CNN Plus Accelerator IP Core - Lattice Radiant Software

User Guide

FPGA-IPUG-02115-1.4

March 2022

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 associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. 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. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CNN	Convolutional Neural Network
DRAM	Dynamic Random Access Memory
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
LMMI	Lattice Memory Mapped Interface
OPN	Ordering Part Number

1. Introduction

The Lattice Semiconductor CNN Plus Accelerator IP Core is a calculation engine for Deep Neural Network with fixed point weight. It calculates full layers of Neural Network including convolution layer, pooling layer, batch normalization layer, and full connect layer by executing sequence code with weight value, which is generated by Lattice sensAI™ Neural Network Compiler. The engine is optimized for convolutional neural network, so it can be used for vision-based application such as classification or object detection and tracking. The IP Core does not require an extra processor; it can perform all required calculations by itself.

The design is implemented in Verilog HDL. It can be targeted to CrossLink™-NX, Certus™-NX and CertusPro™-NX FPGA devices, and implemented using the Lattice Radiant™ software Place and Route tool integrated with the Synplify Pro® synthesis tool.

1.1. Quick Facts

Table 1.1 presents a summary of the CNN Plus Accelerator IP Core.

Table 1.1. Quick Facts

IP Requirements	Supported FPGA Family	CrossLink-NX, Certus-NX, CertusPro-NX
Resource Utilization	Targeted Devices	LIFCL-40, LIFCL-17, LFD2NX-40, LFD2NX-17, LFCPNX-100
	Supported User Interface	AXI4, LMMI (Lattice Memory Mapped Interface) Native interfaces as described in Interface Descriptions section.
	Resources	See Table A.1.
Design Tool Support	Lattice Implementation	IP Core v1.0.x - Lattice Radiant Software 2.0 and 2.1 IP Core v1.1.x - Lattice Radiant Software 2.0 or later IP Core v1.2.x - Lattice Radiant Software 2.0 or later
	Synthesis	Lattice Synthesis Engine
		Synopsys® Synplify Pro for Lattice
	Simulation	For a list of supported simulators, see the Lattice Radiant software user guide.

1.2. Features

The key features of the CNN Plus Accelerator IP Core include:

- Selectable three implementation types:
  - Compact CNN
  - Optimized CNN
  - Extended CNN
- Selectable AXI4 or FIFO interface
- Support for convolution layer, max pooling layer, global average pooling layer, batch normalization layer, and full connect layer
- Configurable bit width of activation (16/8-bit)
- Configurable number of memory blocks for tradeoff between resource and performance
- Optimized for 3 × 3 2D convolution calculation
- Support for 2 × 2 max pooling
- Support for 2 × 2 max pooling/unpooling with max argument – Extended CNN only
- Support for global average pooling by full connect engine
- Configurable input byte mode: signed, unsigned and disable – only Compact CNN supports disable mode
- Partial DRAM access
- Configurable maximum burst length (32, 256) for AXI4 Interface

- Supports paired and quadruple convolution engines to improve performance – Optimized/Extended CNN only
- Configurable Line Buffer Size (512, 1024, 2048) – Optimized/Extended CNN only
- Supports general purpose output signal for controlling external logic through command code

2. Functional Descriptions

2.1. Overview

The CNN Plus Accelerator IP Core performs a series of calculations per command sequence that is generated by the Lattice Neural Network Compiler tool. Commands must be written at the DRAM address specified by the code_base_addr_i signal, which is accessible through AXI BUS. Input data may be read from the DRAM at a pre-defined address or directly written through the input data write port. After command code and input data are available, the CNN Plus Accelerator IP Core starts calculation at the rising edge of the start signal. During calculation, intermediate data and final result may be transferred to the DRAM or fed out through the result write port. All operations are fully programmable by command code.

The CNN Plus Accelerator IP Core offers three types of implementations: the compact CNN type, which is suitable for small FPGA devices due to its low utilization, the optimized CNN type, which can perform four convolution calculations in parallel, making it suitable for high-speed applications, and the extended CNN type, which offers the same features as optimized CNN plus additional support for max pooling/unpooling with max argument. The block diagram of each type of implementation is shown in the following figures: Figure 2.1 for compact CNN type, Figure 2.3 for optimized CNN type, and Figure 2.4 for extended CNN type. The sizes of the Activation Data Storage and the Conv. Scratch Storage are configurable by using the Memory Type and Scratch Pad Memory Size attributes respectively. Refer to Table 2.3 for the description of these attributes.

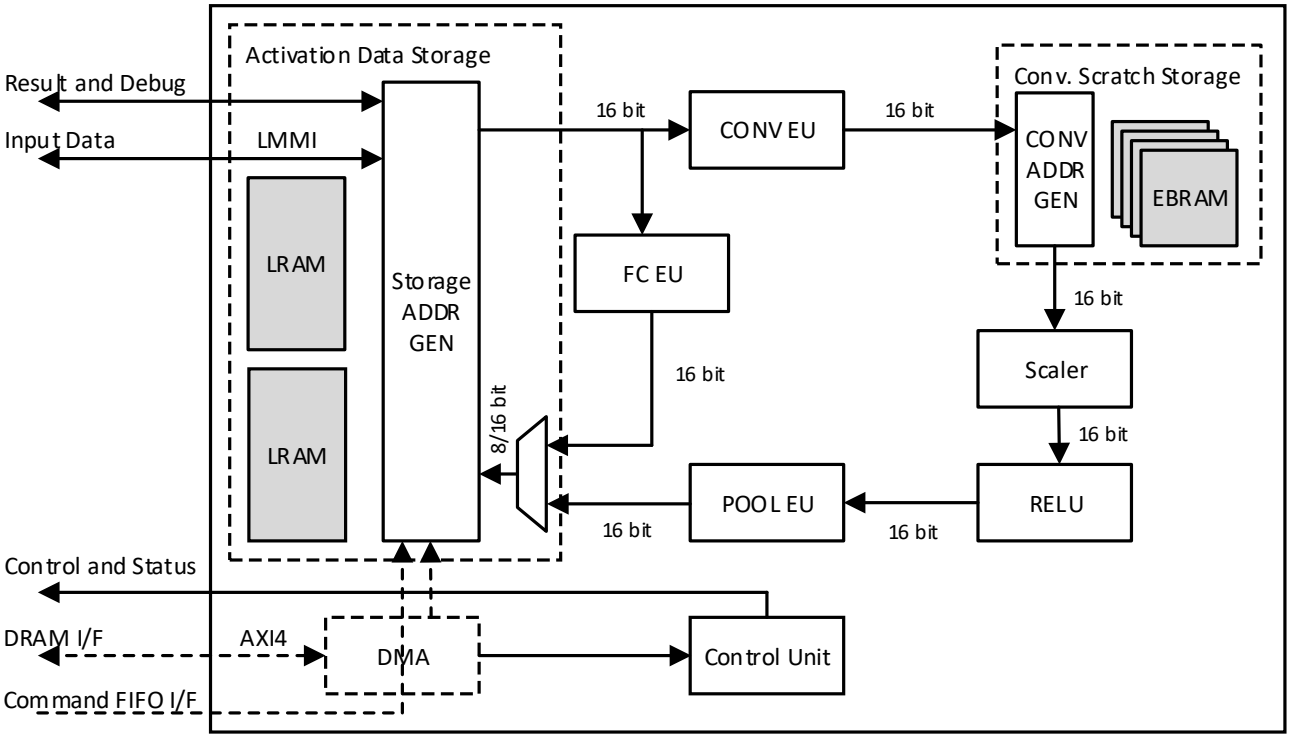

Figure 2.1. Functional Block Diagram of CNN Plus Accelerator (Compact CNN Type)

In Figure 2.1, the input data is stored in Activation Data Storage through LMMI. It can feed data to either the Convolution Engine or the Full Connect Engine. The result of Full Connect Engine goes back to Activation Data Storage. For the other path, the result of Convolution Engine is buffered to Convolution Scratch Storage and is transmitted to Scaler, which performs batch normalization. The Scaler can be bypassed by setting unity value, that is multiply by 1 and add by 0.

That result goes to RELU and then to POOL EU. Both RELU and POOL EU has bypass option. Lastly, the result of POOL EU goes back to Activation Data Storage.

The order of layers for CNN configuration is shown in Figure 2.2, which is similar for both Compact CNN and Optimized CNN implementation type. The only difference is that the Optimized CNN as shown in Figure 2.3 can perform four parallel calculations efficiently. Follow this when designing your network model. The layers inside the parenthesis have bypass option.

