### WS63V100 NV 存储

## 用户指南

文档版本 03

发布日期 2024-05-30

前言

## 概述

本文档主要针对 WS63V100 中 NV 存储模块的使用进行介绍。用于指导工程人员能够快速使用 NV 模块进行二次开发。

## 产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
WS63	V100

## 读者对象

本文档主要适用于以下工程师:

- 技术支持工程师
- 软件工程师

## 符号约定

在本文中可能出现下列标志,它们所代表的含义如下。

符号	说明

2024-05-30 i

用户指南 前言

符号	说明
▲ 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
♪ 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
<u> 注意</u>	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备 损坏、数据丢失、设备性能降低或其它不可预知的结果。 "须知"不涉及人身伤害。
□ 说明	对正文中重点信息的补充说明。 "说明"不是安全警示信息,不涉及人身、设备及环境伤害信息。

## 修改记录

文档版本	发布日期	修改说明
03	2024-05-30	• 更新 "3.1 功能描述" 小节内容。
		• 更新 "3.2 接口说明"小节内容。
02	2024-05-07	更新"3.2 接口说明"小节内容。
01	2024-04-10	第一次正式版本发布。
		• 更新 "3.1 功能描述"小节内容。
		• 更新 "3.2 接口说明"小节内容。
		• 更新 "3.3 开发指引"小节内容。
00B02	2024-03-29	新增"4 NV 项汇总"章节。
00B01	2024-01-10	第一次临时版本发布。

2024-05-30 ii

## 目 录

前言	i
1 NV 简介	1
2 NV 编译预置	2
2.1 新增 NV 项	2
2.1.1 新增 NV 项流程	2
2.1.2 新增 NV 项示例	
2.2 编译生成 NV 镜像	6
3 NV API 指南	7
3.1 功能描述	7
3.2 接口说明	8
3.3 开发指引	
3.4 注意事项	13
4 NV 项汇总	14

1 NV 简介

NV 模块用于本地存储器中存储非易失性数据。NV 中的每项数据以类似 key-value 的方式进行定义,数据项中包含唯一的索引 key 和自定义数据类型的 value。

NV 项可通过两种方式进行存储:编译预置和 API 写入。

- 编译预置是指开发者可在代码编译阶段,通过修改 NV 头文件和 NV 配置文件的方式生成客制化的 NV 镜像,在镜像烧录的过程中统一烧录到存储介质中。预置的 NV 在代码运行阶段可通过 API 接口进行读取和更新。
- API 写入是指用户可直接在代码中调用 API 接口写入新的 NV 项,具体使用方法请参见"3 NV API 指南"。

## **2** NV 编译预置

编译预置的方式不支持加密 NV 项的生成。如需写入加密 NV 项,必须使用 API 接口。

- 2.1 新增 NV 项
- 2.2 编译生成 NV 镜像

## 2.1 新增 NV 项

## 2.1.1 新增 NV 项流程

步骤 1 在头文件中新增 kvalue 的数据类型定义(非必须,如果是通用类型数据可忽略此步骤)。

步骤 2 在 json 文件中新增 NV 描述项。

----结束

#### 新增 kvalue 数据类型

- 通用数据类型:unit8\_t、unit16\_t、unit32\_t、bool。
- 自定义数据类型: 支持自定义枚举 (enum) 类型和结构体 (struct) 类型。
- 自定义数据类型存放路径:
   middleware/chips/ws63/nv/nv\_config/include/nv\_common\_cfg.h
- 当用户使用通用或已定义的数据类型时,不涉及上述文件的修改;当用户要新增 枚举或结构体类型时,需在上述文件中定义。

### 新增 NV 描述项

- NV 描述项文件路径: middleware/chips/ws63/nv/nv\_config/cfg/acore/app.json
- 定义说明:

#### 表2-1 NV 配置选项说明

NV 配置选项	说明
key_id	NV 项的 ID。
key_status	NV 项的状态。
structure_type	NV 项的数据结构类型。
attributions	NV 项的属性值。
value	NV 项的数据。

