SHA Core, Xilinx Edition

May 3, 2017 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

- Implements the SHA-256 algorithm specified in FIPS180-4 Secure Hash Standard (August 2015).
- Targets all modern FPGA families from Xilinx.
- 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

- Authenticated Communications
- Block chain

General Description

This IP core implements the SHA-256 algorithm specified in FIPS180-4 which is widely used for authenticating messages in communications and can also be used to authenticate stored data and in blockchain algorithms. The implementation is based on the most recent version of the standard from August 2015.

The core has a 32 bit data interface and uses a non-pipelined architecture to provide gigabit level performance with good area efficiency on modern FPGA families. Algotronix will implement variants of the core for different area/performance points or with longer message digests on request.

SHA-256 provides a 256 bit message digest and is considered to provider stronger authentication with less chance of collisions than the GF-HASH algorithm used in AES-GCM (which provides a 128 bit hash value), the CBC-MAC algorithm used in AES-CCM (which provides a 128 bit Message Authentication Code) or the EIA3 authentication algorithm based on the ZUC stream cipher which provides a 32 bit message digest. No secret information (such as a cryptographic key) is needed to compute the SHA-256 message digest of a known message. To use SHA-256 as a message authentication code to detect tampering with a message the message digest should be encrypted.

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 SHA under random testing and can also make use of SHA Verification System test vectors provided by NIST in both regression testing against known answer supplied by NIST and qualification testing to generate a response file for validation by a NIST test lab.

Table 1 shows example implementation statistics for the core in the simple configuration most commonly used for comparison purposes. The SHA 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 most recent version of the core in particular core configuration required in your application on request.

Table 1: Example Implementation Statistics for SHA-256, 'Push Button' flow with Fmax specified by clock constraint.

Family	Example Device	Fmax (MHz)	LUT	LUTRAM	GCLK[2]	BRAM[1]	FF	DCM	Throughput (MBit/sec)[3]	Design Tools
Zynq Ultrascale	XCZU2eg-sfva625-3-e	307	1302	32	1	0	1027	0	2,456	Vivado 2016.4

Notes:

1) BRAM count is 36K tiles.

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

3) Typical throughput for long messages.

Functional Description

The main functional blocks are shown in Figure 1.

SHA-256 is a cryptographic hash function, the SHA-256 algorithm works on 512 bit blocks of data and computes a 256 bit message digest. The core has two main modules, the first adds length information to the message and pads it to a 512 bit boundary. The second calculates the message digest based on the padded data.

There is no message_width signal to the SHA module because at that point the data is padded to a 512 bit boundary so there is no partial word at the end of the message. The final signal to the SHA module will usually occur several clock cycles after the final signal to the insert padding module because the data is being expanded.

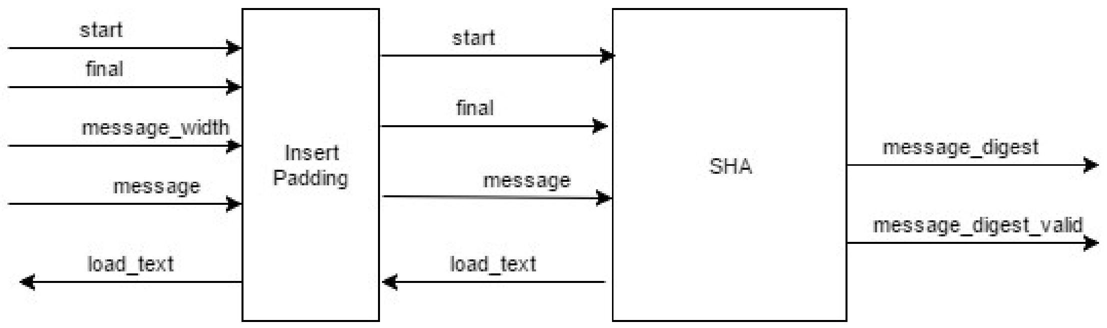

Figure 1, SHA-256 Core Block Diagram

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

- **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 for the SHA core is provided in Table 2.

Signal	Signal Direction	Description
clock	input	Clock – active on rising edge
reset	input	Reset – active high. A global constant in zuc_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. In the description of the functionality of the other signals enable is assumed to be high.
start	input	Instructs the core to start a new message. The first word of text for the message will be loaded when the core brings load_text high.
final	input	End marker,set to high on the last word of the input message.
load_text	output	Indicates the core will load message text on the rising edge of the clock.
message[31:0]	input	Data input: 32-bit word. Message to be hashed. The core loads text in 512 bit blocks using 16 cycles of this bus. The last block can be brought to an end before all 16 cycles using the final signal. After each block is loaded there is a delay before load_text goes high again while the data is processed.
message_width(5 : 0)	input	Width of final word of message in bits, required because message is not required to be an integral number of 32 bit words.
message_digest_valid	output	Valid flag – high when output_text is valid.
message_digest(31:0)	output	Message digest, core requires 8 clock cycles to output entire 256 bit message digest over this bus.