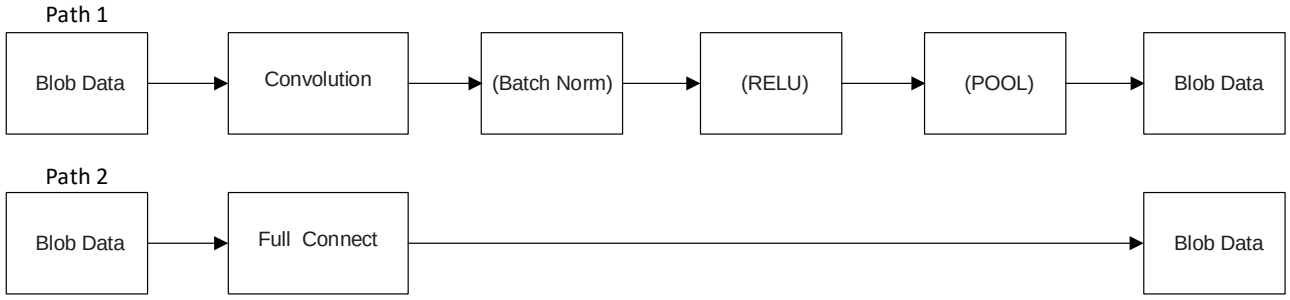

Figure 2.2. Order of Layers for CNN Plus Accelerator

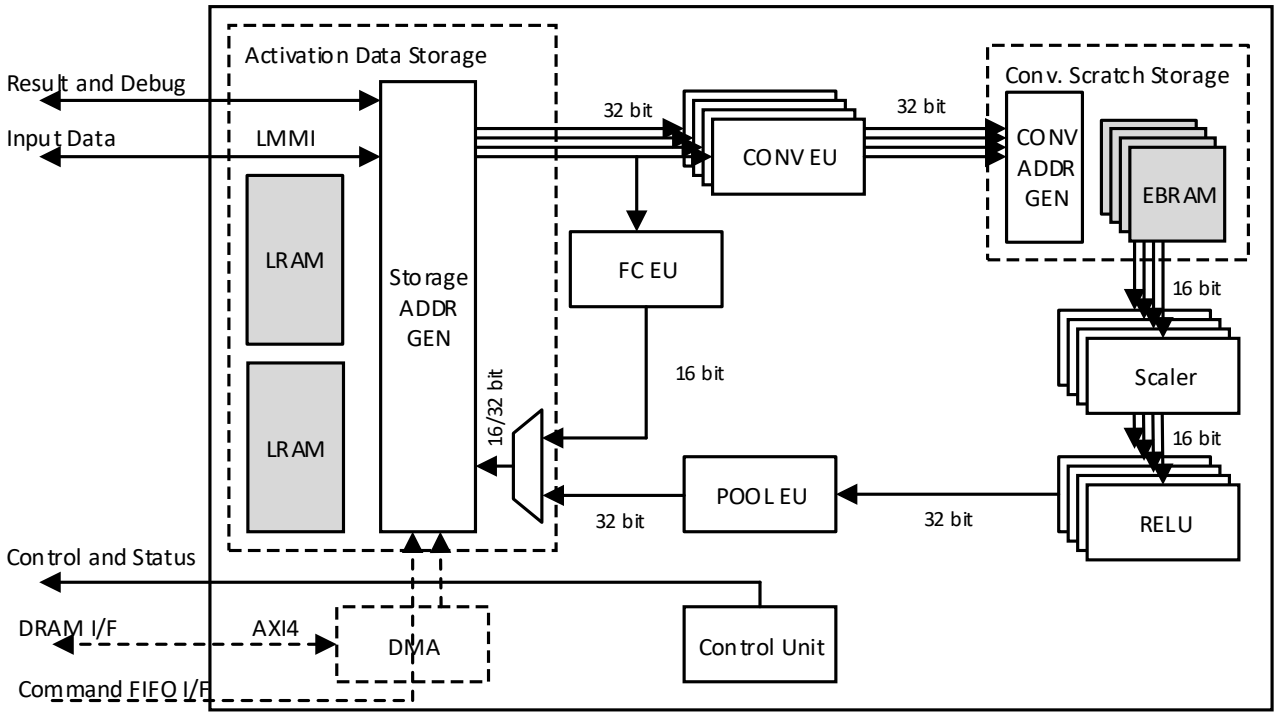

Figure 2.3. Functional Block Diagram of CNN Plus Accelerator (Optimized CNN Type)

In Figure 2.4, two additional computing functions are present in Extended CNN type. First, the max pooling/unpooling that processes the input data in reference to the input max argument. Second, the addition of RELU result and the input data. These new functions also provide additional order of layers as shown in Figure 2.5.

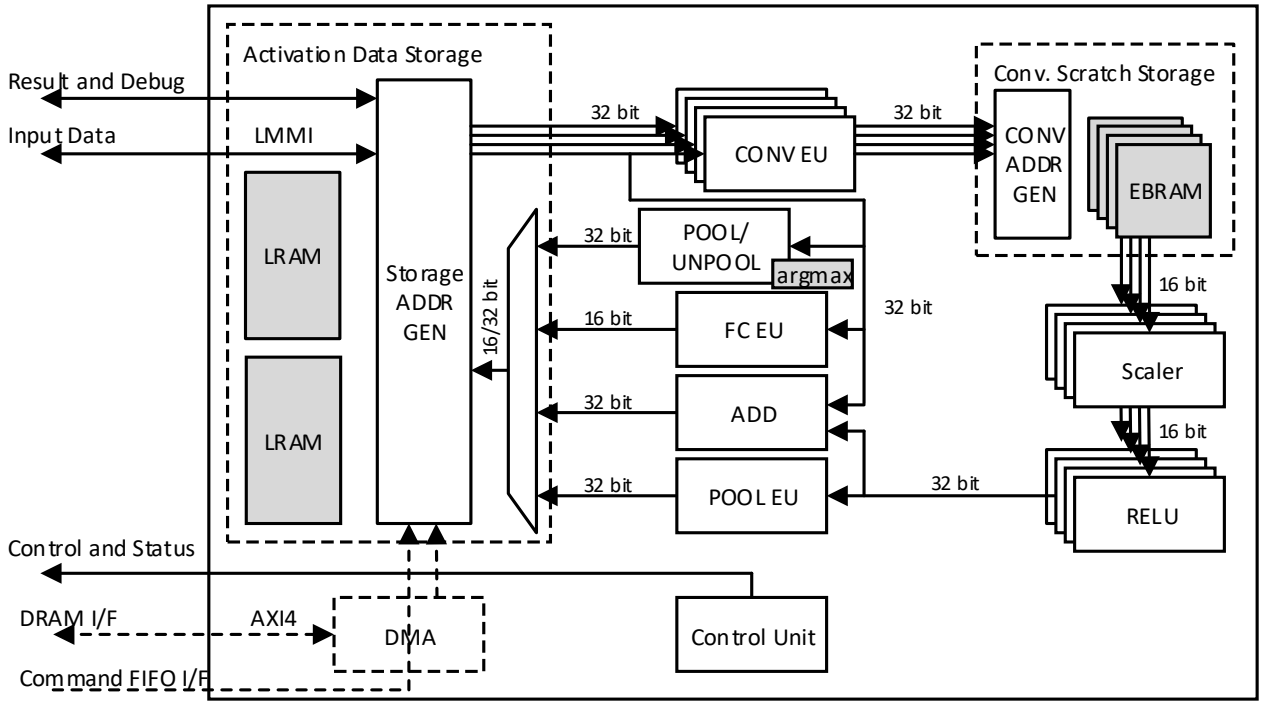

Figure 2.4. Functional Block Diagram of CNN Plus Accelerator (Extended CNN Type)

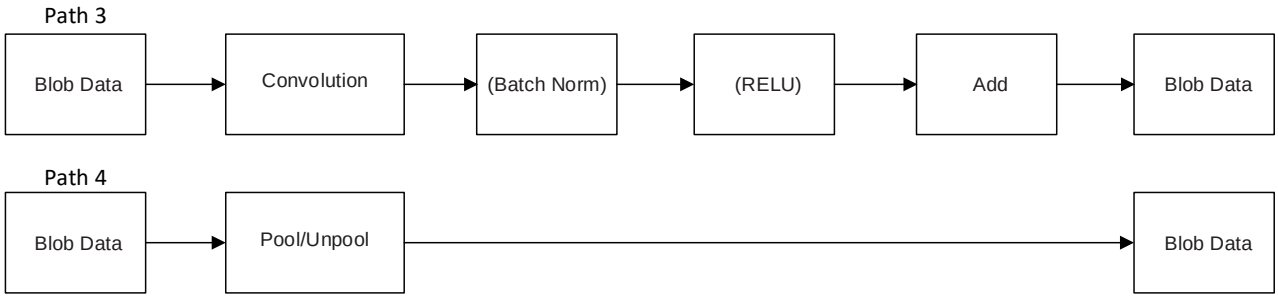

Figure 2.5. Additional Order of Layers for CNN Plus Accelerator (Extended CNN Type)

2.2. Interface Descriptions

Figure 2.6. shows the interface diagram for the CNN Plus Accelerator IP Core. The diagram shows all of the available ports for the IP core.

