Internal Flash Controller IP for MachXO5-NX

IP Version: v2.3.0

User Guide

FPGA-IPUG-02174-1.8

February 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.

Contents

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
AHB-L	Advanced High-Performance Bus – Lite
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
CPU	Central Processing Unit
EBR	Embedded Block RAM
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDL	Hardware Description Language
IP	Intellectual Property
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LMMI	Lattice Memory Mapped Interface
LUT	Look-Up Table
PLL	Phase-Locked Loop
RISC-V	Reduced Instruction Set Computer V
SDK	Software Development Kit
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SSPI	Target Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
UFM	User Flash Memory

1. Introduction

The Lattice MachXO5™-NX device contains an internal flash memory, which can be accessed for various purposes including configuration data storage. This document describes the Internal Flash Controller IP for MachXO5-NX.

1.1. Overview of the IP

The Internal Flash Controller IP for MachXO5-NX enables you to access the internal flash memory of the MachXO5-NX device using the Advanced High-Performance Bus – Lite (AHB-Lite) or Advanced Peripheral Bus (APB) interface.

1.2. Quick Facts

Table 1.1. Summary of the Internal Flash Controller IP for MachXO5-NX

IP Requirements	Supported Devices	MachXO5-NX (except LFMXO5-15D, LFMXO5-55TD)
	IP Changes[1]	Refer to the Internal Flash Controller IP for MachXO5-NX Release Notes (FPGA-RN-02085).
Resource Utilization	Supported User Interface	AHB-Lite, APB
	Resources	Refer to Appendix A. Resource Utilization.
Design Tool Support	Lattice Implementation	IP Core v2.3.0 – Lattice Propel™ 2025.2 Design Environment
	Synthesis	Synopsis® Synplify Pro® for Lattice, Lattice Synthesis Engine
	Simulation	Not supported
Driver Support	API Reference	Refer to the Internal Flash Controller Driver API Reference (FPGA-TN-02421).

Note:

1. In some instances, the IP may be updated without changes to the user guide. This user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates.

1.3. IP Support Summary

Table 1.2. Internal Flash Controller IP for MachXO5-NX Support Readiness

Device Family	IP	Interface	FIFO Depth	FIFO Endian (Write/Read)	Instantiate Internal Oscillator	Hardware Validated
LFMXO5-25	Internal Flash Controller for MachXO5-NX	APB	64	Little/Little	Yes	Yes
		AHB-Lite	64	Little/Little	Yes	Yes
LFMXO5-25	Internal Flash Controller for MachXO5-NX	APB	64	Little/Little	No	Yes
		AHB-Lite	64	Little/Little	No	Yes
LFMXO5-25	Internal Flash Controller for MachXO5-NX	APB	32	Big/Little	No	No
		AHB-Lite	32	Little/Big	No	No

1.4. Features

Key features of the Internal Flash Controller IP for MachXO5-NX include:

- AHB-Lite interface
- APB interface
- Initial user data to be programmed into the flash memory
- Up to 50 MHz input clock frequency

1.5. Licensing and Ordering Information

The Internal Flash Controller IP for MachXO5-NX is provided at no additional cost with the Lattice Propel design environment.

1.6. Hardware Support

Refer to the Example Design section for more information on the board used

1.7. Naming Conventions

1.7.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.7.2. Signal Names

Signal names that end with:

- _*n* are active low signals (asserted when value is logic 0)
- _*i* are input signals
- _*o* are output signals

1.7.3. Attribute Names

Attribute names in this document are formatted in title case and italicized (*Attribute Name*).

2. Functional Description

2.1. IP Architecture Overview

The Internal Flash Controller IP for MachXO5-NX has a system bus interface to access the registers. The input bus can be configured through IP attributes as either an AHB-Lite or APB interface. This IP also has four sub-blocks: register block, data buffer, controller, and flash memory. Figure 2.1 shows the connections of the sub-blocks.

Notes:

- With SSPI persistence enabled in user mode, there is a limitation on operation of the SSPI/JTAG bus when using the Internal Flash Controller IP for MachXO5-NX. Do not access the SSPI/JTAG port when accessing flash memory using the Internal Flash Controller IP for MachXO5-NX or vice versa.
- For this IP to operate properly, you must set the SLAVE_I2C_PORT/SLAVE_I3C_PORT option to DISABLE through the Device Constraint Editor of the Lattice Radiant™ software.

The register block handles all incoming bus transactions. It stores the register values and provides them to the controller. It also performs write or read access to the data buffer depending on the register access. The data buffer implements two FIFO buffers to temporarily store read data from and write data to the flash memory. The controller performs the transactions to the flash memory based on the register values. It also performs write or read access to the data buffer depending on the type of transaction with the flash memory. The flash memory receives the transactions from the controller.

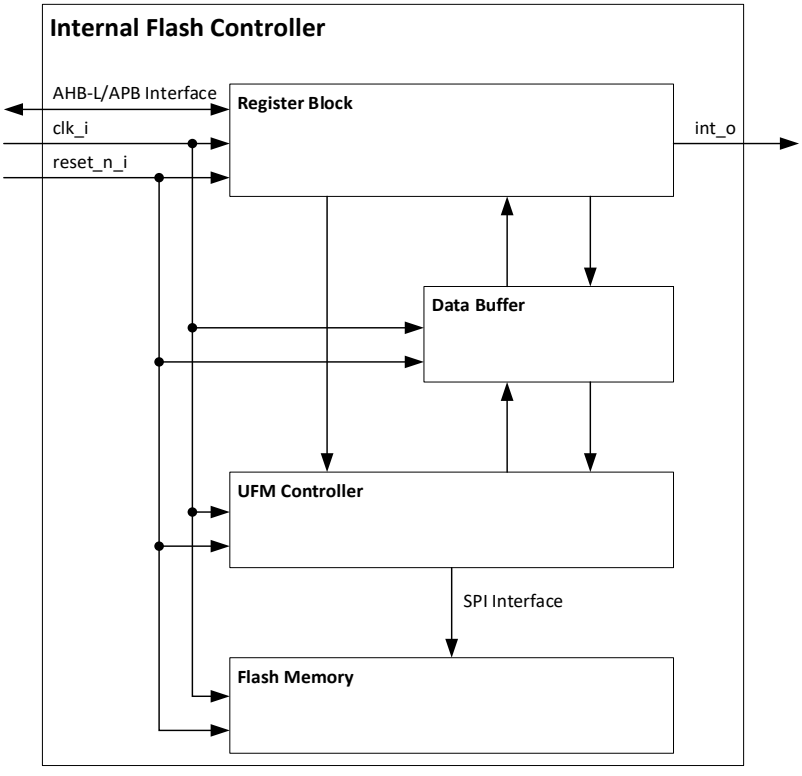

Figure 2.1. Internal Flash Controller Block Diagram (Default)

2.2. Clocking and Reset

The Internal Flash Controller IP for MachXO5-NX contains two internal clock domains: clk_i and lmmi_clk_i. These two clocks must always operate at the same frequency to ensure proper synchronization and reliable operation of the IP. The Internal Flash Controller IP for MachXO5-NX also contains three internal resets: asynchronous resets reset_n_i and lmmi_resetn_i, and soft reset (accessible through the register).

2.2.1. Clocking and Reset Overview

When *Instantiate Internal Oscillator* is enabled, lmmi_clk_i and lmmi_resetn_i for the flash memory are internally generated using an internal oscillator as shown in Figure 2.2. The CONFIG_CLKRST module generates the required clock and signals to ensure proper IP operation. When *Use HFCLK for LMMI CLK* is enabled, the register block, data buffer, and UFM controller use the high-frequency output of the internal oscillator as the clock signal as shown in Figure 2.3. If you are using only clk_i for all blocks including the flash memory, the frequency of the clock source must be set to less than or equal to 50 MHz.

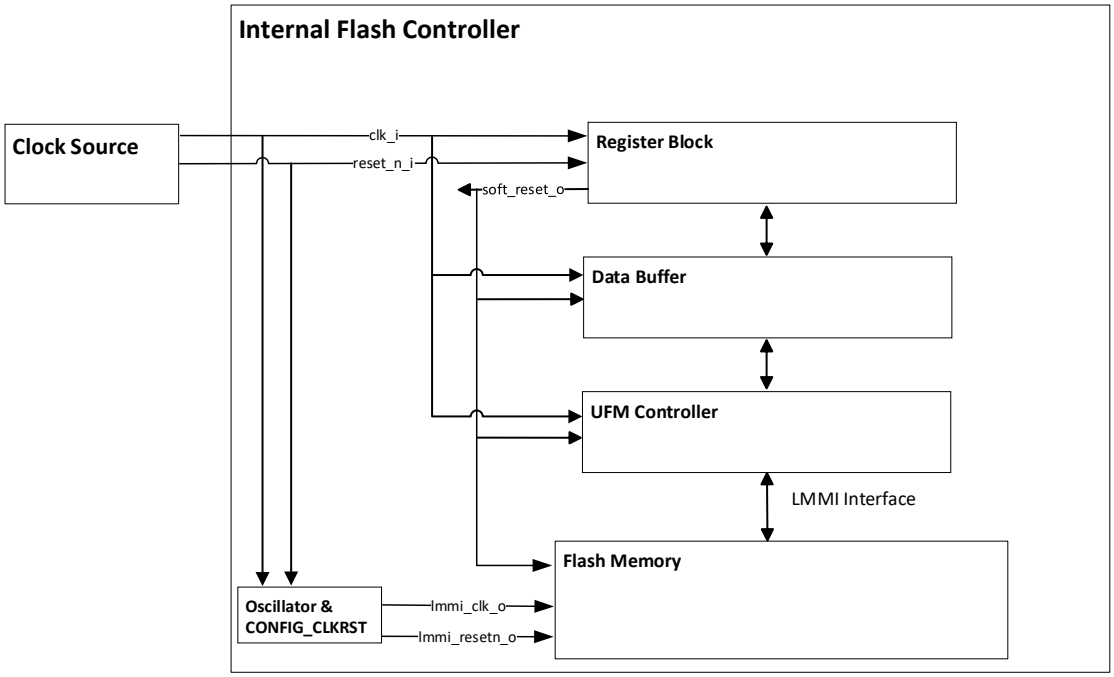

Figure 2.2. Clocking and Reset with Internal Oscillator

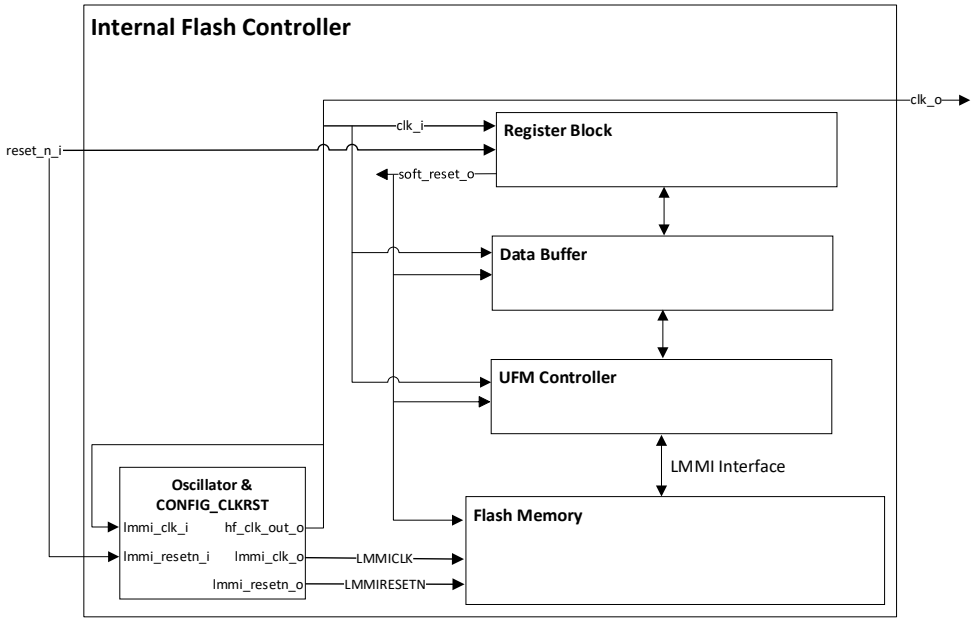

Figure 2.3. Clocking and Reset with Internal Oscillator (*Use HFCLK for LMMI CLK = Checked)*

If you do not want to instantiate the oscillator internally in the IP and intend to share the clock source with other IPs, you must disable *Instantiate Internal Oscillator* and instantiate an oscillator soft IP instead due to physical routing restrictions and for proper handling of clocking and reset. Figure 2.4 shows the connection of the oscillator soft IP to the ports of the Internal Flash Controller IP for MachXO5-NX. lmmi_clk_o and lmmi_resetn_o of the oscillator must drive lmmi_clk_i and lmmi_resetn_i, respectively. The inputs to clk_i and reset_n_i can be used for other modules.

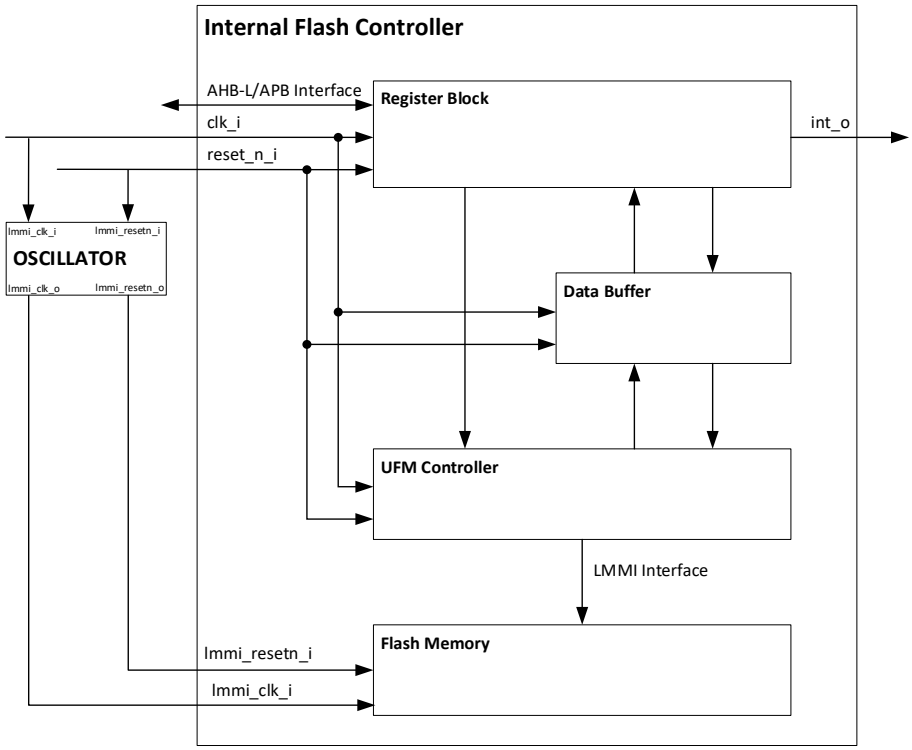

Figure 2.4. Clocking and Reset with Oscillator Soft IP

2.3. Block Partitioning

Table 2.1 shows the whole memory map of the LFMXO5-25 device's flash memory. Table 2.2 shows the whole memory map of the LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T devices' flash memory. Table 2.3 shows the whole memory map of the LFMXO5-55T and LFMXO5-100T devices' flash memory. You can modify the partition sizes by changing the attributes prior to IP generation. There are five options for configuring each of the three CFGx/UFMx partition sizes as shown in Table 2.4. The rules for configuring the partition sizes of USERDATAx are described in Table 3.3. Table 2.5 and Table 2.6 show sample settings for different flash partition block sizes.

Table 2.1. Flash Memory Map for LFMXO5-25 Device

Block Start Address (24-Bit Byte Address)	Block End Address (24-Bit Byte Address)	Block Count	Sector Count	Contents
00 (0x00_0000)	00 (0x00_FFFF)	1	16	—
01 (0x01_0000)	11 (0x0B_FFFF)	11	176	CFG0
12 (0x0C_0000)	15 (0x0F_FFFF)	4	64	UFM0
16 (0x10_0000)	26 (0x1A_FFFF)	11	176	CFG1
27 (0x1B_0000)	30 (0x1E_FFFF)	4	64	UFM1
31 (0x1F_0000)	41 (0x29_FFFF)	11	176	CFG2
42 (0x2A_0000)	45 (0x2D_FFFF)	4	64	UFM2
46 (0x2E_0000)	63 (0x3E_FFFF)	Total ≤ 17	Total ≤ 272	USERDATA0
				USERDATA1
				USERDATA2
				USERDATA3
				USERDATA4
				USERDATA5
				USERDATA6
				USERDATA7
				USERDATA8

Note: A single page has 256 bytes of data.

Table 2.2. Flash Memory Map for LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T Devices

Block Start Address (24-Bit Byte Address)	Block End Address (24-Bit Byte Address)	Block Count	Sector Count	Contents
00 (0x00_0000)	00 (0x00_FFFF)	1	16	—
01 (0x01_0000)	21 (0x15_FFFF)	21	336	CFG0
22 (0x16_0000)	28 (0x1C_FFFF)	7	112	UFM0
29 (0x1D_0000)	49 (0x31_FFFF)	21	336	CFG1
50 (0x32_0000)	56 (0x38_FFFF)	7	112	UFM1
57	62	6	96	USERDATA0

Block Start Address (24-Bit Byte Address)	Block End Address (24-Bit Byte Address)	Block Count	Sector Count	Contents
0x39_0000	(0x3E_FFFF)			
64 (0x40_0000)	84 (0x54_FFFF)	21	336	CFG2
85 (0x55_0000)	91 (0x5B_FFFF)	7	112	UFM2
92 (0x5C_0000)	127 (0x7F_FFFF)	Total ≤ 36	Total ≤ 576	USERDATA1
				USERDATA2
				USERDATA3
				USERDATA4
				USERDATA5
				USERDATA6
				USERDATA7
				USERDATA8

Note: A single page has 256 bytes of data.

