Compact CNN Accelerator IP Core

User Guide

FPGA-IPUG-02038-1.7

December 2020

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 and with all faults, and all risk associated with such information is entirely with 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
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
OPN	Ordering Part Number
SRAM	Static Random Access Memory

1. Introduction

The Lattice Semiconductor Compact CNN Accelerator IP Core is a calculation engine for Deep Neural Networks with fixed-point weight or binarized weight. It calculates many layers of neural networks including convolution, pooling, batch normalization, and full connect by executing sequence code with weight values, which is generated by the Lattice Neural Network Compiler tool. The engine is optimized for convolutional neural networks, which is suitable for analysis of image data and some applications where non-image data can be represented visually. Typical applications include image classification, face detection, and key phrase detection. The engine can operate standalone without the addition of a separate processor.

The design is implemented in Verilog HDL. It can be targeted to iCE40 UltraPlus™ 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 Compact CNN Accelerator IP Core.

Table 1.1. Quick Facts

IP Requirements	FPGA Families Supported	iCE40 UltraPlus
Resource Utilization	Targeted Device	iCE40 UP5K
	Supported User Interface	Native interfaces, refer to Interface Descriptions section.
	Resources	See Table A.1 and Table A.2.
Design Tool Support	Lattice Implementation	IP Core v1.0.x - Lattice Radiant software 1.0 IP Core v1.1.x - Lattice Radiant software 1.1 IP Core v2.0.0 - Lattice Radiant software 1.1 IP Core v2.1.0 - 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 Compact CNN Accelerator IP Core include:

- Support for binarized convolution layer, max/or pooling layer, binarizer and binarized full connect layer
- Configurable Blob memory type and size
- Optimized for 3 x 3 2D convolution calculation
- Dynamic support for various 1D convolution from 1 to 9 taps
- Support for fixed point convolution layer, max pooling layer and scaling layer
- Support for RELU and leaky RELU (fixed negative slope)
- Configurable size of scratch pad of convolution layer (1K, 2K, 4K entries)
- Configurable input byte mode (signed, unsigned, disable)
- Configurable number of 1 x 1 convolution engines in parallel (single, dual, quad)
- Supports general purpose output signal for controlling external logic through command code

Note: For 2D convolution calculation, only 1 x 1 and 3 x 3 are currently supported.

2. Functional Description

2.1. Overview

The Compact CNN Accelerator IP Core performs a series of calculations per a command sequence that is generated by the Lattice Neural Network Compiler tool. Commands must be fed through the command FIFO interface. Input data is directly written through input data write port. After command code and input data are available, the Compact CNN Accelerator IP Core starts calculation at the rising edge of start signal. During calculation, intermediate data and final result are fed out through the result write port. All operations are fully programmable by command code.

The Compact CNN Accelerator IP Core has different implementations for BNN and CNN depending on the Machine Learning Type attribute value. The functional block diagram for BNN and CNN implementation are shown in Figure 2.1 and Figure 2.3, respectively.

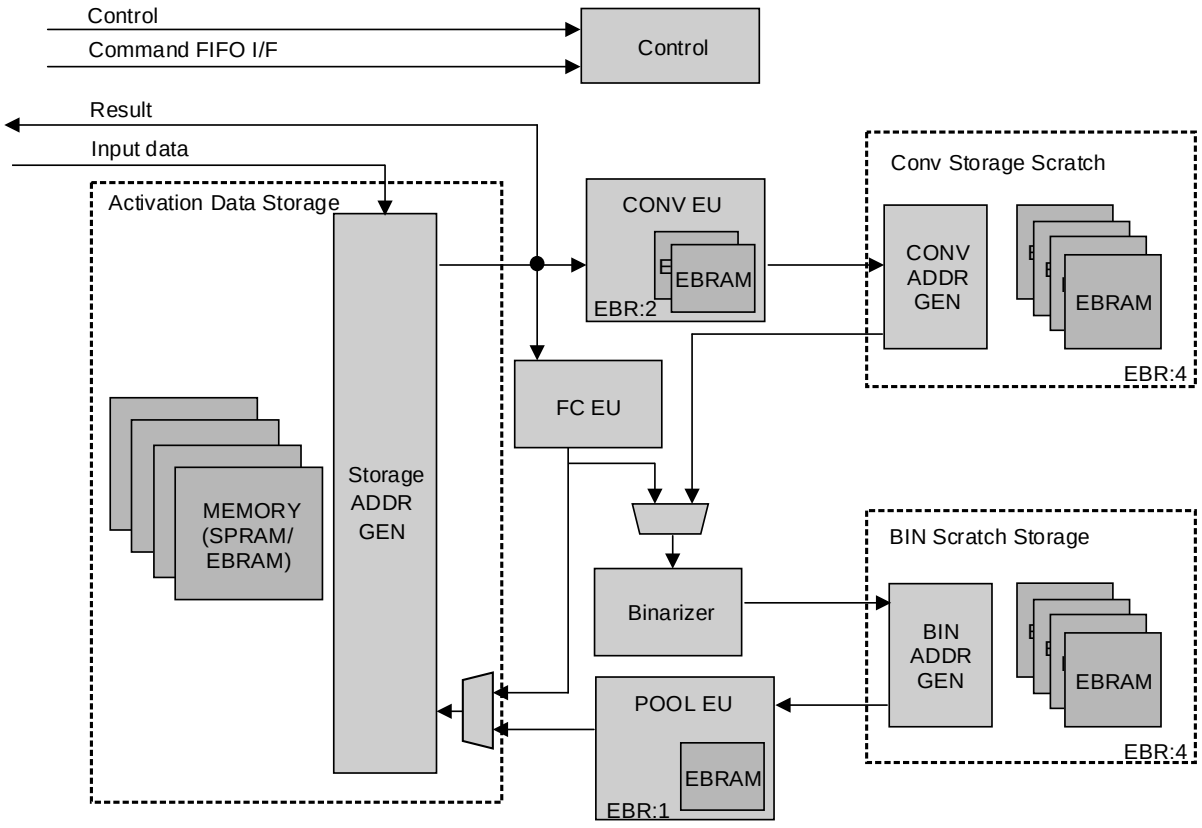

Figure 2.1. Functional Block Diagram (Machine Learning Type == BNN)

In Figure 2.1, the input data is stored in Activation Data Storage. It can feed data to either Convolution Engine or Full Connect Engine. The result of Convolution Engine is buffered to Convolution Scratch Storage and is transmitted to Binarizer. The output of Full Connect Engine can either go back to Activation Data Storage or to Binarizer also. The binarized data is buffered in BIN Scratch Storage before it goes to POOL EU. Both Binarizer and POOL EU has bypass option. Lastly, the result of POOL EU goes back to Activation Data Storage.

The possible order of layers for BNN configuration is shown in Figure 2.2. Follow this when designing your neural network model. The layers inside the parenthesis have bypass option.

The size of Activation Data Storage and Conv/BIN Scratch Storage are configurable, by Memory Type and Scratch Pad Memory Size attributes respectively. Refer to Table 3.2 for the description of these attributes.

