AES Random Number Generator Core, Xilinx Edition

February 10, 2015 Product Specification

Algotronix®
130-10 Calton Road
Edinburgh, Scotland
United Kingdom, EH8 8JQ
Phone: +44 131 556 9242
E-mail: cores@algotronix.com
URL: www.algotronix.com

Core Facts

Provided with Core	
Documentation	User Manual
Design File Formats	VHDL
Verification	Test Bench, Test Vectors
Instantiation templates	VHDL
Simulation Tool Used	
Model Tech ModelSim, Xilinx ISim	
Support	
Support provided by Algotronix	

Features

- Implementation of the CTR-DRBG option using the AES cipher in Draft NIST SP-800-90A, Rev 1 (Nov 2014) "Recommendation for Random Number Generation Using Deterministic Random Bit Generators".
- Supports Block Cipher Derivation Function (DF).
- Supports 128, 192 and 256 bit AES keys
- Targets all modern FPGA families from Xilinx.
- Based on Algotronix AES-G3 core with a 32 bit data bus width. Can be combined with other AES-G3 based cores such as Keywrap to create a security substystem.
- Supplied as easily customizable portable VHDL to allow customers to conduct their own code review in high-security applications. Compilation options to include only required features and save area
- Supplied with comprehensive test bench implementing NIST DRBGVS tests and Algotronix developed directed random tests

Applications

- Generation of Critical Security Parameters (CSP) such as cryptographic keys
- Gaming Machines

General Description

AES-RNG is an implementation of the latest (Nov 2014) draft of the NIST SP-800-90A standard which specifies a way of processing input from a noise source to create random bit streams suitable for use as cryptographic keys or other Critical Security Parameters. The standard was updated following a controversy over the parameters used in the elliptic curve based variant in earlier versions of the standard. This widely reported weakness is not relevant to this AES based core (the controversial Elliptic Curve variant has been removed from the Nov 2014 draft of the standard). An additional advantage of using AES in the Random Number Generator in systems where AES is used for data privacy is that the system is only trusting one cryptographic algorithm.

To support highly sensitive applications the Algotronix AES-RNG core is supplied as VHDL source code allowing customers to carry out a code review to convince themselves no 'Trojan horse' circuitry has been added to the core which could compromise its security.

The core is supplied with a comprehensive test bench implementing a self testing version of the synthesisable core which compares its outputs with a behavioral model developed from the pseudo-code in the standard when stimulated with random test vectors as well as using the Known Answer Tests available on the NIST website.

Table 1 shows example implementation statistics for the core in the configuration used for testing with NIST vectors. There are many implementation options available, some of which have a significant effect on area and performance. The AES-RNG core can be targetted at all Xilinx FPGA families, including older devices not listed in the table below. Algotronix will supply implementation results for the particular core configuration required in your application on request.

Table 1: Example Implementation Statistics for AES-RNG, 'Push Button' flow with Fmax specified by clock constraint. 128 bit AES Key, 128 bit Entropy, 128 bit personalization string, 128 bit additional input, 64 bit nonce, 512 bit requested bit output.

Family	Example Device	Fmax (MHz)	LUT	Memory LUT	IOB[1]	GCLK[2]	BRAM	MULT/ DSP48	DCM	Design Tools
Zynq	XC7Z020-3	200	1284	80	177	1	2.5	0	0	Vivado14.4

Notes:

1) IOB count when all core I/Os and clocks are routed off-chip, which is **not** the intended usage. The core interface is designed to provide flexibility inside a larger FPGA design.

2) GClk signal is normally shared with user's design.

Functional Description

The Algotronix AES-RNG core is aimed at providing ease of use and sufficient performance with good area efficiency for mainstream applications. The core is a Deterministic Random Bit Generator (DRBG) based on the Advanced Encryption Algorithm (AES) in counter mode as opposed to a non-deterministic or True Random Bit Generator (TRBG) based on a real-world noise source. The outputs of a DRBG are completely determined by the input signals but in a way which makes it very difficult to predict previous or future values without knowledge of secret information. Unlike a True Random Bit Generator the core is not subject to physical attacks such as interfering with the power supply voltage or temperature in order to try and reduce randomness and will not have statistical bias in the output values it can also provide high quality random values more frequently than many TRBGs. Usually the DRBG core is seeded with entropy provided by a TRBG and the combination of the two provides the best of both worlds.

