GOST (Kuznyechik) Core, Xilinx Edition

November 7, 2016 Product Specification

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 Vivado Simulator	
Support	
Support provided by Algotronix	

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

Features

- Supports all cipher modes defined in NIST SP800-38A: ECB, CBC, CFB1, CFB8, CFB128, OFB and CTR.
- Supports 256 bit keys
- Targets all modern FPGA families from Xilinx.
- Compile as Encryptor, Decryptor or Encryptor/Decryptor
- 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.

Applications

- Highly secure systems using multiple cipher algorithms to protect against one algorithm becoming compromised
- Communications systems which must interoperate with Russian equipment

General Description

This IP core implements the 'Kuznyechik' cipher specified in the Russian GOST 3412-2015 standard (also available in English as RFC 7801) which replaces the previous 'Magma' algorithm in the GOST 28147-89 standard. Both algorithms are commonly referred to as 'GOST' in English language publications but they have different security properties and resource requirements (GOST is the name of the Russian standardization body rather than the name of an encryption algorithm).

GOST is structurally similar to the AES algorithm and this IP core product has been developed using many of the techniques and some of the code libraries used in the Algotronix AES cores. There is currently no reason to believe the GOST cipher is superior to AES, on the contrary is the de-facto standard algorithm and has had a more transparent standardization process, has been subjected to more intensive scrutiny and has had more effort applied to optimizing its implementation. AES remains the default choice for a symmetric encryption algorithm.

Use of a GOST core rather than an AES core is mainly of interest in three scenarios:

a. To create variants of equipment which normally uses Algotronix AES cores for sale in markets where the GOST cipher is mandated or to create equipment which must occasionally inter-operate with GOST encryption.
b. As a fallback algorithm for sensitive, high reliability applications which use AES but must be able to continue to operate in the unlikely event of a successful attack on AES coming to light. In this circumstance, where AES is believed to be compromised, a fallback algorithm which has been through a completely separate development and standardisation process might be an advantage. With an FPGA implementation the switch to the fallback algorithm could be done by reconfiguring the device. GOST is sufficiently similar to AES to make it practical to swap between them without requiring major changes to the surrounding system.
c. For extremely secure applications where it is desired to use two unrelated encryption algorithms so that security is maintained even if one algorithm is discovered to have a back-door or is later compromised by an attack.

The GOST core currenlty offers similar functionality to the AES-G3 product with a 128 bit data path, it can be extended to provide more of the options of AES-G3 on request if there is customer interest. It is also possible to extend the GOST product range to mirror additional AES products (for example AES-GCM or AES-XTS) on request at price points roughly equivalent to the corresponding AES product.

To support highly sensitive applications, the Algotronix GOST 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 synthesizable core which compares its outputs with a behavioral model of GOST, the behavioral model of GOST is verified against example vectors in the GOST standard.

Table 1 shows example implementation statistics for the core in the simple configuration most commonly used for comparison purposes. There are many implementation options available, some of which have a significant effect on area and performance. The GOST 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 GOST, 'Push Button' flow with Fmax specified by clock constraint. ECB mode, Encrypt Only, 256 Bit Key

Family	Example Device	Fmax (MHz)	Slices[1]	IOB[2]	GCLK[2]	BRAM[1]	MULT/ DSP48	DCM	Throughput (MBit/sec)	Design Tools
Artix-7	xc7a200tffg1156-3	142	485	105	1	8	0	0	1817	Vivado 2016.3

Notes:

1) Actual slice count dependent on percentage of unrelated logic – see Mapping Report File for details. BRAM count is 36K tiles.

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

Comparison with AES

Structurally, GOST is quite similar to AES however there are a few significant differences.

AES offers more flexibility than GOST. It supports 128, 192 and 256 bit keys and provides for efficient implementation with a 32 bit as well as a 128 bit data path.

The 'Transformation L' component of GOST works over all 128 bits of the block where the corresponding 'Mixcol' unit in AES is implemented as four 32 bit subunits. Using the best known implementation the GOST Transformation L is significantly larger and has significantly longer propagation delay than the AES Mixcol. The propagation delay through Transformation L is currently the limiting factor on achievable clock frequency, which is significantly less than with AES.

Conversely, AES uses more rounds of computation than GOST for the same key size. This will correspond to more area in a pipelined implementation or less performance in a non-pipelined one (such as this initial GOST core). One could see the additional rounds of computation in AES as roughly balancing out with the additional complexity per round in GOST.

