Communication Protocols and EVE API Guide for CrossLinkU-NX SoM

User Guide

FPGA-UG-02250-1.0

January 2026

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.

Inclusive Language

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Contents

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Tables

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
ACK	Acknowledgement
AI	Artificial Intelligence
API	Application Programming Interface
BGR	Blue Green Red
BGRA	Blue Green Red Alpha
CPU	Central Processing Unit
EVE	Edge Vision Engine
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
HMI	Human Machine Interface
I2C	Inter-Integrated Circuit
ID	Identification
IPS	Inference Per Second
IR	Infrared
IRQ	Interrupt Request
MJPG	Motion JPEG
NV12	YUV 4:2:0 Semi-planar Format
OS	Operating System
PID	Product ID
RGB	Red Green Blue
RISC-V	Reduced Instruction Set Computer Five
ROI	Region of Interest
RPi	Raspberry Pi
SDK	Software Development Kit
SoM	System on Module
VID	Vendor ID
YUY2	YUV 4:2:2 Packed Format

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

1. Introduction

The CrossLinkU™-NX system-on-module (SoM) for human-machine interface (HMI) reference design offers a purpose-built solution for human-machine interactions. The design enables AI-driven interaction capabilities, including person detection and face recognition, for an intuitive and intelligent user experience. Optimized for edge device processing, the solution supports ultra-low power mode while adhering to industry standards for embedded vision and artificial intelligence (AI) systems. For more information, refer to the CrossLinkU-NX SoM for HMI Demonstration Reference Design (FPGA-RD-02333) available within the Lattice CrossLinkU-NX SoM for HMI Demonstration Reference Design GitHub repository.

This user guide explains how to access data from the CrossLinkU-NX device on the SoM for integrating HMI-based interactions into custom applications. Several methods are available, allowing you to choose based on your objectives and requirements:

- Section 2: FPGA Communication Protocol describes the inter-integrated circuit (I2C) field programmable gate array (FPGA) communication protocol, providing detailed information about the I2C connection and data format for bare-metal implementations.
- Section 3: Communication Protocol Library covers the communication protocol library, explaining how to use helper functions in the *lib_i2clib.so* library to read HMI metadata over I2C.
- Section 4: Edge Vision Engine SDK API introduces the Edge Vision Engine (EVE) software development kit (SDK) application programming interface (API), which offers a user-friendly, high-level interface to connect with the FPGA and process metadata from your application.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2. FPGA Communication Protocol

This section provides details on communication between an FPGA device and a Raspberry Pi (RPi) host. The information provided here can aid in the development of custom solutions, such as using a different operating system (OS), or optimizing for specific use cases.

2.1. Communication with the Device

Communication with the FPGA device is performed through the I2C bus. For more implementation details, see the sensAIReferenceDesign/demo/clnx directory. The library containing the source codes to implement this protocol is in the RESOURCES folder.

2.1.1. I2C Pages and Registers

The device is located on *i2c adapter 0*, accessible at */dev/i2c-0*, with device address 0x30 (decimal 48). The I2C address space has eight pages numbered from 0 to 7. Each page contains 32-bit control registers and data registers. Control registers are the registers from addresses 0x0 to 0xF on every page. To select the active page, write the desired page number into the page control register at address 0x4.

The registers from addresses 0x10 to 0x1F on pages 2, 3, 4, and 5 are data registers. The data registers on pages 2 and 3 are reserved for sending data from the host to the device. The data registers on pages 4 and 5 are reserved for receiving data sent from the device to the host.

For an overview of how to read from and write to I2C devices, refer to I2C/SMBus Subsystem (Linux Kernel documentation).

2.1.2. Receiving Data from the Device

After the device writes data into the data registers, it notifies the host by triggering an interrupt request (IRQ) on general purpose I/O (GPIO) pin 26 and writing to the device-to-host interrupt control register 0x3. Writing 1 to the first bit of this register indicates an active interrupt, while writing 0 indicates no interrupt. Bits [1:6] indicate the type of interrupt as ISH_INT_DATA_TRANSMISSION, which is equivalent to the value of 3. Refer to the i2c_adapter.cc file for details. Once the proper interrupt is triggered, the host proceeds to copy the data from the data registers.

To send data that fits into two pages of 16 registers each (total of 128 bytes), a simple packet protocol is used. Four bytes are reserved for a header, leaving 124 bytes of data per packet. Table 2.1 defines the data packet header. The header is stored in data register 15 of page 5. The remaining data registers are used for the packet data.

Table 2.1. Data Packet Header Definition

Byte	Type	Description
0	uint8_t	Packet identifier Indicates the number of packets for a transmission. Packets are assigned identifiers based on the number of packets in the transmission. 0 is reserved for empty packet. For example, a transmission of 300 bytes requires 3 packets. The initial packet will have the identifier of 3, while the next two packets will have the identifiers of 2 and 1, respectively.
1	uint8_t	Packet data size Indicates the number of bytes being transmitted by the current packet. The maximum packet data size is 124 bytes.
2, 3	uint16_t	Fletcher's checksum[1] Value calculated with the packet data bytes and used to validate packet data correctness.

