altera
An Intel Company

FPGA AI Suite

IP Reference Manual

Updated for FPGA AI Suite: **2024.3**

Online Version
Send Feedback

768974
2025.02.24

Contents

1. FPGA AI Suite IP Reference Manual 4

2. About the FPGA AI Suite IP 5

2.1. Supported Models 7
2.1.1. MobileNet V2 differences between Caffe and TensorFlow models 7
2.2. Model Performance 7
2.2.1. Throughput on the MobileNetV1 model (and other very fast models) 12
2.3. Software Emulation of the FPGA AI Suite IP 12
2.4. FPGA AI Suite Layer / Primitive Ranges 13
2.5. FPGA AI Suite IP Block Configuration 14
2.5.1. Architecture Description File Format for Instance Parameterization 16
2.5.2. Architecture Description File Parameters 17
2.6. IP Block Interfaces 31
2.6.1. Clock and Reset 32
2.6.2. AXI Interfaces 32
2.6.3. AXI Interface Clock and Reset 32
2.6.4. Input Feature Tensor In-Memory Format 33
2.6.5. Output Tensor In-Memory Format 37
2.7. Feature Input and Output Streaming 40
2.7.1. Input Streaming 40
2.7.2. Output Streaming 42
2.8. DDR-Free Operation 43

3. FPGA AI Suite IP Generation Utility 45

3.1. IP Generation Utility Execution Flows 45
3.2. IP Generation Utility Inputs 47
3.3. IP Generation Utility Outputs 47
3.4. IP Generation Utility Command Line Options 47
3.4.1. The `--flow create_ip` Flow 49
3.4.2. The `--flow add_arch` Flow 50
3.4.3. The `--flow list` Flow 51
3.4.4. The `--flow remove_arch` Flow 51

4. FPGA AI Suite Ahead-of-Time Splitter Utility 52

4.1. Files Generated by the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility 52
4.2. Building the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility 53
4.3. Running the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility 53
4.4. FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility Example Application 55

5. CSR Map and Descriptor Queue 56

5.1. Discovery ROM 56
5.2. Interrupt Control 57
5.3. DMA Descriptor Queue 57
5.4. DMA Control Registers 58
5.5. Performance Registers 59
5.6. Debug Network Registers 59
5.7. DMA License Register 60
5.8. DMA Transaction Counters 60

A. FPGA AI Suite IP Reference Manual Archives.....61

B. FPGA AI Suite IP Reference Manual Document Revision History.....62

1. FPGA AI Suite IP Reference Manual

The *FPGA AI Suite IP Reference Manual* provides an overview of the FPGA AI Suite IP and the parameters that you can set to customize the IP. This document also covers the FPGA AI Suite IP generation utility.

About the FPGA AI Suite Documentation Library

Documentation for the FPGA AI Suite is split across a few publications. Use the following table to find the publication that contains the FPGA AI Suite information that you are looking for:

Table 1. FPGA AI Suite Documentation Library

Title and Description	
Release Notes Provides late-breaking information about the FPGA AI Suite including new features, important bug fixes, and known issues.	Link
Getting Started Guide Get up and running with the FPGA AI Suite by learning how to initialize your compiler environment and reviewing the various design examples and tutorials provided with the FPGA AI Suite	Link
IP Reference Manual Provides an overview of the FPGA AI Suite IP and the parameters you can set to customize it. This document also covers the FPGA AI Suite IP generation utility.	Link
Compiler Reference Manual Describes the use modes of the graph compiler (`dla_compiler`). It also provides details about the compiler command options and the format of compilation inputs and outputs.	Link
PCIe-based Design Example User Guide Describes the design and implementation for accelerating AI inference using the FPGA AI Suite, Intel® Distribution of OpenVINO™ toolkit, and a Terasic* DE10-Agilex Development Board.	Link
SoC-based Design Example User Guide Describes the design and implementation for accelerating AI inference using the FPGA AI Suite, Intel Distribution of OpenVINO toolkit, and an Arria® 10 SX SoC FPGA Development Kit (DK-SOC-10AS066S) or Agilex™ 7 FPGA I-Series Transceiver-SoC Development Kit.	Link

Intel Distribution of OpenVINO toolkit Requirement

To use the FPGA AI Suite, you must be familiar with the Intel Distribution of OpenVINO toolkit.

FPGA AI Suite Version 2024.3 requires the Intel Distribution of OpenVINO toolkit Version 2023.3 LTS. For OpenVINO documentation, refer to https://docs.openvino.ai/2023.3/documentation.html.

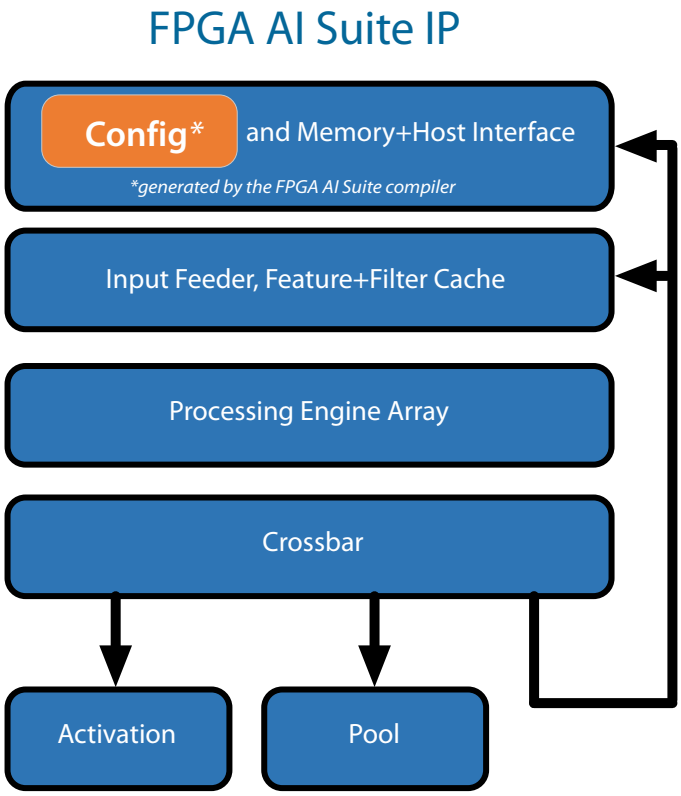
2. About the FPGA AI Suite IP

The FPGA AI Suite IP is an RTL-instantiable configurable IP with AXI interfaces that you can instantiate into a generic embedded FPGA system.

The IP is configured through parameters defined in an Architecture Description File. The Architecture Description File, along with the OpenVINO intermediate representation of your trained model, is compiled by the FPGA AI Suite compiler into configuration instructions for the IP.

The following diagram shows a high-level architecture of the FPGA AI Suite IP.

Figure 1. High-Level Architecture of the FPGA AI Suite IP

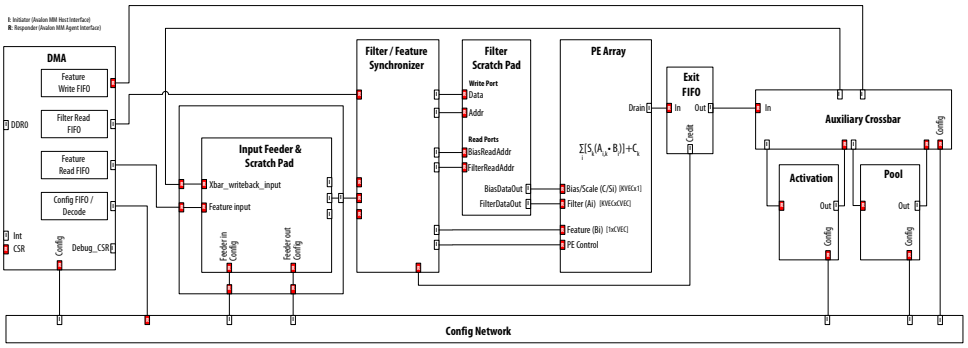


The primary parameters defined in an Architecture Description File cover the following properties:

- PE array vectorization
- Scratch pad sizing
- External memory bus bandwidth
- Types/vectorization of auxiliary layer blocks

The following diagram is an architecture diagram for a specific instantiation of the FPGA AI Suite IP. The blocks connected to the crossbar in this diagram are examples. The selection of blocks connected to the crossbar are determined by compile time parameters.

Figure 2. Architecture of an Example Instantiation of the FPGA AI Suite IP



Two teams are typically involved in the implementation of an AI feature:

- A machine learning (ML) team responsible for developing and delivering an AI model.
- An FPGA team responsible for integrating the FPGA AI Suite IP and runtime together into a system.

Defining the IP architecture straddles the boundary between these two teams. The ML team must develop an AI model that meets the target performance in some parameterization of the configurable IP. The FPGA team must ensure it fits onto the FPGA and closes timing.

Although responsibility for defining the parameterization of the configurable architecture can lie with either team (but is a joint responsibility), it might be easiest for the ML team to define the architecture.

The team responsible for defining the IP parameterization can use the FPGA AI Suite compiler (`dla_compiler` command) area and performance estimator tools to guide their decisions. The *FPGA AI Suite Compiler Reference Manual* describes how to use the `dla_compiler` tool.

In addition to the FPGA team and the ML team, another team is likely responsible for the software integration on the host processor. Depending on the system details, this software is likely responsible for interfacing with OpenVINO and communicating (via the BSP) with the FPGA AI Suite IP. This software will likely be based on the runtime system that is included with the PCIe Example Design or possibly based on an SOC Example Design.

2.1. Supported Models

The FPGA AI Suite supports the following pretrained models from OpenVINO Model Zoo:

Table 2. OpenVINO Model Zoo 2023.3 LTS

Model Zoo 2023.3 LTS path	Model	Framework
`public/mobilenet-v1-1.0-224`	MobileNet V1	Caffe
`public/mobilenet-v2`	MobileNet V2	Caffe
`public/mobilenet-v2-1.4-224`	MobileNet V2	TensorFlow
`public/mobilenet-v3-large-1.0-224-tf`	MobileNet V3	TensorFlow
`public/resnet-50-tf`	ResNet-50	TensorFlow
`intel/unet-camvid-onnx-0001`	UNet	PyTorch
`public/yolo-v3-tf`	YOLO v3	TensorFlow
`public/yolo-v3-tiny-tf`	TinyYOLO v3	TensorFlow
	Yolo v8 (all heads)	PyTorch
`public/squeezenet1.1`	SqueezeNet v1.1	Caffe
`public/i3d_rgb_tf`	Inflated 3D (I3D)	TensorFlow
	Multilayer Perceptrons (MLPs)	

After installation, the location `$COREDLA_ROOT/example_graphs/MLP/` contains an example MLP graph.

In addition, the FPGA AI Suite supports certain signed INT8 symmetric quantized models that use neural network compression framework (NNCF) flows.

Customized models that are similar to the supported models or are derived from the above models are not supported, although in some cases they might work without modification.

The supplied example architectures (or *IP Configurations*) support all of the above models, except for the `Small` and `Small_Softmax` architectures that support only ResNet-50, MobileNet V1, and MobileNet V2.

2.1.1. MobileNet V2 differences between Caffe and TensorFlow models

There are two inverted bottlenecks (group of expand, depthwise, projection) in which TensorFlow has already gone down to 14x14 while Caffe is still at 28x28. This is the only place where the structure of the graph differs. TensorFlow also uses ReLU6, implemented with a clamp in the FPGA AI Suite IP, while Caffe uses ReLU.

2.2. Model Performance

The performance estimator tool (described in the *FPGA AI Suite Compiler Reference Manual*) assumes the following f_{MAX} values for FPGA devices:

- Arria 10: 265 MHz
- Agilex 7: 400 MHz

These assumptions are reasonable and conservative for the standard speed bin. As shown by the results in this section, the achieved f_{MAX} of the example design typically exceeds these assumptions.

The performance results for the designs that follow were achieved using the `dla_build_example_design.py` script that is included with the FPGA AI Suite. The script uses a standard (-2) speed bin with a single seed and uses high-effort compiler settings.

The runtime hosts used for determining the performance results are as follows:

- **Agilex 7 runtime host:** SUSE Linux Enterprise Server 15 host on an Intel Xeon® processor E5-1650 @ 3.5 GHz.

This design uses a dedicated DDR interface for the IP. The batch size is 1. Performance varies based on the clock speed, the DDR latency and bandwidth.

The `dla_build_example_design.py` script includes the following two `.qsf` lines to enable non-default Quartus® Prime options during design compilation:

```
set_global_assignment -name ALLOW_SHIFT_REGISTER_MERGING_ACROSS_HIERARCHIES
ALWAYS
set_global_assignment -name DISABLE_REGISTER_MERGING_ACROSS_HIERARCHIES OFF
```

The architectures in the tables that follow are in the `$COREDLA_ROOT/example_architectures/` directory. Review the README file in that directory for information about each architecture.

The **IP Throughput** column in the tables that follow shows the performance for the portion of the graph that runs on the FPGA device. In many cases, the entire graph runs on the FPGA device. The IP Throughput is representative of performance if the IP is used in a hostless configuration.

The **IP+host Throughput** column in the tables that follow shows the performance including the host. The IP+host performance may be lower than IP-only performance if the host is unable to stream data to the FPGA device quickly enough, or if the host is limited by some of the processing associated with the graph (for example, the host performs NMS for the YOLOv3 graph). Achievable IP+host performance depends on the speed and loading of the host and the FPGA AI Suite IP.