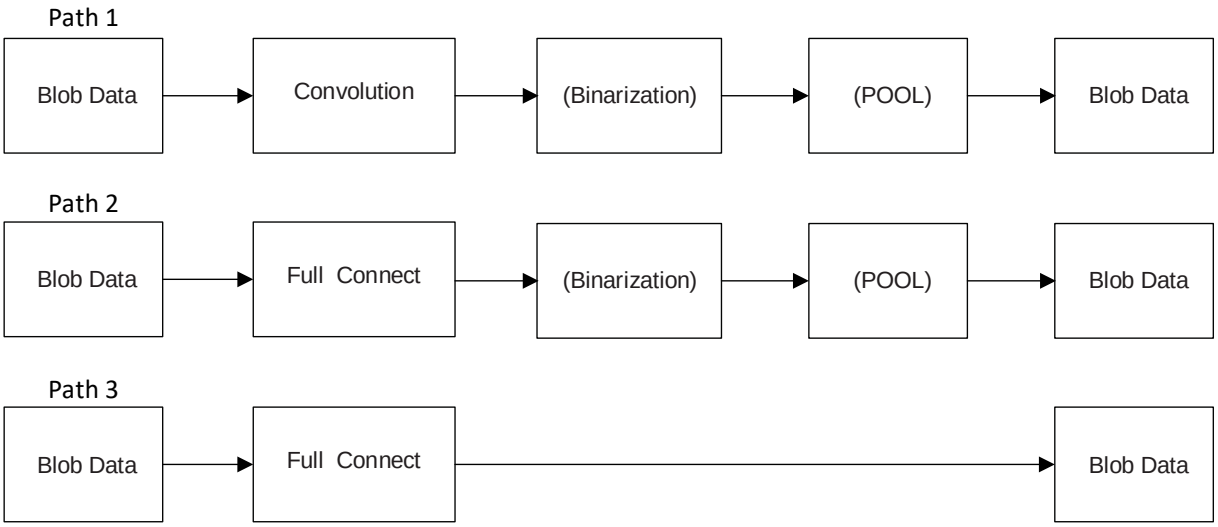

Figure 2.2. Order of Layers for BNN Option

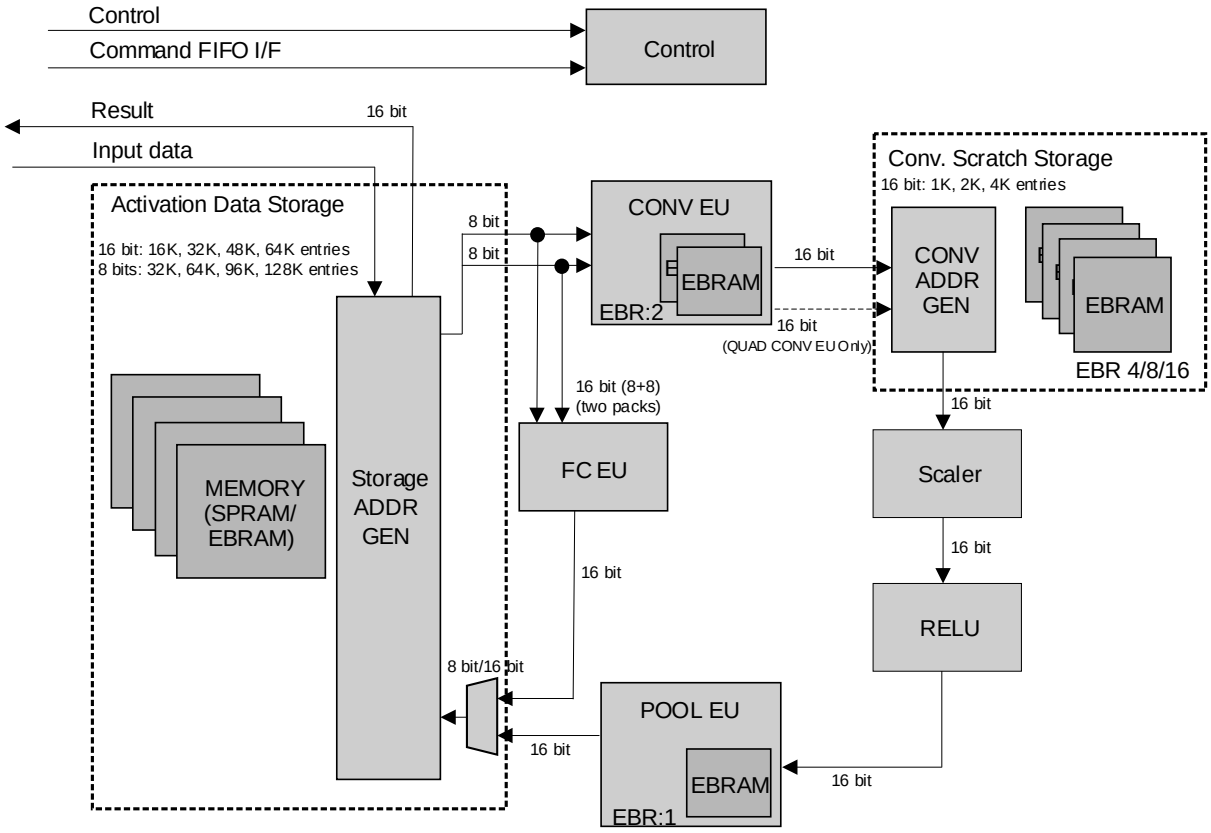

Figure 2.3. Functional Block Diagram (Machine Learning Type == CNN)

In Figure 2.3, the input data is stored in Activation Data Storage. It can feed data to either Convolution Engine or 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. Depending on the *No. of 1x1 Conv. Engines* attribute, two or four parallel convolution calculation can be performed. However, reading from Conv. Scratch Storage and feeding the data to Scaler for batch normalization is only done serially.

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 2.4. Follow this when designing your network model. The layers inside the parenthesis have bypass option.

The size of Activation Data Storage and Conv Scratch Storage are configurable, by Memory Type and Scratch Pad Memory Size attributes respectively. Refer to Table 3.2 for the description of these attributes.

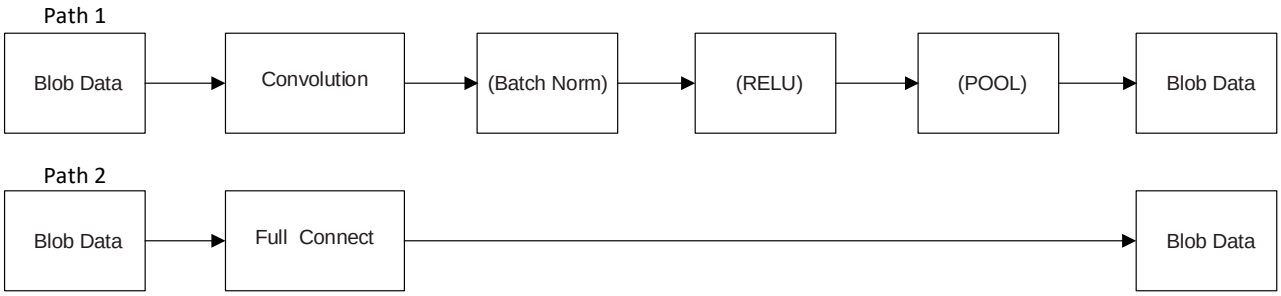

Figure 2.4. Order of Layers for CNN Option

2.2. Interface Descriptions

Figure 2.5 shows the interface diagram for the Compact CNN Accelerator IP Core. The diagram shows all of the available ports for the IP core.