Note:

1. Refer to the i2c_adapter.cc file for an implementation of the Fletcher's checksum algorithm.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.1.3. Sending Data to the Device

Sending data to the device works in a similar manner to receiving data from the device but using the host-to-device pages 2 and 3. The packets are structured identically. The host notifies the device of data to be read by writing to the host-to-device interrupt control register 0x2. Writing 1 to the first bit of this register triggers the interrupt on the device. Bits [1:7] indicate the purpose of the interrupt, which is ISH_INT_DATA_TRANSMISSION. Once the device successfully reads the packet, the packet checksum is written to data register 0x10 on page 1 to acknowledge receipt and allow the host to send the next packet.

A shared library (*libi2c_lib.so*) is provided that implements the communication protocol and handles data packetization. Refer to the Communication Protocol Library section for more information.

2.2. Output Data

Once your board is properly set up, the FPGA pipeline runs automatically when the board is powered up. The pipeline transmits detected information through packets that are regularly sent in the format as described in the Data Format section.

2.2.1. Data Format

Refer to Table 2.2 through Table 2.18 to interpret the pipeline binary output.

2.2.1.1. Data Packet Structure

Table 2.2 describes the overall data packet (pipeline output) structure. All packets start with a flag. Data length only factors in the data field and excludes the start flag, data length, and checksum fields. The Fletcher's checksum algorithm is used to ensure data integrity. The checksum value calculated is based solely on the data, without including the start flag and data length. The host must capture the data, compute its Fletcher's checksum, and compare the checksum value with the value sent by the pipeline. If the two checksum values do not match, the data must be discarded.

Table 2.2. Data Packet Structure

Name	Size (bytes)	Value	Comment
Start flag	1	0x7e	—
Data length	2	0–65535	Length of the data part of the packet only (in bytes).
Data	0–65535	Variable	Refer to the Data Field Format section or Input Data Field Format section for a description of the format of this value.
Fletcher's checksum	2	0–65535	The checksum is computed on the Data part only.

2.2.1.2. Data Field Format

Table 2.3 describes the output data field format.

Table 2.3. Data Field Format

Name	Size (bytes)	Value	Unit	Comment
Response type	1	0–255	—	Refer to Table 2.4.
Response version	1	0–255	—	Version of the data format based on the response type.
Response data	Variable	Variable	—	Dependent on response type and response version. Refer to Table 2.5 through Table 2.7.

Table 2.4 shows the response types. The response type determines the content of the response data.

Table 2.4. Response Types

Name	Description	Numeric Value
RT_NONE	None; the packet is probably wrong	0

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Name	Description	Numeric Value
RT_DATA	The rest of the packet will contain metadata	1
RT_GET	The rest of the packet will contain a requested MT_GET from a client request	2
RT_ACK	The rest of the packet will contain the numbers of acknowledgements (ACKs) processed	3

Table 2.5 through Table 2.7 describe the structure of the response data according to the response type.

Table 2.5. Output Data for RT_DATA Response Type

Name	Size (bytes)	Value	Unit	Comment
Image width	4	0–4294967295	pixels	Interpret as unsigned int32.
Image height	4	0–4294967295	pixels	Interpret as unsigned int32.
ROI left	4	0–4294967295	pixels	Interpret as unsigned int32.
ROI top	4	0–4294967295	pixels	Interpret as unsigned int32.
ROI right	4	0–4294967295	pixels	Interpret as unsigned int32.
ROI bottom	4	0–4294967295	pixels	Interpret as unsigned int32.
Number of users	2	0–65535	—	Interpret as unsigned int16.
Camera streaming enabled	2	0–1	—	0 – Camera ON 1 – Camera OFF
Person 0 data	Variable	Variable	—	Refer to the Person Data Field Format section.
Person 1 data	Variable	Variable	—	Refer to the Person Data Field Format section.
Person x data[1]	Variable	Variable	—	Refer to the Person Data Field Format section.
Ideal user data[2]	Variable	Variable	—	Refer to the Ideal Person Data Field Format section.

Notes:

1. The actual number of person data fields depends on the value of the *number of users* field. If this number is 0, there is no person data field.
2. The *ideal user data* field is only available if at least one person is detected.

Table 2.6. Output Data for RT_GET Response Type

Name	Size (bytes)	Value	Unit	Comment
Network type	1	0–255	—	Refer to the Network Type section.
Input setting type	1	0–255	—	Refer to the Input Setting Type section.
Value	4	0–4294967295	—	Interpret as unsigned int32.

Table 2.7. Output Data for RT_ACK Response Type

Name	Size (bytes)	Value	Unit	Comment
Number of ACKs	1	0–255	—	Interpret as unsigned int8.

2.2.1.3. Person Data Field Format

Table 2.8 describes the output person data field format.

Table 2.8. Person Data Field Format

Name	Size (bytes)	Value	Unit	Comment
Person index	2	0–65535	—	Interpret as unsigned int16.
Face ID index	1	0–255	—	Interpret as unsigned int8.
Face ID status	1	0–255	—	Interpret as unsigned int8. Refer to the Face ID Status section for definitions.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Name	Size (bytes)	Value	Unit	Comment
Person available	4	0–1	—	Interpret as a Boolean. 0 – Bounding box of the person is invalid 1 – Bounding box of the person is valid
Person body bounding box confidence	4	0–100	—	Interpret as signed int32, then divide by 1024.
Person body bounding box left	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the left edge of the image.
Person body bounding box top	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the top edge of the image.
Person body bounding box right	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the left edge of the image.
Person body bounding box bottom	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the top edge of the image.
Person body pose	4	FRONT, SIDE, BACK	—	Interpret as unsigned int32. 0 – FRONT 1 – SIDE 2 – BACK
Person body frontal pose confidence	4	−2147483648 to 2147483647	—	Interpret as signed int32, then divide by 1024.
Person body non-frontal pose confidence	4	−2147483648 to 2147483647	—	Interpret as signed int32, then divide by 1024.
Person distance from camera	4	0–4294967295	cm	Interpret as unsigned int32.

