Tina Linux NPU VIPLite API Description

Version	Date	Reviser	Content
1.0	2022.09.14	AWA1911	First edition, npu function API description

vip_init()	**Description:** Initializes the VIP hardware and the VIPLite software environment. In detail, this API resets and initializes the VIP hardware to a ready state to accept commands. It also initializes the software environment, such as video memory heap, power management, and MMU table. Call this API before the application calls any other VIPLite API to use the VIP hardware. After the application completes, call vip_destroy(). You can call the vip_init() API multiple times. However, the number of vip_destroy() calls should match the number of vip_init() calls. Only the first vip_init() call and the last vip_destroy() call are executed. Other vip_init() and vip_destroy() calls in between do not trigger initialize or destroy operation. **Syntax:** vip_status_e **vip_init**(void); **Parameters:** None **Returns:** vip_status_e
vip_destroy()	**Description:** Terminates the VIPLite driver, releases the resources requested by vip_init(), and shuts down the VIP hardware. Call this API after an application completes. After this API is executed, call vip_init()

	before any other VIPLite API. You can call the vip_init() API multiple times. However, the number of vip_destroy() calls should match the number of vip_init() calls. Only the first vip_init() call and the last vip_destroy() call are executed. Other vip_init() and vip_destroy() calls in between do not trigger initialize or destroy operation. **Syntax:** vip_status_e **vip_destroy**(void); **Parameters:** None **Returns:** vip_status_e
vip_create_network()	**Description:** Creates a network from the given binary. The binary is binary large object (BLOB) data generated by the graph binary generator. The VIPLite driver can interpret it to create a network object. **Syntax:** vip_status_e **vip_create_network**(void *data, vip_uint32_t size_of_data, vip_enum type, vip_network *network); **Parameters:** IN *data The pointer to the graph binary. IN size_of_data The size in bytes of the graph binary. IN type The network type. The supported types are defined in the vip_create_network_type_e enumeration.

	OUT *network The pointer to receive the created network object if the network is created successfully. If the network creation fails, VIP_NULL is returned. **Returns:** vip_status_e
vip_weak_dup_network( )	**Description:** Creates a new network by duplicating the command buffer of an existing network (source). The network coefficients are not duplicated. Before you call this API, make sure that the source network is prepared by calling vip_prepare_network(). This API is useful when you need to add a multi-input network to a network group. For details, see Section *3.2.19, vip_add_network()*. **Note:** Do not destroy the source network if duplicated networks are still in use. For more information, see Section *3.2.3, vip_destroy_network()*. **Syntax:** vip_status_e **vip_weak_dup_network**(vip_network network vip_network *dup_network); **Parameters:** IN network An opaque handle to the source network. OUT dup_network An opaque handle to the target network. **Returns:** vip_status_e
vip_destroy_network()	**Description:** Destroys a network. This API releases all relevant resources allocated to the network. After this API is executed for a specified network, the command buffers of the networks duplicated from the specified network are also released. However, the coefficients of the duplicated networks are retained. **Syntax:**

	vip_status_e **vip_destroy_network**(vip_network network); **Parameters:** IN network An opaque handle to the network to be destroyed. **Returns:** vip_status_e
vip_query_network()	**Description:** Queries a property of a network. **Syntax:** vip_status_e **vip_query_network**(vip_network network, vip_enum property, void *value); **Parameters:** IN network An opaque handle to the network to be queried. IN property The network property to be queried. The following properties are available for query: VIP_NETWORK_PROP_LAYER_COUNT VIP_NETWORK_PROP_INPUT_COUNT VIP_NETWORK_PROP_OUTPUT_COUNT VIP_NETWORK_PROP_NETWORK_NAME VIP_NETWORK_PROP_READ_REG_IRQ VIP_NETWORK_PROP_ADDRESS_INFO VIP_NETWORK_PROP_MEMORY_POOL_SIZE VIP_NETWORK_PROP_PROFILING For details, see Section *2.2.5, vip_network_property_e*. OUT *value A pointer in memory to store the returned property value. The data type of the value varies according to the property queried. **Returns:**
vip_set_network()	**Description:**

