AXI SHA256 Accelerator Reference Manual

Revision 1.0

Table of Contents

1.0 Introduction

This document describes the features and operation of the Secure Hashing Algorithm 256-bit (SHA256) accelerator utilizing an AXI4-Lite interface.

1.1 Features

- AXI4-Lite interface
- Intermediate hash completion in 71 cycles
- Interrupt support
- Big and small endian configuration parameters
- Xilinx drivers

1.2 Functional Description

The AXI SHA256 accelerator has one control/status register, a register indicating the current progression of the hash, a message block buffer, and the output hash registers. The accelerator can hash one message block (512-bits) at a time – the maximum size of one block per the SHA256 specification. All that is required to hash a block is to load the entire message size into the message size registers and copy over bytes from a subsection of the message located elsewhere in memory to the accelerator registers. The intermediate hash computation is started by setting the UPDATE bit with the completion indicated by the BLOCK_DONE and/or HASH_DONE bits in the control/status register. The process of loading 512-bits (64-bytes) from memory to the accelerator's registers/block-buffer and commencing the intermediate hash is repeated until all the message is hashed.

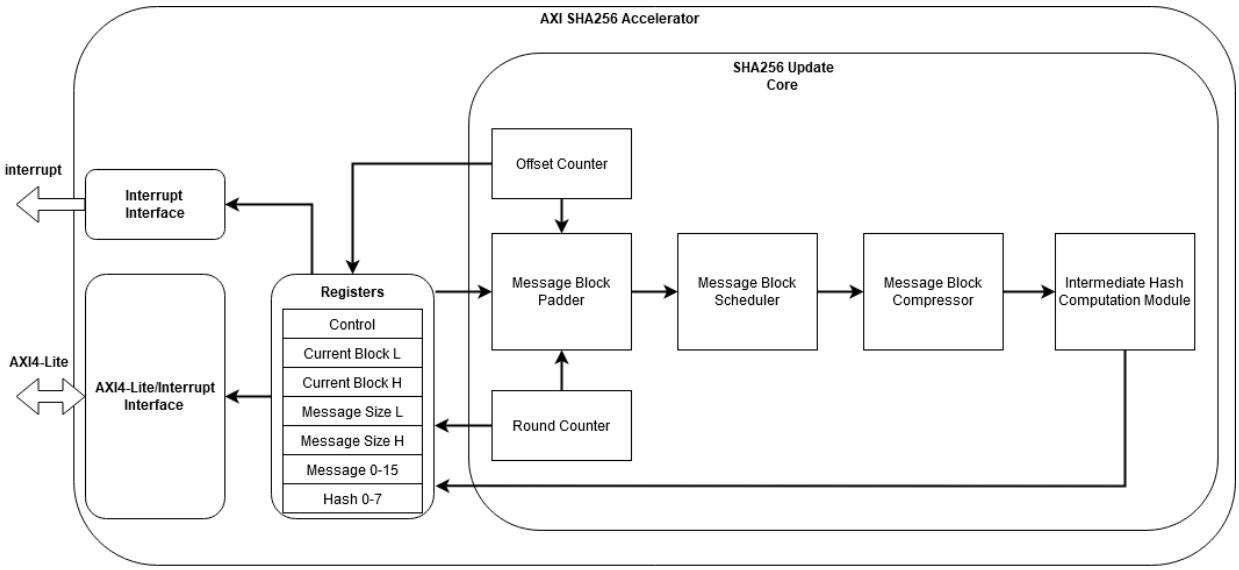

Figure 1: SHA256 AXI4-Lite enabled block diagram

2.0 Accelerator Specification

All specifications given are based off the Vivado Default synthesis and implementation settings.

