SHA-3

Module Specification

Disclaimer

Systemyde International Corporation reserves the right to make changes at any time, without notice, to improve the design or performance and provide the best product possible. Systemyde International Corporation makes no warrant for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make any commitment to update the information contained herein.

Systemyde International Corporation products are not authorized for use in life support devices or systems. Nothing contained herein shall be construed as a recommendation to use any product in violation of existing patents, copyrights or other rights of third parties. No license is granted by implication or otherwise under any patent, patent rights or other rights, of Systemyde International Corporation. All trademarks are trademarks of their respective companies.

Every effort has been made to ensure the accuracy of the information contain herein. If you find errors or inconsistencies please bring them to our attention. In all cases, however, the Verilog HDL source code for the SHA-3 module defines “proper operation”.

This design may be subject to export restrictions. It is the customer's responsibility to check for export restrictions relative to any equipment containing this technology.

Overview

The SHA-3 module implements the SHA3-512, SHA3-384, SHA3-256 and SHA3-224 hash algorithms and the SHAKE128 and SHAKE256 extendable-output functions specified in FIPS PUB 202, the Permutation-Based Hash Standard. This module contains all of the digital logic necessary to generate the 512-bit (or 384-bit, or 256-bit, or 224-bit) message digest for a message of arbitrary bit length.

General Features:

- Technology-independent Verilog HDL implementation.
- Fully synchronous design.
- Simple synchronous 32-bit host interface, compatible with the AMBA APB.
- Generic FIFO input buffer, which can be replaced by an FPGA or ASIC macro.

SHA3 Features:

- One clock per step algorithm implementation.
- Automatic rate selection.
- Automatic pad insertion.
- Any bit length message, including zero-length, is allowed.
- Direct Message Digest output, with strobe.

This page intentionally left blank.

Signal Descriptions

The SHA-3 module connects using a simple one-clock synchronous 32-bit bus, which allows for a simple interface to a standard bus such as AMBA AHB or AMBA APB. The 512-bit (or smaller) hash value is output directly, along with a strobe signal to indicate that the hash computation is complete.

Interface signals:

`clk` (input). The Master Clock is used as both the bus clock and the SHA-3 state machine clock. Because of the input buffer, it is possible to modify the design to use separate clocks for the bus and the SHA-3 state machine if necessary.

`resetb` (input, active-Low). The asynchronous Master Reset signal initializes the module.

`wr_bus[31:0]` (inputs). The Write Data Bus carries write data to the module. This bus is sampled by the clock when the **`wr_pls`** signal is active. Data is assumed to be big-endian within 32-bit words, and right-justified within bytes, consistent with DRAFT FIPS PUB 202.

`wr_bits[2:0]` (inputs). The Write Bit Length Control is used to signal the number of valid bits in the last byte of the write data, according to the table below. Bits are assumed to be right-justified in a byte. This bus is sampled by the clock when the **`wr_pls`** signal is active. Only the last word of a message may be less than the full 32 bits.

`wr_type`	Meaning
000	All 8 bits valid
001	Only bit 0 of last byte is valid
010	Only bits 1:0 of last byte are valid
011	Only bits 2:0 of last byte are valid
100	Only bits 3:0 of last byte are valid
101	Only bits 4:0 of last byte are valid
110	Only bits 5:0 of last byte are valid
111	Only bits 6:0 of last byte are valid

wr_type[2:0] (inputs). The Write Type Control is used to signal the width of the write data, according to the table below. This bus is sampled by the clock when the **wr_pls** signal is active. Only the last word of a message may be less than the full 32 bits.

wr_type	Meaning
000	Zero-length message
001	Last word - wr_bus[31:24] only
010	Last word - wr_bus[31:16] only
011	Last word - wr_bus[31:8] only
100	Last word
101-110	Unused (but interpreted as Normal word)
111	Normal word

wr_pls (input, active-High). The Write Enable signal enables the sampling of the **wr_bus** data bus and the **wr_bits** and **wr_type** control signals on the rising edge of the clock. Continuous writes are allowed.