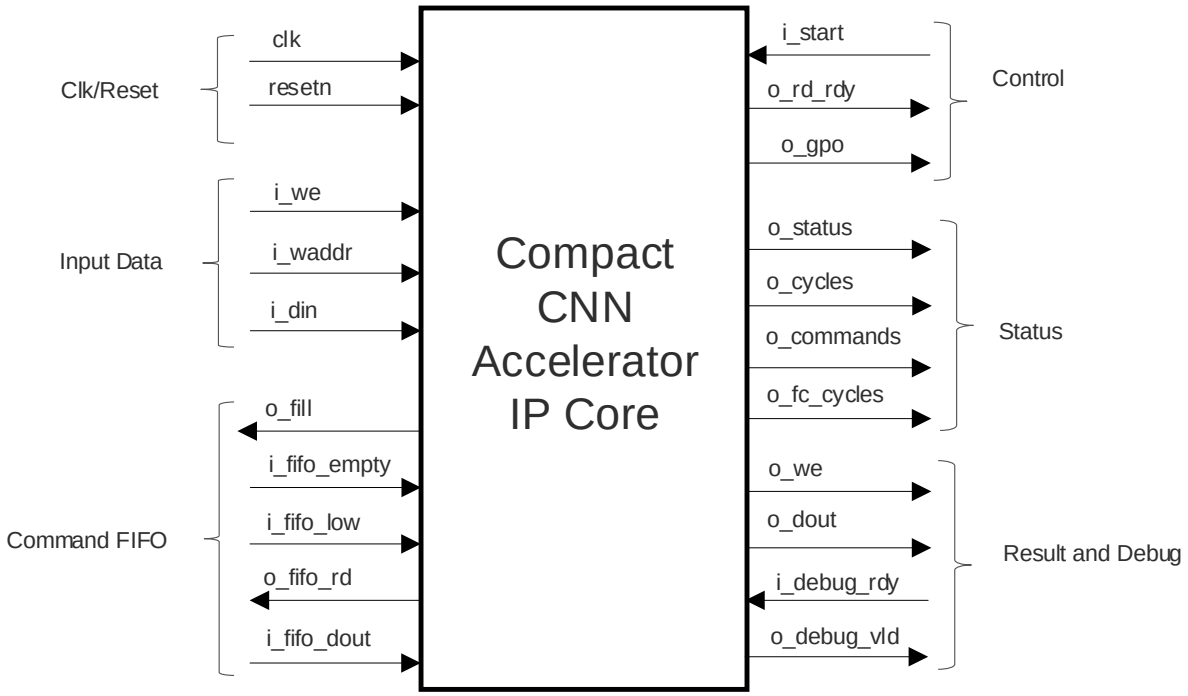

Figure 2.5. Compact CNN Accelerator IP Core Interface Diagram

Table 2.1. Compact CNN Accelerator IP Core Signal Descriptions

Pin Name	Direction	Function Description
Clock/Reset		
clk	Input	System clock Frequency can be chosen by trade-off between power and performance.
resetn	Input	Active low system reset that is synchronous to clk signal 0 – Resets all ports and sets internal registers to their default values. 1 – Reset is NOT active
Control and Status		
i_start	Input	Start execution signal. This signal is level sensitive and must deassert after o_rd_rdy going 0.
o_rd_rdy	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 o_rd_rdy is high.
o_status[7:0]	Output	Debug information [0] – Indicates activity of engines except full connect engine [1] – Indicates activity of full connect engine [2~7] – 0 (for future usage)
o_cycles[31:0]	Output	Indicates the number of clock cycles that Compact CNN Accelerator IP Core is running. The count is reset to 32'b0 on i_start assertion.
o_commands[31:0]	Output	Indicates the number of commands that Compact CNN Accelerator IP Core processed. The count is reset to 32'b0 on i_start assertion.
o_fc_cycles[31:0]	Output	Indicates the number of clock cycles that fully connected layer is running. The count is reset to 32'b0 on i_start assertion.
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		
i_we	Input	Write enable signal for internal memory 0 – No write transaction, i_waddr and i_din are ignored 1 – Write transaction is enabled, i_din and i_waddr are valid
i_waddr[15:0]	Input	Write address signal for internal memory.
i_din[15:0]	Input	Input data signal for internal memory.
Command FIFO		
o_fill	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 Neural Network Compiler must be loaded to external command FIFO. 1 – Normal FIFO operation
i_fifo_empty	Input	Indicates whether external command FIFO is empty or not. 0 – External FIFO is not empty 1 – External FIFO is empty
i_fifo_low	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.
o_fifo_rd	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.
i_fifo_dout[31:0]	Input	External command FIFO read data

Pin Name	Direction	Function Description
Result and Debug		
o_we	Output	Write enable signal of the result, indicates result data is valid. 0 – Result data is NOT valid 1 – Result data is valid
o_dout[15:0]	Output	IF o_we is asserted, o_dout[15:0] contains result data. IF o_debug_vld is asserted, o_dout[15:0] contains debug data.
i_debug_rdy	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. Recommend to connect to 1 if debug feature is not used so that the operation is not halted when the IP Core sends debug data.
o_debug_vld	Output	Indicates that o_dout[15:0] is not result data, but debug data. 0 – Debug data is NOT valid. 1 – Debug data is valid

2.2.1. Control and Status

After reset or when engine is idle, o_rd_rdy is high. During this state, external logic may write input data through the Input Data interface. After writing input data is completed, external logic must assert the i_start signal. The engine starts execution when it reaches i_start = 1 and o_rd_rdy goes 0 during execution. During execution, each bit of o_status indicates activity of sub calculation engine. After completing execution, that is, by getting to finish command, Compact CNN Accelerator IP Core asserts o_rd_rdy (goes idle) and waits for the next execution.

