## HiSVP

# API Reference

**Issue** 05

**Date** 2019-06-25

## HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road, Bantian,
Longgang District,
Shenzhen 518129 P. R. China

Website: http://www.hisilicon.com/en/

Email: support@hisilicon.com

# About This Document

## Purpose

This document provides reference information including the application programming interfaces (APIs), header files, and error codes for the programmers that develop products or solutions using the smart vision processing (SVP) platform of HiSilicon media processors.

**NOTE**

Unless otherwise stated, Hi3559C V100 and Hi3559A V100 contents are consistent.

Unless otherwise stated, Hi3516A V300, Hi3516D V300, and Hi3516C V500 contents are consistent.

## Related Versions

The following table lists the product versions related to this document.

| Product Name | Version |
| --- | --- |
| Hi3559A | V100 |
| Hi3559C | V100 |
| Hi3519A | V100 |
| Hi3516C | V500 |
| Hi3516D | V300 |
| Hi3559 | V200 |
| Hi3516A | V300 |

## Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

# Symbol Conventions

The symbols that may be found in this document are defined as follows.

| Symbol | Description |
| --- | --- |
|  | Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury. |
|  | Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury. |
|  | Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury. |
|  | Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results.<br>NOTICE is used to address practices not related to personal injury. |
|  | Calls attention to important information, best practices and tips.<br>NOTE is used to address information not related to personal injury, equipment damage, and environment deterioration. |

# Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

## Issue 05 (2019-06-25)

This issue is the fifth official release, which incorporates the following changes:

Section 2.1, a Note part is modified.

In section 3.3, HI_NodePlugin_getNodeType and HI_NodePlugin_Compute are added. The **Example** fields of all APIs are modified.

In section 3.4, HI_NodePlugin_Shape_S, HI_NodePlugin_ElemType_E, HI_NodePlugin_Operand_S, and HI_NodePlugin_NodeParam_S are added.

## Issue 04 (2019-05-20)

This issue is the fourth official release, which incorporates the following changes:

Section 3.5 is modified.

## Issue 03 (2019-04-30)

This issue is the third official release, which incorporates the following changes:

Section 1.2.2 is added.

Section 1.6.2 is modified.

In section 2.4, the descriptions in the **Member** fields of SVP_BLOB_S and SVP_NNIE_NODE_S are modified.

## Issue 02 (2019-03-12)

This issue is the second official release, which incorporates the following changes:

In section 2.1, the description that the NNIE and GDC of Hi3559 V200 are mutually exclusive is added.

In section 2.3, the description in the **Note** field of HI_MPI_SVP_NNIE_LoadModel is modified.

The descriptions in the **Parameter** fields of HI_MPI_SVP_NNIE_Forward and HI_MPI_SVP_NNIE_ForwardWithBbox are modified.

In section 2.4, the descriptions in the **Member** fields of SVP_BLOB_S, SVP_NNIE_NODE_S, SVP_NNIE_FORWARD_CTRL_S, and SVP_NNIE_FORWARD_WITHBBOX_CTRL_S are modified.

In section 3.3, the description in the **Parameter** field of HI_SVPRT_RUNTIME_Init is modified.

Section 3.6.2 is modified.

The support for the Hi3559 V200 chip is added.

## Issue 01 (2018-12-10)

This issue is the first official release, which incorporates the following changes:

In section 1.4, SVP_DSP_HANDLE is added.

In section 2.4, the descriptions in the **Syntax** and **Member** fields of SVP_NNIE_NODE_S are modified. SVP_NNIE_NODE_NAME_LEN is added.

In section 2.5, Table 2-2 is modified.

## Issue 00B09 (2018-11-12)

This issue is the ninth draft release, which incorporates the following changes:

In section 3.3, the descriptions in the **Syntax** and **Parameter** fields of HI_SVPRT_RUNTIME_LoadModelGroupSync, HI_SVPRT_RUNTIME_ForwardGroupSync, HI_SVPRT_RUNTIME_ForwardGroupASync, and HI_SVPRT_RUNTIME_UnloadModelGroup are modified.

In section 3.4, HI_RUNTIME_GROUP_HANDLE is modified. HI_RUNTIME_FORWARD_STATUS_CALLBACK_E, HI_RUNTIME_Forward_Callback, and HI_RUNTIME_Connector_Compute are added.

## Issue 00B08 (2018-10-15)

This issue is the eighth draft release, which incorporates the following changes:

In section 2.3, the descriptions in the **Note** fields of HI_MPI_SVP_NNIE_Forward, HI_MPI_SVP_NNIE_ForwardWithBbox, HI_MPI_SVP_NNIE_AddTskBuf, and HI_MPI_SVP_NNIE_RemoveTskBuf are modified.

In section 2.4, the descriptions in the **Note** fields of SVP_NNIE_FORWARD_CTRL_S and SVP_NNIE_FORWARD_WITHBBOX_CTRL_S are modified.

Chapter 4 is added.

## Issue 00B07 (2018-09-04)

This issue is the seventh draft release, which incorporates the following changes:

Chapter 3 is added.

In section 2.3, HI_MPI_SVP_NNIE_AddTskBuf and HI_MPI_SVP_NNIE_RemoveTskBuf are added.

In section 2.6, the NNIE Proc debugging information is added.

## Issue 00B06 (2018-07-30)

This issue is the sixth draft release, which incorporates the following changes:

In section 1.5, Table 1-1 is modified.

The description of the Hi3556A V100 is added.

## Issue 00B05 (2018-05-20)

This issue is the fifth draft release, which incorporates the following changes:

In section 2.3, the description in the **Parameter** field of HI_MPI_SVP_NNIE_GetTskBufSize is modified.

In section 2.4, the descriptions in the **Member** fields of SVP_NNIE_SEG_S, SVP_NNIE_MODEL_S, and SVP_NNIE_FORWARD_CTRL_S are modified.

## Issue 00B04 (2018-04-28)

This issue is the fourth draft release, which incorporates the following changes:

The contents related to the Hi3519A V100 are added.

## Issue 00B03 (2018-02-10)

This issue is the third draft release, which incorporates the following changes:

In section 1.3, HI_MPI_SVP_DSP_PowerOn and HI_MPI_SVP_DSP_PowerOff are added.

In section 2.3, HI_MPI_SVP_NNIE_CNN_ForwardWithBbox is modified.

## Issue 00B02 (2018-01-29)

This issue is the second draft release, which incorporates the following changes:

In section 1.4, the description in the **Syntax** field of SVP_DSP_MEM_TYPE_E is modified.

Section 2.6.2 is modified.

## Issue 00B01 (2018-01-10)

This issue is the first draft release.

# Contents



# Figures



# 1 DSP

## 1.1 Overview

The SVP platform is a platform for the smart vision heterogeneous acceleration of HiSilicon media processors. The digital signal processor (DSP) is a programmable hardware acceleration module of the SVP platform. Intelligent analysis solutions can be developed based on the DSP to accelerate intelligent analysis and reduce the CPU usage.

**NOTE**

The Hi3516C V500/Hi3516D V300/Hi3559 V200 does not support DSP.

## 1.2 Function Description

### 1.2.1 Important Concepts

- Handle

  When you call the DSP to handle tasks, the system allocates a handle to each task for identification purpose.

- Query

  Based on the handle returned by the system, you can query whether the task corresponding to an operator is complete by calling HI_MPI_SVP_DSP_Query.

### 1.2.2 Module Parameters

- **max_node_num**

  Maximum number of nodes in each priority queue of the DSP. A proper value can help reduce the memory usage of nodes in a queue. The value range is [20, 51]. The default value is **30**.

  Instructions:

  - For Linux, set **max_node_num** when loading **hi35xx_dsp.ko**.
  - For Huawei LiteOS, set **u16NodeNum** in the DSP_init function in the **sdk_init.c** file.

- **dsp_init_mode**

  Initialization mode of the DSP. Set the initialization mode as required. The value can be **0** or **1**.

- The value **0** indicates that the media system can be fully exited only after the MPIs for powering off and disabling the DSP are called. To use the DSP next time, you must power on the DSP, load the DSP image, and enable the DSP.
- The value **1** indicates that you do not need to call the MPIs for powering off and disabling the DSP to fully exit the media system. The DSP image system can remain running. You do not need to reload the DSP image to use the DSP next time, either.

Instructions:

- For Linux, set **dsp_init_mode** when loading **hi35xx_dsp.ko**.
- For Huawei LiteOS, set **u16DspInitMode** in the DSP_init function in the **sdk_init.c** file.

# 1.3 API Reference

The ARM end module of HiSilicon DSP uses API interface operation. The following APIs are provided:

- HI_MPI_SVP_DSP_PowerOn: Powers the DSP on
- HI_MPI_SVP_DSP_PowerOff: Powers the DSP off
- HI_MPI_SVP_DSP_LoadBin: Loads the DSP bin file.
- HI_MPI_SVP_DSP_EnableCore: Enables the DSP core.
- HI_MPI_SVP_DSP_DisableCore: Disables the DSP core.
- HI_MPI_SVP_DSP_RPC: Processes tasks remotely.
- HI_MPI_SVP_DSP_Query: Queries the status of a task.

## HI_MPI_SVP_DSP_PowerOn

[Description]

Powers the DSP on.

[Syntax]

```
HI_S32 HI_MPI_SVP_DSP_PowerOn(SVP_DSP_ID_E enDspId);
```

[Parameter]

| Parameter | Description | Input/Output |
|---|---|---|
| enDspId | DSP ID | Input |

[Return Value]

| Return Value | Description |
|---|---|
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 1.5 "Error Codes." |

[Requirement]

- Header files: **hi_dsp.h** and **mpi_dsp.h**
- Library file: **libdsp.a**

[Note]

Before loading the DSP image, you must call this MPI to power the DSP on. Otherwise, image loading will be suspended.

[Example]

None

[See Also]

None

## HI_MPI_SVP_DSP_PowerOff

[Description]

Powers the DSP off.

[Syntax]

```
HI_S32 HI_MPI_SVP_DSP_PowerOff(SVP_DSP_ID_E enDspId);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| enDspId | DSP ID | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 1.5 "Error Codes." |

[Requirement]

- Header files: **hi_dsp.h** and **mpi_dsp.h**
- Library file: **libdsp.a**

[Note]

None

[Example]

None

[See Also]

None

## HI_MPI_SVP_DSP_LoadBin

[Description]

Loads the DSP bin file.

[Syntax]

```
HI_S32 HI_MPI_SVP_DSP_LoadBin(const HI_CHAR *pszBinFileName,
SVP_DSP_MEM_TYPE_E enMemType);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pszBinFileName | Bin file name.<br>This field cannot be null. | Input |
| enMemType | DSP memory type | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 1.5 "Error Codes." |

[Requirement]

- Header files: **hi_dsp.h** and **mpi_dsp.h**
- Library file: **libdsp.a**

[Note]

Ensure the integrity of the loaded file. Otherwise, the DSP may fail to be started or the system may be suspended.

[Example]

None

[See Also]

None

## HI_MPI_SVP_DSP_EnableCore

[Description]

Enables the DSP core.

[Syntax]

```
HI_S32 HI_MPI_SVP_DSP_EnableCore(SVP_DSP_ID_E enDspId);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| enDspId | DSP ID | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 1.5 "Error Codes." |

[Requirement]

- Header files: **hi_dsp.h** and **mpi_dsp.h**
- Library file: **libdsp.a**

[Note]

None

[Example]

None

[See Also]

None

## HI_MPI_SVP_DSP_DisableCore

[Description]

Disables the DSP core.

[Syntax]

```
HI_S32 HI_MPI_SVP_DSP_DisableCore(SVP_DSP_ID_E enDspId);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| enDspId | DSP ID | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 1.5 "Error Codes." |

[Requirement]

- Header files: **hi_dsp.h** and **mpi_dsp.h**
- Library file: **libdsp.a**

[Note]

None

[Example]

None

[See Also]

None

## HI_MPI_SVP_DSP_RPC

[Description]

Processes tasks remotely.

[Syntax]

```
HI_S32 HI_MPI_SVP_DSP_RPC(SVP_DSP_HANDLE *phHandle, const
SVP_DSP_MESSAGE_S *pstMsg,SVP_DSP_ID_E enDspId, SVP_DSP_PRI_E enPri);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| phHandle | Pointer to the handle<br>This field cannot be null. | Output |
| *pstMsg | Message processing body<br>This field cannot be null. | Input |
| enDspId | DSP ID | Input |
| enPri | Task priority | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |

| Return Value | Description |
| --- | --- |
| Other values | Failure. The return value is an error code. For details, see section 1.5 "Error Codes." |

[Requirement]

- Header files: **hi_dsp.h** and **mpi_dsp.h**
- Library file: **libdsp.a**

[Note]

None

[Example]

None

[See Also]

None

## HI_MPI_SVP_DSP_Query

[Description]

Queries the status of a task.

[Syntax]

```
HI_S32 HI_MPI_SVP_DSP_Query(SVP_DSP_ID_E enDspId,SVP_DSP_HANDLE
hHandle,HI_BOOL *pbFinish,HI_BOOL bBlock);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| enDspId | DSP ID | Input |
| hHandle | Task handle | Input |
| pbFinish | Pointer to the task completion status<br>This field cannot be null. | Output |
| bBlock | Flag indicating whether a task is blocked | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 1.5 "Error Codes." |

[Requirement]

- Header files: **hi_dsp.h** and **mpi_dsp.h**
- Library file: **libdsp.a**

[Note]

None

[Example]

None

[See Also]

None

# 1.4 Data Types and Data Structures

The DSP provides the following data types and data structures:

- SVP_DSP_HANDLE: Defines the DSP handle.
- SVP_DSP_ID_E: Defines the DSP ID.
- SVP_DSP_PRI_E: Defines the DSP priority.
- SVP_DSP_MEM_TYPE_E: Defines the memory type.
- SVP_DSP_CMD_E: Defines the command.
- SVP_DSP_MESSAGE_S: Defines the message format.

## SVP_DSP_HANDLE

[Description]

Defines the DSP handle.

[Syntax]

```
typedef HI_S32 SVP_DSP_HANDLE;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_DSP_HANDLE | DSP handle |

[Note]

None

[See Also]

None

## SVP_DSP_ID_E

[Description]

Defines the DSP ID.

[Syntax]

Hi3559A V100:

```
typedef enum hiSVP_DSP_ID_E
{
    SVP_DSP_ID_0 = 0x0,
    SVP_DSP_ID_1 = 0x1,
    SVP_DSP_ID_2 = 0x2,
    SVP_DSP_ID_3 = 0x3,

    SVP_DSP_ID_BUTT
}SVP_DSP_ID_E;
```

Hi3519A V100/Hi3556A V100:

```
typedef enum hiSVP_DSP_ID_E
{
    SVP_DSP_ID_0 = 0x0,
    SVP_DSP_ID_BUTT
}SVP_DSP_ID_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_DSP_ID_0 | DSP ID 0 |
| SVP_DSP_ID_1 | DSP ID 1 |
| SVP_DSP_ID_2 | DSP ID 2 |
| SVP_DSP_ID_3 | DSP ID 3 |

[Note]

None

[See Also]

None

## SVP_DSP_PRI_E

[Description]

Defines the DSP priority.

[Syntax]

```
typedef enum hiSVP_DSP_PRI_E
{
    SVP_DSP_PRI_0 = 0x0,
    SVP_DSP_PRI_1 = 0x1,
    SVP_DSP_PRI_2 = 0x2,

    SVP_DSP_PRI_BUTT
}SVP_DSP_PRI_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_DSP_PRI_0 | Priority 0 (the highest priority) |
| SVP_DSP_PRI_1 | Priority 1 |
| SVP_DSP_PRI_2 | Priority 2 |

[Note]

None

[See Also]

None

## SVP_DSP_MEM_TYPE_E

[Description]

Defines the memory type.

[Syntax]

Hi3559A V100:

```
typedef enum hiSVP_DSP_MEM_TYPE_E
{
    SVP_DSP_MEM_TYPE_SYS_DDR_DSP_0 = 0x0,
    SVP_DSP_MEM_TYPE_IRAM_DSP_0    = 0x1,
    SVP_DSP_MEM_TYPE_DRAM_0_DSP_0  = 0x2,
    SVP_DSP_MEM_TYPE_DRAM_1_DSP_0  = 0x3,

    SVP_DSP_MEM_TYPE_SYS_DDR_DSP_1 = 0x4,
    SVP_DSP_MEM_TYPE_IRAM_DSP_1    = 0x5,
    SVP_DSP_MEM_TYPE_DRAM_0_DSP_1  = 0x6,
    SVP_DSP_MEM_TYPE_DRAM_1_DSP_1  = 0x7,

    SVP_DSP_MEM_TYPE_SYS_DDR_DSP_2 = 0x8,
    SVP_DSP_MEM_TYPE_IRAM_DSP_2    = 0x9,
```

```
    SVP_DSP_MEM_TYPE_DRAM_0_DSP_2  = 0xA,
    SVP_DSP_MEM_TYPE_DRAM_1_DSP_2  = 0xB,

    SVP_DSP_MEM_TYPE_SYS_DDR_DSP_3 = 0xC,
    SVP_DSP_MEM_TYPE_IRAM_DSP_3    = 0xD,
    SVP_DSP_MEM_TYPE_DRAM_0_DSP_3  = 0xE,
    SVP_DSP_MEM_TYPE_DRAM_1_DSP_3  = 0xF,
    SVP_DSP_MEM_TYPE_BUTT
}SVP_DSP_MEM_TYPE_E;
```

Hi3519A V100/Hi3556A V100:

```
typedef enum hiSVP_DSP_MEM_TYPE_E
{
    SVP_DSP_MEM_TYPE_SYS_DDR_DSP_0 = 0x0,
    SVP_DSP_MEM_TYPE_IRAM_DSP_0    = 0x1,
    SVP_DSP_MEM_TYPE_DRAM_0_DSP_0  = 0x2,
    SVP_DSP_MEM_TYPE_DRAM_1_DSP_0  = 0x3,
    SVP_DSP_MEM_TYPE_BUTT
}SVP_DSP_MEM_TYPE_E;
```

[Member]

| Member | Description |
|---|---|
| SVP_DSP_MEM_TYPE_SYS_DDR_DSP_0 | Address space of system DDR memory used by DSP 0 |
| SVP_DSP_MEM_TYPE_IRAM_DSP_0 | IRAM address space inside DSP 0 |
| SVP_DSP_MEM_TYPE_DRAM_0_DSP_0 | DRAM 0 address space inside DSP 0 |
| SVP_DSP_MEM_TYPE_DRAM_1_DSP_0 | DRAM 1 address space inside DSP 0 |
| SVP_DSP_MEM_TYPE_SYS_DDR_DSP_1 | Address space of system DDR memory used by DSP 1 |
| SVP_DSP_MEM_TYPE_IRAM_DSP_1 | IRAM address space inside DSP 1 |
| SVP_DSP_MEM_TYPE_DRAM_0_DSP_1 | DRAM 0 address space inside DSP 1 |
| SVP_DSP_MEM_TYPE_DRAM_1_DSP_1 | DRAM 1 address space inside DSP 1 |
| SVP_DSP_MEM_TYPE_SYS_DDR_DSP_2 | Address space of system DDR memory used by DSP 2 |
| SVP_DSP_MEM_TYPE_IRAM_DSP_2 | IRAM address space inside DSP 2 |
| SVP_DSP_MEM_TYPE_DRAM_0_DSP_2 | DRAM 0 address space inside DSP 2 |
| SVP_DSP_MEM_TYPE_DRAM_1_DSP_2 | DRAM 1 address space inside DSP 2 |
| SVP_DSP_MEM_TYPE_SYS_DDR_DSP_3 | Address space of system DDR memory used by DSP 3 |

| Member | Description |
| --- | --- |
| SVP_DSP_MEM_TYPE_IRAM_DSP_3 | IRAM address space inside DSP 3 |
| SVP_DSP_MEM_TYPE_DRAM_0_DSP_3 | DRAM 0 address space inside DSP 3 |
| SVP_DSP_MEM_TYPE_DRAM_1_DSP_3 | DRAM 1 address space inside DSP 3 |

[Note]

None

[See Also]

None

## SVP_DSP_CMD_E

[Description]

Defines the command.

[Syntax]

```
typedef enum hiSVP_DSP_CMD_E
{
    SVP_DSP_CMD_INIT      = 0x0,
    SVP_DSP_CMD_EXIT      = 0x1,
    SVP_DSP_CMD_ERODE_3X3  = 0x2,
    SVP_DSP_CMD_DILATE_3X3 = 0x3,

    SVP_DSP_CMD_BUTT
}SVP_DSP_CMD_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_DSP_CMD_INIT | This is a system initialization command. It can be ignored. |
| SVP_DSP_CMD_EXIT | This is a system deinitialization command. It can be ignored. |
| SVP_DSP_CMD_ERODE_3X3 | Erode 3 x 3 command |
| SVP_DSP_CMD_DILATE_3X3 | Dilate 3 x 3 command |

[Note]

To run a user-defined command, the value of SVP_DSP_CMD_BUTT should be increased by 1, so that command does not coincide with a HiSilicon command.

