Migrating Designs from AMD CPLD/FPGA Devices to Lattice FPGA Devices

Application Note

FPGA-AN-02081-1.0

March 2024

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 REDUDANCIES, 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

The following table lists abbreviations used in this document.

Abbreviation	Definition
AI	Artificial Intelligence
AIM	Advanced Interconnect Matrix
ADC	Analog to Digital Converter
AES	Advanced Encryption Standard
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASIC	Application-Specific Integrated Circuits
AXI	Advanced eXtensible Interface
CLB	Configurable Logic Blocks
CMT	Clock Management Tile
CRE	Cryptographic Engine
DCC	Dynamic Clock Control
DCM	Digital Clock Management
DCS	Dynamic Clock Select
DDR3	Double Data Rate Three
DLL	Delay-Locked Loop
DSP	Digital Signal Processor
EBR	Embedded Block of RAM
ECDSA	Elliptic Curve Digital Signature Algorithm
ECLK	Edge Clock
FD-SOI	Fully Depleted Silicon on Insulator
FPGA	Field Programmable Gate Array
GLB	Generic Logic Block
GRP	Global Routing Pool
GUI	Graphical User Interface
HDL	Hardware Description Language
HMAC	Hash-based Message Authentication Code
HPIO	High-Performance Input/Output
I2C	Inter-Integrated Circuit
I/O	Input/Output
IDE	Integrated Development Environment
JTAG	Joint Test Action Group
LC	Logic Cells
LFCPNX	CertusPro-NX Code Name
LFD2NX	Certus-NX Code Name
LFMXO5	MachXO5-NX Code Name
LIFCL	CrossLink-NX Code Name
LMMI	Lattice Memory Mapped Interface
LPDDR4	Low Power Double Data Rate Four
LRAM	Large Random Access Memory
LSE	Lattice Synthesis Engine
LUT	Look Up Table
LDI	LVDS Display Interface

Abbreviation	Definition
LSE	Lattice Synthesis Engine
LVDS	Low Voltage Differential Signaling
MIPI	Mobile Industry Processor Interface
ML	Machine Learning
MMCM	Mixed-Mode Clock Manager
MUX	Multiplexer
ORP	Output Routing Pool
OS	Operating System
PAR	Place and Route
PCIe	Peripheral Component Interconnect express
PCLK	Primary Clock
PCS	Physical Coding Sublayer
PIC	Programmable I/O Cell
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PLL	Phase-Locked Loop
PTAT	Proportional to Absolute Temperature
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SDC	Synopsys Design Constraint
SDK	Software Development Kit
SECDED	Single Error Correction - Double Error Detection
SED	Soft Error Detect
SER	Soft Error Rate
SerDes	Serializer Deserializer
SEU	Single Event Upset
SLC	System Logic Cells
SoC	System on Chip
SGMII	Serial Gigabit Media Independent Interface
TCL	Tool Command Language
TDP	True Dual Port
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WRIO	Wide-Range Input/Output
ZIA	Zero-Power Interconnect Array

1. Competitive Positioning

1.1. Lattice FPGA Devices

Lattice Semiconductor offers a wide range of Field Programmable Gate Array (FPGA) devices that suit different industry needs. The selection of one device family depends on multiple factors, such as but not limited to, logic density, number of input/output (I/O) needed, memory requirements, digital signal processing requirements, and security requirements.

The main advantages of FPGA devices are flexibility and adaptability. FPGA devices can implement variety of functions and can be reprogrammed infinitely to adapt to the needs of various industry segments. For example, the needs for communication market may be slightly different than the industrial or server market.

Lattice offers a variety of device families that are tuned to the needs of each market and can be segmented into the following categories. Table 1.1 shows selected device families for each of the categories.

For more information on each device family, refer to the respective device family datasheet. For example, you can refer to the CertusPro-NX Family Data Sheet (FPGA-DS-02086) to learn more about the Lattice CertusPro™-NX device family.

Table 1.1. Selected Device Families for Each Lattice FPGA Device Category

Category	FPGA Device Family	Device Capabilities
Mid-Range General-Purpose FPGA devices (up to 638 system logic cells)	Avant™	The Avant devices have the following capabilities: • Fast and flexible SerDes which supports up to 28 channels of PCIe Gen 4, 25G ethernet. • Fast external memory that support LPDDR4/DDR4 with speed up to 2400 Mbps and DDR5 with speed up to 2100 Mbps. • Consume up to 2.5× lower power than competitive products. • Efficient edge AI processing with up to 7200 INT8 multipliers and 35.6 Mb embedded memory which enables efficient implementation of AI/ML algorithms as well as packet buffering of high-speed interfaces.
Low Density Video Connectivity FPGA devices (up to 40k logic cells)	Crosslink™-NX	The Crosslink-NX devices have the following capabilities: • Consume up to 75% less power when compared to similar FPGA devices. • Have small form factor packaging with sizes as small as 4 mm × 4 mm. • Have large DSP resources and high memory to logic cell ratios (up to 170 bits per logic cell) which accelerate the artificial intelligence (AI) inferencing to provide high vision processing applications performance. • Provide high speed interfaces supporting 2.5 Gb/s Hardened MIPI D-PHY, 5 Gb/s PCIe, 1.5 Gbps programmable I/O, and 1066 Mb/s DDR3. • Support other interfaces such as LVDS, subLVDS, OpenLDI (OLDI), and SGMII.
Low-Density General-Purpose FPGA devices (up to 150k logic Cells)	CertusPro™-NX	The CertusPro-NX devices have the following capabilities: • Have a high power efficiency. • Provide up to 100 times higher reliability due to the 100 times lower Soft Error Rate (SER) from the 28 nm FD-SOI technology. • Support 10 Gb/s SerDes at the lowest power usage and with the smallest package size. • Support LPDDR4 with up to 7.3 Mb of on-chip memory.
Control and Security FPGA devices (up to 25k logic cells)	MachXO5™-NX	The MachXO5-NX devices have the following capabilities: • Have a high I/O to logic ratio. • Have up to 25k logic density, 1.9 Mb of embedded memory, and up to 15.362 Mb of user flash memory. • Have device security features that can protect your intellectual property such as internal flash configuration, AES256 bitstream encryption, ECC256 bitstream authentication, configuration port lock, and run-time security.

Category	FPGA Device Family	Device Capabilities
Ultra-low Power FPGA devices (up to 5k logic cells with static current of 75 µA)	iCE40 UltraPlus™	The iCE40 UltraPlus devices have the following capabilities: • Provide low-power connectivity and low-power computing capabilities. • Solve connectivity issues with a wide variety of interfaces and protocols. • Provide low-power computational resources for higher levels of intelligence on the edge. • Able to implement neural networks for pattern matching necessary to bring always-on intelligence to the edge.

1.2. AMD CPLD/FPGA Devices

The AMD product portfolio focuses on high-end FPGA devices in terms of logic cells density, performance, SerDes speed for system on chip (SoC) integration. Table 1.2 shows the summary of the Complex Programmable Logic Devices (CPLD)/FPGA devices offered by AMD:

Table 1.2. Summary of AMD CPLD/FPGA Devices

Years	CPLD/FPGA Devices
Prior to 2001	XC9500 and CoolRunner™ (EOL)
2001–2007	Virtex®, Spartan®-II (EOL), Virtex-II, Spartan-3(EOL), and Virtex-4 (90 nm)
2006–2009	Virtex-5 (65 nm)
2009	Virtex-6 and Spartan-6 (40/45 nm).
2010–2012	Spartan-7, Artix™-7, Kintex™-7, Virtex-7, and Zynq®-7000 (28 nm)
2013–2018	UltraScale™ and UltraScale+™ (16 nm)
2019	Versal™ (7 nm)

You can refer to the AMD documentation for more details on their device offerings. The need for higher integration level and larger density pushes AMD to move to the smallest process node.

This document focuses on the AMD device offerings that overlap with Lattice in terms of density, features, and performance. Refer to the Competitive Positioning section for more details.

All the recent AMD FPGA devices are using LUT6 as the basic element to implement logic while Lattice FPGA devices are using 4-input lookup tables (LUT4). The difference between the two implementations and the impact it may have on the resource utilization for a given design are discussed in the subsequent sections.

1.3. Device Competitive Positioning Summary

The following Table 1.3 – Table 1.14 in this section provide a rough device mapping between AMD and Lattice devices. In some cases, you may have multiple options as a replacement device. The subsequent sections will clarify what option is most suitable for you to help you choose the right device as a replacement.

1.3.1. FPGA Architecture

Table 1.3. AMD Kintex UltraScale+ Devices and the Closest Lattice Devices Mapping

AMD Device	System Logic Cells	SRAM Dist./EBR/Ultra (Mb)	SerDes (16.3 Gb/s – 32.75 Gb/s)	PLL	External Memory	27 x 18 Mult.	PCIe Lanes	Closest Lattice Devices
KU3P	356k	4.7/12.7/13.5	0/16	4	DDR3, DDR3L, LPDDR3, DDR4, LPDDR4	1368	16	LAV-AT-G50, LAV-AT-X50
KU5P	475k	6.1/16.9/18.0	0/16	4		1824	16	LAV-AT-G70, LAV-AT-X70
KU9P	600k	8.8/32.1/0	28/0	4		2520	0	LAV-AT-G70, LAV-AT-X70

AMD Device	System Logic Cells	SRAM Dist./EBR/Ultra (Mb)	SerDes (16.3 Gb/s – 32.75 Gb/s)	PLL	External Memory	27 x 18 Mult.	PCIe Lanes	Closest Lattice Devices
KU11P	653k	9.1/21.1/22.5	32/20	8		2928	64	LAV-AT-G70, LAV-AT-X70
KU13P	747k	11.3/26.2/31.5	28/0	4		3528	0	LAV-AT-G70, LAV-AT-X70
KU15P	1143k	9.8/34.6/36.0	44/32	11		1968	80	LAV-AT-G70, LAV-AT-X70
KU19P	1843k	11.6/60.8/81.0	0/32	9		1080	0	LAV-AT-G70, LAV-AT-X70

Table 1.4. AMD Artix UltraScale+ Devices and the Closest Lattice Devices Mapping

AMD Device	System Logic Cells	SRAM Dist./EBR (Mb)	SerDes (12.5 Gb/s – 16.3 Gb/s)	PLL	27 x 18 Mult.	Closest Lattice Devices
AU7P	82k	1.1/3.8	4/0	4	216	LFCPNX-100, LFE5UM5G-85, ECP3-95, ECP2M-100, LAV-AT-G30, LAV-AT-X30
AU10P	96k	1.0/3.5	8/12	6	400	LFCPNX-100, LFE5UM5G-85, ECP3-95, ECP2M-100, LAV-AT-G30, LAV-AT-X30
AU15P	170k	2.5/5.1	8/12	6	576	ECP3-150, LAV-AT-G30, LAV-AT-X30
AU20P	238k	3.2/7.0	12	6	900	ECP3-150, LAV-AT-G30, LAV-AT-X30
AU25P	308k	4.7/10.5	12	8	1,200	ECP3-150, LAV-AT-G50, LAV-AT-X50

Table 1.5. AMD Kintex-7 Devices and the Closest Lattice Devices Mapping

AMD Device	Logic Cells	EBR SRAM (Mb)	SerDes (12.5 Gb/s)	PLL	27 x 18 Mult.	PCIe (Gen2)	Closest Lattice Devices
XC7K70T	65.6k	4.8	8	6	240	1	LAV-AT-E30, LFCPNX-100
XC7K160T	162.2k	11.7	8	8	600	1	LAV-AT-E30
XC7K325T	326.0k	16.0	16	10	840	1	LAV-AT-E50
XC7K355T	356.1k	25.7	24	6	1440	1	LAV-AT-E50
XC7K410T	406.7k	28.6	16	10	1540	1	LAV-AT-E70
XC7K420T	416.9k	30.0	32	8	1680	1	LAV-AT-E70
XC7K470T	477.7k	34.3	32	8	1920	1	LAV-AT-E70

Table 1.6. AMD Artix-7 T Devices and the Closest Lattice Devices Mapping

AMD Device	Logic Cells	EBR SRAM (Mb)	SerDes (6.6 Gb/s)	PLL	25 x 18 Mult.	Closest Lattice Devices
XC7A12T	13k	0.7	2	3	40	LFE5UM-25, LFE5UM5G-25, ECP2M-20, ECP3-17
XC7A15T	17k	0.9	0/2/4	5	45	LFE5UM-25, LFE5UM5G-25, ECP2M-20, ECP3-17, LFMXO5-25

AMD Device	Logic Cells	EBR SRAM (Mb)	SerDes (6.6 Gb/s)	PLL	25 x 18 Mult.	Closest Lattice Devices
XC7A25T	23k	1.6	4	3	80	LFE5UM-25, LFE5UM5G-25, ECP2M-20, ECP3-17, LFMXO5-25
XC7A35T	33k	1.8	0/2/4	5	90	LFE5UM-45, LFE5UM5G-45, ECP3-35, LFD2NX-40, LIFCL-40, LFCPNX-50
XC7A50T	52k	2.7	0/2/4	5	120	LFCPNX-50, LFE5UM-45, LFE5UM5G-45, ECP3-70
XC7A75T	75k	3.8	0/4/8	6	180	LFCPNX-100, LFE5UM-85, LFE5UM5G-85, ECP3-70
XC7A100T	101k	4.9	0/4/8	6	240	LFCPNX-100, ECP3-95, ECP3-150
XC7A200T	215k	13.1	4/8/16	10	740	ECP3-150

Table 1.7. AMD Spartan-7 Devices and the Closest Lattice Devices Mapping

AMD Device	Logic Cells	EBR SRAM (kb)	PLL	25 x 18 Mult.	Closest Lattice Devices
XC7S6	6k	180	2	10	MachXO3L-6900, ECP2-6, LP8K
XC7S15	13k	360	2	20	LFE5U-12, ECP2-12, ECP3-17, LIFCL-17, LFD2NX-17, LFMXO5-25
XC7S25	23k	1,620	3	80	LFE5U-25, ECP2-20, ECP3-17, LIFCL-17, LFD2NX-17, LFMXO5-25
XC7S50	52k	2,700	5	120	LFE5U-45, ECP2-50, ECP3-70, LIFCL-40, LFD2NX-40, LFCPNX-50, LFMXO5-55T
XC7S75	77k	3,240	8	140	LFE5U-85, ECP3-70, ECP2-70, LFCPNX-100, LFMXO5-100T
XC7S100	102k	4,320	8	160	ECP3-95, ECP3-150, ECP2M-100, LFCPNX-100, LFMXO5-100T

Table 1.8. AMD Spartan-6 LX Devices and the Closest Lattice Devices Mapping

AMD Device	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	Embedded PCIe I/F	Embedded Mem. Cntl.	Closest Lattice Devices
XC6SLX4	4k	75	216	12	2	4	0	0	XO2-4000HC, XO2-4000HE, ECP2-6, LP8K, MachXO3L-4300
XC6SLX9	9k	90	576	32	2	16	0	2	XO2-7000HC, XO2-7000HE, ECP2-6, ECP2-12, MachXO3L-6900
XC6SLX16	15k	136	576	32	2	32	0	2	ECP3-17, LIFCL-17, LFD2NX-17, LFMXO5-25

AMD Device	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	Embedded PCIe I/F	Embedded Mem. Cntl.	Closest Lattice Devices
XC6SLX25	24k	228	936	52	2	38	0	2	ECP2-20, ECP3-35, LFE5U-25, LIFCL-17, LFD2NX-17, LFMXO5-25
XC6SLX45	44k	401	2088	116	4	58	0	2	ECP3-35, LFE5U-45, LIFCL-40, LFD2NX-40, LFCPNX-50
XC6SLX75	75k	975	4824	268	6	182	0	4	ECP3-70, LFE5U-85, LFCPNX-100
XC6SLX100	101k	975	4824	268	6	182	0	4	ECP3-70, ECP3-95, LFCPNX-100
XC6SLX150	147k	1358	4824	268	6	182	0	4	ECP3-95, ECP3-150

Table 1.9. AMD Spartan-6 LXT Devices and the Closest Lattice Devices Mapping

AMD Device	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	SerDes	DLL	18 x 18 Mult.	Embedded PCIe I/F	Embedded Mem. Cntl.	Closest Lattice Devices
XC6SLX25T	24k	228	936	52	2	2	38	1	2	ECP2M-20, ECP3-35, LFE5UM-25, LIFCL-17, LFD2NX-17, LFCPNX-50
XC6SLX45T	44k	401	2088	116	4	4	58	1	2	ECP3-35, ECP3-70, LFE5UM-45, LIFCL-40, LFD2NX-40, LFCPNX-50
XC6SLX75T	75k	692	3096	172	4/8	6	132	1	4	ECP3-70, LFE5UM-85, LFCPNX-100
XC6SLX100T	101k	975	4824	268	4/8	6	182	1	4	ECP3-95, LFCPNX-100
XC6SLX150T	147k	1358	4824	268	4/8	6	182	1	4	ECP3-95, ECP3-150

Table 1.10. AMD Spartan-3AN Devices and the Closest Lattice Devices Mapping

AMD Device	LUT4	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	User Flash (Mb)	Closest Lattice Devices
XC3S50AN	1.4k	1.6k	11	54	3	2	3	0.6	XO2-1200HC, XP2-5
XC3S200AN	3.6k	4.0k	28	288	16	4	16	3.0	XO2-2000HC, XO2-2000HE, XP2-5

AMD Device	LUT4	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	User Flash (Mb)	Closest Lattice Devices
XC3S400AN	7.2k	8.1k	56	360	20	4	20	2.4	XO2-7000HC, XO2-7000HE, XP2-8
XC3S700AN	11.8k	13.2k	92	360	20	8	20	5.8	XP2-17
XC3S1400AN	22.5k	25.3k	176	576	32	8	32	12.2	XP2-17, XP2-30

Table 1.11. AMD Spartan-3 Devices and the Closest Lattice Devices Mapping

AMD Device	LUT4	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	Closest Lattice Devices
XC3S50	1.5k	1.7k	12	72	4	2	4	XO2-1200HC, XO2-1200UHC, MachXO3L-1300
XC3S200	3.8k	4.3k	30	216	12	4	12	XO2-4000HC, XO2-4000HE, ECP2-6, MachXO3L-4300
XC3S400	7.2k	8.1k	56	288	16	4	16	XO2-7000HC, XO2-7000HE, ECP2-6, ECP2-12, MachXO3L-6900
XC3S1000	15.4k	17.3k	120	432	24	4	24	ECP2-20, ECP2M-20, ECP3-17, LFE5U-12, LFD2NX-17, LIFCL-17
XC3S1500	26.6k	30k	208	576	32	4	32	ECP2-20, ECP2M-20, ECP2-35, ECP2M-35, ECP3-35, LFE5U-25, LFD2NX-40, LIFCL-40
XC3S2000	41k	46k	320	720	40	4	40	ECP2-50, ECP2M-50, ECP3-35, LFE5U-45, LFD2NX-40, LIFCL-40
XC3S4000	55.3k	62.2k	432	1,728	96	4	96	ECP2-50, ECP2M-50, ECP2-70, ECP2M-70, ECP3-70, LFE5U-45, LFCPNX-50
XC3S5000	66.6k	74.9k	520	1,872	104	4	104	ECP2-70, ECP2M-70, ECP3-70, LFE5U-85

Table 1.12. AMD Spartan-3E Devices and the Closest Lattice Devices Mapping

AMD Device	LUT4	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	Closest Lattice Devices
XC3S100E	1.9k	2.2k	15	72	4	2	4	XO2-1200HC, XO2-1200UHC
XC3S250E	4.9k	5.5k	38	216	12	4	12	LP8K, MachXO3L-6900, ECP2-6

AMD Device	LUT4	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	Closest Lattice Devices
XC3S500E	9.3k	10.5k	73	360	20	4	20	LP8K, MachXO3L-6900, ECP2-6, ECP2-12, LFE5U-12
XC3S1200E	17.3k	19.5k	136	504	28	8	28	ECP2-20, ECP2M-20, ECP3-17, LFE5U-12, LFD2NX-17, LIFCL-17
XC3S1600E	29.5k	33.2k	231	648	36	8	36	ECP2-35, ECP2M-35, ECP3-35, LFE5U-25, LFD2NX-40, LIFCL-40

Table 1.13. AMD Spartan-3A Devices and the Closest Lattice Devices Mapping

AMD Device	LUT4	Logic Cells	Dist. RAM (kb)	EBR SRAM (kb)	EBR SRAM Blocks	DLL	18 x 18 Mult.	Closest Lattice Devices
XC3S50A	1.4k	1.6k	11	54	3	2	3	XO2-1200HC, XO2-1200UHC, ECP2-6, MachXO3L-1300
XC3S200A	3.6k	4.0k	28	288	16	4	16	XO2-4000HC, XO2-4000HE, ECP2-6, MachXO3L-4300
XC3S400A	7.2k	8.1k	56	360	20	4	20	XO2-7000HC, XO2-7000HE, ECP2-6, ECP2-12, MachXO3L-6900
XC3S700A	11.8k	13.2k	92	360	20	8	20	ECP2-12, ECP2-20, ECP3-17, LFE5U-12
XC3S1400A	22.5k	25.3k	176	576	32	8	32	ECP2-20, ECP2-35, ECP3-17, LFE5U-25, LFD2NX-17, LIFCL-17

Table 1.14. AMD Spartan-3A DSP Devices and the Closest Lattice Devices Mapping

AMD Device	LUT4	Logic Cells	Dist. RAM (kb)	EBRSRAM (kb)	EBR SRAM Blocks	DLL	Closest Lattice Devices
XC3SD1800A	33k	37k	260	1512	84	8	ECP3-35, ECP2-35, LFD2NX-40, LIFCL-40
XC3SD3400A	48k	54k	373	2268	126	8	ECP2-50, LFE5U-45, LFCPNX-50

1.3.2. CPLD Architecture

Complex programmable logic device (CPLD) is the old programmable logic device (PLD) architecture that is different from the FPGA device. Lattice is committed to offer a variety of products in this category that can replace the AMD discontinued parts, mainly the XC9500 and CoolRunner devices.

CoolRunner-II family of product is based on the AMD XC9500 and CoolRunner XPLA3 families. It ranges from 32 to 512 macrocells. The main difference between all these variants is the power, where Coolrunner-II is offering the lowest power option.

The main characteristic of a CPLD is its deterministic timing, where going from any input pin to any output pin takes 1 × tpd; where, tpd refers to the propagation delay time. The CPLD devices are characterized by how fast the tpd is. For example, the tpd can be 2.5 ns, 5 ns or 7.5 ns.

Figure 1.1 shows the architecture of the Lattice ispMACH™4000 family which has a similar architecture to the AMD CoolRunner.

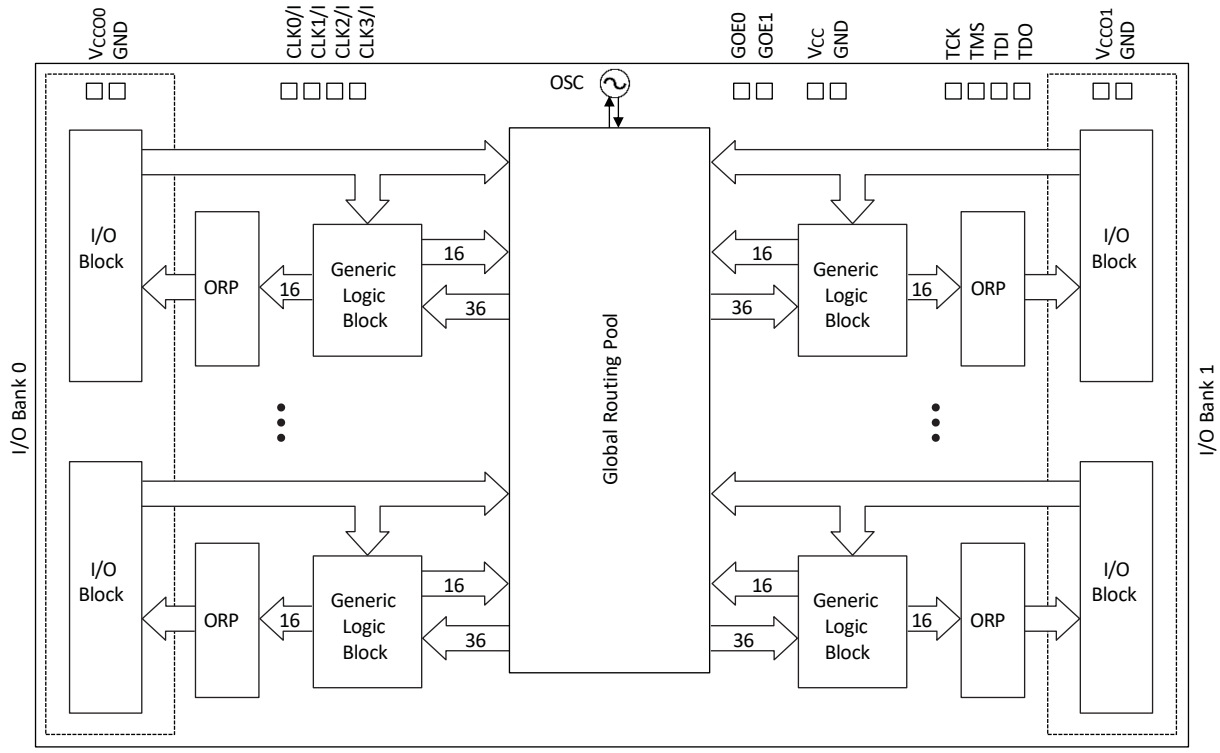


Figure 1.1. IspMACH4000 Family Architecture

The main building blocks of the ispMACH4000 family architecture are as follows:

- GRP: Global Routing Pool
  - AIM: Advanced Interconnect Matrix (CoolRunner)
  - ZIA: Zero-Power Interconnect Array (XPLA)
- GLB: Generic Logic Block
  - PLA: Programmable Logic Array
- ORP: Output Routing Pool
- I/O Banks

The ispMACH4000 is a newer family when compared to the XC9500/CoolRunner family; thus, offers better performance and architecture to support larger product term functions and enhanced IO flexibility. Table 1.15 lists the devices that could be selected to replace the AMD parts.

Table 1.15. AMD CoolRunner-II Devices and the Closest Lattice Devices Mapping

AMD Device	Macro-cells	Distributed RAM (kb)	EBR SRAM (b)	EBR SRAM Blocks	PLLs	Closest Lattice Devices
XC2C32A	32	—	—	—	—	XO2-256, XO2-640, ispMACH 4000V/Z 4032, ispMACH 4000V/Z 4064,
XC2C64A	64	—	—	—	—	
XC2C128	128	—	—	—	—	

AMD Device	Macro-cells	Distributed RAM (kb)	EBR SRAM (b)	EBR SRAM Blocks	PLLs	Closest Lattice Devices
XC2C256	256	—	—	—	—	ispMACH 4000V/Z 4128, ispMACH 4000V/Z 4256, ispMACH 4000V/Z 4512
XC2C384	384	—	—	—	—	
XC2C512	512	—	—	—	—	

1.3.3. Device Temperature Grades

The Lattice FPGA devices are typically offered in commercial, industrial, and automotive temperature grades. Automotive support is offered in a subset of devices, speed grade and package options. Refer to each family datasheet for accurate information.

- Commercial Temperature: (T_j = 0 °C – 85 °C)
- Industrial Temperature (T_j = –40 °C – 100 °C)
- Automotive Temperature (T_j = –40 °C – 125 °C)

Extended temperature range: some competitive devices offer an extended temperature range (T_j=0 °C – 100 °C), these could be replaced with an Industrial temperature range if the extended temperature range is needed. Keep in mind that the Lattice FPGA devices are very low power and will run cooler than the AMD devices.

On Lattice Certus device family, the on-die junction temperature can be monitored using the internal junction temperature monitoring diode. The proportional to absolute temperature (PTAT) diode voltage can be monitored by analog to digital converter (ADC) to provide a digital temperature readout. Refer to the ADC Usage Guide for Nexus Platform (FPGA-TN-02129) for more details.

1.4. Device Part Number and Speed Grade

Device part numbers are similar for both AMD and Lattice, they integrate several acronyms or indices that define the parameters of the device. Detailed meaning of each acronym is shown below.

An important factor is the device speed grade; for example, *-1*, *-2*, and *-3*. Faster devices will have a higher number. For example, *-3* device is faster than a *-2* device. The same logic is used with the Lattice devices; a *-9* device is faster than a *-8* device. The device has been tested at the factory to comply with the datasheet specifications for that speed grade.

Some of the AMD devices are offered in lower power versions coded with an *L* prefix; for example, L1, L2, and G2. Lattice devices are designed for low power and there is no special coding for low power version devices. Lattice Nexus FPGA platform uses 28 nm FD-SOI process technology and offers low power options of the same part number that is configurable at software level (same ordering part number).