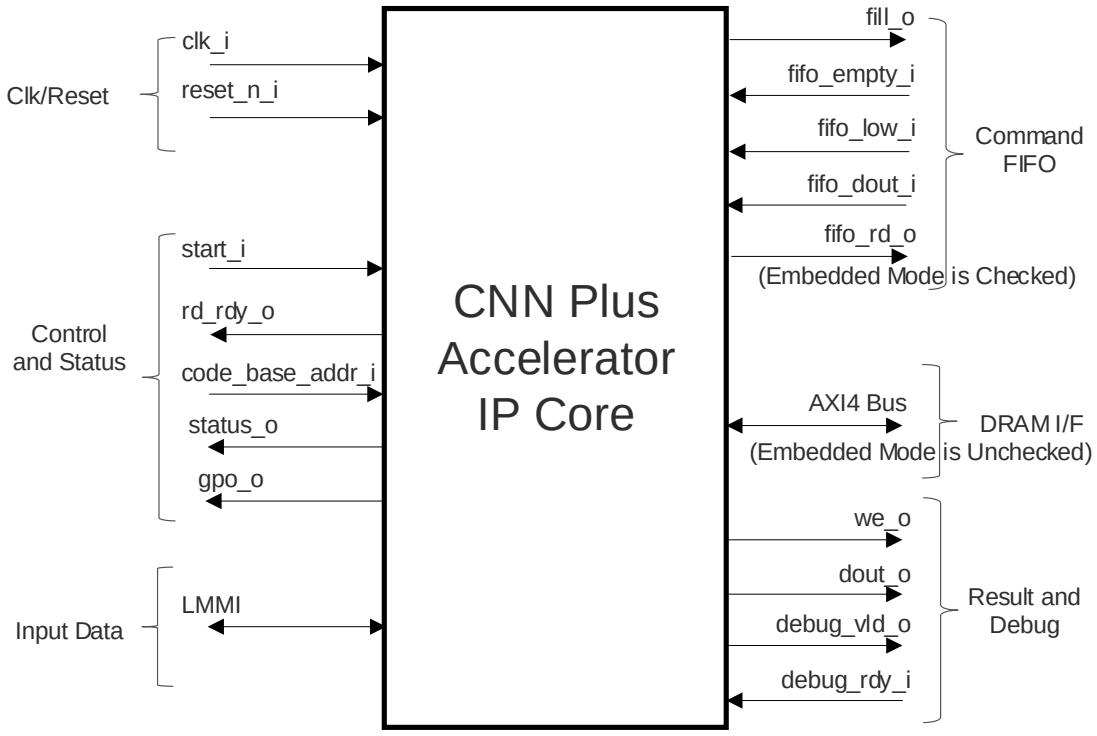

Figure 2.6. CNN Plus Accelerator IP Core Interface Diagram

Table 2.1. CNN Plus Accelerator IP Core Signal Descriptions

Pin Name	Direction	Function Description
Clock/Reset		
clk_i	Input	System clock Frequency can be chosen by trade-off between power and performance.
reset_n_i	Input	Active low system reset that is synchronous to clk signal and is asynchronous to aclk signal. [0] – Resets all ports and sets internal registers to their default values. [1] – Reset is NOT active
Control and Status		
code_base_addr_i[31:0]	Input	This signal specifies the base/start address that is read by CNN Plus Accelerator IP Core to get the command code. External logic should write the command code to this address. This signal must be set before start of operation.
start_i	Input	Start execution signal. Level sensitive. Must deassert after rd_rdy_o going 0.
rd_rdy_o	Output	Ready signal [0] – Engine is busy/running. [1] – Engine is idle and ready to get input. External logic should write input data to internal memory only during rd_rdy_o is high.
status_o	Output	Debug information [0] – If bit value is 1, engines are running except full connect engine. [1] – If bit value is 1, full connect engine is running. [2] – If bit value is 1, AXI bus transfer is ongoing. [3] – If bit value is 1, during engine is running.

Pin Name	Direction	Function Description
		[4] – If bit value is 1, command FIFO is reading. [5] – If bit value is 1, during DRAM access. [6] – If bit value is 1, DRAM command is issued. [7] – If bit value is 1, engine is waiting for FIFO fill.
gpo_o[31:0]	Output	General Purpose Output signal. Use for communication from firmware to outside block such as informing the type of current output, or informing the firmware version.
Input Data (LMMI)		
lmmi_request_i	Input	Start transaction
lmmi_wr_rdn_i	Input	Write = HIGH, Read = LOW
lmmi_offset_i	Input	Address offset for accessing the internal LRAMs in Activation Data Storage.
lmmi_wdata_i	Input	Write data
lmmi_ready_o	Output	Slave Ready signal. When lmmi_ready_o and lmmi_request_i are both HIGH, it indicates write transaction is completed and address phase of ready transition is completed.
lmmi_rdata_valid_o	Output	Indicates read transaction is complete and lmmi_rdata_o contains valid data.
lmmi_rdata_o	Output	Read data
Result and Debug		
we_o	Output	Write enable of result, indicates result data is valid [0] – Result data is NOT valid [1] – Result data is valid
dout_o[15:0]	Output	IF we_o is asserted, dout_o[15:0] contains result data. IF debug_vld_o is asserted, dout_o[15:0] contains debug data.
debug_rdy_i	Input	Ready signal for one burst read of debug data 0 – External logic is not ready to receive 1 burst of debug data. 1 – External logic is ready to receive 1 burst of debug data. The recommendation is to connect to 1 if debug feature is not used so that the operation is not halted when the IP Core sends debug data.
debug_vld_o	Output	Indicates that dout_o[15:0] is not result data, but debug data. 0 – Debug data is NOT valid. 1 – Debug data is valid
DRAM I/F (AXI4 Bus)[1]		
aclk	Input	AXI4 clock signal. Fully asynchronous from clk. Recommend to use DRAM system clock.
axi4_awid_o[7:0]	Output	AXI4 write address channel, write address ID signal *Constant output: 0x10*
axi4_awaddr_o[31:0]	Output	AXI4 write address channel, write address signal
axi4_awregion_o[3:0]	Output	AXI4 write address channel, region identifier signal *Constant output: 0x0 (default)*
axi4_awlen_o[7:0]	Output	AXI4 write address channel, burst length signal
axi4_awsize_o[2:0]	Output	AXI4 write address channel, burst size signal *Constant output: 3'b011 (8 bytes per beat)*
axi4_awburst_o[1:0]	Output	AXI4 write address channel, burst type signal *Constant output: 2'b01 (INCR)*
axi4_awlock_o	Output	AXI4 write address channel, lock type signal *Constant output: 1'b0 (Normal Access)*
axi4_awcache_o[3:0]	Output	AXI4 write address channel, memory type signal *Constant output: 0x00 (Device Non-bufferable)*
axi4_awprot_o[2:0]	Output	AXI4 write address channel, protection type signal *Constant output: 3'b010 (Data, Non-Secure Access, Unpriviledged)*
axi4_awqos_o[3:0]	Output	AXI4 write address channel, quality of service signal *Constant output: 0x0 (no QoS scheme)*
axi4_awvalid_o	Output	AXI4 write address channel, write address valid signal

Pin Name	Direction	Function Description
axi4_awready_i	Input	AXI4 write address channel, write address ready signal
axi4_wid_o[7:0]	Output	AXI4 write data channel, write ID tag signal *Constant output: 0x10*
axi4_wdata_o[63:0]	Output	AXI4 write data channel, write data signal
axi4_wstrb_o[7:0]	Output	AXI4 write data channel, write strobe signal *Constant output: 0xFF*
axi4_wlast_o	Output	AXI4 write data channel, write last signal
axi4_wvalid_o	Output	AXI4 write data channel, write valid signal
axi4_wready_i	Input	AXI4 write data channel, write ready signal
axi4_bid_i[7:0]	Input	AXI4 write response channel, response ID tag signal
axi4_bresp_i[1:0]	Input	AXI4 write response channel, write response signal
axi4_bvalid_i	Input	AXI4 write response channel, write response valid signal
axi4_bready_o	Output	AXI4 write response channel, response ready signal *Constant output: 1'b1 (*AXI4 write response channel is ignored*)*
axi4_arid_o[7:0]	Output	AXI4 read address channel, read address ID signal
axi4_araddr_o[31:0]	Output	AXI4 read address channel, read address signal
axi4_arregion_o[3:0]	Output	AXI4 read address channel, region identifier signal *Constant output: 0x0 (default)*
axi4_arlen_o[7:0]	Output	AXI4 read address channel, burst length signal
axi4_arsize_o[2:0]	Output	AXI4 read address channel, burst size signal
axi4_arburst_o[1:0]	Output	AXI4 read address channel, burst type signal
axi4_arlock_o	Output	AXI4 read address channel, lock type signal *Constant output: 1'b0 (Normal Access)*
axi4_arcache_o[3:0]	Output	AXI4 read address channel, memory type signal
axi4_arprot_o[2:0]	Output	AXI4 read address channel, protection type signal
axi4_arqos_o[3:0]	Output	AXI4 read address channel, quality of service signal *Constant output: 0x0 (no QoS scheme)*
axi4_arvalid_o	Output	AXI4 read address channel, read address valid signal
axi4_arready_i	Input	AXI4 read address channel, read address ready signal
axi4_rid_i[7:0]	Input	AXI4 Read data channel, read ID tag signal
axi4_rdata_i[63:0]	Input	AXI4 Read data channel, read data signal
axi4_rresp_i[1:0]	Input	AXI4 Read data channel, read response signal
axi4_rlast_i	Input	AXI4 Read data channel, read last signal
axi4_rvalid_i	Input	AXI4 Read data channel, read valid signal
axi4_rready_o	Output	AXI4 Read data channel, read ready signal
FIFO I/F[2]		
fill_o	Output	Request for filling the external command FIFO 0 – Request to flush and fill the external command FIFO External command fifo must be flushed and the command code generated by Lattice sensAI Neural Network Compiler software must be loaded to external command FIFO. 1 – Normal FIFO operation
fifo_empty_i	Input	Indicates whether external command FIFO is empty or not. 0 – External FIFO is not empty 1 – External FIFO is empty
fifo_low_i	Input	Indicates whether external command FIFO level is low or not. 0 – FIFO level is not low 1 – FIFO level is low If fifo level is not low, fifo must consecutively provide one full set of full connect weights, that is, by one 32-bit data per two cycles.

Pin Name	Direction	Function Description
fifo_dout_i[31:0]	Input	External command FIFO read data
fifo_rd_o	Output	Read request to external command FIFO 0 – No Read request 1 – Read request to external FIFO is active Value of i_fifo_dout is sampled when i_fifo_empty = 0 and o_fifo_rd = 1.

Notes:

1. DRAM I/F (AXI4 Bus) is available when *Embedded Mode* is unchecked.
2. FIFO I/F is available when *Embedded Mode* is checked.