[See Also]

None

### SVP_DSP_MESSAGE_S

[Description]

Defines the message format.

[Syntax]

```
typedef struct hiSVP_DSP_MESSAGE_S
{
    HI_U32      u32CMD;      /**<CMD ID, user-defined*/
    HI_U32      u32MsgId;    /**<Message ID*/
    HI_U64      u64Body;     /**<Message body*/
    HI_U32      u32BodyLen;  /**<Length of pBody*/
} SVP_DSP_MESSAGE_S;
```

[Member]

| Member | Description |
| --- | --- |
| u32CMD | Message command |
| u32MsgId | Message ID |
| u64Body | Message body |
| u32BodyLen | Size of the message body |

[Note]

None

[See Also]

None

## 1.5 Error Codes

Table 1-1 describes the error codes of APIs in the DSP module.

**Table 1-1** Error codes of APIs in the DSP module

| Error Code | Macro Definition | Description |
| --- | --- | --- |
| 0xA0348001 | HI_ERR_SVP_DSP_INVALID_DEVID | The device ID is invalid. |
| 0xA0348002 | HI_ERR_SVP_DSP_INVALID_CHNID | The channel group ID or the region handle is invalid. |
| 0xA0348003 | HI_ERR_SVP_DSP_ILLEGAL_PARAM | The parameter is invalid. |

| Error Code | Macro Definition | Description |
|---|---|---|
| 00xA0348004 | HI_ERR_SVP_DSP_EXIST | The device, channel, or resource to be created already exists. |
| 0xA0348005 | HI_ERR_SVP_DSP_UNEXIST | The device, channel, or resource to be used or destroyed does not exist. |
| 0xA0348006 | HI_ERR_SVP_DSP_NULL_PTR | The pointer is null. |
| 0xA0348007 | HI_ERR_SVP_DSP_NOT_CONFIG | The module is not configured. |
| 0xA0348008 | HI_ERR_SVP_DSP_NOT_SUPPORT | The parameter or function is not supported. |
| 0xA0348009 | HI_ERR_SVP_DSP_NOT_PERM | The operation, for example, modifying the value of a static parameter, is forbidden. |
| 0xA034800C | HI_ERR_SVP_DSP_NOMEM | The memory fails to be allocated for the reasons such as system memory insufficiency. |
| 0xA034800D | HI_ERR_SVP_DSP_NOBUF | The buffer fails to be allocated. The reason may be that the requested image buffer is too large. |
| 0xA034800E | HI_ERR_SVP_DSP_BUF_EMPTY | There is no image in the buffer. |
| 0xA034800F | HI_ERR_SVP_DSP_BUF_FULL | The buffer is full of images. |
| 0xA0348010 | HI_ERR_SVP_DSP_NOTREADY | The system is not initialized or the corresponding module driver is not loaded. |
| 0xA0348011 | HI_ERR_SVP_DSP_BADADDR | The address is invalid. |
| 0xA0348012 | HI_ERR_SVP_DSP_BUSY | The system is busy. |
| 0xA0348040 | HI_ERR_SVP_DSP_SYS_TIMEOUT | The system times out. |
| 0xA0348041 | HI_ERR_SVP_DSP_QUERY_TIMEOUT | The query times out. |
| 0xA0348042 | HI_ERR_SVP_DSP_OPEN_FILE | The file fails to be opened. |
| 0xA0348043 | HI_ERR_SVP_DSP_READ_FILE | The file fails to be read. |

# 1.6 Proc Debugging Information

## 1.6.1 Overview

The debugging information is obtained from the proc file system on Linux. The information reflects the system status and can be used to locate and analyze problems.

[Document Directory]

**/proc/umap**

[Viewing the Information]

- Run a **cat** command such as **cat /proc/umap/dsp** on the console or run file operation commands such as **cp /proc/umap/dsp ./** to copy all the proc files to the current directory.
- Read the preceding files as common read-only files in the application by using functions such as fopen and fread.

**NOTE**

Note the following when reading parameter descriptions:

- For the parameter whose value is **0** or **1**, if the mapping between the values and definitions is not specified, the value **1** indicates affirmative and the value **0** indicates negative.
- For the parameter whose value is **aaa**, **bbb**, or **ccc**, if the mapping between the values and the definitions is not specified, identify the parameter definitions based on **aaa**, **bbb**, or **ccc**.

## 1.6.2 Proc Information

[Debugging Information]

```
# cat /proc/umap/dsp

[DSP] Version: [Hi3559AV100_MPP_V1.0.0.0 B010 Release], Build Time[Apr  8
2019, 09:27:28]

--------------------------------MODULE PARAM--------------------------------
   max_node_num       dsp_init_mode
            30             0

-------------------------------DSP CORE STATUS-----------------------------
   CoreId    Enable
       0       Yes
       1        No

-------------------------------DSP PRI STATUS-----------------------------
   CoreId  Pri    Create
       0    0       Yes
       0    1        No
       0    2        No
       1    0        No
       1    1        No
```

```
       1    2       No

--------------------------------DSP TASK INFO----------------------------
   CoreId  Pri      Hnd   TaskFsh   HndWrap   FreeNum   WorkNum
       0    0        3         3         0        29         1
       0    1        0         0         0        30         0
       0    2        0         0         0        30         0
       1    0        0         0         0        30         0
       1    1        0         0         0        30         0
       1    2        0         0         0        30         0

-----------------------------------DSP RUN-TIME INFO---------------------
CoreId  Pri  CntPerSec  MaxCntPerSec  TotalIntCntLastSec  TotalIntCnt  TCnt
    0    0         2             2                   3            3            0
    0    1         0             0                   0            0            0
    0    2         0             0                   0            0            0
    1    0         0             0                   0            0            0
    1    1         0             0                   0            0            0
    1    2         0             0                   0            0            0

CostTm   MCostTm   CostTmPerSec   MCostTmPerSec  TotalIntCostTm   RunTm
   1         2             3              3              5               3
   0         0             0              0              0               3
   0         0             0              0              0               3
   0         0             0              0              0               3
   0         0             0              0              0               3
   0         0             0              0              0               3

----------------------DSP INVOKE INFO-------------------------

   CoreId       Pri       RPC
       0         0         3
       0         1         0
       0         2         0
       1         0         0
       1         1         0
       1         2         0
```

[Analysis]

Records the work status and resource information about the current DSP, including DSP queue status, task status, runtime status, and calling information.

[Parameter Description]

| Parameter | | Description |
|---|---|---|
| Module parameter | max_node_num | Maximum number of nodes, that is, the maximum number of processing results that can be saved |
| | dsp_init_mode | Initialization mode. The value can be **0** or **1**. |
| DSP task information | CoreId | DSP core ID |
| | Pri | Task priority |
| | Hnd | Handle ID that can be allocated to the current task |
| | TaskFsh | Handle ID of the completed task |
| | HndWrap | Number of times that the user handle ID is wrapped |
| | FreeNum | Number of free nodes |
| | WorkNum | Number of used tasks |
| DSP runtime information | CoreId | DSP core ID |
| | Pri | Task priority |
| | CntPerSec | Number of times that interrupt functions are called in the last second |
| | MaxCntPerSec | Historical maximum number of times that interrupt functions are called in one second |
| | TotalIntCntLastSec | Number of times when interrupts are reported in the last second |
| | TotalIntCnt | Number of times of DSP interrupts |
| | QTCnt | Number of interrupts due to DSP query timeout |
| | CostTm | Time consumed by the last interrupt<br>Unit: μs |
| | MCostTm | Maximum time consumed by an interrupt<br>Unit: μs |
| | CostTmPerSec | Time consumed by the interrupts during the last second<br>Unit: μs |
| | MCostTmPerSec | Maximum time consumed by the interrupts during one second<br>Unit: μs |
| | TotalIntCostTm | Total time of interrupt handling<br>Unit: μs |

| Parameter | | Description |
| --- | --- | --- |
| | RunTm | Total runtime of the DSP<br>Unit: second |
| DSP INVOKE INFO<br>DSP calling information | CoreId | DSP core ID |
| | Pri | Task priority |
| | RPC | Total calling times of the RPC |

[Note]

None

# 2 NNIE

## 2.1 Overview

The neural network inference engine (NNIE) can be found in the HiSilicon chip intelligent analysis system for media processing. Users can develop intelligent analysis solutions based on the NNIE to reduce CPU usage.

**NOTE**

- Hi3556A V100 does not support the NNIE.
- For Hi3559 V200, when the NNIE is deployed in Linux and the GDC is deployed in Huawei LiteOS, the NNIE module cannot be used if the GDC module is loaded.

## 2.2 Function Description

### 2.2.1 Important Concepts

- Network segment

  For some network layers that are not supported by the NNIE, the compiler allows users to segment the network. The network layer nodes unsupported by the NNIE are not be compiled. Users can segment this network part with the CPU. It is strongly recommended that you use the layer supported by the NNIE to implement the network model. A greater number of segments that are not supported by the NNIE increases network fragments and the interaction frequency between software and hardware. The efficiency is effected as well.

- Handle

  When you call the NNIE APIs to create tasks, the system allocates a handle to each task for identification purpose.

- **bInstant** (instant returned result flag)

  **bInstant** is a flag indicating whether to return results in time. If you want to obtain completion information about a task, you must set **bInstant** to **HI_TRUE** when creating this task. If you do not care whether the task is complete, you are recommended to set **bInstant** to **HI_FALSE** so that the tasks can be executed in automatic subsequence to reduce interrupt times and improve performance.

- Query

  Based on the handle returned by the system, you can query whether the task corresponding to an operator is complete by calling HI_MPI_SVP_NNIE_Query.

- Flushing cache

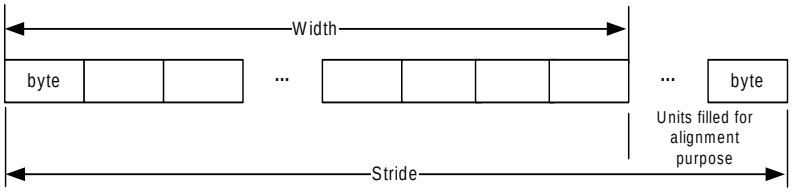
  The NNIE hardware can obtain data only from the DDR. If the CPU has accessed a cacheable space when you call an NNIE task, you need to flush data from the cache to the DDR by calling HI_MPI_SYS_MmzFlushCache. This ensures that the input data and output data of the NNIE are not interfered by the CPU cache. For details about HI_MPI_SYS_MmzFlushCache, see the *HiMPP Vx.y Media Processing Software Development Reference*.

- Stride

  A stride is the valid data bytes in a row plus the invalid data bytes filled for alignment so that the hardware can quickly span to the next row, as shown in Figure 2-1. Note that the variables indicating the number of valid elements stored may differ across data structure rows, and the measurement method is not necessarily the same as that of the stride.

  - The variable indicating the number of valid elements stored in the SVP_BLOB_S row is **width**.
  - The variable indicating the number of valid elements stored in the SVP_SEQ_S row is **dim**.

**Figure 2-1** Stride




- Alignment

  The memory address or memory stride must be a multiple of the alignment coefficient to ensure that hardware can rapidly access the memory start address or access data across rows.

  - Alignment of the start address for the data memory
  - Stride alignment

**NOTICE**

- When Hi3559A V100/Hi3519A V100/Hi3516C V500/Hi3516D V300/Hi3559 V200 uses the DDR4 SDRAM, it is recommended that the start address and stride be 256 bytes aligned to improve the memory access efficiency.
- The stride of the NNIE is measured in bytes. Both stride and width of the IVE are measured in pixels.

- Types of input and output data

  For details, see section 1.4 "Data Types and Data Structures."

- Block data type

  For data types of SVP_BLOB_S, SVP_SRC_BLOB_S, SVP_DST_BLOB_S, see SVP_BLOB_TYPE_E. The memory allocation details are shown in Figure 2-2 to Figure 2-7.

- One-dimensional (1D) data

  SVP_MEM_INFO_S, SVP_SRC_MEM_INFO_S, and SVP_DST_MEM_INFO_S are 1D data, as shown in Figure 2-8.
- Binary large object (BLOB) memory allocation type

**Table 2-1** BLOB memory allocation type

| Type | BLOB Description |
| --- | --- |
| SVP_BLOB_TYPE_S32 | Allocation of the multi-frame signed 32-bit multi-channel data in planar format.<br>In the SVP_BLOB_S structure, **u32Num** indicates the number of frames, **u32Width** indicates the image width, **u32Height** indicates the image height, and **u32Chn** indicates the number of channels of a single frame, as shown in Figure 2-2. |
| SVP_BLOB_TYPE_U8 | Allocation of the multi-frame unsigned 8-bit multi-channel data in planar format.<br>In the SVP_BLOB_S structure, **u32Num** indicates the number of frames, **u32Width** indicates the image width, **u32Height** indicates the image height, and **u32Chn** indicates the number of channels of a single frame, as shown in Figure 2-3. |
| SVP_BLOB_TYPE_YVU420SP | Allocation of the multi-frame YCbCr420 data in semi-planar format.<br>In the SVP_BLOB_S structure, **u32Num** indicates the number of frames, **u32Width** indicates the image width, **u32Height** indicates the image height, and **u32Chn** is **3**, as shown in Figure 2-4. For the chrominance part, the V component is allocated in front of the U component.<br>Note that the height and width must be even numbers. |
| SVP_BLOB_TYPE_YVU422SP | Allocation of the multi-frame YCbCr422 image data in semi-planar format.<br>In the SVP_BLOB_S structure, **u32Num** indicates the number of frames, **u32Width** indicates the image width, **u32Height** indicates the image height, and **u32Chn** is **3**, as shown in Figure 2-5. For the chrominance part, the V component is allocated in front of the U component.<br>Note that the width must be an even number. |
| SVP_BLOB_TYPE_VEC_S32 | Allocation of the multi-frame signed 32-bit vector data.<br>In the SVP_BLOB_S structure, **u32Num** indicates the number of frames, **u32Width** indicates the vector data dimension, **u32Height** indicates the number of vectors (usually, **u32Height** is **1**) in a single frame, and **u32Chn** is **1**, as shown in Figure 2-6. |
| SVP_BLOB_TYPE_SEQ_S32 | Allocation of the multi-segment signed 32-bit sequence data.<br>In the SVP_BLOB_S structure, **u32Num** indicates the number of segments, **u32Dim** indicates the sequence vector data dimension, and **u64VirAddrStep** indicates the array address of the **u32Num** length. The array element indicates the |

| Type | BLOB Description |
| --- | --- |
| | number of vectors in each segment, as shown in Figure 2-7. |

**Figure 2-2** SVP_BLOB_S of SVP_BLOB_TYPE_S32 type (2-channel 2-frame data)

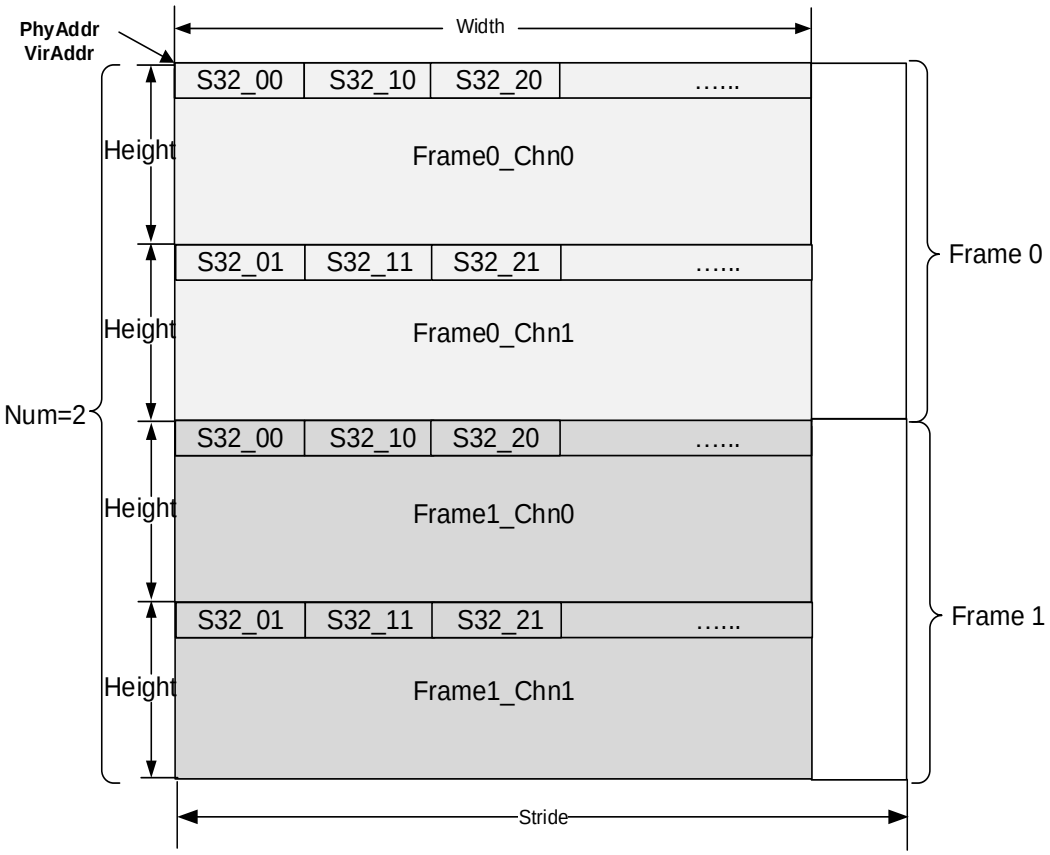
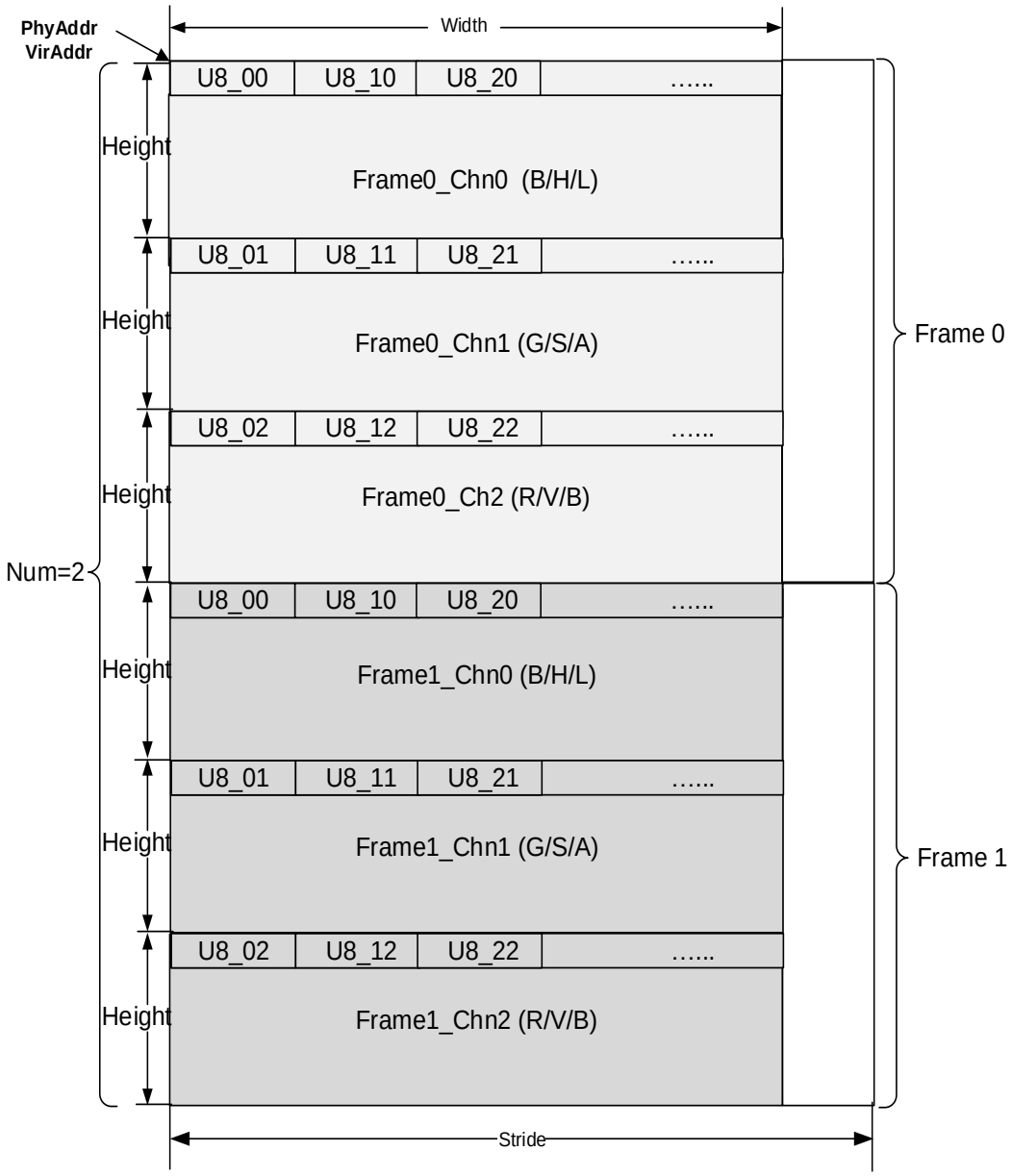