The AES keyschedule algorithm uses a matching number of basic steps to the encryption algorithm. This makes it easy to operate the AES keyschedule 'online' in encrypt mode so as it generates round keys as required. In GOST the number of basic steps in the keyschedule algorithm is significantly higher than the number of steps in the encryption algorithm. To operate 'online' multiple copies of the SBoxes and Transformation L units would be needed in the keyschedule to generate round keys fast enough for a main datapath with a single set of Sboxes and Transformation L. The present GOST implementation does not offer online key schedule generation because of this area cost but it could be added on request.

Functional Description

The main functional blocks are shown in Figure 1 .

The GOST standard document describes the basic ECB mode of the cipher. ECB is normally used as a building block for more complex modes so Algotronix has augmented this with the modes circuitry it provides with its AES cores to include CBC, CTR, OFB, CFB1, CFB8 and CFB128 modes as specified in NIST SP800-38A. It is not recommended to use ECB mode to encrypt streams of data because under ECB mode a given input text will always encrypt (under the same key) to the same output text. Thus, if the same input text occurs many times in a stream of data then the corresponding output will occur in corresponding locations in the output giving an opening for cryptanalysis. The more complex modes of the cipher address this issue. If the modes logic is not required the gost_ecb_cipher module can be used directly within the user design.

The Algotronix GOST core is aimed at providing ease of use, sufficient performance with good area efficiency for mainstream applications. A further design goal is that it should be straightforward for a customer to relate the implementation source code back to the algorithm description in the standard.

ECB Data Path

This block implements the ECB mode of the AES Algorithm. All other modes of AES are built on top on this basic encryption operation.

Key Schedule

This block calculates the round keys for each stage of the GOST algorithm based on the key supplied by the user. The keyschedule is calculated before data starts flowing through the ECB data path and so can share resources with the ECB data path to save area.

Mode Logic

This block contains the feedback paths and additional logic required to implement the more complex modes – CBC, OFB, CFB1, CFB8, CFB128 and CTR. Compilation parameters are supplied so only the logic for those modes which are required by the user will be instantiated.

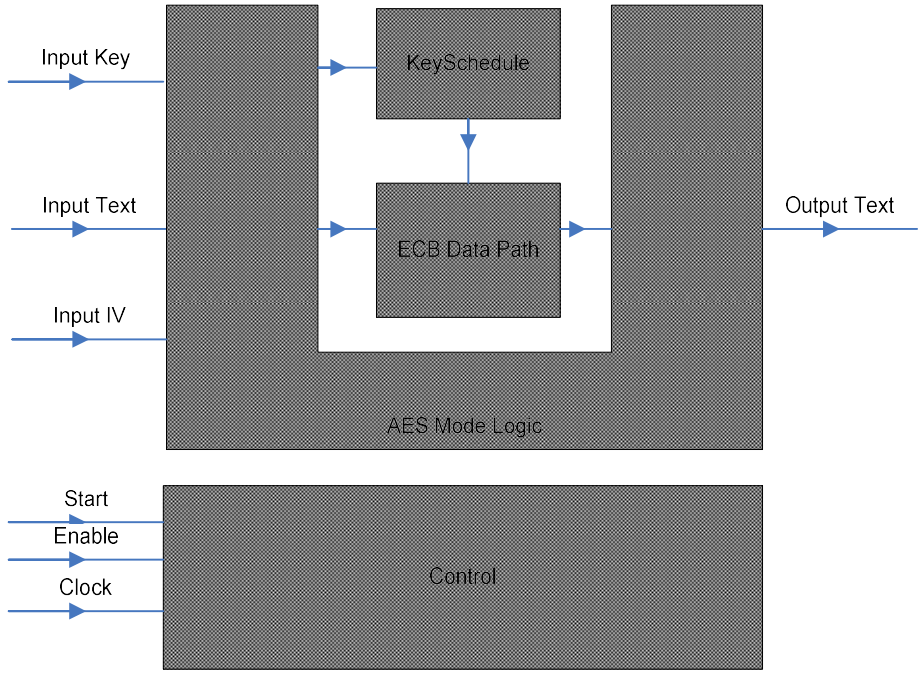

Figure 1, GOST Core Block Diagram

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 gost_cipher.vhd. The compilation parameters are then specified in the file gost_parameters_package and applied in the file gost_cipher_wrapper – the result is a fully parameterized GOST core with no generic parameters at the top level of the design hierarchy. The advantage of having a non-parameterized cell at the top of the design hierarchy is that the top level cell of the 'flat' post-synthesis design has exactly the same interface as the top level cell of the pre-synthesis design. This allows the same testbench to be used for both pre-synthesis and post synthesis simulation. Algotronix can also supply a wrapper to allow the core to be used within a Verilog design.

