CNN Accelerator IP Core

User Guide

FPGA-IPUG-02037-2.2

December 2020

Disclaimers

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
OPN	Ordering Part Number
SRAM	Static Random Access Memory

1. Introduction

The Lattice Semiconductor CNN Accelerator IP Core is a calculation engine for Deep Neural Network with fixed point weight or binarized 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 ECP5 and ECP5-5G FPGA devices, and implemented using the Lattice Diamond® 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 Accelerator IP Core.

Table 1.1. Quick Facts

IP Requirements	FPGA Families Supported	ECP5, ECP5-5G
Resource Utilization	Targeted Device	Full configuration: 85k devices in ECP5 families Reduced configuration: All devices in ECP5 families
	Supported User Interface	AXI4, Native interfaces as described in Interface Descriptions section.
	Resources	See Table A.1.
Design Tool Support	Lattice Implementation	Lattice Diamond software 3.10 or later
	Synthesis	Lattice Synthesis Engine
		Synopsys® Synplify Pro for Lattice
	Simulation	For a list of supported simulators, see the Lattice Diamond User Guide.

1.2. Features

The key features of the CNN Accelerator IP Core include:

- Support for convolution layer, max/ave pooling layer, batch normalization layer, and full connect layer
- Configurable bit width of weight (16-bit, 1-bit)
- Configurable bit width of activation (16/8-bit, 1-bit)
- Dynamic support for 16-bit and 8-bit width of activation
- Configurable number of memory blocks for tradeoff between resource and performance
- Configurable number of convolution engines for tradeoff between resource and performance
- Optimization for 3x3 2D convolution calculation
- Dynamic support for various 1D convolution from 1 to 72 taps
- Supports max pooling with overlap (For example, kernel 3, stride 2)
- Supports average pooling for 2x2 convolution
- Supports global average pooling by full connect engine
- Supports paired convolution engines to improve performance
- Configurable input byte mode (signed, unsigned, disable)
- Partial DRAM access
- Configurable maximum burst length (32, 256)
- Supports MobileNet
- Supports general purpose output signal for controlling external logic through command code

2. Functional Descriptions

2.1. Overview

The CNN 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 DRAM address specified by the i_code_base_addr signal which is accessible through AXI BUS. Input data may be read from DRAM at a pre-defined address or directly written through the input data write port. After command code and input data are available, CNN Accelerator IP Core starts calculation at the rising edge of start signal. During calculation, intermediate data and final result may be transferred to DRAM or fed out through the result write port. All operations are fully-programmable by command code.

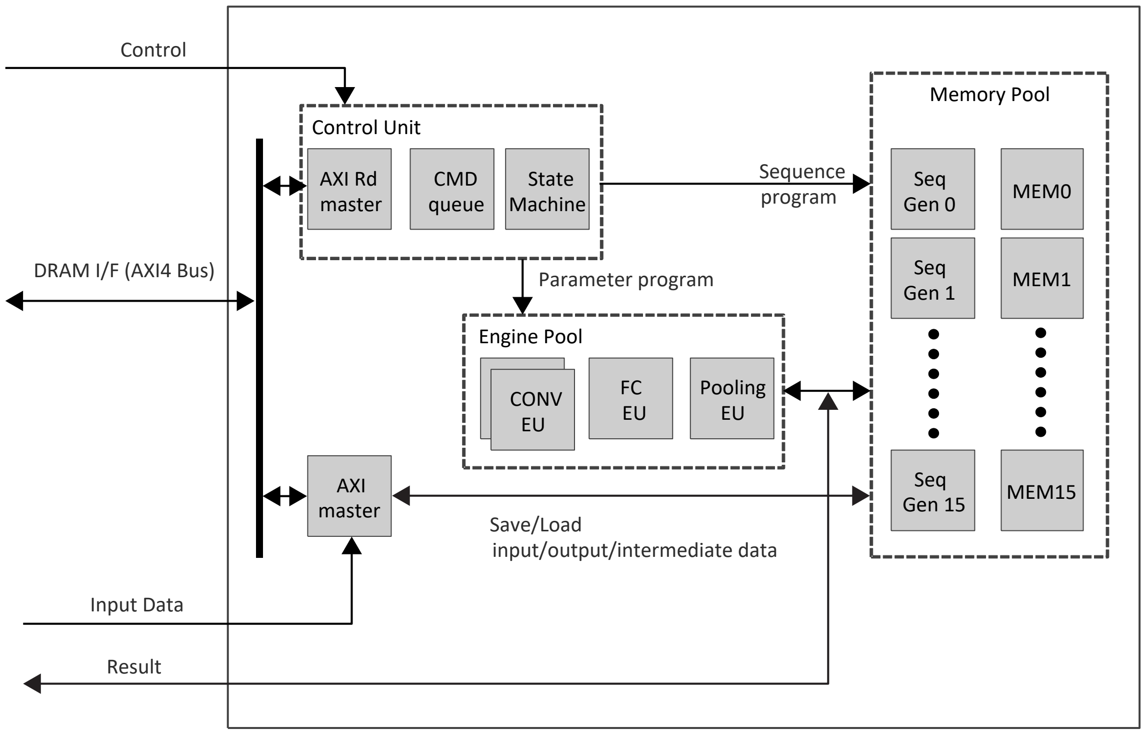


Figure 2.1. Functional Block Diagram

2.2. Interface Descriptions

Figure 2.2 shows the interface diagram for the CNN Accelerator IP Core. The diagram shows all of the available ports for the IP core.

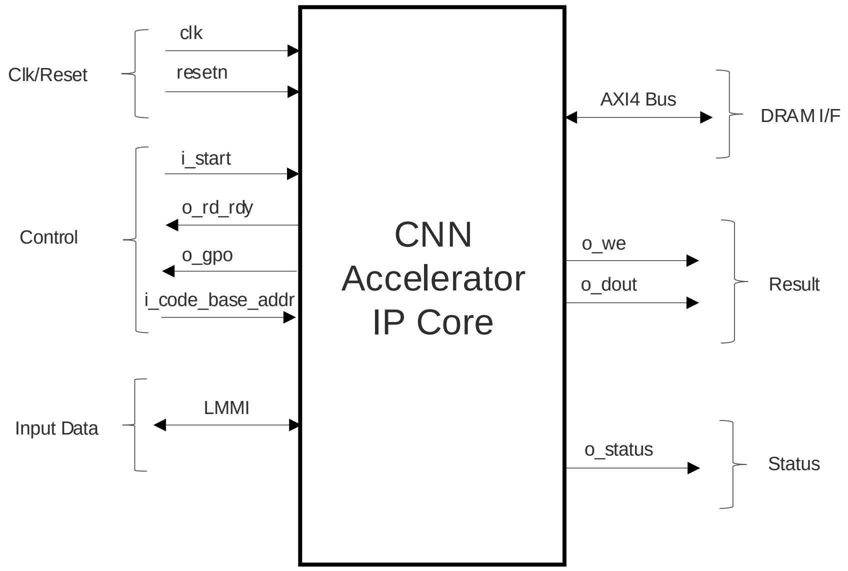


Figure 2.2. CNN Accelerator IP Core Interface Diagram

Table 2.1. 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 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		
i_code_base_addr[31:0]	Input	This signal specifies the base/start address that is read by CNN 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.
i_start	Input	Start execution signal. Level sensitive. 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	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. [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.