This section should be read in conjunction with Chapter 8 of the SP800-90A standard which provides a more formal description of the various inputs to the DRBG and describes important security considerations. It is important to understand and follow the guidance in the SP800-90A standard in order to use this AES-RNG core in a secure way.

The main input to the DRBG core is the entropy, which normally comes from an analogue-domain noise source (on an FPGA the obvious choice is a ring-oscillator based noise source). In some configurations the entropy is taken from another DRBG. The entropy input to the DRBG must be kept secret: if an attacker has access to the entropy being provided to the DRBG security is compromised. The companion draft standard SP800-90B sets out some requirements and design considerations for the entropy source. The interface to the entropy source needs to take account of the possibilty that entropy may not be immediately available and that monitoring circuits in the entropy source may report an error.

The Nonce input is intended to make sure that subsequent instantiations of the DBRG will provide a completely different sequence of random bits even if the entropy inputs are identical. Without a nonce an attacker could potentially use data collected across many instantiations to try and compromise the RNG. The nonce could be a timestamp or a monotonically increasing counter or a random value from an approved random bit generator. The important thing is that each time the RNG is given an instantiate command a different nonce is provided. The nonce does not have to be secret but it must be protected from tampering because the benefit of a nonce is lost if an attacker could prevent the nonce taking a new value for each instantiation.

The Personalization String input is an optional input intended to encode a string of bits which is unique to an instantiation of the DBRG. If the Nonce is used to encode a timestamp the Personalization String could include a unique device identifier such as a serial number or network address. This ensures that two identical copies of the hardware containing the DBRG will generate completely different random values and makes it pointless to try and use one copy of the hardware to try and predict the outcome of another. The DBRG will summarise a long personalization string into the seed so if resources and performance constraints allow quite long personalization strings composed of several different values (network ids, timestamps, serial numbers etc) can be used.

Additional Input is an optional source of additional entropy. Additional input could be connected to various nodes within the larger user design whose value is believed to be unpredictable. It should not be connected to signals whose value is secret and requires to be protected at a security strength greater than the DRBG (e.g. if you have data that needs to be protected with 256 bit AES encryption it should not be fed into a DRBG based on 128 bit AES encryption). Additional input should be kept secret and provides an extra layer of protection if for some reason the main TRBG providing entropy input is compromised or the value of the seed within the DRBG is leaked. There are usually many places in an FPGA design where signals are expected to have difficult to predict values but where it would be difficult to qualify them as a randomness source to the extent needed to trust them as the primary TRNG. Additional Input allows this kind of additional randomness to be added into the mix on the 'if it does no good it will do no harm principle'.

Random Number Generators used for creating cryptographic keys and other critical security parameters are a primary weakpoint in most cryptographic systems and a frequent target of attack. The DRBG with Nonce, Entropy, Personalization String and Additional Input can be looked on as a belt-and-braces approach to make sure that attacking the system by trying to compromise the RNG is as hard as compromising the algorithm used to encrypt the data. Where AES is also used for data encryption or key exchange (AES-Keywrap) there may be a possibility to share the AES core across several functions to save area.

The main functional blocks as shown in Figure 1.

AES-G3

This block instantiates the AES-G3 core in ECB Encrypt mode. As noted above it would be possible to customise the AES-RNG core so the AES-G3 block could be shared with other parts of the user design which require AES.

RNG State

This block contains the registers for the Key and V variables of the RNG, buffers for the RNG seed and storage required by the DF and BCC operations as well as state machines to implement the lower level RNG algorithms.

RNG Function

This block contains a state machine to implement the Instantiate, Generate, Reseed and Uninstantiate operations specified in the standard using the lower level resources provided by RNG State.

RNG Health Check

This block contains logic to isolate the RNG from external signals and conduct a known answer test based health check. This is a relatively complex set of tests as specified in section 11.3 of the SP800-90A standard and is initiated after a reset (the assumption is that the system will assert reset on power on) or in response to the health_check_request signal. As required by the standard if the health check fails the core will assert catastrophic_error and lock up, not responding to any commands until the condition is lifted by user intervention (in this case user intervention would involve asserting reset by means of cycling the power or otherwise).