Table 2.3. Flash Memory Map for LFMXO5-55T and LFMXO5-100T Devices

Block Start Address (24-Bit Byte Address)	Block End Address (24-Bit Byte Address)	Block Count	Sector Count	Contents
00 (0x00_0000)	00 (0x00_FFFF)	1	16	—
01 (0x01_0000)	33 (0x21_FFFF)	33	528	CFG0
34 (0x22_0000)	48 (0x30_FFFF)	15	240	UFM0
49 (0x31_0000)	81 (0x51_FFFF)	33	528	CFG1
82 (0x52_0000)	96 (0x60_FFFF)	15	240	UFM1
97 (0x61_0000)	129 (0x81_FFFF)	33	528	CFG2
130 (0x82_0000)	144 (0x90_FFFF)	15	240	UFM2
145 (0x91_0000)	255 (0xFE_FFFF)	Total ≤ 110	Total ≤ 1760	USERDATA0
				USERDATA1
				USERDATA2
				USERDATA3
				USERDATA4
				USERDATA5
				USERDATA6
				USERDATA7
				USERDATA8

Note: One page is composed of 256 bytes of data. One 4 KB sector is equivalent to 16 pages while one 64 KB block is equivalent to 256 pages.

For the LFMXO5-55T and LFMXO5-100T devices, CFGx and UFMx partitions have 33 and 15 64 KB blocks, respectively. CFGx and UFMx pairs can be combined to form 48 64 KB blocks. There are CFG0, CFG1, CFG2, UFM0, UFM1, and UFM2. USERDATA0 to USERDATA8 have a total of 110 64 KB blocks. Table 2.3 shows the boundary address for each memory partition of LFMXO5-55T and LFMXO5-100T devices.

Table 2.4. Different Combinations of CFGx/UFMx Block Size

Item No.	CFGx Block Count	UFMx Block Count	Use Partition Setting	Description
LFMXO5-25				
1	15	0	Use CFGx with size = 15 Use UFMx with size = 0	Only one partition CFGx having 15 block size
2	11	0	Use CFGx with size = 11 Use UFMx with size = 0	Only one partition CFGx having 11 block size; do not access the partition for UFMx
3	0	4	Use CFGx with size = 0 Use UFMx with size = 4	Only one partition UFMx having 4 block size; do not access the partition for CFGx
4	11	4	Use CFGx with size = 11 Use UFMx with size = 4	Two partitions CFGx having 11 block size and UFMx having 4 block size
5	0	0	Use CFGx with size = 0 Use UFMx with size = 0	No partition is selected. Do not access both.
LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T				
1	28	0	Use CFGx with size = 28 Use UFMx with size = 0	Only one partition CFGx having 28 block size
2	21	0	Use CFGx with size = 21 Use UFMx with size = 0	Only one partition CFGx having 21 block size; do not access the partition for UFMx
3	0	7	Use CFGx with size = 0 Use UFMx with size = 7	Only one partition UFMx having 7 block size; do not access the partition for CFGx
4	21	7	Use CFGx with size = 21 Use UFMx with size = 7	Two partitions CFGx having 21 block size and UFMx having 7 block size
5	0	0	Use CFGx with size = 0 Use UFMx with size = 0	No partition is selected. Do not access both.
LFMXO5-55T and LFMXO5-100T				
1	48	0	Use CFGx with size = 48 Use UFMx with size = 0	Only one partion CFGx having 48 block size.
2	33	0	Use CFGx with size = 33 Use UFMx with size = 0	Only one partion CFGx having 33 block size; you should not access the partiotion for UFMx.
3	0	15	Use CFGx with size = 0 Use UFMx with size = 15	Only one partition UFMx having 15 block size; you should not access the partition for CFGx.
4	33	15	Use CFGx with size = 33 Use UFMx with size = 15	Two partitions CFGx having 33 block size and UFMx having 15 block size.
5	0	0	Use CFGx with size = 0 Use UFMx with size = 0	Neither CFGx nor UFMx should be accessed.

Table 2.5. Sample Settings 1

Block Start Address	Block End Address	Block Count	Use Partition Setting	Contents
LFMXO5-25				
1	15	15	Use CFG0 with size = 15	CFG0
			Use UFM0 with size = 0	Do not access
16	26	11	Use CFG1 with size = 11	CFG1
NA	NA	0	Use UFM1 with size = 0	Do not access
NA	NA	0	Use CFG2 with size = 0	Do not access
42	45	4	Use UFM2 with size = 4	UFM2
46	55	10	USERDATA0 Size = 10	USERDATA0
56	63	8	USERDATA1 Size = 8	USERDATA1

Block Start Address	Block End Address	Block Count	Use Partition Setting	Contents
NA	NA	0	USERDATA2 Size = 0	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T				
1	28	28	Use CFG0 with size = 28	CFG0
NA	NA	0	Use UFM0 with size = 0	Do not access
29	49	21	Use CFG1 with size = 21	CFG1
NA	NA	0	Use UFM1 with size = 0	Do not access
57	62	6	USERDATA0 Size = 6	USERDATA0
NA	NA	0	Use CFG2 with size = 0	Do not access
85	91	7	Use UFM2 with size = 7	UFM2
92	95	4	USERDATA1 Size = 4	USERDATA1
96	99	4	USERDATA2 Size = 4	USERDATA2
100	103	4	USERDATA3 Size = 4	USERDATA3
104	107	4	USERDATA4 Size = 4	USERDATA4
108	111	4	USERDATA5 Size = 4	USERDATA5
112	115	4	USERDATA6 Size = 4	USERDATA6
116	119	4	USERDATA7 Size = 4	USERDATA7
120	127	8	USERDATA8 Size = 8	USERDATA8
LFMXO5-55T and LFMXO5-100T				
1	48	48	Use CFG0 with size = 48	CFG0
			Use UFM0 with size = 0	Do not Access
NA	NA	0	Use CFG1 with size = 0	Do not Access
82	96	15	Use UFM1 with size = 15	UFM1
97	129	33	Use CFG2 with size = 33	CFG2
130	144	15	Use UFM2 with size = 15	UFM2
145	157	13	USERDATA0 Size =13	USERDATA0
158	170	13	USERDATA1 Size =13	USERDATA1
171	183	13	USERDATA2 Size = 13	USERDATA2
184	196	13	USERDATA3 Size = 13	USERDATA3
197	209	13	USERDATA4 Size = 13	USERDATA4
210	222	13	USERDATA5 Size = 13	USERDATA5
223	235	13	USERDATA6 Size = 13	USERDATA6
236	248	13	USERDATA7 Size = 13	USERDATA7
249	254	6	USERDATA8 Size = 6	USERDATA8

Table 2.6. Sample Settings 2

Block Start Address	Block End Address	Block Count	Use Partition Setting	Contents
LFMXO5-25				
1	11	11	Use CFG0 with size = 11	CFG0
12	15	4	Use UFM0 with size = 4	UFM0
NA	NA	0	Use CFG1 with size = 0	Do not access

Block Start Address	Block End Address	Block Count	Use Partition Setting	Contents
NA	NA	0	Use UFM1 with size = 0	Do not access
NA	NA	0	Use CFG2 with size = 0	Do not access
NA	NA	0	Use UFM2 with size = 0	Do not access
46	53	8	USERDATA0 Size = 8	USERDATA0
54	57	4	USERDATA1 Size = 4	USERDATA1
58	59	2	USERDATA2 Size = 2	USERDATA2
60	60	1	USERDATA3 Size = 1	USERDATA3
61	62	2	USERDATA4 Size = 2	USERDATA4
NA	NA	NA	USERDATA5 Size = 0	Do not access
NA	NA	NA	Automatically set to '0'	Do not access
NA	NA	NA	Automatically set to '0'	Do not access
NA	NA	NA	Automatically set to '0'	Do not access
LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T				
1	21	21	Use CFG0 with size = 21	CFG0
22	28	7	Use UFM0 with size = 7	UFM0
29	56	28	Use CFG1 with size = 28	CFG1
NA	NA	0	Use UFM1 with size = 0	Do not access
57	61	5	USERDATA0 Size = 5	USERDATA0
NA	NA	0	Use CFG2 with size = 0	Do not access
85	91	7	Use UFM2 with size = 7	UFM2
92	127	36	USERDATA1 Size = 36	USERDATA1
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
NA	NA	0	Automatically set to '0'	Do not access
LFMXO5-55T and LFMXO5-100T				
NA	NA	0	Use CFG0 with size = 0	Do not access
NA	NA	0	Use UFM0 with size = 0	Do not access
49	81	33	Use CFG1 with size = 33	CFG1
82	96	15	Use UFM1 with size = 15	UFM1
97	129	33	Use CFG2 with size = 33	CFG2
130	144	15	Use UFM2 with size = 15	UFM2
145	208	64	USERDATA0 Size = 64	USERDATA0
209	240	32	USERDATA1 Size = 32	USERDATA1
241	254	14	USERDATA2 Size = 14	USERDATA2
NA	NA	NA	Automatically set to '0'	Do not access
NA	NA	NA	Automatically set to '0'	Do not access
NA	NA	NA	Automatically set to '0'	Do not access
NA	NA	NA	Automatically set to '0'	Do not access
NA	NA	NA	Automatically set to '0'	Do not access
NA	NA	NA	Automatically set to '0'	Do not access

2.4. Setting Initial Data in Flash Memory

This section provides information on how to set the initial data for the flash device using attributes. Table 2.7 describes the five options available for initializing the flash memory partition. Note that initial data is for simulation purpose only. Initial value options create memory initialization files for convenience. These options are intended for optional customer-driven simulation and do not indicate official simulation support.

Table 2.7. Flash Memory Partition Initialization Options

Partition Initial Value Format	Description
ALL 1	An initialization data file is generated automatically containing each bit of the whole memory partition set to *1*.
ALL 0	An initialization data file is generated automatically containing each bit of the whole memory partition set to *0*.
NONE	No initialization data file is generated. Initialization of the MachXO5-NX device flash memory uses the factory default value, which is equivalent to ALL 1.
BINFILE	Initialization data file of the memory partition that you provide. The initialization data file is in the binary memory file format.
HEXFILE	Initialization data file of the memory partition that you provide. The initialization data file is in the hexadecimal memory file format.

2.4.1. Binary File

The binary file is a text file of 0s and 1s. The number of rows is equal to the block size of the partition multiplied by 128. Each row has 4096 bits.

2.4.2. Hexadecimal File

The hexadecimal file is a text file of hexadecimal characters. The number of rows is equal to the block size of the partition multiplied by 128. Each row has 1024 hexadecimal characters.

2.5. Transaction Offset Computation

The Internal Flash Controller IP for MachXO5-NX computes the address of the flash transaction based on TRANSACTION_ADDR.transaction_partition and TRANSACTION_ADDR.transaction_offset. transaction_partition determines the Partition Start Address as shown in the following tables. For a flash transaction address to be valid, Partition Start Address + transaction_offset must not exceed the next Partition Start Address.

Table 2.8. Flash Memory Map (Partition) for LFMXO5-25 Device

Partition Start Address	transaction_partition	Partition
0x01_0000	0x00	CFG0
0x0C_0000	0x01	UFM0
0x10_0000	0x02	CFG1
0x1B_0000	0x03	UFM1
0x1F_0000	0x04	CFG2
0x2A_0000	0x05	UFM2
0x2E_0000	0x06	USERDATA0
0x2E_0000 + (Size of USERDATA0 × 0x01_0000)	0x07	USERDATA1
Address of USERDATA1 + (Size of USERDATA1 × 0x01_0000)	0x08	USERDATA2
Address of USERDATA2 + (Size of USERDATA2 × 0x01_0000)	0x09	USERDATA3
Address of USERDATA3 + (Size of USERDATA3 × 0x01_0000)	0x0A	USERDATA4
Address of USERDATA4 + (Size of USERDATA4 × 0x01_0000)	0x0B	USERDATA5
Address of USERDATA5 + (Size of USERDATA5 × 0x01_0000)	0x0C	USERDATA6
Address of USERDATA6 + (Size of USERDATA6 × 0x01_0000)	0x0D	USERDATA7

Partition Start Address	transaction_partition	Partition
Address of USERDATA7 + (Size of USERDATA7 × 0x01_0000)	0x0E	USERDATA8

Table 2.9. Flash Memory Map (Partition) for LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T Devices

Partition Start Address	transaction_partition	Partition
0x01_0000	0x00	CFG0
0x16_0000	0x01	UFM0
0x1D_0000	0x02	CFG1
0x32_0000	0x03	UFM1
0x39_0000	0x06	USERDATA0
0x40_0000	0x04	CFG2
0x55_0000	0x05	UFM2
0x5C_0000 + (Size of USERDATA1 × 0x01_0000)	0x07	USERDATA1
Address of USERDATA1 + (Size of USERDATA2 × 0x01_0000)	0x08	USERDATA2
Address of USERDATA2 + (Size of USERDATA3 × 0x01_0000)	0x09	USERDATA3
Address of USERDATA3 + (Size of USERDATA4 × 0x01_0000)	0x0A	USERDATA4
Address of USERDATA4 + (Size of USERDATA5 × 0x01_0000)	0x0B	USERDATA5
Address of USERDATA5 + (Size of USERDATA6 × 0x01_0000)	0x0C	USERDATA6
Address of USERDATA6 + (Size of USERDATA7 × 0x01_0000)	0x0D	USERDATA7
Address of USERDATA7 + (Size of USERDATA7 × 0x01_0000)	0x0E	USERDATA8

Table 2.10. Flash Memory Map (Partition) for LFMXO5-55T and LFMXO5-100T Devices

Partition Start Address	transaction_partition	Partition
0x01_0000	0x00	CFG0
0x22_0000	0x01	UFM0
0x31_0000	0x02	CFG1
0x1B_0000	0x03	UFM1
0x52_0000	0x04	CFG2
0x61_0000	0x05	UFM2
0x82_0000	0x06	USERDATA0
0x91_0000 + (Size of USERDATA0 × 0x01_0000)	0x07	USERDATA1
Address of USERDATA1 + (Size of USERDATA1 × 0x01_0000)	0x08	USERDATA2
Address of USERDATA2 + (Size of USERDATA2 × 0x01_0000)	0x09	USERDATA3
Address of USERDATA3 + (Size of USERDATA3 × 0x01_0000)	0x0A	USERDATA4
Address of USERDATA4 + (Size of USERDATA4 × 0x01_0000)	0x0B	USERDATA5
Address of USERDATA5 + (Size of USERDATA5 × 0x01_0000)	0x0C	USERDATA6
Address of USERDATA6 + (Size of USERDATA6 × 0x01_0000)	0x0D	USERDATA7
Address of USERDATA7 + (Size of USERDATA7 × 0x01_0000)	0x0E	USERDATA8

2.6. Programming Flow

2.6.1. Write Access Operation 1

The following is the recommended procedure for executing a write transaction using the Internal Flash Controller IP:

Note: This procedure assumes that the target partition has been properly erased prior to initiating the write operation. Refer to the Erase Operation section.

1. Set INT_ENABLE.done_en to 1 to enable the interrupt that indicates sending command to flash is done.
2. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
   - TRANSACTION_ADDR.transaction_partition
   - TRANSACTION_ADDR.transaction_offset
3. Configure the TRANSACTION_CTRL register to specify the type of transaction – *write access* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
   - TRANSACTION_CTRL.transaction_type – Set to *write access* (2'b00).
   - TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to write. The maximum number of bytes for a write transaction is 256 so you may set this up to 'd255.

   Ensure that the two key fields listed above are properly set.
4. Flush the WR FIFO by writing to the FIFO_CTRL register. For more information, refer to the FIFO_CTRL Register section.
5. Write data to the WR_FIFO_DATA register repeatedly; up to N times, where N is determined by two factors:
   - Configured FIFO depth of WR FIFO
   - TRANSACTION_CTRL.transaction_bytes_m1 setting

   Each write to WR_FIFO_DATA transfers four data bytes. A single write transaction can accommodate a maximum of 256 data bytes, which corresponds to 64 writes to the WR_FIFO_DATA register.

   Note: When writing to the WR_FIFO_DATA register, ensure that the write FIFO is not full to avoid data loss or write errors. You may optionally check the FIFO_STATUS.wr_fifo_full flag before each write operation to the WR_FIFO_DATA register. If FIFO_STATUS.wr_fifo_full is set to 1, defer the write operation until space becomes available.
6. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.
7. Poll FIFO_STATUS.done until it is set to 1. This indicates that the command has been successfully sent to the flash and the controller can queue the next command before interrupt service routine (ISR) processes are complete. It does not indicate completion of the flash operation.
8. After polling FIFO_STATUS.done, monitor the int_o signal, enabled in Step 1, for assertion. This interrupt confirms that the flash operation has completed.
9. Poll INT_STATUS.done_int until it is set to 1 to verify the interrupt, then clear the INT_STATUS register. For more information, refer to the INT_STATUS Register section.
10. Optional: Initiate the next operation (erase, write access, or read access).

Figure 2.5 and Figure 2.6 show the relationship between the data bytes pushed in the WR FIFO and the data bytes written in the flash memory. Data to be written must be sent through the WR_FIFO_DATA register. Each valid write access to the WR_FIFO_DATA register pushes four bytes of data. In an empty WR FIFO, the first write access to WR_FIFO_DATA is pushed as Data0. The second write access is pushed as Data1 and so on. A total of 64 FIFO levels can accommodate the maximum number of bytes (256 bytes) to be written in one transaction.

The address of the transaction is computed by the IP using the TRANSACTION_ADDR settings. The data is written to the flash memory starting from the computed transaction address. The next byte is written to transaction address + 1 until the total number of bytes to be written is completed. Start the write access to the flash memory by setting TRANSACTION_CTRL.start_transaction to *1* with TRANSACTION_CTRL.transaction_type = 0b00. The total number of bytes to be written is set in the TRANSACTION_CTRL.transaction_bytes_m1 field.

