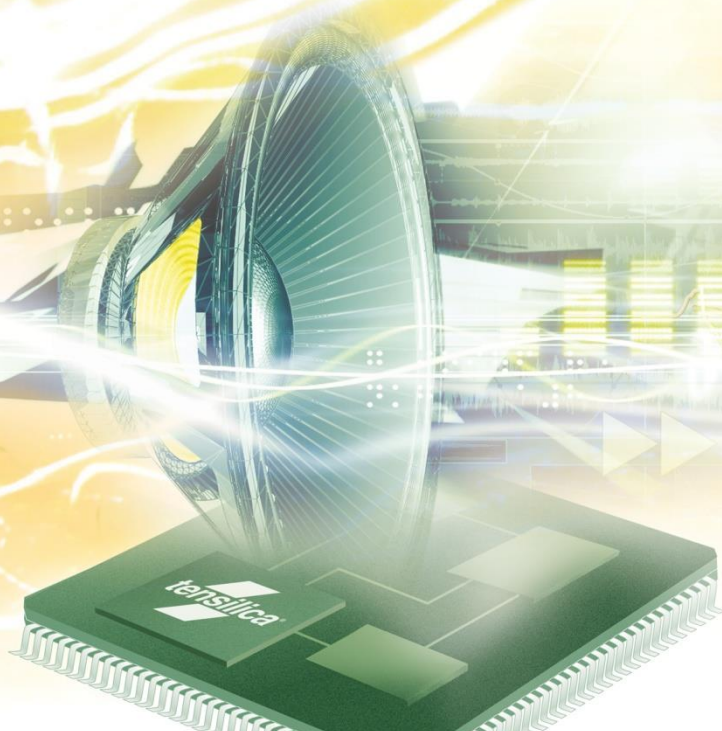




HiFi Neural Network Library

Programmer's Guide

For HiFi DSPs



Cadence Design Systems, Inc.
2655 Seely Ave.
San Jose, CA 95134
www.cadence.com

© 2020 Cadence Design Systems, Inc.
All rights reserved worldwide

This publication is provided “AS IS.” Cadence Design Systems, Inc. (hereafter “Cadence”) does not make any warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Information in this document is provided solely to enable system and software developers to use our processors. Unless specifically set forth herein, there are no express or implied patent, copyright or any other intellectual property rights or licenses granted hereunder to design or fabricate Cadence integrated circuits or integrated circuits based on the information in this document. Cadence does not warrant that the contents of this publication, whether individually or as one or more groups, meets your requirements or that the publication is error-free. This publication could include technical inaccuracies or typographical errors. Changes may be made to the information herein, and these changes may be incorporated in new editions of this publication.

© 2020 Cadence, the Cadence logo, Allegro, Assura, Broadband Spice, CDNLIVE!, Celtic, Chipestimate.com, Conformal, Connections, Denali, Diva, Dracula, Encounter, Flashpoint, FLIX, First Encounter, Incisive, Incyte, InstallScape, NanoRoute, NC-Verilog, OrCAD, OSKit, Palladium, PowerForward, PowerSI, PSpice, Purespec, Puresuite, Quickcycles, SignalStorm, Sigrity, SKILL, SoC Encounter, SourceLink, Spectre, Specman, Specman-Elite, SpeedBridge, Stars & Strikes, Tensilica, TripleCheck, TurboXim, Vectra, Virtuoso, VoltageStorm, Xcelium, Xplorer, Xtensa, and Xtreme are either trademarks or registered trademarks of Cadence Design Systems, Inc. in the United States and/or other jurisdictions.

OSCI, SystemC, Open SystemC, Open SystemC Initiative, and SystemC Initiative are registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission. All other trademarks are the property of their respective holders.

Version 2.4
Feb 2021

Contents

1.	Introduction to the HiFi NN Library	1
1.1	Organization of the HiFi NN Library Package	1
1.1.1	Document Overview.....	3
1.2	HiFi NN Library Specification	3
1.2.1	Low Level Kernels	3
1.2.2	Layers	4
1.2.3	Support for TensorFlow Lite Micro Operators	4
1.2.4	Changes from the Previous Release	4
2.	Generic HiFi NN Layer API	6
2.1	Shape.....	6
2.2	Memory Management.....	8
2.2.1	API Handle / Persistent Memory	9
2.2.2	Scratch Memory.....	9
2.2.3	Weights and Biases Memory.....	9
2.2.4	Input Buffer.....	9
2.2.5	Output Buffer	9
2.3	Generic API Errors	9
2.3.1	Common API Errors.....	10
2.4	C Language API	11
2.4.1	Startup Functions.....	12
2.4.2	Query Functions	13
2.4.3	Initialization Functions.....	13
2.4.4	Execution Functions.....	13
3.	HiFi NN Library – Low-Level Kernels.....	14
3.1	Matrix X Vector Multiplication Kernels	14
3.1.1	Matrix X Vector Kernels	14
3.1.2	Fused (Activation) Matrix X Vector Kernels	19
3.1.3	Matrix X Vector Batch Kernels	23
3.1.4	Matrix Multiplication Kernels.....	26
3.1.5	Matrix X Vector Kernels with Output Stride	29
3.2	Convolution Kernels.....	31
3.2.1	Standard 2D Convolution Kernel	31
3.2.2	Standard 1D Convolution Kernel	36
3.2.3	Depthwise Separable 2D Convolution Kernel	40
3.2.3.1	Depthwise 2D Convolution Kernel	41
3.2.3.2	Pointwise 2D Convolution Kernel	46
3.3	Activation Kernels.....	49
3.3.1	Sigmoid	49

3.3.2	Tanh	51
3.3.3	Rectifier Linear Unit (ReLU)	52
3.3.4	Softmax	54
3.3.5	Activation Min Max	56
3.4	Pooling Kernels	58
3.4.1	Average Pool Kernel	58
3.4.2	Max Pool Kernel	61
3.4.3	Fully Connected Kernel	64
3.5	Miscellaneous Kernels	67
3.5.1	Interpolation Kernel	67
3.5.2	L2 Normalization Kernel	68
3.5.3	Basic Kernels	69
3.5.4	Elementwise Quantize Kernel	71
3.5.5	Memmove Kernel	73
3.5.6	Dot Product Kernels	74
4.	HiFi NN Library – Layers	76
4.1	GRU Layer	76
4.1.1	GRU Layer Specification	76
4.1.2	Error Codes Specific to GRU	77
4.1.3	API Functions Specific to GRU	78
4.1.3.1	Query Functions	78
4.1.3.2	Initialization Stage	80
4.1.3.3	Execution Stage	81
4.1.4	Structures Specific to GRU	84
4.1.5	Enums Specific to GRU	85
4.2	LSTM Layer	87
4.2.1	LSTM Layer Specification	87
4.2.2	Error Codes Specific to LSTM	87
4.2.3	API Functions Specific to LSTM	89
4.2.3.1	Query Functions	89
4.2.3.2	Initialization Stage	91
4.2.3.3	Execution Stage	92
4.2.4	Structures Specific to LSTM	96
4.2.5	Enums Specific to LSTM	97
4.3	CNN Layer	99
4.3.1	CNN Layer Specification	99
4.3.2	Error Codes Specific to CNN	99
4.3.3	API Functions Specific to CNN	100
4.3.3.1	Query Functions	100
4.3.3.2	Initialization Stage	103
4.3.3.3	Execution Stage	105
4.3.4	Structures Specific to CNN	108
4.3.5	Enums Specific to CNN	109
5.	Additional Supporting Libraries	111

5.1	xa_annlib Features	111
5.2	xa_annlib Operations	111
5.2.1	Relu operations	112
5.2.2	Tanh	113
5.2.3	Logistic	113
5.2.4	Softmax	114
5.2.5	Concatenation	115
5.2.6	Convolution Operation	116
5.2.7	Depth-wise Convolution Operation	117
5.2.8	Fully Connected	119
5.2.9	L2 Normalization	120
5.2.10	Pooling operations	120
5.2.11	Basic operations	122
5.2.12	Local Response Norm	123
5.2.13	Reshape Generic	124
5.2.14	Resize Bilinear	125
5.2.15	Depth to Space	126
5.2.16	Space to Depth	126
5.2.17	Pad	127
5.2.18	Batch to Space	128
5.2.19	Space to Batch	129
5.2.20	Squeeze	130
5.2.21	Transpose	131
5.2.22	Mean	131
5.2.23	Strided Slice	132
5.2.24	Dequantize Quant8 to Float32	133
5.2.25	Embedding Lookup	134
5.2.26	Hashtable Lookup	135
5.2.27	LSH Projection	136
5.2.28	LSTM	136
5.2.29	RNN	137
5.2.30	SVDF	138
6.	Introduction to the Example Testbench	140
6.1	Making the Library	140
6.1.1	Controlling Library Code Size	140
6.2	Making the Executable	140
6.2.1	Controlling Executable Code Size	141
6.3	Sample Testbench for Matrix X Vector Multiplication Kernels	142
6.3.1	Usage	142
6.4	Sample Testbench for Convolution Kernels	143
6.4.1	Usage	143
6.5	Sample Testbench for Activation Kernels	146
6.5.1	Usage	146
6.6	Sample Testbench for Pooling Kernels	147

6.6.1	Usage.....	147
6.7	Sample Testbench for Basic Kernels	148
6.7.1	Usage.....	149
6.8	Sample Testbench for Normalization Kernels.....	150
6.8.1	Usage.....	150
6.9	Sample Testbench for GRU Layer	151
6.9.1	Usage.....	151
6.10	Sample Testbench for LSTM Layer.....	152
6.10.1	Usage.....	152
6.11	Sample Testbench for CNN Layer	153
6.11.1	Usage.....	153
6.12	Sample Testbench for ANN Operations	154
6.12.1	Usage.....	155
6.13	Sample Testbench of TF Micro Lite Example	155
6.13.1	Usage.....	155
7.	References	157

Figures

Figure 2-1 HiFi NN Layer Interfaces	6
Figure 2-2 Matrix and Cube (SHAPE_CUBE_DWH_T) Shape Representation	8
Figure 2-3 NN Layer Flow Overview	11

Tables

Table 2-1 Library Identification Functions	12
Table 4-1 GRU Get Persistent Size Function	78
Table 4-2 GRU Get Scratch Size Function	79
Table 4-3 GRU Init Function	80
Table 4-4 GRU Execution Function	81
Table 4-5 GRU Set Parameter Function Details	82
Table 4-6 GRU Get Parameter Function Details	83
Table 4-7 GRU Config Structure xa_nnlib_gru_init_config_t	84
Table 4-8 xa_nnlib_gru_weights_t Parameter Type	84
Table 4-9 xa_nnlib_gru_biases_t Parameter Type	85
Table 4-10 Enum xa_nnlib_gru_precision_t	85
Table 4-11 GRU Specific Parameters	86
Table 4-12 LSTM Get Persistent Size Function	89
Table 4-13 LSTM Get Scratch Size Function	90
Table 4-14 LSTM Init Function	91
Table 4-15 LSTM Execution Function	92
Table 4-16 LSTM Set Parameter Function Details	94
Table 4-17 LSTM Get Parameter Function Details	95
Table 4-18 LSTM Config Structure xa_nnlib_lstm_init_config_t	96
Table 4-19 xa_nnlib_lstm_weights_t Parameter Type	96
Table 4-20 xa_nnlib_lstm_biases_t Parameter Type	97
Table 4-21 Enum xa_nnlib_lstm_precision_t	97
Table 4-22 LSTM Specific Parameters	98
Table 4-23 CNN Get Persistent Size Function	100
Table 4-24 CNN Get Scratch Size Function	101
Table 4-25 CNN Init Function	103

Table 4-26	CNN Execution Function	105
Table 4-27	CNN Set Parameter Function Details.....	106
Table 4-28	CNN Get Parameter Function Details	107
Table 4-29	CNN Config Structure <code>xa_nnlib_cnn_init_config_t</code>	108
Table 4-30	Enum <code>xa_nnlib_cnn_precision_t</code>	109
Table 4-31	Enum <code>xa_nnlib_cnn_algo_t</code>	109
Table 4-32	CNN Specific Parameters.....	110

Document Change History

Version	Changes
0.1	<ul style="list-style-type: none"> ■ Initial release ■ Matrix X vector and activation function kernels added ■ GRU Layer (8x16, 16x16) added
1.0	<ul style="list-style-type: none"> ■ GA release ■ Convolution, pooling kernels added ■ LSTM layer (8x16, 16x16) and CNN layer added
1.0.1	<ul style="list-style-type: none"> ■ Some minor updates
2.0	<ul style="list-style-type: none"> ■ Updated for HiFi NN Library v2.1.0 (Android NN support and TF Micro Lite Example)
2.1	<ul style="list-style-type: none"> ■ Updated for HiFi NN Library v2.2.0
2.2	<ul style="list-style-type: none"> ■ Updated performance tables
2.3	<ul style="list-style-type: none"> ■ Added description of quantized 8-bit variants for standard convolution, depthwise convolution, fully connected and softmax kernels. ■ Added HiFi 3 to the list of supported cores. ■ Updated description of depthwise convolution, average pool and max pool kernels.
2.4	<ul style="list-style-type: none"> ■ Added below kernels used for SVDF, quantize TFLM operators and pointwise convolution <ul style="list-style-type: none"> ○ xa_nn_dot_prod_16x16_asym8s ○ xa_nn_elm_quantize_asym16s_asym8s ○ xa_nn_matmul_per_chan_sym8sxasym8s_asym8s ○ xa_nn_matXvec_out_stride_sym8sxasym8s_16 ○ xa_nn_memmove_16

1. Introduction to the HiFi NN Library

The HiFi Neural Network (NN) Library is a HiFi-optimized implementation of various NN layers and low level NN kernels. The library is designed with speech and audio neural network domain focus. The low level NN kernels are HiFi-optimized building blocks for NN layer implementation with a generic and simple interface. The NN layers are built using low level kernels and accept input in the form of 'shapes'¹ (up to four dimensions) and produce the output, also in the form of shapes. The layers use the weights or coefficients and biases stored 'externally'² for their operation. The shape of the input, output, weights and biases are as per the layer's design. The HiFi NN Library also includes support for Android NN API v1.1 (Android P) NN operations and the operators of TensorFlow Lite for Microcontrollers (TFLM) from TensorFlow v2.2.

This guide refers to the NN layers simply as layers, the low level NN kernels as low-level kernels and the Android NN operations as ANN operations. The current version of the library implements GRU, LSTM (forward path), and CNN layers. It also implements matrix vector multiply, activation, pooling, normalization and convolution functions and some basic elementwise operations as low-level kernels.

-
- | | |
|-------------|--|
| Note | This version of the HiFi NN Library is optimized for HiFi 4 DSP. The same library can be cross compiled for HiFi 3, HiFi 3z, HiFi 5 DSP configurations and Fusion F1 DSP configurations with the AVS and the 16-bit Quad MAC unit options. To enable the cross compilation, few HiFi 4 instructions that are not available in the other configurations, are mapped to sequence of instructions available for the respective configuration. |
| Note | The HiFi NN Library can be built for configurations with or without the optional Vector Floating Point Unit (VFPU). For configurations without VFPU, the floating-point variants of the kernels are not supported. |
| Note | The HiFi NN Library can be built for configurations with newlib or Xtensa C library. The ANN and TFLM example applications and respective supporting libraries need C++11 support and can be built for configurations with Xtensa C library only. |
| Note | This version of the HiFi NN Library is tested with the xt-ccc/xt-xc++ and xt-clang/xt-clang++ compilers using Xtensa Software Tools from RI-2020.5 release. |
-

1.1 Organization of the HiFi NN Library Package

The HiFi NN Library package includes the HiFi NN library containing all layers and low-level kernels implementations and a set of sample test applications (for layers, low-level kernels, ANN operations and TFLM micro speech example).

¹ Refer to Section 2.1 Shape

² Refer to Section 2.2.3 Weights and Biases Memory

The HiFi NN library provides a set of low level NN kernels. The application can use these kernels to implement or optimize performance of NN layers.

The HiFi NN library also implements a set of NN layers. The application can instantiate these layers and connect inputs and outputs across the layers to form a Neural Network system.

The HiFi NN library low level kernels support the datatypes required by the ANN operators from Android NN API v1.1. The HiFi NN Library package also includes a supporting library containing the HiFi implementation of the ANN operators. This library is referred to as ANN library. An application can use the ANN library along with the HiFi NN library to implement the Android NN API.

The sample test applications implement a file-based application to test an instance of a layer or low level NN kernels for the given specification using pre-generated input, weight or coefficients and bias shapes stored in files in raw binary format.

The HiFi NN library package also contains a sample application based on the micro speech example from TFLM (TF v2.2). The micro speech example is included in three parts, the main sample application and two supporting libraries: one for the TFLM framework, TensorFlow Lite Micro library and other library for the preprocessing part, TensorFlow Micro Frontend library.

1.1.1 Document Overview

This document covers all the information required to integrate the HiFi NN Library into a Neural Network system. All the layers implement “HiFi NN layer APIs”, which is generic and explained in Section 2. The low level NN kernels are explained in Section 3. The APIs for each layer are described in Section 4. Section 5 provides details about the included supporting libraries. Section 6 provides details about the available sample testbenches. References are listed in Section 7.

1.2 HiFi NN Library Specification

The current version of the HiFi NN Library provides the following HiFi-optimized low-level kernels, Android NN operations support and layer implementations.

1.2.1 Low Level Kernels

- Matrix-vector multiplication kernels
- Convolution kernels
- Activation kernels
- Pooling kernels
- Basic operations kernels
- Normalization kernels

These kernels support fixed point 8-bit, 16-bit, single precision floating point and asymmetric 8-bit quantized datatypes for the weights, biases, input, and output.

They also support 8-bit quantized data types (asym8u – Asymmetric 8-bit unsigned, asym8s – Asymmetric 8-bit signed, sym8s – Symmetric 8-bit signed) for weights or coefficients, input, and output. Biases are 32-bit quantized values.

8-bit quantized types are either unsigned (0, 255) or signed (-128, 127) 8-bit integer with 3 additional parameters.

Three numbers are associated with a quantized 8-bit value that can be used to convert the 8-bit integer to the real value and vice versa. These numbers are:

- Shift: an integer value indicating the amount of shift. If the value is positive, it is left shift and if negative, it is right shift
- Multiplier: a 32 bit (Q31) fixed point value greater than zero.
- Zero point: a 32 bit integer, in range [0, 255] for unsigned type, in range [-128, 127] for signed type.

The formula is:

$$\text{real_value} = (\text{quantized_value} - \text{zero_point}) * 2^{(\text{shift})} * \text{multiplier}$$

The ‘sym8s’ type is symmetrical around 0, this means that quantized values are between -127 to 127 and zero point is 0, so all the calculation required due to zero point is avoided.

To match the asym8u/asym8s/sym8s APIs with Tensorflow, we define zero point as zero_bias in the NN library APIs. The zero_bias is an integer value having range asym8u - [0, 255], asym8s – [-128, 127] (or asym8u - [-255, 0], asym8s – [-127, 128] in case of the reverse operation depending on the corresponding Tensorflow kernel).

1.2.2 Layers

- GRU layer (8x16, 16x16 precision)
- LSTM (forward path) layer (8x16, 16x16 precision)
- CNN layer (8x8, 8x16, 16x16, and float32xfloat32 precision)

Note, MxN precision above denotes (weights or coefficients) x (input, output, bias) precision. Refer to Section 3 for details.

1.2.3 Support for TensorFlow Lite Micro Operators

The HiFi NN Library low level kernels can be used to implement the following operators of TensorFlow Lite Micro:

No.	Operator	Float32 Datatype Support	Uint8 (asymmetric quantized uint8) Datatype Support	Int8 (quantized int8) Datatype Support
1	FULLY_CONNECTED	Yes	Yes	Yes
2	MAX_POOL_2D	Yes	Yes	Yes
3	SOFTMAX	Yes	Yes	Yes
4	LOGISTIC	Yes		
5	SVDF	Yes		Yes
6	CONV_2D	Yes	Yes	Yes
7	DEPTHWISE_CONV_2D	Yes	Yes	Yes
8	AVERAGE_POOL_2D	Yes	Yes	Yes
9	FLOOR	Yes		
10	RELU	Yes	Yes	
11	RELU6	Yes	Yes	
12	ADD	Yes	Yes	
13	MUL	Yes	Yes	
14	QUANTIZE			Yes

1.2.4 Changes from the Previous Release

- Added support for quantized 8-bit variants for standard convolution, depthwise convolution, fully connected and softmax kernels.

- Enabled HiFi 3 cross compilation support.
- Removed kernel padding requirement for depthwise convolution variants.
- Updated prototype of average pool and max pool 8x8 variants, these variants can be used for pooling operations in TFLM for 'int8' datatype.
- Added support for quantized 8-bit variants for pointwise convolution, SVDF (dot product and matXvec), quantize.
- Updated the TFLM example to newer version of the TFLM framework.

2. Generic HiFi NN Layer API

Note This section explains an API standard which is evolving. The APIs may undergo some changes in future versions.

This section describes the API that is common to all the HiFi NN layers. The API facilitates any layer instance that works in the overall method shown in Figure 2-1.

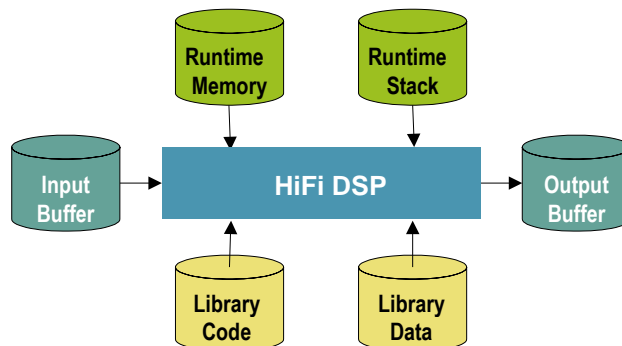


Figure 2-1 HiFi NN Layer Interfaces

All the buffers, input, output, weights and biases are described as shapes. Section 2.1 explains the shape structure.

Section 2.2 discusses all the types of runtime memory required by the layer instances. There is no state information held in static memory, therefore a single thread can perform time division processing of multiple layer instances. Additionally, multiple threads can perform concurrent layer instance processing.

The output from one instance can be fed as input to the next instance if the precision and the dimension matches.

The data types, structures, and error codes explained in this section are declared/defined in `xa_nnlib_standard.h`. By default, the API header file of each layer will include this header file. The application need not include this file.

2.1 Shape

The shapes are used to describe any buffer used in the NN library. The structure `xa_nnlib_shape_t` is defined in `xa_nnlib_standard.h`. The shape can be vector, matrix, or cube.

- Vector is a one-dimensional shape specified by length.
- Matrix is a two-dimensional shape specified by rows, columns, and row_offset. This assumes that the elements in a row are stored at consecutive addresses in memory.

- Cube is a three-dimensional shape specified by height, width, depth, height_offset, width_offset, and depth offset. Cube supports the following shape types:
 - `SHAPE_CUBE_DWH_T`

This assumes that elements are stored in depth (D), width (W), and height (H) order; that is, elements with the same height and width indices are stored consecutively. In other words, in memory, depth is the inner most dimension, width is the middle dimension and height is the outer dimension. This type is also referred to as the NHWC format or the depth-first format (N = Number of batches, H = Height, W = Width, C = Channels / depth).
 - `SHAPE_CUBE_WHD_T`

This assumes that elements are stored in width (W), height (H), and depth (D) order; that is, elements with the same height and depth are stored consecutively. In other words, in memory, width is the inner most dimension, height is the middle dimension and depth is the outer dimension. This type is also referred to as the NCHW format or the width-first format (N = Number of batches, C = Channels / depth, H = Height, W = Width).

Figure 2-1 explains the dimension variables of matrix and cube shapes.

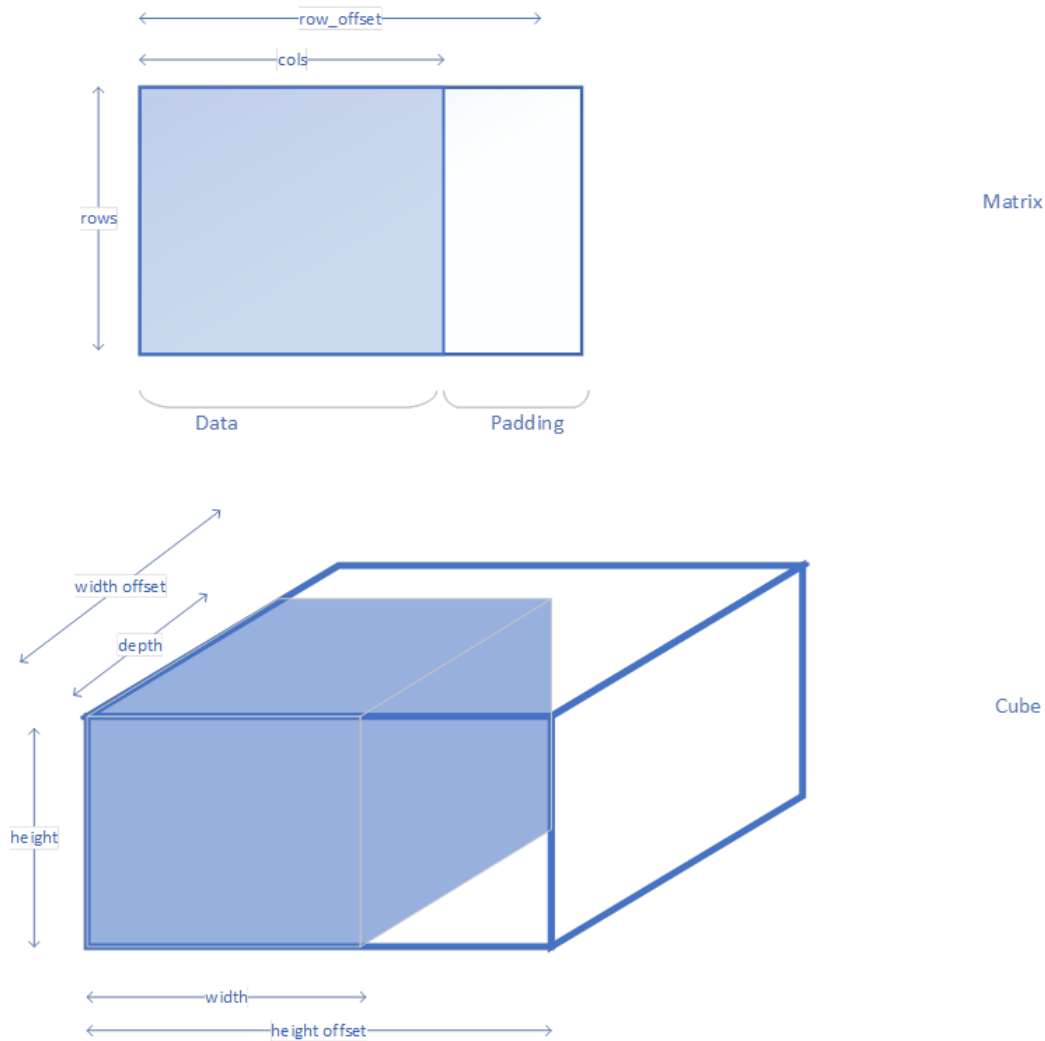


Figure 2-2 Matrix and Cube (SHAPE_CUBE_DWH_T) Shape Representation

2.2 Memory Management

The HiFi NN layer API supports a flexible memory scheme and a simple interface that eases the integration into the final application. The API allows the layers to request the required memory for their operations during runtime.

The runtime memory requirement consists primarily of the scratch and persistent memory. The components also require an input buffer and output buffer for the passing of data into and out of the layer.

2.2.1 API Handle / Persistent Memory

The layer API stores persistent state information in a structure that is referenced via an opaque handle. The handle is passed by the application for each API call. This object contains all state and history information that is maintained from one-layer frame invocation to the next within the same thread or instance. The layers expect that the contents of the persistent memory be unchanged by the system apart from the layer itself for the complete lifetime of the layer.

2.2.2 Scratch Memory

This is the temporary buffer used by the layer during a single frame processing call. The contents of this memory region should not be changed if the actual layer execution process is active; that is, if the thread running the layer is inside any API call. This region can be used freely by the system between successive calls to the layer.

2.2.3 Weights and Biases Memory

The weights or coefficients and biases should be managed by the application, and memory should not be requested by the API. If the design requires DMA access from or to the internal memory for better performance, a ping-pong or circular buffer is allocated as part of the scratch into which the weights, biases, input, and output are copied using DMA. If required, these memories can also be persistent.

2.2.4 Input Buffer

This is the buffer from which the layer reads the input. This buffer must be made available for the layer before its execution call. The input buffer should have an associated shape information to describe the input data format. The input buffer pointer can be changed by the application between calls to the layer, but shape information cannot be changed. This allows the layer to read directly from the output of another layer.

2.2.5 Output Buffer

This is the buffer to which the layer writes the output. This buffer must be made available for the layer before its execution call. The output buffer should have an associated shape information to which the layer can describe the output data format. The output buffer pointer can be changed by the application between calls to the layer. This allows the layer to write directly to the input of another layer.

2.3 Generic API Errors

Layer API functions return an error code of type `Int32`, which is of type `signed int`. The format of the error codes is defined in the following table.

31	30 – 27	26–12	11 – 7	6 – 0
Fatal	Class	Reserved	Component	Sub code

The errors that can be returned from the API are subdivided into those that are fatal, which require resetting the layer, and those that are nonfatal and are provided for information to the application.

The class of an error can be API, Config, or Execution. The API category errors are concerned with the incorrect use of the API. The Config errors are produced when the layer parameters are incorrect or outside the supported usage. The Execution errors are returned after a call to the main process and indicate situations that have arisen due to the input data.

2.3.1 Common API Errors

The following errors are fatal and should not be encountered during normal application operation. They signal that a serious error has occurred in the application that is calling the layer.

- `XA_NNLIB_FATAL_MEM_ALLOC`
At least one of the pointers passed into the API function is NULL.
- `XA_NNLIB_FATAL_MEM_ALIGN`
At least one of the pointers passed into the API function is not properly aligned.
- `XA_NNLIB_FATAL_INVALID_SHAPE`
At least one of the shapes passed to the API function is invalid.

2.4 C Language API

An overview of the NN layer flow is shown in Figure 2-3. The NN layer API consists of query, initialization, and execution functions.

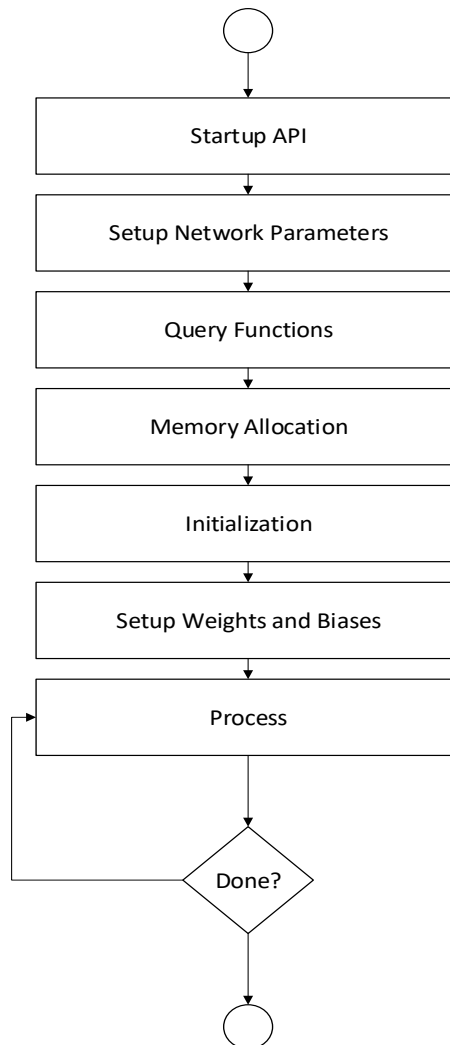


Figure 2-3 NN Layer Flow Overview

2.4.1 Startup Functions

The API startup functions shown in Table 2-1 get the various identification strings from the component library. They are for information only and their usage is optional. These functions do not take any input arguments and return `const char *`.

