Contents

第 1 章 - Azure RTOS LevelX 概述

2021/4/29 •

Azure RTOS LevelX 向嵌入式应用程序提供 NAND 和 NOR 闪存磨损均衡设施。由于 NAND 和 NOR 闪存都只能进行有限次数的擦除，因此，均衡分配闪存的使用至关重要。这通常被称为“磨损均衡”，是通过 LevelX 实现的。

选择重用哪个闪存块的算法主要基于擦除计数，但不是完全基于擦除计数。如果有另一个块的擦除计数在最小擦除计数的可接受增量范围内并且有更多的过时映射，则可能不会选择具有最低擦除计数的块。在这种情况下，具有最多过时映射的块将被擦除和重用，从而节省移动有效映射项的开销。

LevelX 支持 NAND 和/或 NOR 部件的多个实例，也就是说，应用程序可以在同一应用程序中使用 LevelX 的不同实例。每个实例都需要有自己的由应用程序提供的控制块及其自己的闪存驱动程序。

LevelX 向用户提供一个逻辑扇区阵列，这些扇区映射到 LevelX 内的物理闪存。为了增强性能，LevelX 还提供了最新的逻辑扇区映射的缓存。此缓存的大小由编程人员定义。应用程序可以将 LevelX 与 FileX 结合使用，也可以直接读取/写入逻辑扇区。LevelX 不依赖于 FileX，稍微依赖于 ThreadX（只使用基元 ThreadX 数据类型）。

LevelX 是为容错而设计的。闪存更新在多步骤过程中执行，每个步骤都可以中断。在下一操作过程中，LevelX 自动恢复到最佳状态。

LevelX 需要使用闪存驱动程序对基础闪存进行物理访问。我们提供了 NAND 和 NOR 模拟驱动程序的示例，可以从这些示例着手来实现实际的 LevelX 驱动程序。此外，本文档后面部分详细介绍了驱动程序要求。

以下各章介绍了 NAND 和 NOR LevelX 支持的功能操作。

第 2 章 - 安装和使用 Azure RTOS LevelX

2021/4/29 •

本章的以下各节介绍了 Azure RTOS LevelX 的安装和使用，这两项操作起来非常简便。

分发

LevelX 分布在 ANSI C 中，其中每个函数包含在其自己单独的 C 文件中。LevelX 分布文件如下所示。

- lx_api.h
- lx_nand_flash_256byte_ecc_check.c
- lx_nand_flash_256byte_ecc_compute.c
- lx_nand_flash_block_full_update.c
- lx_nand_flash_block_reclaim.c
- lx_nand_flash_close.c
- lx_nand_flash_defragment.c
- lx_nand_flash_extended_cache_enable.c
- lx_nand_flash_initialize.c
- lx_nand_flash_logical_sector_find.c
- lx_nand_flash_next_block_to_erase_find.c
- lx_nand_flash_open.c
- lx_nand_flash_page_ecc_check.c
- lx_nand_flash_page_ecc_compute.c
- lx_nand_flash_partial_defragment.c
- lx_nand_flash_physical_page_allocate.c
- lx_nand_flash_sector_mapping_cache_invalidate.c
- lx_nand_flash_sector_read.c
- lx_nand_flash_sector_release.c
- lx_nand_flash_sector_write.c
- lx_nand_flash_system_error.c
- lx_nor_flash_block_reclaim.c
- lx_nor_flash_close.c
- lx_nor_flash_defragment.c
- lx_nor_flash_extended_cache_enable.c
- lx_nor_flash_initialize.c
- lx_nor_flash_logical_sector_find.c
- lx_nor_flash_next_block_to_erase_find.c
- lx_nor_flash_open.c
- lx_nor_flash_partial_defragment.c
- lx_nor_flash_physical_sector_allocate.c
- lx_nor_flash_sector_mapping_cache_invalidate.c
- lx_nor_flash_sector_read.c
- lx_nor_flash_sector_release.c
- lx_nor_flash_sector_write.c
- lx_nor_flash_system_error.c

还有适用于 LevelX NAND 和 NOR 实例的模拟器和 FileX 驱动程序示例，如下所示。

- demo_filex_nand_flash.c
- fx_nand_flash_simulated_driver.c
- lx_nand_flash_simulator.c
- demo_filex_nor_flash.c
- fx_nor_flash_simulated_driver.c
- lx_nor_flash_simulator.c

当然，如果只要求 NAND 闪存，则只需要 LevelX NAND 闪存文件 (lx_nand_*.c)。同样，如果只要求 NOR 闪存，则只需要 NOR 闪存文件 (**lx_nor_.c***)。

配置选项

可在编译时通过下面所述的条件定义来配置 LevelX。只需将所需的定义添加到每个 LevelX 源的编译即可使用此选项。

- LX_DIRECT_READ：已定义，此选项绕过 NOR 闪存驱动程序读取例程，或直接读取 NOR 内存，从而显著提高性能。
- LX_FREE_SECTOR_DATA_VERIFY：已定义，它会导致 LevelX NOR 实例开放逻辑验证可用的 NOR 扇区是否全部为一。
- LX_NAND_SECTOR_MAPPING_CACHE_SIZE：默认情况下，此值为 16 并定义映射缓存大小的逻辑扇区。较大的值可提高性能，但会占用内存。最小大小为 8，并且所有值必须是 2 的幂。
- LX_NAND_FLASH_DIRECT_MAPPING_CACHE：已定义，它会创建直接映射缓存，这样就不会出现缓存未命中的情况。它还要求 LX_NAND_SECTOR_MAPPING_CACHE_SIZE 表示闪存设备中的总页数。
- LX_NOR_DISABLE_EXTENDED_CACHE：已定义，它禁用了已扩展 NOR 缓存。
- LX_NOR_EXTENDED_CACHE_SIZE：默认情况下，此值为 8，表示最多可以在 NOR 实例中缓存 8 个扇区。
- LX_NOR_SECTOR_MAPPING_CACHE_SIZE：默认情况下，此值为 16 并定义映射缓存大小的逻辑扇区。较大的值可提高性能，但会占用内存。最小大小为 8，并且所有值必须是 2 的幂。
- LX_THREAD_SAFE_ENABLE：已定义，它通过在整个 API 中使用 ThreadX 互斥对象，确保 LevelX 线程安全。