Take note of the page boundaries (last eight bits of the write address bits = 0x00). *Write address + total number of bytes to be written* must not cross the page boundaries. If you want to write 256 bytes of data, the write address must be set to a page boundary (last eight bits of the write address bits = 0x00). If the page boundary is reached, the write address wraps around the start of the page.

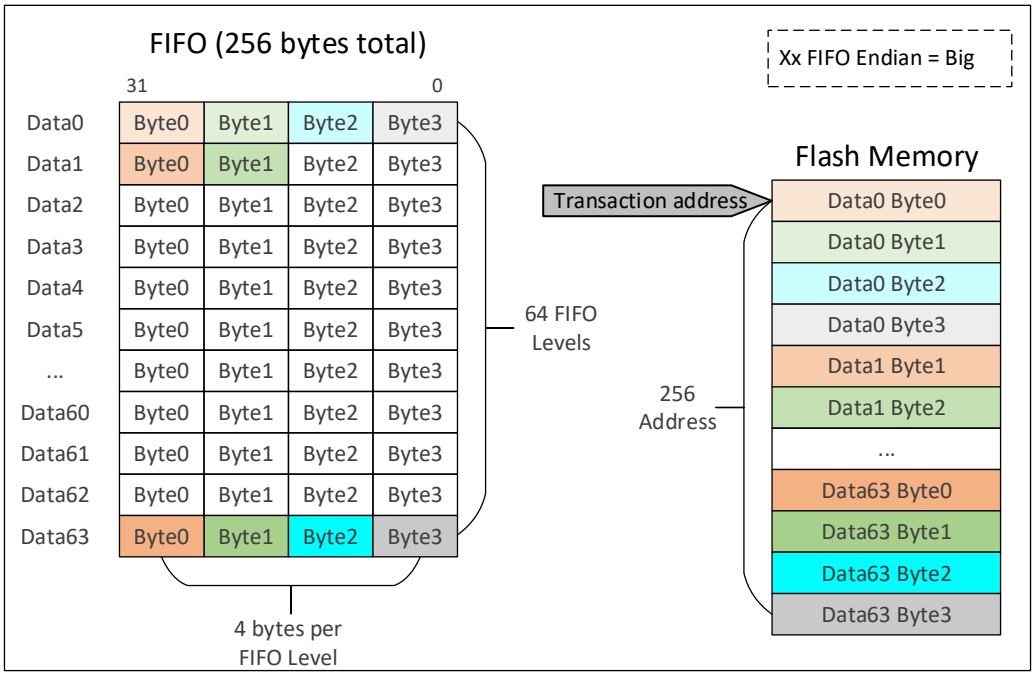

Figure 2.5. FIFO and Flash Memory Relationship (Big Endian)

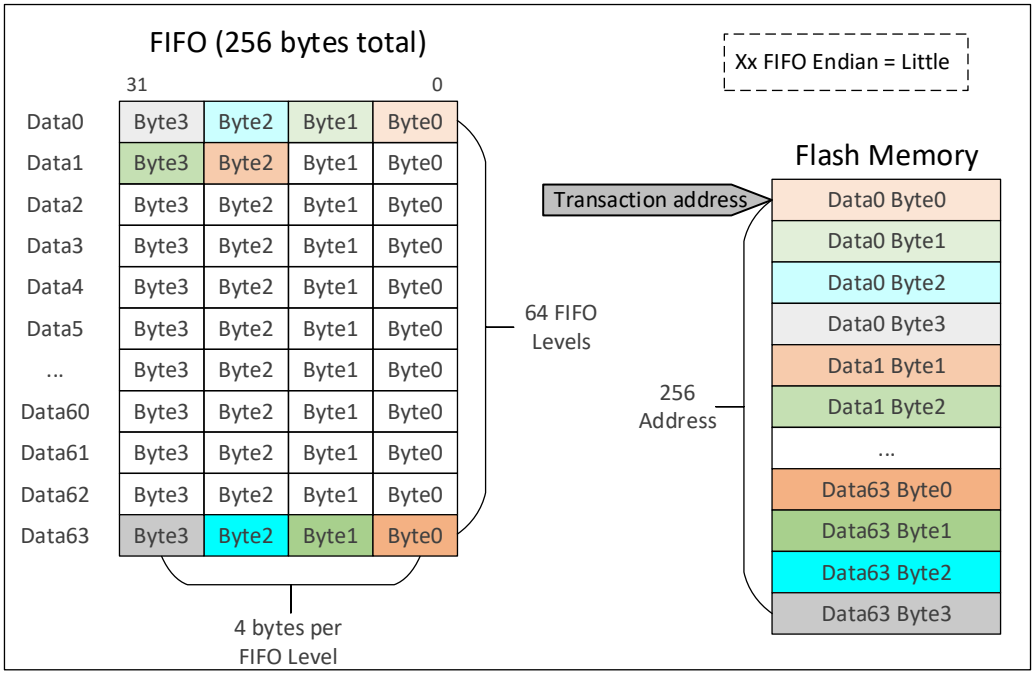

Figure 2.6. FIFO and Flash Memory Relationship (Little Endian)

2.6.2. Write Access Operation 2

The following is the recommended procedure for executing a write transaction with additional monitoring using the Internal Flash Controller IP:

Note: It is assumed that the target partition has been properly erased prior to initiating the write operation.

1. Set INT_ENABLE.done_en to 1 to enable the interrupt that indicates sending command to flash is done. Additionally, if applicable, set INT_ENABLE.wr_fifo_almost_full/almost_empty/empty/full to 1 depending on your monitoring scheme.
2. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
   - TRANSACTION_ADDR.transaction_partition
   - TRANSACTION_ADDR.transaction_offset
3. Configure the TRANSACTION_CTRL register to specify the type of transaction – *write access* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
   - TRANSACTION_CTRL.transaction_type – Set to *write access* (2'b00).
   - TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to write. The maximum number of bytes for a write transaction is 256 so you may set this up to 'd255.

   Ensure that the two key fields listed above are properly set.
4. Flush the WR FIFO by writing to the FIFO_CTRL register. For more information, refer to FIFO_CTRL Register section.
5. Depending on the monitoring scheme and the WR FIFO depth setting, follow the appropriate steps to write data to the WR_FIFO_DATA register repeatedly; up to N times, where N is determined by two factors:
   - Configurated FIFO depth of WR FIFO
   - TRANSACTION_CTRL.transaction_bytes_m1 setting

   Each write to WR_FIFO_DATA transfers four data bytes. A single write transaction can accommodate a maximum of 256 data bytes, which corresponds to 64 writes to the WR_FIFO_DATA register.

   Note: When writing to the WR_FIFO_DATA register, ensure that the WR FIFO is not full to avoid data loss or write errors. You may optionally check the FIFO_STATUS.wr_fifo_full flag before each write operation to the WR_FIFO_DATA register. If FIFO_STATUS.wr_fifo_full is set to 1, defer the write operation until space becomes available.

   The following are the monitoring schemes:
   - Polling-Based Monitoring

     Note: After each write to WR_FIFO_DATA, check the relevant FIFO_STATUS flag based on the desired monitoring behavior.
     - If monitoring FIFO_STATUS.wr_fifo_almost_empty or wr_fifo_empty:
       a. Write data to the WR_FIFO_DATA register as long as the flag remains set.
       b. Pause writing to WR_FIFO_DATA once the flag is cleared.
       c. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.
       d. Resume writing once the wr_fifo_almost_empty or wr_fifo_empty is set.
       e. Repeat these steps until all data has been written.
     - If monitoring FIFO_STATUS.wr_fifo_almost_full or wr_fifo_full:
       a. Write data to the WR_FIFO_DATA register as long as the flag remains clear.
       b. Pause writing to WR_FIFO_DATA once the flag is set.
       c. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.
       d. Resume writing once the wr_fifo_almost_full or wr_fifo_full flag is cleared.
       e. Repeat these steps until all data has been written.
   - Interrupt-Based Monitoring

     Note: When interrupts are enabled for wr_fifo_status flags, after each write to WR_FIFO_DATA, monitor the int_o signal. For interrupt-driven writes, use polling to initially fill the FIFO until it reaches the configured

threshold (full or almost full). After the initial fill, subsequent writes can be managed by the ISR based on the FIFO status interrupts. This ensures efficient startup without waiting for interrupts when the FIFO is empty.

a. Write data to the WR_FIFO_DATA register until int_o asserts.
b. Upon int_o assertion, check and clear the INT_STATUS register to acknowledge the interrupt.
c. Based on the flag set, pause writing to WR_FIFO_DATA.
d. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.
e. Resume writing once the FIFO_STATUS.wr_fifo_almost_empty or wr_fifo_empty flag is cleared.
f. Repeat Steps a through d until all data has been written.

6. After all data has been written, poll FIFO_STATUS.done until it is set to 1. This indicates that the command has been successfully sent to the flash and the controller can queue the next command before ISR processes are complete. It does not indicate completion of the flash operation.
7. After polling FIFO_STATUS.done, monitor the int_o signal, enabled in Step 1 (associated with INT_ENABLE.done_en), for assertion. This interrupt confirms that the flash operation has completed.
8. Poll INT_STATUS.done_int until it is set to 1 to verify the interrupt, then clear the INT_STATUS register. For more information, refer to the INT_STATUS Register section.
9. Optional: Initiate the next operation (erase, write access, or read access).

2.6.3. Read Access Operation 1

The following is the recommended procedure for executing a read transaction using the Internal Flash Controller IP:

1. Set INT_ENABLE.done_en to 1 to enable the interrupt that indicates sending command to flash is done.
2. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section.
   - TRANSACTION_ADDR.transaction_partition
   - TRANSACTION_ADDR.transaction_offset
3. Configure the TRANSACTION_CTRL register to specify the type of transaction – *read access* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
   - TRANSACTION_CTRL.transaction_type – Set to *read access* (2'b01).
   - TRANSACTION_CTRL.access_mode – Set to either QUAD or STANDARD depending on the configured device.
   - TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to read.
   - TRANSACTION_CTRL.read_mode – Set to either fast read or normal read.

   Ensure that the four key fields listed above are properly set.
4. Flush the RD FIFO by writing to the FIFO_CTRL register. For more information, refer to the FIFO_CTRL Register section.
5. Initiate the transaction by setting the TRANSACTION_CTRL.start_transaction field to 1. For more information, refer to the TRANSACTION_CTRL Register section.
6. Poll FIFO_STATUS.rd_fifo_empty until it is 0.
7. Read data from the RD_FIFO_DATA register.
8. Repeat Steps 6 and 7 until the specified number of bytes, as defined by the TRANSACTION_CTRL.transaction_bytes_m1 setting, has been read.
9. Poll FIFO_STATUS.done until it is set to 1. This indicates that the command has been successfully sent to the flash and the controller can queue the next command before ISR processes are complete. It does not indicate completion of the flash operation.
10. After polling FIFO_STATUS.done, monitor the int_o signal, enabled in Step 1, for assertion. This interrupt confirms that the flash operation has completed.
11. Poll INT_STATUS.done_int until it is set to 1 to verify the interrupt, then clear the INT_STATUS register. For more information, refer to the INT_STATUS Register section.
12. Optional: Initiate the next operation (erase, write access, or read access).

Note: In an empty RD FIFO, the first four bytes read from the flash memory are pushed as Data0 in the RD FIFO. The second four bytes read are pushed as Data1, followed by subsequent sets as Data2, Data3, and so on. Data to be read must be accessed through the RD_FIFO_DATA register. Each valid read access to the RD_FIFO_DATA register retrieves four bytes of data.

The address of the transaction is computed by the IP using the TRANSACTION_ADDR settings. Start the read access to the flash memory by setting TRANSACTION_CTRL.start_transaction to *1* with TRANSACTION_CTRL.transaction_type = 2'b01. The data is read from the flash memory starting from the computed transaction address. The next byte is read from transaction address + 1 until the total number of bytes to be read is completed. The total number of bytes to be read is set in the TRANSACTION_CTRL.transaction_bytes_m1 field.

2.6.4. Read Access Operation 2

The following is the recommended procedure for executing a read transaction with additional monitoring using the Internal Flash Controller IP:

1. Set INT_ENABLE.done_en to 1 to enable the interrupt that indicates sending command to flash is done. Additionally, if monitoring the INT_STATUS.rd_fifo_full flag, set INT_ENABLE.rd_fifo_full to 1 to enable the corresponding interrupt.
2. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
   - TRANSACTION_ADDR.transaction_partition
   - TRANSACTION_ADDR.transaction_offset
3. Configure the TRANSACTION_CTRL register to specify the type of transaction – *read access* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
   - TRANSACTION_CTRL.transaction_type – Set to *read access* (2'b01).
   - TRANSACTION_CTRL.access_mode – Set to either QUAD or STANDARD depending on the configured device.
   - TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to read.
   - TRANSACTION_CTRL.read_mode – Set to either fast read or normal read.

   Ensure that the four key fields listed above are properly set.
4. Flush the RD FIFO by writing to the FIFO_CTRL register. For more information, refer to the FIFO_CTRL Register section.
5. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.
6. Depending on the monitoring scheme, follow the appropriate steps to read data from the RD_FIFO_DATA register.
   - If using INT_STATUS.rd_fifo_full:
     a. Wait for the int_o signal to assert. This indicates that the INT_ENABLE.rd_fifo_full interrupt condition has been met.
     b. Check and clear the INT_STATUS.rd_fifo_full flag to acknowledge the interrupt and prepare for the subsequent events.
     c. Read the data from the RD_FIFO_DATA register.

        Check that FIFO_STATUS.rd_fifo_empty is 0 before reading from the RD_FIFO_DATA register.
        - Optional: If the number of bytes to read has already been determined based on the configured RD_FIFO_DEPTH, use that to control the read loop.
        - Important: Do not attempt to read from RD_FIFO_DATA if FIFO_STATUS.rd_fifo_empty is set to 1 as this indicates that the FIFO is empty.

     d. Repeat Steps a through c until the total number of bytes specified by the TRANSACTION_CTRL.transaction_bytes_m1 field has been read.
     e. Poll FIFO_STATUS.done until it is set to 1. This indicates that the command has been successfully sent to the flash and the controller can queue the next command.
     f. Wait for the int_o signal as enabled in Step 1 (associated with INT_ENABLE.done_en) to assert. This indicates that the corresponding interrupt condition has been met.

g. Monitor INT_STATUS.done_int until it is set to 1. This indicates the completion of the operation. Once set, clear the INT_STATUS register to acknowledge the interrupt. For more information, refer to the INT_STATUS Register section.

h. Optional: Initiate the next operation (erase, write access, or read access).

- If polling FIFO_STATUS.rd_fifo_full:

a. Poll FIFO_STATUS.rd_fifo_full until it is set to 1. This indicates that the read FIFO is full and ready for data retrieval.

b. Read the data from the RD_FIFO_DATA register.

Check that FIFO_STATUS.rd_fifo_empty is 0 before reading from the RD_FIFO_DATA register.

- Optional: If the number of bytes to read has already been determined based on the configured RD_FIFO_DEPTH, use that to control the read loop.
- Important: Do not attempt to read from RD_FIFO_DATA if FIFO_STATUS.rd_fifo_empty is set to 1 as this indicates that the FIFO is empty and may result in invalid data access.

c. Repeat Steps a and b until the total number of bytes specified by the TRANSACTION_CTRL.transaction_bytes_m1 field has been read.

d. Poll FIFO_STATUS.done until it is set to 1. This indicates that the command has been successfully sent to the flash and the controller can queue the next command.

e. For interrupt-driven completion, wait for the int_o signal as enabled in Step 1 (associated with INT_ENABLE.done_en) to assert. This indicates that the corresponding interrupt condition has been met. For pure polling scenarios, skip this step because completion is confirmed by FIFO_STATUS.done and INT_STATUS.done_int.

f. Monitor INT_STATUS.done_int until it is set to 1. This indicates the completion of the operation. Once set, clear the INT_STATUS register to acknowledge the interrupt. For more information, refer to the INT_STATUS Register section.

g. Optional: Initiate the next operation (erase, write access, or read access).

2.6.5. Erase Operation

The following is the recommended procedure for executing an erase transaction using the Internal Flash Controller IP:
Note: The default value of flash upon erase transaction is all 1s.

1. Set INT_ENABLE.done_en to 1 to enable the interrupt that indicates sending command to flash is done.
2. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
   - TRANSACTION_ADDR.transaction_partition
   - TRANSACTION_ADDR.transaction_offset
3. Configure the TRANSACTION_CTRL register to specify the type of transaction – *erase* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section.
   - TRANSACTION_CTRL.transaction_type – Set to *erase* (2'b10).
   - TRANSACTION_CTRL.erase_partition_size – Set to either 64 kB (2'b00) or whole flash (2'b01).

   Ensure that the two key fields listed above are properly set.

   Note: The erase function sets the whole partition to the erased state, starting from the transaction address set in the TRANSACTION_ADDR register. The partition size to be erased is set in TRANSACTION_CTRL.erase_partition size. The transaction address must be aligned to the partition size to be erased. The memory address to be written must be in the erased state prior to performing a write access.
4. Start the erase function to the flash memory by setting TRANSACTION_CTRL.start_transaction to 1.
5. Poll FIFO_STATUS.done until it is set to 1.
6. Wait for int_o signal enabled in Step 1 to assert.
7. Poll INT_STATUS.done until it set to 1, then clear the INT_STATUS register. For more information, refer to the INT_STATUS Register section.

2.6.6. Normal Operation (Erase Write Read)

The following is the recommended procedure for performing an erase transaction, followed by a write transaction and then a read transaction, using the Internal Flash Controller IP:

1. Set INT_ENABLE.done_en to 1 to enable the interrupt that indicates sending command to flash is done.
2. Perform erase transaction.
   a. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
      - TRANSACTION_ADDR.transaction_partition
      - TRANSACTION_ADDR.transaction_offset
   b. Configure the TRANSACTION_CTRL register to specify the type of transaction – *erase* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
      - TRANSACTION_CTRL.transaction_type – Set to *erase* (2'b10).
      - TRANSACTION_CTRL.erase_partition_size – Set to either 64 kB (2'b00) or whole flash (2'b01).