Pin Name	Direction	Function Description
Input Data (LMMI)		
i_lmmi_request	Input	Start transaction
i_lmmi_wr_rdn	Input	Write = HIGH, Read = LOW
i_lmmi_offset	Input	Address offset [12:0] – Address signal for an internal memory [16:13] – Selects between internal memories (MEM0, MEM1, …, MEM15)
i_lmmi_wdata	Input	Write data
o_lmmi_ready	Output	Slave Ready signal. When o_lmmi_ready and i_lmmi_request are both HIGH, it indicates write transaction is completed and address phase of ready transition is completed.
o_lmmi_rdata_valid	Output	Indicates read transaction is complete and lmmi_rdata contains valid data.
o_lmmi_rdata	Output	Read data
Result		
o_we	Output	Write enable of result, indicates result data is valid [0] – Result data is NOT valid [1] – Result data is valid
o_dout[15:0]	Output	Result data
DRAM I/F (AXI4 Bus)		
aclk	Input	AXI4 clock signal. Fully asynchronous from clk. Recommend to use DRAM system clock.
A2M_AWID[7:0]	Output	AXI4 write address channel, write address ID signal *Constant output: 0x10*
A2M_AWADDR[31:0]	Output	AXI4 write address channel, write address signal
A2M_AWREGION[3:0]	Output	AXI4 write address channel, region identifier signal *Constant output: 0x0 (default)*
A2M_AWLEN[7:0]	Output	AXI4 write address channel, burst length signal
A2M_AWSIZE[2:0]	Output	AXI4 write address channel, burst size signal *Constant output: 3'b011 (8 bytes per beat)*
A2M_AWBURST[1:0]	Output	AXI4 write address channel, burst type signal *Constant output: 2'b01 (INCR)*
A2M_AWLOCK	Output	AXI4 write address channel, lock type signal *Constant output: 1'b0 (Normal Access)*
A2M_AWCACHE[3:0]	Output	AXI4 write address channel, memory type signal *Constant output: 0x00 (Device Non-bufferable)*
A2M_AWPROT[2:0]	Output	AXI4 write address channel, protection type signal *Constant output: 3’b000 (Secure Access)*
A2M_AWQOS[3:0]	Output	AXI4 write address channel, quality of service signal *Constant output: 0x0 (no QoS scheme)*
A2M_AWVALID	Output	AXI4 write address channel, write address valid signal
A2M_AWREADY	Input	AXI4 write address channel, write address ready signal
A2M_WID[7:0]	Output	AXI4 write data channel, write ID tag signal *Constant output: 0x10*
A2M_WDATA[63:0]	Output	AXI4 write data channel, write data signal
A2M_WSTRB[7:0]	Output	AXI4 write data channel, write strobe signal *Constant output: 0xFF*
A2M_WLAST	Output	AXI4 write data channel, write last signal
A2M_WVALID	Output	AXI4 write data channel, write valid signal
A2M_WREADY	Input	AXI4 write data channel, write ready signal
A2M_BID[7:0]	Input	AXI4 write response channel, response ID tag signal
A2M_BRESP[1:0]	Input	AXI4 write response channel, write response signal
A2M_BVALID	Input	AXI4 write response channel, write response valid signal

Pin Name	Direction	Function Description
A2M_BREADY	Output	AXI4 write response channel, response ready signal *Constant output: 1'b1* (AXI4 write response channel is ignored*)*
A2M_ARID[7:0]	Output	AXI4 read address channel, read address ID signal 0x00: AXI Master for Memory Pool 0x10: AXI Rd Master in Control Unit Refer to Figure 2.1 for the AXI Masters
A2M_ARADDR[31:0]	Output	AXI4 read address channel, read address signal
A2M_ARREGION[3:0]	Output	AXI4 read address channel, region identifier signal *Constant output: 0x0 (default)*
A2M_ARLEN[7:0]	Output	AXI4 read address channel, burst length signal
A2M_ARSIZE[2:0]	Output	AXI4 read address channel, burst size signal *Constant output: 3'b011 (8 bytes per beat)*
A2M_ARBURST[1:0]	Output	AXI4 read address channel, burst type signal *Constant output: 2'b01 (INCR)*
A2M_ARLOCK	Output	AXI4 read address channel, lock type signal *Constant output: 1'b0 (Normal Access)*
A2M_ARCACHE[3:0]	Output	AXI4 read address channel, memory type signal *Constant output: 0x01 (Device Bufferable)*
A2M_ARPROT[2:0]	Output	AXI4 read address channel, protection type signal *Constant output: 3'b010 (Non-secure Access)*
A2M_ARQOS[3:0]	Output	AXI4 read address channel, quality of service signal *Constant output: 0x0 (no QoS scheme)*
A2M_ARVALID	Output	AXI4 read address channel, read address valid signal
A2M_ARREADY	Input	AXI4 read address channel, read address ready signal
A2M_RID[7:0]	Input	AXI4 Read data channel, read ID tag signal
A2M_RDATA[63:0]	Input	AXI4 Read data channel, read data signal
A2M_RRESP[1:0]	Input	AXI4 Read data channel, read response signal
A2M_RLAST	Input	AXI4 Read data channel, read last signal
A2M_RVALID	Input	AXI4 Read data channel, read valid signal
A2M_RREADY	Output	AXI4 Read data channel, read ready signal

2.2.1. Control and Status Interface

After reset or when engine is idle, o_rd_rdy 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 i_code_base_addr signal

After the above steps, external logic must assert i_start signal. Engine starts execution when it gets i_start = 1 and o_rd_rdy goes 0. During execution, each bit of o_status indicates activity of sub calculation engine or AXI BUS. After finishing execution, that is by getting finish command, the CNN Accelerator IP Core asserts o_rd_rdy and waits for the next execution. Repeat from asserting i_start. This is shown in Figure 2.3.