2.2.1. Control and Status Interface

After reset or when engine is idle, rd_rdy_o is high. During this state, external logic may perform the following:

1. Write input data through Input Data interface.
2. Set the start address of command code to code_base_addr signal_i.

After the above steps, external logic must assert start_i signal. Engine starts execution when it gets start_i = 1 and rd_rdy_o goes 0. During execution, each bit of status_o indicates activity of sub calculation engine or AXI BUS. After finishing execution, that is by getting the finish command, the CNN Plus Accelerator IP Core asserts rd_rdy_o and waits for the next execution. Repeat from asserting start_i as shown in Figure 2.7.

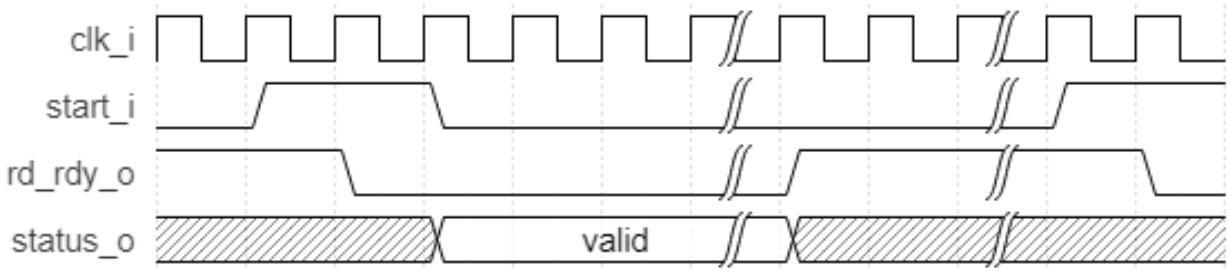

Figure 2.7. Control and Status Interface Timing Diagram

2.2.1.1. General Purpose Output

The general-purpose output signal, gpo_o is available since version v1.1.0 of the IP Core. This signal is controlled by the Lattice sensAI Neural Network Compiler software. One possible application of this signal is shown in Figure 2.8.In this example, different post processing operation needs to be performed on certain outputs. The gpo_o signal may be used to select which post process to perform.

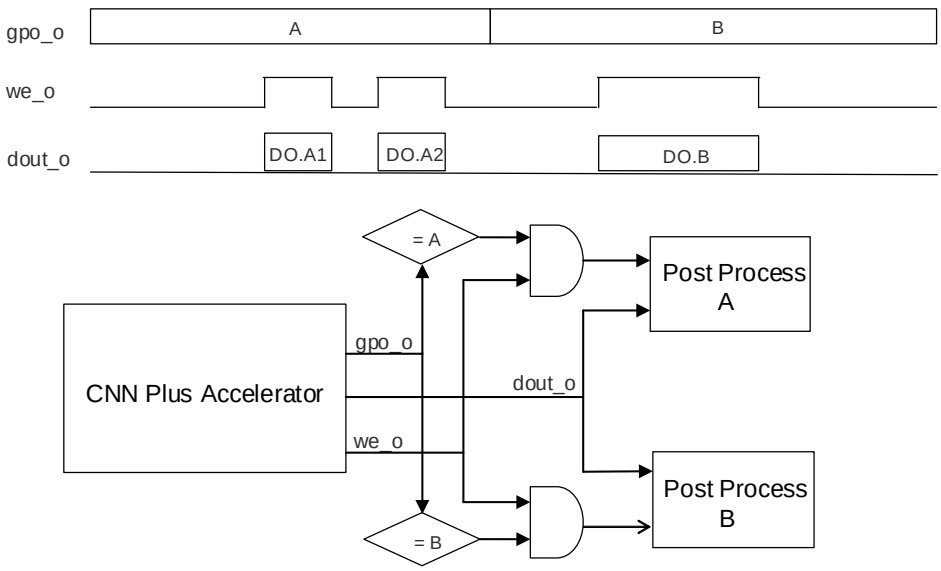

Figure 2.8. General Purpose Output Sample Application

2.2.2. Input Data Interface

Input data can be written to the DRAM by external logic. In this case, loading from the DRAM for input data must be in command codes. This is done by enabling the *Store Input* option and setting the target DRAM address in the Project Implementation window of the Lattice sensAI Neural Network Compiler software. In this case, the external logic needs to store the image data to target the DRAM address.

In addition, if input data is small enough to hold in internal memory, writing to the DRAM and reading back to the CNN Plus Accelerator IP Core may be a waste of cycle time and energy. In that case, the *Store Input* option should be disabled and the external logic should write input data to the internal memory of the CNN Plus Accelerator IP Core during idle state (lmmi_ready_o is high). The Input Data Interface is LMMI. Refer to Lattice Memory Mapped Interface and Lattice Interrupt Interface (FPGA-UG-02039) for more information on LMMI and its timing diagram. The read latency from lmmi_request_i to lmmi_ready_o is three clock cycles. Memory ID and memory address are mapped to the lmmi_offset_i as described in Table 2.1. They should be matched to command code. There is no required order or rule for writing the input data to the internal memory; any random access including memory ID is acceptable. Overwriting of same address is also accepted. However, you should ensure that the input data format is satisfied before asserting start_i signal. Refer to Input Data Format section for details.

2.2.3. Result and Debug Interface

The result, that is, final Blob data of neural network and the debug information are outputted to external logic through this Result and Debug Interface. When we_o signal is asserted, the dout_o signal contains the valid final Blob data. The external logic must have enough buffer to sample the result data whenever we_o asserts. This is usually is a single burst series of 16-bit data based on the command code. On the other hand, when debug_vld_o is asserted, the dout_o signal contains a vaid debug information. The IP Core keeps transmitting available debug information when debug_rdy_i is asserted. The IP Core halts until all debug information are transmitted. Thus, when debug information is not used, you should set debug_rdy_i to a fixed 1'b1 value so that it can flush-out debug information to proceed operation. The we_o and debug_vld_o signals never assert at the same time. Figure 2.9 shows an example timing diagram of Result and Debug interface with debug_rdy_i fixed to 1'b1.

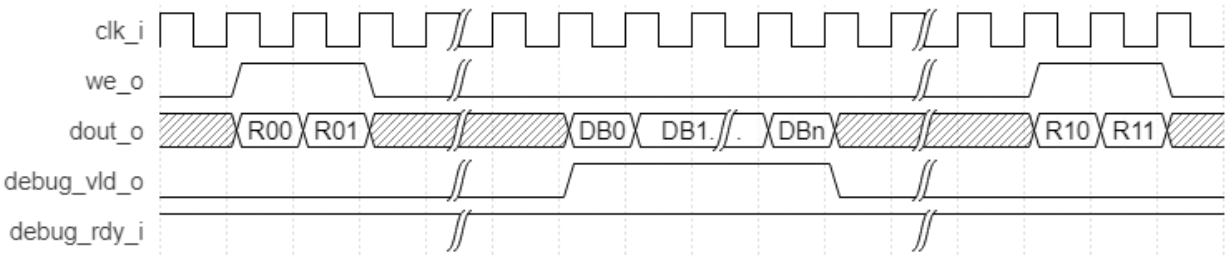

Figure 2.9. Result and Debug Interface Timing Diagram

2.2.4. DRAM Interface

Command code must be written in the DRAM before the execution of CNN Plus Accelerator IP Core. Input data may be written in the DRAM too. During the execution of CNN Plus Accelerator IP Core, it reads command code from the DRAM and performs calculation with internal sub-execution engine per command code. Intermediate data may be transferred from/to the DRAM per command code.

Refer to AXI4 Protocol Specification for the timing diagram of DRAM Interface.

2.2.5. Command FIFO Interface

External command FIFO must provide the command code that is generated by the Lattice sensAI Neural Network Compiler software. When the fill_o signal is at logic 0, the external logic should load the command code to external command FIFO. After the start_i control signal is asserted, the fill_o signal go to logic 1 and normal FIFO operation proceeds as shown in. When fifo_empty_i is low, the fifo_dout_i signal contains a valid command. The command is latched/sampled when fifo_empty_i is low and fifo_rd_o is high. The external command FIFO holds current command (value of fifo_dout_i) when fifo_empty_i and fifo_rd_o are both low.

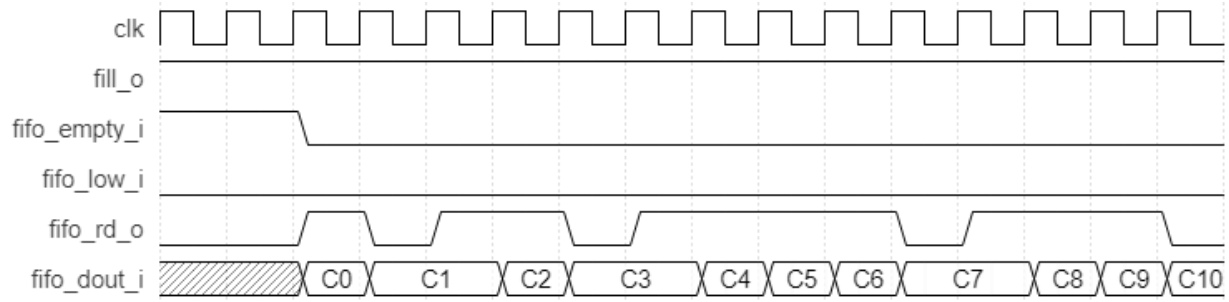

Figure 2.10. Command FIFO Interface Timing Diagram

2.3. Attribute Summary

The configurable attributes of the CNN Plus Accelerator IP Core are shown in Table 2.2 and are described in Table 2.3. The attributes can be configured through the IP Catalog's Module/IP Block wizard of the Lattice Radiant software. For the supported devices, the applicable attributes are listed in Table 2.5.

