Nexus CRE Module IP

IP Version: v1.2.0

User Guide

FPGA-IPUG-02067-1.5

June 2025

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
AES	Advance Encryption Standard
AHB-L	Advanced High Performance Bus - Lite
APB	Advance Peripheral Bus
CRE	Cryptographic Engine
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
FIFO	First-In, First-Out
HMAC	Hash Based Message Authentication Code
IP	Intellectual Property
LMMI	Lattice Memory Mapped Interface
LUT	Lookup Table
SHA	Secure Hash Algorithm
TRNG	True Random Number Generation

1. Introduction

This document provides technical information about the CRE Module that is supported in Lattice FPGA devices built on the Lattice Nexus™ platform. It aims to provide information essential for IP/System developing, verification, integration, testing and validation. In general, design specification from RTL up to IP packaging, IP generation and integration with Lattice Radiant™ software is covered in this document.

1.1. Overview of the IP

CRE stands for Cryptographic Engine which provides user mode cryptographic functions like encryption, authentication, hash, and random number generation.

1.2. Quick Facts

Table 1.1. Summary of the CRE IP

IP Requirements	Supported Devices	CertusPro™-NX, Certus™-NX, MachX05™-NX (except LFMXO5-15D and LFMXO5-55TD), CrossLink™-NX, Certus™-NX-RT, and CertusPro™-NX-RT
Resource Utilization	Supported User Interface	Lattice Memory Mapped Interface (LMMI), FIFO Streaming Interface, Advanced Peripheral Bus (APB), and Advanced High-Performance Bus (AHB) – Lite
Design Tool Support	Lattice Implementation	IP Core v1.2.0 – Lattice Radiant software 2024.1 or later
	Synthesis	Lattice Synthesis Engine, Synopsys® Synplify Pro® for Lattice
	Simulation	QuestaSim

1.3. Features

This module is based on the built-in Security Hard IP with the following features:

- Supports the following user mode security features
  - High throughput Secure Hash Algorithm – 256 bits (SHA-256)
  - Hash Based Message Authentication Code – HMAC-SHA
  - High throughput Advance Encryption Standard – 128/256 bits (AES-128/256)
    - FIFO Interface estimate throughput – 8.53 bits per cycle; 1.28 Gbps at 150 MHz
    - LMMI Interface estimate throughput – 2.2 bits per cycle; 342 Mbps at 100 MHz
- Supports True Random Number Generation
- Supports multiple bus interfaces:
  - Lattice Memory Mapped Interface (LMMI)
  - LMMI + FIFO (High-speed SHA/AES)
  - Advanced High-Performance Bus (AHB) – Lite
  - Advanced Peripheral Bus (APB)

1.4. Licensing and Control Pack Information

The IP specific license string and control pack are required to enable full use of the CRE IP in a complete, top-level design. To request the control pack and IP license string, submit a tech support case to www.latticesemi.com/techsupport.

2. Functional Description

2.1. Block Diagram

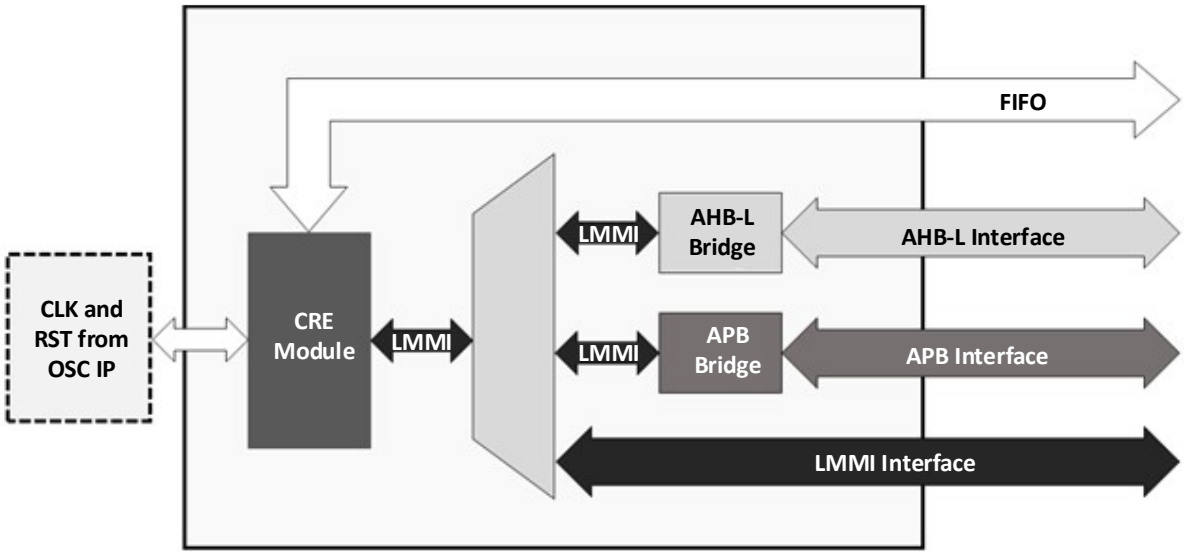

Figure 2.1. CRE Module Block Diagram

Notes:

- The CLK and RST need to be connected to the CRE Module. If you do not use the OSC primitive, the IP has the option to automatically include the OSC IP during generation to avoid encountering Synthesis Errors.
- The FIFO I/O only applies to FIFO control pins; the data pins are shared with the LMMI. The FIFO interface also requires the information to be sent through the LMMI bus, hence the FIFO pins are only available if the LMMI interface is also selected. More information is discussed below.

2.2. Security Functions and Features

2.2.1. Secure Hash Algorithm (SHA-256)

The IP also features a high throughput SHA engine, for fast encryption of messages using the SHA function. The SHA can be configured using a HMAC key with proper data for faster throughput, or through automatic handling of the HMAC by the security engine.

2.2.2. Advance Encryption Standard (AES-128/256)

The IP can also process the encryption and decryption of messages based on the AES-128/256 standard using a common key.

2.2.3. True Random Number Generation

The IP can also provide the true random number generator as a standalone feature for customer use cases, regardless of whether the case is related or not related to security functions.

2.2.4. Differential Power Analysis (DPA) Mitigation

The IP includes an additional level of security to protect itself from several techniques involving power analysis. All features can be turned on or turned off independently.

2.2.4.1. Clock Randomization

The IP uses a 41b LFSR with random seed to make clock enable signal, thus effective clock frequency changes randomly. This affects AES.

2.2.4.2. Random Noise Generation

The IP turns on random number generator, thus random noise hides the current pattern from internal operation. This affects all operations.

2.3. Functional Overview

The CRE Module can be configured using one of four different interfaces for maximum design flexibility.

2.3.1. LMMI Interface

The LMMI Interface is the native interface of the IP, and the most resource-efficient interface of the CRE Module. Using this interface, you can directly use all native IP features without using any fabric or additional control signals.

2.3.2. LMMI + FIFO Interface (Generic Interface)

The LMMI + FIFO interface is similar to the native LMMI interface with the addition of a FIFO control port. The FIFO shares its input and output data connection with the LMMI input and output connection; hence, you must design additional circuitry to fully take advantage of this interface. The benefits of the FIFO data path are the increased throughput for AES/SHA transactions. In this configuration, you can still utilize all the features of the IP while minimizing resource utilization.

Important: The LMMI write and read data ports are shared with the FIFO interface. Proper care must be taken when writing/sampling data to/from the IP using different clocks. This document assumes that you have properly taken care of any possible clock crossing issues which could arise from the use of asynchronous clocks.

2.3.3. AHB-Lite Interface

The CRE Module is designed to be fully compatible with the AHB-Lite standard. The bridge allows translation of signals from the AHB-L bus into LMMI compatible signals, which can be directly interpreted by the core CRE security IP.

2.3.4. APB Interface

The CRE Module is designed to be fully compatible with the APB standard. The bridge allows translation of signals from the APB bus into LMMI compatible signals, which can be directly interpreted by the core CRE Module.