Table 2-1 Library Identification Functions

Function	Description
<code>xa_nnlib_get_lib_name_string</code>	Get the name of the library.
<code>xa_nnlib_get_lib_version_string</code>	Get the version of the library.
<code>xa_nnlib_get_lib_api_version_string</code>	Get the version of the API.

Example

```
const char *name = xa_nnlib_get_lib_name_string();  
const char *ver = xa_nnlib_get_lib_version_string();  
const char *aver = xa_nnlib_get_lib_api_version_string();
```

Errors

- None

2.4.2 Query Functions

The query functions are used in the startup and the memory allocation stages to obtain information about the memory requirements of the library.

Following is the naming convention for query functions:

```
xa_nnlb_<layer>_get_{persistent | scratch}_<placement>
```

Where:

<layer> indicates the module name (such as gru).

<placement> specifies fast or slow.

2.4.3 Initialization Functions

The initialization functions are used to reset the layer to its initial state. Because the layers are fully reentrant, the application can initialize the layer multiple times.

Following is the naming convention for initialization functions:

```
xa_nnlb_<layer>_init
```

2.4.4 Execution Functions

The execution functions are used to generate the output shape by processing one input shape.

Following is the naming convention for execution functions:

```
xa_nnlb_<layer>_process
```

3. HiFi NN Library – Low-Level Kernels

This section explains the low-level kernels provided in the NN library. All the low-level kernels have a generic, simple interface.

The NN library is a single archive containing all low-level kernels and layers implementations. The following sections explain each low-level kernel in detail.

3.1 Matrix X Vector Multiplication Kernels

3.1.1 Matrix X Vector Kernels

Description

These kernels perform the dual matXvec operation with bias addition; that is, $z = \text{mat1} * \text{vec1} + \text{mat2} * \text{vec2} + \text{bias}$. The column dimension of `mat1` must match the row dimension of `vec1` and similarly for `mat2`, `vec2`. The number of rows for `mat1` and `mat2` must be same. Bias and resulting output vector `z` have as many rows as `mat1` and `mat2`.

`bias_shift` and `acc_shift` arguments are provided in the kernel API to adjust Q format of bias and output, respectively. Both `bias_shift` and `acc_shift` can be either positive or negative, where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as matXvec multiplication – accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

`row_stride1` and `row_stride2` arguments are provided in kernel API for row offsets of `mat1` and `mat2`, respectively. Note, input matrices are expected to be appropriately padded in case of `row_stride > cols`.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

The arguments, `mat1_zero_bias`, `mat2_zero_bias`, `vec1_zero_bias`, `vec2_zero_bias`, are provided to convert the `asym8` inputs into their real values and perform matXvec operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to `asym8`.

Function variants available are `xa_nn_matXvec_[p]x[q]_[r]`, where:

- `[p]`: Matrix precision in bits

- $[q]$: Vector precision in bits
- $[r]$: Output precision in bits

Precision

There are twelve variants available:

Type	Description
16x16_16	16-bit matrix inputs, 16-bit vector inputs, 16-bit output
16x16_32	16-bit matrix inputs, 16-bit vector inputs, 32-bit output
16x16_64	16-bit matrix inputs, 16-bit vector inputs, 64-bit output
8x16_16	8-bit matrix inputs, 16-bit vector inputs, 16-bit output
8x16_32	8-bit matrix inputs, 16-bit vector inputs, 32-bit output
8x16_64	8-bit matrix inputs, 16-bit vector inputs, 64-bit output
8x8_8	8-bit matrix inputs, 8-bit vector inputs, 8-bit output
8x8_16	8-bit matrix inputs, 8-bit vector inputs, 16-bit output
8x8_32	8-bit matrix inputs, 8-bit vector inputs, 32-bit output
f32xf32_f32	float32 matrix inputs, float32 vector inputs, float32 output
asym8xasym8_asym8	asym8 matrix inputs, asym8 vector inputs, asym8 output
sym8sxasym8s_asym8s	sym8s matrix inputs, asym8s vector inputs, asym8s output

Algorithm

$$z_n = 2^{acc-shift} \left(\sum_{m=0}^{cols1-1} mat1_{n,m} \cdot vec1_m + \sum_{m=0}^{cols2-1} mat2_{n,m} \cdot vec2_m + 2^{bias-shift} bias_n \right)$$

In case of floating-point and asym8 routine, acc_shift=0 and bias_shift=0.

Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

Prototype

```
WORD32 xa_nn_matXvec_16x16_16
(WORD16 * p_out,          WORD16 * p_mat1,          WORD16 * p_mat2,
 WORD16 * p_vec1,         WORD16 * p_vec2,          WORD16 * p_bias,
 WORD32 rows,            WORD32 cols1,             WORD32 cols2,
 WORD32 row_stride1,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_16x16_32
(WORD32 * p_out,          WORD16 * p_mat1,          WORD16 * p_mat2,
 WORD16 * p_vec1,         WORD16 * p_vec2,          WORD16 * p_bias,
 WORD32 rows,            WORD32 cols1,             WORD32 cols2,
 WORD32 row_stride1,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_16x16_64
(WORD64 * p_out,          WORD16 * p_mat1,          WORD16 * p_mat2,
 WORD16 * p_vec1,         WORD16 * p_vec2,          WORD16 * p_bias,
 WORD32 rows,            WORD32 cols1,             WORD32 cols2,
 WORD32 row_stride1,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_8x16_16
(WORD16 * p_out,          WORD8 * p_mat1,           WORD8 * p_mat2,
 WORD16 * p_vec1,         WORD16 * p_vec2,          WORD16 * p_bias,
 WORD32 rows,            WORD32 cols1,             WORD32 cols2,
 WORD32 row_stride1,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);
```

```

WORD32 xa_nn_matXvec_8x16_32
(WORD32 * p_out,          WORD8 * p_mat1,          WORD8 * p_mat2,
 WORD16 * p_vec1,         WORD16 * p_vec2,         WORD16 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_8x16_64
(WORD64 * p_out,          WORD8 * p_mat1,          WORD8 * p_mat2,
 WORD16 * p_vec1,         WORD16 * p_vec2,         WORD16 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_8x8_8
(WORD8 * p_out,           WORD8 * p_mat1,          WORD8 * p_mat2,
 WORD8 * p_vec1,          WORD8 * p_vec2,         WORD8 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_8x8_16
(WORD16 * p_out,          WORD8 * p_mat1,          WORD8 * p_mat2,
 WORD8 * p_vec1,          WORD8 * p_vec2,         WORD8 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_8x8_32
(WORD32 * p_out,          WORD8 * p_mat1,          WORD8 * p_mat2,
 WORD8 * p_vec1,          WORD8 * p_vec2,         WORD8 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2,
 WORD32 acc_shift,       WORD32 bias_shift);

WORD32 xa_nn_matXvec_f32xf32_f32
(FLOAT32 * p_out,         FLOAT32 * p_mat1,          FLOAT32 * p_mat2,
 FLOAT32 * p_vec1,        FLOAT32 * p_vec2,         FLOAT32 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2);

WORD32 xa_nn_matXvec_asym8xasym8_asym8
(UWORD8 * p_out,          const UWORD8 * p_mat1,  const UWORD8 * p_mat2,
 const UWORD8 * p_vec1,   const UWORD8 * p_vec2,  const WORD32 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2,  WORD32 mat1_zero_bias,
 WORD32 mat2_zero_bias,  WORD32 vec1_zero_bias, WORD32 vec2_zero_bias,
 WORD32 out_multiplier,  WORD32 out_shift,    WORD32 out_zero_bias);

WORD32 xa_nn_matXvec_sym8sxasym8s_asym8s
(WORD8 * p_out,           const WORD8 * p_mat1,  const WORD8 * p_mat2,
 const WORD8 * p_vec1,    const WORD8 * p_vec2,  const WORD32 * p_bias,
 WORD32 rows,             WORD32 cols1,            WORD32 cols2,
 WORD32 row_stridel,     WORD32 row_stride2,  WORD32 vec1_zero_bias,
 WORD32 vec2_zero_bias,  WORD32 out_multiplier, WORD32 out_shift,
 WORD32 out_zero_bias);

```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *	p_mat1	rows*cols1	Input matrix 1, fixed or floating point, asym8u or sym8s

Type	Name	Size	Description
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *	p_mat2	rows*cols2	Input matrix 2, fixed or floating point, asym8u or sym8s
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *	p_vec1	cols1*1	Input vector 1, fixed or floating point, asym8u or sym8s
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *	p_vec2	cols2*1	Input vector 2, fixed or floating point, asym8u or sym8s
WORD16 *, WORD8 *, const WORD32 *, const FLOAT32 *	p_bias	rows*1	Bias vector, fixed or floating point
WORD32	Rows		Number of rows in matrix 1, 2 and bias
WORD32	cols1		Number of columns in matrix 1 and rows in vector 1
WORD32	cols2		Number of columns in matrix 2 and rows in vector 2
WORD32	row_stride1		Row offset of matrix 1
WORD32	row_stride2		Row offset of matrix 2
WORD32	acc_shift		Shift applied to accumulator
WORD32	bias_shift		Shift applied to bias
WORD32	mat1_zero_bias		Zero offset of matrix 1
WORD32	mat2_zero_bias		Zero offset of matrix 2
WORD32	vec1_zero_bias		Zero offset of vector 1
WORD32	vec2_zero_bias		Zero offset of vector 2
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output
WORD32	out_zero_bias		Zero offset of output
Output			
WORD8 *, UWORD8 *, WORD16 *, WORD32 *, WORD64 *, FLOAT32 *	p_out	rows*1	Output, fixed or floating point, asym8u or sym8s

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
-----------	--------------

row_stridel, row_stride2, cols1, cols2	Multiples of 4 (1 for floating point and asym8)
p_mat1, p_mat2, p_vec1, p_vec2	Aligned on 4*(size of one element)-byte boundary ((size of one element)-byte only in case of floating point and asym8) Should not overlap
p_bias, p_out	Aligned on (size of one element)-byte boundary (for kernels supporting multiple bias precision maximum size of one element should be considered as the alignment requirement) Should not overlap
p_mat1, p_vec1, p_bias, p_out	Cannot be NULL
acc_shift, bias_shift, out_shift	{-31, ..., 31}
mat1_zero_bias, mat2_zero_bias, vec1_zero_bias, vec2_zero_bias	{-255, ..., 0} for asym8u, {-127....., 128} for asym8s
out_multiplier	Greater than 0
out_zero_bias	{0, ..., 255} if out type is asym8u, {-128.....,127} if out type is asym8s

3.1.2 Fused (Activation) Matrix X Vector Kernels

Description

These kernels perform the fused dual matXvec operation with an activation function i.e. $z = \text{activation}(\text{mat1} * \text{vec1} + \text{mat2} * \text{vec2} + \text{bias})$. The column dimension of `mat1` must match the row dimension of `vec1` and similarly for `mat2`, `vec2`. Bias and resulting output vector `z` have as many rows as `mat1` and `mat2`.

Intermediate output of $(\text{mat1} * \text{vec1} + \text{mat2} * \text{vec2} + \text{bias})$ is stored in temporary memory provided by the `p_scratch` argument to kernel API. Activation function is applied on this intermediate output to get final output. Note, for fixed point kernels, the activation function always takes input in Q6.25 format.

`bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and intermediate output respectively. Both `bias_shift` and `acc_shift` can be either positive or negative, where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as matXvec multiplication – accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the intermediate output in Q6.25 format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels.

`row_stridel` and `row_stride2` arguments are provided in kernel API for row offsets of `mat1` and `mat2` respectively. Note, input matrices are expected to be appropriately padded in case of `row_stride > cols`.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

Function variants available are `xa_nn_matXvec_[p]x[q]_[r]_<activation>`, where:

- [p]: Matrix precision in bits
- [q]: Vector precision in bits
- [r]: Output precision in bits
- <activation>: activation tag 'tanh' or 'sigmoid'

Precision

There are eight variants available:

Type	Description
16x16_16_tanh	16-bit matrix inputs, 16-bit vector inputs, 16-bit output with tanh activation function
16x16_16_sigmoid	16-bit matrix inputs, 16-bit vector inputs, 16-bit output with sigmoid activation function
8x16_16_tanh	8-bit matrix inputs, 16-bit vector inputs, 16-bit output with tanh activation function
8x16_16_sigmoid	8-bit matrix inputs, 16-bit vector inputs, 16-bit output with sigmoid activation function
8x8_8_tanh	8-bit matrix inputs, 8-bit vector inputs, 8-bit output with tanh activation
8x8_8_sigmoid	8-bit matrix inputs, 8-bit vector inputs, 8-bit output with sigmoid activation
f32xf32_f32_tanh	float32 matrix inputs, float32 vector inputs, float32 output with tanh activation
f32xf32_f32_sigmoid	float32 matrix inputs, float32 vector inputs, float32 output with sigmoid activation

Algorithm

$$z_n = \text{activation} \left(2^{\text{acc-shift}} \left(\sum_{m=0}^{\text{cols1}-1} \text{mat1}_{n,m} \cdot \text{vec1}_m + \sum_{m=0}^{\text{cols2}-1} \text{mat2}_{n,m} \cdot \text{vec2}_m + 2^{\text{bias-shift}} \text{bias}_n \right) \right), \quad n = 0, \dots, \text{rows} - 1$$

In case of floating-point routine, `acc_shift=0` and `bias_shift=0`.

Thus, $2^{\text{acc-shift}} = 2^{\text{bias-shift}} = 1$

activation is tanh or sigmoid

Prototype

```
WORD32 xa_nn_matXvec_16x16_16_tanh
(WORD16 * p_out,          WORD16 * p_mat1,          WORD16 * p_mat2,
 WORD16 * p_vec1,         WORD16 * p_vec2,          VOID * p_bias,
 WORD32 rows,             WORD32 cols1,              WORD32 cols2,
```

```

WORD32 row_stride1,      WORD32 row_stride2,      WORD32 acc_shift,
WORD32 bias_shift,      WORD32 bias_precision, VOID * p_scratch);
WORD32 xa_nn_matXvec_16x16_16_sigmoid
(WORD16 * p_out,          WORD16 * p_mat1,          WORD16 * p_mat2,
WORD16 * p_vec1,          WORD16 * p_vec2,          VOID * p_bias,
WORD32 rows,             WORD32 cols1,             WORD32 cols2,
WORD32 row_stride1,      WORD32 row_stride2,      WORD32 acc_shift,
WORD32 bias_shift,      WORD32 bias_precision, VOID * p_scratch);
WORD32 xa_nn_matXvec_8x16_16_tanh
(WORD16 * p_out,          WORD8 * p_mat1,          WORD8 * p_mat2,
WORD16 * p_vec1,          WORD16 * p_vec2,          VOID * p_bias,
WORD32 rows,             WORD32 cols1,             WORD32 cols2,
WORD32 row_stride1,      WORD32 row_stride2,      WORD32 acc_shift,
WORD32 bias_shift,      WORD32 bias_precision, VOID * p_scratch);
WORD32 xa_nn_matXvec_8x16_16_sigmoid
(WORD16 * p_out,          WORD8 * p_mat1,          WORD8 * p_mat2,
WORD16 * p_vec1,          WORD16 * p_vec2,          VOID * p_bias,
WORD32 rows,             WORD32 cols1,             WORD32 cols2,
WORD32 row_stride1,      WORD32 row_stride2,      WORD32 acc_shift,
WORD32 bias_shift,      WORD32 bias_precision, VOID * p_scratch);
WORD32 xa_nn_matXvec_8x8_8_tanh
(WORD8 * p_out,           WORD8 * p_mat1,          WORD8 * p_mat2,
WORD8 * p_vec1,          WORD8 * p_vec2,          VOID * p_bias,
WORD32 rows,             WORD32 cols1,             WORD32 cols2,
WORD32 row_stride1,      WORD32 row_stride2,      WORD32 acc_shift,
WORD32 bias_shift,      WORD32 bias_precision, VOID * p_scratch);
WORD32 xa_nn_matXvec_8x8_8_sigmoid
(WORD8 * p_out,           WORD8 * p_mat1,          WORD8 * p_mat2,
WORD8 * p_vec1,          WORD8 * p_vec2,          VOID * p_bias,
WORD32 rows,             WORD32 cols1,             WORD32 cols2,
WORD32 row_stride1,      WORD32 row_stride2,      WORD32 acc_shift,
WORD32 bias_shift,      WORD32 bias_precision, VOID * p_scratch);
WORD32 xa_nn_matXvec_f32xf32_f32_tanh
(FLOAT32 * p_out,        FLOAT32 * p_mat1,        FLOAT32 * p_mat2,
FLOAT32 * p_vec1,        FLOAT32 * p_vec2,        FLOAT32 * p_bias,
WORD32 rows,             WORD32 cols1,             WORD32 cols2,
WORD32 row_stride1,      WORD32 row_stride2      FLOAT32 * p_scratch);
WORD32 xa_nn_matXvec_f32xf32_f32_sigmoid
(FLOAT32 * p_out,        FLOAT32 * p_mat1,        FLOAT32 * p_mat2,
FLOAT32 * p_vec1,        FLOAT32 * p_vec2,        FLOAT32 * p_bias,
WORD32 rows,             WORD32 cols1,             WORD32 cols2,
WORD32 row_stride1,      WORD32 row_stride2      FLOAT32 * p_scratch);

```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, FLOAT32 *	p_mat1	rows*cols1	Input matrix 1, fixed or floating point
WORD16 *, WORD8 *, FLOAT32 *	p_mat2	rows*cols2	Input matrix 2, fixed or floating point
WORD16 *, WORD8 *, FLOAT32 *	p_vec1	cols1*1	Input vector 1, fixed or floating point
WORD16 *, WORD8 *, FLOAT32 *	p_vec2	cols2*1	Input vector 2, fixed or floating point

VOID *, FLOAT32 *	p_bias	rows*1	Bias vector, fixed or floating point
WORD32	rows		Number of rows in matrix 1,2, bias and output
WORD32	cols1		Number of columns in matrix 1 and rows in vector 1
WORD32	cols2		Number of columns in matrix 2 and rows in vector 2
WORD32	row_stride1		Row offset of matrix 1
WORD32	row_stride2		Row offset of matrix 2
WORD32	acc_shift		Shift applied to accumulator
WORD32	bias_shift		Shift applied to bias
WORD32	bias_precision		Precision of bias in bytes
Output			
WORD8 *, WORD16 *, FLOAT32 *	p_out	rows*1	Output, fixed (Q7, Q15) or floating point
Temporary			
VOID *, FLOAT32 *	p_scratch	rows*4	Scratch (temporary) memory pointer

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
cols1, cols2	Multiples of 4
row_stride1, row_stride2	Multiples of 4 (2 in case of floating point)
p_mat1, p_mat2, p_vec1, p_vec2, p_out	Aligned on 8-byte boundary Should not overlap
p_bias	Aligned on (size of one element)-byte boundary (for kernels supporting multiple bias precision maximum size of one element should be considered as the alignment requirement) (Aligned on 8-byte for floating point kernels) Should not overlap
p_scratch	Cannot be NULL Aligned on 8-byte boundary Should not overlap
p_mat1, p_vec1, p_bias, p_out	Cannot be NULL
acc_shift, bias_shift	{-31, ..., 31}
bias_precision	{-1, 8, 16, 32, 64} (-1 in case of floating point)

3.1.3 Matrix X Vector Batch Kernels

Description

These kernels perform the operation of multiplication of a single matrix with a series of vectors along with bias addition; that is, $z_i = \text{mat1} * \text{vec1}_i + \text{bias}$. These kernels can also be viewed as matrix X matrix-transpose multiplication kernels. The column dimension of `mat1` must match the row dimension of vectors in `vec1`. Bias and resulting output vector sequence `z` have as many number of rows as `mat1`. `vec1` is a sequence of `vec_count` number of input vectors and bias is added to each resulting vector after multiplication with `mat1`. Thus, output `z` has dimensions `rows*vec_count`. `vec_count` number of input vectors and output vectors are provided as array of pointers arguments to kernel API.

`bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and output respectively. Both `bias_shift` and `acc_shift` can be either positive or negative where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as `matXvec` multiplication – accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

The `row_stridel` argument is provided in kernel API for row offset of `mat1`. Note, input matrix is expected to be appropriately padded in case of `row_stridel > cols1`.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

The arguments, `mat1_zero_bias`, `vec1_zero_bias`, are provided to convert the `asym8` inputs into their real values and perform `matXvec` batch operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to `asym8`.

Function variants available are `xa_nn_matXvec_batch_[p]x[q]_[r]`, where:

- `[p]`: Matrix precision in bits
- `[q]`: Vector precision in bits
- `[r]`: Output precision in bits

Precision

There are five variants available:

Type	Description
<code>16x16_64</code>	16-bit matrix inputs, 16-bit vector inputs, 64-bit output vectors
<code>8x16_64</code>	8-bit matrix inputs, 16-bit vector inputs, 64-bit output vectors
<code>8x8_32</code>	8-bit matrix inputs, 8-bit vector inputs, 32-bit output vectors
<code>f32xf32_f32</code>	float32 matrix inputs, float32 vector inputs, float32 output
<code>asym8xasym8_asym8</code>	asym8 matrix inputs, asym8 vector inputs, asym8 output vectors

Algorithm

$$z_{n,i} = 2^{acc-shift} \left(\sum_{m=0}^{cols1-1} mat1_{n,m} \cdot vec1_{m,i} + 2^{bias-shift} bias_n \right),$$

$$n = 0, \dots, \overline{rows - 1} ; \quad i = 0, \dots, \overline{vec-count - 1}$$

In case of floating-point and asym8 routine, `acc_shift=0` and `bias_shift=0`.

Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

Prototype

```
WORD32 xa_nn_matXvec_batch_16x16_64
(WORD64 ** p_out,          WORD16 * p_mat1,          WORD16 ** p_vec1,
 WORD16 * p_bias,          WORD32 rows,              WORD32 cols1,
 WORD32 row_stridel,       WORD32 acc_shift,          WORD32 bias_shift,
 WORD32 vec_count);

WORD32 xa_nn_matXvec_batch_8x16_64
(WORD64 ** p_out,          WORD8 * p_mat1,            WORD16 ** p_vec1,
 WORD16 * p_bias,          WORD32 rows,              WORD32 cols1,
 WORD32 row_stridel,       WORD32 acc_shift,          WORD32 bias_shift,
 WORD32 vec_count);

WORD32 xa_nn_matXvec_batch_8x8_32
(WORD32 ** p_out,          WORD8 * p_mat1,            WORD8 ** p_vec1,
 WORD8 * p_bias,           WORD32 rows,              WORD32 cols1,
 WORD32 row_stridel,       WORD32 acc_shift,          WORD32 bias_shift,
 WORD32 vec_count);

WORD32 xa_nn_matXvec_batch_f32xf32_f32
(FLOAT32 ** p_out,         FLOAT32 * p_mat1,          FLOAT32 ** p_vec1,
 FLOAT32 * p_bias,         WORD32 rows,              WORD32 cols1,
 WORD32 row_stridel,       WORD32 vec_count);

WORD32 xa_nn_matXvec_batch_asym8xasym8_asym8
(UWORD8 ** p_out,          UWORD8 * p_mat1,           UWORD8 ** p_vec1,
 WORD32 * p_bias,          WORD32 rows,              WORD32 cols1,
 WORD32 row_stridel,       WORD32 vec_count,          WORD32 mat1_zero_bias,
 WORD32 vec1_zero_bias,    WORD32 out_multiplier,     WORD32 out_shift,
 WORD32 out_zero_bias);
```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, UWORD8 *, FLOAT32 *	p_mat1	rows*cols 1	Input matrix, fixed or floating point
WORD16 **, WORD8 **, UWORD8 **, FLOAT32 **	p_vec1	cols1*vec _count	Input vector pointers, fixed or floating point
WORD16 *, WORD8 *, WORD32 *	p_bias	rows*1	Bias vector, fixed or floating point

Type	Name	Size	Description
Float32 *			
WORD32	rows		Number of rows in input matrix, bias and output
WORD32	cols1		Number of columns in input matrix and rows in input vector
WORD32	row_stridel		Row offset of input matrix
WORD32	acc_shift		Shift applied to accumulator
WORD32	bias_shift		Shift applied to bias
WORD32	vec_count		Number of input vectors
WORD32	mat1_zero_bias		Zero offset of matrix 1
WORD32	vec1_zero_bias		Zero offset of vector 1
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output
WORD32	out_zero_bias		Zero offset of output
Output			
WORD32 **, WORD64 **, UWORD8 **, Float32 **	p_out	rows*vec_count	Output vector pointers, fixed or floating point

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
row_stridel, cols1	Multiples of 4 (2 in case of floating point)
p_mat1	Aligned on 8-byte boundary Should not overlap Cannot be NULL
p_vec1	Aligned on 4-byte boundary Cannot be NULL Should not overlap p_vec1[0] to p_vec[vec_count-1] – Aligned on 4*(size of one element)-byte boundary (8-byte for floating point) Cannot be NULL Should not overlap
p_bias	Aligned on (size of one element)-byte boundary Cannot be NULL Should not overlap
p_out	Aligned on 4-byte boundary Cannot be NULL Should not overlap p_out[0] to p_out[vec_count-1] – Aligned on (size of one element)-byte boundary Cannot be NULL

	Should not overlap
acc_shift, bias_shift, out_shift	{-31, ..., 31}
vec_count	Greater than 0
mat1_zero_bias, vec1_zero_bias	{-255, ..., 0}
out_multiplier	Greater than 0
out_zero_bias	{0, ..., 255}

3.1.4 Matrix Multiplication Kernels

Description

These kernels perform the operation of multiplication of a matrix `mat1` with another matrix `mat2` along with bias addition; that is, $z = \text{mat1} * \text{mat2} + \text{bias}$. The first matrix should be stored in row major order and the second matrix should be stored in column major order. The first matrix is of dimensions `rows` x `cols`. The second matrix `mat2` is of dimensions `cols` x `vec_count`. These kernels can also be viewed as a modification of the Matrix X Vector Batch kernels. The column dimension of `mat1` matches the row dimension of `mat2` i.e. the length of each vector in `p_mat2`. Bias and resulting output vector sequence `z` have as many numbers of rows as `mat1`. `mat2` is a sequence of `vec_count` number of input vectors and bias is added to each resulting vector after multiplication with `mat1`. Thus, output `z` has dimensions `rows` * `vec_count`. The arguments `vec_offset` and `out_offset` are offsets to the next vector and output addresses. The argument `out_stride` defines the row offset for the output matrix. For standard matrix multiplication, `vec_offset` should be equal to `cols`, `out_offset` equal to 1 and `out_stride` should be equal to `vec_count` i.e. columns of `mat2`.

The `bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and output respectively. Both `bias_shift` and `acc_shift` can be either positive or negative where positive value denotes a left shift and negative value denotes a right shift.

The `bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as multiplication – accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

The `row_stride` argument indicates the offset to next row of `mat1`.

The `vec_offset` argument refers to the column offset of `mat2`.

Similarly, the `out_offset` and `out_stride` arguments refer to the column offset and row offset of the output matrix `rows` * `vec_count` respectively.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

The arguments, `mat1_zero_bias`, `mat2_zero_bias`, are provided to convert the `asym8` inputs into their real values and perform `matXvec` batch operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to `asym8`.

Function variants available are `xa_nn_matmul_[p]x[q]_[r]`, where:

- `[p]`: Matrix 1 precision in bits
- `[q]`: Matrix 2 precision in bits
- `[r]`: Output precision in bits

Precision

There are five variants available:

Type	Description
<code>16x16_16</code>	16-bit matrix inputs, 16-bit vector inputs, 16-bit output vectors
<code>8x16_16</code>	8-bit matrix inputs, 16-bit vector inputs, 16-bit output vectors
<code>8x8_8</code>	8-bit matrix inputs, 8-bit vector inputs, 8-bit output vectors
<code>f32xf32_f32</code>	float32 matrix inputs, float32 vector inputs, float32 output
<code>asym8xasym8_asym8</code>	asym8 matrix inputs, asym8 vector inputs, asym8 output vectors
<code>per_chan_sym8sxasym8s_asym8s</code>	per channel quantized sym8s matrix inputs, asym8s vector inputs, asym8s output vectors

Algorithm

$$z_{n,i} = 2^{acc-shift} \left(\sum_{m=0}^{cols1-1} mat1_{n,m} \cdot mat2_{m,i} + 2^{bias-shift} bias_n \right),$$

$$n = 0, \dots, \overline{rows} - 1 ; \quad i = 0, \dots, \overline{vec-count} - 1$$

In case of floating-point and `asym8` routine, `acc_shift=0` and `bias_shift=0`.

Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

Prototype

```
WORD32 xa_nn_matmul_16x16_16
(WORD16 * p_out,          WORD16 * p_mat1,          WORD16 * p_mat21,
 WORD16 * p_bias,         WORD32 rows,              WORD32 cols,
 WORD32 row_stride,       WORD32 acc_shift,          WORD32 bias_shift,
 WORD32 vec_count,        WORD32 vec_offset,         WORD32 out_offset,
 WORD32 out_stride);
WORD32 xa_nn_matmul_8x16_16
(WORD16 * p_out,          WORD8 * p_mat1,           WORD16 * p_mat2,
 WORD16 * p_bias,         WORD32 rows,              WORD32 cols,
 WORD32 row_stride,       WORD32 acc_shift,          WORD32 bias_shift,
 WORD32 vec_count,        WORD32 vec_offset,         WORD32 out_offset,
 WORD32 out_stride);
```

```

WORD32 xa_nn_matmul_8x8_8
(WORD8 * p_out,          WORD8 * p_mat1,          WORD16 * p_mat2,
 WORD8 * p_bias,         WORD32 rows,             WORD32 cols,
 WORD32 row_stride,     WORD32 acc_shift,         WORD32 bias_shift,
 WORD32 vec_count,      WORD32 vec_offset,        WORD32 out_offset,
 WORD32 out_stride);
WORD32 xa_nn_matmul_f32xf32_f32
(FLOAT32 * p_out,        FLOAT32 * p_mat1,        FLOAT32 * p_mat2,
 FLOAT32 * p_bias,      WORD32 rows,             WORD32 cols,
 WORD32 row_stride,     WORD32 acc_shift,         WORD32 bias_shift,
 WORD32 vec_count,      WORD32 vec_offset,        WORD32 out_offset,
 WORD32 out_stride);
WORD32 xa_nn_matmul_asym8xasym8_asym8
(UWORD8 * p_out,         UWORD8 * p_mat1,         UWORD16 * p_mat2,
 WORD32 * p_bias,        WORD32 rows,             WORD32 cols,
 WORD32 row_stride,     WORD32 vec_count,        WORD32 vec_offset,
 WORD32 out_offset,     WORD32 out_stride,        WORD32 mat1_zero_bias,
 WORD32 mat2_zero_bias, WORD32 out_multiplier,    WORD32 out_shift,
 WORD32 out_zero_bias);
WORD32 xa_nn_matmul_per_chan_sym8sxasym8s_asym8s
(WORD8 * p_out,          const WORD8 * p_mat1,    const WORD8 * p_mat2,
 const WORD32 * p_bias,  WORD32 rows,             WORD32 cols,
 WORD32 row_stride,     WORD32 vec_count,        WORD32 vec_offset,
 WORD32 out_offset,     WORD32 out_stride,        WORD32 vec1_zero_bias,
 const WORD32 *p_out_multiplier, const WORD32 *p_out_shift,
 WORD32 out_zero_bias);

```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, UWORD8 *, FLOAT32 *	p_mat1	rows*cols	Input matrix, fixed or floating point
WORD16 *, WORD8 *, UWORD8 *, FLOAT32 *	p_mat2	Cols * vec_count	Input matrix, fixed or floating point
WORD16 *, WORD8 *, WORD32 *, FLOAT32 *	p_bias	rows*1	Bias vector, fixed or floating point
WORD32	rows		Number of rows in input matrix, bias and output
WORD32	cols		Number of columns in input matrix and rows in input vector
WORD32	row_stride		Row offset of input matrix
WORD32	acc_shift		Shift applied to accumulator
WORD32	bias_shift		Shift applied to bias
WORD32	vec_count		Number of input vectors
WORD32	vec_offset		Offset to the next vector address
WORD32	out_offset		Offset to the next output address
WORD32	out_stride		Row offset of output matrix
WORD32	mat1_zero_bias		Zero offset of matrix 1

Type	Name	Size	Description
WORD32	vec1_zero_bias		Zero offset of vector 1
WORD32 WORD32 *	out_multiplier , p_out_multiplier		Multiplier value of output, Pointer to output multiplier value
WORD32	out_shift, p_out_shift		Shift value of output, Pointer to output shift value
WORD32	out_zero_bias		Zero offset of output
Output			
WORD16 *, WORD8 *, UWORD8 *, FLOAT32 *	p_out	rows*vec_count	Output matrix, fixed or floating point

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_mat1, p_mat2, p_out,	Aligned on (size of one element)-byte boundary Cannot be NULL Should not overlap
p_bias	Aligned on (size of one element)-byte boundary
acc_shift, bias_shift, out_shift	{-31, ..., 31}
vec_count	Greater than 0
vec_offset, out_offset, out_stride	Should not be 0
mat1_zero_bias, vec1_zero_bias	{-255, ..., 0}
out_multiplier	Greater than 0
p_out_multiplier, p_out_shift	Aligned on (size of one element)-byte boundary Cannot be NULL (range of values are specified for out_multiplier and out_shift)
out_zero_bias	{0, ..., 255}

3.1.5 Matrix X Vector Kernels with Output Stride

Description

These kernels perform a single matXvec operation with bias addition; that is, $z = \text{mat1} * \text{vec1} + \text{bias}$. The column dimension of `mat1` must match the row dimension of `vec1`. Bias and resulting output vector `z` have as many rows as `mat1`.

`row_stride1` is provided in kernel API for row offsets of `mat1`. Note, input matrices are expected to be appropriately padded in case of `row_stride > cols`.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

The argument `out_stride` is helpful in storing the output at a given offset.

The argument `vec1_zero_bias` is provided to convert the quantized 8-bit inputs into their real values and perform `matXvec` operation. The `out_multiplier` and `out_shift` values are used to convert real values of output to 16-bit.

Function variants available are `xa_nn_matXvec_[p]x[q]_[r]`, where:

- `[p]`: Matrix precision in bits
- `[q]`: Vector precision in bits
- `[r]`: Output precision in bits

Precision

There is one variant available:

Type	Description
<code>sym8sxasym8s_16</code>	sym8s matrix inputs, asym8s vector inputs, asym8s output

Algorithm

$$z_n = \left(\sum_{m=0}^{cols1-1} mat1_{n,m} \cdot vec1_m + bias_n \right)$$

Prototype

```
WORD32 xa_nn_matXvec_out_stride_sym8sxasym8s_16
(WORD16 * p_out,      const WORD8 * p_mat1,  const WORD8 * p_vec1,
 const WORD32 * p_bias, WORD32 rows,        WORD32 cols1,
 WORD32 row_stride1,  WORD32 out_stride,    WORD32 vec1_zero_bias,
 WORD32 out_multiplier, WORD32 out_shift);
```

Arguments

Type	Name	Size	Description
Input			
const WORD8 *	<code>p_mat1</code>	<code>rows*cols1</code>	Input matrix, sym8s
const WORD8 *	<code>p_vec1</code>	<code>cols1*1</code>	Input vector, asym8s
const WORD32 *	<code>p_bias</code>	<code>rows*1</code>	Bias vector

Type	Name	Size	Description
WORD32	rows		Number of rows in matrix and number of elements in bias
WORD32	cols1		Number of columns in matrix and elements in vector
WORD32	row_stride1		Row offset of matrix
WORD32	out_stride		Row offset of output
WORD32	vec1_zero_bias		Zero offset of vector
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output
Output			
WORD16 *	p_out	rows*1	Output, 16-bit

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
row_stride1, cols1	row_stride1 >= cols1
p_mat1, p_vec1, p_bias, p_out	Aligned on <size of one element> boundary Should not overlap
p_mat1, p_vec1, p_out	Cannot be NULL
out_shift	{-31, ..., 31}
vec1_zero_bias	{-127....., 128} for asym8s
out_multiplier	Greater than 0

3.2 Convolution Kernels

3.2.1 Standard 2D Convolution Kernel

Description

These kernels perform the 2D convolution operation as $z = \text{inp}(\ast)\text{kernel} + \text{bias}$. A 3D input cube (`input_height` x `input_width` x `input_channels`), is convolved with a 3D kernel cube (`kernel_height` x `kernel_width` x `input_channels`) to produce a 2D convolution output plane (`out_height` x `out_width`). With `out_channels` number of such 3D kernels, output cube (`out_height` x `out_width` x `out_channels`) is produced. The bias having dimension (`out_channels`) is added after the convolution (one bias value is added to each output channel) to produce the final output.

Note, the depth or channels dimension (`input_channels`) of input and kernel must be identical for 2D convolution.

`bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and output, respectively. Both `bias_shift` and `acc_shift` can be either positive or negative where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as convolution - accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

The `x_stride` and `y_stride` arguments in kernel API define the step size of the kernel when traversing the input in width and height dimensions respectively.

The `x_padding` argument defines padding to the left of the input in the width dimension and the `y_padding` argument defines padding to the top of the input in the height dimension.

The right padding is calculated based on `out_width` as `right_paddding = kernel_width + (out_width - 1) * x_stride - (x_padding + input_width)`.

The bottom padding is calculated based on `out_height` as `bottom_paddding = kernel_height + (out_height - 1) * y_stride - (y_padding + input_height)`.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

For the 8x16, 16x16 and the f32 variants the kernel is expected to be padded in the depth or channels dimension if the number of `input_channels` is not a multiple of 4 in case of fixed-point variants, and 2 in case of floating-point variant.

These kernels require temporary buffer for convolution computation. This temporary buffer is provided by `p_scratch` argument of kernel API. The size of temporary buffer should be queried using `xa_nn_conv2d_std_getsize()` helper API.

The arguments `input_zero_bias`, `kernel_zero_bias` are provided to convert the asym8 inputs into their real values and perform Standard 2D Convolution operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to asym8.

These kernels expect input and kernel cubes in `SHAPE_CUBE_DWH_T` shape type and can produce output cube in either `SHAPE_CUBE_DWH_T` or `SHAPE_CUBE_WHD_T` shape type. The `out_data_format` argument to kernel API controls the output cube shape type.

Function variants available are `xa_nn_conv2d_std_[p]`, where:

- `[p]`: precision in bits

Precision

There are five variants available.

Type	Description
16x16	16-bit kernel, 16-bit input, 16-bit output
8x16	8-bit kernel, 16-bit input, 16-bit output
8x8	8-bit kernel, 8-bit input, 8-bit output
f32	float32 kernel, float32 input, float32 output
asym8xasym8	asym8 kernel, asym8 input, asym8 output
per_chan_sy m8sxasym8s	per channel quantized sym8s kernel, asym8s input, asym8s output

Algorithm

$$z_{h,w,d} = 2^{acc-shift} \left(\sum_{i=0}^{K_H-1} \sum_{j=0}^{K_W-1} \sum_{k=0}^{I_C-1} in_{pad}(h*y-stride+i),(w*x-stride+j),k \cdot ker_{pad}_{d,i,j,k} + 2^{bias-shift} b_d \right)$$

$$h = 0, \dots, \overline{out-height - 1}, w = 0, \dots, \overline{out-width - 1},$$

$$d = 0, \dots, \overline{out-channels - 1}$$

In case of floating-point and asym8 kernel , `acc_shift=0` and `bias_shift=0`.

Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

in_{pad}, ker_{pad} denote the padded `p_inp` and padded `p_ker` shapes, respectively.

K_H, K_W, I_C denote `kernel_height`, `kernel_width`, and `input_channels`, respectively.

b denotes the `bias` shape.

Prototype

```
WORD32 xa_nn_conv2d_std_getsize
(WORD32 input_height,      WORD32 input_channels, WORD32 kernel_height,
 WORD32 kernel_width,     WORD32 y_stride,      WORD32 y_padding,
 WORD32 out_height,       WORD32 input_precision);

WORD32 xa_nn_conv2d_std_16x16
(WORD16 * p_out,           WORD16 * p_inp,        WORD16 * p_ker,
 WORD16 * p_bias,          WORD32 input_height,   WORD32 input_width,
 WORD32 input_channels,    WORD32 kernel_height, WORD32 kernel_width,
 WORD32 out_channels,      WORD32 x_stride,     WORD32 y_stride,
 WORD32 x_padding,         WORD32 y_padding,   WORD32 out_height,
 WORD32 out_width,         WORD32 bias_shift,    WORD32 acc_shift,
 WORD32 out_data_format,   VOID * p_scratch);

WORD32 xa_nn_conv2d_std_8x16
(WORD16 * p_out,           WORD16 * p_inp,        WORD8 * p_ker,
 WORD16 * p_bias,          WORD32 input_height,   WORD32 input_width,
 WORD32 input_channels,    WORD32 kernel_height, WORD32 kernel_width,
 WORD32 out_channels,      WORD32 x_stride,     WORD32 y_stride,
 WORD32 x_padding,         WORD32 y_padding,   WORD32 out_height,
 WORD32 out_width,         WORD32 bias_shift,    WORD32 acc_shift,
 WORD32 out_data_format,   VOID * p_scratch);

WORD32 xa_nn_conv2d_std_8x8
```

```

(WORD8 * p_out,          WORD8 * p_inp,          WORD8 * p_ker,
WORD8 * p_bias,          WORD32 input_height,    WORD32 input_width,
WORD32 input_channels,   WORD32 kernel_height, WORD32 kernel_width,
WORD32 out_channels,     WORD32 x_stride,      WORD32 y_stride,
WORD32 x_padding,        WORD32 y_padding,     WORD32 out_height,
WORD32 out_width,        WORD32 bias_shift,    WORD32 acc_shift,
WORD32 out_data_format,  VOID * p_scratch);
WORD32 xa_nn_conv2d_std_f32
(FLOAT32 * p_out,        const FLOAT32 * p_inp, const FLOAT32 * p_ker,
Const FLOAT32 * p_bias,  WORD32 input_height,    WORD32 input_width,
WORD32 input_channels,   WORD32 kernel_height, WORD32 kernel_width,
WORD32 out_channels,     WORD32 x_stride,      WORD32 y_stride,
WORD32 x_padding,        WORD32 y_padding,     WORD32 out_height,
WORD32 out_width,        WORD32 out_data_format, VOID * p_scratch);
WORD32 xa_nn_conv2d_std_asym8xasym8
(UWORD8* p_out,          const UWORD8* p_inp,      const UWORD8* p_kernel,
const WORD32* p_bias,    WORD32 input_height,    WORD32 input_width,
WORD32 input_channels,   WORD32 kernel_height, WORD32 kernel_width,
WORD32 out_channels,     WORD32 x_stride,      WORD32 y_stride,
WORD32 x_padding,        WORD32 y_padding,     WORD32 out_height,
WORD32 out_width,        WORD32 input_zero_bias, WORD32 kernel_zero_bias,
WORD32 out_multiplier,   WORD32 out_shift,      WORD32 out_zero_bias,
WORD32 out_data_format,  VOID *p_scratch);
WORD32 xa_nn_conv2d_std_per_chan_sym8sxasym8s
(WORD8* p_out,           const WORD8* p_inp,          const WORD8* p_kernel,
const WORD32* p_bias,    WORD32 input_height,    WORD32 input_width,
WORD32 input_channels,   WORD32 kernel_height, WORD32 kernel_width,
WORD32 out_channels,     WORD32 x_stride,      WORD32 y_stride,
WORD32 x_padding,        WORD32 y_padding,     WORD32 out_height,
WORD32 out_width,        WORD32 input_zero_bias, WORD32* p_out_multiplier,
WORD32 * p_out_shift,    WORD32 out_zero_bias,  WORD32 out_data_format,
VOID *p_scratch);

```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *,	p_inp	input_height* input width* input_channels	Input cube, fixed, floating point, asym8u or asym8s, in SHAPE_CUBE_DWH_T
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *,	p_ker	out_channels* (kernel_height* kernel width* input_channels)	Kernel cube, fixed, floating point, asym8u or sym8s in SHAPE_CUBE_DWH_T
WORD16 *, WORD8 *, const WORD32 *, FLOAT32 *,	p_bias	out_channels	Bias vector, fixed or floating point
WORD32	input_height		Input height
WORD32	input_width		Input width
WORD32	input_channels		Number of input channels
WORD32	kernel_height		Kernel height
WORD32	kernel_width		Kernel width

WORD32	out_channels		Number of output channels
WORD32	x_stride		Horizontal stride over input
WORD32	y_stride		Vertical stride over input
WORD32	x_padding		Left padding width on input
WORD32	y_padding		Top padding height on input
WORD32	out_height		Output height
WORD32	out_width		Output width
WORD32	bias_shift		Shift applied to bias
WORD32	acc_shift		Shift applied to accumulator
WORD32	input_zero_bias		Zero offset of input
WORD32	kernel_zero_bias		Zero offset of kernel
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output
WORD32	out_zero_bias		Zero offset of output
WORD32	out_data_format		Output data format 0:SHAPE_CUBE_DWH_T 1:SHAPE_CUBE_WHD_T
VOID *	p_scratch	xa_nn_conv2d_std_getsize()	Scratch memory pointer
Output			
WORD16 *, WORD8 *, const UWORD8 *, FLOAT32 *,	p_out	(out_height* out_width)* out_channels	Output cube, fixed, floating point, asym8u or asym8s as per the out_data_format argument.

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_ker, p_scratch	Cannot be NULL
	Should not overlap
	Aligned on 8-byte boundary (p_bias needs to be only 4-byte aligned for asym8 variant)
	For p_scratch - memory size >= size returned by xa_nn_conv2d_std_getsize()
p_out, p_inp, p_bias	Cannot be NULL
	Should not overlap
	Aligned on (size of one element)-byte boundary

input_height, input_width, input_channels	Greater than or equal to 1
kernel_height	{1, 2, ..., input_height}
kernel_width	{1, 2, ..., input_width}
out_channels	Greater than or equal to 1
x_stride	Greater than or equal to 1
y_stride	Greater than or equal to 1
x_padding, y_padding	Greater than or equal to 0
out_height, out_width	Greater than or equal to 1
acc_shift, bias_shift, out_shift	{-31 31} for fixed point APIs
input_zero_bias, kernel_zero_bias	{-255, ..., 0}
out_multiplier	Greater than 0
out_zero_bias	{0 ..., 255}
out_data_format	Can be 0: SHAPE_CUBE_DWH_T or 1: SHAPE_CUBE_WHD_T

3.2.2 Standard 1D Convolution Kernel

Description

These kernels perform the 1D convolution operation as $z = \text{inp}(*)\text{kernel} + \text{bias}$. A 3D input cube (`input_height` x `input_width` x `input_channels`) is convolved with a 3D kernel cube (`kernel_height` x `input_width` x `input_channels`) to produce a 1D convolution output vector (`out_height`). With `out_channels` number of such 3D kernels, output matrix (`out_height` x `out_channels`) is produced. The bias having dimension (`out_channels`) is added after the convolution (one bias value is added to each output column) to produce the final output.

Note, the depth or channels dimension (`input_channels`) of input and kernel must be identical, and width dimension (`input_width`) of input and kernel also must be identical for 1D convolution.

`bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and output, respectively. Both `bias_shift` and `acc_shift` can be either positive or negative, where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as convolution - accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

The `y_stride` argument to kernel API defines the step size of the kernel when traversing the input in height dimension.

The `y_padding` argument defines padding to the top of the input in the height dimension.

The bottom padding is calculated based on `out_height` as `bottom_paddding = kernel_height + (out_height - 1) * y_stride - (y_padding + input_height)`.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

The kernel is expected to be padded if the product `input_channels*input_width` is not a multiple of 4 in case of fixed-point variants, and 2 in case of floating-point variant.

These kernels require temporary buffer for convolution computation. This temporary buffer is provided by `p_scratch` argument of kernel API. The size of temporary buffer should be queried using `xa_nn_conv1d_std_getsize()` helper API.

The arguments `input_zero_bias`, `kernel_zero_bias` are provided to convert the `asym8` inputs into their real values and perform Standard 1D Convolution operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to `asym8`.

These kernels expect input and kernel cubes in `SHAPE_CUBE_DWH_T` shape type and can produce output matrix with either `(out_height x out_channels)` or `(out_channels x out_height)` dimensions. The `out_data_format` argument to kernel API controls the output matrix height and width order.

Function variants available are `xa_nn_conv1d_std_[p]`, where:

- `[p]`: precision in bits

Precision

There are five variants available:

Type	Description
16x16	16-bit kernel, 16-bit input, 16-bit output
8x16	8-bit kernel, 16-bit input, 16-bit output
8x8	8-bit kernel, 8-bit input, 8-bit output
f32	float32 kernel, float32 input, float32 output
asym8xasym8	asym8 kernel, asym8 input, asym8 output

Algorithm

$$z_{h,d} = 2^{acc-shift} \left(\sum_{i=0}^{K_H-1} \sum_{j=0}^{I_W-1} \sum_{k=0}^{I_C-1} in_{pad(h*y-stride+i),j,k} \cdot ker_{pad_{d,i,j,k}} + 2^{bias-shift} b_d \right)$$

$$h = 0, \dots, \overline{out_height - 1}, d = 0, \dots, \overline{out_channels - 1}$$

In case of floating-point and `asym8` kernel, `acc_shift=0` and `bias_shift=0`.

Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

n_{pad} , ker_{pad} denote the padded `p_inp` and padded `p_ker` shapes, respectively.

K_H, I_W, I_C denote kernel_height, input_width, and input_channels, respectively.

b denotes the bias shape.

Prototype

```
WORD32 xa_nn_conv1d_std_getsize
(WORD32 kernel_height, WORD32 input_width, WORD32 input_channels,
 WORD32 input_precision);

WORD32 xa_nn_conv1d_std_16x16
(WORD16 * p_out, WORD16 * p_inp, WORD16 * p_ker,
 WORD16 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 out_channels,
 WORD32 y_stride, WORD32 y_padding, WORD32 out_height,
 WORD32 bias_shift, WORD32 acc_shift, WORD32 out_data_format,
 VOID * p_scratch);

WORD32 xa_nn_conv1d_std_8x16
(WORD16 * p_out, WORD16 * p_inp, WORD8 * p_ker,
 WORD16 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 out_channels,
 WORD32 y_stride, WORD32 y_padding, WORD32 out_height,
 WORD32 bias_shift, WORD32 acc_shift, WORD32 out_data_format,
 VOID * p_scratch);

WORD32 xa_nn_conv1d_std_8x8
(WORD8 * p_out, WORD8 * p_inp, WORD8 * p_ker,
 WORD8 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 out_channels,
 WORD32 y_stride, WORD32 y_padding, WORD32 out_height,
 WORD32 bias_shift, WORD32 acc_shift, WORD32 out_data_format,
 VOID * p_scratch);

WORD32 xa_nn_conv1d_std_f32
(FLOAT32 * p_out, FLOAT32 * p_inp, FLOAT32 * p_ker,
 FLOAT32 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 out_channels,
 WORD32 y_stride, WORD32 y_padding, WORD32 out_height,
 WORD32 out_data_format, VOID * p_scratch);

WORD32 xa_nn_conv1d_std_asym8xasym8
(UWORD8 * p_out, UWORD8 * p_inp, UWORD8 * p_kernel,
 WORD32 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 out_channels,
 WORD32 y_stride, WORD32 y_padding, WORD32 out_height,
 WORD32 input_zero_bias, WORD32 kernel_zero_bias, WORD32 out_multiplier,
 WORD32 out_shift, WORD32 out_zero_bias, WORD32 out_data_format,
 VOID * p_scratch);
```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, const UWORD8 *, FLOAT32 *,	p_inp	input_height* input width* input_channels	Input cube, fixed or floating point, in SHAPE_CUBE_DWH_T
WORD16 *, WORD8 *, const UWORD8 *, FLOAT32 *,	p_ker	out_channels* (kernel_height* input width* input_channels)	Kernel cube, fixed or floating point, in SHAPE_CUBE_DWH_T

WORD16 *, WORD8 *, const WORD32 *, FLOAT32 *,	p_bias	out_channels	Bias vector, fixed or floating point
WORD32	input_height		Input height
WORD32	input_width		Input width
WORD32	input_channels		Number of input channels
WORD32	kernel_height		Kernel height
WORD32	out_channels		Number of output channels
WORD32	y_stride		Vertical stride over input
WORD32	y_padding		Top padding height on input
WORD32	out_height		Output height
WORD32	bias_shift		Shift applied to bias
WORD32	acc_shift		Shift applied to accumulator
WORD32	input_zero_bias		Zero offset of input
WORD32	kernel_zero_bias		Zero offset of kernel
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output
WORD32	out_zero_bias		Zero offset of output
WORD32	out_data_format		Output matrix order 0: out_height x out_channels 1: out_channels x out_height
VOID *	p_scratch	xa_nn_convld_std_getsize()	Scratch memory pointer
Output			
WORD16 *, WORD8 *, const UWORD8 *, FLOAT32 *,	p_out	out_height* out_channels	Output matrix, fixed or floating point, as per the out_data_format argument.

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_out, p_inp, p_ker, p_bias, p_scratch	Cannot be NULL Should not overlap Aligned on 8-byte boundary For p_scratch - memory size >= size returned by xa_nn_conv1d_std_getsize()
input_height, input_width, input_channels	Greater than or equal to 1
kernel_height	{1, 2, ..., input_height}
out_channels	Greater than or equal to 1
y_stride	{1, 2, ..., kernel_height}
y_padding	Greater than or equal to 0
out_height	Greater than or equal to 1
acc_shift, bias_shift, out_shift	{-31 31} for fixed point APIs
input_zero_bias, kernel_zero_bias	{-255, ..., 0}
out_multiplier	Greater than 0
out_zero_bias	{0, ..., 255}
out_data_format	Can be 0: out_height x out_channels or 1: out_channels x out_height

3.2.3 Depthwise Separable 2D Convolution Kernel

Depthwise Separable 2D Convolution is computed in two steps using following two low level kernels:

- First step: xa_nn_conv2d_depthwise_xx() low level kernel

These kernels convolve each input 2D plane (input_height x input_width) from input cube (input_height x input_width x input_channels) with channels_multiplier number of 2D kernels (kernel_height x kernel_width) to produce channels_multiplier number of 2D output planes (out_height x out_width). Thus, with kernel cube of dimension (kernel_height x kernel_width x (channels_multiplier * input_channels)), output cube of dimension (out_height x out_width x (channels_multiplier * input_channels)) is produced. Bias is added to the convolution output. There is one bias value for each output 2D plane; that is, bias is a vector of dimension (channels_multiplier * input_channels).

- Second step: xa_nn_conv2d_pointwise_xx() low level kernel

These kernels take output cube (out_height x out_width x (channels_multiplier * input_channels)) of first step as input and perform pointwise multiplication with kernel vector (channels_multiplier * input_channels) in depth dimension to produce output 2D plane (out_height x out_width). Thus, with out_channels kernel vectors, output cube of dimension (out_height x out_width x out_channels) is produced. Bias is added to the pointwise multiplication output. There is one bias value for each output 2D plane; that is, bias is a vector of dimension out_channels.

Note, for depthwise separable 2D convolution, $(\text{channels_multiplier} * \text{input_channels})$ must be multiple of 4 (see Section 3.2.3.2 for details).

Following are the descriptions for these two low level kernels.

3.2.3.1 Depthwise 2D Convolution Kernel

Description

These kernels perform the 2D depthwise convolution operation as $z = \text{inp} (*) \text{kernel} + \text{bias}$. These kernels convolve each input 2D plane ($\text{input_height} \times \text{input_width}$) from input cube ($\text{input_height} \times \text{input_width} \times \text{input_channels}$) with $\text{channels_multiplier}$ number of 2D kernels ($\text{kernel_height} \times \text{kernel_width}$) to produce $\text{channels_multiplier}$ number of 2D output planes ($\text{out_height} \times \text{out_width}$). Thus, with kernel cube of dimension ($\text{kernel_height} \times \text{kernel_width} \times (\text{channels_multiplier} * \text{input_channels})$), output cube of dimension ($\text{out_height} \times \text{out_width} \times (\text{channels_multiplier} * \text{input_channels})$) is produced. Bias is added to the convolution output. There is one bias value for each output 2D plane; that is, bias is a vector of dimension $(\text{channels_multiplier} * \text{input_channels})$.

`bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and output respectively. Both `bias_shift` and `acc_shift` can be either positive or negative where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as convolution - accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

The `x_stride` and `y_stride` arguments in kernel API define the step size of the kernel when traversing the input in width and height dimensions, respectively.

The `x_padding` argument defines padding to the left of the input in the width dimension, and `y_padding` argument defines padding to the top of the input in the height dimension.

The right padding is calculated based on `out_width` as $\text{right_padding} = \text{kernel_width} + (\text{out_width} - 1) * \text{x_stride} - (\text{x_padding} + \text{input_width})$.

The bottom padding is calculated based on `out_height` as $\text{bottom_padding} = \text{kernel_height} + (\text{out_height} - 1) * \text{y_stride} - (\text{y_padding} + \text{input_height})$.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

These kernels require a temporary buffer for convolution computation. This temporary buffer is provided by the `p_scratch` argument of kernel API. The size of temporary buffer should be queried using `xa_nn_conv2d_depthwise_getsize()` helper API.

The arguments `input_zero_bias`, `kernel_zero_bias` are provided to convert the `asym8` inputs into their real values and perform Depthwise 2D Convolution operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to `asym8`.

The depthwise kernels expect input cube in `SHAPE_CUBE_DWH_T` and `SHAPE_CUBE_WHD_T` shape type and produce output cube in `SHAPE_CUBE_DWH_T` shape type respectively. The `inp_data_format` argument to the kernel API can be 0 or 1 to indicate input cube shape respectively.

The `out_data_format` argument to the kernel API must be 0 for all the kernels to indicate output cube shape.

Function variants available are `xa_nn_conv2d_depthwise_[p]`, where:

- `[p]`: precision in bits

Precision

There are six variants available:

Type	Description
16x16	16-bit kernel, 16-bit input, 16-bit output
8x16	8-bit kernel, 16-bit input, 16-bit output
8x8	8-bit kernel, 8-bit input, 8-bit output
f32	float32 kernel, float32 input, float32 output
asym8xasym8	asym8 kernel, asym8 input, asym8 output
per_chan_sym8sxasym8s	per channel quantized sym8s kernel, asym8s input, asym8s output

Algorithm

$$z_{h,w,d \cdot C_M+m} = 2^{acc-shift} \left(\sum_{i=0}^{K_H-1} \sum_{j=0}^{K_W-1} in_{pad_{(h*y-stride+i),(w*x-stride+j),d}} \cdot ker_{pad_{i,j,(d \cdot C_M+m)}} + 2^{bias-shift} b_{0,0,d \cdot C_M+m} \right)$$

$$h = 0, \dots, \overline{out-height - 1}, w = 0, \dots, \overline{out-width - 1},$$

$$d = 0, \dots, \overline{input-channels - 1},$$

$$m = 0, \dots, \overline{channels-multiplier - 1}$$

In case of floating-point and `asym8` kernel, `acc_shift=0` and `bias_shift=0`.

Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

in_{pad}, ker_{pad} denote the padded `p_inp` and padded `p_ker` shapes, respectively.

K_H, K_W, C_M denote `kernel_height`, `kernel_width`, and `channels_multiplier`, respectively.

b denotes the `bias` shape.

