**[Overview](https://opentitan.org/book/hw/ip/aon_timer/index.html" \l "overview)**

This document specifies ENTROPY\_SRC hardware IP functionality. This module conforms to the [Comportable guideline for peripheral functionality.](https://opentitan.org/book/doc/contributing/hw/comportability/index.html)

[**Features**](https://opentitan.org/book/hw/ip/entropy_src/index.html#features)

* This revision provides an interface to an external physical random noise generator (also referred to as a physical true random number generator or PTRNG). The PTRNG external source is a physical true random noise source. A noise source and its relation to an entropy source are defined by [SP 800-90B.](https://csrc.nist.gov/publications/detail/sp/800-90b/final)
* A set of registers is provided for firmware to obtain entropy bits.
* Interrupts are supported:
  + Entropy bits are available for firmware consumption.
  + The internal health tests have detected a test failure.
  + An internal FIFO error has occurred.
* Two health checks that are defined by SP 800-90B are performed by this revision: Repetition Count and Adaptive Proportion tests.
* Two additional hardware health checks are performed by this revision: Bucket and Markov tests.
* Firmware-defined (mailbox-based) and vendor-defined health checks are also supported.
* A health check failure alert is supported by this revision.

[**Description**](https://opentitan.org/book/hw/ip/entropy_src/index.html#description)

This IP block provides an entropy source that is capable of using a PTRNG (Physical True Random Number Generator) noise source to generate random values in a manner that is compliant both with FIPS (though NIST SP 800-90B) and CC (AIS31) recommendations.

At a high-level, the ENTROPY\_SRC block, when enabled, will continuously collect entropy bits from the PTRNG noise source, perform health tests on the collected entropy bits, pack them, send them through a conditioning unit, and finally store them into a FIFO. The content of this FIFO can either be sent out through a hardware interface or alternatively (but not simultaneously) read by firmware over the TL-UL bus. The random values generated by the ENTROPY\_SRC block serve as non-deterministic seeds for the CSRNG block. The outputs of the CSRNG are then used either directly by firmware or are distributed to other hardware blocks through the Entropy Distribution Network.

In the terms of AIS31 classes, this block is meant to satisfy the requirements for a PTG.2 class physical entropy source, with “internal” entropy (an AIS31 term, meaning the min-entropy as measured just *before the output pins*) exceeding 0.997 entropy-bits/output-bit. In NIST terms, this block satisfies the requirements for a “full-entropy” source, which requires even smaller deviations from ideal entropy, at the level of less than one part in 264.

The raw noise bits from the PTRNG noise source (external to this block) are subjected to a sequence of health-checks to screen the raw signals for statistical defects which would cause any significant deviations from ideal entropy at the output of the conditioning block. These tests include:

* The Repetition Count test, which screens for stuck-bits, or a complete failure of the PTRNG noise source,
* The Adaptive Proportion test, which screens for statistical bias in the number of 1’s or 0’s output by the noise source,
* The “Bucket Test”, which looks for correlations between the individual noise channels that the external noise source concatenates together to produce the raw noise sequence,
* The “Markov Test”, which looks for unexpected first-order temporal correlations between bits output by the individual noise channels,
* The “Mailbox Test”, in which raw-noise data is transferred to firmware in contiguous blocks, so that firmware can perform custom tests, and signal a failure through the same path as the other tests, and
* Optional Vendor Specific tests, which allow silicon creators to extend the health checks by adding a top-level block external to this IP.

The Repetition Count and Adaptive Proportion test are specifically recommended by SP 800-90B, and are implemented in accordance with those recommendations. In FIPS/CC compliant mode, all checks except the Repetition Count test are performed on a fixed window of data of configurable size, by default consisting of 2048 bits each. Per the definition in SP 800-90B, the Repetition Count test does not operate on a fixed window. The repetition count test fails if any sequence of bits continuously asserts the same value for too many samples, as determined by the programmable threshold, regardless of whether that sequence crosses any window boundaries. The thresholds for these tests should be chosen to achieve a low false-positive rate (α) given a conservative estimate of the manufacturing tolerances of the PTRNG noise source. The combined choice of threshold and window size then determine the false-negative rate (β), or the probability of missing statistical defects at any particular magnitude.