2.4. Signal Descriptions

Table 2.1 CRE Module Ports

Pin Name	Direction	Width (Bits)	Description
Core IP Signals			
cfg_clk_i	IN	1	Config Clock Signal (from OSC IP)
cre_clk_i	IN	1	CRE Clock Signal (from OSC IP)
cre_rstn_i	IN	1	CRE Engine Reset Signal
LMMI Slave Interface[1]			
lmmi_clk_i	IN	1	Clock Signal of the LMMI Interface
lmmi_resetn_i	IN	1	LMMI Reset Signal. Active LOW, LMMI interface is in reset when asserted.
lmmi_request_i	IN	1	Active HIGH signal, indicates that the master wants to initiate a transaction when asserted.
lmmi_wr_rdn_i	IN	1	Active HIGH signal, indicates a write transaction when asserted.
lmmi_offset_i	IN	18	Offset address, the accessed location of the current active transaction.
lmmi_wdata_i	IN	32	Input data, the data to be written in the offset address. (This port is shared with the FIFO data input.)
lmmi_rdata_o	OUT	32	Output data, the data result from the previous transaction. (This port is shared with the FIFO data output.)
lmmi_rdata_valid_o	OUT	1	Active HIGH, indicates that the data is valid when asserted.
lmmi_ready_o	OUT	1	Active HIGH, indicates that the slave is ready to receive transactions when asserted.
FIFO Interface[2]			
async_fifo_clk_i	IN	1	Clock Signal of the FIFO Interface
async_fifo_rst_i	IN	1	FIFO Reset Signal, Active HIGH, indicates that the FIFO interface is in reset when asserted.
async_fifo_wr_en_i	IN	1	Active HIGH, indicates that an input data would be written to the FIFO if the FIFO is not full.
async_fifo_rd_en_i	IN	1	Active HIGH, indicates that an output data would be generated from the FIFO if the FIFO is not empty.
async_fifo_full_o	OUT	1	Active HIGH, indicates that the FIFO is full.
async_fifo_empty_o	OUT	1	Active HIGH, indicates that the FIFO is empty.
AHB-Lite Slave Interface[3]			
ahbl_hclk_i	IN	1	Clock Signal of the AHB-L Interface.
ahbl_hresetn_i	IN	1	AHB-L reset signal. Active LOW, AHB-L interface is in reset when asserted.
ahbl_hsel_i	IN	1	Active HIGH select signal. Allows the device to function when asserted.
ahbl_hready_i	IN	1	When HIGH, this indicates that there are currently no transfers being executed.
ahbl_haddr_i	IN	32	Contains the address of the data to be written/read.
ahbl_hburst_i	IN	3	Determines the type of burst used in the burst transfer.
ahbl_hsize_i	IN	3	Indicates the size of the transfer.
ahbl_hmastlock_i	IN	1	Indicates if the transfer is part of a locked sequence.
ahbl_hprot_i	IN	4	The protection control signals provide additional information about a bus access.
ahbl_htrans_i	IN	2	Determines if the transfer is a single transfer or forms a part of the burst.
ahbl_hwrite_i	IN	1	Indicates the transfer direction. (HIGH = write, LOW = read).
ahbl_hwdata_i	IN	32	Input data for the CRE IP

Pin Name	Direction	Width (Bits)	Description
ahbl_hreadyout_o	OUT	1	Indicates that the CRE IP is busy when pin is set LOW.
ahbl_hresp_o	OUT	1	Indicates that an error has occurred in the transfer when asserted to HIGH.
ahbl_hrdata_o	OUT	32	Output data for the CRE IP
APB Slave Interface[4]			
apb_pclk_i	IN	1	Clock Signal of the APB interface
apb_presetn_i	IN	1	APB reset signal. Active LOW, APB interface is in reset when asserted.
apb_psel_i	IN	1	Active HIGH select signal. Allows the device to function when asserted.
apb_paddr_i	IN	32	Contains the address of the data to be written/read.
apb_pwdata_i	IN	32	Input data for the CRE IP
apb_pwrite_i	IN	1	Indicates the transfer direction. (HIGH = write, LOW = read).
apb_penable_i	IN	1	Active HIGH, initiates a transfer request when asserted.
apb_pready_o	OUT	1	Active HIGH, indicates that the slave is ready to receive transactions when asserted.
apb_prdata_o	OUT	32	Output data for the CRE IP

Notes:

1. LMMI interface is available when INTERFACE = “lmmi” or INTERFACE = “generic”.
2. FIFO interface is available when INTERFACE = “generic”. The FIFO interface ports must by tied to 0, if unused or just set INTERFACE = “lmmi”.
3. AHB-Lite interface is available when INTERFACE = “AHB-Lite”.
4. APB interface is available when INTERFACE = “APB”.

2.5. Attribute Summary

Table 2.2. CRE Module Parameters

Parameter Name	Values	Default	Description
INTERFACE	“generic”, “lmmi”, “AHB-Lite”, “APB”	“generic”	Instantiates the IP with the selected bus interface: “generic”: LMMI + FIFO “lmmi”: LMMI only “AHB-Lite”: AHB-L bus “APB”: APB bus

2.6. Register Map

The CRE Module contains several registers which control the different security functions of the IP. Some registers change function depending on the current function that the IP is doing.

Table 2.3. CRE Instruction Register Map

Name	LMMI [17:0]	APB/AHB-L [31:0]	Size	R/W	Description
RI_CTRL1	0x2 000C	0x0000 080C	4B	W/O	Instruction register, writing to this register defines the current function of the CRE Engine and automatically starts the Engine: 0x00: Clears previous instruction 0x02: True Random Generation 0x05: Starts SHA256 0x06: Starts HMAC-SHA256 0x09: Starts AES Engine
RI_CTRL3	0x2 0014	0x0000 0814	4B	W/O	Sets the size of the message to be encrypted/decrypted (HMAC-SHA [1980B] max).
AES_SIZE	0x2 0018	0x0000 0818	4B	W/O	Sets the size of the key used in the encryption/decryption process (AES). 0x00: 128-bits (16B) 0x01: 256-bits (32B)
RO_GP0	0x2 0020	0x0000 0820	4B	R/O	Shows the status of the CRE Engine: 0x0B0: Engine is ready to accept instructions 0x0B1: Engine is busy 0x0B2: Engine has completed performing instructions
DPA_CON	0x2 0030	0x0000 0830	4B	W	Writes the information controlling the Differential Power Analysis features of the IP. Bit[0] controls “Clock Randomization” Bit[1] controls “Random Noise Addition”
DATA_SRC	0x2 003C	0x0000 083C	4B	W/O	Sets the data source for the SHA/AES Engine: 0x00: Sets the AES Engine data source to the bus 0x02: Sets the SHA Engine data source to the bus 0x03: Sets the SHA Engine data source to the FIFO 0x04: Sets the AES Engine data source to the FIFO
AES_CON	0x2 2040	0x0000 2840	4B	W/O	AES Control Register, sets the current function of the AES Engine to either encrypt or decrypt. 0x00: Encryption 0x01: Decryption
AES_STAT	0x2 2044	0x0000 2844	4B	R/O	Shows the status of the AES Engine: AES_STAT[0] = 0: AES is busy expanding the key AES_STAT[0] = 1: AES key expansion ready AES_STAT[1] = 0: AES is encrypting/decrypting AES_STAT[1] = 1: AES process finished
SHA_INIT	0x2 3070	0x0000 3870	4B	W/O	Initializes the SHA Engine, must be written with 0x01 followed by 0x00.

Table 2.4. CRE Data Register Map

LMMI [17:0]	APB/AHB-L [31:0]	Description
0x1 F800	0x0000 0000	**HMAC-SHA:** Generation: INPUT (before process): HMAC-key (32B)
0x1 F820	0x0000 0020	**HMAC-SHA:** Generation: OUTPUT (after process): SHA message (32B)
0x1 F840	0x0000 0040	**HMAC-SHA:** Generation: INPUT (before process): message (n bytes depending on length)
0x1 F880	0x0000 0080	**TRNG:** OUTPUT (after process): Q (32B), output of TRNG engine
0x2 2000	0x0000 2800	**AES:** Input (before process): AES Key (32B) [Big Endian]
0x2 2020	0x0000 2820	**AES:** Encryption Mode: • Input (before process): Private Message (16B) [Big Endian]. Total Message Size can be larger than 16B, but must be processed at 16B at a time. Decryption Mode: • Output (after process): Decrypted Message (16B) [Big Endian]. Total Message Size can be larger than 16B, but must be processed at 16B at a time.
0x2 2030	0x0000 2830	**AES:** Encryption Mode: • Output (after process): Encrypted Message (16B) [Big Endian]. Total Message Size can be larger than 16B, but must be processed at 16B at a time. Decryption Mode: • Input (before process): Encrypted Message (16B) [Big Endian]. Total Message Size can be larger than 16B, but must be processed at 16B at a time.
0x2 304C	0x0000 384C	**SHA:** Input (before process): SHA data size (n bytes depending on message length) [Big Endian] Address does not increment, and data must be written on the same address for every word.
0x2 3050	0x0000 3850	**SHA:** Output (after process): SHA message (32B)

2.7. CRE Function Access

Note: When the transaction exceeds 4B of data, it is assumed that you automatically write or read at the address position of the succeeding addresses in an incremental fashion to complete the entire data set, unless otherwise indicated. For example, if 16B of data would need to be written at address 0x0000 0000, you can write four 4B data on addresses 0x0000 00000, 0x0000 0004, 0x0000 0008, and 0x0000 000C in four different transactions. This applies to all bus interfaces: LMMI, AHB-L, and APB. All data are LITTLE ENDIAN unless otherwise specified.

