Delivering New Platform Technologies

George Cox

Security Architect

Intel Corporation

Dreaming up a new technology is just the first of many steps in getting the technology developed, making it deployable, delivering it broadly into the market place, getting it widely enabled, and having it be effectively used. In this talk, we discuss this the various steps required to successfully carry out such a process using Intel's new RdRand/DRNG technology as an example.

Agenda

Delivering New Platform Technologies

The Platform Entropy Problem

Have Entropy?

How Make It Have Known Quality

How Make Sure Design Meets Goals

How Make Sure That Design Gets Delivered

How Make Sure Can Deliver Across Product Lines

How Get Entropy To Users

How Get Users To Use

How Find New Uses

Delivering New Platform Technologies

Common problem for product developers

Existing platform architecture framework as context

- No “clean sheet of paper”

How embed new technology so that value gets delivered through applications and services

Security technologies bring the added problem of how to embed them securely

The Platform's Need For Entropy

Entropy is valuable in a variety of uses, the first of which that comes to mind being for "keying material" in cryptography.

Cryptography is a basic building block for modern computer security and is based upon the use of comparably high quality algorithms and keys.

Either being "weak" has resulted in successful attacks on cryptographic systems.

Over time, cryptographic algorithms and their implementations have continually been improved, as needed (e.g., our AES NI).

Comparably, the availability, quality, and performance of entropy sources have not.

The Platform Entropy Problem

Historically, computing platforms have had a perennial problem of the absence of any high quality/high performance “entropy source”.

Older approaches were almost all based upon the premise that true “raw” entropy accumulation was a very slow process.

Entropy was slowly gathered in small quantities from sources of true entropy (some HW source) at slow rates - in the bits/sec (e.g., key strokes, mouse click timing, disk seek times) up to kilobits/sec (analog ring oscillator-based TRNGs)

As a scarce resource, entropy had to be accumulated (in an entropy pool) and used to seed/reseed a SW PRNG that could cryptographically spread that scarce entropy resource out over numerous requests with acceptable performance.

With little availability of quality entropy early in boot, OSes can have difficulty generating good boot time keys.

The Platform Entropy Problem

OS mediated access to HW entropy sources reduces application or networking performance further.

Many of these SW PRNGs did not met quality standards such as NIST SP 800-90 or were not FIPS 140-2 certified as such.

Such SW PRNGs also have a history of being error prone and easily monitorable/attackable.

As servers become headless (e.g., with no keyboard or mouse) and move to SSDs (instead of disks), platform sources of entropy are going away.

As one moves to virtualized environments, the virtualized OS that thinks it can get at the platform's HW entropy probably can't and will suffer further performance loss caused by hypervisor mediation.

Have Entropy?

True Random Number Generators (TRNGs) – aka Nondeterministic Random Bit Generators (NRBGs)

- True random numbers must come from some HW source
- TRNG/NRBG are really misnomers as don't they generate entropy
- They just sample and digitize some form of HW noise (e.g., thermal)

Existing Entropy Sources are analog (e.g., triple ring oscillators)

- Hard to manufacture
- Hard to migrate across process generations - particularly as feature sizes continue to shrink
- Usually relatively slow (e.g., a few hundred Kbits/sec)

Opportunity arises out of circuit research

- A digital entropy source
- Blindingly faster by comparison (e.g., 2.5 – 3.5 Gbits/sec)

How embed entropy in platforms?

How Make It Have Known Quality

Entropy Sources "raw" output can be

- Correlated
- Biased

Need post processing to generate uniformity in output

Today's solutions are to use "raw" entropy to "seed" SW PRNGs

Historically, numerous problems with SW PRNGs

Solve by

- Meeting a standard (e.g., NIST SP 800-90)
- Do it in HW
- To make it
  - Less attackable
  - Faster

How Make Sure Design Meets Goals

Certify that it meets the standard it is designed for (e.g., NIST SP 800-90 compliance via FIPS 140-2/3 Level 2 process)

Self certification

- Why should we trust you? – financial institutions through security ISVs

Third party certification (e.g., via SAIC)

- Costs
- TTM delay

How Make Sure That Design Gets Delivered

Design For Test (DFT)

- Same pre/post silicon
  - Consistency across all testing regimes (unit, uncore, full chip)
  - Otherwise, do not know whether (or not) all versions of implementation are equivalent
- Comprehensive Built In Self Test (BIST) – "test from inside" - use in
  - High Volume Manufacturing (HVM) and
  - Product
  - At every "reset" to guarantee feature health
- Test Port
  - Complete internal state examination/alteration and sequencing
  - Available only in development/debug

Deliver "securely"

- Build inside an enforced "security boundary"
  - Interlocks to guarantee no internal state examination in production use
- Allow only authorized users to access

How Make Sure Can Deliver Across Product Lines

Design for reuse

- Just write nice RTL and done?
- Just run pre-silicon simulations and done?
- Just synthesize and "place and route" and done?

Across (semiconductor) processes

- Generations (e.g., 45, 32, 22, 14, and now waiting for 10 nm)
- Variations – normal versus low power

Across on die interconnects (e.g., range of busses, JTAG, …)

Across design styles (e.g., coding standards)

Across clocking models (e.g., clock gating) – autonomous or forced

Across power models (e.g., power gating) – autonomous or forced

Across post-silicon validation (process, voltage, temperature)

Across 31 product embeddings by EOY'2012

How Get Entropy To Users

Drive for as little dependency on others => less enabling cost/time

- HW OEMs
- VMMVs
- OSVs
- ISVs

Interface choices

- Protected register interface – requires
  - OS "driver" deployment
  - user/OS call/return
- Instruction interface - RdRand
  - Available in all domains and states
  - No OS or other SW dependency
  - Application/service SW can invoke directly
  - Not always applicable if function is "long running" as could interfere with real time response
- Results in "vertical IP" – full value stack – from primitive (Entropy Source) to instruction provided in HW without any need for SW enabling

How Get Users To Use

Motivation

- Entropy (and algorithms) basis for cryptography
- Cryptography basis for many security solutions …
- Bad (quality) keys more often the cause of cryptography breaks

Understand potential usages

Make easy to adopt

- Make it FAST – Butler Lampson
- Embedding in existing libraries (e.g., seeding SW PRNGs)
- Direct use via compiler intrinsic (avoid trusting library and call/return overheads)
- "White papers" on
  - Efficient/correct use
  - Performance/quality advantages
- Sample embeddings in key OSV/ISV applications/services
- Advertising of value

New "brand promise" of high quality/high performance entropy on ALL Intel platforms

How Find New Uses

Always want to find/enable new uses/users

Explore other entropy uses beyond cryptography

What other uses might benefit from or be enabled by more high quality/high performance entropy being available

