Classification Level: Top Secret( ) Secret( ) Internal( ) Public(√)

RKNN API For RK356X User Guide

(Technology Department, Graphic Compute Platform Center)

Mark: [ ] Changing [√] Released	Version:	0.7.0
	Author:	HPC
	Completed Date:	April 1, 2021
	Reviewer:	Vincent
	Reviewed Date:	April 1, 2021

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

Revision History

Version	Modifier	Date	Modify Description	Reviewer
v0.6.0	HPC	Mar 2, 2021	Initial version	Vincent
V0.7.0	HPC	April 1, 2021	Update version number	Vincent

Table of Contents

1 Overview

RKNN SDK provides programming interfaces for RK3566 and RK3568 chip platforms with NPU, which can help users deploy RKNN models exported using RKNN-Toolkit2 and accelerate the implementation of AI applications.

2 Supported hardware platforms

This document applies to the following hardware platforms:

RK3566, RK3568

Note: RK356X is used in some parts of the document to indicate RK3566 and RK3568.

3 Instructions

3.1 RKNN SDK Development Process

Before using the RKNN SDK, users first need to use the RKNN-Toolkit2 tool to convert the user's model to the RKNN model.

After getting the RKNN model file, users can choose using C interface to develop the application. The following chapters will explain how to develop application based on the RKNN SDK on RK3566, RK3568 platform.

3.2 RKNN Linux Platform Development Instructions

3.2.1 RKNN API Library For Linux

For RK3566 and RK3568, the SDK library file is librknn_api.so under <sdk>/rknpu2/ directory.

3.2.2 Demo For Linux

The SDK provides MobileNet image classification, SSD object detection demos. These demos provide reference for developer to develop applications based on the RKNN SDK. The demo code is located in the <sdk>/rknpu2/examples directory. Let's take rknn_mobilenet_demo as an example to explain how to get started quickly.

1) Compile Demo Source Code

```
cd examples/rknn_mobilenet_demo
# set GCC_COMPILER in build-linux.sh to the correct compiler path
./build-linux.sh
```

2) Deploy to the RK3566 or RK3568 device

```
adb push install/rknn_mobilenet_demo_Linux /userdata/
```

3) Run Demo

```
adb shell
cd /userdata/rknn_mobilenet_demo_Linux/
./rknn_mobilenet_demo model/mobilenet_v1.rknn model/dog_224x224.jpg
```

3.3 RKNN Android platform development instructions

3.3.1 RKNN API Library For Android

For Android devices that need to pass the CTS test, you can use the RKNN API of the Android platform. The RKNN API of the Android platform is located in the vendor/rockchip/hardware/interfaces/neuralnetworks directory of the Android system SDK. When the Android system is compiled, some NPU-related libraries will be generated, as shown below:

```
/system/lib64/librknn_api.so
/system/lib64/librknnhal_bridge.rockchip.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0.so
/vendor/lib64/rockchip.hardware.neuralnetworks@1.0-adapter-helper.so
/vendor/lib64/librknnrt.so
/vendor/lib64/hw/rockchip.hardware.neuralnetworks@1.0-impl.so
```

For the application, you only need to link librknn_api.so. The interface of this library is RKNN C API.

3.3.2 Example Usage

Currently, the SDK provides examples of MobileNet image classification, SSD object detection. The demo code is located in the <sdk>/rknpu2/examples directory. Users can use NDK to compile the demo executed in the Android command line. Let's take rknn_mobilenet_demo as an example to explain how to use this demo on the Android platform.

4) Compile Demo Source Code

```
cd examples/rknn_mobilenet_demo
#set ANDROID_NDK_PATH under build-android.sh to the correct NDK path
./build-android.sh
```

5) Deploy to the RK3566 or RK3568 device

```
adb push install/rknn_mobilenet_demo_Android /data/
```

6) Run Demo

```
adb shell
cd /data/rknn_mobilenet_demo_Andorid/
./rknn_mobilenet_demo model/mobilenet_v1.rknn model/dog_224x224.jpg
```

3.4 RKNN C API

3.4.1 API process description

First, the user uses the rknn_inputs_set function to set the model input, and after the inference is over, use the rknn_outputs_get function to obtain the output of the inference. The API call process is shown in Figure 3-1.