sel_hash[2:0] (inputs). The Select Hash bus is used to select which algorithm to use, according to the table below. The state of this signal is latched by the clock when the **start_pls** signal is active.

sel_hash	Algorithm	Rate (Block size)
000	SHA3-224	1152 bits (36 words)
001	SHA3-256	1088 bits (34 words)
010	SHA3-384	832 bits (26 words)
011	SHA3-512	576 bits (18 words)
100	SHAKE128	1344 bits (42 words)
101	SHAKE256	1088 bits (34 words)

start_pls (input, active-High). The Start Enable signal is used to signal the start of a new hash calculation. To start a new hash calculation, this signal must be asserted for one clock sometime after the previous hash computation is complete, but prior to the last word a block or the last word of the new message being written to the module.

buffer_full (output, active-High). The Buffer Full signal is asserted by the input FIFO when it is full. No further writes to the module should occur until this signal is deasserted. The timing of this signal will be dependent on how the FIFO is implemented.

hash_done (output, active-High). The Hash Done signal is asserted for one clock cycle when the hash for the last block is complete and available on the **hash_reg** bus.

hash_reg[511:0] (outputs). The Hash Value bus is updated at the end of each message block, but the final hash value is only valid when accompanied by the **hash_done** signal. The table below shows the position of the hash value for the different hash lengths.

sel_hash	Hash location
000	**hash_reg[511:288]**
001	**hash_reg[511:256]**
010	**hash_reg[511:128]**
011	**hash_reg[511:0]**
100	**hash_reg[511:256]**
101	**hash_reg[511:0]**

Timing

The SHA-3 module is a synchronous design that uses the **clk** signal for all timing. Inputs are sampled by the rising edge of the clock and outputs normally change only in response to the rising edge of the clock. The only exception is the case of the assertion of the asynchronous **resetb** signal.

The SHA-3 module is supplied with a pair of generic FIFO input buffers, to match the 32-bit width of the **wr_bus** data bus to the native 64-bit datapath of the hash state machine. The expectation is that these generic FIFOs will be replaced with technology-specific macros by the user. The characteristics (width, depth, and timing) of the generic FIFOs are consistent with FIFO macros available with all major FPGA families.

One 32-bit word can be written to the FIFO input buffers on every clock edge. When the hash state machine is ready for data, it can read one pair of 32-bit words from the input buffer every other clock cycle.

All of the SHA-3 algorithms use 25 rounds, and each of these rounds requires one clock cycle in this design. However, because the rate (block size) is different for each algorithm, the overall throughput depends on the algorithm. The table below shows the rate (block size) for each algorithm.

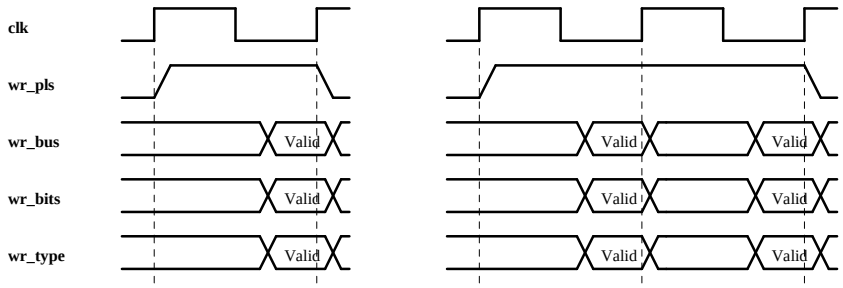
Algorithm	Rate (Block size, in 32-bit words)
SHA3-224	1152 bits (36 words)
SHA3-256	1088 bits (34 words)
SHA3-384	832 bits (26 words)
SHA3-512	576 bits (18 words)
SHAKE128	1344 bits (42 words)
SHAKE256	1088 bits (34 words)

Some of the timing is dependent on the rate, as the number of 32-bit words. This delay is denoted as "n" in the diagrams.