Table 2.5. SHA Message Generation Procedure (DATA_SOURCE = bus)

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 083C	0x2 003C	4B	[*DATA_SRC ← 0x02*] Sets the SHA data source from the LMMI/AHB-L/APB bus.
Write	0x0000 3870	0x2 3070	4B	[*SHA_INIT ← 0x01*] Initializes SHA (1st step).
Write	0x0000 3870	0x2 3070	4B	[*SHA_INIT ← 0x00*] Initializes SHA (2nd step).
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x05*] Starts the SHA Engine.
Write	0x0000 384C	0x2 304C	1B	Message to be hashed. Written 1B at a time on the same address on every write iteration, no increment required. **[BIG ENDIAN]** (The upper 3B must be padded by 0, to fill the 32-bit word. The MSB of the last word written must be 1. That is, if the message is 0x12345678, it must be written as 0x0000 0012, 0x0000 0034, 0x0000 0045 and 0x8000 0078).
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Read	0x0000 3850	0x2 3050	32B	SHA Message
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Table 2.6. SHA Message Generation Procedure (DATA_SOURCE = FIFO)

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 083C	0x2 003C	4B	[*DATA_SRC ← 0x03*] Sets the SHA data source from the LMMI/AHB-L/APB bus.
Write	0x0000 3870	0x2 3070	4B	[*SHA_INIT ← 0x01*] Initializes SHA (1st step).
Write	0x0000 3870	0x2 3070	4B	[*SHA_INIT ← 0x00*] Initializes SHA (2nd step).
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x05*] Starts the SHA Engine.
Write	N/A	lmmi_wrdata port	N bytes	Message to be hashed. Written 1B at a time if the signal async_full_o is LOW. Hold the data when the async_full_o is HIGH, this gives time for the buffer to clear. Write the entire message first before reading hash output. **[BIG ENDIAN]** (The upper 3B must be padded by 0, to fill the 32-bit word. The MSB of the last word written must be 1. For example, if the message is 0x12345678, it must be written as 0x0000 0012, 0x0000 0034, 0x0000 0045 and 0x8000 0078).
Read	N/A	lmmi_rddata port	32B	SHA Message, start reading when the signal async_empty_o is LOW. Continue reading until the async_empty_o signal is HIGH.[1]
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Note:

1. The FIFO interface is designed specifically for high throughput applications. You must keep rd_en_i until async_empty_o signal transitions from LOW to HIGH.

Table 2.7. HMAC-SHA Message Generation Procedure

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 0814	0x2 0014	4B	[*RI_CTRL3 ← <message size in bytes>*] Size of message to be hashed
Write	0x0000 083C	0x2 003C	4B	[*DATA_SRC ← 0x02*] Sets the SHA data source from the LMMI/AHB-L/APB bus.
Write	0x0000 0000	0x1 F800	32B	HMAC Key
Write	0x0000 0040	0x1 F840	N bytes	Message to be hashed. If N is not divisible by 4B, pad the missing bytes on the last transaction with 0.
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x06*] Starts the HMAC-SHA Engine.
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Read	0x0000 0020	0x1 F820	32B	SHA Message.
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Table 2.8. AES Encryption Procedure (DATA_SOURCE = bus)

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 083C	0x2 003C	4B	[*DATA_SRC ← 0x00*] Sets the AES data source from the LMMI/ AHB-L/APB bus.
Write	0x0000 2840	0x2 2040	4B	[*AES_CON ← 0x00*] Sets the AES mode to encryption.
Write	0x0000 0818	0x2 0018	4B	[*AES_SIZE ← <size>*] Sets the AES key size for encryption. **0x00**: 128-bits (16B) **0x01**: 256-bits (32B)
Write	0x0000 2800 0x0000 2810[1]	0x2 2000 0x2 2010[1]	32B/16B[1]	AES Key **[BIG ENDIAN]**
Read	0x0000 2844	0x2 2044	4B	[*AES_STAT[0] == 1*] Waits for the AES to finish key expansion. AES_STAT[0] = 0: Busy AES_STAT[0] = 1: Ready
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x09*] Starts the AES Engine.
Write	0x0000 2820	0x2 2020	16B	AES message to be encrypted. For larger data sizes: you must write 16B → wait for encryption to finish → read the output data (see succeeding procedures) before continuing to the next set of data. **[BIG ENDIAN]**
Read	0x0000 2844	0x2 2044	4B	[*AES_STAT[1] == 1*] Waits for the AES to finish encryption. AES_STAT[1] = 0: Busy AES_STAT[1] = 1: Ready
Read	0x0000 2830	0x2 2030	16B	Encrypted Message. For larger data sizes: you must write 16B → wait for encryption to finish → read the output data (see preceding procedures) before continuing to the next set of data. **[BIG ENDIAN]**
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Note:

1. if AES_SIZE is for 128-bits.

Table 2.9. AES Decryption Procedure (DATA_SOURCE = bus)

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 083C	0x2 003C	4B	[*DATA_SRC ← 0x00*] Sets the AES data source from the LMMI/AHB-L/APB bus.
Write	0x0000 2840	0x2 2040	4B	[*AES_CON ← 0x01*] Sets the AES mode to decryption.
Write	0x0000 0818	0x2 0018	4B	[*AES_SIZE ← <size>*] Sets the AES key size for decryption. 0x00: 128-bits (16B) 0x01: 256-bits (32B)
Write	0x0000 2800 0x0000 2810[1]	0x2 2000 0x2 2010[1]	32B/16B[1]	AES Key **[BIG ENDIAN]**
Read	0x0000 2844	0x2 2044	4B	[*AES_STAT[0] == 1*] Waits for the AES to finish key expansion. AES_STAT[0] = 0: Busy AES_STAT[0] = 1: Ready
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x09*] Starts the AES Engine.
Write	0x0000 2820	0x2 2020	16B	AES message to be decrypted. For larger data sizes: you must write 16B → wait for decryption to finish → read the output data (see succeeding procedures) before continuing to the next set of data. [**BIG ENDIAN**]
Read	0x0000 2844	0x2 2044	4B	[*AES_STAT[1] == 1*] Waits for the AES to finish decryption. AES_STAT[1] = 0: Busy AES_STAT[1] = 1: Ready
Read	0x0000 2830	0x2 2030	16B	Decrypted Message. For larger data sizes: you must write 16B → wait for decryption to finish → read the output data (see preceding procedures) before continuing to the next set of data. [**BIG ENDIAN**]
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Note:

1. if AES_SIZE is for 128-bits.

Table 2.10. AES Encryption Procedure (DATA_SOURCE = FIFO)

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 083C	0x2 003C	4B	[*DATA_SRC ← 0x04*] Sets the AES data source from the FIFO interface.
Write	0x0000 2840	0x2 2040	4B	[*AES_CON ← 0x00*] Sets the AES mode to encryption.
Write	0x0000 0818	0x2 0018	4B	[*AES_SIZE ← <size>*] Sets the AES key size for encryption. **0x00**: 128-bits (16B) **0x01**: 256-bits (32B)
Write	0x0000 2800 0x0000 2810[1]	0x2 2000 0x2 2010[1]	32B/16B[1]	AES Key **[BIG ENDIAN]**
Read	0x0000 2844	0x2 2044	4B	[*AES_STAT[0] == 1*] Waits for the AES to finish key expansion. AES_STAT[0] = 0: Busy AES_STAT[0] = 1: Ready
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x09*] Starts the AES Engine.
Write	N/A	lmmi_wrdata port	32B	Message to be encrypted. Written 4B at a time as long as the signal async_full_o is LOW. Hold the data when the async_full_o is HIGH, this gives time for the buffer to clear. Write the entire message first before waiting for the async_empty_o to go LOW. [**BIG ENDIAN**]
Read	N/A	lmmi_rddata port	32B	Encrypted message, start reading when the signal async_empty_o is LOW. When the async_empty_o goes HIGH, while there is remaining data to be read; wait for the signal to go LOW again before reading.[2] This gives the AES engine enough time to process the input data.[**BIG ENDIAN**]
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Notes:

1. If AES_SIZE is for 128-bits.
2. The FIFO interface is designed specifically for high throughput applications. You must keep rd_en_i until async_empty_o signal transitions from LOW to HIGH.

