Using MachXO3D ESB to Implement HMAC SHA256

Reference Design

FPGA-RD-02052-1.0

August 2021

Disclaimers

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AES	Advanced Encryption Standard
ECDSA	Elliptic Curve Digital Signature Algorithm
ESB	Embedded Security Block
HMAC	Keyed-hash Message Authentication Code or Hash-based Message Authentication Code
SHA256	256-bit Secure Hash Algorithm

1. Introduction

The integrity and authenticity of information is increasingly important in Internet protocols today. Keyed-hash Message Authentication Code (HMAC) is a widely accepted authentication code that is commonly used in many Internet protocols. It involves a cryptographic hash function and a secret cryptographic key. Many cryptographic hash functions, such as MD5, or SHA-1, can be used in the calculation of an HMAC. Thus, HMAC-SHA256 uses SHA256 hash function and a secret key for calculation and verification of the message authentication values. For example, after you sent a piece of data along with its hash, the receiver can verify the integrity of the message by calculating the hash of the message and by comparing it with the received hash value. However, the receiver does not know for sure if the message and the hash come from in a trusted resource. In order to establish authenticity, the receiver needs to recalculate the HMAC using a secret key that only the receiver and the trusted resource know. To share the secret key between a trusted resource and receiver, you can use the ECDH function of the ESB engine.

The MachXO3D™ device is the next-generation product family from the MachXO™ product line with key features such as security and on-chip dual boot. The Embedded Security Block (ESB) of the MachXO3D family focuses on security feature and involves cryptographic functions such as Elliptic Curve Digital Signature Algorithm (ECDSA), Advanced Encryption Standard (AES), and HMAC SHA256. This reference design takes advantage of the ESB to deliver the HMAC function abstracting algorithmic details of HMAC from the end user.

2. Reference Design Overview

2.1. Block Diagram

Figure 2.1 shows the block diagram of the reference design.

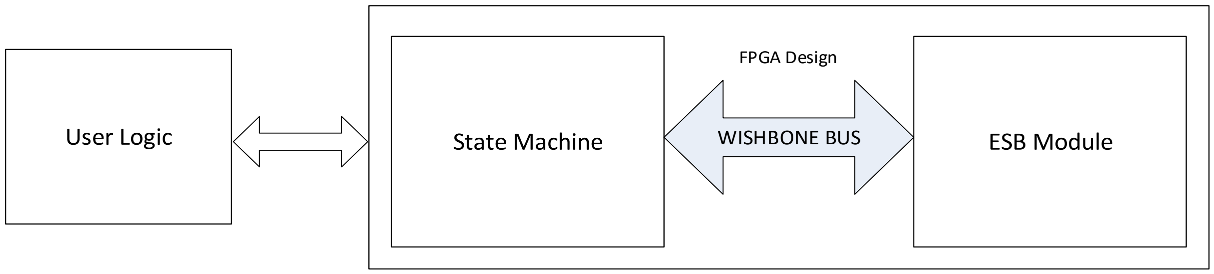


Figure 2.1. Top-Level Block Diagram

2.2. Overview

This design shows how to use the ESB to implement HMAC-SHA256. The interface of the ESB is the WISHBONE bus. The state machine provides detailed steps of accessing the register and the memory of the ESB. The user logic can communicate with the state machine for HMAC-SHA256 implementation.

The features of this reference design include:

- HMAC-SHA256 algorithm using the ESB primitive instance
- Parameterized message size in bytes
- User-defined keys, 32 bytes

3. Functional Description

Message Authentication Code (MAC), as the names suggests, is used for checking the authenticity of any message. MAC is generated from any associated message, and it serves two purposes:

- Assuring message integrity
- Establishing authenticity of the message source

In general, there are two types of algorithms that are used for computing MAC:

- MAC algorithms based on approved block cipher algorithms;
- MAC algorithms based on hash functions, called HMAC algorithms.

In MachXO3D devices, MAC algorithms based on hash functions are used. In particular, ESB supports MAC algorithm that is based on SHA256 hash function, as specified in the name HMAC SHA256. In this algorithm, the input message along with a secret key, is processed through the MAC algorithm. The output of this MAC algorithm is then processed using SHA256 hash function to generate the final 256-bit output.

HMAC SHA256 uses SHA256 hash function and a secret key for calculation and verification of the message authentication values, as shown in Figure 3.1.