#### 图2-1 NV 配置文件示例

```
"common":{
    "module_id": "0x0",

    "key_id": "0x0",
        "key_status": "reserve",
        "structure_type": "uint8_t",
        "attributions": 1,
        "value": [0]
},

"sample":{
        "key_id": "0x1",
        "key_status": "alive",
        "structure_type": "sample_type_t",
        "attributions": 1,
        "value": [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]
}
```

在 NV 的配置中,每个字段的详细描述如下:

- key\_id:

以十六进制形式给出的 NV 项 ID。key\_id 必须唯一,不能重复,因此建议用户在"key\_id.h"中预留的用户区间内取值,避免不同模块使用 NV 互相影响。

key\_status:

用于标记是否将该项的 NV 值编到生成的 bin 文件中。该字段为 "alive",表示 NV 项生效,当前固件版本正在使用此 key; 若为其他的字段或空,则不生效。

- structure\_type:

NV 项的数据类型。已在"新增 kvalue 数据类型"中详细描述。

- attributions: NV 项属性值。1 、2、4 为互斥关系,三选一。
  - 1: Normal nv (普通 NV,可修改)。
  - 2: Permanent nv (不可修改)。
  - 4: Un-upgrade nv (不随版本升级而修改)。
- value:

如果 value 不是上述通用数据类型,任何结构都必须以列表的形式书写,有如下两种情况:

- 列表所有成员全部赋值。
- 只对列表前面若干个成员赋值。这表明对末尾未赋值的成员缺省赋值为0。

## 2.1.2 新增 NV 项示例

- 在 "middleware/chips/ws63/nv/nv\_config/include/nv\_common\_cfg.h" 文件中新 增自定义结构体。新增自定义数据类型示例如下:
  - 新增结构体类型且结构体内都为基础类型:

```
typedef struct {
    int8_t param1;
    int8_t param2;
    int8_t param3;
    int8_t param4;
    int8_t param5;
    uint32_t param6;
    uint32_t param7;
    int32_t param8;
    uint32_t param9;
    uint32_t param10;
    uint32_t param11;
```

```
uint32_t param12;
uint32_t param13;
uint32_t param14;
uint32_t param15;
uint32_t param16;
uint32_t param17;
} sample_type_t;
```

- 新增结构体类型且结构体内有数组类型:

```
typedef struct {
    uint16_t param1;
    uint16_t param2;
    uint16_t param3;
    uint16_t param4[2];
} sample_two;
```

- 新增枚举类型:

```
typedef enum {
    PARAM1,
    PARAM2,
    PARAM3,
    PARAM4
} sample_three;
```

- 在 "middleware/chips/ws63/nv/nv\_config/cfg/acore/app.json" 文件中添加新的NV 项。
  - 当 kvalue 预置值类型是基础类型时,添加 kvalue 预置值如图 2-2 所示,可在 "app.json"配置文件直接添加,无需在头文件中新增。

#### 图2-2添加基础类型 kvalue 预置值

- 当 kvalue 预置值类型是结构体类型时,添加 kvalue 预置值如图 2-3 所示,该 kvalue 值要对应头文件中已有的结构体,如果没有需手动添加自定义结构体。

#### 图2-3 添加结构体类型 kvalue 预置值

## 2.2 编译生成 NV 镜像

在全量编译时自动生成 NV 镜像,如图 2-4 所示即为成功,即可在输出路径下生成 "ws63\_all\_nv.bin" 文件,可直接烧录使用。

输出路径: output\ws63\acore\nv\_bin\ws63\_all\_nv.bin

图2-4 build\_nvbin.py 脚本执行成功

build nv bin success!!