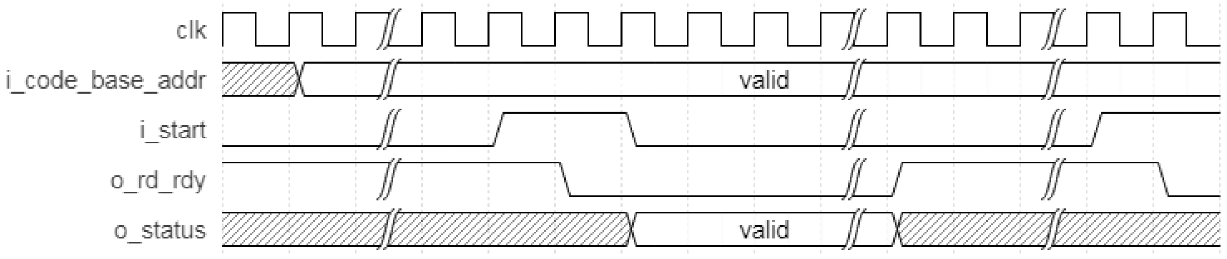


Figure 2.3. Control and Status Interface Timing Diagram

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 2.4. 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.

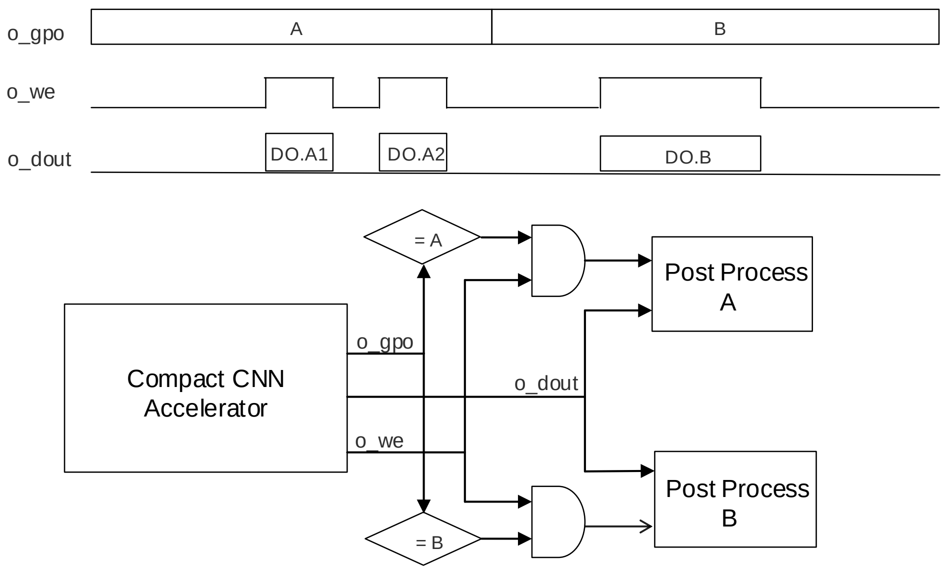


Figure 2.4 General Purpose Output Sample Application

2.2.2. Input Data Interface

Input data can be written to DRAM by external logic. In this case, loading from DRAM for input data must be in command codes. This is done by enabling the *Store Input* option in Project Implementation Window of Lattice SensAI Neural Network Compiler software. In this case, the external logic needs to store the image data to logical address 0x0f00_0000.

Also, if input data is small enough to hold internal memory, writing to DRAM and reading back to CNN Accelerator IP Core may be 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 internal memory of CNN Accelerator IP Core during idle state (o_lmmi_ready is high). The Input Data Interface is LMMI, please refer to Lattice Memory Mapped Interface and Lattice Interrupt Interface (FPGA-UG-02039) for more information on LMMI and its timing diagram. The read latecy from i_lmmi_request to o_lmmi_ready is 7 clock cycles. Memory ID and memory address are mapped to the i_lmmi_offset 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 okay. Overwriting of same address is also accepted. However, you should ensure that the input data format is satisfied before asserting i_start signal. Please refer to Input Data Format section for details.

CNN Accelerator IP Core v2.0 and earlier versions use a simple SRAM interface to input data. This is changed to LMMI starting from IP Core v2.1. For compatibility with existing designs The input data interface connection should be matched as follows:

IP Core v2.0 connections below:

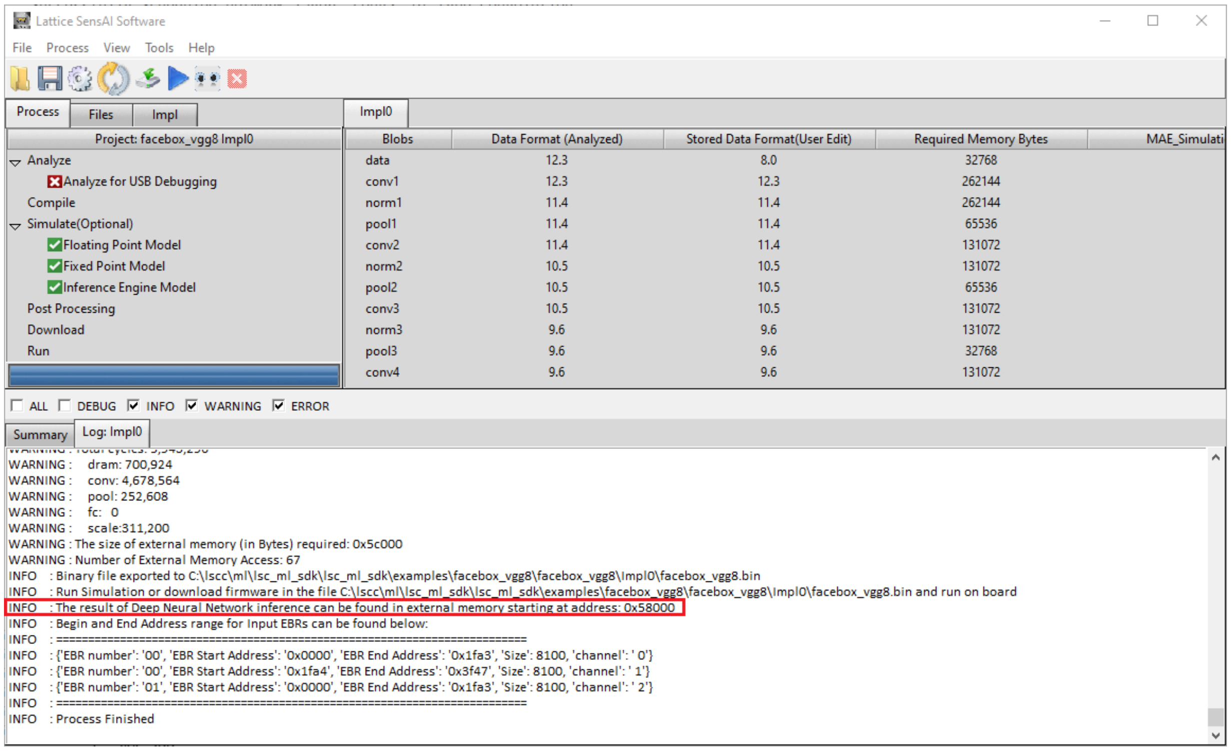
```
    .i_we          (w_we          ),
    .i_mem_sel     (w_wmemsel     ),
    .i_waddr       (w_waddr       ),
    .i_din         (w_dout        ),
```

may be mapped as follows for IP Core v2.1:

```
    .i_lmmi_request     (w_we    ),
    .i_lmmi_wr_rdn      (1'b1    ),
    .i_lmmi_offset      ({w_wmemsel, w_waddr[12:0]}),
    .i_lmmi_wdata       (w_dout  ),
    .o_lmmi_ready       (),
    .o_lmmi_rdata_valid(),
    .o_lmmi_rdata       (),
```