Table 2.11. AES Decryption Procedure (DATA_SOURCE = FIFO)

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 083C	0x2 003C	4B	[*DATA_SRC ← 0x04*] Sets the AES data source from the FIFO interface.
Write	0x0000 2840	0x2 2040	4B	[*AES_CON ← 0x01*] Sets the AES mode to decryption.
Write	0x0000 0818	0x2 0018	4B	[AES_SIZE ← <size>] Sets the AES key size for decryption. 0x00: 128-bits (16B) 0x01: 256-bits (32B)
Write	0x0000 2800 0x0000 2810[1]	0x2 2000 0x2 2010[1]	32B/16B[1]	AES Key **[BIG ENDIAN]**
Read	0x0000 2844	0x2 2044	4B	[AES_STAT[0] == 1] Waits for the AES to finish key expansion. AES_STAT[0] = 0: Busy AES_STAT[0] = 1: Ready
Write	0x0000 080C	0x2 000C	4B	[RI_CTRL1 ← 0x09] Starts the AES Engine.
Write	N/A	lmmi_wrdata port	16B	Message to be decrypted. Written 4B at a time if the signal async_full_o is LOW. Hold the data when the async_full_o is HIGH, this gives time for the buffer to clear. Write the entire message first before waiting for the async_empty_o to go LOW. **[BIG ENDIAN]**
Read	N/A	lmmi_rddata port	16B	Decrypted message, start reading when the signal async_empty_o is LOW. When the async_empty_o goes HIGH, while there is remaining data to be read; wait for the signal to go LOW again before reading.[2] This gives the AES engine enough time to process the input data. **[BIG ENDIAN]**
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Notes:

1. If AES_SIZE is for 128-bits
2. The FIFO interface is designed specifically for high throughput applications. You must keep the rd_en_i until async_empty_o signal transitions from LOW to HIGH.

Table 2.12. AES-Key Size Change (128-bit to 256-bit)

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 081C	0x2 001C	4B	[*RI_CTRL1 ← AES*] Write AES old 128-bit key LSB [31:0].
Read	0x0000 2844	0x2 2044	4B	[1][*AES_STAT[0] == 1*] Waits for the AES to finish key expansion. AES_STAT[0] = 0: Busy AES_STAT[0] = 1: Ready
Write	0x0000 0818	0x2 0018	4B	[1][*AES_SIZE ← <size>*] Sets the AES key size to 0x01 for 256-bit selection.

Note:

1. Follow the instructions depending on the next AES transaction. Refer to Table 2.8, Table 2.9, Table 2.10, and Table 2.11 for the available protocols related to AES.

Table 2.13. True Random Number Generator Procedure

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x02*] Starts the True Random Number Generation Process.
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Read	0x0000 0080	0x1 F880	32B	Random Generated Number output.
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

Table 2.14. DPA Feature Access

Transaction	AHB-L/APB	LMMI	Data	Description
Read	0x0000 0820	0x2 0020	4B	Poll if IP is Ready. [*RO_GP0 == 0xB0*]
Write	0x0000 0830	0x2 0030	4B	[*DPA_CON ← <DPA setting>*] Bit [0] = Clock Randomization Bit [1] = Random Noise Addition
The steps below are for DPA_CON[0] = 1				
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x03*] API generates a random seed using TRNG, to be loaded as a clock enable signal.
Read	0x0000 0820	0x2 0020	4B	Poll if transaction is done. [*RO_GP0 == 0xB2*]
Write	0x0000 080C	0x2 000C	4B	[*RI_CTRL1 ← 0x00*] Clears the previous transaction and sets the IP ready for the next.

2.8. CRE Timing Diagrams

The CRE timing diagrams are compliant with the supported bus interfaces: LMMI, AHB-L and APB. The IP is designed to support the protocols for this bus interface standard. A special case would be using the generic LMMI interface where the IP configuration is fed through the LMMI bus, and data is processed using the FIFO interface.

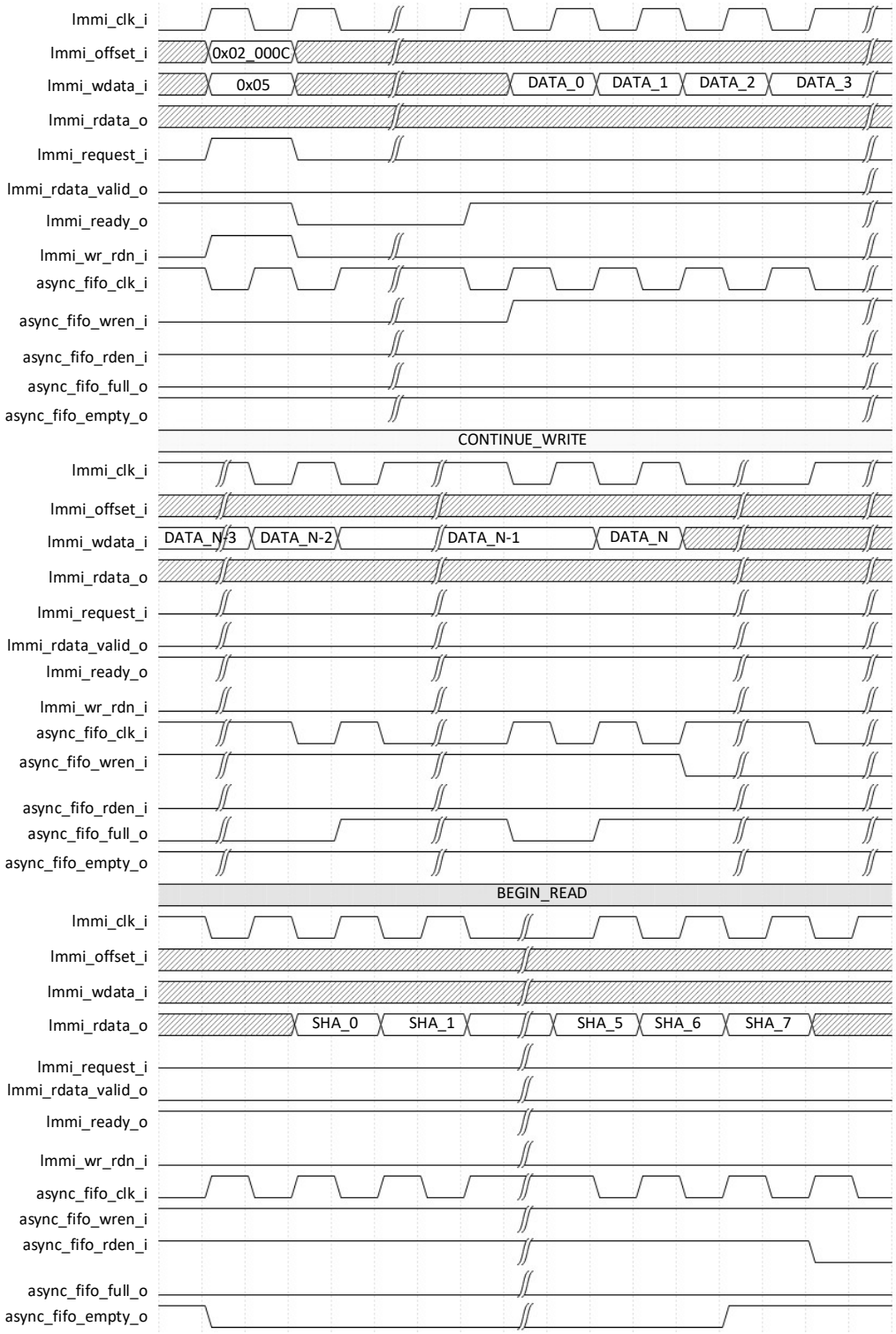

Figure 2.2. SHA-256 Hash Done on Size N-Bytes Message

2.9. Interface Limitations

Table 2.15. CRE Interface Limitations

Signal/Command	Interface	Description
RESET	“generic”, “lmmi”, “AHB-Lite”, “APB”	The IP requires 500 clock cycles minimum to initialize the Security Engine Core. This means that you must wait 500 clock cycles before initiating any bus access request. The 500-cycle measurement is taken from the clock fed to cre_clk_i port.
HSIZE	“AHB-Lite”	The AHB-Lite interface cannot support byte/half-word transactions, and any AHB-Lite access must be limited with HSIZE[2:0] = 010.
Write on Read-only register	“lmmi”, “AHB-Lite”, “APB”	Any write on a read-only register would cause an overwrite, if done on a register which contains process result. This command would overwrite the result.
Write on Read register	“lmmi”, “AHB-Lite”, “APB”	Any read on a write register would return garbage data. The IP would still return an output as compliance to bus specifications.