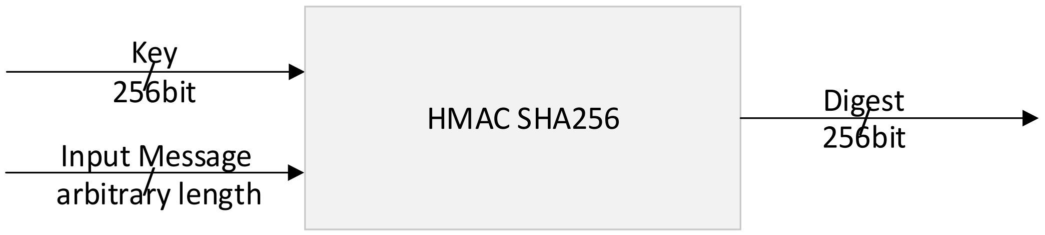


Figure 3.1. High-level Overview of HMAC SHA256 Function

The ESB implements HMAC with SHA-256 hash function. The ESB takes inputs with sizes up to 1984 bytes and produces 32 bytes output. To implement HMAC-SHA256 with a message of a size larger than 1984 bytes, you can use the SHA256 function of the ESB. In addition, you need to implement those XOR operations and ipad/opad concatenation in fabric logic.

4. Design Description

4.1. Parameters of the Design

Table 4.1 lists the parameters of the reference design.

Table 4.1. HMAC SHA256 Parameter Descriptions

Parameter	Description	Value
NUM_BYTE	Specifies the size of message in bytes. Since the size of the memory used to store the message in the ESB is less than 1984 bytes, the maximum value of this parameter is 1984.	1 to 1984
KEY	Message authentication secret key defined by the user. The length of the key is 32 bytes.	Defined by the user.

4.2. Input/Output of the Design

Figure 4.1 is the I/O diagram of the HMAC-SHA256 reference design.

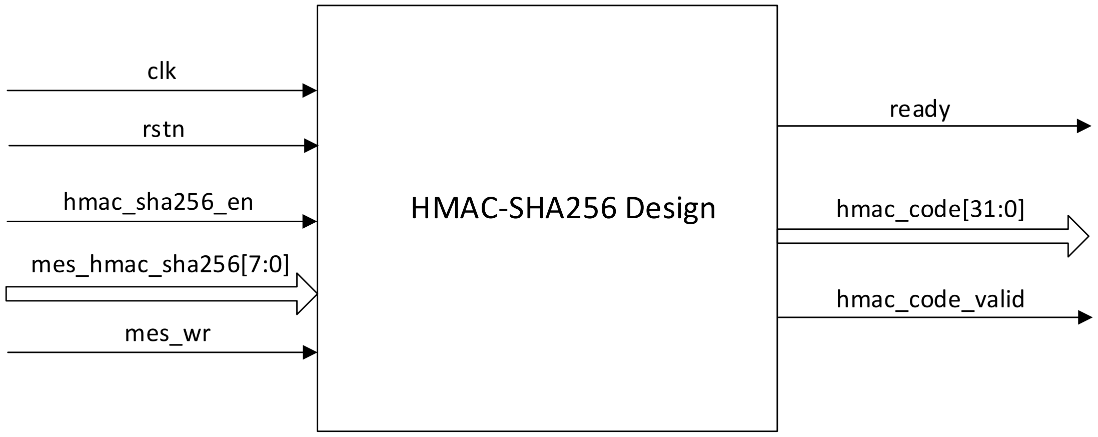


Figure 4.1. I/O Diagram of the HMAC-SHA256 Design

Table 4.2 lists the I/O ports of the reference design.

Table 4.2. Pin Descriptions

Signal	Width	Type	Active	Description
clk	1	Input	Rising edge	System clock
rstn	1	Input	Low	Asynchronous reset
hmac_sha256_en	1	Input	High	Enable signal. When asserted, the design runs HMAC-SHA256 function in the ESB.
mes_hmac_sha256	8	Input	N/A	Message to which HMAC SHA256 is applied.
mes_wr	1	Input	High	Message valid signal. When asserted, message is written.
ready	1	Output	High	Ready output. When asserted, it indicates that the design is ready to accept the next message byte.
hmac_code	32	Output	N/A	HMAC code. 32 bytes.
hmac_code_valid	1	Output	High	HMAC code valid signal. When asserted, HMAC code is validated.