When the IP is disabled by clearing the [MODULE\_ENABLE](https://opentitan.org/book/hw/ip/entropy_src/doc/registers.html#MODULE_ENABLE) register, all health checks are disabled and all counters internal to the health checks are reset.

When operating in FIPS/CC compliant mode, the raw outputs of the PTRNG noise source are passed through a conditioning function based on a suitable secure hash function (SHA-3) which has been vetted by NIST to meet the stringent requirements for a full-entropy source. In order to compensate for the fact our tests (like *all* realistic statistical tests) have finite resolution for detecting defects, we conservatively use 2048 bits of PTRNG noise source to construct each 384 bit conditioned entropy sample by default. The effectively used number of bits is equal to the configured health test window size. When passed through the conditioning block, the resultant entropy stream will be full entropy unless the PTRNG noise source has encountered some statistical defect serious enough to reduce the raw min-entropy to a level below 0.375 bits of entropy per output bit. We choose this level as our definition of “non-tolerable statistical defects” for the purposes of evaluating this system under AIS31. Given this definition of “non-tolerable defects”, the health-checks as implemented for this block will almost certainly detect any of the previously mentioned defects in a single iteration of the health checks (i.e. such serious defects will be detected with very low β).

In addition to the brief, low-latency health checks, various long-term statistics are accumulated in registers for additional diagnostic purposes or for in-depth analysis.

There are two modes in which entropy bits are delivered, boot-time / bypass mode and FIPS/CC compliant mode. Boot-time / bypass mode will deliver bits sooner for specific on-boot obfuscation applications, though the bits may not yet have been subjected to the same level of startup health checks required for FIPS or CC compliance.

In boot-time / bypass mode health checks only operate on a window of 384 bits. The boot-time / bypass mode health checks are the same as the FIPS/CC health-checks, though with different thresholds. They are sensitive to the same types of statistical defects, though at reduced statistical resolution. With suitable thresholds, the boot-time health checks can be operate both with low false-alarm rates (low α), while still confirming with low β that the total entropy of the first seed contains at least 80 bits of total entropy. During start up the initial 384 bits are held in a buffer until the boot-time start-up health checks are performed. Storing the seed in this buffer, allows this seed to released to the CSRNG immediately after the entropy has been confirmed.

Boot-time / bypass mode also has the feature that it bypasses the SHA conditioning function, as only 384 bits are used in the initial boot-time seed.

For maximal flexibility in normal operation, the conditioning function can also be implemented by firmware. When this firmware conditioning feature is activated, data read directly out of the noise source can be re-inserted into the entropy pipeline via a TL-UL register after it has been processed by firmware. It should be noted that this firmware algorithm must be vetted by NIST to satisfy the requirements for a full-entropy source. This feature can also be disabled for security purposes, either by locking the feature via the [REGWEN](https://opentitan.org/book/hw/ip/entropy_src/doc/registers.html#regwen) register at boot, or by a write to one-time programmable (OTP) memory.

[**Compatibility**](https://opentitan.org/book/hw/ip/entropy_src/index.html#compatibility)

This IP block does not have any direct hardware compatibility requirements. However, the general design of this block follows the overall NIST recommendations, as described by SP 800-90B.

**Theory of Operation**

The ENTROPY\_SRC hardware block collects entropy bits from the PTRNG (Physical True Random Number Generator) noise source, performs health tests on the collected entropy bits, packs them, sends them through a conditioning unit, and finally stores them into the esfinal FIFO for consumption by firmware or hardware.

It operates in a manner compliant with both FIPS (though NIST SP 800-90B) and CC (AIS31) recommendations. This revision supports only an external interface for a PTRNG noise source implementation.

**Initialization and Enabling**

After power-up, the ENTROPY\_SRC block is disabled. The first step is initialization and enabling.