使用 LevelX

要单独使用 LevelX 或与 FileX 一起使用 LevelX，请在引用 LevelX API 的代码中包含文件 lx_api.h。同时还要确保在链接时可以使用 LevelX 对象代码。请检查文件 demo_filex_nand_flash.c 和 demo_filex_nor_flash.c 以获取有关如何使用 LevelX 的示例。

第 3 章 - Azure RTOS LevelX NAND 支持

2021/4/29 •

NAND 闪存通常用于大数据存储，大数据存储是文件系统的典型特征。NAND 内存由块组成。每个 NAND 块中都有一系列页。NAND 块是可擦除的，这意味着 NAND 块中的所有页会被擦除（设置为全部）。每个 NAND 块页都有一组备用字节，Azure RTOS LevelX 将其用于簿记、错误块管理和错误检测。NAND 块页有多种大小可供选择。最常见的页大小为：

PAGE SIZE	□□□
256	8
512	16
2048	64

NAND 内存与 NOR 内存的区别在于没有直接访问，也就是说，NAND 内存不能像 NOR 内存那样直接从处理器读取。NAND 内存只能在擦除后写入有限的次数。这也与 NOR 内存不同，只要写入请求清除设置位，NOR 内存就可以无限次写入。最后，与每个页相关联的备用字节对于 NAND 闪存来说是唯一的。典型的备用字节配置如下表中所示。

□□□	□□	□
8	字节 0-2：	ECC 字节
	字节 3、4、6、7：	LevelX 扇区映射
	字节 5：	错误的块标志
16	字节 0-3、6-7：	ECC 字节
	字节 8-11：	LevelX 扇区映射
	字节 12-15：	未使用
	字节 5：	错误的块标志
64	字节 0：	错误的块标志
	字节 2-5：	LevelX 扇区映射
	字节 6-39：	未使用
	字节 40-63：	ECC 字节

LevelX 利用每个 NAND 页的 4 个备用字节来跟踪映射到物理 NAND 页的逻辑扇区。这 4 个字节用于实现 LevelX 专有格式的 32 位无符号整数。32 位字段的高位（位 31）用于指示逻辑扇区到页面的映射有效。如果此位为 0，则此页中的信息将不再有效。下一位（位 30）用于指示此页即将过时，正在写入新的扇区。位 29 用于指示映射项写入完成的时间。如果位 29 为 0，表示映射项写入完成。如果设置了位 29，表示映射项正在写入过程

中。位 30 和 29 用于在更新新的闪存页时从潜在断电中恢复。最后，低 29 位 (28-0) 包含页的逻辑扇区号。

LevelX 映射条目

▯	▯▯
31	有效标志。如果逻辑扇区并非设置为全部，表示映射有效
30	过时标志。如果清除此项，表示该映射可能已过时或即将过时。
29	如果此位为 0，表示映射条目写入完成
0-28	映射到此物理页面的逻辑扇区（并非全部）。

LevelX 还利用每个 NAND 块的第一页进行块擦除计数，并将其用作块满时的映射页列表。LevelX 中 NAND 块的第一页格式如下所示：

LEVELX ▯▯ 0 ▯▯
[块擦除计数]
[第 1 页扇区映射]
...
[第“n”页扇区映射]
[0xF0F0F0F0]

> **NOTE**
> 只有在块已满时（即，块的所有页都已写入），才能写入页映射信息。这样就可以在运行时更快速地搜索免费页和逻辑扇区。

NAND 错误块支持

NAND 内存也比 NOR 内存更容易出现错误块。这在很大程度上是因为 NAND 制造商可以通过允许错误块并要求软件绕过这些错误块来提高产量。LevelX 通过简单地映射错误块来处理 NAND 错误块管理。

LevelX 还为基础 LevelX 驱动程序提供了 256 字节汉明纠错编码 (ECC) 的 API，以用于计算新的 ECC 代码或对页面的每个 256 字节部分中的页面读取执行 1 位纠错。

NAND 驱动程序要求

LevelX 需要基础的 NAND 闪存驱动程序，该驱动程序特定于基础闪存部分和硬件实现。在初始化期间，通过 API lx_nand_flash_open 将驱动程序指定给 LevelX。LevelX 驱动程序的原型如下所示。

```
INT nand_driver_initialize(LX_NAND_FLASH *instance);
```

实例参数指定 LevelX NAND 控制块。驱动程序初始化函数负责为关联的 LevelX 实例设置所有其他的驱动程序级服务。以下列表显示了每个 LevelX NAND 实例所需的服务。

- 阅读页

- 写入页
- 块擦除
- 验证擦除的块
- 验证擦除的页
- 获取块状态
- 设置块状态
- 获取块额外字节
- 设置块额外字节
- 系统错误处理程序

驱动程序初始化

这些服务是通过在驱动程序初始化函数内的 LX_NAND_FLASH 实例中设置函数指针来设置的。驱动程序初始化函数还指定块总数、每个块的页数、每页的字节数，以及足够将一个页读入内存的 RAM 区域。在返回 LX_SUCCESS 之前，驱动程序初始化函数可能还执行其他特定于设备和/或实现的初始化任务。

驱动程序读取页

LevelX NAND 驱动程序"读取页"服务负责读取 NAND 闪存的特定块中的特定页面。所有错误检查和纠正逻辑都由驱动程序服务负责。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"读取页"服务的原型如下所示。

```
INT nand_driver_read_page(
    ULONG block,
    ULONG page,
    ULONG *destination,
    ULONG words);
```

其中块和页标识要读取的页，目标和字指定要放置页内容的位置以及要读取的 32 位字的数量 。

驱动程序写入页

LevelX NAND 驱动程序"写入页"服务负责将特定页面写入 NAND 闪存的指定块。所有错误检查和 ECC 计算都由驱动程序服务负责。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"写入页"服务的原型如下所示。

```
INT nand_driver_write_page(
    ULONG block,
    ULONG page,
    ULONG *source,
    ULONG words);
```

其中块和页标识要写入的页，源和字指定写入的源以及要写入的 32 位字的数量 。

> **NOTE**
> LevelX 在写入闪存页时依赖驱动程序进行低级错误检测，这通常涉及读回页并与写入缓冲区进行比较，以确保写入成功。

驱动程序块擦除

LevelX NAND 驱动程序"块擦除"服务负责擦除 NAND 闪存的指定块。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"块擦除"服务的

原型如下所示。

```
INT nand_driver_block_erase(ULONG block,
    ULONG erase_count);
```