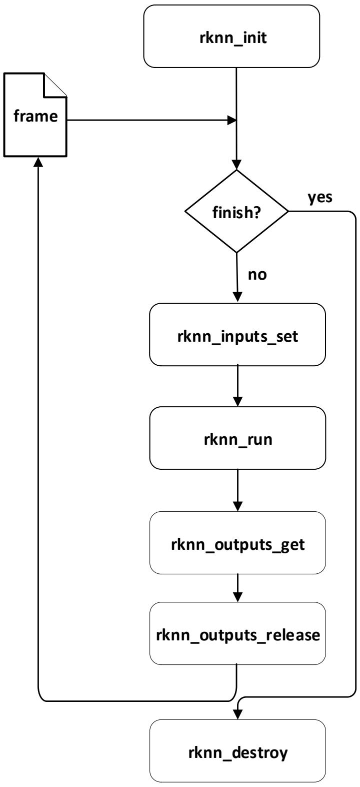

Figure 3-1 RKNN SDK API call process

3.4.1.1 API internal processing flow

During inference of RKNN model, the original data has to go through three processes: input processing, NPU running model, and output processing. In a typical picture inference scenario, assuming that the input data is a 3-channel picture and is in the NHWC layout format, the data processing flow at runtime is shown in Figure 3-2. At the API level, the rknn_inputs_set interface (when pass_through=0, see the rknn_input structure for details) includes the process of swapping color channel, normalization, quantization, and conversion of NHWC to NCHW/NC1HWC2. The rknn_outputs_get interface (when want_float=1, see rknn_output structure) contains the process of conversion of NC1HWC2 to NCHW and dequantization.

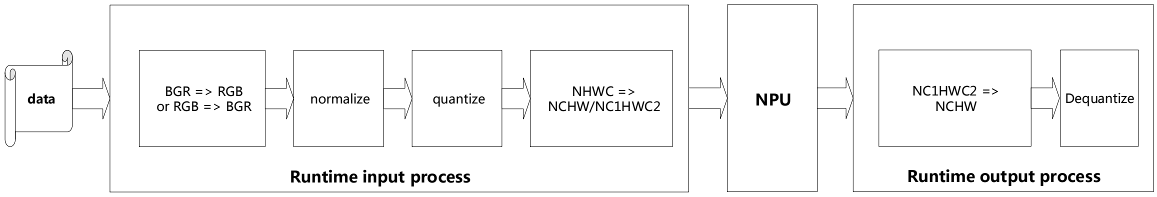

Figure 3-2 Complete image data processing flow

3.4.1.2 Quantification and dequantization

The quantization method, quantization data type and quantization parameters used in quantization and dequantization can be queried through the rknn_query interface.

Currently, the NPU of RK3566/RK3568 only supports asymmetric quantization, and does not support dynamic fixed-point quantization. The combination of data type and quantization method includes:

- int8 (asymmetric quantization)
- int16 (asymmetric quantization, not yet implemented)
- float16

Normally, the normalized data is stored in 32-bit floating point data. For conversion of 32-bit floating point data to 16-bit floating point data, please refer to the IEEE-754 standard. Assuming that the normalized 32-bit floating point data is D, the following describes the quantization process:

1) float32 to int8(asymmetric quantization)

Assuming that the asymmetric quantization parameter of the input tensor is S_q, ZP, the data quantization process is expressed as the following formula:

$$D_q = round(clamp(D / S_q + ZP,-128,127))$$

In the above formula, clamp means to limit the value to a certain range. round means rounding processing.

2) float32 to int16(asymmetric quantization)

Assuming that the asymmetric quantization parameter of the input tensor is S_q, ZP, the data quantization process is expressed as the following formula:

$$D_q = round(clamp(D / S_q + ZP,-32768,32767))$$

The dequantization process is the inverse process of quantization, and the inverse quantization formula can be deduced according to the above quantization formula, which will not be repeated here.

3.4.2 API Reference

3.4.2.1 rknn_init

The rknn_init function will create a *rknn_context* object, load the RKNN model, and perform specific initialization behavior based on the flag.

API	rknn_init
Description	Initialize rknn
Parameters	*rknn_context *context*:Pointer to *rknn_context* object. After the function is called, the context object will be assigned.
	*void *model*:Binary data for the RKNN model.
	uint32_t size:Model size
	uint32_t flag:A specific initialization flag. Currently RK3566 and RK3568 platforms do not support setting flags.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0);
```

3.4.2.2 rknn_destroy

The rknn_destroy function will release the *rknn_context* object and its associated resources.

API	rknn_destroy
Description	Destroy the rknn_context object and its related resources.
Parameters	*rknn_context context*: The rknn_context object to be destroyed.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
int ret = rknn_destroy (ctx);
```

3.4.2.3 rknn_query

The rknn_query function can query the information of model input and output tensor attribute and SDK version etc.

API	rknn_query
Description	Query the information about the model and the SDK.
Parameters	*rknn_context context*: The object of *rknn_contex*.
	rknn_query_cmd cmd: Query command.
	void info*: Structure object that stores the result of the query.
	uint32_t size: the size of the info Structure object.
Return	int: Error code (See RKNN Error Code).