For simplicity of initialization, only a single write operation to the [MODULE\_ENABLE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#module_enable) register is needed to start functional operation of the ENTROPY\_SRC block. This assumes that proper defaults are chosen for the health test thresholds and other registers such as the [CONF](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#conf) register.

Once the ENTROPY\_SRC block is enabled, also the PTRNG noise source starts up. The ENTROPY\_SRC block will then start to collect entropy bits indefinitely until disabled.

**Boot-Time / Bypass Mode**

After a reset, the ENTROPY\_SRC block will start up in boot-time / bypass mode by default. This feature is designed to provide an initial seed's worth of entropy with lower latency than the normal FIPS/CC compliant health check process. Health testing will still be performed on boot-time mode entropy, but the window of checking is, by default, 384 bits instead of 2048 bits.

**FIPS/CC Compliant Mode**

Once the initial boot-time mode phase has completed, the ENTROPY\_SRC block can be switched to FIPS/CC compliant mode (for simplicity referred to as FIPS mode) by setting the FIPS\_ENABLE field in the [CONF](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#conf) register to kMultiBitBool4True. In this mode, once the raw entropy has been health checked, it will be passed into a conditioner block. This block will compress the bits such that the entropy bits/physical bits, or min-entropy value, should be improved over the raw data source min-entropy value. The compression operation will compress every [HEALTH\_TEST\_WINDOWS.FIPS\_WINDOW](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#health_test_windows--fips_window) x 4 tested bits into 384 full-entropy bits. By default, 2048 tested bits are used. Note that a seed is only produced if the last [HEALTH\_TEST\_WINDOWS.FIPS\_WINDOW](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#health_test_windows--fips_window) x 4 tested bits have passed the health tests. If a health test fails, the conditioner block continues absorbing the next window unless [ALERT\_SUMMARY\_FAIL\_COUNTS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#alert_summary_fail_counts) reaches the configured [ALERT\_THRESHOLD](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#alert_threshold). Once the threshold is reached, the ENTROPY\_SRC block stops serving entropy and signals a recoverable alert. Firmware then needs to disable/re-enable the block to restart operation.

When entropy is delivered to the downstream hardware block, a signal will indicate what type of entropy it is - FIPS/CC compliant or not. This signal is determined by the FIPS\_FLAG field in the [CONF](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#conf) register. When RNG\_FIPS field in the [CONF](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#conf) register is set to kMultiBitBool4True, the ENTROPY\_SRC block will request higher quality entropy from the noise source by asserting the rng\_fips output signal.

**Startup Health Testing**

Note that after enabling the ENTROPY\_SRC block, the health tests need to pass for two subsequent windows of [HEALTH\_TEST\_WINDOWS.FIPS\_WINDOW](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#health_test_windows--fips_window) x 4 tested bits (startup health testing). By default, 1024 samples of 4 bits (4096 1-bit samples when running in single-channel mode), i.e., 4096 tested bits, are used for producing the startup seed. If a health test fails, the startup health testing starts over and the conditioner block continues absorbing the next window. If the health tests don't pass for two subsequent windows, the ENTROPY\_SRC block stops operating and signals a recoverable alert. Firmware then needs to disable/re-enable the block to restart operation including the startup health testing.

**Firmware Override Mode**

The hardware conditioning can also be bypassed and replaced in normal operation with a firmware-defined conditioning algorithm. This firmware conditioning algorithm can be disabled on boot for security purposes.