Before running HMAC SHA256, set the parameter NUM_BYTE and KEY based on the size of the message in bytes and the secret key value. Then assert the hmac_sha256_en signal to run HMAC SHA256. Once the design detects the rising edge of the enable signal hmac_sha256_en, it initiates the ESB for the HMAC operation and asserts the ready signal to indicate that it is ready to accept message from user logic. The user logic needs to provide the message bytes along with the control signal mes_wr. The message is sent using the procedure below:

1. The design informs the user logic that it is ready to receive the new message by asserting the ready signal for one clock cycle.
2. After asserting the ready signal (Figure 4.2), the design waits for the signal mes_wr to go HIGH. If the design finds the signal mes_wr asserted by the user logic, it takes the signal mes_hmac_sha256 as the next message byte. The message byte order is Little Endian.
3. The design counts the size of the message in bytes until it reaches the value of NUM_BYTE. After the last byte of the message is delivered, the user logic needs to wait for the hmac_code_valid signal to be active.

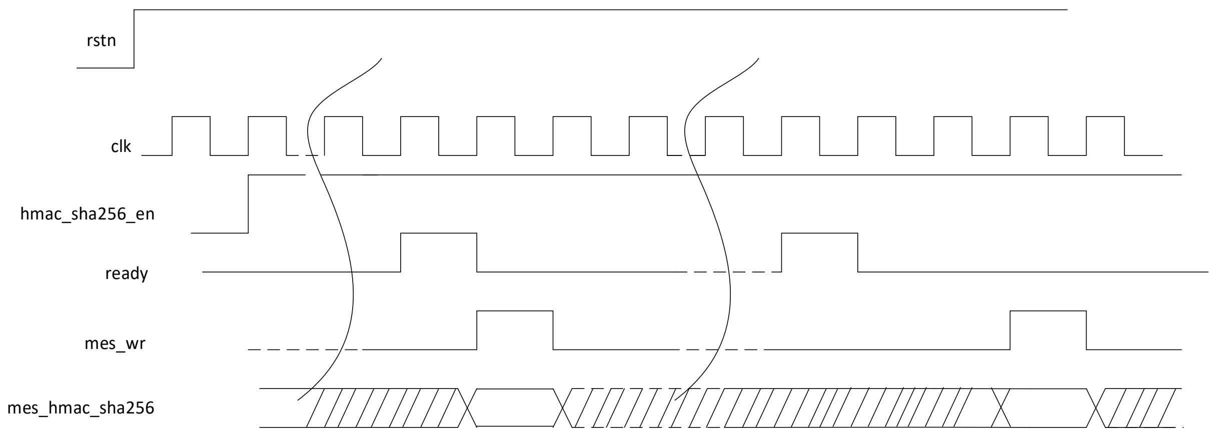


Figure 4.2. Timing Diagram of Signal ready, Message Valid, and Message

The design asserts the signal hmac_code_valid to tell the user logic to receive the HMAC code (Figure 4.3). The signal hmac_code_valid is asserted eight times to provide the 32 bytes code, and each time, it is asserted for one clock cycle. The HMAC code is Little Endian.

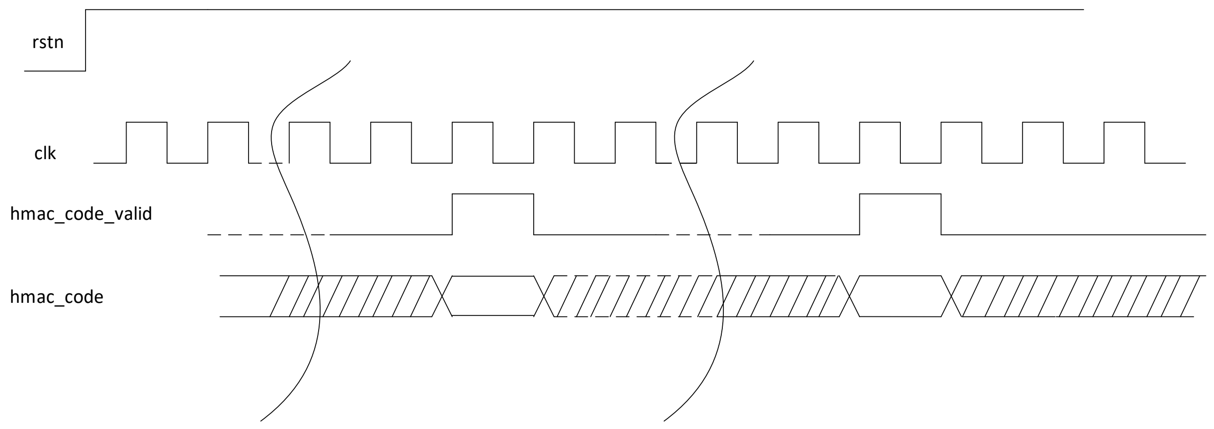


Figure 4.3. Timing Diagram of Signal hmac_code_valid and hmac_code

4.3. ESB Registers for HMAC-SHA256

Table 4.3 lists the registers of the design. Refer to the MachXO3D Embedded Security Block (FPGA-TN-02091) for detailed information on the registers listed below.