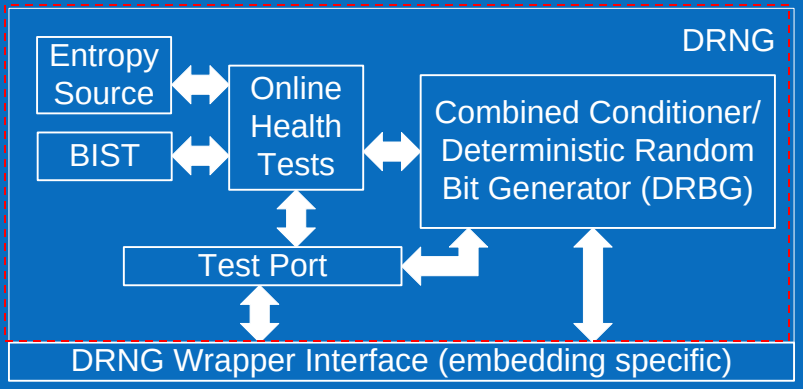
- We thought:
  - Nondeterministic simulations
  - Nondeterministic gaming

  but they require "replayable" entropy streams => separate SW PRNGs per application instance

  still use outputs for high volume "seeding" of parallel SW PRNGs
- We have characterized:
  - Communication concentrator/transaction frontend servers – IV per packet
  - Memory/disk clearing

  and have ISV enabling under way

Basic DRNG Module



A reusable IP module that Provides each embedding with an autonomous/self contained, high quality/high performance, "complete" DRNG

Composed of

- An all-digital Entropy Source (NRBG), runtime entropy quality measurement (OHT),
- Conditioning (via AES CBC-MAC mode) and DRBGing (via AES CTR mode) post processing and
- BIST and Test Port

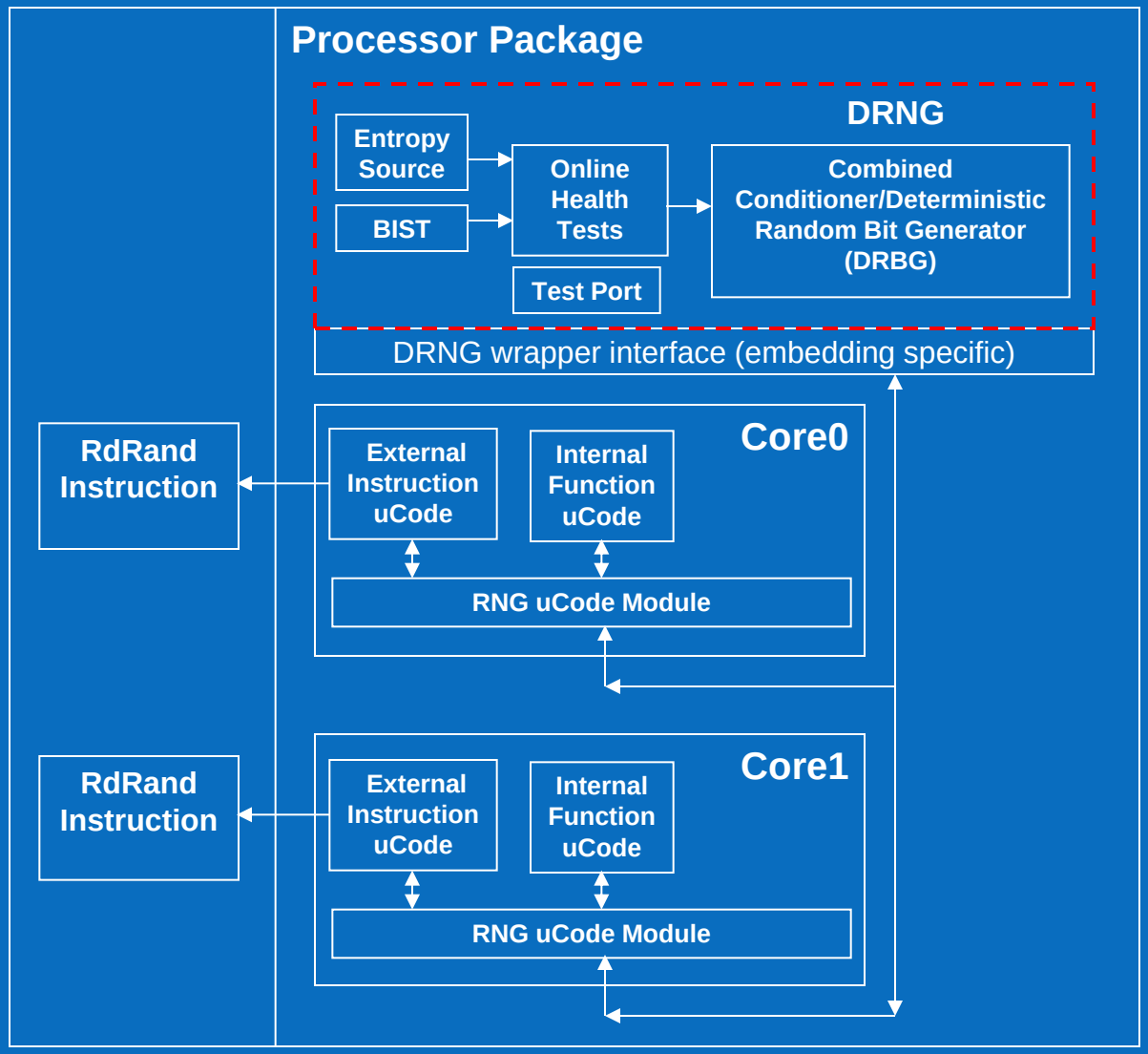
"Standards" compliant (NIST SP 800-90) and FIPS 140-2/3 Level 2 certified as such and

Designed for ease of testability, debug, and validation in HVM and in end user platforms

- Comprehensive BIST test with "known value" injection/sampling and
- Test Port (and associated tools) for full pre/post silicon debug flexibility