Prototype

```

WORD32 xa_nn_conv2d_depthwise_getsize
(WORD32 input_height, WORD32 input_width, WORD32 input_channels,
 WORD32 kernel_height, WORD32 kernel_width, WORD32 channels_multiplier,
 WORD32 x_stride, WORD32 y_stride, WORD32 x_padding,
 WORD32 y_padding, WORD32 output_height, WORD32 output_width,
 WORD32 circ_buf_precision, WORD32 inp_data_format);

WORD32 xa_nn_conv2d_depthwise_16x16
(WORD16 * p_out, const WORD16 * p_kernel, const WORD16 * p_inp,
 Const WORD16 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 kernel_width,
 WORD32 channels_multiplier, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 acc_shift, WORD32 bias_shift,
 WORD32 inp_data_format, WORD32 out_data_format, VOID * p_scratch);

WORD32 xa_nn_conv2d_depthwise_8x16
(WORD16 * p_out, const WORD8 * p_kernel, const WORD16 * p_inp,
 const WORD16 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 kernel_width,
 WORD32 channels_multiplier, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 acc_shift, WORD32 bias_shift,
 WORD32 inp_data_format, WORD32 out_data_format, VOID * p_scratch);

WORD32 xa_nn_conv2d_depthwise_8x8
(WORD8 * p_out, const WORD8 * p_kernel, const WORD8 * p_inp,
 const WORD8 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 kernel_width,
 WORD32 channels_multiplier, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 acc_shift, WORD32 bias_shift,
 WORD32 inp_data_format, WORD32 out_data_format, VOID * p_scratch);

WORD32 xa_nn_conv2d_depthwise_f32
(FLOAT32 * p_out, const FLOAT32 * p_kernel, const FLOAT32 * p_inp,
 const FLOAT32 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 kernel_width,
 WORD32 channels_multiplier, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 inp_data_format, WORD32 out_data_format,
 VOID * p_scratch);

WORD32 xa_nn_conv2d_depthwise_asym8xasym8
(pUWORD8 p_out, const UWORD8 * p_kernel, const UWORD8 * p_inp,
 const WORD32 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 kernel_width,
 WORD32 channels_multiplier, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 input_zero_bias, WORD32 kernel_zero_bias,
 WORD32 out_multiplier, WORD32 out_shift, WORD32 out_zero_bias,
 WORD32 inp_data_format, WORD32 out_data_format, pVOID p_scratch);

WORD32 xa_nn_conv2d_depthwise_per_chan_sym8sxasym8s
(pWORD8 p_out, const WORD8 * p_kernel, const WORD8 * p_inp,
 const WORD32 * p_bias, WORD32 input_height, WORD32 input_width,
 WORD32 input_channels, WORD32 kernel_height, WORD32 kernel width,
 WORD32 channels_multiplier, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out height,
 WORD32 out width, WORD32 input_zero_bias, const WORD32 * p_out_multiplier,
 const WORD32 * p_out_shift, WORD32 out_zero_bias, WORD32 inp_data_format,
 WORD32 out_data_format, pVOID p_scratch);

```

Arguments

Type	Name	Size	Description
Input			
const WORD16 *, const WORD8 *, const UWORD8 *, const FLOAT32 *	p_ker	kernel_height* kernel_width* input_channels* channels_multiplier	Kernel cube, fixed or floating point, asym8u or sym8s, in SHAPE_CUBE_DW H or SHAPE_CUBE_WH D_T
const WORD16 *, const WORD8 *, const UWORD8 *, const FLOAT32 *	p_inp	input_height* input_width* input_channels	Input cube, fixed or floating point, asym8u or asym8s in SHAPE_CUBE_DW H or SHAPE_CUBE_WH D_T
const WORD16 *, const WORD8 *, const WORD32 *, const FLOAT32 *	p_bias	input_channels*chan nels_multiplier	Bias vector, fixed or floating point
WORD32	input_height		Input height
WORD32	input_width		Input width
WORD32	input_channels		Number of input channels
WORD32	kernel_height		Kernel height
WORD32	kernel_width		Kernel width
WORD32	channels_multipl ier		Multiplier value for each input channel
WORD32	x_stride		Horizontal stride over input
WORD32	y_stride		Vertical stride over input
WORD32	x_padding		Left padding width on input
WORD32	y_padding		Right padding height on input
WORD32	out_height		Output height
WORD32	out_width		Output width
WORD32	acc_shift		Shift applied to accumulator
WORD32	bias_shift		Shift applied to bias
WORD32	input_zero_bias		Zero offset of input
WORD32	kernel_zero_bias		Zero offset of kernel
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output

Type	Name	Size	Description
WORD32	out_zero_bias		Zero offset of output
WORD32	inp_data_format		Input and Kernel data format 0:SHAPE_CUBE_DWH_T 1:SHAPE_CUBE_WHD_T
WORD32	out_data_format		Output data format 0:SHAPE_CUBE_DWH_T
VOID *	p_scratch	xa_nn_conv2d_depthwise_getsize()	Scratch memory pointer
Output			
WORD16 *, WORD8 *, const UWORD8 *, FLOAT32 *	p_out	out_height* out_width* input_channels* channels_multiplier	Output cube, fixed or floating point, asym8u or asym8s, in SHAPE_CUBE_DWH_T

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_kernel, p_inp	Cannot be NULL
	Should not overlap
	Aligned on 8-byte boundary
p_out, p_bias	Cannot be NULL
	Should not overlap
	Aligned on (size of one element)-byte boundary
p_scratch	Cannot be NULL
	Should not overlap
	Aligned on 8-byte boundary
	memory size >= size returned by xa_nn_conv2d_depthwise_getsize()
input_height, input_width, input_channels	Greater than or equal to 1
kernel_height	{1,2,...,input_height}
kernel_width	{1,2,...,input_width}
channels_multiplier	Greater than or equal to 1
x_stride	{1,2,...,kernel_width}
y_stride	{1,2,...,kernel_height}
x_padding, y_padding	Greater than or equal to 0
out_height, out_width	Greater than or equal to 1

<code>acc_shift, bias_shift, out_shift</code>	{-31 31} for fixed point APIs
<code>input_zero_bias</code>	{-255,....., 0} for asym8u input, {-127....., 128} for asym8s input
<code>Kernel_zero_bias</code>	{-255....., 0} for asym8u kernel
<code>out_multiplier</code>	Greater than 0
<code>out_zero_bias</code>	{0,.....,255} for asym8u output, {-128....., 127} for asym8s output
<code>inp_data_format</code>	can be 0: SHAPE_CUBE_DWH_T or 1: SHAPE_CUBE_WHD_T
<code>out_data_format</code>	must be 0: SHAPE_CUBE_DWH_T

3.2.3.2 Pointwise 2D Convolution Kernel

Description

These kernels perform pointwise multiplication of input cube (`input_height` x `input_width` x `input_channels`) with kernel vector (`input_channels`) in depth dimension to produce output 2D plane (`input_height` x `input_width`). Thus, with `out_channels` kernel vectors, output cube of dimension (`input_height` x `input_width` x `out_channels`) is produced. Bias is added to the pointwise multiplication output. There is one bias value for each output 2D plane; that is, bias is a vector of dimension `out_channels`.

The `bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and output respectively. Both `bias_shift` and `acc_shift` can be either positive or negative, where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as convolution - accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

The arguments `input_zero_bias`, `kernel_zero_bias` are provided to convert the asym8 inputs into their real values and perform Pointwise 2D Convolution operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to asym8.

The pointwise kernels expect input cube in SHAPE_CUBE_DWH_T shape type, kernel as matrix, bias as vector and produce output cube in SHAPE_CUBE_DWH_T or SHAPE_CUBE_WHD_T shape type as per the `out_data_format` argument value 0 or 1 to kernel API.

Function variants available are `xa_nn_conv2d_pointwise_[p]`, where:

- `[p]`: precision in bits

Precision

There are six variants available:

Type	Description
16x16	16-bit kernel, 16-bit input, 16-bit output
8x16	8-bit kernel, 16-bit input, 16-bit output
8x8	8-bit kernel, 8-bit input, 8-bit output
f32	float32 kernel, float32 input, float32 output
asym8xasym8	asym8u kernel, asym8u input, asym8 output
sym8xasym8	sym8s kernel, asym8s input, asym8s output

Algorithm

$$z_{h,w,d} = 2^{acc-shift} \left(\sum_{k=0}^{I_C-1} in_{h,w,k} \cdot ker_{d,0,0,k} + 2^{bias-shift} b_{0,0,d} \right)$$

$$h = 0, \dots, \overline{input-height - 1}, w = 0, \dots, \overline{input-width - 1},$$

$$d = 0, \dots, \overline{out_channels - 1}$$

In case of floating-point and asym8 kernel, `acc_shift=0` and `bias_shift=0`. Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

`in`, `ker` denote the `p_inp`, and `p_ker` shapes respectively.

`IC` denotes `input_channels`

`b` denotes the `bias` shape

Prototype

```
WORD32 xa_nn_conv2d_pointwise_16x16
(WORD16 * p_out,          WORD16 * p_kernel,      WORD16 * p_inp,
 WORD16 * p_bias,         WORD32 input_height,    WORD32 input_width,
 WORD32 input_channels,   WORD32 out_channels,    WORD32 acc_shift,
 WORD32 bias_shift,       WORD32 out_data_format);
WORD32 xa_nn_conv2d_pointwise_8x16
(WORD16 * p_out,          WORD8 * p_kernel,        WORD16 * p_inp,
 WORD16 * p_bias,         WORD32 input_height,    WORD32 input_width,
 WORD32 input_channels,   WORD32 out_channels,    WORD32 acc_shift,
 WORD32 bias_shift,       WORD32 out_data_format);
WORD32 xa_nn_conv2d_pointwise_8x8
(WORD8 * p_out,           WORD8 * p_kernel,        WORD8 * p_inp,
 WORD8 * p_bias,          WORD32 input_height,    WORD32 input_width,
 WORD32 input_channels,   WORD32 out_channels,    WORD32 acc_shift,
 WORD32 bias_shift,       WORD32 out_data_format);
WORD32 xa_nn_conv2d_pointwise_f32
(FLOAT32 * p_out,         FLOAT32 * p_kernel,    FLOAT32 * p_inp,
 FLOAT32 * p_bias,        WORD32 input_height,    WORD32 input_width,
 WORD32 input_channels,   WORD32 out_channels,
 WORD32 out_data_format);
WORD32 xa_nn_conv2d_pointwise_asym8xasym8
(pUWORD8 p_out,           pUWORD8 p_kernel,        pUWORD8 p_inp,
 pWORD32 p_bias,          WORD32 input_height,    WORD32 input_width,
```



```

WORD32 input_channels,      WORD32 out_channels,      WORD32 input_zero_bias,
WORD32 kernel_zero_bias,   WORD32 out_multiplier, WORD32 out_shift,
WORD32 out_zero_bias,      WORD32 out_data_format);
WORD32 xa_nn_conv2d_pointwise_per_chan_sym8sxasym8s
(WORD8 * p_out,              const WORD8 * p_ker,        const WORD8 * p_inp,
 const WORD32 * p_bias,      WORD32 input_height,        WORD32 input_width,
 WORD32 input_channels,      WORD32 out_channels,        WORD32 input_zero_bias,
 WORD32 * p_out_multiplier, WORD32 * p_out_shift,        WORD32 out_zero_bias,
 WORD32 out_data_format);

```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, FLOAT32 *, const UWORD8 *, const WORD8 *	p_ker	out_channels * input_channels	Kernel matrix, fixed or floating point
WORD16 *, WORD8 *, FLOAT32 *, const UWORD8 *, const WORD8 *	p_inp	input_height* input width* input_channels	Input cube, fixed or floating point, in SHAPE_CUBE_DWH_T
WORD16 *, WORD8 *, FLOAT32 *, const WORD32 *	p_bias	out_channels	Bias vector, fixed or floating point
WORD32	input_height		Input height
WORD32	input_width		Input width
WORD32	input_channels		Number of input channels
WORD32	out_channels		Number of output channels
WORD32	acc_shift		Shift applied to accumulator
WORD32	bias_shift		Shift applied to bias
WORD32	input_zero_bias		Zero offset of input
WORD32	kernel_zero_bias		Zero offset of kernel
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output
WORD32	out_zero_biast		Zero offset of output
WORD32	out_data_format		Output data format 0:SHAPE_CUBE_DWH_T 1:SHAPE_CUBE_WHD_T
Output			
WORD16 *, WORD8 *, FLOAT32 *, UWORD8 *	p_out	(out_height* out_width)* out_channels	Output cube, fixed, floating point, asym8u or asym8s, as per the out_data_format argument.

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_out, p_ker, p_inp, p_bias	Cannot be NULL Should not overlap
input_height, input_width	Greater than or equal to 1
input_channels, out_channels	Greater than or equal to 1
acc_shift, bias_shift	{-31 31} for fixed point APIs
input_zero_bias, kernel_zero_bias	{-255, ..., 0}
out_multiplier	Greater than 0
out_zero_bias	{0, ..., 255}
out_data_format	can be 0: SHAPE_CUBE_DWH_T or 1: SHAPE_CUBE_WHD_T

3.3 Activation Kernels

3.3.1 Sigmoid

Description

These kernels perform the sigmoid operation on input vector x and give output vector as $y = \text{sigmoid}(x)$. Both the input and output vectors have size `vec_length`.

The fixed-point kernels accept 32-bit input in Q6.25 format and give output in Q16.15 (32-bit), Q15 (16-bit), or Q7 (8-bit) format.

For the asym8 kernels both the input and output are of asym8 datatype.

Function variants available are `xa_nn_vec_sigmoid_[p]_[q]`, where:

- [p]: Input precision in bits
- [q]: Output precision in bits

Precision

There are five variants available.

Type	Description
32_32	32-bit input, 32-bit output

32_16	32-bit input, 16-bit output
32_8	32-bit input, 8-bit output
f32_f32	float32 input, float32 output
asym8xasym8	asym8 input, asym8 output

Algorithm

$$y_n = \frac{1}{1 + \exp(-x_n)}, \quad n = 0, \dots, \overline{vec_length} - 1$$

Prototype

```
WORD32 xa_nn_vec_sigmoid_32_32
(WORD32 * p_out,          const WORD32 * p_vec,      WORD32 vec_length);
WORD32 xa_nn_vec_sigmoid_32_16
(WORD16 * p_out,          const WORD32 * p_vec,      WORD32 vec_length);
WORD32 xa_nn_vec_sigmoid_32_8
(WORD8 * p_out,           const WORD32 * p_vec,      WORD32 vec_length);
WORD32 xa_nn_vec_sigmoid_f32_f32
(FLOAT32 * p_out,         const FLOAT32 * p_vec,     WORD32 vec_length);
WORD32 xa_nn_vec_sigmoid_asym8_asym8
(UWORD8 * p_out,          const UWORD8 * p_vec,      WORD32 zero_point,
WORD32 input_range_radius, WORD32 input_multiplier, WORD32 input_left_shift,
WORD32 vec_length);
```

Arguments

Type	Name	Size	Description
Input			
const WORD32 *, const UWORD8 *, const FLOAT32 *	p_vec	vec_length	Input vector, Q6.25, floating point or asym8
WORD32	zero_point		bias value
WORD32	input_range_radius		Range radius: output = (abs (x _i - zero_point) <= radius)? sigmoid() : 0
WORD32	input_multiplier		Multiplier value of input
WORD32	input_left_shift		Left Shift value of input
Output			
WORD32 *, WORD16 *, WORD8 *, UWORD8 *, FLOAT32 *	p_out	vec_length	Output vector, fixed (Q16.15, Q15, Q7) floating point or asym8

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_vec, p_out	Aligned on (size of one element)-byte boundary
	Cannot be NULL

3.3.2 Tanh

Description

These kernels perform the hyperbolic tangent operation on input vector x and give output vector as $y = \tanh(x)$. Both the input and output vectors have size `vec_length`.

The fixed-point kernels accept 32-bit input in Q6.25 format and give output in Q16.15 (32-bit), Q15 (16-bit), or Q7 (8-bit) format.

Function variants available are `xa_nn_vec_tanh_[p]_[q]`, where:

- `[p]`: Input precision in bits
- `[q]`: Output precision in bits

Precision

There are four variants available:

Type	Description
32_32	32-bit input, 32-bit output
32_16	32-bit input, 16-bit output
32_8	32-bit input, 8-bit output
f32_f32	float32 input, float32 output

Algorithm

$$y_n = \tanh(x_n), \quad n = 0, \dots, \overline{vec_length} - 1$$

Prototype

```
WORD32 xa_nn_vec_tanh_32_32
(WORD32 * p_out,      const WORD32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_tanh_32_16
(WORD16 * p_out,      const WORD32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_tanh_32_8
(WORD8 * p_out,       const WORD32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_tanh_f32_f32
(FLOAT32 * p_out,     const FLOAT32 * p_vec,  WORD32 vec_length);
```

Arguments

Type	Name	Size	Description
------	------	------	-------------

Input			
const WORD32 *, const FLOAT32 *	p_vec	vec_length	Input vector, Q6.25 or floating point
WORD32	vec_length		Length of input vector
Output			
WORD32 *, WORD16 *, WORD8 *, FLOAT32 *	p_out	vec_length	Output vector, fixed (Q16.15, Q15, Q7) or floating point

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_vec, p_out	Aligned on (size of one element)-byte boundary Cannot be NULL

3.3.3 Rectifier Linear Unit (ReLU)

Description

These kernels compute the rectifier linear unit function of input vector x and give output vector as $y = \text{relu}(x)$. Both the input and output vectors have size `vec_length`.

The fixed-point routines accept 32-bit input in Q6.25 format and gives 32-bit output in Q16.15 format.

The `threshold` argument to `relu` kernel API allows to set upper threshold for proper compression of output signal and is expected in Q16.15 format. In `relu1` and `relu6` kernels, the thresholds are set to 1 and 6, respectively. The `relu1` and `relu6` kernels are available only for 32-bit fixed point and floating-point precision.

The standard ReLU kernels `relu_std` can be used when the `threshold` is not required.

Function variants available are `xa_nn_vec_relu_[p]_[q]`, `xa_nn_vec_relu1_[p]_[q]`, and `xa_nn_vec_relu6_[p]_[q]` `xa_nn_vec_relu_std_[p]_[q]`, where:

- [p]: Input precision in bits
- [q]: Output precision in bits

Precision

There are four variants available:

Type	Description
------	-------------

32_32	32-bit input, 32-bit output
f32_f32	float32 input, float32 output
16_16	16-bit input, 16-bit output
8_8	8-bit input, 8-bit output

Algorithm

$$y_n = \max(0, \min(K)), \quad n = 0, \dots, \overline{vec_length} - 1$$

K represents threshold

For standard ReLU, there is no threshold

$$y_n = \max(0, x_n), \quad n = 0, \dots, \overline{vec_length} - 1$$

Prototype

```
WORD32 xa_nn_vec_relu_32_32
(WORD32 * p_out,      const WORD32 * p_vec,   WORD32 threshold,
 WORD32 vec_length);
WORD32 xa_nn_vec_relu_f32_f32
(FLOAT32 * p_out,     const FLOAT32 * p_vec,  FLOAT32 threshold,
 WORD32 vec_length);
WORD32 xa_nn_vec_relu_16_16
(WORD16 * p_out,      const WORD16 * p_vec,   WORD16 threshold,
 WORD32 vec_length);
WORD32 xa_nn_vec_relu_8_8
(WORD8 * p_out,       const WORD8 * p_vec,    WORD8 threshold,
 WORD32 vec_length);
WORD32 xa_nn_vec_relu1_32_32
(WORD32 * p_out,      const WORD32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_relu1_f32_f32
(FLOAT32 * p_out,     const FLOAT32 * p_vec,  WORD32 vec_length);
WORD32 xa_nn_vec_relu6_32_32
(WORD32 * p_out,      const WORD32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_relu6_f32_f32
(FLOAT32 * p_out,     const FLOAT32 * p_vec,  WORD32 vec_length);
WORD32 xa_nn_vec_relu_std_32_32
(WORD32 * p_out,      const WORD32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_relu_std_f32_f32
(FLOAT32 * p_out,     const FLOAT32 * p_vec,  WORD32 vec_length);
WORD32 xa_nn_vec_relu_std_16_16
(WORD16 * p_out,      const WORD16 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_relu_std_8_8
(WORD8 * p_out,       const WORD8 * p_vec,    WORD32 vec_length);
```

Arguments

Type	Name	Size	Description
Input			
const WORD32 *, const FLOAT32 *, const WORD16 *	p_vec	vec_length	Input vector, fixed-point or floating point

const WORD8 *			
WORD32	vec_length		length of input vector
WORD32 FLOAT32 WORD16 WORD8	threshold		threshold, fixed or floating point
Output			
WORD32 *, FLOAT32 *, WORD16 *, WORD8 *	p_out	vec_length	Output vector, fixed or floating point

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_vec, p_out	Aligned on (size of one element)-byte boundary
	Cannot be NULL

3.3.4 Softmax

Description

These kernels compute the softmax (normalized exponential function) of input vector x and give output vector as $y = \text{softmax}(x)$. Both the input and output vectors have size `vec_length`.

The fixed-point kernels accept 32-bit input in Q6.25 format and give 32-bit output in Q16.15 format.

For the `asym8u` and `asym8s` kernels, both the input and output are of same precision.

Function variants available are `xa_nn_vec_softmax_[p]_[q]`, where:

- [p]: Input precision in bits
- [q]: Output precision in bits

Precision

There are three variants available:

Type	Description
32_32	32-bit input, 32-bit output
f32_f32	float32 input, float32 output
asym8_asym8	asym8 input, asym8 output
asym8s_asym8s	asym8s input, asym8s output

Algorithm

$$y_n = \frac{\exp(x_n)}{\sum_k \exp(x_k)}, \quad n = 0, \dots, \overline{vec_length} - 1$$

Prototype

```
WORD32 xa_nn_vec_softmax_32_32
(WORD32 * p_out,          const WORD32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_softmax_f32_f32
(FLOAT32 * p_out,        const FLOAT32 * p_vec,   WORD32 vec_length);
WORD32 xa_nn_vec_softmax_asym8_asym8
(UWORD8 * p_out,          const UWORD8 * p_vec,   WORD32 diffmin,
WORD32 input_left_shift, WORD32 input_multiplier,
WORD32 vec_length,       pVOID p_scratch);
WORD32 xa_nn_vec_softmax_asym8s_asym8s
(WORD8 * p_out,           const WORD8 * p_vec,     WORD32 diffmin,
WORD32 input_left_shift,  WORD32 input_multiplier,
WORD32 vec_length,       pVOID p_scratch);
```


Arguments

Type	Name	Size	Description
Input			
const WORD32 *, const UWORD8 *, const FLOAT32 *	p_vec	vec_length	Input vector, Q6.25, floating point, asym8u or asym8s
WORD32	diffmin		Diffmin value: output = $((x_i - \max) > \text{diffmin}) ? \text{softmax}() : 0$
WORD32	input_ left_shift		left shift value of input
WORD32	input_ multiplier		multiplier value of input
WORD32	vec_length		Length of input vector
Output			
WORD32 *, UWORD8 *, FLOAT32 *	p_out	vec_length	Output vector, Q16.15, floating point, asym8u or asym8s
Temporary			
VOID *, FLOAT32 *	p_scratch		Scratch (temporary) memory pointer

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_vec, p_out	Should not overlap
	Cannot be NULL

3.3.5 Activation Min Max

Description

These kernels compute the activation minimum and maximum value of input vector x and give output vector as $y = \text{activation_min_max}(x)$. Both the input and output vectors have size `num_elm`.

The routine accepts asym8 or float32 input and gives asym8 or float32 output.

The `activation_min` and `activation_max` arguments to the kernel API allow to set the threshold for proper compression of the output. The kernel is a generic implementation of the ReLU function.

Function variant available is `xa_nn_vec_activation_min_max_[p]_[q]`, where:

- [p]: Input precision in bits
- [q]: Output precision in bits

Precision

There are four variants available:

Type	Description
f32_f32	float32 input, float32 output
asym8xasym8	asym8 input, asym8 output
16_16	16-bit input, 16-bit output
8_8	8-bit input, 8-bit output

Algorithm

$$y_n = \max(\text{activation_min}, \min(x_n, \text{activation_max})), \quad n = 0, \dots, \text{vec_length} - 1$$

activation_min represents lower threshold.

activation_max represents upper threshold.

Prototype

```
WORD32 xa_nn_vec_activation_min_max_f32_f32
(FLOAT32 * p_out,      const FLOAT32 * p_vec, FLOAT32 activation_min,
 FLOAT32 activation_max, WORD32  vec_length);
WORD32 xa_nn_vec_activation_min_max_asym8_asym8
(UWORD8 * p_out,      const UWORD8 * p_vec,  int activation_min,
 int activation_max,   WORD32  vec_length);
WORD32 xa_nn_vec_activation_min_max_16_16
(WORD16 * p_out,      const WORD16 * p_vec,  int activation_min,
 int activation_max,   WORD32  vec_length);
WORD32 xa_nn_vec_activation_min_max_8_8
(WORD8 * p_out,      const WORD8 * p_vec,   int activation_min,
 int activation_max,   WORD32  vec_length);
```

Arguments

Type	Name	Size	Description
Input			
const UWORD8 *, const FLOAT32 *, const WORD16 *, const WORD8 *	p_vec	vec_length	Input vector, floating-point, asym8 or fixed point.
WORD32	vec_length		Length of input vector
WORD32, FLOAT32	activation_min		Lower threshold value, floating-point, asym8 or fixed point.
WORD32, FLOAT32	activation_max		Upper threshold value, floating-point, asym8 or fixed point
Output			

UWORD8 *, FLOAT32 *, WORD16 *, WORD8 *	p_out	vec_length	Output vector, floating-point, asym8 or fixed point
---	-------	------------	--

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_vec, p_out	Aligned on (size of one element)-byte boundary
	Cannot be NULL

3.4 Pooling Kernels

3.4.1 Average Pool Kernel

Description

These kernels compute 2D average pool on a set of input planes (matrices) x and give a set of planes y as output.

The pooling region is defined by `kernel_height` and `kernel_width`. It is shifted over the input plane in steps of `x_stride` horizontally and in steps of `y_stride` vertically to generate the specified output plane size. The input is extended by zero padding as specified by the padding region. The padding is determined by the parameters `x_padding`, `y_padding` for left and top side padding respectively, and `out_width`, `out_height` for right and bottom padding respectively. Around the edges of input planes, if only a part of pooling region is covering input plane then only average of those elements is calculated and the denominator is the number of elements from input in current pooling region.

The average pool kernels accept input as 8-bit, 16-bit integer, asym8 or single precision floating point format and give output in same precision as input.

These kernels require temporary buffer for average pool computation. This temporary buffer is provided by the `p_scratch` argument of kernel API. The size of temporary buffer should be queried using `xa_nn_avgpool_getsize()` helper API.

The average pool kernels expect input cube in `SHAPE_CUBE_DWH_T` and `SHAPE_CUBE_WHD_T` shape type and produce output cube in `SHAPE_CUBE_DWH_T` and `SHAPE_CUBE_WHD_T` shape type respectively. The `inp_data_format` and `out_data_format` arguments to the kernel API can be 0 or 1 to indicate input and output cube shapes respectively.

The value of `inp_data_format` and `out_data_format` must be equal.

Note, the fixed-point 8-bit average pool kernel, `xa_nn_avgpool_8` can be used for the quantized int8 datatype.

Function variants available are `xa_nn_avgpool_[p]`, where:

- `[p]`: Input and Output precision in bits

Precision

There are four variants available:

Type	Description
8	8-bit input, 8-bit output
16	16-bit input, 16-bit output
f32	float32 input, float32 output
asym8	asym8 input, asym8 output

Algorithm

$$z_{h,w,d} = \frac{1}{K_H K_W} \left(\sum_{i=0}^{K_H-1} \sum_{j=0}^{K_W-1} in_{(h*y-stride+i),(w*x-stride+j),d} \right)$$

$$h = 0, \dots, out_height - 1, \quad w = 0, \dots, out_width - 1,$$

$$d = 0, \dots, out_channels - 1$$

in denotes padded input cube, *z* denotes output

K_H, K_W denote `kernel_height`, `kernel_width` respectively.

Prototype

```
WORD32 xa_nn_avgpool_getsize
(WORD32 input_channels, WORD32 inp_precision, WORD32 out_precision,
 WORD32 input_height, WORD32 input_width, WORD32 kernel_height,
 WORD32 kernel_width, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 inp_data_format, WORD32 out_data_format);

WORD32 xa_nn_avgpool_8
(WORD8 * p_out, const WORD8 * p_inp, WORD32 input_height,
 WORD32 input_width, WORD32 input_channels, WORD32 kernel_height,
 WORD32 kernel_width, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 inp_data_format, WORD32 out_data_format,
 VOID * p_scratch);

WORD32 xa_nn_avgpool_16
(WORD16 * p_out, const WORD16 * p_inp, WORD32 input_height,
 WORD32 input_width, WORD32 input_channels, WORD32 kernel_height,
 WORD32 kernel_width, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 inp_data_format, WORD32 out_data_format,
 VOID * p_scratch);

WORD32 xa_nn_avgpool_f32
(FLOAT32 * p_out, const FLOAT32 * p_inp, WORD32 input_height,
 WORD32 input_width, WORD32 input_channels, WORD32 kernel_height,
```

```

WORD32 kernel_width,      WORD32 x_stride,      WORD32 y_stride,
WORD32 x_padding,        WORD32 y_padding,      WORD32 out_height,
WORD32 out_width,        WORD32 inp_data_format, WORD32 out_data_format,
VOID * p_scratch);
WORD32 xa_nn_avgpool_asym8
(WORD8* p_out,             const UWORD8* p_inp,      WORD32 input_height,
WORD32 input_width,       WORD32 input_channels,    WORD32 kernel_height,
WORD32 kernel_width,     WORD32 x_stride,        WORD32 y_stride,
WORD32 x_padding,        WORD32 y_padding,      WORD32 out_height,
WORD32 out_width,        WORD32 inp_data_format, WORD32 out_data_format,
VOID *p_scratch);

```

Arguments

Type	Name	Size	Description
Input			
WORD8 *, WORD16 *, const UWORD8 *, const FLOAT32 *	p_inp	input_height * input_width * input_channels	Input cube
WORD32	input_height		Input height
WORD32	input_width		Input width
WORD32	input_channels		Input number of channels
WORD32	kernel_height		Pooling window height
WORD32	kernel_width		Pooling window width
WORD32	x_stride		Horizontal stride over input
WORD32	y_stride		Vertical stride over input
WORD32	x_padding		Left padding width on input
WORD32	y_padding		Top padding height on input
WORD32	out_height		Output height
WORD32	out_width		Output width
WORD32	inp_data_format		Input data format: 0: SHAPE_CUBE_DWH_T 1: SHAPE_CUBE_WHD_T
WORD32	out_data_format		Output data format: 0: SHAPE_CUBE_DWH_T 1: SHAPE_CUBE_WHD_T
Output			
WORD8 *, WORD16 *, UWORD8 *, FLOAT32 *	p_out	out_height * out_width * input_channels	Output
Temporary			
VOID *	p_scratch	xa_nn_avgpool_ getsize()	Temporary / scratch memory

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
<code>p_inp, p_out</code>	Cannot be NULL Should not overlap
<code>p_scratch</code>	Cannot be NULL Aligned on 8-byte boundary Should not overlap Memory size \geq size returned by <code>xa_nn_avgpool_getsize()</code>
<code>input_height, input_width</code>	Greater than or equal to 1
<code>input_channels</code>	Greater than or equal to 1
<code>kernel_height</code>	{1, 2, ..., min(input_height, 256)} (for 8-bit and 16-bit) {1, 2, ..., input_height} (for float32)
<code>kernel_width</code>	{1, 2, ..., min(input_width, 256)} (for 8-bit and 16-bit) {1, 2, ..., input_width} (for float32)
<code>x_stride, y_stride</code>	Greater than or equal to 1
<code>x_padding, y_padding</code>	Greater than or equal to 0
<code>out_height, out_width</code>	greater than or equal to 1
<code>inp_data_format</code>	Can be 0: SHAPE_CUBE_DWH_T or 1: SHAPE_CUBE_WHD_T
<code>out_data_format</code>	Must be equal to <code>inp_data_format</code>

3.4.2 Max Pool Kernel

Description

These kernels perform 2D max pooling operation over a set of input planes `x` and give as output, a set of planes `y`.

The pooling region is defined by `kernel_height` and `kernel_width`. It is shifted over the input plane horizontally in steps of `x_stride` and vertically in steps of `y_stride` to generate the specified output plane size.

The input plane, padded with the maximum negative values, is considered while performing the max pooling operation. The padding region is determined by the parameters `x_padding`, `y_padding` for left and top side padding respectively, and `out_width`, `out_height` for right and bottom padding respectively.

The max pool kernels accept input as 8-bit, 16-bit integer, or single precision floating point format and give output in the same precision as input.

These kernels require temporary buffer for max pool computation. This temporary buffer is provided by the `p_scratch` argument of kernel API. The size of temporary buffer should be queried using the `xa_nn_maxpool_getsize()` helper API.

The max pool kernels expect input cube in SHAPE_CUBE_DWH_T and SHAPE_CUBE_WHD_T shape type and produce output cube in SHAPE_CUBE_DWH_T and SHAPE_CUBE_WHD_T shape type

respectively. The `inp_data_format` and `out_data_format` arguments to the kernel API can be 0 or 1 to indicate input and output cube shapes respectively.

The value of `inp_data_format` and `out_data_format` must be equal.

Note, the fixed-point 8-bit max pool kernel, `xa_nn_maxpool_8` can be used for the quantized int8 datatype.

Function variants available are `xa_nn_maxpool_[p]`, where:

- `[p]`: Input and Output precision in bits

Precision

There are four variants available:

Type	Description
8	8-bit input, 8-bit output
16	16-bit input, 16-bit output
f32	float32 input, float32 output
asym8	asym8 input, asym8 output

Algorithm

$$z_{h,w,d} = \max(in_{(h*y-stride+i),(w*x-stride+j),d})$$

$$h = 0, \dots, \overline{out_height - 1}, \quad w = 0, \dots, \overline{out_width - 1},$$

$$d = 0, \dots, \overline{out_channels - 1}$$

$$i = 0, \dots, K_H - 1, \quad j = 0, \dots, K_W - 1$$

in denotes padded input cube, *z* denotes output.

