EMBEDEEP
壹 普 智 慧

# EEP-TPU 应用开发接口使用手册

## eep-ug053 (v0.1.0)



**2023-02-01**

**厦门壹普智慧科技有限公司**

修改历史:

| 版本 | 日期 | 描述 | 作者 |
|---|---|---|---|
| 0.1.0 | 2023-02-01 | 初始版本 | 何 xx |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 目　　录

# 1、前言

本文档是“libeeptpu_pub.so”库文件的接口说明，通过该库文件，可对 EEP-TPU 进行操作。

libeeptpu_pub 库必须配合 public 版本的 bin 文件。在 EEP-TPU 编译器编译神经网络时，加上“--public_bin”参数，即可编译生成“.pub.bin”格式的 bin 文件。

## 1.1 编译环境

**arm-32 位**

交叉编译工具链：arm-linux-gnueabihf-g++，gcc 版本 6.3.1 20170404 （Linaro GCC 6.3-2017.05），交叉编译工具链下载地址：

http://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/arm-linux-gnueabihf/gcc-linaro-6.3.1-2017.05-i686_arm-linux-gnueabihf.tar.xz

**aarch-64 位**

交叉编译工具链：aarch64-linux-gnu-g++，gcc 版本 6.3.1 20170404 （Linaro GCC 6.3-2017.05），交叉编译工具链下载地址：

http://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu.tar.xz

**X86 平台**

编译器：g++，gcc version 7.5.0（Ubuntu 7.5.0-3ubuntu1~18.04）（Ubuntu 18.04.6 LTS）

**注意：程序使用的交叉编译工具链需与上述版本一致，否则可能会出现兼容性问题。库文件 libeeptpu_pub.so v0.7.0 版本及以上，编译器需配套使用 eeptpu_compiler v2.4.1 版本及以上；EEP-TPU 硬件需使用 v0.8.4 版本。**

# 2、API 接口

## 2.1 初始化

初始化的函数，调用一次即可。

### 2.1.1 初始化库文件调用接口

| | |
|---|---|
| 函数 | EEPTPU* init(); |
| 功能 | 初始化库文件调用接口，在使用所有类成员函数前需先调用本初始化函数。 |
| 参数 | 无 |
| 返回 | 返回 EEPTPU 类指针 |
| 示例 | *EEPTPU *tpu = NULL; // 可声明为全局变量*<br>*if (tpu == NULL) tpu = tpu->init(); // 可放置于 main 函数开头进行初始化* |

### 2.1.2 设置接口类型

| | |
|---|---|
| 函数 | int eeptpu_set_interface(int interface_type); |
| 功能 | 设置数据交互接口类型。<br>在 init()函数执行后，加载 EEPTPU BIN 文件之前，需调用本函数来设置接口类型。 |
| 参数 | 接口类型：<br>可用接口类型有：eepInterfaceType_SOC 和 eepInterfaceType_PCIE，请根据开发板实际情况进行选择。若程序与 EEP-TPU 同在板载内存，则设置为 eepInterfaceType_SOC；若使用 PCIE 插槽，程序在主机端通过 PCIE 与 EEP-TPU 进行交互，则设置为 eepInterfaceType_PCIE。<br>*enum {*<br>*eepInterfaceType_NONE = 0,*<br>*eepInterfaceType_SOC = 1,*<br>*eepInterfaceType_PCIE,*<br>*eepInterfaceType_EOF, /* dummy data. */*<br>*};* |
| 返回 | 0: 成功 |

| | |
|---|---|
| | < 0: 失败。（错误码参考附件 1） |
| 示例 | *tpu->eeptpu_set_interface(eepInterfaceType_SOC);* |

### 2.1.3 设置 PCIE 设备名

| | |
|---|---|
| 函数 | int eeptpu_set_interface_info_pcie(const char* dev_reg, const char* dev_h2c, const char* dev_c2h); |
| 功能 | 设置 PCIE 设备名，适用于 Xilinx 的板卡。 |
| 参数 | **dev_reg**：PCIE 寄存器设备名；<br>**dev_h2c**: PCIE host to card 的设备名；<br>**dev_c2h**: PCIE card to host 的设备名。 |
| 返回 | 0：成功<br>< 0：失败。（错误码参考附件 1） |
| 示例 | *tpu->eeptpu_set_interface_info_pcie("/dev/xdma0_user","/dev/xdma0_h2c_0","/dev/xdma0_c2h_0");* |

### 2.1.4 设置 EEP-TPU 可访问的内存起始地址

| | |
|---|---|
| 函数 | int eeptpu_set_tpu_mem_base_addr(unsigned long mem_base_addr); |
| 功能 | 用于 eepInterfaceType_PCIE 接口模式。<br>eepInterfaceType_SOC 模式下，可以不配置。 |
| 参数 | **mem_base_addr**: EEP-TPU 可访问和使用的内存起始地址。<br>PCIE 模式下，在 EEP-TPU 端，对 PCIE 设备读写时的起始内存地址。 |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |
| 示例 | *tpu->eeptpu_set_tpu_mem_base_addr(0x00000000);* |