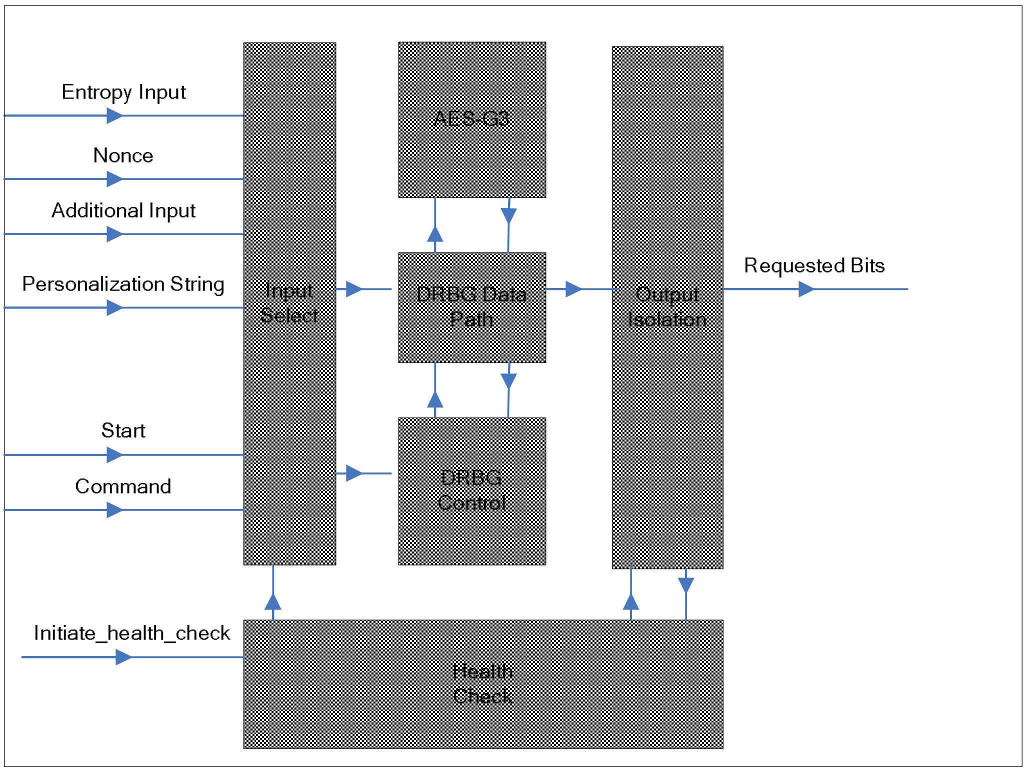

Figure 1, AES RNG Core Block Diagram

Compilation Options

The core can be configured easily using a set of VHDL generic parameters. Normally, it is unnecessary for users to modify the design source code although the code is supplied and they are free to do so if they wish. Algotronix can also customise the core as a service for users with particular requirements which are not met by the standard product.

The parameterisable version of the core is declared in the file aes_cipher.vhd. The compilation parameters are then specified in the file aes_rng_parameters_package and applied in the file aes_rng_cipher_wrapper – the result is a fully parameterized AES core with no generic parameters at the top level of the design hierarchy which can be convenient, for example in mixed language designs.

The following compilation options are specified by editing constant definitions in the aes_rng_parameters_package.vhd file (there are no compilation options corresponding to reseed_supported and df_supported in the standard: the core supports both these options and does not provide an option to disable them so these would always be true). The aes_rng_parameters_package.vhd file is the only source code in the AES core release which will normally be edited by the user.

- **aes_key_size** – specifies the maximum key length the core should implement. The user can select any key length up to and including this using control signals. For example, if max_crypt_size is aes256 then the core would deal with 256, 192 and 128 bit keys.