Table 2.2. Attributes Table

Attribute	Selectable Values	Default	Dependency on Other Attributes
Embedded Mode[1]	Checked, Unchecked	Unchecked	—
Machine Leaning Type	COMPACT_CNN, OPTIMIZED_CNN, EXTENDED_CNN	COMPACT_CNN	—
Convolution Engine[2]	SINGLE_CONV, DUAL_CONV, QUAD_CONV	SINGLE_CONV	Only for *Machine Leaning Type* = OPTIMIZED_CNN or EXTENDED_CNN
Memory Type	SINGLE_LRAM, DUAL_LRAM, QUAD_LRAM	DUAL_LRAM	—
LRAM Enable Output Register[2]	Checked, Unchecked	Unchecked	Only for *Machine Leaning Type* = COMPACT_CNN or OPTIMIZED_CNN
Scratch Pad Memory Size	1K, 4K, 8K, 16K	1K	v1.0.0: *Machine Leaning Type* = COMPACT_CNN
	1K, 2K, 4K, 8K		v1.0.0: *Machine Leaning Type* = OPTIMIZED_CNN v1.1.x: Same value for both *Machine Leaning Type* 8K is not supported for *Scratch Pad Memory Mode* = OCTA
Scratch Pad Memory Mode[2]	QUAD, OCTA	QUAD	Only for *Machine Leaning Type* = COMPACT_CNN
Line Buffer Size[1]	512, 1024, 2048	512	*Machine Leaning Type* = OPTIMIZED_CNN
Byte Mode	UNSIGNED, SIGNED, DISABLE	UNSIGNED	The DISABLE option is only available when *Machine Leaning Type* = COMPACT_CNN
Byte Shift	b000, b001, b010, b011, b100, b101, b110, b111	b101	v1.0.0: Fixed to b101 Only for *Machine Leaning Type* = COMPACT_CNN
Maximum Burst Length	32, 256	32	*Embedded Mode* is unchecked
Maximum Argument Size[2]	4096, 8192	4096	*Machine Leaning Type = EXTENDED_CNN*

Notes:

1. The attributes Embedded Mode and Line buffer Size are available from v1.1.0.
2. The attributes *Convolution Engine*, *LRAM Enable Output Register, Scratch Pad Memory Mode* and *Maximum Argument Size* are available from v1.2.0.

Table 2.3. Attributes Descriptions

Attribute	Description
Embedded Mode	Selects between AXI4 I/F (unchecked) and command FIFO I/F (checked). When command FIFO I/F is enabled, the external logic feeds the command codes to this interface.
Machine Leaning Type	This option allows you to choose between the following two implementation options: COMPACT_CNN – Low footprint, similar to Compact CNN Accelerator for thunderplus but enhanced with external DRAM access capability and supports larger network. OPTIMIZED_CNN – High performance – four parallel engines that efficiently calculate 3 × 3 convolution, depth-wise convolution and 1 × 1 convolution. EXTENDED_CNN – Same performance as OPTIMIZED_CNN plus support to max pooling/unpooling with max argument and adding of RELU result and input data.

Attribute	Description
Convolution Engine	Specifies the number of convolution engines. SINGLE_CONV – Use Single Convolution Engine DUAL_CONV – Use Paired Convolution Engine QUAD_CONV – Use Quadruple Convolution Engine This option is only for *Machine Leaning Type* == OPTIMIZED_CNN or EXTENDED_CNN.
Memory Type	Specifies the memory size of Activation Data Storage. SINGLE_LRAM – Total size is 64 kB. DUAL_LRAM – Total size is 128 kB. QUAD_LRAM – Total size is 256 kB. Set this attribute based on required size of peak Blob data in the Neural Network Compiler: • Use SINGLE_LRAM when network blob data ≤ 64 kB • Use DUAL_LRAM when network blob data > 64 kB and ≤ 128 kB • Use QUAD_LRAM when network blob data > 128 kB and ≤ 256 kB
LRAM Enable Output Register	Enables the use of output register for LRAM Unchecked – Disable the use of output register Checked – Enable the use of output register This option is only for *Machine Leaning Type* == COMPACT_CNN or OPTIMIZED_CNN.
Scratch Pad Memory Size	Configures the memory size of each EBRAM in Convolution Scratch Storage. 1K – Size of each EBRAM is 2 kB (1024 × 2 bytes). 2K – Size of each EBRAM is 4 kB (2048 × 2 bytes). 4K – Size of each EBRAM is 8 kB (4096 × 2 bytes). 8K – Size of each EBRAM is 16 kB (8192 × 2 bytes). The Lattice sensAI Neural Network Compiler software schedules data access to Convolution Scratch Storage based on the size of this memory and size of the input image. It is recommended to allocate more size to this memory when your design and target device allows it.
Scratch Pad Memory Mode	Specifies the total size of scratch pad QUAD – 2x of scratch pad OCTA – 4x of scratch pad This option is only for *Machine Leaning Type* == COMPACT_CNN.
Line Buffer Size	Selects the size of line buffer. 512 – Select this when the input image is ≤ QVGA (320×240). 1024 – Select this when the input image is VGA 2048 – Select this when the input image size is ≥ VGA
Byte Mode	Specifies the byte mode of input data. SIGNED – Input data is signed 16-bit/8-bit data. UNSIGNED – Input data is signed 16-bit data or unsigned 8-bit data. DISABLE – Input data is unsigned 16-bit data only. This option saves LUT because the byte mode support is not implemented. This option is only for *Machine Leaning Type* == COMPACT_CNN.
Byte Shift	Specifies the shift amount for converting byte data and 16-bit data. Conversion occurs when storing the result of convolution, pooling, and full connect layers into the Activation Data Storage as Byte Mode (signed or unsigned) and reading out the byte data to the 16-bit dout_o signal. Refer to Table 2.4. for the shifting of data byte based on Byte Shift and Byte Mode. This is fixed to default value (b101) for IP Core v1.0 and sensAI v3.0.
Maximum Burst Length	Specifies the maximum burst length of AXI4 bus. This should be set less than or equal to the maximum burst length that is supported by the connected slave device/memory.
Maximum Argument Size	Specifies the number of output data which can be processed by max pooling/unpooling without updating table. This option is only for *Machine Leaning Type* == EXTENDED_CNN.

Table 2.4. Shifting of Data Byte Based on Byte Shift and Byte Mode

Byte Shift	Byte Mode == SIGNED	Byte Mode == UNSIGNED
b000	{Byte[7:0], 8'b0}	{1'b0, Byte[6:0], 8'b0}
b001	{Byte[7], Byte[7:0], 7'b0}	{1'b0, Byte[7:0], 7'b0}
b010	{2{Byte[7]}, Byte[7:0], 6'b0}	{2'b0, Byte[7:0], 6'b0}
b011	{3{Byte[7]}, Byte[7:0], 5'b0}	{3'b0, Byte[7:0], 5'b0}
b100	{4{Byte[7]}, Byte[7:0], 4'b0}	{4'b0, Byte[7:0], 4'b0}
b101	{5{Byte[7]}, Byte[7:0], 3'b0}	{5'b0, Byte[7:0], 3'b0}
b110	{6{Byte[7]}, Byte[7:0], 2'b0}	{6'b0, Byte[7:0], 2'b0}
b111	{7{Byte[7]}, Byte[7:0], 1'b0}	{7'b0, Byte[7:0], 1'b0}

Note: Byte[7:0] is the result of convolution, pooling, and full connect layers. Byte[7:0] = 8'h00 when sign of 16-bit value is negative and Byte[7:0] = 8'hFF when saturated.

Table 2.5. Supported Devices and Attributes

Attribute	LIFCL-17/LFD2NX-17	LIFCL-40/LFD2NX-40	LFCPNX-100
Machine Leaning Type == COMPACT_CNN	O	O	O
Machine Leaning Type == COMPACT_CNN Scratch Pad Memory Size == 1K/2K/4K Scratch Pad Memory Mode == QUAD	O	O	O
Machine Leaning Type == COMPACT_CNN Scratch Pad Memory Size == 8K Scratch Pad Memory Mode == QUAD	X	O	O
Machine Leaning Type == COMPACT_CNN Scratch Pad Memory Size == 1K/2K Scratch Pad Memory Mode == OCTA	O	O	O
Machine Leaning Type == COMPACT_CNN Scratch Pad Memory Size == 4K Scratch Pad Memory Mode == OCTA	X	O	O
Machine Leaning Type == OPTIMIZED_CNN	O	O	O
Machine Leaning Type == OPTIMIZED_CNN Use Paired Convolution Engine == Checked	X	O	O
Machine Leaning Type == OPTIMIZED_CNN Use Quadrupled Convolution Engine == Checked	X	X	O
Machine Leaning Type == EXTENDED_CNN	O	O	O
Machine Leaning Type == EXTENDED_CNN Use Paired Convolution Engine == Checked	X	O	O
Machine Leaning Type == EXTENDED_CNN Use Quadrupled Convolution Engine == Checked	X	X	O
Memory Type == SINGLE_LRAM	O	O	O
Memory Type == DUAL_LRAM	O	O	O
Memory Type == QUAD_LRAM	O	X	O

2.4. Clock Domain

The clk_i and aclk_i domains for Compact CNN implementation type are shown in Figure 2.11. , the aclk_i domain is limited to the AXI4 interface. The difference in clock is absorbed by Lattice Dual Clocked FIFO IP (FIFO_DC) and is implemented in the DMA sub block. This is the same for the Optimized CNN implementation type.