### 2.1.5 设置 EEP-TPU 寄存器组的信息

| | |
|---|---|
| 函数 | int eeptpu_set_tpu_reg_zones(std::vector<struct EEPTPU_REG_ZONE>& regzones); |
| 功能 | 配置 EEP-TPU 模块的寄存器组的信息。<br>对于单核 SOC 模式下，库文件里已经配置了默认的基址，一般情况下可不用调用此 |

| | |
|---|---|
| | 函数；若芯片设计人员更改了默认的基址，则需告知软件开发人员调用此函数进行配置；对于多核情况，需要调用此函数对各个核进行寄存器的配置。 |
| 参数 | **regzones**: EEP-TPU 模块的寄存器组的配置信息。包含各个 EEP-TPU core 的 ID、寄存器基址、偏移地址、寄存器区域大小。<br>其中，<br>寄存器基址：对于 **SOC** 接口模式，传入的是**实际寄存器地址**。对于 **PCIE** 接口模式，传入的是 EEP-TPU 模块相对于整体寄存器的**偏移地址**；例如，整体寄存器起始地址为 0x60000000，EEP-TPU 模块的 Core0 寄存器从 0x60020000 开始，那么可以配置这里的寄存器基址为 0x00020000。 |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |
| 示例 | **# SOC 模式：**<br>*vector<struct EEPTPU_REG_ZONE> regzones;*<br>*struct EEPTPU_REG_ZONE zone;*<br>*zone.core_id = 0;*<br>*zone.addr = 0xA0000000;*<br>*zone.size = 0x1000;*<br>*regzones.push_back(zone);*<br>*zone.core_id = 1;*<br>*zone.addr = 0xA0040000;*<br>*zone.size = 0x1000;*<br>*regzones.push_back(zone);*<br>*tpu->eeptpu_set_tpu_reg_zones(regzones);*<br>**# PCIE 模式：**<br>*vector<struct EEPTPU_REG_ZONE> regzones;*<br>*struct EEPTPU_REG_ZONE zone;*<br>*zone.core_id = 0;*<br>*zone.addr = 0x00020000;*<br>*zone.size = 256*1024;*<br>*regzones.push_back(zone);*<br>*tpu->eeptpu_set_tpu_reg_zones(regzones);* |
| 备注 | # PCIE 模式下如何获取寄存器空间的大小：<br>**方法 1：**<br>第一步，加载 Xilinx PCIE xdma 驱动； |

| | |
|---|---|
| | 第二步，在命令行窗口执行命令：lspci -v；<br>第三步，查找有“Kernel driver in use: xdma”的设备：<br>*01:00.0 Memory controller: Xilinx Corporation Device 8028*<br>*Subsystem: Xilinx Corporation Device 0007*<br>*Flags: bus master, fast devsel, latency 0, IRQ 16*<br>***Memory at a1000000 (32-bit, non-prefetchable) [size=2M]***<br>***Memory at a1200000 (32-bit, non-prefetchable) [size=64K]***<br>*Capabilities: <access denied>*<br>*Kernel driver in use: xdma*<br>查看 Memory 所在行，通常，第 1 行即表示 PCIE 设备的 Bar0，即我们需要的寄存器所在位置，获取其 size 为 2M，转换为十进制（2097152）或 16 进制（0x200000）作为参数传入本函数。<br>**方法 2：**<br>也可以在加载完 Xilinx PCIE xdma 驱动后，通过 dmesg 命令，找到下面这段日志：<br>*BAR0 at 0xa1000000 mapped at 0x00000000512efc91, length=2097152(/2097152)*<br>*BAR1 at 0xa1200000 mapped at 0x000000000ee6abb6, length=65536(/65536)*<br>*config bar 1, pos 1.*<br>*2 BARs: config 1, user 0, bypass -1.*<br>其中，最后一句表示有 2 个 Bar，config bar 为 Bar1, user bar 为 Bar0；我们需要找的是 user bar，即 Bar0。查看上面的地址可以看到 Bar0 的 length 为 2097152。 |

### 2.1.6 设置 EEP-TPU 数据的内存基址（2 个基址）

| | |
|---|---|
| 函数 | int eeptpu_set_base_address(unsigned long base0, unsigned long base1); |
| 功能 | 设置 EEP-TPU 的内存基址。与 boot 内核文件的配置有关，不同开发板可能需要不同的内存配置，需根据实际情况进行配置，请在获取 boot 文件时咨询可用的基址。 |
| 参数 | base0：基址 0<br>base1：基址 1<br>**算法内存空间说明：**<br>每个算法使用的 EEP-TPU 数据分为以下 4 类：<br>（1）参数数据和算法指令数据（参数数据：算法中各个层所使用的参数；算法指令数据：该算法的 TPU 指令数据。参数数据和算法指令数据在 TPU 推理过程中始终不变。）<br>（2）输入数据<br>（3）输出数据<br>（4）临时数据，即 TPU 在推理过程中计算生成的临时数据。 |

这 4 类数据可以分配到上述两个不同的基址中。调用此函数，会将输入输出数据放在一起，使用基址 0；将参数数据和算法指令数据、临时数据放在一起，使用基址 1。当基址 0 和基址 1 设置的地址相同时，则上述 4 类数据使用同一块内存空间。