The firmware override function has the capability to completely override the hardware health tests and the conditioner paths. In the case of health tests, firmware can turn off one or all of the health tests and perform the tests in firmware. A data path is provided in the hardware such that the inbound entropy can be trapped in the observe FIFO. Once a pre-determined threshold of entropy has been reached in this FIFO, and interrupt is raised to let firmware read the entropy bits out of the FIFO. The exact mechanism for this functionality starts with setting the [FW\_OV\_MODE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_control--fw_ov_mode) field in the [FW\_OV\_CONTROL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_control) register. This will enable firmware to monitor post-health test entropy bits (including entropy bits used for startup health testing) by reading from the [FW\_OV\_RD\_DATA](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_rd_data) register. Firmware can use the [OBSERVE\_FIFO\_THRESH](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#observe_fifo_thresh) and [OBSERVE\_FIFO\_DEPTH](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#observe_fifo_depth) to determine the state of the OBSERVE FIFO. At this point, firmware can do additional health checks on the entropy. Optionally, firmware can do the conditioning function, assuming the hardware is configured to bypass the conditioner block. Once firmware has processed the entropy, it can then write the results back into the [FW\_OV\_WR\_DATA](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_wr_data) register (pre-conditioner FIFO). The [FW\_OV\_ENTROPY\_INSERT](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_control--fw_ov_entropy_insert) in the [FW\_OV\_CONTROL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_control) register will enable inserting entropy bits back into the entropy flow. If enabled, any startup health testing has to be performed by the firmware.

An additional feature of the firmware override function is to insert entropy bits into the flow and still use the conditioning function in the hardware. Setting the [FW\_OV\_INSERT\_START](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_sha3_start--fw_ov_insert_start) field in the [FW\_OV\_SHA3\_START](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_sha3_start) register will prepare the hardware for this flow. Once this field is set true, the [FW\_OV\_WR\_DATA](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_wr_data) register can be written with entropy bits. The [FW\_OV\_WR\_FIFO\_FULL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_wr_fifo_full) register should be monitored after each write to ensure data is not dropped. Once all of the data has been written, the [FW\_OV\_INSERT\_START](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_sha3_start--fw_ov_insert_start) field should be set to false. The normal SHA3 processing will continue and finally push the conditioned entropy through the module.

For more details, refer to the [programmer's guide](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/programmers_guide.md/#firmware-override--bypass-modes).

**Health Tests**

Health checks are performed on the input raw data from the PTRNG noise source when in that mode. There are four health tests that will be performed: repetitive count, adaptive proportion, bucket, and Markov tests. Each test has a pair of threshold values that determine that pass/fail of the test, one threshold for boot-time / bypass mode, and one for FIPS mode. By default, all tests are enabled, but can be turned off by setting the thresholds to the maximum value. Because of the variability of the PTRNG noise source, there are several registers that log statistics associated with the health tests. For example, the adaptive proportion test has a high watermark register that logs the highest measured number of ones. The [ADAPTP\_HI\_WATERMARKS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#adaptp_hi_watermarks) register has an entry for both FIPS and boot-time modes. This register allows for determining how close the threshold value should be set to the fail over value. Specific to the adaptive proportion test, there is also the [ADAPTP\_LO\_WATERMARKS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#adaptp_lo_watermarks) register, which will hold the lowest number of ones measured. To help understand how well the thresholds work through time, a running count of test fails is kept in the [ADAPTP\_HI\_TOTAL\_FAILS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#adaptp_hi_total_fails) register. The above example for the adaptive proportion test also applies to the other health tests, with the exception of the low watermark registers. See the timing diagrams below for more details on how the health tests work. It should be noted that for all error counter registers, they are sized for 16 bits, which prevents any case where counters might wrap.

Additional, vendor-specific tests are supported through an [external health test interface (xht)](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/theory_of_operation.md#external-health-tests).

**Health Test Failure Alert**

The [ALERT\_THRESHOLD](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#alert_threshold) register determines during how many subsequent health test windows one or more health test failures can occur before a recoverable alert is raised and the ENTROPY\_SRC block stops operating. In case this threshold is reached, firmware needs to disable/re-enable the block to restart operation including the startup health testing. By default, the threshold is set to two, such that the occurrence of two failing test cycles back-to-back would provide a very low α value. When reaching the threshold while running in [Firmware Override: Extract & Insert mode](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/programmers_guide.md#firmware_override_-_extract_&_insert), the recoverable alert is not raised nor does the block stop operating. In other modes, the generation of the recoverable alert can be disabled by configuring a value of zero.