K_H, K_W denote `kernel_height`, `kernel_width` respectively.

Prototype

```
WORD32 xa_nn_maxpool_getsize
(WORD32 input_channels, WORD32 inp_precision, WORD32 out_precision,
 WORD32 input_height, WORD32 input_width, WORD32 kernel_height,
 WORD32 kernel_width, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 inp_data_format, WORD32 out_data_format);

WORD32 xa_nn_maxpool_8
(WORD8 * p_out, WORD8 * p_inp, WORD32 input_height,
 WORD32 input_width, WORD32 input_channels, WORD32 kernel_height,
 WORD32 kernel_width, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
 WORD32 out_width, WORD32 inp_data_format, WORD32 out_data_format,
 VOID * p_scratch);

WORD32 xa_nn_maxpool_16
(WORD16 * p_out, WORD16 * p_inp, WORD32 input_height,
 WORD32 input_width, WORD32 input_channels, WORD32 kernel_height,
 WORD32 kernel_width, WORD32 x_stride, WORD32 y_stride,
 WORD32 x_padding, WORD32 y_padding, WORD32 out_height,
```

```

WORD32 out_width,          WORD32 inp_data_format, WORD32 out_data_format,
VOID * p_scratch);
WORD32 xa_nn_maxpool_f32
(FLOAT32 * p_out,          const FLOAT32 * p_inp, WORD32 input_height,
WORD32 input_width,       WORD32 input_channels, WORD32 kernel_height,
WORD32 kernel_width,      WORD32 x_stride,      WORD32 y_stride,
WORD32 x_padding,         WORD32 y_padding,      WORD32 out_height,
WORD32 out_width,         WORD32 inp_data_format, WORD32 out_data_format,
VOID * p_scratch);
WORD32 xa_nn_maxpool_asym8
(UWORD8 * p_out,          const UWORD8 * p_inp, WORD32 input_height,
WORD32 input_width,       WORD32 input_channels, WORD32 kernel_height,
WORD32 kernel_width,      WORD32 x_stride,      WORD32 y_stride,
WORD32 x_padding,         WORD32 y_padding,      WORD32 out_height,
WORD32 out_width,         WORD32 inp_data_format, WORD32 out_data_format,
VOID * p_scratch);

```

Arguments

Type	Name	Size	Description
Input			
WORD8 *, WORD16 *, const UWORD8 *, const FLOAT32 *	p_inp	input_height * input_width * input_channels	Input cube
WORD32	input_height		Input height
WORD32	input_width		Input width
WORD32	input_channels		Input number of channels
WORD32	kernel_height		Pooling window height
WORD32	kernel_width		Pooling window width
WORD32	x_stride		Horizontal stride over input
WORD32	y_stride		Vertical stride over input
WORD32	x_padding		Left padding width on input
WORD32	y_padding		Top padding height on input
WORD32	out_height		Output height
WORD32	out_width		Output width
WORD32	inp_data_format		Input data format: 0: SHAPE_CUBE_DWH_T 1: SHAPE_CUBE_WHD_T
WORD32	out_data_format		Output data format: 0: SHAPE_CUBE_DWH_T 1: SHAPE_CUBE_WHD_T
Output			
WORD8 *, WORD16 *, UWORD8 *, FLOAT32 *	p_out	out_height * out_width * input_channels	Output
Temporary			
VOID *	p_scratch	xa_nn_maxpool_ getsize()	Temporary / scratch memory

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_inp, p_out	Cannot be NULL Should not overlap
p_scratch	Cannot be NULL Aligned on 8-byte boundary Should not overlap Memory size \geq size returned by <code>xa_nn_maxpool_getsize()</code>
input_height, input_width	Greater than or equal to 1
input_channels	Greater than or equal to 1
kernel_height	{1, 2, ..., input_height}
kernel_width	{1, 2, ..., input_width}
x_stride, y_stride	Greater than or equal to 1
x_padding, y_padding	Greater than or equal to 0
out_height, out_width	Greater than or equal to 1
inp_data_format	Can be 0: SHAPE_CUBE_DWH_T or 1: SHAPE_CUBE_WHD_T
out_data_format	Must be equal to <code>inp_data_format</code>

3.4.3 Fully Connected Kernel

Description

These kernels perform the operation of multiplication of weight matrix with input vectors in a fully connected neural network layer i.e. $z = \text{weight} \times \text{input} + \text{bias}$. The column dimension of weight must match the row dimension of input. Bias and resulting output vector z have as many numbers of rows as weight matrix.

The `bias_shift` and `acc_shift` arguments are provided in kernel API to adjust Q format of bias and output, respectively. Both `bias_shift` and `acc_shift` can be either positive or negative, where positive value denotes a left shift and negative value denotes a right shift.

`bias_shift` is the shift in number of bits applied to the bias to make it in the same Q format as weight X input multiplication – accumulation result. `acc_shift` is the shift in number of bits applied to the accumulator to obtain the output in desired Q format.

Note, the `acc_shift` and `bias_shift` arguments are not relevant in case of floating-point kernels and asymmetric 8-bit kernels.

For conversion from higher precision accumulator to lower precision output, symmetric rounding is used.

The precision of output is the same as precision of input vector.

The arguments `input_zero_bias`, `weight_zero_bias` are provided to convert the `asym8` inputs into their real values and perform Fully Connected kernel operation. The `out_zero_bias`, `out_multiplier` and `out_shift` values are used to quantize real values of output back to `asym8`.

Function variants available (for fixed point) are `xa_nn_fully_connected_[p]x[q]_[r]`, where:

- [p]: Weight matrix precision in bits
- [q]: Input vector precision in bits
- [r]: Output vector precision in bits

Precision

There are six variants available:

Type	Description
16x16_16	16-bit matrix inputs, 16-bit vector inputs, 16-bit output
8x16_16	8-bit matrix inputs, 16-bit vector inputs, 16-bit output
8x8_8	8-bit matrix inputs, 8-bit vector inputs, 8-bit output
f32	float32 matrix inputs, float32 vector inputs, float32 output
asym8xasym8_asym8	asym8 matrix inputs, asym8 vector inputs, asym8 output
sym8sxasym8s_asym8s	sym8s weight matrix, asym8s input vector, asym8s output

Algorithm

$$z_n = 2^{acc-shift} \left(\sum_{m=0}^{W_D-1} weight_{n,m} \cdot input_m + 2^{bias-shift} bias_n \right),$$

$n = 0, \dots, \overline{out-depth} - 1$

where W_D represents `weight_depth`

In case of floating-point and asym8 routines, `acc_shift=0` and `bias_shift=0`

Thus, $2^{acc-shift} = 2^{bias-shift} = 1$

Prototype

```
WORD32 xa_nn_fully_connected_16x16_16
(WORD16 * p_out,          WORD16 * p_weight,          WORD16 * p_inp,
 WORD16 * p_bias,         WORD32 weight_depth,         WORD32 out_depth,
 WORD32 acc_shift,        WORD32 bias_shift);
WORD32 xa_nn_fully_connected_8x16_16
(WORD16 * p_out,          WORD8 * p_weight,            WORD16 * p_inp,
 WORD16 * p_bias,         WORD32 weight_depth,         WORD32 out_depth,
 WORD32 acc_shift,        WORD32 bias_shift);
WORD32 xa_nn_fully_connected_8x8_8
(WORD8 * p_out,           WORD8 * p_weight,            WORD8 * p_inp,
 WORD8 * p_bias,          WORD32 weight_depth,         WORD32 out_depth,
 WORD32 acc_shift,        WORD32 bias_shift);
WORD32 xa_nn_fully_connected_f32
(FLOAT32 * p_out,         const FLOAT32 * p_weight,    const FLOAT32 * p_inp,
 const FLOAT32 * p_bias,   WORD32 weight_depth,        WORD32 out_depth);
WORD32 xa_nn_fully_connected_asym8xasym8_asym8
(UWORD8 * p_out,          const UWORD8 * p_weight,     const UWORD8 * p_inp,
 const WORD32 * p_bias,    WORD32 weight_depth,         WORD32 out_depth,
 WORD32 input_zero_bias,   WORD32 weight_zero_bias      WORD32 out_multiplier,
```

```

WORD32 out_shift,          WORD32 out_zero_bias);
WORD32 xa_nn_fully_connected_sym8sxasym8s_asym8s
(WORD8 * p_out,            const WORD8 * p_weight,    const WORD8 * p_inp,
 const WORD32 * p_bias, WORD32 weight_depth,        WORD32 out_depth,
 WORD32 input_zero_bias, WORD32 out_multiplier,      WORD32 out_shift,
 WORD32 out_zero_bias);

```

Arguments

Type	Name	Size	Description
Input			
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *	p_weight	out_depth* weight_depth	Weight matrix, fixed, floating point, asym8u or sym8s
WORD16 *, WORD8 *, const UWORD8 *, const FLOAT32 *	p_inp	weight_depth* 1	Input vector, fixed, floating point, asym8u or asym8s
WORD16 *, WORD8 *, const WORD32 *, const FLOAT32 *	p_bias	out_depth*1	Bias vector, fixed or floating point
WORD32	out_depth		Number of rows in weight matrix, bias and output vector
WORD32	weight_depth		Number of columns in weight matrix and rows in input vector
WORD32	acc_shift		Shift applied to accumulator
WORD32	bias_shift		Shift applied to bias
WORD32	input_zero_bias		Zero offset of input
WORD32	weight_zero_bias		Zero offset of weights
WORD32	out_multiplier		Multiplier value of output
WORD32	out_shift		Shift value of output
WORD32	out_zero_bias		Zero offset of output
Output			
WORD8 *, WORD16 *, UWORD8 *, FLOAT32 *	p_out	out_depth*1	Output vector, fixed, floating point, asym8u or asym8s

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
weight_depth	Multiple of 4 (1 in case of floating point and asym8)

p_weight, p_inp, p_bias, p_out	Aligned on 8-byte boundary (Aligned on (size of one element)-byte boundary for floating point and asym8) Should not overlap Cannot be NULL
out_depth	Greater than or equal to 1
acc_shift, bias_shift, out_shift	{-31, ...,31}
input_zero_bias	{-255, ...,0} for asym8u, {-127, ...,128} for asym8s
weight_zero_bias	{-255, ...,0}
out_multiplier	Greater than 0
out_zero_bias	{-255, ...,0} for asym8u, {-127, ...,128} for asym8s

3.5 Miscellaneous Kernels

3.5.1 Interpolation Kernel

Description

This kernel performs interpolation between two input vectors h and y using interpolation factor from vector x to get output vector z .

The interpolation kernel accepts 16-bit inputs and 16-bit interpolation factor in Q15 format and produces 16-bit output in Q15 format.

Precision

Type	Description
16-bit	16-bit input, 16-bit interpolation factor, 16-bit output

Algorithm

$$z_n = x_n * y_n + (1 - x_n) * h_n, \quad n = 0 \dots, \overline{num-elements} - 1$$

x_n represents interpolation factor.

y_n represents first input, h_n represents second input.

z_n represents output.

Prototype

```
WORD32 xa_nn_vec_interpolation_q15(
    (WORD16 * p_out,          WORD16 * p_ifact,      WORD16 * p_inp1,WORD16 * p_inp2,          WORD32
    num_elements);
```

Arguments

Type	Name	Size	Description
Input			
WORD16 *	p_ifact	num_elements	Interpolation factor vector
WORD16 *	p_inp1	num_elements	First input vector
WORD16 *	p_inp2	num_elements	Second input vector
WORD32	num_elements		Number of elements
Output			
WORD16 *	p_out	num_elements	Output vector

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_ifact, p_inp1, p_inp2, p_out	Aligned on 8-byte boundary
	Should not overlap
	Cannot be NULL
num_elements	Multiple of 4

3.5.2 L2 Normalization Kernel

Description

This kernel performs L2 normalization of an input vector x to get output vector z , which means every element of input vector x is divided by L2 norm of x , this gives an output vector z whose L2 norm is 1.

The L2 Normalization kernel accepts float32 input vector and produces float32 output vector.

Precision

Type	Description
f32	float32 input, float32 output

Algorithm

$$z_n = \frac{x_n}{\sqrt{\sum_{n=1}^N |x_n|^2}}, \quad n = 1 \dots, \overline{num-elements}$$

x_n represents input vector.

z_n represents output vector.

Prototype

```
WORD32 xa_nn_l2_norm_f32
(FLOAT32 * p_out,      const FLOAT32 * p_inp,      WORD32 num_elm);
```

Arguments

Type	Name	Size	Description
Input			
const FLOAT32 *	p_inp	num_elm	Input vector
WORD32	num_elm		Number of elements
Output			
WORD16 *	p_out	num_elm	Output vector

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions

Arguments	Restrictions
p_inp, p_out	Aligned on 4-byte boundary
	Should not overlap
	Cannot be NULL
num_elm	Greater than 0

3.5.3 Basic Kernels

Description

These kernels perform basic elementwise operations between two input vectors x and y to get output vector z . The supported operations are: add, subtract, multiply, multiply and accumulate, divide and floor. The supported precisions are: float32 and asym8.

Function variants available are `xa_nn_[o]_[p]_[q]`, where:

- [o]: Operations: `elm_add`, `elm_sub`, `elm_mul`, `elm_mul_acc`, `elm_div`, `elm_floor`
- [p]: Input Precision in bits- `input1xinput2`
- [q]: Output Precision in bits

Precision

Type	Description
------	-------------

f32xf32_f32	float32 input, float32 output
asym8xasym8_asym8	asym8 input, asym8 output

Algorithm

elm_add: $z_n = x_n + y_n$, $n = 0 \dots, \overline{num_elm - 1}$
 elm_sub: $z_n = x_n - y_n$, $n = 0 \dots, \overline{num_elm - 1}$
 elm_mul: $z_n = x_n * y_n$, $n = 0 \dots, \overline{num_elm - 1}$
 elm_mul_acc: $z_n = z_n + x_n * y_n$, $n = 0 \dots, \overline{num_elm - 1}$
 elm_div: $z_n = x_n / y_n$, $n = 0 \dots, \overline{num_elm - 1}$
 elm_floor: $z_n = \lfloor x_n \rfloor$, $n = 0 \dots, \overline{num_elm - 1}$

x_n represents first input, y_n represents second input.

z_n represents output.

Prototype

```

WORD32 xa_nn_elm_add_f32xf32_f32
(FLOAT32 * p_out,          const FLOAT32 * p_inpl,          const FLOAT32 * p_inp2,
 WORD32 num_elm);
WORD32 xa_nn_elm_sub_f32xf32_f32
(FLOAT32 * p_out,          const FLOAT32 * p_inpl,          const FLOAT32 * p_inp2,
 WORD32 num_elm);
WORD32 xa_nn_elm_mul_f32xf32_f32
(FLOAT32 * p_out,          const FLOAT32 * p_inpl,          const FLOAT32 * p_inp2,
 WORD32 num_elm);
WORD32 xa_nn_elm_mul_acc_f32xf32_f32
(FLOAT32 * p_out,          const FLOAT32 * p_inpl,          const FLOAT32 * p_inp2,
 WORD32 num_elm);
WORD32 xa_nn_elm_div_f32xf32_f32
(FLOAT32 * p_out,          const FLOAT32 * p_inpl,          const FLOAT32 * p_inp2,
 WORD32 num_elm);
WORD32 xa_nn_elm_floor_f32_f32
(FLOAT32 * p_out,          const FLOAT32 * p_inpl,          WORD32 num_elm);
WORD32 xa_nn_elm_add_asym8xasym8_asym8
(UWORD8 * p_out,          WORD32 out_zero_bias,          WORD32 out_shift,
 WORD32 out_multiplier,   WORD32 out_activation_min,      WORD32 out_activation_max,
 const UWORD8 * p_inpl,   WORD32 inpl_zero_bias,          WORD32 inpl_shift,
 WORD32 inpl_multiplier,  const UWORD8 * p_inp2,          WORD32 inpl_zero_bias,
 WORD32 inpl_shift,       WORD32 inpl_multiplier,          WORD32 left_shift,
 WORD32 num_elm);
WORD32 xa_nn_elm_mul_asym8xasym8_asym8
(UWORD8 * p_out,          WORD32 out_zero_bias,          WORD32 out_shift,
 WORD32 out_multiplier,   WORD32 out_activation_min,      WORD32 out_activation_max,
 const UWORD8 * p_inpl,   WORD32 inpl_zero_bias,          const UWORD8 * p_inp2,
 WORD32 inpl_zero_bias,   WORD32 num_elm);

```

Arguments

Type	Name	Size	Description
Input			
const UWORD8 *	p_inpl, p_inp	num_elm	First input vector

FLOAT32 *			
const UWORD8 * FLOAT32 *	p_inp2	num_elm	Second input vector
WORD32	num_elm		Number of elements
WORD32	out_zero_bias		Zero bias of output
WORD32	out_shift		Shift value of output
WORD32	out_multiplier		Multiplier value of output
WORD32	out_activation_min		Activation min of output
WORD32	out_activation_max		Activation max of output
WORD32	inp1_zero_bias		Zero bias of input 1
WORD32	inp1_shift		Shift value of input 1
WORD32	inp1_multiplier		Multiplier value of input 1
WORD32	inp2_zero_bias		Zero bias of input 2
WORD32	inp2_shift		Shift value of input 2
WORD32	inp2_multiplier		Multiplier value of input 2
Output			
UWORD8 * FLOAT32 *	p_out	num_elm	Output vector

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions:

Arguments	Restrictions
p_inp1, p_inp2, p_inp, p_out,	Aligned on (size of one element)-byte boundary
	Should not overlap
	Cannot be NULL
num_elm	Greater than 0

3.5.4 Elementwise Quantize Kernel

Description

These kernels perform the quantization operation of the `p_inp1` input vector elements to get the output vector `p_out`. The kernel is developed in reference to the Quantize operator implementation in Tensorflow Lite Micro.

Function variants available are `xa_nn_elm_quantize_[p]_[q]`, where:

- `[p]`: Input precision
- `[q]`: Output precision

Precision

Type	Description
asym16s_asym8s	asym16s input, asym8s output

Prototype

```
WORD32 xa_nn_elm_quantize_asym16s_asym8s
(WORD8 * p_out,          const WORD16 * p_inp,          WORD32 inp_zero_bias,
 WORD32 out_zero_bias,   WORD32 out_shift,              WORD32 out_multiplier,
 WORD32 num_elm);
```

Arguments

Type	Name	Size	Description
Input			
const WORD16 *	p_inp	num_elm	Input vector
WORD32	inp_zero_bias		Zero offset of input
WORD32	out_zero_bias		Zero offset of output
WORD32	out_shift		Shift value of output
WORD32	out_multiplier		Multiplier value of output
WORD32	num_elm		Number of input elements
Output			
WORD8 *	p_out	num_vecs	Output vector

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions:

Arguments	Restrictions
p_inp, p_out	Aligned on (size of one element)-byte boundary
	Cannot be NULL
num_elm	Greater than 0
out_shift	{-31, ..., 31}
out_multiplier	Greater than 0
inp_zero_bias	{-32768...,32767} for out type asym8s
out_zero_bias	{-128...,127} for out type asym8s

3.5.5 Memmove Kernel

Description

These kernels perform the copy of `n` characters from `psrc` input to the `pdst` output. It takes care of overlapping memory blocks as well.

Function variants available are `xa_nn_memmove_[p]`, where:

- `[p]`: Input and Output precision

Precision

Type	Description
16	16 bit input and 16 bit output pointers

Prototype

```
WORD32 xa_nn_memmove_16( void *pdst,
    const void *psrc,
    WORD32 n );
```

Arguments

Type	Name	Size	Description
Input			
<code>const void *</code>	<code>psrc</code>	<code>n</code>	Input vector
<code>WORD32</code>	<code>n</code>		number of char elements
Output			
<code>void *</code>	<code>pdst</code>	<code>n</code>	Output vector

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions:

Arguments	Restrictions
<code>psrc, pdst</code>	Aligned on 2-byte boundary
	Cannot be NULL
<code>num_elm</code>	Greater than 0

3.5.6 Dot Product Kernels

Description

These kernels perform the dot product operations between two sets of input vectors `p_inp1` and `p_inp2` to get output vector `p_out`. The supported precisions are: `f32xf32_f32` and `16x16_asym8s`.

Function variants available are `xa_nn_elm_quantize_[p]x[q]_[r]`, where:

- `[p], [q]`: Input precision
- `[r]`: Output precision

Precision

There are two variants available:

Type	Description
<code>f32xf32_f32</code>	float32 input, float32 output
<code>16x16_asym8s</code>	16-bit input, asym8s output

Prototype

```
WORD32 xa_nn_dot_prod_f32xf32_f32(FLOAT32 * __restrict__ p_out,
    const FLOAT32 * __restrict__ p_inp1, const FLOAT32 * __restrict__ p_inp2,
    WORD32 vec_length, WORD32 num_vecs);
WORD32 xa_nn_dot_prod_16x16_asym8s(WORD8 * __restrict__ p_out,
    const WORD16 * __restrict__ p_inp1_start,
    const WORD16 * __restrict__ p_inp2_start,
    const WORD32 * bias_ptr, WORD32 vec_length,
    WORD32 out_multiplier, WORD32 out_shift,
    WORD32 out_zero_bias, WORD32 vec_count);
```

Arguments

Type	Name	Size	Description
Input			
const FLOAT32 * const WORD16 *	<code>p_inp1</code>	<code>vec_length</code>	First input vector
const FLOAT32 * const WORD16 *	<code>p_inp2</code>	<code>vec_length</code>	Second input vector
const WORD32 *	<code>Bias_ptr</code>	<code>vec_count</code>	
WORD32	<code>vec_length</code>		Length of each vector
WORD32	<code>out_multiplier</code>		Multiplier value of output
WORD32	<code>out_shift</code>		Shift value of output
WORD32	<code>out_zero_bias</code>		Zero offset of output

WORD32	num_vecs, vec_count		number of vectors in each input
Output			
FLOAT32 * WORD8 *	p_out	num_vecs	Output vector

Returns

- 0: no error
- -1: error, invalid parameters

Restrictions:

Arguments	Restrictions
p_inp1, p_inp2, p_out	Aligned on (size of one element)-byte boundary Cannot be NULL
vec_length, num_vecs	Greater than 0
out_shift	{-31, ..., 31}
out_multiplier	Greater than 0
out_zero_bias	{-128..., 127} for out type asym8s

4. HiFi NN Library – Layers

This section explains the APIs of each layer implementation in the NN library. All the layers conform to the “generic NN Layer API” and flow explained in Section 2.

The NN library is a single archive containing all layers and low-level kernels implementations. Each layer has its own header file that defines the APIs specific to the layer. The following sections explain each layer in detail.

Note This version of the library supports GRU, LSTM, and CNN layers

4.1 GRU Layer

The GRU APIs are defined in `xa_nnlb_gru_api.h`. Refer to the overall signal flow diagram of GRU in [\[1\]](#).

4.1.1 GRU Layer Specification

GRU layer implements the following input-output equations [\[1\]](#):

$$\begin{aligned} z_t &= \text{sigmoid}(W_z * x_t + U_z * \text{prev-h} + b_z) \\ r_t &= \text{sigmoid}(W_r * x_t + U_r * \text{prev-h} + b_r) \\ g &= \tanh(W_h * x_t + U_h * (r_t \cdot \text{prev-h}) + b_h) \\ y_t = h_t &= z_t \cdot g + (1 - z_t) \cdot \text{prev-h} \\ \text{prev-h} &= h_t \end{aligned}$$

x_t : input vector

y_t, h_t : output vector

W, U : weight matrices

prev-h : previous output vector

z_t : update gate vector

r_t : reset gate vector

b : bias vectors

4.1.2 Error Codes Specific to GRU

Other than common error codes explained in Section 2.3, the GRU layer may also report the following error codes, which may be generated during the initialization stage.

- `XA_NNLIB_GRU_CONFIG_FATAL_INVALID_IN_FEATS3`
Number of input features is not supported
- `XA_NNLIB_GRU_CONFIG_FATAL_INVALID_OUT_FEATS`
Number of output features is not supported
- `XA_NNLIB_GRU_CONFIG_FATAL_INVALID_PRECISION`
I/O precision is not supported
- `XA_NNLIB_GRU_CONFIG_FATAL_INVALID_COEFF_QFORMAT`
Number of fractional bits for coefficients is not supported.
- `XA_NNLIB_GRU_CONFIG_FATAL_INVALID_IO_QFORMAT`
Number of fractional bits for input-output is not supported.
- `XA_NNLIB_GRU_CONFIG_FATAL_INVALID_MEMBANK_PADDING`
Membank padding should be 0 or 1.
- `XA_NNLIB_GRU_CONFIG_FATAL_INVALID_PARAM_ID`
Parameter identifier (param_id) is not valid

The following error codes may be generated during the execution stage.

- `XA_NNLIB_GRU_EXECUTE_FATAL_INSUFFICIENT_DATA`
Input data passed in is insufficient
- `XA_NNLIB_GRU_EXECUTE_FATAL_INSUFFICIENT_OUTPUT_BUFFER_SPACE`
Output buffer size is not sufficient.

³ FEATS := features

4.1.3 API Functions Specific to GRU

4.1.3.1 Query Functions

Table 4-1 GRU Get Persistent Size Function

Function	<code>xa_nnlib_gru_get_persistent_fast</code>
Syntax	<pre>Int32 xa_nnlib_gru_get_persistent_fast(xa_nnlib_gru_init_config_t *config)</pre>
Description	Returns persistent memory size in bytes required by GRU layer.
Parameters	Input: <code>config</code> Initial configuration parameters (see Table 4-7).
Errors	<p>If return value is less than 0, then it is an error. Following are the possible error codes:</p> <ul style="list-style-type: none"> ■ <code>XA_NNLIB_FATAL_MEM_ALLOC</code> ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_IN_FEATS</code> Number of input features is not supported ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_OUT_FEATS</code> Number of output features is not supported ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_PRECISION</code> I/O precision is not supported ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_COEFF_QFORMAT</code> Number of fractional bits for coefficients is not supported. ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_IO_QFORMAT</code> Number of fractional bits for input-output is not supported.

Table 4-2 GRU Get Scratch Size Function

Function	<code>xa_nnlib_gru_get_scratch_fast</code>
Syntax	<pre>Int32 xa_nnlib_gru_get_scratch_fast(xa_nnlib_gru_init_config_t *config)</pre>
Description	Returns scratch memory size in bytes required by GRU layer.
Parameters	Input: <code>config</code> Initial configuration parameters (see Table 4-7).
Errors	<p>If return value is less than 0, then it is an error. Following are the possible error codes:</p> <ul style="list-style-type: none"> ■ <code>XA_NNLIB_FATAL_MEM_ALLOC</code> ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_IN_FEATS</code> Number of input features is not supported ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_OUT_FEATS</code> Number of output features is not supported ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_PRECISION</code> I/O precision is not supported ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_COEFF_QFORMAT</code> Number of fractional bits for coefficients is not supported ■ <code>XA_NNLIB_GRU_CONFIG_FATAL_INVALID_IO_QFORMAT</code> Number of fractional bits for input-output is not supported

4.1.3.2 Initialization Stage

Table 4-3 GRU Init Function

Function	<code>xa_nnl-lib-gru-init</code>
Syntax	<pre> Int32 xa_nnl-lib-gru-init (xa_nnl-lib-handle_t handle, xa_nnl-lib-gru-init-config_t *config) </pre>
Description	Reset the GRU Layer API handle into its initial state. Set up the GRU Layer to the specified initial configuration parameters. This function sets <code>prev_h</code> vector to 0; the user can put the desired values in <code>prev_h</code> by using set config <code>XA_NNL-LIB-GRU-RESTORE-CONTEXT</code> (refer to Table 4-11 for more information).
Parameters	<p>Input: <code>handle</code> Pointer to the component persistent memory. This is the opaque handle. Required size: see <code>xa_nnl-lib-gru-get-persistent-fast</code>. Required alignment: 8 bytes.</p> <p>Input: <code>config</code> Initial configuration parameters (see Table 4-7). Note that the initial configuration parameters <i>must</i> be identical to those passed to query functions.</p>
Errors	<p>If the return value is not <code>XA_NNL-LIB-NO-ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA_NNL-LIB-FATAL-MEM-ALLOC</code> One of the pointers is invalid. ■ <code>XA_NNL-LIB-FATAL-MEM-ALIGN</code> One of the pointers is not properly aligned. ■ <code>XA_NNL-LIB-GRU-CONFIG-FATAL-INVALID-IN-FEATS</code> Number of input features is not supported ■ <code>XA_NNL-LIB-GRU-CONFIG-FATAL-INVALID-IN-FEATS</code> Number of output features is not supported ■ <code>XA_NNL-LIB-GRU-CONFIG-FATAL-INVALID-PRECISION</code> I/O precision is not supported. ■ <code>XA_NNL-LIB-GRU-CONFIG-FATAL-INVALID-COEFF-QFORMAT</code> Number of fractional bits for coefficients is not supported. ■ <code>XA_NNL-LIB-GRU-CONFIG-FATAL-INVALID-IO-QFORMAT</code> Number of fractional bits for input-output is not supported.

4.1.3.3 Execution Stage

Table 4-4 GRU Execution Function

Function	<code>xa_nnlib_gru_process</code>
Syntax	<pre> Int32 xa_nnlib_gru_process(xa_nnlib_handle_t handle, void *scratch, void *input, void *output, xa_nnlib_shape_t *p_in_shape, xa_nnlib_shape_t *p_out_shape) </pre>
Description	Processes one input shape to generate one output shape.
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>scratch</code> A pointer to the scratch buffer. Required alignment: 8 bytes.</p> <p>Input: <code>input</code> A pointer to the input buffer. Input buffer contains input data. Required alignment: 8 bytes.</p> <p>Output: <code>output</code> A pointer to the output buffer. Output is written to output buffer. Required alignment: 8 bytes.</p> <p>Input/Output: <code>p_in_shape</code> Pointer to the shape containing input buffer dimensions. Contains the length of input data passed to GRU layer. Required alignment: 4 bytes.</p> <p>Input/Output: <code>p_out_shape</code> Pointer to the shape for output buffer dimensions. On return, <code>*p_out_shape</code> is filled with the length of output generated by HiFi GRU Layer. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA_NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA_NNLIB_FATAL_MEM_ALLOC</code> One of the pointers is NULL. ■ <code>XA_NNLIB_FATAL_MEM_ALIGN</code> One of the pointers is not properly aligned.

	<ul style="list-style-type: none"> ■ XA_NNLIB_FATAL_INVALID_SHAPE Either input or output shape is invalid. ■ XA_NNLIB_GRU_EXECUTE_FATAL_INSUFFICIENT_DATA Input data passed in insufficient. ■ XA_NNLIB_GRU_EXECUTE_FATAL_INSUFFICIENT_OUTPUT_BUFFER_SPACE Output buffer size is not sufficient.
--	--

Table 4-5 GRU Set Parameter Function Details

Function	<code>xa_nnl-lib-gru-set-config</code>
Syntax	Int32 <pre>xa_nnl-lib-gru-set-config (xa_nnl-lib-handle_t handle, xa_nnl-lib-gru-param-id_t param_id, void *params)</pre>
Description	Sets the parameter specified by <code>param_id</code> to the value passed in the buffer pointed to by <code>params</code> .
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>param_id</code> Identifies the parameter to be written. Refer to Table 4-11 for the list of supported parameters.</p> <p>Input: <code>params</code> A pointer to a buffer that contains the parameter value. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA_NNLIB_NO_ERROR</code>, it implies that function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ XA_NNLIB_FATAL_MEM_ALLOC One of the pointers (<code>handle</code> or <code>params</code>) is NULL. ■ XA_NNLIB_FATAL_MEM_ALIGN One of the pointers (<code>handle</code> or <code>params</code>) is not aligned correctly. ■ XA_NNLIB_GRU_CONFIG_FATAL_INVALID_PARAM_ID Parameter identifier (<code>param_id</code>) is not valid.