2.2.1.4. Person Face Field Format

Table 2.9 describes the output person face field format.

Table 2.9. Person Face Field Format

Name[1]	Size (bytes)	Value	Unit	Comment
Face available	4	0–1	—	Interpret as a Boolean. 0 – Face's bounding box is invalid 1 – Face's bounding box is valid
Person's face bounding box confidence	4	0–100	—	Interpret as signed int32, then divide by 1024.
Person's face bounding box left	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the left edge of the image.
Person's face bounding box top	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the top edge of the image.
Person's face bounding box right	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the left edge of the image.
Person's face bounding box bottom	4	−2147483648 to 2147483647	pixels	Interpret as signed int32. Pixels from the top edge of the image.
Face distance from camera	4	0–4294967295	cm	Interpret as unsigned int32.

Note:

1. Fields related to a person's face are only valid when the person's *face available* field is set to 1.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

2.2.1.5. Ideal Person Data Field Format

Table 2.10 describes the output ideal person data field format.

Table 2.10. Ideal Person Data Field Format

Name	Size (bytes)	Value	Unit	Comment
Ideal person index	2	0–65535	—	Interpret as unsigned int16. Indicates which user is the ideal user. This value must be in the range 0 to (*number of users* – 1).
Face ID Index	0.5	0–15	—	Interpret as unsigned int8.
Last registered ID	0.5	0–15	—	Interpret as unsigned int8.
User count in face ID gallery	0.5	0–15	—	Interpret as unsigned int8.
Size of gallery	0.5	0–15	—	Interpret as unsigned int8.
Ideal person face ID status	4	0–5	—	Interpret as unsigned int32. Refer to the Face ID Status section for definitions.
Person's face yaw angle	4	−180 to 180	°	Interpret as signed int32, then divide by 1024.
Person's face pitch angle	4	−180 to 180	°	Interpret as signed int32, then divide by 1024.
Person's face roll angle	4	−180 to 180	°	Interpret as signed int32, then divide by 1024.
Face validation confidence	4	0–100	—	Interpret as signed int32, then divide by 1024.

2.2.1.6. Face ID Status

Table 2.11 shows the output face ID status definitions.

Table 2.11. Face ID Status Definitions

Numeric Value	Name	Description
0	USER_REGISTERED	User has been found in the gallery.
1	USER_UNREGISTERED	User has not been found in the gallery.
2	USER_STATUS_UNKNOWN	Default value.
3	USER_REQUIREMENTS_UNMET	Face ID was not run on this user because the angle and distance requirements were not met.
4	USER_FID_DISABLED	Face ID feature is disabled.
5	USER_NO_GALLERY	No user has been registered in the gallery.

2.2.1.7. Input Data Field Format

You can send data to the board to change the configuration of the HMI pipeline at runtime. The same data packet structure as shown in Table 2.2 is used.

Table 2.12 describes the input data field format.

Table 2.12. Input Data Field Format

Name	Size (bytes)	Value	Unit	Comment
Message type	1	0–255	—	Refer to Table 2.13.
Message version	1	0–255	—	Message version. Use 0 unless instructed differently.
Message data	Variable	Variable	—	Dependent on the message type. Refer to Table 2.14 through Table 2.16.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 2.13 shows the message types. The message type determines the content of the message data.

Table 2.13. Message Types

Name	Description	Numeric Value
MT_NONE	None; the packet is probably wrong	0
MT_SET	The rest of the packet will set a setting value	1
MT_GET	The rest of the packet will contain a request to get a setting value	2
MT_GET_BATCH	The rest of the packet will contain a request to get multiple setting values	3

Table 2.14 through Table 2.16 describe the structure of the message data according to the message type.

Table 2.14. Input Data for MT_SET Message Type

Name	Size (bytes)	Value	Unit	Comment
Network type	1	0–255	—	Refer to the Network Type section.
Setting type	1	0–255	—	Refer to the Input Setting Type section.
Setting value	4	0–4294967295	—	Interpret as unsigned int32.

Table 2.15. Input Data for MT_GET Message Type

Name	Size (bytes)	Value	Unit	Comment
Network type	1	0–255	—	Refer to the Network Type section.
Setting type	1	0–255	—	Refer to the Input Setting Type section.
Notify	1	0–1	—	Interpret as a Boolean. 1 – Pipeline sends RT_GET messages when the value changes from any source.

Table 2.16. Input Data for MT_GET_BATCH Message Type

Name	Size (bytes)	Value	Unit	Comment
Network types	2	0–65535	—	Refer to the Network Type section. Mask of the requested network types. Bit 0 corresponds to network type 0 (face detection), bit 1 corresponds to network type 1 (landmarks and face validation), and so on.
Setting type	4	0–4294967295	—	Refer to the Input Setting Type section. Mask of the requested setting types. Bit 0 corresponds to network type 0 (ENABLED), bit 1 corresponds to network type 1 (IPS), and so on.
Notify	1	0–1	—	Interpret as a Boolean 1 – Pipeline sends RT_GET messages when the value changes from any source; for all settings requested in the masks.