Details - FPGA AI Suite 2024.3

Architecture	f_{MAX}	ALMs	DSPs	M20Ks	Registers
AGX7_FP16_Generic	600 MHz	33.6 k	186	511	95 k
AGX7_FP16_Performance	605 MHz	103.9 k	1162	1533	324 k
AGX7_Small_NoSoftmax	610 MHz	17.2 k	80	296	49 k
AGX7_Small_Softmax	616 MHz	18.6 k	90	304	57 k
AGX7_Generic	600 MHz	38.9 k	202	778	113 k
AGX7_Performance	585 MHz	70.5 k	650	1278	207 k
AGX7_Performance_Giant	535 MHz	127.8 k	1546	2371	359 k

public/mobilenet-v1-1.0-224

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	2261	171	171	71.2	89.5
AGX7_FP16_Performance	103.9 k	1162	9117	572	567	71.2	89.5
AGX7_Small_NoSoftmax	17.2 k	80	2770	167	167	70.9	89.6
AGX7_Small_Softmax	18.6 k	90	2796	169	168	70.9	89.5
AGX7_Generic	38.9 k	202	3306	255	251	70.9	89.5
AGX7_Performance	70.5 k	650	8893	566	399	70.9	89.5
AGX7_Performance_Giant	127.8 k	1546	8987	1483	764	71.0	89.6

public/mobilenet-v2

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	3653	148	147	71.8	89.6
AGX7_FP16_Performance	103.9 k	1162	6948	372	367	71.8	89.6
AGX7_Small_NoSoftmax	17.2 k	80	4609	141	138	71.6	89.6
AGX7_Small_Softmax	18.6 k	90	4645	142	139	71.8	89.4
AGX7_Generic	38.9 k	202	2720	203	198	71.8	89.4
AGX7_Performance	70.5 k	650	7166	343	276	71.7	89.4
AGX7_Performance_Giant	127.8 k	1546	6370	1081	726	71.8	89.4

public/mobilenet-v2-1.4-224

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	4085	122	121	74.8	91.9
AGX7_FP16_Performance	103.9 k	1162	8717	290	288	74.8	91.9
AGX7_Generic	38.9 k	202	4184	151	145	74.7	91.8
AGX7_Performance	70.5 k	650	8716	290	226	74.7	91.8
AGX7_Performance_Giant	127.8 k	1546	7539	847	618	74.7	91.7

(*) **DDR** is estimated minimum average read + write (that is, read + write require at least this much bandwidth on average). Peak bandwidth is higher.

public/mobilenet-v3-large-1.0-224-tf

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	3774	169	165	75.8	92.1
AGX7_FP16_Performance	103.9 k	1162	11260	240	234	75.8	92.1
AGX7_Generic	38.9 k	202	4530	181	174	72.3	90.7
AGX7_Performance	70.5 k	650	11293	246	201	72.1	90.5
AGX7_Performance_Giant	127.8 k	1546	8492	355	304	72.6	90.6

public/resnet-50-tf

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	3005	32	32	76.8	92.9
AGX7_FP16_Performance	103.9 k	1162	11715	166	164	76.8	92.9
AGX7_Small_NoSoftmax	17.2 k	80	5935	28	28	77.0	92.9
AGX7_Small_Softmax	18.6 k	90	5989	28	28	77.1	92.9
AGX7_Generic	38.9 k	202	4206	60	60	77.1	92.9
AGX7_Performance	70.5 k	650	11540	163	143	76.9	92.9
AGX7_Performance_Giant	127.8 k	1546	8067	237	229	76.9	92.8

Resnet50 v1 (Caffe)

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	2822	38	38	74.4	91.4
AGX7_FP16_Performance	103.9 k	1162	12139	195	195	74.4	91.4
AGX7_Small_NoSoftmax	17.2 k	80	4161	37	37	74.1	91.4
AGX7_Small_Softmax	18.6 k	90	4203	37	37	74.2	91.3
AGX7_Generic	38.9 k	202	4489	73	73	74.2	91.3
AGX7_Performance	70.5 k	650	12119	195	162	74.0	91.4
AGX7_Performance_Giant	127.8 k	1546	8379	270	247	74.1	91.4

intel/unet-camvid-onnx-0001

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]
AGX7_FP16_Generic	33.6 k	186	825	1.09
AGX7_FP16_Performance	103.9 k	1162	4552	7.57
AGX7_Small_NoSoftmax	17.2 k	80	1140	1.10

continued...

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]
AGX7_Small_Softmax	18.6 k	90	1153	1.11
AGX7_Generic	38.9 k	202	1319	2.14
AGX7_Performance	70.5 k	650	4331	7.36
AGX7_Performance_Giant	127.8 k	1546	5426	11.71

public/yolo-v3-tf

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Detection mAP @0.5	Detection mAP @0.5:0.95
AGX7_FP16_Generic	33.6 k	186	1428	4.2	4	62.27	31.58
AGX7_FP16_Performance	103.9 k	1162	6347	27.9	28	62.25	31.58
AGX7_Generic	38.9 k	202	1901	8.2	8	62.28	31.49
AGX7_Performance	70.5 k	650	6170	27.0	11	62.22	31.47
AGX7_Performance_Giant	127.8 k	1546	6634	40.5	30	62.25	31.46

public/yolo-v3-tiny-tf

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Detection mAP @0.5	Detection mAP @0.5:0.95
AGX7_FP16_Generic	33.6 k	186	1200	41	36	35.79	14.77
AGX7_FP16_Performance	103.9 k	1162	4680	116	113	35.81	14.78
AGX7_Generic	38.9 k	202	2433	82	66	35.76	14.74
AGX7_Performance	70.5 k	650	4647	115	40	35.73	14.72
AGX7_Performance_Giant	127.8 k	1546	5028	109	64	35.81	14.75

public/yolo-v8-nano detection

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Detection mAP @0.5	Detection mAP @0.5:0.95
AGX7_FP16_Performance	103.9 k	1162	6728	94	91	51.15	36.52
AGX7_Generic	38.9 k	202	2427	50	39	51.14	36.50
AGX7_Performance	70.5 k	650	6720	95	32	51.10	36.48

public/yolo-v8-nano classification

Architecture	ALMs	DSPs	DDR(*) [MB/s]	Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Performance	103.9 k	1162	10345	1384	67.92	87.72
AGX7_Generic	38.9 k	202	5489	943	67.96	87.86
AGX7_Performance	70.5 k	650	10178	1358	67.72	87.72

public/squeezenet1.1

Architecture	ALMs	DSPs	DDR(*) [MB/s]	IP Throughput [fps]	IP+host Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	631	218	219	58.5	81.1
AGX7_FP16_Performance	103.9 k	1162	4679	940	886	58.5	81.1
AGX7_Small_NoSoftmax	17.2 k	80	923	220	219	58.5	81.0
AGX7_Small_Softmax	18.6 k	90	933	222	222	58.5	81.0
AGX7_Generic	38.9 k	202	1722	535	536	58.5	81.0
AGX7_Performance	70.5 k	650	4654	932	419	58.4	81.0
AGX7_Performance_Giant	127.8 k	1546	3631	951	735	58.3	81.1

public/i3d_rgb_tf

Architecture	ALMs	DSPs	DDR(*) [MB/s]	Throughput [fps]	Top-1 [%]	Top-5 [%]
AGX7_FP16_Generic	33.6 k	186	442	0.61	65.79	82.89
AGX7_FP16_Performance	103.9 k	1162	2562	4.14	65.79	82.89
AGX7_Small_NoSoftmax	17.2 k	80	492	0.58	65.35	82.89
AGX7_Small_Softmax	18.6 k	90	496	0.59	65.57	82.89
AGX7_Generic	38.9 k	202	742	1.36	65.57	83.11
AGX7_Performance	70.5 k	650	2486	4.01	65.13	83.11
AGX7_Performance_Giant	127.8 k	1546	2839	4.64	65.79	82.89

2.2.1. Throughput on the MobileNetV1 model (and other very fast models)

Due to the high system throughput, the MobileNetV1 performance with large IP instances is strongly dependent on the host.

2.3. Software Emulation of the FPGA AI Suite IP

The FPGA AI Suite includes a compiled software model of the FPGA AI Suite IP that is bit-accurate(*). The emulation of the FPGA AI Suite IP is accessible through the OpenVINO plugin interface. This emulation models the numeric details of the IP, including the behavior of the block floating point numerics (when used).

The OpenVINO emulation plugin is enabled in the `$COREDLA_ROOT/bin/plugins_emulation.xml` plugins file. Because it uses the OpenVINO plugin architecture, it works with both the OpenVINO Python API and the C++ API. For an example that shows how to use emulation as the inference engine for the runtime `dla_benchmark` utility, refer to "Performing Inference Without an FPGA Board" in the *FPGA AI Suite Getting Started Guide*.

(*) Minor rounding differences between software emulation and hardware will typically result in differences of less than two units of least precision (ulps).

Because the emulation executes on the CPU and does not benefit from the FPGA acceleration, the emulation is much slower than inference on the FPGA. Typical inference times for a single image with ResNet50 are on the order of minutes of time. The inference speed varies dramatically depending on the architecture configuration and the graph.

2.4. FPGA AI Suite Layer / Primitive Ranges

The following table lists the hyperparameter ranges supported by key primitive layers:

Layer / Primitive	Hyperparameter	Supported Range
Fully connected	None	n/a
2D Conv	Filter Size	Width = [1..28] Height = [1..28] Height does not have to equal width. Default value for each is 14.
	Stride	Maximum stride is 15
	Pad	Maximum pad is (2^{16}) - 1
3D Conv	Filter Size	Width = [1..28] Height = [1..28] Depth = [1..14] Filter volume should fit into the filter cache size.
	Stride	Maximum stride is 15.
	Pad	Maximum pad is (2^{16}) - 1
Depthwise	Filter Size	Same as 2D Conv filter size Depth = 1
	Stride	Same as 2D Conv stride Depth = 1
	Pad	Same as 2D conv padding Depth = 1
Scale-Shift	Scale factor	FP16 float range
	Bias term	FP16 float range
Deconv / Transpose Convolution	Filter Size	Any – Same as convolution, and height/width can be different Depth = 1
	Stride	1, 2, 4, 8 (stride width == stride height) Depth = 1
	Pad	Restricted to filter_[height, width] - 1 Depth = 1
ReLU	n/a	n/a
pReLU	Scaling parameter (a) (1 per filter / conv output channel)	float range Depth = 1
Leaky ReLU	Scaling parameter (a) (1 per tensor)	float range
Clamp	Limit parameters (a, b) (1 per tensor)	float range

continued...

Layer / Primitive	Hyperparameter	Supported Range
Round_Clamp	Limit parameters (a, b) (1 per tensor)	float range
H-sigmoid	n/a	n/a
H-swish	n/a	n/a
Sigmoid	n/a	FP16 float range
Swish	n/a	FP16 float range
Tanh	n/a	FP16 float range
Max Pool	Window Size	up to 13x13x13
	Pad	1, 2
	Stride	1, 2, 3, 4
Average Pool	Window Size	Up to 27x27 (one less than the maximum 2D convolution size) Width == Height Depth = 1 or 2
	Pad	1, 2
	Stride	1, 2, 3, 4
Softmax	Maximum Number of Channels	4096
Elementwise Multiplication of feature * filter and feature * feature tensors.(1)	n/a	Tensor sizes are expanded if necessary to support the multiplication. Depth = 1
ChannelToSpace DepthToSpace PixelShuffle	block_mode	`blocks_first` or `blocks_last`
	block_size	2, 4, 8

2.5. FPGA AI Suite IP Block Configuration

The FPGA AI Suite IP block has many important parameters that describe arithmetic precision, feature set, size of various modules (such as the PE Array), and details regarding the internal buses and the external AXI interface.

Configurable parameters are specified in the Architecture Description (`.arch`) file, as described in Architecture Description File Format for Instance Parameterization on page 16 and Architecture Description File Parameters on page 17.

The table below shows the major parameters, some of which are not configurable, that describe the IP block.

Common Parameter Name	Description	Valid Range
c_vector (CVEC)	Size of the dot product performed by each PE in the PE Array. Typically optimized when generating an optimized architecture with the FPGA AI Suite compiler.	[4,8,16,32,64]
k_vector	Number of PEs in the PE Array	[4-128]

continued...

(1) This is an element-wise multiplication, not a matrix multiply operation.

Common Parameter Name	Description	Valid Range
(PE KVEC)		Must be a multiple of c_vector
num_lanes	Number of execution lane. Each lane contains a set of auxiliary modules (aux_modules) and a number (equal to k_vector) of PEs	[1,2,4]
N/A	Number of auxiliary modules connecting to the crossbar (XBAR)	[1-4]
pool k_vector (Pool KVEC)	Width of the pool interface. Typically optimized when generating an optimized architecture with the FPGA AI Suite compiler.	[1,2,4,8,16,32,64]
pool max_window_height pool max_window_width	Size of the pooling window	[3x3, 7x7,13x13]
depthwise k_vector (Depthwise KVEC)	Number of output channels processed in parallel. Typically optimized when generating an optimized architecture with the FPGA AI Suite compiler.	[16, 32, 64] Must be equal to k_vector
depthwise max_window_height depthwise max_window_width	Size of the depthwise filter	[3x3, 5x5, 7x7]
depthwise max_dilation_vertical depthwise max_dilation_horizontal	Maximum supported value for the depthwise dilation	[1-6]
activation k_vector (Activation KVEC)	Width of the activation interface. Typically optimized when generating an optimized architecture with the FPGA AI Suite compiler.	[2,4,8,16,32,64]
enable_clamp	Enables clamp activation function	[true, false]
enable_relu	Enables ReLU activation function	[true, false]
enable_leaky_relu	Enables Leaky ReLU activation function	[true, false]
enable_prelu	Enables PReLU activation function	[true, false]
enable_round_clamp	Enables round clamp activation function	[true, false]
enable_sigmoid	Enables Sigmoid and Swish activation functions	[true, false]
enable_tanh	Enables Tanh activation function	[true, false]
enable_parameter_rom	Enables storing graph parameters in on-chip memory, which requires input and output streaming to be enabled and configured. For details about DDR-free operation, refer to "Generating Artifacts for DDR-Free Operation" in the *FPGA AI Suite Compiler Reference Manual*.	[true, false]
arch_precision (PE precision)	Precision of features and weights in the PE Array. For details about how this parameter affects DSP utilization, refer to the "Parameter: `arch_precision`" section of Parameter Group: Global Parameters on page 17.	"FP11" (INT7-BFP / 1s.6m.5e) "FP12AGX" (INT8-BFP / 8m.5e, two's complement) "FP13AGX" (INT9-BFP / 9m.5e, two's complement) "FP16" (INT12-BFP / 1s.11m.5e)
PE bias add precision	Precision of accumulator bias value in the PE Array.	fp16
PE accumulator precision	Precision of the accumulators in the PE Array.	fp32
PE drain precision	Precision of values drained from the PE Accumulators to the XBAR and AUX Modules.	fp16

continued...

Common Parameter Name	Description	Valid Range
PE interleave factor	Multi-threading factor for the features x filters in the PE array accumulators.	**Agilex 5 devices:** 12x1 **Agilex 7 devices:** 2x3, 3x2, 5x1, 1x5 **Arria 10 devices:** 2x2, 4x1, 1x4 **Stratix® 10 devices:** 2x3, 3x2, 5x1, 1x5 1×1 supported for graphs with no bias
PE scale precision	Precision of scale multiplier in the PE array	fp16
Aux module precision	Precision of the Aux Modules	fp16
Memory port width	Width of memory port	[64, 128, 256, 512]
enable_debug	Toggle the FPGA AI Suite debug network that includes interface profiling counters that can be queried with the CSR. Enabled by default.	[true, false]
enable_layout_transform	Enables the dedicated input tensor layout transform module.	[true, false]

The major constraints include:

- PE KVEC must be a multiple of CVEC
- PE KVEC must be divisible by XBAR and AUX KVECs
- PE drain width must be equal to XBAR KVEC

Graph limitations include:

- Convolution filter size: 1×1 -> 28×28, including asymmetric
- Convolution filter stride: 1 .. 15
- No limitation on convolution padding
- The limits of the depthwise layers are the same as normal convolution. Depthwise convolution is handled with software emulation using regular convolution passes.

The maximum supported DDR size is 4GB.

2.5.1. Architecture Description File Format for Instance Parameterization

The FPGA AI Suite IP has a highly configurable architecture. Configuring the design allows for different trade-offs between inference performance (throughput and latency) and utilization of FPGA resources (area). Configurations are specified through Architecture Description Files. The IP instances corresponding to these configurations can be compiled as part of an FPGA design into an FPGA bitstream.