Table 4-6 GRU Get Parameter Function Details

Function	<code>xa_nnlib_gru_get_config</code>
Syntax	<pre>Int32 xa_nnlib_gru_get_config (xa_nnlib_handle_t handle, xa_nnlib_gru_param_id_t param_id, void *params)</pre>
Description	Gets the value of the parameter specified by <code>param_id</code> in the buffer pointed to by <code>params</code> .
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>param_id</code> Identifies the parameter to be read. Refer to Table 4-11 for the list of supported parameters.</p> <p>Output: <code>params</code> A pointer to a buffer that is filled with the parameter value when the function returns. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA>NNLIB_NO_ERROR</code>, it implies that function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA>NNLIB_FATAL_MEM_ALLOC</code> One of the pointers (<code>handle</code> or <code>params</code>) is <code>NULL</code>. ■ <code>XA>NNLIB_FATAL_MEM_ALIGN</code> One of the pointers (<code>handle</code> or <code>params</code>) is not aligned correctly. ■ <code>XA>NNLIB_GRU_CONFIG_FATAL_INVALID_PARAM_ID</code> Parameter identifier (<code>param_id</code>) is not valid.

4.1.4 Structures Specific to GRU

Table 4-7 GRU Config Structure `xa_nnlb_gru_init_config_t`

Element Type	Element Name	Range	Default	Description
Int32	<code>in_feats</code>	4-2048	256	Number of input features (must be multiple of 4)
Int32	<code>out_feats</code>	4-2048	256	Number of output features (must be multiple of 4)
Int32	<code>pad</code>	0, 1	1	Padding 8 bytes for HiFi 4
Int32	<code>mat_prec</code>	8, 16	16	Matrix input precision
Int32	<code>vec_prec</code>	16	16	Vector input precision
<code>xa_nnlb_gru_precision_t</code>	<code>precision</code>	XA_NNLB_GRU_16b16b, XA_NNLB_GRU_8b16b	XA_NNLB_GRU_16b16b	Coef and I/O precision. Note: Current library supports only 16b16b and 8b16b precision for GRU
Int16	<code>coeff_Qformat</code>	0-15	15	Number of fractional bits for weights and biases
Int16	<code>io_Qformat</code>	0-15	12	Number of fractional bits for input and output

Table 4-8 `xa_nnlb_gru_weights_t` Parameter Type

Element Type	Element Name	Range	Default	Description
<code>coeff_t *</code>	<code>w_z</code>	NA	NA	Pointer to coefficient matrix <code>w_z</code> .
<code>xa_nnlb_shape_t</code>	<code>shape_w_z</code>	NA	NA	Shape information about <code>w_z</code> .
<code>coeff_t *</code>	<code>u_z</code>	NA	NA	Pointer to coefficient matrix <code>u_z</code> .
<code>xa_nnlb_shape_t</code>	<code>shape_u_z</code>	NA	NA	Shape information about <code>u_z</code> .
<code>coeff_t *</code>	<code>w_r</code>	NA	NA	Pointer to coefficient matrix <code>w_r</code> .
<code>xa_nnlb_shape_t</code>	<code>shape_w_r</code>	NA	NA	Shape information about <code>w_r</code> .
<code>coeff_t *</code>	<code>u_r</code>	NA	NA	Pointer to coefficient matrix <code>u_r</code> .
<code>xa_nnlb_shape_t</code>	<code>shape_u_r</code>	NA	NA	Shape information about <code>u_r</code> .
<code>coeff_t *</code>	<code>w_h</code>	NA	NA	Pointer to coefficient matrix <code>w_h</code> .
<code>xa_nnlb_shape_t</code>	<code>shape_w_h</code>	NA	NA	Shape information about <code>w_h</code> .
<code>coeff_t *</code>	<code>u_h</code>	NA	NA	Pointer to coefficient matrix <code>u_h</code> .
<code>xa_nnlb_shape_t</code>	<code>shape_u_h</code>	NA	NA	Shape information about <code>u_h</code> .

Table 4-9 xa_nnlib_gru_biases_t Parameter Type

Element Type	Element Name	Range	Default	Description
coeff_t *	b_z	NA	NA	Pointer to coefficient matrix b_z.
xa_nnlib_shape_t	shape_b_z	NA	NA	Shape information about b_z.
coeff_t *	b_r	NA	NA	Pointer to coefficient matrix b_r.
xa_nnlib_shape_t	shape_b_r	NA	NA	Shape information about b_r.
coeff_t *	b_h	NA	NA	Pointer to coefficient matrix b_h.
xa_nnlib_shape_t	shape_b_h	NA	NA	Shape information about b_h.

Note GRU requires all weight matrices' and bias vectors' pointers to be 8 bytes aligned.

4.1.5 Enums Specific to GRU

Table 4-10 Enum xa_nnlib_gru_precision_t

Element	Description
XA_NNLIB_GRU_16bx16b	Coef: 16 bits, I/O: 16 bits Fixed Point
XA_NNLIB_GRU_8bx16b	Coef: 8 bits, I/O: 16 bits Fixed Point
XA_NNLIB_GRU_8bx8b	Not supported
XA_NNLIB_flt16xflt16	Not supported

Note Currently, GRU only supports XA_NNLIB_GRU_16bx16b, XA_NNLIB_GRU_8bx16b precision setting.

Table 4-11 describes parameter IDs for parameters supported by GRU. It contains the following columns:

- Parameter ID: Parameter identifier (`param_id`).
- Value type: A pointer (`params`) to a variable of this type is to be passed.
- RW: Indicates whether the parameter can be read (`get`) and/or written (`set`).
- Range: Indicates valid values of the parameter.
- Default: Default value of the parameter
- Description: Brief description of the parameter.

Table 4-11 GRU Specific Parameters

Parameter ID	Value Type	RW	Range	Default	Description
XA_NNLIB_GRU_RESTORE_CONTEXT	vect_t []	RW	NA	NA	Set previous output. This can be used to set prev_h to specific context (size should be equal to number of output features). Upon set config, the buffer passed is copied to persistent memory; upon get config, it returns the prev_h state in the given buffer.
XA_NNLIB_GRU_WEIGHT	xa_nnlib_gru_weights_t	RW	NA	NA	Weight matrices, pointers to weight matrices along with shape information must be passed via xa_nnlib_gru_weights_t structure for set config. Upon get config, it returns pointers to weight matrices along with their shape information in same structure.
XA_NNLIB_GRU_BIAS	xa_nnlib_gru_biases_t	RW	NA	NA	Bias vectors, pointers to bias vectors along with shape information must be passed via xa_nnlib_gru_biases_t structure for set config. Upon get config, it returns pointers to bias vectors along with their shape information in same structure.
XA_NNLIB_GRU_INPUT_SHAPE	xa_nnlib_shape_t	R	NA	NA	Input shape information, get information of the input shape expected by the layer.
XA_NNLIB_GRU_OUTPUT_SHAPE	xa_nnlib_shape_t	R	NA	NA	Output shape information, get information of the output shape expected by layer.

4.2 LSTM Layer

The LSTM APIs are defined in `xa_nnlib_lstm_api.h`.

4.2.1 LSTM Layer Specification

The LSTM layer implements the following forward path input-output equations:

$$\begin{aligned} f_f &= \text{sigmoid}(w_{xf} * \text{frame}_f + \text{prev-h} * w_{hf} + b_f) \\ i_f &= \text{sigmoid}(w_{xi} * \text{frame}_f + \text{prev-h} * w_{hi} + b_i) \\ c\text{-hat}_f &= \tanh(w_{xc} * \text{frame}_f + \text{prev-h} * w_{hc} + b_c) \\ c_f &= f_f * \text{prev-c} + i_f * c\text{-hat}_f \\ o_f &= \text{sigmoid}(w_{xo} * \text{frame}_f + \text{prev-h} * w_{ho} + b_o) \\ h_f &= o_f * \tanh(c_f) \end{aligned}$$

i_f : input gate

h_t : output vector

$c\text{-hat}_f$: intermediate cell state vector

f_f : forget gate

frame_f : Input vector

w_x : weight matrices of input connections

prev-h : previous output vector

prev-c : previous cell output

b : bias vectors

o_f : output gate

c_f : cell state vector

w_h : weight matrices of recurrent connections

4.2.2 Error Codes Specific to LSTM

Other than common error codes explained in Section 2.3, the LSTM layer may also report the following error codes, which may be generated during the initialization stage:

- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_IN_FEATS4`
Number of input features is not supported
- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_OUT_FEATS`
Number of output features is not supported
- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_PRECISION`
I/O precision is not supported
- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_COEFF_QFORMAT`
Number of fractional bits for coefficients is not supported.
- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_CELL_QFORMAT`
Number of fractional bits for cells is not supported

⁴ FEATS: = features

- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_IO_QFORMAT`
Number of fractional bits for input-output is not supported.
- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_MEMBANK_PADDING`
Membank padding should be 0 or 1.
- `XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_PARAM_ID`
Parameter identifier (param_id) is not valid

The following error codes may be generated during the execution stage.

- `XA_NNLIB_LSTM_EXECUTE_FATAL_INSUFFICIENT_DATA`
Input data passed in insufficient
- `XA_NNLIB_LSTM_EXECUTE_FATAL_INSUFFICIENT_OUTPUT_BUFFER_SPACE`
Output Buffer Size is not sufficient

4.2.3 API Functions Specific to LSTM

4.2.3.1 Query Functions

Table 4-12 LSTM Get Persistent Size Function

Function	<code>xa_nnlstm_get_persistent_fast</code>
Syntax	<pre>Int32 xa_nnlstm_get_persistent_fast (xa_nnlstm_init_config_t *config)</pre>
Description	Returns persistent memory size in bytes required by LSTM layer.
Parameters	Input: <code>config</code> Initial configuration parameters (see Table 4-18).
Errors	<p>If return value is less than 0 then it is an error. Following are the possible error codes:</p> <ul style="list-style-type: none"> ■ <code>XA_NNLSTM_FATAL_MEM_ALLOC</code> ■ <code>XA_NNLSTM_CONFIG_FATAL_INVALID_IN_FEATS</code> Number of input features is not supported ■ <code>XA_NNLSTM_CONFIG_FATAL_INVALID_OUT_FEATS</code> Number of output features is not supported ■ <code>XA_NNLSTM_CONFIG_FATAL_INVALID_PRECISION</code> I/O precision is not supported ■ <code>XA_NNLSTM_CONFIG_FATAL_INVALID_COEFF_QFORMAT</code> Number of fractional bits for coefficients is not supported. ■ <code>XA_NNLSTM_CONFIG_FATAL_INVALID_CELL_QFORMAT</code> Number of fractional bits for cells is not supported ■ <code>XA_NNLSTM_CONFIG_FATAL_INVALID_IO_QFORMAT</code> Number of fractional bits for input-output is not supported. ■ <code>XA_NNLSTM_CONFIG_FATAL_INVALID_MEMBANK_PADDING</code> Membank padding should be 0 or 1.

Table 4-13 LSTM Get Scratch Size Function

Function	<code>xa_nnlib_lstm_get_scratch_fast</code>
Syntax	<code>Int32 xa_nnlib_lstm_get_scratch_fast (</code> <code>xa_nnlib_lstm_init_config_t *config)</code>
Description	Returns scratch memory size in bytes required by LSTM layer.
Parameters	Input: <code>config</code> Initial configuration parameters (see Table 4-18).
Errors	<p>If return value is less than 0 then it is an error, the possible error codes are:</p> <ul style="list-style-type: none"> ■ <code>XA_NNLIB_FATAL_MEM_ALLOC</code> ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_IN_FEATS</code> Number of input features is not supported ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_OUT_FEATS</code> Number of output features is not supported ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_PRECISION</code> I/O precision is not supported ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_COEFF_QFORMAT</code> Number of fractional bits for coefficients is not supported. ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_CELL_QFORMAT</code> Number of fractional bits for cells is not supported ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_IO_QFORMAT</code> Number of fractional bits for input-output is not supported. ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_MEMBANK_PADDING</code> Membank padding should be 0 or 1.

4.2.3.2 Initialization Stage

Table 4-14 LSTM Init Function

Function	<code>xa_nnlib_lstm_init</code>
Syntax	<pre>Int32 xa_nnlib_lstm_init (xa_nnlib_handle_t handle, xa_nnlib_lstm_init_config_t *config)</pre>
Description	<p>Reset the LSTM layer API handle into its initial state. Set up the LSTM layer to the specified initial configuration parameters. This function sets <code>prev_h</code> vector and <code>prev_c</code> vector to 0; the user can put the desired values in <code>prev_h</code> and <code>prev_c</code> by using set config <code>XA_NNLIB_LSTM_RESTORE_CONTEXT_OUTPUT</code> and <code>XA_NNLIB_LSTM_RESTORE_CONTEXT_CELL</code> respectively (refer to Table 4-22 for more information).</p>
Parameters	<p>Input: <code>handle</code> Pointer to the component persistent memory. This is the opaque handle. Required size: see <code>xa_nnlib_lstm_get_persistent_fast</code>. Required alignment: 8 bytes.</p> <p>Input: <code>config</code> Initial configuration parameters (see Table 4-18). Note that the initial configuration parameters MUST be identical to those passed to query functions.</p>
Errors	<p>If the return value is not <code>XA_NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA_NNLIB_FATAL_MEM_ALLOC</code> One of the pointers is invalid. ■ <code>XA_NNLIB_FATAL_MEM_ALIGN</code> One of the pointers is not properly aligned. ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_IN_FEATS</code> Number of input features is not supported ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_OUT_FEATS</code> Number of output features is not supported ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_PRECISION</code> I/O precision is not supported ■ <code>XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_COEFF_QFORMAT</code> Number of fractional bits for coefficients is not supported.

- **XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_CELL_QFORMAT**
Number of fractional bits for cells is not supported
- **XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_IO_QFORMAT**
Number of fractional bits for input-output is not supported
- **XA_NNLIB_LSTM_CONFIG_FATAL_INVALID_MEMBANK_PADDING**
Membank padding should be 0 or 1.

4.2.3.3 Execution Stage

Table 4-15 LSTM Execution Function

Function	<code>xa_nnl-lib_lstm_process</code>
Syntax	<pre> Int32 xa_nnl-lib_lstm_process (xa_nnl-lib_handle_t handle, void *scratch, void *input, void *output, xa_nnl-lib_shape_t *p_in_shape, xa_nnl-lib_shape_t *p_out_shape) </pre>
Description	Processes one input shape to generate one output shape.
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>scratch</code> A pointer to the scratch buffer. Required alignment: 8 bytes.</p> <p>Input: <code>input</code> A pointer to the input buffer. Input buffer contains input data. Required alignment: 8 bytes.</p> <p>Output: <code>output</code> A pointer to the output buffer. Output is written to the output buffer. Required alignment: 8 bytes.</p> <p>Input/Output: <code>p_in_shape</code> Pointer to the shape containing input buffer dimensions. Contains the length of input data passed to LSTM layer. Required alignment: 4 bytes.</p> <p>Input/Output: <code>p_out_shape</code></p>

	<p>Pointer to the shape for output buffer dimensions. On return, *p_out_shape is filled with the length of output generated by HiFi LSTM layer.</p> <p>Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not XA_NNLIB_NO_ERROR, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ XA_NNLIB_FATAL_MEM_ALLOC One of the pointers is NULL. ■ XA_NNLIB_FATAL_MEM_ALIGN One of the pointers is not having proper alignment. ■ XA_NNLIB_FATAL_INVALID_SHAPE Either input or output shape is invalid. ■ XA_NNLIB_LSTM_EXECUTE_FATAL_INSUFFICIENT_DATA Input data passed in insufficient ■ XA_NNLIB_LSTM_EXECUTE_FATAL_INSUFFICIENT_OUTPUT_BUFFER_SPACE Output Buffer Size is not sufficient

Table 4-16 LSTM Set Parameter Function Details

Function	<code>xa_nnlib_lstm_set_config</code>
Syntax	<pre>Int32 xa_nnlib_lstm_set_config (xa_nnlib_handle_t handle, xa_nnlib_lstm_param_id_t param_id, void *params)</pre>
Description	Sets the parameter specified by <code>param_id</code> to the value passed in the buffer pointed to by <code>params</code> .
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>param_id</code> Identifies the parameter to be written. Refer to Table 4-11 for the list of supported parameters.</p> <p>Input: <code>params</code> A pointer to a buffer that contains the parameter value. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA>NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA>NNLIB_FATAL_MEM_ALLOC</code> One of the pointers (<code>handle</code> or <code>params</code>) is NULL. ■ <code>XA>NNLIB_FATAL_MEM_ALIGN</code> One of the pointers (<code>handle</code> or <code>params</code>) is not aligned correctly. ■ <code>XA>NNLIB_LSTM_CONFIG_FATAL_INVALID_PARAM_ID</code> Parameter identifier (<code>param_id</code>) is not valid.

Table 4-17 LSTM Get Parameter Function Details

Function	<code>xa_nnlib_lstm_get_config</code>
Syntax	<pre>Int32 xa_nnlib_lstm_get_config (xa_nnlib_handle_t handle, xa_nnlib_lstm_param_id_t param_id, void *params)</pre>
Description	Gets the value of the parameter specified by <code>param_id</code> in the buffer pointed to by <code>params</code> .
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>param_id</code> Identifies the parameter to be read. Refer to Table 4-11 for the list of supported parameters.</p> <p>Output: <code>params</code> A pointer to a buffer that is filled with the parameter value when the function returns. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA>NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA>NNLIB_FATAL_MEM_ALLOC</code> One of the pointers (<code>handle</code> or <code>params</code>) is <code>NULL</code>. ■ <code>XA>NNLIB_FATAL_MEM_ALIGN</code> One of the pointers (<code>handle</code> or <code>params</code>) is not aligned correctly. ■ <code>XA>NNLIB_LSTM_CONFIG_FATAL_INVALID_PARAM_ID</code> Parameter identifier (<code>param_id</code>) is not valid.

4.2.4 Structures Specific to LSTM

Table 4-18 LSTM Config Structure `xa_nnlib_lstm_init_config_t`

Element Type	Element Name	Range	Default	Description
Int32	<code>in_feats</code>	4-2048	256	Number of input features (must be multiple of 4)
Int32	<code>out_feats</code>	4-2048	256	Number of output features (must be multiple of 4)
Int32	<code>pad</code>	0, 1	1	Padding 8 bytes for HiFi 4 DSP
Int32	<code>mat_prec</code>	8, 16	16	Matrix input precision
Int32	<code>vec_prec</code>	16	16	Vector input precision
<code>xa_nnlib_lstm_precision_t</code>	<code>precision</code>	XA_NNLIB_LSTM_16bx16b, XA_NNLIB_LSTM_8bx16b	XA_NNLIB_LSTM_16bx16b	Coef and I/O precision. Note: The current library supports only 16bx16b and 8bx16b precision for LSTM.
Int16	<code>coeff_Qformat</code>	0-15	15	Number of fractional bits for weights and biases
Int16	<code>cell_Qformat</code>	0-26		Number of fractional bits for cells.
Int16	<code>io_Qformat</code>	0-15	12	Number of fractional bits for input and output

Table 4-19 `xa_nnlib_lstm_weights_t` Parameter Type

Element Type	Element Name	Range	Default	Description
<code>coeff_t *</code>	<code>w_xf</code>	NA	NA	Pointer to coefficient matrix <code>w_xf</code> .
<code>xa_nnlib_shape_t</code>	<code>shape_w_xf</code>	NA	NA	Shape information about <code>w_xf</code> .
<code>coeff_t *</code>	<code>w_xi</code>	NA	NA	Pointer to coefficient matrix <code>w_xi</code> .
<code>xa_nnlib_shape_t</code>	<code>shape_w_xi</code>	NA	NA	Shape information about <code>w_xi</code> .
<code>coeff_t *</code>	<code>w_xc</code>	NA	NA	Pointer to coefficient matrix <code>w_xc</code> .
<code>xa_nnlib_shape_t</code>	<code>shape_w_xc</code>	NA	NA	Shape information about <code>w_xc</code> .
<code>coeff_t *</code>	<code>w_xo</code>	NA	NA	Pointer to coefficient matrix <code>w_xo</code> .
<code>xa_nnlib_shape_t</code>	<code>shape_w_xo</code>	NA	NA	Shape information about <code>w_xo</code> .
<code>coeff_t *</code>	<code>w_hf</code>	NA	NA	Pointer to coefficient matrix <code>w_hf</code> .
<code>xa_nnlib_shape_t</code>	<code>shape_w_hf</code>	NA	NA	Shape information about <code>w_hf</code> .
<code>coeff_t *</code>	<code>w_hi</code>	NA	NA	Pointer to coefficient matrix <code>w_hi</code> .

Element Type	Element Name	Range	Default	Description
xa_nnlb_shape_t	shape_w_hi	NA	NA	Shape information about w_hi.
coeff_t *	w_hc	NA	NA	Pointer to coefficient matrix w_hc.
xa_nnlb_shape_t	shape_w_hc	NA	NA	Shape information about w_hc.
coeff_t *	w_ho	NA	NA	Pointer to coefficient matrix w_ho.
xa_nnlb_shape_t	shape_w_ho	NA	NA	Shape information about w_ho.

Table 4-20 xa_nnlb_lstm_biases_t Parameter Type

Element Type	Element Name	Range	Default	Description
coeff_t *	b_f	NA	NA	Pointer to coefficient matrix b_f.
xa_nnlb_shape_t	shape_b_f	NA	NA	Shape information about b_f.
coeff_t *	b_i	NA	NA	Pointer to coefficient matrix b_i.
xa_nnlb_shape_t	shape_b_i	NA	NA	Shape information about b_i.
coeff_t *	b_c	NA	NA	Pointer to coefficient matrix b_c.
xa_nnlb_shape_t	shape_b_c	NA	NA	Shape information about b_c.
coeff_t *	b_o	NA	NA	Pointer to coefficient matrix b_o.
xa_nnlb_shape_t	shape_b_o	NA	NA	Shape information about b_o.

Note LSTM requires all weight matrices' and bias vectors' pointers to be 8 bytes aligned.

4.2.5 Enums Specific to LSTM

Table 4-21 Enum xa_nnlb_lstm_precision_t

Element	Description
XA_NNLB_LSTM_16bx16b	Coef: 16 bits, I/O: 16 bits Fixed Point
XA_NNLB_LSTM_8bx16b	Coef: 8 bits, I/O: 16 bits Fixed Point
XA_NNLB_LSTM_8bx8b	Not supported
XA_NNLB_flt16xflt16	Not supported

Note Currently, LSTM only supports the XA_NNLB_LSTM_16bx16b, XA_NNLB_LSTM_8bx16b precision setting.

Table 4-22 describes parameter IDs for parameters supported by LSTM. It contains the following columns:

- Parameter ID: Parameter identifier (`param_id`).
- Value type: A pointer (`params`) to a variable of this type is to be passed.
- RW: Indicates whether the parameter can be read (`get`) and/or written (`set`).
- Range: Indicates valid values of the parameter.
- Default: Default value of the parameter.
- Description: Brief description of the parameter.

Table 4-22 LSTM Specific Parameters

Parameter ID	Value Type	RW	Range	Default	Description
XA_NNLIB_LSTM_RESTORE_CONTEXT_OUTPUT	<code>vect_t []</code>	RW	NA	NA	Set previous output. This can be used to set <code>prev_h</code> to specific context (size should be equal to number of output features). Upon set config, the buffer passed is copied to persistent memory; upon get config, it returns the <code>prev_h</code> state in the given buffer.
XA_NNLIB_LSTM_RESTORE_CONTEXT_CELL	<code>vect_t []</code>	RW	NA	NA	Set previous cell state. This can be used to set <code>prev_c</code> to specific cell context (size should be equal to number of output features). Upon set config, the buffer passed is copied to persistent memory; upon get config, it returns the <code>prev_c</code> state in the given buffer.
XA_NNLIB_LSTM_WEIGHT	<code>xa_nnl-lib_lstm_weights_t</code>	RW	NA	NA	Weight matrices, pointers to weight matrices along with shape information needs to be passed via <code>xa_nnl-lib_lstm_weights_t</code> structure for set config. Upon get config, it returns pointers to weight matrices along with their shape information in same structure.
XA_NNLIB_LSTM_BIAS	<code>xa_nnl-lib_lstm_biases_t</code>	RW	NA	NA	Bias vectors, pointers to bias vectors along with shape information needs to be passed via <code>xa_nnl-lib_lstm_biases_t</code> structure for set config. Upon get config, it returns pointers to bias vectors along with their shape information in same structure.
XA_NNLIB_LSTM_INPUT_SHAPE	<code>xa_nnl-lib_shape_t</code>	R	NA	NA	Input shape information, get information of the input shape expected by the layer.
A_NNLIB_LSTM_OUTPUT_SHAPE	<code>xa_nnl-lib_shape_t</code>	R	NA	NA	Output shape information, get information of the output shape expected by layer.

4.3 CNN Layer

The CNN APIs are defined in `xa_nnlb_cnn_api.h`.

4.3.1 CNN Layer Specification

The CNN layer implements Standard 2D Convolution, Standard 1D Convolution, and Depthwise Separable 2D Convolution. Refer to the equations in Section 3.2.1 for Standard 2D Convolution, Section 3.2.2 for Standard 1D Convolution, and Section 3.2.3 for Depthwise Separable 2D Convolution.

4.3.2 Error Codes Specific to CNN

Other than common error codes explained in Section 2.3, the CNN layer may also report the following error codes, which may be generated during the initialization stage.

- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_ALGO`
Algorithm is not supported
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_PRECISION`
I/O precision is not supported.
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_BIAS_SHIFT`
Value of Bias shift is not supported
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_ACC_SHIFT`
Value of Accumulator shift is not supported.
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_STRIDE`
Value of strides is not supported
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_PADDING`
Value of padding is not supported.
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_INPUT_SHAPE`
Input shape dimension is not supported.
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_OUTPUT_SHAPE`
Out shape dimension is not supported.
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_KERNEL_SHAPE`
Kernel shape dimension is not supported.
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_BIAS_SHAPE`
Bias shape dimension is not supported.
- `XA_NNLB_CNN_CONFIG_FATAL_INVALID_PARAM_ID`
Parameter identifier (`param_id`) is not valid

- **XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_COMBINATION**

Parameter combination (param_id) is not valid

The following error codes may be generated during the execution stage.

- **XA_NNLIB_CNN_CONFIG_FATAL_INVALID_INPUT_SHAPE**

Input shape passed during execution does not match with the input shape passed during initialization

4.3.3 API Functions Specific to CNN

4.3.3.1 Query Functions

Table 4-23 CNN Get Persistent Size Function

Function	<code>xa_nnlb_cnn_get_persistent_fast</code>
Syntax	<pre>Int32 xa_nnlb_cnn_get_persistent_fast (xa_nnlb_cnn_init_config_t *config)</pre>
Description	Returns persistent memory size in bytes required by CNN layer.
Parameters	Input: <code>config</code> Initial configuration parameters (see Table 4-29).
Errors	If return value is less than 0, then it is an error. Following are the possible error codes: <ul style="list-style-type: none"> ■ XA_NNLIB_FATAL_MEM_ALLOC ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_ALGO Algorithm is not supported ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PRECISION I/O precision is not supported. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_BIAS_SHIFT Value of Bias shift is not supported ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_ACC_SHIFT Value of Accumulator shift is not supported. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_STRIDE Value of strides is not supported ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PADDING Value of padding is not supported.

	<ul style="list-style-type: none"> ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_INPUT_SHAPE Input shape dimension is not supported. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_OUTPUT_SHAPE Out shape dimension is not supported. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_KERNEL_SHAPE Kernel shape dimension is not supported. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_BIAS_SHAPE Bias shape dimension is not supported ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_ID Parameter identifier (param_id) is not valid ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_COMBINATION Parameter combination (param_id) is not valid
--	--

Table 4-24 CNN Get Scratch Size Function

Function	<code>xa_nnlb_cnn_get_scratch_fast</code>
Syntax	<pre>Int32 xa_nnlb_cnn_get_scratch_fast (xa_nnlb_cnn_init_config_t *config)</pre>
Description	Returns scratch memory size in bytes required by CNN layer.
Parameters	Input: <code>config</code> Initial configuration parameters (see Table 4-29).
Errors	If return value is less than 0, then it is an error. Following are the possible error codes: <ul style="list-style-type: none"> ■ XA_NNLIB_FATAL_MEM_ALLOC ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_ALGO Algorithm is not supported ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PRECISION I/O precision is not supported. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_BIAS_SHIFT Value of bias shift is not supported ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_ACC_SHIFT Value of Accumulator shift is not supported. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_STRIDE Value of strides is not supported

- `XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PADDING`
Value of padding is not supported.
- `XA_NNLIB_CNN_CONFIG_FATAL_INVALID_INPUT_SHAPE`
Input shape dimension is not supported.
- `XA_NNLIB_CNN_CONFIG_FATAL_INVALID_OUTPUT_SHAPE`
Out shape dimension is not supported.
- `XA_NNLIB_CNN_CONFIG_FATAL_INVALID_KERNEL_SHAPE`
Kernel shape dimension is not supported.
- `XA_NNLIB_CNN_CONFIG_FATAL_INVALID_BIAS_SHAPE`
Bias shape dimension is not supported.
- `XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_ID`
Parameter identifier (param_id) is not valid
- `XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_COMBINATION`

Parameter combination (param_id) is not valid

4.3.3.2 Initialization Stage

Table 4-25 CNN Init Function

Function	<code>xa_nnlib_cnn_init</code>
Syntax	<pre>int xa_nnlib_cnn_init (xa_nnlib_handle_t handle, xa_nnlib_cnn_init_config_t *config)</pre>
Description	Reset the CNN layer API handle into its initial state. Set up the CNN layer to the specified initial configuration parameters.
Parameters	<p>Input: <code>handle</code> Pointer to the component persistent memory. This is the opaque handle. Required size: see <code>xa_nnlib_cnn_get_persistent_fast</code>. Required alignment: 8 bytes.</p> <p>Input: <code>config</code> Initial configuration parameters (see Table 4-29). Note that the initial configuration parameters <i>must</i> be identical to those passed to query functions.</p>
Errors	<p>If the return value is not <code>XA>NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA>NNLIB_FATAL_MEM_ALLOC</code> One of the pointers is invalid. ■ <code>XA>NNLIB_FATAL_MEM_ALIGN</code> One of the pointers is not properly aligned. ■ <code>XA>NNLIB_CNN_CONFIG_FATAL_INVALID_ALGO</code> Algorithm is not supported. ■ <code>XA>NNLIB_CNN_CONFIG_FATAL_INVALID_PRECISION</code> I/O precision is not supported. ■ <code>XA>NNLIB_CNN_CONFIG_FATAL_INVALID_BIAS_SHIFT</code> Value of Bias shift is not supported. ■ <code>XA>NNLIB_CNN_CONFIG_FATAL_INVALID_ACC_SHIFT</code> Value of Accumulator shift is not supported. ■ <code>XA>NNLIB_CNN_CONFIG_FATAL_INVALID_STRIDE</code> Value of strides is not supported. ■ <code>XA>NNLIB_CNN_CONFIG_FATAL_INVALID_PADDING</code> Value of padding is not supported.