- **prediction_resistance_supported** - when true the input signal prediction_resistance_request is signficant and determines whether the core will reseed in every generate operation making it harder to predict the next set of generated bits based on the previous ones.
- **number_of_requests_between_reseeds** - number of times the generate operation may be used before the core automatically does a reseed.
- **entropy_input_length** - length of the entropy input, must be a multiple of 32 bits.
- **personalization_string_length** - length of the personalization string input, must be a multiple of 32 bits.
- **additional_input_length** - length of the additional input, must be a multiple of 32 bits.
- **nonce_length** - length of the nonce, must be a multiple of 32 bits.
- **requested_number_of_bits** - number of bits provided by a generate operation, must be a multiple of 32 bits.
- **implement_sboxes_in_ram** – specifies that FPGA RAM blocks rather than logic gates should be used to implement SBoxes and Inverse SBoxes. This is the most efficient option if RAM blocks are available after mapping the remainder of the user design.
- **target_device** – In the Xilinx Edition version of the core only Xilinx devices are supported. Target device should be set to XILINX for legacy devices with 4 input LUTs or XILINX_VIRTEX_5 for newer devices with 6 input LUTs.

When the compilation options are changed it is also necessary to update the aes_rng_health_check.vhd source code file which contains a table of computed known-answer data for the health check. The expected results for the AES-RNG operations tested in the health check change when the compilation parameters are altered so they need to be re-computed. This is done using an option in the aes_rng_testbench which creates a text file of expected data. This expected data then needs to be copied into the aes_rng_health_check.vhd file where it will be compiled into a Read Only Memory used by the health check hardware. The easiest approach is to provide Algotronix with the configuration information when ordering the core and it will be supplied in a Xilinx project configured and ready for use.

If any of the compilation parameters aes_key_size, prediction_resistance_supported, entropy_input_lengthm personalization_string_length, additional_input_length, nonce_length and requested_number_of_bits are changed without the corresponding update to the expected results in the health check code the core will detect a health check error after the initial reset and lock up with a catastrophic error.

Core I/O Signals

The core signal I/O have not been fixed to specific FPGA device pins to provide flexibility for interfacing with user logic. Descriptions of all signal I/O are provided in Table 2.

Signal	Signal Direction	Description
clock	input	Clock – active on rising edge
reset	input	Reset – active high. A global constant in aes_package.vhd USE_ASYNCHRONOUS_RESET determines whether this is a synchronous or asynchronous reset. On Xilinx FPGAs a synchronous reset is usually more efficient.
enable	input	Module clock enable – 0: module is inactive, 1: module operates
initiate_health_check	input	Initiates the health check which also occurs immediately after reset. The health check may last several hundred clocks as multiple known-answer tests are required to cover all the requirements specificed in the standard. While the health check is running inputs to the core execpt for clock, reset and enable are ignored. The health check will uninstantiate the present state of the RNG. If the health check fails the core will be left in the catastrophic error condition and require a reset or power cycle to restore normal function (it is a requirement of the standard that 'user intervention' is necessary to clear the catastrophic error condition).
start	input	Starts a new operation specified by the command input. If the core is still running a previous operation 'busy' will be high and this input will be ignored.
command	input	Specifies the RNG operation to be carried out: Instantiate, Uninstantiate, Reseed, or Generate.
prediction_resistance_request	input	Causes the core to reseed before generating bits so that it is impossible to predict the next set of bits based on the previous outputs.
request_nonce	output	Indicates the core requires nonce information on the next clock cycle.
nonce(31:0)	input	Nonce value - longer nonces will be transferred in multiple words.
request_personalization_string	output	Indicates the core requires personalization string information on the next clock cycle.
personalization_string	input	Personalization string value - longer strings will be transferred in multiple words.
request_additional_input	output	Indicates the core requires additional_input information on the next clock cycle.
additional_input[31:0]	input	Additional input value - longer strings will be transferred in multiple words.
request_entropy	output	Indicates the core would like entropy information on the next clock cycle. Unlike the other _require signals there is a possibility that entropy will not be available.
entropy_available	input	Indicates that entropy is available to load on the current clock cycle.
entropy_input(31:0)	input	Entropy value - longer strings will be transferred in multiple words.
entropy_source_error	input	Indicates the entropy source has an error condition and currently cannot

