SHA-256

Module Specification

Disclaimer

Systemyde International Corporation reserves the right to make changes at any time, without notice, to improve 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-256 design 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-256 module implements both the SHA-256 and SHA-224 hash algorithms specified in FIPS PUB 180-4, the Secure Hash Standard. This module contains all of the digital logic necessary to generate the 256-bit (or 224-bit) message digest for a message up to 2^{40} bytes in 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.

SHA Features:

- One clock per step algorithm implementation.

- Automatic pad insertion.

- Handles byte-aligned messages from zero up to 2^{40}-1 bytes in length. Longer messages can be accommodated by increasing the width of the message-length counter.

- Direct Message Digest ouput, with strobe.

- Verified using SHAVS ShortMsg and LongMsg response files.

This page intentionally left blank.

Signal Descriptions

The SHA-256 module connects using a simple one-clock synchronous 32-bit bus, which allows for simple interface to a standard bus such as AMBA AHB or AMBA APB. The 224-bit or 256-bit 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 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 state machine if necessary.

`resetb` (input, active-Low). The asynchronous Master Reset signal completely 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, consistent with FIPS PUB 180-4.

`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 **`wr_type`** control signals on the rising edge of the clock. Continuous writes are allowed.

select_224 (input, High selects SHA-224). The Select SHA-224 signal is used to select which algorithm to use. The state of this signal is latched by the clock when the **start_pls** signal is active.

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 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 willbe 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[255:0] (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. In the case of SHA-224, the hash value is on **hash_reg[255:32]**.

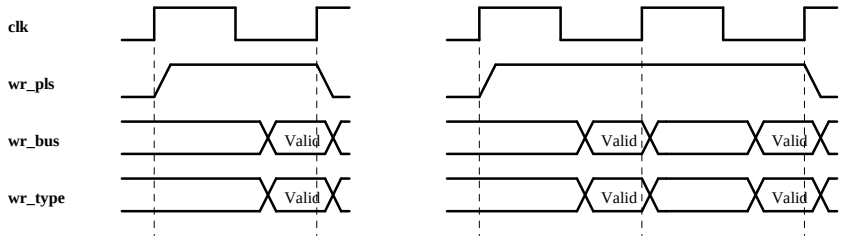
Timing

The SHA-256 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-256 module is supplied with a generic FIFO input buffer. The expectation is that this generic FIFO will be replaced with a technology-specific macro by the user. The characteristics (width, depth, and timing) of this generic FIFO are consistent with FIFO macros available with all major FPGA families.

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.

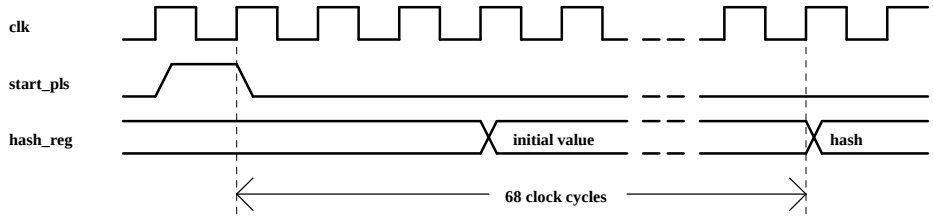


Starting a Hash computation:

The **start_pls** signal must be asserted to initialize hash state machine at the beginning of a message. Since the hash state machine will automatically begin when a word tagged as the last in a message is written to the input buffer, the **start_pls** signal must be asserted prior to this condition.

Because the **start_pls** signal latches the state of the **select_224** 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.

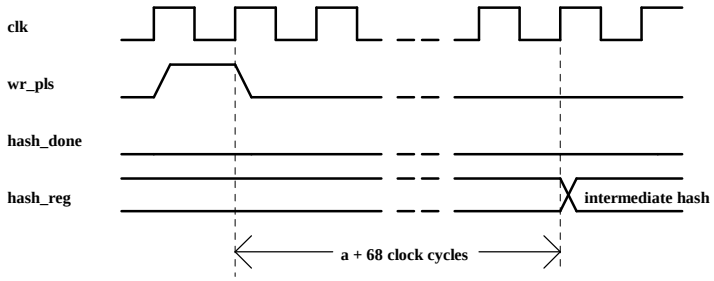
The figure below shows the timing in the case where the buffer already contains sixteen words when the **start_pls** signal is asserted.



Intermediate Hash computation:

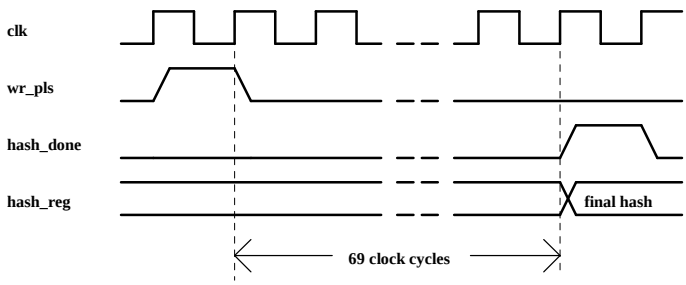
For intermediate blocks of a message, the hash state machine automatically starts when there are at least sixteen words in the input buffer, because this is the block size for SHA-224 and SHA-256. This hold-off relies on the **almost_empty** status signal from the FIFO used for the input buffer. When replacing the generic FIFO with a technology-specific FIFO macro, the **almost_empty** signal must be programmed to be asserted when fifteen or less words are in the FIFO. Common FPGA FIFO macros provide this capability.

The figure below shows the timing for an intermediate block of a message. In this figure the "a" is the number of clocks of delay to the deassertion of the **almost_empty** signal, which is specific to the FIFO implementation.



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 nine bytes, so if a message ends at least nine bytes prior to a 512-byte block boundary, this timing will apply. This timing assumes that the FIFO is not empty when the **wr_pls** signal occurs. If the FIFO is empty when this write occurs there will be an additional delay until the FIFO empty signal is deasserted. The number of clocks of delay to the deassertion of the **empty** signal is specific to the FIFO implementation.



The figure below shows the timing for the last block of a message when the padding does require an additional block. If the FIFO is empty when this write occurs there will be an additional delay until the FIFO empty signal is deasserted. The number of clocks of delay to the deassertion of the **`empty`** signal is specific to the FIFO implementation.