2.2.1.8. Network Type

Table 2.17 shows the network types. Network types are used in responses and messages to specify the configuration of networks.

Table 2.17. Network Types

Name	Description	Numeric Value
FD	Face detection	0

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Name	Description	Numeric Value
LM_FV	Landmarks and face validation; depends on face detection	1
FID	Face ID; depends on landmarks face validation	2
PD	Person detection	3
HD[1]	Hand detection	4
HLMV[1]	Hand landmarks validation; depends on hand detection	5

Note:

1. Only available with a specific FPGA firmware that supports hand data. Contact Lattice Semiconductor for more information.

2.2.1.9. Input Setting Type

Table 2.18 shows the input setting types used in responses and messages to specify settings to change (with MT_SET) or to get (with MT_GET) for networks.

Table 2.18. Input Setting Types

Name	Description	Numeric Value
ENABLED	0 – Disable network 1 – Enable network	0
IPS	Maximum inference per second (IPS). Network will not run faster than this IPS value.	1
RESERVED	Reserved	2–7
FACE_ID_CLEAR	Supported only when network type is FID. When the setting value is not 0, the firmware clears all registrations in the gallery. Resets to 0 after operation is done.	8
FACE_ID_REGISTER	Supported only when network type is FID. When the setting value is not 0, the firmware attempts to register the ideal user in the gallery. If the user has already been added to the gallery, the firmware either improves the accuracy by adding another entry in the gallery associated with this user or skips the operation. Resets to 0 after operation is done.	9
FACE_ID_CLEAR_ID	Supported only when network type is FID. When the setting value is not 0, the firmware clears the entry matching the setting value in the face ID gallery (if available). Resets to 0 after operation is done.	10
FACE_ID_SECONDARY_USERS	Supported only when network type is FID. If the setting value is 1, the face ID operation attempts to recognize secondary users. If the setting value is 0, the face ID operation only attempts to recognize the ideal user.	16

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3. Communication Protocol Library

The *libi2c_lib.so* library provides a synchronous communication protocol over I2C to send data bidirectionally between the RPi host and RISC-V device firmware. The library handles data packetization, so you only need to provide the buffer and its length when sending or receiving data.

Code samples are available in the RESOURCES folder within the sensAIReferenceDesign/demo/clnx directory.

3.1. Communication with the Host

3.1.1. Overview

On the host-side, use the *I2CDevice* class to send data to or receive data from the device. Key functions are *write_to_device(uint8_t* data, uint32_t length)* and *read_from_device().* Include the *i2c_adapter.h* header file in your application code.

3.1.2. Initializing the Device

Initialize the I2CDevice class with the i2c adapter number and device number for your device. If you are using a non-standard IRQ GPIO pin, specify this when instantiating the class.

```
#include "i2c_adapter.h"
I2CDevice dev(0, 0x30); // adapter_number = 0, device_number = 0x30
// Optional: specify IRQ GPIO pin
// I2CDevice dev(0, 0x30, 26);
```

3.1.3. Sending Data to the Device

I2CDevice::write_to_device handles sending data to the device. Provide a uint8_t* buffer containing the bytes to send and the buffer length. The function returns 0 on success. Otherwise, it returns the number of remaining unsent bytes.

```
constexpr uint32_t length = 8;
uint8_t bytes[length] = {1,2,3,4,5,6,7,8};
int32_t remaining = dev.write_to_device(bytes, length);
if (remaining != 0) { /* Implement retry/backoff or error handling */}
```

Note: I2CDevice::write_to_device is a blocking function. It waits for the device to read the sent data. The function triggers an interrupt on the device to notify it that data is waiting to be read.

3.1.4. Receiving Data from the Device

I2CDevice::read_from_device handles receiving data from the device. This function returns a vector containing the bytes read from the device. Call this method periodically or within your event loop to retrieve data. The function returns an empty vector if no data is pending.

```
std::vector data = dev.read_from_device();
 if (!data.empty()) { /* Process the received payload */}
```

Note: The device-side write function is a blocking function. It waits for the host to read the data.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

3.2. Pure C Interface and Python ctypes

3.2.1. Pure C API

A pure C wrapper is provided around the C++ I2CDevice class for projects that require C or foreign function interfaces. For more information, refer to the *i2c_pure_c.h* header file. The following functions are provided:

```
I2CDeviceHandle i2c_device_create(uint8_t adapter_number, uint8_t
device_number);

void i2c_device_destroy(I2CDeviceHandle handle);

int32_t i2c_write_to_device(I2CDeviceHandle handle, uint8_t* data, uint32_t
length);

int32_t i2c_read_from_device(I2CDeviceHandle handle, uint8_t* out_data, uint32_t
max_length);
```

These functions mirror the high-level usage described in the Communication with the Host section and can be called from C code directly.

3.2.2. Python ctypes Integration

The libi2c_lib.so library provides a pure C API. This enables Python ctypes to call its functions. For an example, refer to the i2c_serial.py file, which implements a serial-style interface using the I2C library as its foundation.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4. Edge Vision Engine SDK API

This section explains how the EVE SDK API provides a user-friendly, high-level programming interface for connecting to the CrossLinkU-NX SoM and processing HMI metadata for custom applications. The EVE SDK API approach is particularly recommended for HMI-based interaction proofs of concept or scenarios where quick implementation is required.