      Ensure that the two key fields listed above are properly set.

      Note: The erase function sets the whole partition to the erased state, starting from the transaction address set in the TRANSACTION_ADDR register. The partition size to be erased is set in TRANSACTION_CTRL.erase_partition size. The transaction address must be aligned to the partition size to be erased. The memory address to be written must be in the erased state prior to performing a write access.
   c. Start the erase function to the flash memory by setting TRANSACTION_CTRL.start_transaction to 1.
   d. Poll FIFO_STATUS.done until it is set to 1.
   e. Wait for the int_o signal enabled in Step 1 to assert.
   f. Poll INT_STATUS.done_int until it set to 1, then clear the INT_STATUS register. For more information, refer to the INT_STATUS Register section.
3. Perform write transaction.
   a. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
      - TRANSACTION_ADDR.transaction_partition
      - TRANSACTION_ADDR.transaction_offset
   b. Configure the TRANSACTION_CTRL register to specify the type of transaction – *write access* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
      - TRANSACTION_CTRL.transaction_type – Set to *write access* (2'b00).
      - TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to write. The maximum number of bytes for a write transaction is 256 so you may set this up to 'd255.

      Ensure that the two key fields listed below are properly set.
   c. Flush the WR FIFO by writing to the FIFO_CTRL register. For more information, refer to FIFO_CTRL Register section.
   d. Write data to the WR_FIFO_DATA register repeatedly; up to N times, where N is determined by two factors:
      - Configured FIFO depth of WR FIFO
      - TRANSACTION_CTRL.transaction_bytes_m1 setting

      Each write to WR_FIFO_DATA transfers four data bytes. A single write transaction can accommodate a maximum of 256 data bytes, which corresponds to 64 writes to the WR_FIFO_DATA register.

      Note: When writing to the WR_FIFO_DATA register, ensure that the WR FIFO is not full to avoid data loss or write errors. You may optionally check the FIFO_STATUS.wr_fifo_full flag before each write operation to the WR_FIFO_DATA register. If FIFO_STATUS.wr_fifo_full is set to 1, defer the write operation until space becomes available.
   e. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.

f. Poll FIFO_STATUS.done until it is set to 1. This indicates that the command has been successfully sent to the flash and the controller can queue the next command.

g. Wait for the int_o signal enabled in Step 1 to assert.

h. Poll INT_STATUS.done_int until it set to 1, then clear the INT_STATUS register. For more information, refer to the INT_STATUS Register section.

4. Perform read transaction.

a. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
- TRANSACTION_ADDR.transaction_partition
- TRANSACTION_ADDR.transaction_offset

b. Configure the TRANSACTION_CTRL register to specify the type of transaction – *read access* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
- TRANSACTION_CTRL.transaction_type – Set to *read access* (2'b01).
- TRANSACTION_CTRL.access_mode – Set to either QUAD or STANDARD depending on the configured device.
- TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to read.
- TRANSACTION_CTRL.read_mode – Set to either fast read or normal read.

Ensure that the four key fields listed above are properly set.

c. Flush the RD FIFO by writing to the FIFO_CTRL register. For more information, refer to the FIFO_CTRL Register section.

d. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.

e. Poll FIFO_STATUS.rd_fifo_empty until it is 0.

f. Read data from the RD_FIFO_DATA register.

g. Repeat Steps e and f until the specified number of bytes, as defined by the TRANSACTION_CTRL.transaction_bytes_m1 setting, has been read.

h. Poll FIFO_STATUS.done until it is set to 1. This indicates that the command has been successfully sent to the flash and the controller can queue the next command.

i. Wait for the int_o signal enabled in Step 1 to assert.

j. Poll INT_STATUS.done_int until it set to 1, then clear the INT_STATUS register. For more information, refer to the INT_STATUS Register section.

2.6.7. Normal Operation (Erase Write Read) with TRANSACTION_STATUS

The following is the recommended procedure for performing an erase transaction, followed by a write transaction and then a read transaction, using the Internal Flash Controller IP while monitoring the TRANSACTION_STATUS.idle field to ensure proper sequencing and completion of each operation.

1. Poll TRANSACTION_STATUS.idle until it set to 1. This indicates that the controller is idle and ready for the next operation. For more information, refer to the TRANSACTION_STATUS Register section.
2. Perform erase transaction.

a. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
- TRANSACTION_ADDR.transaction_partition
- TRANSACTION_ADDR.transaction_offset

b. Configure the TRANSACTION_CTRL register to specify the type of transaction – *erase* in this case. For more information, refer to the TRANSACTION_CTRL Register section:
- TRANSACTION_CTRL.transaction_type – Set to *erase* (2'b10).
- TRANSACTION_CTRL.erase_partition_size – Set to either 64 kB (2'b00) or whole flash (2'b01).

Ensure that the two key fields listed above are properly set.

Note: The erase function sets the whole partition to the erased state, starting from the transaction address set in the TRANSACTION_ADDR register. The partition size to be erased is set in TRANSACTION_CTRL.erase_partition size. The transaction address must be aligned to the partition size to be erased. The memory address to be written must be in the erased state prior to performing a write access.

c. Start the erase function to the flash memory by setting TRANSACTION_CTRL.start_transaction to 1.

d. Poll TRANSACTION_STATUS.idle until it set to 1. This indicates that the controller is idle and ready for the next operation. For more information, refer to the TRANSACTION_STATUS Register section.

3. Perform write transaction.

a. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the following fields in the TRANSACTION_ADDR Register section:
- TRANSACTION_ADDR.transaction_partition
- TRANSACTION_ADDR.transaction_offset

b. Configure the TRANSACTION_CTRL register to specify the type of transaction – *write access* in this case. For more information, refer to the following fields in the TRANSACTION_CTRL Register section:
- TRANSACTION_CTRL.transaction_type – Set to *write access* (2'b00).
- TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to write. The maximum number of bytes for a write transaction is 256 so you may set this up to 'd255.

Ensure that the two key fields listed above are properly set.

c. Flush the WR FIFO by writing to the FIFO_CTRL register. For more information, refer to FIFO_CTRL Register section.

d. Write data to the WR_FIFO_DATA register repeatedly; up to N times, where N is determined by two factors:
- Configured FIFO depth for WR FIFO
- TRANSACTION_CTRL.transaction_bytes_m1 setting

Each write to WR_FIFO_DATA transfers four data bytes. A single write transaction can accommodate a maximum of 256 data bytes, which corresponds to 64 writes to the WR_FIFO_DATA register.

Note: When writing to the WR_FIFO_DATA register, ensure that the write FIFO is not full to avoid data loss or write errors. You may optionally check the FIFO_STATUS.wr_fifo_full flag before each write operation to the WR_FIFO_DATA register. If FIFO_STATUS.wr_fifo_full is set to 1, defer the write operation until space becomes available.

e. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.

f. Poll TRANSACTION_STATUS.idle until it set to 1. This indicates that the controller is idle and ready for the next operation. For more information, refer to the TRANSACTION_STATUS Register section.

4. Perform read transaction.

a. Configure the TRANSACTION_ADDR register to specify the target partition and address for the access operation. For more information, refer to the TRANSACTION_ADDR Register section:
- TRANSACTION_ADDR.transaction_partition
- TRANSACTION_ADDR.transaction_offset

b. Configure the TRANSACTION_CTRL register to specify the type of transaction – *read access* in this case. For more information, refer to the TRANSACTION_CTRL Register section:
- TRANSACTION_CTRL.transaction_type – Set to *read access* (2'b01).
- TRANSACTION_CTRL.access_mode – Set to either QUAD or STANDARD depending on the configured device.
- TRANSACTION_CTRL.transaction_bytes_m1 – Set the number of bytes to read.
- TRANSACTION_CTRL.read_mode – Set to either fast read or normal read.

Ensure that the four key fields listed above are properly set.

c. Flush the RD FIFO by writing to the FIFO_CTRL register. For more information, refer to the FIFO_CTRL Register section.

d. Initiate the transaction by setting TRANSACTION_CTRL.start_transaction to 1. For more information, refer to the TRANSACTION_CTRL Register section.

e. Poll FIFO_STATUS.rd_fifo_empty until it is 0.

f. Read data from the RD_FIFO_DATA register.

g. Repeat steps e and f until the specified number of bytes, as defined by the TRANSACTION_CTRL.transaction_bytes_m1 setting, has been read.

h. Poll TRANSACTION_STATUS.idle until it set to 1. This indicates that the controller is idle and ready for the next operation. For more information, refer to the TRANSACTION_STATUS Register section.

3. IP Parameter Description

The configurable attributes of the Internal Flash Controller IP for MachXO5-NX are shown in the following tables. You can configure the IP by setting the attributes accordingly in the IP Catalog Module/IP wizard of the Lattice Propel Builder software.

Wherever applicable, default values are in bold.

3.1. General

Table 3.1. General Attributes

Attribute	Selectable Values	Description
General		
Interface	**AHBL**, APB	Specifies the type of interface for register access. Unselected interface is not available in the generated IP.
Interface Data Width	**32**	Specifies the data width of the selected interface. This also the value of the FIFO width. Informational only.
Interface Address Width	**32**	Specifies the address width of the selected interface Informational only.
RD FIFO Settings		
RD FIFO Width	**32**	Specifies the bit width of each data word of the internal RD FIFO. Value is the same as Interface Data Width. Informational only.
FIFO Depth	4, 8, 16, 32, **64**	Specifies the number of FIFO levels. Only power of 2 value is allowed.
Implementation of RD FIFO	EBR, **LUT**	Selects the FPGA resource that is used to implement the FIFO: EBR or LUT.
Endian of RD FIFO	**Little**, Big	Sets the endianness of RD FIFO. Refer to Figure 2.5 and Figure 2.6.
RD FIFO Almost Full Flag	1 to (RD FIFO Depth – 1), **59**	Specifies the threshold value for signaling that the FIFO is almost full.
RD FIFO Almost Empty Flag	1 to RD FIFO Almost Full Flag, **4**	Specifies the threshold value for signaling that the FIFO is almost empty.
WR FIFO Settings		
WR FIFO Width	**32**	Specifies the bit width of each data word of the internal WR FIFO. Value is the same as Interface Data Width. Informational only.
FIFO Depth	4, 8, 16, 32, **64**	Specifies the number of FIFO levels. Only power of 2 value is allowed.
Implementation of WR FIFO	EBR, **LUT**	Selects the FPGA resource that is used to implement the FIFO: EBR or LUT.
Endian of WR FIFO	**Little**, Big	Sets the endianness of WR FIFO. Refer to Figure 2.5 and Figure 2.6.
WR FIFO Almost Full Flag	1 to (WR FIFO Depth – 1), **59**	Specifies the threshold value for signaling that the FIFO is almost full.
WR FIFO Almost Empty Flag	1 to WR FIFO Almost Full Flag, **4**	Specifies the threshold value for signaling that the FIFO is almost empty.

3.2. CFG0/1/2 and UFM0/1/2 Settings

Table 3.2. CFG0/1/2 and UFM0/1/2 Attributes

Attribute	Selectable Values[1]	Description
CFG0/1/2 and UFM0/1/2 Settings		
Use CFGx with size (blocks)	A: **0**, 11, 15 B: **0**, 21, 28 C: **0**, 33, 48	Selects use of the CFGx partition when *Use CFGx with size (blocks)* > 0. There are two partition size options available. Do not access the partition if the partition size is set to 0.
CFGx Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use CFGx with size (blocks)* > 0.
CFGx Initial Value File Path	—	Allows browsing of memory files to select a custom memory file. Applicable when *CFGx Initial Value Format* = HEXFILE or BINFILE.
Use UFMx with size (blocks)	UFM0 A: 0, **4** B: 0, **7** C: 0, **15** UFM1/2 A: **0**, 4 B: **0**, 7 C: **0**, 15	Selects use of the UFMx partition when *Use UFMx with size (blocks)* > 0. If the partition size of the corresponding CFG (*Use CFGx with size (blocks)*) is set to the maximum, this parameter is automatically set to 0. Do not access the partition if the partition size is set to 0. Applicable when *Use CFGx with size (blocks)* < maximum.
UFMx Initial Value Format[2]	**ALL 1**, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use UFMx with size (blocks)* > 0.
UFMx Initial Value File Path	—	Allows browsing of memory files to select a custom memory file. Applicable when *UFMx Initial Value Format* = HEXFILE or BINFILE.

Notes:

1. *A* applies to LFMXO5-25; *B* applies to LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T; *C* applies to LFMXO5-55T and LFMXO-100T.
2. Initial value options create memory initialization files for convenience. These options are intended for optional customer-driven simulation and do not indicate official simulation support.

3.3. USERDATA0 to USERDATA8 Settings

Table 3.3. USERDATA0 to USERDATA8 Attributes

Attribute	Selectable Values[1]	Description
Use USERDATA0 to USERDATA8	Checked, **Unchecked**	Selects use of the USERDATA0 to USERDATA8 partitions when checked. Do not access the partitions if unchecked. If checked, at the very least, the USERDATA0 partition must be used. Use of USERDATA1 to USERDATA8 requires the preceding USERDATAx to have at least one block used.
USERDATA0 Settings		

Use USERDATA0 with size (blocks)	A: 1–17
	B: 1–6
	C: 1–110
		Selects the size of the USERDATA0 partition (in blocks).
		The rules of USERDATA block size selection are as follows:
		• Selecting *0* setting means that the partition is not used. This is not applicable for USERDATA0 because USERDATA0 must be used when *Use USERDATA0 to USERDATA8* = Checked.
		• If a partition is set to 0, succeeding partitions cannot be used. For example, if USERDATA3 is set to 0, USERDATA4 to USERDATA8 are automatically set to 0.
		• The total number of blocks must not exceed:
		  • LFMXO5-25: 17 blocks
		  • LFMXO5-35/35T/65/56T: 6 blocks for USERDATA0 and 36 blocks for USERDATA1 to USERDATA8.
		  • LFMXO5-55T/100T: 110 blocks
		Applicable when *Use USERDATA0 to USERDATA8* = Checked.

USERDATA0 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA0 to USERDATA8* = Checked.
USERDATA0 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA0 Initial Value Format* = HEXFILE or BINFILE.
USERDATA1 Settings		
Use USERDATA1 with size (blocks)	A: 0–17 B: 1–36 C: 0–110	Selects the size of the USERDATA1 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Note that for LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T devices, USERDATA1 must be at least 1 block. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA1 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA1 with size (blocks)* ≥ 1.
USERDATA1 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA1 Initial Value Format* = HEXFILE or BINFILE.

Attribute	Selectable Values[1]	Description
USERDATA2 Settings		
Use USERDATA2 with size (blocks)	A: 0–17 B: 0–36 C: 0–110	Selects the size of the USERDATA2 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA2 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA2 with size (block)* ≥ 1.
USERDATA2 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA2 Initial Value Format* = HEXFILE or BINFILE.
USERDATA3 Settings		
Use USERDATA3 with size (blocks)	A: 0–17 B: 0–36 C: 0–110	Selects the size of the USERDATA3 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA3 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA3 with size (blocks)* ≥ 1.
USERDATA3 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA3 Initial Value Format* = HEXFILE or BINFILE.
USERDATA4 Settings		
Use USERDATA4 with size (blocks)	A: 0–17 B: 0–36 C: 0–110	Selects the size of the USERDATA4 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA4 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA4 with size (blocks)* ≥ 1.
USERDATA4 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA4 Initial Value Format* = HEXFILE or BINFILE.
USERDATA5 Settings		
Use USERDATA5 with size (blocks)	A: 0–17 B: 0–36 C: 0–110	Selects the size of the USERDATA5 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA5 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA5 with size (blocks)* ≥ 1.
USERDATA5 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA5 Initial Value Format* = HEXFILE or BINFILE.

Attribute	Selectable Values[1]	Description
USERDATA6 Settings		
Use USERDATA6 with size (blocks)	A: 0–17 B: 0–36 C: 0–110	Selects the size of the USERDATA6 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA6 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA6 with size (blocks)* ≥ 1.
USERDATA6 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA6 Initial Value Format* = HEXFILE or BINFILE.
USERDATA7 Settings		
Use USERDATA7 with size (blocks)	A: 0–17 B: 0–36 C: 0–110	Selects the size of the USERDATA7 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA7 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA7 with size (blocks)* ≥ 1.
USERDATA7 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA7 Initial Value Format* = HEXFILE or BINFILE.
USERDATA8 Settings		
Use USERDATA8 with size (blocks)	A: 0–17 B: 0–36 C: 0–110	Selects the size of the USERDATA8 partition (in blocks). Refer to the description of *Use USERDATA0 with size (blocks)* for the rules of USERDATA block size selection. Applicable when *Use USERDATA0 to USERDATA8* = Checked. Available options change depending on the selected value of other USERDATAx partitions.
USERDATA8 Initial Value Format[2]	ALL 1, ALL 0, NONE, HEXFILE, BINFILE	Selects how to initialize the flash memory partition. Refer to Table 2.7 for descriptions of the available options. Applicable when *Use USERDATA8 with size (blocks)* ≥ 1.
USERDATA8 Initial Value File Path	—	Selects a custom memory file. Applicable when *USERDATA8 Initial Value Format* = HEXFILE or BINFILE.

Notes:

1. *A* applies to LFMXO5-25; *B* applies to LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T; *C* applies to LFMXO5-55T and LFMXO-100T.
2. Initial value options create memory initialization files for convenience. These options are intended for optional customer-driven simulation and do not indicate official simulation support.