The architecture determines how much FPGA area is consumed by the FPGA AI Suite IP and strongly affects the achieved inference fps and ease of timing closure.

Achieving the best performance of a given graph for a given FPGA area (or the smallest FPGA area for a given performance target) requires optimizing the architecture. The architecture optimization function of the FPGA AI Suite compiler is designed to produce good architectures for a given graph or set of graphs. For more details about the architecture optimization function of the compiler, refer to the *FPGA AI Suite Compiler Reference Manual*.

The FPGA AI Suite Architecture Description Files use the protobuf format and have a `.arch` file extension. While these files are human readable and editable, manually optimizing an architecture requires a deep knowledge of the FPGA AI Suite IP design and is not recommended.

You adjust some of the architecture parameters by hand, because the Architecture Optimizer does not modify them. For example, the optimizer does not modify the numerical precision (for example, fp16 or fp11) in the architecture file. Similarly, the optimizer does not modify details related to the AXI interfaces on the IP. In some case, you can improve performance of the resulting optimized architecture by the choice of these values.

When possible, modifying the graph or batch size might also result in performance improvements. For example, a graph that requires FP16 precision might have sufficient accuracy at FP11 or FP12 if a few extra layers are added. Reducing the internal precision enables a large memory and area reduction. Very small and fast graphs might achieve a higher performance on hardware by using a batch size that is greater than one.

The `example_architectures/` directory includes an example that shows how to enable the hardware-accelerated softmax function.

The comment character in the `.arch` format is #.

2.5.2. Architecture Description File Parameters

2.5.2.1. Parameter Group: Global Parameters

Parameter: `family`

This parameter specifies the target FPGA device family for the architecture.

Legal Values

Table 3. Valid Values for `family` Global Parameter

Value	Description
`A10`	Target Arria 10 devices.
`AGX5`	Target Agilex 5 devices.
`AGX7`	Target Agilex 7 devices.
`C10`	Target Cyclone® 10 devices.
`S10`	Target Stratix 10 devices.

Parameter: `k_vector`

This parameter, also called KVEC, describes the number of filters that the PE Array is able to process simultaneously.

Typically the architecture optimizer is used to set this parameter.

Legal values: [4-128]

- The k_vector value must be a multiple of the c_vector value.
- The k_vector value must be divisible by the xbar_k_vector and auxiliary k_vector values.
- When you use the depthwise module, the k_vector value must equal the c_vector value.

Parameter: c_vector

This parameter, also called CVEC, describes the size of the dot product within each PE in the PE Array.

Typically the architecture optimizer is used to set this parameter.

Legal values: [4,8,16,32,64]

- When you use the depthwise module, the c_vector value must equal the k_vector value.

Parameter: num_lanes

This parameter describes how many output-height slices the architecture can compute in parallel.

Using the num_lanes architecture parameter has the following effects:

- Setting the num_lanes parameter scales the PE array in the FPGA AI Suite IP by the given number and provides additional parallelism at the cost of more DSPs and area.
- The total stream buffer size scales with the num_lanes parameter. Because the feature surface of a graph is divided across multiple lanes, adjust the stream_buffer_depth parameter listed in the .arch file by the inverse of the num_lanes parameter value. For example, a 4-lane architecture with 10k stream buffer depth indicates a 40k total stream buffer size.

When the value of the num_lanes parameter of architecture is greater than 1, the architecture is subject to the following limitations:

- All c_vector and k_vector values in the architecture must be the same
- The softmax auxiliary module is not supported.

Legal values: [1,2,4]

Parameter: arch_precision

This parameter sets the precision (in bits) of the internal numeric representation used by FPGA AI Suite IP. Lower values increase fps and reduce area, but at the cost of inference accuracy.

Each internal precision option corresponds to a different number of sign and mantissa bits, and uses either two's complement or sign+magnitude. For details, refer to the table in FPGA AI Suite IP Block Configuration on page 14.

The FP16 precision significantly increases the size of the resulting IP, but can improve accuracy (particularly in models that have not been retrained for low precision).

All numeric options use block floating point format. In block floating point format, each block of size CVEC shares a common exponent. Both CVEC (`c_vector`) and `arch_precision` affect the accuracy of the inference. However, the impact of `c_vector` is generally small, while the impact of the `arch_precision` setting is relatively large.

The block floating point format used by the FPGA AI Suite IP is directly compatible with graphs that use INT8 symmetric quantization. INT8 symmetric quantization requires that all operations going from floating point to integer, and vice versa, require only scaling (multiplication or division). When given a graph with INT8 weights, the FPGA AI Suite compiler sets the exponent of the block floating point weights so the original INT8 weights can be used directly as the mantissa. This setting limits the use of INT8 weights to architectures where the mantissa is 8-bits or larger.

The use of INT8 graphs does not significantly affect either the inference speed or the FPGA resource consumption. All inference, regardless of whether block floating point is used, is performed with the same hardware.

In addition to selecting a compatible numeric precision, set the `pe_array/enable_scale` parameter to true in order to support graphs with INT8 quantization.

The example architectures that are included with the FPGA AI Suite are already set to the recommended `arch_precision` parameter values for their supported FPGA family. In some cases, it is useful to select a different `arch_precision` value. FP11 is the lowest precision option, but requires the least number of RAM blocks, and slightly reduces the amount of external memory traffic. The FP12AGX significantly reduces the number of DSPs required to implement the PE array, but logic utilization may increase.

Agilex 5 devices implement enhanced DSPs with AI tensor blocks. To take advantage of AI tensor blocks, set the `arch_precision` value to FP12AGX or FP11 and use interleave values of `12x1`.

For more details about the block floating point format, refer to the *Low-Precision Networks for Efficient Inference on FPGAs* white paper.

Legal values:

FPGA Device Family	Supported `arch_precision` Values
Agilex 5	• FP11 • FP12AGX • FP13AGX • FP16 (less common)
Agilex 7	• FP11 • FP13AGX • FP16 (less common)

continued...

FPGA Device Family	Supported `arch_precision` Values
Arria 10	• FP11 • FP16 (less common)
Cyclone 10 GX	• FP11 • FP16 (less common)
Stratix 10	• FP11 • FP16 (less common)

Table 4. Multiplication Operations per DSP

Precision	FPGA Device Family		
	Arria 10 Cyclone 10 GX Stratix 10	Agilex 7	Agilex 5
FP11	4	4	20(*)
FP12AGX	–	–	20(*)
FP13AGX	–	4	4(**)
FP16	2	2	2

INT8 symmetric quantization is enabled by FP12AGX and higher precision options.

The total number of multipliers required by the PE Array will be equal to CVEC * KVEC * num_lanes. Due to quantization, this calculation underpredicts the number of DSPs required when using Agilex 5 DSP tensor mode. In addition, the PE Array requires KVEC * num_lanes DSPs to build the FP32 accumulators.

Parameter: `stream_buffer_depth`

This parameter controls the depth of the stream buffer. The stream buffer is used as the on-chip cache for feature (image) data. Larger values increase area (logic and block RAM) but also increase performance.

Typically the architecture optimizer is used to set this parameter.

Legal values: [2048-262144]

Parameter: `enable_eltwise_mult`

This parameter enables the Elementwise multiplication layer. This layer is required for MobileNetV3.

(*) Assumes a 12x1 interleave.

(**) The Agilex 5 DSP block supports six multipliers per block, but currently FPGA AI Suite uses only four of them.

Parameters: `filter_size_width_max`, `filter_size_height_max`

These parameters determine the maximum size of a convolution filter, which also relates the maximum window size for Average Pool.

The maximum window size for Average Pool is no larger than the value determined by the following formula: $\min(\text{filter\_size\_width\_max}, \text{file\_size\_height\_max}) - 1$. In addition, the Average Pool window size may be limited by the filter_scratchpad and filter_depth parameters.

Legal values: [14,28]

Parameters: `output_image_height_max`, `output_image_width_max`, `output_channels_max`

These parameters control the maximum size of the output tensor.

The default maximum size is 128x128, with up to 8192 channels

Parameter: `enable_debug`

This parameter toggles the FPGA AI Suite debug network to allow forwarding of read requests from the CSR to one of many externally-attached debug-capable modules.

Generally not required for production architectures.

Legal values: `[true,false]`

Parameter: `enable_layout_transform`

The parameter enables the dedicated input tensor transform module in the FPGA AI Suite IP. When enabled, the dedicated layout transform hardware transforms the input tensor format and folds the inputs into channels.

You can use the layout transform in streaming and non-streaming configurations of the FPGA AI Suite IP. The transform is particularly useful for doing fast and deterministic tensor preprocessing in hostless applications, or applications where the hard-processor is slow or highly loaded.

However, the layout transform comes with an FPGA area cost that scales mainly with the input data bus width, maximum tensor/stride dimensions, and CVEC. In particular, instances where the value of `max_stride_width` × `max_stride_height` × `max_channels` is greater than the CVEC value consume significant memory resources due to the buffer space required for the overflowing CVEC.

In graphs where the first convolution stride dimensions are unity, no folding can be done, and the layout transform cannot optimize the layout of the input tensor. In such a case, try doing a lighter-weight transformation operation outside of the FPGA AI Suite IP.

When this parameter is enabled, configure the transform with the parametersas described in Parameter Group: layout_transform_params on page 31. For information about the layout transformation operation and hardware, refer to Input Layout Transform Hardware on page 36.

The hardware layout transform is not supported in SoC designs in streaming-to-memory (S2M) mode. the S2M design uses a lightweight, external transform module.

2.5.2.2. Parameter Group: `activation`

This parameter group configures the activation module. These activation functions are common in deep learning, and it is beyond the scope of this document to describe them.

Different activation functions can be enabled or disabled to suit the graph to be run. Disabling unnecessary activations functions can reduce area.

Parameter: `activation/enable_relu`

This parameter enables or disables the Rectified Linear Unit (ReLU) activation function.

Legal values: [true, false]

Parameter: `activation/enable_leaky_relu`

This parameter enables or disables the Leaky ReLU activation function. This activation function is a superset of the ReLU activation function.

Legal values: [true, false]

Parameter: `activation/enable_prelu`

This parameter enables or disables the Parametric ReLU activation function. This activation function is a superset of the Leaky ReLU activation function.

Legal values: [true, false]

Parameter: `activation/enable_clamp`

This parameter enables or disables the clamp function. Enabling the clamp function also enables a ReLU6 activation function.

Legal values: [true, false]

Parameter: `activation/enable_round_clamp`

This parameter enables or disables the round_clamp function. Enabling the round_clamp function also enables ReLU.

If both enable_clamp and enable_round_clamp are set, enable_round_clamp takes priority over enable_clamp when implementing ReLU.

Legal values: [true, false]

Parameter: `activation/enable_sigmoid`

This parameter enables or disables the Sigmoid and Swish activation functions.

As a side-effect, enabling these activation functions also enables the Tanh and Reciprocal activation function. This side-effect might change in a future release, so the best practice is to enable activation function explicitly instead of depending on the side-effect.

Legal values: [true, false]

Parameter `activation/enable_tanh`

This parameter enables or disables the Tanh activation function.

As a side-effect, enabling this activation functions also enables the Sigmoid, Swish, and Reciprocal activation functions. This side-effect might change in a future release, so the best practice is to enable activation functions explicitly instead of depending on the side-effect.

Legal value: [true, false]

2.5.2.3. Parameter Group: `pe_array`

This parameter group configures the PE Array. The PE Array is used to calculate dot products.

Parameter: `pe_array/dsp_limit`

Use this parameter to force the PE array to implement multipliers in ALM logic on the FPGA.

The number of multipliers that the PE requires is determined by the `k_vector` and `c_vector` global parameters. Given the value of the `arch_precision` global parameter and the target architecture (for example, Arria 10 or Agilex 7), the number of multipliers determines the number of DSPs that the PE Array tries to use. If this number exceeds the value set in the `dsp_limit` parameter, then some multipliers are implemented in ALM logic to ensure that the PE Array DSP usage does not exceed the limit set by the `dsp_limit` parameter.

If this option is omitted, then all multipliers are implemented in the FPGA AI Suite IP as DSPs.

Typically, this parameter is set by the architecture optimizer.

Parameters: `pe_array/num_interleaved_features`, `pe_array/num_interleaved_filters`

To support layers with bias values, the PE array uses a threaded accumulator that is time-multiplexed to handle multiple accumulations. Each accumulation corresponds to an output filter and feature.

Common Values:	*Agilex 5 devices*	12x1
	Agilex 7 devices	5x1, 3x2

Arria 10 devices	4x1, 2x2
Cyclone 10 GX	
Stratix 10 devices	5x1, 3x2

All architectures support a 1x1 interleave. Selecting a 1x1 interleave typically reduces ALM consumption, but the IP associated with this architecture does not support layers with bias. Because most deep learning graphs include bias, the 1x1 interleave is typically not used.

The architecture optimizer does not modify the `num_interleaved_features` and `num_interleaved_filters` values. You must set them manually.

The filter interleave multiplies the effective KVEC, which means that graphs with a depthwise convolution (such as certain versions of MobileNet) might perform best when using `num_interleaved_filters=1`. Multilayer perceptron graphs might perform best when using `num_interleaved_features=1`.

Except in the 1x1 case, the value of `num_interleaved_features` multiplied by `num_interleaved_filters` must meet the following requirements:

Agilex 5 devices	The value of `num_interleaved_features` must be greater than or equal to 12.
Agilex 7 devices	The value of `num_interleaved_features` multiplied by `num_interleaved_filters` must be greater than or equal to five.
Arria 10 devices	The value of `num_interleaved_features` multiplied by `num_interleaved_filters` must be greater than or equal to four.
Cyclone 10 GX devices	
Stratix 10 devices	The value of `num_interleaved_features` multiplied by `num_interleaved_filters` must be greater than or equal to five.

There is no advantage in choosing interleave factors larger than the minimum required.

Parameter: `pe_array/exit_fifo_depth`

This parameter controls the depth of the PE Array exit FIFO. Larger values might reduce the incidence of stalling, but at the cost of area.

Typically, this parameter is not modified.

Parameter: pe_array/enable_scale

This parameter controls whether the IP supports scaling feature values by a per-channel weight. This is used to support batch normalization and INT8 scaling (in graphs that are INT-8 quantized and do not use block floating point).

In most graphs, the graph compiler (`dla_compiler` command) adjusts the convolution weights to account for scale, so this option is usually not required. (Similarly, if a shift is required, then the convolution bias values are adjusted).

Legal values: true, false

2.5.2.4. Parameter Group: pool

This parameter group configures the pool module. The pool module is used for max pooling only; average pooling is performed using the convolution engine.

Parameter: pool/k_vector

This parameter controls the width of the pool interface.

Typically, the architecture optimizer is used to set this parameter.

Legal values: [1,2,4,8,16,32, 64]

Parameters: pool/max_window_height, pool/max_window_width

These parameters set the maximum window height and width that the architecture can support.