1.4.1. AMD Part Number Description

Figure 1.2 describes the AMD 7 series part numbers (Artix-7, Kintex-7, and Vertex 7).

Example: Kintex-7 device with a part number of XC7K325T-2FBG900C.

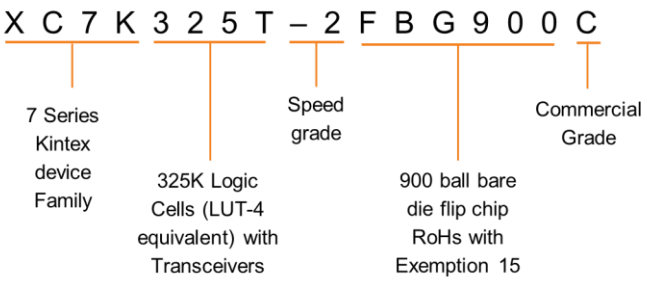


Figure 1.2. Kintex-7 Part Number Description

1.4.2. Lattice Part Number Description

Figure 1.3 describes the Lattice ECP5/ECP-5G part number and Figure 1.4 describes the Lattice CertusPro-NX part number.

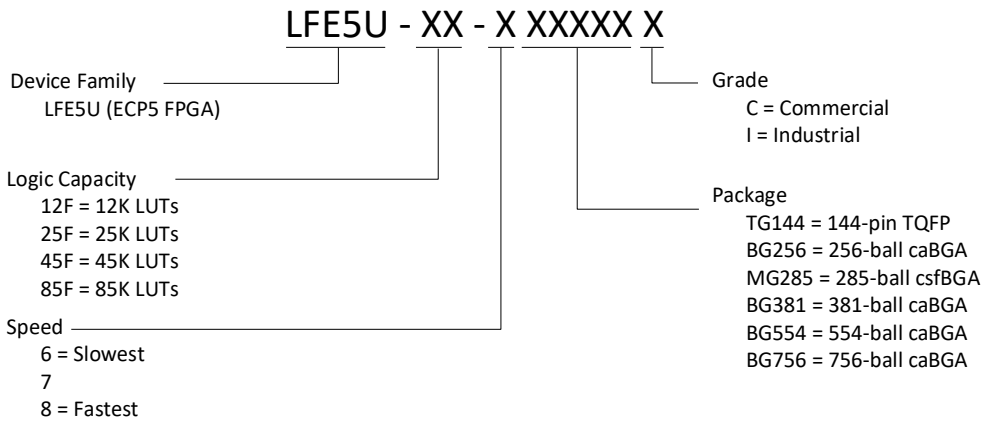


Figure 1.3. Lattice ECP5/ECP5-5G Part Number Description

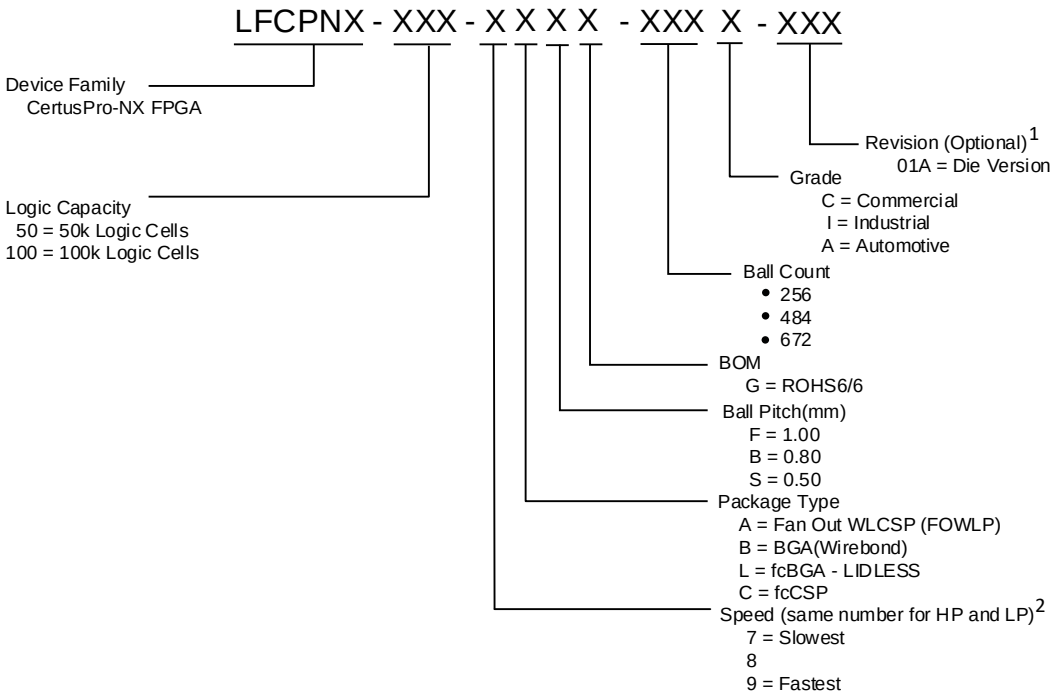


Figure 1.4. Lattice CertusPro-NX Part Number Description

1.4.3. Lattice Ordering Part Numbers Example

For example, CertusPro-NX family part number *LFCPNX-100-7ASG256C* would be described as follow:

- *LFCPNX*: device Family
- *100*: 100K Logic Cells (LUT4)
- *-7*: Speed grade
- *ASG256*: 256 ball WLCSP (Wafer level Chip Scale Package) with 0.5mm pitch
- *C*: Commercial Grade

2. Architecture Differences

The section provides you the overview and analysis of the differences between Lattice and AMD architecture focusing on the AMD 7-Series devices; namely the Spartan-7, Artix-7, Kintex-7, and Virtex-7, and older generation FPGA as well as their equivalent parts from Lattice Nexus and Avant device families.

This section coves the following topics:

- Old AMD terminology
- LUT4 versus LUT6
- I/O voltage and banks
- Available clocking resources
- Internal memory configuration
- External memory interface
- SerDes/Transceiver blocks
- DSP blocks
- Other hardened functions

2.1. Old AMD FPGA Terminology

Older AMD FPGA such as Spartan-II, Spartan-3, or Virtex II use LUT4 based architecture and have some specific terminology (shown in Table 2.1) that is discussed in this section to provide a way to compare them with the Lattice architecture elements.

Table 2.1. AMD Spartan-3 Device Architecture

AMD Device	XC3S50	XC3S200	XC3S400	XC3S1000	XC3S1500	XC3S2000	XC3S4000	XC3S5000
System Gate	50k	200k	400k	1000k	1500k	2000k	4000k	500k
Equivalent Logic Cells[1]	1728	4320	8064	17 280	29 952	46 000	62 208	74 880
CLB	192	480	896	1920	3328	5120	6912	8320
Calculated LUT4[2]	1.5k	3.8k	7.2k	15.4k	26.6k	41k	55.3k	66.6k

Notes:

1. Calculated LUT4 = Number of CLB × 8 (not listed in the datasheet).
2. Equivalent Logic Cells = Number of LUT4 × 1.125 (listed in the datasheet).

Further description of the terms used in Table 2.1 are as follows:

- System gates are used to name the devices as shown in the example below. System gate represents the device density in equivalent Application-Specific Integrated Circuit (ASIC) gates which is not fully representative of the device usable logic. In reality, all the architecture elements (such as RAM, Routing, and DSP) are counted in the system gate numbers.
- Logic Cells = 4-input Look Up Table (LUT) + a *D* flip-flop.
- Configurable Logic Block (CLB) is composed of 8 × Logic Cells
- Equivalent Logic Cells = CLB × 8 Logic Cells × 1.125. The multiplier factor (1.125 ×) is an effectiveness factor that stays as a gross marketing number.

When migrating design from these older AMD families, the equivalent logic cell numbers are comparable to Lattice logic cell terminology. In the absence of logic cell details, LUT4 number is also a good comparison metric.

The following example shows the comparison between the AMD Spartan-3 device and Lattice device:

XC3S200 is a Spartan 3 device family with 200k system gates.

To compare this device with Lattice offering, you need to refer to the number of logic cells available in the device.

In this case:

Number of logic cells = Number of CLB × 8 = 480 × 8 = 3840 logic cells (LUT4+FF)

Proper equivalent device can be XO2-4000HC/XO2-4000HE and XO3L-4300 with 4000 and 4300 LUT4 respectively.

2.2. LUT4 vs LUT6

2.2.1. Architecture Description and Differences

One of the main differences between Lattice and the new AMD architecture is the size of the basic element of the FPGA, which is the Lookup Table (LUT). A LUT6 is a 6-input memory block that is used to implement any logic function with 6 inputs. A LUT4 on the other hand implements any function with 4 inputs. In a design the number of inputs for a given logic equation varies a lot depending on how you write your HDL and how complex the logic is.

As an example, if you have a 10-input logic function, you need 2 LUT6 to implement your design whereas on a LUT4 based architecture it takes 3. In this specific case, you need 50% more LUT4 than LUT6 (3 LUT4 versus 2 LUT6). For details, refer to Figure 2.1.

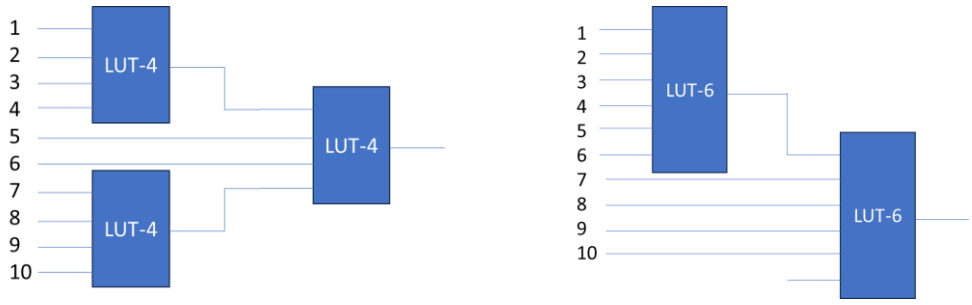


Figure 2.1. LUT4 Versus LUT6 Based Architectures for 10-input Logic Function

Silicon footprint for LUT4 and LUT6 is not the same. The number of transistors required to build a LUT6 are more than double the LUT4 implementation and LUT6 implementation will run slightly slower than LUT4.

Every design is different, and the number of levels of logic and pipelining will play a role in the overall design performance. Some designs may run faster in a LUT4 architecture, the other way around is also true.

To be able to compare both LUT4 and LUT6, AMD literature uses the term “Logic Cell” or “system logic cell” to calculate the equivalent LUT6 density of their devices. That is, 1 logic cell = 1.6 × number of LUT6.

Logic cell listed in the AMD literature represent the equivalent LUT4 device logic density. This terminology has been adopted since the move to LUT6 based architecture with Virtex 5, it is a way to compare logic resources to Virtex 4 (using LUT4).

It is obvious that a LUT6 can integrate more logic than a LUT4 as explained in the example above. AMD set that number to be 1.6 based on some benchmarking. In reality, the multiplier factor will change for every design and could only be used as an estimated number.

All the AMD FPGA devices after Virtex-4 have been adopting this multiplier for the naming of the device. Refer to Table 2.2 and the following example for more details:

Table 2.2. AMD Spartan-7 Device Architecture

AMD Device	XC7S6	XC7S15	XC7S25	XC7S50	XC7S75	XC3S100
Logic Cells	6000	12800	23360	52160	76800	102400
Slices	938	2000	3650	8150	12000	16000
Calculated LUT6*	3752	8000	14 600	32 600	48 000	64 000

***Note**: Calculated number of LUT6 = Number of slices × 4

The *XC7S6* device has *6000* logic cells and *938* number of slices.

Each 7 series FPGA slice contains four LUT6 and eight flip-flops.

The logic cell number listed is calculated using this equation:

Logic cells = Number of slices × Number of LUT6 per slice × 1.6

Logic cells = 938 × 4 × 1.6 = 6003

Hence, the *XC7S6* device name has 6000 logic cells.

System logic cell has been introduced with the AMD UltraScale device family. AMD changed their terminology to consider extra features added in UltraScale devices architecture. On a high level, these extra features can be summarized in the addition of dedicated inputs to the CLB flipflop, enabling wider Multiplexer (MUX) function, enhanced carry chain, and improved clock enable and reset.

All these enhancements allow more logic packing compared to original LUT4 in Virtex-4 architecture. It has been estimated that these features offer 20% – 30% more capacity for a given design. That is the justification of the *2.1875* multiplication factor used to convert *System Logic Cells* to Virtex-4 LUT4 equivalent similar to *Logic Cell* metric used in 7-Series devices. Refer to the following formula, Table 2.3, and example for more detail:

1 System logic cell = 2.1875 × Number of LUT6

Table 2.3. AMD Kintex UltraScale Device Architecture

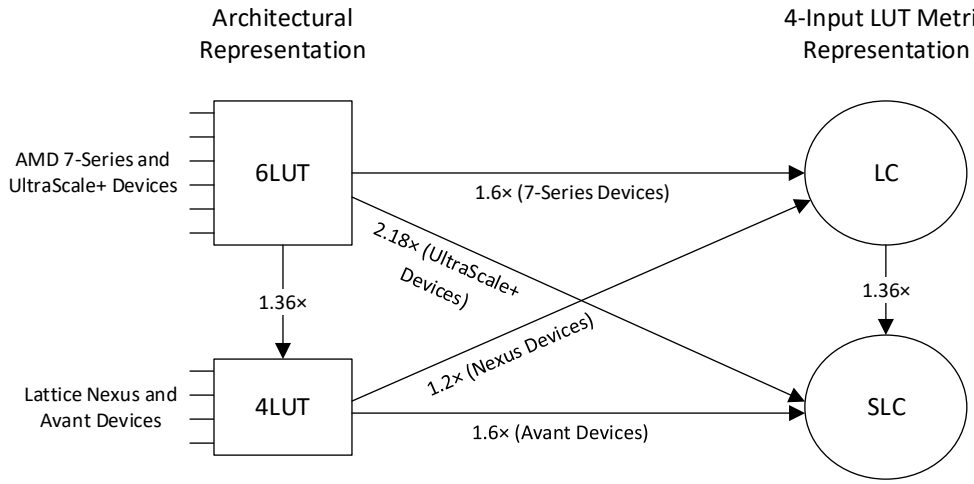
AMD Device	System Logic Cells (SLC)	CLB LUT
KU025	318	145 440
KU035	444	203 128
KU040	530	242 400

Number of LUT6 is listed as CLB LUT = 145 440 LUT6

Number of SLC = CLB LUT × 2.1875

Number of SLC = 145 440 × 2.1875 = 318 150 LUT4 Equivalent

Lattice also uses some multipliers to convert LUT4 to LC-logic cells (for Nexus Family) and SLC-System Logic Cells (for Avant Family). Figure 2.2 provides a summary of different multipliers that are used by both AMD and Lattice to move from the architectural representation to the 4-input LUT metric (LC or SLC).



Note:

These logic cell (LC) or system logic cell (SLC) multipliers, also called effectiveness factors in some literature, are rough estimate based on some benchmarking. The value of this effectiveness factor will always be design dependent.

Figure 2.2. AMD and Lattice Logic Cell and System Logic Cell Conversion Diagram

2.2.2. Architecture LUT Size is a Strategic Decision

Selecting LUT4 versus LUT6 is a strategic architectural decision and depends on multiple factors. LUT6 is suitable for very high-density FPGA devices to allow logic density integration and improve performance. It is also suitable for bleeding edge technology node.

The same architecture usually scales down to smaller devices. You can see this with the Ultrascale and 7-Series AMD devices.

Lattice strategic positioning focuses on low to mid-range density FPGA devices with an architecture and technology nodes that are suitable for the device density based on the offering provided.

LUT4 architecture is optimal for up to 500k LC at 350 MHz in 16 nm. Devices with LUT4 architecture, such as the Lattice Avant family, offers the required performance with lower power advantage.

LUT4 is lower power when compared to LUT6 because it has 4× less bits. LUT4 has 16 bits, whereas LUT6 has 64 bits. Less bit count is equivalent to less leakage. The decoding logic is also simpler with LUT4 and will consequently consume less power.

Figure 2.3 shows a high-level representation of a LUT4 versus LUT6 in logic gates. At the same process node, LUT6 could be up to 4 times bigger than LUT4 in terms of decoding logic associated with it. The footprint on the silicon is different based on the process node used and the ratio is also different. However, there is an advantage of using LUT4 for Lattice target class of devices which are small and mid-range FPGA devices.

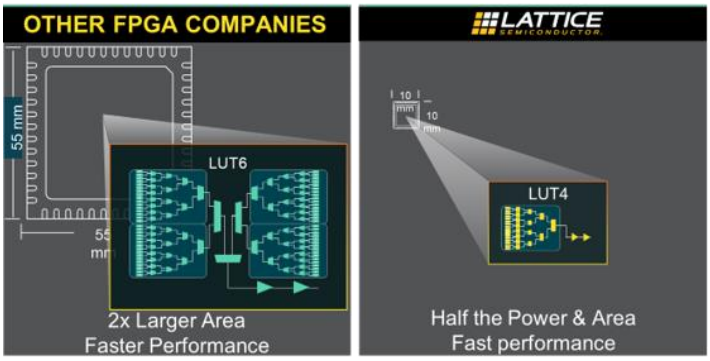


Figure 2.3. High-level Representation of LUT4 Versus LUT6 Logic Gates

2.2.3. Design Conversion Recommendations

From the design conversion perspective, there is not much to do to accommodate LUT4 or LUT6 architecture. Usually, the HDL code is agnostic and is not specific to either architecture. The synthesis tools used will convert your HDL code to the selected target technology library. For a high-performance design, you may need some pipelining techniques to optimize implementation for either structure.

In the case where you are using LUT to build distributed memory or shift register, be aware that a LUT4 is a 16-bit memory block whereas LUT6 is a 64-bit memory block. For more information, refer to the Internal Memory Configuration section.

2.3. I/O, Voltage, and Bank

2.3.1. Architecture Description and Differences

Both Lattice and AMD FPGA devices have configurable I/O with versatility of voltage support. Both group I/Os in banks with pre-determined size; for example, AMD 50 I/Os. Refer to each vendor for specific I/O related information and voltage standard support.

For more information on the Lattice CertusPro-NX device family (other Lattice device families have equivalent documents), refer to the following documents:

- sysI/O User Guide for Nexus Platform (FPGA-TN-02067)
- CertusPro-NX High-Speed I/O Interface (FPGA-TN-02244)
- Sub-LVDS Signaling Using Lattice Devices (FPGA-TN-02028)

There are differences in I/O, voltage and banks terminology used by Lattice and AMD which is summarized in Table 2.4. Lattice offers flexibility in voltage mixing within the same bank which can give an advantage in terms of usable pins per bank. For more details, refer to the Voltage Mix within the Same Bank section.

Table 2.4. I/O, Voltage, and Bank Terminology Comparison Between Lattice and AMD Devices

AMD Device	Lattice Device	Descriptions
V_{CCO}	V_{CCIO}	The bank I/O voltage.
V_{REF}	V_{REF}	The reference voltage per bank.
VCC_{AUX}	VCC_{AUX}	The auxiliary voltage.
HP I/O Banks	HPIO Banks or High performance I/O Banks (usually are the bottom banks)	The designated I/O banks that support high performance signals interface (up to 1.8 V I/O interface).
HR I/O Banks	WRIO Banks or Wide Range I/O Banks	Designated I/O banks that support up to 3.3 V I/O interface.

2.3.2. Voltage Mix within the Same Bank

Lattice devices, such as the Nexus platform devices, offer a flexible I/O architecture that allows voltage mixing within the same bank. For more details, refer to the *VCCIO Requirement for I/O Standards* section of the sysI/O User Guide for Nexus Platform (FPGA-TN-02067) document.

Table 2.5. Input Mixed Mode for Wide Range Input Buffers

V_{CCIO} (V)	Input Signaling (V)					
	LVCMOS10	LVCMOS12	LVCMOS15	LVCMOS18	LVCMOS25	LVCMOS33
	V_{CC} Powered Buffer		V_{CCAUX} Powered Buffer		V_{CCIO} Powered Buffer	
1.2	✓[2]	✓[2]	✓[1, 3]	—	—	—
1.5	✓[2]	✓[2]	✓[1, 3]	✓	—	—
1.8	✓[2]	✓[2]	✓[1, 3]	✓	—	—
2.5	✓[2]	✓[2]	✓[1, 3]	✓	✓	—
3.3	✓[2]	✓[2]	✓[1, 3]	✓	✓[3]	✓

Notes:
1. Increased ICC is due to underdrive.
2. No Hysteresis.
3. Reduced Hysteresis.

Table 2.5 shows that there are multiple compatible voltages for a given V_{CCIO} bank voltage. For example, any voltage standard can be an input for a V_{CCIO} of 3.3 V. If the number of I/O fits within the bank, only one bank is needed to implement the requirements. Voltage output will always follow the V_{CCIO}.

This I/O bank voltage mixing flexibility is not offered with AMD devices. For example, for the AMD 7-Series FPGA devices, the V_{CCO} conditions all the I/O voltages that are accepted in that bank for input and output.

2.3.2.1. Example Case

If your design must support LVCMOS12 and LVCMOS18 for a number of I/O, you may need to split them between 2 banks which make these 2 banks dedicated to these voltages or any compatible voltage. This will have an impact on the usable I/O for either bank. You may be losing pins because of these non-compatible voltages within the same bank.

For AMD 7-Series FPGA devices, there are requirements for each standard to be supported within the bank. The I/O voltages can be mixed if they share the same voltage requirements for V_{CCO}. In this case, if you want to support input for LVCMOS12, LVCMOS18, LVCMOS25, LVCMOS33 for your design, you will have to use 4 different banks of I/O.

2.3.3. Design Conversion Recommendations

When converting a design, make sure to pay attention to the voltage mix within the bank. Take advantage of the Bank I/O flexibility in the Lattice devices. You may be able to fit your design into smaller package options.

Use PCLK pins for clock input which have a direct connection to the clock routing or internal PLL. Even though the primary clock or the PLL can have its input from routing or any pin, it will inject extra delay which is not optimal. If the clock is single ended, use the PCLK (+) to connect your input.

Designated HPIO banks need to be used for high-speed interfaces including external memory interfaces. These banks have the required I/O architecture and clocking structure to run at higher speed than the regular I/O.

2.4. Clocking Resources

2.4.1. Clocking Architectures Comparison

AMD architecture offers Primary clocks, Region clocks for the fabric and I/O clocks. MMCM and CMT refer to the clock management and PLL. Table 2.6 lists the different resources available in the AMD 7-Series architecture and their equivalent with Lattice.

Table 2.6. Clocking Architecture Comparison Between Lattice and AMD Devices

AMD Clock Buffer	Descriptions	Lattice Equivalent
BUFG	32 global clock buffers	Up to 64 high fanout Primary clock (PCLK) Internal clock buffer is inferred automatically.
BUFH, BUFR, BUFMR	Horizontal, Region clocks Multi-region clock	Up to 64 high fanout Primary clock (PCLK) Internal clock buffer is inferred automatically.
BUFIO	I/O clock	Edge clock (ECLK) Internal clock buffer is inferred automatically.
BUFGMUX	Glitch less clock switch	DCS (Dynamic clock select)
BUFGCE	Clock gating buffer	DCC (Dynamic clock Control)
CMT	Clock management tile	PLL/DLL
MMCM	Mixed mode clock manager	PLL/DLL

Note: The clocking structure in the AMD UltraScale devices has been changed and the number of buffers has been reduced to 3 types only. The table information is still valid.

Lattice device clocking architecture includes optimized skew primary clocks that are distributed all over the fabric. Depending on the density and device family, the number of primary clocks vary. The clock network is divided into four clocking quadrants as shown in Figure 2.4: Top Left (TL), Bottom Left (BL), Top Right (TR), and Bottom Right (BR).

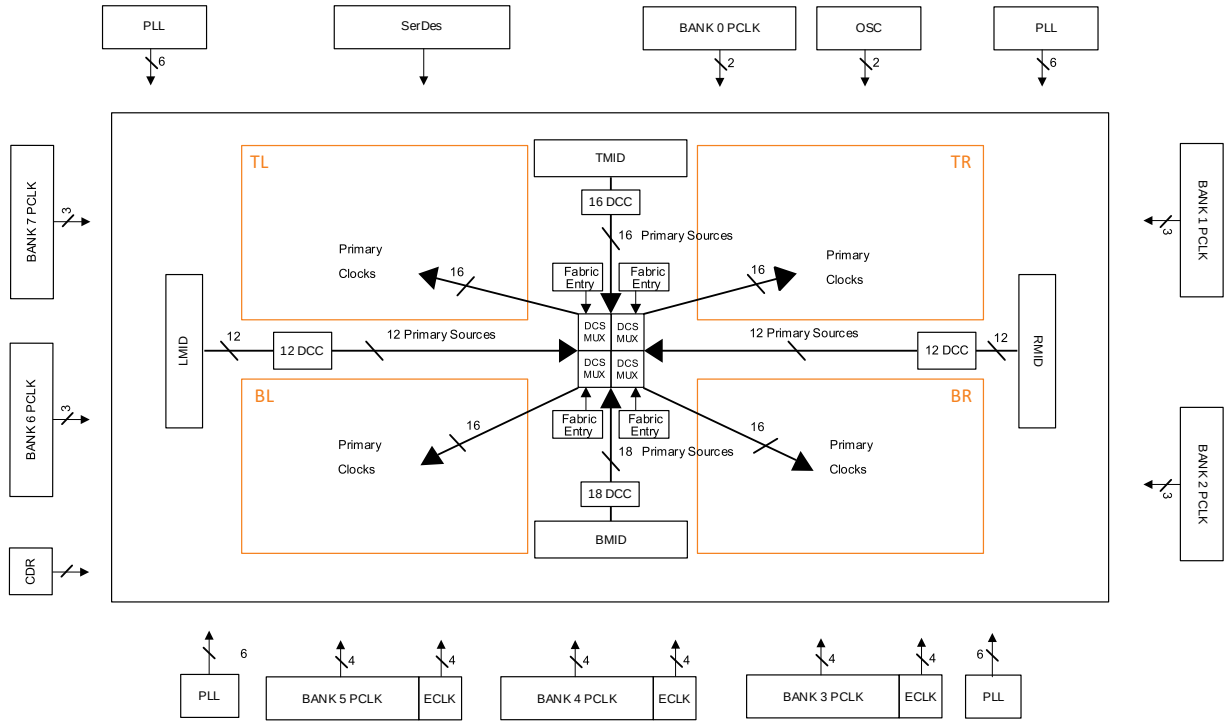


Figure 2.4. Lattice CertusPro-NX Clocking Structure

Each of these quadrants has 16 primary clocks that can be distributed to the fabric in the quadrant. The Lattice Diamond software can automatically route each clock to one of the four quadrants up to a maximum of 16 clocks per quadrant. You can change how the clocks are routed by specifying a preference in the Lattice software (USE_PRIMARY).

Refer to the specific device datasheet for details about clocking structure of each device. For more information on the Lattice CertusPro-NX device family (other Lattice device families have equivalent documents), refer to the following documents:

- CertusPro-NX Family Data Sheet (FPGA-DS-02086)
- sysCLOCK PLL Design and User Guide for Nexus Platform (FPGA-TN-02095)

2.4.2. Design Conversion Recommendations

Software tools assign clocks based on the signal load and connectivity defined in the HDL code. Check for the following when converting your design:

- Replace the clock buffer with a single net and verify that primary clocks are allocated properly. If not, use design constraint: USE_PRIMARY to force the software to implement a specific clock on the primary clock routing.
- Replace primitives with Lattice equivalent for BUFGMUX and BUFGCE. For more information, refer to the HDL Attributes section for the HDL code conversion examples.
- Make sure you are using the PCLK pins for your clock input. They have the most optimal path to the clock network, PLLs or edge clocks.

2.5. PLL/MMCM/DCM

Both AMD and Lattice offer clock management modules. The names of these entities differ. Lattice offer comparable functionalities using equivalent modules as described in Table 2.7.

Table 2.7. Clock Management Module Comparison Between Lattice and AMD Devices

AMD Device	Lattice Device	Descriptions
CMT	GPLL/DLL	CMT (Clock Management Tile) includes a mixed-mode clock manager (MMCM) and a phase-locked loop (PLL). The Lattice equivalent is a PLL or DLL and other clocking resources within the architecture.
MMCM	GPLL/DLL	MMCM is a Mixed-Mode Clock Manager. The Lattice equivalent is a PLL or DLL.
DCM	GPLL/DLL	Older generation AMD devices use the DCM (Digital Clock Management) which is replaced by the CMT and MMCM in newer generation devices. The Lattice equivalent is a PLL or DLL.

Both the AMD UltraScale and 7-series FPGA devices integrate the CMT that includes a MMCM and one PLL for 7-Series devices or two PLLs for UltraScale devices.