		deliver entropy which it is safe to use. In the companion standard SP800-90B the entropy source is specified to have self test circuitry which checks the statistical properties of the generated random bitstrings and reports an error if the tests fail.
busy	output	Indicates the core is currently processing and is not available for a new operation.
returned_bits_valid	output	Indicates the core is currently outputting valid random bits on the requested_bits output.
returned_bits(31:0)	output	Random bits, longer random outputs will use multiple words.
status_valid	output	Indicates the catastrophic_error and error signals for the current operation are valid. Active on the last cycle of the operation so can also be used as a marker for the end of the operation.
catastropic_error	output	Indicates the core has encountered a catastrophic error. This signal goes high and stays high until the user clears the error using the reset input or by cycling power. While in the catastrophic_error condition the core is locked-up and will not process commands. It is a requirement of the standard that user intervention is needed to clear a catastrophic error.
error	output	Indicates an error on the current operation. This type of error does not stop the core processing further commands. Errors reported by the entropy source on entropy_source_error wil cause a corresponding error on this signal.

Table 2: Core I/O Signals.

Verification Methods

The testbench includes a self-checking configuration of the top level entity in the VHDL design which uses a behavioral model of the AES-RNG algorithm to check the results from the synthesisable implementation code. This is implemented using the VHDL facility to provide multiple architecture definitions for a particular entity: the top level entity in the design has a self_checking and a synthesis architecture defined. As shown in Figure 2, the self checking architecture has an identical interface to the synthesisable architecture and instances the synthesisable architecture within itself but also contains behavioral code to capture all input and output signals and check their values against expected values computed using a behavioral model. When errors are detected assertions are triggered and the simulation is stopped with an error message.

This self-checking configuration of the AES-RNG core can also be instantiated within the user's own simulations. This makes it easy to verify the core operates properly when connected to the user circuitry surrounding the core. In addition, the assertions within the self checking code will detect and report many situations where the user design is not driving the core correctly simplifying the task of integrating the core with the larger user design easier.

The AES-RNG testbench supplied with the core also makes use of the self checking configuration of the core for random testing. The testbench stimulates the self checking core with a random sequence of packets, writes of key information and key activations and the self checking core takes responsibility for detecting any errors.

The AES-RNG behavioral model used in the self checking core is also operational in regression and qualification testing so that its outputs along with the outputs of the synthesisable logic are checked by the NIST test vectors.