**Figure 2-3** SVP_BLOB_S of SVP_BLOB_TYPE_U8 type (3-channel 2-frame data)

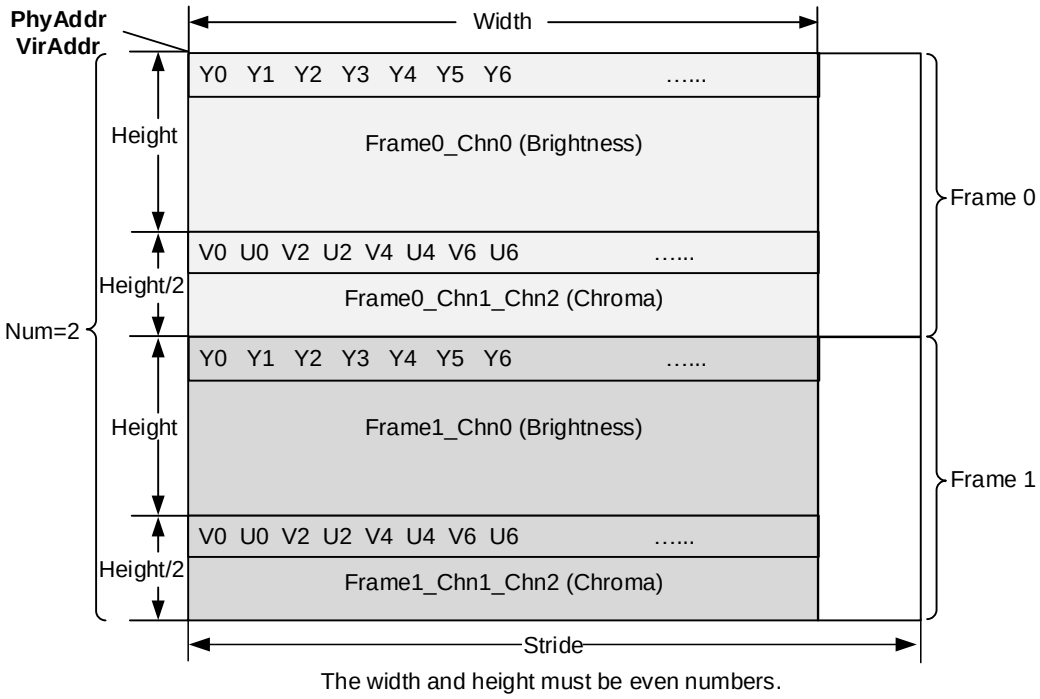



**NOTE**

Typical RGB, HSV, and LAB images are stored in planar format. However, the storage sequence of these images in the NNIE is changed to BGR, HSV, and LAB by default, as shown in Figure 2-3.

**Figure 2-4** SVP_BLOB_S of SVP_BLOB_TYPE_YVU420SP type (2-frame YVU420SP data)

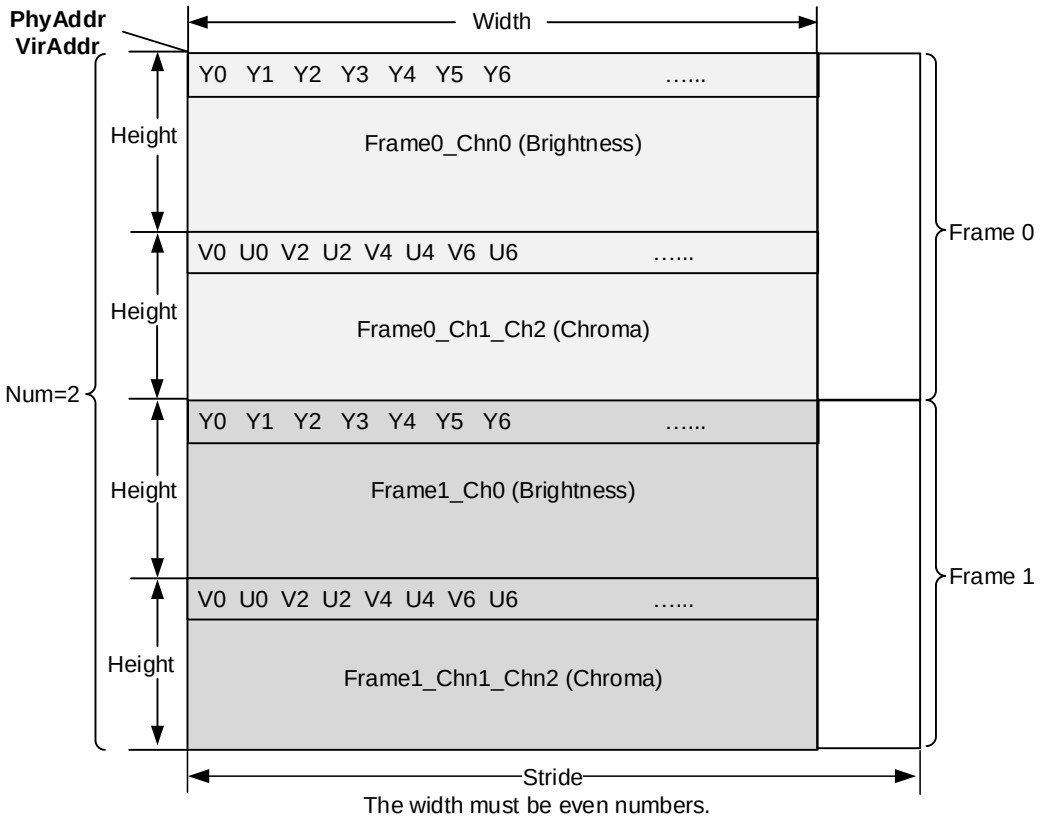



**NOTE**

For the chrominance part, the V component is allocated in front of the U component.

**Figure 2-5** SVP_BLOB_S of SVP_BLOB_TYPE_YVU422SP type (2-frame YVU422SP data)




**NOTE**

For the chrominance part, the V component is allocated in front of the U component.

**Figure 2-6** SVP_BLOB_S of SVP_BLOB_TYPE_VEC_S32 type (2-frame data)

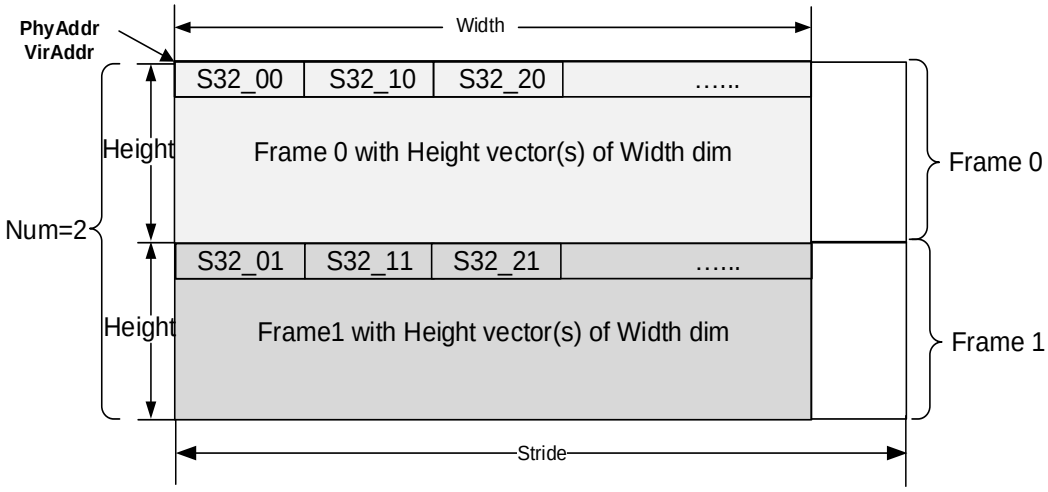


**Figure 2-7** SVP_BLOB_S of SVP_BLOB_TYPE_SEQ_S32 type (Num = *N*)

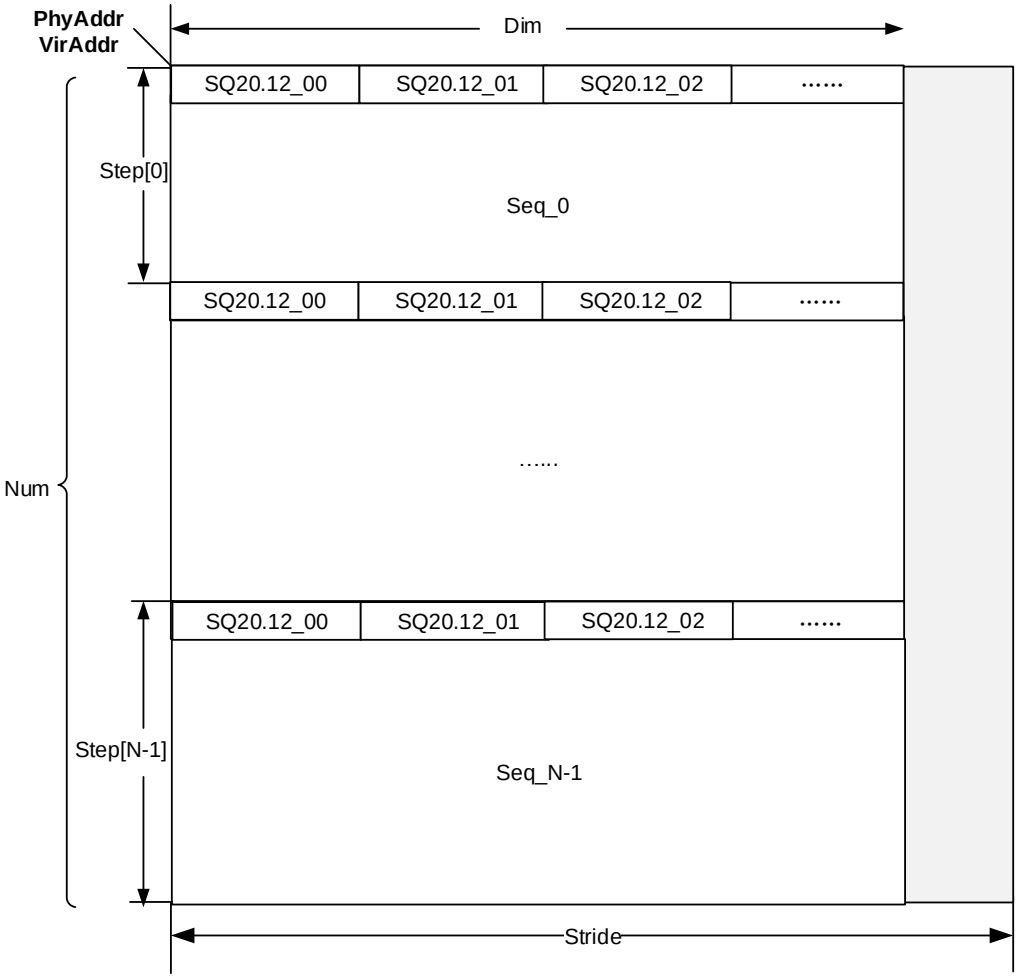
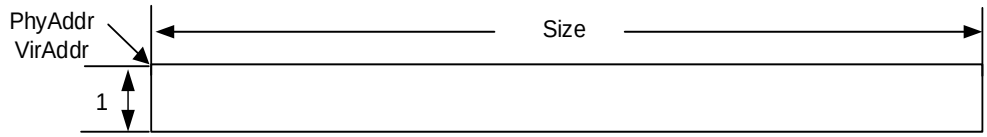



**Figure 2-8** SVP_MEM_INFO_S data memory




## 2.2.2 Instructions

- Call a corresponding operator MPI to create a task, specify the value of **bInstant**, and record the returned handle ID of the task.
- Specify the block mode based on the returned handle ID to query the completion status of a task.

  For details, see the **Example** field of HI_MPI_SVP_NNIE_Query.

# 2.3 API Reference

The NNIE provides the following MPIs for task creation and query:

- HI_MPI_SVP_NNIE_LoadModel: Parses the network model from the model loaded to the buffer by the user in advance.
- HI_MPI_SVP_NNIE_GetTskBufSize: Obtains the size of the auxiliary memory of each segment of a specified network task.
- HI_MPI_SVP_NNIE_Forward: Predicts the CNN-type network with multi-node input and output.
- HI_MPI_SVP_NNIE_ForwardWithBbox: Inputs the multi-node feature map.
- HI_MPI_SVP_NNIE_UnloadModel: Unloads a model.
- HI_MPI_SVP_NNIE_Query: Queries the status of a task.
- HI_MPI_SVP_NNIE_AddTskBuf: Adds TskBuf address information.
- HI_MPI_SVP_NNIE_RemoveTskBuf: Removes the TskBuf address information.

## HI_MPI_SVP_NNIE_LoadModel

[Description]

Parses the network model from the model loaded to the buffer by the user in advance.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_LoadModel (const SVP_SRC_MEM_INFO_S *pstModelBuf,
SVP_NNIE_MODEL_S *pstModel);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pstModelBuf | Buffer of the storage model. The buffer should be allocated in advance and the **wk** file obtained by the NNIE compiler should be loaded to the buffer.<br>This field cannot be null. | Input |
| pstModel | Network model structure | Output |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**

- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

- Ensure the integrity and correctness of the model data stored in the memory pointed to by the address in **pstModelBuf**.
- The memory pointed to by the address in **pstModelBuf** can be released only when the stored model is no longer used and the data in the memory cannot be modified.
- Ensure that the parsed content in **pstModel** is not modified.

[Example]

None

[See Also]

None

## HI_MPI_SVP_NNIE_GetTskBufSize

[Description]

Obtains the size of the auxiliary memory of each segment of a specified network task.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_GetTskBufSize(HI_U32 u32MaxBatchNum, HI_U32
u32MaxBboxNum, const SVP_NNIE_MODEL_S *pstModel, HI_U32 au32TskBufSize[],
HI_U32 u32NetSegNum);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| u32MaxBatchNum | Maximum number of image frames that can be input to the current network. The value must be greater than or equal to the **u32Num** value of **astSrc[]** of HI_MPI_SVP_NNIE_Forward and HI_MPI_SVP_NNIE_ForwardWithBbox.<br>Value range: [1, 256] | Input |
| u32MaxBboxNum | Maximum number of output candidate bounding boxes (Bboxes) when there is the RPN layer on the network. If the network has the RPN layer and the **max_roi_frame_cnt** value is configured in **nnie_mapper**, the **u32MaxBboxNum** value must be less than or equal to the **max_roi_frame_cnt** value. It is recommended that their values be the same. | Input |
| pstModel | Network model structure | Input |
| au32TaskBufSize[] | Auxiliary memory of each segment of a network task | Output |

| Parameter | Description | Input/Output |
| --- | --- | --- |
| u32NetSegNum | Number of segments of a network task, which must match the number of segments in **pstModel**<br>Value range: [1, 8] | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**
- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

- To run one network model for a single thread, you can select either of the following solutions to allocate TskBuf based on the segment mode of the network:
  - For the network segmented in a regular NNIE/non-NNIE (segmented by the CPU or DSP) mode like NNIE→non-NNIE→NNIE→non-NNIE, users can allocate the maximum value for one **au32TskBufSize[]** segment, which can be multiplexed by all segments.
  - For the network with irregular execution intervals where *N* consecutive NNIE segments are found, TskBuf cannot be multiplexed by consecutive NNIE segments. To minimize the memory size, users can allocate the smallest TskBuf for $N - 1$ consecutive NNIE segments and the largest TskBuf for the remaining segments in regular NNIE/non-NNIE execution intervals. For example, on the network segmented in a mode like NNIE→NNIE→non-NNIE→NNIE→non-NNIE, the smallest TskBuf for NNIE→NNIE should be allocated first. Then, the non-NNIE→NNIE→non-NNIE part can multiplex TskBuf of the largest size.
- To run the same network model for multiple threads, independent TskBuf is required by each thread. For details, see the TskBuf allocation methods for running one network model for a single thread.