Table 4.3. Register Descriptions

Register Type	Register Name	Address	Read/Write	Description
Control Register	r0_gp0	18’h2_0020	Read	Check for busy status of ESB: 0xB0: READY to get a new command 0xB2: ESB operation done
	ri_ctrl3	18'h2_000c	Write	Set the size of message in bytes
	ri_ctrl1	18'h2_000c	Write	ESB function: 0x06: Enable HMAC-SHA256 0x00: Disable HMAC-SHA256
Memory	HMAC_key_0	18’h1_F800	Write/Read	HMAC-SHA256 Key [31:0]
	HMAC_key_1	18’h1_F804	Write/Read	HMAC-SHA256 Key [63:32]
	HMAC_key_2	18’h1_F808	Write/Read	HMAC-SHA256 Key [95:64]
	HMAC_key_3	18’h1_F80C	Write/Read	HMAC-SHA256 Key [127:96]
	HMAC_key_4	18’h1_F810	Write/Read	HMAC-SHA256 Key [159:128]
	HMAC_key_5	18’h1_F814	Write/Read	HMAC-SHA256 Key [191:160]
	HMAC_key_6	18’h1_F818	Write/Read	HMAC-SHA256 Key [223:192]
	HMAC_key_7	18’h1_F81C	Write/Read	HMAC-SHA256 Key [255:224]
	HMAC_result_0	18’h1_F820	Write/Read	HMAC-SHA256 Result [31:0]
	HMAC_result_1	18’h1_F824	Write/Read	HMAC-SHA256 Result [63:32]
	HMAC_result_2	18’h1_F828	Write/Read	HMAC-SHA256 Result [95:64]
	HMAC_result_3	18’h1_F82C	Write/Read	HMAC-SHA256 Result [127:96]
	HMAC_result_4	18’h1_F830	Write/Read	HMAC-SHA256 Result [159:128]
	HMAC_result_5	18’h1_F834	Write/Read	HMAC-SHA256 Result [191:160]
	HMAC_result_6	18’h1_F838	Write/Read	HMAC-SHA256 Result [223:192]
	HMAC_result_7	18’h1_F83C	Write/Read	HMAC-SHA256 Result [255:224]
	Message	18’h1_F840~18’h1_FFFF	Write/Read	Message input

The interface between this reference design and the ESB is WISHBONE. The state machine (Figure 2.1) uses the procedure below:

1. Poll the register ro_gpo in the ESB until the value of this register is 0xB0.
2. Write the size of message in bytes to the register ri_ctrl3 in the ESB based on the value of NUM_BYTE.
3. Write the secret key to the memory in the ESB based on the KEY provided by the user.
4. Assert the ready signal and wait for the signal mes_wr to be active.
5. If the asserting of the signal mes_wr is detected, receive the message byte, and send message byte to the ESB.
6. If all the message bytes are received, set register ri_ctrl1 in the ESB to 0x06.
7. Read the register ro_gpo in the ESB until the value of this register is 0xB2.
8. Read the HMAC code from the memory in the ESB with the assertion of the signal hmac_code_valid.