Red dotted line is a FIPS boundary

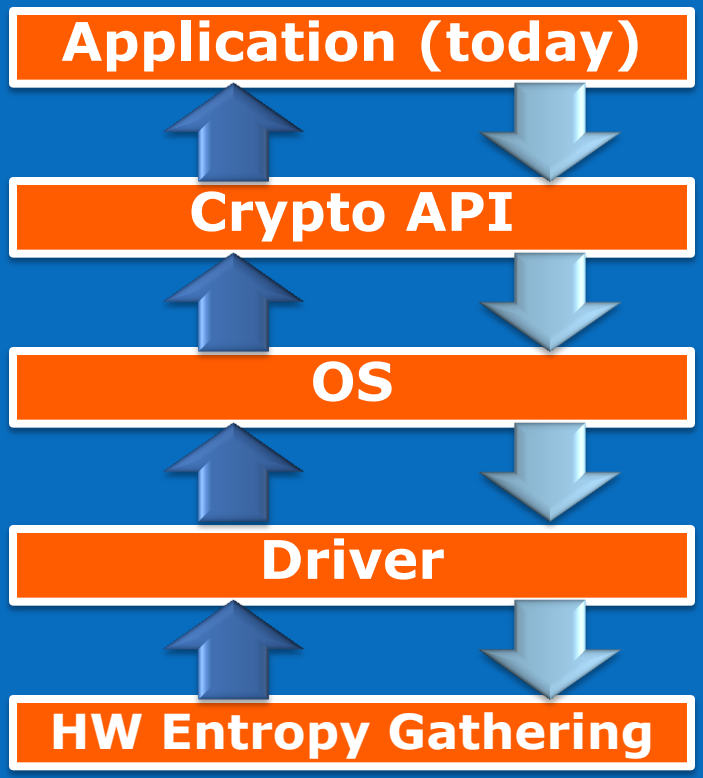
Digital RNG – DRNG

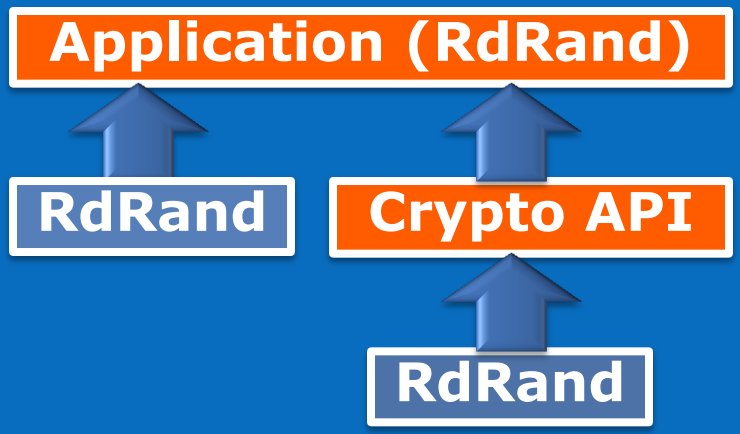


Provides each processor package with a chipset independent, autonomous/self contained, high quality/high performance, "complete", shared, uncore resident DRNG

Performance

Direct access to random numbers through RdRand bypasses OS, driver, and associated overhead





On-chip entropy source - no off-chip bus or I/O delays

Latency comparable to software PRNGs

Highly scalable

Measured Throughput

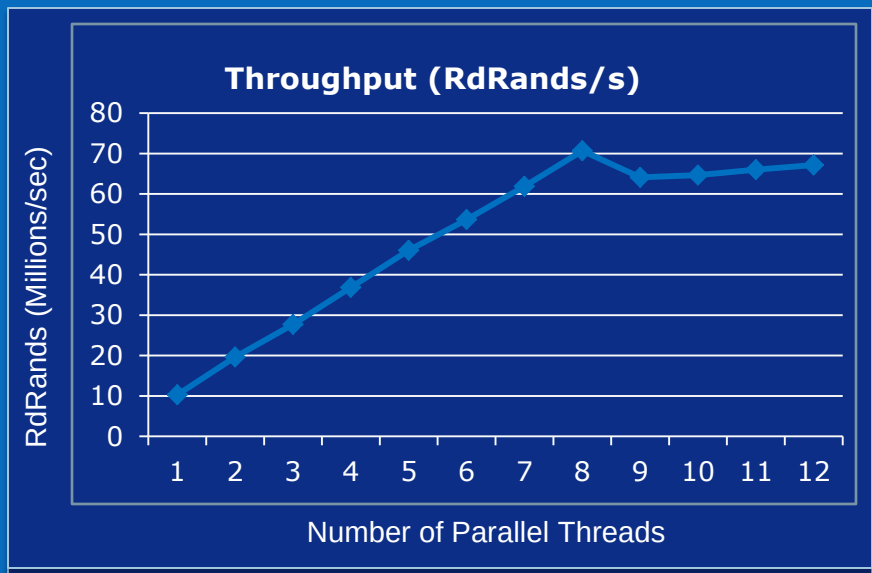
Preliminary data from pre-production Ivy Bridge sample[1]

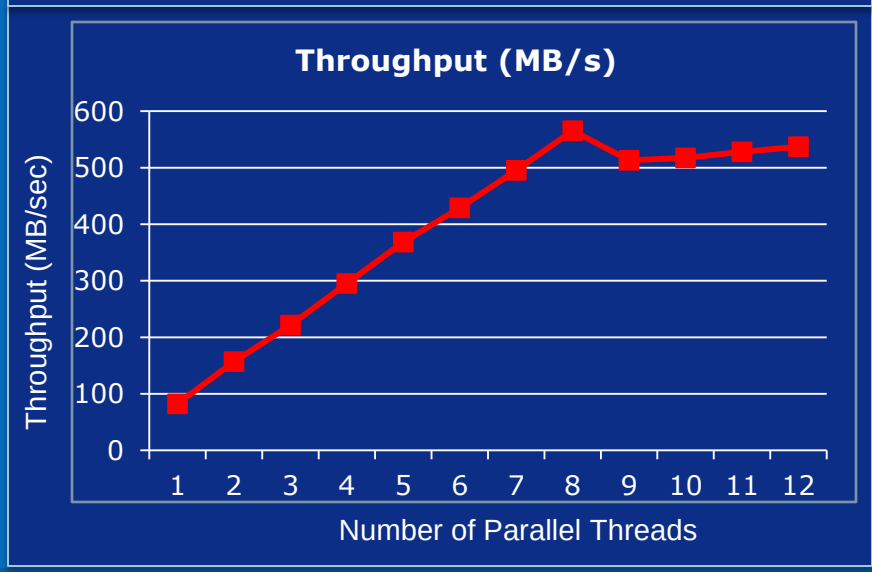
Up to:

- 70 million RdRand invocations per second
- 500+ Million Bytes of random data per second

Throughput ceiling is insensitive to number of contending parallel threads

- Steady state maintained at peak performance





[1]Data taken from Intel® processor codename Ivy Bridge early engineering sample board. Quad core, 1.4 GHz, 4 GB memory, hyper-threading enabled. Software: LINUX* Fedora 14, gcc version 4.6.0 (experimental) with RdRand support, test uses pthreads kernel API.

Response Time and Reseeding Frequency

Preliminary data from pre-production Ivy Bridge sample[1]

RdRand Response Time

~150 clocks per invocation

(Note: Varies with CPU clock frequency since constraint is shared data path from DRNG to cores.)

Little contention until 8 threads

- (or 4 threads on 2 core chip)