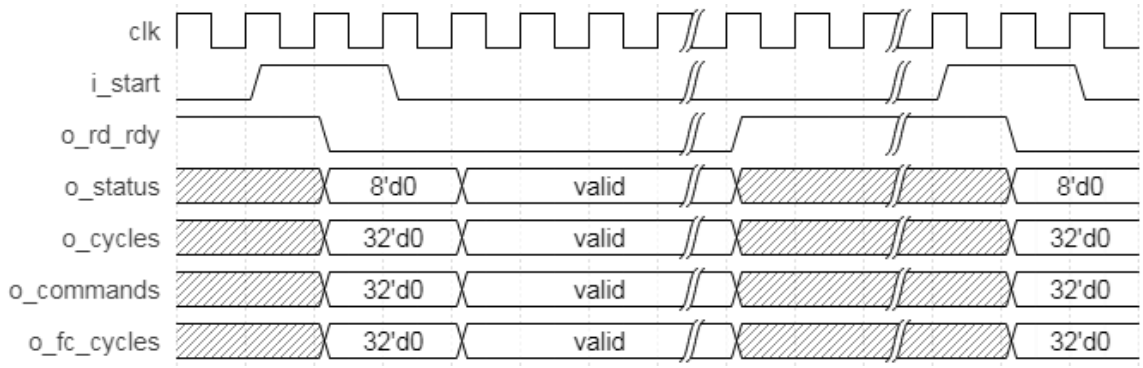

Figure 2.6. Control and Status Interface Timing Diagram

2.2.1.1. General Purpose Output

The general-purpose output signal, gpo_o is available since version v2.1.0 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 2.7. 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.

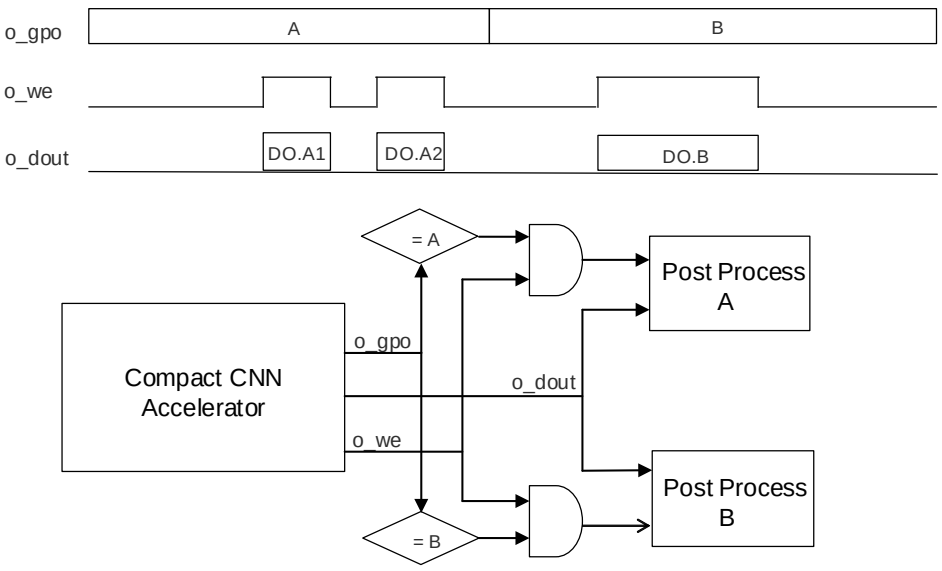

Figure 2.7 General Purpose Output Sample Application

2.2.2. Input Data Interface

External logic should write input data to internal memory of Compact CNN Accelerator IP Core only during idle state (o_rd_rdy is high). Writing to internal memory while o_rd_rdy is low is ignored. The write address for the input data must matched to command code. Input Data Interface is based on simple SRAM interface as shown in Figure 2.8. Since input data is written to internal SRAM, there is no required order or rule. Any random access is acceptable. Overwriting of the same address is also accepted.

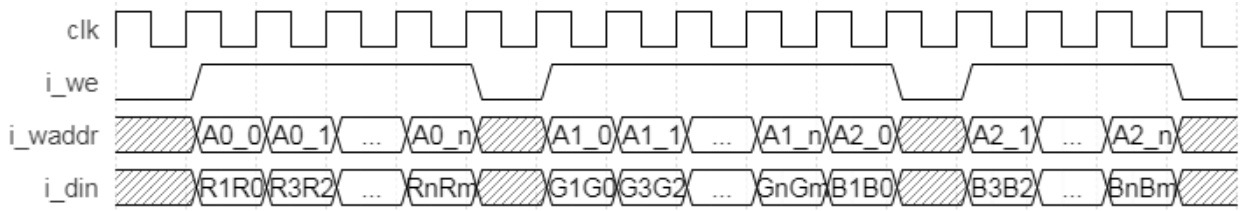

Figure 2.8. Input Data Interface Timing Diagram

2.2.3. Command FIFO Interface

External command FIFO must provide command code that is generated by the Lattice Neural Network Compiler tool. When o_fill signal is at logic 0, external logic should load the command code to external command FIFO. After the i_start control signal is asserted, o_fill go to logic 1 and normal FIFO operation proceeds as shown in Figure 2.9. When i_fifo_empty is low, i_fifo_dout contains a valid command. The command is latched/sampled when i_fifo_empty is low and o_fifo_rd is high. The external command FIFO holds current command (value of i_fifo_dout) when i_fifo_empty and o_fifo_rd are both low.

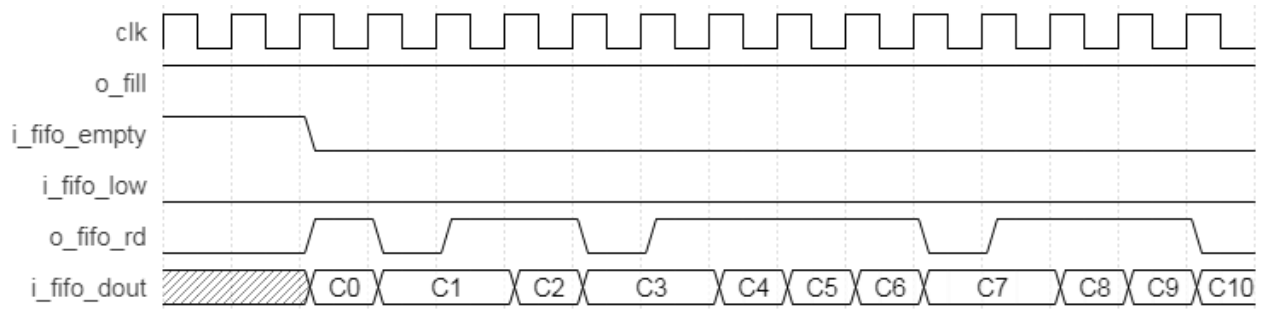

Figure 2.9. Command FIFO Interface Timing Diagram

2.2.4. Result and Debug Interface

This interface transmits the result, that is, by the final Blob data of the neural network. Interface consists of o_we as valid indicator and o_dout as 16-bit data. Usually, it may be a single burst series of 16-bit data. The amount of output data is programmable by command code. This interface does not have a ready signal. Thus, the receiving module should be able to receive the entire Blob.