# 3 NV API 指南

- 3.1 功能描述
- 3.2 接口说明
- 3.3 开发指引
- 3.4 注意事项

## 3.1 功能描述

NV 当前支持存储最多 16K(存在管理结构体占用空间,实际略小于 16K)数据,备份分区大小与 NV 主区一致,总计占用 32K flash 空间,API 主要提供以下几种功能:

● NV 项写入:

保存需要存储的格式化数据。除普通属性的 NV 外,还可以设置 NV 项是否永久存储、是否加密存储和是否不可升级。

● NV 项读取:

从本地存储器读取 NV 数据。

- NV 信息查询:
  - 查询 NV 是否已存储于本地存储器中。
  - 查询 NV 空间的使用状态。
- NV 数据备份
  - 对 NV 数据进行备份,仅在退出产测模式时会自动进行备份,不接受手动备份。
- NV 数据恢复

- 对 NV 数据进行恢复,可实现全量和部分恢复。

NV 项的写入接口可设置 NV 项的属性,对于通过 API 动态添加的 NV 项,可在其写入的接口中传入其所拥有的特殊属性。

#### 须知

NV 数据写入 flash 不可避免的会增加 flash 的擦写次数,消耗 flash 寿命,甚至缩短产品使用年限。因此,一定要避免频繁写入 NV 数据。电池类产品设备,运行数据不会丢失,建议只在关机前写入要保存的 NV 数据,减少数据写入次数

## 3.2 接口说明

使用 NV 接口需要引用 NV 接口头文件,路径:include/middleware/utils/nv.h

NV 模块主要提供以下 API:

errcode\_t uapi\_nv\_write(uint16\_t key, const uint8\_t \*kvalue, uint16\_t kvalue\_length)

uapi_nv_write	写入 NV 数据项,默认属性 Normal,无回调函数,写入成功时返回 ERRCODE_SUCC,其他返回错误码。
key	要写入的 NV 项的 key ID,用于索引。
*kvalue	指向要写入的 NV 项的值的指针。
kvalue_length	写入数据的长度,单位: Byte。对于非加密 NV 项,支持最大值为 4060; 加密 NV 项为 4048。

errcode\_t uapi\_nv\_write\_with\_attr(uint16\_t key, const uint8\_t \*kvalue, uint16\_t kvalue\_length,nv\_key\_attr\_t \*attr, nv\_storage\_completed\_callback func)

uapi_nv_write_wi th_attr	写入 NV 数据项,并根据业务需求配置属性及回调函数,写入成功时返回 ERRCODE_SUCC,其他返回错误码。
key	要写入的 NV 项的 key ID,用于索引。
*kvalue	指向要写入的 NV 项的值的指针。
kvalue_length	写入数据的长度,单位: Byte。对于非加密 NV 项,支持最大值为 4060; 加密 NV 项为 4048。

uapi_nv_write_wi th_attr	写入 NV 数据项,并根据业务需求配置属性及回调函数,写入成功时返回 ERRCODE_SUCC,其他返回错误码。
*attr	要配置的 NV 项的属性。
func	kvalue 写入 Flash 后调用的回调函数。当前不支持。

errcode\_t uapi\_nv\_read(uint16\_t key, uint16\_t kvalue\_max\_length, uint16\_t \*kvalue\_length, uint8\_t \*kvalue)

uapi_nv_read	读取指定 NV 数据项的值,默认不获取 key 的属性值,读取成功时返回 ERRCODE_SUCC,其他返回错误码。
key	要读取的 NV 项的 key ID,用于索引。
kvalue_max_leng th	*kvalue 指向空间所能存储数据的最大值,单位:Byte。
*kvalue_length	实际读取到的数据长度。
*kvalue	指向保存读取数据的 buffer 的指针。

errcode\_t uapi\_nv\_read\_with\_attr(uint16\_t key, uint16\_t kvalue\_max\_length, uint16\_t \*kvalue\_length,uint8\_t \*kvalue, nv\_key\_attr\_t \*attr)