**数据的排列顺序为：**

（1）当 base0 等于 base1 时：

输入数据，输出数据，参数数据，指令数据，临时数据。

（2）当 base0 不等于 base1 时：

Base0: 输入数据，输出数据；

Base1: 参数数据，指令数据，临时数据。

**当使用多个算法时：**

使用 eeptpu_load_bin API 函数加载算法时，参数 fg_multi 用于标记多个算法的加载。当加载第 1 个算法时，fg_multi 设为 0，加载后续算法时 fg_multi 设为 1。

在库文件内部会维护 base0 和 base1 这两个基址，当加载完一个 bin 数据后，库文件内部的 base0 和 base1 会更新成紧接着上一个算法的可用的内存地址。

**多个算法情况下的内存数据的前后排列顺序为：**

（1）当 base0 等于 base1 时：

算法 1[输入数据，输出数据，参数数据，指令数据，临时数据]；算法 2[输入数据，输出数据，参数数据，指令数据，临时数据]；……

（2）当 base0 不等于 base1 时：

Base0: 算法 1[输入数据，输出数据]；算法 2[输入数据，输出数据]；……

Base1: 算法 1[参数数据，指令数据，临时数据]； 算法 2[参数数据，指令数据，临时数据]；……

**算法使用示例：**

算法 1：

```
// eeptpu_set_base_address 只需在开始时配置一次。
tpu1->eeptpu_set_base_address(0x30000000, 0x30000000);
tpu1->eeptpu_load_bin(path_bin1, 0);
```

算法 2：紧接着算法 1 来分配内存地址，不需再重复配置基址。

```
tpu2->eeptpu_load_bin(path_bin2, 1);
```

eeptpu_set_base_address 函数只需第一次加载算法前调用 1 次。第一个算法加载时 eeptpu_load_bin 函数的 fg_multi 参数为 0，后续算法加载时 fg_multi 设置为 1。

**多个算法情况下的特殊使用内存示例：**

| | |
|---|---|
| | 上述第 2 点是由库文件内部来自动维护多算法情况下的各算法的内存基址的分配。而多算法情况下，用户也可以手动维护每个算法的内存基址，只要给各算法分配的内存数据空间不要覆盖其他算法的数据空间即可。例如：<br>算法 1：使用内存空间的大小为 0xA00000，从 0x30000000 开始分配，空间使用到 0x30A00000 为止。<br>*tpu1->eeptpu_set_base_address(0x30000000, 0x30000000);*<br>*tpu1->eeptpu_load_bin(path_bin1, 0);*<br>算法 2：从 0x31000000 开始分配，未覆盖算法 1 的空间。<br>*tpu2->eeptpu_set_base_address(0x31000000, 0x31000000);*<br>*tpu2->eeptpu_load_bin(path_bin2, 0);*<br>eeptpu_set_base_address 函数每次都需要调用，eeptpu_load_bin 函数的 fg_multi 参数为 0，表示使用前面 eeptpu_set_base_address 函数设置的基址。<br>**获取算法使用的内存空间大小：（内存空间大小的获取）**<br>可使用 eeptpu_get_memory_used_size( )API 函数来获取该算法所占用的内存空间。若有多个算法，则将每个算法所获取的空间大小加起来就是总共需要使用的内存空间大小。总共需要使用的内存空间大小是每个算法所获取的空间大小之和。<br>每个算法使用的数据将限制在为其分配的内存地址空间范围内，不会额外再动态使用别的内存地址空间。<br>**示例：**<br>*unsigned long memsize1 = tpu1->eeptpu_get_memory_used_size();*<br>*int MBytes1 = memsize1 / (1024*1024);*<br>*int KBytes1 = memsize1 % (1024*1024);*<br>*unsigned long memsize2 = tpu2->eeptpu_get_memory_used_size();*<br>*unsigned long memtotal = memsize1 + memsize2;* |
| 返回 | 0: 成功<br>< 0: 失败。 |
| 示例 | *tpu->eeptpu_set_base_address(0x30000000, 0x30000000);* |

## 2.1.7 设置 EEP-TPU 数据的内存基址（4 个基址）

| | |
|---|---|
| 函数 | int eeptpu_set_base_address(unsigned long base_par, unsigned long base_in, unsigned long base_out, unsigned long base_tmp); |
| 功能 | 设置 EEP-TPU 的内存基址，可设置 4 个不同类型的内存基址。与 boot 内核文件的配置有 |

| | |
|---|---|
| | 关，不同开发板可能需要不同的内存配置，需根据实际情况进行配置，请在获取 boot 文件时咨询可用的基址。 |
| 参数 | base_par: 算法参数数据（包含算法指令数据）<br>base_in: 输入数据<br>base_out: 输出数据<br>base_tmp: 临时数据<br>**算法内存空间说明：**<br>每个算法使用的 EEP-TPU 数据分为以下 4 类：<br>（1）参数数据及算法指令数据 （参数数据：算法中各个层所使用的参数；算法指令数据：该算法的 TPU 指令数据。参数数据和算法指令数据在 TPU 推理过程中始终不变。）<br>（2）输入数据<br>（3）输出数据<br>（4）临时数据，即 TPU 在推理过程中计算生成的临时数据。<br>这 4 类数据可以分别配置到上述基址中。4 个基址中，可以有相同的基址。若存在相同的基址，则表示对应的数据是存放在该基址起始的内存空间内，按照这个顺序安排数据存放：参数数据、输入数据、输出数据、临时数据。<br>使用多个算法的情况，与“2.1.6 节”里的说明一致。 |
| 返回 | 0: 成功<br>< 0: 失败。 |
| 示例 | *tpu->eeptpu_set_base_address(0x60000000, 0x60000000, 0x60000000, 0x60000000);* |