The devices in the Lattice CertusPro-NX family support three to four full-featured general purpose PLLs (GPLLs), each with up to 6 different outputs. In addition, the dedicated DDRDLL units are also available for high-speed interfaces including external memory interfaces.

Refer to the specific device datasheet for more details about the clocking resources that are available for each device. For more information on the Lattice CertusPro-NX device family (other Lattice device families have equivalent documents), refer to the following documents:

- CertusPro-NX Family Data Sheet (FPGA-DS-02086)
- sysCLOCK PLL Design and User Guide for Nexus Platform (FPGA-TN-02095)

2.5.1. Architecture Primitives Comparison

Unlike Lattice, where only one primitive could be parameterized to define the PLL operation and features, AMD propose two types of MMCM units: The **MMCME#_ADV** primitive provides access to all **MMCME#_BASE** features plus additional ports for clock switching, access to the dynamic reconfiguration port (DRP), dynamic fine-phase shifting and spread spectrum support.

Table 2.8 describes the AMD primitives and the equivalent ports from Lattice.

The AMD primitives are as listed below:

- 7-Series devices: MMCME2_BASE, MMCME2_ADV, PLLE2_BASE, PLLE2_ADV
- UltraScale devices: MMCME3_BASE, MMCME3_ADV, PLLE3_BASE, PLLE3_ADV, PLLE4_BASE, PLLE4_ADV

The Lattice primitives are as listed below:

- LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, and UT24CP device families: PLL/PLLA
- Avant device family: PLLC
- iCE40 UltraPlus device family: PLL_B

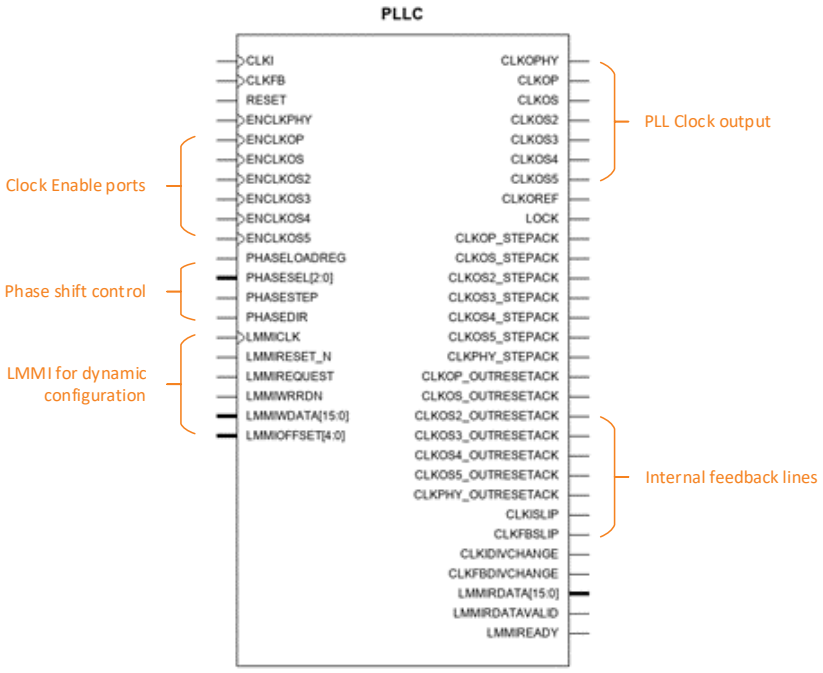


Figure 2.5. PLLC Functional Block

Table 2.8. Architecture Primitive Comparison Between Lattice and AMD Devices

AMD Device	Lattice Device	Descriptions
CLKIN1, CLKIN2	REFCK	The general clock input. For Lattice device, only one input is available for the PLL. You can use the DCS primitive to MUX clock inputs.
CLKFBIN	CLKFB	For AMD device, CLKFBIN refers to the feedback clock input. For Lattice device, the PLLREFCS primitive is used to support the dynamic reference clock switching using the SEL signal.
CLKINSEL	N/A	For AMD device the primitive signal controls the state of the clock input MUX: CLKIN1/2. For Lattice device, you can use the DCS unit to MUX clocks to the PLL.
RST	RST	The asynchronous reset signal. For Lattice, set *active high* to reset the PLL.
DADDR[6:0], DI[15:0], DWE, DEN, DCLK, DO[15:0], DRDY	LMMI_CLK, LMMI_OFFSET[6:0], LMMI_REQUEST, LMMI_RESETN, LMMI_WDATA[7:0], LMMI_WR_RDN, LMMI_RDATA[7:0], LMMI_RDATA_VALID, LMMI_READY	The dynamic reconfiguration interface signals. For Lattice device, the LMMI or APB interface is used for dynamic reconfiguration. Refer to the sysCLOCK PLL Design and User Guide for Nexus Platform (FPGA-TN-02095) or the document for respective family for details of operation. APB serial interface can also be used. The register map is identical to the LMMI.

AMD Device	Lattice Device	Descriptions
CLKOUTx, CLKOUTxB	CLKOP, CLKOSx	The AMD primitives are referring to the clock output and its inverted version. For Lattice device, the inverted version of the clock could be done at the clock tree level.
CLKFBOUT, CLKFBOUTB	Refclk	The dedicated MMCM feedback output and its inverted version. For Lattice device, the inverted version of the clock could be done at the clock tree level.
LOCKED	LOCK	The status of the PLL.
PWRDWN	PLLPD_EN_N	The AMD primitive is used to power down the MMCMs. For Lattice device, the same feature exist with the PLLPD_EN_N signal.
PSEN, PSINCDEC, PSCLK, PSDONE	PHASESEL[2:0], PHASEDIR, PHASESTEP, PHASELOADREG	The phase shift control signals.

2.5.2. CMT Features Comparison

UltraScale MMCMs are similar to the MMCM in the AMD 7-Series devices. PLLs are considered as a subset of MMCM with reduced features. Table 2.9 summarize the difference between AMD MMCM, PLL and their Lattice equivalent replacement unit.

Table 2.9. CMT Features Comparison Between Lattice and AMD Devices

CMT Features	AMD Device MMCM	AMD Device PLL	Lattice Device PLL (CertusPro-NX)
Input Frequency Range	10 MHz – 800 MHz	19 – 800 MHz	10 MHz – 500 MHz
Output Frequency Range	4.69 MHz – 800 MHz	6.25 – 800 MHz	6.25 MHz – 800 MHz
VCO Frequency	600 MHz – 1600 MHz	800 – 2133 MHz	800 MHz – 1600 MHz
Spread Spectrum	Yes	No	Yes (20 kHz – 200 kHz)
Phase Shift	Yes	Yes	Yes
Number of outputs	8 outputs per MMCM	7 outputs per PLL	6 outputs per PLL (CLKOP, CLKOS, CLKOS2, CLKOS3, CLKOS4, and CLKOS5)
Fractional Divide	Yes	Yes	Yes
Programmable and dynamic phase control	Yes	Yes	Yes
Register interface	AXI Bus	AXI Bus	LMMI or APB

The Nexus PLL supports Spread spectrum clock generation through Lattice Radiant™ IP Catalog. The spread spectrum function is integrated with the Fractional-N controls and supports Centered Spread or Down Spread, triangle wave, 0.25% per step from 1.00% to 2.00% with modulation frequency range from 24.42 kHz to 200 kHz.

2.5.3. Design Conversion Recommendations

To convert AMD MMCM and PLL, it is recommended to use the IP Catalog or IP Express to generate equivalent PLL unit for your design.

Note that using the AMD IP Catalog gives you the option to enable differential buffer when configuring the MMCM or PLL. This automatically generates a PLL with differential clock input buffer (IBUFDS).

Lattice does not offer the differential buffer generation option at the PLL interface level. In general, all design signals within the HDL code are defined as single ended pin. The pin type could be assigned at the constraint editor for single ended or differential type. When locking a differential pin on the constraint editor, the complement is automatically reserved.

Lattice offers the option of instantiating a differential buffer in the HDL code if desired (use DIFFCLKIO primitive for LFCPNX and UT24CP). For details, refer to Figure 2.6.

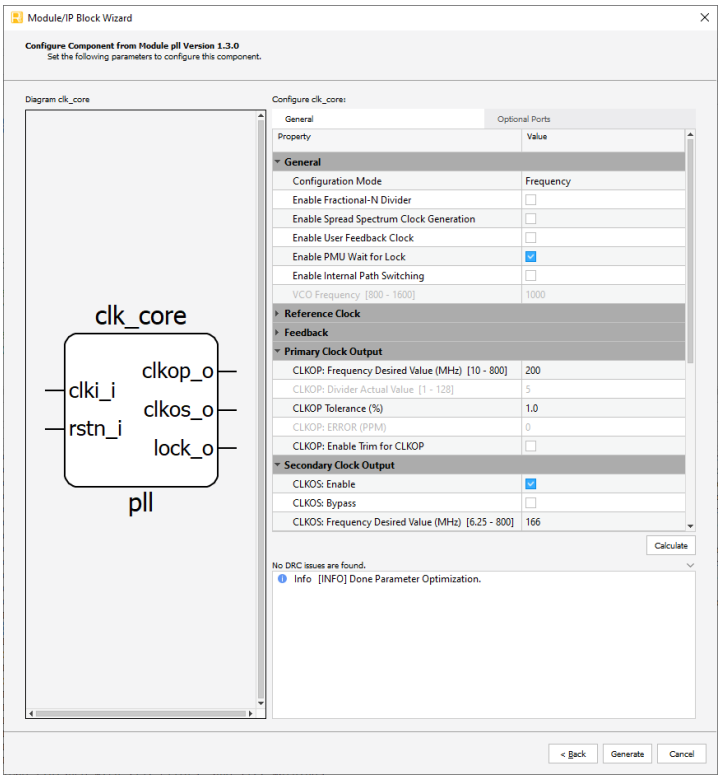


Figure 2.6. Lattice IP Catalog for a PLL Configuration Interface

2.6. Internal Memory Configuration

2.6.1. Embedded and Distributed Memory Comparison

This section compares the difference of architecture between Lattice and AMD internal memories. Both architectures integrate distributed and embedded blocks of RAM. For a small buffering, distributed memory is used. Bigger memory is implemented in the RAM blocks. Both types could be inferred or instantiated in the code.

If the memory is coded in the HDL code, the synthesis tool will automatically infer either distributed or EBR, based on the size of the memory. You can also force one or the other implementation by using the *syn-ramstyle* synthesis attribute. The *syn_ramstyle* attribute specifies the implementation to use for an inferred RAM. You can apply *syn_ramstyle* globally to a module or a RAM instance.

The following values can be specified globally or on a module or a RAM instance:

- *Registers* – Causes an inferred RAM to be mapped to registers (flip-flops and logic) rather than the technology-specific RAM resources. This implementation is not resource efficient. If your RAM resources are limited, you can use this attribute to map additional RAMs to registers instead of the dedicated or distributed RAM resources.
- *Distributed* – Causes the RAM to be implemented using the distributed RAM or PFU resources.
- *Block_ram* – Causes the RAM to be implemented using the dedicated RAM blocks or EBR.

Both Lattice and AMD architectures allow a certain percentage of the available LUTs to be configured as small memory blocks. These are usually used in complementarity with Block RAM (EBR) to implement small shift register, buffering or to store small amount of initialization data or filter coefficients.

Since AMD use a LUT6 architecture, each LUT6 can be converted to a 64×1 memory element (512 bits: 64×8 by AMD CLB).

For Lattice architecture a LUT4 is used which translates to a 16×1 memory element. Each Lattice PFU can be converted to a 16×4 memory block (64 bit per PFU). Only slice 0 and slice 1 are available for distributed memory. Slice 2 is used to provide memory address and control signals. Refer to the specific device datasheet for more details.

Multiple PFUs could be cascaded to generate a larger memory. You can use the Lattice software IP generation tool to generate such memory.

The source code provided in the Source HDL Code Example section is not specific to Lattice or AMD devices. It can be targeted to either one with no changes required. You may only need to adjust the attributes to guide the implementation of the memory (EBR or Distributed).

Refer to the following synthesis tool usage guides for more details about the attributes:

- Reference directory: **Reference Guides** > **Constraints Reference Guide** > **Lattice Synthesis Engine Constraints** > **Lattice Synthesis Engine-Supported HDL Attributes** > **syn_ramstyle**
- Lattice Reference Manual in the Synplify Pro® installation directory.

More details about this subject are discussed in the Attributes Conversion Examples section.

Table 2.10 provides a summary of different resources available for each architecture.

Table 2.10. RAM Type Comparison Between Lattice and AMD Devices

RAM Type	AMD Device	Lattice Device	Descriptions
Distributed Memory	LUT6 based	LUT4 based	The LUT6 has 64-bit memory and LUT4 has 16-bit memory.
Block RAM Size	18 kb/36 kb	18 kb	The Lattice Avant device offers a 32 kb block RAM size. You can use the Lattice software (IP Catalog) to generate the equivalent module and select the required options. Port names may differ, refer to Table 2.15 for more details.
Single Port	Yes	Yes	
Pseudo Dual Port	Yes	Yes	
True Dual Port	Yes	Yes	
ROM	Yes	Yes	
FIFO Dual Clock	Yes	Yes	
Memory initialization	Yes	Yes	You can initialize memories to all *1*, *0*, or provide a custom initialization memory file.
Power Up Condition	0	0	The default value is *0*.

Note: Older generation devices have smaller block RAM sizes.

2.6.2. Source HDL Code Example

The following code is an example of a source HDL code:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
library synplify;

entity ram is
generic (
  addr_width : natural := 4;
  data_width : natural := 8);
port (
  addr : in  std_logic_vector (addr_width - 1 downto 0);
  write_en : in  std_logic;
  clk : in  std_logic;
  din : in  std_logic_vector (data_width - 1 downto 0);
  dout : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
 type mem_type is array ((2** addr_width) - 1 downto 0) of
    std_logic_vector(data_width - 1 downto 0);
signal mem : mem_type;
    -- Define RAM as an indexed memory array.
attribute syn_ramstyle : string;
--attribute syn_ramstyle of mem : signal is "block_ram";
--attribute syn_ramstyle of mem : signal is "distributed";
```

```
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then --Control with clock edge
      if (write_en = '1') then -- Control with a write enable.
              mem(conv_integer(addr)) <= din;
      end if;
       dout <= mem(conv_integer(addr));
    end if;
  end process;
end rtl;
```

2.6.3. Large Memory Blocks

Large block RAMs are available in the UltraScale+ devices which are called the UltraRAM blocks. These are large blocks of RAMs meant to serve as additional memory resources beyond what is available in the EBR and PFU for certain video or communication applications which require a large amount of memory.

Lattice devices has an equivalent Large RAM (LRAM) block with some differences which are listed in Table 2.11. Note that even smaller density Lattice devices integrate the LRAM blocks.

Table 2.11. LRAM and UltraRAM Features Comparison Between Lattice and AMD

Large RAM Features	AMD Device	Lattice Device	Descriptions
Name	UltraRAM	LRAM	LRAM is offered on the CertusPro-NX devices.
Data width	Fixed (72-bits)	Configurable (1-64 bits)	Refer to the Memory User Guide for Nexus Platform (FPGA-TN-02094) for more details.
Block RAM Size	288 kb	512 kb	—
Single Port	Yes	Yes	UltraRAM can only support read or write per port per cycle (not a real dual port). It has a fixed read behavior; there are no user definable read-first, write-first, no-change like with EBR. Lattice devices have more flexibility. You can configure the LRAM operation based on attributes like EBR. Refer to the Read and Write Priority section for more information.
Pseudo Dual Port	Single clock	Single clock	
True Dual Port	Single clock	Single clock	
ROM	No	Yes	UltraRAM does not support ROM mode.
FIFO	No	No	No native support for both devices.
ECC support	Yes	Yes	Both architectures provide ECC module with Single Error Correction - Double Error Detection (SECDED) capability.
Memory initialization	No	Yes	You can initialize memories to all *1*, *0*, or provide a custom initialization memory file.
Power up condition	0	0	The default value is *0*.

2.6.4. Read and Write Priority

To avoid data collision and better control the memory behavior, any port which has both Read access and Write access has a Write Mode attribute. Both AMD and Lattice devices offer this control.

For AMD devices, this attribute is available for Port A in the Single Port Mode and for both Port A and Port B in True Dual-Port Mode. In Pseudo Dual-Port and ROM Modes, no Write Mode attribute is available as there are no ports with both Read access and Write access.

For Lattice devices, there are three possible values for Write Mode attribute: Normal, Write Through, and Read Before Write. All three modes are supported in the Single Port LRAM, while only Normal and Write Through are supported in the True Dual-Port LRAM.

The description of each value is as follows:

• In the Normal mode, the output data is not changed nor updated during the write operation.

• In the Write Through mode, the output data is updated with the input data during the write cycle.

• In the Read Before Write mode, the output data port is updated with the existing data stored in the write address, during a write cycle. This mode is supported only in the Single Port LRAM.

For more information, refer to the Memory Modules User Guide (FPGA-IPUG-02033).

Table 2.12 provides AMD device memory attributes and their Lattice device memory attributes equivalent.

Table 2.12. Memory Attribute Comparison Between Lattice and AMD Devices

AMD Device Attribute	Lattice Device Attribute	Descriptions
NO_CHANGE Mode	Normal mode (default)	Data output remains the last read data and is unaffected by a write operation on the same port.
READ_FIRST or Read-Before-Write Mode	Read Before Write mode	This mode is supported only in the Single Port LRAM.
WRITE_FIRST or Transparent Mode (Default)	Write Through mode	The output data is updated with the input data during the write cycle.

When the Memory IP module is generated by the IP Catalog, the default parameter is used. You can change it to the desired value. Below is a DPRAM IP example for a CertusPro-NX device in Verilog.

The module wrapper, will have the WRITE_MODE default parameter *normal* for both ports A and B. You can force these values to the desired behavior by changing this parameter.

.WRITE_MODE_A("normal") example:

```
module TDPRAM_test (
        clk_a_i,
        clk_b_i,
        rst_a_i,
        rst_b_i,
        clk_en_a_i,
        clk_en_b_i,
        wr_en_a_i,
        wr_en_b_i,
        wr_data_a_i,
        addr_a_i,
        rd_data_a_o,
        wr_data_b_i,
        addr_b_i,
        rd_data_b_o) ;
    input clk_a_i ;
    input clk_b_i ;
    input rst_a_i ;
    input rst_b_i ;
    input clk_en_a_i ;
    input clk_en_b_i ;
    input wr_en_a_i ;
    input wr_en_b_i ;
    input [17:0] wr_data_a_i ;
    input [8:0] addr_a_i ;
    output [17:0] rd_data_a_o ;
    input [17:0] wr_data_b_i ;
    input [8:0] addr_b_i ;
    output [17:0] rd_data_b_o ;
```

```
    parameter MEM_ID = "TDPRAM_test" ; TDPRAM_test_ipgen_lscc_ram_dp_true #(.FAMILY("LFCPNX"),
            .MEM_ID(MEM_ID),
            .MEM_SIZE("18,512"),
            .ADDR_DEPTH_A(512),
            .DATA_WIDTH_A(18),
            .ADDR_DEPTH_B(512),
            .DATA_WIDTH_B(18),
            .ADDR_WIDTH_A(9),
            .REGMODE_A("reg"),
            .RESETMODE_A("sync"),
            .RESET_RELEASE_A("sync"),
            .REGMODE_B("reg"),
            .RESETMODE_B("sync"),
            .RESET_RELEASE_B("sync"),
            .BYTE_ENABLE_A(0),
            .BYTE_SIZE_A(9),
            .BYTE_SIZE_B(9),
            .BYTE_WIDTH_A(2),
            .BYTE_ENABLE_B(0),
            .WRITE_MODE_A("normal"),
            .WRITE_MODE_B("normal"),
            .BYTE_WIDTH_B(2),
            .ADDR_WIDTH_B(9),
            .ECC_ENABLE(0),
            .INIT_MODE("none"),
            .INIT_FILE("none"),
            .INIT_FILE_FORMAT("hex"),
.INIT_VALUE_00("0x0000000000000000000000000000000000000000000000000000000000000000"),
….
.INIT_VALUE_3F("0x0000000000000000000000000000000000000000000000000000000000000000"))

lscc_ram_dp_true_inst (
.clk_a_i(clk_a_i),
                .clk_b_i(clk_b_i),
                .rst_a_i(rst_a_i),
                .rst_b_i(rst_b_i),
                .clk_en_a_i(clk_en_a_i),
                .clk_en_b_i(clk_en_b_i),
                .wr_en_a_i(wr_en_a_i),
                .wr_en_b_i(wr_en_b_i),
                .wr_data_a_i(wr_data_a_i[17:0]),
                .addr_a_i(addr_a_i[8:0]),
                .rd_data_a_o(rd_data_a_o[17:0]),
                .wr_data_b_i(wr_data_b_i[17:0]),
                .addr_b_i(addr_b_i[8:0]),
                .ben_a_i(2'b11),
                .ben_b_i(2'b11),
                .rd_data_b_o(rd_data_b_o[17:0]),