Simple linear increase as additional threads are added

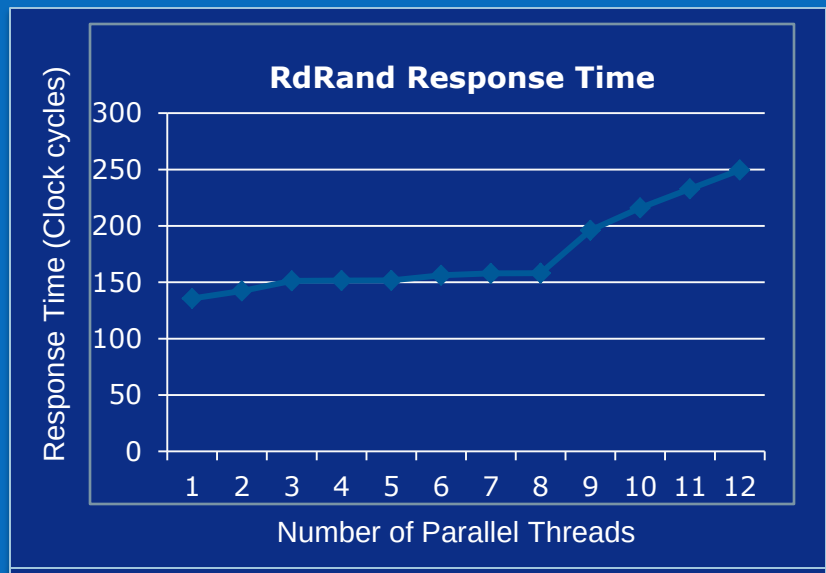
DRNG Reseed Frequency

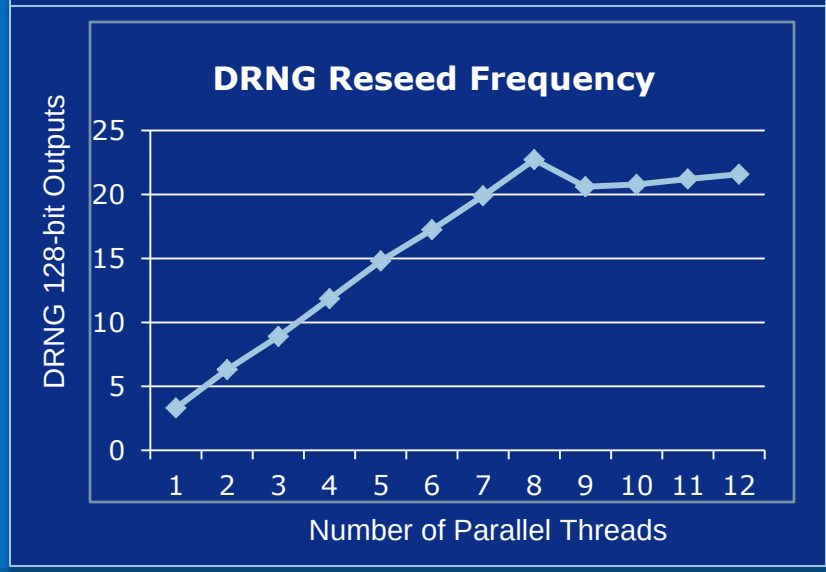
Single thread worst case: Reseeds every 4 RdRand invocations

Multiple thread worst case: Reseeds every 23 RdRand invocations

At slower invocation rate, can expect reseed before every 2 RdRand calls

- NIST SP 800-90 recommends ≤ 2^{48}





[1]Data taken from Intel® processor codename Ivy Bridge early engineering sample board. Quad core, 1.4 GHz, 4 GB memory, hyper-threading enabled. Software: LINUX* Fedora 14, gcc version 4.6.0 (experimental) with RdRand support, test uses pthreads kernel API.

Acknowledgements

The original RdRand/DRNG architecture team - Ernie Brickell, James Coke, George Cox, Charles Dike, Martin Dixon, Steve Fischer, Ed Gamsaragan, Shay Gueron, Howard Herbert, DJ Johnston, Greg Piper, Guna Thuraisingham, Jesse Walker

The DRNG design/implementation team - George Cox, Charles Dike, and DJ Johnston

The integration teams across 30 product embeddings

The product enabling teams working with OSVs/ISVs

Backup

DRNG Direction

- Support the RdRand instruction and other product technology uses
- Be standards based to provide known quality (NIST SP 800-90 compliant)
  - Approved in Security Initiative, CART, and IPTR reviews as the designated vehicle to drive a cross Intel, uniform brand promise of high quality/high performance entropy
- Be FIPS certified to demonstrate adherence to that standard
  - As requested by Microsoft, Apple, RSA, EMC, Oracle, the US and British governments, …
- Be totally Intel IP (parent application filed)
- Be a reusable IP module for reuse/deployment on ALL Intel silicon wherever entropy is needed => replacing any/all alternatives

We are well on the way to accomplishing these goals over the next couple of process/product generations:

- With MPS1/1269, X10B/1270, and MPS3/1269.8 test chips in progress and X11B/1271 in planning
- It is POR for:
  - Large core processors:
    - Starting with Ivy Bridge and Haswell (in 1270) and
    - Associated server processors (Ivy Town)
  - Small core processors: starting with Silvermont/Pondicherry-based SOCs
    - Valley View (in 1271.1) and
    - Tangier (in 1271.4); and
  - Chipsets: starting with Lynxpoint/Wellsburg PCH (in 1269.8)
  - Integrated graphics – starting w/GEN in Haswell

DRNG Description

Directly available for use by ALL processor-based SW use (via RdRand) instruction in all privilege levels and all operating modes and

Also available for other uncore and processor uCode usages such as implementing other IA instructions that need random numbers (e.g., Secure Enclaves IA instructions)

Process Independence

- Standard cell-based design w/NO need for process specific validation (e.g., as with analog entropy sources) or separate power supplies

Robustness

- HW/uCode implementation protects the DRNG (both NRBG and DRBG units and their states) from SW attack

Shared Resource

- Across "n" processor cores, threads, VMs, communication stacks, and apps in package
- Designed to scale under multiple consumer load

Power Management

- Self scheduling, compute ahead, low latency, consumption driven, queued interface
- Clock gates itself off when queues full (and prepared to support power gating, but not yet called for)

DRNG Value

Technical Value

- "Securing" anything on a computer requires
  - Use of high quality cryptography which requires
  - High quality keys which require
  - High quality random numbers/entropy which require
  - A high quality Entropy Source (in HW) and
  - High quality (e.g., standards compliant) post processing (e.g., via conditioner/DRBG) certified as such (e.g., via FIPS 140 2/3 Level 2)
- DRNG
  - Satisfies these needs and
  - Is the first/only, commercially available, autonomous/self contained, "complete" DRNG product

Business Value

- Establish Intel products as providing the highest quality/highest performance entropy sources in the Industry
- Exceed/meet competition feature – ubiquitous (non-SKUed), brand promise, new instruction feature (RdRand) across ALL IVB/HSW processors

DRNG Value

Target Segments

- Initially POR on Ivy Bridge/Haswell processors => Eventually as a reusable IP collateral circuit providing uniform brand promise across ALL Intel silicon - platform types (MIDs – Servers) – on processors, chip sets, and anywhere else that entropy is required

Competitive Position

- Existing competition, led by VIA (C7 and Nano) with their XSTORE instruction, provide relatively slower Entropy Sources and do not integrate entropy post processing

Customer Input

- Microsoft, Apple, RSA, EMC, Oracle, Google, and Sun ALL very positive about "complete" DRNG (see Microsoft response below)

The Microsoft Letter

To: Renee James, Justin Rattner, Ernie Brickell, Pat Gelsinger, Rob Crooke, Tom Kilroy

CC: George Cox, Jim Rottsolk, Dave Aucsmith, Craig Mundie, David Pritchard

Subject: Intel Random Number Generation Hardware Proposal

Date: August 4, 2008

George Cox and a team of Intel engineers have developed requirements and a compliant design for a random number generator intended for inclusion in Intel CPUs. There is no such capability in current mass market CPU's (although a somewhat standard PC component, the TPM, has a low rate entropy source).

The lack of a high quality entropy source has been a significant contributing factor in the failure of many cryptographic systems. There is ample evidence that this has caused harm to customers of both Intel and Microsoft and knowledgeable customers have consistently demanded such a capability. As a result, Microsoft considers the broad availability of hardware providing high quality entropy as a critical need in current PC systems.

A hardware entropy subsystem must:

Have an analytically validated entropy quality model including a known "min-entropy" value for each raw sample.

Comply with existing standards (e.g., ANSI X9.82 and NIST SP 800-90) and, ideally, be FIPS 140-2/3 certified.

Provide entropy at a higher rate than current TPMs without the "part fatigue" associated with some TPM implementations.

Deliver high quality entropy without significant latency even early in system initialization without complex driver support.