4.1. Getting Started and Code Samples

Code samples and the CMakeLists.txt file are available in the RESOURCES folder within the sensAIReferenceDesign/demo/clnx directory. Resources include code samples to compile, run, and retrieve data using the EVE SDK and information on how to create a sample application on the host.

4.2. EVE SDK API Endpoints

4.2.1. CreateEve()

This API creates and initializes the EVE processing pipeline. The first run might take a comparatively longer time because EVE compiles and caches OpenCL kernels, which is normal behavior.

Note: Call this API before any other EVE APIs. If use of a camera is required, the typical process is to select or select and format the camera (using camera APIs) after calling this API, then call StartEve().

```
enum EveError EveSDK_EXPORT CreateEve(struct EveStartupParameters
startupParameters);
```

In/Out	Parameter	Description	Returns
In	startupParameters	Instance of struct EveStartupParameters that specifies the EVE configurable startup parameters.	EveError: • EVE_ERROR_NO_ERROR • EVE_ERROR_ CAMERA_MANAGER _NOT_FOUND • EVE_ERROR_ PIPELINE_NOT_FOUND

4.2.2. EveRegisterDataCallback()

This API registers your data callback. This is a mechanism for receiving results each time EVE processes a frame. EVE calls your function with a pointer to an EveProcessingCallbackReturnData instance on every processed frame. The typical process flow is as follows:

- Pull processed outputs (for example, call EveGetFpgaData() from EveFpga.h or read image analysis results).
- Decide to continue or stop.
- Write to the provided EveProcessingCallbackReturnData instance.

Note: Call this API before StartEve(). Keep the callback fast and thread safe, and avoid long blocking operations.

```
enum EveError EveSDK_EXPORT EveRegisterDataCallback(void (*callback)(struct
EveProcessingCallbackReturnData *));
```

In/Out	Parameter	Description	Returns
In	*callback	Pointer to a user-provided function. EVE invokes this function on every processed frame with a pointer to struct EveProcessingCallbackReturnData.	EveError: • EVE_ERROR_NO_ ERROR • EVE_ERROR_NO_ CALLBACK
In/Out	*	Parameter of the callback function above. Pointer to struct EveProcessingCallbackReturnData. Do not free or manage this pointer. Set requestedState to one of the following:	

In/Out	Parameter	Description	Returns
		• EVE_REQUESTED_PROCESSING_STATE_CONTINUE – Keep running. • EVE_REQUESTED_PROCESSING_STATE_STOP – Request graceful stop.	

4.2.3. EveGetNumberOfCameras()

This API returns the number of cameras detected by EVE on the system. Some physical devices present multiple logical cameras so you may see more than one entry per device. For example, RGB and IR streams are reported separately.

Note: Call this API after CreateEve() is successful. Iterate indices from [0] to [count – 1] with EveGetCamera() to inspect details.

```
struct EveNumberOfCameras EveSDK_EXPORT EveGetNumberOfCameras();
```

In/Out	Parameter	Description	Returns
—	—	—	EveNumberOfCameras[1]

Note:

1. Refer to struct EveNumberOfCameras.

4.2.4. EveGetCamera()

This API returns information about the camera at the specified index, as reported by the OS and queried by EVE. The indices are assigned in the order the cameras are reported by the OS. When you exceed available indices, an EVE_NO_MORE_DATA error is reported in the returned instance of EveCamera. The typical process flow is as follows:

- Enumerate with EveGetNumberOfCameras().
- Pick an endpoint with EveGetCamera(i), where *i* represents the index of the camera to get information from.

Note: Call this API to locate special cameras such as cameras with a built-in FPGA device.

```
struct EveCamera EveSDK_EXPORT EveGetCamera(unsigned int camera);
```

In/Out	Parameter	Description	Returns
In	camera	Zero-based camera index (camera index starts at 0).	EveCamera[1]

Note:

1. Refer to struct EveCamera.

4.2.5. EveConfigureFpga()

This API configures the EVE connection and operational parameters for communicating with the FPGA.

Note: Call EveConfigureFpga() only after CreateEve() and after selecting the FPGA camera and format, for example, through EveGetCamera() and EveSetCamera(). Use the correct adapter number, device address, and IRQ GPIO for your SoM or board. The values in the code sample provided are values for the CrossLinkU-NX SoM.

```
struct EveFpgaOptions EveSDK_EXPORT EveConfigureFpga(struct EveFpgaOptions
options);
```

In/Out	Parameter	Description	Returns
In	options	Instance of struct EveFpgaOptions containing the connection and operational parameters for communicating with the FPGA.	EveFpgaOptions[1]: • On success, contains validated parameters to be used. On error, contains input parameters.

Note:

1. Refer to struct EveFpgaOptions.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.2.6. StartEve()

This API starts the EVE processing thread. The following conditions are required before starting EVE:

- CreateEve() is successful.
- Camera, FPGA, or both are configured.
- EveRegisterDataCallback() has been called.

```
enum EveError EveSDK_EXPORT StartEve();
```

In/Out	Parameter	Description	Returns
—	—	—	EveError: • EVE_ERROR_NO_ ERROR • EVE_ERROR_NOT_ CREATED • EVE_ERROR_NO_ CALLBACK

4.2.7. EveGetFpgaData()

This API retrieves the latest FPGA data from EVE associated with the most recently processed frame. This is typically called inside your data callback and registered through EveRegisterDataCallback() so that you can consume the FPGA outputs such as pipelineData.dataContent.numberOfUsers in the provided code sample.