2.2.3. Result Interface

Result, that is, final Blob data of neural network can be written to DRAM per command code. This is done by enabling the *Store Output* option in Project Implementation window of Lattice SensAI Neural Network Compiler software. In this case, the Lattice SensAI software automatically assigns the output address in the generated command code. This information is provided as an INFO message in the output log, an example is shown in Figure 2.5. For this example, the external logic should read the result data from DRAM at logical addess 0x0F00_0000 (base address) + 0x5_8000.



Blobs	Data Format (Analyzed)	Stored Data Format(User Edit)	Required Memory Bytes	MAE_Simulati
data	12.3	8.0	32768	
conv1	12.3	12.3	262144	
norm1	11.4	11.4	262144	
pool1	11.4	11.4	65536	
conv2	11.4	11.4	131072	
norm2	10.5	10.5	131072	
pool2	10.5	10.5	65536	
conv3	10.5	10.5	131072	
norm3	9.6	9.6	131072	
pool3	9.6	9.6	32768	
conv4	9.6	9.6	131072	

```
WARNING :   dram: 700,924
WARNING :   conv: 4,678,564
WARNING :   pool: 252,608
WARNING :   fc:  0
WARNING :   scale:311,200
WARNING : The size of external memory (in Bytes) required: 0x5c000
WARNING : Number of External Memory Access: 67
INFO    : Binary file exported to C:\lscc\ml\lsc_ml_sdk\lsc_ml_sdk\examples\facebox_vgg8\facebox_vgg8\Impl0\facebox_vgg8.bin
INFO    : Run Simulation or download firmware in the file C:\lscc\ml\lsc_ml_sdk\lsc_ml_sdk\examples\facebox_vgg8\facebox_vgg8\Impl0\facebox_vgg8.bin and run on board
INFO    : The result of Deep Neural Network inference can be found in external memory starting at address: 0x58000
INFO    : Begin and End Address range for Input EBRs can be found below:
INFO    : ===========================================================================
INFO    : {'EBR number': '00', 'EBR Start Address': '0x0000', 'EBR End Address': '0x1fa3', 'Size': 8100, 'channel': ' 0'}
INFO    : {'EBR number': '00', 'EBR Start Address': '0x1fa4', 'EBR End Address': '0x3f47', 'Size': 8100, 'channel': ' 1'}
INFO    : {'EBR number': '01', 'EBR Start Address': '0x0000', 'EBR End Address': '0x1fa3', 'Size': 8100, 'channel': ' 2'}
INFO    : ===========================================================================
INFO    : Process Finished
```

Figure 2.5. Result Interface Timing Diagram

The command code can also simply feed result data to external logic through this Result interface when the *Store Output* option is disabled. Interface consists o_we as valid indicator and o_dout as 16-bit data as shown in Figure 2.6. Usually, it is a single burst series of 16-bit data. Also, it is fully programmable by command code.

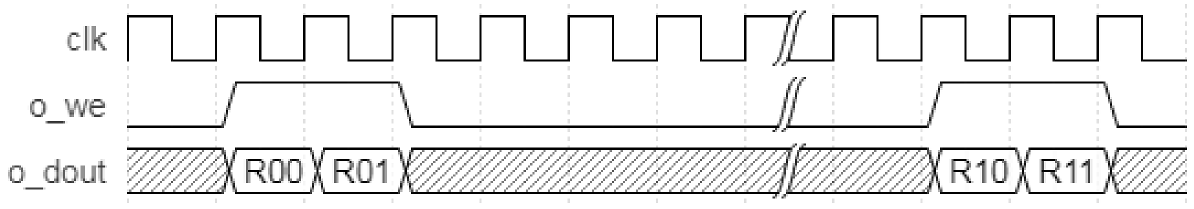


Figure 2.6. Result Interface Timing Diagram

2.2.4. DRAM Interface

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

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

2.3. Clock Domain

The clk and aclk domains are shown in Figure 2.7. The difference in clock is absorbed by Lattice Dual Clocked FIFO IP (FIFO_DC) and is implemented in AXI Rd Master and AXI Master sub-blocks.

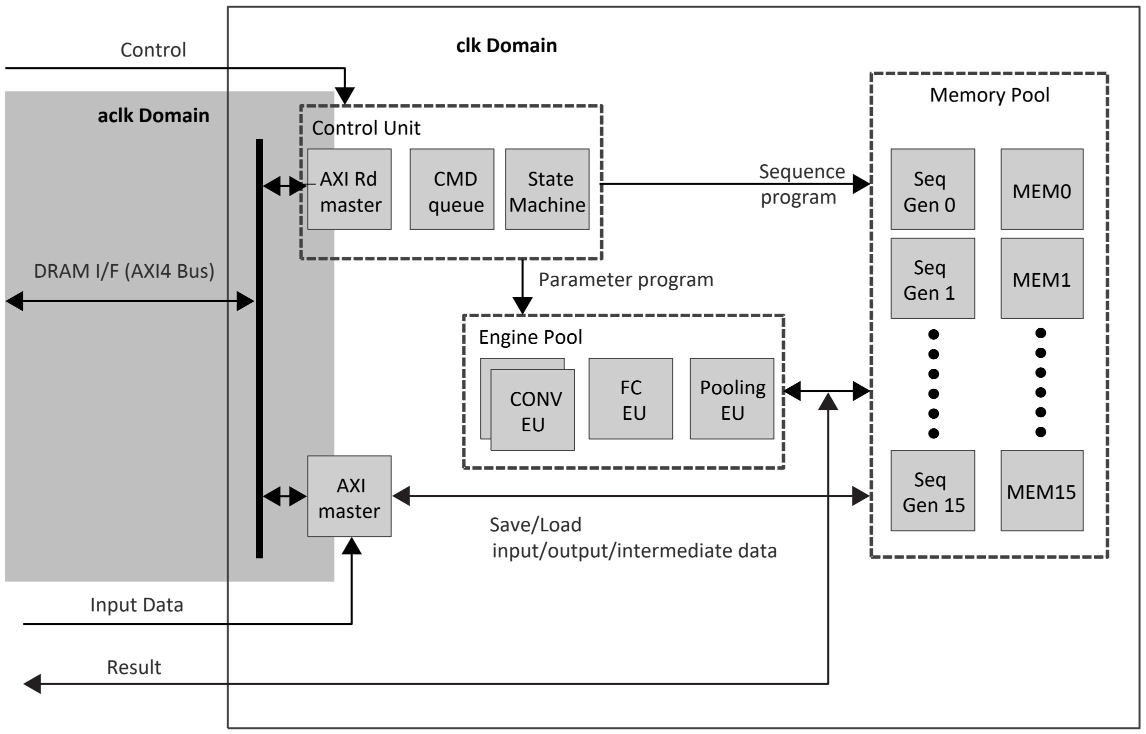


Figure 2.7 Clock Domain Diagram

2.4. 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 2.8. Not all AXI4 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 1 cycle of the slower clock between clk and aclk.

