request from VIA Technologies.

[4] VIA processor documentation describes procedures for determining if an RNG-enabled Nehemiah C3 is present. The complete evaluation report contains implementation recommendations for configuring and using output from the RNG.

[5] *Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator*, by J. Kelsey, B. Schneier, and N. Ferguson, 1999.