Endmodule ;
```

2.6.5. Memory Size and Configuration

Table 2.13 lists the maximum amount of memory available per Lattice device family for EBR and LRAM.

Table 2.13. Maximum Memory Available per Lattice Device Family

Device Family/ Max. Memory	iCE40 UltraPlus	Mach-NX/MachXO/ MachXO2/ MachXO3/ MachXO3D/MachXO5-NX	ECP5/ECP5-5G	CrossLink/ CrossLink-NX	CertusPro-NX	Avant
Block RAM	0.12 Mb	1.4 Mb	3.7 Mb	1.1 Mb	3.7 Mb	35.6 Mb
Large RAM Blocks	1 Mb	0.5 Mb	—	2.5 Mb	3.5 Mb	—
Distributed Memory	—	0.184 Mb	0.64 Mb	0.24 Mb	0.64 Mb	4.14 Mb
UFM	—	15.36 Mb	—	—	—	—

Notes:

- Large RAM blocks are offered in smaller density devices.
- Memory driven design may be able to fit in smaller devices.

2.6.6. Memory Primitives Comparison

AMD 7-Series device memory blocks are 36 kb and can be split in two 18 kb. The Lattice device EBR are 18 kb each for the Nexus device family and 32 kb for the Avant device family. Table 2.14 lists the different primitives that you may see with AMD device architecture and potential Lattice device equivalent primitives for Nexus devices.

Table 2.14. Memory Primitives for Lattice and AMD Devices

Primitive	Device	Descriptions
RAMB18E1	AMD Device	The RAM_MODE attribute determines the mode of the RAM block, either the SDP mode or true dual-port (TDP) mode.
RAMB36E1	AMD Device	
FIFO18E1	AMD Device	
FIFO36E1	AMD Device	
URAM288_BASE	AMD Device	
DP16K	Lattice Device	16 kb Dual Port Block RAM
DPR16X4	Lattice Device	Distributed Pseudo Dual Port RAM with Synchronous Write and Asynchronous Read
DPSC512K	Lattice Device	512 kb Single Clock Dual Port Block RAM
FIFO16K	Lattice Device	16 kb FIFO
PDP16K	Lattice Device	16 kb Pseudo Dual Port Block RAM
PDPSC16K	Lattice Device	16 kb Pseudo Dual Port Single Clock Block RAM
PDPSC512K	Lattice Device	512 kb Single Clock Pseudo Dual Port Block RAM
SP16K	Lattice Device	16 kb Single Port Block RAM
SP512K	Lattice Device	512 kb Single Port Block RAM
SPR16X4	Lattice Device	Distributed Single Port RAM with Synchronous Write and Asynchronous Read

Figure 2.7 shows the block diagrams of Lattice DP16K and FIFO16K primitives for Nexus platform devices.

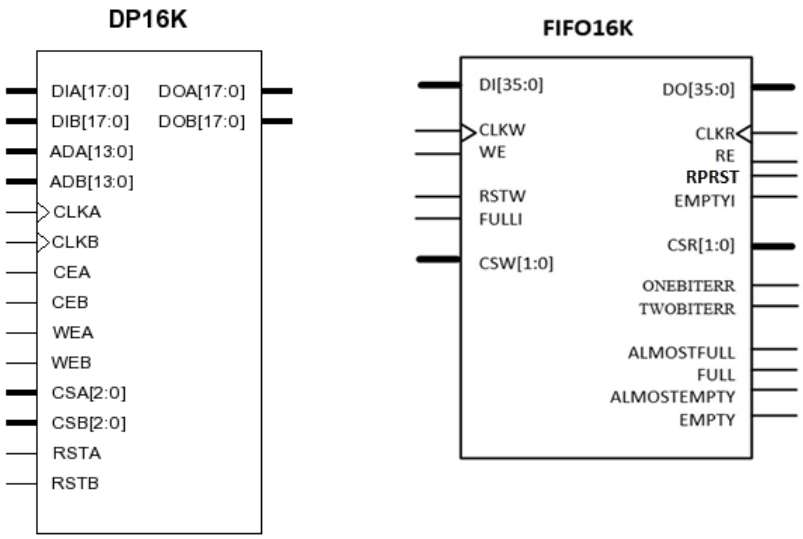


Figure 2.7. Lattice DP16K and FIFO16K Primitives for Nexus Platform Devices

Table 2.15 provides detailed differences in port naming between Lattice FPGA devices, which is mainly for the CertusPro-NX devices, and the AMD 7-Series/UltraScale devices.

Table 2.15. Port Naming Comparison Between Lattice and AMD Devices

AMD Device	Lattice Device	Descriptions
DI[A|B]	DI[A|B]	Data input bus. DIP parity bus. Can be used for additional data inputs. Lattice DI attribute can be used for both DI and DIP.
DIP[A|B]	DI [A|B]	
ADDR[A|B]	AD[A|B]	Address bus.
WE[A|B]	WE[A|B]	Byte-wide write enable.
EN[A|B]	CS[A|B]	When inactive no data is written to the block RAM and the output bus remains in its previous state.
RSTREG[A|B]	RST[A|B]	Synchronous Set/Reset the output registers (DO_REG = 1). The RSTREG_PRIORITY attribute determines the priority over REGCE.
RSTRAM[A|B]	RST[A|B]	Synchronous Set/Reset the output data latches.
CLK[A|B]	CLK[A|B]	Clock input.
DO[A|B]	DO[A|B]	Data output bus. Data output parity bus. Can be used for additional data outputs. Lattice DO attribute can be used for both DO and DOP.
DOP[A|B]	DO[A|B]	
REGCE[A|B]	CE[A|B]	Output Register clock enable.
CASCADEIN[A|B]	N/A	Cascade input for 64k × 1 mode. Lattice device uses fabric routing to implement this. It is transparent to you.
CASCADEOUT[A|B]	N/A	

2.6.7. Design Conversion Recommendations

When converting a memory block from AMD device architecture to Lattice device architecture, you can refer to the following recommendations:

- Memory Primitives between the two architectures may be too different, it is best to use the IP Express or IP Catalog to generate a Lattice equivalent memory. Define the size, parameter and attributes using the knowledge acquired in this section to have an equivalent functional memory.
  - Memory types: SPRAM, DPRAM, PDPRAM, FIFO, and ROM.
  - Memory sizes: Address and data bus.
  - Enable Input/Output registers as needed.
  - Enable *Byte enable* if needed.

- Reset: Synchronous or Asynchronous
- Memory initialization values.
- Enable ECC if required.
- Set WRITE_MODE operation. Note that the default value differs between AMD and Lattice devices.

- The Lattice IP generated should have equivalent ports.
- Choose the same IP module name to minimize source code changes.
- In the source code, change the module instantiated with the one generated. Or create a wrapper that maps the names of the Lattice module to the AMD module.

Note: When using memory element in your design for filter implementation or other use cases, the latency of data in and out of the memory is important for the proper functionality of the design. Having a different latency of data out of the memory element could impact the logic integrity. Latency could be caused by pipeline registers that you enable for the generated module.

2.6.7.1. Example Case

When you use a FIFO in a digital FIR filter implementation, latency is compensated in the design to have the right filter logic implemented.

When you generate a FIFO, you have the options to enable the output register and high-speed implementation. Enabling these options will add extra latency, but the design will have a higher performance and the Memory Tco will be faster. For a proper conversion, you need to keep latency consistent with the original design. Figure 2.8 shows the FIFO configuration interface of the Lattice Radiant IP catalog.

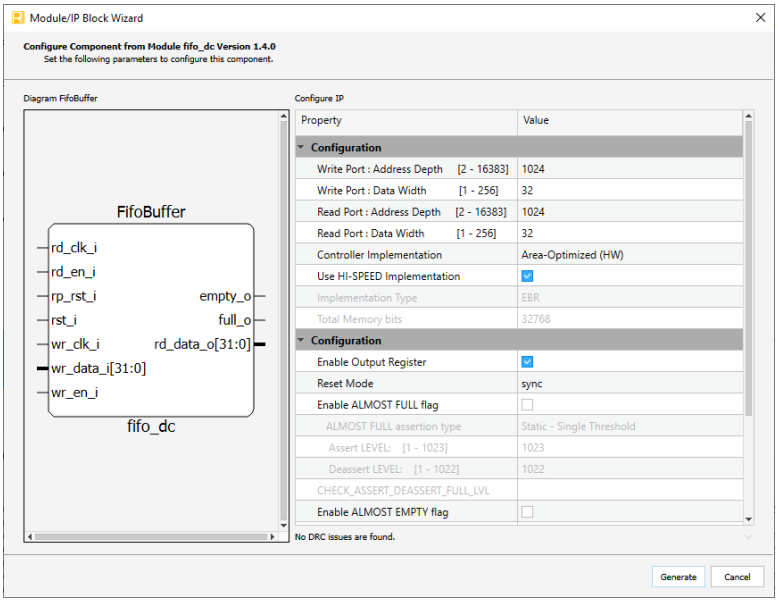


Figure 2.8. FIFO Configuration Interface of Lattice Radiant IP Catalog

The difference in behavior between enabling and disabling the output register is illustrated in the waveforms in Figure 2.9.

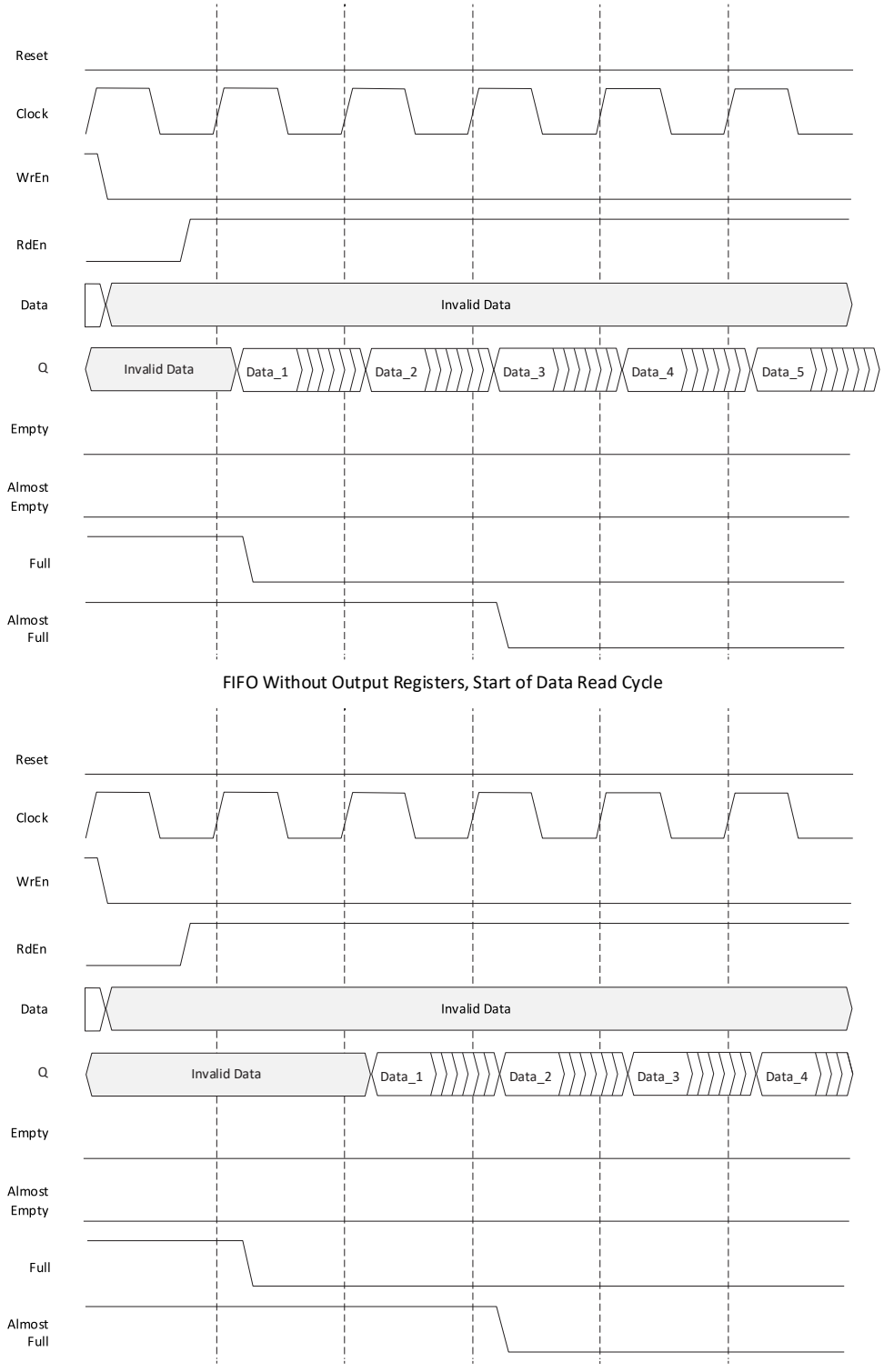


Figure 2.9. Wave Forms of FIFO Dual Clock Module With and Without Registers

2.7. DSP Blocks

2.7.1. DSP Architectures Comparison

Both Lattice and AMD devices have hardware DSP units as part of their FPGA architectures. The different modes of operations available are as follows:

- MULT (Multiply)
- MAC (Multiply, Accumulate)
- MULTADDSUB (Multiply, Addition/Subtraction)
- MULTADDSUBSUM (Multiply, Addition/Subtraction, Summation)

DSP primitives can be directly instantiated in the design HDL source file. Each of the primitives has a fixed set of attributes that can be customized to meet the design requirements.

Lattice device DSP blocks are configurable to support multiple bus widths. Each block could be configured to support one 36 × 36, two 18 × 36, four 18 × 18, or eight 9 × 9 bus width.

For more information on the Lattice device DSP, refer to the following documents:

- sysDSP User Guide for Nexus Platform (FPGA-TN-02096), other Lattice device families have equivalent document.
- DSP Arithmetic Modules User Guide (FPGA-IPUG-02050)
- Arithmetic Modules User Guide (FPGA-IPUG-02032)

Primitives for both AMD and Lattice devices are listed in Table 2.16.

Table 2.16. DSP Primitive Comparison Between Lattice and AMD Devices

AMD Primitive	Lattice Primitive (Nexus Device Family)	Descriptions
DSP48E2 (UltraScale device) DSP48E1 (7-Series and Virtex 6 device) DSP48E (Virtex-5 device) DSP48E/1 (25 × 18) DSP48E2 (27 × 18) Larger multipliers can be built by cascading	MULT18X18	18 × 18 Multiplier with Optional Input/Output Registers
	MULT18X36	18 × 36 Multiplier with Optional Input/Output Register
	MULT36X36	36 × 36 Multiplier with Optional Input/Output Registers
	MULTADDSUB18X18	18 × 18 Multiplier and Accumulator
	MULTADDSUB18X36	18 × 36 Multiplier and Adder/Subtractor
	MULTPREADD18X18	18 × 18 Multiplier with Pre-Adder
	MULTPREADD9X9	9 × 9 Multiplier with Pre-adder
	There are more options available, refer to the sysDSP User Guide for Nexus Platform (FPGA-TN-02096) for more details.	

2.7.2. DSP Features Comparison

Table 2.17 compares the AMD DSP48E1 multiplier core and the Lattice DSP core features.

Table 2.17. DSP Features Comparison Between Lattice and AMD Devices

DSP Features	AMD Device	Lattice Device
Pre-adder	Yes	Yes
Multiplier	Only 25 × 18 (7-Series Device) And 27 × 18 (UltraScale Device)	36 × 36, 18 × 18, 9 × 9
Add/Subtract/accumulate	Yes	Yes
Pattern detector	Yes	Need to be implemented in the logic
Basic logic function (XOR,NOR…)	Yes	Need to be implemented in the logic
Saturation and rounding options	Yes	Yes
Cascading DSP blocks	Yes	Yes
Configurable Pipeline Register	Yes	Yes

2.7.3. DSP Port Mapping Comparison

Table 2.18 shows the port mapping between the AMD Multiplier Core and the Lattice IP core.

Table 2.18. DSP Port Comparison Between Lattice and AMD Devices

AMD Device	Lattice Device	Descriptions
A [ ]	Data_a_i [ ]	Data Input Port A
B [ ]	Data_b_i [ ]	Data Input Port B
CLK	clk_i	Clock Port
CE	clk_en_i	Clock Enable Port
SCLR	rst_i (Sync only)	Active HIGH reset for both (sync/Async)
P [ ]	Result_o[ ]	Multiplication result

It is highly recommended to use the IP Catalog to convert the Multiplier Core that targets an AMD device into multipliers for Lattice FPGA devices. Figure 2.10 shows the difference between AMD and Lattice IP Catalog Interfaces.

Note: The AMD IP Catalog allows you to configure the DSP block from one main interface to implement multiplication, accumulation, adder, and so on. With Lattice IP Catalog, you have multiple IP Catalog units that you need to select from depending on the operation mode that you have to implement.

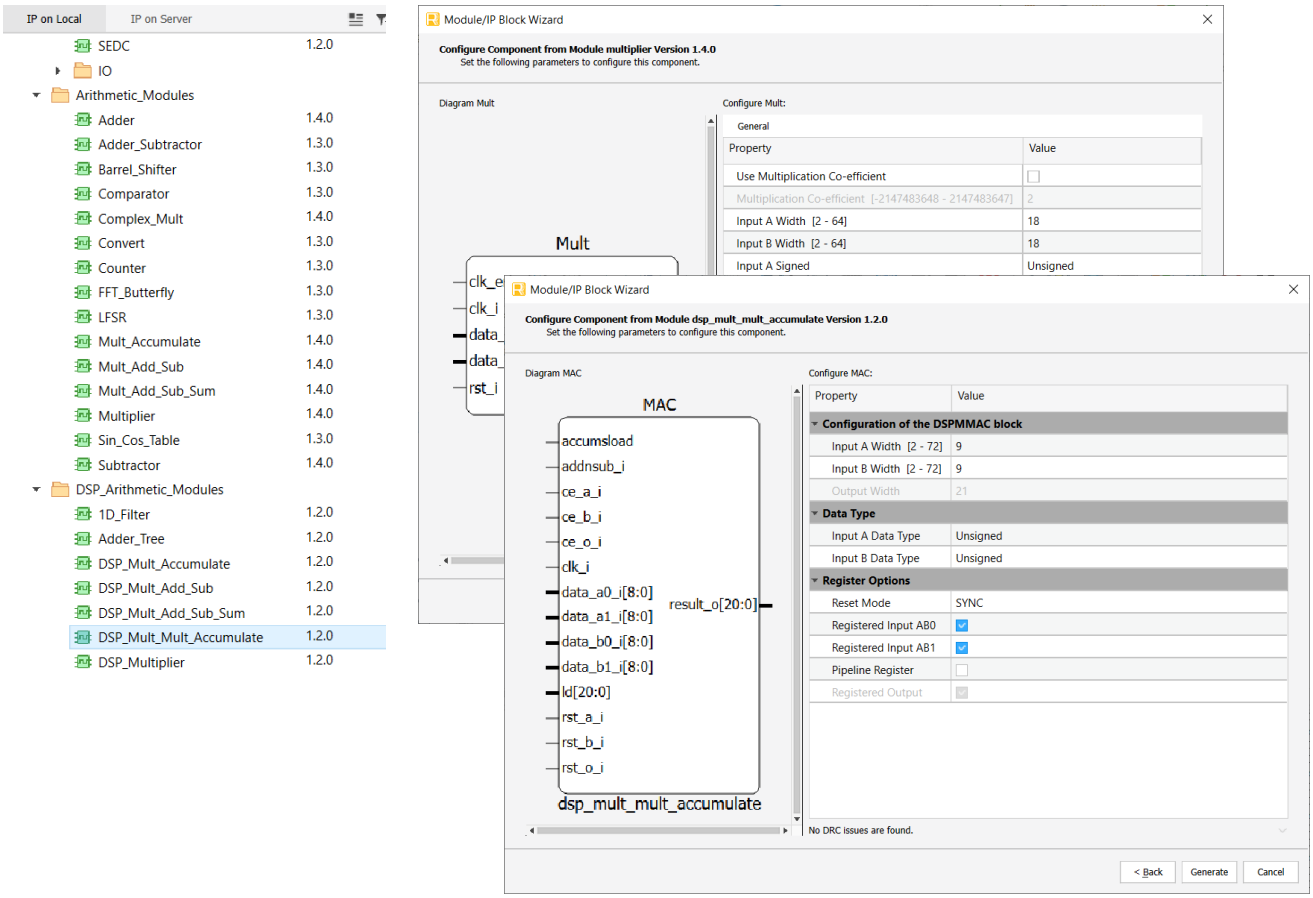


Figure 2.10. Comparison Between AMD and Lattice IP Catalog Interfaces

2.7.4. Design Conversion Recommendations

You can refer to the following recommendations when performing the conversion:

- In the original code, identify if the DSP or ALU function is inferred or hard coded using the primitives.
  - If the function is inferred, Lattice synthesis tool should be able to convert the function using the proper configuration. Verify implementation and use HDL attribute to force a desired implementation.
  - If the source code refers to the primitives, you will need to generate equivalent functional block and instantiate it in the HDL code. You can use the primitive listed in this section or use the IP Catalog which is recommended.
- Verify the bus size and data sign (signed versus unsigned). Lattice device support signed and unsigned on both ports A and B.
- Verify the number of pipeline registers. Lattice device DSP blocks offer pipeline stage on top of the input and output registers. Using the IP Catalog, you can generate a DSP-based or a LUT-based implementation. DSP implementation is recommended as it offer better performance and does not use fabric logic.
- Replace the AMD component in the source code by the one generated for Lattice device architecture.
- In some designs you may have attributes that guide the implementation to use the DSP blocks of the architecture. Refer to the followings for more details:
  - For these AMD device attributes:

```
attribute use_dsp48 : string;
attribute use_dsp48 of coreTransform : entity is “yes”;
```

  You can use these equivalent Lattice device attributes:

```
attribute syn_multstyle : string ;
attribute syn_multstyle of coreTransform : entity is " block_mult " ;
```

Note: The input, output and pipeline registers will enhance the design performance but may have an impact on the design functional integrity. Make sure you keep the number of pipeline registers consistent between the original design and the converted one.

2.7.5. DSP Inferring Design Example

The following VHDL code is not technology dependent, you could target this to either AMD or Lattice device. The synthesis tool will automatically infer a DSP block to implement the required function unless you have an attribute to force the implementation in the logic gates.

The function described in the code is a multiplier with pre-adder and uses the input and output registers. The implementation in either AMD or Lattice should take only one DSP block.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity presubmult is
generic (     AWIDTH : natural := 12; -- Width of A input
              BWIDTH : natural := 16; -- Width of B input
              CWIDTH : natural := 16 -- Width of C input);
port (
clk : in std_logic; -- Clock
ain : in std_logic_vector(AWIDTH-1 downto 0); -- A input
bin : in std_logic_vector(BWIDTH-1 downto 0); -- B input
cin : in std_logic_vector(CWIDTH-1 downto 0); -- C input
pout : out std_logic_vector(BWIDTH+CWIDTH downto 0) – Output
      );
end presubmult;

architecture rtl of presubmult is

signal a: signed(AWIDTH-1 downto 0);
```

```
signal b : signed(BWIDTH-1 downto 0);
signal c : signed(CWIDTH-1 downto 0);
signal add : signed(BWIDTH downto 0);
signal mult, p, : signed(BWIDTH+CWIDTH downto 0);

begin


process(clk)
begin
if rising_edge(clk) then
          a <= signed(ain);
          b <= signed(bin);
          c <= signed(cin);
          mult <= (resize(a, BWIDTH+1) + resize(b, BWIDTH+1))* c;
end if;
end process;
pout <= std_logic_vector(mult);
end rtl;
```

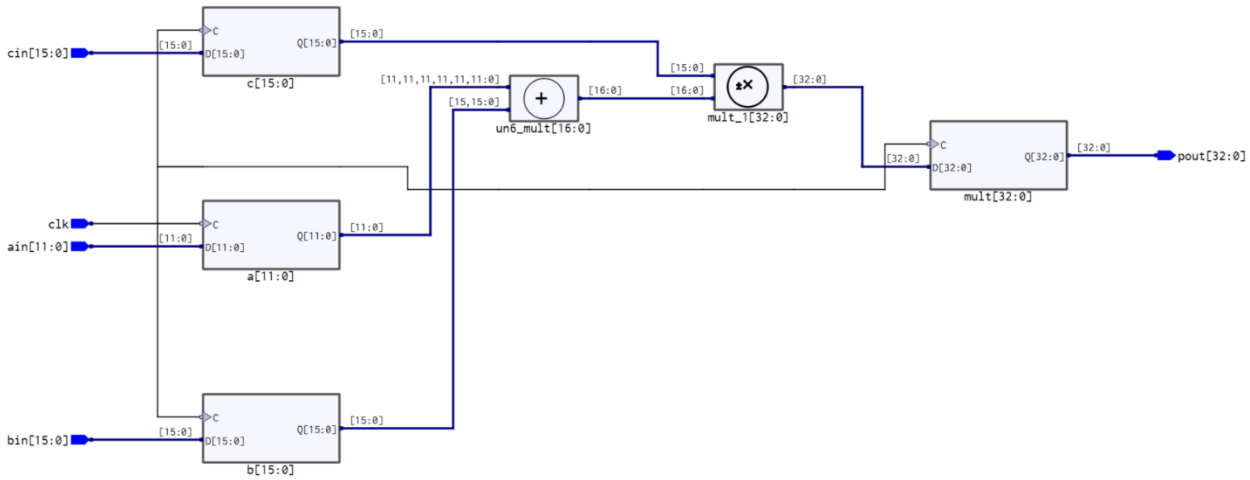


Figure 2.11. Synplify Pro RTL View

Going through the Map process, the pipeline registers are no longer visible as they are integrated in the DSP block. The AMD Vivado™ tool infers a DSP48E1. Based on the block configuration, there is a similar information to the one listed for Lattice MULTPREADD18X18 that is inferred by the Lattice Radiant software. Input A, B, and C are all registered and output registers are also enabled. Refer to Figure 2.12 for more details.

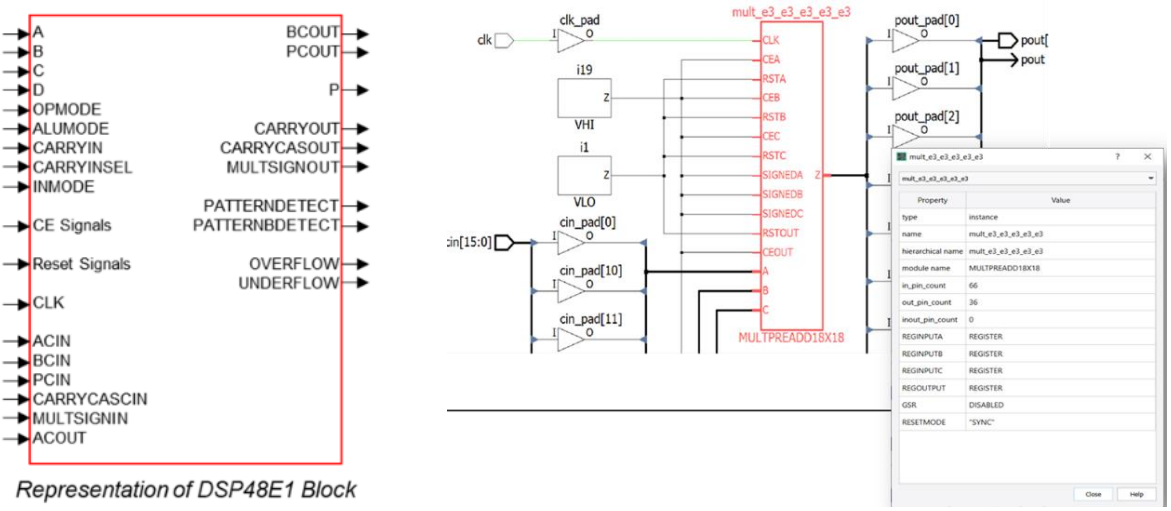


Figure 2.12. Design Implementation Comparison Between AMD DSP48E1 and Lattice MULTPREADD18X18

2.8. SerDes/Transceivers

2.8.1. SerDes/Transceivers Comparison

The Lattice device SerDes or transceivers are comparable to those in the AMD devices. The speed and protocols that are supported may vary between families. Refer to the Lattice specific device summary table or Table 3.2 for details about the number of SerDes and max speed supported by each family.

For a device to support a certain standard, a compliance to the specification is required. In addition to the SerDes speed, the requirement is applicable to other aspects such as jitter, eye diagram Rx and Tx, and serial data stream characteristics support.

Lattice devices are tested for compliance with the standard and for guarantee with the conformity of the signal to the specification parameters. Refer to the specific device family datasheet for supported protocols by the device. Table 2.19 shows the comparison of the number and speed of SerDes between AMD and Lattice devices.

Table 2.19. SerDes Specification Comparison Between Lattice and AMD Devices

Device Family/SerDes Specification	AMD Artix-7	AMD Kintex-7	AMD Virtex-7	Lattice ECP5/ECP5-5G	Lattice CertusPro-NX	Lattice Avant
# High speed Transceivers	16	32	96	8	8	28
Max Speed per Transceiver	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s	5 Gb/s	10.315 Gb/s	25 Gb/s
PCIe Hard Core	Gen 2	Gen 2	Gen 2/3	Soft	Gen 1/2/3	Gen 1/2/3/4

Note: For Lattice devices, the transceivers are referred to as SerDes in the literature with different speed supported. For AMD devices, the transceivers are named differently based on their maximum speed supported. This means that GTP = 6.6 Gb/s, GTX = 12.5 Gb/s, GTH = 13.1 Gb/s, and GTZ = 28.05 Gb/s.

Each protocol implemented using these SerDes/transceivers is compliant to a standard. A physical coding sublayer (PCS) works behind the SerDes to implement the physical layer and interface to the fabric. This is applicable to most FPGA devices which includes encoding, decoding, scrambler, and elastic buffer among many others. Figure 2.13 shows a simplified example of the CertusPro-NX device PCS block.

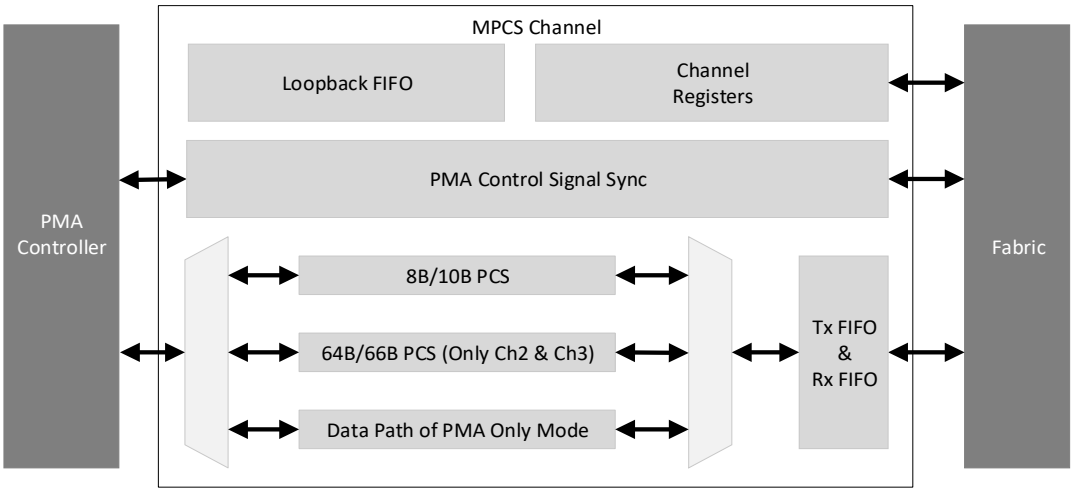


Figure 2.13. Lattice CertusPro-NX Device PCS Block

2.8.2. Design Conversion Recommendations

At a high level, the implemented protocol using SerDes needs to have the same functionalities and to comply to the standard such as the PCIexpress and Gigabit Ethernet. You need to provide special attention to optional features and IP core register interface. Multiple options are usually offered. Depending on the IP, one or more of the following options are available:

- AHB: Advanced High-performance Bus (AHB) is a bus protocol introduced in Advanced Microcontroller Bus Architecture published by the Arm company. AHB is used for a high-frequency design.
- APB: Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) family protocols. APB is used to interface to peripherals that are low bandwidth and takes low power.
- AXI4: The Advanced eXtensible Interface (AXI), is an on-chip communication bus protocol developed by the Arm company.
- LMMI: Lattice Memory Mapped Interface which is a parallel bus interface.

The choice of the bus interface protocol will have an impact on the rest of the design, and the way to interface and control the registers of the IP Module. Some work may be needed to adapt the design to the Lattice IP. For more information, refer to the Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide (FPGA-UG-02039).

2.8.3. Lattice Device Supported SerDes Based Standards

The number of protocols supported per device will vary. Using the CertusPro-NX device family as an example, the protocols shown in Table 2.20 can be implemented using the SerDes and associated MPCS.

Table 2.20. Standards Supported by the CertusPro-NX Device Family SerDes/PCS

Standard	Data Rate (Mb/s)	System Reference Clock (MHz)	FPGA Clock (MHz)	Number of Link Width	Encoding Style
PCI Express Gen1[1]	2500	100, 125	125	×1, ×2, ×4	8b10b
PCI Express Gen2[1]	5000	100, 125	125	×1, ×2, ×4	8b10b
PCI Express Gen3[1]	8000	100, 125	250	×1, ×2, ×4	128b130b
Ethernet 1000BASE-X	1250	125	125	x1	8b10b
Ethernet SGMII	1250	125	125	×1	8b10b
Ethernet XAUI	3125	156.25	156.25	×4	8b10b
Ethernet QSGMII	5000	125	125	×1	8b10b
Ethernet 10GBASE-R[2]	10312.5	161.1328125	161.1328125	×1	64b66b
SLVS-EC Grade1	~1250	—	~125	×1, ×2, ×4, ×6, ×8	8b10b
SLVS-EC Grade2	~2500	—	~125	×1, ×2, ×4, ×6, ×8	8b10b
SLVS-EC Grade3	~5000	—	~125	×1, ×2, ×4, ×6, ×8	8b10b

Standard	Data Rate (Mb/s)	System Reference Clock (MHz)	FPGA Clock (MHz)	Number of Link Width	Encoding Style
CoaXPress CXP-1	1250	125	125	×1~×8	8b10b
CoaXPress CXP-2	2500	125	125	×1~×8	8b10b
CoaXPress CXP-3	3125	156.25	156.25	×1~×8	8b10b
CoaXPress CXP-5	5000	125	125	×1~×8	8b10b
CoaXPress CXP-6	6250	156.25	156.25	×1~×8	8b10b
DP/eDP RBR	1620	108	162	×1, ×2, ×4	8b10b
DP/eDP HBR	2700	135	135	×1, ×2, ×4	8b10b
DP/eDP HBR2	5400	135	135	×1, ×2, ×4	8b10b
DP/eDP HBR3	8100	135	202.5	×1, ×2, ×4	8b10b
10-bit/20-bit/40-bit SerDes	625 – 8100	—	—	×1~×8	N/A
8-bit/16-bit/32-bit SerDes	625 – 8100	—	—	×1~×8	N/A
Generic 8b10b	625 – 8100	—	—	×1~×8	8b10b

Notes:

1. CertusPro-NX supports a maximum of four lanes PCIe with hard IP.
2. CertusPro-NX SerDes does not support Ethernet 10GBASE-KR. Moreover, not all channels can support 10GBASE-R.

For more information on the CertusPro-NX device family (other Lattice device families have equivalent documents), refer to the following documents:

- CertusPro-NX Family Data Sheet (FPGA-DS-02086)
- CertusPro-NX SerDes/PCS User Guide (FPGA-TN-02245)
- CertusPro-NX Hardware Checklist (FPGA-TN-02255)

All these IPs could be parametrized using the IP Catalog or IP Express tools integrated with the Lattice Radiant and Diamond software platforms. A direct connection to the IP server is available for you to download the latest version of any of these IPs. Figure 2.14 shows an example interface that is displayed for the connectivity IPs that could be installed on a local machine.