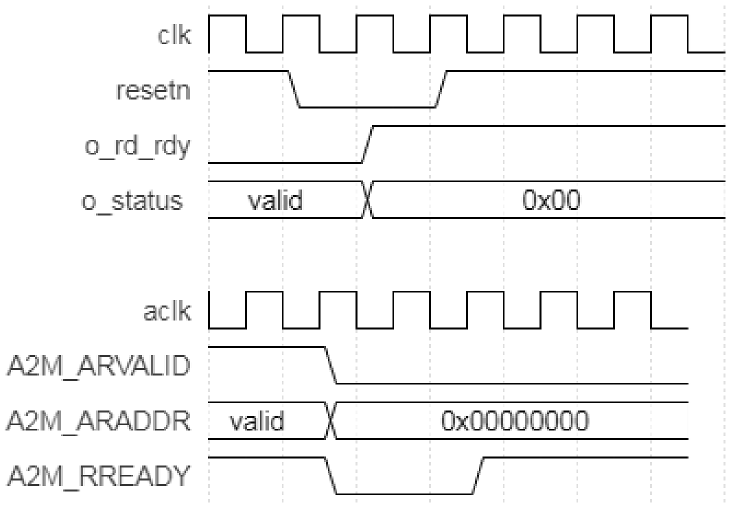


Figure 2.8. Reset Timing Diagram

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

2.5. Register Description

CNN Accelerator IP Core has no user-configurable register.

2.6. Operation Sequence

Operation sequence must be executed in the following steps:

1. Assert Reset.
2. Deassert Reset, i_start must be deasserted.
3. Write command sequence code which is generated by the Lattice SensAI Neural Network Compiler software into DRAM starting at the address specified by i_code_base_addr signal.
4. Check whether o_rd_rdy is high or not. o_rd_rdy must be high. Otherwise, go back to step 1.
5. Write input data into DRAM at the proper address, which is decided by command sequence, or directly write into the internal memory block of the CNN Accelerator IP Core through input data ports.
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 if code has direct output commands. Collect o_dout while o_we == 1.
9. Check o_rd_rdy and read result from DRAM if command code has storing result to DRAM code after o_rd_rdy going high.
10. Repeat from step 5.

2.6.1. Command Format

Command is a sequence of 32-bit data with or without additional parameters or weights as shown in Figure 2.9. It should be loaded at DRAM address specified by i_code_base_addr signal before execution. 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).

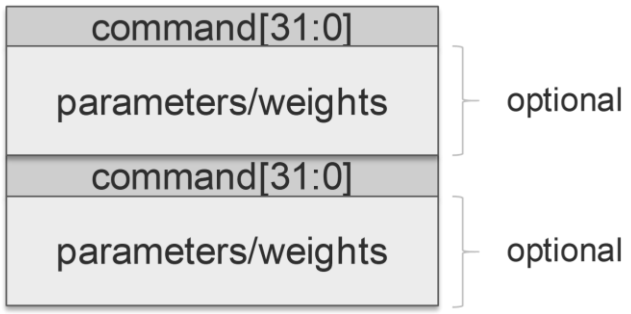


Figure 2.9. Command format

2.6.2. 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 Lattice CNN Accelerator IP Core through input data write interface. Since CNN Accelerator IP Core has only 16-bit width interface, external block should pack two of 8-bit data if 8-bit width is used for input data layer.

For example, face detection neural network may take 32 x 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 x 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 CNN 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 3.2 for details.

2.6.3. Output Data Format

Output data is a sequence of 16-bit data which is controlled by commands. 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.

2.7. Supported Commands

Command sequences are generated by Lattice SensAI Neural Network Compiler Software.