3. Getting Started

The Module/IP Block Wizard in Lattice Radiant Software allows you to generate, create, or open modules for the target device.

To generate the module:

1. From the Lattice Radiant Software, select the **IP Catalog** tab, and select **IP on Local**.

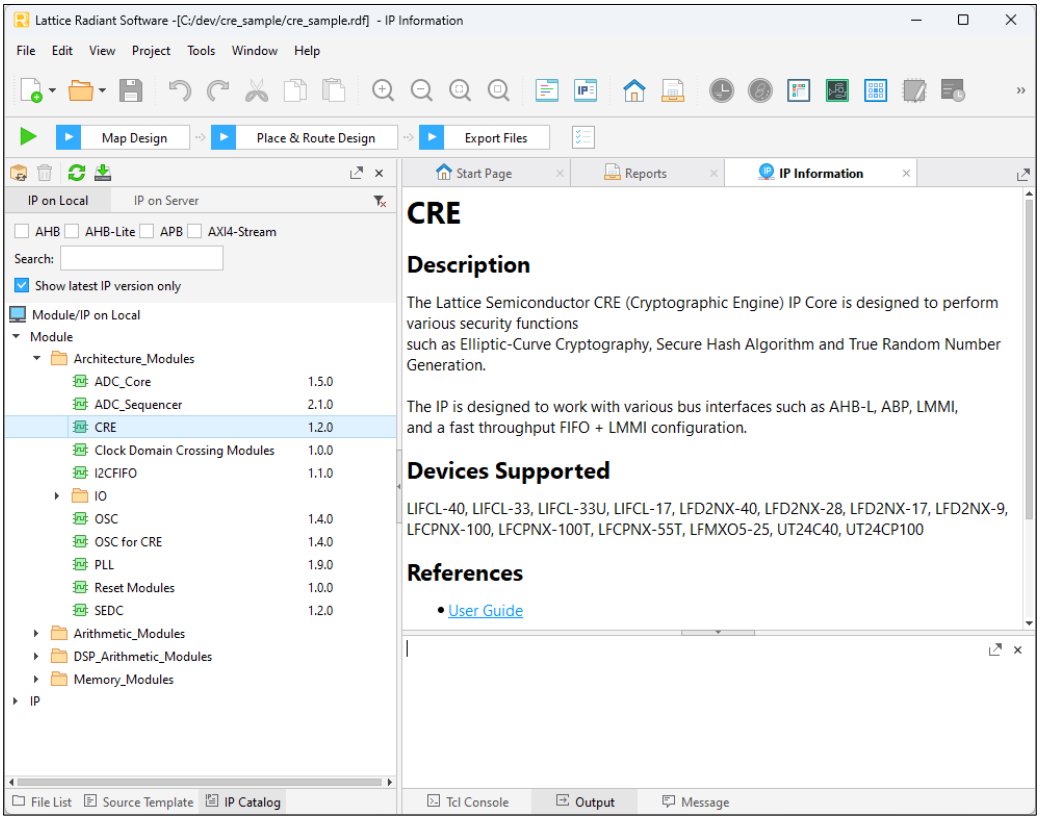

Figure 3.1. IP on Local, with CRE Selected Under Architecture

2. Double-click on **CRE IP** to open the **Module/IP Block Wizard**. Provide a file name and click **Next**.

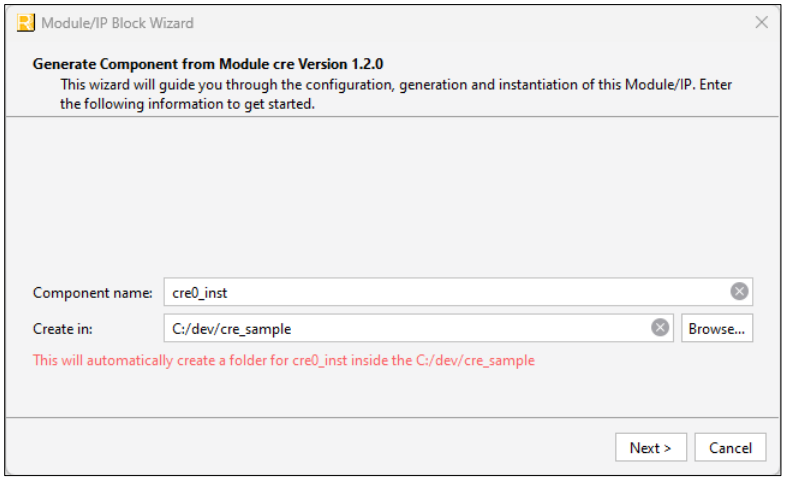

Figure 3.2. Module/IP Block Wizard

3. On the next screen, the Configuration mode for the module is displayed. The available user parameters are shown. Use the default settings for now. Click **Generate**. After generation, the IP is automatically added to the active project.

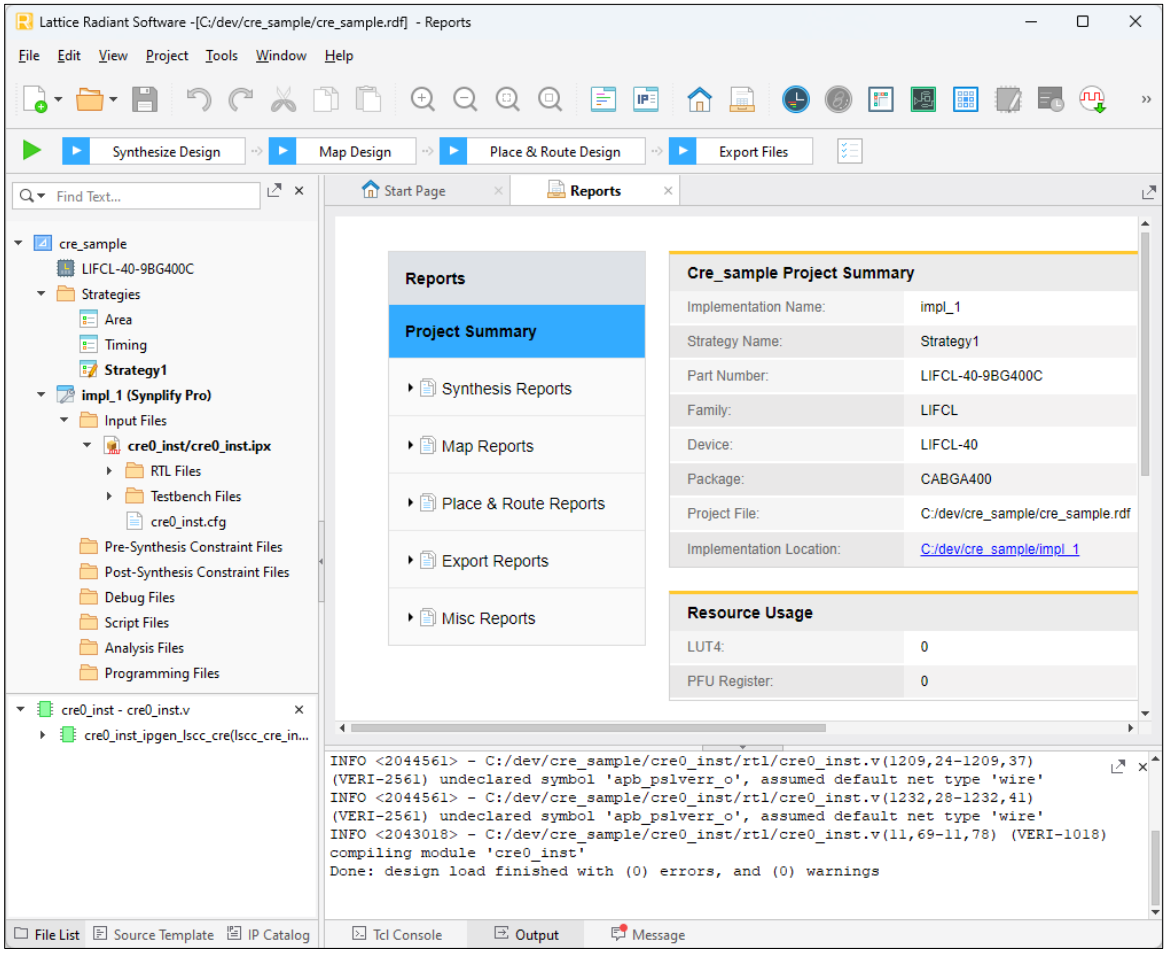

Figure 3.3. IP Structure

3.1. Generated Files and Top-Level Directory Structure

Table 3.1 shows the list of generated files and provides a short description of each file.