Our initial analysis of the Intel design convinces us that Intel's proposal will likely fulfill these requirements and thus deliver high value to customers as well as significantly improve security in the PC ecosystem. We appreciate Intel's leadership in delivering this critical capability and strongly support its development and the widest possible deployment.

We look forward to working with Intel in providing this vital capability to our customers.

John L. Manferdelli, Distinguished Engineer, Microsoft Corporation

DRNG Reuse Scope & Feasibility

Si Impact – [Medium] – Mostly in uncore on processors

- Uncore: Totally self contained uncore DRNG (Entropy Source, Online Health Test, Conditioner, DRGB, Output Queue, BIST, and Test Port) plus "wrapper"
  - ~148,035 um2 (1269)
  - ~83,724 um2 (1270) and
  - ~81,251 um2 (1271)
- Core: ~50 uops to implement RdRand instruction

Effort – [Low] – Given prior Ivy Bridge and test chip work

Power impact – [Low] - Expect to be negligible due an clock gating ability when entropy computation/post processing is not in-progress

- ~4.40mw active, ~2.78 mw leakage on Ivy Bridge (1270)

Technical risks – [Low] – Given prior Ivy Bridge and test chip work

Process dependency – [None]

- Planned availability as reusable Intel Reuse Repository (IRR) IP library block across processes (e.g., 1269, 1270, 1271, …)

DRNG Maturity & Roadmap

Maturity - High

- Current tech readiness – passed TRP L2 (10/14/08), SAFE G2/SDL S1/SPARC passed, CART closed (9/18/08)
- Appropriate HAS, DRNG and Wrapper MAS specifications per embedding
- Product quality, collateral circuit of uncore elements (test chips 1269 (broadly functional) and X10B/1270 in mid-summer 2010), detailed element sizings available
- FIPSing direction approved, designed for FIPSing, and SOW for FIPSing of 1269 test chip closed w/Atlan (our FIPSing lab)
- Totally Intel IP with parent application filed

- X-generational Roadmap - It is POR for:
  - Large core processors:
    - Starting with Ivy Bridge and Haswell (in 1270) and
    - Associated server processors (Ivy Town)
  - Small core processors: starting with Silvermont/Pondicherry-based SOCs
    - Valley View (in 1271.1) and
    - Tangier (in 1271.4); and
  - Chipsets: starting with Lynxpoint PCH (in 1269.8)
  - Integrated and discrete graphics – starting w/GEN in Haswell and Forest Isle (dLRB3)
  - Eventually as a reusable IP collateral circuit providing uniform brand promise across ALL Intel silicon - platform types (MIDs – Servers) – on processors, chip sets, and anywhere else that entropy is required
- External dependencies
  - Designed to meet/exceed NIST SP 800-90 and be FIPS 140-2/3 Level 2 certified as such

DRNG Reuse – Interconnect Mix and Match

	Process	Primary Bussing Interface	Scan	Volume Output	Clock Rate
1266	1266				800 MHz
MPS1/1269.0	1269.0	Test wrapper pins			800 MHz
X10B/1270	1270		TMG scan	Special pins	800 MHz
MPS4/1269.8	1269.8		Slave TAP	VISA	207 - 400 MHz
Ivy Bridge and Ivy Town	1270	Message Channel +	TAP to MC	-	800 MHz
Haswell Client and Server	1270	IOSF SB (16 bit)	TAP to IOSF	VISA	800 MHz
Lynxpoint and Wellsburg	1269.8	ME AUX	TAP to ME AUX	VISA	207 - 400 MHz
Valley View	1271.1	IOSF SB (8 bit) + AC	Slave TAP	VISA	207 - 400 MHz
Tangier	1271.4	IOSF SB (8 bit) + AC	Slave TAP	VISA	207 - 400 MHz
MoonRun	1272	?	Slave TAP	VISA	? MHz
Canonical Wrapper for IRR		IOSF (8 bit + AC and 16 bit) and ME AUX	Slave Tap	VISA	scalable

DRNG Reuse Learnings

Initial processor-based embeddings used a "shared" DRNG module located in the processor uncore connected to the processor cores through a "shared" interconnect

The "shared" interconnects employed (i.e., Message Channel and IOSF) resulted in huge latency penalties (e.g. 2-400 uncore clocks) per reference even on unloaded configurations executing back-to-back RdRands

Ivy Bridge has implemented "a one off alternative" interconnect in order to cut this latency down to ~100 uncore clocks

This points to migrating the DRNG :

- Closer to the processors on a wider/faster "shared" bus (e.g., for Broadwell) or
- Into the processor as the DRNG becomes ever smaller by:
  - Process related shrinkage and/or
  - A slower (but smaller) DRNG design that would deliver faster overall results by avoiding interconnect latency

DRNG Enabling

Initial SW enabling in the Ivy Bridge context (which will be reusable across ALL Intel processor implementations) should include having externally deliverable/supported:

- RdRand instruction enabling (e.g., in SW development tools
  - Instruction level simulators
  - SDV and/or SDK enabling
  - Assembler, debugger, and complier intrinsic/instruction inlining support, and
  - An Embeddable RdRand "inner loop" (to be developed by SSG VSL/MKL team?)) and
- Along with these tools, support working with selected OSVs (e.g., Microsoft, Apple, and Linux), Security ISVs (e.g., RSA), and ISVs (e.g., EMC, Oracle, Google) for RdRand-enabling their SW PRNGs by:
  - Embedding the RdRand "inner loop" in their SW PRNGs (e.g., for seeding/reseeding) and
  - Direct use of RdRand (really more likely direct use of the RdRand "inner loop" via a complier intrinsic).
- Intel library enabling – for RdRand-enabling SW PRNGs in:
  - Intel's Vector Statistical Library (VSL) sub-component of Math Kernel Library (MKL) and
  - Intel's IPP libraries
- Inner loop (direct instruction use) enabling for other interested ISVs (e.g., nondeterministic simulation and gaming)

Intel internal documentation (e.g., FAQ – see attached draft); and

Intel external/product documentation (e.g., FAQ, instruction interface)

DRNG Usages

RNGs are foundational enablers (e.g., for making keys) of all cryptographic usages in

- Communications- ALL levels of ALL stacks (XML, TLS, VPN, IP, WiFi, …)
- Signing – digital certificates, integrity manifests, attestation, transactions
- Storage – file and volume
- DRM

In the crypto arena,

- Security ISVs and OSVs

will embed DRNG support into their cryptographic "libraries"

- VPN vendors
- SSL/TLS:
  - Browser vendors
  - SSL VPN vendors
- Authentication Services vendors
- Data base and storage ISVs

will use DRNG for making better keys, IVs, nonces

In the non-crypto arena,

- Non-deterministic simulation ISVs (Monte Carlo and weather) and simulation vendors
- Non-deterministic gaming ISVs (interactive and gambling)

will use DRNG for better simulation and gaming

FIPS Certification

Initial certification - significant work/cost ($35-50K)

- Approved by Security Initiative/BCG management
- In process of FIPS 140-2/3 Level 2 certification to NIST SP 800-90 compliance by an external lab (Atlan)
- On 1269/MPS1 test chip
- Accumulation and review of initial documentation
- Capture and analysis of “raw” entropy
- Running Known Answer Tests (KATS) and review of test results

Incremental recertification for reuse – much less work/cost ($5-15K) – drive to a simple repeatable process

- Once per process – on 1270/X10B test chip
- Once per product (family – Ivy Bridge, Haswell, Silvermont, LynxPoint)
- Update and review of documentation
- Capture and analysis of “raw” entropy
- Rerunning Known Answer Tests (KATS) and review of test results
- Not to be redone per stepping

FIPS Certification (2)

Once initial DRNG module FIPS certification is achieved

- Our plans are to incrementally recertify on a per product basis
- As an expected checklist item by OSVs and security, database, and storage ISVs

If certification is delayed, there will be no holdup of product shipment

- Until certified, we can say:
  - Product xyz provides a DRNG module designed to meet the NIST SP 800-90 standard.
- Once certified, we can say:
  - Product xyz provides a DRNG module certified to meet the NIST SP 800-90 standard to FIPS 140-2/3 Level 2.