### 2.1.8 加载 EEP-TPU BIN 文件

| | |
|---|---|
| 函数 | int eeptpu_load_bin(const char* path_bin, int fg_multi=0); |
| 功能 | 加载 eeptpu 编译器生成的 bin 文件。<br>**注意**：一个 EEPTPU 对象只能加载一个 EEPTPU BIN 文件。如果需要使用多个 EEPTPU BIN 文件，则需定义多个 EEPTPU 对象。 |
| 参数 | **path_bin**：bin 文件的路径<br>**fg_multi**: 是否已加载多个 bin 文件。程序中第一次调用本函数时，fg_multi 必须设置为 0；若一个程序中需要加载多个 bin 文件，则第 2 个开始及之后的 fg_multi 必须设置为 1。<br>多核情况下，也一样，第一次调用本函数时 fg_multi 设置为 0，之后调用时设置为 1。 |

| | |
|---|---|
| 返回 | 0: 成功<br>< 0: 失败。 |
| 示例 | *tpu->eeptpu_load_bin(path_bin);*<br>若需加载多个 bin 文件：<br>*tpu1->eeptpu_load_bin(path_bin); // 第 2 个参数不填的话，默认=0*<br>*tpu2->eeptpu_load_bin(path_bin, 1);*<br>*tpu3->eeptpu_load_bin(path_bin, 1);* |

### 2.1.9 配置算法跳转

| | |
|---|---|
| 函数 | int eeptpu_jump_update(EEPTPU* alg2); |
| 功能 | EEP-TPU 支持从算法 1 跳转到算法 2 去执行。算法 1 的输出作为算法 2 的输入。应用算法跳转时，算法 1 和算法 2 都需各自初始化，然后调用此函数进行跳转设置。在推理时，只需调用算法 1 的推理即可。TPU 内部在算法 1 推理结束后会自动跳转到算法 2 继续执行，推理返回的结果数据也是算法 2 的推理结果。 |
| 参数 | **alg2：**第 2 个算法 |
| 返回 | 0: 成功<br>< 0: 失败。 |
| 示例 | *EEPTPU *tpu1 = NULL;*<br>*EEPTPU *tpu2 = NULL;*<br>*>> 初始化 tpu1 和 tpu2 // 从 tpu1 跳转到 tpu2 去执行*<br>*ret = tpu1->eeptpu_jump_update(tpu2);*<br>*>> tpu1 写输入数据*<br>*tpu1->eeptpu_forward(result);*<br>*>> 只需 tpu1 推理，在 tpu1 推理完成后会自动跳转到 tpu2 去执行，后续读出的推理结果是 tpu2 网络的推理结果。* |

### 2.1.10 获取该算法的内存使用大小

| | |
|---|---|
| 函数 | unsigned long eeptpu_get_memory_used_size(); |
| 功能 | 获取该算法所使用的内存空间大小。 |
| 参数 | 无 |
| 返回 | 占用内存的字节数。 |

| | |
|---|---|
| 示例 | *unsigned long memsize = tpu->eeptpu_get_memory_used_size();*<br>*int MBytes = memsize / (1024*1024);*<br>*int KBytes = memsize % (1024*1024);* |

### 2.1.11 获取库文件的版本

| | |
|---|---|
| 函数 | char* eeptpu_get_lib_version(); |
| 功能 | 获取库文件版本信息。 |
| 参数 | |
| 返回 | 版本字符串 |
| 示例 | *char* ptr = tpu->eeptpu_get_lib_version();*<br>*printf("EEPTPU library version: %s\n", ptr);* |

### 2.1.12 获取 EEP-TPU 硬件版本

| | |
|---|---|
| 函数 | char* eeptpu_get_tpu_version(); |
| 功能 | 获取 EEP-TPU 硬件版本信息。 |
| 参数 | |
| 返回 | 版本字符串 |
| 示例 | *char* ptr = tpu->eeptpu_get_tpu_version();*<br>*printf("EEPTPU hardware version: %s\n", ptr);* |

### 2.1.13 获取 EEP-TPU 硬件配置信息

| | |
|---|---|
| 函数 | char* eeptpu_get_tpu_info(); |
| 功能 | 获取 EEP-TPU 硬件配置信息。 |
| 参数 | |
| 返回 | 字符串 |
| 示例 | *char* ptr = tpu->eeptpu_get_tpu_info();*<br>*printf("EEPTPU hardware info : %s\n", ptr);* |