[Example]

None

[See Also]

None

## HI_MPI_SVP_NNIE_Forward

[Description]

Predicts the CNN-type network with multi-node input and output.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_Forward(SVP_NNIE_HANDLE *phSvpNnieHandle, const
SVP_SRC_BLOB_S astSrc[],const SVP_NNIE_MODEL_S *pstModel, const
SVP_DST_BLOB_S astDst[],const SVP_NNIE_FORWARD_CTRL_S *pstForwardCtrl,
HI_BOOL bInstant);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| phSvpNnieHandle | Pointer to the handle<br>This field cannot be null. | Output |
| astSrc[] | Multi-node input. The sequence of nodes is the same as that defined on the network. Multiple frames can be input at the same time. | Input |
| pstModel | Network model structure | Input |
| astDst[] | Multi-node output from the network segment, including the user-tagged intermediate layer result that needs to be reported and the final result of the network segment | Output |
| pstForwardCtrl | Control structure | Input |
| bInstant | Flag indicating whether results need to be returned instantly | Input |

| Parameter | Supported Type | Address Alignment | Resolution |
| --- | --- | --- | --- |
| astSrc[] | YVU420SP/<br>YVU422SP/<br>U8/S32/<br>VEC_S32/<br>SEQ_S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | The dimension information must be the same as that of **astSrcNode[]** in **pstModel-> astSeg[pstForwardCtrl-> u32NetSegId]**. |
| pstModel | Supported network segment type: CNN/Current. | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | None |

| Parameter | Supported Type | Address Alignment | Resolution |
| --- | --- | --- | --- |
| astDst[] | S32/VEC_S32/SEQ_S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | The dimension information must be the same as that of **astDstNode[]** in **pstModel-> astSeg[pstForwardCtrl-> u32NetSegId]**. |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

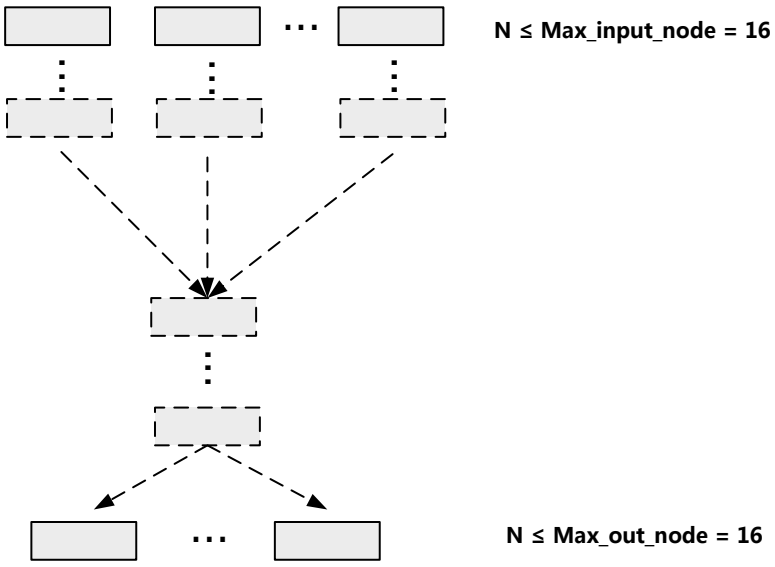
[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**
- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

- Ensure that the data in the memory to which the address in **pstModel -> stBase** points is complete and correct.
- Ensure that the content in the pstModel structure and the data in the memory to which the address in **pstModel -> stBase** points are obtained by parsing the same model file.
- For U8 image input, one channel for gray scale images and three channels for RGB images are supported only.
- When multi-BLOB inputs and outputs are used, the mapping between the BLOBs and the layers is displayed based on the dot diagram generated by the dot description file output from the compiler.

**Figure 2-9** Multi-node input/output network supported by NNIE_Forward




[Example]

None

[See Also]

None

## HI_MPI_SVP_NNIE_ForwardWithBbox

[Description]

Inputs the multi-node feature map.

**astBbox[]** is the Bbox input to several ROIPooling/PSROIPooling layers of the network segment obtained through the RPN layer. The network grades and readjusts the position of the **astBbox[]** input.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_ForwardWithBbox(SVP_NNIE_HANDLE *phSvpNnieHandle,
const SVP_SRC_BLOB_S astSrc[], const SVP_SRC_BLOB_S astBbox[],const
SVP_NNIE_MODEL_S *pstModel, const SVP_DST_BLOB_S astDst[],const
SVP_NNIE_FORWARD_WITHBBOX_CTRL_S *pstForwardCtrl, HI_BOOL bInstant);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| phSvpNnieHandle | Pointer to the handle<br>This field cannot be null. | Output |
| astSrc[] | Multi-node input. The sequence of nodes is the same as that defined on the network. Each node supports only single-frame input. | Input |
| astBbox[] | Bbox input to the network segment. In the BLOB, **Height** indicates the number of Bboxes, and **Width** is **4**. For details, see Figure 2-3. | Input |
| pstModel | Network model structure | Input |
| astDst[] | Multi-node output from the network segment, including scores, Bbox adjustment values, and middle layer output.<br>For the BLOB indicating scores, the class number of **Width** is the same as that of **pstModel**, and **Height** equals **Height** in astBbox. For details, see Figure 2-4.<br>For the BLOB indicating Bbox adjustment values, **Height** equals **Height** in astBbox. For details, see Figure 2-5, Figure 2-6, and the **Note** field.<br>If there is a report of other intermediate layers, the output feature map must be the same as that recorded in the pstModel. For details, see Figure 2-2. | Output |
| pstForwardCtrl | Control structure | Input |
| bInstant | Flag indicating whether results need to be returned instantly | Input |

| Parameter | Supported Type | Address Alignment | Resolution |
| --- | --- | --- | --- |
| astSrc[] | S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | The dimension information must be the same as that of **astSrcNode[]** in **pstModel-> astSeg[pstForwardCtrl-> u32NetSegId]**. |

| Parameter | Supported Type | Address Alignment | Resolution |
| --- | --- | --- | --- |
| astBbox[] | S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | Width = 4<br>Height ≤ 5000 and not greater than **u32MaxBboxNum** transferred when the HI_MPI_SVP_NNIE_GetTskBufSize is called to calculate **TskBufSize** |
| pstModel | Supported network segment type: ROI/PSROI | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | - |
| astDst[] | VEC_S32/S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | The dimension information must be the same as that of **astDstNode[]** in **pstModel-> astSeg[pstForwardCtrl-> u32NetSegId].** |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**
- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

- Ensure that the data in the memory to which the address in **pstModel -> stBase** points is complete and correct.
- Ensure that the content in the pstModel structure and the data in the memory to which the address in **pstModel -> stBase** points are obtained by parsing the same model file.
- The current astBbox[] array supports only one element, that is, pstForwardCtrl→u32ProposalNum = 1, as shown in Figure 2-10.
- The coordinates in astBbox use the SQ20.12 fixed-point input, as shown in Figure 2-11.
- For details about the output score, see Figure 2-12.

- **Bbox_Delta** indicates the output Bbox coordinate adjustment value. According to the training setting, there are three cases with **Bbox_Delta**:
  - Each class has its own **Bbox_Delta**. The output dimension of each **Bbox_Delta** is class_num x 4, as shown in Figure 2-13.
  - All classes share one group of **Bbox_Delta**. The output dimension of each **Bbox_Delta** is **4**, as shown in Figure 2-14.
  - If the background class has its own group of **Bbox_Delta** and the foreground class shares a group of **Bbox_Delta**, the output dimension of each **Bbox_Delta** is **8**, as shown in Figure 2-15.

**Figure 2-10** Multi-node input/output network supported by NNIE_ForwardWithBbox

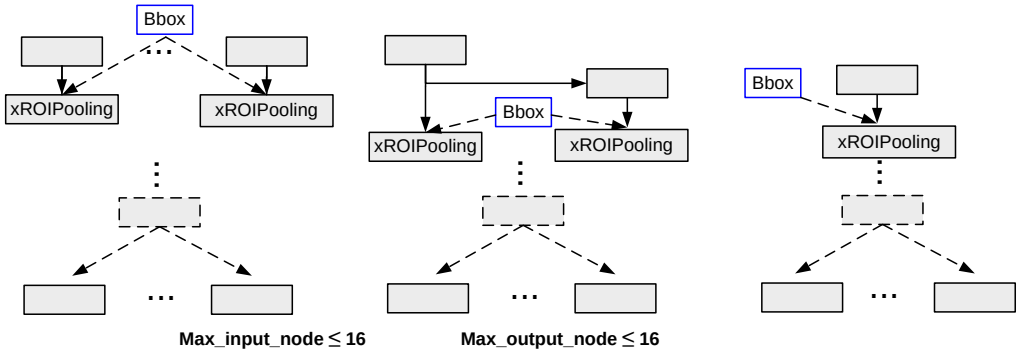



**Figure 2-11** NNIE_ForwardWithBbox astBbox[i] input diagram

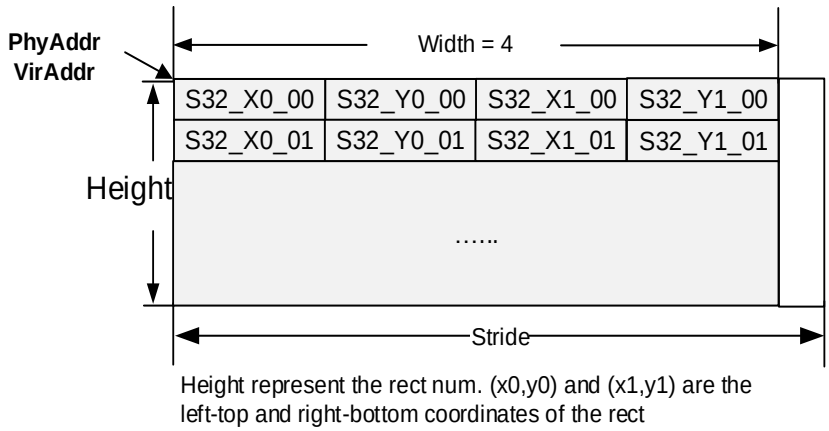


**Figure 2-12** NNIE_ForwardWithBbox score output diagram

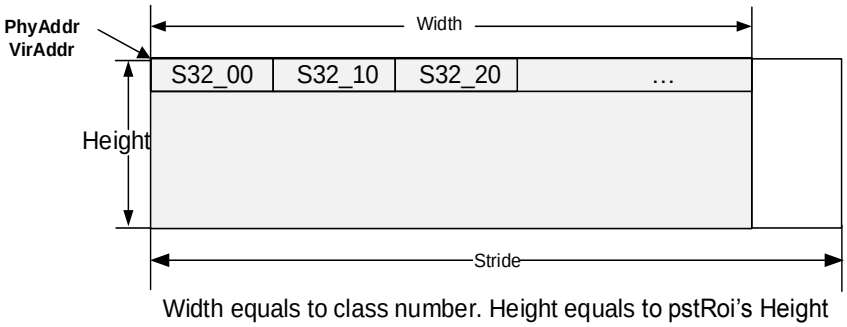



**Figure 2-13** Diagram A of NNIE_ForwardWithBbox Bbox adjustment value

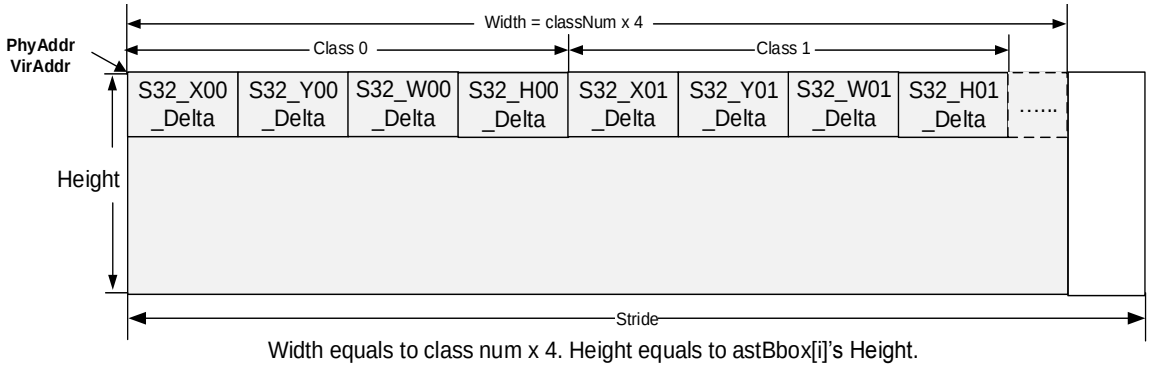



**Figure 2-14** Diagram B of NNIE_ForwardWithBbox Bbox adjustment value

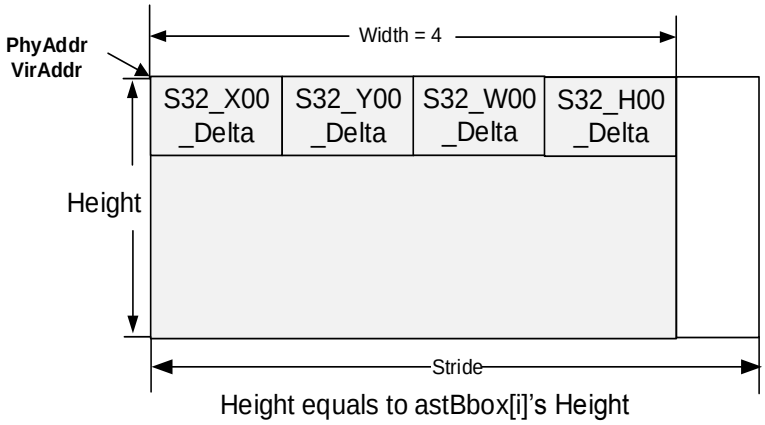
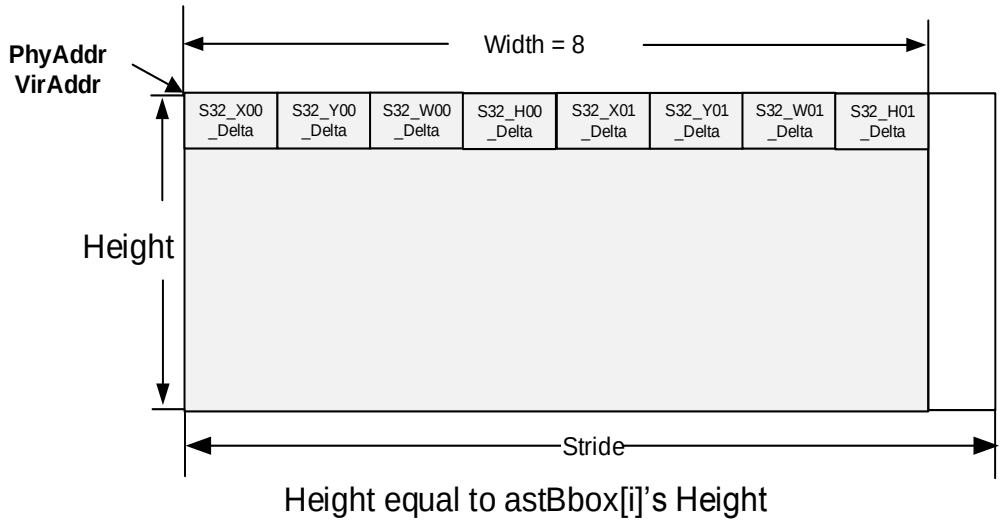


**Figure 2-15** Diagram C of NNIE_ForwardWithBbox Bbox adjustment value




[Example]

None

[See Also]

None

## HI_MPI_SVP_NNIE_UnloadModel

[Description]

Unloads the CNN model.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_UnloadModel(SVP_NNIE_MODEL_S *pstModel);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pstModel | Network model structure | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**
- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

None

[Example]

None

[See Also]

None

## HI_MPI_SVP_NNIE_Query

[Description]

Queries the status of a task.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_Query(SVP_NNIE_ID_E enNnieId,SVP_NNIE_HANDLE
svpNnieHandle,HI_BOOL *pbFinish,HI_BOOL bBlock);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| enNnieId | Flag indicating the NNIE core running for the task | Input |
| svpNnieHandle | Handle | Input |
| pbFinish | Completion flag | Output |
| bBlock | Flag indicating whether a task is blocked | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**
- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

None

[Example]

None

[See Also]

None

## HI_MPI_SVP_NNIE_AddTskBuf

[Description]

Adds the TskBuf address information.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_AddTskBuf(const SVP_MEM_INFO_S* pstTskBuf);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pstTskBuf | Pointer to TskBuf<br>This field cannot be null. | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**
- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

- The TskBuf address information is added to reduce the memory mapping count in kernel mode and improve efficiency.
- The TskBuf address information is managed through the linked list. The default length of the linked list is 32. The length can be configured by using the module parameter **nnie_max_tskbuf_num**.
- If the address information is not added to the system by calling HI_MPI_SVP_NNIE_AddTskBuf, the TskBuf virtual address in kernel mode will be mapped or unmapped each time HI_MPI_SVP_NNIE_Forward or HI_MPI_SVP_NNIE_ForwardWithBbox is called, reducing the efficiency.
- This MPI must be used together with HI_MPI_SVP_NNIE_RemoveTskBuf.
- You are advised to call this MPI to add the TskBuf address information required by HI_MPI_SVP_NNIE_Forward and HI_MPI_SVP_NNIE_ForwardWithBbox to the

system. If the TskBuf address is no longer required, you can call HI_MPI_SVP_NNIE_RemoveTskBuf to remove the TskBuf address information. Add the TskBuf address information during initialization. Then, the TskBuf address information can be used directly. Remove it when it is no longer required.

- When **pstTskBuf** -> **u64VirAddr** is not used, the parameter exception check is not performed.
- The value of **pstTskBuf** -> **u32Size** cannot be **0**.
- The TskBuf memory is released by users. The recorded TskBuf can be released only after being removed.

[Example]

None

[See Also]

- HI_MPI_SVP_NNIE_GetTskBufSize
- HI_MPI_SVP_NNIE_Forward
- HI_MPI_SVP_NNIE_ForwardWithBbox
- HI_MPI_SVP_NNIE_RemoveTskBuf

## HI_MPI_SVP_NNIE_RemoveTskBuf

[Description]

Removes the TskBuf address information.

