**AXI SHA256 Accelerator Reference Manual**

**Revision 1.0**

Table of Contents

[1.0 Introduction 3](#_Toc39748960)

[1.1 Features 3](#_Toc39748961)

[1.2 Functional Description 3](#_Toc39748962)

[2.0 Accelerator Specification 4](#_Toc39748963)

[2.1 Performance 4](#_Toc39748964)

[2.2 Resource Utilization 4](#_Toc39748965)

[2.3 Port Descriptions 4](#_Toc39748966)

[2.0 Configuration Parameters 5](#_Toc39748967)

[3.0 Registers 6](#_Toc39748968)

[SHA256\_CON 7](#_Toc39748969)

[SHA256\_MSG\_SIZE\_L 8](#_Toc39748970)

[SHA256\_MSG\_SIZE\_H 9](#_Toc39748971)

[SHA256\_CUR\_BLOCK\_L 10](#_Toc39748972)

[SHA256\_CUR\_BLOCK\_H 11](#_Toc39748973)

[SHA256\_MSGx 12](#_Toc39748974)

[SHA256\_HASHx 13](#_Toc39748975)

[4.0 Operation 14](#_Toc39748976)

[4.1 Overview 14](#_Toc39748977)

[4.2 Enable bit 14](#_Toc39748978)

[4.3 Update and busy bits 14](#_Toc39748979)

[4.4 Block and hash completion bits 15](#_Toc39748980)

[4.5 Message size registers 15](#_Toc39748981)

[4.6 Current block registers 15](#_Toc39748982)

[4.7 Message block registers 16](#_Toc39748983)

[4.8 Hash registers 16](#_Toc39748984)

[4.9 Padding 16](#_Toc39748985)

[4.10 Interrupts 16](#_Toc39748986)

[5.0 Revision History 17](#_Toc39748987)

[6.0 Resources 17](#_Toc39748988)

# 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.

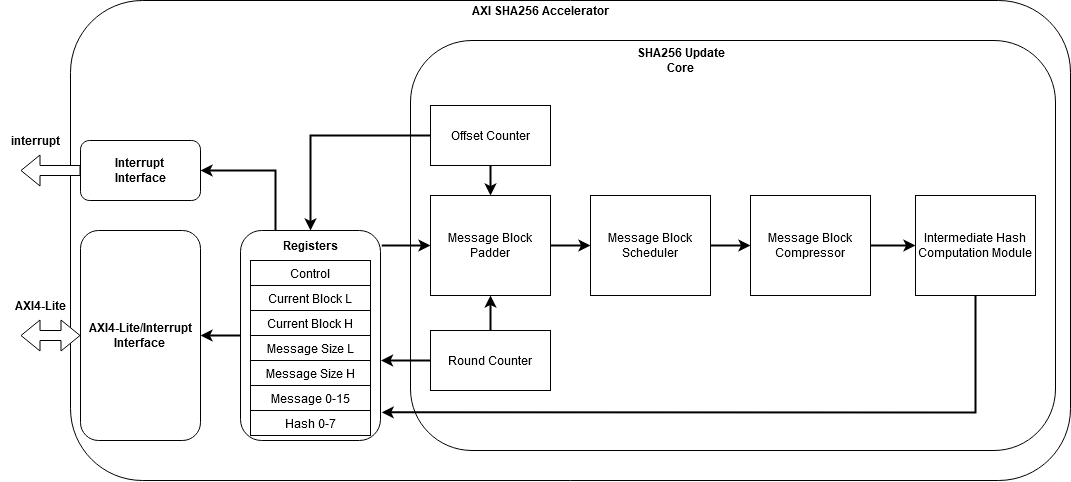


Figure : SHA256 AXI4-Lite enabled block diagram

# 2.0 Accelerator Specification

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

## 2.1 Performance

|  |  |  |
| --- | --- | --- |
| **Device** | **Speed Grade** | **F­MAX** |
| Artix 7 | -1 | 100 MHz |

## 2.2 Resource Utilization

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter Values** | | **Device Resources** | | |
| **Generate Padder** | **Pipeline Stages** | **Slices** | **Flip-Flops** | **LUTs** |
| No | 0 |  |  |  |
| No | 1 |  |  |  |
| No | 2 |  |  |  |
| Yes | 0 | 651 | 1796 | 1593 |
| Yes | 1 |  |  |  |
| Yes | 2 |  |  |  |

## 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 | R/W | R/W |
| - | HASH\_DONE | BLOCK\_DONE | BUSY | UPDATE | EN |
| [31:5] | [4] | [3] | [2] | [1] | [0] |

|  |  |
| --- | --- |
| bit 31-5 | **Unimplemented:** Read as ‘0’ |
| bit 4 | **HASH\_DONE:** Hash of message done status bit **(2)**  1 = Hash of message is done  0 = Hash of message is not done |
| bit 3 | **BLOCK\_DONE:** Hash of message block done status bit **(2)**  1 = Hash of message block is done  0 = Hash of message block is not done |
| bit 2 | **BUSY:** Hash computation in progress status bit  1 = The hash computation is in progress  0 = There is no current process running |
| bit 1 | **UPDATE:** Start hash of message block control bit  1 = Start intermediate hash  0 = Don’t start intermediate hash **(1)** |
| bit 0 | **EN:** SHA256 enable control bit  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.**