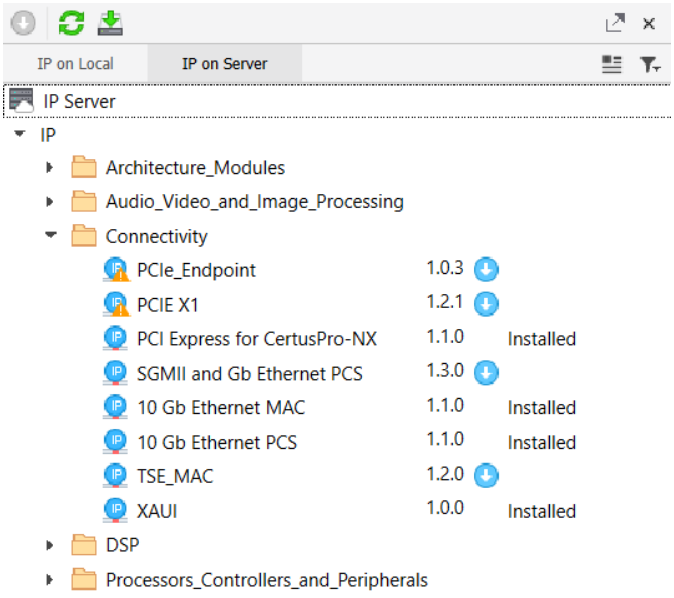


Figure 2.14. Lattice Radiant Software IP Catalog GUI

Note: Not all the IPs are available for all devices. You may see an exclamation mark if the IP does not apply to the selected device when opening the IP Catalog.

2.9. External Memory Interface

Both AMD and Lattice devices support multiple external memory interface protocols such as DDR2, LPDDR2, DDR3, LPDDR3, and LPDDR4. DDR memory interfaces use a non-continuous bidirectional strobe signal called DQS which is edge aligned with the data bus (DQ).

The most challenging part of the DDR Memory interface is the clock domain crossing from the DQS strobe signal to the system fabric clock domain. A 90-degree phase shift needs to be performed on the DQS to clock the data that is received by the FPGA fabric. Most of Lattice FPGA architectures include hardened PHY layer on specific PIC (Programmable I/O Cell) to facilitate this transfer.

The CertusPro-NX devices and other device families integrate the following hardened function on the I/O cell:

- DQS Clock Tree spanning the DQS group
- DDRDLL used to generate the 90-degree delay codes
- DLL-compensated DQS delay elements
- Input FIFO for read data clock domain transfer
- Dedicated DDR memory input and output registers
- Dynamic Margin Control Circuit to adjust Read and Write delays
- Input/Output Data Delay used to compensate for DQS clock tree delay
- ×4 or ×8 gearing box to demux the data bus

When migrating from AMD device to Lattice device, pay extra attention to the way the memory controller has been implemented. For example, AMD Spartan-6 FPGA devices incorporate full hardened memory-controller blocks. Whereas the AMD 7-Series FPGA devices implement the memory controller with a soft IP core and a hardened PHY layer that integrate similar functionalities like described above, which is similar to the Lattice design implementation. This method provides more design flexibility for you to customize your memory controller to meet your requirements.

Older generation AMD devices may have implementations that use the fabric logic for DQ/DQS to FPGA clock domain transfer. It is recommended to generate the required memory interface with the IPexpress (Diamond) or IP Catalog (Radiant) to have an optimized implementation and to avoid errors.

Figure 2.15 shows an example of an IP Catalog GUI for memory interface generation. Note that there are multiple options that can be set which includes the gearing ratio that allows you to have a wider bus with lower speed on the FPGA fabric.

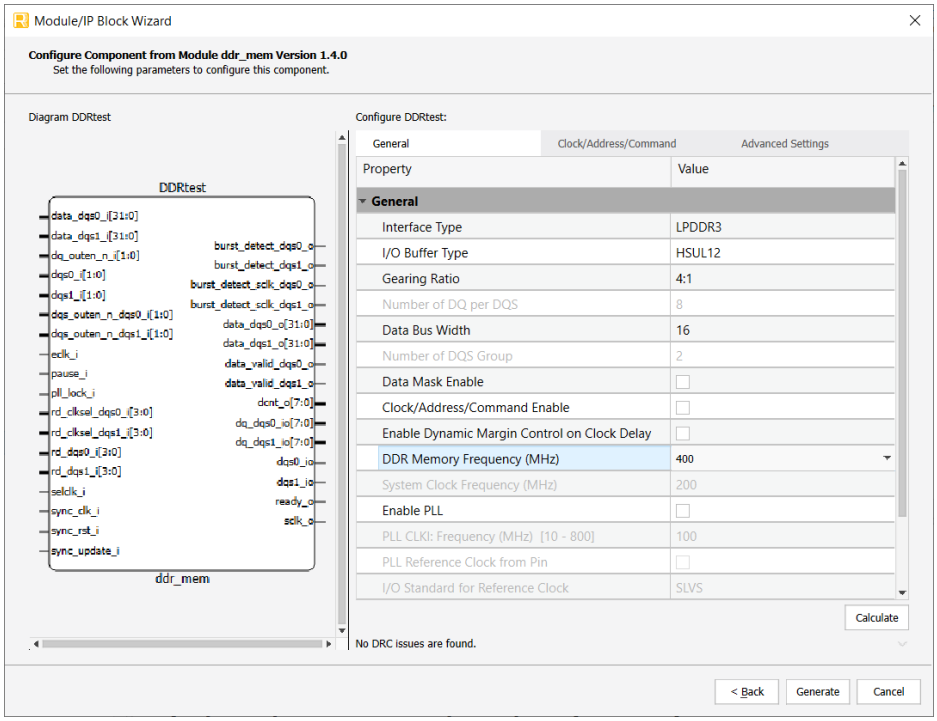


Figure 2.15. Lattice Radiant Software IP Catalog GUI for Memory Interface Generation

2.9.1. Lattice Device Supported Standards

The external memory support standard will depend on the device family. For example, the CertusPro-NX device can be used to support the DDR3, DDR3L, LPDDR2, LPDDR3 and LPDDR4 memory interfaces. Refer to Table 2.21 for more details.

For more information on the CertusPro-NX device family (other Lattice device families have equivalent documents), refer to the following documents:

- CertusPro-NX Family Data Sheet (FPGA-DS-02086)
- sysI/O User Guide for Nexus Platform (FPGA-TN-02067)
- CertusPro-NX High-Speed I/O Interface (FPGA-TN-02244)
- CertusPro-NX Hardware Checklist (FPGA-TN-02255)

Table 2.21. DDR Memory Configurations Support

DDR Memory	Data Width	VCCIO	DQ	DQS	Module Types	Rank	Chip Selects	Write Leveling	CMD/ADDR Timing	Fmax
DDR3	8, 16, 24, 32 bits	1.5 V	SSTL15_I	SSTL15D_I	UDIMM, SODIMM, RDIMM	Single, Dual	1, 2,4	Yes	2T[3]	533 MHz
	8, 16, 24, 32 bits	1.5 V	SSTL15_I	SSTL15D_I	Embedded	Single, Dual	1, 2,4	Yes[1]	2T[3]	533 MHz
DDR3L	8, 16, 24, 32 bits	1.35 V	SSTL135_II	SSTL135D_II	UDIMM, SODIMM, RDIMM	Single, Dual	1, 2,4	Yes	2T[3]	533 MHz
	8, 16, 24, 32 bits	1.35 V	SSTL135_II	SSTL135D_II	Embedded	Single, Dual	1, 2,4	Yes[1]	2T[3]	533 MHz
LPDDR2	16 and 32 bits	1.2 V	HSUL12	HSUL12D	Embedded (Single Channel)	Single	1	No	ODDRX2	533 MHz
LPDDR4	16, 32, 64 bits	1.1 V	LVSTL11	LVSTL11D	Embedded (Single Channel)	Single	1, 2,4	Yes[1]	2T[3]	533MHz

Notes:

1. If fly-by wiring is implemented.
2. Fly-by wiring is emulated using board traces.
3. CSN uses 1T timing.

All the memory parameters can be set when configuring the memory controller IP with the IP Catalog or IP Express.

Even though Lattice device support a DDR memory controller IP, you can implement your own custom design and take advantage of the hardened PHY Layer. For this type of implementation, you can refer to the IP Catalog Architecture module to configure the PHY for a specific memory interface as shown in Figure 2.16.

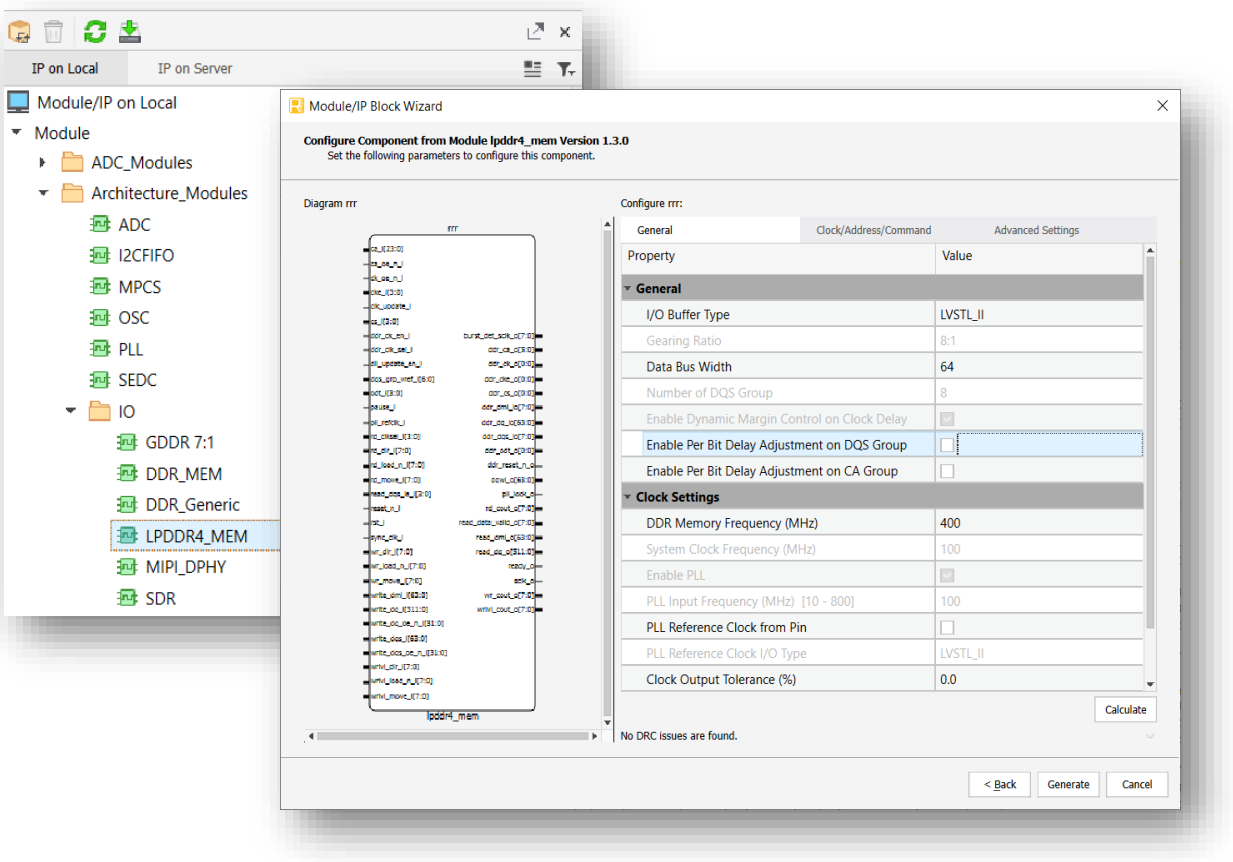


Figure 2.16. IP Catalog LPDDR4 Memory Interface Configuration Window

2.10. Other FPGA Device Hardened Functions

When converting the AMD designs, you may encounter some other hardened functions like the embedded microprocessor IP (microblaze), ADC, and cryptographic engine. Table 2.22 shows the comparison of these features between the AMD architecture and their equivalent for Lattice architecture.

Table 2.22. Hardened Functions Comparison Between Lattice and AMD Devices

AMD Device	Lattice Device	Descriptions
Microblaze: A family of drop-in, modifiable preset with 32-bit/ 64-bit RISC microprocessor configurations.	RISC-V: A soft IP which contains a 32-bit RISC-V processor core and optional submodules.	Lattice Propel™ software is the Embedded Design Environment to implement the RISC-V soft processor systems in the Lattice FPGA devices.
XADC: Dual 12-bit 1 MSPS analog-to-digital converters (ADCs).*	ADC: Dual ADC – 1 MSPS, 12-bit, with Simultaneous Sampling.	Use the IP Catalog to configure the ADC unit.
Encryption: 256-bit AES encryption with the HMAC/ SHA-256 authentication.*	• Cryptographic engine (CRE) • Bitstream encryption – using the AES-256. • Bitstream authentication – using the ECDSA. • Hashing algorithms – SHA, HMAC • True Random Number Generator • AES 128/256 Encryption	Crypto Engine are used during the FPGA configuration. These functions are available after the configuration for you to implement various cryptographic functions into your FPGA design. CRE bus interface is the LMMI.
Built-in SEU detection and correction	• Single Event Upset (SEU) Mitigation Support • Soft Error Detect (SED) – Embedded hard macro. • Soft Error Correction (SEC) – Transparent to user design operation. • Soft Error Injection – Emulate SEU event to debug system error handling.	Lattice CertusPro-NX devices offer extremely low Soft Error Rate (SER) due to the FD-SOI technology used. SEU features are also available.

***Note**: Applicable to all AMD 7-Series FPGA devices except for the XC7S6 and XC7S15 devices.

For more information on the Lattice Nexus device family (other Lattice device families have equivalent documents), refer to the following documents:

- Soft Error Detection (SED)/Correction (SEC) User Guide for Nexus Platform (FPGA-TN-02076)
- ADC User Guide for Nexus Platform (FPGA-TN-02129)
- Single Event Upset (SEU) Report for Nexus Platform (FPGA-TN-02174)
- Advanced Configuration Security User Guide for Nexus Platform (FPGA-TN-02176)
- Using TraceID Technical Note (FPGA-TN-02084)

3. Selecting the Right Equivalent Target Device

3.1. Step 1: Collect Information from the AMD Report File

AMD tools provide you with a report file that can be found in the project directory or through the Vivado tool synth_design – *Utilization*. In the report file, you can find the information shown in Figure 3.1.

```
Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Slice Logic Distribution
3. Memory
4. DSP
5. IO and GT Specific
6. Clocking
7. Specific Feature
8. Primitives
9. Black Boxes
10. Instantiated Netlists
```

Figure 3.1. Utilization Design Information from AMD Report File

Alternatively, you can find the file with the .rpt extension and collect the information shown in Table 3.1. This information will help you to identify the best fit for your design in terms of used resources.

Table 3.1. List of Information to Collect from the AMD Report File

AMD Design Information	Lattice Design Information	Notes
Logic Cells	Logic Cells	For older devices, 1 Equivalent Logic Cell = LC * 1.125 1 logic Cell = 1.6 LUT4 equivalent 1 System Logic Cell = 2.1875 LUT4 equivalent
Block RAM	EBR	Block RAM can have different sizes between the 2 architectures. Keep in mind that you can use the distributed memory if available.
Distributed RAM	Distributed Memory	The total amount of distributed memory, not the number of LUTs used as distributed memory. Keep in mind that you can use the EBRs if available.
UltraRAM	LRAM	AMD design offers 288 kb versus 512 kb per block.
DSP	DSP	Take into consideration the configuration mode of each DSP block (MULT,ADD/SUB, and MAC) and the size (18×18, and so on).
DCM/MMCM	sysPLL	The number of PLLs (DCM or MMCM). Refer to the PLL/MMCM/DCM section for differences and flexibilities of Lattice device PLL structure.
Transceivers	SerDes	Collect the information on the number of transceivers and list of implemented protocols. Refer to the SerDes/Transceivers section to confirm the protocols supported by Lattice devices.
External Memory interface	External Memory interface	The external memory interface speed and size. Refer to the External Memory Interface section to confirm the interfaces supported by Lattice devices.
N/A	User Flash Memory	AMD architecture does not include user flash memory. Look on the board level if there is such device. Device with user flash memory could be integrated.
PCIe Interface	PCIe Interface	The number of interfaces and PCIe Gen required.
ADC	ADC	—
I/O Pins (Max)	I/O Pins (Max)	Max number of I/Os used.
I/O Voltage	I/O Voltage	The different interface voltages required.

AMD Design Information	Lattice Design Information	Notes
Cryptographic engine	Cryptographic engine	Lattice Cryptographic Engine supports the following user-mode features: • True Random Number Generation (TRNG) • Secure Hashing Algorithm (SHA) – 256 bits • Message Authentication Codes (MACs) – HMAC
Internal Oscillator	Internal Oscillator	—
MIPI (D-PHY)	MIPI (D-PHY)	The speed required for the MIPI interface.
Non-Volatile[1]	Non-Volatile[1]	Some of the Lattice devices are non-volatile which eliminate the need to have an external boot flash and have the advantage of fast device boot with controlled I/Os.
Temperature Grade[2]	Temperature Grade[2]	All devices are available in the commercial and industrial grades but not all of them are available in the automotive grade. It is important to note if you need an automotive temperature grade device. Refer to the Device Temperature Grades section for more information.

Notes:
1. Device integrates a non-volatile memory for single or multiple boots.
2. Junction temperature, Commercial operation is from 0 °C to +85 °C, Industrial operation is from –40 °C to +100 °C, and Automotive operation is from –40 °C to +125 °C.

3.2. Step 2: Reconsider Your Device Size

Keep in mind that you may be able to fit your design in a smaller size Lattice device. The design resource bottleneck may have driven the selection of a larger AMD device.

For example, you need 300 I/Os for your design. In AMD Spartan-7 Series, XC7S75 is the smallest device option that fits your requirement. In the AMD Artix-7 Series, XC7A75T is the smallest device option that fits your requirement. Both options include over 70k logic cells.

If the logic density is not required, you could potentially select the Lattice MachXO2 device family that offers over 300 I/Os with only 10k logic cells. This selection will give an advantage on the price and power consumption of the device.

3.3. Step 3: Select the Equivalent Device

Select the equivalent device using the information from the previous steps and options provided in Table 3.2.

Table 3.2. Summary of Lattice Device Specifications Based on Different Device Family

Device Family/ Specification	iCE40	MachXO2/ MachXO3/ MachXO5-NX	ECP5	CrossLink/ CrossLink-NX	Certus-NX/ CertusPro-NX	Avant
Logic Cells	5k	25k	150k	39k	100k	637k
Block RAM	0.12 Mb	1.4 Mb	3.7 Mb	1.5 Mb	3.7Mb	36 Mb
Large RAM Blocks	1 Mb	0.5 Mb	—	1 Mb	3.5 Mb	—
DSP	8	20	156	64	156	1800
PLL/DLL	1	2/2	4/2	3/2	4/2	11
SerDes	—	—	4	—	8	28
SerDes Max Speed	—	—	5 Gb/s	—	10.3 Gb/s	25 Gb/s
Memory interface	—	1066 Mb/s	800 Mb/s	—	1066 Mb/s	2400 Mb/s
User Flash Memory	—	15.36 Mb	—	—	—	—
PCIe Interface	—	—	Gen1/2	Gen1/2	Gen1/2/3	Gen1/2/3/4
ADC	—	2	—	2	2	—

Device Family/ Specification	iCE40	MachXO2/ MachXO3/ MachXO5-NX	ECP5	CrossLink/ CrossLink-NX	Certus-NX/ CertusPro-NX	Avant
I/O Pins (Max)	39	306	365	192	305	500
I/O Voltage	1.2 V – 3.3 V	1.0 V – 3.3 V	1.2 V – 3.3 V	1.2 V – 3.3 V	1.0 V – 3.3 V	1.0 V – 3.3 V
Cryptographic engine	—	AES128/256, ECDSA, SHA, HMAC	AES128	AES128/256, ECDSA, HMAC	AES256, ECDSA	AES256-GCM, ECC521, RSA4096, PUF
Internal Oscillator	2	2	2	2	2	1
MIPI (D-PHY)	—	1.25G/Lane	—	2.5G/lane	—	1.8 Gbps/Lane
Non-Volatile[1]	Yes	Yes	No	No	No	No
Temperature Grade[2]	Commercial and Industrial	Commercial, Industrial and Automotive	Commercial, Industrial and Automotive	Commercial, Industrial and Automotive	Commercial and Industrial	Commercial and Industrial

Notes:

1. Device integrates a non-volatile memory for single or multiple boots.
2. Junction temperature, Commercial operation is from 0 °C to +85 °C, Industrial operation is from –40 °C to +100 °C, and Automotive operation is from –40 °C to +125 °C.

4. HDL Code Compatibility

4.1. Introduction

The first step in a conversion is to identify and import the HDL code to the Lattice Radiant or Diamond software. These files are either VHDL or Verilog. The synthesis output netlist needs to be recompiled to target Lattice technology and cannot be imported directly.

When importing source files to the Lattice platform, you may get warning and error messages like shown below during the parse or synthesis phase. All the items listed represent compatibility issues that need to be resolved. There can be multiple reasons causing the issues and some of the common reasons are as follows:

- Library declaration that needs to be set within the Lattice tool. The library can define the HDL design parameters, and functions.
- Verilog Include file path needs to be properly set in the software.
- Using unrecognized AMD primitives such as IBUF, OBUF, and BUFHCE.
- Unrecognized IP module that was generated by the AMD tool such as the PLL and FIFO.

The followings are examples of the warning and error messages:

```
WARNING - <File path and name>/ (185,3-185,57) (VERI-1063) instantiating unknown module 'IBUF'
WARNING - <File path and name>/ (189,3-189,58) (VERI-1063) instantiating unknown module 'OBUF'
WARNING - <File path and name>/ (81,3-88,5) (VERI-1063) instantiating unknown module 'clk_core'
WARNING - <File path and name>/ (91,3-99,5) (VERI-1063) instantiating unknown module 'BUFHCE'
WARNING - <File path and name>/ (255,3-296,5) (VERI-1063) instantiating unknown module
'cmd_parse'
WARNING - <File path and name>/ (314,3-334,5) (VERI-1063) instantiating unknown module
'resp_gen'
WARNING - <File path and name>/ (339,3-349,5) (VERI-1063) instantiating unknown module
'char_fifo'
WARNING - <File path and name>/ (79,3-86,5) (VERI-1063) instantiating unknown module
'debouncer'
WARNING - <File path and name>/ (382,3-394,5) (VERI-1063) instantiating unknown module
'clkx_bus'
WARNING - <File path and name>/ (50,4-62,6) (VERI-1063) instantiating unknown module 'ODDR'
ERROR - <File path and name> (69): cannot open include file design_par.vh. VERI-1245
ERROR - Stopping Synthesis Tool flow due to error.
ERROR - <File path and name> (160): module ignored due to previous errors. VERI-1072
ERROR - Stopping Synthesis Tool flow due to error.
WARNING - <File path and name> (87): parameter declaration becomes local in BUS_SIZE_MIN with
formal parameter declaration list. VERI-1199
WARNING - <File path and name> (88): parameter declaration becomes local in BUS_SIZE_MAX with
formal parameter declaration list. VERI-1199
WARNING - IP Module ODDR not found in top. Skipping Constraint Propagation...
WARNING - IP Module char_fifo not found in top. Skipping Constraint Propagation...
WARNING - IP Module clk_core not found in top. Skipping Constraint Propagation...
ERROR - Can't open file design_par.vh
```

The warning and error messages provide a good starting point to the conversion process. The next sections provide solutions to overcome these conversion problems.

In general, you have to perform the following tasks for the conversion process:

- Comment out any AMD-specific library and add the Lattice library, if required.
- Replace the AMD-specific primitives, such as I/O buffers and global clock buffers, with the Lattice primitives or behavioral HDL code and preferences.

- Replace the AMD core modules, such as MMCM, PLL, memory, and multipliers with the Lattice modules.
- Replace the AMD timing and device constraints (.ucf or .xdc) file with a Lattice source constraints or preferences file (.sdc or ldc). Refer to the Tools Constraint Compatibility section for more details.
- Optimize the HDL-inferred modules such as the shift registers, counters, and multipliers.

4.2. Library Declaration and Include Files

4.2.1. VHDL

In VHDL, you may have a library declaration in the header of each VHDL code. The first section is a standard IEEE VHDL library declaration whereas the second section is about a user library that is used. These could reference package file, design parameters or other initialization data. An example of a VHDL code is shown below:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_SIGNED.all;

library userlib;
use userlib.userPackage.all;
```

When compiling the design, you need to make sure the compiled code is in the user library so that it can be referenced properly by the source code. Note that the default library in the software tools is *work*. You can change the source code to use *work* library or change the software setting to point to the user library.

To set the library in the Lattice Diamond or Radiant software, you can right click on the source file and select **Properties** from the selection menu. A window where you can set the VHDL library name will open as shown in Figure 4.1. Do this for all the VHDL files that reference the user library.

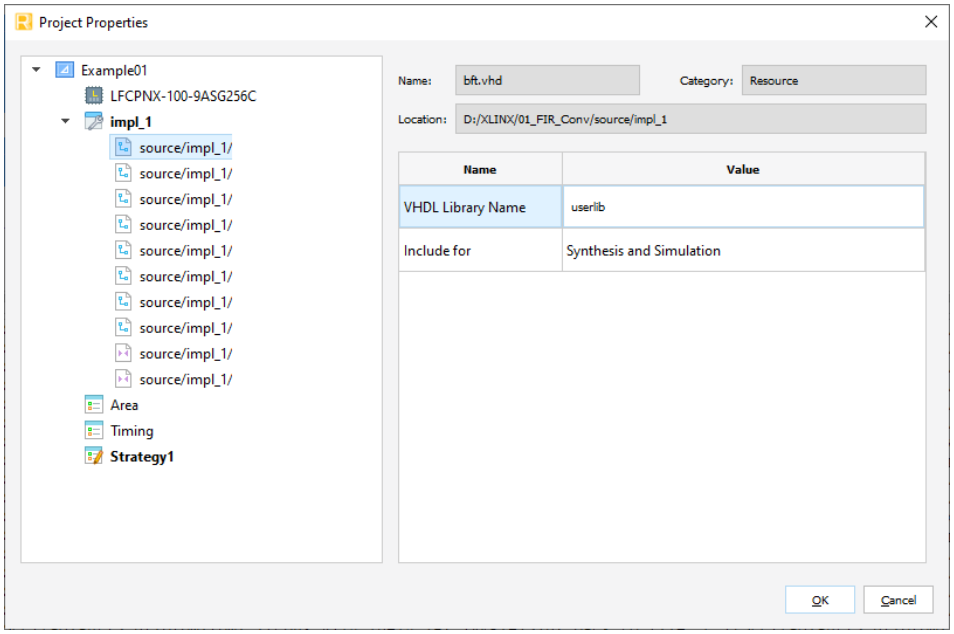


Figure 4.1. Lattice Radiant Software VHDL Library Name

4.2.2. Verilog

For Verilog designs you may have an include file that is called by the design source code. In order for the design to compile properly, you need to set a search path for that include file. The parameter file needs to be tied with the project. See the following code for more details:

```
// user_param file include design parameters and functions
`include "user_param.vh"
```

In the Radiant software platform, you can by right click on the active implementation and select **Properties** from the selection menu to access the include search path. Multiple search paths can be set. See Figure 4.2 for more details.

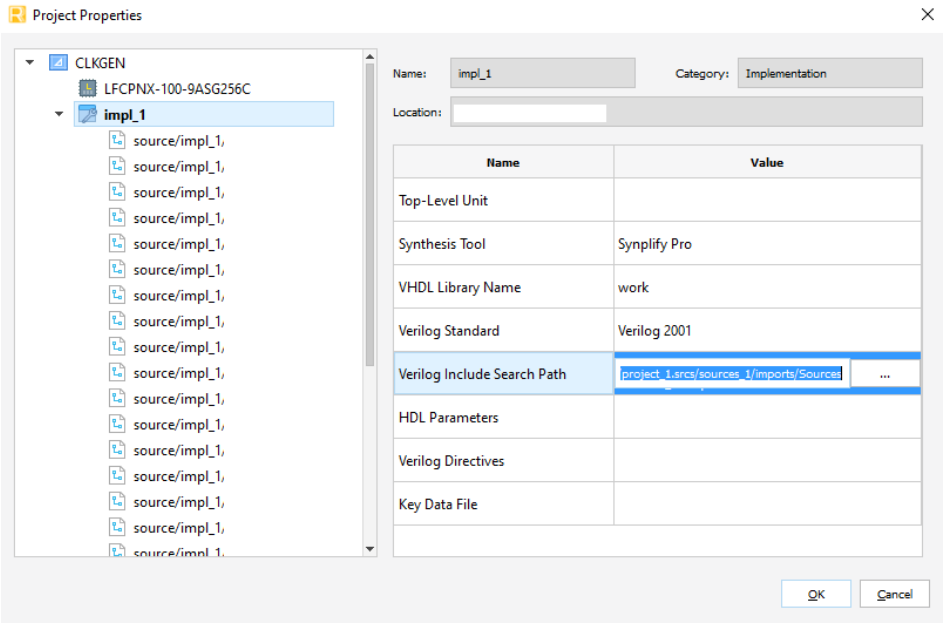


Figure 4.2. Lattice Radiant Software Verilog Include Search Path

4.3. Unrecognized Primitive Modules

During the parse or synthesis phase, you may see a warning message of an unknown module like shown below. This message is usually related to a module that is instantiated in the design but no lower-level HDL source file is found for it.