uapi_nv_read_wit h_attr	读取指定 NV 数据项的值,同时获取 key 的属性值,读取成功时返回 ERRCODE_SUCC,其他返回错误码。
key	要读取的 NV 项的 key ID,用于索引。
kvalue_max_leng th	*kvalue 指向空间所能存储数据的最大值,单位:Byte。
*kvalue_length	实际读取到的数据长度。
*kvalue	指向保存读取数据的 buffer 的指针。
*attr	获取到的 NV 项的属性。

errcode\_t uapi\_nv\_get\_store\_status(nv\_store\_status\_t \*status)

uapi_nv_get_stor e status	获取 NV 存储空间使用情况,获取成功时返回
e_status	ERRCODE_SUCC,其他返回错误码。

uapi_nv_get_stor e_status	获取 NV 存储空间使用情况,获取成功时返回 ERRCODE_SUCC,其他返回错误码。
*status	指向保存 NV 状态数据的指针。

errcode\_t uapi\_nv\_set\_restore\_mode\_all(void);

uapi_nv_set_restore_m ode_all 设置 NV 全量恢复出厂标记,备份成功时返回 ERRCODE_SUCC,其他返回错误码。
--

说明:该接口可实现备份区全部数据的恢复,工作区内没有进行备份数据将会被删除,即全量恢复后工作区和备份区的内容完全一致,调用后不会立即恢复,将在重新启动后进行恢复操作。

errcode\_t uapi\_nv\_set\_restore\_mode\_partitial(const nv\_restore\_mode\_t \*restore\_mode);

uapi_nv_set_restore_m	设置 NV 部分恢复出厂标记,设置成功时返回
ode_partitial	ERRCODE_SUCC,其他返回错误码。
*restore_mode	指向恢复标记结构体的指针。

通过控制恢复标记结构体内 region\_mode 数组中的标记为 0 或者 1,实现对指定的 region 区的数据恢复出厂。

说明:该接口可实现将备份区内指定 region 的数据恢复到工作区,标记为 0 的 region 区的数据将会保留工作区原数据,标记为 1 的 region 区内的数据恢复为备份 区的数据(注:该 region 区域内未备份的数据将会被删除),调用后不会立即恢复,将在重新启动后进行恢复操作。

#### 表3-1 region 区域划分表

恢复区域	key_id
region_0	[0x0001,0x1000)
region_1	[0x1000,0x2000)
region_2	[0x2000,0x3000)
region_3	[0x3000,0x4000)
region_4	[0x4000,0x5000)
region_5	[0x5000,0x6000)

恢复区域	key_id
region_6	[0x6000,0x7000)
region_7	[0x7000,0x8000)
region_8	[0x8000,0x9000)
region_9	[0x9000,0xA000)
region_10	[0xA000,0xB000)
region_11	[0xB000,0xC000)
region_12	[0xC000,0xD000)
region_13	[0xD000,0xE000)
region_14	[0xE000,0xF000)
region_15	[0xF000,0xFFFF)

#### □ 说明

- NV恢复出厂时,最小单位为一个 region 区,不支持进行单独 NV 项的恢复。
- "uapi\_nv\_set\_restore\_mode\_all" 和 "uapi\_nv\_set\_restore\_mode\_partitial" 两个函数仅是设置 NV 恢复出厂的标记,实际恢复出厂的操作需要在复位设备后执行。

## 3.3 开发指引

下述为 NV 读写接口的使用指引。API 调用处以加粗突出。

步骤 1 写入默认 Normal 类型 NV。

```
uint8_t *test_nv_value; /* 要写入的NV value保存在test_nv_value中 */
uint32_t test_len = 15; /* 长度为test_len , 示例中为15*/
uint16_t key = TEST_KEY; /* TEST_KEY 为该key的ID*/
errcode_t nv_ret_value = uapi_nv_write(key, test_nv_value, test_len);
if (nv_ret_value!= ERRCODE_SUCC) {
    return ERRCODE_FAIL;
}
return ERRCODE_SUCC;
```