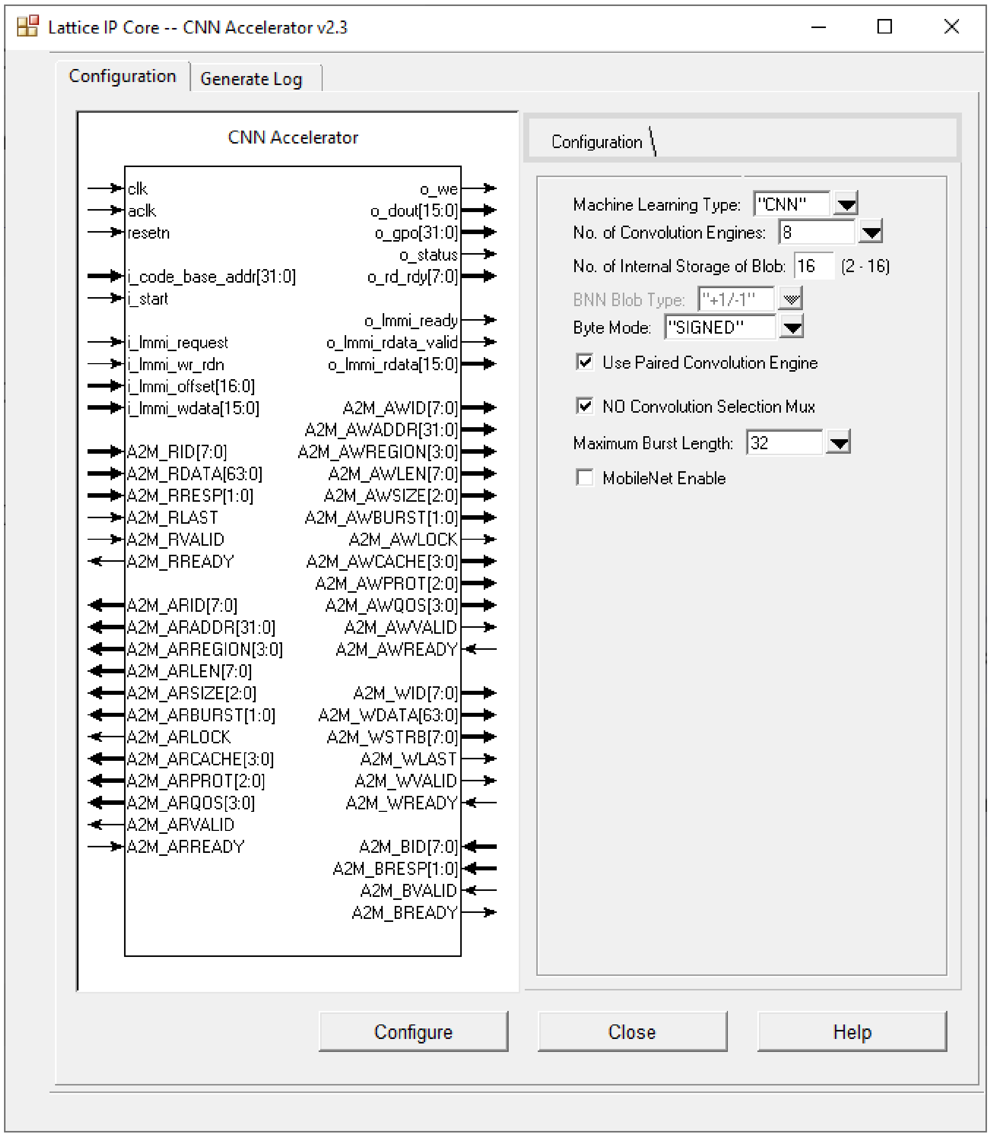
3. Parameter Settings

The IP Catalog is used to create IP and architectural modules in the Diamond software. You may 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 CNN Accelerator IP Core. The attribute values are specified using the IP core Configuration user interface in Clarity Designer 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	If MobileNet Enable is checked, selected value becomes CNN.
No. of Convolution Engines	1 - 8	8	If NO Convolution Selection Mux is Checked, Selectable Values are reduced to {1, 2, 4, 8}. If MobileNet Enable is checked, selected value becomes 8.
No. of Internal Storage of Blob	2 - 16	16	If MobileNet Enable is checked, selected value becomes 16.
BNN Blob Type	+1/-1, +1/0	+1/-1	Valid only when Machine Learning Type = *BNN*
Byte Mode	SIGNED, UNSIGNED, DISABLE	SIGNED	—
Use Paired Convolution Engine	Unchecked, Checked	Unchecked	If MobileNet Enable is checked, selected value becomes Unchecked.
NO Convolution Selection Mux	Unchecked, Checked	Unchecked	If MobileNet Enable is checked, selected value becomes Checked.
Maximum Burst Length	32, 256	32	—
MobileNet Enable	Unchecked, Checked	Unchecked	—



Machine Learning Type: "CNN"
No. of Convolution Engines: 8
No. of Internal Storage of Blob: 16 (2 - 16)
BNN Blob Type: "+1/-1"
Byte Mode: "SIGNED"
[✓] Use Paired Convolution Engine
[✓] NO Convolution Selection Mux
Maximum Burst Length: 32
[ ] MobileNet Enable

Figure 3.1. CNN Accelerator IP Core Configuration User Interface

Table 3.2. Attributes Descriptions

Attribute	Description
Machine Learning Type	This option allows you to choose between CNN engine and BNN engine. CNN engine always uses 16-bit fixed point weight, while BNN only uses 1-bit weight. Based on required performance, accuracy and available capacity, you should select Machine Learning Type.
No. of Convolution Engines	This option allows you to specify the number of convolution engines. For 1D convolution: • 1-9 taps can be performed for each engine • Up to 72 taps by chaining 8 engines For 2D Convolution: • 3 x 3 convolution per cycle per engine You should trade-off between required size and performance.
No. of Internal Storage of Blob	Each storage can store up to 16 kB. You should trade-off between required size and performance. In order to fully utilize convolution engines, number of storage must be larger than number of convolution engines. Recommend set 2x of number of convolution engines.

Attribute	Description
BNN Blob Type	Selects the type of binary blob data, either +1/-1 or +1/0. This setting should be matched to the Lattice Neural Network Compiler.
Byte Mode	Specifies the byte mode of input data. SIGNED – input data is signed 16-bit/8-bit data, similar to v1.1. 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.
Use Paired Convolution Engine	Enables the use of paired convolution engine. Unchecked – Not use paired convolution engine, similar to v1.1. Checked – Use paired convolution engine. The total number of convolution engine is double of the No. of Convolution Engines value.
NO Convolution Selection Mux	Disables the use of convolution selection multiplexor. Unchecked – Use of convolution selection multiplexor, similar to v1.1. Checked – Use dedicated connection instead of multiplexor.
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.
MobileNet Enable	Enables MobileNet mode. Unchecked – Not use MobileNet mode, similar to v2.0. Checked – MobileNet mode. MobileNet improves 1x1 convolution and depthwise convolution up to 8x at the cost of more LUT and slightly reduced Fmax. This option should not be Checked when the neural network does not have 1x1 convolution and depthwise convolution.

Any combination of Use Paired Convolution Engine and NO Convolution Selection Mux settings are supported. It is recommended to use NO Convolution Selection Mux=Checked because it uses less LUT with no negative side effect. However, this is only supported in SensAI v2.0. The NO Convolution Selection Mux=Unchecked is for backward compatibility.

The Use Paired Convolution Engine has its pros and cons:

- Pro – Enhances performance of convolution calculation
- Con – Increase DSP and LUT consumption and may reduce operation frequency of the core clock

You should carefully choose setting based on resource and calculation requirement. For example, if neural network does not have much convolution calculation, overall performance may be reduced when using Use Paired Convolution Engine=Checked due to slower clock. The summary of Use Paired Convolution Engine and NO Convolution Selection Mux setting combination is shown in Table 3.3.

Table 3.3 Combinations of Use Paired Convolution Engine and NO Convolution Selection Mux Settings

Use Paired Convolution Engine	NO Convolution Selection Mux	Description
Unchecked	Unchecked	Backward compatible mode. Use for existing firmware and SensAI 1.x.
Unchecked	Checked	Recommended for SensAI 2.0 or later.
Checked	Unchecked	Not recommended.
Checked	Checked	Enhances performance of convolution calculation at the cost of increase in DSP and LUT utilization. Note that this setting may also reduce clock frequency.

4. IP Generation and Evaluation

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

4.1. Licensing the IP

An IP core-specific device-specific license string is required to enable full, unrestricted use of the Lattice CNN 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 http://www.latticesemi.com/Products/DesignSoftwareAndIP.aspx.