2.1 Performance

Device	Speed Grade	F_{MAX}
Artix 7	-1	100 MHz

2.2 Resource Utilization

Device Parameters	Device Resources		
Generate Padder	**Slices**	**Registers**	**LUTs**
No	570	1692	1422
Yes	617	1798	1602

2.3 Port Descriptions

Signal Name	Interface	Description
s_sha256_irq	Interrupt	SHA256 Interrupt request
s_axi_aclk	Clock	AXI Clock
s_axi_aresetn	Reset	AXI Reset, active-low
s_axi_*	AXI4-Lite	AXI4-Lite Slave Interface [1]

2.0 Configuration Parameters

C_SHA256_BIG_ENDIAN	CUR_BLOCK, MSG_SIZE, and MSGx registers are in big endian mode. 1 = Big endian mode 0 = Little endian mode (default)
C_SHA256_BIG_ENDIAN_HASH	HASHx registers are in big endian mode. 1 = Big endian mode (default) 0 = Little endian mode
C_SHA256_GEN_PADDER	Generate the padder module in the accelerator. If one isn't generated, then it is up to the drivers to pad the message before loading words into the SHA256_MSGx registers. 1 = Generate padder (default) 0 = Don't generate padder

3.0 Registers

This table lists the address offsets from the base address of the SHA256 accelerator.

Register	Offset
SHA256_CON	0x00
SHA256_MSG_SIZE_L	0x04
SHA256_MSG_SIZE_H	0x08
SHA256_CUR_BLOCK_L	0x0C
SHA256_CUR_BLOCK_H	0x10
SHA256_MSG0	0x14
SHA256_MSG1	0x18
SHA256_MSG2	0x1C
SHA256_MSG3	0x20
SHA256_MSG4	0x24
SHA256_MSG5	0x28
SHA256_MSG6	0x2C
SHA256_MSG7	0x30
SHA256_MSG8	0x34
SHA256_MSG9	0x38
SHA256_MSG10	0x3C
SHA256_MSG11	0x40
SHA256_MSG12	0x44
SHA256_MSG13	0x48
SHA256_MSG14	0x4C
SHA256_MSG15	0x50
SHA256_HASH0	0x54
SHA256_HASH1	0x58
SHA256_HASH2	0x5C
SHA256_HASH3	0x60
SHA256_HASH4	0x64
SHA256_HASH5	0x68
SHA256_HASH6	0x6C
SHA256_HASH7	0x70

SHA256_CON

U	R/W (2)	R/W (2)	R/W (2)	R	R/W	R/W
-	HASH_DONE (3)	BLOCK_DONE	ERROR	BUSY	UPDATE	EN
[31:6]	[5]	[4]	[3]	[2]	[1]	[0]

bit 31-6 **Unimplemented:** Read as '0'

HASH_DONE: Hash of message done status bit (2)
bit 5 1 = Hash of message is done
0 = Hash of message is not done

BLOCK_DONE: Hash of message block done status bit (2)
bit 4 1 = Hash of message block is done
0 = Hash of message block is not done

ERROR: Unexpected error status bit (2)
bit 3 1 = Accelerator has encountered an unexpected error
0 = Accelerator is operating normally

BUSY: Hash computation in progress status bit
bit 2 1 = The hash computation is in progress
0 = There is no current process running

UPDATE: Start hash of message block control bit
bit 1 1 = Start intermediate hash
0 = Don't start intermediate hash (1)

EN: SHA256 enable control bit
bit 0 1 = The module is enabled
0 = The module is disabled, and all internal states are reset

(1) This bit is automatically cleared by the hardware. For acknowledgement of an update, the busy bit should transition from a '0' to '1'.
(2) The HASH_DONE and BLOCK_DONE bits can only be cleared with '0' after they are '1' to acknowledge the completion. See section 4.4 for more details.
(3) The HASH_DONE bit will always read '0' if a hardware padder is not used. See section 4.10.