Table 2: SHA-256 Core I/O Signals.

Description of Operation

The message is input over the message bus 32 bits at a time when load_text is high using as many clock cycles as required. Processing is done in units of 512 bits and after a block is loaded the load_text signal will be brought low while the core is processing. Words are transferred in order starting with the most significant word. The message can end on any bit boundary so the final 32 bit input word may be incomplete and a message_width input is provided to specify how many bits of the final word are valid. Bits are regarded as being in order starting with the most significant bit so the 'empty' section of the final word is in the least signficant bits.

The SHA-256 standard specifies a padding algorithm to pad out the final block to 512 bits before processing. The padding algorithm also inserts message length information into the data to be processed: a message which was an exact multiple of 512 bits before padding will not be a multiple of 512 bits after including the length information and so would still need padded to a 512 bit boundary. The padding is done automatically by the core but users need to be aware of the consequences for flow control and maximum throughput of the message being extended to a 512 bit boundary. The core is effectively processing a longer message than that supplied by the user so the delay between the final signal marking the end of the input message and the message_digest_valid signal marking that the message_digest has been computed will vary depending on how much padding was required. For short messages the padding length may be significant relative to message data length.

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 starts processing a new message and will bring load_text high as soon as it is ready for the first word of data. If start is brought high while the core is midway through processing a message the previous message will be abandoned. Normally the user design containing the core will wait until the message_digest for the previous message has been output before bringing start high for the next message.

The user design can use the enable signal to implement flow control if it is not ready to provide input data or accept the output data – bringing enable low will stop the core 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.

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 SHA 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 SHA behavioral model is validated using known-answer examples provided in the ZUC 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 SHA 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 SHA 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.

As well as random testing the testbench implements known-answer-tests using Secure Hash Algorithm Validation System (SHAVS) test vectors supplied by NIST and can generate response files for SHA algorithm validation by a NIST approved test laboratory.

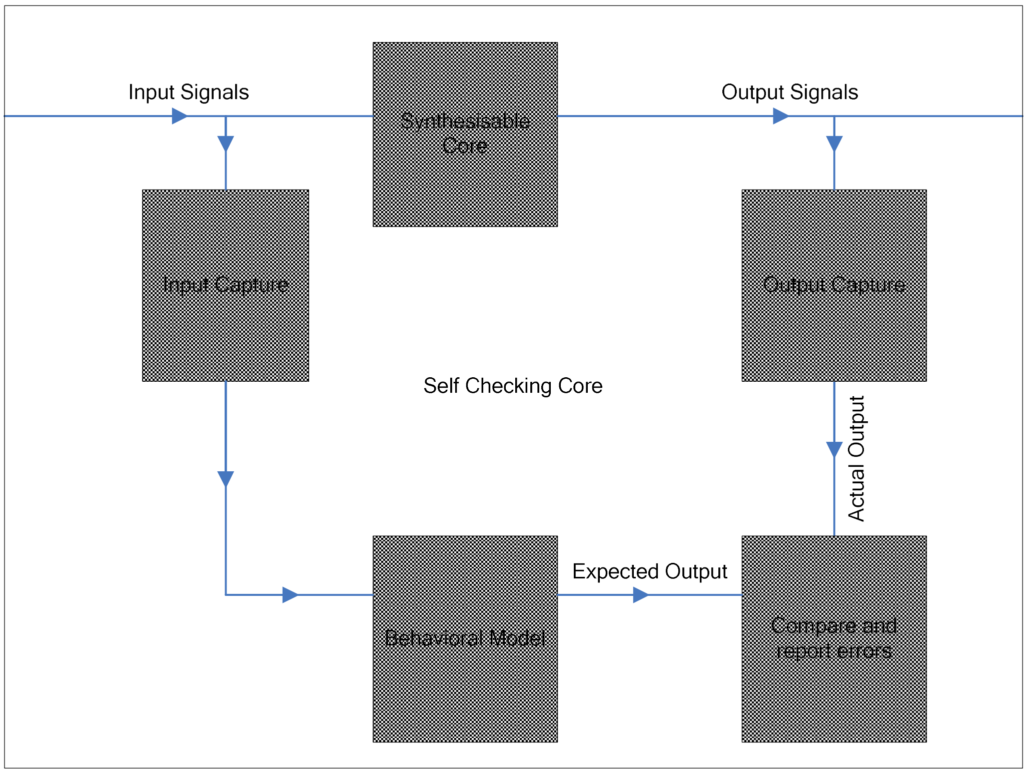

Figure 2, SHA Self Checking Architecture

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 ZUC in a larger system.

Related Standards

1. FIPS PUB 180-4 “Secure Hash Standard” August 2015. http://dx.doi.org/10.6028/NIST.FIPS.180-4
2. “The Secure Hash Algorithm Validation System (SHAVS)”, NIST report, updated May 2014.

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 SHA 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 ZUC 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