[Syntax]

```
HI_S32 HI_MPI_SVP_NNIE_RemoveTskBuf(const SVP_MEM_INFO_S* pstTskBuf);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pstTskBuf | Pointer to TskBuf<br>This field cannot be null. | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 2.5 "Error Codes." |

[Requirement]

- Header files: **hi_comm_svp.h**, **hi_nnie.h** and **mpi_nnie.h**
- Library file: **libnnie.a** (**nniefc1.1.lib** or **nnieit1.1.lib** for simulation on the PC)

[Note]

- If TskBuf is no longer required, the added TskBuf address information needs to be removed from the linked list.
- This MPI must be used together with HI_MPI_SVP_NNIE_AddTskBuf.
- When **pstTskBuf** -> **u64VirAddr** is not used, the parameter exception check is not performed.
- The value of **pstTskBuf** -> **u32Size** cannot be **0**.
- The TskBuf memory is released by users. The recorded TskBuf can be released only after being removed.

[Example]

None

[See Also]

- HI_MPI_SVP_NNIE_GetTskBufSize
- HI_MPI_SVP_NNIE_Forward
- HI_MPI_SVP_NNIE_ForwardWithBbox
- HI_MPI_SVP_NNIE_AddTskBuf

# 2.4 Data Structures

The NNIE provides the following data structures:

## Fixed-Point Data Types

[Description]

Defines the types of fixed-point data.

[Syntax]

```
typedef unsigned char          HI_U0Q8;
typedef unsigned char          HI_U1Q7;
typedef unsigned char          HI_U5Q3;
typedef unsigned short         HI_U0Q16;
typedef unsigned short         HI_U4Q12;
typedef unsigned short         HI_U6Q10;
typedef unsigned short         HI_U8Q8;
typedef unsigned short         HI_U14Q2;
typedef unsigned short         HI_U12Q4;
typedef short                 HI_S14Q2;
typedef short                 HI_S9Q7;
typedef unsigned int           HI_U22Q10;
typedef unsigned int           HI_U25Q7;
typedef int                    HI_S25Q7;
typedef int                    HI_S20Q12;
typedef unsigned short         HI_U8Q4F4;   /*8-bit unsigned integer, 4-
bit decimal fraction, 4-bit flag*/
```

[Member]

| Member | Description |
|---|---|
| HI_U0Q8 | No bit expresses the integral part, and 8 bits express the decimal part. This type is called UQ0.8 in this document. |
| HI_U1Q7 | The upper 1-bit unsigned data expresses the integral part, and lower 7 bits express the decimal part. This type is represented by UQ1.7 in this document. |
| HI_U5Q3 | The upper 5-bit unsigned data expresses the integral part, and lower 3 bits express the decimal part. This type is represented by UQ5.3 in this document. |
| HI_U0Q16 | No bit expresses the integral part, and 16 bits express the decimal part. This type is represented by UQ0.16 in this document. |
| HI_U4Q12 | The upper-4-bit unsigned data expresses the integral part, and lower 12 bits express the decimal part. This type is represented by UQ4.12 in this document. |
| HI_U6Q10 | The upper 6 bits unsigned data expresses the integral part, and lower 10 bits express the decimal part. This type is represented by UQ6.10 in this document. |
| HI_U8Q8 | The upper-8-bit unsigned data expresses the integral part, and lower 8 bits express the decimal part. This type is represented by UQ8.8 in this document. |
| HI_U14Q2 | The upper-14-bit unsigned data expresses the integral part, and lower 2 bits express the decimal part. This type is represented by UQ14.2 in this document. |
| HI_U12Q4 | The upper-12-bit unsigned data expresses the integral part, and lower 4 bits express the decimal part. This type is represented by UQ12.4 in this document. |
| HI_S14Q2 | The upper-14-bit signed data expresses the integral part, and lower 2 bits express the decimal part. This type is represented by SQ14.2 in this document. |
| HI_S9Q7 | The upper-9-bit signed data expresses the integral part, and lower 7 bits express the decimal part. This type is represented by SQ9.7 in this document. |
| HI_U22Q10 | The upper-22-bit unsigned data expresses the integral part, and lower 10 bits express the decimal part. This type is represented by UQ22.10 in this document. |
| HI_U25Q7 | The upper-25-bit unsigned data expresses the integral part, and lower 7 bits express the decimal part. This type is represented by UQ25.7 in this document. |
| HI_S25Q7 | The upper-25-bit signed data expresses the integral part, and lower 7 bits express the decimal part. This type is represented by SQ25.7 in this document. |

| Member | Description |
| --- | --- |
| HI_S20Q12 | The upper-20-bit signed data expresses the integral part, and lower 12 bits express the decimal part. This type is represented by SQ20.12 in this document. |
| HI_U8Q4F4 | The upper-8-bit unsigned data expresses the integral part, and middle 4 bits express the decimal part, and the lower four bits express the flag. This type is represented by UQF8.4.4 in this document. |

[Note]

HI_UxQyFz/HI_SxQy:

- The variable *x* after U indicates that *x*-bit unsigned data expresses the integral part.
- The variable *x* after S indicates that *x*-bit signed data expresses the integral part.
- The variable *y* after Q indicates that *y*-bit data expresses the decimal part.
- The variable *z* after F indicates that *z*-bit data expresses the flag.
- The bits are upper bits to lower bits from left to right.

[See Also]

None

## SVP_NNIE_HANDLE

[Description]

Defines the NNIE handle.

[Syntax]

```
typedef HI_S32 SVP_NNIE_HANDLE;
```

[Member]

None

[Note]

None

[See Also]

None

## SVP_MEM_INFO_S

[Description]

Defines 1D memory information.

[Syntax]

```
typedef struct hiSVP_MEM_INFO_S
{
    HI_U64  u64PhyAddr; /* RW;The physical address of the memory */
```

```
    HI_U64  u64VirAddr; /* RW;The virtual address of the memory */
    HI_U32  u32Size;    /* RW;The size of memory */
}SVP_MEM_INFO_S;
```

[Member]

| Member | Description |
| --- | --- |
| u64VirAddr | Virtual address of the memory block |
| u64PhyAddr | Physical address of the memory block |
| u32Size | Byte count of memory blocks. See Figure 2-8. |

[Note]

None

[See Also]

- SVP_SRC_MEM_INFO_S
- SVP_DST_MEM_INFO_S

## SVP_SRC_MEM_INFO_S

[Description]

Defines the source sequence.

[Syntax]

```
typedef SVP_MEM_INFO_S SVP_SRC_MEM_INFO_S;
```

[Member]

None

[Note]

None

[See Also]

- SVP_MEM_INFO_S
- SVP_DST_MEM_INFO_S

## SVP_DST_MEM_INFO_S

[Description]

Defines the output sequence.

[Syntax]

```
typedef SVP_MEM_INFO_S SVP_DST_MEM_INFO_S;
```

[Member]

None

[Note]

None

[See Also]

- SVP_MEM_INFO_S
- SVP_SRC_MEM_INFO_S

## SVP_BLOB_TYPE_E

[Description]

Defines the data memory allocation of the BLOB.

[Syntax]

```
typedef enum hiSVP_BLOB_TYPE_E
{
    SVP_BLOB_TYPE_S32      =  0x0,
    SVP_BLOB_TYPE_U8       =  0x1,
    /*channel = 3*/
    SVP_BLOB_TYPE_YVU420SP  =  0x2,
    /*channel = 3*/
    SVP_BLOB_TYPE_YVU422SP  =  0x3,
    SVP_BLOB_TYPE_VEC_S32   =  0x4,
    SVP_BLOB_TYPE_SEQ_S32   =  0x5,
    SVP_BLOB_TYPE_BUTT
}SVP_BLOB_TYPE_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_BLOB_TYPE_S32 | BLOB data element (S32). For details, see Figure 2-2. |
| SVP_BLOB_TYPE_U8 | BLOB data element (U8). For details, see Figure 2-3. |
| SVP_BLOB_TYPE_YVU420SP | BLOB data memory allocation (YVU420SP). For details, see Figure 2-4. |
| SVP_BLOB_TYPE_YVU422SP | BLOB data memory allocation (YVU422SP). For details, see Figure 2-5. |
| SVP_BLOB_TYPE_VEC_S32 | Storage vector in the BLOB. Each element is of the S32 type. For details, see Figure 2-6. |
| SVP_BLOB_TYPE_SEQ_S32 | Storage sequence in the BLOB. The data element is of the S32 type. For details, see Figure 2-7. |

[Note]

None

[See Also]

SVP_BLOB_S

## SVP_BLOB_S

[Description]

Defines the information of multiple consecutive BLOBs.

[Syntax]

```
typedef struct hiSVP_BLOB_S
{
    SVP_BLOB_TYPE_E enType;     /*BLOB type*/
    HI_U32 u32Stride;          /*Stride, a line bytes num*/
    HI_U64 u64VirAddr;         /*virtual addr*/
    HI_U64 u64PhyAddr;         /*physical addr*/
   HI_U32     u32Num;        /*N: frame num or sequence num,correspond to
caffe blob's n*/
   union
   {
       struct
       {
          HI_U32 u32Width;    /*W: frame width, correspond to caffe blob's
w*/
          HI_U32 u32Height;   /*H: frame height, correspond to caffe
blob's h*/
          HI_U32 u32Chn;     /*C: frame channel,correspond to caffe
blob's c*/
       }stWhc;
       struct
       {
          HI_U32 u32Dim;         /*D: vector dimension*/
          HI_U64 u64VirAddrStep;  /*T: virtual address of   time steps
array in each sequence*/
       }stSeq;
   }unShape;
}SVP_BLOB_S;
```

[Member]

| Member | Description |
| --- | --- |
| enType | BLOB type<br>Value range: [SVP_BLOB_TYPE_S32, SVP_BLOB_TYPE_BUTT) |

| Member | Description |
| --- | --- |
| u32Stride | Number of aligned bytes of single-line data in the BLOB |
| u64VirAddr | Start virtual address of the BLOB |
| u64PhyAddr | Start physical address of the BLOB |
| u32Num | Number of consecutive memory blocks. If one frame of data corresponds to one block, then there are **u32Num** frames in the blob.<br>When **enType** is set to **YVU420SP**, **YVU422SP**, **U8**, or **SEQ_S32**, the value range is [1, 256].<br>When **enType** is set to **S32** or **VEC_S32**, the value range is [1, 5000]. |
| u32Width | BLOB width<br>When **enType** is set to **YVU420SP**, **YVU422SP**, or **U8**, the value range is [8, 4096].<br>When **enType** is set to **S32**, the value range is [1, 0xFFFFFFFF].<br>When **enType** is set to **VEC_S32**, the value range is [1, 0xFFFFFFFF]. |
| u32Height | BLOB height<br>When **enType** is set to **YVU420SP**, **YVU422SP**, or **U8**, the value range is [8, 4096].<br>When **enType** is set to **S32**, the value range is [1, 0xFFFFFFFF].<br>When **enType** is set to **VEC_S32**, the value range is [1, 1]. |
| u32Chn | Number of BLOB channels<br>When **enType** is set to **YVU420SP** or **YVU422SP**, the value is **3**.<br>When **enType** is set to **U8**, the value is **1** or **3**.<br>When enType is set to **S32**, the value range is [1, 0xFFFFFFFF].<br>When **enType** is set to **VEC_S32**, the value range is [1, 1]. |
| u32Dim | Vector length. It is used only to indicate the RNN/LSTM data.<br>When **enType** is set to **SEQ_S32**, the value range is [1, 0xFFFFFFFF]. |
| u64VirAddrStep | Array address. The array element indicates the number of vectors in each sequence. |

[Note]

- Table 2-2 describes the data memory blocks used to indicate memory shapes in the Caffe.

**Table 2-2** Data memory blocks indicating memory shapes in the Caffe

| Data Block | Dim0 | Dim1 | Dim2 | Dim3 |
|---|---|---|---|---|
| Image/feature map | N | C | H | W |
| FC (normal) vector | N | C | - | - |
| RNN/LSTM vector | T | N | D | - |

- Table 2-3 describes the mapping between the data memory blocks and the BLOBs discussed in this document.

**Table 2-3** Mapping between the data memory blocks and the BLOBs

| Data Block | Dim0 | Dim1 | Dim2 | Dim3 |
|---|---|---|---|---|
| Image/feature map | u32Num | 32Chn | u32Height | u32Width |
| FC (normal) vector | u32Num | u32Width | - | - |
| RNN/LSTM vector | u64VirAddrStep[i] | u32Num | u32Dim | - |

- **u32Stride** indicates the number of bytes after data alignment in both the **u32Width** and **u32Dim** directions.
- When **enType** is set to **S32**, the value range of the product result of (u32Chn x u32Height x u32Stride) is [1, 0xFFFFFFFF].
- When **enType** is set to **SEQ_S32** or **VEC_S32**, the value range of **u32Stride** is [1, 0xFFFFFFFF].

[See Also]

SVP_BLOB_TYPE_E

## SVP_SRC_BLOB_S

[Description]

Defines the source sequence.

[Syntax]

```
typedef SVP_BLOB_S  SVP_SRC_BLOB_S;
```

[Member]

None

[Note]

None

[See Also]

- SVP_BLOB_S
- SVP_DST_BLOB_S

## SVP_DST_BLOB_S

[Description]

Defines the output sequence.

[Syntax]

```
typedef SVP_BLOB_S  SVP_DST_BLOB_S;
```

[Member]

None

[Note]

None

[See Also]

- SVP_BLOB_S
- SVP_SRC_BLOB_S

## SVP_NNIE_ID_E

[Description]

Defines the enumeration type of the NNIE hardware IDs.

[Syntax]

Hi3559A V100:

```
typedef enum hiSVP_NNIE_ID_E
{
    SVP_NNIE_ID_0      = 0x0,
    SVP_NNIE_ID_1      = 0x1,
    SVP_NNIE_ID_BUTT
}SVP_NNIE_ID_E;
```

Hi3519A V100/Hi3516C V500/Hi3516D V300/Hi3559 V200:

```
typedef enum hiSVP_NNIE_ID_E
{
    SVP_NNIE_ID_0      = 0x0,
    SVP_NNIE_ID_BUTT
}SVP_NNIE_ID_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_NNIE_ID_0 | NNIE whose subscript is 0 |

| Member | Description |
| --- | --- |
| SVP_NNIE_ID_1 | NNIE whose subscript is 1 |

[Note]

None

[See Also]

None

## SVP_NNIE_RUN_MODE_E

[Description]

Defines the running mode.

[Syntax]

```
typedef enum hiSVP_NNIE_RUN_MODE_E
{
    SVP_NNIE_RUN_MODE_CHIP     = 0x0,  /* on SOC chip running */
    SVP_NNIE_RUN_MODE_FUNC_SIM  = 0x1,  /* functional simulation */

    SVP_NNIE_RUN_MODE_BUTT
}SVP_NNIE_RUN_MODE_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_NNIE_RUN_MODE_CHIP | The NNIE can only run on the chip. |
| SVP_NNIE_RUN_MODE_FUNC_SIM | The NNIE can only be used for the function emulation on the PC. |

[Note]

None

[See Also]

None

## SVP_NNIE_NET_TYPE_E

[Description]

Defines the network type.

[Syntax]

```
typedef enum hiSVP_NNIE_NET_TYPE_E
```

```
{
    SVP_NNIE_NET_TYPE_CNN       = 0x0, /* Normal cnn net */
    SVP_NNIE_NET_TYPE_ROI       = 0x1, /* With ROI input cnn net*/
    SVP_NNIE_NET_TYPE_RECURRENT = 0x2, /* RNN or LSTM net */

    SVP_NNEI_NET_TYPE_BUTT
}SVP_NNIE_NET_TYPE_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_NNIE_NET_TYPE_CNN | Common CNN/DNN network type |
| SVP_NNIE_NET_TYPE_ROI | Network type that contains the output box information, active box information and confidence regression of the RPN layer. |
| SVP_NNIE_NET_TYPE_RECURRENT | Cyclic neural network type |

[Note]

None

[See Also]

None

## SVP_NNIE_ROIPOOL_TYPE_E

[Description]

Defines the ROIPooling type.

[Syntax]

```
typedef enum hiSVP_NNIE_ROIPOOL_TYPE_E
{
    SVP_NNIE_ROIPOOL_TYPE_NORMAL = 0x0,     /* Roipooling*/
    SVP_NNIE_ROIPOOL_TYPE_PS     = 0x1,     /* Position-Sensitive
roipooling */

    SVP_NNIE_ROIPOOL_TYPE_BUTT
}SVP_NNIE_ROIPOOL_TYPE_E;
```

[Member]

| Member | Description |
| --- | --- |
| SVP_NNIE_ROIPOOL_TYPE_NORMAL | Normal ROIPooling |
| SVP_NNIE_ROIPOOL_TYPE_PS | Position-sensitive ROIPooling |

[Note]

None

[See Also]

None

## SVP_NNIE_NODE_S

[Description]

Defines the network input/output node type.

[Syntax]

```
typedef struct hiSVP_NNIE_NODE_S
{
    SVP_BLOB_TYPE_E  enType;
    union
    {
        struct
        {
            HI_U32 u32Width;
            HI_U32 u32Height;
            HI_U32 u32Chn;
        }stWhc;
        HI_U32 u32Dim;
    }unShape;
    HI_U32 u32NodeId;
    HI_CHAR szName[SVP_NNIE_NODE_NAME_LEN];
}SVP_NNIE_NODE_S;
```

[Member]

| Member | Description |
| --- | --- |
| enType | Node type |
| u32Width | Width of the node memory<br>When **enType** is set to **YVU420SP**, **YVU422SP**, or **U8**, the value range is [8, 4096].<br>When **enType** is set to **S32**, the value range is [1, 0xFFFFFFFF].<br>When **enType** is set to **VEC_S32**, the value range is [1, 0xFFFFFFFF]. |

| Member | Description |
| --- | --- |
| u32Height | Height of the node memory<br>When **enType** is set to **YVU420SP**, **YVU422SP**, or **U8**, the value range is [8, 4096].<br>When **enType** is set to **S32**, the value range is [1, 0xFFFFFFFF].<br>When **enType** is set to **VEC_S32**, the value range is [1, 1]. |
| u32Chn | Channel count of the node memory<br>When **enType** is set to **YVU420SP** or **YVU422SP**, the value is **3**.<br>When **enType** is set to **U8**, the value is **1** or **3**.<br>When **enType** is set to **S32**, the value range is [1, 0xFFFFFFFF].<br>When **enType** is set to **VEC_S32**, the value range is [1, 1]. |
| u32Dim | Vector dimension of the node memory<br>When **enType** is set to **SEQ_S32**, the value range is [1, 0xFFFFFFFF]. |
| u32NodeId | Node ID on the network |
| szName | Node name |

[Note]

None

[See Also]

SVP_NNIE_SEG_S

## SVP_NNIE_NODE_NAME_LEN

[Description]

Defines the length of a node name.

[Syntax]

```
#define SVP_NNIE_NODE_NAME_LEN     32 /*NNIE node name length*/
```

[Member]

None

[Note]

None

[See Also]

None

## SVP_NNIE_MAX_NET_SEG_NUM

[Description]

Defines the maximum number of network segments.

[Syntax]

```
#define SVP_NNIE_MAX_NET_SEG_NUM     8 /*NNIE max segment num that the net being cut into*/
```

[Member]

None

[Note]

None

[See Also]

None

## SVP_NNIE_MAX_INPUT_NUM

[Description]

Defines the maximum number of network input nodes.

[Syntax]

```
#define SVP_NNIE_MAX_INPUT_NUM           16 /*NNIE max input num in each seg*/
```

[Member]

None

[Note]

None

[See Also]

None

## SVP_NNIE_MAX_OUTPUT_NUM

[Description]

Defines the maximum number of network output nodes.

[Syntax]

```
#define SVP_NNIE_MAX_OUTPUT_NUM          16 /*NNIE max output num in each seg*/
```

[Member]

None

[Note]

None

[See Also]

None

## SVP_NNIE_MAX_ROI_LAYER_NUM_OF_SEG

[Description]

Defines the maximum number of ROIPooling and PSROIPooling layers in a network segment.

[Syntax]

```
#define SVP_NNIE_MAX_ROI_LAYER_NUM_OF_SEG   2  /*NNIE max roi layer num in each seg*/
```

[Member]

None

[Note]

None

[See Also]

None

## SVP_NNIE_MAX_ROI_LAYER_NUM

[Description]

Defines the maximum number of ROIPooling and PSROIPooling layers on the network.

[Syntax]

```
#define SVP_NNIE_MAX_ROI_LAYER_NUM         4  /*NNIE max roi layer num*/
```

[Member]

None

[Note]

None

[See Also]

None

## SVP_NNIE_SEG_S

[Description]

Defines the network segment structure.

[Syntax]

```
typedef struct hiSVP_NNIE_SEG_S
{
    SVP_NNIE_NET_TYPE_E enNetType;
    HI_U16              u16SrcNum;
    HI_U16              u16DstNum;
```

```
    HI_U16              u16RoiPoolNum;
    HI_U16              u16MaxStep;
    HI_U32              u32InstOffset;
    HI_U32              u32InstLen;
    SVP_NNIE_NODE_S     astSrcNode[SVP_NNIE_MAX_INPUT_NUM];
    SVP_NNIE_NODE_S     astDstNode[SVP_NNIE_MAX_OUTPUT_NUM];
    HI_U32              au32RoiIdx[SVP_NNIE_MAX_ROI_LAYER_NUM_OF_SEG];
/*Roipooling info index*/
}SVP_NNIE_SEG_S;
```

[Member]

| Member | Description |
| --- | --- |
| enNetType | Network type |
| u16SrcNum | Number of input nodes in a network segment<br>Value range: [1, 16] |
| u16DstNum | Number of output nodes in a network segment<br>Value range: [1, 16] |
| u16RoiPoolNum | Actual number of ROIPooling and PSROIPooling layers in a network segment<br>The number of xRoiPooling layers that can be included in a single-segment network is [0, SVP_NNIE_MAX_ROI_LAYER_NUM_OF_SEG]. The number of xRoiPooling layers that can be included in the entire network is [0, SVP_NNIE_MAX_ROI_LAYER_NUM]. |
| u16MaxStep | Maximum number of "frames" in the sequence of the RNN or LSTM network<br>Value range: [1, 1024] |
| astSrcNode[i] | Information about the *ith* input node in a network segment. |
| astDstNode[i] | Information about the *ith* output node in a network segment. |
| au32RoiIdx[i] | Subscript of the SVP_NNIE_ROIPOOL_INFO_S array of the *ith* ROIPooling or PSROIPooling in SVP_NNIE_MODEL_S in a network segment. |

[Note]

None

[See Also]

SVP_NNIE_MODEL_S

## SVP_NNIE_MODEL_S

[Description]

Defines NNIE model structure.

[Syntax]