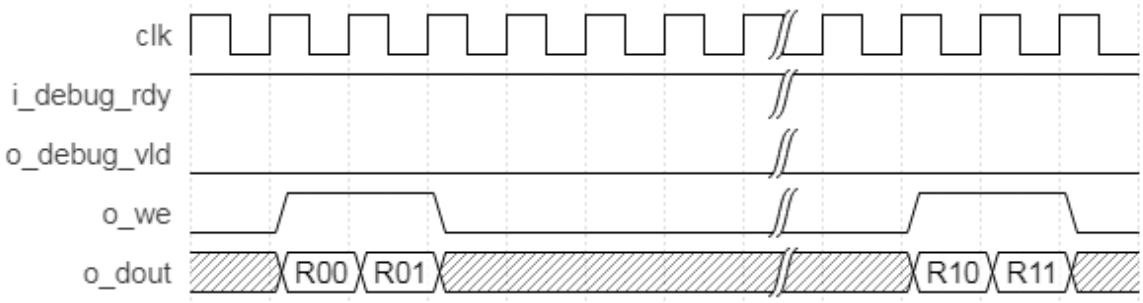

Figure 2.10. Result and Debug Interface Timing Diagram (No Debug Data)

Compact Compact CNN Accelerator IP Core may transfer debug data through o_dout port by asserting o_debug_vld signal per command code. The amount of debug data is also set in the command code. Debug data or intermediate result can be transferred in Result and Debug Interface without affecting normal operation. Unlike the Result data, transfer of debug data can be controlled by i_debug_rdy signal. Initially, external logic must be able to accept one burst read of debug data, and should assert i_debug_rdy signal to 1. If buffer is not available for next consecutive burst read, external logic must deassert i_debug_rdy signal to 0, in order to hold operation of Compact CNN Accelerator IP Core. A sample transmission of debug data and result data is shown in Figure 2.11. In this example, only two bursts of debug data are set in the command code thus, only two bursts are transmitted even if i_debug_rdy signal is still asserted.

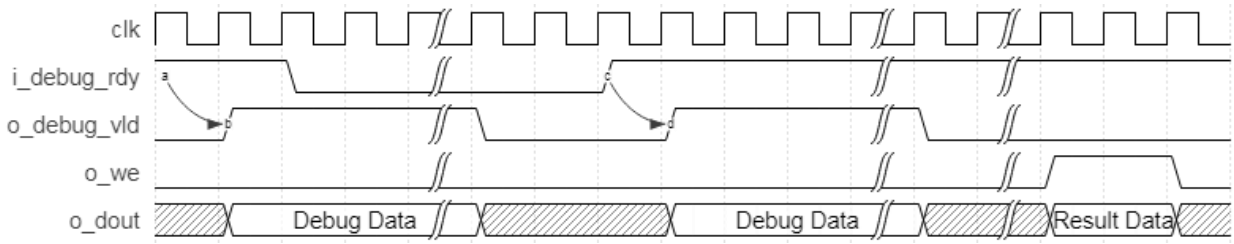

Figure 2.11. Result and Debug Interface Timing Diagram (With Debug Data)

2.3. Reset Behavior

When resetn signal asserts, output ports return to default value in the next cycle. The default value of o_rd_rdy signal is 1, while all other output signals are 0. A timing diagram of reset during operation is shown as an example in Figure 2.12. The minimum resetn assert period is 1 clk cycle.

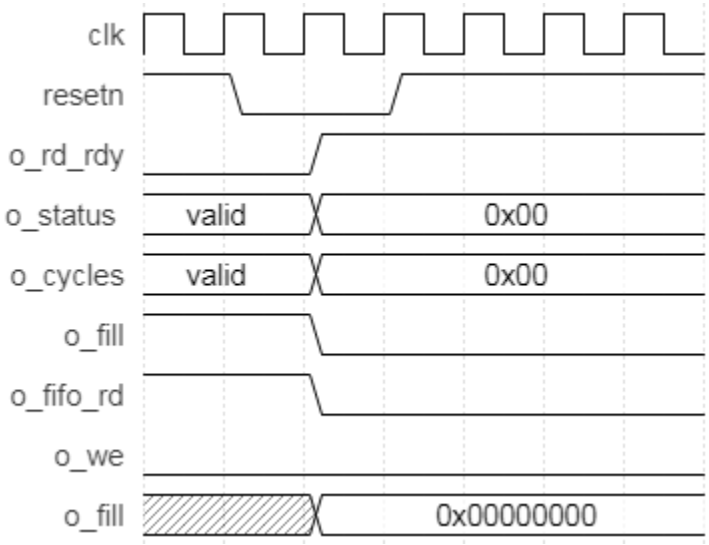

Figure 2.12. Reset Timing Diagram

2.4. Register Description

Compact CNN Accelerator IP Core has no user-configurable registers.

2.5. Operation Sequence

Operation must be executed in the following sequence:

1. Assert Reset.
2. Deassert Reset; i_start must be deasserted.
3. Write the command sequence code, which is generated by the Lattice Neural Network Compiler tool, into FIFO.
4. Check whether o_rd_rdy is high or not. o_rd_rdy must be high. Otherwise, go back to step 1.
5. Write into memory block of Compact CNN Accelerator IP Core through Input Data interface at proper address, which is decided by command sequence.
6. Assert i_start and check o_rd_rdy. o_rd_rdy signal should be 0 after asserting i_start.
7. Deassert i_start.
8. Check o_we and collect o_dout while o_we == 1.
9. Repeat from step 5 for the next Input data.

2.5.1. Command Format

Command is a sequence of 32-bit data with or without additional parameters or weights. It should be loaded at address 0x0000 before execution. Engine expects command in little-endian order. Command is generated by the Lattice Neural Network Compiler tool. For more information, refer to Lattice SensAI Neural Network Compiler Software User Guide (FPGA_UG-02052).

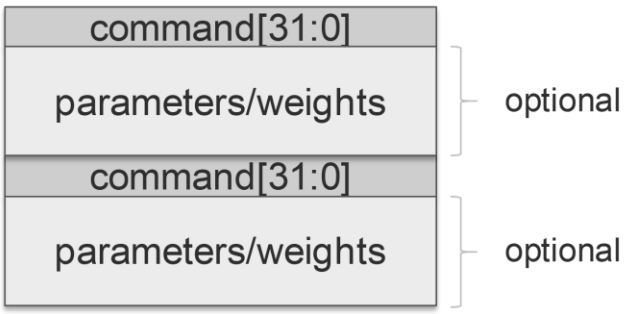

Figure 2.13. Command Format

2.5.2. Input Data Format

Input data is a sequence of 8-bit or 16-bit data depending on Byte Mode attribute setting, please see Table 3.2 for details. Address is decided by the neural network. Therefore, external block should process input raw data and write input data to the Compact CNN 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 x 32 of R, G, B planes at memory with address 0x0000 for Red plane, 0x0400 for Green plane, and 0x0800 for Blue plane. Because memory assignment is defined by the neural network, external block should handle input raw data and write it to proper position of internal memory of Compact CNN Accelerator IP Core. The IP Core expects data in little-endian order.

2.5.3. 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 the neural network, that is, by output Blobs. External block should interpret output sequence and generate usable information. For example, face detection may have two classes. Therefore, output may be two of 16-bit data and final result may be simple comparison of these two 16-bit data. 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.