## 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

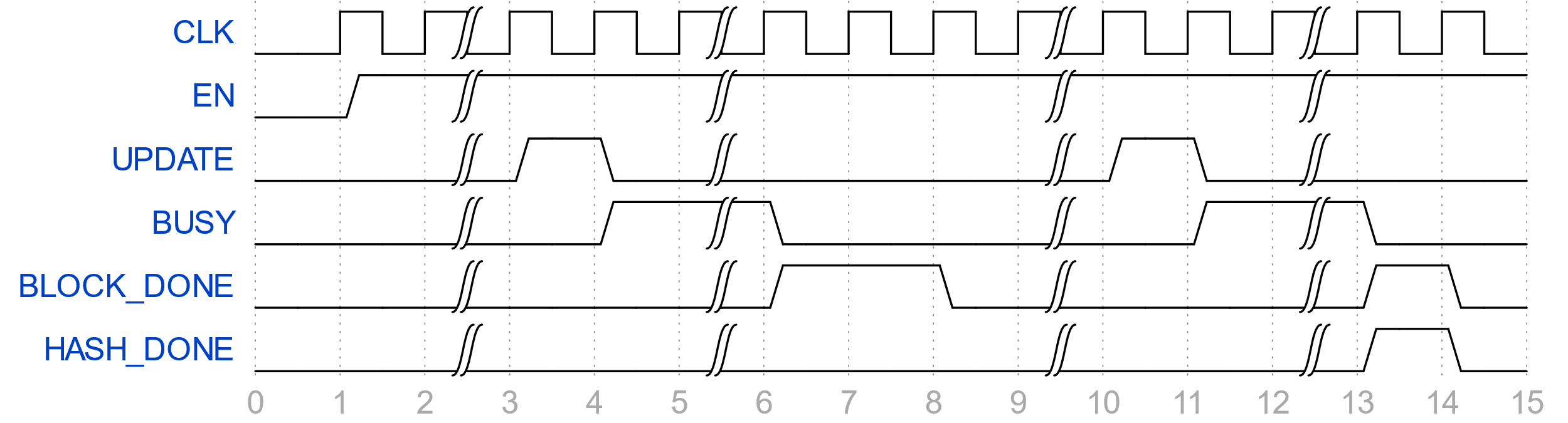


Figure : Hash of a message requiring 2 message blocks

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.

## 4.4 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.

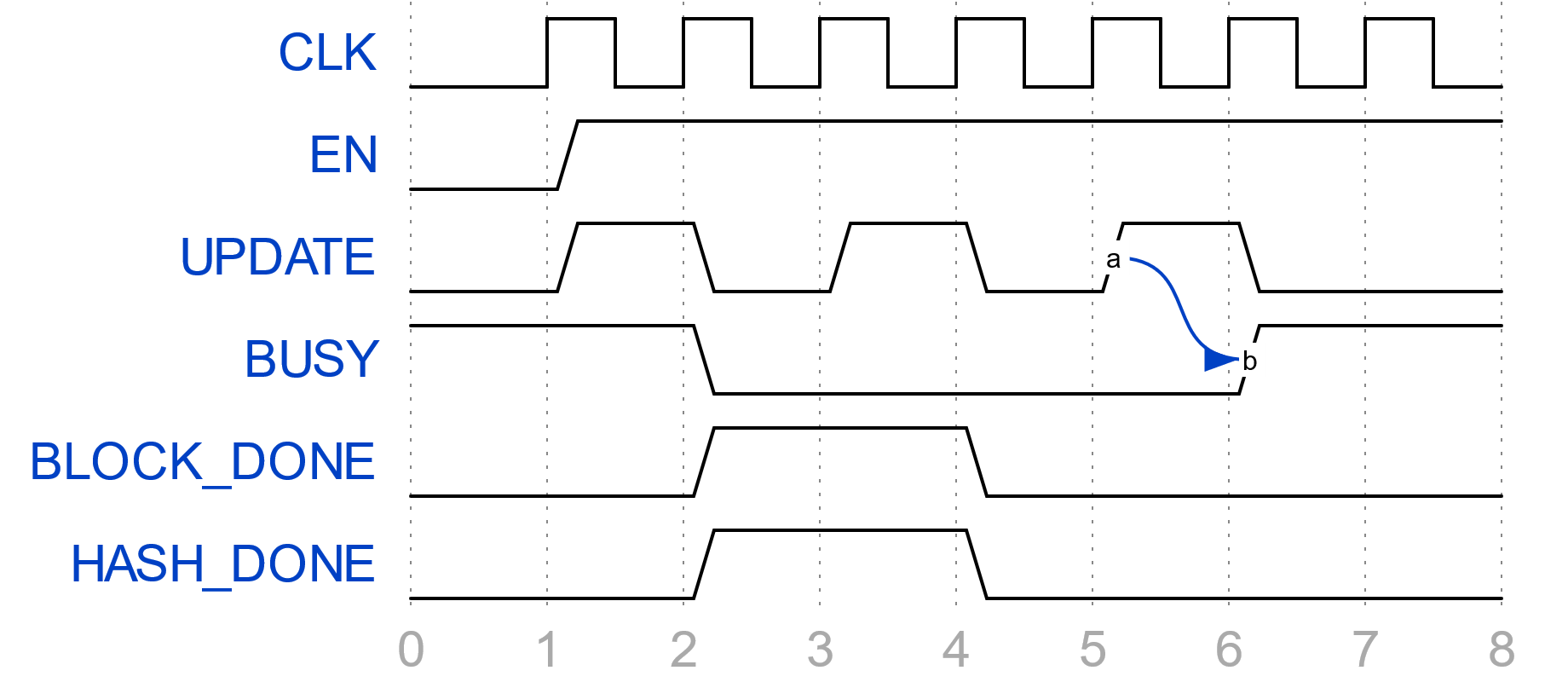


Figure : UPDATE acknowledgement

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’.

## 4.5 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.

## 4.6 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

## 4.7 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.

## 4.8 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

## 4.9 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, see the SHA256 NIST specification [2].

## 4.10 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. |