Table 3.1. CRE Generated File Description

File	Sim	Synthesis	Description
rtl/<IP_name>.v	Yes	Yes	Top Level RTL file with the selected configuration. The main IP file
testbench/dut_params	Yes	—	Top level parameters of the generated RTL file
testbench/dut_inst	Yes	—	Instantiated version of the <IP_name>.v file for simulation use
testbench/tb_top.v	Yes	—	Test bench template, you can modify this to match the specific needs. The tb_top contains a master module and a checker module, which is automatically included when the IP is simulated.
<IP_name>.cfg	—	—	This file contains the configuration options used to recreate or modify the core IP in the IP Platform.

File	Sim	Synthesis	Description
<IP_name>.ipx	—	—	The IPX file holds references to all the elements of an IP or Module after it is generated from the IP Platform user interface. This file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP Platform when the IP/Module is being re-generated.

3.2. Instantiating the Module

To instantiate the module, type in the module name and then the instance name, following the Verilog format. Alternatively, a sample instance can be seen in the testbench/dut_inst folder.

3.3. Running Functional Simulation

IP module generation includes a Verilog-based testbench to check for integrity of the generated IP. Testbench consumes the parameters extracted from user configuration to match the IP. Below is a high-level block diagram showing the testbench.

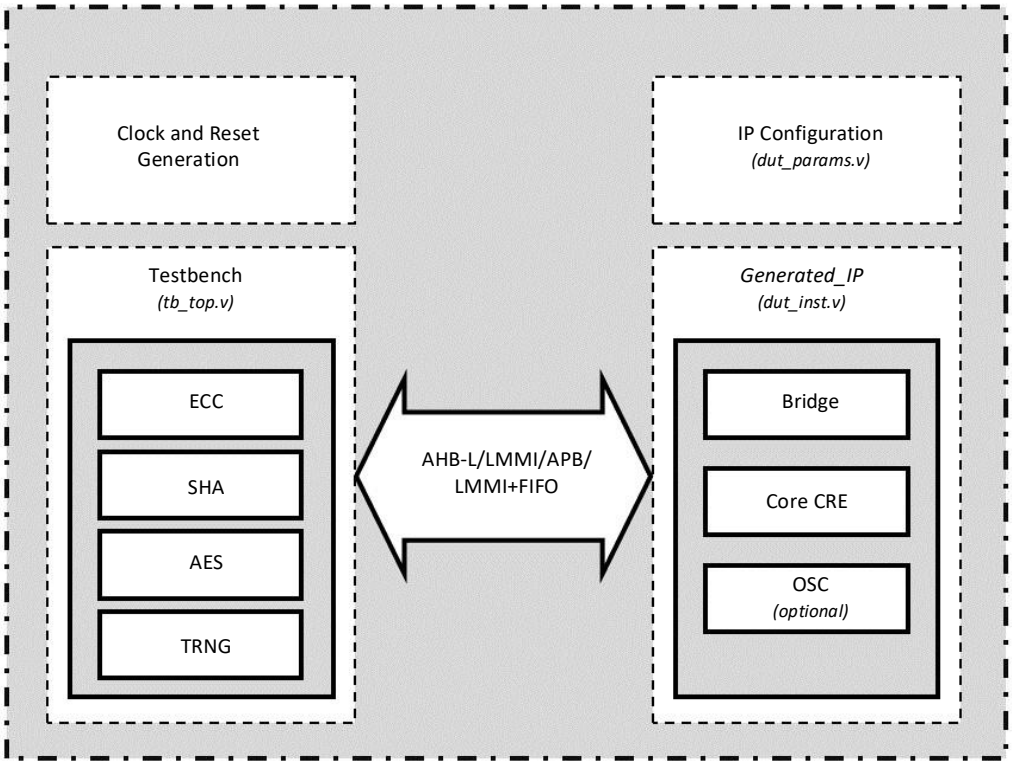

Figure 3.4. IP Testbench Block Diagram

To run the testbench:

1. Once the simulation file is added, simulate the project. Click **Tools > Simulation Wizard** to open the Lattice Radiant Software Simulation Wizard.
2. The splash window opens. Configure the testbench and click **Next**.

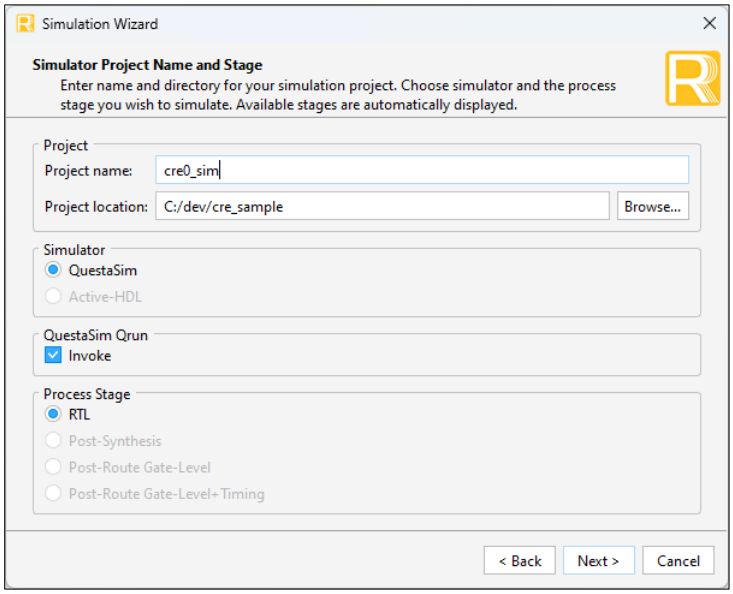

Figure 3.5. Simulation Wizard

3. For this example, the project is named *cre0_sim*, using the QuestaSim simulator and RTL simulation only. Click **Next**.
4. The list of files to be added appears. Click **Next**.

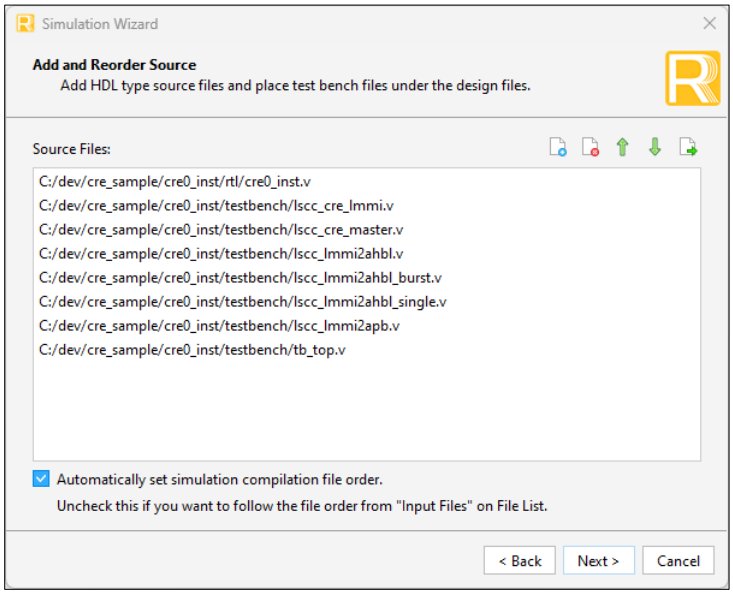

Figure 3.6. File List

5. Simulation Wizard does not parse the entire set of included RTL. In this section, make sure that the Simulation Top Model is tb_top. Click **Next**.