Note: Call this API after EVE is created and started, ideally from the callback. Always check that fpgaData.data is non-null before de-referencing, and handle error fields if present. The code sample prints numberOfUsers only when data is non-null.

```
struct EveFpgaData EveSDK_EXPORT EveGetFpgaData();
```

In/Out	Parameter	Description	Returns
—	—	—	EveFpgaData[1]

Note:

1. Refer to struct EveFpgaData.

4.2.8. ShutdownEve()

This API shuts down the EVE processing pipeline and frees internal resources. This allows an application to exit safely and re-initialize later without issues. The typical process flow is as follows:

- Stop the processing thread.
- Close camera connections (if any).

Note: Call this API once a stop request has been made through the callback requestedState and processing has stopped.

```
enum EveError EveSDK_EXPORT ShutdownEve();
```

In/Out	Parameter	Description	Returns
—	—	—	EveError: • EVE_ERROR_NO_ERROR • EVE_ERROR_NOT_ CREATED

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.2.9. EveSetCamera()

This optional API sets the camera and format to be used by EVE. By default, EVE connects to the first reported camera by the system. The filter parameter is optional. You can leave any of its members zero initialized to allow EVE to set them. For example, if you set frames per second (FPS) to 30 while leaving the other fields at 0, EVE will connect to the highest resolution that offers 30 FPS. The preferred format is YUY2 because this format is typically provided by the OS and camera drivers without additional processing such as scaling or color conversion. For example, if you set the resolution to 640 x 480 while leaving other fields at 0, EVE will connect to a 640 x 480 YUY2 feed at the highest available FPS.

Note: Call this API after CreateEve().

```
struct EveError EveSDK_EXPORT EveSetCamera(unsigned int camera, struct
CCameraFormat filter);
```

In/Out	Parameter	Description	Returns
Out	camera	Pointer to EveProcessingCallbackReturnData instance on every processed frame. Do not free or manage this pointer.	EveError: • EVE_CAMERA_ INTERACTION_ WITHOUT_CAMERA • EVE_ERROR_NOT_CREATED • EVE_INVALID_CAMERA_ID
In	filter	Instance of struct CCameraFormat containing camera filters. For example, resolution, format, and FPS. The exact members can be found in CCameraStructs.h.	

4.2.10. EveSendImageForProcessing()

This API sends an image to EVE for processing. Results can be accessed after this function returns. To use this function, ensure that EVE is started with the imageProvider option of the EveStartupParameters instance set to EVE_CLIENT_PROVIDED.

```
enum EveError EveSDK_EXPORT EveSendImageForProcessing(struct EveInputImage
image);
```

In/Out	Parameter	Description	Returns
In	image	Instance of struct EveInputImage to be processed by EVE.	EveError: • EVE_CAMERA_ INTERACTION_ WITHOUT_CAMERA • EVE_ERROR_NOT_CREATED • EVE_INVALID_ IMAGE_ENCODING

4.2.11. EveConfigureFpgaDebug()

This API sets optional features for debugging and drawing purposes.

```
struct EveFpgaDebugOptions EveSDK_EXPORT EveConfigureFpgaDebug(struct
EveFpgaDebugOptions options);
```

In/Out	Parameter	Description	Returns
In	options	Instance of struct EveFpgaDebugOptions containing the debug feature toggles for the FPGA device or EVE pipeline.	EveFpgaDebugOptions[1]: • On success, contains validated parameters to be used. On error, contains input parameters.

Note:

1. Refer to struct EveFpgaDebugOptions.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.3. EVE SDK API Data Structures

4.3.1. struct EveStartupParameters

This struct defines the startup configuration for EVE when CreateEve() is called.
Note: You must zero-initialize this struct to ensure default behavior for unspecified fields.

Table 4.1. EveStartupParameters Parameters

Data Type	Struct Member	Description
enum EveGpuPreference	gpuPreference	Specifies graphics processing unit (GPU) preference for OpenCL acceleration. • EVE_NO_GPU – skips OpenCL initialization, which disables most algorithms that rely on GPU acceleration. This setting is useful for CPU-only or headless environments.
enum EveImageProvider	imageProvider	Defines the mechanism for EVE to receive images. • EVE_CAMERA – EVE connects to a camera directly. You cannot call EveSendImageForProcessing() in this mode. Frames are obtained from the camera pipeline. • EVE_CLIENT_PROVIDED – your application provides images manually through EveSendImageForProcessing(). Callback is not automatic for camera frames. You control when images are processed.
char *	pathOverride	Optional path override for locating EVE libraries. Leave as zero-initialized for default behavior. Specify this if your EVE SDK binaries are in a non-standard location. The maximum path length is 512 characters.

4.3.2. struct EveFpgaOptions

This struct contains the connection and operational parameters for communicating with the FPGA, and the status from configuring the FPGA connection. You pass CFpgaParameters through the corresponding API such as EveConfigureFpga(). EVE returns this struct, which allows you to check the error status.

Table 4.2. EveFpgaOptions Parameters

Data Type	Struct Member	Description
struct CFpgaParameters	parameters	Instance of the struct CFpgaParameters containing FPGA connection and operational parameters supplied.
enum EveError	error	Error code indicating the configuration status. Check this after calling EveConfigureFpga(). Possible values (as seen in the comment section of the FPGA header file EveFpga.h) are as follows: • EVE_ERROR_NO_ERROR • EVE_CAMERA_INTERACTION_WITHOUT_CAMERA • EVE_ERROR_NOT_STARTED

Table 4.3 shows the structure containing the FPGA connection and operational parameters.