其中块标识要清除的块。出于诊断目的提供了参数 erase_count。例如，当擦除计数超过特定阈值时，驱动程序可能需要警告应用程序软件的另一部分。

> **NOTE**
> LevelX 在删除块时依赖驱动程序进行低级错误检测，这通常涉及确保块的所有页都是一整页。

验证擦除的驱动程序块

LevelX NAND 驱动程序"验证擦除的块"服务负责验证 NAND 闪存的指定块是否被擦除。如果已擦除，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未擦除该块，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"验证块擦除"服务的原型如下所示：

```
INT nand_driver_block_erased_verify(ULONG block);
```

其中块指定要验证是否被擦除的块。

> **NOTE**
> LevelX 依赖驱动程序检查所有页面以及每页的所有字节（包括备用字节和数据字节），以确保将其擦除（包含所有字节）。

验证擦除的驱动程序页

LevelX NAND 驱动程序"验证擦除的页"服务负责验证 NAND 闪存的指定块的指定页是否被擦除。如果已擦除，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未擦除该页，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"验证页擦除"服务的原型如下所示：

```
INT nand_driver_page_erased_verify(
    ULONG block,
    ULONG page);
```

其中块指定是哪个块，页指定要验证是否已擦除的页 。

> **NOTE**
> LevelX 依赖驱动程序检查指定页面的所有字节（包括备用字节和数据字节），以确保将其擦除（包含所有字节）。

获取驱动程序块状态

LevelX NAND 驱动程序"获取块状态"服务负责检索 NAND 闪存的指定块的错误块标志。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"获取块状态"服务的原型如下所示。

```
INT nand_driver_block_status_get(
    ULONG block,
    UCHAR *bad_block_byte);
```

其中块指定是哪个块，bad_block_byte 指定错误块标志的目标 。

设置驱动程序块状态

LevelX NAND 驱动程序"设置块状态"服务负责设置 NAND 闪存的指定块的错误块标志。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"设置块状态"服务的原型如下所示：

```
INT nand_driver_block_status_set(
    ULONG block,
    UCHAR bad_block_byte);
```

其中块指定是哪个块，bad_block_byte 指定错误块标志的值 。

获取驱动程序块额外字节

LevelX NAND 驱动程序"获取块额外字节"服务负责检索与 NAND 闪存的特定块的特定页相关的额外字节。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"获取块额外字节"服务的原型如下所示：

```
INT nand_driver_block_extra_bytes_get(
    ULONG block,
    ULONG page,
    UCHAR *destination,
    UINT size);
```

其中块指定是哪个块，页指定特定的页，目标指定额外字节的目标 。参数大小指定要获取的额外字节数。

设置驱动程序块额外字节

LevelX NAND 驱动程序"设置块额外字节"服务负责在 NAND 闪存的特定块的特定页中设置额外字节。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"设置块额外字节"服务的原型如下所示：

```
INT nand_driver_block_extra_bytes_set(
    ULONG block,
    ULONG page,
    UCHAR *source,
    UINT size);
```

其中块指定是哪个块，页指定特定的页，源指定额外字节的源 。参数大小指定要设置的额外字节数。

驱动程序系统错误

LevelX NAND 驱动程序"系统错误处理程序"服务负责设置 LevelX 检测到的处理系统错误。此例程中的处理依赖于应用程序。如果成功，LevelX NAND 驱动程序将返回 LX_SUCCESS。如果未成功，LevelX NAND 驱动程序将返回 LX_ERROR。LevelX NAND 驱动程序"系统错误"服务的原型如下所示：

```
INT nand_driver_system_error(
    UINT error_code,
    ULONG block,
    ULONG page);
```

其中块指定是哪个块，页指定发生 error_code 表示的错误的特定页 。

NAND 模拟驱动程序

LevelX 提供模拟的 NAND 闪存驱动程序，该驱动程序只使用 RAM 来模拟 NAND 闪存部件的操作。默认情况下，NAND 模拟驱动程序提供 8 个 NAND 闪存块，每个块 16 页，每页 2048 字节。

模拟的 NAND 闪存驱动程序初始化函数为 lx_nand_flash_simulator_initialize，在 lx_nand_flash_simulator.c 中定义。这个驱动程序还为编写特定的 NAND 闪存驱动程序提供了一个很好的模板。

NAND FileX 集成

如前文所述，LevelX 不依赖于 FileX 进行操作。应用程序软件可以直接调用所有 LevelX API，以将原始数据存储/检索到 LevelX 提供的逻辑扇区。但是，LevelX 也支持 FileX。

文件 fx_nand_flash_simulated_driver.c 包含用于 NAND 闪存模拟的示例 FileX 驱动程序。该驱动程序有趣的一点是，它将通常由 FileX 使用的 512 字节逻辑扇区合并为使用 2048 字节页面对 LevelX 模拟器的单个逻辑扇区读/写请求。这样可以更高效地使用 NAND 闪存。LevelX 的 NAND 闪存 FileX 驱动程序为编写自定义 FileX 驱动程序提供了一个很好的起点。

> **NOTE**
>
> FileX NAND 闪存格式应该是小于 NAND 闪存提供的扇区的一个完整块大小。这将有助于确保在损耗均衡处理期间获得最佳性能。在 LevelX 损耗均衡算法中提高写入性能的其他技巧包含以下内容。

1. 确保所有写入大小都恰好是一个或多个群集，并且在确切的群集边界上启动。
2. 通过 API 的 FileX fx_file_allocate 类执行大文件写入操作之前，请先预分配群集。
3. 确保已启用 FileX 驱动程序以接收释放扇区信息，并通过调用 lx_nor_flash_sector_release 在驱动程序中处理对驱动程序发出的释放扇区的请求。
4. 定期使用 lx_nand_flash_defragment 释放尽可能多的 NAND 块，从而提高写入性能。
5. 利用 lx_nand_flash_extended_cache_enable API 提供各种 NAND 块资源的 RAM 缓存，以提高性能。