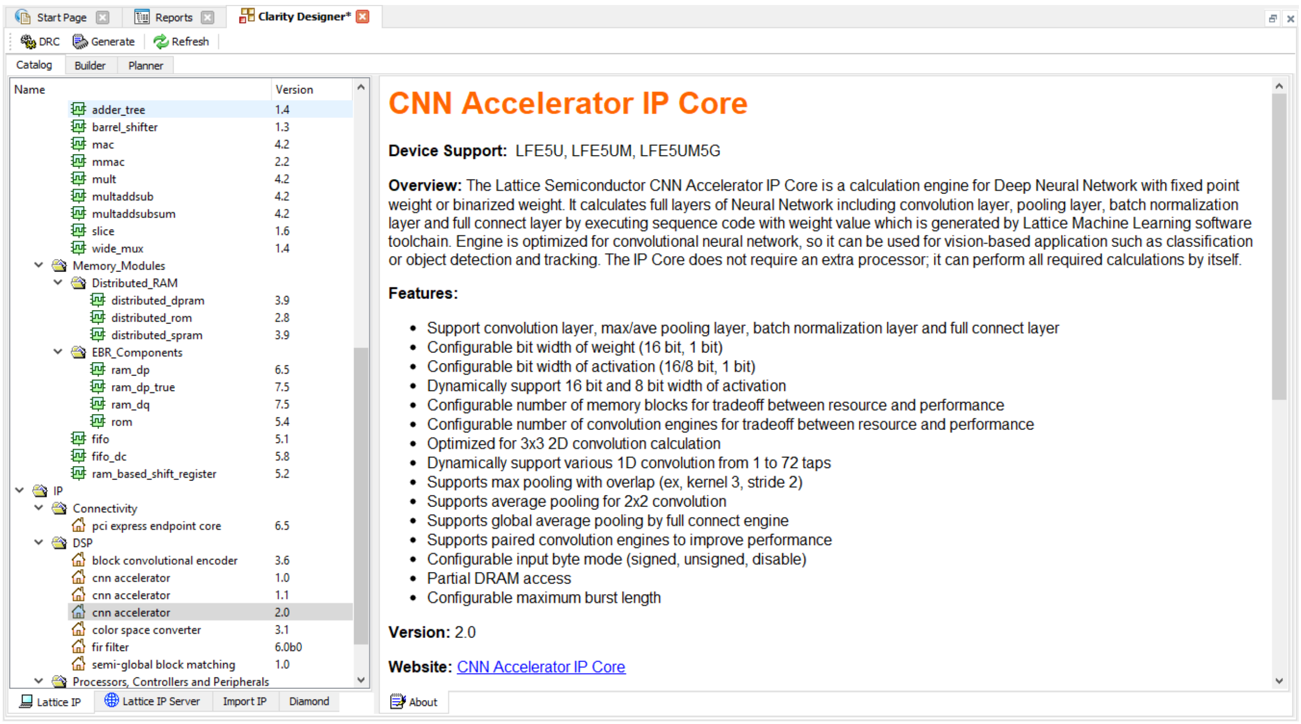
You may download and generate the CNN Accelerator IP Core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license string. The CNN Accelerator IP Core supports Lattice's IP hardware evaluation capability, which makes it possible 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 enable timing simulation to open the design in the Diamond software, and 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.

4.2. Generation and Synthesis

4.2.1. Getting Started

The CNN Accelerator IP Core is available for download in the Lattice IP Server using the Diamond Clarity Designer tool. The IP files can be automatically installed using InstallShield® technology in any customer-specified directory. After the IP core is installed, the IP core is listed in the Catalog tab of the Clarity Designer user interface, under DSP category as shown in Figure 4.1.



Name	Version
adder_tree	1.4
barrel_shifter	1.3
mac	4.2
mmac	2.2
mult	4.2
multaddsub	4.2
multaddsubsum	4.2
slice	1.6
wide_mux	1.4
Memory_Modules	
Distributed_RAM	
distributed_dpram	3.9
distributed_rom	2.8
distributed_spram	3.9
EBR_Components	
ram_dp	6.5
ram_dp_true	7.5
ram_dq	7.5
rom	5.4
fifo	5.1
fifo_dc	5.8
ram_based_shift_register	5.2
IP	
Connectivity	
pci express endpoint core	6.5
DSP	
block convolutional encoder	3.6
cnn accelerator	1.0
cnn accelerator	1.1
cnn accelerator	2.0
color space converter	3.1
fir filter	6.0b0
semi-global block matching	1.0
Processors, Controllers and Peripherals	

CNN Accelerator IP Core

Device Support: LFE5U, LFE5UM, LFE5UM5G

Overview: The Lattice Semiconductor CNN Accelerator IP Core is a calculation engine for Deep Neural Network with fixed point weight or binarized 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 Machine Learning software toolchain. 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.

Features:

- Support convolution layer, max/ave pooling layer, batch normalization layer and full connect layer
- Configurable bit width of weight (16 bit, 1 bit)
- Configurable bit width of activation (16/8 bit, 1 bit)
- Dynamically support 16 bit and 8 bit width of activation
- Configurable number of memory blocks for tradeoff between resource and performance
- Configurable number of convolution engines for tradeoff between resource and performance
- Optimized for 3x3 2D convolution calculation
- Dynamically support various 1D convolution from 1 to 72 taps
- Supports max pooling with overlap (ex, kernel 3, stride 2)
- Supports average pooling for 2x2 convolution
- Supports global average pooling by full connect engine
- Supports paired convolution engines to improve performance
- Configurable input byte mode (signed, unsigned, disable)
- Partial DRAM access
- Configurable maximum burst length

Version: 2.0

Website: CNN Accelerator IP Core

Figure 4.1. CNN Accelerator IP Core in Clarity Designer Catalog Tab

4.2.2. Configuring the IP Core in Clarity

The CNN Accelerator IP Core should be configured and generated before it can be used in the Diamond project. This is done through the CNN Accelerator IP Core configuration user interface (see Figure 3.1); it provides options for setting the desired parameters and invoking the IP core generator.

To configure the CNN Accelerator IP Core:

1. Double-click the CNN Accelerator IP Core in the Catalog tab of the Diamond Clarity tool.
2. Specify the following in the dialog box:
   - Instance Path
   - Instance Name

   Note: All generated files are placed under the directory <Instance Path>/<Instance Name>.
3. Enter configuration parameters.
4. Click **Customize.** This closes the dialog box and launches the CNN Accelerator IP Core configuration user interface.
5. Enter the desired parameters in the CNN Accelerator IP Core configuration user interface. Ensure that same parameters are used in the Lattice Neural Network Compiler to be able to generate compatible command sequence code.
6. Click the **Generate** button, and close the user interface to generate the IP core and supporting files.

Table 4.1 provides a list of key files and directories created by the IPexpress tool and how they are used.