2.6. Supported Commands

Command sequences are generated by the Lattice Neural Network Compiler tool. For more information, refer to Lattice SensAI Neural Network Compiler Software User Guide (FPGA-UG-02052).

3. Parameter Settings

The IP Catalog is used to create IP and architectural modules in the Lattice Radiant Software. Refer to the IP Generation and Evaluation section on how to generate the IP.

Table 3.1 provides the list of user-configurable attributes for the Compact CNN Accelerator IP Core. The attribute values are specified using the IP core Configuration user interface in the Lattice Radiant Software as shown in Figure 3.1.

Table 3.1. Attributes Table

Attribute	Selectable Values	Default	Dependency on Other Attributes
Machine Learning Type	CNN, BNN	CNN	—
Memory Type	EBRAM, DUAL_SPRAM, SINGLE_SPRAM, TRI_SPRAM, QUAD_SPRAM	DUAL_SPRAM	TRI_SPRAM and QUAD_SPRAM are only available for Machine Learning Type == CNN
BNN Blob Type	+1/-1, +1/0	+1/0	Machine Learning Type == BNN
No. of 1x1 Conv. Engines[*2]	SINGLE, DUAL, QUAD	SINGLE	Machine Learning Type == CNN AND Byte Mode != DISABLE
Scratch Pad Memory Size[*3]	1K, 2K, 4K	1K	Machine Learning Type == CNN, No. of 1x1 Conv. Engines
Byte Mode	SIGNED, UNSIGNED, DISABLE	DISABLE	—
Byte Shift	b000, b001, b010, b011, b100, b101, b110, b111	b101	Machine Learning Type == CNN AND Byte Mode != DISABLE[*1]

[*]**Notes:**

1. Not editable for IP Core v2.0.x and SensAI v2.1.
2. No. of 1 x 1 Conv. Engines is available from IP core version v2.1.0.
3. When *No. of 1x1 Conv. Engines*==QUAD, two of 1K, 2K memory are used, effectively using 2K, 4K entries.

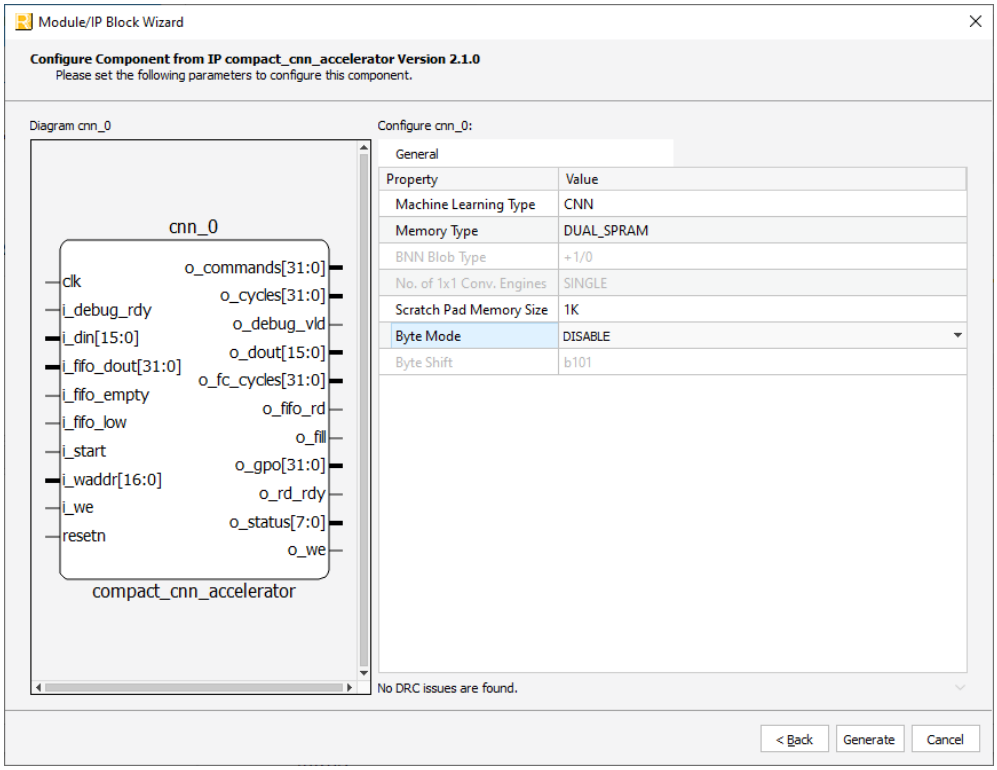

Figure 3.1. Compact CNN Accelerator IP Core Configuration User Interface

Table 3.2. Attributes Descriptions

Attribute	Description
Machine Learning Type	Selects the machine learning engine type between CNN and BNN
Memory Type	EBRAM – Use 16 EBRAM. Total size is 8 kB. SINGLE_SPRAM – Use 1 SPRAM. Total size is 32 kB. DUAL_SPRAM – Use 2 SPRAMs. Total size is 64 kB. TRI_SPRAM – Use 3 SPRAMs. Total size is 96 kB. QUAD_SPRAM: Use 4 SPRAMs. Total size is 128 kB. Set this attribute based on required size of peak Blob data in the Neural Network compiler: • Use SINGLE_SPRAM when network blob data <= 32 kB • Use DUAL SPRAM when network blob data > 32 kB and <= 64 kB • Use TRI SPRAM when network blob data > 64 kB and <= 96 kB • Use QUAD SPRAM when network blob data > 96 kB and <= 128 kB EBRAM option is for special usage when no SPRAM is available.
BNN Blob Type	Selects the type of binary blob data, either +1/−1 or +1/0. This setting should be matched to Lattice Neural Network Compiler. This setting is only valid when Machine Learning Type == BNN.
No. of 1x1 Conv. Engines	Selects the number of 1x1 Convolution calculations in parallel. SINGLE: Single 1x1 convolution only DUAL: Supports single and dual 1x1 convolution QUAD: Supports single, dual and quad 1x1 convolution Parallel convolution engine speeds up the convolution calculation at the cost of more resource.
Scratch Pad Memory Size	Configures the memory size of scratch pad of convolution. 1K – Size of scratch pad memory is 2 kB (1K x 2 bytes). 2K – Size of scratch pad memory is 4 kB (2K x 2 bytes). 4K – Size of scratch pad memory is 8 kB (4K x 2 bytes). This setting is only valid when Machine Learning Type == CNN. The BNN engine uses fixed scratch memory of 1K (2 kB) for the BIN and convolution functions. Set this attribute based on the image size for convolution: • Use 1K when image size <= 32 x 32 • Use 2K/4K when image size > 32 x 32
Byte Mode	Specifies the byte mode of input data. SIGNED – input data is signed 8-bit data. UNSIGNED – input data is unsigned 8-bit data. DISABLE – input data is unsigned 16-bit data only, similar to v1.1.
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 blob memory as Byte Mode (signed or unsigned) and reading out the byte data to the 16-bit o_dout signal. Refer to Table 3.3 for the shifting of data byte based on Byte Shift and Byte Mode. This is fixed to default value (b101) for IP Core v2.1 and SensAI v2.1.