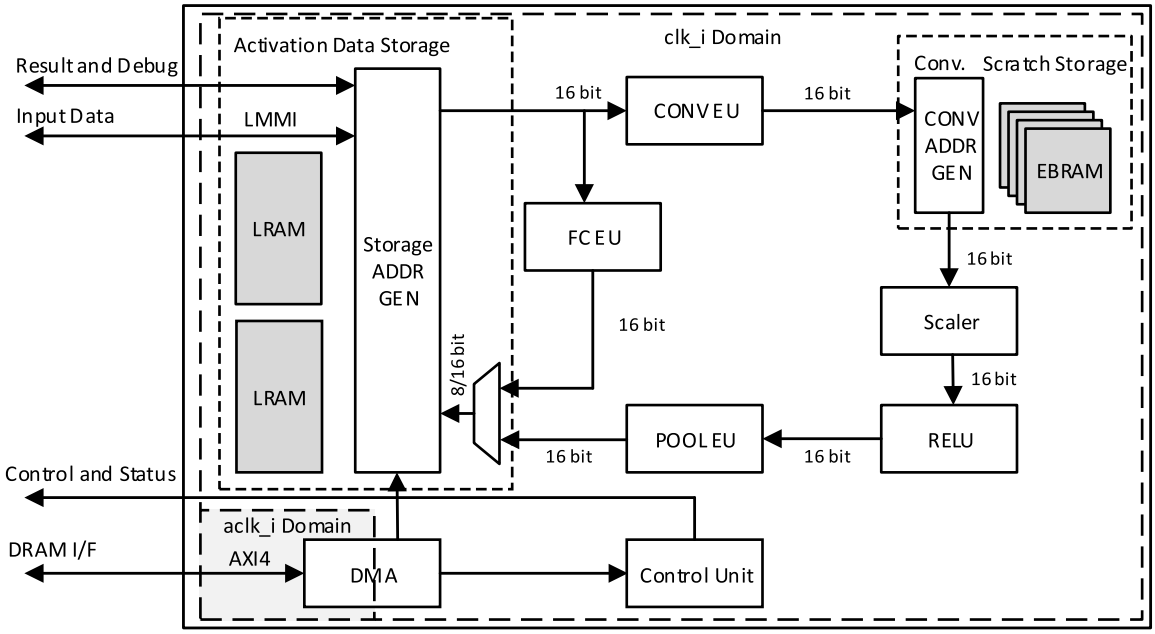

Figure 2.11. Clock Domain Diagram

2.5. Reset Behavior

When resetn signal asserts, output ports return to logic 0 in the next cycle. When resetn deasserts, output ready signals assert in the next cycle. A timing diagram of reset during AXI4 access is shown as an example in Figure 2.12. Only a few output signals are shown in this figure. The clk and aclk signals are 50% out-of-phase to show asynchronous relationship. The minimum resetn assert period is one cycle of the slower clock between clk_i and aclk_i.

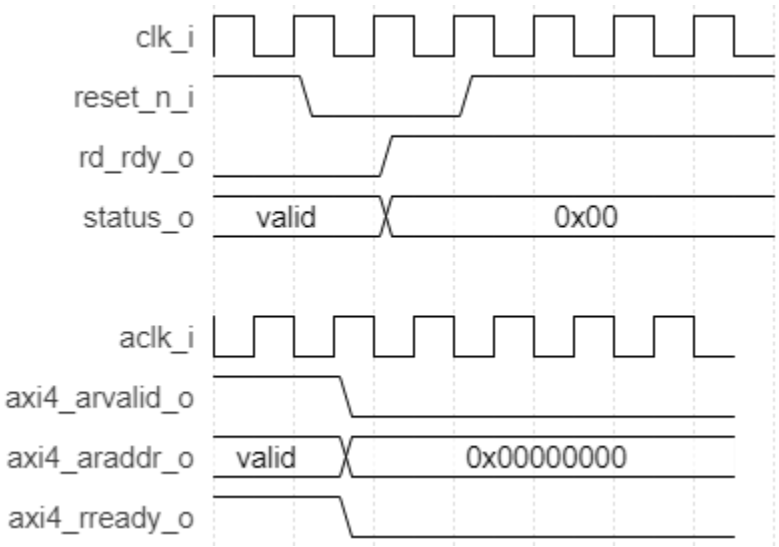

Figure 2.12. Reset Timing Diagram

Some AXI4 output signals are constant outputs; these are not affected by reset. Refer to Table 2.1 for the AXI4 output signals that are constant.

2.6. Register Description

CNN Plus Accelerator IP Core has no user-configurable register.

2.7. Operation Sequence

Operation sequence must be executed in the following steps. debug_rdy_i is tied to 1 in this example.

1. Assert Reset.
2. Deassert Reset. start_i must be deasserted.
3. Write command sequence code, which is generated by the Lattice sensAI Neural Network Compiler software into the DRAM starting at the address specified by code_base_addr_i signal.
4. Check that rd_rdy_o is high, if not, go back to step 1.
5. Write input data into the DRAM at the proper address, which is decided by command sequence, or directly write into the internal memory block of the CNN Plus Accelerator IP Core through the input data LMMI interface.
6. Assert start_i and check rd_rdy_o. The rd_rdy_o signal deasserts to 0 after asserting start_i.
7. Deassert start_i.
8. Check if we_o if code has direct output commands. Collect dout_o while we_o == 1.
9. Repeat from step 5.

2.7.1. Command Format

Command is a sequence of 32-bit data with or without additional parameters or weights as shown in Figure 2.13. . It should be loaded at DRAM address specified by code_base_addr_i signal before execution. The command is generated by the Lattice sensAI Neural Network Compiler software. For more information, refer to Lattice SensAI Neural Network Compiler Software User Guide (FPGA-UG-02052).

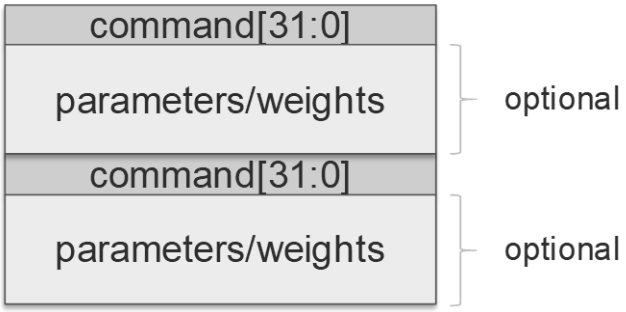

Figure 2.13. Command Format

2.7.2. Input Data Format

Input data is a sequence of 8-bit or 16-bit data. Memory index and address are decided by Neural Network. Therefore, the external block should process input raw data and write input data to the Lattice CNN Plus Accelerator IP Core through the input data write interface. The input data interface of this IP core is 16-bit wide. If input data is 8-bit (Byte Mode is SIGNED/UNSIGNED), use only the lower 8-bit of the input data interface and set the upper 8-bit to 8'h00.

For example, face detection neural network may take 32 × 32 of R, G, B planes at memory index 0 with address 0x0000 for Red plane, 0x0400 for Green plane and 0x0800 for Blue plane. Another example is object detection neural network may take 90 × 90 of R, G, B planes, which are assigned to memory index 0, 1, and 2 respectively. Because memory assignment is defined by neural network, external block should handle input raw data and write it to proper position of internal memory of the CNN Plus Accelerator IP Core.

Writing input data to DRAM and using Load command to fetch input data are also possible in the case of large input data. The IP core expects data in little-endian order.

The input data must agree with the Byte Mode attribute settings. Refer to Table 2.3 for details.

2.7.3. Output Data Format

Output data is a sequence of 16-bit data which is controlled by commands. The amount of data is also decided by Neural Network, that is, by output blobs. External block should interpret output sequence and generate usable information. For example, face detection outputs 2-beat burst (two consecutive) of 16-bit data, the first is confidence of non-face while the second one is confidence of face. Whenever the latter is larger than the former, conclusion is Face. The IP core outputs data in little-endian order.

2.8. Supported Commands

Command sequences are generated by Lattice SensAI Neural Network Compiler Software (FPGA-UG-02052).

3. Core Generation Simulation and Validation

This section provides information on how to generate the CNN Plus Accelerator IP Core using the Lattice Radiant software and how to run simulation and synthesis. For more details on the Lattice Radiant software, refer to the Lattice Radiant Software 2.0 User Guide.

3.1. Licensing the IP

An IP core-specific device-specific license string is required to enable full, unrestricted use of the Lattice CNN Plus Accelerator IP Core in a complete, top-level design. You may refer to the instructions on how to obtain licenses for Lattice IP cores at https://www.latticesemi.com/Support/Licensing.

You may download and generate the CNN Plus Accelerator IP Core and fully evaluate the core through functional simulation and implementation (synthesis, map, place, and route) without an IP license string. The CNN Plus Accelerator IP Core supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core which operate in hardware for a limited time (approximately four hours) without requiring an IP license string. See Hardware Evaluation section for further details. However, a license string is required to generate bitstream file that does not include the hardware evaluation timeout limitation.

Note: All IP has a license whether in eval mode or full mode. Difference is license string.

3.2. Generation and Synthesis

The Lattice Radiant software allows you to customize and generate modules and IPs and integrate them into the device's architecture. The procedure for generating the CNN Plus Accelerator IP Core in Lattice Radiant software is described below.