Typically, you set this value to the size of the largest max pooling window in your graph. Larger values cost more FPGA area.

Legal values: [3-7]

Parameters: pool/max_stride_vertical, pool/max_stride_horizontal

These parameters set the maximum stride values that the architecture can support.

Typically, you set these to the largest value your graph requires after a max pool. Larger values cost FPGA area.

Legal values: [1-4]

2.5.2.5. Parameter Group: depthwise

This parameter group configures the depthwise module. The depthwise module accelerates the depthwise convolutions.

If a depthwise layer has more parameters than the depthwise module can support, the layer is executed on the general PE array.

Parameter: depthwise/K_vector

This parameter controls the width of the depthwise interface. Typically, the architecture optimizer is used to set this parameter.

Legal Values: [16, 32, 64]

Parameters: depthwise/max_window_height, depthwise/max_window_width

These parameters set the maximum window height and width that the architecture can support. Typically, you set this value to the size of the largest max depthwise window in your graph.

Larger values cost more FPGA area.

Legal Values: [3, 5, 7]

Parameters: depthwise/max_stride_vertical, depthwise/max_stride_horizontal

These parameters set the maximum stride values that the architecture can support. Typically, you set these to the largest value your graph requires after a depthwise convolution.

Larger values cost FPGA area.

Legal Values: [1-4]

Parameters: depthwise/max_dilation_vertical, depthwise/max_dilation_horizontal

These parameters set the maximum dilation values that the architecture can support. Typically, you set these to the largest value your depthwise convolutions in your graph require

Larger values cost FPGA area.

Legal Values: [1-6]

2.5.2.6. Module: softmax

The `softmax` module is enabled or disabled by including a custom auxiliary primitive (`custom_aux_primitive`) with a `layer_type` and `name` set to `softmax`. The primitive must connect to the crossbar (`xbar`).

The file `example_architectures/Generic_softmax.arch` has an example of this connectivity.

Parameter: softmax/max_num_channels

This parameter sets the maximum number of channels for SoftMax that the architecture can support.

Legal Values: [1-4096]

2.5.2.7. Parameter Group: dma

This parameter group defines key attributes related to the external AXI interface for the IP.

Parameters: dma/csr_addr_width, dma/csr_data_bytes

These parameters define the interface to the CSR.

Parameters: dma/ddr_addr_width, dma/ddr_burst_width, dma/ddr_data_bytes, dma/ddr_read_id_width

These parameters define the AXI interface to off-chip memory.

2.5.2.8. Parameter Group: xbar

For each layer of the graph, data passes through the convolution engine (referred to as the processing element [PE] array), followed by zero or more auxiliary modules. The auxiliary modules perform operations such as activation or pooling.

After the output data for a layer has been computed, it can be sent to one of the following places:

- An internal buffer while waiting for the start of the next convolution layer.
- The external memory for reading by the host program.

Internally, a crossbar (xbar) connects the modules together. The `xbar` parameters specify the connections between the PE array, the auxiliary modules, the input feeder (which holds data waiting for the next convolution layer), and the output writer (which writes the data to the external memory).

Consider the following example:

```
xbar {
      xbar_k_vector : 16
      max_input_interfaces : 4
      max_output_interfaces : 4
      xbar_ports {
        xbar_aux_port {
          name : 'activation'
          input_connection : 'xbar_in_port'
        }
        xbar_aux_port {
          name : 'pool'
          input_connection : 'xbar_in_port'
          input_connection : 'activation'
        }
      }
      xbar_in_port {
        external_connection  : 'pe_array'
      }
      xbar_out_port {
        external_connection : 'input_feeder'
        external_connection : 'output_writer'
        input_connection    : 'xbar_in_port'
        input_connection    : 'pool'
      }
    }
```

The crossbar always has the following elements:

- An `xbar_in_port` element that accepts the incoming connection from the PE array.
- An `xbar_out_port` element that connects externally to the input feeder and output writer.

This example architecture also has two auxiliary modules defined with `xbar_aux_port` elements: a pool module and an activation module. The `xbar_aux_port` elements are specified in the `xbar_ports` section.

In this configuration, the activation module can accept data from the `xbar_in_port` (the PE array), while the pool module can accept data from the `xbar_in_port` or from the activation module.

Finally, the output port can accept data either directly from the `xbar_in_port` or from the pool module. These connections limit how data flows through the system.

In this example, the activation module cannot write out from the layer. Activations must be followed by pooling layers. Also, an activation layer cannot follow a pooling layer. However, the convolution can be followed by a pooling layer without an activation layer in between.

Connections cost area and can reduce f_{MAX}. You can reduce area by including only the connections that are required for a given graph (sometimes called "depopulating" the crossbar). If you use this architecture as the starting point for the Architecture Optimizer, the result can improve throughput.

To see examples of other crossbar configurations, review the `example_architectures/` directory.

Parameter: `xbar/xbar_k_vector`

This parameter defines the width of the interface into the crossbar. Typically, this parameter is set to be equal to the width of the widest interface into any of the auxiliary modules.

Typically, the architecture optimizer is used to set this parameter.

Legal values: [2,4,8,16,32,64]

2.5.2.9. Parameter Group: `filter_scratchpad`

These parameters define the on-chip cache used for the filter weights, bias values, and scale values (if scale and bias are enabled).

Parameter: `filter_scratchpad/filter_depth`

This parameter defines the size of the on-chip cache used to store convolution filter weights. Larger values use more FPGA resources, but might increase fps.

Typically, the architecture optimizer is used to set this parameter.

Legal values: Depends on the setting of `enable_parameter_rom`:

- `enable_parameter_rom=false`

  2^n where *n* is 9, 10, or 11
- `enable_parameter_rom=true`

  2^n where *n* is 4 or greater.

Parameter: `filter_scratchpad/bias_scale_depth`

This parameter defines the size of the on-chip cache used to store feature bias and scale weights, if the corresponding support is enabled in the PE Array. Larger values use more FPGA resources, but might increase inference throughput.

Typically this is set equal to the `filter_scratchpad/filter_depth` parameter.

Legal values: Depends on the setting of `enable_parameter_rom`:

- `enable_parameter_rom=false`

  2^n where *n* is 9, 10, or 11
- `enable_parameter_rom=true`

  2^n where *n* is 4 or greater.

2.5.2.10. Parameter Group: `input_stream_interface`

Enable and configure the width of the input AXI4-Stream interface. If enabled, input streaming requires that the layout transform is also enabled. Neural network graphs compiled with an input streaming-enabled architecture cannot be sliced in the input layer, therefore the stream buffer depth must be large enough to fit the input layer.

Parameter: input_stream_interface/enable

Enables the input streaming module. Enable_layout_transform must also be true, and the transform must be configured for the target neural network graph.

Legal values: true, false

Parameter: input_stream_interface/data_width

Sets the width of the AXI4-streaming input bus in bits.

Legal values: 2^n where *n* is 4 or greater.

Parameter: input_stream_interface/fifo_depth

Sets the depth of the input FIFO

Legal values: 2^n where *n* is 4 or greater.

For best performance, set the `fifo_depth` value to the size of one transformed input (refer to Input Transform Mapping on page 35) For example, if the transformed input is 32×28×28×x1 and the `c_vector` size is 16, set `fifo_depth` to 2048.

For example, an architecture with a 128-bit input AXI streaming interface would include the following options:

```
input_stream_interface {
 enable: true
 data_width: 128
 fifo_depth: 2048
}
```

2.5.2.11. Parameter Group: `output_stream_interface`

Enable and configure the AXI4-Stream output interface.

Parameter: `output_stream_interface/enable`

Enables the output streaming module.

Legal values: true, false

Parameter: `output_stream_interface/data_width`

Sets the width of the AXI4-steaming output interface in bits.

Legal values: 2^n where *n* is 4 or greater.

Parameter: `output_stream_interface/fifo_depth`

Sets the depth of the output FIFO.

Legal values: 2^n where *n* is 4 or greater.

For best performance, the FIFO depth should hold the entire output $\text{ceil}\left(\frac{\text{channel}}{\text{xbar k\_vector}}\right) \times \text{width} \times \text{height} \times \text{depth}$. For example, if the output is 64×10×10×1 and the xbar k_vector is 32, the FIFO depth should be set to 256.

For example, and architecture file with an 128-bit wide output AXI streaming interface would include the following options:

```
output_stream_interface {
 enable: true
 data_width: 128
 fifo_depth: 1024
}
```

2.5.2.12. Parameter Group: config_network

These parameters define the configuration network that connects internal components within the IP.

Typically, this section is changed only minimally when modifying the examples in the example_architectures/ directory.

2.5.2.13. Parameter Group: layout_transform_params

These parameters configure the input tensor layout transformation module of the FPGA AI Suite IP.

Parameter: layout_transform_params/do_u8_fp16_conversion

When true, this parameter enables hardware to convert 8-bit integer input values to FP16 format, and 8-bit unsigned integers must be given as inputs. Otherwise, no conversion is done and you must write FP16 values at the input.

Legal values: [true, false]

Parameters: layout_transform_params/max_channels, layout_transform_params/max_feature_height, layout_transform_params/max_feature_width, layout_transform_params/max_feature_depth, layout_transform_params/max_stride_height, layout_transform_params/ max_stride_width, layout_transform_params/max_stride_depth, layout_transform_params/max_pad_top, layout_transform_params/max_pad_left, layout_transform_params/max_pad_depth, layout_transform_params/max_filter_width, layout_transform_params/max_filter_height, layout_transform_params/max_filter_depth

This group configures the range of feature shapes, padding, and convolution strides that the layout transform hardware module supports. The values in this configuration represent the maximum allowed values for each parameter. However, the resource usage of the layout transform is sensitive to the parameterization, so set these values as close to the actual values as possible.

The parameters of the first convolution in the graph must fit within the maximum range configured here. The exact parameters required by the layout transform module are reported by the FPGA AI Suite compiler in a file named input_transform_dump_<*graphname*>.csv.

For more information about the input tensor layout transform, refer to Input Feature Tensor In-Memory Format on page 33.

2.6. IP Block Interfaces

This document uses the terms *initiator* and *responder* where the terms *master* and *slave* might have been used in the past.

2.6.1. Clock and Reset

Table 5. Clocks

Name	Description
dla_clk	Clock used by internal processing logic
ddr_clk	Clock used by DDR memory and CSR interfaces
irq_clk	Clock used for interrupt request (IRQ) interface

Table 6. Resets

Name	Description
dla_resetn	Global asynchronous reset This reset must be held for at least three cycles of the slowest of the clocks listed in the Clocks table. The IP becomes responsive sometime after the reset is released, but not immediately due to an internal reset cycle in the FPGA AI Suite IP.

2.6.2. AXI Interfaces

Name	Type	Description
DDR0 Initiator	AXI4	Initiator port for connecting to DDR memory
CSR Responder	AXI4-Lite	Exposes IP MMIO region
Interrupt Initiator	Interrupt Sender	Level sensitive interrupt

2.6.3. AXI Interface Clock and Reset

Name	Clock	Reset	Note
DDR0 Initiator	ddr_clk	dla_resetn	N/A
CSR Responder	ddr_clk	dla_resetn	The CSR initiator operates on the ddr_clk clock.
Interrupt Initiator	irq_clk	dla_resetn	N/A

The following parameters are used by the AXI interfaces. The parameter values can be modified in the Architecture Description files as described in IP Generation Utility.

Name	Supported Value	Entry in Architecture Description
C_CSR_AXI_ADDR_WIDTH	11	= dma.csr_addr_width
C_CSR_AXI_DATA_WIDTH	32	= dma.csr_data_bytes * 8
C_DDR_AXI_ADDR_WIDTH	1~32	= dma.ddr_addr_width
C_DDR_AXI_BURST_WIDTH	1~8	= dma.ddr_burst_width
C_DDR_AXI_DATA_WIDTH	64, 128, 256, 512 (bits)	= dma.ddr_data_bytes * 8
C_DDR_AXI_THREAD_ID_WIDTH	2	= ddr_read_id_width

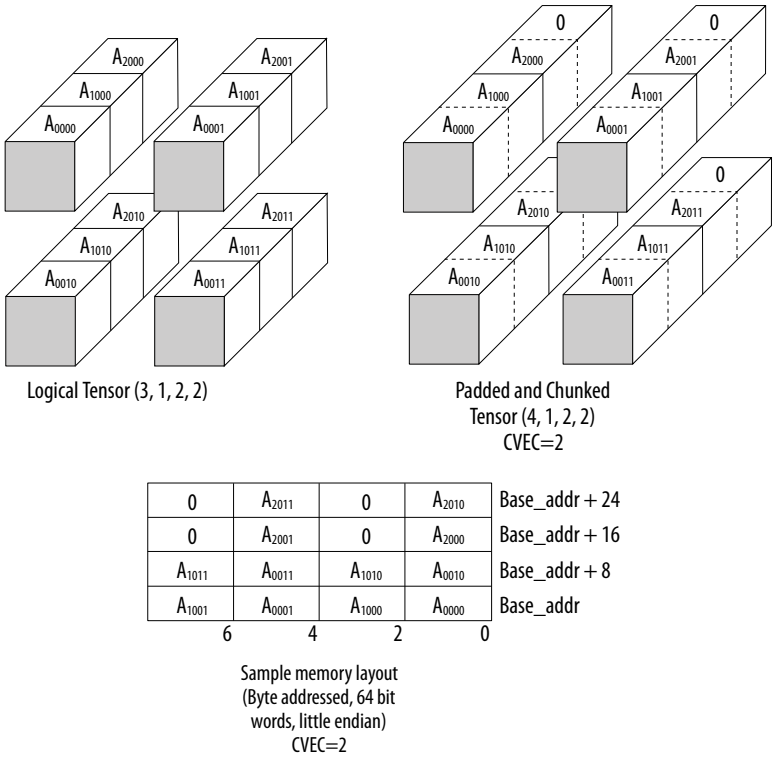
2.6.4. Input Feature Tensor In-Memory Format

Input features are stored in FP16 format. FP16 format has 1 sign bit, 10 mantissa bits, and 5 exponent bits. The input features are converted by the IP hardware to its native format using a *round to nearest, ties to even* (RNE) rounding rule.

Feature elements are packed into CVEC-sized chunks in the channel dimension from low to high. The final CVEC chunk, at a given (d,h,w), is padded with zeros. The CVEC chunks are stored in NCDHW format. The order is as follows: batch, channel, depth, height, width, and CVEC, where CVEC is the fastest changing index and batch the slowest.

The following figure shows a sample memory layout for a 1×3×1×2×2 input tensor to a CVEC=2 architecture:

Figure 3. Input Tensor In-Memory Layout



2.6.4.1. Multiple Input Graphs

For graphs with more than one input, each tensor is structured as described in the previous section. The multiple input tensors must be packed together at address offsets as reported by the FPGA AI Suite compiler.