If errata interfere with or preclude the certification process?

- We can still say:
  - Product xyz provides a DRNG module designed to meet the NIST SP 800-90 standard.

Some higher security customers/vendors may require an S-spec part that has gone through certification

As part of explaining RdRand and our DRNG, we will deliver white papers on:

- The RdRand instruction, its usages, and available SW tools/libraries;
- The DRNG module architecture; and
- How FIPS certification was achieved

RdRand/DRNG – To SKU Or Not To SKU

Since its inception, our RdRand instruction and its underlying DRNG enabling functionality have been:

- Intended for solving a fundamental platform security problem uniformly and
- Committed as a "common brand promise" of high quality entropy across ALL Intel silicon.

In all our negotiations with OSVs (e.g., Microsoft and Apple), security ISVs (e.g., RSA/EMC), and ISVs (e.g., Oracle and Google), their commitment to enabling/use has been predicated on their insistence of ubiquitous presence across the spectrum of Intel-based platforms.

Thus, our DRNG is a cross Intel security enabling element and is POR for large core processors (client and server), small core processors (all SOCs), chipsets, and integrated graphics

Beyond being used by processor ucode in the implementation of the RdRand instruction, on large core processors, starting with Haswell, our DRNG is also used by:

- PCU
- GEN and
- PAVP elements to provide entropy

Comparably, on small core processors, starting with Silvermont-based SOCs, our DRNG is also used by:

- The SEC (in Valley View)
- The Chaabi (in Tangier) elements to provide entropy

RdRand/DRNG – To SKU Or Not To SKU

On both large and small core processors, our DRNG will be used by ucode for making keys for memory encryption for PPPE and SE and by ucode in implementation of SE

On chipsets, starting with Lynxpoint and Wellsburg, our DRNG is used to replace their old analog TRNG.

Our DRNG is NOT a "performance" feature (e.g., like AES NI) that can be implemented more slowly in SW on lower end processors and much faster (e.g., using DRNG HW) on higher end processors – it is a fundamental platform enabling feature – there is no way to generate entropy in SW

Not all of our new security technologies have a direct monetization strategy

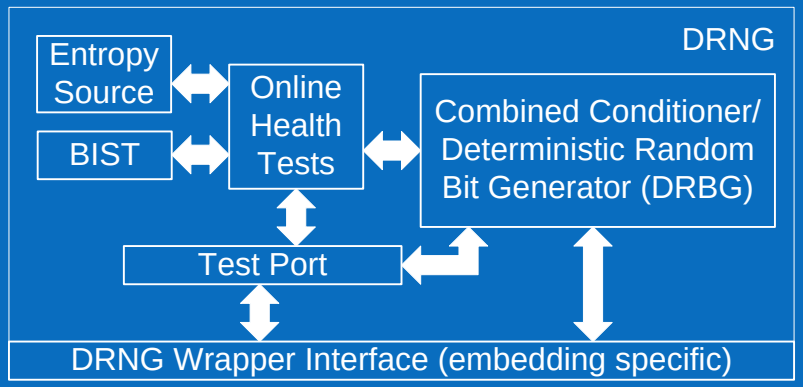
Intel is building a "Protection" value vector into our brand… at a level with power and performance

As described above, our RdRand/DRNG implementation is POR for delivery across all Intel silicon, from top to bottom

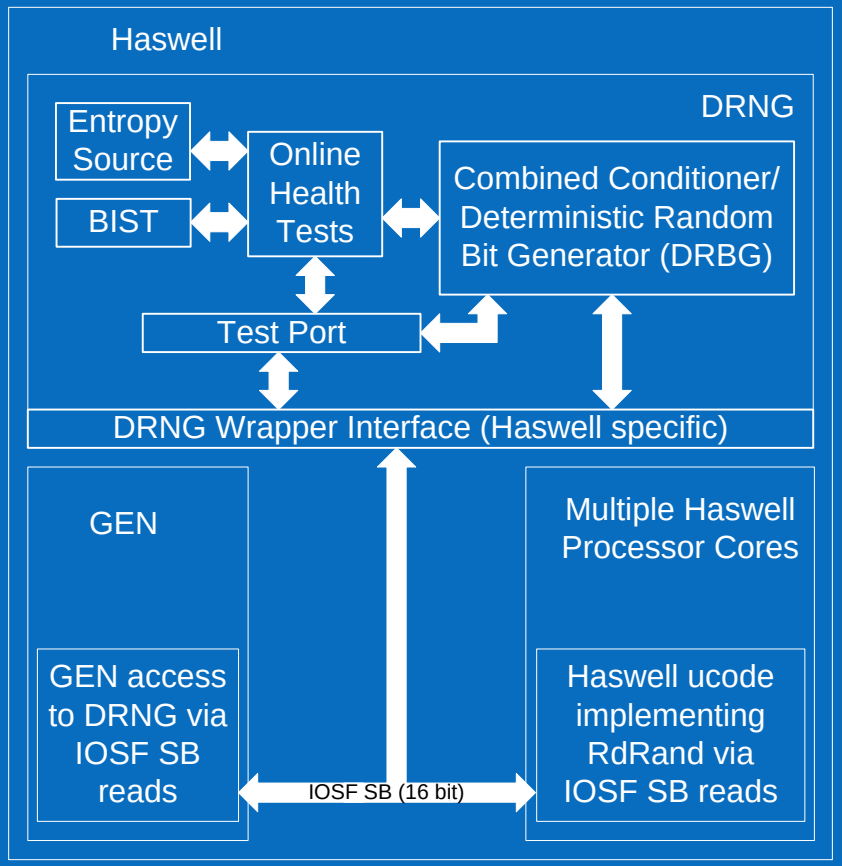
RdRand/DRNG is not a technology that Intel plans to SKU and target for sell-up, or consider for revenue-share or licensing models

As such, it is not a focus of the Security Monetization task force (an effort which is currently underway)