3.4. Oscillator Setting

Table 3.4. Oscillator Attributes

Attribute	Selectable Values	Description
Instantiate Internal Oscillator	**Checked**, Unchecked	Instantiates the internal oscillator when checked.
Use HFCLK for LMMI CLK	Checked, **Unchecked**	Uses the HLCK output of the internal oscillator as LMMI CLK when checked. clk_o is available when this attribute is checked. The clock output should be used to sync the LMMI transactions. Applicable when *Instantiate Internal Oscillator* = Checked.
Frequency of the internal LMMI CLK (MHz)	**50.0**, 45.0, 40.909, 30.0	Selects the frequency of the HFCLK. Applicable when *Use HFCLK for LMMI CLK* = Checked

3.5. Access Mode

Table 3.5. Access Mode Attribute

Attribute	Selectable Values	Description
Read Access Mode	**STANDARD**, QUAD	Selects standard or quad access mode. QUAD is only applicable when LFMXO5-35, LFMXO5-35T, LFMXO5-55T, LFMXO5-65, LFMXO5-65T, or LFMXO5-100T is used. Default value is STANDARD.

4. Signal Description

This section describes the Internal Flash Controller IP for MachXO5-NX ports.

Table 4.1. Internal Flash Controller Ports

Port	Type	Description
System Clock and Reset		
clk_i[1]	Input	Clock signal input Available when *Use HFCLK for LMMI CLK* = Unchecked.
clk_o	Output	Clock signal output Available when *Use HFCLK for LMMI CLK* = Checked.
lmmi_clk_i[1]	In	LMMI clock signal input Available when *Instantiate Internal Oscillator* = Unchecked.
lmmi_resetn_i	In	LMMI reset signal input Available when *Instantiate Internal Oscillator* = Unchecked.
reset_n_i	Input	Active low reset signal
Interrupt		
int_o	Output	Interrupt signal Initial value after reset is tied to low.
AHB-Lite Interface[2]		
ahbl_hsel_i	Input	AHB-L select signal
ahbl_hready_i	Input	AHB-L master ready signal
ahbl_haddr_i[x:0]	Input	AHB-L address signal **Note:** x = *Interface Address Width* – 1
ahbl_hburst_i[2:0]	Input	AHB-L burst signal Indicates the number of transfers in a burst. **Note:** Only single burst is supported (b000).
ahbl_hsize_i[2:0]	Input	AHB-L size signal Indicates the size of the transfer.
ahbl_hprot_i[3:0]	Input	AHB-L protection control signal
ahbl_htrans_i[1:0]	Input	AHB-L transfer type signal Indicates the transfer type.
ahbl_hwrite_i	Input	AHB-L transfer direction signal 0 – Read 1 – Write
ahbl_hwdata_i[x:0]	Input	AHB-L write data signal **Note:** x = *Interface Data Width* – 1
ahbl_hreadyout_o	Output	AHB-L subordinate ready signal Used to extend AHB-L transfer. Initial value after reset is tied to high.
ahbl_hrdata_o[x:0]	Output	AHB-L read data signal **Note:** x = *Interface Data Width* – 1 Initial value after reset is tied to low.
ahbl_hresp_o	Output	AHB-L transfer response signal 0 – OKAY 1 – ERROR Initial value after reset is tied to low.
APB Interface[2]		
apb_psel_i	Input	APB select signal Indicates that the target device is selected, and a data transfer is required.
apb_paddr_i[x:0]	Input	APB address signal **Note:** x = *Interface Address Width* – 1

Port	Type	Description
apb_pwdata_i[x:0]	Input	APB write data signal **Note:** x = *Interface Data Width* – 1
apb_pwrite_i	Input	APB write signal 0 – Read 1 – Write
apb_penable_i	Input	APB enable signal Indicates the second and subsequent cycles of an APB transfer.
apb_pready_o	Output	APB ready signal Indicates transfer completion. Completer uses this signal to extend an APB transfer. Initial value after reset is tied to low.
apb_pslverr_o	Output	APB error signal Initial value after reset is tied to low.
apb_prdata_o[x:0]	Output	APB read data signal **Note:** x = *Interface Data Width* – 1 Initial value after reset is tied to low.

Notes:

1. The flash memory clock source can either be lmmi_clk_i or clk_i depending on the GUI setting. When the clock being delivered to the flash memory is equal to or greater than 25 MHz, it is mandatory by design that CONFIG_IOSLEW be set to FAST. This guarantees proper operation of the internal flash. Failure to comply might cause configuration related issues resulting in the device not being able to boot from its internal flash or issues in operation specifically during the read sequence.
2. Only one of the two interfaces is available at any one time as selected by the *Interface* attribute. Refer to AMBA 3 AHB-Lite Protocol V1.0 and AMBA 3 APB Protocol V1.0 for more information.

5. Register Description

5.1. Overview

This section defines the registers of the Internal Flash Controller IP for MachXO5-NX.

Table 5.1 defines the register access types. Table 5.2 lists the address map and specifies the registers available. These registers are accessible through the AHB-Lite or APB interface.

Table 5.1. Register Access Types

Access Type	Access Type Abbreviation	Behavior on Read Access	Behavior on Write Access
Read only	RO	Returns register value	Ignores write access
Write only	WO	Returns 0	Updates register value
Read and write	RW	Returns register value	Updates register value
Read and write 1 to clear	RW1C	Returns register value	Writing 1'b1 on register bit clears the bit to 1'b0. Writing 1'b0 on register bit is ignored.
Reserved	RSVD	Returns 0	Ignores write access

Table 5.2. Summary of Internal Flash Controller IP for MachXO5-NX Registers

Offset Address	Register Name	Description
0x00	UFM_CTRL_REG	Sets soft_reset function of the IP.
0x04	TRANSACTION_ADDR	Sets the transaction partition and offset to be accessed for write, read, or erase function.
0x08	WR_FIFO_DATA	Writing to this register pushes four bytes of data to WR_FIFO.
0x0C	RD_FIFO_DATA	Reading from this register pulls four bytes of data from RD_FIFO.
0x10	TRANSACTION_CTRL	Sets the type of transaction, number of bytes, access mode for read and write transactions, erase size for erase transactions, and read mode for read transactions.
0x14	FIFO_STATUS	Contains the status flags for both WR_FIFO and RD_FIFO.
0x18	FIFO_CTRL	Flushes WR_FIFO and RD_FIFO.
0x1C	INT_STATUS	Shows the interrupt status for transaction done and FIFO flags.
0x20	INT_ENABLE	Enables interrupts.
0x24	INT_SET	Forcibly asserts any interrupt for register debugging.
0x28	TRANSACTION_STATUS	Reading from this register triggers an internal polling sequence to check the flash status.

5.2. UFM_CTRL_REG Register

Table 5.3. UFM_CTRL_REG Register

Field	Name	Description	Access	Default
[31]	soft_reset	Writing 1 to this bit resets all the internal logic, flushes the FIFOs, and restores all registers to the default settings. Reset takes effect two clock cycles after the write transaction. Intended for error recovery.	WO	—
[30:0]	reserved	Reserved	RSVD	—

5.3. TRANSACTION_ADDR Register

Table 5.4. TRANSACTION_ADDR Register

Field	Name	Description	Access	Reset
[31:24]	transaction_partition	Selects the partition to access. Access to unused partitions based on the IP attributes is ignored. 0x00 – CFG0 0x01 – UFM0 0x02 – CFG1 0x03 – UFM1 0x04 – CFG2 0x05 – UFM2 0x06 – USERDATA0 0x07 – USERDATA1 0x08 – USERDATA2 0x09 – USERDATA3 0x0A – USERDATA4 0x0B – USERDATA5 0x0C – USERDATA6 0x0D – USERDATA7 0x0E – USERDATA8	RW	0x00
[23]	reserved	Reserved	RSVD	—
[22:0]	transaction_offset	Refer to the Transaction Offset Computation section.	RW	0x00_0000

5.4. WR_FIFO_DATA Register

Table 5.5. WR_FIFO_DATA Register

Field	Name	Description	Access	Reset
[31:24]	wr_data0	Writing to this register pushes four bytes of data to WR FIFO. The relationship between WR FIFO and the write access to the flash memory is discussed in the Programming Flow section.	WO	—
[23:16]	wr_data1		WO	—
[15:8]	wr_data2		WO	—
[7:0]	wr_data3		WO	—

Note: Ensure that a write command to the WR_FIFO_DATA register is not issued when the FIFO_STATUS.wr_fifo_full flag is asserted during a valid write transaction or access. Additionally, verify the available FIFO depth before issuing any write command. Refer to the Programming Flow section for the procedure to access the register correctly.

5.5. RD_FIFO_DATA Register

Table 5.6. RD_FIFO_DATA Register

Field	Name	Description	Access	Reset
[31:24]	rd_data0	Reading from this register retrieves four bytes of data from RD FIFO. The relationship between RD FIFO and the read access from the flash memory is discussed in the Programming Flow section.	RO	—
[23:16]	rd_data1		RO	—
[15:8]	rd_data2		RO	—
[7:0]	rd_data3		RO	—

Note: Ensure that a read command to the RD_FIFO_DATA register is not issued when the FIFO_STATUS.rd_fifo_empty flag is asserted during a valid read transaction or access. Additionally, consider the current FIFO depth before issuing any read command. Refer to the Programming Flow section for the procedure to access the register correctly.

5.6. TRANSACTION_CTRL Register

Table 5.7. TRANSACTION_CTRL Register

Field	Name	Description	Access	Reset
[31]	start_transaction[1]	Writing 1 to this bit starts the transaction. Read data returns 0.	WO	—
[30:16]	reserved	Reserved	RSVD	—
[15]	access_mode	Specifies the access mode to be used. Not applicable to the LFMXO5-25 device. The *Read Access Mode* attribute must be set to QUAD if you intend to use this bit. Otherwise, this bit is not supported and write access is ignored. 0 – STANDARD 1 – QUAD	RW	0b0
[14]	read_mode	0 – Normal read 1 – Fast read	RW	0b0
[13:12]	erase_partition_size	00 – 64 kB 01 – Whole flash 10 – Reserved 11 – Reserved	RW	0b0
[11:2]	transaction_bytes_m1	Specifies the number of bytes to write or read. *transaction_bytes_m1 = Number of bytes – 1* Note that *Number of bytes* must be divisible by 4. Write access settings range from 0 to 255. Read access settings range from 0 to 1023. 0 – 1 byte of write/read access 1 – 2 bytes of write/read access 2 – 3 bytes of write/read access … 255 – 256 bytes of write/read access (maximum for write access) … 1023 – 1024 bytes of read acccess (maximum for read access)	RW	0x0
[1:0]	transaction_type	Type of transaction to be performed. 00 – Write access 01 – Read access 10 – Erase 11 – Reserved	RW	0b00

Note:

1. After each start_transaction, a flash busy status polling sequence is initiated to ensure the flash device is ready before proceeding with the transaction.

5.7. FIFO_STATUS Register

Table 5.8. FIFO_STATUS Register

Field	Name	Description	Access	Reset
[31]	done	0 – Sending command to flash is not done. You must wait until done = 1 before queueing the next transaction. 1 – Sending command to flash is done. You can queue the next transaction. Note that if an invalid transaction_offset is set, done is asserted immediately after the transaction is started. The done status indicates whether the controller is done sending a command to the flash. It does not indicate that the transaction is done.	RO	0x0
[30:8]	reserved	Reserved	RSVD	—
[7]	wr_fifo_full	0 – WR FIFO is not full 1 – WR FIFO is full	RO	0x0
[6]	wr_fifo_almost_full	0 – WR FIFO has less than *WR FIFO Almost Full Flag* bytes 1 – WR FIFO has equal or more than *WR FIFO Almost Full Flag* bytes	RO	0x0
[5]	wr_fifo_almost_empty	0 – WR FIFO has more than *WR FIFO Almost Empty Flag* bytes 1 – WR FIFO has equal or less than *WR FIFO Almost Empty Flag* bytes	RO	0x1
[4]	wr_fifo_empty	0 – WR FIFO is not empty 1 – WR FIFO is empty	RO	0x1
[3]	rd_fifo_full	0 – RD FIFO is not full 1 – RD FIFO is full	RO	0x0
[2]	rd_fifo_almost_full	0 – RD FIFO has less than *RD FIFO Almost Full Flag* bytes 1 – RD FIFO has equal or more than *RD FIFO Almost Full Flag* bytes	RO	0x0
[1]	rd_fifo_almost_empty	0 – RD FIFO has more than *RD FIFO Almost Empty Flag* bytes 1 – RD FIFO has equal or less than *RD FIFO Almost Empty Flag* bytes	RO	0x1
[0]	rd_fifo_empty	0 – RD FIFO is not empty 1 – RD FIFO is empty	RO	0x1

Note: The FIFO_STATUS register is read-only. Clear operations, such as clear INT_STATUS, do not affect the FIFO_STATUS register.

5.8. FIFO_CTRL Register

Table 5.9. FIFO_CTRL Register

Field	Name	Description	Access	Reset
[31:2]	reserved	Reserved	RSVD	—
[1]	wr_fifo_flush	Writing 1 to this bit flushes contents of WR FIFO (reset read and write pointers).	WO	—
[0]	rd_fifo_flush	Writing 1 to this bit flushes contents of RD FIFO (reset read and write pointers).	WO	—

5.9. INT_STATUS Register

Table 5.10. INT_STATUS Register

Field	Name	Description	Access	Reset
[31]	done_int	Done interrupt (sending command to flash is done) Writing 1 to the bit clears the interrupt.	RW1C	0x0
[30:8]	reserved	Reserved	RSVD	—
[7]	wr_fifo_full_int	WR FIFO full interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0
[6]	wr_fifo_almost_full_int	WR FIFO almost full interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0
[5]	wr_fifo_almost_empty_int	WR FIFO almost empty interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0
[4]	wr_fifo_empty_int	WR FIFO empty interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0
[3]	rd_fifo_full_int	RD FIFO full interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0
[2]	rd_fifo_almost_full_int	RD FIFO almost full interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0
[1]	rd_fifo_almost_empty_int	RD FIFO almost empty interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0
[0]	rd_fifo_empty_int	RD FIFO empty interrupt Writing 1 to the bit clears the interrupt.	RW1C	0x0

Note: FIFO interrupts are triggered on the rising edge of the corresponding FIFO condition and stay asserted until cleared by writing 1 to the corresponding bit.

5.10. INT_ENABLE Register

Table 5.11. INT_ENABLE Register

Field	Name	Description	Access	Reset
[31]	done_en	Enable Done interrupt (sending command to flash is done) 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0
[30:8]	reserved	Reserved	RSVD	—
[7]	wr_fifo_full_en	Enable WR FIFO full interrupt 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0
[6]	wr_fifo_almost_full_en	Enable WR FIFO almost full interrupt 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0
[5]	wr_fifo_almost_empty_en	Enable WR FIFO almost empty interrupt 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0
[4]	wr_fifo_empty_en	Enable WR FIFO empty interrupt 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0
[3]	rd_fifo_full_en	Enable RD FIFO full interrupt 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0
[2]	rd_fifo_almost_full_en	Enable RD FIFO almost full interrupt 0 – Interrupt disabled	RW	0x0

Field	Name	Description	Access	Reset
		1 – Interrupt enabled		
[1]	rd_fifo_almost_empty_en	Enable RD FIFO almost empty interrupt 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0
[0]	rd_fifo_empty_en	Enable RD FIFO empty interrupt 0 – Interrupt disabled 1 – Interrupt enabled	RW	0x0

Note: The INT_ENABLE bits control whether corresponding interrupts in the INT_STATUS register assert their respective int_o signals. They do not affect the contents of the INT_STATUS register. Refer to the Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide (FPGA-UG-02039) for more information.

5.11. INT_SET Register

Table 5.12. INT_SET Register

Field	Name	Description	Access	Reset
[31]	done_set	Set Done interrupt (sending command to flash is done) Writing 1 to the bit sets the bit of the interrupt.	WO	—
[30:8]	reserved	Reserved	RSVD	—
[7]	wr_fifo_full_set	Set WR FIFO full interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—
[6]	wr_fifo_almost_full_set	Set WR FIFO almost full interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—
[5]	wr_fifo_almost_empty_set	Set WR FIFO almost empty interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—
[4]	wr_fifo_empty_set	Set WR FIFO empty interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—
[3]	rd_fifo_full_set	Set RD FIFO full interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—
[2]	rd_fifo_almost_full_set	Set RD FIFO almost full interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—
[1]	rd_fifo_almost_empty_set	Set RD FIFO almost empty interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—
[0]	rd_fifo_empty_set	Set RD FIFO empty interrupt Writing 1 to the bit sets the bit of the interrupt.	WO	—

5.12. TRANSACTION_STATUS Register

Table 5.13. TRANSATION_STATUS Register

Field	Name	Description	Access	Reset
[31]	Idle[1]	0 – Busy; flash is currently busy 1 – Idle; flash is not busy (ready for the new transaction)	RO	0x0[2]
[30:0]	reserved	Reserved	RSVD	—

Notes:

1. You must monitor the idle bit (bit [31]) of the TRANSACTION_STATUS register. While polling is in progress, no other transactions should be initiated, as any concurrent operations will be discarded. This restriction ensures the integrity of the polling mechanism managed by the Internal Flash Controller IP.
2. The value of the idle bit is valid only after the flash completes status polling and returns the actual status. Therefore, its value immediately after reset is not meaningful. For a more detailed use case of this bit, refer to the Normal Operation (Erase Write Read) with TRANSACTION_STATUS section.