The compiler generates CSV files that describe the input and output tensor, unless you specify the `--fno-transform-tables` option. Each row of the CSV file gives information about one input. For more details, refer to the *FPGA AI Suite Compiler Reference Manual*.

For multiple inputs, the inputs are stored by batch and then by input number. For example, for 3 inputs and 2 batches, the input tensors would be stored as follows:

- input 1, batch 1
- input 1, batch 2
- input 2, batch 1
- input 2, batch 2
- input 3, batch 1
- input 3, batch 2

2.6.4.2. Input Folding

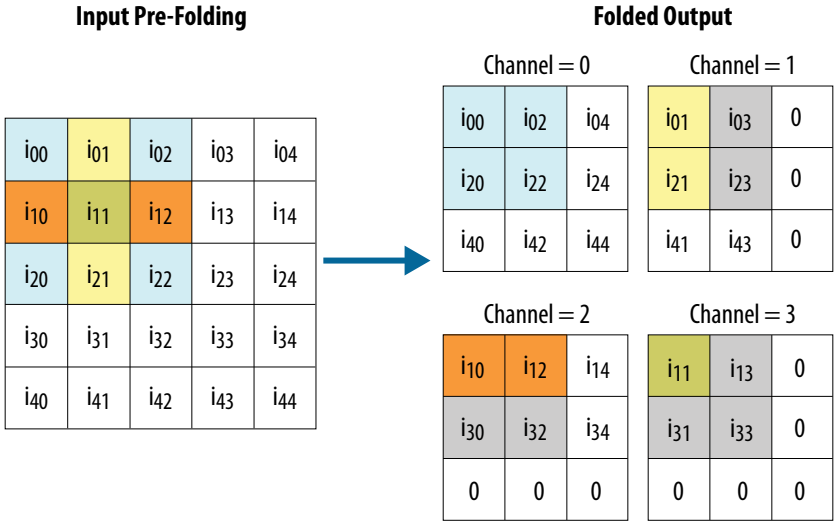
Many graphs, particularly those processing image data, have very shallow input channels. In high CVEC instantiations, very shallow input channels can lead to low computational efficiency.

The folding can be done in conjunction with the FPGA AI Suite compiler and the FPGA AI Suite IP, or it can be performed in hardware using the hardware layout transform described in Input Layout Transform Hardware on page 36.

Input folding is typically most beneficial to the first layer of a graph.

The following figure illustrates an example of the functionality of the folding transform performed by the FPGA AI Suite compiler.

Figure 4. Illustration of the folding transform for a 1x1×5×5 input, 1×1×3×3 filter and stride_height = stride_width = 2



In this transformation, the input depth, height, and width are folded into the channel dimension by a factor corresponding to the stride of the first convolution of a network. In the earlier figure, this factor corresponds to transforming the input channels from 1 to 4 ($1 \times \text{stride\_depth} \times \text{stride\_height} \times \text{stride\_width}$), input height from 5 to 3 ($\text{ceil}\left(\frac{5}{2}\right)$) and input width from 5 to 3. Each color corresponds to the new filter window, which in this

case would be 4×1×2×2, with the gray boxes corresponding to 0 padding for the filters. Folding is done in a similar way for inputs with depths greater than one, but the folding transform illustration excludes it for simplicity.

The FPGA AI Suite IP has various enhancements that reduce, but not eliminate the efficiency hit of shallow first layers. In many cases, you can disable first layer folding in the compiler and pass shallow-channel tensors directly to the IP hardware.

You can disable or enable folding with the following FPGA AI Suite compiler options:

- NoFolding

  No folding is performed by the host or an external module. This leads to low efficiency for the FPGA AI Suite IP but might be useful for debugging purposes.
- ExternalFullFolding

  Folding performed by the host or an external module for the depth, height, and width stride of the first convolution layer.
- ExternalFullExtraPEFolding

  Folding is performed by the host or an external module for the depth, height, and width stride of the first convolution layer with additional folding performed afterward via the FPGA AI Suite IP. This might lead to better performance than ExternalFullFolding depending on the instantiation parameters of IP.
- PEFolding

  Folding is performed entirely by the FPGA AI Suite IP without the need for any host or external module. This should lead to similar performance to ExternalFullFolding depending on the instantiation parameters of the FPGA AI Suite IP and the neural network topology.

  PEFolding mode does not support input with a depth greater than one.

2.6.4.3. Input Scale and Shift

Many graphs require that input data be pre-scaled and pre-shifted. These scale and shift operations are supported in the FPGA AI Suite IP if they are sent to the device. Depending on the folding options specified, the method of support differs for the FPGA AI Suite IP. Input preprocessing is not supported for 5D inputs.

For the ExternalFullFolding or ExternalFullExtraPEFolding options, external modules of the FPGA AI Suite are responsible for replacing the zero padding of the input data with non-zero shift values received as input from the FPGA AI Suite compiler.

For the NoFolding or PEFolding options, the scaling and shifting are performed entirely by the FPGA AI Suite IP without any additional support from the host or external modules. If the scale and shift operation is not mapped to the device, then it is typically performed as an operation fused with the conversion to FP16. This fused operation is performed either by a host CPU or by an external hardware block.

2.6.4.4. Input Transform Mapping

To help make sense of how input data is transformed for a compiled graph, the FPGA AI Suite compiler creates the following CSV files:

- `input_transform_dump_<graph-name>.csv`

  This file describes the tensor shape, padding, and stride for each input tensor.

- `input_transform_mapping_<graph-name>.csv`

  This file shows the element-wise mapping of the logical input tensor elements (domain) to the FPGA AI Suite IP input tensor format (co-domain) described earlier.

The transform mapping file has columns that correspond to the offset and subscript indices for the logical input tensor elements, and the corresponding elements in the transformed FPGA AI Suite input tensor.

For a graph with the input example given in Input Tensor In-Memory Layout, the transform mapping CSV output from the DLA compiler would be as follows:

Table 7. Example Transform Mapping CSV Output

Logical Tensor Offset	Input Channel (C)	Input Depth (D)	Input Height (H)	Input Width (W)	→	Input Tensor Offset	Transformed Channel (C)	Transformed Depth (D)	Transformed Height (H)	Transformed Width (W)	Transformed C-vector (Cvec)
0	0	0	0	0	→	0	0	0	0	0	0
1	0	0	0	1	→	2	0	0	0	1	0
2	0	0	1	0	→	4	0	0	1	0	0
3	0	0	1	1	→	6	0	0	1	1	0
4	1	0	0	0	→	1	1	0	0	0	0
5	1	0	0	1	→	3	1	0	0	1	0
6	1	0	1	0	→	5	1	0	1	0	0
7	1	0	1	1	→	7	1	0	1	1	0
8	2	0	0	0	→	8	0	0	0	0	1
9	2	0	0	1	→	10	0	0	0	1	1
10	2	0	1	0	→	12	0	0	1	0	1
11	2	0	1	1	→	14	0	0	1	1	1

The tensor offsets in this table are logical, not address offsets and are thus independent of data type. The transformation typically implicitly adds zero padding to the data. As such, not all transformed output logical offsets are mapped from a given input logical offset.

2.6.4.5. Input Layout Transform Hardware

The input tensor layout transform and folding operations described in this section can be done on the FPGA AI Suite when the layout transform is enabled in the IP architecture file.

The hardware implementation assumes that the input tensors are in HWC format, and that the data elements are either FP16 or U8 format. The hardware implementation of the input transform supports input folding for any feature, stride, and padding values.

When active, the layout transform hardware folds the input tensor and converts it to the CHWCvec format as described in Input Feature Tensor In-Memory Format on page 33. If configured for U8 inputs, the data elements are also converted to FP16 format before tensors are sent downstream for inference.

Use the hardware layout transform with the `--ffolding_option 1` compiler option described in "Compilation Options (`dla_compiler` Command Options)" in the *FPGA AI Suite Compiler Reference Manual*. The layout transform hardware does not currently support multi-batch inputs (N>1) or 5-dimensional input tensors. Scale and shift values are also not applied in the hardware layout transform. You must apply scale and shift values to inputs before inferencing.

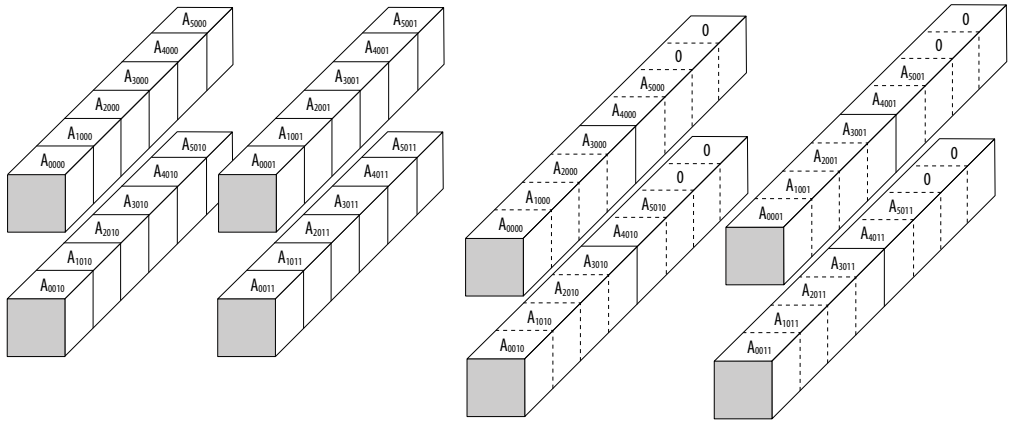
2.6.5. Output Tensor In-Memory Format

The output tensor in-memory format is similar to the input tensor in-memory format described in Input Feature Tensor In-Memory Format on page 33. However, the output tensor is padded to the nearest multiple of KVEC rather than CVEC, with the padding being done at the boundaries between FPGA AI Suite IP outputs rather than strictly at the edge of the logical tensor output.

While the logical tensor output might be a single tensor, the FPGA AI Suite IP might compute this output as one single output or by slicing the output into smaller pieces.

Each scenario can result in a different output layout, with the first resulting in padding only on the boundary of the tensor as shown in the following figure:

Figure 5. Output Tensor In-Memory Layout Without Slicing

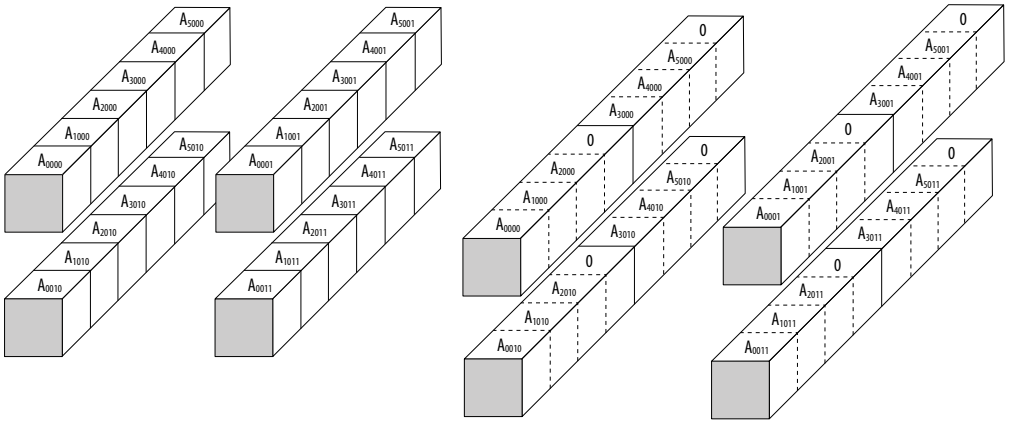


0	0	0	0	Base_addr + 56
0	0	0	0	Base_addr + 48
A_{5011}	A_{4011}	A_{5010}	A_{4010}	Base_addr + 40
A_{5001}	A_{4001}	A_{5000}	A_{4000}	Base_addr + 32
A_{3011}	A_{2011}	A_{3010}	A_{2010}	Base_addr + 24
A_{3001}	A_{2001}	A_{3000}	A_{2000}	Base_addr + 16
A_{1011}	A_{0011}	A_{1010}	A_{0010}	Base_addr +8
A_{1001}	A_{0001}	A_{1000}	A_{0000}	Base_addr
6	4	2	0	

Sample memory layout
(Byte addressed, 64 bit words, little endian)
CVEC=2, KVEC=4

The second scenario results in padding within the logical tensor as well as on the boundary of the logical tensor as shown in the following figure:

Figure 6. Output In-Memory Layout With Slicing



0	A_{5011}	A_{4011}	A_{3011}	Base_addr + 56
0	A_{5010}	A_{4010}	A_{3010}	Base_addr + 48
0	A_{5001}	A_{4001}	A_{3001}	Base_addr + 40
0	A_{5000}	A_{4000}	A_{3000}	Base_addr + 32
0	A_{2011}	A_{1011}	A_{0011}	Base_addr + 24
0	A_{2010}	A_{1010}	A_{0010}	Base_addr + 16
0	A_{2001}	A_{1001}	A_{0001}	Base_addr +8
0	A_{2000}	A_{1000}	A_{0000}	Base_addr
6	4	2	0	

Sample memory layout
(Byte addressed, 64 bit words, little endian)
KVEC=4

To enable the plugin to determine the appropriate output layout and offsets, the compiler provides a `DlaRuntimeOutputConfiguration` object. One of the fields of this object is the `output_tensor_mapping` field, which provides a mapping from FPGA AI Suite IP tensor output to logical tensor output.

To show how the IP output tensor data is mapped to the logical output tensor for a compiled graph, the FPGA AI Suite compiler creates the following CSV files:

- `output_transform_dump_<graph-name>.csv`

  This file describes the tensor shape and offsets.

- `output_transform_mapping_<graph-name>.csv`

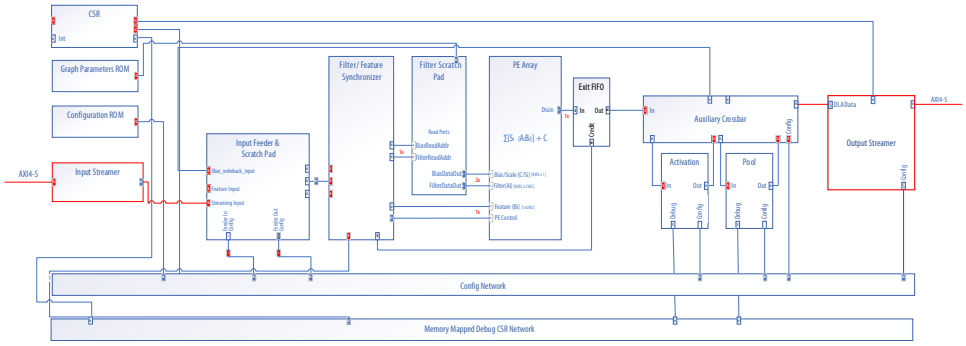
  This file shows the element-wise mapping of the FPGA AI Suite output tensor to the logical output tensor. This mapping is in the inverse of the input mapping described in Input Transform Mapping on page 35. Refer to the example in that topic for a description of the transform mapping CSV file.