### 2.1.14 获取 EEP-TPU BIN 文件扩展信息

| | |
|---|---|
| 函数 | char* eeptpu_get_extinfo(); |

| | |
|---|---|
| 功能 | 获取 EEP-TPU bin 文件自定义的扩展信息字符串。该字符串在编译器编译时通过命令行参数“*--extinfo*”传入，可由用户自定义格式并自行解析。 |
| 参数 | |
| 返回 | 字符串 |
| 示例 | *char* ptr = tpu->eeptpu_get_extinfo();*<br>*printf("extinfo: %s\n", ptr);* |

## 2.2 输入数据

### 2.2.1 等待输入缓存区可写

| | |
|---|---|
| 函数 | int eeptpu_wait_input_writable(unsigned int timeout_ms = 2000); |
| 功能 | 等待输入缓存变为可写入状态。当输入缓冲区的数据还未被使用时，不能再继续写入数据，否则会导致输入数据错乱。使用本函数可以等待输入缓存转变为可写入状态。 |
| 参数 | **timeout_ms**：等待的超时时间，单位：毫秒。（默认值 2000ms） |
| 返回 | 0: 成功。可以继续写入数据。<br>< 0: 失败。（错误码参考附件 1）。如果当前数据缓冲区的数据还未使用完，建议设置更长的超时时间。如果强行写入，可能造成输入数据错乱，影响推理结果。 |
| 示例 | *ret = tpu->eeptpu_wait_input_writable(2000);* |

### 2.2.2 设置输入数据（单输入情况）

| | |
|---|---|
| 函数 | int eeptpu_set_input(void* input_data, int dim1, int dim2, int dim3, int mode = 0, int data_type = DType_FP32); |
| 功能 | 在神经网络为单个输入的情况下，设置该网络的输入数据。 |
| 参数 | **input_data**：预处理后的输入数据，其维度需与神经网络输入大小一致。<br>预处理：例如 resize、减均值、归一化等操作。若输入为图像数据，均值和归一化操作可由本库文件自动处理，即用户程序在预处理中可以不用做均值和归一化。<br>若输入为图像数据：这里可以是 Opencv 的 Mat 格式数据指针。<br>**dim1、dim2、dim3**：输入数据的维度。 |

| | |
|---|---|
| | 若输入为图像数据：dim1、dim2、dim3 分别指图像的通道数、高度、宽度。<br>**Mode**：输入数据的模式。<br>Mode=0：Mode 默认值为 0。此时可以通过库文件进行均值/归一化处理。适用于输入图像数据，在使用 eeptpu 编译器编译神经网络时设置好均值，调用此函数时可以自动进行均值计算。<br>Mode=1：在库文件内部不进行均值/归一化计算。所以，在调用本函数前，需要由用户对输入数据做好全部预处理操作。<br>上述 mode 等于 0 和 1 的情况，输入数据写入内存前会由库文件将该数据转换成 TPU 内部可识别的默认格式。<br>Mode=2：将这里的原始输入数据不做任何转换，直接写入内存给 TPU 使用。适用于应用“pack”模式的输入。<br>**data_type**：输入数据的数据类型，适用于 mode=1 的情况。默认是使用 float32 类型。支持的数据类型有：FP32/FP16/INT8/INT16/INT32/UINT8。<br>神经网络输入数据的维度可在加载完 bin 文件后，通过 tpu->input_shape[n]来获取。N 取值范围：0~3；分别表示 batch_size、C、H、W（图像维度）。<br>**注意：batch_size 始终为 1。** |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |
| 示例 | *cv::Mat cvimg_resized;*<br>*>> read&process image to cvimg_resized*<br>*tpu->eeptpu_set_input(cvimg_resized.data,tpu->input_shape[1],tpu->input_shape[2],tpu->input_shape[3]);* |

### 2.2.3 设置输入数据（多输入情况）

| | |
|---|---|
| 函数 | int eeptpu_set_input(int input_id, void* input_data, int dim1, int dim2, int dim3, int mode = 0, int data_type = DType_FP32); |
| 功能 | 在神经网络为多个输入的情况下，设置该网络的输入数据。 |
| 参数 | **input_id**：输入数据的索引号。该索引号可在编译器编译 bin 文件时打印出来。<br>**其他参数**：与单输入情况的说明一致。 |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |

| | |
|---|---|
| 示例 | *cv::Mat cvimg_resized_0;*<br>*>> read&process image to cvimg_resized_0*<br>*tpu->eeptpu_set_input(0, cvimg_resized.data,c0, h0, w0);*<br>*cv::Mat cvimg_resized_1;*<br>*>> read&process image to cvimg_resized_1*<br>*tpu->eeptpu_set_input(1, cvimg_resized.data,c1, h1, w1);* |