Currently, the SDK supports the following query commands:

Query command	**Return result structure**	**Function**
RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	Query the number of input and output Tensor.
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	Query input Tensor attribute.
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	Query output Tensor attribute.
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	Query the running time of each layer of the network(RK3566 and RK3568 do not support this function temporarily).

RKNN_QUERY_SDK_VERSION	rknn_sdk_version	Query the SDK version.

Next we will explain each query command in detail.

1） Query the number of input and output Tensor

The *RKNN_QUERY_IN_OUT_NUM* command can be used to query the number of model input and output Tensor. You need to create the *rknn_input_output_num* structure object first.

Sample Code:

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num,
                        sizeof(io_num));
printf("model input num: %d, output num: %d\n", io_num.n_input,
                        io_num.n_output);
```

2） Query input Tensor attribute

The *RKNN_QUERY_INPUT_ATTR* command can be used to query the attribute of the model input Tensor. You need to create the *rknn_tensor_attr* structure object first.

Sample Code:

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
    input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]),
                        sizeof(rknn_tensor_attr));
}
```

3） Query output Tensor attribute

The *RKNN_QUERY_OUTPUT_ATTR* command can be used to query the attribute of the model output Tensor. You need to create the *rknn_tensor_attr* structure object first.

Sample Code:

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
        output_attrs[i].index = i;
        ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]),
                                    sizeof(rknn_tensor_attr));
}
```

4） Query the SDK version

The *RKNN_QUERY_SDK_VERSION* command can be used to query the version information of the RKNN SDK. You need to create the *rknn_sdk_version* structure object first.

Sample Code:

```
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version,
                                    sizeof(rknn_sdk_version));
printf("sdk api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

3.4.2.4 rknn_inputs_set

The input data of the model can be set by the rknn_inputs_set function. This function can support multiple inputs, each one is a *rknn_input* structure object. Developers needs to set these object field before passing in.

API	rknn_inputs_set
Description	Set the model input data.
Parameter	*rknn_context context*: The object of rknn_contex.
	uint32_t n_inputs: Number of inputs.
	rknn_input inputs[]: Array of rknn_input.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index = 0;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].size = img_width*img_height*img_channels;
inputs[0].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;

ret = rknn_inputs_set(ctx, 1, inputs);
```

3.4.2.5 rknn_run

The rknn_run function will perform a model inference. The input data need to be set by the *rknn_inputs_set* function before *rknn_run is* called.

API	rknn_run
Description	Perform a model inference.
Parameter	*rknn_context context*: The object of rknn_contex.
	rknn_run_extend extend*: Reserved for extension, currently not used, you can pass NULL.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
ret = rknn_run(ctx, NULL);
```

3.4.2.6 rknn_outputs_get

The rknn_outputs_get function can get the output data of the model. This function can get multiple output data. Each of these outputs is a *rknn_output* structure object, which needs to be created and set in turn before the function is called.

There are two ways to store buffers for output data:

1) Developer allocate and release buffers themselves. At this time, the *rknn_output.is_prealloc* needs to be set to 1, and the *rknn_output.buf* points to users' allocated buffer;

2) The other is allocated by SDK. At this time, the *rknn_output .is_prealloc* needs to be set to 0. After the function is executed, *rknn_output.buf* will be created and store the output data.

API	rknn_outputs_get
Description	Get model inference output data.
Parameter	*rknn_context context*: The object of rknn_context.
	uint32_t n_outputs: Number of output.
	rknn_output outputs[]: Array of rknn_output.
	rknn_run_extend extend*: Reserved for extension, currently not used, you can pass NULL.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
        outputs[i].want_float = 1;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

3.4.2.7 rknn_outputs_release

The rknn_outputs_release function will release the relevant resources of the *rknn_output* object.

API	rknn_outputs_release
Description	Release the rknn_output object
Parameter	*rknn_context context*: rknn_context object
	uint32_t n_outputs: Number of output.
	rknn_output outputs[]: rknn_output array to be release.
Return	int: Error code (See RKNN Error Code).