To generate the CNN Plus Accelerator IP Core:

1. Create a new Lattice Radiant software project or open an existing project.
2. In the **IP Catalog** tab, double-click on **CNN_Plus_Accelerator** under **IP, DSP** category. The **Module/IP Block Wizard** opens as shown in Figure 3.1. Enter values in the **Component name** and the **Create in** fields and click **Next**.

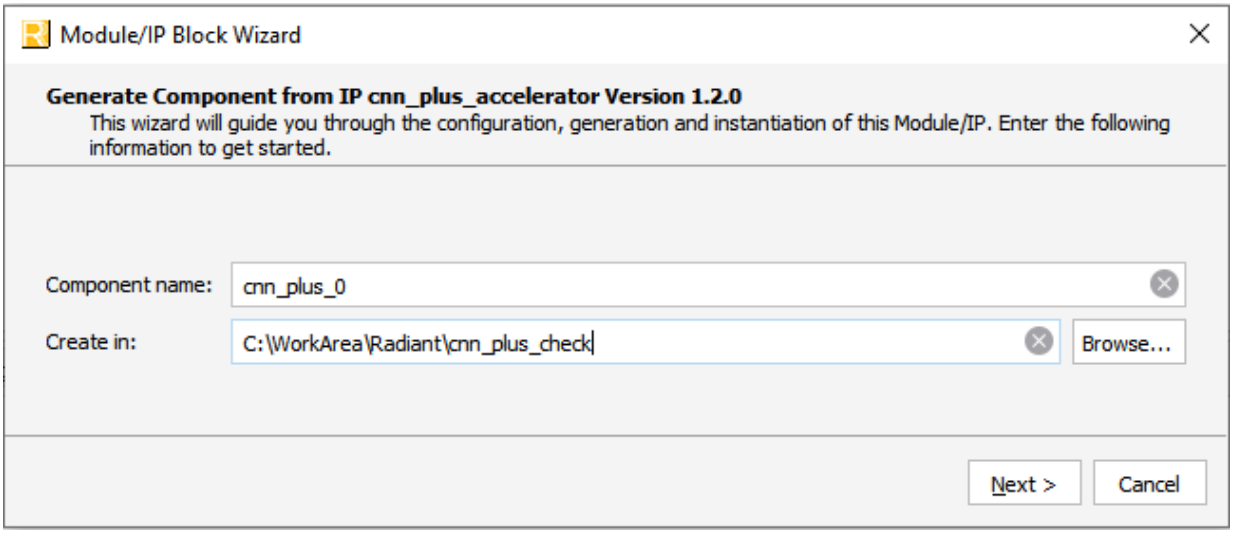

Figure 3.1. Module/IP Block Wizard

3. In the module's dialog box of the **Module/IP Block Wizard** window, customize the selected CNN Plus Accelerator IP Core using drop-down menus and check boxes. As a sample configuration, see Figure 3.2. For configuration options, see the Attribute Summary section.

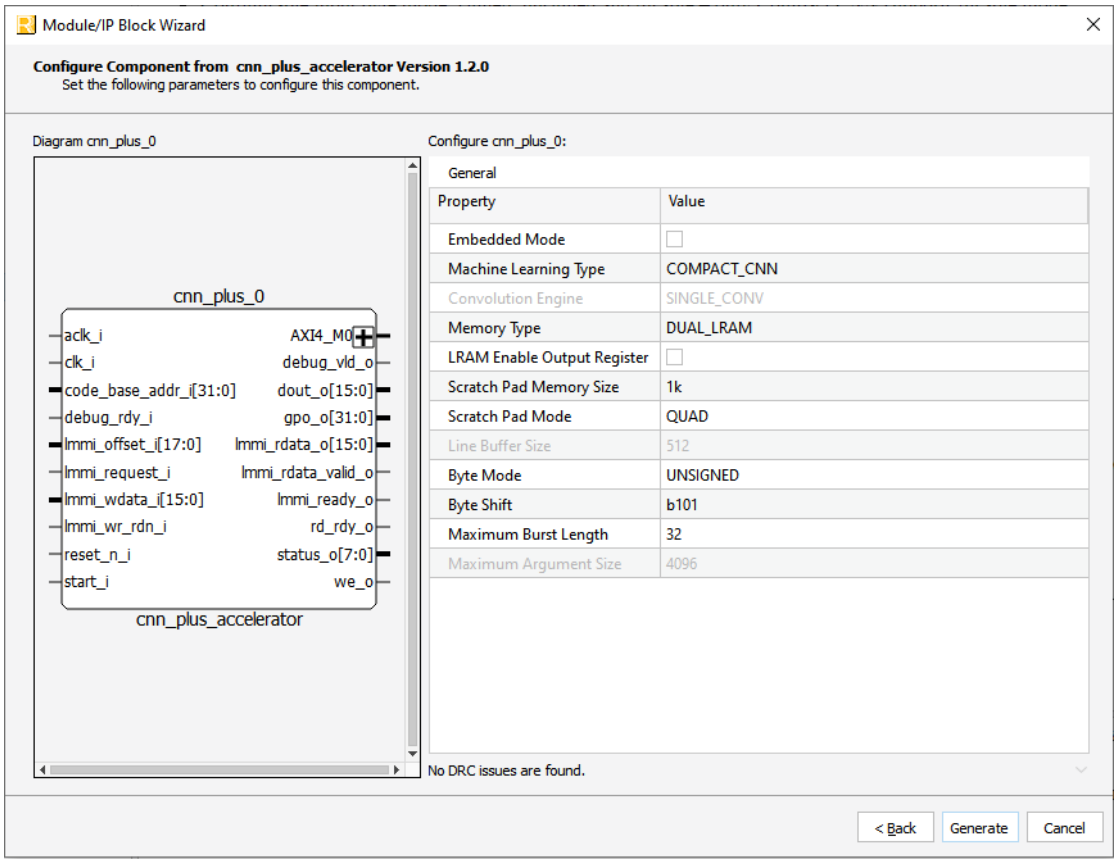

Figure 3.2. Configure User Interface of CNN Plus Accelerator IP Core

4. Click **Generate**. The **Check Generating Result** dialog box opens, showing design block messages and results as shown in Figure 3.3.

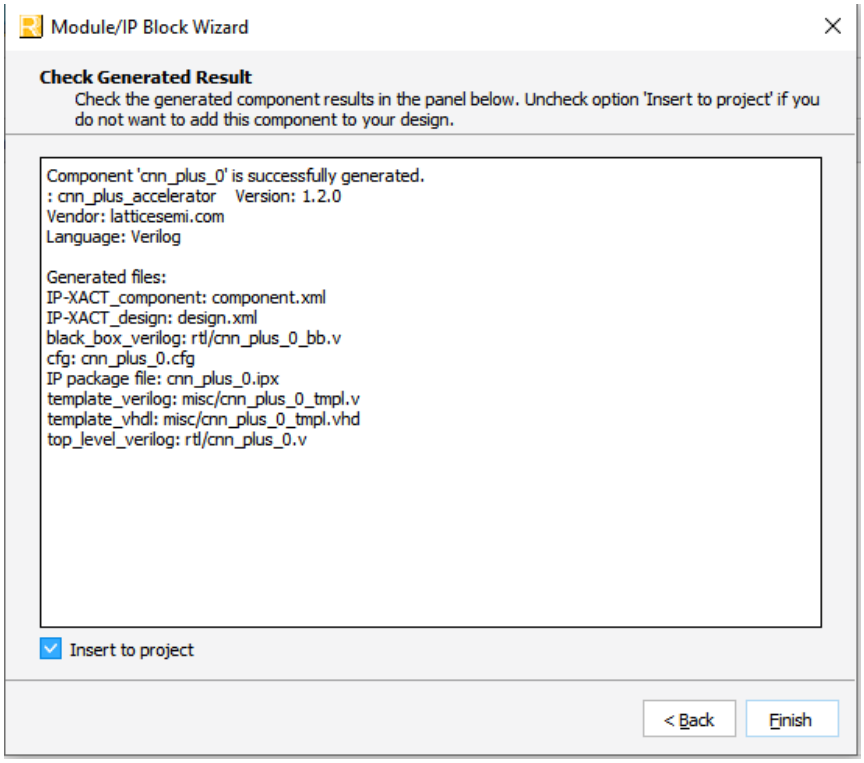

Figure 3.3. Check Generating Result

5. Click the **Finish** button. All the generated files are placed under the directory paths in the **Create in** and the **Component name** fields shown in Figure 3.1.

The generated CNN Plus Accelerator IP Core package includes the black box (<Component name>_bb.v) and instance templates (<Component name>_tmpl.v/vhd) that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file (<Component name>.v) that can be used as an instantiation template for the IP core is also provided. You may also use this top-level reference as the starting template for the top-level for their complete design. The generated files are listed in Table 3.1.

Table 3.1. Generated File List

Attribute	Description
<Component name>.ipx	Contains the information on the files associated to the generated IP.
<Component name>.cfg	Contains the parameter values used in IP configuration.
component.xml	Contains the ipxact:component information of the IP.
design.xml	Documents the configuration parameters of the IP in IP-XACT 2014 format.
rtl/<Component name>.v	Provides an example RTL top file that instantiates the IP core.
rtl/<Component name>_bb.v	Provides the synthesis black box.
misc/<Component name>_tmpl.v misc /<Component name>_tmpl.vhd	Provide instance templates for the IP core.

3.3. Running Functional Simulation

The CNN Plus Accelerator IP does not have a functional simulation testbench.