- | | |
|--|--|
| | <ul style="list-style-type: none">■ <code>XA_NNLIB_CNN_CONFIG_FATAL_INVALID_INPUT_SHAPE</code>
Input shape dimension is not supported.■ <code>XA_NNLIB_CNN_CONFIG_FATAL_INVALID_OUTPUT_SHAPE</code>
Out shape dimension is not supported.■ <code>XA_NNLIB_CNN_CONFIG_FATAL_INVALID_KERNEL_SHAPE</code>
Kernel shape dimension is not supported.■ <code>XA_NNLIB_CNN_CONFIG_FATAL_INVALID_BIAS_SHAPE</code>
Bias shape dimension is not supported.■ <code>XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_ID</code>
Parameter identifier (<code>param_id</code>) is not valid.■ <code>XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_COMBINATION</code>
Parameter combination (<code>param_id</code>) is not valid. |
|--|--|

4.3.3.3 Execution Stage

Table 4-26 CNN Execution Function

Function	<code>xa_nnlib_cnn_process</code>
Syntax	<pre>int xa_nnlib_cnn_process (xa_nnlib_handle_t handle, void *scratch, void *input, void *output, xa_nnlib_shape_t *p_in_shape, xa_nnlib_shape_t *p_out_shape)</pre>
Description	Processes one input shape to generate one output shape.
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>scratch</code> A pointer to the scratch buffer. Required alignment: 8 bytes.</p> <p>Input: <code>input</code> A pointer to the input buffer. Input buffer contains input data. Required alignment: 8 bytes.</p> <p>Output: <code>output</code> A pointer to the output buffer. Output is written to the output buffer. Required alignment: 8 bytes.</p> <p>Input/Output: <code>p_in_shape</code> Pointer to the shape containing input buffer dimensions. Contains the length of input data passed to the CNN layer. Required alignment: 4 bytes.</p> <p>Output: <code>p_out_shape</code> Pointer to the shape for output buffer dimensions. Upon return, <code>*p_out_shape</code> is filled with the length of output generated by the CNN layer. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA_NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA_NNLIB_FATAL_MEM_ALLOC</code> One of the pointers is NULL

	<ul style="list-style-type: none"> ■ XA_NNLIB_FATAL_MEM_ALIGN One of the pointers is not having required alignment ■ XA_NNLIB_FATAL_INVALID_SHAPE Input shape passed during execution does not match with the input shape passed during initialization
--	--

Table 4-27 CNN Set Parameter Function Details

Function	<code>xa_nnl-lib_cnn_set_config</code>
Syntax	<pre>int xa_nnl-lib_cnn_set_config (xa_nnl-lib_handle_t handle, xa_nnl-lib_cnn_param_id_t param_id, void *params)</pre>
Description	Sets the parameter specified by <code>param_id</code> to the value passed in the buffer pointed to by <code>params</code> .
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>param_id</code> Identifies the parameter to be written. Refer to Table 4-32 for the list of supported parameters.</p> <p>Input: <code>params</code> A pointer to a buffer that contains the parameter value. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA_NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ XA_NNLIB_FATAL_MEM_ALLOC One of the pointers (<code>handle</code> or <code>params</code>) is <code>NULL</code>. ■ XA_NNLIB_FATAL_MEM_ALIGN One of the pointers (<code>handle</code> or <code>params</code>) is not aligned correctly. ■ XA_NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_ID Parameter identifier (<code>param_id</code>) is not valid.

Table 4-28 CNN Get Parameter Function Details

Function	<code>xa_nnlib_cnn_get_config</code>
Syntax	<pre>int xa_nnlib_cnn_get_config(xa_nnlib_handle_t handle, xa_nnlib_cnn_param_id_t param_id, void *params)</pre>
Description	Gets the value of the parameter specified by <code>param_id</code> in the buffer pointed to by <code>params</code> .
Parameters	<p>Input: <code>handle</code> The opaque component handle. Required alignment: 8 bytes.</p> <p>Input: <code>param_id</code> Identifies the parameter to be read. Refer to Table 4-32 for the list of supported parameters.</p> <p>Output: <code>params</code> A pointer to a buffer that is filled with the parameter value when the function returns. Required alignment: 4 bytes.</p>
Errors	<p>If the return value is not <code>XA>NNLIB_NO_ERROR</code>, it implies that the function has encountered one of the following errors:</p> <ul style="list-style-type: none"> ■ <code>XA>NNLIB_FATAL_MEM_ALLOC</code> One of the pointers (<code>handle</code> or <code>params</code>) is <code>NULL</code>. ■ <code>XA>NNLIB_FATAL_MEM_ALIGN</code> One of the pointers (<code>handle</code> or <code>params</code>) is not aligned correctly. <p><code>XA>NNLIB_CNN_CONFIG_FATAL_INVALID_PARAM_ID</code> Parameter identifier (<code>param_id</code>) is not valid.</p>

4.3.4 Structures Specific to CNN

Table 4-29 CNN Config Structure xa_nnlib_cnn_init_config_t

Element Type	Element Name	Range	Default	Description
xa_nnlib_shape_t	input_shape	NA	height = 16 width = 16 channels = 4	Input shape dimensions
Int32	output_height	NA	16	Output height
Int32	output_width	NA	16	Output width
Int32	output_channels	NA	4	Output depth or channels
Int32	output_format	0 or 1	0	Output data format 0: SHAPE_CUBE_DWH_T 1: SHAPE_CUBE_WHD_T
xa_nnlib_shape_t	kernel_std_shape	NA	height = 16 width = 16 channels = 4	Standard 1D/2D Convolution Kernel (Filter) shape dimensions output_channels indicate number of kernels
xa_nnlib_shape_t	kernel_ds_depth_shape	NA	NA	Depthwise Separable 2D Convolution - Depthwise Kernel (filter) Dimensions
xa_nnlib_shape_t	kernel_ds_point_shape	NA	NA	Depthwise Separable 2D Convolution - Pointwise Kernel (filter) Dimensions
xa_nnlib_shape_t	bias_std_shape	NA	channels = 4	Standard 1D/2D Convolution Bias dimensions
xa_nnlib_shape_t	bias_ds_depth_shape	NA	NA	Depthwise Separable 2D Convolution - Depthwise Bias Dimensions
xa_nnlib_shape_t	bias_ds_point_shape	NA	NA	Depthwise Separable 2D Convolution – Pointwise Bias Dimensions
xa_nnlib_cnn_precision_t	precision	XA_NNLIB_CNN_16b16b, XA_NNLIB_CNN_8b16b, XA_NNLIB_CNN_8b8b, XA_NNLIB_CNN_f32xf32	XA_NNLIB_CNN_8b16b	Kernel (filter), input, output precision setting
Int32	bias_shift	-31 to 31	7	Q-format adjustment for bias before addition into

Element Type	Element Name	Range	Default	Description
				accumulator, +/- value - left/right shift
Int32	acc_shift	-31 to 31	-7	Q-format adjustment for accumulator before rounding to result, +/- value - left/right shift
Int32	channels_multiplier	NA	NA	Depthwise Separable 2D Convolution - channel multiplier. (channels_multiplier * input_channels) must be multiple of 4
Int32	x_padding	NA	2	Left side padding to be added to input
Int32	y_padding	NA	2	Top padding to be added to input
Int32	x_stride	NA	2	Strides over padded input in width dimension
Int32	y_stride	NA	2	Strides over padded input in height dimension
xa_nnlib_cnn_algo_t	algo	NA	XA_NNLIB_CNN_CONV2D_STD	Convolution algorithm

4.3.5 Enums Specific to CNN

Table 4-30 Enum xa_nnlib_cnn_precision_t

Element	Description
XA_NNLIB_CNN_16b×16b	Coef: 16 bits, I/O: 16 bits fixed point
XA_NNLIB_CNN_8b×16b	Coef: 8 bits, I/O: 16 bits fixed point
XA_NNLIB_CNN_8b×8b	Coef: 8 bits, I/O: 8 bits fixed point
XA_NNLIB_CNN_f32×f32	Coef: single precision float, I/O: single precision float

Table 4-31 Enum xa_nnlib_cnn_algo_t

Element	Description
XA_NNLIB_CNN_CONV1D_ST	Standard 1D Convolution
XA_NNLIB_CNN_CONV2D_STD	Standard 2D Convolution
XA_NNLIB_CNN_CONV2D_DS	Depthwise Separable 2D Convolution

Table 4-32 describes parameter IDs for parameters supported by CNN. It contains the following columns:

- Parameter ID: Parameter identifier (`param_id`).
- Value type: A pointer (`params`) to a variable of this type is to be passed.
- RW: Indicates whether the parameter can be read (`get`) and/or written (`set`).
- Range: Indicates valid values of the parameter.
- Default: Default value of the parameter
- Description: Brief description of the parameter.

Table 4-32 CNN Specific Parameters

Parameter ID	Value Type	RW	Range	Default	Description
XA>NNLIB_CNN_KERNEL	<code>vect_t</code> []	<i>RW</i>	NA	NA	Kernel shape information, get or set information of the kernel shape expected by the layer
XA>NNLIB_CNN_BIAS	<code>vect_t</code> []	<i>RW</i>	NA	NA	Bias shape information, get or set information of the bias shape expected by the layer
XA>NNLIB_CNN_INPUT_SHAPE	<code>xa_nnl-lib_shape_t</code>	<i>R</i>	NA	NA	Input shape information, get information of the input shape expected by the layer.
XA>NNLIB_CNN_OUTPUT_SHAPE	<code>xa_nnl-lib_shape_t</code>	<i>R</i>	NA	NA	Output shape information, get information of the output shape produced by layer.

5. Additional Supporting Libraries

The HiFi NN library package contains additional support libraries that demonstrate the integration of the library into the NN frameworks like Android NN API and TensorFlow Lite for Microcontrollers (TFLM). These libraries are provided for reference and they can be modified as per the target application.

The HiFi NN library package contains an example speech commands application based on the TFLM micro speech example. It uses two additional libraries along with the NN library: the `xa_tf_lite_micro_lib` implements the TFLM framework and the `xa_tf_micro_frontend_lib` implements the preprocessing functionality required by the micro speech commands example. Please refer the reference source code and documentation here [2].

The HiFi NN library package also includes a library, `xa_nnlib`, that demonstrates the implementation of Android NN API v1.1 using the HiFi NN library. The below sections describe the main features and the operations supported by the `xa_nnlib` library.

5.1 *xa_nnlib Features*

- All the Android NN operations from Android NN API v1.1 are supported in the library
- Majority of the operations are supported using HiFi 4 optimized low level kernels while providing API similar to that of the reference Android NN implementation.
- The library is tested using the testcases provided in the Android CTS tests for Android NN API v1.1.

5.2 *xa_nnlib Operations*

The `xa_nnlib` includes functions that support easy integration with the Android NN API v1.1. The library supports all operations of the Android NN API v1.1 [3].

These functions are provided with similar API and the same functionality as that of the reference implementation. In few cases, the operations need additional scratch memory for the optimizations. In such cases, the APIs are modified accordingly. Please refer the reference ANN API implementation, documentation and the provided sample testbench for more details.

An example testbench that demonstrates the usage and testing of these operations is also provided, as described in Section 6.12. The operations are tested using the testcases provided with the reference implementation as part of the Android CTS test suite.

The rest of this section describes the individual ANN functions. The related function prototypes are provided in the header files included in 'test/android_nn/include/xa_nnlib_ann_api.h'.

5.2.1 Relu operations

Description

These functions perform elementwise rectified linear activation on the input. They are implemented using the HiFi optimized low-level kernels.

Algorithm

```
Relu: output = max(0, input)
Relu1: output = min(1.f, max(-1.f, input))
Relu6: output = min(6, max(0, input))
```

Prototype

```
bool genericActivationPrepare(const Shape& input, Shape* output,
                             const Operation& operation, int32_t& scratch_size);

bool reluFloat32(const float* inputData, const Shape& inputShape,
                 float* outputData, const Shape& outputShape);
bool relu1Float32(const float* inputData, const Shape& inputShape,
                  float* outputData, const Shape& outputShape);
bool relu6Float32(const float* inputData, const Shape& inputShape,
                  float* outputData, const Shape& outputShape);

bool reluQuant8(const uint8_t* inputData, const Shape& inputShape,
                uint8_t* outputData, const Shape& outputShape);
bool relu1Quant8(const uint8_t* inputData, const Shape& inputShape,
                  uint8_t* outputData, const Shape& outputShape);
bool relu6Quant8(const uint8_t* inputData, const Shape& inputShape,
                  uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float *	inputData	Pointer to the input operand
uint8_t *		
const Shape &	inputShape	Shape of the input operand
Output		
float *	outputData	Pointer to the output
uint8_t *		
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.2 Tanh

Description

This function performs elementwise hyperbolic tangent operation on the input. This function is implemented using the HiFi optimized low-level kernel.

Algorithm

```
output = tanh(input)
```

Prototype

```
bool genericActivationPrepare(const Shape& input, Shape* output,
                             const Operation& operation, int32_t& scratch_size);
```

```
bool tanhFloat32(const float* inputData, const Shape& inputShape,
                 float* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float *	inputData	Pointer to the input operand
const Shape &	inputShape	Shape of the input operand
Output		
float *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.3 Logistic

Description

These functions perform elementwise logistic or sigmoid operation on the input. They are implemented using the HiFi optimized low-level kernels.

Algorithm

$$y_n = \frac{1}{1 + \exp(-x_n)}, \quad n = 0, \dots, \overline{vec-length} - 1$$

Prototype

```
bool genericActivationPrepare(const Shape& input, Shape* output,
                             const Operation& operation, int32_t& scratch_size);

bool logisticFloat32(const float* inputData, const Shape& inputShape,
                    float* outputData, const Shape& outputShape);

bool logisticQuant8(const uint8_t* inputData, const Shape& inputShape,
                   uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float * uint8_t *	inputData	Pointer to the input operand
const Shape &	inputShape	Shape of the input operand
Output		
float * uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.4 Softmax

Description

These functions perform elementwise softmax operation on the input. They are implemented using the HiFi optimized low-level kernels.

Algorithm

$$y_n = \frac{\exp(\beta x_n)}{\sum_k \exp(\beta x_k)}, \quad n = 0, \dots, \overline{vec-length} - 1$$

Prototype

```
bool genericActivationPrepare(const Shape& input, Shape* output,
                             const Operation& operation, int32_t& scratch_size);

bool softmaxFloat32(const float* inputData, const Shape& inputShape,
                   const float beta, float* outputData,
                   const Shape& outputShape);
```

```
bool softmaxQuant8(const uint8_t* inputData, const Shape& inputShape,
                  const float beta, uint8_t* outputData,
                  const Shape& outputShape, void *p_scratch);
```

Arguments

Type	Name	Description
Input		
const float * uint8_t *	inputData	Pointer to the input operand
const Shape &	inputShape	Shape of the input operand
const float	beta	Input multiplier
const Operation&	operation	Operation
Output		
float * uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Temporary		
int32_t&	scratch_size	Size of the required scratch memory
void *	p_scratch	Pointer to scratch memory

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.5 Concatenation

Description

These functions perform concatenation of input tensors along the given dimension. These functions are included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool concatenationPrepare(const std::vector<Shape>& inputShapes,
                        int32_t axis,
                        Shape* output);

bool concatenationFloat32(const std::vector<const float*>& inputDataPtrs,
                        const std::vector<Shape>& inputShapes, int32_t axis,
                        float* outputData, const Shape& outputShape);

bool concatenationQuant8(const std::vector<const uint8_t*>& inputDataPtrs,
                        const std::vector<Shape>& inputShapes, int32_t axis,
                        uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float * uint8_t *	inputDataPtrs	Pointer to the array of pointers to input operands
const Shape &	inputShapes	Pointer to Shape of the input operand
int32_t	axis	Concatenation axis
Output		
float * uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.6 Convolution Operation

Description

These functions perform 2D convolution on the input data. These functions are implemented using the HiFi optimized low-level kernels.

Prototype

```
bool convPrepare(const Shape& input,
                const Shape& filter,
                const Shape& bias,
                int32_t padding_left, int32_t padding_right,
                int32_t padding_top, int32_t padding_bottom,
                int32_t stride_width, int32_t stride_height,
                Shape* output, int32_t& scratch_size);

bool convFloat32(const float* inputData, const Shape& inputShape,
                const float* filterData, const Shape& filterShape,
                const float* biasData, const Shape& biasShape,
                int32_t padding_left, int32_t padding_right,
                int32_t padding_top, int32_t padding_bottom,
                int32_t stride_width, int32_t stride_height,
                int32_t activation, float* outputData,
                const Shape& outputShape, void *p_scratch);

bool convQuant8(const uint8_t* inputData, const Shape& inputShape,
                const uint8_t* filterData, const Shape& filterShape,
                const int32_t* biasData, const Shape& biasShape,
```

```
int32_t padding_left, int32_t padding_right,
int32_t padding_top, int32_t padding_bottom,
int32_t stride_width, int32_t stride_height,
int32_t activation, uint8_t* outputData,
const Shape& outputShape, void *p_scratch);
```

Arguments

Type	Name	Description
Input		
const float * const uint8_t *	inputData, filterData, biasData	Pointer to the input, filter and bias operands
const Shape &	inputShape, filterShape, biasShape	Pointer to Shape of the input, filter and bias operands
int32_t	padding_left, padding_right, padding_top, padding_bottom	Padding values.
int32_t	stride_width, stride_height	Stride values
int32_t	activation	Fused activation function selection
Output		
float * uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Temporary		
int32_t&	scratch_size	Size of the required scratch memory
void *	p_scratch	Pointer to scratch memory

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.7 Depth-wise Convolution Operation

Description

These functions perform depth-wise 2D convolution on the input data. They are implemented using the HiFi optimized low-level kernels.

Prototype

```
bool depthwiseConvPrepare(const Shape& input,
                          const Shape& filter,
                          const Shape& bias,
                          int32_t padding_left, int32_t padding_right,
```

```
int32_t padding_top, int32_t padding_bottom,
int32_t stride_width, int32_t stride_height,
Shape* output, int32_t& scratch_size);
```

```
bool depthwiseConvFloat32(const float* inputData, const Shape& inputShape,
    const float* filterData, const Shape& filterShape,
    const float* biasData, const Shape& biasShape,
    int32_t padding_left, int32_t padding_right,
    int32_t padding_top, int32_t padding_bottom,
    int32_t stride_width, int32_t stride_height,
    int32_t depth_multiplier, int32_t activation,
    float* outputData, const Shape& outputShape, void* p_scratch);
```

```
bool depthwiseConvQuant8(const uint8_t* inputData, const Shape& inputShape,
    const uint8_t* filterData, const Shape& filterShape,
    const int32_t* biasData, const Shape& biasShape,
    int32_t padding_left, int32_t padding_right,
    int32_t padding_top, int32_t padding_bottom,
    int32_t stride_width, int32_t stride_height,
    int32_t depth_multiplier, int32_t activation,
    uint8_t* outputData, const Shape& outputShape,
    void *p_scratch);
```

Arguments

Type	Name	Description
Input		
const float * const uint8_t *	inputData, filterData, biasData	Pointer to the input, filter and bias operands
const Shape &	inputShape, filterShape, biasShape	Pointer to Shape of the input, filter and bias operands
int32_t	padding_left, padding_right, padding_top, padding_bottom	Padding values.
int32_t	stride_width, stride_height	Stride values
int32_t	depth_multiplier	Depthwise multiplier
int32_t	activation	Fused activation function selection
Output		
float * uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Temporary		
int32_t&	scratch_size	Size of the required scratch memory
void *	p_scratch	Pointer to scratch memory

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.8 Fully Connected

Description

These functions perform multiplication of the weight matrix with the input vectors in a fully connected neural network layer i.e. $z = \text{weight} \times \text{input} + \text{bias}$. They are implemented using the HiFi optimized low-level kernels.

Prototype

```
bool fullyConnectedPrepare(const Shape& input,
                          const Shape& weights,
                          const Shape& bias,
                          Shape* output);

bool fullyConnectedFloat32(const float* inputData, const Shape& inputShape,
                          const float* weights, const Shape& weightsShape,
                          const float* biasData, const Shape& biasShape,
                          int32_t activation, float* outputData,
                          const Shape& outputShape);

bool fullyConnectedQuant8(const uint8_t* inputData, const Shape& inputShape,
                          const uint8_t* weights, const Shape& weightsShape,
                          const int32_t* biasData, const Shape& biasShape,
                          int32_t activation, uint8_t* outputData,
                          const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float * uint8_t *	inputData, weights, biasData	Pointer to the input operands
const Shape &	inputShape, weightsShape, biasShape	Shape of the input operand
int32_t	activation	Fused activation function selection
Output		
float * uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.9 L2 Normalization

Description

These functions perform l2 normalization on the input to get output which has unity l2-norm. They are included as is from the reference implementation without any HiFi optimization.

Algorithm

$$z_n = \frac{x_n}{\sqrt{\sum_{n=1}^N |x_n|^2}}, \quad n = 1 \dots, \overline{num-elements}$$

x_n represents input vector.

z_n represents output vector.

Prototype

```
bool l2normFloat32(const float* inputData, const Shape& inputShape,
                  float* outputData, const Shape& outputShape);
```

```
bool l2normQuant8(const uint8_t* inputData, const Shape& inputShape,
                  uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float * uint8_t *	inputData	Pointer to the input operand
const Shape &	inputShape	Shape of the input operand
Output		
float *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.10 Pooling operations

Description

Pooling functions perform 2D pooling (average, max, L2) on the input data. They are implemented using the HiFi optimized low-level kernels.

Prototype

```
bool genericPoolingPrepare(const Shape& input,
                          int32_t padding_left, int32_t padding_right,
                          int32_t padding_top, int32_t padding_bottom,
                          int32_t stride_width, int32_t stride_height,
                          int32_t filter_width, int32_t filter_height,
                          Shape* output, const Operation& operation,
                          int32_t& scratch_size);

bool averagePoolFloat32(const float* inputData, const Shape& inputShape,
                       int32_t padding_left, int32_t padding_right,
                       int32_t padding_top, int32_t padding_bottom,
                       int32_t stride_width, int32_t stride_height,
                       int32_t filter_width, int32_t filter_height, int32_t activation,
                       float* outputData, const Shape& outputShape, void* p_scratch);

bool averagePoolQuant8(const uint8_t* inputData, const Shape& inputShape,
                      int32_t padding_left, int32_t padding_right,
                      int32_t padding_top, int32_t padding_bottom,
                      int32_t stride_width, int32_t stride_height,
                      int32_t filter_width, int32_t filter_height, int32_t activation,
                      uint8_t* outputData, const Shape& outputShape, void* p_scratch);

bool l2PoolFloat32(const float* inputData, const Shape& inputShape,
                  int32_t padding_left, int32_t padding_right,
                  int32_t padding_top, int32_t padding_bottom,
                  int32_t stride_width, int32_t stride_height,
                  int32_t filter_width, int32_t filter_height, int32_t activation,
                  float* outputData, const Shape& outputShape);

bool maxPoolFloat32(const float* inputData, const Shape& inputShape,
                   int32_t padding_left, int32_t padding_right,
                   int32_t padding_top, int32_t padding_bottom,
                   int32_t stride_width, int32_t stride_height,
                   int32_t filter_width, int32_t filter_height, int32_t activation,
                   float* outputData, const Shape& outputShape, void* p_scratch);

bool maxPoolQuant8(const uint8_t* inputData, const Shape& inputShape,
                  int32_t padding_left, int32_t padding_right,
                  int32_t padding_top, int32_t padding_bottom,
                  int32_t stride_width, int32_t stride_height,
                  int32_t filter_width, int32_t filter_height, int32_t activation,
                  uint8_t* outputData, const Shape& outputShape, void* p_scratch);
```

Arguments

Type	Name	Description
Input		
const float * uint8_t *	inputData	Pointer to the input, filter and bias operands
const Shape &	inputShape	Pointer to Shape of the input, filter and bias operands

int32_t	padding_left, padding_right, padding_top, padding_bottom	Padding values.
int32_t	stride_width, stride_height	Stride values
int32_t	filter_width, filter_height	Filter dimensions
int32_t	activation	Fused activation function selection
Output		
float *	outputData	Pointer to the output
uint8_t *	outputShape	Shape of the output
Temporary		
int32_t&	scratch_size	Size of the required scratch memory
void *	p_scratch	Pointer to scratch memory

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.11 Basic operations

Description

These functions perform basic elementwise operations. They are implemented using the HiFi optimized low-level kernels.

Prototype

```
bool addFloat32(const float* in1, const Shape& shape1,
               const float* in2, const Shape& shape2,
               int32_t activation,
               float* out, const Shape& shapeOut);

bool addQuant8(const uint8_t* in1, const Shape& shape1,
               const uint8_t* in2, const Shape& shape2,
               int32_t activation,
               uint8_t* out, const Shape& shapeOut);

bool mulFloat32(const float* in1, const Shape& shape1,
               const float* in2, const Shape& shape2,
               int32_t activation,
               float* out, const Shape& shapeOut);

bool mulQuant8(const uint8_t* in1, const Shape& shape1,
               const uint8_t* in2, const Shape& shape2,
               int32_t activation,
               uint8_t* out, const Shape& shapeOut);
```

```

bool floorFloat32(const float* inputData,
                  float* outputData,
                  const Shape& shape);

bool subFloat32(const float* in1, const Shape& shape1,
                const float* in2, const Shape& shape2,
                int32_t activation,
                float* out, const Shape& shapeOut);

bool divFloat32(const float* in1, const Shape& shape1,
                const float* in2, const Shape& shape2,
                int32_t activation,
                float* out, const Shape& shapeOut);

```

Arguments

Type	Name	Description
Input		
const float *	in1, in2	Pointer to the input operand
const Shape &	shape1, shape2	Shape of the input operand
Output		
float *	out	Pointer to the output
const Shape &	shapeOut	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.12 Local Response Norm

Description

This function performs local response normalization along the depth dimension of a 4-D tensor. It is implemented using the HiFi optimized low-level kernels.

Prototype

```

bool localResponseNormFloat32(const float* inputData, const Shape& inputShape,
                              int32_t radius, float bias, float alpha, float beta,
                              float* outputData, const Shape& outputShape);

```

Arguments

Type	Name	Description
------	------	-------------

Input		
const float *	inputData	Pointer to the input operand
const Shape &	inputShape	Shape of the input operand
int32_t	radius	Depth radius
float	bias	Bias value that is added to product of squared sum and multiplication factor.
float	alpha	Multiplication factor of squared sum
float	Beta	Power factor
Output		
float *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.13 Reshape Generic

Description

This function reshapes a tensor in newly specified shape. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool reshapePrepare(const Shape& input,
                   const int32_t* targetDims,
                   const int32_t targetDimsSize,
                   Shape* output);

bool reshapeGeneric(const void* inputData, const Shape& inputShape,
                   void* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const void *	inputData	Pointer to input operands
const Shape &	inputShape	Shape of the input operand
int32_t *	targetDims	Pointer to target dimension.
int32_t	targetDimsSize	Target dimension size
Output		
void *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output

Shape *	output	Pointer to output shape
---------	--------	-------------------------

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.14 Resize Bilinear

Description

This function resizes images using bilinear interpolation. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool resizeBilinearPrepare(const Shape& input,
                          int32_t height,
                          int32_t width,
                          Shape* output);

bool resizeBilinearFloat32(const float* inputData,
                          const Shape& inputShape,
                          float* outputData,
                          const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float *	inputData	Pointer to input operands
const Shape &	inputShape	Shape of the input operand
int32_t	height	Target height.
int32_t	width	Target width.
Output		
float *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Shape *	output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.15 Depth to Space

Description

This function rearranges data from depth to spatial blocks. It unfolds depth data into non-overlapping spatial blocks of size blockSize * blockSize. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool depthToSpacePrepare(const Shape& input,
                        int32_t blockSize,
                        Shape* output);

bool depthToSpaceGeneric(const uint8_t* inputData, const Shape& inputShape,
                        int32_t blockSize,
                        uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float *	inputData	Pointer to input operands
const Shape &	inputShape	Shape of the input operand
int32_t	blockSize	Target blocksize.
Output		
float *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Shape *	Output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.16 Space to Depth

Description

This function rearranges data from spatial blocks to depth. It folds non-overlapping spatial blocks of size blockSize * blockSize into depth data. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool spaceToDepthPrepare(const Shape& input,
                        int32_t blockSize,
                        Shape* output);
```

```
bool spaceToDepthGeneric(const uint8_t* inputData, const Shape& inputShape,
                        int32_t blockSize,
                        uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float *	inputData	Pointer to input operands
const Shape &	inputShape	Shape of the input operand
int32_t	blockSize	Target blocksize.
Output		
float *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Shape *	Output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.17 Pad

Description

This operation pads input with zeros according to the specified paddings.

Prototype

```
bool padPrepare(const Shape& input,
               const int32_t* paddingsData,
               const Shape& paddingsShape,
               Shape* output);
```

```
bool padGeneric(const uint8_t* inputData, const Shape& inputShape,
               const int32_t* paddings,
               uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const float *	inputData	Pointer to input operands
const Shape &	inputShape, paddingsShape	Shape of the input operand
int32_t *	paddingsShape, paddings	Target padding
Output		
float *	outputData	Pointer to the output

const Shape &	outputShape	Shape of the output
Shape *	Output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.18 Batch to Space

Description

BatchToSpace for N-dimensional tensors.

This operation reshapes the batch dimension (dimension 0) into $M + 1$ dimensions of shape `block_shape + [batch]`, interleaves these blocks back into the grid defined by the spatial dimensions `[1, ..., M]`, to obtain a result with the same rank as the input.

This is the reverse of `SpaceToBatch`.

It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool batchToSpacePrepare(const Shape& input,
                        const int32_t* blockSizeData,
                        const Shape& blockSizeShape,
                        Shape* output);

bool batchToSpaceGeneric(const uint8_t* inputData, const Shape& inputShape,
                        const int32_t* blockSize,
                        uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const uint8_t *	inputData	Pointer to input operands
const Shape &	inputShape, blockSizeShape	Shape of the input operand
Const int32_t *	blockSize, blockSizeData	Target block size.
Output		
uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Shape *	Output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.19 Space to Batch

Description

SpaceToBatch for N-Dimensional tensors.

This operation divides "spatial" dimensions [1, ..., M] of the input into a grid of blocks of shape `block_shape`, and interleaves these blocks with the "batch" dimension (0) such that in the output, the spatial dimensions [1, ..., M] correspond to the position within the grid, and the batch dimension combines both the position within a spatial block and the original batch position. Prior to division into blocks, the spatial dimensions of the input are optionally zero padded according to `paddings`.

