General-Purpose I/O （GPIO） 驱动程序设计指南

2025/07/18

本部分介绍如何为常规用途 I/O（GPIO） 控制器设备编写驱动程序。 GPIO 控制器将 GPIO 引脚配置为执行低速数据 I/O 操作、充当设备选择信号以及接收中断请求。 从 Windows 8 开始，GPIO 框架扩展（GpioClx）简化了为 GPIO 控制器编写驱动程序的任务。 此外，GpioClx 为与连接到控制器上的 GPIO 引脚的设备通信的外围设备驱动程序提供了统一的 I/O 请求接口。

本部分内容

展开表

主题	DESCRIPTION
GPIO 驱动程序支持概述	从 Windows 8 开始，GPIO 框架扩展（GpioClx）简化了为 GPIO 控制器设备编写驱动程序的任务。 此外，GpioClx 为连接到 GPIO 引脚的外围设备提供驱动程序支持。 GpioClx 是内核模式驱动程序框架（KMDF）的系统提供的扩展，它执行 GPIO 设备类成员通用的处理任务。
GpioClx I/O 和中断接口	通常，GPIO 控制器的客户端是连接到 GPIO 引脚的外围设备的驱动程序。 这些驱动程序使用 GPIO 引脚作为低带宽数据通道、设备选择输出和中断请求输入。 外围设备驱动程序打开与配置为数据输入或输出的 GPIO 引脚的逻辑连接。 通过这些连接发送 I/O 请求到这些引脚。 此外，外围设备驱动程序可以在逻辑上将其中断服务例程连接到配置为中断请求输入的 GPIO 引脚。
GPIO-Based 硬件资源	从 Windows 8 开始，由 GPIO 控制器驱动程序控制的常规用途 I/O（GPIO）引脚作为系统管理的硬件资源可供其他驱动程序使用。 GPIO I/O 引脚（配置为数据输入或数据输出的引脚）可用作新的 Windows 资源类型 、*GPIO I/O 资源*。 此外，GPIO 中断引脚（配置为中断请求输入的引脚）可用作普通 Windows 中断资源。
GPIO 中断	某些通用 I/O （GPIO）控制器设备可以将其 GPIO 引脚配置为中断请求输入。 这些中断请求输入由物理连接到 GPIO 引脚的外围设备驱动。 这些 GPIO 控制器的驱动程序可以在单个 GPIO 引脚上启用、禁用、屏蔽、取消屏蔽和清除中断请求。

主题	DESCRIPTION
GpioClx DDI	常规用途 I/O (GPIO) 控制器驱动程序通过 GpioClx 设备驱动程序接口 (DDI) 与 GPIO 框架扩展 (GpioClx) 通信。 此 DDI 在 Gpioclx.h 头文件中定义，并在 General-Purpose I/O （GPIO） 驱动程序参考中介绍。 作为此 DDI 的一部分，GpioClx 实现多个驱动程序支持方法，这些方法由 GPIO 控制器驱动程序调用。 此驱动程序实现了一组事件回调函数，这些函数由 GpioClx 调用。 GpioClx 使用这些回调来管理已配置为中断输入的 GPIO 引脚提供的中断请求，并将数据传输到已配置为数据输入和输出的 GPIO 引脚，或者从其传输出来。

GPIO 驱动程序支持概述

项目 • 2023/06/15

从 Windows 8 开始，GPIO 框架扩展 (GpioClx) 简化了为 GPIO 控制器设备编写驱动程序的任务。 另外，GpioClx 还为连接到 GPIO 引脚的外围设备提供驱动程序支持。 GpioClx 是系统提供的针对内核模式驱动程序框架 (KMDF) 的扩展，其执行的处理任务对于 GPIO 设备类的成员来说很常见。