6. Example Design

The Internal Flash Controller example design allows you to compile, simulate, and test the Internal Flash Controller IP for MachXO5-NX on the following Lattice evaluation board:

- MachXO5-NX Development Board

6.1. Example Design Supported Configuration

Table 6.1. Internal Flash Controller IP for MachXO5-NX Configuration Supported by the Example Design

Internal Flash Controller IP for MachXO5-NX GUI Attribute	Internal Flash Controller IP for MachXO5-NX Configuration
General Tab	
General	
Interface	APB
RD FIFO Settings	
FIFO Depth	64
Implementation of RD FIFO	LUT
RD FIFO Endian	Little
RD FIFO Almost Full Flag	59
RD FIFO Almost Empty Flag	4
WR FIFO Settings	
FIFO Depth	64
Implementation of WR FIFO	LUT
Endian of WR FIFO	Little
WR FIFO Almost Full Flag	59
WR FIFO Almost Empty Flag	4
CFG0/1/2 and UFM0/1/2 Settings Tab	
CFG0 and UFM0 Settings	
Use CFG0 with size (blocks)	11
CFG0 Initial Value Format	ALL 1
Use UFM0 with size (blocks)	4
UFM0 Initial Value Format	ALL 1
CFG1 and UFM1 Settings	
Use CFG1 with size (blocks)	11
CFG1 Initial Value Format	ALL 1
Use UFM1 with size (blocks)	4
UFM1 Initial Value Format	ALL 1
CFG2 and UFM2 Settings	
Use CFG2 with size (blocks)	11
CFG2 Initial Value Format	ALL 1
Use UFM2 with size (blocks)	4
UFM2 Initial Value Format	ALL 1

Internal Flash Controller IP for MachXO5-NX GUI Attribute	Internal Flash Controller IP for MachXO5-NX Configuration
USERDATA0 to USERDATA8 Settings Tab	
Use USERDATA0 to USERDATA8	Checked
USERDATA0 Settings	
Use USERDATA0 with size (blocks)	2
USERDATA0 Initial Value Format	ALL 1
USERDATA1 Settings	
Use USERDATA1 with size (blocks)	2
USERDATA1 Initial Value Format	ALL 1
USERDATA2 Settings	
Use USERDATA2 with size (blocks)	2
USERDATA2 Initial Value Format	ALL 1
USERDATA3 Settings	
Use USERDATA3 with size (blocks)	2
USERDATA3 Initial Value Format	ALL 1
USERDATA4 Settings	
Use USERDATA4 with size (blocks)	2
USERDATA4 Initial Value Format	ALL 1
USERDATA5 Settings	
Use USERDATA5 with size (blocks)	2
USERDATA5 Initial Value Format	ALL 1
USERDATA6 Settings	
Use USERDATA6 with size (blocks)	2
USERDATA6 Initial Value Format	ALL 1
USERDATA7 Settings	
Use USERDATA7 with size (blocks)	2
USERDATA7 Initial Value Format	ALL 1
USERDATA8 Settings	
Use USERDATA8 with size (blocks)	2
USERDATA8 Initial Value Format	ALL 1
Oscillator Setting Tab	
Instantiate Internal Oscillator	Checked
Use HFCLK for LMMI CLK	Unchecked

6.2. Overview of the Example Design and Features

The example design discussed in this section is created using the *RISC-V MC SoC Project* template in the Lattice Propel™ Design Environment. The generated project includes the following components:

- Processor – RISC-V
- GPIO
- System Memory
- UART – Serial port
- PLL
- Glue Logic

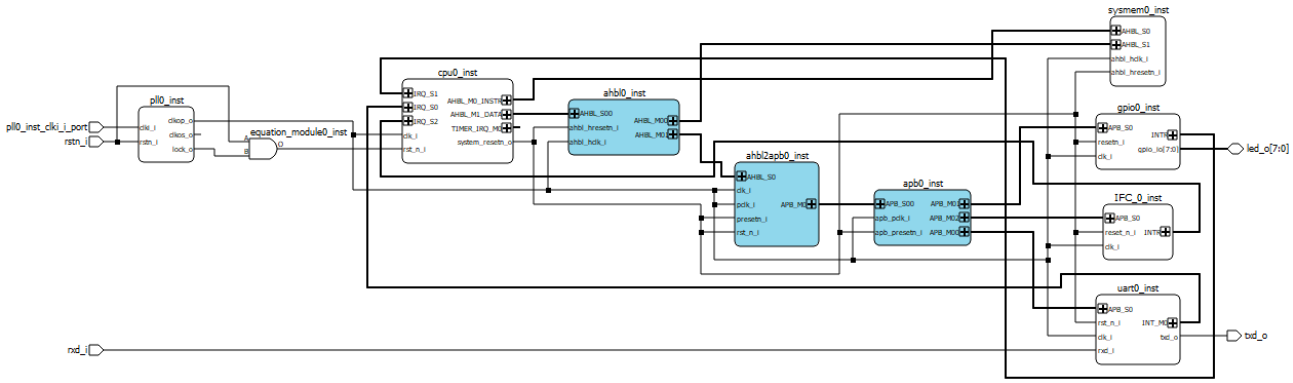

Figure 6.1. Internal Flash Controller IP for MachXO5-NX in Propel SoC Project

An embedded C/C++ project is also created in the Propel software to enable development and debugging of the application code for different IP features, such as soft_reset, read, write, and erase. The Internal Flash Controller features can be tested by sending APB transactions from the processor to the IP. Figure 6.2 shows an example routine of writing to and reading from the CFG1 partition of the flash memory through the IP.

```
//*******************************Write CFG1*************************//
    printf("Writing to CFG1...\n");


    transaction_addr_set(CFG1, 0x000000);
    transaction_ctrl_set(send_byte, trans_wr, 0);
    fifo_ctrl(wr_fifo_flush);

    fifo_status_check();

    uint32_t data_cfg1 = 0x00000000; // Starting data with running 0
    for (int i = 0; i < 64; i++) { // 8 iterations to send 32 bytes (4 bytes each iteration)
        wr_fifo_ctrl(data_cfg1);
        printf("Data sent: 0x%08X\n", data_cfg1);
        data_cfg1 += 0x01010101; // Increment data for running 0 pattern
    }

    fifo_status_check();


    transaction_start();

    done_status_check();

    done_int_status_check();

//*******************************READ DATA of CFG1*************************//
    printf("Reading data from CFG1...\n");

    uint32_t rd_data_cfg1[64]; // Array to store read data
    uint32_t *rd_pt_cfg1; // Pointer to store read data address

    // Initialize read data array
    for (uint32_t i = 0; i < 64; i++) {
        rd_data_cfg1[i] = 0;
    }


    transaction_addr_set(CFG1, 0x000000);
    transaction_ctrl_set(send_byte, trans_rd, 0);
    fifo_ctrl(rd_fifo_flush);

    transaction_start();
```

Figure 6.2. Sample C Code Test Routine

6.3. Example Design Components

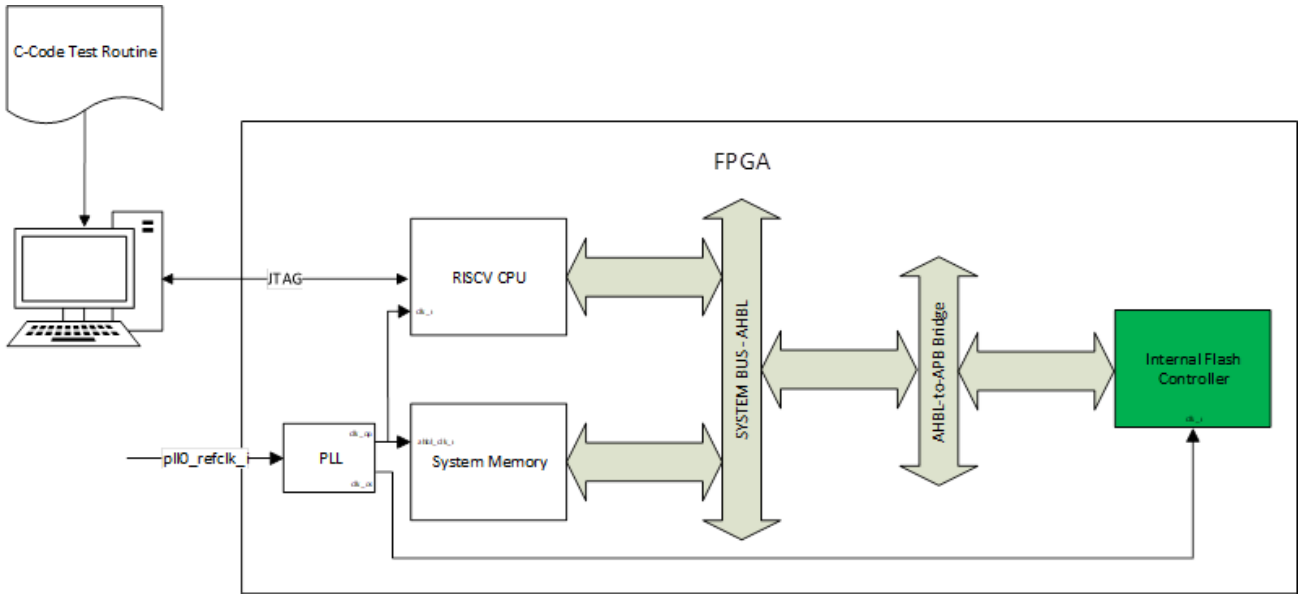

Figure 6.3. Internal Flash Controller Example Design Block Diagram

The Internal Flash Controller example design includes the following blocks:

- RISC-V CPU – Passes the C code test routine from system memory to system bus. Handles interrupts.
- Memory – Contains commands for testing.
- System Bus – AHB-Lite system bus for transfers between memory and IP.
- AHBL-to-APB Bridge – Converts AHB-Lite transactions to APB transactions.
- Internal Flash Controller – IP instance.

6.4. Generating the Example Design

The procedure for generating an example design for the Internal Flash Controller is described below. For more detailed instructions, refer to the Lattice Propel Builder User Guide and Lattice Propel SDK User Guide.

To create an SoC project:

1. Launch the Lattice Propel software and set up your workspace directory.
2. In the Propel software, create a new SoC design project by selecting **File** > **New** > **Lattice SoC Design Project**. The **Create SoC Project** window opens.

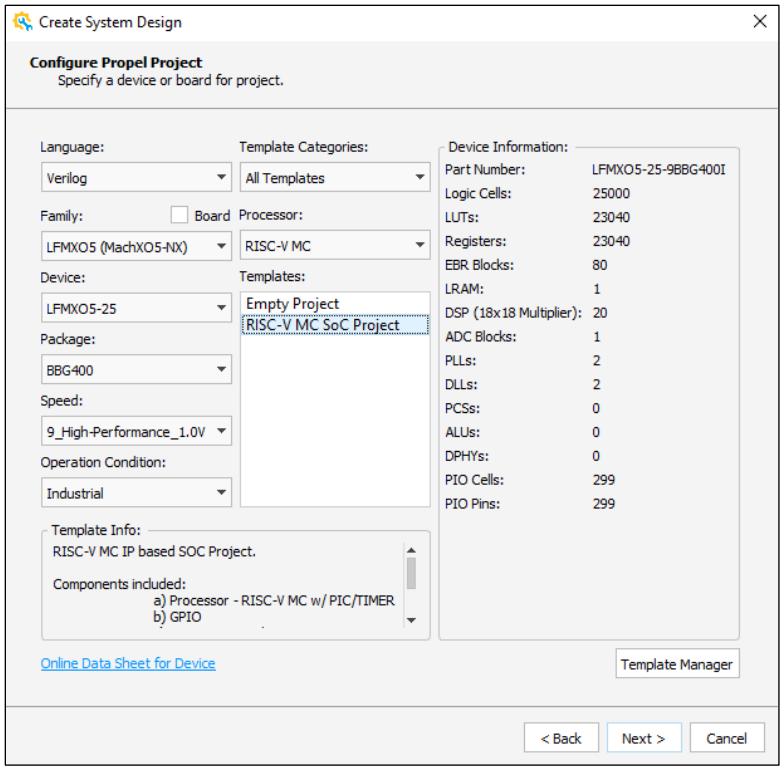

Figure 6.4. Create System Design

3. Customize the options under **Device** to reflect the device that you are using. For this example design, in the Propel Builder project, set the device to LFMXO5-25-9BBG400C because hardware testing uses the MachXO5-NX Development Board.
4. Select the template design from the **Template Design** list. For this example design, select RISC-V MC SoC Project.
5. Click **Finish**.
6. Select the generated project, then run Propel Builder by clicking the icon or selecting **LatticeTools** > **Open Design** in Propel Builder**.** Propel Builder opens and loads the design template.
7. On the **IP on Local** tab, instantiate the Internal Flash Controller IP for MachXO5-NX with the configuration as shown in Table 6.1.
   Note: If the IP is not available on the **IP on Local** tab, download the IP from the **IP on Server** tab.
8. After generating the IP, the **Define Instance** window opens. Modify the instance name if needed, then click **OK.**

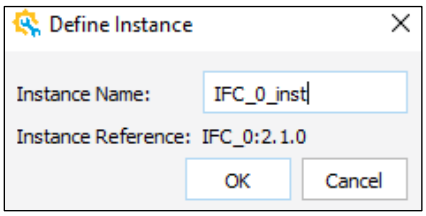

Figure 3.6. Define Instance

9. Connect the instantiated IP to the system. Refer to Figure 6.1 for the connections used in this IP. You will need to update other components of the system for clock and reset sources, interrupt, and bus interface.
10. Save the system design.

To create a Radiant project:

1. Click the icon or select **Design** > **Run Radiant** to launch the Lattice Radiant software.
2. Update the IP constraints file accordingly.
3. Generate the programming file.

To create a C/C++ project:

1. In the Lattice Propel software, build your SoC project to generate the system environment needed for the embedded C/C++ project. Select the generated SoC project followed by **Project** > **Build Project**.
2. Check the build result from the **Console** tab.

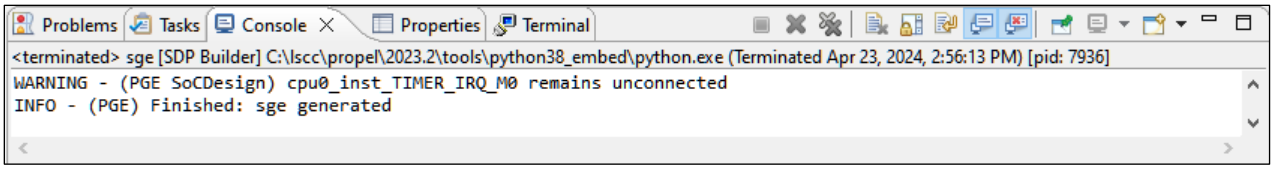

Figure 6.5. Build SoC Project Result

3. Generate a new Lattice C/C++ project by selecting **File** > **New** > **Lattice C/C++ Project**. Update the **Project name**, then click **Next** followed by **Finish**.

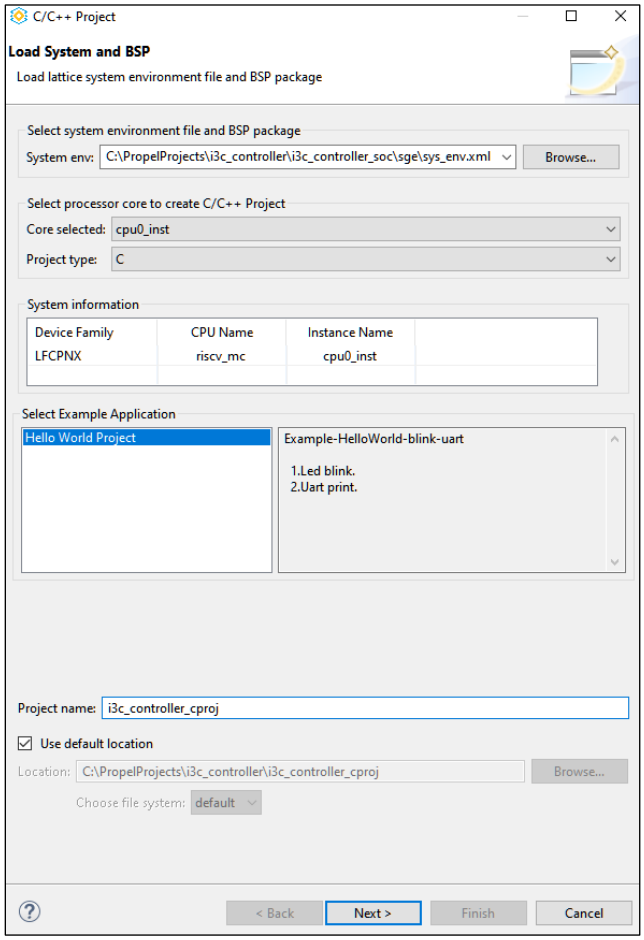

Figure 6.6. C/C++ Project

4. Select the C/C++ project to build followed by **Project** > **Build**.

5. Check the build result from the **Console** tab.

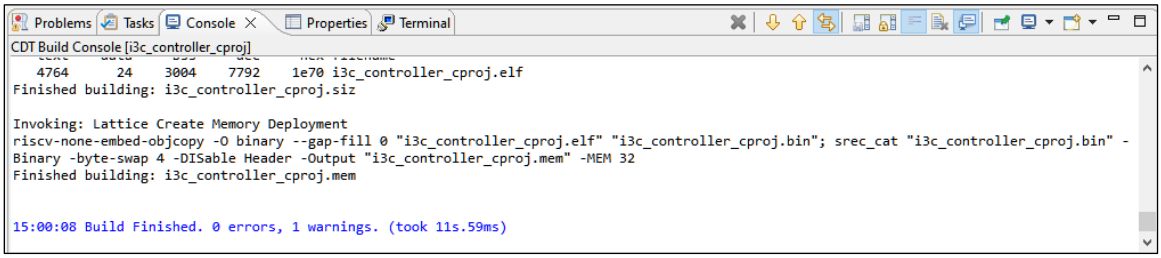

Figure 6.7. Build C/C++ Project Result

This environment is now ready for running your tests on the device. Refer to the *Running Demo on MachXO3D Breakout Board – Hello World* section of the Lattice Propel SDK User Guide for a step-by-step guide.

6.5. Hardware Testing

6.5.1. Hardware Testing Setup

Download the generated bitstream file from the Generating the Example Design section to the MachXO5-NX Development Board through the Lattice Radiant Programmer.

6.5.2. Expected Output

Figure 6.8 shows a sample console output of the APB transactions between the CPU and the IP. Data written to and read from the Internal Flash Controller IP for MachXO5-NX are displayed.