```
WARNING - <File path and name>/ (185,3-185,57) (VERI-1063) instantiating unknown module 'IBUF'
WARNING - <File path and name>/ (189,3-189,58) (VERI-1063) instantiating unknown module 'OBUF'
WARNING - <File path and name>/ (91,3-99,5) (VERI-1063) instantiating unknown module 'BUFHCE'
```

One of the following options should apply:

- If you missed to import the file to the project, look for the files in the source code that could describe the behavior of the missing module.
- If an AMD architecture primitive is used in the source code, look for an equivalent Lattice architecture module. Some of the replacement code could be as simple as a signal assignment. For example, buffer instantiation is not required for Lattice designs. They are inferred during synthesis and map phase automatically. Refer to the I/O Buffer Primitives section for more information.
- If the module is generated by the AMD IP Catalog tool, an equivalent module needs to be generated in the Lattice environment using the IP Catalog in the Radiant software or the IPexpress in the Diamond software.

4.4. Unrecognized IP Modules

When there are unrecognized IP modules, it means that the module is generated by the AMD IP Catalog tool. An equivalent module needs to be generated in the Lattice software environment.

```
WARNING - <File path and name>/(81,3-88,5) (VERI-1063) instantiating unknown module 'clk_core'
WARNING - <File path and name>/(255,3-296,5) (VERI-1063) instantiating unknown module
'cmd_parse'
```

In the AMD environment, you can look at the details of the module to determine if it is a primitive or a generated core. In the example below, *Clk_core* is an AMD generated IP for a PLL or DCM. It is identified with an orange rectangle in the Vivado tool and will have an .xci file extension. In this case, an equivalent module in the Lattice environment needs to be configured.

Once the module is generated, proceed to the code replacement in the HDL source. It is important to keep the same module name to minimize changes in the HDL code. In this case, see the Verilog source code below as an example:

```
// AMD HDL code
 //clk_core clk_core_i0 (
    //.clk_in1_p          (clk_pin_p),
    //.clk_in1_n          (clk_pin_n),
    //.clk_rx             (clk_rx),
    //.clk_tx             (clk_tx),
    //.reset              (rst_i),
    //.locked             (clock_locked)
  //);
// LSCC conversion
  clk_core clk_core_i0 (
    .clki_i          (clk_pin_p),
    .clkop_o        (clk_rx ),
    .clkos_o        (clk_tx ),
    .rstn_i         (rst_i),
    .lock_o         (clock_locked)
  );
```

Make sure to pay attention to the upper case and lower case in the naming of the module to convert. Especially for mixed language design. VHDL is not case sensitive whereas Verilog is case sensitive. See the example below for a FIFO module generated from the AMD tool and a Lattice equivalent instantiation in Verilog. When configured properly, you will have the same ports, but naming will differ. See Table 4.1 for more details. The same approach is valid for other modules.

```
//AMD HDL Code

char_fifo char_fifo_i0 (
    .din        (char_fifo_din), // Bus [7 : 0]
    .rd_clk     (clk_tx),
    .rd_en      (char_fifo_rd_en),
    .rst        (rst_i),
    .wr_clk     (clk_rx),
    .wr_en      (char_fifo_wr_en),

    .dout       (char_fifo_dout), // Bus [7 : 0]
    .empty      (char_fifo_empty),
    .full       (char_fifo_full)
  );
```

```
//LSCC Conversion

  char_fifo char_fifo_i0 (
    .wr_data_i        (char_fifo_din), // Bus [7 : 0]
    .rd_clk_i     (clk_tx),
    .rd_en_i      (char_fifo_rd_en),
    .rst_i        (rst_i),
   .rp_rst_i        (),
    .wr_clk_i     (clk_rx),
    .wr_en_i      (char_fifo_wr_en),
    .rd_data_o       (char_fifo_dout), // Bus [7 : 0]
    .empty_o      (char_fifo_empty),
    .full_o       (char_fifo_full)
  );
```

Table 4.1. FIFO_DC Port Comparison Between Lattice and AMD Generated Modules.

AMD Module	Lattice Module
.din	.wr_data_i
.rd_clk	.rd_clk_i
.rd_en	.rd_en_i
.rst	.rst_i
—	.rp_rst_i*
.wr_clk	.wr_clk_i
.wr_en	.wr_en_i
.dout	.rd_data_o
.empty	.empty_o
.full	.full_o

***Note**: .rp_rst_i is the FIFO pointer reset. AMD module does not have this port. You can keep it open or tie it to the module reset signal.

4.5. Unrecognized Architecture Primitive

The example below is a *BUFHCE* module port map. There is no lower-level module or IP generated module that correspond to *BUFHCE* in the AMD project environment. This is a primitive that is part of the AMD architecture and does not need to be described. It will be recognized automatically by the synthesis tool as it is part of the AMD target device library.

In this case, find an equivalent component in the Lattice architecture that does the same function. In majority of cases, there are equivalent architecture elements that could be used. In the absence of that, find a way to emulate that function with a custom design.

In the example below, the equivalent primitive to *BUFHCE* is *DCC*:

```
// AMD HDL code
BUFHCE (       .O        (clk_samp),   // 1-bit The output of the BUFH
       .CE       (en_clk_samp),// 1-bit Enables propagation of signal from I to O
       .I        (clk_tx)      // 1-bit The input to the BUFH
  );
//LSCC Conversion
DCC DCSInst0 (
.CLKI (clk_tx),
.CE (en_clk_samp),
.CLKO (clk_samp)
);
```

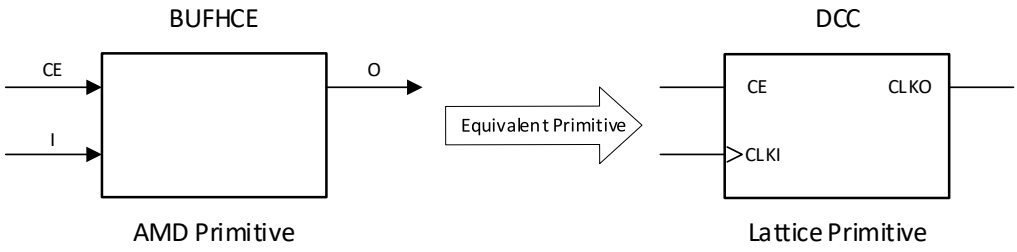


Figure 4.3. AMD BUFHCE Equivalent Primitive is Lattice DCC Primitive

4.6. I/O Buffer Primitives

Some of the older generation AMD devices require the use of buffer instantiation for differential signals and special buffer such as clock and GSR. Lattice software does not require this extra step, the buffer is automatically inferred during the synthesis phase based on the signal used in the design. For example, the differential buffer can be treated as single ended in the HDL code, the software automatically infers the differential buffer when the signal is defined as differential signal in the constraint editor. Table 4.2 lists the commonly used buffers in the AMD FPGA devices.

Table 4.2. Commonly Used Buffers

AMD Primitive	Description	Lattice equivalent
IBUF	Input Buffer	All these buffer types can be replaced with Signal or wire declaration in the HDL code with pin or signal attribute in the software constraint editor.
OBUF	Output Buffer	
BUFG_FABRIC	Global Clock Buffer driven by fabric interconnect	
IBUFDS	Differential Input Buffer	

4.6.1. Design Conversion Recommendations

Refer to the following list of recommendations for design conversion:

- Remove primitive from the HDL code.
- Replace the primitive with a signal declaration in the entity/module or in the signal declaration section.
- If the primitive is an I/O with special type such as differential and clock, use the lattice constraint editor to insert the required attribute.

Refer to the following codes for a conversion example:

- Verilog Example:

```
// AMD HDL code
IBUF IBUF_rst_i0      (.I (rst_pin),      .O (rst_i));
OBUF OBUF_txd         (.I(txd_o),         .O(txd_pin));
//LSCC Conversion
  assign rst_i = rst_pin;
  assign txd_pin = txd_o      ;
```

The following HDL code does not instantiate any input or output buffer nor clock buffer.

At the synthesis level, buffers are added to the design with differentiation between clock buffer and I/O buffer. Refer to the example below for the output of the Synplify Pro Synthesis tool.

- VHDL Example:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity XORGate is
    Port (       clk, rst : in STD_LOGIC;
                 A,B : in STD_LOGIC;
                 C : out STD_LOGIC);
end XORGate;

architecture Behavioral of XORGate is
begin
process (clk, rst)
begin
          if rst='1' then c <='0';
          elsif clk'event and clk='1' then C <= A Xor B;
          end if;
end process;
end Behavioral;
```

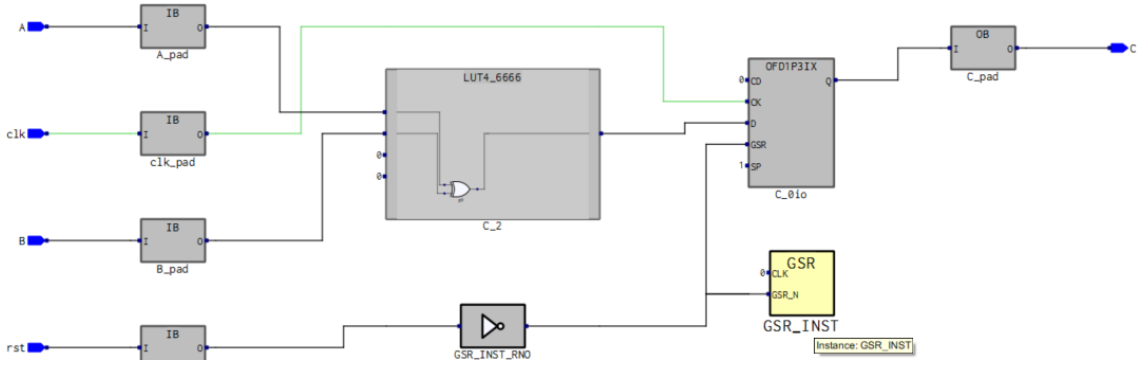


Figure 4.4. RTL View of Input, Output and Clock Buffers Automatically Inferred by Lattice Software

By using the software constraint editor, you can add any I/O type specific attribute. For example, as shown in Figure 4.5, input A is selected as an LVDS type.

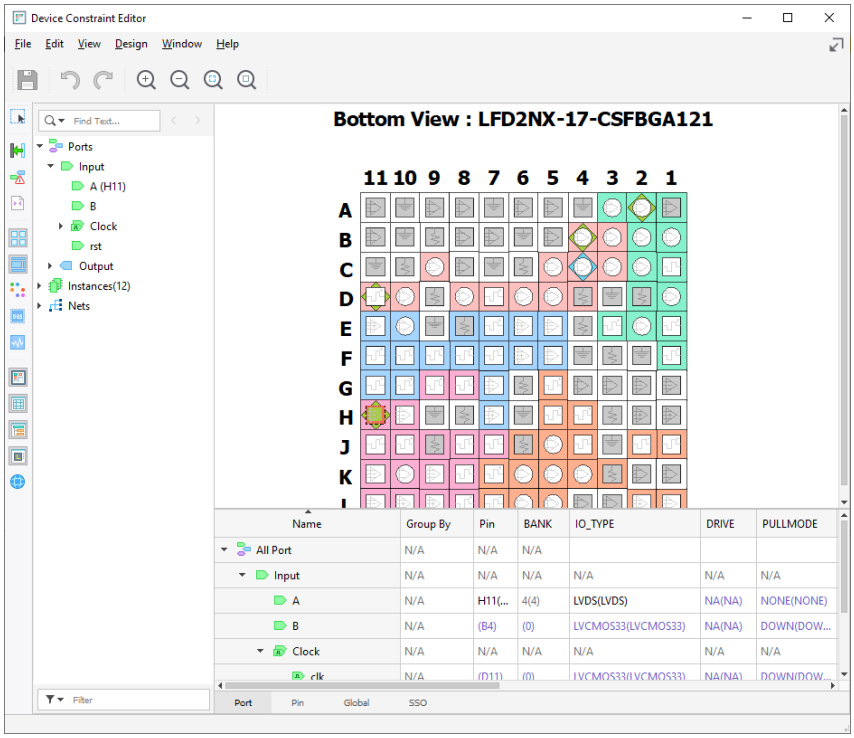


Figure 4.5. Device constraint Editor view of LVDS Buffer Type attribute (Input A)

Map report shows that pin A has been locked to pin H11 with buffer type LVDS.

At the Map level, there is a DRC check to validate the pin location selection and its compatibility with the I/O bank as well as any other constraints with the I/O Bank. Refer to Figure 4.6 for more information.

IO (PIO) Attributes

IO Name	Direction	Levelmode IO_TYPE	IO REG	IO DDR	Special IO Buffer
C	OUTPUT	LVCMOS33	O		
B	INPUT	LVCMOS33			
A	INPUT	LVDS			
rst	INPUT	LVCMOS33			
clk	INPUT	LVCMOS33			

Figure 4.6. I/O Attributes in Map Report

During the place and route (PAR) stage, the input buffer will be replaced with a differential buffer. Signal A is assumed to be the positive pair of the differential signal (*A+*). Locking *A+* will automatically reserves the *A–* to be used by the internal buffer. PAR report shows the pin allocation with the buffer type.

Notice that *A+* is located at pin H11 and pin H10 is reserved for the *A–* complement version of the same signal.

Similarly, you can see the *clk* signal is placed on a dedicated clock I/O. PCLKT0_0 is a dual function pin that can be used as a clock or regular I/O. The *T* in the clock name refers to the *True* version of the pin. *C* is the reference complement version of the clock if it is a differential signal (PCLKC0_0 will reference the complementary pin when the clock is used in the differential mode). Refer to Figure 4.7 for more information.

Pinout by Pin Number:

Pin/Bank	Pin Info	Preference	Buffer Type	Site	Dual Function
A2/1	rst		LVCMOS33_IN	PR5B	TDO/SSO/SD1/S3_OUT
A3/1	unused, PULL:DOWN			PR5A	TDI/SSI/SD0/S5_OUT
APIO_R25C74E/0				APIO_R25C74E	
B1/1	unused, PULL:DOWN			PR3B	TMS/SCSN/S5_IN
B2/1	unused, PULL:DOWN			PR7A	SD3/SDA/USER_SDA/S4_IN
B3/0	unused, PULL:DOWN			PT67A	S7_IN
B4/0	B		LVCMOS33_IN	PT57A	INITN
C1/1	unused, PULL:DOWN			PR9A	TCK/SCLK/PMU_EXT_CLK/S9_IN
C2/1	unused, PULL:DOWN			PR7B	SD2/SCL/USER_SCL/S3_IN
C3/0	unused, PULL:DOWN			PT67B	S7_OUT
C4/0	C		LVCMOS33_OUT	PT57B	PROGRAMN
C5/0	unused, PULL:DOWN			PT63B	MD2/SD6/S2_OUT
C9/0	unused, PULL:DOWN			PT65A	MD3/SD7/S6_IN/OSC_BURST
D1/1	unused, PULL:DOWN			PR9B	PMU_WAKEUPN/S4_OUT/EIO
D5/0	unused, PULL:DOWN			PT63A	MISO/MD1/SD5/S2_IN/OSC_HI
D6/0	unused, PULL:DOWN			PT61B	MOSI/MD0/SD4/S1_OUT
D7/0	unused, PULL:DOWN			PT61A	MCSN/S1_IN/PCLKT0_1
D8/0	unused, PULL:DOWN			PT65B	MCSNO/MSDO/S6_OUT
D10/0	unused, PULL:DOWN			PT59B	DONE/S0_OUT
D11/0	clk		LVCMOS33 IN	PT59A	MCLK/S0 IN/PCLKT0 0
H10/4	A-		LVDS_IN	PB44B	BDQ46
H11/4	A+	LOCATED	LVDS_IN	PB44A	BDQ46

Figure 4.7. Pinout by Pin Number in Map Report

4.7. HDL Attributes

4.7.1. Introduction

Both AMD and Lattice devices have attributes that can be used to guide the implementation of a design. The syntax of these attributes may differ between the two devices. This section covers the common attributes and provide AMD equivalent syntax that you can use for Lattice implementation. The attributes are segmented into the following categories:

- Synthesis specific attributes (Syn_)
- FPGA architecture attributes
- Physical placement attributes

In general, the HDL attributes are constraints that are attached as text to design objects which are interpreted by the software. A design object can be a specific port, component pin, net, instance, instantiation, or even an entire design.

An attribute provides information about the object. For example, an attribute might specify where a component in the logical design must be placed in the physical device, or it might specify a frequency constraint for a net that timing-driven place and route will attempt to meet.

HDL attributes are typically declared using comment notation in the Verilog HDL or Attribute keyword in the VHDL.

For more information, refer to the following documents:

- FPGA Libraries Reference Guide
- HDL Coding Guidelines
- Design Planning in Diamond

4.7.2. Common Synthesis Attributes Conversion Table

Lattice device has two synthesis options which are LSE (Lattice Synthesis Engine) and Synplify Pro. Most source code HDL attributes are compatible these two options. When using Synplify Pro, the AMD HDL attributes stay unchanged unless there are architecture specific attributes.

Refer to the respective documentation for details about all the attributes. Table 4.3 lists the commonly used attributes as examples.

Table 4.3. Commonly Used HDL Attributes

AMD Attribute	Lattice Attribute	Descriptions
use_dsp	syn_multstyle	Specifies whether the multipliers are implemented as dedicated hardware blocks or as logic.
fsm_extract	syn_state_machine	Enables/disables the state-machine optimization.
fsm_encoding	syn_encoding	Specifies the encoding style for a finite state machine (FSM), overriding the default synthesis encoding.

AMD Attribute	Lattice Attribute	Descriptions
box_type	syn_black_box	Specifies that a Verilog module or VHDL architecture declaration is for a box without lower-level description or has an encrypted netlist.
ram_style	syn_ramstyle	Specifies the implementation to use for an inferred RAM.
iob	syn_useioff	Packs registers into I/O pad cells.
translate_off/ translate_on	translate_off/ translate_on	Allows you to synthesize designs originally written for use with other synthesis tools without needing to modify the source code.

4.7.3. Common Architecture Attributes Conversion Table

The architecture physical attribute can be entered in the HDL file to guide the design implementation. The architecture attributes can be related to the pin placement, I/O voltage, and programmable features, such as slow rate and drive, or physical placement in the device (floor planning).

Table 4.4 describes the HDL attributes that are commonly used as constraints in the HDL source files. These attributes generally apply to all Lattice designs and are not specific to any device family.

AMD attributes have different syntax but offer the same capabilities. When converting an AMD design, these attributes need to be replaced.

Table 4.4. Commonly Used Architecture Attributes

AMD Attribute	Lattice Attribute	Descriptions	Supporting Lattice Device
LOC	LOC	Specifies a site location for a component or an I/O pin.	All
DRIVE	DRIVE	This attribute is available for output standards that support programmable drive strength.	All
IOSTANDARD	IO_TYPE	Sets the I/O standard for an I/O (input, output, and bidirectional buffers such as IB, OB, and BB).	All
DIFF_TERM	DIFFRESISTOR	This attribute is attached to the input and output buffers such as the IB and OB. It is used to provide differential termination.	LIFCL, LFD2NX, and UT24C
PULLTYPE	PULLMODE	The following PULLMODE values are available: • UP (default) • DOWN • NONE • KEEPER • PCICLAMPPULLMODE = 100K, 3P3K, 6P8K, 10K, NA	All
SLEW	SLEWRATE	Controls each I/O pin that has an individual slew rate control.	LIFCL, LFD2NX, and UT24C

Refer to the **Reference Guides** > **Constraints Reference Guide** > **HDL Attributes** in the Lattice Radiant or Diamond software Help menu for more details.

Refer to the following AMD attributes for a conversion example:

- Verilog Example:

```
(* LOC = "SLICE_X0Y0" *) reg placed_reg;
(* DRIVE = "2" *) output STATUS,
(* IOSTANDARD = "LVCMOS12" *) output STATUS,
```

- VHDL Example:

```
attribute LOC : string;
attribute LOC of placed_reg : label is "SLICE_X0Y0";

attribute DRIVE : integer;
attribute DRIVE of STATUS : signal is 2;

attribute IOSTANDARD : string;
attribute IOSTANDARD of STATUS: signal is "LVCMOS12";
```

Conversion to Lattice attributes :

- Verilog Example:

```
reg placed_reg /* synthesis loc="R40C47" */;
PinType STATUS /* synthesis IO_TYPE="[type_name]" DRIVE="[drive_strength]"
PULLMODE="[mode]" SLEWRATE="[value]"*/;
```

- VHDL Example:

```
ATTRIBUTE LOC : string;
ATTRIBUTE LOC OF placed_reg: SIGNAL IS "R5C5D";

ATTRIBUTE IO_TYPE : string;
ATTRIBUTE IO_TYPE OF [pin_name]: SIGNAL IS "[type_name]";
```

4.7.4. Physical Placement Attributes

Physical placement constraints are used to control the design part placement on a fabric. Both Lattice and AMD software platforms support placement features. The followings are multiple ways available to enter these constraints:

- HDL code through attributes (Lattice and AMD devices have different HDL syntax)
- Constraint files:
  - AMD software: .ucf or .xdc file
  - Lattice software: .ldc or .pdc file
- GUI interface part of the software suite:
  - AMD software: PlanAhead™ or Floorplanning view
  - Lattice software: Floorplan or Physical Designer

Table 4.5 summarizes the placement attributes that are used in the Lattice software for both Radiant and Diamond software platforms. Note that Radiant and Diamond software have different attribute syntax.

Table 4.5. Lattice Radiant and Diamond Software Placement Attributes

Lattice Software	Attribute	Descriptions
Radiant Software	BBOX	Indicates the bounding box or the area given in number of rows and columns for a given GRP. This attribute must appear on the same block as the GRP attribute.
	GRP	Universal grouping construct. Use this attribute to group blocks within different hierarchies or with no hierarchy.
	RBBOX	Indicates the area size of a region. This attribute must appear on the same block as the REGION attribute.
	REGION	Indicates the region to which a given GRP belongs to. This attribute must appear on a block that has a GRP attribute.
	RLOC/GLOC	RLOC/GLOC (Region Lock/Group Lock) can only be used with REGION or GRP attributes.
Diamond Software	BBOX	Indicates the bounding box or the area given in number of rows and columns for a given UGROUP.
	HGROUP	Hierarchical grouping construct. Use this attribute to group components that are to be instantiated multiple times.
	HULOC	Indicates the physical location of the northwest corner of an HGROUP or UGROUP assignment.
	HURLOC	Indicates the northwest corner of a region for a given HGROUP or UGROUP definition. This attribute must appear on the same block as the REGION attribute.
	RBBOX	RBBOX indicates the area size of a region. This attribute must appear on the same block as the REGION attribute.
	REGION	REGION indicates the region to which a given HGROUP or UGROUP belongs to. This attribute must appear on a block that has a HGROUP or UGROUP attribute.
	UGROUP	Universal grouping construct. Use the UGROUP attribute to group blocks within different hierarchies or with no hierarchy. UGROUP differs from HGROUP attribute in that its identifier is not changed by pre-appending the hierarchy and the block instance.

Refer to the following codes for the Radiant software HDL examples:

- VHDL example:

```
attribute REGION: string;
attribute REGION of <object>: label is "<REGION_name>";
attribute RLOC: string;
attribute RLOC of <object>: label is "<site>";
attribute RBBOX: string;
attribute RBBOX of <object>: label is "<height>,<width>";
attribute GRP: string;
attribute GRP of <object>: label is "<reg_GRP>";
attribute GLOC: string;
attribute GLOC of <object>: label is "<site>";
attribute BBOX: string;
attribute BBOX of <object>: label is "<height>,<width>";
```

- Verilog example:

```
"module serial_reg_custom(D, CLK, CE, RST, Q) /*  synthesis REGION="reg_REGION"
 RLOC= "R5C19D"
 RBBOX= "20,15"
 GRP= "reg_GRP"
 GLOC="R10C20D"
 BBOX= "5,5"*/;

serial_reg_custom inst1A(.D(A), .CLK(CLK), .CE(CE), .RST(RST), .Q(adder1_in1));
serial_reg_custom inst1B(.D(B), .CLK(CLK), .CE(CE), .RST(RST), .Q(adder1_in2));
serial_reg_custom inst1C(.D(adder1_sum), .CLK(CLK), .CE(CE), .RST(RST), .Q(SUM));

count count_inst(A,CLK,RST) /* synthesis REGION="reg_REGION" */;
```

Refer to the following codes for the Diamond software HDL examples:

- VHDL Syntax:

```
attribute HGROUP: string;
attribute BBOX: string;
```

- VHDL Example Code:

```
attribute HGROUP of struct: architecture is "reg_group";
attribute BBOX  of struct: architecture is "5,5";
```

- Verilog Syntax – Synplify

```
/* synthesis HGROUP= "<hgroup_name>"
BBOX= "<h,w>" */;
```

- Verilog Example Code – Synplify

```
/* synthesis HGROUP= "reg_group"
BBOX= "5,5" */;
```

4.7.5. Attributes Conversion Examples

This section provides the conversion examples for the RAM_STYLE, BLACK_BOX, and FSM_ENCODING attributes:

- RAM_STYLE attribute:
  - AMD VHDL example:
    ```
    attribute ram_style : string;
    attribute ram_style of myram : signal is "distributed";
    ```
  - Conversion to Lattice attribute:
    ```
    attribute syn_ramstyle: string;
    attribute syn_ramstyle of myram : signal is "distributed";
    ```
  - Conversion to Lattice Verilog attribute:
    ```
    module myram (datain,dataout,clk);
    output [31:0] dataout;
    input clk;
    input [31:0] datain;
    reg [7:0] dataout[31:0] /* synthesis syn_ramstyle="block_ram" */;

    * Lattice Attribute options: Registers, Distributed, Block_ram
    ```
- BLACK_BOX attribute:
  - AMD VHDL example:
    ```
    attribute black_box  : string;
    attribute black_box  of beh : architecture is "yes";
    ```
  - Conversion to Lattice attribute:
    ```
    attribute syn_black_box: string;
    attribute syn_black_box of beh : architecture is true;
    ```
  - Conversion to Lattice Verilog attribute:
    ```
    module bl_box(out,data,clk) /* synthesis syn_black_box */;
    ```
- FSM_ENCODING attribute:
  - AMD VHDL example:
    ```
    type count_state is (zero, one, two, three, four, five, six, seven);
    signal my_state : count_state;
    attribute fsm_encoding : string;
    attribute fsm_encoding of my_state : signal is "sequential";
    ```
  - AMD Verilog example:
    ```
    (* fsm_encoding = "one_hot" *) reg [7:0] my_state;
    ```
  - Conversion to Lattice VHDL attribute:
    ```
    attribute syn_encoding : string;
    attribute syn_encoding of my_state : signal is "sequential";
    ```

- Conversion to Lattice Verilog attribute:

```
reg [7:0] my_state /* synthesis syn_encoding = "onehot" */;
reg [7:0] my_state /* synthesis syn_encoding = "Safe, onehot" */;

* Lattice Attribute options: Sequential, onehot, Gray, Safe (combined with any other
type)
```

A comprehensive guide to library element HDL attributes is available in the FPGA Libraries Reference Guide. You can access this guide by navigating to **Reference Guides** > **FPGA Libraries Reference Guide** from the Lattice Radiant software Help menu. You can also find more information in the **Reference Guides** > **Constraints Reference Guide** > **Lattice Synthesis Engine Constraints** > **Lattice Synthesis Engine-Supported HDL Attributes** from the Lattice Radiant software Help menu.

It is not recommended to edit these library element attributes, as they are usually generated from an array of choices that are made for module generation using the IP Catalog.

5. Software Tools Comparison

5.1. Introduction

The Lattice design flow for FPGA and CPLD devices is similar in conception and implementation to the AMD design flow. Both software support hardware description language (VHDL and/or Verilog) as input, that can be targeted to a specific FPGA device family. Table 5.1 lists the different tools available for the AMD devices and their equivalent Lattice tools.

Table 5.1. Software Tools Comparison Between Lattice and AMD Devices

AMD Software Tool	Lattice Software Tool	Descriptions
Vivado Design Suite: HL Design Edition for the following AMD devices: • UltraScale • UltraScale+ • 7-Series	Lattice Radiant Lattice Diamond	Radiant software is the new generation software platform that supports new devices. Refer to Table 5.3 for list of supported devices.
ISE* Design Suite for the following AMD devices: • Spartan-6 • Virtex-6 • CoolRunner* • Previous generations	Lattice Radiant or Lattice Diamond	Depending on the selected equivalent device, Radiant or Diamond software could be used. Refer to Table 5.3 for list of supported devices.
Vitis™	Lattice Propel	Vivado/Diamond and Radiant software offer a hardware-centric approach to designing a hardware, while Vitis/Propel software offer a software-centric approach to develop both hardware and software SoC.