	Configures a network. Before you can run the network, you need to validate the configurations by calling vip_prepare_network(). **Syntax:** vip_status_e **vip_set_network**(vip_network network, vip_enum property, void *value); **Parameters:** IN network An opaque handle to the network to be configured. IN property The network property to be configured. The supported properties are: VIP_NETWORK_PROP_CHANGE_PPU_PARAM VIP_NETWORK_PROP_SET_MEMORY_POOL VIP_NETWORK_PROP_SET_DEVICE_ID VIP_NETWORK_PROP_SET_PRIORITY For details, see Section *2.2.5, vip_network_property_e*. IN *value A pointer in memory to the property value. **Returns:** vip_status_e
vip_prepare_network()	**Description:** Validates the configurations of a network. This API allocates internal memory resources to the network, deploys resources for all operations to the internal memory pool, allocates and patches a command buffer for the resources in the internal memory pool. After this API is executed successfully, the network is considered prepared for running on VIP hardware. Prior to this API, use the vip_set_network() API to configure the network. If this API is called more than once with the network configurations unchanged, the driver silently ignores the API calls except for the first call. **Syntax:**

	vip_status_e **vip_prepare_network**(vip_network network); **Parameters:** IN network An opaque handle to the network to be prepared. **Returns:** vip_status_e
vip_query_input()	**Description:** Queries the properties of a network input. **Syntax:** vip_status_e **vip_query_input**(vip_network network, vip_uint32_t index, vip_enum property, void *value); **Parameters:** IN network An opaque handle to the network to be queried. IN index The index of the network input to be queried. IN property The input buffer property to be queried. The following properties are available for query: VIP_BUFFER_PROP_QUANT_FORMAT VIP_BUFFER_PROP_NUM_OF_DIMENSION VIP_BUFFER_PROP_SIZES_OF_DIMENSION VIP_BUFFER_PROP_DATA_FORMAT VIP_BUFFER_PROP_FIXED_POINT_POS VIP_BUFFER_PROP_TF_SCALE VIP_BUFFER_PROP_TF_ZERO_POINT VIP_BUFFER_PROP_NAME VIP_BUFFER_PROP_DATA_TYPE For details, see Section *2.2.6, vip_buffer_property_e*.

	OUT *value A pointer in memory to store the returned property value. **Returns:** vip_status_e
vip_query_output()	**Description:** Queries a property of a network output. **Syntax:** vip_status_e **vip_query_output**(vip_network network, vip_uint32_t index, vip_enum property, void *value); **Parameters:** IN network An opaque handle to the network to be queried. IN index The index of the network output to be queried. IN property The output buffer property to be queried. The following properties are available for query: VIP_BUFFER_PROP_QUANT_FORMAT VIP_BUFFER_PROP_NUM_OF_DIMENSION VIP_BUFFER_PROP_SIZES_OF_DIMENSION VIP_BUFFER_PROP_DATA_FORMAT VIP_BUFFER_PROP_FIXED_POINT_POS VIP_BUFFER_PROP_TF_SCALE VIP_BUFFER_PROP_TF_ZERO_POINT VIP_BUFFER_PROP_NAME VIP_BUFFER_PROP_DATA_TYPE For details, see Section *2.2.6, vip_buffer_property_e*. OUT *value A pointer in memory to store the returned property value. **Returns:** vip_status_e

vip_set_input()	**Description:** Attaches an input buffer to a network. When attaching the input buffer to the network, the VIPLite driver patches the network command buffer to fill in the input buffer. You can also call this API to update the input buffers. The update takes effect from the next network execution. Before attaching input buffers to a network, make sure that the network is prepared using the vip_prepare_network() API. Before using vip_run_network() to run a network, make sure that each valid network input is attached with a buffer. Otherwise, VIP_ERROR_MISSING_INPUT_OUTPUT is returned once the vip_run_network() API is called. **Syntax:** vip_status_e **vip_set_input**(vip_network network, vip_uint32_t index, vip_buffer input); **Parameters:** IN network An opaque handle to the network to be configured. IN index The index of the network input to be configured. IN input An opaque handle to the buffer to be attached to the network input. **Returns:** vip_status_e
vip_set_output()	**Description:** Attaches the output buffer to a network. When attaching the output buffer to the network, the VIPLite driver patches the network command buffer to fill in the output buffer. You can also call this API to update the output buffer. The update takes effect from the next network execution.