Table 4.3. FPGA Parameters Structure

C++ Struct Name	Description
struct CFpgaParameters	Contains the FPGA connection and operational parameters. For example, connection type, I2C adapter number, device address, IRQ GPIO pin, pipeline version, and flags such as forceCameraOn. The exact members can be found in CFpgaData.h.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.3.3. struct EveFpgaDebugOptions

This struct contains the debug feature toggles for the FPGA device or EVE pipeline and an error code. When drawing is enabled, EVE renders annotations onto the image and might enable a YUY2-to-BGR decoder on systems with no configured GPU. This ensures you get visible overlays even without GPU acceleration.

Table 4.4. EveFpgaDebugOptions Parameters

Data Type	Struct Member	Description
unsigned int	enableDrawingOnImage	Turns on overlays or annotation drawing on the output image. 1 – On 0 – Off
enum EveError	error	Error code indicating the status of applying debug options. Check this after calling EveConfigureFpgaDebug(). There are no errors possible in the current version of EVE.

4.3.4. struct EveProcessingCallbackReturnData

This struct is the return payload your callback uses to notify EVE of the next action. EVE provides a pointer to this struct when it invokes your callback. Do not allocate or free this pointer yourself. Only modify the value of requestedState before returning in your callback.

Table 4.5. EveProcessingCallbackReturnData Parameters

Data Type	Struct Member	Description
enum EveRequested ProcessingState	requestedState	Depending on your application logic (for example, time limit reached, error condition, or user command), set to one of the following: • EVE_REQUESTED_PROCESSING_STATE_CONTINUE – Keep running. • EVE_REQUESTED_PROCESSING_STATE_STOP – Request graceful stop.

4.3.5. struct EveFpgaData

This struct combines a pointer to the latest FPGA data with an error code so you can safely consume results per processed frame. This is typically what you read inside your data callback, registered through EveRegisterDataCallback(), using APIs such as EveGetFpgaData(). Always check the pointer and error before accessing the FPGA data.

Table 4.6. EveFpgaData Parameters

Data Type	Struct Member	Description
struct CFpgaData *	data	Pointer to the structured FPGA data for the current or last processed frame. The CFpgaData type (defined in CFpgaData.h) contains your pipeline's payload such as counters and detections. This pointer can be null so check that it is not before dereferencing.
enum EveError	error	Error code indicating the status of data retrieval. Check this in conjunction with the pointer. Possible values are defined in the EVE SDK API Enumeration section.

The FPGA data contains several structures as summarized in Table 4.7. For more information, see the RESOURCES folder within the sensAIReferenceDesign/demo/clnx directory.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Table 4.7. FPGA Data Structures

C++ Struct Name	Description
CFpgaMessage	Contains the response type and version for each FPGA frame.
CFpgaImageDimensions	Holds the image width, height, and region of interest (ROI) for the frame.
CFpgaDataContent	Provides the number of users detected in the frame and the index of the ideal user.
CFpgaUserData	Main container for all per-user data, including face and person information.
CFpgaPersonData	Stores person-specific data for each user: availability, confidence, bounding box, posture, and distance.
CFpgaFaceData	Stores face-specific data for each user: availability, confidence, bounding box, and distance.
CFpgaFaceIdData	Contains gallery-related information: user ID, last registered FaceID, number of users, and gallery size.

4.3.6. struct EveNumberOfCameras

This struct contains a count of the number of cameras and an error code.

Table 4.8. EveNumberOfCameras Parameters

Data Type	Struct Member	Description
unsigned int	count	Number of camera entries.
enum EveError	error	Error code reported by EVE. Set to one of the following: • EVE_ERROR_NO_ ERROR • EVE_CAMERA _INTERACTION_ WITHOUT_ CAMERA • EVE_ERROR_NOT_ CREATED

4.3.7. struct EveCamera

This struct contains information on a camera and an error code. Always check the error before accessing the camera data.

Table 4.9. EveCamera Parameters

Data Type	Struct Member	Description
struct CCamera	data	Camera information. Refer to Table 4.10.
enum EveError	error	Error code reported by EVE. Set to one of the following: • EVE_CAMERA_INTERACTION_WITHOUT_CAMERA • EVE_ERROR_NOT_CREATED • EVE_INVALID_CAMERA_ID

The CCamera instance represents a camera in EVE and includes various parameters describing the device, which are implementation-specific, as shown in Table 4.10.

Table 4.10. CCamera Parameters

Data Type	Struct Member	Description
int	Id	Numerical ID; starts at 0 for the first detected camera
char *	pid	Product ID (PID) of the camera
char *	vid	Vendor ID (VID) of the camera
char *	name	Name of the camera
unsigned int	isHardwareCamera	1 for a physical camera; 0 for a software camera
unsigned int	isFpgaCamera	1 if the camera contains a Lattice FPGA device; 0 if not
unsigned int	isIrCamera	1 if the camera is an infrared camera; 0 if not

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.3.8. struct EveInputImage

This struct contains information on the input image.