### 2.2.4 获取输入数据的信息（多输入情况）

| | |
|---|---|
| 函数 | int eeptpu_get_input_info(std::vector<struct NET_INPUT_INFO>& input_info); |
| 功能 | 获取该网络的输入数据的信息。适用于单输入和多输入的情况。 |
| 参数 | **input_info**：输入数据的信息。NET_INPUT_INFO 结构体数据。包含的信息有：<br>input_id：输入数据索引号。（单输入的话，索引号为 0）<br>c/h/w：数据的 3 个维度。<br>name：输入数据的名称。与编译器编译过程中打印的输入数据名称对应。也可在 Netron 网络可视化程序查看到对应的输入数据名称。<br>mean/norm：该输入数据对应的均值和归一化配置。<br>pack_type: “Pack”模式时，pack 的输入数据的类型。<br>pack_out_c/pack_out_h/pack_out_w：“Pack”模式时，pack 完成后的维度。 |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |
| 示例 | *std::vector<struct NET_INPUT_INFO> inputs_info;*<br>*tpu->eeptpu_get_input_info(inputs_info);* |

### 2.2.5 获取输入数据的均值和归一化配置（单输入情况）

| | |
|---|---|
| 函数 | void eeptpu_get_mean_norm(std::vector<float>& get_mean, std::vector<float>& get_norm); |
| 功能 | 获取输入数据的均值和归一化配置。 |
| 参数 | get_mean：均值数据。对应编译器的参数“--mean”的配置。<br>get_norm：归一化数据。对应编译器的参数“--norm”的配置。 |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |
| 示例 | *vector<float> net_mean;* |

| | |
|---|---|
| | *vector<float> net_norm;*<br>*tpu->eeptpu_get_mean_norm(net_mean, net_norm);* |

# 2.3 推理

## 2.3.1 设置等待推理结束的等待超时时间

| | |
|---|---|
| 函数 | int eeptpu_set_forward_timeout(unsigned int ms); |
| 功能 | 执行推理函数时，有一个等待推理完成的等待超时时间；默认值是 20000 毫秒。用户可自行配置一个合适的等待时长。 |
| 参数 | **ms**: 超时时长（单位：毫秒） |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |
| 示例 | *int ret = tpu-> eeptpu_set_forward_timeout(3000);* |

## 2.3.2 前向推理

| | |
|---|---|
| 函数 | int eeptpu_forward(std::vector<struct EEPTPU_RESULT>& result); |
| 功能 | 进行图像推理。 |
| 参数 | **result**: 推理结果<br>*struct EEPTPU_RESULT*<br>*{*<br>*float* data;  // 最终数据*<br>*int shape[4];  // 输出数据的维度（shape[0]=1）*<br>*};* |
| 返回 | 0: 成功<br>< 0: 失败。（错误码参考附件 1） |
| 示例 | *std::vector<struct EEPTPU_RESULT> results;*<br>*int ret = tpu->eeptpu_forward(results);* |

## 2.3.3 获取 EEP-TPU 硬件部分推理时间

| | |
|---|---|
| 函数 | unsigned int eeptpu_get_tpu_forward_time(); |

| | |
|---|---|
| 功能 | 获取 EEP-TPU 硬件部分推理时间。（单位：微秒） |
| 参数 | |
| 返回 | 硬件部分推理时间（单位：微秒） |
| 示例 | *unsigned int hwus = tpu->eeptpu_get_tpu_forward_time();*<br>*printf("EEPTPU hw cost: %.3f ms\n", (float)hwus/1000);* |

## 2.4 关闭

### 2.4.1 中止 EEP-TPU 运行

| | |
|---|---|
| 函数 | void eeptpu_terminate(bool b_term); |
| 功能 | EEP-TPU‘中止运行’或‘取消中止运行状态’。 |
| 参数 | **b_term**: true-中止运行；false-取消中止运行状态。 |
| 返回 | 无 |
| 示例 | *tpu->eeptpu_terminate(true);*<br>示例场景：<br>在应用程序中捕捉到 Ctrl+C 等中断信号后，可调用本函数来及时中止并退出 EEP-TPU 的推理函数。或者，在多线程环境中，线程 1 在循环中不断进行 TPU 推理；线程 2 可以调用本函数传入 true 来中止该 TPU 的推理，然后在下一次需要推理的时候再传入 false 来结束这一中止状态。 |

### 2.4.2 关闭 EEP-TPU

| | |
|---|---|
| 函数 | void eeptpu_close(); |
| 功能 | 关闭 EEP-TPU |
| 参数 | 无 |
| 返回 | 无 |
| 示例 | *tpu->eeptpu_close();*<br>在应用程序关闭时，需调用此函数来关闭 EEP-TPU。 |

## 2.5 内存映射

### 2.5.1 内存映射

| | |
|---|---|
| 函数 | void* eeptpu_memory_map(unsigned int addr, unsigned int len); |
| 功能 | 对某段内存进行映射，获取其指针。 |
| 参数 | **addr**：需要映射的内存的起始地址；<br>**len**：需要映射的内存大小。 |
| 返回 | 返回映射的内存指针。若是 NULL，表示映射失败。 |
| 示例 | *unsigned char* ptr = tpu-> eeptpu_memory_map(0x65000000, 0x1000);* |

### 2.5.2 解除内存映射

| | |
|---|---|
| 函数 | int eeptpu_memory_unmap(void* start, unsigned int len); |
| 功能 | 解除对某段内存的内存映射。 |
| 参数 | **start**：eeptpu_memory_map 函数返回的内存指针；<br>**len**：已映射的内存大小。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *unsigned char* ptr = tpu->eeptpu_memory_map(0x65000000, 0x1000);*<br>*int ret = tpu->eeptpu_memory_unmap(ptr, 0x1000);* |