#### 步骤 2 写入带属性 NV (配置永久属性, 其他略)。

```
uint8_t *test_nv_value; /* 要写入的NV value保存在test_nv_value中 */
```

```
uint32_t test_len = 15; /* 长度为test_len, 例中为15 */
uint16_t key = TEST_KEY;
nv_key_attr_t attr = {0};
attr.permanent = true;/* 永久属性设为true */
attr.encrypted = false;
attr.non_upgrade = false;
errcode_t nv_ret_value = uapi_nv_write_with_attr(key, test_nv_value, test_len, &attr, NULL);
if (nv_ret_value! = ERRCODE_SUCC) {
    return ERRCODE_FAIL;
}
/* APP PROCESS */
return ERRCODE_SUCC;
```

#### 步骤 3 读取 NV。

```
uint16_t key = TEST_KEY;
uint16_t key_len= test_len;
uint16_t real_len= 0;
uint8_t *read_value = malloc(key_len);
if (read_value == NULL) {
    return ERRCODE_MALLOC;
}
if (uapi_nv_read(key, key_len, &real_len, read_value) != ERRCODE_SUCC) {
    /* ERROR PROCESS */
    uapi_free(read_value);
    return ERRCODE_FAIL;
}
/* APP PROCESS */
ree(read_value);
return ERRCODE_SUCC;
```

#### 步骤 4 读取 NV 及属性。

```
uint16_t key = TEST_KEY;
uint16_t key_len = test_len;
uint16_t real_len = 0;
uint8_t *read_value = malloc(key_len);
nv_key_attr_t attr = {false, false, o};
ext_errno nv_ret = uapi_nv_read_with_attr(key, key_len, &real_len, read_value, &attr);
if (nv_ret!= ERRCODE_SUCC ) {

free(read_value);
    return ERRCODE_FAIL;
}
```

uapi\_free(read\_value);
return ERRCODE\_SUCC;

#### ----结束

#### □ 说明

开发指引只是 API 接口的测试用例,为用户提供简单的 sample 参考,sample 中省略了宏、部分变量、回调函数的定义过程和业务处理过程。

## 3.4 注意事项

- uapi\_nv\_write: 默认不对所存储的 key 添加额外属性(是否永久存储、是否加密存储等)。
- uapi\_nv\_write\_with\_attr:可同时配置 key 属性和注册回调函数。在 WS63V100中回调函数可以忽略,传 NULL 即可。
- NV 项存储在 Flash 中时,以 Flash 器件的 sector 为单位进行管理。NV 页的数量 默认配置为 4 页。对于 4060Byte 的 sector,除去管理结构,单个非加密 NV 项的 有效数据最大不应超过 4060Byte。
- NV 属性结构体和 NV 空间状态结构体说明详见"nv.h"文件。

# **4** NV 项汇总

NV_ID	NV 说明	NV value 说明
0x3	保留 NV 项	N/A
0x4	保留 NV 项	N/A
0x5	MAC 地址	MAC 地址的 6 字节。
0x6	频偏温补开关	<ul><li>0: 关闭</li><li>1: 开启</li></ul>
0x7	频偏温补补偿值	细调补偿值,取值范围[-127, 127], 共 8 个值, 对应补偿到温度区间[-40,-20), [-20,0), [0,20), [20,40),[40,60),[60,80),[80,100),[100,~)。
0x2003	国家码	• 67: 表示字符 'C' • 78: 表示字符 'N'
0x2004	数采开关状态	<ul><li>0: 关闭状态</li><li>1: 开启状态</li></ul>
0x2005	漫游开关	<ul><li>0: 关闭漫游</li><li>1: 开启漫游</li></ul>
0x2006	11r 开关	<ul><li>0: 开启 over air</li><li>1: 开启 over ds</li></ul>
0x2007	配置 Wi-Fi 11ax TXBF 能力位	• 0: 关闭 • 1: 开启