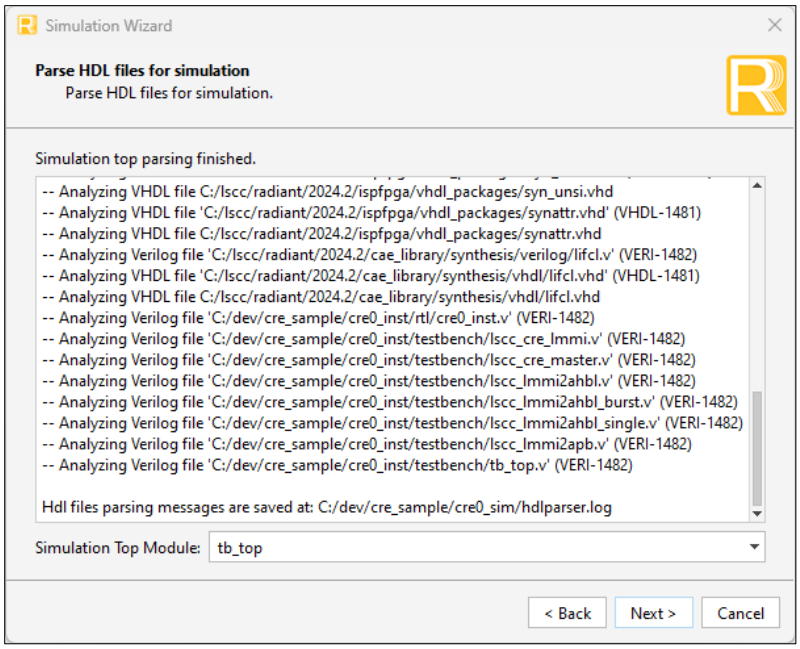

Figure 3.7. Parse HDL Files

6. Verify that the options **Run simulator**, **Add top-level signals to waveform display**, and **Run simulation** are checked.