SHA256_MSG_SIZE_L

C_SHA256_BIG_ENDIAN = 0

R/W
MSG_SIZE [31:0]
[31:0]

bit 31-0 **MSG_SIZE_L:** Lower 32 bits of MSG_SIZE

C_SHA256_BIG_ENDIAN = 1

R/W
MSG_SIZE [0:31]
[0:31]

bit 0-31 **MSG_SIZE_L:** Lower 32 bits of MSG_SIZE

SHA256_MSG_SIZE_H

C_SHA256_BIG_ENDIAN = 0

R/W
MSG_SIZE [63:32]
[31:0]

bit 31-0 **MSG_SIZE_H:** Upper 32 bits of MSG_SIZE

C_SHA256_BIG_ENDIAN = 1

R/W
MSG_SIZE [32:63]
[0:31]

bit 0-31 **MSG_SIZE_H:** Upper 32 bits of MSG_SIZE

SHA256_CUR_BLOCK_L

C_SHA256_BIG_ENDIAN = 0

R
CUR_BLOCK [31:0]
[31:0]

bit 31-0 **CUR_BLOCK_L:** Lower 32 bits of CUR_BLOCK

C_SHA256_BIG_ENDIAN = 1

R
CUR_BLOCK [0:31]
[0:31]

bit 0-31 **CUR_BLOCK_L:** Lower 32 bits of CUR_BLOCK

SHA256_CUR_BLOCK_H

C_SHA256_BIG_ENDIAN = 0

U	R
-	CUR_BLOCK [54:32]
[31:23]	[22:0]

bit 31-23 **Unimplemented:** Read as ‘0’

bit 22-0 **CUR_BLOCK_H:** Upper 23 bits of CUR_BLOCK

C_SHA256_BIG_ENDIAN = 1

R	U
CUR_BLOCK [32:54]	-
[0:22]	[23:31]

bit 0-22 **CUR_BLOCK_H:** Upper 23 bits of CUR_BLOCK

bit 23-31 **Unimplemented:** Read as ‘0’

SHA256_MSGx

C_SHA256_BIG_ENDIAN = 0

R/W
MSGx [31:0]
[31:0]

bit 31-0 **MSGx:** Message block word

C_SHA256_BIG_ENDIAN = 1

R/W
MSGx [0:31]
[0:31]

bit 0-31 **MSGx:** Message block word

SHA256_HASHx

C_SHA256_BIG_ENDIAN_HASH = 0

R
HASHx [31:0]
[31:0]

bit 31-0 **HASHx:** Message block word

C_SHA256_BIG_ENDIAN_HASH = 1

R
MSGx [0:31]
[0:31]

bit 0-31 **HASHx:** Message block word

4.0 Operation

4.1 Overview

1. Enable accelerator by setting the EN bit in the SHA256_CON register
2. Load message size (MSG_SIZE) into SHA256_MSG_SIZE_L and SHA256_MSG_SIZE_H registers
3. Load the 16 message block registers with the next 16 words of the message
4. Set the UPDATE bit in the SHA256_CON register
5. Wait for the BLOCK_DONE bit in the SHA256_CON register to go high
6. If the HASH_DONE bit in the SHA256_CON register is high, then copy the HASHx values and acknowledge the completion by clearing the HASH_DONE bit. If the HASH_DONE bit is low, then repeat steps 3-6 and acknowledge the BLOCK_DONE bit by clearing it