It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool spaceToBatchPrepare(const Shape& input,
                        const int32_t* blockSizeData,
                        const Shape& blockSizeShape,
                        const int32_t* paddingsData,
                        const Shape& paddingsShape,
                        Shape* output);

bool spaceToBatchGeneric(const uint8_t* inputData, const Shape& inputShape,
                        const int32_t* blockSize,
                        const int32_t* padding, const Shape& paddingShape,
                        uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
<code>const uint8_t *</code>	<code>inputData</code>	Pointer to input operands
<code>const Shape &</code>	<code>inputShape,</code> <code>paddingShape</code>	Shape of the input operand
<code>const int32_t *</code>	<code>blockSize,</code> <code>blockSizeData</code>	Target block size.
<code>const int32_t *</code>	<code>padding,</code> <code>paddingsData</code>	Target Padding.
Output		
<code>uint8_t *</code>	<code>outputData</code>	Pointer to the output
<code>const Shape &</code>	<code>outputShape</code>	Shape of the output
<code>Shape *</code>	<code>Output</code>	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.20 Squeeze

Description

This function removes dimensions of size 1 from the input tensor.
It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool squeezePrepare(const Shape& input,
                   const int32_t* squeezeDims,
                   const Shape& squeezeDimsShape,
                   Shape* output);

bool squeezeGeneric(const void* inputData, const Shape& inputShape,
                   void* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const void *	inputData	Pointer to input operands
const Shape &	inputShape, squeezeDimsShape	Shape of the input operand
const int32_t *	squeezeDims	Target squeeze dimension.
Output		
void *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Shape *	Output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.21 Transpose

Description

This function transposes the input tensor according to permute tensor. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool transposePrepare(const Shape& input,
                    const int32_t* permData,
                    const Shape& permShape,
                    Shape* output);

bool transposeGeneric(const uint8_t* inputData, const Shape& inputShape,
                    const int32_t* perm, const Shape& permShape,
                    uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const uint8_t *	inputData	Pointer to input operands
const Shape &	inputShape, permShape	Shape of the input operand
const int32_t *	permData, perm	Target permutation.
Output		
uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Shape *	Output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.22 Mean

Description

Computes the mean of elements across dimensions of a tensor.

Reduces the input tensor along the given dimensions to reduce. Unless keep_dims is true, the rank of the tensor is reduced by 1 for each entry in axis. If keep_dims is true, the reduced dimensions are retained with length 1.

It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool meanPrepare(const Shape& input,
                const int32_t* axisData,
                const Shape& axisShape,
                bool keepDims,
                Shape* output);

bool meanGeneric(const uint8_t* inputData, const Shape& inputShape,
                const int32_t* axis, const Shape& axisShape, bool keepDims,
                uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const uint8_t *	inputData	Pointer to input operands
const Shape &	inputShape, axisShape	Shape of the input operand
const int32_t *	axis, axisData	Mean axis.
bool	keepDims	Flag: true if dimension to be retained, false if output dimension is to be reduced.
Output		
uint8_t *	outputData	Pointer to the output
const Shape &	outputShape	Shape of the output
Shape *	Output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.23 Strided Slice

Description

This function extracts a strided slice of a tensor.

More specifically this operation extracts a slice of size (end - begin) / stride from the given input tensor. Starting at the location specified by begin the slice continues by adding stride to the index until all dimensions are not less than end. Note that a stride can be negative, which causes a reverse slice.

It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool stridedSlicePrepare(const Shape& input,
                        const int32_t* beginData, const Shape& beginShape,
                        const int32_t* endData, const Shape& endShape,
                        const int32_t* stridesData, const Shape& stridesShape,
                        int32_t beginMask, int32_t endMask, int32_t shrinkAxisMask,
                        Shape* output);

bool stridedSliceGeneric(const uint8_t* inputData, const Shape& inputShape,
                        const int32_t* beginData, const int32_t* endData,
                        const int32_t* stridesData,
                        int32_t beginMask, int32_t endMask, int32_t shrinkAxisMask,
                        uint8_t* outputData, const Shape& outputShape);
```

Arguments

Type	Name	Description
Input		
const uint8_t *	inputData	Pointer to input operands
const Shape &	inputShape, beginShape, endShape, stridesShape	Shape of the operands
const int32_t *	beginData, endData, stridesData	Pointer to the begin, end and stride values
int32_t	beginMask, endMask, shrinkAxisMask	Begin, end and shrink mask values
Output		
uint8_t *	outputData	Pointer to the output
Shape *	Output	Pointer to output shape
const Shape &	outputShape	Shape of the output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.24 Dequantize Quant8 to Float32

Description

This function performs dequantization of quant8 format to float32 data. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool dequantizePrepare(const Shape& input, Shape* output);
```

```
bool dequantizeQuant8ToFloat32(const uint8_t* inputData,
                              float* outputData,
                              const Shape& shape);
```

Arguments

Type	Name	Description
Input		
const uint8_t *	inputData	Pointer to the input operand
const Shape &	shape, input	Shape of the input operand
Output		
float *	outputData	Pointer to the output
Shape *	output	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.25 Embedding Lookup

Description

This module implements the embedded lookup operation as specified in the Android NN API v1.1 reference implementation. It concatenates sub-tensors from the given input tensor according to the given indices tensor. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool embeddingLookupPrepare(const Shape &valueShape,
                           const Shape &lookupShape,
                           Shape *outputShape);

EmbeddingLookup::EmbeddingLookup(
    const android::hardware::neuralnetworks::V1_1::Operation &operation,
    std::vector<RunTimeOperandInfo> &operands);

bool EmbeddingLookup::Eval()
```

Arguments

Type	Name	Description
Input		
const Shape &	valueShape, lookupShape	Reference to input and lookup shape.
std::vector<RunTimeOperandInfo> &	operands	List of operands specified as RunTimeOperandInfo
Output		
Shape *	outputShape	Pointer to outputShape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.26 Hashtable Lookup

Description

This module implements the hashtable lookup operation as specified in the Android NN API v1.1 reference implementation. It concatenates sub-tensors from the given input tensor according to the given key-value map. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
bool hashtableLookupPrepare(const Shape &lookupShape,
                           const Shape &keyShape,
                           const Shape &valueShape,
                           Shape *outputShape,
                           Shape *hitShape);

HashtableLookup::HashtableLookup(
    const android::hardware::neuralnetworks::V1_1::Operation &operation,
    std::vector<RunTimeOperandInfo> &operands);

bool HashtableLookup::Eval()
```

Arguments

Type	Name	Description
Input		
Operation &	operation	ANN operation structure instance of the type LSH_PROJECTION
const Shape &	lookupShape, keyShape, valueShape	Shapes of the inputs: lookup, key and values
std::vector<RunTimeOperandInfo> &	operands	List of operands specified as RunTimeOperandInfo
Output		
Shape *	outputShape	Pointer to output shape
Shape *	hitShape	Pointer to the hits output

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.27 LSH Projection

Description

This module implements the LSH projection operation as specified in the Android NN API v1.1 reference implementation. It projects an input to a bit vector using locality sensitive hashing. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
LSHProjection::LSHProjection(
    const android::hardware::neuralnetworks::V1_1::Operation &operation,
    std::vector<RunTimeOperandInfo> &operands);

bool LSHProjection::Prepare(
    const android::hardware::neuralnetworks::V1_1::Operation &operation,
    std::vector<RunTimeOperandInfo> & operands,
    Shape *outputShape);

bool LSHProjection::Eval();
```

Arguments

Type	Name	Description
Input		
Operation &	operation	ANN operation structure instance of the type LSH_PROJECTION
std::vector<RunTimeOperandInfo> &	operands	List of operands specified as RunTimeOperandInfo
Output		
Shape *	outputShape	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.28 LSTM

Description

These functions perform a single time step in a LSTM layer as specified in the Android NN API v1.1 reference implementation. They are implemented using the HiFi optimized low-level kernels.

Prototype

```
LSTMCell::LSTMCell(const android::hardware::neuralnetworks::V1_1::Operation &operation,
```

```

        std::vector<RunTimeOperandInfo> &operands);

static bool LSTMCell::Prepare(const android::hardware::neuralnetworks::V1_1::Operation &operation,
                             std::vector<RunTimeOperandInfo> &operands,
                             Shape *scratchShape,
                             Shape *outputStateShape,
                             Shape *cellStateShape,
                             Shape *outputShape);

bool LSTMCell::Eval();

```

Arguments

Type	Name	Description
Input		
Operation	operation	ANN operation instance of the type LSTM
std::vector<RunTimeOperandInfo> &	operands	List of operands specified as RunTimeOperandInfo
Shape *	cellStateShape	Pointer to cell state shape
Output		
Shape *	outputShape	Pointer to output shape
Shape *	outputStateShape	Pointer to output state shape
Temporary		
Shape *	scratchShape	Pointer to scratch shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.29 RNN

Description

These functions implement a basic recurrent neural network as specified in the Android NN API v1.1 reference implementation. They are implemented using the HiFi optimized low-level kernels.

Prototype

```

RNN::RNN(const android::hardware::neuralnetworks::V1_1::Operation &operation,
         std::vector<RunTimeOperandInfo> &operands);

bool RNN::Prepare(const android::hardware::neuralnetworks::V1_1::Operation &operation,
                  std::vector<RunTimeOperandInfo> &operands,
                  Shape *hiddenStateShape,
                  Shape *outputShape);

bool RNN::Eval();

```

Arguments

Type	Name	Description
Input		
Operation	operation	ANN operation instance of the type RNN
std::vector<RunTimeOperandInfo> &	operands	List of operands specified as RunTimeOperandInfo
Shape *	hiddenStateShape	Pointer to shape of the state
Output		
Shape *	outputShape	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

5.2.30 SVDF

Description

This module implements the SVDF operation as specified in the Android NN API v1.1 reference implementation. It is included as is from the reference implementation without any HiFi optimization.

Prototype

```
SVDF::SVDF(const android::hardware::neuralnetworks::V1_1::Operation &operation,
           std::vector<RunTimeOperandInfo>& operands);

bool SVDF::Prepare(
    const hardware::neuralnetworks::V1_1::Operation &operation,
    std::vector<RunTimeOperandInfo> &operands, Shape *stateShape,
    Shape *outputShape);

bool SVDF::Eval();
```

Arguments

Type	Name	Description
Input		
Operation	operation	ANN operation instance of the type SVDF
std::vector<RunTimeOperandInfo> &	operands	List of operands specified as RunTimeOperandInfo
Shape *	stateShape	Pointer to state shape
Output		
Shape *	outputShape	Pointer to output shape

Returns

- 1 (true): no error
- 0 (false): error, invalid parameters

6. Introduction to the Example Testbench

6.1 Making the Library

To build and execute the application from Xtensa Xplorer workspace (.xws) based release package, please refer to the readme.html file available in the imported application project.

To build the library in makefile based (.tgz) package, following steps are required.

If you have source code distribution, you must build the NN library before you can build the testbench. To do so, follow these steps:

1. Go to build.
2. From the command prompt, enter:

```
xt-make -f makefile detected_core=hifi4 clean all install
```

The NN library `xa_nnlib.a` will be built and copied to the `lib` directory.

6.1.1 Controlling Library Code Size

The HiFi NN Library code size can be reduced by discarding unused functions at the time of linking.

The library is compiled with the `'-ffunction-sections'` option. With this option, the compiler puts each function in a separate section. This enables the linker to discard unused functions when linking the executable, using the `'-Wl,-gc-sections'` linker option.

Additionally, to remove unused function sections during the library creation, the `'-Wl,-gc-sections'` linker option is enabled while building the library. The list of required functions is provided in the linker script file `build/ldscript_nnlib.txt`. While building the library, the linker discards functions not listed as `'EXTERN'` in the linker script file. By appropriately modifying the linker script, the library can be built with only the kernels required for particular application.

6.2 Making the Executable

To build and execute the application from Xtensa Xplorer workspace (.xws) based release package, please refer to the readme.html file available in the imported application project.

To build the library in makefile based (.tgz) package, following steps are required.

To build the testbenches, follow these steps:

1. Go to `test/build`.
2. From the command-line prompt, enter:

```
xt-make -f makefile_testbench_sample detected_core=hifi4 clean all
```

This will build the example testbenches for all the kernels and layers.

The following header files are common and used by all testbenches.

- Testbench header files (`test/include`)
 - `xt_profiler.h`
 - `cmdline_parser.h`
 - `file_io.h`
 - `xt_manage_buffers.h`

The following sections describe each low-level kernel and layer testbench.

6.2.1 Controlling Executable Code Size

The code size of the executable binaries can be reduced by discarding unused functions at the time of linking.

The library is compiled with the `'-ffunction-sections'` option. With this option, the compiler puts each function in a separate section. This enables the linker to discard unused functions when linking the executable, using the `'-Wl,-gc-sections'` linker option.

6.3 Sample Testbench for Matrix X Vector Multiplication Kernels

The NN library Matrix X Vector Multiplication Kernels are provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_matXvec_testbench.c

6.3.1 Usage

The NN library Matrix X Vector Multiplication Kernels executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_matXvec_test [options]
```

Following are available options:

Option	Description	Additional Information
-rows	Rows of mat1.	
-cols1	Columns of mat1 and rows of mat2 (Default=32)	Columns of mat1 must be multiple of 4
-cols2	Columns of mat2 (Default=32)	Columns of mat2 must be multiple of 4
-row_stride1	Row stride for mat1(Default=32)	
-row_stride2	Row stride for mat2(Default=32)	
-vec_count	Vec count for Time batching (Default=1)	
-acc_shift	Accumulator shift (Default=-7)	
-bias_shift	Bias shift (Default=7)	
-mat_precision	8, 16, -1(single precision float), -3 (asym8u) or -5 (sym8s); (Default=16)	
-inp_precision	8, 16, -1(single precision float), -3(asym8u) or -4 (asym8s); (Default=16)	
-out_precision	8, 16, 32, 64, -1(single precision float), -3(asym8u) or -4 (asym8s); (Default=16)	
-bias_precision	16, 64 -1(single precision float), 32(asym8); (Default=16)	
-mat1_zero_bias	Matrix1 zero bias for asym8 -255 to 0; Default=-128	
-mat2_zero_bias	Matrix2 zero bias for asym8 -255 to 0; Default=-128	
-inp1_zero_bias	Input1 zero bias for asym8 -255 to 0; Default=-128	

Option	Description	Additional Information
-inp2_zero_bias	Input2 zero bias for asym8 -255 to 0; Default=-128	
-out_multiplier	Output multiplier for asym8 0 to 0x7ffffff; Default=0x40000000	
-out_shift	Output shift for asym8 -31 to 31; Default=-8	
-out_zero_bias	Output zero bias for asym8 0 to 255; Default=128	
-membank_padding	0 or 1 (Default=1)	
-frames	Positive number; (Default=2)	
-activation	Sigmoid, tanh, relu or softmax (Default= bypass i.e. no activation for output)	
-write_file	Set to 1 to write input and output vectors to file; (Default=0)	
-read_inp_file_name	Full filename for reading inputs (order - mat1, vec1, mat2, vec2, bias)	
-read_ref_file_name	Full filename for reading reference output	
-write_inp_file_name	Full filename for writing inputs (order - mat1, vec1, mat2, vec2, bias)	
-write_out_file_name	Full filename for writing output	
-verify	Verify output against provided reference	0: Disable, 1: Bit exact match (Default=1)
-batch	Flag to execute time batching kernels	0: Disable, 1: Enable (Default=0)
-fc	Flag to execute fully connected kernels	0: Disable, 1: Enable (Default=0)
-help	Prints help	

If no command line arguments are given, the Matrix X Vector Multiplication Kernels sample testbench runs with default values.

6.4 Sample Testbench for Convolution Kernels

The NN library Convolutional Kernels are provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_conv_testbench.c

6.4.1 Usage

The NN Library Convolutional Kernels executable can be run with command-line options as follows.


```
$ xt-run [--mem_model] [--turbo] xa_nn_conv_test [options]
```

Following are available options:

Option	Description
-input_height	Input height (Default=16)
-input_width	Input width (Default=16)
-input_channels	Input channels (Default=4)
-kernel_height	Kernel height (Default=3)
-kernel_width	Kernel width (Default=3)
-out_channels	Out channels (Default=4)
-channels_multiplier	Channel Multiplier (Default=1)
-x_stride	Stride in width dimension (Default=2)
-y_stride	Stride in height dimension (Default=2)
-x_padding	Left padding in width dimension (Default=2)
-y_padding	Top padding in height dimension (Default=2)
-out_height	Output height (Default=16)
-out_width	Output width (Default=16)
-bias_shift	Bias shift (Default=7)
-acc_shift	Accumulator shift (Default=-7)
-inp_data_format	Input data format, 0 (SHAPE_CUBE_DWH_T), 1 (SHAPE_CUBE_WHD_T); Default=1, ignored for conv2d_std and conv1d_std kernels
-out_data_format	0 (SHAPE_CUBE_DWH_T), 1 (SHAPE_CUBE_WHD_T) (Default=0)
-inp_precision	8, 16, -1(single precision float), -3(asym8u) or -4 (asym8s); (Default=16)
-kernel_precision	8, 16, -1(single precision float), -3(asym8u) or -5 (sym8s); (Default=8)
-out_precision	8, 16, -1(single precision float), -3(asym8u) or -4 (asym8s); (Default=16)
-bias_precision	8, 16, -1(single precision float), 32(asym8); (Default=16)
-input_zero_bias	Input zero bias for asym8, 0 to -255; Default=-128
-kernel_zero_bias	Kernel zero bias for asym8, 0 to -255; Default=-128
-out_multiplier	Output multiplier in Q31 format for asym8, 0x0 to 0x7ffffff; Default=0x40000000
-out_shift	Output shift for asym8, 31 to -31; Default=-8

Option	Description
-out_zero_bias	Output zero bias for asym8, 0 to 255; Default=128
-frames	Positive number (Default=2)
-kernel_name	conv2d_std, conv2d_depth, conv1d_std; (Default= conv2d_std)
-write_file	Set to 1 to write input and output vectors to file; (Default=0)
-read_inp_file_name	Full filename for reading inputs (order - input, kernel, bias, (pointwise kernel, pointwise bias for depth separable))
-read_ref_file_name	Full filename for reading reference output
-write_inp_file_name	Full filename for writing inputs (order - input, kernel, bias, (pointwise kernel, pointwise bias for depth separable))
-write_out_file_name	Full filename for writing output
-verify	Verify output against provided reference; 0: Disable, 1: Bit exact match (Default=1)
-help	Prints help

If no command line arguments are given, the Convolutional Kernels sample testbench runs with default values.

6.5 Sample Testbench for Activation Kernels

The NN library Activation kernels are provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_activations_testbench.c

6.5.1 Usage

The NN library Activation Kernels executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_activation_test [options]
```

Following are available options:

Option	Description
-num_elements	Number of elements (Default=32)
-relu_threshold	Threshold for relu in Q16.15 (Default= 32768 i.e. =1 in Q16.15)
-inp_precision	8,16, 32, -1(single precision float), -3(asym8u) or -4 (asym8s); (Default=32)
-out_precision	8,16, 32, -1(single precision float), -3(asym8u) or -4 (asym8s); (Default=32)
-frames	Positive number (Default=2)
-activation	Sigmoid, tanh, relu, relu1, relu6, relu_std or softmax (Default= sigmoid)
-write_file	Set to 1 to write input and output vectors to file; (Default=0)
-read_inp_file_name	Full filename for reading input
-read_ref_file_name	Full filename for reading reference output
-write_inp_file_name	Full filename for writing input
-write_out_file_name	Full filename for writing output
-verify	Verify output against provided reference; 0: Disable, 1: Bit exact match (Default=1)
ASYM8 specific parameters	
-diffmin	Diffmin; Default=-15
-input_left_shift	Input_left_shift; Default=27
-input_multiplier	Input_multiplier; Default=2060158080

Option	Description
-activation_max	Asym8 input data activation max; Default=0
-activation_min	Asym8 input data activation min; Default=0
-input_range_radius	Sigmoid_asym8 input parameter; Default=128
-zero_point	Sigmoid_asym8 input parameter; Default=0
-help	Prints help

If no command line arguments are given, the Activation Kernels sample testbench runs with default values.

6.6 Sample Testbench for Pooling Kernels

The NN library Pooling Kernels are provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_pool_testbench.c

6.6.1 Usage

The NN library Pooling Kernels executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_pool_test [options]
```

Following are available options:

Option	Description
-input_height	Input height (Default=16)
-input_width	Input width (Default=16)
-input_channels	Input channels (Default=4)
-kernel_height	Kernel height (Default=3)
-kernel_width	Kernel width (Default=3)
-x_stride	Stride in width dimension (Default=2)
-y_stride	Stride in height dimension (Default=2)
-x_padding	Left padding in width dimension (Default=2)
-y_padding	Top padding in height dimension (Default=2)
-out_height	Output height (Default=16)
-out_width	Output width (Default=16)

Option	Description
-acc_shift	Accumulator shift (Default=-7)
-inp_data_format	Input data format, 0 (SHAPE_CUBE_DWH_T), 1 (SHAPE_CUBE_WHD_T); (Default=1 (SHAPE_CUBE_WHD_T))
-out_data_format	Output data format, 0 (SHAPE_CUBE_DWH_T), 1 (SHAPE_CUBE_WHD_T); (Default=1 (SHAPE_CUBE_WHD_T))
-inp_precision	8, 16, -1(single precision float), -3(asym8); (Default=16)
-out_precision	8, 16, -1(single precision float), -3(asym8); (Default=16)
-frames	Positive number (Default=2)
-kernel_name	avgpool, maxpool (Default=avgpool)
-write_file	set to 1 to write input and output vectors to file; (Default=0)
-read_inp_file_name	Full filename for reading inputs (order - inp)
-read_ref_file_name	Full filename for reading reference output
-write_inp_file_name	Full filename for writing inputs (order - inp)
-write_out_file_name	Full filename for writing output
-verify	Verify output against provided reference; 0: Disable, 1: Bit exact match (Default=1)
-help	Prints help

If no command line arguments are given, the Pooling Kernels sample testbench runs with default values.

6.7 Sample Testbench for Basic Kernels

The NN library Basic Kernels are provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_basic_testbench.c

6.7.1 Usage

The NN library Basic Kernels executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_basic_test [options]
```

Following are available options:

Option	Description
-io_length	Input/output vector length; Default=1024
-io_precision	-3 (asym8), -1 (single prec float); Default=-1
-frames	Positive number; Default=2
-kernel_name	; Default=avgpool
-write_file	Set to 1 to write input and output vectors to file; Default=0
-read_inp1_file_name	Full filename for reading inputs (order - inp)
-read_inp2_file_name	Full filename for reading inputs (order - inp)
-read_ref_file_name	Full filename for reading reference output
-write_inp1_file_name	Full filename for writing inputs (order - inp)
-write_inp2_file_name	Full filename for writing inputs (order - inp)
-write_out_file_name	Full filename for writing output
-verify	Verify output against provided reference; 0: Disable, 1: Bit exact match; Default=1
ASYM8 specific parameters	
-output_zero_bias	Output zero bias; Default=127
-output_left_shift	Output_left_shift; Default=1
-output_multiplier	Output_multiplier; Default=0x7fff
-output_activation_min	Output_activation_min; Default=0
-output_activation_max	Output_activation_max; Default=225
-input1_zero_bias	Input1 zero bias (Only needed in add_asym8); Default=-127
-input1_left_shift	Input1_left_shift (Only needed in add_asym8); Default=0
-input1_multiplier	Input1_multiplier (Only needed in add_asym8); Default=0x7fff
-input2_zero_bias	Input2 zero bias (Only needed in add_asym8); Default=-127
-input2_left_shift	Input2_left_shift (Only needed in add_asym8); Default=0

Option	Description
-input2_multiplier	Input2_multiplier (Only needed in add_asym8); Default=0x7fff
-left_shift	Global left_shift (Only needed in add_asym8); Default=0
-h	Prints help

If no command line arguments are given, the Basic Kernels sample testbench runs with default values.

6.8 Sample Testbench for Normalization Kernels

The NN library Normalization Kernels are provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_norm_testbench.c

6.8.1 Usage

The NN library Normalization Kernels executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_norm_test [options]
```

Following are available options:

Option	Description
-num_elms	Number of elements; Default=256
-inp_precision	8, 16, -1(single prec float); Default=16
-out_precision	8, 16, -1(single prec float); Default=16
-frames	Positive number; Default=2
-kernel_name	L2_norm; Default=l2_norm
-write_file	Set to 1 to write input and output vectors to file; Default=0
-read_inp_file_name	Full filename for reading inputs (order - inp)
-read_ref_file_name	Full filename for reading reference output
-write_inp_file_name	Full filename for writing inputs (order - inp)
-write_out_file_name	Full filename for writing output

Option	Description
-verify	Verify output against provided reference; 0: Disable, 1: Bit exact match; Default=1
-h	Prints help

If no command line arguments are given, the Normalization Kernels sample testbench runs with default values.

6.9 Sample Testbench for GRU Layer

The NN library GRU layer is provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_gru_testbench.c

6.9.1 Usage

The NN library GRU executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_gru_test [options]
```

Following are available options:

Option	Description	Additional Information
--in_feats	Input length (Default=256)	Range: 4-2048 NOTE: Input length must be multiple of 4
--out_feats	Output length (Default=256)	Range: 4-2048 NOTE: Output length must be multiple of 4
--membank_padding	Memory bank padding (Default=1)	Must be 0 or 1
--mat_prec	Coefficient precision (Default=16)	Must be 8 or 16
--vec_prec	Input precision (Default=16)	Must be 16
--verify	Verify output against ref output (Default=1)	Supported values: 0: Disable, 1: Enable
--input_file	Input file name	
--filter_path	Path where file containing filter are stored	
--output_file	File to which output will be written	
--prev_h_file	File containing context data	
--ref_file	File which has ref output	
-help	Prints help	

If no command line arguments are given, the GRU sample testbench runs with default values.

6.10 Sample Testbench for LSTM Layer

The NN library LSTM layer is provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_lstm_testbench.c

6.10.1 Usage

The NN library LSTM executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_lstm_test [options]
```

Following are available options:

Option	Description	Additional Information
--in_feats	Input length (Default=256)	Range: 4-2048 NOTE: Input length must be multiple of 4
--out_feats	Output length (Default=256)	Range: 4-2048 NOTE: Output length must be multiple of 4
--membank_padding	Memory bank padding (Default=1)	Must be 0 or 1
--mat_prec	Coefficient precision (Default=16)	Must be 8 or 16
--vec_prec	Input precision (Default=16)	Must be 16
--verify	Verify output against ref output (Default=1)	Supported values: 0: Disable, 1: Enable
--input_file	File containing input shape	
--filter_path	Path where file containing filter are stored	
--output_file	File to which output will be written	
--output_cell_file	File to which cell output will be written	
--prev_h_file	File containing context (previous output) data	
--prev_c_file	File containing context (previous cell state) data	
--ref_file	File which has ref output	
--ref_cell_file	File which has ref cell output	
-help	Prints help	

If no command line arguments are given, the LSTM sample testbench runs with default values.

6.11 Sample Testbench for CNN Layer

The NN library CNN layer is provided with a sample testbench application. The supplied testbench consists of the following files:

- Testbench source files (test/src)
 - xa_nn_cnn_testbench.c

6.11.1 Usage

The NN Library CNN executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_cnn_test [options]
```

Following are available options:

Option	Description
-input_height	Input height (Default=16)
-input_width	Input width (Default=16)
-input_channels	Input channels (Default=4)
-kernel_height	Kernel height (Default=3)
-kernel_width	Kernel width (Default=3)
-out_channels	Out channels (Default=4)
-channels_multiplier	Channel Multiplier (Default=1)
-x_stride	Stride in width dimension (Default=2)
-y_stride	Stride in height dimension (Default=2)
-x_padding	Left padding in width dimension (Default=2)
-y_padding	Top padding in height dimension (Default=2)
-out_height	Output height (Default=16)
-out_width	Output width (Default=16)
-bias_shift	Bias shift (Default=7)
-acc_shift	Accumulator shift (Default=7)
-out_data_format	Output data format, 0 (SHAPE_CUBE_DWH_T), 1 (SHAPE_CUBE_WHD_T); (Default=0)
-inp_precision	8, 16, -1(single precision float); (Default=16)
-kernel_precision	8, 16, -1(single precision float); (Default=8)

Option	Description
-out_precision	8, 16, -1(single precision float); (Default=16)
-bias_precision	8, 16, -1(single precision float); (Default=16)
-frames	Positive number; (Default=2)
-kernel_name	conv2d_std, conv2d_depth, conv1d_std; (Default=conv2d_std)
-write_file	Set to 1 to write input and output vectors to file; (Default=0)
-read_inp_file_name	Full filename for reading inputs (order - input, kernel, bias, (pointwise kernel, pointwise bias for depth separable))
-read_ref_file_name	Full filename for reading reference output
-write_inp_file_name	Full filename for writing inputs (order - input, kernel, bias, (pointwise kernel, pointwise bias for depth separable))
-write_out_file_name	Full filename for writing output
-verify	Verify output against provided reference; 0: Disable, 1: Bit exact match; Default=1
-help	Prints help

If no command line arguments are given, the CNN sample testbench runs with default values.

6.12 Sample Testbench for ANN Operations

The NN library package is provided with a sample testbench application for the ANN operations. This testbench is based on the test application provided in the Android NN API reference implementation in the Android Open Source Project [3] [4]. It builds and runs the tests given in the reference implementation using the ANN operations provided by the library. The supplied testbench consists of the following files:

- Testbench source files (test/android_nn)
 - runtime/... The test application derived from ANN reference
 - common/... Supporting files for the ANN test application
 - android_deps/... Supporting files for the ANN test application
 - tools/... Supporting files for the ANN test application

6.12.1 Usage

The ANN testbench executable can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_ann_test
```

Currently the testbench does not accept any command line options. The test to run is selected at compile time through a preprocessor definition of testcase identifier. For e.g. defining “HIFI_ADD” selects the ANN testcase for ADD operation.

The file “test/android_nn/runtime/test/generated/all_generated_tests_hifi.cpp” contains the list of all ANN testcase identifiers and testcase specification (model, input and output).

To run a test, the executable should be built with the corresponding test case identifier defined.

6.13 Sample Testbench of TF Micro Lite Example

The NN library package contains a sample testbench of the TF Micro Lite example application. This application is based on the Micro Speech Example provided by Google [2] in the TensorFlow GitHub depository (TensorFlow v2.3). It accepts 1 sec wav audio from the input file and outputs the scores for the twelve classes: silence, unknown, yes, no, up, down, left, right, on, off, stop, go. It uses optimized NN Library kernels for the inference as per the network model specified in the application. The TensorFlow Lite model ‘low_latency_conv’, comprising of one depthwise convolutional layer followed by three fully connected layers is used in this application. The model uses three operations depthwiseConv, FullyConnected and Softmax. The testbench consists of the following files:

- Testbench source files (test/tf_micro_lite)
 - test/src/... The TF Micro Lite application source
 - tensorflow/... Supporting files from the tensorflow package

6.13.1 Usage

The TF Micro Lite example can be run with command-line options as follows.

```
$ xt-run [--mem_model] [--turbo] xa_nn_tfulite_test <input-wav-file.wav>
<test_result.txt>
```

Option	Description
<input-wav-file.wav>	Full filename for reading input wav audio
<test_result.txt>	Output text file for storing the classification score
-help	Shows help

If no command line arguments are given, the sample testbench runs with default values.

7. References

- [1] Reference Wiki page for GRU. https://en.wikipedia.org/wiki/Gated_recurrent_unit

- [2] TF Micro Lite speech recognition example:
https://github.com/tensorflow/tensorflow/tree/r2.3/tensorflow/lite/micro/examples/micro_speech

- [3] Android Neural Networks API.
<https://developer.android.com/ndk/guides/neuralnetworks/>

- [4] Android Neural Networks API reference implementation:
https://android.googlesource.com/platform/frameworks/ml/+refs/tags/android-9.0.0_r35/nn/