2.7. Feature Input and Output Streaming

The FPGA AI Suite can be configured to accept AXI-streaming (AXI4-Stream) input features, produce AXI-streaming (AXI4-Stream) output features, or both. Only graphs that fit entirely on the FPGA device are supported, You cannot use the OpenVINO HETERO:FPGA,CPU plugin with streaming.

When input and output streaming is enabled, the system architecture can be described as shown in the following diagram:

Figure 7. FPGA AI Suite IP with Input and Output Streaming



For more information about the configuration and operation of these streaming interfaces, refer to the following sections:

- Input Streaming on page 40
- Output Streaming on page 42

2.7.1. Input Streaming

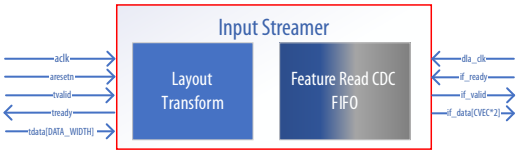
When input streaming is configured in the IP system architecture as described in Parameter Group: input_stream_interface on page 29, and the `Activate_streaming` DMA control register is set to 1, the FPGA AI Suite IP accepts inputs through the AXI4-Stream signals that are exposed at the top-level instead of being memory mapped. The exposed signals are as follows:

Table 8. FPGA AI Suite IP AXI4-Stream Input Interface Signals

Signal	Source	Width	Description
ACLK	Clock	1	Data source clock
ARESETn	Reset	1	Data source active-low reset
TVALID	Data source	1	Input signal that indicates whether the values in TDATA are valid.
TREADY	FPGA AI Suite IP	1	Output signal that indicates whether the FPGA AI Suite IP is ready to accept data.
TDATA	Data source	DATA_WIDTH	Input data bus.

Schematically, the input streaming component is constructed as follows:

Figure 8. Input Streamer Schematic View



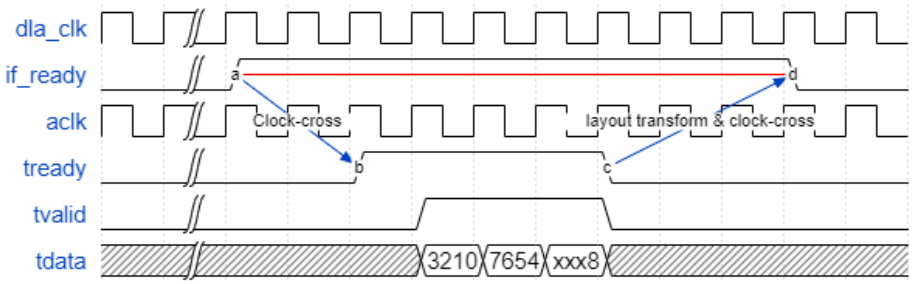
The streamed input tensor format must be in HWC, where `channels` is the fastest-changing dimension. The HWC tensors are internally folded and vectorized to CHWCvec tensors that the PE array can ingest (as described in Input Feature Tensor In-Memory Format on page 33). Data also internally crosses from the source clock domain to the FPGA AI Suite IP internal clock domain.

This subset of the AXI4 streaming protocol signals implements a streaming interface where transfers take place whenever the TREADY and TVALID signals are asserted. The input streaming interface does not implement the TSTRB or TKEEP signals, which means that all data in a valid TDATA signal transfer must be valid unless it is the last transfer. The TLAST signal is also unused at the input, because the FPGA AI Suite IP tracks the state of the transfer internally.

Any data beyond the boundary of the input tensor in the final transfer of an input feature that is not a multiple of `DATA_WIDTH` is ignored. The data stream producer is responsible for padding features, if needed, so that adjacent features to not share the same data transfer at the boundary.

The following timing example shows a 3x3x1 input tensor with monotonically increasing pixel values and DATA_WIDTH of 4 bytes. Note the padding in the third transfer.

Figure 9. Streaming Input Waveform



The TREADY signal is asserted by the FPGA AI Suite IP whenever the IP is ready for a new input feature and streaming has been activated in the CSR. The TREADY signal comes from the input streamer module and is first asserted once the FPGA AI Suite IP is configured and the input streamer FIFO is not full. The FPGA AI Suite IP accepts the input feature as long as there is space in the FIFO and no backpressure from the downstream system, which means the TREADY signal can be deasserted mid input feature.

When the input streaming interface is enabled, it requires the input layout transform to be enabled and configured as described in Parameter Group: input_stream_interface on page 29.

When streaming data is received, it is converted from HWC format, where channels is the fastest-changing dimension in the memory representation, to CHWCvec format, where the input has been vectorized into Cvec-lines that can be input to the PE array.

The layout transform can be configured to either accept FP16 input data or uint8 input data that is converted internally to FP16.

The input streaming module handles clock-domain crossing from the input stream clock domain to that of the FPGA AI Suite IP and also handles width conversion from DATA_WIDTH to CVEC.

2.7.2. Output Streaming

Use the output streaming component to stream output data from FPGA AI Suite IP to a downstream module using an AXI4-Stream interface.

When the output streaming interface enabled, it produces data in HWC format (where channels is the fastest changing dimension), on the output bus. The bus width is configured in the architecture parameters as described in Parameter Group: output_stream_interface on page 30. The data is converted from the FPGA AI Suite IP internal clock domain to the clock domain of the AXI4-Stream receiver.

The following signals implement the output AXI4-Stream interface:

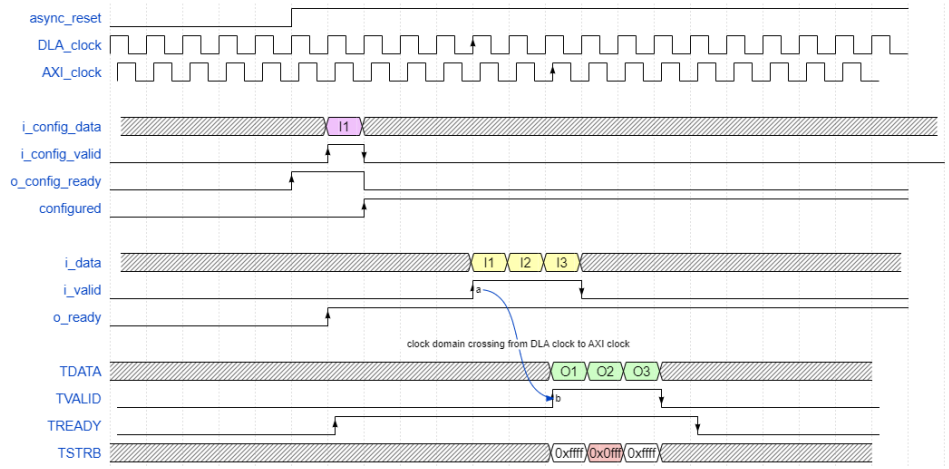
Table 9. FPGA AI Suite IP AXI4-Stream Output Interface Signals

Signal	Source	Width	Description
`clk_axi`	Clock	1	Downstream AXI clock
`i_resetn_axi`	Reset	1	Downstream AXI active-low reset
`o_ostream_axi_t_valid`	FPGA AI Suite IP	1	Output signal that indicates whether the values in TDATA are valid
`i_ostream_axi_t_ready`	Downstream AXI User	1	Output signal that indicates whether the AXI receiver is ready to accept data
`o_ostream_axi_t_data`	FPGA AI Suite IP	`DATA_WIDTH`	Output data bus
`o_ostream_axi_t_strb`	FPGA AI Suite IP	`DATA_WIDTH/8`	Output signal that indicates which bytes of TDATA are valid
`o_ostream_axi_t_last`	FPGA AI Suite IP	1	Indicates the last transmission for the current frame

Data from the FPGA AI Suite IP is produced in CVEC multiples. When the number of output channels is not a multiple of CVEC, the last AXI4-Stream transaction for a single pixel in height/width dimensions might have only some valid elements that are indicated by the `o_ostream_axi_t_strb` signal.

Consider the following example: Assume the output data tensor has a shape of 3x3x10 (HWC), with CVEC = 8 elements (each 16 bits), and AXI `TDATA_WIDTH` = 128 bits. For each pixel in the 3x3 surface, we need to produce 10 channels, which fit in two AXI transactions. The first transaction has all valid elements (`o_ostream_axi_t_strb` = 0xffff). The second transaction has only two valid elements and the rest are zeros (`o_ostream_axi_t_strb` = 0x000f). The downstream receiver of AXI transactions is responsible for intercepting the `o_ostream_axi_t_strb` signal and processing only the valid elements.

The figure that follows demonstrates an example transaction flow with 3 input transactions (I1, I2 and I3, each 128 bits) and three output AXI transactions (O1, O2, and O3, each 128 bits). In this example the first and third 128-bit transactions are all valid data (`o_ostream_axi_t_strb` =0xffff), but the second transaction has only 6 valid elements (16 valid bytes). For the second transaction, the `o_ostream_axi_t_strb` =0x0fff (marked in red color). Notice that data transactions happen only when the block is configured and ready to produce output.



2.8. DDR-Free Operation

To avoid use of external memory for storing graph weights and FPGA AI Suite IP configurations during inference, you can store these parameters within the IP using on-chip memory.

In this case, three types of memory initialization files (`.mif`) are required:

- `ddrfree_filter_hw*.mif`: Contains the graph filters.
- `ddrfree_bias_scale_hw*.mif`: Contains the graph biases and scaling factors.
- `ddrfree_config.mif`: Stores the FPGA AI Suite instructions for the compiled graph.

These files are generated using the `dla_compiler` tool that takes the architecture definition (`.arch` file) and the target neural network graph as inputs.

A total of K_VECTOR DDR-free filter `.mif` files are generated. Each file is postfixed with an integer representing which PE the filter is to be loaded to.

Similarly, a total of K_VECTOR DDR-free bias and scale files are generated, using the same postfix scheme to indicate the PE that should load this file.

The FPGA AI Suite IP requires a stream of instructions that describe the order in which convolutions, activations, and other operations must be performed. This instruction stream is stored in a single `.mif` file (`ddrfree_config.mif`).

These files are used to initialize read-only memories while building an FPGA bitstream, which means that bitstreams are graph-specific for DDR-free operation.

You must also enable input and output streaming when using DDR-free operation. The graph must have a large enough stream buffer depth to accommodate all of the intermediate results during inference.

The `dla_build_example_design.py` command can build design examples with DDR-free operation. The only streaming design example option currently supported is the `0_STREAMING` option. The design example created by this option targets an Agilex 7 FPGA I-Series Development Kit (DK-DEV-AGI027RBES).

To compile a bitstream with a DDR-free architecture, specify the directory that contains the DDR-free `.mif` files with the `–parameter_rom_dir` option of the `dla_build_example_design.py` command.

For details about creating the `.mif` file required for DDR-free operation, refer to "Generating Artifacts for DDR-Free Operation" in the *FPGA AI Suite Compiler Reference Manual*.

3. FPGA AI Suite IP Generation Utility

The FPGA AI Suite IP generation utility reads an input Architecture Description File (`.arch`) and places generated IP into an IP library that can be imported into Platform Designer or used directly in a pure RTL design. The generated IP is configured based on the Architecture Description (`.arch`) File, which defines the following items:

- Top RTL parameters are set to reflect values in the Architecture Description File
- The Discovery ROM MIF is created to specify an architecture hash and FPGA AI Suite compiler version string

The generation utility also copies all the required RTL files into a single directory. As convenience options, the utility can also help to create `.qsf` files and wrappers for the PCIe Example Design or to compile an instance of the IP with a dummy BSP interface.

The IP generation utility generates either an unlicensed or a licensed copy of the IP:

Unlicensed IP The unlicensed IP has a limit of 10000 inferences. After 10000 inferences, the unlicensed IP refuses to perform any additional inference and a bit in the CSR is set. For details about the CSR bit, refer to DMA Descriptor Queue on page 57.

Licensed IP The licensed IP has no inference limitation

The IP generation utility checks for an FPGA AI Suite IP license before generating the IP. The utility prints messages to `stdout` that show the license status.

You can use either licensed and unlicensed IP for bitstream generation so that you can fully test your design during the evaluation process.

3.1. IP Generation Utility Execution Flows

The IP generation utility (`dla_create_ip` command) has the following flows:

- Creating an IP Directory (`--flow create_ip`)

  Use this flow to create a new FPGA AI Suite IP library directory, generate an IP, and place it in the library.
- Adding an Architecture to an IP Directory (`--flow add_arch`)

  Use this flow to generate an IP and place it in an existing FPGA AI Suite IP library directory.
- Listing Architectures in an IP Directory (`--flow list`)

  Use this flow to list the IPs in an existing IP library directory.
- Removing an Architecture from an IP Directory (`--flow remove_arch`)

  Use this flow to remove an IP from an existing IP library directory.

Creating an IP Directory

This flow is typically first flow that you use. It creates an FPGA AI Suite IP library directory and adds an architecture as follows:

1. Creates the IP library directory and copies all contents from <*ai_suite_root*>/fpga/ip_template/ into the new directory. The copied folder (<*ip_template*>) contains some basic Tcl scripts and Platform Designer IP configurations.
2. Creates a <*ip_directory*>/*Verilog* directory and copies over RTL files that are common to any generated FPGA AI Suite IP architecture.

   These files are listed in `static_files.tcl`, which is used to ensure that other flows (the Platform Designer flow and design assembly flows outside Platform Designer) have access to the list of RTL source files.
3. Creates the following directory for each architecture-family pair to add to the IP library:

   <*ip_directory*>/Verilog/<*architecture*>_<*family*>

   This location stores architecture-specific files.
4. Invokes an internal utility to read the architecture description file (`.arch`).

   Output files are architecture-specific and are copied to the <*ip_directory*>/Verilog/<*architecture*>_<*family*> directory.
5. For the specific architecture, creates a `generated_files.tcl` file and a `dla_ip.qsf` file in the <*ip_directory*>/Verilog/<*architecture*>_<*family*> directory.

Adding an Architecture to an Existing IP Directory

This flow adds a new architecture to an existing IP library directory as follows:

1. Creates a <*ip_directory* >/*Verilog* directory and copies over RTL files that are common to any generated FPGA AI Suite IP architecture.

   These files are listed in `static_files.tcl`, which is used to ensure that other flows (the Platform Designer flow and design assembly flows outside Platform Designer) have access to the list of RTL source files.
2. Creates the following directory for each architecture-family pair to add to the IP library:

   <*ip_directory*>/Verilog/<*architecture*>_<*family*>

   This location stores architecture-specific files.
3. Invokes an internal utility to read the architecture description file (`.arch`).

   Output files are architecture-specific and are copied to the <*ip_directory*>/Verilog/<*architecture*>_<*family*> directory.
4. For the specific architecture, creates a `generated_files.tcl` file and a `dla_ip.qsf` file in the <*ip_directory*>/Verilog/<*architecture*>_<*family*> directory.

Listing Architectures in an IP Directory

This flow lists all available architectures in the IP library directory. The utility looks for all <*architecture_family*> folders and displays them.