The most expensive sub-units in area terms within the GOST design are the 'Transformation L' unit and the 'Transformation S' substitution or S-boxes. These units are needed both to calculate the keyschedule and to run the encryption or decryption and are different for ECB encryption and decryption. Many of the

compilation parameters are concerned with making efficient use of the Transformation L and Transformation S circuits.

The following compilation options are specified by editing constant definitions in the gost_parameters_package file. This is the only file in the GOST core release which will normally be edited by the user.

- **cipher_function** - specifies whether an Encryptor, Decryptor or Encryptor/Decryptor is required.
- **implement_sboxes_in_ram** – specifies that FPGA RAM blocks rather than LUT memory should be used to implement SBoxes and Inverse SBoxes. This is often the most efficient option if RAM blocks are available after mapping the remainder of the user design.
- **omit_ecb_mode, omit_cbc_mode, omit_ofb_mode, omit_ctr_mode, omit_cfb1_mode, omit_cfb8_mode, omit_cfb128_mode** - Used to request that logic to support cipher modes that will not be required is omitted from the design. The CTR and CFB modes require quite large amounts of additional logic.
- **keyschedule_shares_lsx_unit** – specifies that the same 'Transformation S' and 'Transformation L' circuits are used for the encryption datapath and the keyschedule unit. This saves considerable area but inserts latency after the key is changed while a new keyschedule is calculated. In the initial release of the core this parameter should always be set to true.
- **force_output_low_until_valid** – When true the core will hold the output low at all times when valid output data is not present. When this signal is false the circuitry to hold the output zero will be omitted, saving some area. In this case the core output 'output_text' will show the values at intermediate rounds of the cipher as well as the final round. This data is not fully encrypted and, if available to an attacker, could compromise security of both the key and data. If the user's design which contains the core can guarantee that an attacker will not be able to monitor the core output directly then this parameter can be set to false which will save some area.
- **target_device** – In the Xilinx Edition version of the core only Xilinx devices are supported, target device should be set to XILINX_VIRTEX_5 which will support all recent device families with 6 input LUTs.

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
mode	input	Mode signal – specifies which mode of AES is to be implemented. See also the omit_* compilation options in the section below. If compilation options have specified that logic for a particular mode should be omitted then incorrect behaviour will result if that mode is selected.
do_encrypt	input	Specifies whether the core should operate in ENCRYPT or DECRYPT mode. This input is only significant if the compilation option cipher_function is set to ENCRYPT_DECRYPT i.e. hardware for both encryption and decryption has been included.
start	input	Starts a new encryption operation or block of operations in the chained modes. The control signals mode and do_encrypt are sampled and the parameters fixed for the next operation. The key is assumed to have changed and the keyschedule is recalculated.
load_text	output	Load flag – high when input_text is being loaded
load_key	output	Load flag – high when the key is being loaded
output_valid	output	Valid flag – high when output_text is valid.
advanced_output_valid	output	High on the four clock cycles immediately preceding output_valid. This signal gives advanced warning that the core is about to input and output data and can by external control circuitry to stop the core using the enable signal until the system is ready to provide new input data and accept output data.
input_text[127:0]	input	Data input: current 32-bit word of the 128-bit plain text
output_text[127:0]	output	Data output: current 32-bit word of the 128-bit cipher text
initial_value [127:0]	input	current 32-bit word of the 128-bit initial value for the chained modes of operation (ECB mode does not use the initial value).
input_key[127:0]	input	current 128-bit word of the key – it takes two clock cycles to load a complete 256 bit key

Table 2: Core I/O Signals.

Description of Operation

The input_text, output_text and initial_value buses transfer 128 bit GOST block values over 128 bit wide buses in a single clock cycle. The input_key bus transfers 256 bits of key information over a 128 bit bus in two clock cycles. Words are transferred in order starting with the most significant word.

Synchronising between the user design and the core is straightforward. The user controls when the core processing starts using the 'start' signal. When the core samples 'start' as being high on a rising edge of the clock it prepares for a new chain of encryptions and on the following clock cycle starts to load key data and compute the keyschedule.