```
typedef struct hiSVP_NNIE_MODEL_S
{
    SVP_NNIE_RUN_MODE_E     enRunMode;

    HI_U32                u32TmpBufSize; /*temp buffer size*/
    HI_U32                u32NetSegNum;
    SVP_NNIE_SEG_S          astSeg[SVP_NNIE_MAX_NET_SEG_NUM];
    SVP_NNIE_ROIPOOL_INFO_S astRoiInfo[SVP_NNIE_MAX_ROI_LAYER_NUM];
/*ROIPooling info*/

    SVP_MEM_INFO_S          stBase;
}SVP_NNIE_MODEL_S;
```

[Member]

| Member | Description |
| --- | --- |
| enRunMode | Running mode of the network model |
| u32TmpBufSize | tmpBuf size of the auxiliary memory<br>Value range: (0, 4294967295] |
| u32NetSegNum | Number of NNIE network segments in the network model<br>Value range: [1, 8] |
| astSeg | Information of the segment executed by the network on the NNIE |
| astRoiInfo | Information array of ROIPooling and PSROIPooling in the network model |
| stBase | Other network information |

[Note]

None

[See Also]

None

## SVP_NNIE_FORWARD_CTRL_S

[Description]

Defines the CNN/DNN/RNN network prediction control parameter

[Syntax]

```
typedef struct hiSVP_NNIE_FORWARD_CTRL_S
{
    HI_U32            u32SrcNum;     /* input node num, [1, 16] */
    HI_U32            u32DstNum;     /* output node num, [1, 16]*/
    HI_U32            u32NetSegId;    /* net segment index running on NNIE
*/
    SVP_NNIE_ID_E      enNnieId;      /* device target which running the
seg*/
    SVP_MEM_INFO_S     stTmpBuf;      /* auxiliary temp mem */
    SVP_MEM_INFO_S     stTskBuf;      /* auxiliary task mem */
}SVP_NNIE_FORWARD_CTRL_S;
```

[Member]

| Member | Description |
| --- | --- |
| u32SrcNum | Number of input nodes to an NNIE network segment<br>Value range: [1, 16] |
| u32DstNum | Number of output nodes from an NNIE network segment<br>Value range: [1, 16] |
| u32NetSegId | Sequence number of a network segment<br>Value range: [0, 8). The value must be smaller than the number of segments on the network. |
| enNnieId | NNIE ID of an NNIE network segment<br>Value range: [SVP_NNIE_ID_0, SVP_NNIE_ID_BUTT) |
| stTmpBuf | Auxiliary memory |
| stTskBuf | Auxiliary memory |

[Note]

- **stTmpBuf** and **stTskBuf** can be released only when they are no longer used by tasks.
- After a task is executed by calling HI_MPI_SVP_NNIE_Forward and before the task is complete, the memories that **stTmpBuf** and **stTskBuf** point to cannot be used by other tasks.

[See Also]

None

## SVP_NNIE_FORWARD_WITHBBOX_CTRL_S

[Description]

Defines the prediction control parameter of the target detection network with Bbox inputs

[Syntax]

```
typedef struct hiSVP_NNIE_FORWARD_WITHBBOX_CTRL_S
```

```
{
    HI_U32            u32SrcNum;     /* input node num, [1, 16] */
    HI_U32            u32DstNum;     /* output node num, [1, 16]*/
    HI_U32            u32ProposalNum; /* element num of  roi array */
    HI_U32            u32NetSegId;    /* net segment index running on NNIE
*/
    SVP_NNIE_ID_E      enNnieId;      /* device target which running the
seg*/
    SVP_MEM_INFO_S     stTmpBuf;      /* auxiliary temp mem */
    SVP_MEM_INFO_S     stTskBuf;      /* auxiliary task mem */
}SVP_NNIE_FORWARD_WITHBBOX_CTRL_S;
```

[Member]

| Member | Description |
| --- | --- |
| u32SrcNum | Number of input nodes to an NNIE network segment<br>Value range: [1, 16] |
| u32DstNum | Number of output nodes from an NNIE network segment<br>Value range: [1, 16] |
| u32ProposalNum | Number of proposal layers that generate Bbox network layers, the same as the number of elements in the **astBbox[]** array of the HI_MPI_SVP_NNIE_ForwardWithBbox interface<br>Value range: [1, SVP_NNIE_MAX_ROI_LAYER_NUM_OF_SEG] |
| u32NetSegId | Sequence number of a network segment<br>Value range: [0, 8). The value must be smaller than the number of segments on the network. |
| enNnieId | NNIE ID of an NNIE network segment<br>Value range: [SVP_NNIE_ID_0, SVP_NNIE_ID_BUTT) |
| stTmpBuf | Auxiliary memory |
| stTskBuf | Auxiliary memory |

[Note]

- **stTmpBuf** and **stTskBuf** can be released only when they are no longer used by tasks.
- After a task is executed by calling HI_MPI_SVP_NNIE_CNN_ForwardWithBbox and before the task is complete, the memories that **stTmpBuf** and **stTskBuf** point to cannot be used by other tasks.

[See Also]

None

# 2.5 Error Codes

Table 2-4 describes the NNIE API error codes.

**Table 2-4** NNIE API error codes

| Error Code | Macro Definition | Description |
|---|---|---|
| 0xA0338001 | HI_ERR_SVP_NNIE_INVALID_DEVID | The device ID is invalid. |
| 0xA0338002 | HI_ERR_SVP_NNIE_INVALID_CHNID | The channel group ID or the region handle is invalid. |
| 0xA0338003 | HI_ERR_SVP_NNIE_ILLEGAL_PARAM | The parameter is invalid. |
| 0xA0338004 | HI_ERR_SVP_NNIE_EXIST | The device, channel, or resource to be created already exists. |
| 0xA0338005 | HI_ERR_SVP_NNIE_UNEXIST | The device, channel, or resource to be used or destroyed does not exist. |
| 0xA0338006 | HI_ERR_SVP_NNIE_NULL_PTR | The pointer is null. |
| 0xA0338007 | HI_ERR_SVP_NNIE_NOT_CONFIG | The module is not configured. |
| 0xA0338008 | HI_ERR_SVP_NNIE_NOT_SUPPORT | The parameter or function is not supported. |
| 0xA0338009 | HI_ERR_SVP_NNIE_NOT_PERM | The operation, for example, modifying the value of a static parameter, is forbidden. |
| 0xA033800C | HI_ERR_SVP_NNIE_NOMEM | The memory fails to be allocated for the reasons such as system memory insufficiency. |
| 0xA033800D | HI_ERR_SVP_NNIE_NOBUF | The buffer fails to be allocated. The reason may be that the requested image buffer is too large. |
| 0xA033800E | HI_ERR_SVP_NNIE_BUF_EMPTY | There is no data in the buffer. |
| 0xA033800F | HI_ERR_SVP_NNIE_BUF_FULL | The buffer is full of data. |
| 0xA0338010 | HI_ERR_SVP_NNIE_NOTREADY | The system is not initialized or the corresponding module driver is not loaded. |
| 0xA0338011 | HI_ERR_SVP_NNIE_BADADDR | The address is invalid. |
| 0xA0338012 | HI_ERR_SVP_NNIE_BUSY | The system is busy. |
| 0xA0338040 | HI_ERR_SVP_NNIE_SYS_TIMEOUT | The system times out. |

| Error Code | Macro Definition | Description |
| --- | --- | --- |
| 0xA0338041 | HI_ERR_SVP_NNIE_QUERY_TIMEOUT | The query times out. |
| 0xA0338042 | HI_ERR_SVP_NNIE_CFG_ERR | A configuration error occurs. |
| 0xA0338043 | HI_ERR_SVP_NNIE_OPEN_FILE | Opening a file fails. |
| 0xA0338044 | HI_ERR_SVP_NNIE_READ_FILE | Reading a file fails. |
| 0xA0338045 | HI_ERR_SVP_NNIE_WRITE_FILE | Writing to a file fails. |

# 2.6 Proc Debugging Information

## 2.6.1 Overview

The debugging information is obtained from the proc file system on Linux. The information reflects the system status and can be used to locate and analyze problems.

[Document Directory]

**/proc/umap**

[Viewing the Information]

- Run a **cat** command such as **cat /proc/umap/nnie** on the console or run file operation commands such as **cp /proc/umap/nnie ./** to copy all the proc files to the current directory.
- Read the preceding files as common read-only files in the application by using functions such as fopen and fread.

**NOTE**

Note the following when reading parameter descriptions:

- For the parameter whose value is **0** or **1**, if the mapping between the values and definitions is not specified, the value **1** indicates affirmative and the value **0** indicates negative.
- For the parameter whose value is **aaa**, **bbb**, or **ccc**, if the mapping between the values and the definitions is not specified, identify the parameter definitions based on **aaa**, **bbb**, or **ccc**.

## 2.6.2 Proc Information

[Debugging Information]

```
# cat /proc/umap/nnie

[NNIE] Version: [Hi3559AV100_MPP_Vx.x.x.x B0xx Release], Build Time[mm
dd yyyy, hh:mm:ss]
--------------------NNIE MODULE PARAM---------------------------------
nnie_save_power      nnie_max_tskbuf_num
        1                32
```

```
--------------------NNIE QUEUE INFO-----------------------------------
   CoreId     Wait     Busy WaitCurId WaitEndId BusyCurId BusyEndId
       0        0       -1        0        0        0        0
       1        0       -1        0        0        0        0


-------------------- NNIE TASK INFO-----------------------------------
   CoreId      Hnd   TaskFsh    LastId    TaskId   HndWrap   FshWrap
       0        0        0        0        0        0        0
       1        0        0        0        0        0        0
   FreeTskBufNum      UseTskBufNum
       32                  0
       32                  0

--------------------- NNIE RUN-TIME INFO-----------------------------
CoreId  LastInst  CntPerSec  MaxCntPerSec  TotalIntCntLastSec
      0        0          0             0                   0
         1        0         0           0                  0

TotalIntCnt   QTCnt   STCnt    CfgErrCnt  CostTm     MCostTm
          0        0         0           0        0          0
          0        0         0           0        0          0

     CostTmPerSec  MCostTmPerSec  TotalIntCostTm      RunTm
        0        0           0            0            0


        0        0           0            0            0

----------------------NNIE INVOKE INFO-----------------------------------
CoreId   Forward   ForwardWithBbox
        0        0         0
        1        0         0
```

[Analysis]

Records the work status and resource information about the current NNIE, including NNIE queue status, task status, runtime status, and calling information.

[Parameter Description]

| Parameter | | Description |
|---|---|---|
| NNIE module parameter | nnie_save_power | Low power consumption flag<br>**0**: Low power consumption is disabled.<br>**1**: Low power consumption is enabled. |

| Parameter | | Description |
|---|---|---|
| | nnie_max_tskbuf_num | Maximum number of TskBuf items that can be added |
| NNIE queue information | CoreId | NNIE core ID |
| | Wait | Waiting queue ID (0 or 1) |
| | Busy | ID of the queue being scheduled (0, 1, or –1). The value **–1** indicates that the NNIE hardware is idle. |
| | WaitCurId | ID of the first valid task in the waiting queue |
| | WaitEndId | ID of the last valid task in the waiting queue + 1 |
| | BusyCurId | ID of the first valid task in the queue being scheduled |
| | BusyEndId | ID of the last valid task in the queue being scheduled + 1 |
| NNIE task information | CoreId | NNIE core ID |
| | Hnd | Handle ID that can be allocated to the current task |
| | TaskFsh | Number of completed tasks |
| | LastId | ID of the previously completed task |
| | TaskId | ID of the currently completed task |
| | HndWrap | Number of times that the user handle ID is wrapped |
| | FshWrap | Number of times that the number of completed tasks is wrapped |
| | FreeTskBufNum | Number of idle nodes in the TskBuf linked list |
| | UseTskBufNum | Number of used nodes in the TskBuf linked list |
| NNIE runtime information | CoreId | NNIE core ID |
| | LastInst | **bInstant** value transferred when users submit tasks last time |
| | CntPerSec | Number of times that interrupt functions are called in the last second |
| | MaxCntPerSec | Historical maximum number of times that interrupt functions are called in one second |
| | TotalIntCntLastSec | Number of times when interrupts are reported in the last second |
| | TotalIntCnt | Number of times of NNIE interrupts |
| | QTCnt | Number of interrupts due to NNIE query timeout |

| Parameter | | Description |
|---|---|---|
| | STCnt | Number of times of NNIE system timeouts |
| | CfgErrCnt | Number of interrupts of NNIE configuration error |
| | CostTm | Time consumed by the last interrupt<br>Unit: μs |
| | MCostTm | Maximum time consumed by an interrupt<br>Unit: μs |
| | CostTmPerSec | Time consumed by the interrupts during the last second<br>Unit: μs |
| | MCostTmPerSec | Maximum time consumed by the interrupts during one second<br>Unit: μs |
| | TotalIntCostTm | Total time of handling interrupts<br>Unit: μs |
| | RunTm | Total runtime of the NNIE<br>Unit: second |
| NNIE calling information | CoreId | NNIE core ID |
| | Forward | Number of times that NNIE **Forward** is called |
| | ForwardWithBbox | Number of times that NNIE **ForwardWithBbox** is called |

# 3 Runtime

## 3.1 Overview

Runtime is a software system developed based on the NNIE. Users can develop intelligent analysis solutions using Runtime without caring about the logic such as scheduling, maximizing the multiplexing of the NNIE hardware.

## 3.2 Function Description

### 3.2.1 Important Concepts

- Network model group

  In most cases, a model is connected to another model. With Runtime, cascaded models can form a model group. The system assigns a handle to each model group. Different handles indicate different model groups.

- Connector

  In the construction of a model group, connector objects are required between models. Two networks can be connected using a connector object. You can also customize the service logic internally.

- Priority

  You can set the priority of a model group based on service requirements. Runtime executes the model groups based on the priority.

## 3.3 API Reference

Runtime provides the following APIs for creating and querying tasks:

- HI_SVPRT_RUNTIME_Init: Initializes the Runtime running environment.
- HI_SVPRT_RUNTIME_LoadModelGroup: Parses the network model from the model group that is loaded to the buffer by the user (synchronization).
- HI_SVPRT_RUNTIME_ForwardGroupSync: Predicts the network with multi-node input and output (synchronization).

- HI_SVPRT_RUNTIME_ForwardGroupASync: Predicts the network with multi-node input and output (asynchronization).
- HI_SVPRT_RUNTIME_UnloadModelGroup: Uninstalls the loaded model group.
- HI_SVPRT_RUNTIME_DeInit: Deinitializes the Runtime environment.
- HI_NodePlugin_getNodeType: Obtains the plug-in type. The current plug-in type is **Proposal**.
- HI_NodePlugin_Compute: Computes a plug-in.

## HI_SVPRT_RUNTIME_Init

[Description]

Initializes the Runtime running environment.

[Syntax]

```
HI_S32 HI_SVPRT_RUNTIME_Init(IN const HI_CHAR* pszGlobalSetting, IN
HI_RUNTIME_MEM_CTRL_S *pstMemCtrl);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pszGlobalSetting | Initializes global configurations, including the CPU thread affinity and the global plug-in library | Input |
| pstMemCtrl | Initializes the memory management function. When **HI_NULL** is entered, the MMZ memory is used by default. | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

[Requirement]

- Header files: **hi_runtime_comm.h** and **hi_runtime_api.h**
- Library file: **libsvpruntime.a** and **libsvpruntime.so** (**libsvpruntime.a**, **svpruntime.lib**, and **svpruntime.dll** for simulation on the PC)

[Example]

For details, see the **sample/hirt/src** directory in the SDK.

[See Also]

None

## HI_SVPRT_RUNTIME_LoadModelGroup

[Description]

Parses the network model from the model group that is loaded to the buffer by the user.

[Syntax]

```
HI_S32 HI_SVPRT_RUNTIME_LoadModelGroup(IN const HI_CHAR*
pstModelGroupConfig, IN HI_RUNTIME_GROUP_INFO_S *pstModelGroupAttr, OUT
HI_RUNTIME_GROUP_HANDLE* phGroupHandle);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pstModelGroupConfig | Network structure of the network model group and network topology in prototxt format | Input |
| pstModelGroupAttr | Internal information structure of the network model group | Input |
| phGroupHandle | Handle to a group | Output |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

[Requirement]

- Header files: **hi_runtime_comm.h** and **hi_runtime_api.h**
- Library file: **libsvpruntime.a** and **libsvpruntime.so** (**libsvpruntime.a**, **svpruntime.lib**, and **svpruntime.dll** for simulation on the PC)

[Note]

None

[Example]

For details, see the **sample/hirt/src** directory in the SDK.

[See Also]

None

## HI_SVPRT_RUNTIME_ForwardGroupSync

[Description]

Predicts the network with multi-node input and output (synchronization).

[Syntax]

```
HI_S32 HI_SVPRT_RUNTIME_ForwardGroupSync(IN const HI_RUNTIME_GROUP_HANDLE hGroupHandle, IN const HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_PTR pstSrc, OUT HI_RUNTIME_GROUP_DST_BLOB_ARRAY_PTR pstDst, IN HI_U64 u64SrcId);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| hGroupHandle | Handle to a group<br>This field cannot be null. | Input |
| pstSrc | BLOB object that consists of the multi-node input of each model in the model group. Multiple frames can be input at the same time. | Input |
| pstDst | Multi-node output of the network segments of each model in the model group, including the results of the flagged intermediate layers that need to be reported and the final result of the model group | Output |
| u64SrcId | Input frame ID | Input |

| Parameter | Supported Type | Address Alignment | Resolution |
| --- | --- | --- | --- |
| pstSrc | YVU420SP/<br>YVU422SP/<br>U8/S32/<br>VEC_S32/<br>SEQ_S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | 8 x 8–4096 x 4096<br>Vector dimension: 1–0xFFFFFFFF |
| pstDst | S32/VEC_S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | 8 x 8–4096 x 4096<br>Vector dimension: 1–0xFFFFFFFF |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

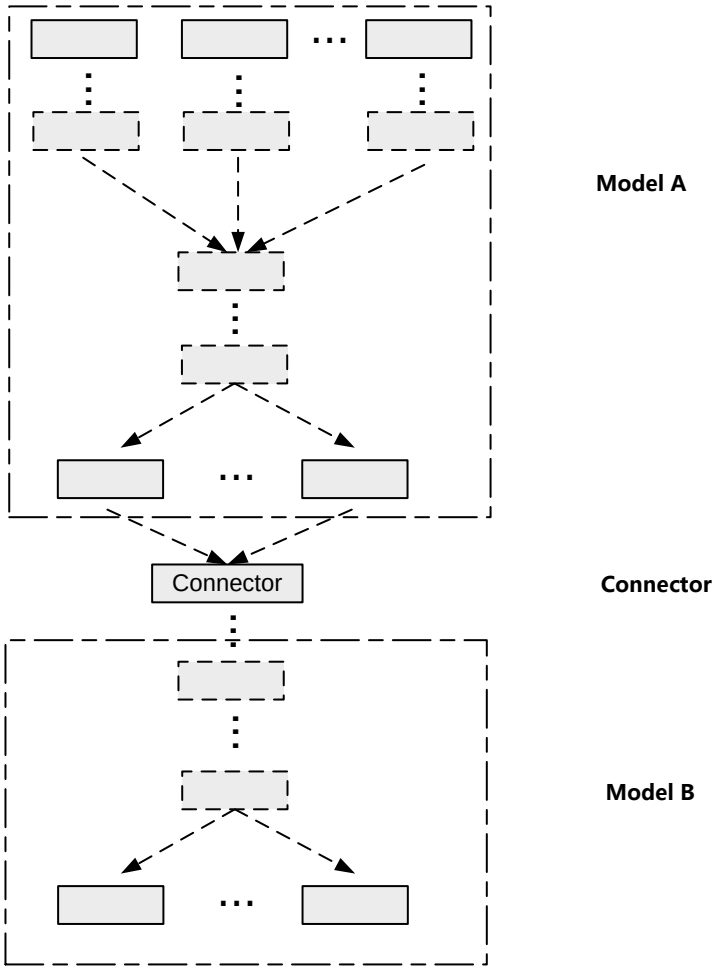
[Requirement]

- Header files: **hi_runtime_comm.h** and **hi_runtime_api.h**
- Library file: **libsvpruntime.a** and **libsvpruntime.so** (**libsvpruntime.a**, **svpruntime.lib**, and **svpruntime.dll** for simulation on the PC)

[Note]

- The U8 image input supports only channel 1 (gray scale image) and channel 3 (RGB image).
- When multiple BLOBs are input and output, the mapping between the BLOBs and the layers is shown in the .dot diagram generated by the dot description file output by the compiler.