	Before attaching the output buffer to a network, make sure that the network is prepared using the vip_prepare_network() API. Before using vip_run_network() to run a network, make sure that the network output is attached with a buffer. Otherwise, VIP_ERROR_MISSING_INPUT_OUTPUT is returned once the vip_run_network() API is called. **Syntax:** vip_status_e **vip_set_output**(vip_network network, vip_uint32_t index, vip_buffer output); **Parameters:** IN network An opaque handle to the network to be configured. IN index The index of the network output to be configured. IN output An opaque handle to the buffer to be attached to the network output. **Returns:** vip_status_e
vip_run_network()	**Description:** Commits an execution task for the network. The VIP hardware executes the task of the highest priority among the committed tasks. You can call this API multiple times. The API execution status is returned after the VIP hardware completes the execution. If you need the status to be immediately returned without waiting for the execution to complete, use the vip_trigger_network() API. To set the network priority, use the vip_set_network() API. Before running a network, make sure that the network is prepared using vip_prepare_network(). In addition, make sure that each network input and the network output are attached with buffers by using

	vip_set_input() and vip_set_output(). Otherwise, VIP_ERROR_MISSING_INPUT_OUTPUT is returned once vip_run_network() is called. To run multiple networks in a group, use vip_run_group() or vip_trigger_group(). **Syntax:** vip_status_e **vip_run_network**(vip_network network); **Parameters:** IN network An opaque handle to the network to be run. **Returns:** vip_status_e
vip_trigger_network()	**Description:** Commits an execution task for the network. The VIP hardware executes the task of the highest priority among the committed tasks. You can call this API multiple times. The API execution status is returned immediately without waiting for the hardware to complete the execution. To acquire the status, call vip_wait_network() for synchronization. If you need the status to be returned after the VIP hardware completes the execution, use the vip_run_network() API. To set the network priority, use the vip_set_network() API. Before running a network, make sure that the network is prepared using vip_prepare_network(). In addition, make sure that each network input and the network output are attached with buffers by using vip_set_input() and vip_set_output(). Otherwise, VIP_ERROR_MISSING_INPUT_OUTPUT is returned once vip_trigger_network() is called. To run multiple networks in a group, use vip_run_group() or vip_trigger_group(). **Syntax:** vip_status_e **vip_trigger_network**(vip_network network); **Parameters:** IN network An opaque handle to the network to be executed.

	Returns: vip_status_e
vip_wait_network()	**Description:** Waits for the VIP hardware to finish the inference for the specified network. Call this API after vip_trigger_network() is called. **Syntax:** vip_status_e **vip_wait_network**(vip_network network); **Parameters:** IN network An opaque handle to the network. **Returns:** vip_status_e
vip_finish_network()	**Description:** Releases the resources of a prepared network. After this API is called, all internal memory resources allocated to the network are released with the network not destroyed. If the network is no long needed, destroy it by using the vip_destroy_network() API. Call the vip_finish_network() API to finish a prepared network only if the network is no longer used or the remaining system resources are limited for other networks. If the network is still needed, do not call this API because the preparation of a network is time consuming. After a vip_finish_network() call is successfully executed for a prepared network, repeated calls are silently ignored until the network is re-prepared with the vip_prepare_network() API. For an unprepared network, vip_finish_network() calls are silently ignored. **Important:** Do not call the vip_finish_network() API for a running network. **Syntax:** vip_status_e **vip_finish_network**(vip_network network); **Parameters:** IN network An opaque handle to the network to be finished. **Returns:** vip_status_e

vip_create_buffer()	**Description:** Creates a VIP buffer of the specified size with no padding between lines, slices, or batches. **Syntax:** vip_status_e **vip_create_buffer**(vip_buffer_create_params_t *create_param, vip_uint32_t size_of_param, vip_buffer *buffer); **Parameters:** IN *create_param The pointer to a vip_buffer_create_params_t structure. IN size_of_param The size of the data structure created by *create_param in bytes. OUT *buffer The pointer to receive the created buffer object if the VIP buffer is created successfully. If the VIP buffer creation fails, VIP_NULL is returned. **Returns:** vip_status_e If VIP_SUCCESS is returned, a VIP buffer is created successfully. If VIP_ERROR_<*error_type*> is returned, no buffer is created.
vip_create_buffer_from_handle()	**Description:** Creates a VIP buffer from a handle and maps the handle associated physical address to the buffer. Before using this API, enable the VIP MMU. Otherwise, the API returns VIP_ERROR_FAILURE. **Syntax:** vip_status_e **vip_create_buffer_from_handle**(vip_buffer_create_params_t *create_param, vip_ptr handle_logical, vip_uint32_t handle_size, vip_buffer *buffer

	) **Parameters:** IN *create_param A pointer to a vip_buffer_create_params_t structure. IN handle_logical The address of the handle from which the new VIP buffer is to be created. For a non-real-time Linux operating system, specify a logical address. The address is allocated by the Linux malloc() function. For a real-time operating system, specify a physical address. **Note:** Address alignment to 64 bytes is recommended. IN handle_size The size of the memory to which the handle points. **Note:** Size alignment to 64 bytes is recommended. OUT *buffer The pointer to receive the created buffer object if the VIP buffer is created successfully. If the VIP buffer creation fails, VIP_NULL is returned. **Returns:** vip_status_e If VIP_SUCCESS is returned, a VIP buffer is created successfully. If VIP_ERROR_<*error_type*> is returned, no buffer is created.
	vip_destroy_buffer() **Description:** Destroys a VIP buffer and frees the memory used by the buffer. **Syntax:** vip_status_e **vip_destroy_buffer**(vip_buffer buffer); **Parameters:** IN buffer The opaque handle of the buffer to be destroyed. **Returns:** vip_status_e
vip_map_buffer()	**Description:** Creates a pointer to the specified VIP buffer. The pointer can be used by applications to access the buffer.