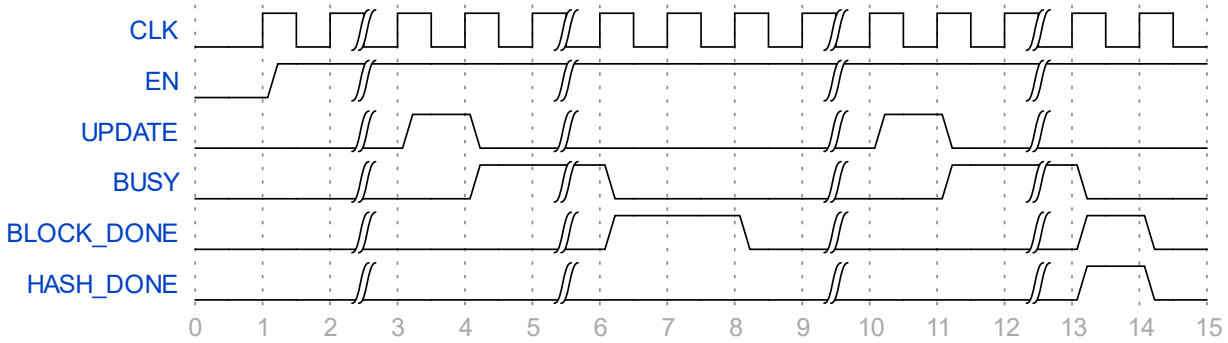

Figure 2: Hash of a message requiring 2 message blocks

The example waveform above shows the hash of a message requiring two message blocks. First, the accelerator is enabled, then between time step 2 and 3, the message size and message block registers were loaded (SHA256_MSG_SIZE_L, SHA256_MSG_SIZE_H, and SHA256_MSG0 through SHA256_MSG15). At timestamp = 3, the UPDATE control bit is set and acknowledged on the following clock when the BUSY status bit transitions from low to high. Between timestamps 5 and 6, the hash is being computed, were on timestamp 6, BUSY goes low and the BLOCK_DONE status bit goes high. The BLOCK_DONE bit is acknowledged on timestamp 8, and the remainder of the message words were loaded into the SHA256_MSGx registers between timestamps 9 and 10. Another UPDATE was issued, and the last message block hash was started. Finally, between timestamps 12 and 13, the message hash completed with the HASH_DONE and BLOCK_DONE status bits going high, then being acknowledged on timestamp 14.

4.2 Enable bit

The EN bit (SHA256_CON[0]) globally enables the accelerator. When enabled, it allows for hashes to be computed. When disabled, it holds the accelerator in a reset state.

4.3 Update and busy bits

The UPDATE bit (SHA256_CON[1]) starts the hash computation given that the module is enabled. When set, it is automatically cleared on the next cycle and is only acknowledged when the BUSY bit transitions from ‘0’ to ‘1’. Once the hash computation has begun, it cannot be paused and the BUSY

bit (SHA256_CON[2]) goes high. Until the BUSY bit goes low and the BLOCK_DONE (SHA256_CON[3]) and HASH_DONE (SHA256_CON[4]) bits are acknowledged (written with ‘0’s) will the UPDATE bit be able to be acknowledged again. This means before any other intermediate hash computation can begin, the previous hash needs to have finished and its completion needs to be acknowledged.

If the hardware padder is used, the accelerator will automatically update itself if it detects that the padding has overflowed onto a second message block. If software padding is used, then it is up to the software to correctly pad the message and manually update the overflowed block.

4.4 Error bit

The error bit indicates that the accelerator has entered some unknown state and requires a reset by disabling and re-enabling the accelerator.

4.5 Block and hash completion bits

The BLOCK_DONE and HASH_DONE indicate the completion of a message block hash and message hash, respectfully. Once either are set, they block further updates from occurring (as with the BUSY bit). When only the BLOCK_DONE bit is high, the hash computation is not complete and requires at least one more intermediate hash completion. Once acknowledged, another UPDATE can be issued. If the HASH_DONE bit is high, then the BLOCK_DONE bit will also be high. If a hash of another message is desired, this bit should be cleared, and by clearing the HASH_DONE bit, it also clears the BLOCK_DONE bit and resets the accelerator state. However, the HASHx registers will remain unchanged to allow for the software drivers to copy the results before or after acknowledging the HASH_DONE bit, but before the start of the next hash.