***Note**: AMD ISE Design Suite software has been discontinued and is replaced by the Vivado Design Suite software. The latest version of AMD ISE Design Suite software was released in October 2013.

Table 5.2 provides the descriptions of each Lattice software tool.

Table 5.2. Lattice Software Tools Descriptions

Lattice Software Tool	Descriptions
Lattice Radiant	The latest generation FPGA design software environment for Lattice devices. It will replace the Diamond software in the future. Only new devices are supported by this platform.
Lattice Diamond	Provides an optimized, and tailored design and verification environment for Lattice FPGAs featuring extensive constraints, advanced optimization, accurate analysis, extensive verification, and fast iterations.
Lattice Propel	Provides a complete set of graphical and command-line tools to create, analyze, compile, and debug both the hardware design of an FPGA-based processor system, and the software design for that processor system.

Table 5.3 provides the summary of Lattice device families supported by each Lattice software tool.

Table 5.3. Lattice Software Tools and Supported Device Families

Device Family	Radiant Software	Diamond Software	Propel Software	Programmer
iCE40 UltraPlus	✓	×	×	✓
iCE40	×	×	×	✓
MachXO2/MachXO3/ MachXO3D	×	✓	✓	✓
ECP5-5G	×	✓	×	✓
CrossLink/CrossLinkPlus	×	✓	×	✓
Crosslink-NX	✓	×	×	✓
Certus-NX	✓	×	✓	✓
CertusPro-NX	✓	×	✓	✓

Device Family	Radiant Software	Diamond Software	Propel Software	Programmer
LatticeXP2	✖	✓	✖	✓
Avant	✓	✖	✓	✓

5.2. Design Flow Using GUI

This chapter covers the following information:

- The differences in terminology and process between the AMD and Lattice tools.
- Methods using similar tools when converting designs from one platform to another.

5.2.1. Introduction

Both AMD and Lattice hardware development platform integrate multiple GUI tools that allow you to go through the FPGA design process and generate device bitstream to download in the FPGA.

Most of the capabilities of the Vivado tools have their equivalent within the Diamond or Radiant tool. However, the terminology used by each vendor and how to access these tools differs.

There are different views within the AMD Vivado tool to access design or reporting tools based on the phases of your design (IO Planning, Floor planning, and Timing analysis).

All the same tools are accessible through the main interface without changing the view and are classified under the menu “Tools” or through shortcuts in the Lattice Radiant or Diamond tools bar. An example of the Radiant software main interface is shown in Figure 5.1. Notice that all the software tools are accessible from the tool’s menu or top shortcut bar. Multiple tabs are available on each view to customize the interface.

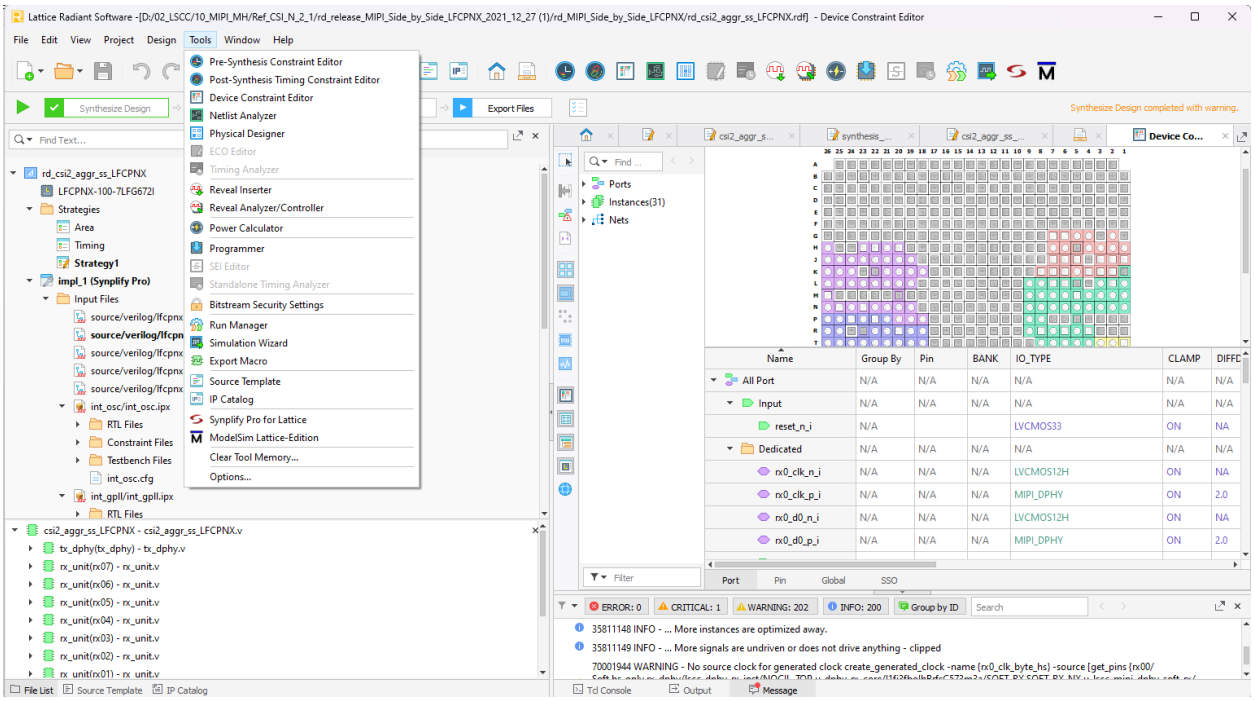


Figure 5.1. Lattice Radiant Software Main Interface

Table 5.4 lists most of the available AMD software tools and their equivalent Lattice software tools focusing on the Radiant software. Lattice Diamond software has similar tools and naming. For further details on how to use these features refer to the Lattice Radiant Software Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02165) and Lattice Diamond Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02169).

Table 5.4. Software Tool Comparison Between Lattice and AMD Software

AMD Vivado Design Suite Software	Lattice Radiant Software	Purpose
Vivado main interface	Radiant main interface	The main tool interface which can be started from the Windows or Linux operating systems. TCL console is also available.
Project Manager > Setting	Strategies	Project setting, and optimization or implementation settings.
Project Manager > IP Catalog	IP Catalog	GUI tool that allows you to customize the architecture primitive or IP.
Project Manager > Language template	Source Template	Provides an example source code for a VHDL or Verilog construct.
Simulation > run simulation	ModelSim™ Simulation Wizard	The simulation wizard that allows you to set and run simulation environment.
RTL Analysis (Schematic)	Netlist Analyzer with LSE HDL Analyst with Synplify Pro	Design RTL schematic view
Vivado Design Suite Synthesis	Lattice Synthesis Engine (LSE) Synplify Pro	The two options of Synthesis tools are available with Lattice free version.
Vivado Implementation	Map Design Place & Route Design	Design Map, and Place and Route tools.
I/O Planning Device view and package view.	Device Constraint Editor Physical Designer	Tools to help select and lock different pins of your design in package view. Physical design shows the floorplan view.
Implementation > Constraint Wizard	Timing Constraint Editor	An interface that helps you set all the timing constraint of your design.
Timing Analysis	Timing Analyzer	A GUI tool to analyze timing with cross probing capability.
Design Runs	Run Manager	Run Manager helps you to manage running multiple project implementations and to compare the results.
Vivado logic analyzer (Hardware Manager)	Reveal Inserter Reveal Analyzer/Viewer	On-chip-logic analyzer.
AMD Power Estimator (XPE) Report Power	Power Calculator	An integrated power calculator tool.
Vivado Programmer (Hardware Manager)	Programmer	Hardware programming tool
AMD Reporting	Reports	Report viewer tab summarizes all your design reporting: Synthesis, Map, PAR, Timing, and Bitgeneration.
Vitis Core Development Kit AMD Software Development Kit (XSDK)	Lattice Propel Builder	Note that Lattice uses RISC-V as the embedded processor.

When going through the implementation process, multiple files are generated for each phase. The extension of these files differ between AMD and Lattice software. Table 5.5 shows a summary of commonly used files and their file extensions that can be used for debug purposes.

Table 5.5. Extension File Comparison Between Lattice and AMD Software

File Type	AMD Vivado Design Suite Software	Lattice Radiant or Diamond Software	Descriptions
Project file	.xpr	.rdf .ldf	• Radiant software project file is .rdf. • Diamond software project file is .ldf.
Design constraint file	.xdc	.ldc .pdc .lpf .sdc	• Radiant software Pre-synthesis constraints file is *.ldc. • Radiant software Post synthesis constraint file is *.pdc. • Diamond software post synthesis constraints file is *.lpf. • Pre-synthesis design and timing constraints based on SDC format file is *.sdc.

File Type	AMD Vivado Design Suite Software	Lattice Radiant or Diamond Software	Descriptions
IP generated module	.xci	.ipx	IP generated files with IP Catalog. Generated HDL or .ipx file could be added to the design project.

5.2.2. FPGA Design Flow

When creating designs for FPGA devices, Lattice and AMD software tools have similarities in terms of concepts, approach, and functionality. Lattice Radiant and Diamond software framework technology uses the typical FPGA design flow that adheres to a sequence of steps, which initially requires setting up the design environment and ends with the generation of programming files that are used to program the hardware.

The AMD Vivado tools design process is segmented as follows:

- Project Manager: To set all project parameters.
- RTL Analysis: To view design architecture and hierarchy.
- Synthesis: To run synthesis and view results.
- Implementation: To set design implementation and constraints settings.
- Program and debug: Device programming related tools.

The Lattice design process, shown in Figure 5.2, includes the following:

- Project Setting: Device, Strategy, one or multiple implementations.
- Synthesis Design: Synthesis using LSE or Synplify Pro.
- Map Design: The process of mapping the design to the target device.
- Place & Route Design: The placement and routing on the target device.
- Export Files: To generate programming files, IBIS Model or other simulation files.

For more information on the design flow, refer to the Lattice Synthesis Engine for Diamond User Guide and see Figure 5.3 for more details on the Lattice design process.

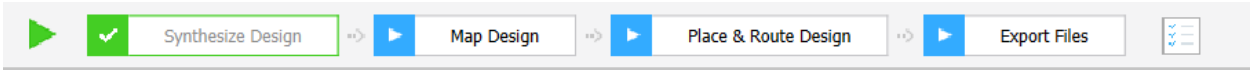


Figure 5.2. Lattice Radiant software Design Process

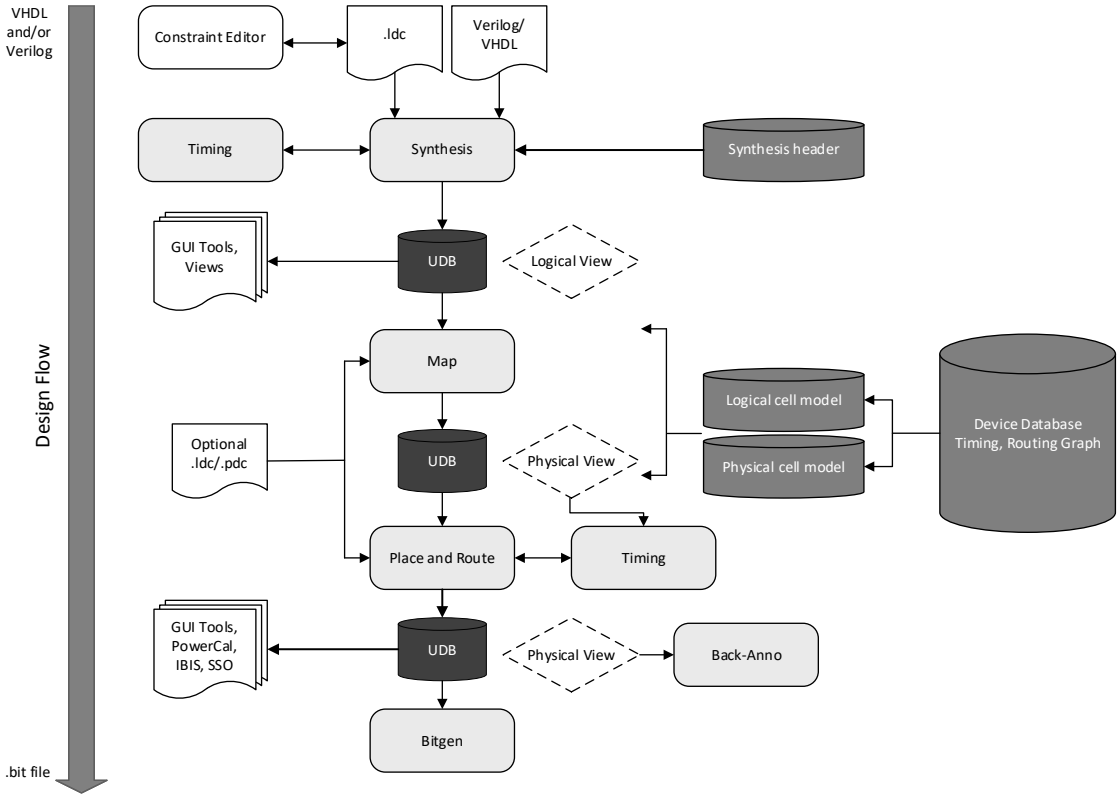


Figure 5.3. Lattice Design Flow

5.3. Design Flow Using TCL

Similar to AMD Vivado Design Suite software, Lattice Diamond software also supports TCL (Tool Command Language) scripting feature that enables a batch capability for running tools in the Diamond software graphical interface. TCL commands can be used through the command line/terminal or the Lattice Radiant or Diamond stand-alone TCL console that is included in the software package.

For further details on this subject refer to the following Lattice documentation:

- Tools Help: **Reference Guides** > **TCL Command Reference Guide**
- Lattice Radiant Software Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02165)
- Lattice Diamond Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02169)
- Lattice Radiant Software 3.2 User Guide
- Lattice Diamond 3.12 User Guide

6. Tools Constraint Compatibility

6.1. Introduction

The following types of constraint files are used in the Radiant software:

- *.ldc (via LSE)
- *.sdc (via Synplify Pro and LSE)
- *.fdc (via Synplify Pro)
- *.pdc constraint files

Figure 6.1 details the entry mechanism for the input files and their data flow through the Lattice Radiant software constraints flow process. Note that the Radiant software also generates a post-synthesis timing report just for LSE.

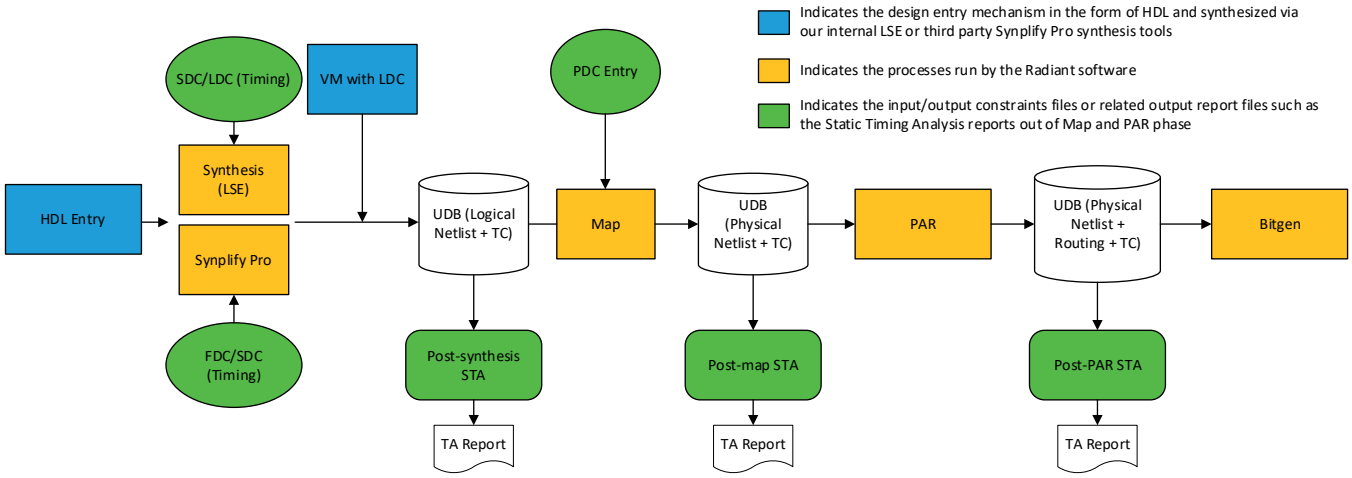


Figure 6.1. Lattice Radiant Software Constraints Flow Process

The important process to note is that the Unified Database (.udb) used to store the processed data dictates that certain stages of the constraints need not be re-run in the tool saving processing time. For example, physical constraints on RTL entered as attributes in the HDL, or pre-synthesis constraints entered using the Pre-Synthesis Timing Constraint Editor/Synplify Pro SCOPE (or text editor) including .ldc/.fdc (.sdc converted to .fdc) timing constraints would initially run through synthesis and the data stored is in the .udb file.

Subsequent timing and physical constraints entered using tools such as the Post-Synthesis Timing Constraint Editor, Placement Mode or Device Constraint Editor and stored in the .pdc file are processed via the mapping run without the need to re-synthesize the design since the pre-synthesis data resides in the .udb database. As the constraints flow proceeds with each process like the synthesis, Map, and PAR process, the .udb successively stores a physical netlist, routing, and timing constraint. The storage of accumulating data throughout the constraints flow in the .udb and reduction in the re-running processes is one reason for the improved speed and efficiency for the Lattice Radiant software.

In terms of the necessity of pre-synthesis versus post-synthesis timing constraints, usage depends on the complexity and desired performance of the design results. A simple design requiring few timing constraints with a relaxed fMAX may require only pre-synthesis timing constraints and runs through the flow from synthesis to place and route phase.

A complex design requiring a higher fMAX performance may have initial pre-synthesis timing constraints entered for synthesis. Later in the flow, you may want to override or fine tune these timing constraints in addition to adding physical constraints to reach the desired performance by driving the place and route process.

For example, you might want to specify a higher fMAX to reduce logic levels in the initial stage but lower the fMAX constraint later in the post-synthesis flow so that the place and route process is not strained when routing the design.

The Radiant software LSE enables you to set pre-synthesis Synopsys® Design (formatted) Constraints, which are directly interpreted by the synthesis engine.

When you use the LSE, these .sdc constraints are saved to a Lattice Design Constraints file (.ldc). You can create several .ldc files and select one of them to serve as the active synthesis constraint file for an implementation.

Refer to the Lattice Radiant Timing Constraints Methodology (FPGA-AN-02059) for further details on the use of the constraints and the Lattice Radiant Software 3.2 User Guide for further details on the Radiant software.

Figure 6.2 shows a detailed comprehensive representation of all the files involved on the logical domain and physical domain.

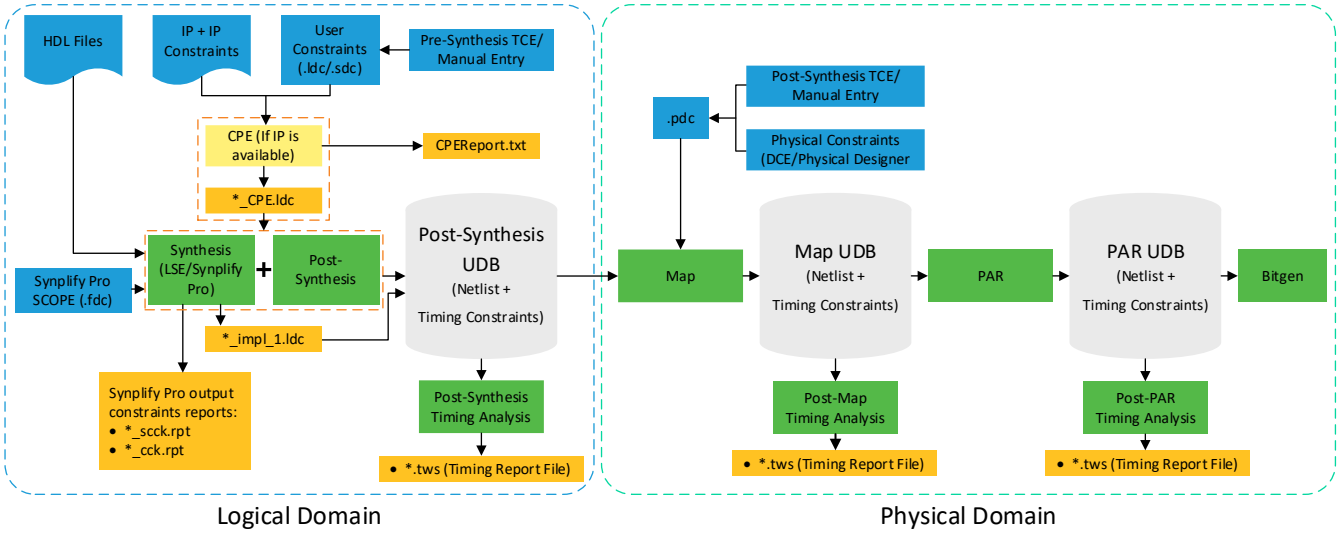


Figure 6.2. Input Files and Data Flow of Logical and Physical Domains

Notes:

- Regarding the physical/timing constraints, constraints entered via any GUI tool such as the Device Constraint Editor or Timing constraint editor will take precedence over the constraints that are entered via an HDL attribute in the RTL or .ldc TCL sdc command.
- Constraints entered later in the design flow such as the Post-Synthesis Timing Constraint editor (.pdc) will override a constraint entered in the Pre-Synthesis Constraint (.ldc or .sdc) editor tool.
- If there are conflicts in the constraint file such as a ldc_set_location TCL command conflicting with a ldc_prohibit on the same location, then the Radiant software will issue an error message in the design flow.

To complete the design conversion process, it is important to replace the AMD timing and device constraints (.ucf .sdc or .xdc) file with a Lattice Semiconductor source constraints or preferences file (.prf, .pdc or .sdc). See Table 6.1 for the equivalent Lattice Semiconductor preferences.

6.2. Converting SDC File

A synopsis design constraint standard format defined in the ASCII text file (with the extension .sdc) that contains design timing constraints. This usually does not require any changes as both AMD and Lattice software support SDC file format.

6.3. Converting UCF File

UCF files are used to interact with the ISE™ Design Suite software which is an old AMD implementation tool. UCF files are unique to AMD software and they provide a format for feeding physical and timing constraints into the AMD ISE tools. The conversion of this type of file can be difficult as you may have constraints that do not have direct equivalents. The same problem arises with the conversion of UCF to XDC (when you move from the old generation AMD tool ISE to Vivado). The AMD tool, PlanAhead tcl commands allows you to convert UCF to XDC. You can type in the following syntax:

```
write_xdc - file <file path and name>
```

This will automatically generate an XDC file compatible with Vivado tool (pay attention to the warning generated using this process as some constraints may not have direct equivalents.

XDC file is based on the SDC file construct and will be much easier to convert to Lattice constraint (.ldc or .pdc).

6.4. Converting XDC File

The Vivado IDE supports the AMD design constraint (XDC) and Synopsys design constraint (SDC) file formats. The SDC format is for timing constraints while the XDC format is for both timing and physical constraints. Constraints can include placement, timing, and I/O restrictions.

Most SDC files should be compatible without modification with some limitations. The current LSE timing does not take the PLL/DLL frequency or phase shift properties into account. It also does not model the different IO_TYPE in the PIO. Therefore, it is necessary to adjust the timing constraint accordingly.

6.5. Timing Constraint

6.5.1. Timing Constraints Conversion Table

Table 6.1 lists all the constraints that can be used by both Lattice (.ldc) and AMD (.xdc) software. The listed constraints are based on the SDC construct and are fully compatible.

Table 6.1. Timing Constraints Comparison Between Lattice and AMD Software

AMD Constraint (.xdc)	Lattice Constraint (.ldc)	Descriptions
create_clock	create_clock	Creates a clock and defines its characteristics.
create_generated_clock	create_generated_clock	Creates an internally generated clock and defines its characteristics.
set_clock_groups	set_clock_groups	Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during timing analysis.
set_clock_latency	set_clock_latency	Specifies the behavior of the clock outside of the FPGA device.
set_clock_uncertainty	set_clock_uncertainty	This constraint indicates that the clock of interest has uncertainties in its period.
set_false_path	set_false_path	Identifies paths that are considered false and excluded from timing analysis.
set_input_delay	set_input_delay	Defines the arrival time of an input relative to a clock.
set_load	set_load	To accurately perform timing and SSO analysis, the tool needs information regarding the external load capacitance of nets connected to output ports of the FPGA device.
set_max_delay	set_max_delay	Specifies the maximum delay for the timing paths.
set_min_delay	set_min_delay	Specifies the minimum delay for the timing paths.
set_multicycle_path	set_multicycle_path	Defines a path that takes multiple clock cycles.
set_output_delay	set_output_delay	Defines the output delay of an output relative to a clock.

6.5.2. Timing Constraint Best Practice

Timing constraint is very important for a given design implementation as it guides how the design are optimized and has an impact on the implementation process. Both Synthesis and implementation tools are timing driven. The algorithm will try to optimize timing scores (all negative Slacks in your design) to a value of zero. Having a partial coverage for timing constraint may lead the tool to optimize in the wrong area of the design and miss timing in other important regions.

Both the Radiant and Diamond software provide you with valuable information on your timing coverage. You can refer to the MAP and PAR report files to look for:

- Clock summary: Provides you with a summary of all identified clocks in your design with clock domain crossing to other clock domains.
- Timing constraint coverage: Provides you with a percentage of constraint coverage. Having a low number means that your design is not well constrained. It is recommended to be above 90% (ideally, 100% coverage).
- Total timing scores: Provides you with an indication of the total Negative Slack in picosecond. This is the score that the tool tries to optimize to zero.
- A list of Unconstraint paths.
- Setup and hold time report.

To have a good timing constraint coverage, it is recommended to define the following constraints in this order of priority:

- Identify different clocks and define your frequency constraint.
- Identify and constraint any generated clocks.
- Define input and output delay constraints.
- Define different clock relationship if any.
- Define false paths.
- Define multicycle paths.

For more information on timing constraints, refer to the **Reference Guides** > **Constraints Reference Guide** > **Lattice Synthesis Engine Constraints** > **Synopsys Design Constraints Timing/Physical Constraints** or **Reference Guides** > **Constraints Reference Guide** > **Lattice Synthesis Engine Constraints** > **Synopsys Design Constraints** from the Radiant software Help menu.

6.6. Physical Constraint

6.6.1. Definition

Physical constraints are related to the physical domain. The physical constraint guides the Map and PAR tools in the implementation process and can include the following:

- Pin placement, voltage per bank, and V_{ref}.
- Force the use of certain clock resources (Primary, region, and secondary)
- Creating a group with logical components
- Defining an anchor point for a group or a component
- Defining the configuration mode of the device

6.6.2. Physical Constraint Files

You can assign physical constraints using one or both of the following methods:

- Assign Lattice design constraints (.PDC) via the Device Constraint Editor tool. See the Device Constraint Editor Help menu for a complete description of each constraint, including syntax rules and examples.
- Assign HDL or schematic-based attributes using design source files. These attributes are used to direct Map and PAR tools. See HDL Attributes section from the Lattice Radiant/Diamond software Help menu for complete descriptions of each attribute, including conventions and examples.

Post-Synthesis Design Constraints (.pdc) is the Radiant software design constraints that contain both physical constraints and any post-synthesis timing constraints. The .pdc file is generated from higher level constraints and is the one that is used by Map and PAR tools to complete the physical implementation of the design.

6.6.3. Physical Constraints Conversion

The conversion of physical constraints have to be done manually. Table 6.2 lists the supported Lattice physical constraints.