Table 4.1. File List

Attribute	Description
<Instance Name>.lpc	This file contains the tool options used to recreate or modify the core in the Clarity tool.
<Instance Name>.ngo	This file provides the synthesized IP core.
<Instance Name>_bb.v	This file provides the synthesis black box for the user's synthesis.
<Instance Name>_inst.v	This file provides an instance template for the IP core.
<Instance Name>_top.v	This file provides an example RTL top file that instantiates the IP core.
beh_rtl.v*	This file provides cycle-accurate simulation model for the IP core. The top level module name in this file is <Instance Name>.
generate_core.tcl	This file is created when the user interface Generate button is pushed. This file may be run from command line.
<Instance Name>_generate.log	This is the synthesis and map log file.
<Instance Name>_gen.log	This is the IP Core generation log file.

***Note:** This file is located in the following path:
<Instance Path>/<Instance Name>/ml_engine_ecp5_eval/<Instance Name>/src/beh_rtl/<device>

4.2.3. Instantiating the IP Core

The generated CNN Accelerator IP Core package includes black-box (<Instance Name>_bb.v) and instance (<Instance Name>_inst.v) templates that can be used to instantiate the core in a top-level design. A sample of RTL top-level reference source file (<Instance Name>_top.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.

4.3. Running Functional Simulation

The CNN Accelerator IP Core does NOT contain a sample test bench for performing simple simulation test. However, a cycle-accurate simulation model (beh_rtl.v) is provided as shown in Table 4.1. This may be instantiated in a test bench. To successfully elaborate the simulation model:

1. Instantiate GSR and PUR in testbench top RTL file.
   The following codes may be copied and paste to testbench top RTL file:

   ```
   PUR PUR_INST(<active_low_reset_signal>);
   GSR GSR_INST(<active_low_reset_signal>);
   ```

2. Compile the simulation primitives of selected device. These are located in the following path:
   <Diamond Install Path>/cae_library/simulation/verilog/<device>

4.4. Hardware Evaluation

The CNN 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.

5. Ordering Part Number

The Ordering Part Numbers (OPN) for CNN Accelerator IP Core targeting ECP5 and ECP5-5G FPGA devices are the following:

- CNN-ACCEL-E5-U – Project License
- CNN-ACCEL-E5-UT – Site License

References

- ECP5 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 the ECP5UM using Lattice Diamond 3.11.0.396.4. The following settings are used in generating this data. The new attributes are default for this setting.

- Synthesis Tool – Synplify Pro
- Device Part Number – LFE5UM-85F-8BG756I
- BNN Blob Type – +1/0 (For Machine Learning Type: BNN)

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

No. of Internal Storage of Blob[2]	Byte Mode	Use Paired Conv. Engine	NO Conv. Selection Mux	Max Burst Length	Register	LUTs	Slices	DSP MULT	DSP ALU	Block RAMs	clk Fmax[3] (MHz)
Machine Learning Type: CNN											
2	Signed	Unchecked	Unchecked	32	3888	5642	3981	26	2	26	132.679
8	Signed	Unchecked	Unchecked	32	9646	14642	10600	80	2	77	127.081
16	Signed	Unchecked	Unchecked	32	17419	27609	19314	152	2	145	121.566
16	Unsigned	Unchecked	Unchecked	32	16870	27648	19099	152	2	145	118.864
16	Disable	Unchecked	Unchecked	32	16413	24683	17541	152	2	145	124.116
16	Signed	Unchecked	Checked	32	17800	26651	19000	152	2	145	127.763
16	Signed	Checked	Checked	32	22937	33675	24057	296	2	153	123.335
16	Signed	Unchecked	Unchecked	256	17498	27520	19316	152	2	145	122.220
Machine Learning Type: CNN (MobileNet Enabled)											
16	Signed	Unchecked	Checked	32	18915	29091	20496	152	2	145	124.008
Machine Learning Type: BNN											
2	Signed	Unchecked	Unchecked	32	3565	4982	3561	0	0	25	137.099
8	Signed	Unchecked	Unchecked	32	9808	15200	10730	0	0	76	129.116
16	Signed	Unchecked	Unchecked	32	17989	28541	19941	0	0	144	120.511
16	Unsigned	Unchecked	Unchecked	32	18048	28174	19842	0	0	144	125.156
16	Disable	Unchecked	Unchecked	32	17711	25882	18384	0	0	144	126.008
16	Signed	Unchecked	Checked	32	18080	27477	19399	0	0	144	105.496
16	Signed	Checked	Checked	32	18107	27908	19621	0	0	144	117.028
16	Signed	Unchecked	Unchecked	256	17996	28523	19942	0	0	144	126.008

Notes:

1. Performance may vary when using a different software version or targeting a different device density or speed grade.
2. It is recommended to use No. of Internal Storage of Blob = 2 * No. of Convolution Engines.
3. Fmax is generated when the FPGA design only contains the CNN Accelerator IP Core. These values may be reduced when user logic is added to the FPGA design.

Revision History

Revision 2.2, December 2020

Section	Change Summary
Acronyms in This Document	Added this section.
Introduction	• Added General Purpose Output feature in Features section. • Updated Table 1.1.
Functional Descriptions	• Added o_gpo signal in Figure 2.2 and Table 2.1. • Added General Purpose Output section.
Parameter Settings	Updated Figure 3.1.
References	Updated this section.

Revision 2.1, October 2019

Section	Change Summary
Introduction	Added MobileNet feature in Features section.
Interface Descriptions	• Updated interfaces in Table 2.1. Added more information in Result Interface section. • Updated Input Data Interface and DRAM Interface sections for interface change.
Parameter Settings	Updated Table 3.1. Attributes Table, Table 3.2. Attributes Descriptions and Figure 3.1. CNN Accelerator IP Core Configuration User Interface for the new attributes.
IP Generation and Evaluation	Updated Figure 4.1 in Getting Started section.
Appendix A. Resource Utilization	Updated Table A.1. Performance and Resource Utilization[1] per result of CNN Accelerator IP Core v2.1.

Revision 2.0, May 2019

Section	Change Summary
All	• Added Disclaimers section. • Updated last page of the document.
Introduction	Added new features in Features section.
Parameter Settings	Updated Table 3.1. Attributes Table, Table 3.2. Attributes Descriptions and Figure 3.1. CNN Accelerator IP Core Configuration User Interface for the new attributes.
IP Generation and Evaluation	Updated Figure 4.1 in Getting Started section.
Appendix A. Resource Utilization	Updated Table A.1. Performance and Resource Utilization[1] per result of CNN Accelerator IP Core v2.0.

Revision 1.1, September 2018

Section	Change Summary
Functional Description	Updated Overview section. Updated status signal description in Table 2.1. CNN Accelerator IP Core Signal Descriptions.
Appendix A. Resource Utilization	Updated Table A.1. Performance and Resource Utilization[1] per result of CNN Accelerator IP Core v1.1.
Revision History	Updated revision history table to new template.

Revision 1.0, May 2018

Section	Change Summary
All	Initial release