### 2.5.3 读内存

| | |
|---|---|
| 函数 | int eeptpu_mem_rd(unsigned char* buf, unsigned long addr, unsigned int len); |
| 功能 | 读取某段内存的数据。 |
| 参数 | **buf**：数据指针，读取的内存数据存放于此。用户需在调用此函数前预先为其开辟好足够大小的内存空间。<br>**addr**：内存地址。<br>**len**：读取的内存字节长度。 |
| 返回 | 0: 成功 |

| | |
|---|---|
| | < 0: 失败。（错误码参考附件 1） |
| 示例 | *unsigned char* buf = (unsigned char*)malloc(0x1000);*<br>*ret = tpu->eeptpu_mem_rd(buf, 0x60000000, 0x1000);* |

### 2.5.4 写内存

| | |
|---|---|
| 函数 | int eeptpu_mem_wr(unsigned char* buf, unsigned long addr, unsigned int len); |
| 功能 | 内存写数据。 |
| 参数 | **buf**：数据指针，欲写入内存的数据存放于此。<br>**addr**：内存地址。<br>**len**：写入数据的字节长度。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *unsigned char* buf = (unsigned char*)malloc(0x1000);*<br>*>> 填充 buf 数据*<br>*ret = tpu->eeptpu_mem_wr(buf, 0x60000000, 0x1000);* |

## 2.6 参数复用功能

对于多核 EEP-TPU，当某个算法（bin 文件）想给多个核同时使用时，可使用本节介绍的参数复用功能。

在常用的单核或多核推理时，每个核可能负责不同的算法。在初始化的过程中，每个核都要进行各自的初始化（例如配置各个核的内存基址、寄存器信息、load bin 文件等），特别是 load bin 文件，会把 bin 里面的算法数据写入内存中，占用一定的内存空间。

当多个核需要用到同一个算法时，为了节省内存空间，避免多次 load 同一个 bin 文件造成内存空间的浪费，可使用本节的参数复用功能，这样只需要把这个 bin 文件里的参数数据加载一次即可，参数数据、输入数据、输出数据、临时数据可自定义灵活分配内存地址空间，使算法可以更灵活地进行配置。

### 2.6.1 加载算法配置信息

| | |
|---|---|
| 函数 | int eeptpu_nn_load_info(const char* path_bin); |
| 功能 | 加载 bin 文件里的神经网络算法配置信息，不会将算法的参数和指令信息写入内存。 |
| 参数 | **path_bin**: bin 文件路径。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *#define ALG_NUM 2 // count of used bin files*<br>*EEPTPU* alg[ALG_NUM];*<br>*for (unsigned int i = 0; i < ALG_NUM; i++)*<br>*{*<br>*printf("\nReading alg[%d] info...\n", i);*<br>*alg[i] = alg[i]->init();*<br>*ret = alg[i]->eeptpu_set_interface(eepInterfaceType_SOC);*<br>*if (ret < 0) return ret;*<br>*ret = alg[i]->eeptpu_nn_load_info(bins[i].c_str());*<br>*if (ret < 0) {*<br>*printf("Load info fail, ret=%d\n", ret);*<br>*return ret;*<br>*}*<br>*}* |

### 2.6.2 加载算法参数数据

| | |
|---|---|
| 函数 | int eeptpu_nn_load_data(unsigned long addr, unsigned int len, int flag); |
| 功能 | 加载 bin 文件里的神经网络算法参数或算法指令，将其写入内存。 |
| 参数 | **Addr**：内存地址。<br>**len**：数据长度。<br>**flag**：标志位，1 表示加载算法参数数据，2 表示加载算法指令数据。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *ret = alg[0]->eeptpu_nn_load_data(0x60000000, 0x18000, 1);* |

### 2.6.3 设置 EEP-TPU 数据的内存基址

| | |
|---|---|
| 函数 | int eeptpu_set_base_address(struct NET_MEM_ADDR mem_base); |
| 功能 | 设置数据存放的内存起始地址。 |
| 参数 | **mem_base**：NET_MEM_ADDR 结构体数据。可配置 5 类数据的内存存放基址：参数、算法指令、输入、输出、临时数据。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *EEPTPU* cores[2];*<br>*vector<struct NET_MEM_ADDR> cores_membase;*<br>*>> 初始化 cores 和 cores_membase*<br>*ret = cores[0]->eeptpu_set_base_address(cores_membase[0]);* |

### 2.6.4 拷贝神经网络相关数据

| | |
|---|---|
| 函数 | int eeptpu_net_copy(EEPTPU* alg); |
| 功能 | 拷贝另一个已经初始化并加载了数据的 EEPTPU 对象的相关信息。 |
| 参数 | **alg**：已经初始化并加载了数据的 EEPTPU 对象指针。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *ret = cores[0]->eeptpu_net_copy(alg[1]);* |