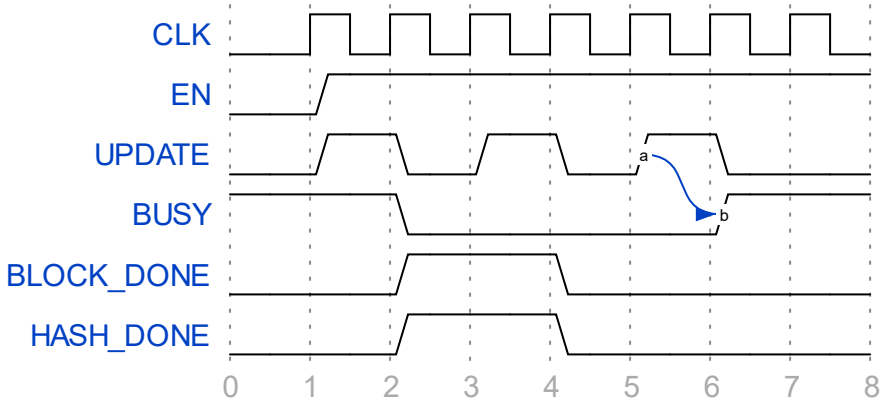

Figure 3: UPDATE acknowledgement

The waveform above shows scenarios where an update issue was ignored and where it was accepted. At timestamp 2, the update issue was ignored because the BUSY bit was high. Likewise, at timestamp 4, the update issue was ignored because the BLOCK_DONE and HASH_DONE bits were high, although the update request would have been ignored if either one of them were high. Finally, in timestamp 6, the update request was accepted as the BUSY bit transitions from ‘0’ to ‘1’.

4.6 Message size registers

SHA256_MSG_SIZE_L and SHA256_MSG_SIZE_H together make up the message size, MSG_SIZE, a 64-bit parameter. MSG_SIZE is the size of bytes in the message to be hashed where SHA256_MSG_SIZE_L holds the lower 32 bits and SHA256_MSG_SIZE_H holds the upper 32 bits for both endianness.

4.7 Current block registers

SHA256_CUR_BLOCK_L and SHA256_CUR_BLOCK_H together make up the current block, CUR_BLOCK, a 55-bit parameter. SHA256_CUR_BLOCK_L is the lower 32 bits of CUR_BLOCK and SHA256_CUR_BLOCK_H is the upper 23 bits. CUR_BLOCK indicates the block currently being hashed, where a block is a 512-bit, or 64-byte, section of the overall message. CUR_BLOCK is used to indicate the progress of the hash and can be used by software drivers to load the correct data from the message into the message block registers (SHA256_MSGx), acting like a pointer offset. It can also be used as a secondary source for indicating the end of hash. The message size in bytes and last block are related by the following expression:

Last Block = MSG_SIZE >> 6

4.8 Message block registers

The SHA256_MSGx registers make up the 512-bit message block. Not all registers need to be filled as long as the message size (MSG_SIZE) is correctly set. If using the hardware padder and if the message is not word aligned, i.e. its message size is not an even multiple of 4, then the unused bytes are don’t-cares as it will know which byte is the last in the buffer. If a software padder is used, then the entire message block buffer needs to be filled with a padded message block.

4.9 Hash registers

The SHA256_HASHx registers contain parts of the full 256-bit hash. The full hash is found through concatenation in the following order (dropping the SHA256_ prefix):

HASH0 | HASH1 | HASH2 | HASH3 | HASH4 | HASH5 | HASH6 | HASH7

4.10 Padding

The role of padding in the SHA256 algorithm is to take a message of any size and concatenate it to make the size (in bits) an integer multiple of 512. The accelerator can be configured to have either the hardware padding, to maximize performance, or software padding, to save area. For details about the padding algorithm, see the SHA256 NIST specification [2]. When using software padding, the HASH_DONE bit will always be zero and requires the software to keep track of the progress.

In the SHA256 specification, the message words are big endian while the bit-field for the message size is little endian. When performing software padding, it will be required that the endianness of message size field regardless of the configured accelerator endianness.

4.11 Interrupts

The interrupt signal follows the BLOCK_DONE bit in the SHA256_CON register, so when BLOCK_DONE is ‘0’, there is no interrupt request (IRQ), and when it is ‘1’, there is an IRQ. The accelerator only indicates that there is a pending interrupt and does not have any other logic pertaining to interrupt handling. It is up to the CPU, interrupt controller, or other mechanism to configure/enable the SHA256 IRQ, see to the IRQ, and jump to the interrupt service routine (ISR). The ISR should acknowledge the BLOCK_DONE bit if the block is done or the HASH_DONE bit if the hash is done, which also acknowledges the BLOCK_DONE bit simultaneously.

5.0 Revision History

Date	Revision	Author	Description
5/7/20	1.0	PW	Initial draft

6.0 Resources

[1] National Institute of Standards and Technology (NIST), "NIST," August 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[2] ARM, "AXI and ACE Protocol Specification," [Online]. Available: https://developer.arm.com/docs/ihi0022/d.