3.4. Hardware Evaluation

The CNN Plus Accelerator IP Core supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs. Choose Project > Active Strategy > Translate Design Settings. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

4. Ordering Part Number

The Ordering Part Numbers (OPN) for CNN Plus Accelerator IP Core are the following:

- CNNPLUS-ACCEL-CNX-U – CNN Plus Accelerator for CrossLink-NX - Single Design License
- CNNPLUS-ACCEL-CNX-UT – CNN Plus Accelerator for CrossLink-NX - Multi-Site License
- CNNPLUS-ACCEL-CTNX-U – CNN Plus Accelerator for Certus-NX - Single Design License
- CNNPLUS-ACCEL-CTNX-UT – CNN Plus Accelerator for Certus-NX - Multi-Site License
- CNNPLUS-ACCEL-CPNX-U – CNN Plus Accelerator for CertusPro-NX - Single Design License
- CNNPLUS-ACCEL-CPNX-UT – CNN Plus Accelerator for CertusPro-NX - Multi-Site License

References

- CrossLink-NX FPGA Web Page in latticesemi.com
- Certus-NX FPGA Web Page in latticesemi.com
- CertusPro-NX FPGA Web Page in latticesemi.com

Technical Support Assistance

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

Appendix A. Resource Utilization

Table A.1 shows configuration and resource utilization for LFCPNX-100-9BBG484I using Lattice Synthesis Engine of Lattice Radiant software.

Table A.1. Performance and Resource Utilization[1]

Configuration[3]	clk_i, aclk_i Fmax (MHz)[2]	Registers	LUTs	LRAMs	EBRs[4]	Logical DSP	
						MULT9, MULT18	REG18, PREADD9
Default	121.448, 127.081	2566	3626	2	13	13, 1	13, 13
Scratch Pad Memory Size=2K, Others=Default	119.104, 125.078	2578	3639	2	15	13, 1	13, 13
Scratch Pad Memory Size=4K, Others=Default	116.469, 124.270	2582	3650	2	19	13, 1	13, 13
Scratch Pad Memory Size=8K, Others=Default	122.160, 130.022	2590	3657	2	27	13, 1	13, 13
Scratch Pad Memory Mode=OCTA, Others=Default	120.685, 127.275	2710	3844	2	15	13, 1	13, 13
Scratch Pad Memory Mode=OCTA, *Scratch Pad Memory Size*=2K, Others=Default	123.183, 129.149	2714	3840	2	19	13, 1	13, 13
Scratch Pad Memory Mode=OCTA, *Scratch Pad Memory Size*=4K, Others=Default	122.085, 123.047	2722	3858	2	27	13, 1	13, 13
Memory Type=SINGLE_LRAM, Others=Default	130.005, 130.463	2565	3608	1	13	13, 1	13, 13
Memory Type=QUAD_LRAM, Others=Default	116.564, 122.011	2573	3677	4	13	13, 1	13, 13
Machine Leaning Type=OPTIMIZED_CNN, Others=Default	113.869, 129.601	5470	7226	2	17	48, 4	48, 48
Machine Leaning Type=OPTIMIZED_CNN, *Scratch Pad Memory Size*=2K, Others=Default	112.803, 123.686	5475	7240	2	21	48, 4	48, 48
Machine Leaning Type=OPTIMIZED_CNN, *Scratch Pad Memory Size*=4K, Others=Default	114.863, 124.409	5486	7265	2	29	48, 4	48, 48
Machine Leaning Type=OPTIMIZED_CNN, *Scratch Pad Memory Size*=8K, Others=Default	113.353, 122.234	5490	7279	2	45	48, 4	48, 48
Machine Leaning Type=OPTIMIZED_CNN, *Line Buffer Size*=1024, Others=Default	115.473, 126.662	5475	7250	2	21	48, 4	48, 48
Machine Leaning Type=OPTIMIZED_CNN, *Line Buffer Size*=2048, Others=Default	118.161, 122.609	5477	7266	2	30	48, 4	
Machine Leaning Type=OPTIMIZED_CNN, *Scratch Pad Memory Size*=8K, *Maximum Burst Length*=256, Others=Default	119.446, 126.263	5491	7279	2	45	48, 4	48, 48
Machine Leaning Type=OPTIMIZED_CNN *Convolution Engine*=DUAL_CONV, Others=Default	113.225, 124.657	7394	9585	2	21	84, 4	84, 84
Machine Leaning Type=OPTIMIZED_CNN *Convolution Engine*=QUAD_CONV, Others=Default	117.233, 124.008	11023	14216	2	29	156, 4	156, 156

Machine Leaning Type=OPTIMIZED_CNN LRAM Enable Output Register=Checked, Others=Default	147.842, 131.027	5475	7252	2	17	48, 4	48, 48
Embedded Mode=Checked, Others=Default	127.259, N/A	1737	2474	2	5	13, 1	13, 13
Embedded Mode=Checked, *Machine Leaning Type*=OPTIMIZED_CNN, Others=Default	120.715, N/A	4673	6109	2	9	48, 4	48, 48
Embedded Mode=Checked, *Line Buffer Size*=1024, *Machine Leaning Type*=OPTIMIZED_CNN, Others=Default	114.116, N/A	4677	6112	2	13	48, 4	48, 48
Embedded Mode=Checked, *Machine Leaning Type*=OPTIMIZED_CNN, *Convolution Engine*=DUAL_CONV, Others=Default	118.991, N/A	6596	8463	2	13	84, 4	84, 84
Machine Leaning Type=EXTENDED_CNN, Others=Default	134.120, 124.750	5798	8049	2	20	48, 4	48, 48
Machine Leaning Type=EXTENDED_CNN, *Scratch Pad Memory Size*=2K, Others=Default	126.406, 130.497	5803	8051	2	24	48, 4	48, 48
Machine Leaning Type=EXTENDED_CNN, *Scratch Pad Memory Size*=4K, Others=Default	150.466, 114.403	5811	8059	2	32	48, 4	48, 48
Machine Leaning Type=EXTENDED_CNN, *Scratch Pad Memory Size*=8K, Others=Default	132.732, 127.926	5815	8041	2	48	48, 4	48, 48
Machine Leaning Type=EXTENDED_CNN, Maximum Argument Size=8192, Others=Default	143.947, 128.287	5798	8051	2	22	48, 4	48, 48
Machine Leaning Type=EXTENDED_CNN, *Convolution Engine*=DUAL_CONV, Others=Default	144.071, 127.665	7722	10404	2	24	84, 4	84, 84
Machine Leaning Type=EXTENDED_CNN, *Convolution Engine*=QUAD_CONV, Others=Default	146.585, 135.208	11352	15002	2	32	156, 4	156, 156
Embedded Mode=Checked, *Machine Leaning Type*=EXTENDED _CNN, Others=Default	134.88, N/A	5000	6929	2	12	48, 4	48, 48

Notes:

1. Performance may vary when using a different software version or targeting a different device density or speed grade.
2. Fmax is generated when the FPGA design only contains the CNN Plus Accelerator IP Core. These values may be reduced when user logic is added to the FPGA design.
3. The K value in “Scratch Pad Memory Size=*K” is equivalent to 1024 entries × 2 bytes. For example, 4K is equal to 8 kB of scratch pad memory.
4. The OPTIMIZED_CNN implementation has a lot more EBRs because it duplicates the EBRs in Convolution scratch storage to enable parallel processing. In addition, some duplicated submodules have their own EBRs: CONV_EU (1 EBR per unit) and POOL (1 EBR shared by 2 units).

Revision History

Revision 1.4, March 2022

Section	Change Summary
Introduction	Removed “Dynamic support for various 1D convolution from 1 to 9 taps” from the list of features.

Revision 1.3, October 2021

Section	Change Summary
All	Minor adjustments in formatting across the document.
Acronyms in This Document	Added this section.
Introduction	• Added CertusPro-NX as supported FPGA Family and LFCPNX-100 as target device in Table 1.1. • Added new features in Features section.
Functional Description	• Added extended CNN type in Overview. • Updated Figure 2.1 and Figure 2.3. • Added Figure 2.4 and Figure 2.5 for extended CNN type block diagram and order of layers. • Updated Table 2.2 and Table 2.3 for the new attributes. • Added Table 2.5 for supported devices and attributes.
Core Generation Simulation and Validation	Updated Figure 3.1, Figure 3.2, and Figure 3.3.
Ordering Part Number	Updated content.
References	Added reference to the CertusPro-NX web page.
Resource Utilization	Added new rows in **Table A.1** to update the resource utilization result and changed target device to LFCPNX-100-9BBG484I.

Revision 1.2, May 2021

Section	Change Summary
Functional Description	Updated Input Data Format section.
Resource Utilization	Updated resource utilization result.

Revision 1.1, December 2020

Section	Change Summary
Introduction	• Added Certus-NX as supported FPGA Family, LFD2NX-40 and LFD2NX-17 as target devices in Quick Facts section. • Added new features in Features section.
Functional Description	• Added gpo_o and Command FIFO interface in Figure 2.1, Figure 2.4, and Table 2.1. • Added new sections: General Purpose Output and Command FIFO Interface. • Updated Table 2.2 and Table 2.3 for the new attributes.
Core Generation Simulation and Validation	Updated Figure 3.1, Figure 3.2, and Figure 3.3.
Ordering Part Number	Updated content.
References	Added Certus-NX Web page in latticesemi.com.
Resource Utilization	Updated resource utilization result.

Revision 1.0, May 2020

Section	Change Summary
All	Initial release

www.latticesemi.com