### 2.6.5 获取各数据的内存空间大小

| | |
|---|---|
| 函数 | struct NET_MEM_SIZE eeptpu_get_memory_zone_size(); |
| 功能 | 获取当前加载的算法的各类数据的内存空间大小。 |
| 参数 | |
| 返回 | NET_MEM_SIZE 结构体数据，包含 5 类数据的内存大小：参数、算法指令、输入、输出、临时数据。 |
| 示例 | *vector<struct NET_MEM_SIZE> memsize;*<br>*for (unsigned int i = 0; i < ALG_NUM; i++)*<br>*{* |

| | |
|---|---|
| | *memsize[i] = alg[i]->eeptpu_get_memory_zone_size();*<br>*printf("alg[%d] memsize: par 0x%08x; alg 0x%08x; in 0x%08x; out 0x%08x; tmp 0x%08x\n", i, memsize[i].par, memsize[i].alg, memsize[i].in, memsize[i].out, memsize[i].tmp);*<br>*}* |

### 2.6.6 获取各类数据的内存实际地址

| | |
|---|---|
| 函数 | int eeptpu_get_memory_zone_addr(int flag); |
| 功能 | 获取各类数据的内存实际地址。 |
| 参数 | **flag**：标志位，eeptpu.h 头文件定义的 idxPar、idxIn、idxTmp、idxOut、idxAlg。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *unsigned long addr_par = tpu->eeptpu_get_memory_zone_addr(idxPar);* |

### 2.6.7 获取各类数据的内存占用大小

| | |
|---|---|
| 函数 | unsigned int eeptpu_get_memory_zone_size(int flag); |
| 功能 | 获取各类数据的内存占用字节数。 |
| 参数 | **flag**：标志位，eeptpu.h 头文件定义的 idxPar、idxIn、idxTmp、idxOut、idxAlg。 |
| 返回 | 0: 成功；<br>< 0: 失败。 |
| 示例 | *unsigned int len_par = tpu->eeptpu_get_memory_zone_size(idxPar);* |

# 3、同一程序中多个神经网络算法交替使用

EEP-TPU API 支持在同一个程序中交替使用多个神经网络（eeptpu bin）。

初始化示例代码：

```
EEPTPU *tpu1 = NULL;
EEPTPU *tpu2 = NULL;
EEPTPU *tpu3 = NULL;
char path_bin1[ ] = (char*)"./eeptpu1.pub.bin";
char path_bin2[ ] = (char*)"./eeptpu2.pub.bin";
char path_bin3[ ] = (char*)"./eeptpu3.pub.bin";
int eeptpu_init(int interface_type)
{
    int ret;
    if (tpu1 == NULL) tpu1 = tpu1->init();
    if (tpu2 == NULL) tpu2 = tpu2->init();
    if (tpu3 == NULL) tpu3 = tpu3->init();
    // Configure the 1st EEPTPU object
    ret = tpu1->eeptpu_set_interface(interface_type);
    if (ret < 0) return ret;
    ret = tpu1->eeptpu_load_bin(path_bin1);
    if (ret < 0) return ret;
    // Configure the 2nd EEPTPU object
    ret = tpu2->eeptpu_set_interface(interface_type);
    if (ret < 0) return ret;
    ret = tpu2->eeptpu_load_bin(path_bin2, 1);
    if (ret < 0) return ret;
    // Configure the 3rd EEPTPU object
    ret = tpu3->eeptpu_set_interface(interface_type);
    if (ret < 0) return ret;
```

```
    ret = tpu3->eeptpu_load_bin(path_bin3, 1);
    if (ret < 0) return ret;
    return 0;
}
```

# 附件 1 - 错误码列表

| 错误码 | 错误描述 |
|---|---|
| 0 | 成功 |
| -1 | 失败 |
| -2 | 参数错误 |
| -3 | 不支持的操作 |
| -4 | 文件打开失败 |
| -5 | 文件读取失败 |
| -6 | 文件写入错误 |
| -7 | 内存分配失败 |
| -8 | 超时 |
| -11 | 加载 Bin 文件失败 |
| -12 | EEP-TPU 初始化失败 |
| -13 | 错误的图像类型 |
| -14 | 不支持的像素类型 |
| -15 | 均值设置失败 |
| -16 | 错误的 Blob 输入 |
| -17 | 错误的 Blob 格式 |
| -18 | 错误的 Blob 大小 |
| -19 | 错误的 Blob 数据 |
| -20 | 错误的 Blob 输出 |
| -21 | 内存地址错误 |
| -22 | 内存数据写入失败 |
| -23 | 内存数据读取失败 |

| | |
|---|---|
| -24 | 内存映射失败 |
| -25 | 设备打开失败 |
| -26 | 设备未初始化 |
| -27 | EEP-TPU 接口类型设置失败 |
| -28 | EEP-TPU 接口初始化失败 |
| -29 | EEP-TPU 接口操作失败 |
| -40 | EEP-TPU Bin 文件版本过旧，请用新版本编译器重新生成 Bin 文件 |
| -41 | EEP-TPU 库文件版本过旧，请使用新版本的库文件 |
| -42 | EEP-TPU 硬件版本过旧，请升级 EEP-TPU |