Figure 6.8. C/C++ Project Log

The Reveal analyzer can be added to the Radiant project to check the waveforms of the IP. Refer to the relevant sections in the Lattice Radiant Software User Guide for more information on using the Reveal Inserter and Reveal Analyzer tools.

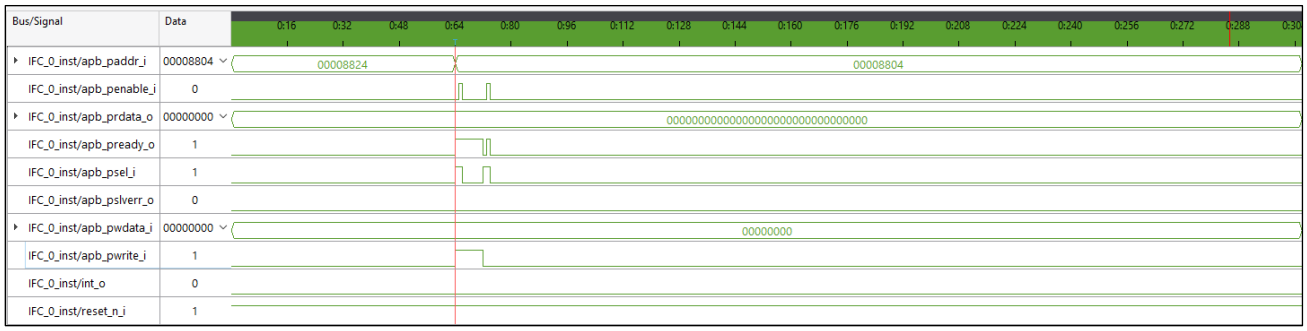

Figure 6.9. APB Interface Reveal Waveform

7. Designing with the IP

This section provides information on how to generate the IP Core using the Lattice Propel Builder software. For more details on the Lattice Propel Builder software, refer to the Lattice Propel Builder User Guide.

Note: The screenshots provided are for reference only. Details may vary depending on the version of the IP or software being used. If there have been no significant changes to the GUI, a screenshot may reflect an earlier version of the IP.

Note: The IP generation flow does not include a testbench. Any helper scripts or memory files in the IP package are provided as-is. You can choose to simulate your SoC design using your own testbench and simulator.

7.1. Generating and Instantiating the IP

You can use the Lattice Propel Builder software to generate IP modules and integrate them into the device architecture. The steps below describe how to generate the Internal Flash Controller IP for MachXO5-NX in the Lattice Propel Builder software.

To generate the Internal Flash Controller IP for MachXO5-NX:

1. Double-click **UFM Access** under **Architecture_Modules** to open the Module/IP Block Wizard.
2. Enter the name of the component in **Component name** and the target location in **Create in**.

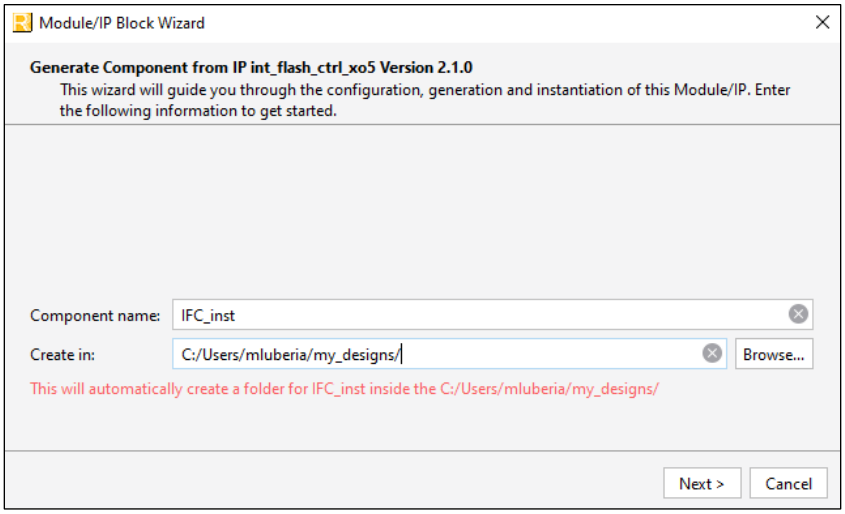

Figure 7.1. Generating Internal Flash Controller Using Module/IP Block Wizard

3. Click **Next**.
4. Choose the partition settings**.** Figure 7.2, Figure 7.3, Figure 7.4, Figure 7.5, and Figure 7.6 show the attribute tabs.

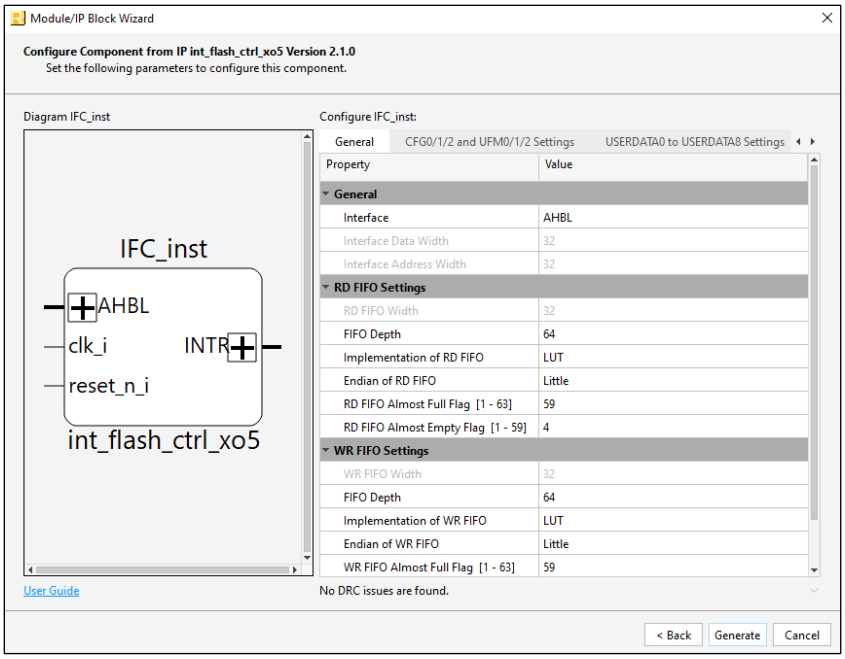

Figure 7.2. Module/IP Block Wizard – General Tab

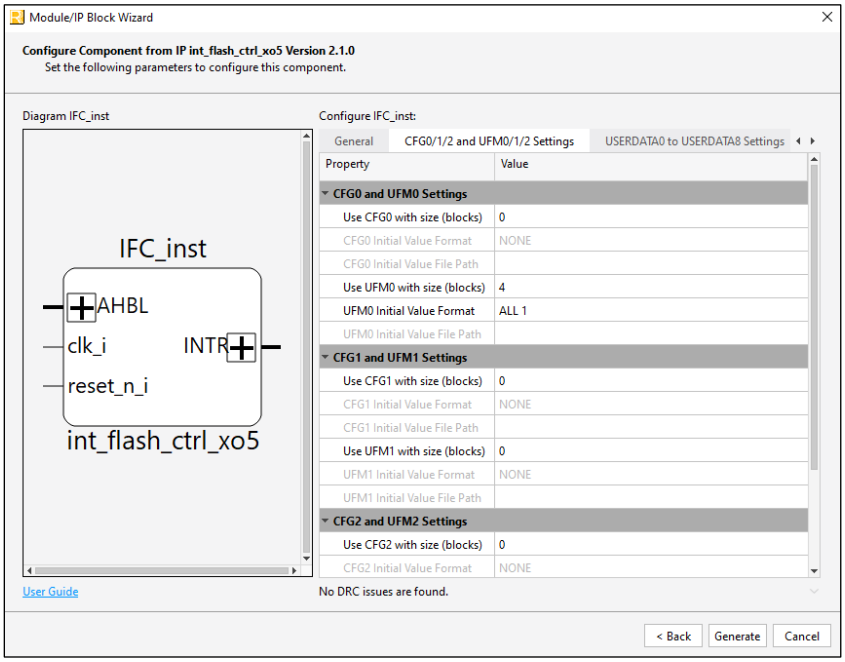

Figure 7.3. Module/IP Block Wizard – CFG0/1/2 and UFM0/1/2 Settings Tab

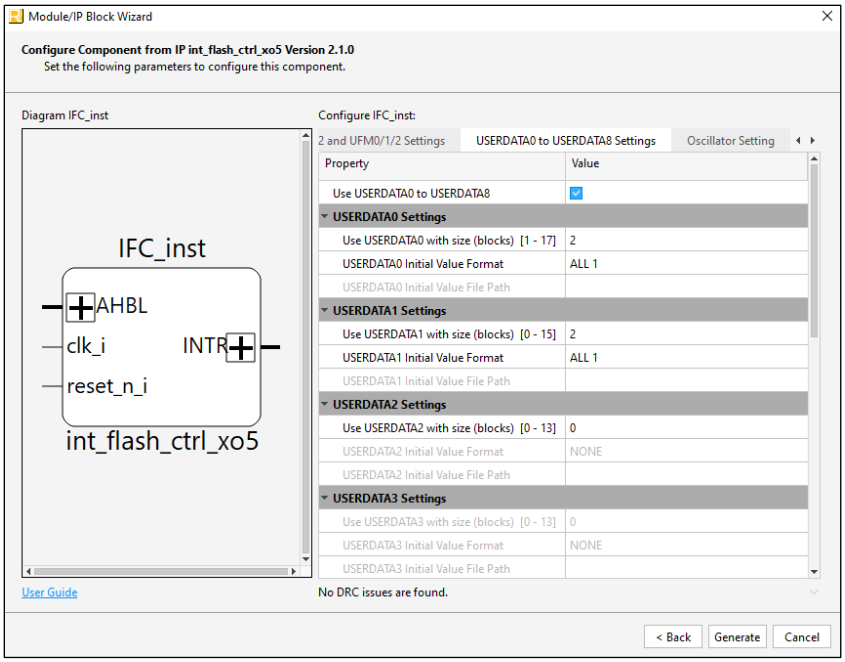

Figure 7.4. Module/IP Block Wizard – USERDATA0 to USERDATA8 Settings Tab

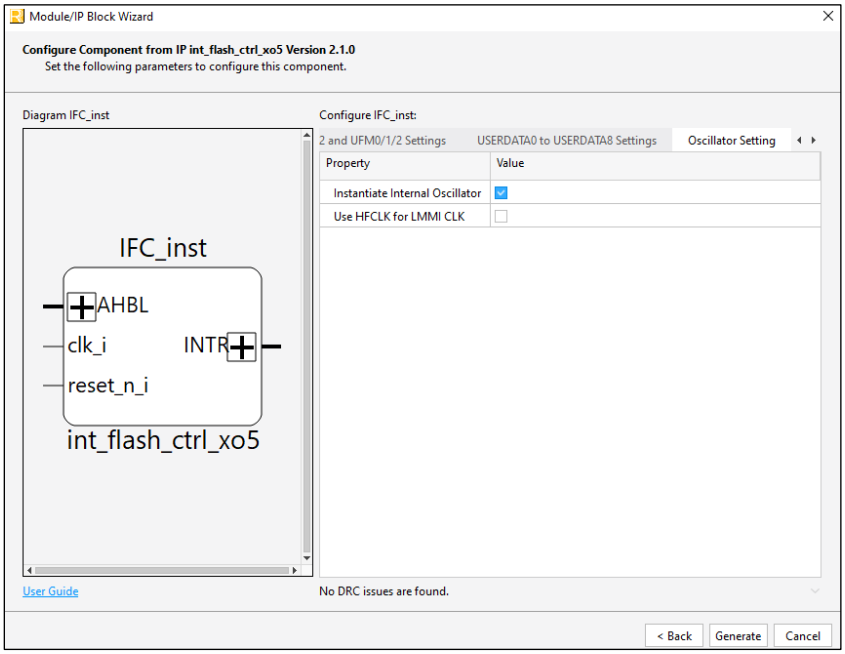

Figure 7.5. Module/IP Block Wizard – Oscillator Setting Tab

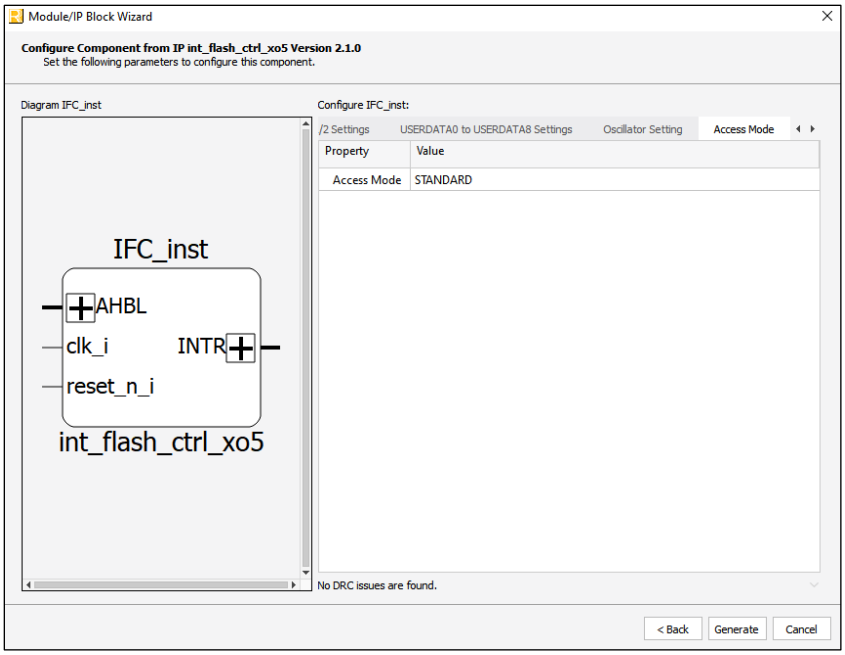

Figure 7.6. Module/IP Block Wizard – Access Mode Tab

5. When all required attributes are set, click **Generate.**
6. Click **Finish**.

Once the IP module is in the Lattice Propel Builder software project, it can be instantiated in other modules within the project. You can view the files and instances added in the **File List** tab.

Appendix A. Resource Utilization

Table A.1 and Table A.2 show the resource utilization of the Internal Flash Controller IP for MachXO5-NX for different devices using the Lattice Synthesis Engine of the Lattice Propel Builder software. The default configuration is used, and some attributes are changed from the default value to show the effect on the resource utilization.

Table A.1. Resource Utilization Using LFMXO5-25-7BBG256C

Configuration	Clk f_{MAX} (MHz)[1]		Slice Registers	LUTs	EBRs
	7_Low_Power	7_High_ Performance			
Interface = AHBL, RD FIFO Depth = 64, Implementation of RD FIFO = LUT, WR FIFO Depth = 64, Implementation of WR FIFO = LUT	81.70	121.91	4792	3929	0
Interface = AHBL, RD FIFO Depth = 64, Implementation of RD FIFO = EBR, WR FIFO Depth = 64, Implementation of WR FIFO = EBR	*80.95*	112.0	629	964	2
Interface = AHBL, RD FIFO Depth = 32, Implementation of RD FIFO = LUT, WR FIFO Depth = 32, Implementation of WR FIFO = LUT	95.44	126.94	2712	2342	0
Interface = AHBL, RD FIFO Depth = 32, Implementation of RD FIFO = EBR, WR FIFO Depth = 32, Implementation of WR FIFO = EBR	106.09	120.14	607	915	2
Interface = APB, RD FIFO Depth = 64, Implementation of RD FIFO = LUT, WR FIFO Depth = 64, Implementation of WR FIFO = LUT	105.25	139.82	4520	3454	0

Note:

1. f_{MAX} is generated when the FPGA design only contains the Internal Flash Controller IP for MachXO5-NX and the target frequency is 133 MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.2. Resource Utilization Using LFMXO5-25-9BBG256C

Configuration	Clk f_{MAX} (MHz)[1]	Slice Registers	LUTs	EBRs
Interface = AHBL, RD FIFO Depth = 64, Implementation of RD FIFO = LUT, WR FIFO Depth = 64 Implementation of WR FIFO = LUT	144.53	4784	3996	0
Interface = AHBL, RD FIFO Depth = 64, RD FIFO Implementation = EBR, WR FIFO Depth = 64, Implementation of WR FIFO = EBR	153.8	622	980	2
Interface = AHBL, RD FIFO Depth = 32, Implementation of RD FIFO = LUT, WR FIFO Depth = 32, Implementation of WR FIFO = LUT	154.68	2706	2362	0
Interface = AHBL, RD FIFO Depth = 32, Implementation of RD FIFO = EBR, WR FIFO Depth = 32, Implementation of WR FIFO = EBR	182.05	594	897	2
Interface = APB, RD FIFO Depth = 64, Implementation of RD FIFO = LUT, WR FIFO Depth = 64, Implementation of WR FIFO = LUT	159.185	4520	3451	0

Note:

1. f_{MAX} is generated when the FPGA design only contains the Internal Flash Controller IP for MachXO5-NX and the target frequency is 133 MHz. These values may be reduced when user logic is added to the FPGA design.

References

- Internal Flash Controller IP for MachXO5-NX Release Notes (FPGA-RN-02085)
- Internal Flash Controller Driver API Reference (FPGA-TN-02421)
- Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide (FPGA-UG-02039)
- AMBA 3 AHB-Lite Protocol V1.0
- AMBA 3 APB Protocol V1.0
- MachXO5-NX web page
- MachXO5-NX Development Board web page
- Internal Flash Controller IP Core web page
- Lattice Propel Design Environment web page
- Lattice Radiant Software web page
- Lattice Solutions IP Cores web page
- Lattice Solutions Reference Designs web page
- Lattice Insights for Lattice Semiconductor training courses and learning plans

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.

Revision History

Note: In some instances, the IP may be updated without changes to the user guide. The user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates.