Once the keyschedule is ready the initial value and first block of input text is loaded the core begins processing. The user design can use the enable signal to implement flow control if it is not ready to accept the output data – bringing enable low will stop the core processing. The advanced_output_valid signal gives a warning that the core is ready to output text, assuming that enable is left high, one clock cycle later the core will start outputting the processed plaintext or ciphertext (depending on whether this is an encryption or decryption operation). Simultaneously, if this is a chained operation it will load the next block of plaintext or ciphertext for processing.

The best way to get an understanding of the timing of the interface signals to the core is to simulate the core using the testbench provided and examine the waveforms on the signals in the table above during operation in the mode of the cipher you wish to use. Algotronix also provides a 'Getting Started' application note which includes timing diagrams for the core.

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 GOST 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. The GOST behavioral model is validated using known-answer examples provided in the GOST standard and by loop-back testing.

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 GOST 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 GOST 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 test data and the self checking core takes responsibility for detecting any errors.

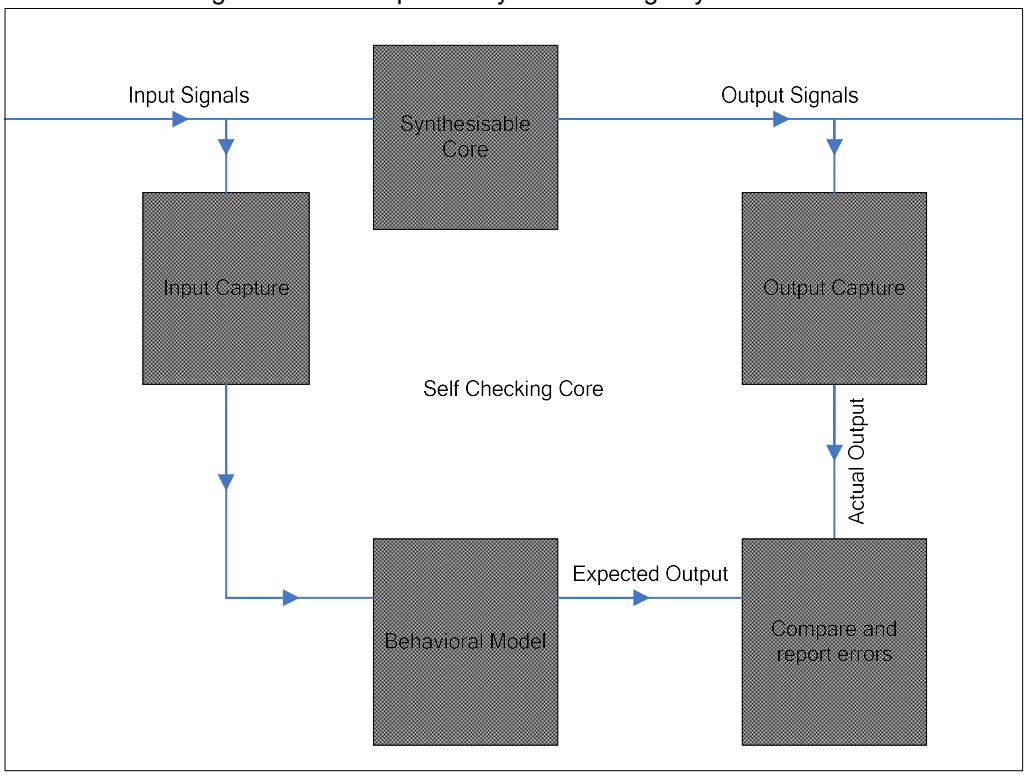

Figure 2, GOST Self Checking Architecture

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. 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 GOST 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 GOST 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 GOST technology.

Related Information

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

Copyright © 2002-2016 Algotronix Ltd., All Rights Reserved.

Algotronix® is a registered trademark of Algotronix Ltd. in the United States and United Kingdom and a trademark of Algotronix Ltd. in other countries.
The supply of the product described in this document is the subject of a separate license agreement with Algotronix Ltd. which defines the legal terms and conditions under which the product is supplied. This product description does not constitute an offer for sale, a warranty of any aspects of the product described or a license under the intellectual property rights of Algotronix or others. Algotronix products are continuously being improved and are subject to change without notice. Algotronix products are supplied 'as is' without further warranties, including warranties as to merchantability or suitability for a given purpose. Algotronix' products are not intended for use in safety critical applications.

URL: www.algotronix.com