Basic DRNG Element



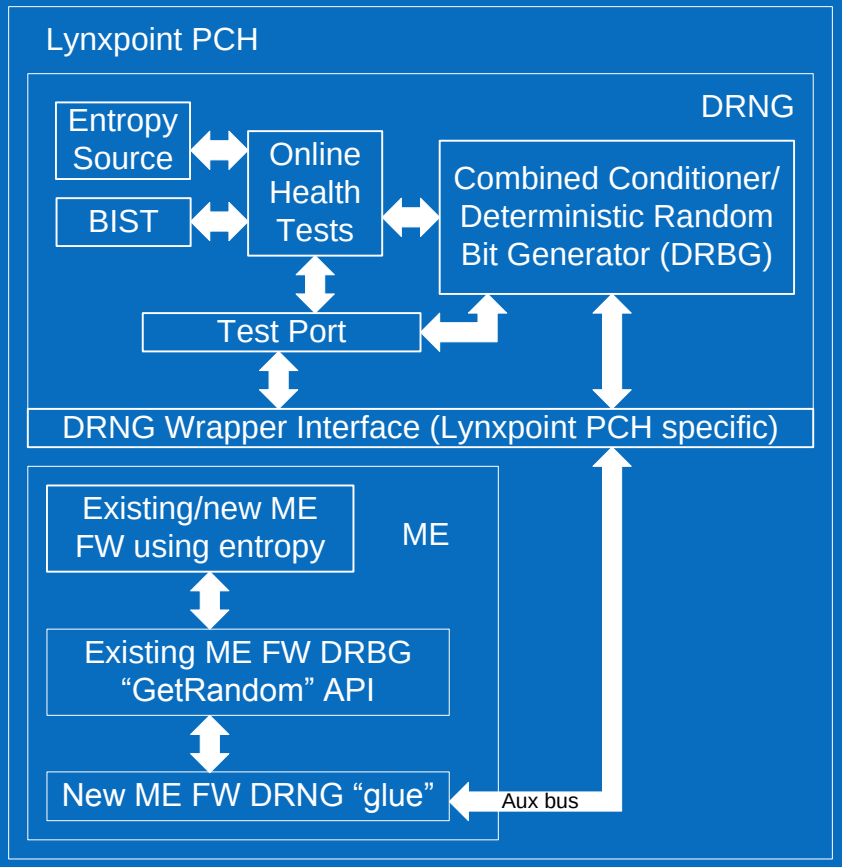
Haswell DRNG Embedding



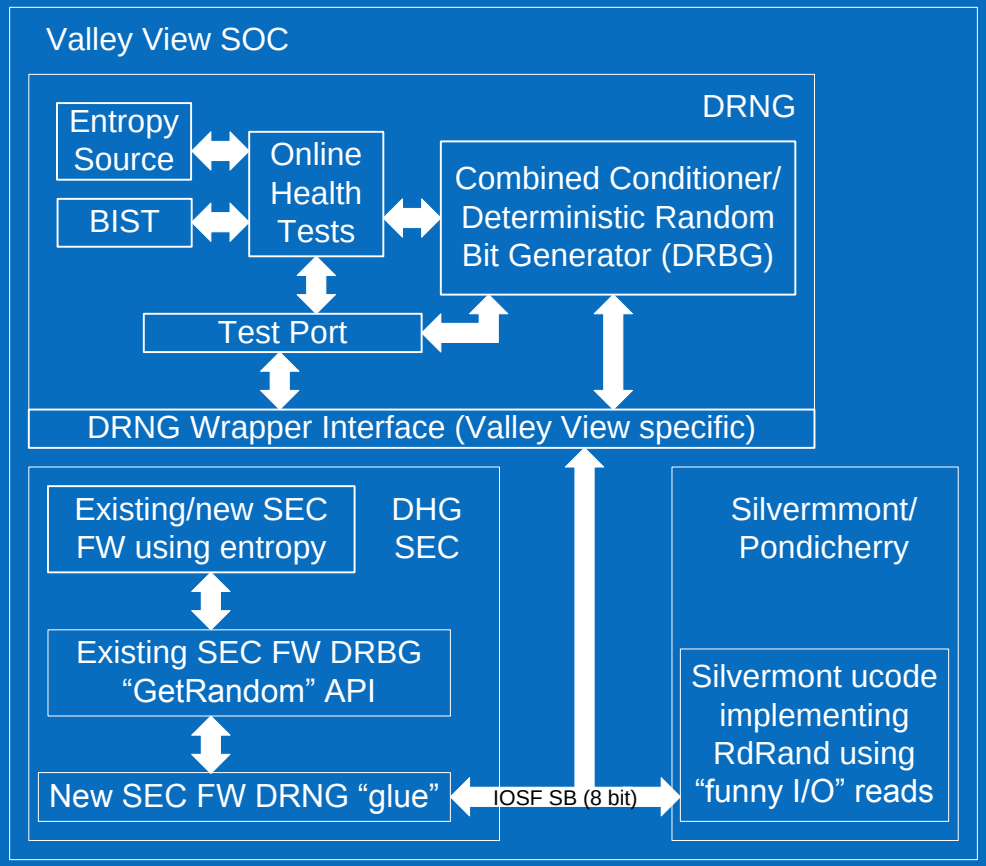
Previous Ivy Bridge configuration similar but for:

- Using older Message Channel (instead of newer IOSF SB) and
- IVB GEN not using DRNG