The [ALERT\_SUMMARY\_FAIL\_COUNTS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#alert_summary_fail_counts) register holds the total number of health test windows during which one or more health test failures occurred. The register is automatically cleared after every passing health test window unless the ENTROPY\_SRC is configured in [Firmware Override: Extract & Insert mode](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/programmers_guide.md#firmware_override_-_extract_&_insert).

The [ALERT\_FAIL\_COUNTS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#alert_fail_counts) reports the number of health test failures since the last passing health test window on a per-test basis. If multiple health tests fail for a certain window, the value in [ALERT\_SUMMARY\_FAIL\_COUNTS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#alert_summary_fail_counts) is incremented by just one whereas multiple fields in [ALERT\_FAIL\_COUNTS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#alert_fail_counts) may get incremented. The same holds for [EXTHT\_FAIL\_COUNTS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#extht_fail_counts) which reports the number of external health test failures since the last passing health test window on a per-test basis.

**Routing Entropy to Firmware**

Firmware has a path to read entropy from the ENTROPY\_SRC block. The [ES\_ROUTE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#entropy_control--es_route) field in the [ENTROPY\_CONTROL](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#entropy_control) register allows firmware to set the internal multiplexers to steer entropy data to the [ENTROPY\_DATA](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#entropy_data) register. When enabled, no entropy is being passed to the block hardware interface. The control field [ES\_TYPE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#entropy_control--es_type) sets whether the entropy will come from the conditioning block or be sourced through the bypass path. A status bit will be set that can either be polled or generate an interrupt when the entropy bits are available to be read from the [ENTROPY\_DATA](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#entropy_data) register. The firmware needs to read the [ENTROPY\_DATA](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#entropy_data) register twelve times in order to cleanly evacuate the 384-bit seed from the hardware path (12 \* 32 bits = 384 bits in total).

**Disabling**

At any time, the [MODULE\_ENABLE](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#module_enable) field can be cleared to halt the entropy generation and health testing. See the [programmer's guide section](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/programmers_guide.md/#entropy-source-module-disable) for more details on the ENTROPY\_SRC block disable sequence.

**Design Details**

**Entropy Dropping and Conditioner Back Pressure**

When enabled, the ENTROPY\_SRC block has no way to exercise back pressure onto the PTRNG noise source. Any sample coming in from the noise source is being health tested and continues to flow down the entropy pipeline. However, if the ENTROPY\_SRC block experiences internal back pressure, health tested samples may be dropped in different locations depending on the source of the back pressure and the configured mode of operation.

* **Input of esfinal FIFO**: If the esfinal FIFO is full, e.g., because the hardware entropy interface doesn't request entropy, entire 384-bit seeds coming out of the conditioner or the bypass packer FIFO get dropped. Firmware is not informed when dropping entire seeds. When running in [Firmware Override: Extract & Insert mode](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/programmers_guide.md#firmware_override_-_extract_&_insert), firmware may want to infer whether the next seed is likely going to be dropped at this moment by reading the current depth of the esfinal FIFO from the [DEBUG\_STATUS](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#debug_status) register.
* **Input of Observe FIFO**: When running in any of the [Firmware Override modes](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/programmers_guide.md#firmware_override_/_bypass_modes) health tested entropy bits are collected in the Observe FIFO for observation / extraction by firmware. Whenever the Observe FIFO runs full, the [FW\_OV\_RD\_FIFO\_OVERFLOW](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#fw_ov_rd_fifo_overflow) bit is asserted and the hardware stops collecting entropy bits in the Observe FIFO until firmware has emptied the FIFO. This helps ensuring that observed / extracted entropy bits are contiguous.
* **Input of postht / esbit FIFO**: When experiencing back pressure from the hardware conditioner (see below), health tested entropy bits may get dropped at the input of the postht or esbit FIFO when running in multi-channel or single-channel mode, respectively. Note that the dropped bits are still health tested and contribute to the overall results for the current window. However, to keep the number of bits passed to the conditioner constant, the health test window is stretched by the number of dropped samples. Whenever post-health test entropy bits are being dropped from the hardware pipeline, a recoverable alert is triggered and the [RECOV\_ALERT\_STS.POSTHT\_ENTROPY\_DROP\_ALERT](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#recov_alert_sts--postht_entropy_drop_alert) bit is asserted. Note that this may also happen in [Firmware Override: Observe mode](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/programmers_guide.md#firmware_override_-_observe). Firmware should thus explicitly check the [RECOV\_ALERT\_STS.POSTHT\_ENTROPY\_DROP\_ALERT](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#recov_alert_sts--postht_entropy_drop_alert) bit to ensure the bits retrieved from the Observe FIFO are indeed contiguous.