第 4 章 - Azure RTOS LevelX NAND API

2021/4/29 •

可用于应用程序的 Azure RTOS LevelX NAND API：

- lx_nand_flash_close：关闭 NAND 闪存实例*
- lx_nand_flash_defragment：对 NAND 闪存实例进行碎片整理*
- lx_nand_flash_extended_cache_enable：启用/禁用扩展 NAND 缓存*
- lx_nand_flash_initialize：初始化 NAND 闪存支持*
- lx_nand_flash_open：打开 NAND 闪存实例*
- lx_nand_flash_page_ecc_check：检查页是否有 ECC 错误和更正*
- lx_nand_flash_page_ecc_compute：计算页的 ECC*
- lx_nand_flash_partial_defragment：NAND 闪存实例的部分碎片整理*
- lx_nand_flash_sector_read：读取 NAND 闪存扇区*
- lx_nand_flash_sector_release：释放 NAND 闪存扇区*
- lx_nand_flash_sector_write：写入 NAND 闪存扇区*

lx_nand_flash_close

关闭 NAND 闪存实例

原型

```
UINT lx_nand_flash_close(LX_NAND_FLASH *nand_flash);
```

说明

此服务关闭先前打开的 NAND 闪存实例。

输入参数

- nand_flash：NAND 闪存实例指针。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0x01) 关闭闪存实例时出错。

允许来自

线程数

示例

```
/* Close NAND flash instance "my_nand_flash". */
status = lx_nand_flash_close(&my_nand_flash);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize

- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_defragment

对 NAND 闪存实例进行碎片整理

原型

```
UINT lx_nand_flash_defragment(LX_NAND_FLASH *nand_flash);
```

说明

此服务对先前打开的 NAND 闪存实例进行碎片整理。碎片整理进程最大限度地增加可用页和块的数量。

输入参数

- nand_flash：NAND 闪存实例指针。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0X01) 对闪存实例进行碎片整理时出错。

允许来自

线程数

示例

```
/* Defragment NAND flash instance "my_nand_flash". */
status = lx_nand_flash_defragment(&my_nand_flash);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_extended_cache_enable

启用/禁用扩展 NAND 缓存

原型

```
UINT lx_nand_flash_extended_cache_enable(
    LX_NAND_FLASH
    *nand_flash,
    VOID *memory,
    ULONG size);
```

说明

此服务使用应用程序提供的内存在 RAM 中实现缓存层。完全缓存操作所需的内存总量可以按如下方式计算：

```
size (in_bytes) = number_of_blocks (rounded up to be divisible by 4) +
    ((number_of_blocks * pages_per_block) * 4)  +
    ((number_of_blocks * (pages_per_block + 1)) * 4)
```

如果提供的内存不够大，无法容纳完全 NAND 缓存，则此例程会根据提供的内存尽可能多地启用 NAND 闪存缓存。

如果指定的内存地址为 NULL，将禁用 NAND 缓存。因此，NAND 缓存可能以临时方式使用。

输入参数

- nand_flash：NAND 闪存实例指针。
- memory：缓存内存的起始地址，为 ULONG 访问而对齐。LX_NULL 值禁用缓存。
- size：提供的内存大小（以字节为单位）。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0x01) NAND 缓存的一个元素的内存不足。

允许来自

线程数

示例

```
/* Enable the NAND flash cache for the instance "my_nand_flash". */
status = lx_nand_flash_extended_cache_enable(&my_nand_flash,
    &my_memory, sizeof(my_memory));

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_initialize

初始化 NAND 闪存支持

原型

```
UINT lx_nand_flash_initialize(void);
```

说明

此服务初始化 LevelX NAND 闪存支持。必须在任何其他 LevelX NAND API 之前调用它。

输入参数

- **无**

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0x01) 初始化 NAND 闪存支持时出错。

允许来自

初始化、线程

示例

```
/* Initialize NAND flash support. */
status = lx_nand_flash_initialize();

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_open

打开 NAND 闪存实例

原型

```
UINT lx_nand_flash_open(
    LX_NAND_FLASH *nand_flash,
    CHAR *name,
    UINT (*nand_driver_initialize) (LX_NAND_FLASH *));
```

说明

此服务使用指定的 NAND 闪存控制块和驱动程序初始化函数打开一个 NAND 闪存实例。请注意，驱动程序初始化函数负责安装各种函数指针，用于读取、写入和擦除与此 NAND 闪存实例相关联的 NAND 硬件的块/页。

输入参数

- nand_flash：NAND 闪存实例指针。
- name：NAND 闪存实例的名称。
- nand_driver_initialize：指向 NAND 闪存驱动程序初始化函数的函数指针。有关 NAND 闪存驱动程序责任的详细信息，请参阅本指南的第 3 章。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0x01) 打开 NAND 闪存实例时出错。
- LX_NO_MEMORY：(0x08) 驱动程序未提供用于将页读取到 RAM 的缓冲区。

允许来自

线程数

示例

```
/* Open NAND flash instance "my_nand_flash" with the driver "my_nand_driver_initialize". */
status = lx_nand_flash_open(&my_nand_flash,"my nand flash",
    my_nand_driver_initialize);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_page_ecc_check

检查页是否有 ECC 错误和更正

原型

```
UINT lx_nand_flash_page_ecc_check(
    LX_NAND_FLASH *nand_flash,
    UCHAR *page_buffer,
    UCHAR *ecc_buffer);
```

说明

此服务通过提供的 ECC 验证所提供的 NAND 页缓冲区的完整性。页大小（在 NAND 闪存实例指针中定义）假定为 256 字节的倍数，提供的 ECC 代码能够在页的每 256 字节部分纠正 1 位错误。

输入参数

- nand_flash：NAND 闪存实例指针。
- page_buffer：指向 NAND 闪存页缓冲区的指针。
- ecc_buffer：指向 NAND 闪存页的 ECC 的指针。请注意，页的每 256 字节部分有 3 个 ECC 字节。

返回值

- LX_SUCCESS：(0x00) NAND 页没有错误。
- LX_NAND_ERROR_CORRECTED：(0x06) 在 NAND 页中更正了一个或多个 1 位错误（更正位于页缓冲区）。
- LX_NAND_ERROR_NOT_CORRECTED：(0x07) NAND 页上有太多错误，无法更正。

允许来自

LevelX 驱动程序

示例

```
/* Check the NAND page pointed to by "page_pointer" of the NAND flash instance "my_nand_flash" . */
status = lx_nand_flash_page_ecc_check(&my_nand_flash, page_pointer, ecc_pointer);

/* If status is LX_SUCCESS the page is valid. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_page_ecc_compute

计算页的 ECC

原型

```
UINT lx_nand_flash_page_ecc_compute(
    LX_NAND_FLASH *nand_flash,
    UCHAR *page_buffer,
    UCHAR *ecc_buffer);
```