Table 3.3. 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.

4. IP Generation and Evaluation

This section provides information on how to generate the IP using the Lattice Radiant software, and how to run simulation, synthesis, and hardware evaluation. For more details, refer to the Lattice Radiant Software User Guide.

4.1. Licensing the IP

IP core v1.1.x and later versions require an IP core-specific license string and a Lattice Radiant software 1.1 license patch, or later radiant Software version, to enable full use of the Lattice Compact CNN Accelerator IP Core in a complete, top-level design. Without a license string and the software license patch, the bitstream file is not generated. Compact CNN Accelerator IP Core license string is available in 30-day evaluation license and full license with yearly renewal. The evaluation license may be used only for the purpose of checking the usability of the IP core. The bitstream generated using evaluation license must not be used in actual product.

If you want to use the Compact CNN Accelerator IP Core in your product, you may obtain a 30-day evaluation license by going to http://www.latticesemi.com/Support/Licensing/IPCore/CompactCNN and entering your 12-digit hexadecimal Host Physical Address (MAC address). For details on enabling the 30-day evaluation license, go to http://www.latticesemi.com/en/Support/AnswerDatabase/5/8/0/5808.

Ensure that you are logged in using your organization/company email as the license file is sent to this email.

If you have purchased and received the IP Core product, you may obtain the full license by going to http://www.latticesemi.com/en/Support/Licensing/IPCore/IPCoreNew and providing all required information.

The IP core v1.1.x and v2.0.0 supports Lattice Radiant software 1.1 and future versions. For more details, refer to the Lattice Radiant Software 1.1 User Guide and Lattice Radiant Software 1.1 Tutorial.

You can obtain the Lattice Radiant 1.1. software patch file from the Lattice website through Lattice Radiant 1.1 Software Patch.

4.2. Generation and Synthesis

The Lattice Radiant software allows customization and generation of modules. The procedure for generating Compact CNN Accelerator IP Core in Lattice Radiant software is described below:

1. Create a new Lattice Radiant software project or open an existing project.
2. In the IP Catalog tab, double-click on **Compact_CNN_Accelerator** under IP, DSP category. The Module/IP Block Wizard - appears as shown in Figure 4.1. Enter the values in the **Instance name** and the **Create in** fields.

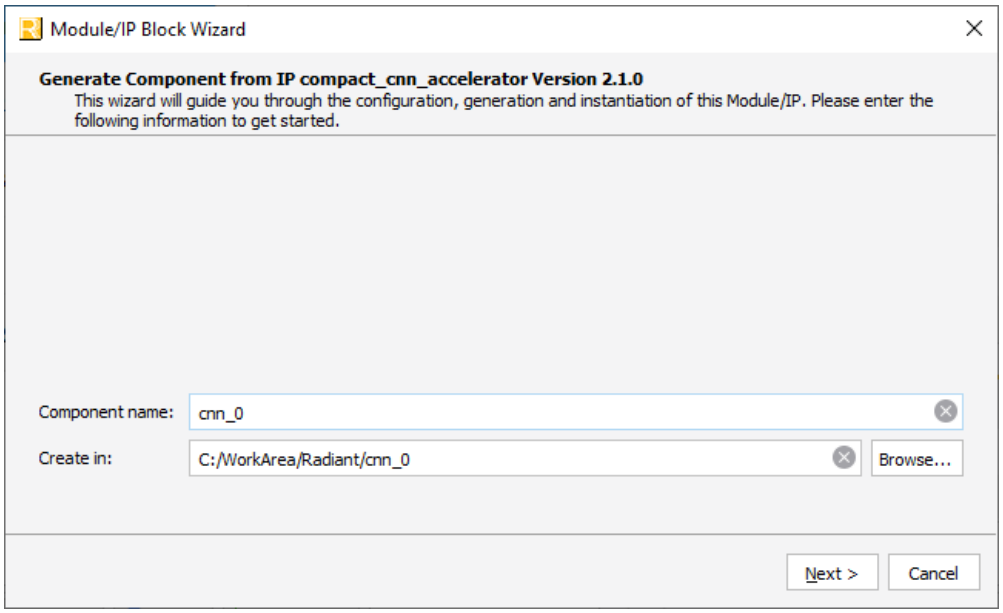

Figure 4.1. Module/IP Block Wizard

3. Click **Next**.
4. Update the configuration parameters as necessary as shown in Figure 4.1.
5. Click the **Generate** button to generate the IP. Confirm the generation result as shown in Figure 4.2. Check the **Insert to project** option to add the generated IP to the project.
6. Click **Finish**. All the generated files are placed under the directory paths in the Create in and the Instance name fields shown in Figure 4.1.

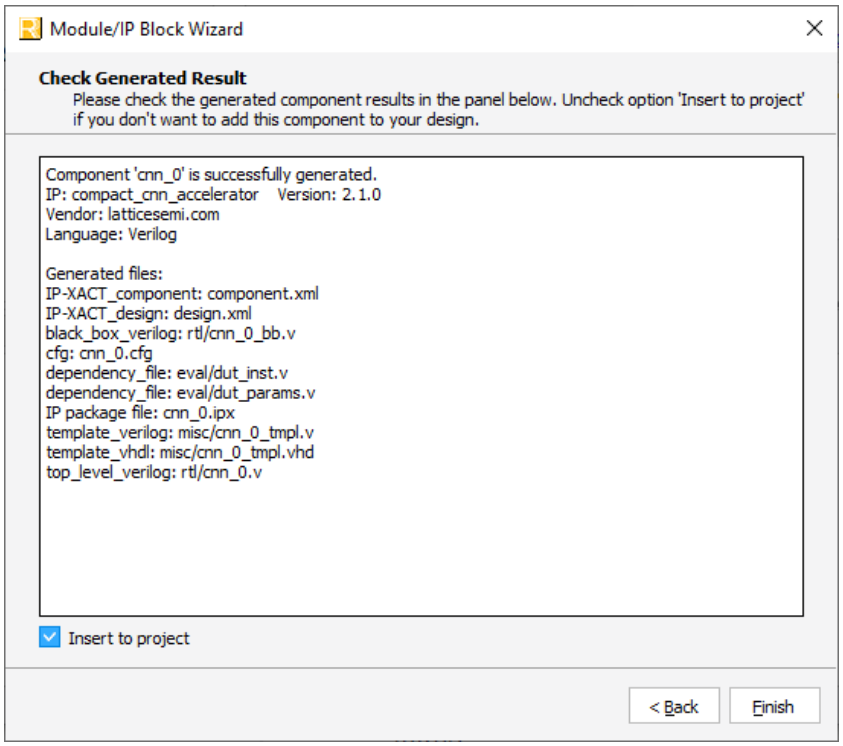

Figure 4.2. Check Generating Result

The generated Compact CNN Accelerator IP Core package includes black-box (<Instance Name>_bb.v) and instance templates (<Instance 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 (<Instance 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 4.1.