用户指南 4 NV 项汇总

NV_ID	NV 说明	NV value 说明
0x2008	配置 Wi-Fi LDPC 模 式开关	<ul><li>0: 关闭</li><li>1: 开启</li></ul>
0x2009	配置 Wi-Fi 接收 STBC 特性开关	<ul><li>0: 关闭</li><li>1: 开启</li></ul>
0x200A	配置是否使能以 HE ER SU 格式发包	<ul><li>0: 开启 SU 配置</li><li>1: 关闭 SU 配置</li></ul>
0x200B	配置 DCM 发送能力 开关	<ul><li>0: 不支持 DCM</li><li>1: BPSK</li><li>2: QPSK</li><li>3: 16-QAM</li></ul>
0x200C	配置是否支持 106- tone 发包	<ul><li>0: 不支持</li><li>1: 支持</li></ul>
0x200D	配置 Wi-Fi AMSDU 小包聚合 (<128Byte) 的最大 聚合个数。	支持配置范围: 1~4。
0x200E	配置发送 amsdu 开 关	<ul><li>0: 关闭</li><li>1: 开启</li></ul>
0x200F	配置 ampdu+amsdu 联合使能开关	<ul><li>0: 关闭</li><li>1: 开启</li><li>若关闭则仅能使用 ampdu 进行报文发送。</li></ul>
0x2010	配置 Wi-Fi 最大发送 A-MPDU 聚合报文数 量	支持配置范围: 2~32。
0x2011	配置 Wi-Fi 最大接收 A-MPDU 聚合报文数 量	支持配置范围: 1~32。
0x2012	配置聚合报文发送窗	支持配置范围: 2~64。

用户指南 4 NV 项汇总

NV_ID	NV 说明	NV value 说明
	口大小	
0x2013	最大 user 数量	支持范围 1~6
0x2015	WiFi 断连后不上报 lwip	<ul><li>0: 上报</li><li>1: 不上报</li></ul>
0x2016	linkloss 开关	0: 开启 linkloss     1: 关闭 linkloss
0x2050	射频插损	插损补偿值,精度 1dB。
0x2051	射频校准开关	每 bit 表示一个校准项。
0x2052	射频数据开关	每 bit 表示一个数据项。
0x2053	大区功率配置 FCC	根据国家码选择大区功率配置,参考软件开发指南的国家码功能配置。
0x2054	大区功率配置 ETSI	根据国家码选择大区功率配置,参考软件开发指南的国家码功能配置。
0x2055	大区功率配置 JAPAN	根据国家码选择大区功率配置,参考软件开发指南的国家码功能配置。
0x2056	大区功率配置通用	根据国家码选择大区功率配置,参考软件开发指南的国家码功能配置。
0x2057	RSSI 补偿值	RSSI 默认补偿值,单位:dB。
0x2058	校准参考功率	上电校准功率参考,单位: 0.1dB。
0x2059	高功率拟合曲线默认值	拟合曲线系数。
0x205A	低功率拟合曲线默认 值	拟合曲线系数。
0x205B	拟合曲线放大系数默 认值	拟合曲线系数。
0x205C	校准数据	数据。

用户指南 4 NV 项汇总

NV_ID	NV 说明	NV value 说明
0x205D	认证开关	<ul><li>0: 关闭</li><li>1: 开启</li></ul>
0x20A0	BSLE 最大功率档位	<ul> <li>若设置为 7,可以使用的档位是 0~7,每档对应-6、-2、2、6、10、14、16、20;</li> <li>若设置为 3,可以使用的档位是 0~5,每档对应-6、-2、2、6。</li> </ul>
0x2100~0x 211F	雷达性能参数	靠近、存在、距离跟踪参数等,按照架高、敏 感度、遮挡材料等情况的标定值。
0x2140	雷达控制参数	架高(0: 1 米, 1: 2 米, 2: 3 米)、敏感度 (0: 一般, 1: 更敏感)、WIFI模式(0: STA, 1: softAP)、遮挡材料(0: 无遮挡, 1: 金属 或 PCB 板遮挡)、融合距离跟踪(0: 不启用, 1: 启用)、融合 AI(0: 不启用, 1: 启用)等控 制。