The reduce the probability of dropping post-health test entropy bits, the **Distribution FIFO** can be used. This FIFO has pass-through mode enabled meaning it doesn't add latency to hardware pipeline. It has a width of 32 bits. Its depth is configurable via compile-time Verilog parameter and should match the expected level of conditioner back pressure. The level of conditioner back pressure depends on the following factors:

* The maximum latency for the conditioner to add the padding bits \(n\_{pad}\) (25 clock cycles) and to run the internal SHA3 primitive \(n\_{sha3}\) (24 clock cycles).
* The maximum latency of the [CS AES halt request interface](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/interfaces.md/#inter-module-signals) \(n\_{halt}\) (48 clock cycles corresponding to CSRNG performing the Update function).

The required depth \(d\_{distr}\) of the Distribution FIFO can be computed as $$ d\_{distr} = { (r\_{ptrng} \* s\_{symbol}) \* (2 \* (n\_{halt} + n\_{sha3}) + n\_{pad} + n\_{uarch}) - 32 - 64 \over 32} $$

where

* \(r\_{ptrng}\) is the rate at which the PTRNG noise source generates entropy samples,
* \(s\_{symbol}\) (= 4) is the symbol size of these samples in bits,
* \(n\_{uarch}\) (= 5) is the latency of the ENTROPY\_SRC micro architecture between producing seeds, and
* the -32 and -64 terms are to account for the number of entropy bits stored in the postht and the precon FIFOs, respectively.

For [Top Earlgrey](https://github.com/lowRISC/opentitan/blob/master/hw/top_earlgrey/README.md), the assumption is that the PTRNG noise source generates entropy bits at a rate of roughly 50 kbps. With the ENTROPY\_SRC block running at 100 MHz, this leads to noise source rate \(r\_{ptrng}\) = 1/8000.

The noise source model inside the DV environment generates symbols with an average rate of 1 4-bit symbol every 6.5 clock cycles. To reach functional coverage metrics, the entropy\_src\_rng\_max\_rate configures the noise source to generate a 4-bit symbol every other clock cycle (\(r\_{ptrng}\) = 1/2) an the CS AES halt request interface to always respond immediately (\(n\_{halt}\) = 0). With these settings, the ENTROPY\_SRC block should never drop samples due to conditioner back pressure if a depth of two is chosen for the Distribution FIFO (\(d\_{distr}\) = 2).

**Security**

All module assets and countermeasures performed by hardware are listed in the [countermeasures section](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/interfaces.md/#security-countermeasures). Labels for each instance of asset and countermeasure are located throughout the RTL source code.

A configuration and control register locking function is performed by the [REGWEN](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#regwen) register. Clearing the bit in this register will prevent future modification of the [CONF](https://github.com/lowRISC/opentitan/blob/master/hw/ip/entropy_src/doc/registers.md#conf) register or other writeable registers by firmware.

For all of the health test threshold registers, these registers could be protected with shadow registers. A design choice was made here to not use shadow registers and save on silicon cost. The threshold registers are protected by software. It is expected that software will read the threshold registers on a periodic basis, and compare these values to what was originally programmed into the threshold registers.

Bus integrity checking is performed for the final seed delivery to CSRNG. This is done to make sure repeated values are not occurring. Only 64 bits (out of 384 bits) are checked, since this is statistically significant, and more checking would cost more silicon.