Revision 1.8, IP v2.3.0, February 2026

Section	Change Summary
Functional Description	In the IP Architecture Overview section: • Added note on the need to disable the SLAVE_I2C_PORT/SLAVE_I3C_PORT option.

Revision 1.7, IP v2.3.0, December 2025

Section	Change Summary
All	• Added a note on the IP version in the Quick Facts and Revision History sections. • Made minor editorial fixes.
Abbreviations in This Document	Added *ISR*.

Introduction	• In Table 1.1. Summary of the Internal Flash Controller IP for MachXO5-NX:
	  • Updated IP core version and software version in *Lattice Implementation*.
	  • Added *Driver Support* row.
	• Removed the Ordering Part Number section.

Function Description	• Made minor editorial changes in the IP Architecture Overview and Clocking and Reset Overview sections.
	• In the Block Partitioning section:
	  • Made minor editorial changes.
	  • Updated column headers to *Block Count* and *Sector Count* in Table 2.1. Flash Memory Map for LFMXO5-25 Device through Table 2.3. Flash Memory Map for LFMXO5-55T and LFMXO5-100T Devices.
	  • Updated column headers to *Item No.*, *CFGx Block Count*, and *UFMx Block Count* in Table 2.4. Different Combinations of CFGx/UFMx Block Size.
	  • Updated column header to *Block Count* in Table 2.5. Sample Settings 1 and Table 2.6. Sample Settings 2.
	• In the Setting Initial Data in Flash Memory section:
	  • Added clarification on initial data for simulation purpose.
	• In the Write Access Operation 1 section:
	  • Added reference to the Erase Operation section.
	  • Updated Steps 7 through 9 to describe further the need for polling FIFO_STATUS.done and INT_STATUS.done_int.
	• In the Write Access Operation 2 section:
	  • Under Interrupt-Based Monitoring in Step 5:
	  • Added description to note on polling to initially fill the FIFO.
	  • Added relevant flag names to Step e.
	  • Updated Step f to specifically mention the steps to repeat.
	  • Updated Steps 6 through 8 to describe further the need for polling FIFO_STATUS.done and INT_STATUS.done_int.
	• In the Read Access Operation 1 section:
	  • Updated Steps 9 through 11 to describe further the need for polling FIFO_STATUS.done and INT_STATUS.done_int.
	• In the Read Access Operation 2 section:
	  • Under polling.FIFO_STATUS.rd_fifo_full in Step 6:
	  • Updated Step e to indicate that this step is only applicable for interrupt-driven completion and not pure polling scenarios.

IP Parameter Description	• In Table 3.1. General Attributes:

Section	Change Summary

	• Substituted – symbol representing range with *to* for selectable values of attributes *RD FIFO Almost Full Flag*, *RD FIFO Almost Empty Flag*, *WR FIFO Almost Full Flag*, and *WR FIFO Almost Full Flag*.
	• In Table 3.2. CFG0/1/2 and UFM0/1/2 Attributes:
	  • Removed statement on initial data for simulation purpose from descriptions of *CFGx Initial Value Format* and *UFMx Initial Value Format*.
	  • Added Note 2 clarifying purpose of initial value options.
	• In Table 3.3. USERDATA0 to USERDATA8 Attributes:
	  • Removed statement on initial data for simulation purpose from descriptions of *USERDATA0 Initial Value Format* and *USERDATA1 Initial Value Format*.
	  • Added Note 2 clarifying purpose of initial value options.

Example Design	• Removed note on check and cross symbols in the Example Design Supported Configuration section. • In Table 6.1. Internal Flash Controller IP for MachXO5-NX Configuration Supported by the Example Design: • Replaced check symbol with *Checked* and cross symbol with *Unchecked*. • Updated Step 7 to reference the IP on Local tab and added note on IP on Server tab in the Generating the Example Design section.
Designing with the IP	• Added note on IP version in GUI. • Added note clarifying availability of testbench.
References	Added Internal Flash Controller Driver API Reference.

Revision 1.6, IP v2.2.0, July 2025

Section	Change Summary
Abbreviations in This Document	Added *AMBA*, *CPU*, *EBR*, *FIFO*, *FPGA*, *GPIO*, *GUI*, *HDL*, *LUT*, *PLL*, *RISC-V*, *SDK*, *SoC*, *SPI*, and *UART*.
Introduction	In Table 1.1. Summary of the Internal Flash Controller IP for MachXO5-NX: • Updated IP core version for *Lattice Implementation*.
Functional Description	Updated labeling from *I/F* to *Interface* in the following figures: • Figure 2.1. Internal Flash Controller Block Diagram (Default) • Figure 2.2. Clocking and Reset with Internal Oscillator • Figure 2.3. Clocking and Reset with Internal Oscillator (Use HFCLK for LMMI CLK = Checked) • Figure 2.4. Clocking and Reset with Oscillator Soft IP

Revision 1.5, IP v2.1.0, July 2025

Section	Change Summary
All	Updated references from Radiant to Propel Builder where appropriate.
Abbreviations in This Document	• Updated section title and introducing sentence. • Added *APB*, *IP*, and *UFM.*

Introduction	• Updated description.
	• Added the Overview of the IP, Quick Facts, IP Support Summary, Licensing and Ordering Information, and Hardware Support sections.
	• Updated description in the Features section.
	• In the Naming Conventions section:
	  • Updated section title.
	  • Added the Signal Names and Attribute Names sections.

Functional Description	• In the IP Architecture Overview section:
	  • Updated section title and description.
	  • Updated Figure 2.1. Internal Flash Controller Block Diagram (Default).

Section	Change Summary

	• Moved the Internal Flash Controller Block Diagram (Use HFCLK for LMMI CLK = Checked) figure into the Clocking and Reset section.
	• Added the Clocking and Reset section.
	• Moved content in the Attributes Summary, Signal Description, and Register Description sections into the IP Parameter Description, Signal Description, and Register Description sections.
	• Removed the Connecting the External Oscillator section.
	• In the Block Partitioning section:
	  • Added introducing sentences for Table 2.2. Flash Memory Map for LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T Devices and Table 2.3. Flash Memory Map for LFMXO5-55T and LFMXO5-100T Devices.
	  • In Table 2.1. Flash Memory Map for LFMXO5-25 Device:
	  • Removed note to table regarding dual-boot implementation.
	  • Added Table 2.2. Flash Memory Map for LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T Devices.
	  • In Table 2.3. Flash Memory Map for LFMXO5-55T and LFMXO5-100T Devices:
	  • Removed note to table regarding dual-boot implementation.
	  • Updated discussion about CFG and UFM for LFMXO5-55T and LFMXO5-100T devices and reference to table with boundary address for each memory partition.
	  • In Table 2.4. Different Combinations of CFGx/UFMx Block Size, Table 2.5. Sample Settings 1, and Table 2.6. Sample Settings 2:
	  • Reworked table.
	  • Added information for LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T.
	  • Added LFMXO5-55T.
	• In the Setting Initial Data in Flash Memory section:
	  • Made minor editorial changes to description.
	  • Added note on initial data for simulation only.
	  • In Table 2.7. Flash Memory Partition Initialization Options:
	  • Updated table title.
	  • Updated description for NONE.
	  • Removed text *(block size x 128)* in the Binary File section.
	  • Removed text *(block size x 8) i*n the Hexadecimal File section.
	• In the Transaction Offset Computation section:
	  • Updated section title and description.
	  • Updated transaction_address to transaction_offset.
	  • Updated Table 2.8. Flash Memory Map (Partition) for LFMXO5-25 Device and Table 2.10. Flash Memory Map (Partition) for LFMXO5-55T and LFMXO5-100T Devices titles.
	  • Added Table 2.9. Flash Memory Map (Partition) for LFMXO5-35, LFMXO5-35T, LFMXO5-65, and LFMXO5-65T Devices.
	• Updated section title from *Sample Operation Control* to *Programming Flow*.
	• In the Write Access Operation 1 section:
	  • Updated section title.
	  • Added description of recommended procedure for executing a write transaction using the Internal Flash Controller IP.
	  • Removed Write Access Flow Sample 1 figure.
	  • Removed Write Access Flow Sample 2 figure and related descriptions.
	• Added the Write Access Operation 2 section.
	• In the Read Access Operation 1 section:
	  • Updated section title.
	  • Added description of recommended procedure for executing a read transaction using the Internal Flash Controller IP.

Section	Change Summary
	• Removed Read Access Flow 1 (User) figure and related descriptions. • Removed Read Access Flow 2 (User) for transaction_bytes/4 = Depth (User) figure and related descriptions. • Added the Read Access Operation 2 section. • In the Erase Operation section: • Updated section title. • Removed the Erase Access Flow (User) figure and related descriptions. • Added description of recommended procedure for executing an erase transaction using the Internal Flash Controller IP. • Added the Normal Operation (Erase Write Read) section. • Added the Normal Operation (Erase Write Read) with TRANSACTION_STATUS section.
IP Parameter Description	• Added section with reworked content from Attributes Summary. • In Table 3.1. General Attributes: • Updated default values for *Implementation of RD FIFO* and *Implementation of WR FIFO* to *LUT*. • Updated RD FIFO Almost Empty Flag selectable values range. • Updated WR FIFO Almost Empty Flag selectable values range. • Updated attribute names for RD FIFO and WR FIFO to *FIFO Depth*. • Updated attributes names to *Endian of RD FIFO* and *Endian of WR FIFO*. • In Table 3.2. CFG0/1/2 and UFM0/1/2 Attributes: • Streamlined CFG0/1/2 and UFM0/1/2 attribute references to CFGx and UFMx. • Added selectable values for LFMXO5-35/35T/65/65T for the following attributes: Use CFGx with size (blocks) and Use UFMx with size (blocks). • Added dependencies for the following attribute: Use UFMx with size (blocks). • Updated descriptions for the following attributes: Use CFGx with size (blocks), Use UFMx with size (blocks), CFG0/1/2 Initial Value Format, and UFM0/1/2 Initial Value Format. • Split selectable values for Use UFMx with size (blocks) into UFM0 and UFM1/2. • In Table 3.3. USERDATA0 to USERDATA8 Attributes: • Added selectable values for LFMXO5-35/35T/65/65T for the following attributes: Use USERDATA0 with size (blocks), Use USERDATA1 with size (blocks), Use USERDATA2 with size (blocks), Use USERDATA3 with size (blocks), Use USERDATA4 with size (blocks), Use USERDATA5 with size (blocks), Use USERDATA6 with size (blocks), Use USERDATA7 with size (blocks), and Use USERDATA8 with size (blocks). • Updated selectable values for the following attributes: Use USERDATA1 with size (blocks), Use USERDATA2 with size (blocks), Use USERDATA3 with size (blocks), Use USERDATA4 with size (blocks), Use USERDATA5 with size (blocks), Use USERDATA6 with size (blocks), Use USERDATA7 with size (blocks), and Use USERDATA8 with size (blocks). • Updated descriptions for the following attributes: Use USERDATA0 to USERDATA8, Use USERDATA0/1/2/3/4/5/6/7/8 with size (blocks), and USERDATA0/1/2/3/4/5/6/7/8 Initial Value Format. • In Table 3.5. Access Mode Attribute: • Added description and updated dependency for Read Access Mode attribute.
Signal Description	• Reorganized and reworked previous signal description content into this section. • In Table 4.1. Internal Flash Controller Ports: • Added ports lmmi_clk_i and lmmi_resetn_i. • Added note on supported burst type to description for ahbl_hburst_i. • Removed port ahbl_hmastlock_i. • Updated descriptions for int_o, all AHB-Lite Interface ports, and all APB Interface ports. • Added Note 1 to table regarding CONFIG_IOSLEW setting in relation to flash memory clock.

Section	Change Summary

Register Description	• Reorganized and reworked previous register description content into this section.
	• In Table 5.1. Register Access Types:
	  • Updated table title.
	  • Renamed column to *Access Type Abbreviation* and added column *Access Type*.
	• In Table 5.2. Summary of Internal Flash Controller IP for MachXO5-NX Registers:
	  • Added TRANSACTION_STATUS register.
	• In Table 5.3. UFM_CTRL_REG Register:
	  • Updated description of soft_reset.
	• In Table 5.4. TRANSACTION_ADDR Register:
	  • Added reserved bit.
	  • Updated transaction_address to transaction_offset and associated bits.
	• In Table 5.5. WR_FIFO_DATA Register:
	  • Updated cross reference for relationship between WR FIFO and write access to flash memory.
	  • Added note on FIFO_STATUS.wr_fifo_full flag and FIFO depth.
	• In Table 5.6. RD_FIFO_DATA Register:
	  • Removed reset values.
	  • Updated cross reference for relationship between RD FIFO and read access from flash memory.
	  • Added note on FIFO_STATUS.rd_fifo_empty flag and FIFO depth.
	• In Table 5.7. TRANSACTION_CTRL Register:
	  • Updated description for access_mode.
	  • Updated transaction_bytes_div4_m1 to transaction_bytes_m1 and associated description and field.
	  • Added note on flash busy status polling sequence.
	• In Table 5.8. FIFO_STATUS Register:
	  • Updated description for done.
	  • Added note on FIFO_STATUS register being read-only.
	• In Table 5.10. INT_STATUS Register:
	  • Updated description for done_int.
	  • Updated reserved field.
	• In Table 5.11. INT_ENABLE Register:
	  • Updated descriptions for done_en, wr_fifo_full_en, wr_fifo_almost_full_en, wr_fifo_almost_empty_en, wr_fifo_empty_en, rd_fifo_full_en, rd_fifo_almost_full_en, rd_fifo_almost_empty_en, and rd_fifo_empty_en.
	  • Added note to clarify relationship between INT_ENABLE register and INT_STATUS register.
	• In Table 5.12. INT_SET Register:
	  • Updated description for done_set.
	• Added the TRANSACTION_STATUS Register section.

Example Design	Added new section.

Designing with the IP	• Reworked section from Internal Flash Controller for MachXO5-NX Generation.
	• In the Generating and Instantiating the IP section:
	  • Updated Figure 7.1. Generating Internal Flash Controller Using Module/IP Block Wizard, Figure 7.2. Module/IP Block Wizard – General Tab, Figure 7.3. Module/IP Block Wizard – CFG0/1/2 and UFM0/1/2 Settings Tab, Figure 7.4. Module/IP Block Wizard – USERDATA0 to USERDATA8 Settings Tab, and Figure 7.5. Module/IP Block Wizard – Oscillator Setting Tab.
	  • Added Figure 7.6. Module/IP Block Wizard – Access Mode Tab.
	  • Made minor editorial changes.

Appendix A. Resource Utilization	Updated description by referencing Lattice Propel Builder software.
References	• Added Internal Flash Controller IP for MachXO5-NX Release Notes and Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide.

Section	Change Summary
	• Added MachXO5-NX Development Board, Internal Flash Controller IP Core, and Lattice Propel Design Environment web pages. • Updated listings for IP and reference design. • Changed listing from Lattice Radiant FPGA design software to Lattice Radiant Software web page.

Revision 1.4, January 2025

Section	Change Summary

Functional Description	• In Table 2.4. Summary of Internal Flash Controller for MachXO5-NX Registers:
	  • Updated *transaction_bytes_div4_m1* description for the TRANSACTION_CTRL register.
	• In Table 2.6. Flash Memory Map LFMXO5-25:
	  • Updated *Contents* for *00* block start address row.
	  • Added notes regarding primary image, golden image, and jump instruction programming.
	• In Table 2.7. Flash Memory Map LFMXO5-55T and LFMXO5-100T:
	  • Updated *Contents* for *00* block start address row.
	  • Added notes regarding primary image, golden image, and jump instruction programming.
	• In Table 2.9. Sample Settings 1:
	  • Updated *Use Partition Setting* for LFMXO5-100T at block start address 249 to *USERDATA8 Size = 6*.

Revision 1.3, March 2024

Section	Change Summary
Acronyms in This Document	Added JTAG and SSPI.
Functional Description	Added note to Overview section regarding SSPI/JTAG bus operation limitation with SSPI persistence enabled in user mode when using the Internal Flash Controller IP to access flash memory.

Revision 1.2, December 2023

Section	Change Summary
All	• Performed editorial and formatting fixes. • Made minor fixes to comply with Lattice's inclusive language guidelines.
Disclaimers	• Updated boilerplate.
Inclusive Language	• Added this section.
Introduction	• Updated input clock frequency in section 1.1 Features.

Functional Description	• Updated the following tables:
	  • Table 2.1. Internal Flash Controller Signal Description
	  • Table 2.2. Attributes Table
	  • Table 2.3. Attributes Description
	  • Table 2.4. Summary of Internal Flash Controller for MachXO5-NX Registers
	  • Table 2.8. Different Combinations of CFGx/UFMx Block Size
	  • Table 2.9. Sample Settings 1
	  • Table 2.10. Sample Settings 2
	• Updated table caption for the following tables:
	  • Table 2.6. Flash Memory Map LFMXO5-25
	  • Table 2.12. Flash Memory Map LFMXO5-25
	• Added the following tables:

Section	Change Summary
	• Table 2.7. Flash Memory Map LFMXO5-55T and LFMXO5-100T1 • Table 2.13. Flash Memory Map LFMXO5-100T and LFMXO5-55T • Updated description in section 2.6. Block Partitioning. • Updated instances of *UFM_CTRL.start_transaction* to *TRANSACTION_CTRL.start_transaction*. • Updated figure caption for Table 2.9. Sample Settings 1.
References	Updated this section with relevant links.
Technical Support Assistance	Added link to the Lattice Answer Database.

Revision 1.1, August 2022

Section	Change Summary
All	Minor adjustments in formatting across the document.
Functional Description	• Updated Selectable and Default values for Frequency of the Internal LMMI CLK in Table 2.2. Attributes Table. • Updated Table 5.3. to change CFG0/1/2 value for TRANSACTION_ADDR from 0x00_0000–0x0F_FFFF to 0x00_0000–0x0A_FFFF.

Revision 1.0, April 2022

Section	Change Summary
All	Production release

www.latticesemi.com