Table 4.1. Generated File List

Attribute	Description
<Instance Name>.ipx	This file contains the information on the files associated to the generated IP.
<Instance Name>.cfg	This file contains the parameter values used in IP configuration.
rtl/<Instance Name>.v	This file provides an example RTL top file that instantiates the IP core.
rtl/<Instance Name>_bb.v	This file provides the synthesis black box for the user's synthesis.
misc/<Instance Name>_tmpl.v misc /<Instance Name>_tmpl.vhd	These files provide instance templates for the IP core.

4.3. Running Functional Simulation

The Compact CNN Accelerator IP Core does NOT contain a sample test bench.

5. Ordering Part Number

The Ordering Part Numbers (OPN) for Compact CNN Accelerator IP Core targeting iCE40 UltraPlus FPGA devices are the following:

- CNN-CPACCEL-UP-U – Project License
- CNN-CPACCEL-UP-UT – Site License

References

- iCE40UP Web Page at latticesemi.com

Technical Support Assistance

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

Appendix A. Resource Utilization

Table A.1 and Table A.2 show the resource utilization of the Compact CNN Accelerator IP Core for the iCE40 UltraPlus device, using Lattice Radiant software 2.2. Default configuration is used and some attributes are changed from the default value to show the effect on the resource utilization. Refer to Table 3.1 for the default configuration.

The following settings are used in generating this data:

- Synthesis Tool: Lattice Synthesis Engine
- Device Part No.: iCE40UP5K-SG48I

Table A.1. Performance and Resource Utilization (Machine Learning Type == BNN)[1]

Configuration	Register	LUT4s	EBR	SRAM	clk Fmax[2] (MHz)
Memory Type = EBRAM, Others = Default	2055	2662	27	0	39.305
Memory Type = SINGLE_SPRAM, Others = Default	2035	2652	11	1	41.873
Default	2036	2654	11	2	42.380
BNN Blob Type = +1/-1, Others = Default	2026	2496	11	2	42.380

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 Compact CNN Accelerator IP Core, these values may be reduced when user logic is added to the FPGA design.

Table A.2. Performance and Resource Utilization (Machine Learning Type == CNN)[1]

Configuration	Register	LUT4s	EBR	SRAM	clk Fmax[2] (MHz)
Memory Type == EBRAM, Others = Default	1879	2265	15	0	40.228
Memory Type == SINGLE_SPRAM, Others = Default	1891	2296	7	1	38.300
Default	1892	2312	7	2	37.859
Memory Type == TRI_SPRAM, Others = Default	1902	2351	7	3	40.743
Memory Type == QUAD_SPRAM, Others = Default	1902	2348	7	4	41.161
Scratch Pad Memory Size = 2K, Others = Default[3]	1895	2314	11	2	38.725
Scratch Pad Memory Size = 4K, Others = Default[3]	1903	2330	19	2	40.112
Byte Mode = SIGNED, Other = Default	1892	2323	7	2	39.749
Byte Mode = SIGNED, No of 1x1 Conv. Engines == DUAL, Other = Default	1925	2384	7	2	39.456
Byte Mode = UNSIGNED, Others = Default	1901	2329	7	2	38.513

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 Compact CNN Accelerator IP Core, these values may be reduced when user logic is added to the FPGA design.
3. The *K* value in Scratch Pad is equivalent to 1K entries x 2 bytes. For example, 4K is equal to *8 kB* of scratch pad memory.

For more information on Lattice Radiant software, visit the Lattice web site at www.latticesemi.com/Products/DesignSoftwareAndIP.

Revision History

Revision 1.7, December 2020

Section	Change Summary
Acronyms in This Document	Added this section.
Introduction	• Added features. • Updated Table 1.1.
Functional Description	• Updated Figure 2.3 and its description. • Added o_gpo signal in Figure 2.5 and Table 2.1. • Added General Purpose Output section.
Parameter Settings	Updated Table 3.1 and Table 3.2.
IP Generation and Evaluation	Updated content.
References	Updated content.
Appendix A. Resource Utilization	Updated Table A.2.

Revision 1.6, March 2020

Section	Change Summary
Introduction	Updated content.
Functional Description	• Updated Figure 2.1, Figure 2.3, and Figure 2.5. • Updated content of Overview, Register Description, and Operation Sequence section. • Updated Table 2.1.
Parameter Settings	Updated Table 3.2.
Appendix A. Resource Utilization	Updated Table A.2.

Revision 1.5, November 2019

Section	Change Summary
Input Data Format	Updated description.
Parameter Settings	Updated Byte Mode description in Table 3.2. Attributes Descriptions.

Revision 1.4, October 2019

Section	Change Summary
Introduction	Updated Features section.
Parameter Settings	• Added parameters in Table 3.1. Attributes Table. • Updated Table 3.2. Attributes Descriptions.
Appendix A. Resource Utilization	• Updated Table A.1. Performance and Resource Utilization (Machine Learning Type == BNN). • Added Table A.2. Performance and Resource Utilization (Machine Learning Type == CNN).

Revision 1.3, July 2019

Section	Change Summary
Introduction	Updated Features section.
Functional Description	• Added description for Figure 2.1. Functional Block Diagram (Machine Learning Type == BNN) and Figure 2.3. Functional Block Diagram (Machine Learning Type == CNN). • Added Figure 2.2. Order of Layers for BNN Option and Figure 2.4. Order of Layers for CNN Option.
IP Generation and Evaluation	Added reference to enabling 30-day evaluation license in Licensing the IP section.

Revision 1.2, May 2019

Section	Change Summary
All	• Added Disclaimers section. • Updated last page of the document.
Introduction	Added IP Core version – Radiant software version compatibility in Quick Facts section.
IP Generation and Evaluation	Added licensing information in Licensing the IP section.
Ordering Part Number	Added Ordering Part Number information.

Revision 1.1, September 2018

Section	Change Summary
All	Renamed the IPUG from BNN Accelerator IP Core to Compact CNN Accelerator IP Core. Added support for convolutional neural network with fix point weights.
Introduction	Updated Features section. Added additional features.
Functional Description	• Updated text, Figure 2.1. Functional Block Diagram (Machine Learning Type == BNN) and Figure 2.2. Functional Block Diagram (Machine Learning Type == CNN) in Overview section. • Updated Figure 2.3. Compact CNN Accelerator IP Core Interface Diagram and Table 2.1. Compact CNN Accelerator IP Core Signal Descriptions in Interface Descriptions section.
Parameter Settings	Updated Figure 3.1. Compact CNN Accelerator IP Core Configuration User Interface, Table 3.1. Attributes Table, and Table 3.2. Attributes Descriptions.
Appendix A. Resource Utilization	• Updated Table A.1. Performance and Resource Utilization (Machine Learning Type == BNN). • Added Table A.2. Performance and Resource Utilization (Machine Learning Type == CNN).

Revision 1.0, May 2018

Section	Change Summary
All	Initial release

www.latticesemi.com