Some of the timing is dependent on the delay for the de-assertion of the "empty" signal out of the FIFO. This delay is 3 clock cycles for the generic FIFO supplied, but is denoted as "e" in the diagrams.

Writing data:

Write transactions are shown in the figure below. Back-to-back transactions are allowed, up to the depth of the input buffer, as shown on the right.

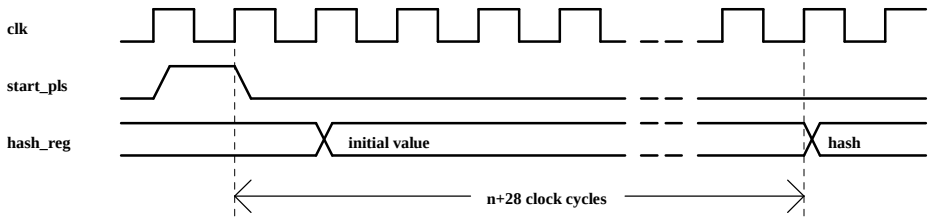


Starting a Hash computation:

The **`start_pls`** signal must be asserted to initialize hash state machine at the beginning of a message. The hash state machine will not automatically begin extracting words from the input buffer until the **`start_pls`** signal has been asserted.

Because the **`start_pls`** signal latches the state of the **`sel_hash`** signal, it should never be asserted while the previous hash is still being computed, especially when switching algorithms. In practice, the **`start_pls`** signal will usually be asserted after the previous hash computation is complete, but prior to the first word of a new message being written to the module.

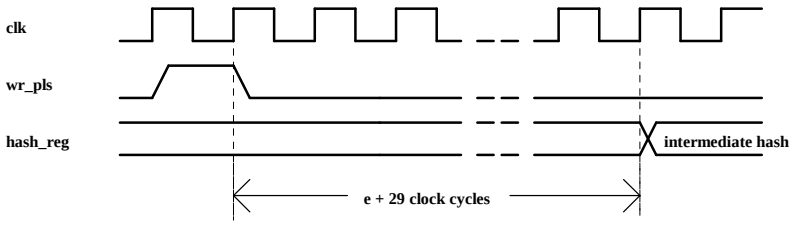
The figure below shows the timing in the case where the buffer already contains a full block of words when the **`start_pls`** signal is asserted.



Intermediate Hash computation:

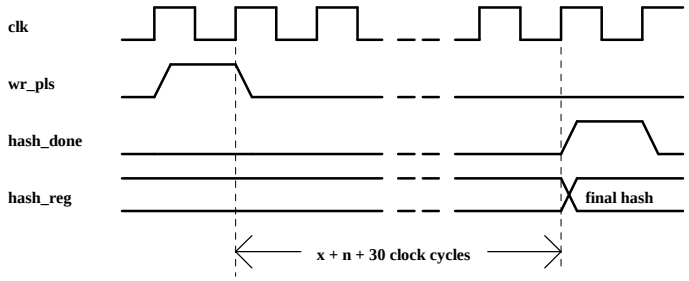
For intermediate blocks of a message, the hash state machine automatically starts when there are at least two words in the input buffer, but the algorithm rounds cannot start until the block is complete.

The figure below shows the timing for an intermediate block of a message, where the write is for the last word in a block.



Finishing a Hash computation:

The figure below shows the timing for the last block of a message when the padding does not require an additional block. Padding requires at least four bits for a hash computation and at least six bits for SHAKE128 and SHAKE256, so if a message ends at least this number of bits prior to a block boundary, this timing will apply. If the last write does not correspond to a block boundary the hash state machine must insert 64-bit pad words. The number of 64-bit pad words in this last block is denoted as "x" in the figure.



The figure below shows the timing for the last block of a message when the padding does require an additional block.