说明

此服务计算提供的 NAND 页缓冲区的 ECC，并返回提供的 ECC 缓冲区中的 ECC。页面大小假设为 256 字节的倍数（在 NAND 闪存实例指针中定义）。ECC 代码用于在稍后读取页时验证页的完整性。

输入参数

- nand_flash：NAND 闪存实例指针。
- page_buffer：指向 NAND 闪存页缓冲区的指针。
- ecc_buffer：指向 NAND 闪存页 ECC 的目标的指针。请注意，页的每 256 字节部分必须有 3 字节的 ECC 存储。例如，2048 字节页需要 24 字节的 ECC。

返回值

- LX_SUCCESS：(0x00) 已成功计算 ECC。
- LX_ERROR：(0x01) 计算 ECC 时出错。

允许来自

LevelX 驱动程序

示例

```
/* Compute ECC for the NAND page pointed to by "page_pointer" of the NAND flash instance "my_nand_flash". */
status = lx_nand_flash_page_ecc_compute(&my_nand_flash, page_pointer, ecc_pointer);

/* If status is LX_SUCCESS the ECC was calculated and Can be found in the memory pointed to by
"ecc_pointer." */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_partial_defragment

NAND 闪存实例的部分碎片整理

原型

```
UINT lx_nand_flash_partial_defragment(
    LX_NAND_FLASH *nand_flash,
    UINT max_blocks);
```

说明

此服务将以前打开的 NAND 闪存实例碎片整理到指定的最大块数。碎片整理进程最大限度地增加可用页和块的数量。

输入参数

- nand_flash：NAND 闪存实例指针。
- max_blocks：最大块数。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0X01) 对闪存实例进行碎片整理时出错。

允许来自

线程数

示例

```
/* Defragment 1 block of NAND flash instance "my_nand_flash". */
status = lx_nand_flash_partial_defragment(&my_nand_flash, 1);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_sector_read

读取 NAND 闪存扇区

原型

```
UINT lx_nand_flash_sector_read(
    LX_NAND_FLASH *nand_flash,
    ULONG logical_sector,
    VOID *buffer);
```

说明

此服务从 NAND 闪存实例读取逻辑扇区，如果成功，则返回所提供的缓冲区中的内容。请注意，NAND 扇区大小始终是基础 NAND 硬件的页大小。

输入参数

- nand_flash：NAND 闪存实例指针。
- logical_sector：要读取的逻辑扇区。
- buffer：指向逻辑扇区内容的目标的指针。请注意，缓冲区的大小假定为 NAND 闪存页大小，并为 ULONG 访问而对齐。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0x01) 读取 NAND 闪存扇区时出错。

允许来自

线程数

示例

```
/* Read logical sector 20 of the NAND flash instance "my_nand_flash" and place contents in "buffer". */
status = lx_nand_flash_sector_read(&my_nand_flash, 20, buffer);

/* If status is LX_SUCCESS, "buffer" contains the contentsnof logical sector 20. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_release
- lx_nand_flash_sector_write

lx_nand_flash_sector_release

释放 NAND 闪存扇区

原型

```
UINT lx_nand_flash_sector_release(
    LX_NAND_FLASH *nand_flash,
    ULONG logical_sector);
```

说明

此服务在 NAND 闪存实例中释放逻辑扇区映射。在未使用时释放逻辑扇区会使 LevelX 的损耗均衡更有效率。

输入参数

- nand_flash：NAND 闪存实例指针。
- logical_sector：要释放的逻辑扇区。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_ERROR：(0x01) 写入 NAND 闪存扇区时出错。

允许来自

线程数

示例

```
/* Release logical sector 20 of the NAND flash instance "my_nand_flash". */
status = lx_nand_flash_sector_release(&my_nand_flash, 20);

/* If status is LX_SUCCESS, logical sector 20 has been released. */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read

- lx_nand_flash_sector_write

lx_nand_flash_sector_write

写入 NAND 闪存扇区

原型

```
UINT lx_nand_flash_sector_write(
    LX_NAND_FLASH *nand_flash,
    ULONG logical_sector,
    VOID *buffer);
```

说明

此服务在 NAND 闪存实例中写入指定的逻辑扇区。

输入参数

- nand_flash：NAND 闪存实例指针。
- logical_sector：要写入的逻辑扇区。
- buffer：指向逻辑扇区内容的指针。请注意，缓冲区的大小假定为 NAND 闪存页大小，并为 ULONG 访问而对齐。

返回值

- LX_SUCCESS：(0x00) 成功的请求。
- LX_NO_SECTORS：(0x02) 没有更多可用扇区来执行写入操作。
- LX_ERROR：(0x01) 释放 NAND 闪存扇区时出错。

允许来自

线程数

示例

```
/* Write logical sector 20 of the NAND flash instance "my_nand_flash" with the contents pointed to by
"buffer". */
status = lx_nand_flash_sector_write(&my_nand_flash, 20, buffer);

/* If status is LX_SUCCESS, logical sector 20 has been written with the contents of "buffer". */
```

另请参阅

- lx_nand_flash_close
- lx_nand_flash_defragment
- lx_nand_flash_extended_cache_enable
- lx_nand_flash_initialize
- lx_nand_flash_open
- lx_nand_flash_page_ecc_check
- lx_nand_flash_page_ecc_compute
- lx_nand_flash_partial_defragment
- lx_nand_flash_sector_read
- lx_nand_flash_sector_release

第 5 章 - Azure RTOS LevelX NOR 支持

2021/4/29 •

NOR 闪存由块组成，这些块通常被平均分隔为 512 字节。NOR 闪存中没有闪存页的概念。另外，NOR 闪存中也没有备用字节，因此 Azure RTOS LevelX 必须对所有管理信息使用 NOR 闪存本身。在 NOR 闪存中可以直接进行读取访问。写入访问通常需要一系列特殊操作。只要清除位，就可以多次写入 NOR 闪存。通过擦除块操作，NOR 闪存中的位只能设置一次。

LevelX 将每个或 NOR 闪存块分割为 512 字节的逻辑扇区。此外，LevelX 使用每个 NOR 闪存块的前“n”个扇区来存储管理信息。LevelX NOR 闪存管理信息的格式如下所示：

LevelX NOR 块格式

tttttt	tt
0	[块擦除计数]
4	[最小映射扇区]
8	[最大映射扇区]
12	[可用扇区位图]
m	[扇区 0 映射条目]
	...
m+4*(n-1)	[扇区“n”映射条目]
	...
s	[扇区 0 内容]
	...
s+512*(n-1)	[扇区“n”内容]