Table 4.11. EveInputImage Parameters

Data Type	Struct Member	Description
unsigned char *	data	Image data
int	width	Image width
int	height	Image height
enum EveVideoFormat	encoding	Encoding of the input image: • EVE_NONE – No format is set • EVE_BGRA – Video format is BGRA (4 channels, 8 bits per channel) • EVE_YUY2 – Video format is YUY2 (2 channels, 8 bits per channel) • EVE_NV12 – Video format is NV12 (1 channel, 8 bits per channel) • EVE_MJPG – Video format is MJPG • EVE_BGR – Video format is BGR (3 channels, 8 bits per channel) • EVE_GRAYSCALE – Video format is Grayscale (1 channel, 8 bits per channel)

4.4. EVE SDK API Enumeration

Table 4.12. EVE SDK API Variables

Enum Value	Description
EVE_ERROR_NO_ERROR	Success
EVE_ERROR_NOT_CREATED	EVE processing pipeline has not been created; CreateEve() has not been called; operation requested before CreateEve()
EVE_ERROR_NOT_STARTED	EVE processing thread has not started; StartEve() has not been called **Note:** Potential call order issue. Ensure processing pipeline has been created and camera has been selected before requesting operation.
EVE_ERROR_PIPELINE_NOT_FOUND	Pipeline configuration for EVE is not available; pipeline missing (install issue)
EVE_ERROR_CAMERA_MANAGER_NOT_FOUND	Required camera components are unavailable; camera manager missing (install issue)
EVE_ERROR_NO_CALLBACK	Callback has not been registered; register callback before StartEve()
EVE_ERROR_NOT_ACCESSED_FROM_CALLBACK	Provided function pointer is invalid; access data only inside callback
EVE_INVALID_IMAGE_ENCODING	Unsupported image format
EVE_CALLBACK_WITHOUT_CONNECTING_TO_CAMERA	Callback registered without camera
EVE_CAMERA_INTERACTION_WITHOUT_CAMERA	Camera API used in non-camera mode; operation expects a camera but none is connected or selected
EVE_NO_MORE_DATA	End of data (for example, end of camera list)
EVE_INVALID_CAMERA_ID	Camera ID invalid
EVE_FACE_ID_INVALID_THRESHOLD	Threshold out of range [0.0, 1.0]
EVE_ERROR_NO_CAMERA_INTERACTION_WITH_CAMERA	Wrong API for active camera mode

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

4.5. Data Available with Current Version of HMI

The pipeline available in the current version of HMI only outputs a subset of the data available through the EVE SDK API. For more information on the data available, refer to the CBasicStruct.h, CCameraStructs.h, and CFpgaData.h header files in the RESOURCES folder within the Lattice CrossLinkU-NX SoM for HMI Demonstration Reference Design GitHub repository.

The data currently available are as follows:

- CFpgaMessage.serialStatus
- CFpgaPipelineData.pipelineType
- CFpgaImageDimensions.width
- CFpgaImageDimensions.height
- CFpgaImageDimensions.cropArea
- CFpgaDataContent.isCameraStreaming
- CFpgaDataContent.numberOfUsers
- CFpgaUserData.id
- CFpgaUserData.isStatusAvailable
- CFpgaPersonData.isPersonDataAvailable
- CFpgaPersonData.personConfidence
- CFpgaPersonData.personBox
- CFpgaPersonData.personPosture
- CFpgaPersonData.personFrontalPostureConfidence
- CFpgaPersonData.personNotFrontalPostureConfidence
- CFpgaPersonData.personDistance
- CFpgaFaceData.isFacePositionAvailable
- CFpgaFaceData.isFaceConfidenceAvailable
- CFpgaFaceData.faceConfidence
- CFpgaFaceData.isFaceGeometricBoxAvailable
- CFpgaFaceData.faceBox
- CFpgaFaceData.isFaceDistanceAvailable
- CFpgaFaceData.faceDistance
- CFpgaFaceData.isStatusAvailable
- CFpgaFaceData.faceIDStatus
- CFpgaFaceData.faceID
- CFpgaDataContent.isObjectDetectionAvailable
- CFpgaDataContent.isIdealUserDataAvailable
- CFpgaDataContent.idealUserDetected
- CFpgaDataContent.isIdealUserIndexValid
- CFpgaDataContent.idealUserIndex
- CFpgaFaceIdData.lastRegisteredFaceID
- CFpgaFaceIdData.userId
- CFpgaFaceIdData.gallerySize
- CFpgaFaceIdData.usersInGallery
- CFpgaUserData.isStatusAvailable
- CFpgaUserData.status
- CFpgaFaceData.isEulerAnglesIcsAvailable
- CFpgaFaceData.anglesICS
- CFpgaFaceData.faceLandmarksConfidence
- CFpgaFaceData.isFaceLandmarksConfidenceAvailable
- CFpgaDataContent.isHandGestureDataAvailable

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

References

- CrossLinkU-NX SoM for HMI Demonstration Reference Design (FPGA-RD-02333)
- CrossLinkU-NX web page
- Lattice CrossLinkU-NX SoM for HMI Demonstration Reference Design GitHub repository
- Lattice sensAI Edge Vision Engine web page
- I2C/SMBus Subsystem (Linux Kernel documentation) web page
- Lattice Insights for Lattice Semiconductor training courses and learning plans

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Technical Support Assistance

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

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

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

Revision History

Revision 1.0, January 2026

Section	Change Summary
All	Initial release.

© 2026 Lattice Semiconductor Corp. All Lattice trademarks, registered trademarks, patents, and disclaimers are as listed at www.latticesemi.com/legal.
All other brand or product names are trademarks or registered trademarks of their respective holders. The specifications and information herein are subject to change without notice.

www.latticesemi.com