**Figure 3-1** Input and output diagram of the network model group




[Example]

For details, see the **sample/hirt/src** directory in the SDK.

[See Also]

None

## HI_SVPRT_RUNTIME_ForwardGroupASync

[Description]

Predicts the network with multi-node input and output (asynchronization).

[Syntax]

```
HI_S32 HI_SVPRT_RUNTIME_ForwardGroupASync(IN const
HI_RUNTIME_GROUP_HANDLE hGroupHandle, IN const
HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_PTR pstSrc, OUT
HI_RUNTIME_GROUP_DST_BLOB_ARRAY_PTR pstDst, IN HI_U64 u64SrcId, IN
HI_RUNTIME_Forward_Callback pCbFun);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| hGroupHandle | Handle to a group<br>This field cannot be null. | Input |
| pstSrc | BLOB object that consists of the multi-node input of each model in the model group. Multiple frames can be input at the same time. | Input |
| pstDst | Multi-node output of the network segments of each model in the model group, including the results of the flagged intermediate layers that need to be reported and the final result of the model group | Output |
| u64SrcId | Input frame ID | Input |
| pCbFun | Result callback function | Input |

| Parameter | Supported Type | Address Alignment | Resolution |
| --- | --- | --- | --- |
| pstSrc | YVU420SP/<br>YVU422SP/<br>U8/S32/<br>VEC_S32/<br>SEQ_S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | 8 x 8–4096 x 4096<br>Vector dimension: 1–0xFFFFFFFF |

| Parameter | Supported Type | Address Alignment | Resolution |
| --- | --- | --- | --- |
| pstDst | S32/VEC_S32 | DDR3: 16-byte alignment.<br>DDR4: 32-byte alignment.<br>256-byte alignment is recommended for better performance. | 8 x 8–4096 x 4096<br>Vector dimension: 1–0xFFFFFFFF |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

[Requirement]

- Header files: **hi_runtime_comm.h** and **hi_runtime_api.h**
- Library file: **libsvpruntime.a** and **libsvpruntime.so** (**libsvpruntime.a**, **svpruntime.lib**, and **svpruntime.dll** for simulation on the PC)

[Note]

- The U8 image input supports only channel 1 (gray scale image) and channel 3 (RGB image).
- When multiple BLOBs are input and output, the mapping between the BLOBs and the layers is shown in the .dot diagram generated by the dot description file output by the compiler. For details, see HI_SVPRT_RUNTIME_ForwardGroupSync.

[Example]

For details, see the **sample/hirt/src** directory in the SDK.

[See Also]

None

## HI_SVPRT_RUNTIME_UnloadModelGroup

[Description]

Uninstalls the model group.

[Syntax]

```
HI_S32 HI_SVPRT_RUNTIME_UnloadModelGroup(IN const HI_RUNTIME_GROUP_HANDLE hGroupHandle);
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| hGroupHandle | Handle to a group | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

[Requirement]

- Header files: **hi_runtime_comm.h** and **hi_runtime_api.h**
- Library file: **libsvpruntime.a** and **libsvpruntime.so** (**libsvpruntime.a**, **svpruntime.lib**, and **svpruntime.dll** for simulation on the PC)

[Note]

None

[Example]

For details, see the **sample/hirt/src** directory in the SDK.

[See Also]

None

## HI_SVPRT_RUNTIME_DeInit

[Description]

Deinitializes the Runtime running environment.

[Syntax]

```
HI_S32 HI_SVPRT_RUNTIME_DeInit();
```

[Parameter]

None

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

[Requirement]

- Header files: **hi_runtime_comm.h** and **hi_runtime_api.h**
- Library file: **libsvpruntime.a** and **libsvpruntime.so** (**libsvpruntime.a**, **svpruntime.lib**, and **svpruntime.dll** for simulation on the PC)

[Note]

None

[Example]

For details, see the **sample/hirt/src** directory in the SDK.

[See Also]

None

## HI_NodePlugin_getNodeType

[Description]

Obtains the plug-in type. The current plug-in type is **Proposal**.

[Syntax]

```
HI_S32 HI_NodePlugin_getNodeType(HI_CHAR pszNodeType[])
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pszNodeType | Character array for obtaining the plug-in node type | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

[Requirement]

- Header files: **hi_plugin.h**, **hi_runtime_comm.h**, and **proposal_common.h**
- Library files: **libruntime_plugin_proposal.a** and **libruntime_plugin_proposal.so** (**libruntime_plugin_proposal.dll** used for simulation on the PC)

[Note]

- This function implements a customized proposal plug-in. Currently, the NNIE does not support the direct implementation of the proposal layer. Therefore, the layer is implemented as a plug-in.

- Similarly, other layers that are not supported by the NNIE can also be implemented as plug-ins.
- This function is called by the runtime library and does not need to be called directly by users.

[Example]

For details, see the **sample/hirt/plugins** directory in the SDK.

[See Also]

None

## HI_NodePlugin_Compute

[Description]

Computes a plug-in.

[Syntax]

```
HI_S32 HI_NodePlugin_Compute(const HI_NodePlugin_Operand_S *pstInputs,
HI_U32 u32InputNum, HI_NodePlugin_Operand_S *pstOutputs,
HI_U32 u32Outputs, HI_NodePlugin_NodeParam_S* pstHyperParam,
HI_NodePlugin_NodeParam_S* pstTrainingParam)
```

[Parameter]

| Parameter | Description | Input/Output |
| --- | --- | --- |
| pstInputs | Blobs input to the plug-in | Input |
| u32InputNum | Number of Blobs input to the plug-in | Input |
| pstOutputs | Blobs output by the plug-in | Output |
| u32Outputs | Number of output blobs | Output |
| pstHyperParam | Hyperparameter configured for the plug-in | Input |
| pstTrainingParam | Training parameter configured for the plug-in | Input |

[Return Value]

| Return Value | Description |
| --- | --- |
| 0 | Success |
| Other values | Failure. The return value is an error code. For details, see section 3.5 "Error Codes." |

[Requirement]

- Header files: **hi_plugin.h**, **hi_runtime_comm.h**, and **proposal_common.h**
- Library files: **libruntime_plugin_proposal.a** and **libruntime_plugin_proposal.so** (**libruntime_plugin_proposal.dll** used for simulation on the PC)

[Note]

- This function implements a customized proposal plug-in. Currently, the NNIE does not support the direct implementation of the proposal layer. Therefore, the layer is implemented as a plug-in.
- Similarly, other layers that are not supported by the NNIE can also be implemented as plug-ins.
- This function is called by the runtime library and does not need to be called directly by users.
- **pstTrainingParam** is a reserved field for the plug-in. The user can determine whether to use this field as required when implementing a plug-in library.

[Example]

For details, see the **sample/hirt/plugins** directory in the SDK.

[See Also]

None

# 3.4 Data Structures

Runtime provides the following data structures:

- HI_RUNTIME_MEM_S: Defines the 1D memory information.
- HI_RUNTIME_BLOB_TYPE_E: Defines the data memory layout of the BLOB.
- HI_RUNTIME_GROUP_PRIORITY_E: Defines the priorities of the model groups.
- HI_RUNTIME_BLOB_S: Defines the information of multiple consecutive BLOBs.
- HI_RUNTIME_BLOB_ARRAY_S: Defines the source sequence.
- HI_RUNTIME_SRC_BLOB_ARRAY_S: Defines the input sequence.
- HI_RUNTIME_DST_BLOB_ARRAY_S: Defines the output sequence.
- HI_RUNTIME_SRC_BLOB_ARRAY_PTR: Defines the pointer to the input sequence.
- HI_RUNTIME_DST_BLOB_ARRAY_PTR: Defines the pointer to the output sequence.
- HI_RUNTIME_GROUP_HANDLE: Defines the handle of a model group.
- HI_RUNTIME_GROUP_BLOB_S: Defines the BLOB structure corresponding to a group.
- HI_RUNTIME_GROUP_BLOB_ARRAY_S: Defines the information of the BLOB array corresponding to a group.
- HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_S: Defines the information of the input BLOB array corresponding to a group.
- HI_RUNTIME_GROUP_DST_BLOB_ARRAY_S: Defines the information of the output BLOB array corresponding to a group.
- HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_PTR: Defines the pointer to the information of the input BLOB array corresponding to a group.

- HI_RUNTIME_GROUP_DST_BLOB_ARRAY_PTR: Defines the pointer to the information of the output BLOB array corresponding to a group.
- HI_RUNTIME_GROUP_SRC_BLOB_S: Defines the input BLOB of a group.
- HI_RUNTIME_GROUP_DST_BLOB_S: Defines the output BLOB of a group.
- HI_RUNTIME_MEM_CTRL_S: Defines the memory management structure.
- HI_RUNTIME_WK_INFO_S: Defines the WK model.
- HI_RUNTIME_WK_INFO_ARRAY_S: Defines the WK model array information.
- HI_RUNTIME_COP_ATTR_S: Defines the attributes of a user-defined layer.
- HI_RUNTIME_COP_ATTR_ARRAY_S: Defines the array information of the user-defined layer.
- HI_RUNTIME_CONNECTOR_ATTR_S: Defines the attributes of the connector object.
- HI_RUNTIME_CONNECTOR_ATTR_ARRAY_S: Defines the array information of the connector object.
- HI_RUNTIME_GROUP_INFO_S: Defines the Runtime group information.
- HI_RUNTIME_FORWARD_STATUS_CALLBACK_E: Defines the status information of asynchronous forward.
- HI_RUNTIME_Forward_Callback: Defines the callback function of asynchronous forward.
- HI_RUNTIME_Connector_Compute: Defines the callback function of the connector.
- MAX_OPERAND_NAME_LEN: Defines the maximum length of the operation description string for a plug-in.
- HI_NodePlugin_Shape_S: Defines the H, W, and C dimensions of the input and output of a plug-in.
- HI_NodePlugin_ElemType_E: Defines the enumeration of data alignment modes of a plug-in.
- HI_NodePlugin_Operand_S: Defines the data and attributes of the input and output of a plug-in.
- HI_NodePlugin_NodeParam_S: Defines the hyperparameter and training parameter of a plug-in.

## MAX_NAME_LEN

[Description]

Defines the maximum length of the model and BLOB names.

[Syntax]

```
#define MAX_NAME_LEN 64
```

[Member]

None

[Note]

None

[See Also]

None

## HI_RUNTIME_MEM_S

[Description]

Defines the 1D memory information.

[Syntax]

```
typedef struct hiRUNTIME_MEM_S
{
    HI_U64  u64PhyAddr; /* RW;The physical address of the memory */
    HI_U64  u64VirAddr; /* RW;The virtual address of the memory */
     HI_U32  u32Size;    /* RW;The size of memory */
}HI_RUNTIME_MEM_S;
```

[Member]

| Member | Description |
| --- | --- |
| u64VirAddr | Virtual address of the memory block |
| u64PhyAddr | Physical address of the memory block |
| u32Size | Number of bytes in the memory block. For details, see Figure 2-8. |

[Note]

None

[See Also]

None

## HI_RUNTIME_BLOB_TYPE_E

[Description]

Defines the data memory layout of the BLOB.

[Syntax]

```
typedef enum hiRUNTIME_BLOB_TYPE_E
{
    HI_RUNTIME_BLOB_TYPE_S32      =  0x0,
    HI_RUNTIME_BLOB_TYPE_U8       =  0x1,
    HI_RUNTIME_BLOB_TYPE_YVU420SP  =  0x2,
    HI_RUNTIME_BLOB_TYPE_YVU422SP  =  0x3,
    HI_RUNTIME_BLOB_TYPE_VEC_S32   =  0x4,
    HI_RUNTIME_BLOB_TYPE_SEQ_S32   =  0x5,
    HI_RUNTIME_BLOB_TYPE_BUTT
}HI_RUNTIME_BLOB_TYPE_E;
```

[Member]

| Member | Description |
| --- | --- |
| HI_RUNTIME_BLOB_TYPE_S32 | The BLOB data element is of the S32 type. |
| HI_RUNTIME_BLOB_TYPE_U8 | The BLOB data element is of the U8 type. |
| HI_RUNTIME_BLOB_TYPE_YVU420SP | The data memory layout of the BLOB is in YVU420SP format. |
| HI_RUNTIME_BLOB_TYPE_YVU422SP | The data memory layout of the BLOB is in YVU422SP format. |
| HI_RUNTIME_BLOB_TYPE_VEC_S32 | Storage vector in BLOB. The data element is of the S32 type. |
| HI_RUNTIME_BLOB_TYPE_SEQ_S32 | Storage sequence in the BLOB. The data element is of the S32 type. |

[Note]

For details about the memory layout, see SVP_BLOB_TYPE_E of the NNIE.

[See Also]

None

## HI_RUNTIME_GROUP_PRIORITY_E

[Description]

Defines the priorities of the model groups.

[Syntax]

```
typedef enum hiRUNTIME_GROUP_PRIORITY_E
{
    HI_RUNTIME_GROUP_PRIORITY_HIGHEST = 0x0,
    HI_RUNTIME_GROUP_PRIORITY_HIGH,
    HI_RUNTIME_GROUP_PRIORITY_MEDIUM,
    HI_RUNTIME_GROUP_PRIORITY_LOW,
    HI_RUNTIME_GROUP_PRIORITY_LOWEST,
    HI_RUNTIME_GROUP_PRIORITY_BUTT
} HI_RUNTIME_GROUP_PRIORITY_E;
```

[Member]

| Member | Description |
| --- | --- |
| HI_RUNTIME_GROUP_PRIORITY_HIGHEST | Highest priority |
| HI_RUNTIME_GROUP_PRIORITY_HIGH | High priority |
| HI_RUNTIME_GROUP_PRIORITY_MEDIUM | Medium priority |

| Member | Description |
| --- | --- |
| HI_RUNTIME_GROUP_PRIORITY_LOW | Low priority |
| HI_RUNTIME_GROUP_PRIORITY_LOWEST | Lowest priority |

[Note]

None

[See Also]

None

## HI_RUNTIME_BLOB_S

[Description]

Defines the information of multiple consecutive BLOBs.

[Syntax]

```
typedef struct hiRUNTIME_BLOB_S
{
    SVP_BLOB_TYPE_E enBlobType;     /*Blob type*/
    HI_U32 u32Stride;          /*Stride, a line bytes num*/
    HI_U64 u64VirAddr;         /*virtual addr*/
    HI_U64 u64PhyAddr;         /*physical addr*/
    HI_U32 u32Num;        /*N: frame num or sequence num,correspond to
caffe blob's n*/
   union
   {
      struct
      {
         HI_U32 u32Width;    /*W: frame width, correspond to caffe blob's
w*/
         HI_U32 u32Height;   /*H: frame height, correspond to caffe
blob's h*/
         HI_U32 u32Chn;     /*C: frame channel,correspond to caffe
blob's c*/
      }stWhc;
      struct
      {
         HI_U32 u32Dim;         /*D: vector dimension*/
         HI_U64 u64VirAddrStep;  /*T: virtual address of time steps
array in each sequence*/
      }stSeq;
   }unShape;
}HI_RUNTIME_BLOB_S,*HI_RUNTIME_BLOB_PTR;
```

[Member]

| Member | Description |
| --- | --- |
| enBlobType | BLOB type |
| u32Stride | Number of aligned bytes of single-line data in the BLOB |
| u64VirAddr | Start virtual address of the BLOB |
| u64PhyAddr | Start physical address of the BLOB |
| u32Num | Number of consecutive memory blocks. If one frame of data corresponds to one block, then the number of frames in the BLOB is **u32Num**. |
| u32Width | Width of the BLOB |
| u32Height | Height of the BLOB |
| u32Chn | Number of channels in the BLOB |
| u32Dim | Length of the vector. It is used only as the representation of the recurrent neural network (RNN) or Long-Short Term Memory (LSTM) data. |
| u64VirAddrStep | Array address. The array element indicates the number of vectors in each sequence. |

[Note]

- The following table describes the dimensions of the data memory blocks in Caffe.

| Data Block | Dimension 0 | Dimension 1 | Dimension 2 | Dimension 3 |
| --- | --- | --- | --- | --- |
| Image/Feature map | N | C | H | W |
| FC (normal) vector | N | C | - | - |
| RNN/LSTM vector | T | N | D | - |

- The following table describes the corresponding BLOB representation.

| Data Block | Dimension 0 | Dimension 1 | Dimension 2 | Dimension 3 |
| --- | --- | --- | --- | --- |
| Image/Feature map | u32Num | 32Chn | u32Height | u32Width |
| FC (normal) vector | u32Num | u32Width | - | - |
| RNN/LSTM vector | u64VirAddrStep[i] | u32Num | u32Dim | - |

- **u32Stride** indicates the number of aligned bytes in the **u32Width** and **u32Dim** directions.

[See Also]

HI_RUNTIME_BLOB_TYPE_E

## HI_RUNTIME_BLOB_ARRAY_S

[Description]

Defines the source sequence.

[Syntax]

```
typedef struct hiRUNTIME_SRC_BLOB_ARRAY_S
{
    HI_U32 u32BlobNum;
    HI_RUNTIME_BLOB_S *pstBlobs;
} HI_RUNTIME_BLOB_ARRAY_S, *HI_RUNTIME_BLOB_ARRAY_PTR;
```

[Member]

| Member | Description |
| --- | --- |
| u32BlobNum | Number of input BLOBs |
| pstBlobs | Input BLOB pointer, pointing to the BLOB array |

[Note]

None

[See Also]

HI_RUNTIME_BLOB_S

## HI_RUNTIME_SRC_BLOB_ARRAY_S

[Description]

Defines the input sequence.

[Syntax]

```
typedef HI_RUNTIME_BLOB_ARRAY_S HI_RUNTIME_SRC_BLOB_ARRAY_S;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_BLOB_ARRAY_S

## HI_RUNTIME_DST_BLOB_ARRAY_S

[Description]

Defines the output sequence.

[Syntax]

```
typedef HI_RUNTIME_BLOB_ARRAY_S HI_RUNTIME_DST_BLOB_ARRAY_S;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_BLOB_ARRAY_S

## HI_RUNTIME_SRC_BLOB_ARRAY_PTR

[Description]

Defines the pointer to the input sequence.

[Syntax]

```
typedef HI_RUNTIME_BLOB_ARRAY_PTR HI_RUNTIME_SRC_BLOB_ARRAY_PTR;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_BLOB_ARRAY_S

## HI_RUNTIME_DST_BLOB_ARRAY_PTR

[Description]

Defines the pointer to the output sequence.