32 位块擦除计数包含块的擦除次数。LevelX 的主要目标是使所有块的擦除计数相对接近，以帮助防止任何一个块过早耗尽。仅当块中的所有逻辑扇区都已被映射并写入时，才会写入 32 位的最小映射扇区和最大映射扇区字段 。这些字段有助于优化扇区读取操作。“可用扇区位图”条目是一个位图，其中每个设置的位对应于块中未映射的扇区。此字段用于提高可用扇区搜索的效率。这是一个可变长度字段，该字段需要块中每 32 个扇区有一个字。“扇区映射条目”数组包含块中每个扇区的映射信息。每个条目均具有以下格式：

扇区映射条目

t	tt
31	有效标志。如果逻辑扇区并非设置为全部扇区，表示映射有效

[	[[
30	过时标志。如果清除此项，表示该映射可能已过时或即将过时。
29	如果此位为 0，表示映射条目写入完成
0-28	映射到此物理扇区的逻辑扇区（并非全部扇区）。

32 位字段的高位（位 31）用于指示逻辑扇区映射有效。如果此位为 0，则此条目中的信息（以及相应的扇区内容）将不再有效。下一位（位 30）用于指示此扇区即将过时，将写入新的扇区。位 29 用于指示映射条目写入完成的时间。如果位 29 为 0，表示映射条目写入完成。如果设置了位 29，表示映射条目正在写入过程中。位 30 和 29 用于在更新新的扇区映射时从潜在断电中恢复。最后，低 29 位 (28-0) 包含扇区的逻辑扇区号。如果未映射扇区，则会设置所有位，也就是说，其值将为 0xFFFFFFFF。

NOR 驱动程序要求

LevelX 需要基础的 NOR 闪存驱动程序，该驱动程序特定于基础闪存部分和硬件实现。在初始化期间，通过 API lx_nor_flash_open 将驱动程序指定给 LevelX。LevelX 驱动程序的原型如下：

```
INT nor_driver_initialize(LX_NOR_FLASH *instance);
```

“instance”参数指定 LevelX NOR 控制块。驱动程序初始化函数负责为关联的 LevelX 实例设置所有其他的驱动程序级服务。每个 LevelX NOR 实例所需的服务如下：

- 读取扇区
- 写入扇区
- 块擦除
- 验证擦除的块
- 系统错误处理程序

驱动程序初始化

这些服务是通过在驱动程序初始化函数内的 LX_NOR_FLASH 实例中设置函数指针来设置的。驱动程序初始化函数还负责：

1. 指定闪存的基址。
2. 指定块总数和每个块的字词数。
3. RAM 缓冲区，用于读取闪存的某一扇区（512 字节）并保持一致以获取 ULONG 访问。

在返回 LX_SUCCESS 之前，驱动程序初始化函数可能还执行其他特定于设备和/或实现的初始化任务。

驱动程序读取扇区

LevelX NOR 驱动程序“读取扇区”服务负责读取 NOR 闪存的特定块中的特定扇区。所有错误检查和纠正逻辑都由驱动程序服务负责。如果成功，LevelX NOR 驱动程序将返回 LX_SUCCESS。如果失败，LevelX NOR 驱动程序将返回 LX_ERROR。LevelX NOR 驱动程序“读取扇区”服务的原型如下：

```
INT nor_driver_read_sector(
    ULONG *flash_address,
    ULONG *destination,
    ULONG words);
```

其中"flash_address"指定内存的 NOR 闪存块内某个逻辑扇区的地址，"destination"和"words"指定扇区内容的放置位置以及要读取的 32 位字的数量 。

驱动程序写入扇区

LevelX NOR 驱动程序"写入扇区"服务负责写入 NOR 闪存的块中的特定扇区。所有错误检查都由驱动程序服务负责。如果成功，LevelX NOR 驱动程序将返回 LX_SUCCESS。如果失败，LevelX NOR 驱动程序将返回 LX_ERROR。LevelX NOR 驱动程序"写入扇区"服务的原型如下：

```
INT nor_driver_write_sector(
    ULONG *flash_address,
    ULONG *source,
    ULONG words);
```

其中"flash_address"指定内存的 NOR 闪存块内某个逻辑扇区的地址，"source"和"words"指定写入的源以及要写入的 32 位字的数量 。

> **NOTE**
> LevelX 依赖驱动程序来验证写入扇区是否已成功。这通常通过读取回编程值来实现，以确保它与要写入的请求的值相匹配。

驱动程序块擦除

LevelX NOR 驱动程序"块擦除"服务负责擦除 NOR 闪存的指定块。如果成功，LevelX NOR 驱动程序将返回 LX_SUCCESS。如果失败，LevelX NOR 驱动程序将返回 LX_ERROR。LevelX NOR 驱动程序"块擦除"服务的原型如下：

```
INT nor_driver_block_erase(ULONG block,
    ULONG erase_count);
```

其中"block"标识要清除的 NOR 块。出于诊断目的提供了参数"erase_count"。例如，当擦除计数超过特定阈值时，驱动程序可能需要警告应用程序软件的另一部分。

> **NOTE**
> LevelX 依赖驱动程序检查块的所有字节，以确保将其擦除(包含所有字节)。

验证擦除的驱动程序块

LevelX NOR 驱动程序"验证擦除的块"服务负责验证 NOR 闪存的指定块是否被擦除。如果已被擦除，LevelX NOR 驱动程序将返回 LX_SUCCESS。如果未擦除该块，LevelX NOR 驱动程序将返回 LX_ERROR。LevelX NOR 驱动程序"验证块擦除"服务的原型如下：

```
INT nor_driver_block_erased_verify(ULONG block);
```

其中"block"指定要验证是否被擦除的块。

> **NOTE**
> LevelX 依赖驱动程序检查指定块的所有字节，以确保将其擦除(包含所有字节)。

驱动程序系统错误

LevelX NOR 驱动程序"系统错误处理程序"服务负责设置 LevelX 检测到的处理系统错误。此例程中的处理依赖于应用程序。如果成功，LevelX NOR 驱动程序将返回 LX_SUCCESS。如果失败，LevelX NOR 驱动程序将返回 LX_ERROR。LevelX NOR 驱动程序"系统错误"服务的原型如下：

```
INT nor_driver_system_error(UINT error_code);
```