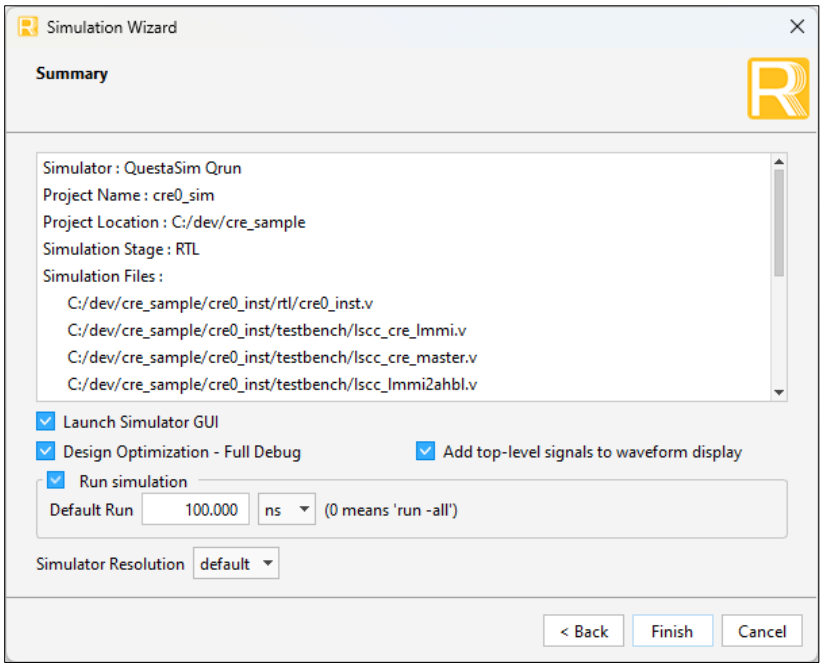

Figure 3.8. Simulation Wizard Summary

7. Click **Finish** to automatically open Active-HDL and run the first 1 µs of the simulation.

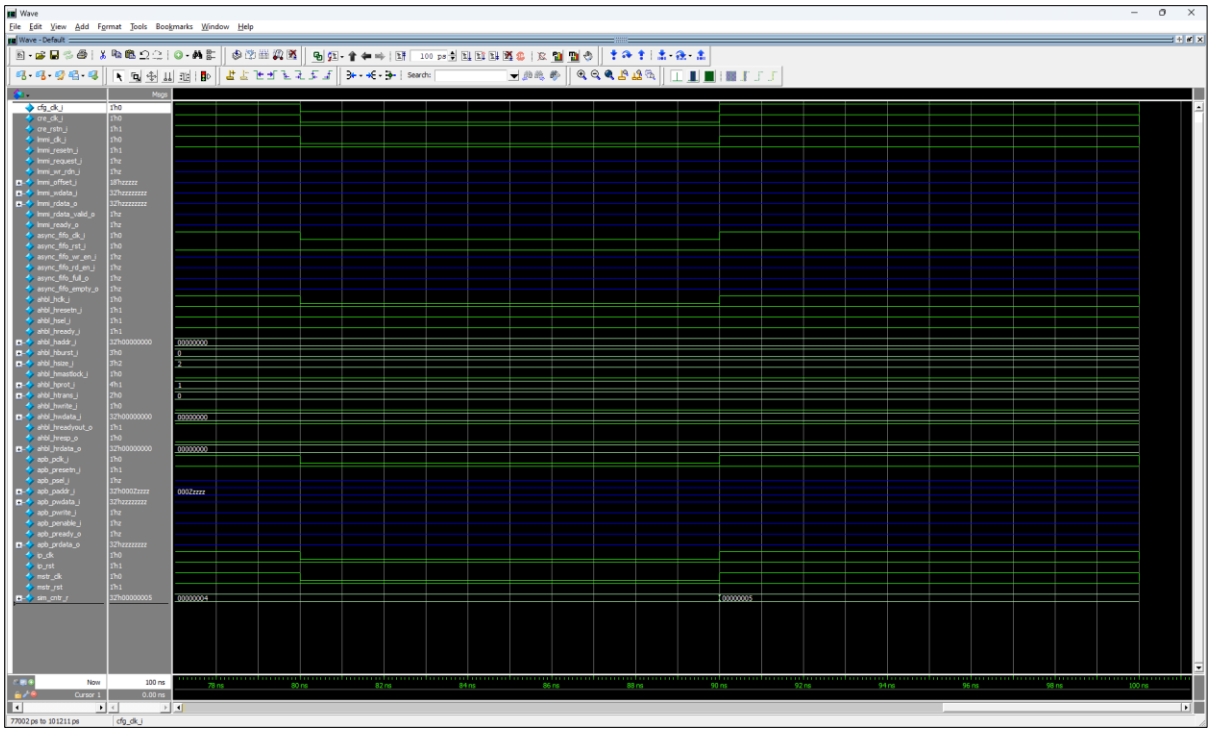

Figure 3.9. QuestaSim Simulation Start

8. Click **Run -All** to continue the simulation and view the rest of the waveform.

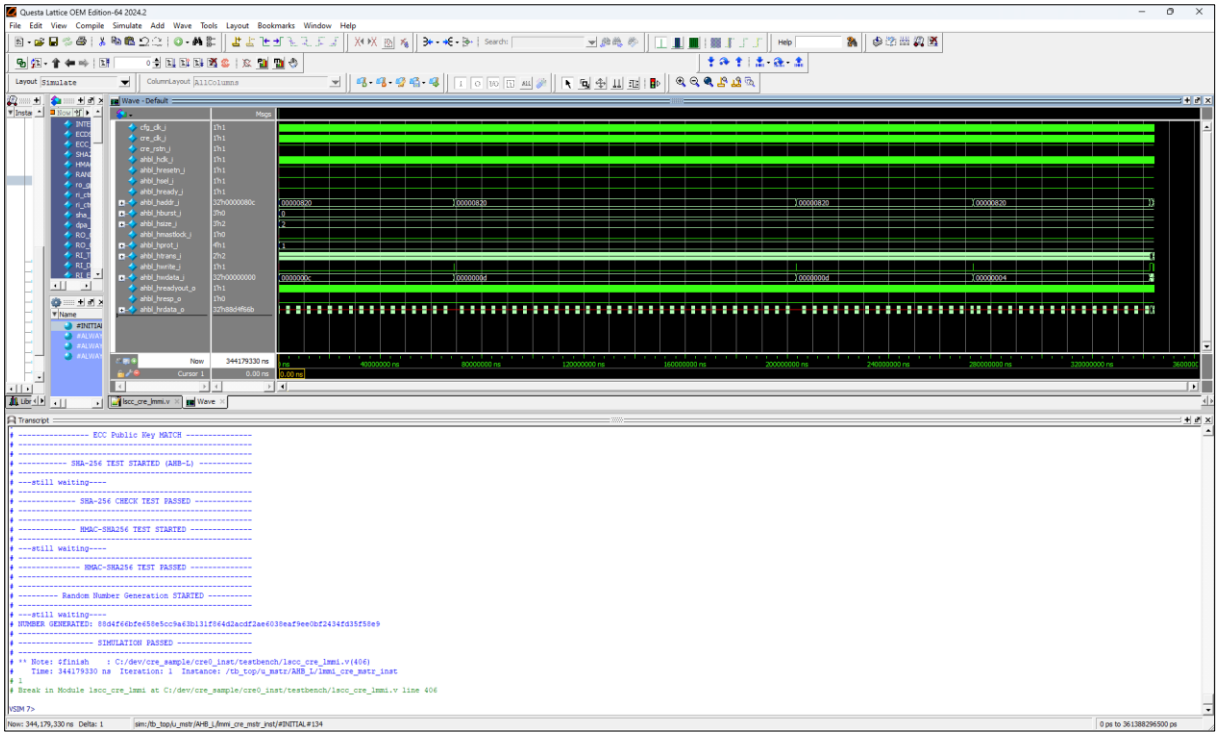

Figure 3.10. QuestaSim Simulation Finished

9. Once the simulation is completed, a list of all tests ran with their results, either data, pass, or fail are viewable. The speed of the test is dependent on the configuration of the clocks and whether the USE_OSC parameter is enabled or disabled.

Appendix A. Resource Utilization

Table A.1 shows configuration and resource utilization for LIFCL-40-9BG400I using Lattice LSE of Lattice Radiant Software 3.1.

Table A.1. Resource Utilization[1] (LIFCL)

Interface	Registers	LUTs[2]	EBR	Cryptographic Block	Maximum Frequency
LMMI	0	1	0	1	200 MHz
Generic (LMMI + FIFO)	0	1	0	1	200 MHz
AHB-L	200	341	0	1	200 MHz
APB	3	19	0	1	200 MHz

Notes:

1. In all configurations, the oscillator clock and reset pins are connected externally to the CRE module.
2. The distributed RAM utilization is accounted for in the total LUT4s utilization. The actual LUT4 utilization is distribution among logic, distributed RAM, and ripple logic.

Table A.2 shows configuration and resource utilization for LFD2NX-40-9BG256I using Lattice LSE of Lattice Radiant Software 3.1.

Table A.2. Resource Utilization[1] (LFD2NX)

Interface	Registers	LUTs[2]	EBR	Cryptographic Block	Maximum Frequency
LMMI	0	1	0	1	200 MHz
Generic (LMMI + FIFO)	0	1	0	1	200 MHz
AHB-L	200	342	0	1	200 MHz
APB	3	19	0	1	200 MHz

Notes:

1. In all configurations, the oscillator clock and reset pins are connected externally to the CRE module.
2. The distributed RAM utilization is accounted for in the total LUT4s utilization. The actual LUT4 utilization is distribution among logic, distributed RAM, and ripple logic.

Table A.3 shows configuration and resource utilization for LFCPNX-100-7LFG672I using Lattice LSE of Lattice Radiant Software 2022.1.

Table A.3. Resource Utilization[1] (LFCPNX)

Interface	Registers	LUTs[2]	EBR	Cryptographic Block	Maximum Frequency
LMMI	0	1	0	1	200 MHz
Generic (LMMI + FIFO)	0	1	0	1	200 MHz
AHB-L	200	261	0	1	200 MHz
APB	3	14	0	1	200 MHz

Notes:

1. In all configurations, the oscillator clock and reset pins are connected externally to the CRE module.
2. The distributed RAM utilization is accounted for in the total LUT4s utilization. The actual LUT4 utilization is distribution among logic, distributed RAM, and ripple logic.

References

- AMBA 3 AHB-Lite Protocol Specification
- Certus-NX Family Devices Web Page
- CertusPro-NX Family Devices Web Page
- CrossLink-NX Family Devices Web Page
- MachXO5-NX Family Devices Web Page
- Lattice Insights for Lattice Semiconductor training courses and learning plans
- Lattice Radiant FPGA design software

Technical Support Assistance

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

For frequently asked questions, refer to the Lattice Answer Database at http://www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.5, IP v1.2.0, June 2025

Section	Change Summary
All	• Added IP version to the cover page and revision history. • Updated document title from CRE Module IP to *Nexus CRE Module IP*, including in the filename.
Introduction	• Added Overview of the IP, Quick Facts, and Licensing and Control Pack Information sections. • Moved Security Hard IP content in the Features section and added FIFO and LMMI interfaces under the *High throughput Advance Encryption Standard* bullet point.
Getting Started	Updated section content, including figures, to be aligned with the current Radiant software.
References	Updated Radiant and Lattice Insights bullet items.

Revision 1.4, August 2023

Section	Change Summary

Functional Description	• In Table 2.1 CRE Module Ports:
	  • removed duplicated FIFO Interface ports: async_fifo_clk_i, async_fifo_rst_i, async_fifo_wr_en_i, async_fifo_rd_en_i, async_fifo_full_o, and async_fifo_empty_o;
	  • changed *HSE IP* to *CRE IP* in the Description of apb_pwdata_i and apb_prdata_o.
	• In the Register Map section, changed *HSE Module contains several registers which control the different security functions of the IP* to *CRE Module contains several registers which control the different security functions of the IP*.
	• In Table 2.7. HMAC-SHA Message Generation Procedure:
	  • changed the address *0x2 2014* to *0x2 0014* in the LMMI column;
	  • changed the address *0x0000 2814* to *0x0000 0814* in the AHB-L/APB column.
	• In Table 2.8. AES Encryption Procedure (DATA_SOURCE = bus):
	  • changed *Waits for the AES to finish decryption* to *Waits for the AES to finish encryption* in the Description for the Read transaction, [AES_STAT[1] == 1];
	  • changed *AES Key* to *AES Key [BIG ENDIAN]* in the Description for the Write transaction on 0x0000 2800, 0x0000 2810.
	• In Table 2.9. AES Decryption Procedure (DATA_SOURCE = bus):
	  • changed *wait for encryption to finish* to *wait for decryption to finish* in the Description for the Write transaction on 0x0000 2820;
	  • changed *wait for encryption to finish* to *wait for decryption to finish* in the Description for the Read transaction on 0x0000 2830;
	  • changed *AES Key* to *AES Key [BIG ENDIAN]* in the Description for the Write transaction on 0x0000 2800, 0x0000 2810.
	• In Table 2.10. AES Encryption Procedure (DATA_SOURCE = FIFO), changed *AES Key* to *AES Key [BIG ENDIAN]* in the Description for the Write transaction on 0x0000 2800, 0x0000 2810.
	• In Table 2.11. AES Decryption Procedure (DATA_SOURCE = FIFO), changed *AES Key* to *AES Key [BIG ENDIAN]* in the Description for the Write transaction on 0x0000 2800, 0x0000 2810.

References	Newly added the links to Certus-NX Family Devices Web Page, CertusPro-NX Family Devices Web Page, CrossLink-NX Family Devices Web Page, MachXO5-NX Family Devices Web Page, and Lattice Insights for Training Series and Learning Plans.

Revision 1.3, December 2022

Section	Change Summary
Appendix A. Resource Utilization	Added Table A.3. Resource Utilization1 (LFCPNX).
Technical Support Assistance	Added Lattice FAQ website link.

Revision 1.2, November 2021

Section	Change Summary
Functional Overview	Added footnote in Table 2.12. AES-Key Size Change (128-bit to 256-bit).
Appendix A. Resource Utilization	General update to this section.

Revision 1.1, June 2021

Section	Change Summary
—	Changed document status from Preliminary to final.
Introduction	• Replaced specific product names with Lattice FPGA devices built on the Lattice Nexus platform. • Revised statement to *This module is based on the built-in Security Hard IP having the following features*.

Revision 1.0, August 2020

Section	Change Summary
All	Preliminary release

www.latticesemi.com