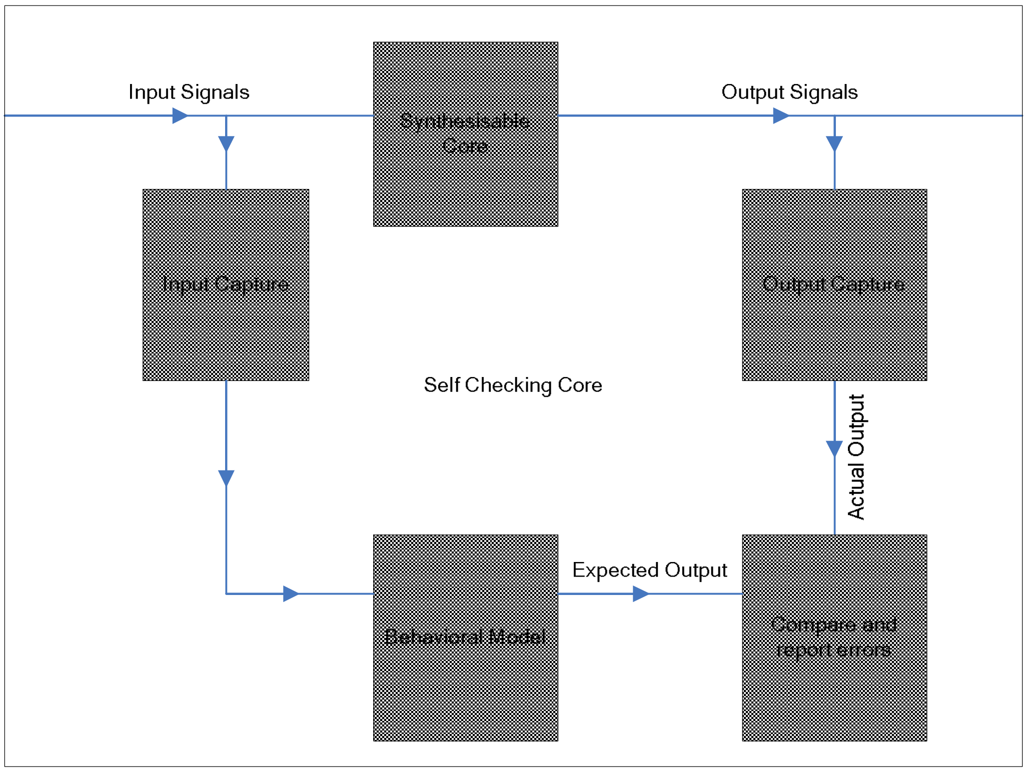

Figure 2, AES-RNG Self Checking Architecture

The AES-RNG testbench runs in one of three basic modes:

- In RANDOM mode the testbench generates random test vectors and applies them to the core. Results from the core are checked against a behavioral model. This is a more comprehensive test than the NIST test vectors as can involve large numbers of vectors and there is no fixed structure to the order in which operations are applied.
- In REGRESSION mode the testbench automatically compares the results of test operations against known answers provided in the test file. These files have the extension .gld for ‘golden’ because they contain known good results.
- In QUALIFICATION mode the testbench processes vector files with extension .req for ‘request’. These files do not contain correct answer data. In this mode the testbench generates output files with extension ‘res’ for ‘result’. The result files would be required if the AES-RNG core was submitted to a NIST approved laboratory for qualification.

As well as random testing using the self-checking version of the core in RANDOM mode the test bench has access to an example set of known-answer test vectors provided by NIST in the format specified in the NIST DRBGVS document in REGRESSION and QUALIFICATION modes.

Customization Service

Algotronix can offer a cost effective customization service for this core in order to tune the implementation for easy integration into a larger system. One option would be to allow a single AES-G3 core to be shared for use both by the AES-RNG core and for data encryption or wrapping of keys for transmission to a remote location using AES-Keywrap.

It is also possible to produce variants with significantly higher performance at the expense of increased area and to create optimized variants of the core targeted at particular FPGA products.

Recommended Design Experience

It is recommended that the user is familiar with the VHDL language and with the Xilinx design flow and simulation tools. The core can also be instantiated inside a wrapper to allow use with a Verilog design flow.

It is also recommended that the user has a background in data security or takes appropriate advice when considering how to implement AES-RNG in a larger system.

Ordering Information

This product is available directly from Algotronix under the terms of the SignOnce IP License. Please contact Algotronix for pricing and additional information about this product using the contact information on the front page of this datasheet. To learn more about the SignOnce IP License program, contact Algotronix or visit the web:

Email:	commonlicense@xilinx.com
URL:	www.xilinx.com/ipcenter/signonce

Export Control

Strong encryption technology such as AES is the subject of international export regulations. Algotronix is located in the United Kingdom and export of this core is regulated by the UK government.

The core is freely available within the European Union and in addition can be supplied immediately to the following countries: United States, Australia, New Zealand, Canada, Norway, Switzerland, Japan.

Export to other countries requires an export licence. The UK Department of Business, Enterprise and Regulatory Reform publishes information on their website (www.berr.gov.uk) which gives an indication of average export licence processing times for various countries and the percentage of licence requests which are granted. For many countries obtaining an export licence can be done relatively quickly and with only a small amount of paperwork.

It is the the responsibility of the customer to comply with all applicable requirements with respect to re-export of products containing the AES technology.

Related Information

Industry Information

The AES standard documents FIPS197, SP800-38A and AESAVS are available from the National Institute of Standards and Technology, Computer Security Resouce Center website (www.csrc.nist.gov).

Xilinx Programmable Logic

For information on Xilinx programmable logic or development system software, contact your local Xilinx sales office, or:

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124

Phone:	+1 408-559-7778
Fax:	+1 408-559-7114
URL:	www.xilinx.com

URL: www.algotronix.com

Version Control Information	
Subversion Revision Number	**71**
Date	2015/04/01 16:06:15
Document	Aes Rng Data Sheet, Xilinx Edition
Status (blank field indicates OK/no warnings)	
	(Table auto-updates, do not edit field values by hand)