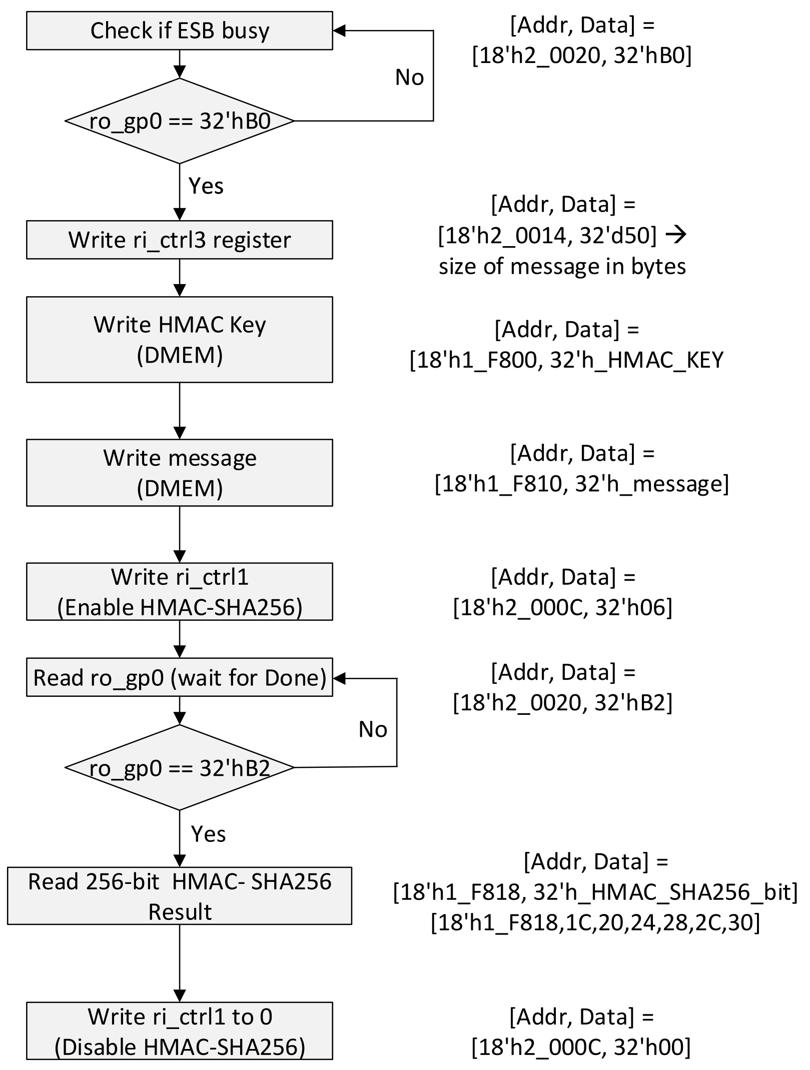

Check if ESB busy
[Addr, Data] = [18'h2_0020, 32'hB0]
No
ro_gp0 == 32'hB0
Yes
Write ri_ctrl3 register
[Addr, Data] = [18'h2_0014, 32'd50] → size of message in bytes
Write HMAC Key (DMEM)
[Addr, Data] = [18'h1_F800, 32'h_HMAC_KEY
Write message (DMEM)
[Addr, Data] = [18'h1_F810, 32'h_message]
Write ri_ctrl1 (Enable HMAC-SHA256)
[Addr, Data] = [18'h2_000C, 32'h06]
Read ro_gp0 (wait for Done)
[Addr, Data] = [18'h2_0020, 32'hB2]
No
ro_gp0 == 32'hB2
Yes
Read 256-bit HMAC- SHA256 Result
[Addr, Data] = [18'h1_F818, 32'h_HMAC_SHA256_bit] [18'h1_F818,1C,20,24,28,2C,30]
Write ri_ctrl1 to 0 (Disable HMAC-SHA256)
[Addr, Data] = [18'h2_000C, 32'h00]

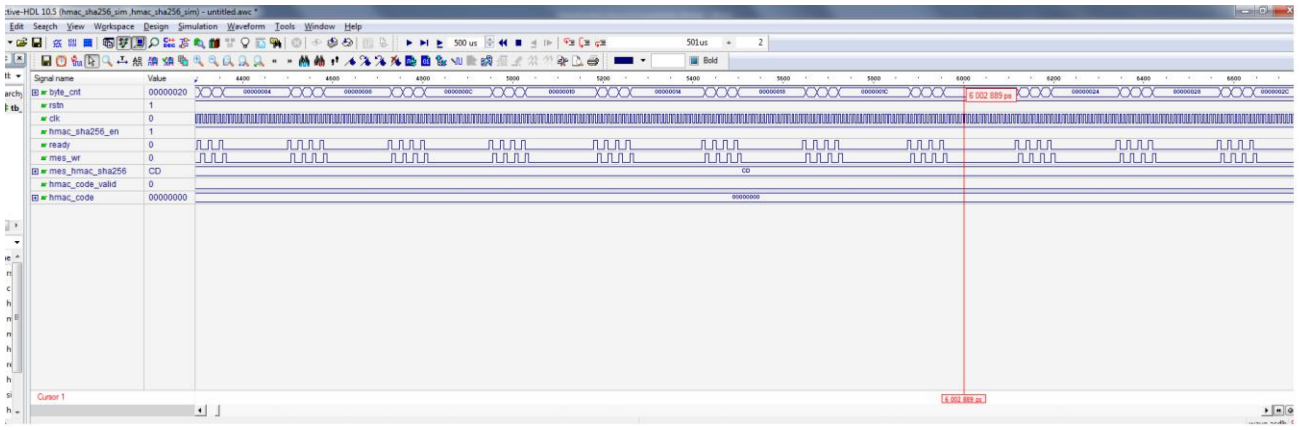
Figure 4.4. Configuration Flow for Enabling HMAC SHA256 Function