Lynxpoint PCH DRNG Embedding

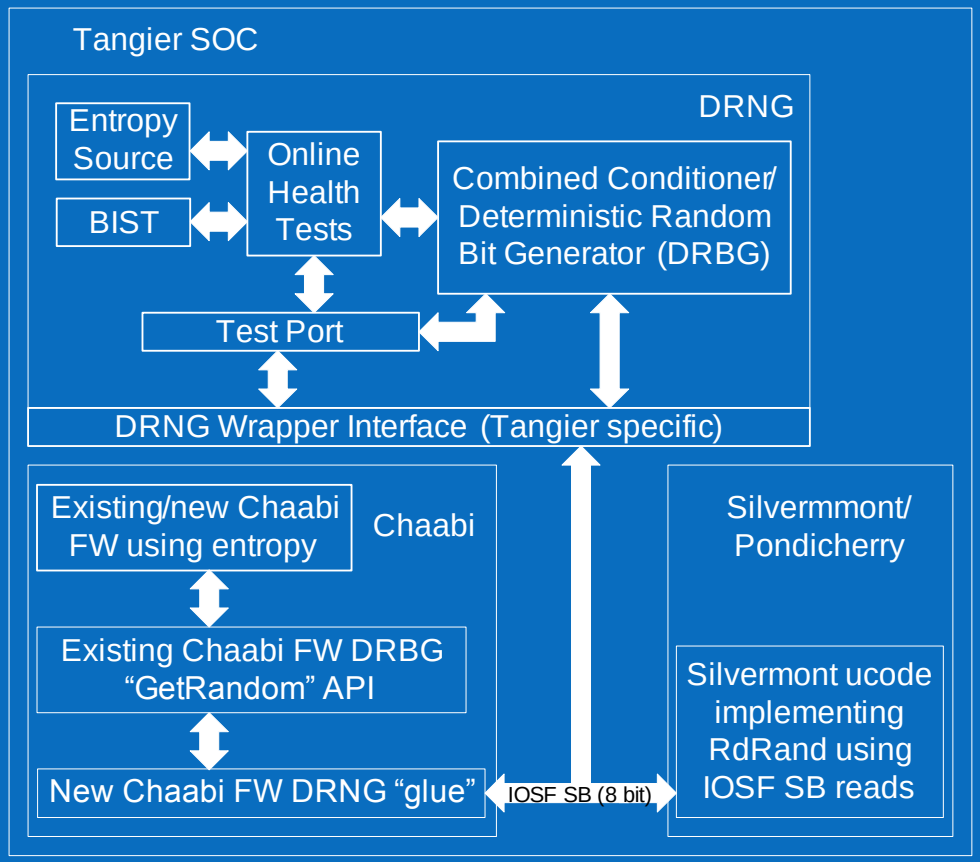


Valley View DRNG Embedding

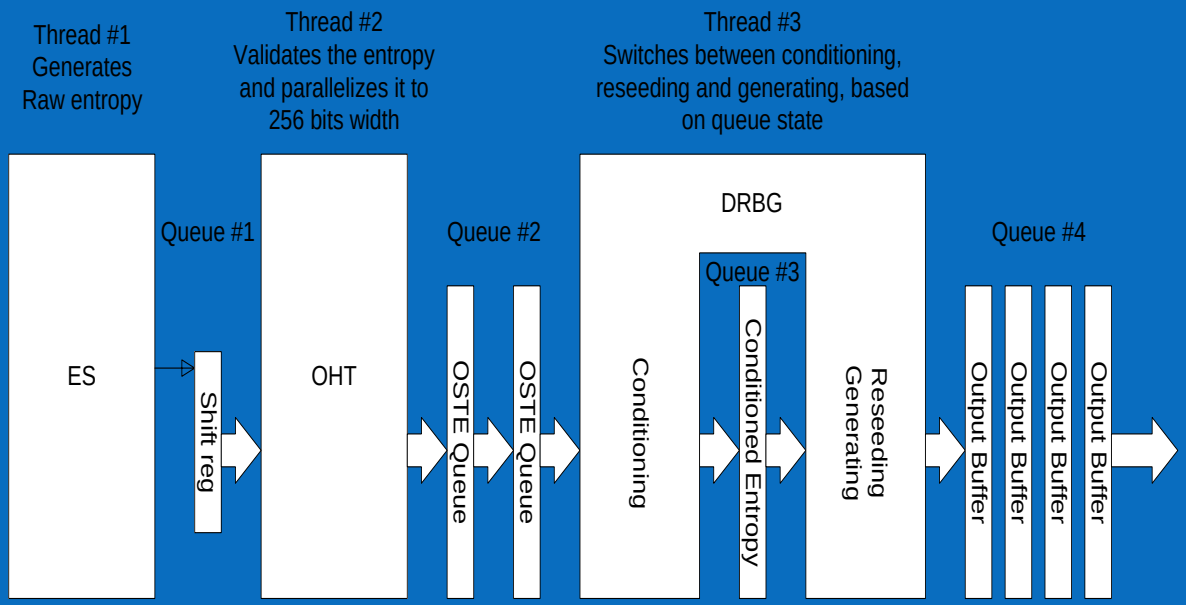


Valley View SOC not currently intending to have SEC use DRNG

Tangier DRNG Embedding



Temporal Asynchrony Between Subunits

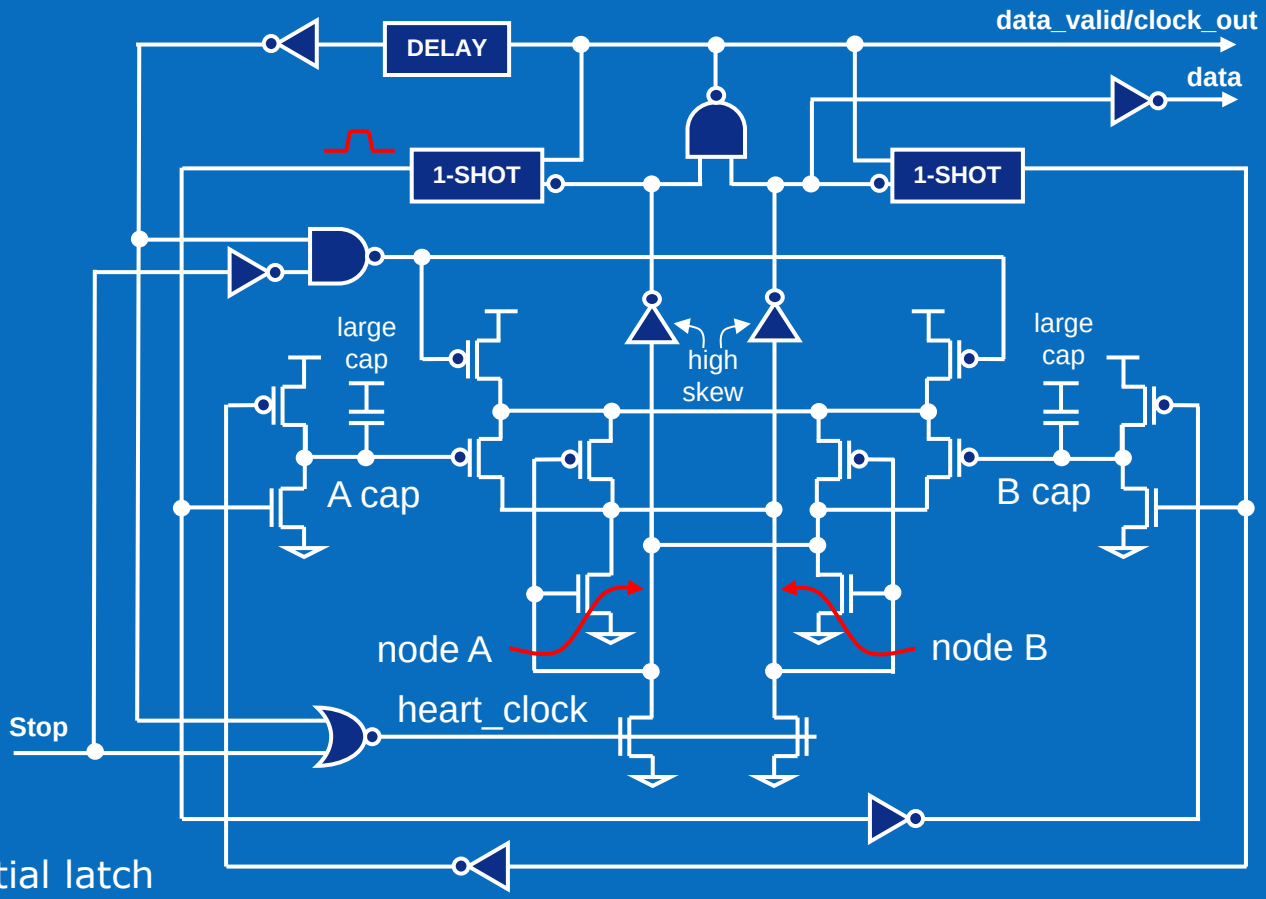


Our DRNG is logically a three stage/subunit asynchronous production pipeline (composed of the Entropy Source, Online Health Test, and Conditioner and DRBG)

For "flow control" purposes, these subunits each have what amounts to an "output queue" between them and their nearest neighbor in the DRNG production sequence

Depending on the subunit production rate and the next subunit consumption rate, unpredictable dynamic synchronization behaviors ensue

Entropy Source For 1269 Process (MPS1)



Differential latch

Push into metastable state with resolution driven by thermal noise

Dynamic, bilateral, step-based feedback loop to deal with any circuit bias

Designed to be stable across process, temperature, and voltage/power variations