[Syntax]

```
typedef HI_RUNTIME_BLOB_ARRAY_PTR HI_RUNTIME_DST_BLOB_ARRAY_PTR;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_BLOB_ARRAY_S

## HI_RUNTIME_GROUP_HANDLE

[Description]

Defines the handle of a model group.

[Syntax]

```
typedef  HI_VOID*  HI_RUNTIME_GROUP_HANDLE;
```

[Member]

None

[Note]

None

[See Also]

None

## HI_RUNTIME_GROUP_BLOB_S

[Description]

Defines the BLOB structure corresponding to a group.

[Syntax]

```
typedef struct hiRUNTIME_GROUP_BLOB_S
{
    HI_RUNTIME_BLOB_PTR pstBlob;
    HI_CHAR acOwnerName[MAX_NAME_LEN+1];
    HI_CHAR acBlobName[MAX_NAME_LEN+1];
} HI_RUNTIME_GROUP_BLOB_S;
```

[Member]

| Member | Description |
| --- | --- |
| pstBlob | Pointer to the BLOB |
| acOwnerName | Name of the BLOB owner |
| acBlobName | Name of the BLOB |

[Note]

None

[See Also]

- HI_RUNTIME_BLOB_S
- MAX_NAME_LEN

## HI_RUNTIME_GROUP_BLOB_ARRAY_S

[Description]

Defines the information of the BLOB array corresponding to a group.

[Syntax]

```
typedef struct hiRUNTIME_GROUP_BLOB_ARRAY_S
{
    HI_U32 u32BlobNum;
    HI_RUNTIME_GROUP_BLOB_S* pstBlobs;
} HI_RUNTIME_GROUP_BLOB_ARRAY_S, *HI_RUNTIME_GROUP_BLOB_ARRAY_PTR;
```

[Member]

| Member | Description |
| --- | --- |
| u32BlobNum | BLOB count |
| pstBlobs | BLOB pointer, pointing to the BLOB array |

[Note]

None

[See Also]

HI_RUNTIME_GROUP_BLOB_S

## HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_S

[Description]

Defines the information of the input BLOB array corresponding to a group.

[Syntax]

```
typedef HI_RUNTIME_GROUP_BLOB_ARRAY_S HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_S;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_GROUP_BLOB_ARRAY_S

## HI_RUNTIME_GROUP_DST_BLOB_ARRAY_S

[Description]

Defines the information of the output BLOB array corresponding to a group.

[Syntax]

```
typedef HI_RUNTIME_GROUP_BLOB_ARRAY_S HI_RUNTIME_GROUP_DST_BLOB_ARRAY_S;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_GROUP_BLOB_ARRAY_S

## HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_PTR

[Description]

Defines the pointer to the information of the input BLOB array corresponding to a group.

[Syntax]

```
typedef HI_RUNTIME_GROUP_BLOB_ARRAY_S*
HI_RUNTIME_GROUP_SRC_BLOB_ARRAY_PTR;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_GROUP_BLOB_ARRAY_S

## HI_RUNTIME_GROUP_DST_BLOB_ARRAY_PTR

[Description]

Defines the pointer to the information of the output BLOB array corresponding to a group.

[Syntax]

```
typedef HI_RUNTIME_GROUP_BLOB_ARRAY_S*
HI_RUNTIME_GROUP_DST_BLOB_ARRAY_PTR;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_GROUP_BLOB_ARRAY_S

## HI_RUNTIME_GROUP_SRC_BLOB_S

[Description]

Defines the input BLOB of a group.

[Syntax]

```
typedef HI_RUNTIME_GROUP_BLOB_S HI_RUNTIME_GROUP_SRC_BLOB_S;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_GROUP_BLOB_S

## HI_RUNTIME_GROUP_DST_BLOB_S

[Description]

Defines the output BLOB of a group.

[Syntax]

```
typedef HI_RUNTIME_GROUP_BLOB_S HI_RUNTIME_GROUP_DST_BLOB_S;
```

[Member]

None

[Note]

None

[See Also]

HI_RUNTIME_GROUP_BLOB_S

## HI_RUNTIME_MEM_CTRL_S

[Description]

Defines the memory management structure.

[Syntax]

```
typedef struct hiRUNTIME_MEM_CTRL_S
{
    HI_RUNTIME_AllocMem allocMem;
    HI_RUNTIME_FlushCache flushMem;
    HI_RUNTIME_FreeMem freeMem;
} HI_RUNTIME_MEM_CTRL_S;
```

[Member]

| Member | Description |
| --- | --- |
| allocMem | Callback function for memory allocation |
| flushMem | Callback function for memory refresh |
| freeMem | Memory release callback function |

[Note]

None

[See Also]

For details about the callback functions HI_RUNTIME_AllocMem, HI_RUNTIME_FlushCache, and HI_RUNTIME_FreeMem, see **hi_runtime_comm.h**.

## HI_RUNTIME_WK_INFO_S

[Description]

Defines the WK model.

[Syntax]

```
typedef struct hiRUNTIME_WK_MEM_S
{
    HI_CHAR acModelName[MAX_NAME_LEN+1];
    HI_RUNTIME_MEM_S stWKMemory;
} HI_RUNTIME_WK_INFO_S;
```

[Member]

| Member | Description |
| --- | --- |
| acModelName | Model name |
| stWKMemory | Memory information stored in the model |

[Note]

None

[See Also]

- HI_RUNTIME_MEM_S
- MAX_NAME_LEN

## HI_RUNTIME_WK_INFO_ARRAY_S

[Description]

Defines the WK model array information.

[Syntax]

```
typedef struct hiRUNTIME_WK_MEM_ARRAY_S
{
    HI_U32 u32WKNum;
    HI_RUNTIME_WK_INFO_S *pstAttrs;
} HI_RUNTIME_WK_INFO_ARRAY_S, *HI_RUNTIME_WK_INFO_ARRAY_PTR;
```

[Member]

| Member | Description |
| --- | --- |
| u32WKNum | Model count |
| pstAttrs | Model pointer, pointing to the model array |

[Note]

None

[See Also]

HI_RUNTIME_WK_INFO_S

## HI_RUNTIME_COP_ATTR_S

[Description]

Defines the attributes of a user-defined layer.

[Syntax]

```
typedef struct hiRUNTIME_COP_ATTR_S
{
    HI_CHAR acModelName[MAX_NAME_LEN+1];
    HI_CHAR acCopName[MAX_NAME_LEN+1];
    HI_U32 u32ConstParamSize;
    HI_VOID* pConstParam;
} HI_RUNTIME_COP_ATTR_S;
```

[Member]

| Member | Description |
| --- | --- |
| acModelName | Name of the model where the user-defined layer is located. |
| acCopName | Name of the user-defined layer |
| u32ConstParamSize | Size of the hyperparameter |
| pConstParam | Pointer to the content of the hyperparameter |

[Note]

None

[See Also]

MAX_NAME_LEN

## HI_RUNTIME_COP_ATTR_ARRAY_S

[Description]

Defines the array information of the user-defined layer.

[Syntax]

```
typedef struct hiRUNTIME_COP_ATTR_ARRAY_S
{
    HI_U32 u32CopNum;
    HI_RUNTIME_COP_ATTR_S *pstAttrs;
} HI_RUNTIME_COP_ATTR_ARRAY_S, *HI_RUNTIME_COP_ATTR_ARRAY_PTR;
```

[Member]

| Member | Description |
| --- | --- |
| u32CopNum | Number of user-defined layers |
| pstAttrs | User-defined layer pointer, pointing to the custom operator (COP) array |

[Note]

None

[See Also]

HI_RUNTIME_COP_ATTR_S

## HI_RUNTIME_CONNECTOR_ATTR_S

[Description]

Defines the attributes of the connector object.

[Syntax]

```
typedef struct hiRUNTIME_CONNECTOR_ATTR_S
{
    HI_CHAR acName[MAX_NAME_LEN +1];
    HI_RUNTIME_Connector_Compute pConnectorFun;
    HI_VOID *pParam;
} HI_RUNTIME_CONNECTOR_ATTR_S;
```

[Member]

| Member | Description |
| --- | --- |
| acName | Name of the connector object |
| pConnectorFun | Connector callback function |
| pParam | Pointer to the connector parameter content |

[Note]

For details about the HI_RUNTIME_Connector_Compute callback function, see **hi_runtime_comm.h**.

[See Also]

MAX_NAME_LEN

## HI_RUNTIME_CONNECTOR_ATTR_ARRAY_S

[Description]

Defines the array information of the connector object.

[Syntax]

```
typedef struct hiRUNTIME_CONNECTOR_ATTR_ARRAY_S
{
    HI_U32 u32ConnectorNum;
    HI_RUNTIME_CONNECTOR_ATTR_S *pstAttrs;
} HI_RUNTIME_CONNECTOR_ATTR_ARRAY_S, *HI_RUNTIME_CONNECTOR_ATTR_ARRAY_PTR;
```

[Member]

| Member | Description |
| --- | --- |
| u32ConnectorNum | Number of connector objects |
| pstAttrs | Connector object pointer, pointing to the connector array |

[Note]

None

[See Also]

HI_RUNTIME_CONNECTOR_ATTR_S

## HI_RUNTIME_GROUP_INFO_S

[Description]

Defines the Runtime group information.

[Syntax]

```
typedef struct hiRUNTIME_GROUP_INFO_S
{
    HI_RUNTIME_WK_INFO_ARRAY_S stWKsInfo;
    HI_RUNTIME_COP_ATTR_ARRAY_S stCopsAttr;
    HI_RUNTIME_CONNECTOR_ATTR_ARRAY_S stConnectorsAttr;
} HI_RUNTIME_GROUP_INFO_S;
```

[Member]

| Member | Description |
| --- | --- |
| stWKsInfo | Information of the WK object array structure |
| stCopsAttr | Information of the COP object array structure |
| stConnectorsAttr | Information of the connector object array structure |

[Note]

None

[See Also]

- HI_RUNTIME_WK_INFO_ARRAY_S
- HI_RUNTIME_COP_ATTR_ARRAY_S
- HI_RUNTIME_CONNECTOR_ATTR_S

## HI_RUNTIME_FORWARD_STATUS_CALLBACK_E

[Description]

Defines the status information of asynchronous forward.

[Syntax]

```
typedef enum hiRUNTIME_FORWARD_STATUS_CALLBACK_E
{
    HI_RUNTIME_FORWARD_STATUS_SUCC = 0x0,
    HI_RUNTIME_FORWARD_STATUS_FAIL,
    HI_RUNTIME_FORWARD_STATUS_ABORT,
    HI_RUNTIME_FORWARD_STATUS_BUTT
} HI_RUNTIME_FORWARD_STATUS_CALLBACK_E;
```

[Member]

| Member | Description |
| --- | --- |
| HI_RUNTIME_FORWARD_STATUS_SUCC | A forward success is returned. |
| HI_RUNTIME_FORWARD_STATUS_FAIL | A forward failure is returned. |
| HI_RUNTIME_FORWARD_STATUS_ABORT | The forward ends unexpectedly. |

[Note]

None

[See Also]

None

## HI_RUNTIME_Forward_Callback

[Description]

Defines the callback function of asynchronous forward.

[Syntax]

```
typedef HI_S32
(*HI_RUNTIME_Forward_Callback)(HI_RUNTIME_FORWARD_STATUS_CALLBACK_E
enEvent, HI_RUNTIME_GROUP_HANDLE hGroupHandle, HI_U64 u64SrcId,
HI_RUNTIME_GROUP_DST_BLOB_ARRAY_PTR pstDst);
```

[Member]

None

[Note]

None

[See Also]

- HI_RUNTIME_FORWARD_STATUS_CALLBACK_E
- HI_RUNTIME_GROUP_HANDLE
- HI_RUNTIME_GROUP_DST_BLOB_ARRAY_PTR

## HI_RUNTIME_Connector_Compute

[Description]

Defines the callback function of the connector.

[Syntax]

```
typedef HI_S32
(*HI_RUNTIME_Connector_Compute)(HI_RUNTIME_SRC_BLOB_ARRAY_S*
pstConnectorSrc, HI_RUNTIME_DST_BLOB_ARRAY_S* pstConnectorDst, HI_U64
u64SrcId, HI_VOID* pParam);
```

[Member]

None

[Note]

None

[See Also]

- HI_RUNTIME_SRC_BLOB_ARRAY_S
- HI_RUNTIME_DST_BLOB_ARRAY_S

## MAX_OPERAND_NAME_LEN

[Description]

Defines the maximum length of the operation description string for a plug-in.

[Syntax]

```
#define MAX_OPERAND_NAME_LEN 64
```

[Member]

None

[Note]

None

## HI_NodePlugin_Shape_S

[Description]

Defines the H, W, and C dimensions of the input and output of a plug-in.

[Syntax]

```
typedef struct hiNodePlugin_Shape_S {
    HI_S32 s32H;
    HI_S32 s32W;
    HI_S32 s32C;
} HI_NodePlugin_Shape_S;
```

[Member]

| Member | Description |
| --- | --- |
| s32H | Dimension H of the plug-in |
| s32W | Dimension W of the plug-in |
| s32C | Dimension C of the plug-in |

[Note]

None

[See Also]

None

## HI_NodePlugin_ElemType_E

[Description]

Defines the enumeration of data alignment modes of a plug-in.

[Syntax]

```
typedef enum hiNodePlugin_ElemType_E {
    ELEM_TYPE_U8,
    ELEM_TYPE_U16,
    ELEM_TYPE_U32
} HI_NodePlugin_ElemType_E;
```

[Member]

| Member | Description |
| --- | --- |
| ELEM_TYPE_U8 | 8-bit aligned |
| ELEM_TYPE_U16 | 16-bit aligned |
| ELEM_TYPE_U32 | 32-bit aligned |

[Note]

None

[See Also]

None

## HI_NodePlugin_Operand_S

[Description]

Defines the data and attributes of the input and output of a plug-in.

[Syntax]

```
typedef struct hiNodePlugin_Operand_S {
    HI_U64 u64Offset;  // addr
    HI_CHAR mName[MAX_OPERAND_NAME_LEN + 1];
    HI_NodePlugin_ElemType_E enElemType;
    HI_U32 u32Num;
    HI_U32 u32Stride;
    HI_NodePlugin_Shape_S stShape;
} HI_NodePlugin_Operand_S
```

[Member]

| Member | Description |
| --- | --- |
| u64Offset | Address for storing input and output data |
| mName | String for the alignment type |
| enElemType | Alignment type |

| Member | Description |
| --- | --- |
| u32Num | Number of consecutive memory blocks. If one frame of data corresponds to one block, then there are **u32Num** frames in the Blob. |
| u32Stride | Stride |
| stShape | H, W, and C dimensions of input and output data |

[Note]

None

[See Also]

- HI_NodePlugin_ElemType_E
- HI_NodePlugin_Shape_S

## HI_NodePlugin_NodeParam_S

[Description]

Defines the hyperparameter and training parameter of a plug-in.

[Syntax]

```
typedef struct HiNodeParam {
    HI_VOID *pParam;
    HI_U32 u32Size;
} HI_NodePlugin_NodeParam_S;
```

[Member]

| Member | Description |
| --- | --- |
| pParam | Pointer to the plug-in parameter |
| u32Size | Size of a plug-in parameter, in bytes |

[Note]

**pParam** is the pointer to a custom structure. This is a generic pointer, because the parameters of the plug-in are determined based on the network on which the plug-in runs. **u32Size** is the data size.

[See Also]

None

# 3.5 Error Codes

| Error Code | Macro Definition | Description |
| --- | --- | --- |
| 0xFF000F01 | HI_ERR_SVP_RUNTIME_ILLEGAL_STATE | The state is illegal. |
| 0xFF000F02 | HI_ERR_SVP_RUNTIME_MODEL_NOLOAD | The pointer is null. |
| 0xFF000F03 | HI_ERR_SVP_RUNTIME _NULL_PTR | The parameter is incorrect. |
| 0xFF000F04 | HI_ERR_SVP_RUNTIME_INVALID_PARAM | An SDK interface execution error occurred. |
| 0xFF000F05 | HI_ERR_SVP_RUNTIME_SDK_ERROR | The memory fails to be allocated for the reasons such as system memory insufficiency. |
| 0xFF000F06 | HI_ERR_SVP_RUNTIME_SDK_NOMEM | The model is not loaded. |

# 3.6 Proc Debugging Information

## 3.6.1 Overview

The debugging information is obtained from the proc file system on Linux. The information reflects the status of Runtime and can be used to locate and analyze problems.

[File Path]

**/proc/hisi/svprt/task**

[Information Viewing Method]

- Run a **cat** command such as **cat /proc/hisi/svprt/task** on the console or run a file operation command such as **cp /proc/hisi/svprt/task ./** to copy all the proc files to the current directory.
- Read the preceding files as common read-only files in the application by using functions such as fopen and fread.

## 3.6.2 Proc Information

[Debugging Information]

```
cat /proc/hisi/svprt/task
Left Seg Num Info
================================================================================
Left Unready Seg Num: 6
Left Ready Seg Num On NNIE: 0
Left Ready Seg Num On CPU: 0
```

```
NNIE_0 Cost Time / Total Cost Time / Use Rate
================================================================================
1802111        /3654734              /49%

NNIE_1 Cost Time / Total Cost Time / Use Rate
================================================================================
637106         /3693870              /17%
GroupName    FrameId ForwardTime   EnterPriQTime  EnterExecQTime
ExecStartTime      ExecEndTime TotalTime   SegInfo
================================================================================
rfcn_alexnet    0(24)   364099995   585    25    10     355993  356614
rfcn(Vop0)(171377, NNIE_0)
rfcn(proposal)(73911, CPU_2)
rfcn(Vop2)(11764, NNIE_1)
rfcn(Vop3)(8016, NNIE_0)
rfcn_conn_alexnet(rfcn_conn_alexnet)(4315, CPU_2)
alexnet(Vop0)(47226, NNIE_1)
```

[Analysis]

The information about the current NNIE status and the resource information are recorded, including the Runtime queue status, task status, and runtime status.

[Parameter Description]

| Parameter | | Description |
|---|---|---|
| Left Seg Num info | Left Unready Seg Num | Data segment unit that is not ready in the queue |
| | Left Ready Seg Num on NNIE | NNIE data segment unit that is ready in the queue |
| | Left Ready Seg Num on CPU | CPU data segment unit that is ready in the queue |
| NNIE info | Cost Time | Total data execution time of the NNIE |
| | Total Cost Time | Total NNIE consumption time, including the idle time and total data execution time of the NNIE |
| | Use Rate | NNIE usage rate |
| Group info | Group Name | Model group name |
| | FrameId | ID of the external input frame. The value inside the brackets is the ID of the internally recorded frame. |

| Parameter | | Description |
| --- | --- | --- |
| | ForwardTime | UTC time when HI_SVPRT_RUNTIME_ForwardGroupSync or HI_SVPRT_RUNTIME_ForwardGroupASync is called (in μs) |
| | EnterPriQTime | Time when the priority queue is entered, relative to **ForwardTime** (in μs) |
| | EnterExecQTime | Time when the execution queue is entered, relative to **EnterPriQTime** (in μs) |
| | ExecStartTime | Time when the first segment is executed, relative to **EnterExecQTime** (in μs) |
| | ExecEndTime | Time when a model group is executed, relative to **ExecStartTime** (in μs) |
| | TotalTime | Time from when HI_SVPRT_RUNTIME_ForwardGroupSync or HI_SVPRT_RUNTIME_ForwardGroupASync is called to when the execution of a model group is complete (in μs) |
| | SegInfo | Execution status of each segment of a model group, including execution time by segment (in μs) and the device where the execution happens |