其中"error_code"表示发生的错误。

NOR 模拟驱动程序

LevelX 提供模拟的 NOR 闪存驱动程序，该驱动程序只使用 RAM 来模拟 NOR 闪存部件的操作。默认情况下，NOR 模拟驱动程序提供 8 个 NOR 闪存块，每个块 16 个扇区，每扇区 512 字节。

模拟的 NOR 闪存驱动程序初始化函数为 lx_nor_flash_simulator_initialize，在 x_nor_flash_simulator.c 中定义。此驱动程序还为编写特定的 NOR 闪存驱动程序提供了一个很好的模板。

NOR FileX 集成

如前文所述，LevelX 不依赖于 FileX 进行操作。应用程序软件可以直接调用所有 LevelX API，以将原始数据存储/检索到 LevelX 提供的逻辑扇区。但是，LevelX 也支持 FileX。

文件 fx_nor_flash_simulated_driver.c 包含用于 NOR 闪存模拟的示例 FileX 驱动程序。LevelX 的 NOR 闪存 FileX 驱动程序为编写自定义 FileX 驱动程序提供了一个很好的起点。

> **NOTE**
>
> FileX NOR 闪存格式应该是小于 NOR 闪存提供的扇区的一个完整块大小。这将有助于确保在损耗均衡处理期间获得最佳性能。在 LevelX 损耗均衡算法中提高写入性能的其他技巧包含：
>
> 1. 确保所有写入大小都恰好是一个或多个群集，并且在确切的群集边界上启动。
> 2. 通过 API 的 FileX fx_file_allocate 类执行大文件写入操作之前，请先预分配群集。
> 3. 定期使用 lx_nor_flash_defragment 释放尽可能多的 NOR 块，从而提高写入性能。
> 4. 确保已启用 FileX 驱动程序以接收释放扇区信息，并通过调用 lx_nor_flash_sector_release 在驱动程序中处理对驱动程序发出的释放扇区的请求。

第 6 章 - Azure RTOS LevelX NOR API

2021/4/29 •

可用于应用程序的 Azure RTOS LevelX NOR API 函数如下所示。

- lx_nor_flash_close*_：关闭 NOR 闪存实例
- lx_nor_flash_defragment*_：对 NOR 闪存实例进行碎片整理
- lx_nor_flash_extended_cache_enable*_：启用/禁用扩展 NOR 缓存
- lx_nor_flash_initialize*_：初始化 NOR 闪存支持
- lx_nor_flash_open*_：打开 NOR 闪存实例
- lx_nor_flash_partial_defragment*_：NOR 闪存实例的部分碎片整理
- lx_nor_flash_sector_read*_：读取 NOR 闪存扇区
- lx_nor_flash_sector_release*_：释放 NOR 闪存扇区
- lx_nor_flash_sector_write*_：写入 NOR 闪存扇区

lx_nor_flash_close

关闭 NOR 闪存实例

原型

```
UINT lx_nor_flash_close(LX_NOR_FLASH *nor_flash);
```

说明

此服务关闭先前打开的 NOR 闪存实例。

输入参数

- nor_flash：NOR 闪存实例指针。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0x01) 关闭闪存实例时出错。

允许来自

线程数

示例

```
/* Close NOR flash instance "my_nor_flash". */
status = lx_nor_flash_close(&my_nor_flash);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nor_flash_defragment
- lx_nor_flash_extended_cache_enable
- lx_nor_flash_initialize
- lx_nor_flash_open
- lx_nor_flash_partial_defragment

- lx_nor_flash_sector_read
- lx_nor_flash_sector_release
- lx_nor_flash_sector_write

lx_nor_flash_defragment

对 NOR 闪存实例进行碎片整理

原型

```
UINT lx_nor_flash_defragment(LX_NOR_FLASH *nor_flash);
```

说明

此服务对先前打开的 NOR 闪存实例进行碎片整理。碎片整理进程最大限度地增加可用扇区和块的数量。

输入参数

- nor_flash：NOR 闪存实例指针。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0X01) 对闪存实例进行碎片整理时出错。

允许来自

线程数

示例

```
/* Defragment NOR flash instance "my_nor_flash". */
status = lx_nor_flash_defragment(&my_nor_flash);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_extended_cache_enable
- lx_nor_flash_initialize
- lx_nor_flash_open
- lx_nor_flash_partial_defragment
- lx_nor_flash_sector_read
- lx_nor_flash_sector_release
- lx_nor_flash_sector_write

lx_nor_flash_extended_cache_enable

启用/禁用扩展 NOR 缓存

原型

```
UINT lx_nor_flash_extended_cache_enable(
    LX_NOR_FLASH *nor_flash,
    VOID *memory,
    ULONG size);
```

说明

此服务使用应用程序提供的内存在 RAM 中实现 NOR 扇区缓存层。提供的每个 512 字节的内存都转换为可以缓存的一个 NOR 扇区。缓存的扇区是包含块控制信息的扇区，例如清除计数、可用扇区映射和扇区映射信息。数据扇区不存储在此缓存中。

输入参数

- nor_flash：NOR 闪存实例指针。
- memory：缓存内存的起始地址，为 ULONG 访问而对齐。LX_NULL 值禁用缓存。
- size：提供的内存的大小（以字节为单位，应为 512 字节的倍数）。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0X01) 一个 NOR 扇区的内存不足。
- LX_DISABLED：(0x09) 配置选项禁用的 NOR 扩展缓存。

允许来自

线程数

示例

```
/* Enable the NOR flash cache for the instance "my_nor_flash". */
status = lx_nor_flash_extended_cache_enable(&my_nor_flash,
    &my_memory, sizeof(my_memory));

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_defragment
- lx_nor_flash_initialize
- lx_nor_flash_open
- lx_nor_flash_partial_defragment
- lx_nor_flash_sector_read
- lx_nor_flash_sector_release
- lx_nor_flash_sector_write

lx_nor_flash_initialize

初始化 NOR 闪存支持

原型

```
UINT lx_nor_flash_initialize(void);
```

说明

此服务初始化 LevelX NOR 闪存支持。必须在任何其他 LevelX NOR API 之前调用它。

输入参数

- **无**

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0x01) 初始化 NOR 闪存支持时出错。

允许来自

初始化、线程

示例

```
/* Initialize NOR flash support. */
status = lx_nor_flash_initialize();