Table 6.2. Lattice Physical Constraints

Lattice Physical Constraint	Descriptions
ldc_create_group	Defines a single identifier that refers to a group of objects. Only slice and IO can be created currently. Refer to the following example: ldc_create_group -name group1[get_ports{a*}]
ldc_create_region	Defines a rectangular area. Refer to the following example: ldc_create_region -name region0 -site R16C2D -width 11 -height 30
ldc_set_location	When applied to a specified component, it places the component at a specified site or bank and locks the component to the site or bank. Refer to the following examples: • ldc_set_location -site 11 [get_ports {A}] • ldc_set_location -region region0 [get_group {group1}]
ldc_create_vref	Defines a voltage reference. The PIO site serves as the input pin for an on-chip voltage reference. Refer to the following example: ldc_create_vref -name VREF1_BANK_3 -site N21
ldc_set_vcc	Sets the voltage and/or derate for the bank or core. Refer to the following examples: • ldc_set_vcc -bank 1 3.3 • ldc_set_vcc -bank 1 -derate -3
ldc_set_port	Sets the constraint attributes to ports; -iobuf, is used exclusively; -vref must be combined with –iobuf. Refer to the following example: ldc_set_port -iobuf {IO_TYPE=HSTL15_II PULLMODE=UP} [get_ports {A}]
ldc_set_sysconfig	Sets the sysConfig port attributes. Refer to the following example: ldc_set_sysconfig {JTAG_PORT=ENABLE PROGRAMN_PORT=ENABLE MCCLK_FREQ=56.2 DONE_OD=ON}
ldc_set_attribute	Sets the constraint attributes to the objects or the design if no object is specified. Refer to the following examples: • ldc_set_attribute {USE_PRIMARY=TRUE} {USE_PRIMARY_REGION=0,1} [get_nets {clk1_c}] • ldc_set_attribute GSR_NET=TRUE [get_nets {my_gsr}]
ldc_prohibit	Prohibits the use of a site or all sites in a region. Refer to the following examples: • ldc_prohibit -site AB • ldc_prohibit -region regionA

For more information on timing constraints, refer to the **Reference Guides** > **Constraints Reference Guide** > **Lattice Synthesis Engine Constraints** > **Synopsys Design Constraints Timing/Physical Constraints** or **Reference Guides** > **Constraints Reference Guide** > **Lattice Synthesis Engine Constraints** > **Synopsys Design Constraints** from the Radiant software Help menu.

6.7. XDC File Conversion Example

This section provides an example of converting an AMD XDC file to the Lattice LDC file:

- AMD XDC file example:

```
create_clock -period 10 -name wClk [get_ports wClk]
create_clock -period 5 -name rClk [get_ports rClk]
set_property PACKAGE_PIN P20 [get_ports {wbtData[7]}]
set_property PACKAGE_PIN V22 [get_ports {wbData[6]}]
set_property PACKAGE_PIN E21 [get_ports {wbData[5]}]
set_property PACKAGE_PIN P23 [get_ports {wbData[4]}]
set_property PACKAGE_PIN V23 [get_ports {wbData[3]}]
set_property PACKAGE_PIN E24 [get_ports {wbData[2]}]
set_property PACKAGE_PIN P25 [get_ports {wbtData[1]}]
set_property PACKAGE_PIN V26 [get_ports {wbData[0]}]
```

```
set_property PACKAGE_PIN E27 [get_ports wclk]
set_property PACKAGE_PIN E28 [get_ports rclk]
set_property IOSTANDARD LVCMOS18 [all_inputs]
set_property IOSTANDARD LVCMOS18 [all_outputs]
create_pblock pblock_1
add_cells_to_pblock [get_pblocks pblock_1] [get_cells -quiet [list arnd1 arnd2 arnd3
arnd4]]
resize_pblock [get_pblocks pblock_1] -add {SLICE_X10Y170:SLICE_X35Y194}
```

- Conversion to Lattice LDC file:

```
# Timing constraint
create_clock -name {rClk} -period 10 [get_ports rClk]
create_clock -name {wClk} -period 5 [get_ports wClk]

# I/O pin locking
ldc_set_location -site {L16} [get_ports wClk]
ldc_set_location -site {L18} [get_ports rClk]
ldc_set_location -site {L14} [get_ports {wbData[0]}]
ldc_set_location -site {L13} [get_ports {wbData[1]}]
ldc_set_location -site {M16} [get_ports {wbData[2]}]
ldc_set_location -site {M15} [get_ports {wbData[3]}]
ldc_set_location -site {M14} [get_ports {wbData[4]}]
ldc_set_location -site {M13} [get_ports {wbData[5]}]
ldc_set_location -site {P14} [get_ports {wbData[6]}]
ldc_set_location -site {P13} [get_ports {wbData[7]}]

# Default setting for all I/O to have LVCMOS33 voltage
ldc_set_port -iobuf {IO_TYPE=LVCMOS33 DIFFDRIVE=NA DIFFRESISTOR=OFF}

# Create a group box with defined size (10x10) and physical placement guideline (anchor
point R2C2D)
ldc_create_group -name MyGroup -bbox {10 10} [get_cells {arnd1 arnd2 arnd3 arnd4}]
ldc_set_location -site {R2C2D} [ldc_get_groups MyGroup]
```

7. Design Simulation

7.1. Supported Simulation Tools and Process

Both AMD Vivado Design Suite and Lattice Radiant/Diamond software support a number of third-party simulators, as shown in Table 7.1.

Table 7.1. Supported Simulation Tools and Process by AMD Vivado Design Suite and Lattice Radiant/Diamond Software

Simulation Tool	AMD Vivado Design Suite Software	Lattice Radiant/Diamond Software
Siemens EDA Questa™ Advanced Simulator	✓	✓
Siemens EDA ModelSim™ Simulator	✓	✓
Synopsys® VCS® (Verilog Compiler Simulator)	✓	✓
Aldec® Riviera-PRO™ Simulator	✓	✓
Aldec Active-HDL™	✓	✓
Cadence® Xcelium™ Parallel Simulator	✓	×
Cadence Incisive® Enterprise Simulator (IES)	✓	×
AMD Vivado™ simulator	✓	×
Cadence NC-VHDL	×	✓
Cadence NCSim	×	✓
Cadence NC-Verilog	×	✓

In the AMD Vivado Design Suite software, you can run a simulation by clicking **Run Simulation** under **Simulation** on the **Flow Navigator** pane. You can choose between Behavioral Simulation, Post-Synthesis Simulation, and Post-Implementation Simulation.

In the Diamond or Radiant software, you can click on the **Simulation Wizard** icon (or **tools** > **simulation wizard**) to run a simulation using Modelsim OEM tool. The simulation wizard is used to generate a Simulation Wizard Project (.spf) file and a simulation script DO file that is executed by ModelSim. Figure 7.1 shows an example of the Radiant software simulation wizard GUI.

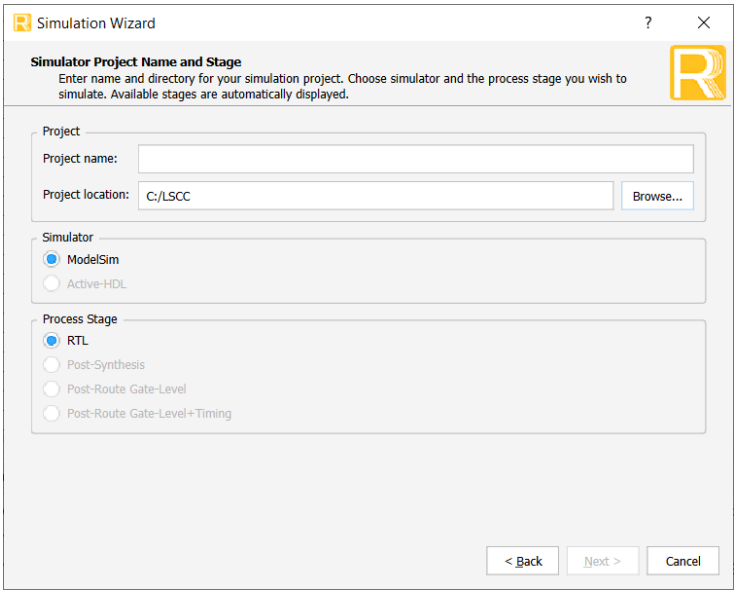


Figure 7.1. Lattice Radiant Software Simulation Wizard

Figure 7.2 shows the files involved in each type of simulation for the Radiant software.

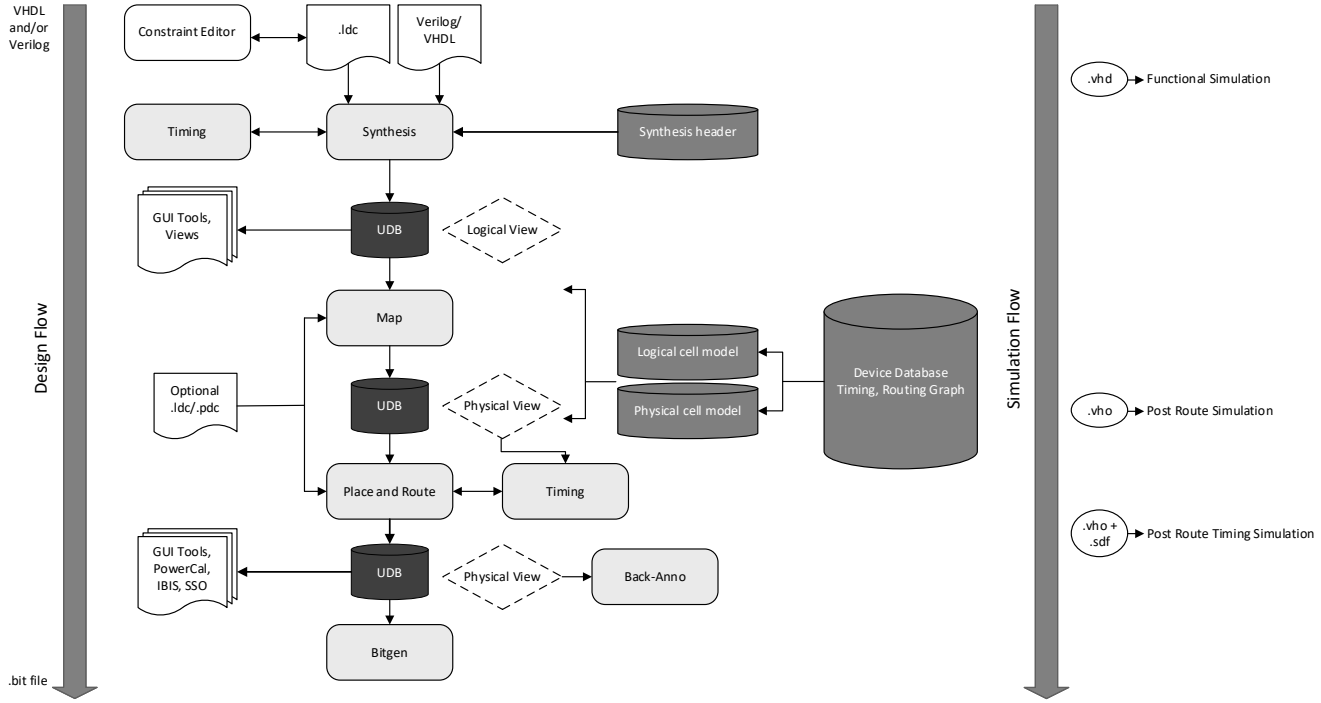


Figure 7.2. Radiant Software Design and Simulation Flows

For more information, refer to the user guides from the Lattice Radiant/Diamond software Help menu. Figure 7.3 shows an example of user guides listed in the Lattice Radiant software Help menu.

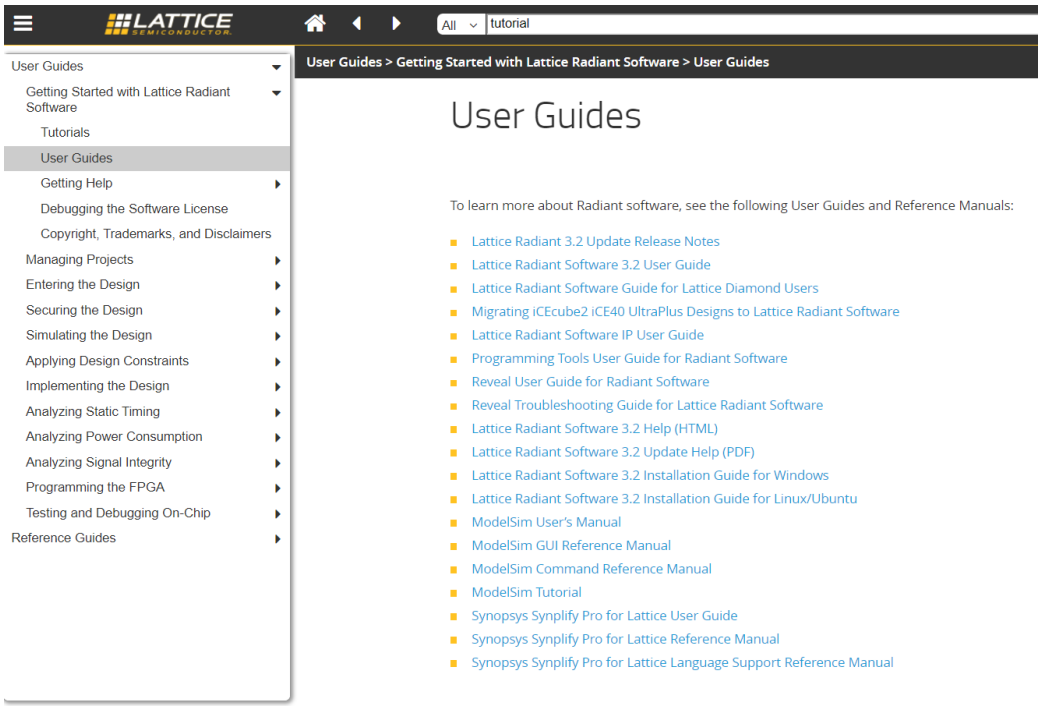


Figure 7.3. Lattice Radiant Software User Guides

8. Device Programming

8.1. Programming Mode Options

After you have created and verified your design, you can use the final output data file to download a bitstream to the FPGA device using the appropriate programming tool. There are multiple modes of device programming for both Lattice and AMD devices. For details about these modes, refer to the device datasheet or sysConfig user guide of a specific device family. A high-level summary of supported modes is listed in Table 8.1.

Table 8.1. Programming Mode Options Available By Device Family for AMD and Lattice Devices

Device Family/ Programming Mode	AMD 7-Series	Lattice MachXO	Lattice CertusPro-NX	Lattice ICE40 UltraPlus	Lattice CrossLink-NX
SDM*	×	✓ (Flash)	×	✓ (NVCM)	×
SPI	✓ (×1, ×2, and ×4)	✓	✓ (×1, ×2, and ×4)	✓	✓
I2C	×	✓	✓	×	✓
I3C	×	✓ (NX Version)	✓	×	✓
BPI (parallel I/F)	✓ (×8 and ×16)	×	×	×	×
Multiboot	✓	✓	✓	✓	✓
Encryption	✓	✓ (NX Version)	✓	×	✓ (NX Version)
Authentication	✓	✓ (NX Version)	✓	×	✓ (NX Version)
Partial Configuration	✓	×	×	×	×
Daisy Chaining	✓	✓	✓	×	✓
JTAG	×	✓	✓	✓	✓

***Note**: SDM stands for Self-Download-Mode. Some Lattice devices integrate Flash or NVCM. No external boot device is needed.

8.2. Bitstream Generation

There are different configuration options that are available in the Lattice Radiant and AMD Vivado Design Suite software that allow you to customize the generated bitstream.

The Vivado Design Suite software interface allows you to select the type of interface, bitstream format and the programming mode using the **Program and Debug** step of the flow.

Configuring the **additional bitstream settings** allows you to access the following advanced settings:

- Configuration mode
- Selection of the SPI flash mode (×1, ×2, and ×4)
- Bitstream encryption

In the Lattice software, all these parameters can be set from the following interfaces:

- Strategy Settings (bitstream format)
- Device Constraint Manager (to set the sysConfig port parameters)
- Programmer and Programmer file utility which include the bitstream conversion debug tools and device programming utility.

For more information, refer to the following documents:

- Lattice Radiant Software Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02165)
- Advanced Configuration Security User Guide for Nexus Platform (FPGA-TN-02176)
- Lattice Diamond Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02169)
- Lattice Radiant Software 3.2 User Guide
- Lattice Diamond 3.12 User Guide

8.2.1. Bitstream Strategy Settings

In the Radiant software, you can double-click on the **strategy** to set the output format of the bitstream file:

- Bit File (Binary) – Generates a binary configuration file (.bin) that contains the default outputs of the Bit Generation process.
- Raw Bit File (ASCII) – Generates an ASCII raw bit text file (.rbt) of ASCII ones and zeros that represent the bits in the bitstream file. If you are using a microprocessor to configure a single FPGA device, you can include the Raw Bit file in the source code as a text file to represent the configuration data. The sequence of characters in the Raw Bit file is the same as the bit sequence that will be written into the FPGA device.

Figure 8.1 shows an example of the Radiant software Strategies GUI.

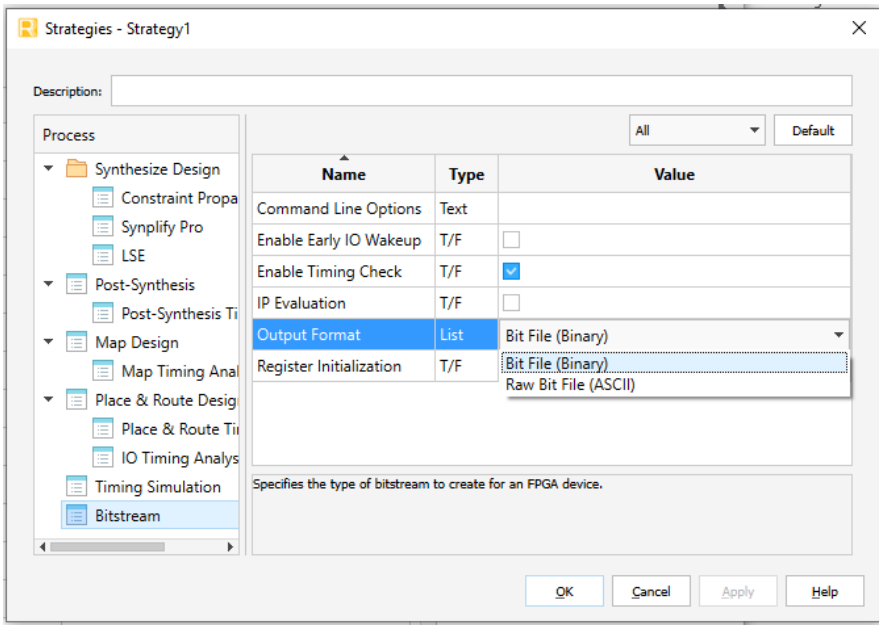


Figure 8.1. Lattice Radiant Software Strategies GUI

8.2.2. Device Constraint Options (sysConfig)

In the Radiant software, you can find all the SysConfig settings under the **General** tab. You can set these settings using the **Global** tab in the Device Constraint Editor or manually. If you do not specify these settings in the .pdc file, some default sysConfig constraints will automatically be generated based on the device selection. Figure 8.2 shows an example of the Radiant software Device Constraint Editor GUI.

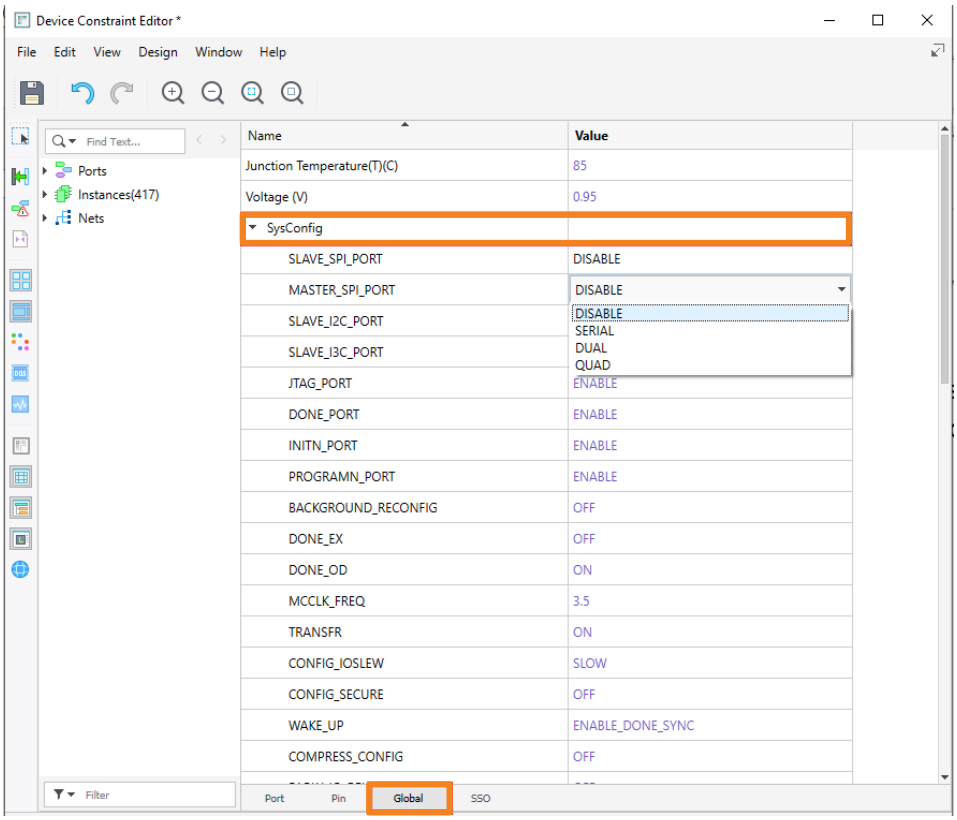


Figure 8.2. Lattice Radiant Software Device Constraint Editor GUI

8.2.3. Programmer and Programmer File Utility

The Radiant Programmer in the Radiant software allows you to connect with the hardware and program the device or flash memory. It offers several views to help you set up your connection to a target board and to program the FPGA devices. If an item is not showing, choose it in the **View** menu.

The configuration created by the Programmer will be stored in an .xcf file. The .xcf file contains information about each device, the data files targeted, and the operations to be performed.

Figure 8.3 shows an example of the Radiant Programmer GUI.

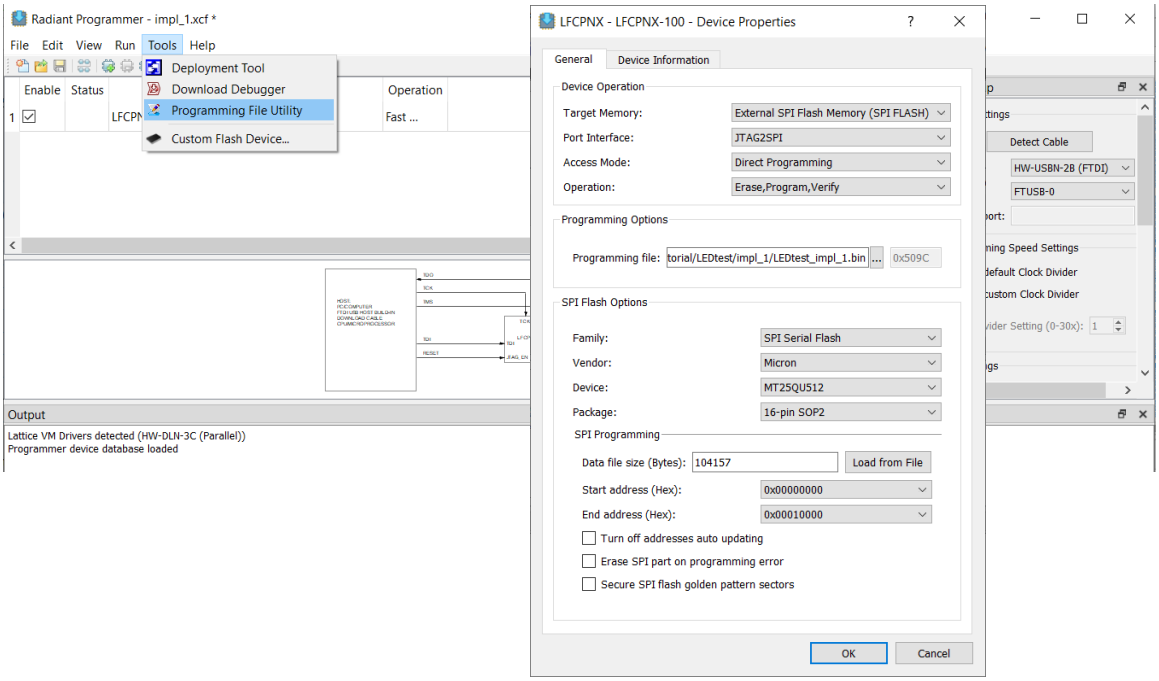


Figure 8.3. Lattice Radiant Programmer GUI

Accessible from the same interface is the **Programming File Utility**, shown in Figure 8.4, it is a stand-alone tool that allows you to view, compare, and edit data files. The tool covers the following functions:

- Viewing Data Files
- Comparing Two Data Files
- Editing Feature Row Values
- Editing Control Register Values
- Control Register Dialog Box
- Editing the USERCODE in the Data File

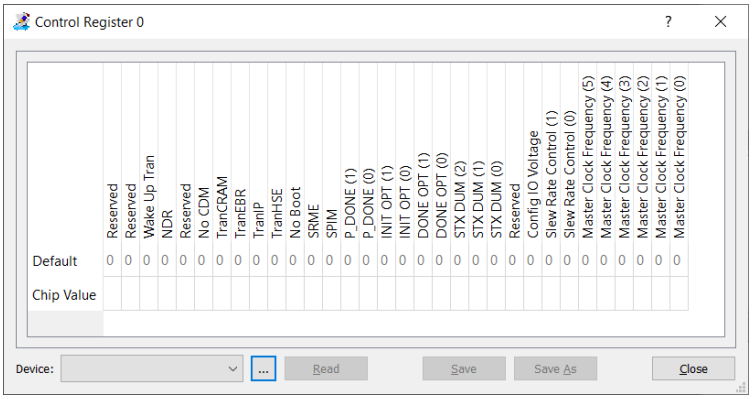


Figure 8.4. Lattice Programming File Utility Control Register GUI

The Radiant Deployment Tool, which is part of the Radiant Programmer interface, allows you to generate other type of files such as ISC, HEX, and BSDL files as shown in Figure 8.5.

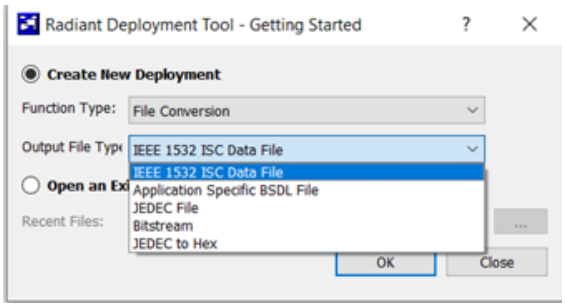


Figure 8.5. Lattice Radiant Deployment Tool GUI

References

For more information, refer to the following documents:

- CertusPro-NX Family Data Sheet (FPGA-DS-02086)
- Package Diagrams Data Sheet (FPGA-DS-02053)
- sysI/O User Guide for Nexus Platform (FPGA-TN-02067)
- CertusPro-NX High-Speed I/O Interface (FPGA-TN-02244)
- Sub-LVDS Signaling Using Lattice Devices (FPGA-TN-02028)
- sysCLOCK PLL Design and User Guide for Nexus Platform (FPGA-TN-02095)
- Lattice Radiant Software Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02165)
- Memory User Guide for Nexus Platform (FPGA-TN-02094)
- Lattice Radiant Software 3.1 User Guide
- Memory Modules User Guide (FPGA-IPUG-02033)
- sysDSP User Guide for Nexus Platform (FPGA-TN-02096)
- DSP Arithmetic Modules User Guide (FPGA-IPUG-02050)
- Arithmetic Modules User Guide (FPGA-IPUG-02032)
- Lattice Radiant Software 3.2 User Guide
- CertusPro-NX SerDes/PCS User Guide (FPGA-TN-02245)
- CertusPro-NX Hardware Checklist (FPGA-TN-02255)
- Lattice Memory Mapped Interface and Lattice Interrupt Interface User Guide (FPGA-UG-02039)
- Soft Error Detection (SED)/Correction (SEC) User Guide for Nexus Platform (FPGA-TN-02076)
- ADC User Guide for Nexus Platform (FPGA-TN-02129)
- Single Event Upset (SEU) Report for Nexus Platform (FPGA-TN-02174)
- Using TraceID Technical Note (FPGA-TN-02084)
- Advanced Configuration Security User Guide for Nexus Platform (FPGA-TN-02176)
- FPGA Libraries Reference Guide
- HDL Coding Guidelines
- Design Planning in Diamond
- Lattice Diamond Design Flow Overview for Xilinx Vivado Users User Guide (FPGA-UG-02169)
- Lattice Diamond 3.12 User Guide
- Lattice Synthesis Engine for Diamond User Guide

For more information, refer to the following Lattice device and training web pages:

- Avant-E web page
- Avant-G web page
- Avant-X web page
- Certus-NX web page
- CertusPro-NX web page
- CrossLink web page
- CrossLink-NX web page
- CrossLinkPlus web page
- LatticeECP2/M web page
- Lattice ECP3 web page
- ECP5/ECP5-5G web page
- LatticeXP2 web page
- iCE40 LP/HX web page
- iCE40 UltraPlus web page
- ispMACH 4000ZE web page
- ispMACH 4000V/Z web page
- MachXO web page
- MachXO2 web page

- MachXO3 web page
- MachXO3D web page
- MachXO5-NX web page
- Lattice Insights web page 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

Revision 1.0, March 2024

Section	Change Summary
All	Initial release.

www.latticesemi.com