5. Simulation and Verification

The simulation takes a golden pattern as an example. It first sets the parameter NUM_BYTE to be 50 and the KEY to be 256'h0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f20.

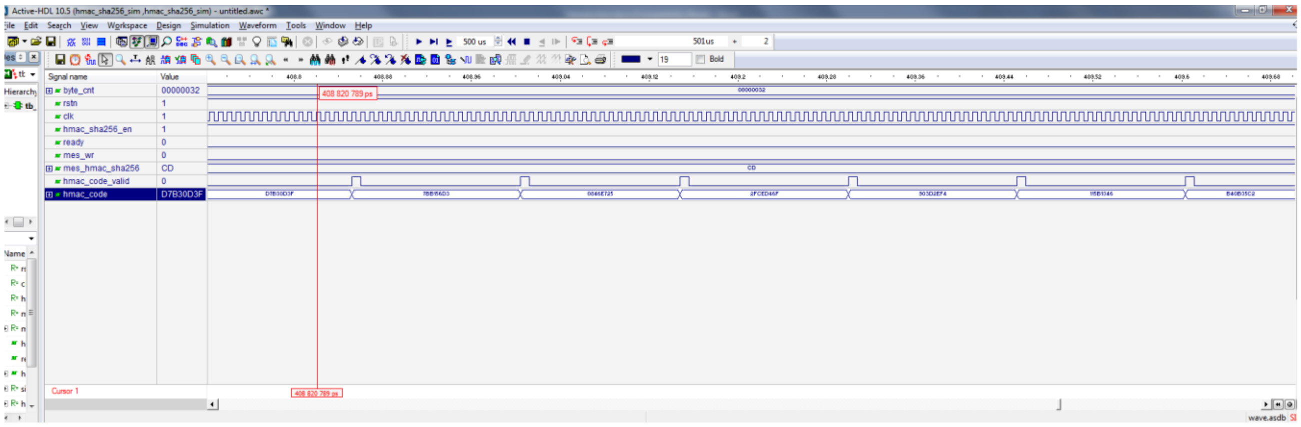
After reset (Figure 5.1), it asserts the hmac_sha256_en signal and then input the message ('hcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcdcd).

The design processes on the message with the key and generate the HMAC code output using the ESB engine (Figure 5.1). The testbench compares HMAC code output with the golden output using the ESB engine to check the functionality (Figure 5.2).



Signal name	Value
byte_cnt	00000020
rstn	1
clk	0
hmac_sha256_en	1
ready	0
mes_wr	0
mes_hmac_sha256	CD
hmac_code_valid	0
hmac_code	00000000

Figure 5.1. Emulated HMAC SHA256 Message Written Transmission



Signal name	Value
byte_cnt	00000032
rstn	1
clk	1
hmac_sha256_en	1
ready	0
mes_wr	0
mes_hmac_sha256	CD
hmac_code_valid	0
hmac_code	D7B30D3F

Figure 5.2. Emulated HMAC SHA256 Code Output Transmission

6. Implementation

This reference design is implemented in Verilog HDL using Lattice Diamond® software. The synthesis tool is set to Synplify Pro®. When using this design in a different device, density, speed, or grade, performance and utilization may vary.

Table 6.1. Performance and Resource Utilization

Device Family	Language	Utilization	Operating Frequency	ESB Primitive	OSC Primitive	Number of I/O
LCMXO3D-9400HC	Verilog HDL	230 LUTs	>50 MHz	Yes	Yes	46

Note: Performance and utilization characteristics are generated with LCMXO3D-9400HC, using Diamond 3.11 design software.

Reference

MachXO3D Embedded Security Block (FPGA-TN-02091)

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.0, August 2021

Section	Change Summary
All	Production release.