	Syntax: void ***vip_map_buffer**(vip_buffer buffer); **Parameters:** IN buffer The opaque handle of the buffer for which a pointer is to be created. **Returns:** A pointer to the buffer that applications can use to read or write the buffer data
vip_unmap_buffer()	**Description:** Releases the pointer that applications use to access a VIP buffer. **Syntax:** vip_status_e ***vip_unmap_buffer**(vip_buffer buffer); **Parameters:** IN buffer The opaque handle of the buffer whose pointer is to be released. **Returns:** vip_status_e
vip_get_buffer_size()	**Description:** Retrieves the size of the buffer in bytes. **Syntax:** vip_uint32_t **vip_get_buffer_size**(vip_buffer buffer); **Parameters:** IN buffer The opaque handle of the buffer whose size is requested. **Returns:** vip_uint32_t The buffer size in bytes.
vip_flush_buffer()	**Description:** Flushes or invalidates the cache of a VIP buffer created from the vip_create_buffer()

	or vip_create_buffer_from_handle() API. Call this API in the following scenarios: If the VIP buffer in use contains a CPU cache, flush the cache with this API before calling vip_run_network(). After return from vip_wait_network() or vip_run_network(), use this API to invalidate the buffer cache. **Syntax:** vip_status_e **vip_flush_buffer**(vip_buffer buffer, vip_buffer_operation_type_e type); **Parameters:** IN buffer The opaque handle of the buffer whose cache is to be flushed or invalidated. IN type The buffer cache operation to be executed on the buffer. The buffer catch operations are defined in the vip_buffer_operation_type_e enumeration. **Returns:** vip_status_e
Running a Single Network	The procedure to run a network is detailed as follows: 1. Call vip_init() to initialize the VIPLite engine, including the software and the hardware. This API resets the hardware to get it ready for use and initializes the software resources. It sets up the video memory and other hardware resources, such as interrupt and register memory, for VIPLite to use. 2. Read the network binary graph (NBG) data from a file or memory. 3. Create a network with the vip_create_network() API. This API performs a sanity check on the NBG data. Therefore, it is recommended that you check the returned error code to verify that the network is successfully generated. 4. Query the input and output properties by using vip_query_input() and vip_query_output(). This step is recommended to avoid errors caused by mismatched input or output

properties.

5. Create input and output buffers with the following APIs: vip_create_buffer(), vip_create_buffer_from_handle(), vip_create_buffer_from_physical(), and vip_create_buffer_from_fd().

6. (Optional) Configure the network properties by using vip_set_network().

7. Prepare the network command buffer with the vip_prepare_network() API.

It is recommended that you check the error code returned by this API. This is because the API may fail because
of resource limitation, for example, out of memory.

8. Load data from the input and output buffers with the assistance of the vip_map_buffer() API.

9. Attach the input and output buffers to the network by using vip_set_input() and vip_set_output().

10. Run the network with the vip_run_network() or vip_trigger_network() API.

If vip_run_network() is used, it returns together with the result after the VIP hardware completes the
execution.

If vip_trigger_network() is used, it immediately returns without waiting for the VIP hardware to
complete execution. This optimizes the CPU usage if the CPU workload is heavy.

In this case, when the CPU requires the API result, call vip_wait_network() for synchronization.

Note: Multiple networks can be created. However, only one network can be run at a time. The application runs the
networks one by one according to the network priorities configured with the vip_set_network() API.

11. Flush the network execution result from the CPU cache to the output buffer by using vip_flush_buffer().

12. Check the network execution result from the output buffer with the assistance of vip_map_buffer().

13. (Optional) Repeat steps 6 to 12 to run the network multiple times.

14. Call vip_finish_network() to free the internal memory allocated to the network.

15. Call vip_destroy_network() to release all the other resources allocated to the network.

16. Call vip_destroy_buffer() to free the memory allocated to the input and output buffers.

17. Call vip_destroy() to release the VIPLite resources and exit.