Removing an Architecture from an IP Directory

This flow removes an architecture from an IP library directory. The utility looks for a <*architecture_family*> folder and removes it.

3.2. IP Generation Utility Inputs

The only external inputs to the IP creation flow are FPGA AI Suite architecture description files. The file format for architecture descriptions files is described in Architecture Description File Format for Instance Parameterization on page 16.

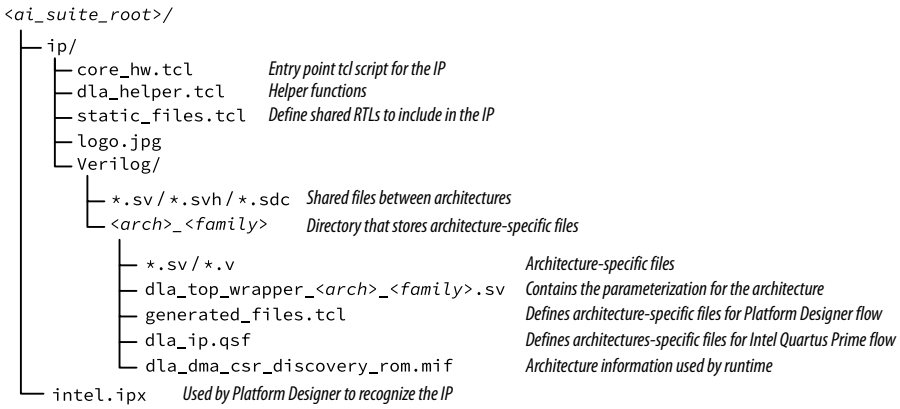
The internal inputs to the scripts are:

- <*ai_suite_root*>/fpga/<*modules*>. Module folders that contain RTL source files.
- <*ai_suite_root*>/fpga/ip_template. Contains basic files used to create the IP in the Platform Designer flow.

3.3. IP Generation Utility Outputs

The IP generation utility (`dla_create_ip` command) generates an FPGA AI Suite IP directory as its output.

The default directory structure is as follows:



3.4. IP Generation Utility Command Line Options

The actions of the IP generation utility (`dla_create_ip` command) are controlled primarily by the `--flow` option. Depending on your choice for the `--flow` option, you might need to specify other command options.

The `--flow` Option

This option specifies the IP generation utility execution flow to run.

Syntax `--flow <flow>`

Where <*flow*> specifies the execution flow to run.

Valid values

- `create_ip`

  For details, refer to The --flow create_ip Flow on page 49.
- `add_arch`

  For details, refer to The --flow add_arch Flow on page 50.
- `list`

  For details, refer to The --flow list Flow on page 51.
- `remove_arch`

  For details, refer to The --flow remove_arch Flow on page 51.

Default Value `create_ip`

The `--arch` Option

This option specifies one or more FPGA AI Suite architecture description files (`.arch`) to use in the flow.

If you specify this option, you cannot specify the `--arch_dir` option.

Syntax `--arch <architecture> [<architecture2>] [<architecture3] ...`

For the `create_ip` and `add_arch` flows, specify a path to the architecture description file or files to add to an IP library.

For the `remove_arch` flow, specify the name of the architecture description file or files to remove from an IP library.

The `--arch_dir` Option

Use this option with the `create_ip` and `add_arch` flows to specify the path to a directory that contains architecture description files. If you specify this option, the IP generation utility adds each architecture in the specified directory to the FPGA AI Suite IP library.

If you specify this option, you cannot specify the `--arch` option.

Syntax `--arch_dir <path_to_arch_folder>`

The `--ip_dir` Option

This option specifies the path of the FPGA AI Suite IP library directory to use in the flow.

In the `create_ip` flow, this directory is created if it does not exist. For all other flows, the specified IP library directory must exist.

Syntax `--ip_dir <ip_directory>`

Default Value `<ai_suite_root>/ip`

The `--unlicenced/--licenced` Options

Generate an unlicensed or licensed copy of the FPGA AI Suite IP:

- **Unlicensed IP:** Unlicensed IP has a limit of 10000 inferences. After 10000 inferences, the unlicensed IP refuses to perform any additional inference and a bit in the CSR is set. For details about the CSR bit, refer to DMA Descriptor Queue on page 57.
- **Licensed IP:** Licensed IP has no inference limitation.

If you generate a licensed copy of the IP but do not have a license, then Quartus cannot to generate a programming bitstream for your FPGA.

If neither option is specified, then the tool queries the `lmutil` license manager to determine the correct option.

Syntax `--unlicenced`

`--licenced`

The `--overwrite` Option

For the `create_ip` flow, use this option to overwrite the IP library directory when you specify an existing IP library directory.

If you specify an existing directory with the `create_ip` flow and do not specify the `--overwrite` option, you are asked if you want to overwrite the directory.

Syntax `--overwrite`

3.4.1. The `--flow create_ip` Flow

The default flow for the IP generation utility (`dla_create_ip` command) is the `--flow create_ip` flow. This flow creates a new IP library directory (`<ip_directory>`).

To generate a new IP, provide one of the following options as the location of architecture description file or files:

- Use the `--arch` option to specify one or more architecture description files (`.arch`).
- Use the `--arch_dir` option to specify a directory that contains one or more architecture description files. All architecture description files in the directory are added to the IP library.

Use the `--ip_dir` option to specify the output FPGA AI Suite IP library directory. If unspecified, the default value of `<ai_suite_root>/ip` is used as the IP library directory.

You can specify the `--overwrite` option to overwrite the IP directory and create a new IP if the IP directory already exists. If the `--overwrite` option is not set, the script prompts you if want to overwrite the IP library directory.

Usage Synopsis

```
dla_create_ip [--flow create_ip] \
   --arch <path to .arch File> [<path to .arch file> ...] \
   [--ip_dir <ip_directory>] \
   [--overwrite]
```

```
dla_create_ip [--flow create_ip] \
   --arch_dir=<path to  directory with .arch files> \
   [--ip_dir <ip_directory>] \
   [--overwrite]
```

Sample Call

```
dla_create_ip --flow create_ip \
   --arch=$COREDLA_ROOT/example_architectures/A10_Generic.arch \
   --overwrite \
   --ip_dir ./ip
```

Sample Output

```
============================================================
            Start IP Creation Flow
============================================================
Generate file path ip/intel_ai_ip/Verilog/Generic_A10
============================================================
            IP Creation finished
============================================================
```

3.4.2. The `--flow add_arch` Flow

The `--flow add_arch` flow adds additional generated IP to an existing IP library directory (previously created with the `--flow create_ip` flow).

If you add an architecture that already exists in the IP library directory, the architecture is overwritten with newly generated RTL files.

Usage Synopsis

```
dla_create_ip --flow add_arch \
   --arch <path to .arch file> \
   [--ip_dir <ip_directory>]
```

Sample Call

```
dla_create_ip --flow add_arch \
   --arch $COREDLA_ROOT/example_architectures/A10_Generic.arch \
   --ip_dir ./ip
```

Sample Output

```
============================================================
       Adding Generic_A10 to the existing IP
============================================================
```

```
Generate file path ./ip/intel_ai_ip/Verilog/Generic_A10

================================================================
       Finished adding Generic_A10
================================================================
```

3.4.3. The `--flow list` Flow

Use `--flow list` flow to list all available architecture in an IP library folder. An architecture description file (`.arch`) specifies the target FPGA family device.

If an architecture has been added to the IP library with different FPGA families, those architecture–FPGA family combinations are displayed as different architectures in the IP folder.

Usage Synopsis

```
dla_create_ip --flow list [--ip_dir <ip_directory>]
```

Sample Call

```
dla_create_ip --flow list --ip_dir $COREDLA_ROOT/example_ip_cores
```

Sample Output

```
================================================================
     Listing available architectures from <ai_suite_rootdir>/ip
================================================================
1x1x16x16_fp11_sb30240_reluk16_poolk16_A10
1x1x16x16_fp11_sb30240_reluk16_A10
```

3.4.4. The `--flow remove_arch` Flow

Remove an architecture from the IP library with the `--flow remove_arch` flow.

If you specify an architecture description file that does not exist in the IP library, the IP generation utility returns a warning.

Usage Synopsis

```
dla_create_ip --flow remove_arch \
   --arch <path to .arch file> \
  [--ip_dir <ip_directory>]
```

Sample Call

```
dla_create_ip --flow remove_arch \
   --arch A10_Generic.arch \
   --ip_dir ./ip
```

Sample Output

```
================================================================
removing Generic_A10 from existing IP
================================================================
```

4. FPGA AI Suite Ahead-of-Time Splitter Utility

Typically, the FPGA AI Suite runtime OpenVINO plugin provisions the control-and-status registers (CSR) and memory needed to run inference. If you want to run inference on the FPGA AI Suite IP without using the OpenVINO Inference Engine, you can use the ahead-of-time (AOT) splitter utility (`dla_aot_splitter`) to extract the raw data that is provisioned in memory.

An example system where you might not want to use the OpenVINO Inference Engine is when you have an embedded soft processor that manages a FPGA AI Suite IP instance in the FPGA fabric. The embedded system can provision DDR memory with the files generate by the AOT splitter and configure the IP to use the provisioned data.

Review the source code provided in `dla_aot_splitter_example` to see how to use the files that are generated by the splitter utility in an application that performs inference without using the OpenVINO Inference Engine.

4.1. Files Generated by the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility

The FPGA AI Suite AOT splitter utility converts a model and its associated input or inputs compiled with the `dla_compiler` command into a set of files. The model must target the OpenVINO `HETERO:FPGA` plugin.

The AOT splitter utility generates the following files:

- `input.bin`

  This file is the layout transformed model input (or inputs) intended for direct consumption by the FPGA AI Suite IP.

  The splitter utility also provides `input.mem` as a plain-text representation of `input.bin`. The plain-text file is suitable to be interpreted as C/C++ constants.

- `config.bin`

  This file is the configuration word data stream that the FPGA AI Suite IP uses to execute the model.

  The splitter utility also provide `config.mem` as a plain-text representation of `config.bin`. The plain-text file is suitable to be interpreted as C/C++ constants.

- `filter.bin`

  This file contains the transformed weights of the model.

  The splitter utility also provide `filter.mem` as a plain-text representation of `filter.bin`. The plain-text file is suitable to be interpreted as C/C++ constants.

- `inter_size.mem`

  This file defines the intermediate buffer size. The intermediate buffer is a temporary area of memory that must be allocated but not initialized.

- `output_size.mem`

  This file defines the output buffer size. The output buffer is where the FPGA AI Suite IP writes the results of the output layers of the model.

- `arch_build.bin`

  This file contains the architecture file hash and build version.

  The splitter utility also provide `arch_build.mem` as a plain-text representation of `filter.bin`. The plain-text file is suitable to be interpreted as a C/C++ constant.

4.2. Building the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility

The FPGA AI Suite AOT splitter utility is provided along with the example runtime component. The utility is built when you build the example runtime.

To build the AOT splitter utility:

1. Build the FPGA AI Suite example runtime with the following commands:

   ```
   cd $COREDLA_ROOT/runtime
   rm -rf build_Release
   ./build_runtime.sh --target_de10_agilex
   ```

   When the build finishes, the AOT splitter utility command (`dla_aot_splitter`) is in the following directory:

   ```
   $COREDLA_ROOT/runtime/build_Release/dla_aot_splitter/
   ```

The `dla_aot_splitter` executable generated by building the example runtime remains the same regardless of any board- or device-specific options that you might specify. However, the files generated by the utility are specific to the FPGA device family and FPGA AI Suite architecture that you specify when compiling your inference model with the `dla_compiler` command.

4.3. Running the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility

Before you run the FPGA AI Suite AOT splitter utility, you must have the following files ready:

- A model compiled by the `dla_compiler` command. The model must target the `HETERO:FPGA` format and the output format must be `open_vino_hetero`.

  For example, the following command compiles a model that targets a board with an Agilex 7 device to a file called `RN50_Performance_b1.bin`:

  ```
  cd $COREDLA_ROOT/demo/models/public/resnet-50-tf/FP32
  dla_compiler \
    --march $COREDLA_ROOT/example_architectures/AGX7_Performance.arch \
    --network-file ./resnet-50-tf.xml \
    --foutput-format=open_vino_hetero \
  ```

```
  --o $COREDLA_ROOT/demo/RN50_Performance_b1.bin \
  --branch=1 \
  --fanalyze-performance
```

- The input or inputs for model inference.

The AOT splitter utility outputs the artifacts to the current working directory. The command line options must have a compiled model, model inputs, and a path to the splitter utility plugins XML file.

To run the AOT splitter utility:

1. Change directories to where you want the output files from the AOT splitter utility to go.
2. Run the splitter utility command as follows:

```
$COREDLA_ROOT/runtime/build_Release/dla_aot_splitter/dla_aot_splitter \
  -cm <full_path_to_compiled_model_file> \
  -i < XXX-delimited list of inference input files> \
  -plugins <full_path_to_plugins_aot_splitter.xml> \
  [-bgr]
```

   The optional `-bgr` command option tells the AOT splitter to reverse the R, G, and B input channels for images before the channels are passed to the model.

   For example:

```
runtime/build_Release/dla_aot_splitter/dla_aot_splitter \
  -cm $COREDLA_ROOT/demo/RN50_Performance_b1.bin \
  -i $COREDLA_ROOT/demo/sample_images/val_00000000.bmp \
  -plugins \
   runtime/dla_aot_splitter/dla_aot_splitter_plugin/plugins_aot_splitter.xml
  -bgr
```

Running the splitter generates the following files:

- `arch_build.mem` and `arch_build.bin`
- `config.mem` and `config.bin`
- `filter.mem` and `filter.bin`
- `input.mem` and `input.bin`
- `inter_size.mem`
- `output_size.mem`

  For a description of the files, refer to Files Generated by the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility on page 52.

4.4. FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility Example Application

To learn more about how to use the files generated by the AOT splitter utility, review the `dla_aot_splitter_example` example application. This example application uses a PCIe interface to interact with either the Terasic DE10-Agilex Development Board.

Building the AOT Splitter Utility Example Application

You build the AOT splitter utility example application with the `build_runtime.sh` script with the `-aot_splitter_example` option.

The CMake build for the example runs the `dla_compiler` command on a model and the `dla_aot_splitter` command on the compiled model and model input.

The example application is built for a specific target board as follows:

- **Terasic DE10-Agilex Development Board**

```
export AOT_SPLITTER_EXAMPLE_MODEL=<path/to/model.xml>
export AOT_SPLITTER_EXAMPLE_INPUT=<path/to/image.bmp>
sh build_runtime.sh -aot_splitter_example -target_de10_agilex
```

Running the AOT Splitter Utility Example Application

Remember: The AOT splitter utility example application supports only the Terasic DE10-Agilex Development Board.

After you build the example application, you can review its source code to learn how to apply the files generated by the splitter to your system.