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_defragment
- lx_nor_flash_extended_cache_enable
- lx_nor_flash_partial_defragment
- lx_nor_flash_open
- lx_nor_flash_sector_read
- lx_nor_flash_sector_release
- lx_nor_flash_sector_write

lx_nor_flash_open

打开 NOR 闪存实例

原型

```
UINT lx_nor_flash_open(
    LX_NOR_FLASH *nor_flash,
    CHAR *name,
    UINT (*nor_driver_initialize) (LX_NOR_FLASH *));
```

说明

此服务使用指定的 NOR 闪存控制块和驱动程序初始化函数打开一个 NOR 闪存实例。请注意，驱动程序初始化函数负责安装各种函数指针，用于读取、写入和清除与此 NOR 闪存实例相关联的 NOR 硬件的块。

输入参数

- nor_flash：NOR 闪存实例指针。
- name：NOR 闪存实例的名称。
- nor_driver_initialize：指向 NOR 闪存驱动程序初始化函数的函数指针。有关 NOR 闪存驱动程序责任的详细信息，请参阅本指南的第 5 章。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0x01) 打开 NOR 闪存实例时出错。
- LX_NO_MEMORY：(0X08) 驱动程序未提供用于将无扇区读取到 RAM 的缓冲区。

允许来自

线程数

示例

```
/* Open NOR flash instance "my_nor_flash" with the driver "my_nor_driver_initialize". */
status = lx_nor_flash_open(&my_nor_flash,"my NOR flash",
    my_nor_driver_initialize);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_defragment
- lx_nor_flash_extended_cache_enable
- lx_nor_flash_initialize
- lx_nor_flash_partial_defragment
- lx_nor_flash_sector_read
- lx_nor_flash_sector_release
- lx_nor_flash_sector_write

lx_nor_flash_partial_defragment

NOR 闪存实例的部分碎片整理

原型

```
UINT lx_nor_flash_partial_defragment(
    LX_NOR_FLASH *nor_flash,
    UINT max_blocks);
```

说明

此服务将以前打开的 NOR 闪存实例碎片整理到指定的最大块数。碎片整理进程最大限度地增加可用扇区和块的数量。

输入参数

- nor_flash：NOR 闪存实例指针。
- max_blocks：最大块数。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0X01) 对闪存实例进行碎片整理时出错。

允许来自

线程数

示例

```
/* Defragment of one block in NOR flash instance* *"my_nor_flash". */
status = lx_nor_flash_partial_defragment(&my_nor_flash, 1);

/* If status is LX_SUCCESS the request was successful. */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_defragment
- lx_nor_flash_extended_cache_enable

- lx_nor_flash_initialize
- lx_nor_flash_open
- lx_nor_flash_sector_read
- lx_nor_flash_sector_release
- lx_nor_flash_sector_write

lx_nor_flash_sector_read

读取 NOR 闪存扇区

原型

```
UINT lx_nor_flash_sector_read(
    LX_NOR_FLASH *nor_flash,
    ULONG logical_sector,
    VOID *buffer);
```

说明

此服务从 NOR 闪存实例读取逻辑扇区，如果成功，则返回所提供的缓冲区中的内容。请注意，NOR 扇区大小始终为 512 字节。

输入参数

- nor_flash：NOR 闪存实例指针。
- logical_sector：要读取的逻辑扇区。
- buffer：指向逻辑扇区内容的目标的指针。请注意，该缓冲区假定为 512 字节，并为 ULONG 访问而对齐。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0x01) 读取 NOR 闪存扇区时出错。

允许来自

线程数

示例

```
/* Read logical sector 20 of the NOR flash instance "my_nor_flash" and place contents in "buffer". */
status = lx_nor_flash_sector_read(&my_nor_flash, 20, buffer);

/* If status is LX_SUCCESS, "buffer" contains the contents of logical sector 20. */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_defragment
- lx_nor_flash_extended_cache_enable
- lx_nor_flash_initialize
- lx_nor_flash_open
- lx_nor_flash_partial_defragment
- lx_nor_flash_sector_release
- lx_nor_flash_sector_write

lx_nor_flash_sector_release

释放 NOR 闪存扇区

原型

```
UINT lx_nor_flash_sector_release(
    LX_NOR_FLASH *nor_flash,
    ULONG logical_sector);
```

说明

此服务在 NOR 闪存实例中释放逻辑扇区映射。在未使用时释放逻辑扇区会使 LevelX 的损耗均衡更有效率。

输入参数

- nor_flash：NOR 闪存实例指针。
- logical_sector：要释放的逻辑扇区。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_ERROR：(0x01) NOR 闪存扇区写入时出错。

允许来自

线程数

示例

```
/* Release logical sector 20 of the NOR flash instance "my_nor_flash". */
status = lx_nor_flash_sector_release(&my_nor_flash, 20);

/* If status is LX_SUCCESS, logical sector 20 has been released. */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_defragment
- lx_nor_flash_extended_cache_enable
- lx_nor_flash_initialize
- lx_nor_flash_open
- lx_nor_flash_partial_defragment
- lx_nor_flash_sector_read
- lx_nor_flash_sector_write

lx_nor_flash_sector_write

写入 NOR 闪存扇区

原型

```
UINT lx_nor_flash_sector_write(
    LX_nor_FLASH *NOR_flash,
    ULONG logical_sector,
    VOID *buffer);
```

说明

此服务在 NOR 闪存实例中写入指定的逻辑扇区。

输入参数

- nor_flash：NOR 闪存实例指针。

- logical_sector：要写入的逻辑扇区。
- buffer：指向逻辑扇区内容的指针。请注意，该缓冲区假定为 512 字节，为 ULONG 访问而对齐。

返回值

- LX_SUCCESS：(0X00) 成功的请求。
- LX_NO_SECTORS：(0X02) 没有更多可用扇区来执行写入操作。
- LX_ERROR：(0x01) 释放 NOR 闪存扇区时出错。

允许来自

线程数

示例

```
/* Write logical sector 20 of the NOR flash instance "my_nor_flash" with the contents pointed to by
"buffer". */
status = lx_nor_flash_sector_write(&my_nor_flash, 20, buffer);

/* If status is LX_SUCCESS, logical sector 20 has been written with the contents of "buffer". */
```

另请参阅

- lx_nor_flash_close
- lx_nor_flash_defragment
- lx_nor_flash_extended_cache_enable
- lx_nor_flash_initialize
- lx_nor_flash_open
- lx_nor_flash_partial_defragment
- lx_nor_flash_sector_read
- lx_nor_flash_sector_release