Sample Code:

```
ret = rknn_outputs_release(ctx, io_num.n_output, outputs);
```

3.4.3 RKNN Data Structcture Definition

3.4.3.1 rknn_input_output_num

The structure *rknn_input_output_num* represents the number of input and output Tensor， The following table shows the definition:

Field	Type	Meaning
n_input	uint32_t	The number of input tensor
n_output	uint32_t	The number of output tensor

3.4.3.2 rknn_tensor_attr

The structure *rknn_tensor_attr* represents the attribute of the model's Tensor. The following table shows the definition:

Field	Type	Meaning
index	uint32_t	Indicates the index position of the input and output Tensor.
n_dims	uint32_t	The number of Tensor dimensions.
dims	uint32_t[]	Values for each dimension.
name	char[]	Tensor name.
n_elems	uint32_t	The number of Tensor data elements.
size	uint32_t	The memory size of Tensor data.
fmt	rknn_tensor_format	The format of Tensor dimension, has the following format: **RKNN_TENSOR_NCHW** **RKNN_TENSOR_NHWC**

type	rknn_tensor_type	Tensor data type, has the following data types: **RKNN_TENSOR_FLOAT32** **RKNN_TENSOR_FLOAT16** **RKNN_TENSOR_INT8** **RKNN_TENSOR_UINT8** **RKNN_TENSOR_INT16**
qnt_type	rknn_tensor_qnt_type	Tensor Quantization Type, has the following types of quantization: **RKNN_TENSOR_QNT_NONE:** Not quantified; **RKNN_TENSOR_QNT_DFP**: Dynamic fixed point quantization; **RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC**: Asymmetric quantification.
fl	int8_t	RKNN_TENSOR_QNT_DFP quantization parameter
zp	uint32_t	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC quantization parameter.
scale	float	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC quantization parameter.

3.4.3.3 rknn_input

The structure *rknn_input* represents a data input to the model, used as a parameter to the rknn_inputs_set function. The following table shows the definition:

Field	Type	Meaning
index	uint32_t	The index position of this input.
buf	void*	Point to the input data Buffer.

size	uint32_t	The memory size of the input data Buffer.
pass_through	uint8_t	When set to 1, buf will be directly set to the input node of the model without any pre-processing.
type	rknn_tensor_type	The type of input data.
fmt	rknn_tensor_format	The format of input data.

3.4.3.4 rknn_tensor_mem

The structure *rknn_tensor_mem* represents the storage state information after tensor initialization. At present, RK3566 and RK3568 have not yet provided an interface to use this structure, and they are temporarily reserved for subsequent expansion. The definition of the structure is shown in the following table:

Field	**Type**	**Meaning**
logical_addr	void*	The virtual address of this input.
physical_addr	uint64_t	The physical address of this input.
fd	int32_t	The fd of this input.
size	uint32_t	The memory size of the input tensor.
handle	uint32_t	The handle of this input.
priv_data	void*	Reserved data.
reserved_flag	int32_t	Reserved flag.

3.4.3.5 rknn_output

The structure *rknn_output* represents a data output of the model, used as a parameter to the rknn_outputs_get function. The following table shows the definition:

Field	**Type**	**Meaning**
want_float	uint8_t	Indicates if the output data needs to be converted to float type.

is_prealloc	uint8_t	Indicates whether the Buffer that stores the output data is pre-allocated.
index	uint32_t	The index position of this output.
buf	void*	Pointer which point to the output data Buffer.
size	uint32_t	Output data Buffer memory size.

3.4.3.6 rknn_perf_detail

The structure *rknn_perf_detail* represents the performance details of the model. The following table shows the definition:

Field	Type	Meaning
perf_data	char*	Performance details include the run time of each layer of the network.
data_len	uint64_t	The Length of perf_data.

3.4.3.7 rknn_sdk_version

The structure *rknn_sdk_version* is used to indicate the version information of the RKNN SDK. The following table shows the definition:

Field	Type	Meaning
api_version	char[]	SDK API Version information.
drv_version	char[]	Driver version information.

3.4.4 RKNN Error Code

The return code of the RKNN API function is defined as shown in the following table.

Error Code	Message
RKNN_SUCC（0）	Execution is successful
RKNN_ERR_FAIL (-1)	Execution error

RKNN_ERR_TIMEOUT (-2)	Execution timeout
RKNN_ERR_DEVICE_UNAVAILABLE (-3)	NPU device is unavailable
RKNN_ERR_MALLOC_FAIL (-4)	Memory allocation is failed
RKNN_ERR_PARAM_INVALID (-5)	Parameter error
RKNN_ERR_MODEL_INVALID (-6)	RKNN model is invalid
RKNN_ERR_CTX_INVALID (-7)	rknn_context is invalid
RKNN_ERR_INPUT_INVALID (-8)	rknn_input object is invalid
RKNN_ERR_OUTPUT_INVALID (-9)	rknn_output object is invalid
RKNN_ERR_DEVICE_UNMATCH (-10)	Version does not match
RKNN_ERR_INCOMPATILE_OPTIMIZATION_LEVEL_VERSION (-12)	This RKNN model use optimization level mode, but not compatible with current driver.
RKNN_ERR_TARGET_PLATFORM_UNMATCH (-13)	This RKNN model don’t compatible with current platform.