Run the example application with the following commands:

```
cd $COREDLA_ROOT/runtime/build_Release/dla_aot_splitter/
./dla_aot_splitter_example/dla_aot_splitter_example
```

5. CSR Map and Descriptor Queue

The CSR interface uses a 32-bit data path in which all accesses are aligned to 32 bits; however the address is a byte address. The size of the CSR address space is 2048 bytes (11 bit addressable). The regions within the CSR address space are listed in the table that follows.

Table 10. CSR Address Space Regions

Base Address	Feature
0x000	Discovery ROM (512 word)
0x200	Interrupt Control
0x210	DMA Descriptor Queue
0x220	DMA Control Registers
0x240	Performance Registers
0x250	Debug Network Registers
0x260	DMA License Register on page 60
0x264	DMA Transaction Counters on page 60

Register and Bit Attribute Definitions

The following notation describes the CSR registers.

Table 11. Register and Bit Attribute Definitions

Attribute	Expansion	Description
RW	Read/Write	This bit can be read or written by software.
RO	Read Only	The bit is set by hardware only. Software can only read this bit. Writes have no effect.
RW1C	Read/Write 1to Clear	Software can read or clear this bit. Software must write 1 to clear this bit. Writing zero to an RW1C bit has no effect. A multibit RW1C field can exist. In that case, all bits in the field are cleared if a 1 is written to any of the bits.
RsvdZ	Reserved and zero	Reserved for future RW1C implementations. When you write to a register with RsvdZ bits, only write zeros to these bits.

5.1. Discovery ROM

The discovery ROM stores metadata. The metadata includes a hash for the architecture that the IP corresponds to and the FPGA AI Suite version that was used to create the IP.

The host runtime can use this information to determine whether the incoming inference job can be run on the IP instances. For example, if the architectures do not match each other, then inference is not possible.

The layout of the discovery ROM is as follows:

Table 12. Discovery ROM Layout

Base Byte Address	Length (in bytes)	Feature
0x000	16	Hash of the Architecture Description File (`.arch`)
0x010	32	Human-readable FPGA AI Suite version string

5.2. Interrupt Control

The interrupt control feature registers are as follows:

Table 13. Interrupt Control Feature Registers

Register	Offset	Attribute	Description
ICR	0x000	RW1C	DMA Interrupt control register
IMR	0x004	RW	DMA Interrupt mask register

The DMA optionally generates level sensitive interrupt signals in response to various events.

The hardware sets the corresponding bit within the ICR register whenever such an event occurs.

An interrupt is generated upon a 0-to-1 transition of a bit within ICR only if the corresponding bit in the IMR is set to one. A 0-to-1 transition of a bit within the IMR also generates an interrupt if the corresponding bit within the ICR is set to 1.

Table 14. Interrupt Control Register (ICR) Fields

Field	Bit	Description
Reserved	31:2	RsvdZ (Reserved; software must write 0)
Inference_complete	1	Indicates that an inference request has completed
Error	0	Indicates that an error condition has been triggered

Table 15. Interrupt Mask Register (IMR) Fields

Field	Bit	Description
Reserved	31:2	RsvdZ (Reserved; software must write 0)
Inference_complete_mask	1	Set to one to enable interrupt generation on inference completion
Error_mask	0	Set to one to enable interrupt generation on error condition

5.3. DMA Descriptor Queue

The DMA contains a single descriptor FIFO for enqueuing inference requests. Descriptors potentially require multiple register writes and are added to the queue upon writing to the `desc_input_output_base_addr` register.

The `desc_cfg_filter_base_addr` and `desc_cfg_num_words` are registers that hold their value.

If you already enqueued a DMA descriptor and want to enqueue another descriptor with the same values for the `desc_cfg_filter_base_addr` and `desc_cfg_num_words` registers, then write to the `desc_input_output_base_addr` register.

If you want to change the `desc_cfg_filter_base_addr` and `desc_cfg_num_words` registers for the next descriptor, then you must set new values before writing to the `desc_input_output_base_addr` register.

Table 16. DMA Descriptor Queue Registers

Register	Offset	Attribute	Description
`desc_cfg_filter_base_addr`	0x000	RW	Base address pointer for the configuration buffer and for the filter buffer. The filters are located at `desc_cfg_filter_base_addr` + `desc_cfg_num_words`, which is encoded in the address provided to the filter reader as configuration data. Must be aligned to a multiple of the DDR word size.
`desc_cfg_num_words - 2`	0x004	RW	Length of the configuration buffer - 2, in config words (64 bits – 32 for instruction, 32 for data)
`desc_input_output_base_addr`	0x008	RW	Base address pointer for the input feature data to be used for inference, and also the base address to place the output inference results into. Must be aligned to a multiple of the DDR word size. Writing to this register enqueues a descriptor into the internal DMA descriptor queue.
`desc_diagnostics`	0x00C	RO	This register is useful for debugging. Production software should not need to read from this. Bit 0: Asserts if the descriptor queue overflows; this is a sticky bit which only clears after reset. Bit 1: Descriptor queue is full or almost full. Bit 2: Asserts if the inference limit for an unlicensed IP is reached. When asserted, inference requests are rejected. All other bits are reserved.

5.4. DMA Control Registers

Table 17. DMA Control Registers

Register	Offset	Attribute	Description
`Intermediate_ddr_base_address`	0x000	RW	Base address for the DDR intermediate data. This is a shared address across all graphs. Only required to be set once upon startup. Must be aligned to a multiple of the DDR word size.
`Inference_completion_count`	0x004	RO	Number of inference request completions by the FPGA AI Suite IP.
`IP_reset`	0x008	RW	Write any non-zero value to this address to trigger a reset of the FPGA AI Suite IP. The value is automatically cleared upon reset. Reading from this register always returns `0`.
`Activate_streaming`	0x00C	RW	When streaming is enabled in the architecture, writing "1" to this register makes the FPGA AI Suite IP begin queuing descriptors and start listening for streaming inputs.

continued...

Register	Offset	Attribute	Description
			Writing "0" stops queuing descriptors and turns off the input streaming interface.

5.5. Performance Registers

Hardware counters are provided to measure how many clock cycles that the IP is active. A job is considered active after the first word of its descriptor is read from the descriptor queue. A job is considered finished just before the done interrupt is raised and the completion count is updated.

The IP and supporting host form an elastic pipeline in which multiple jobs can be in flight. The IP tracks both the overall latency (for example, the length of time required to process 100 jobs) as well as the average latency for each of those jobs. The hardware tracks the total latency of every job but knowing the total number of jobs software can compute the average.

64-bit counters mitigate against overflow. There is no synchronization between reading the lower or upper 32 bits of a counter, therefore the software should not read the counters while the IP is active.

Table 18. Performance Registers

Register	Offset	Attribute	Description
Total clocks active (lower 32 bits)	0x000	RO	On each clock cycle, if any IP job is active, increment the counter by 1.
Total clocks active (upper 32 bits)	0x004	RO	Same as above.
Total clocks for all jobs (lower 32 bits)	0x008	RO	On each clock cycle, if there are *N* IP jobs active, increment the counter by *N*.
Total clocks for all jobs (upper 32 bits)	0x00C	RO	Same as above.

5.6. Debug Network Registers

The debug network has the following registers available from the CSR:

Table 19. Debug Network Registers

Register	Offset	Attribute	Description
DLA_DMA_CSR_OFFSET_DEBUG_NETWORK_ADDR	0x000	RO	Address that the debug network uses to issue a read request.
DLA_DMA_CSR_OFFSET_DEBUG_NETWORK_VALID	0x004	RO	Indicates that a read response has been received from the debug network.
DLA_DMA_CSR_OFFSET_DEBUG_NETWORK_DATA	0x008	RO	Data from debug network.

5.7. DMA License Register

Table 20. DMA License Register

Register	Offset	Attribute	Description
`license_flag`	0x000	RO	Indicates whether the IP is licensed: • `0`: unlicensed • `1`: licensed

5.8. DMA Transaction Counters

Hardware counters are provided to measure the number of data words accessed by the DMA from the external DDR memory.

The counter values are separated into input feature reads, input weights and biases reads, and output feature writes. The width of each memory word in bytes matches the `dma/ddr_data_bytes` value in the architecture description file.

Table 21. DMA Transaction Counter Registers

Register	Offset	Attribute	Description
Total number of input feature words read by the FPGA AI Suite IP (lower 32 bits)	0x000	RO	This counter is incremented by 1 for every input feature word transferred from the external memory to the IP DMA on the AXI4 read bus.
Total number of input feature words read by the FPGA AI Suite IP (upper 32 bits)	0x004	RO	Same as above.
Total number of input filter and biases words read by the FPGA AI Suite IP (lower 32 bits)	0x008	RO	This counter is incremented by 1 for every filter-bias word transferred from the external memory to the IP DMA on the AXI4 read bus.
Total number of input filter and biases words read by the FPGA AI Suite IP (upper 32 bits)	0x00C	RO	Same as above.
Total number of output feature words written by the FPGA AI Suite IP (lower 32 bits)	0x010	RO	This counter is incremented by 1 for every feature word written to the external memory by the IP DMA on the AXI4 write bus.
Total number of output feature words written by the FPGA AI Suite IP (upper 32 bits)	0x00C	RO	Same as above.

A. FPGA AI Suite IP Reference Manual Archives

For the latest and previous versions of this reference manual, refer to FPGA AI Suite IP Reference Manual. If an FPGA AI Suite software version is not listed, the reference manual for the previous software version applies.

B. FPGA AI Suite IP Reference Manual Document Revision History

Document Version	FPGA AI SuiteVersion	Changes
2025.02.24	2024.3	• Fixed typos in "Running the FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility" and "FPGA AI Suite Ahead-of-Time (AOT) Splitter Utility Example Application".
2024.12.16	2024.3	• For *Parameter: arch_precision* in "Parameter Group: Global Parameters", corrected the number of operations per DSP for Agilex 7 devices with FP13AGX precision • Fixed minor errors and typos.
2024.11.25	2024.3	• Updated "Model Performance" with Version 2024.3 values. • Many of the `layout_transform_params` parameters have changed. In most cases, the old parameters represented an exact value to apply while the new parameters represent the maximum value to apply. **Old Parameter** → **New Parmater** channels → max_channels feature_height → max_feature_height feature_width → max_feature_width feature_depth → max_feature_depth stride_height → max_stride_height stride_width → max_stride_width stride_depth → max_stride_depth pad_top → max_pad_top pad_left → max_pad_left pad_depth → max_pad_depth • Added the following new `layout_transform_params` parameters: — max_filter_width — max_filter_height — max_filter_depth • Removed the following `layout_transform_params` parameters: — output_channels — output_height — output_width — output_depth

continued...

Document Version	FPGA AI SuiteVersion	Changes
2024.09.06	2024.2	• Replaced occurrences of "memoryless" with "DDR-free". • Renamed "FPGA AI Suite Software Reference Model" to "Software Emulation of the FPGA AI Suite IP". • Replaced occurrences of "software reference model" with "software emulation model".
2024.07.31	2024.2	• Updated "Model Performance" with Version 2024.2 values • Added "FPGA AI Suite Software Reference Model" • Updated "FPGA AI Suite Layer/Primitive Ranges" as follows: — Added Tanh — Revised Max Pool window size range — Revised pool max_windows_height/width valid range • Updated "FPGA AI Suite IP Block Configuration" as follows: — Added `enable_parameter_rom` — Added `enable_tanh` • Updated "Parameter Group: Global Parameters" as follows: — Revised description of `arch_precision` parameter — Added section *Parameters: `output_image_height_max`, `output_image_width_max`, `output_channels_max`* • Updated "Parameter Group: `activation`" to add *Parameter: `activation/enable_tanh`* • Added "Parameter Group: `input_stream_interface`" • Added "Parameter Group: `ouput_stream_interface`" • Added "Feature Input and Output Streaming" • Added "Input Streaming" • Added "Output Streaming" • Added "Memoryless Operation" • Updated"DMA Control Registers" as follows: — Added `IP_reset` register — Added `Activate_streaming` register
2024.03.29	2024.1	• Updated "Model Performance" with Version 2024.1 values • Rebranded from "Intel FPGA AI Suite" to "FPGA AI Suite". • Added updates to support new dedicated layout transform module. • Updated "FPGA AI Suite Layer / Primitive Ranges" as follows: — Updated description of Softmax. — Added Sigmoid. — Added Swish. • Updated "FPGA AI Suite IP Block Configuration" as follows: — Expanded the ranges of the `c_vector`, `pool k_vector`, and `activation k_vector` parameters. — Added the `enable_sigmoid` parameter. — Expanded the range of the `arch_precision` parameter. • Updated "Parameter Group: Global Parameters" with additional information about Agilex 5 device family support. • Added `activation/enable_sigmoid` to "Parameter Group: `activation`". • Added `softmax/max_num_channels` parameter description to "Module: `softmax`".
2023.12.01	2023.3	• Updated "Model Performance" with Version 2023.3 values • Updated `arch_precision` parameter description in "Parameter group: Global Parameters". • Added "Performance Registers". • Added "Debug Network Registers".

continued...

Document Version	FPGA AI SuiteVersion	Changes
		• Added "PE scale precision" to "FPGA AI Suite IP Block Configuration". • Updated "Parameter Group: Global Parameters" for Agilex 5 support. • Updated "Parameter Group: `pe_array`" for Agilex 5 support.
2023.09.08	2023.2.1	• Added Multilayer Perceptrons (MLPs) to supported models.
2023.07.03	2023.2	• Updated "Model Performance" with Version 2023.2 values and the enabling of non-default Quartus Prime options (register merging). • Added additional information about modifying the internal precision of a graph for performance improvement in "Architecture Description File Format for Instance Parameterization".
2023.04.05	2023.1	• Added ChannelToSpace, DepthToSpace, and PixelShuffle to "FPGA AI Suite Layer / Primitive Ranges". • Added enable_debug to "FPGA AI Suite IP Block Configuration and Interfaces". • Added description of enable_round_clamp activation parameter where needed. • Added "Input Transform Mapping". • Added output transform mapping information to "Output Tensor In-Memory Format". • Renamed the`dlac` command. The FPGA AI Suite compiler command is now `dla_compiler`. • Updated "Model Performance" with Version 2023.1 values. • Updated the Intel Agilex product family name to "Intel Agilex® 7."
2023.02.03	2022.2	• Correct the description of the `-bgr` option of the AOT splitter utility.
2022.12.23	2022.2	• Added SqueezeNet to the list of supported models. • Added the `family` architecture description file global parameter. • Removed the `--family` IP generation utility command option.
2022.04.27	2022.1	• Removed references to HLS generation. • Updated descriptions of parameters in the `.arch` file. • Added a Model Performance section.
2021.09.10	2021.2	• Added information for Intel Agilexdevice support. • Added information for MobileNet v3 support.
2021.04.30	2021.1	• Various corrections and updates.
2020.12.04	2020.2	• Renamed *Intel FPGA AI Suite IP Core* to *Intel FPGA AI Suite IP*".
2020.10.30	2020.1	• Initial release.