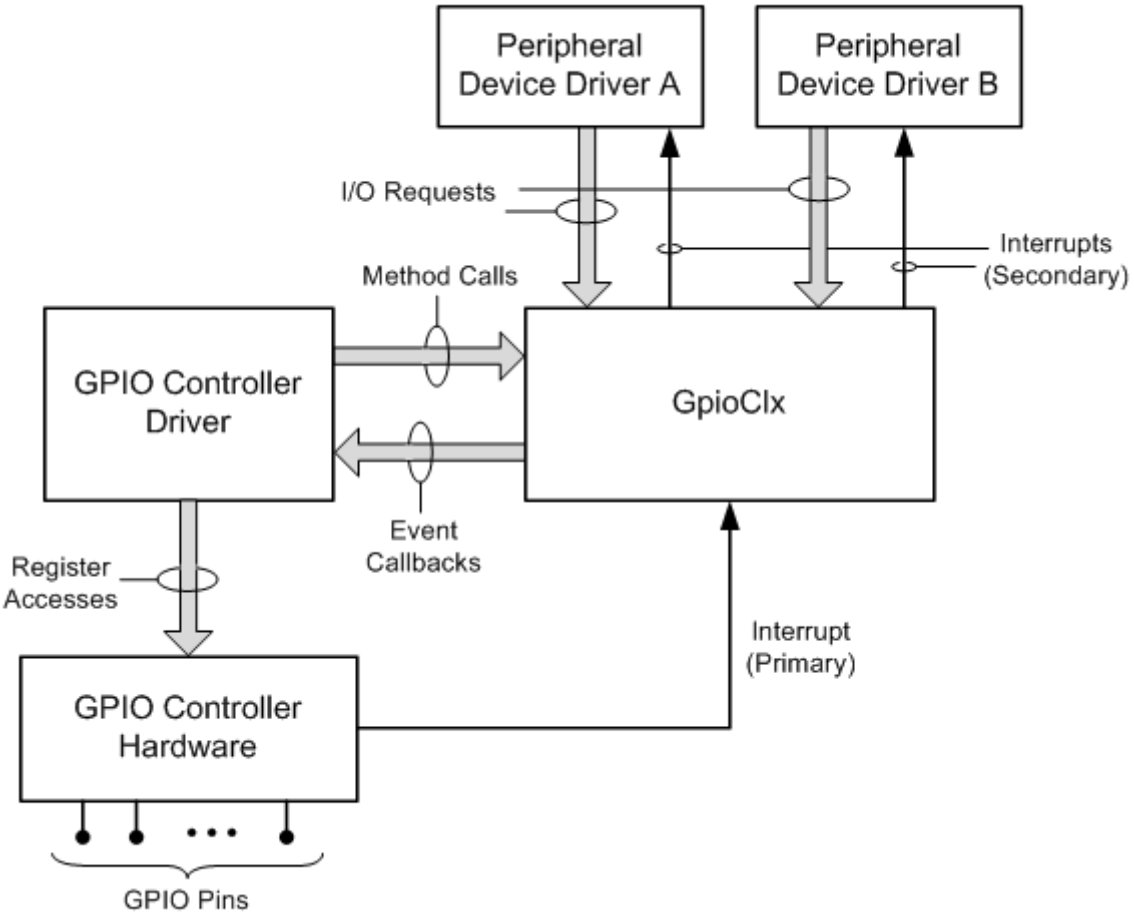
本概述讨论以下主题：

GPIO 控制器驱动程序

硬件供应商提供驱动程序来控制其 GPIO 控制器。 GPIO 控制器驱动程序是一种 KMDF 驱动程序，用于管理 GPIO 控制器的所有特定于硬件的操作。 GPIO 控制器驱动程序与 GpioClx 协作处理配置为数据输入和数据输出的 GPIO 引脚组的 I/O 请求。 此外，此驱动程序与 GpioClx 合作处理来自配置为中断输入的 GPIO 引脚的中断请求。

GPIO 控制器设备具有一定数量的 GPIO 引脚。 这些引脚可以物理连接到外围设备。 GPIO 引脚可配置为数据输入、数据输出或中断请求输入。 通常，GPIO 引脚专用于外围设备，而不是由两个或更多设备共享。 GPIO 引脚和外围设备之间的连接是固定的，用户(无法更改，例如，通过移除外围设备并将其替换为另一个设备) 。 因此，可以在平台固件中描述 GPIO 引脚分配给外围设备的方式。

下图显示了 GPIO 控制器驱动程序和 GpioClx。



GPIO 控制器驱动程序和 GpioClx 通过 GpioClx 设备驱动程序接口 (DDI) 相互通信。 GPIO 控制器驱动程序调用由 GpioClx 实现的 驱动程序支持方法 。 GpioClx 调用由 GPIO 控制器驱动程序实现 的事件回调函数 。

GPIO 控制器驱动程序直接访问 GPIO 控制器设备的硬件寄存器。

GpioClx 处理来自物理连接到 GPIO 引脚的外围设备的驱动程序的 I/O 请求。 GpioClx 将这些 I/O 请求转换为简单的硬件操作，通过调用由 GPIO 控制器驱动程序实现的事件回调函数来执行这些操作。 例如，若要从一组 GPIO 引脚读取数据或将数据写入一组，GpioClx 会调用事件回调函数，*例如CLIENT_ReadGpioPins* 和 *CLIENT_WriteGpioPins*。 GpioClx 管理 GPIO 控制器的 I/O 队列，从而解除此任务的 GPIO 控制器驱动程序。

此外，GpioClx 处理来自 GPIO 控制器设备的主要中断，并将这些中断映射到辅助中断，这些中断由外围设备驱动程序处理。 主要中断是由硬件设备生成的中断。 辅助中断由操作系统生成，以响应某些主要中断。 主要中断和辅助中断均由全局系统中断 (GSIs) 标识。 硬件平台的 ACPI 固件将 GSI 分配给主要中断，在运行时，操作系统将 GSIS 分配给次要中断。

例如，固件将 GSI 分配给来自 GPIO 控制器的硬件中断，操作系统将 GSI 分配给配置为中断输入的 GPIO 引脚。

GpioClx 实现一个 ISR，用于处理来自 GPIO 控制器设备的硬件生成的主要中断。 当外围设备断言 GPIO 引脚上的中断，并启用并取消屏蔽此引脚上的中断时，GPIO 控制器会中

断处理器。 作为响应，内核陷阱处理程序计划运行 GpioClx ISR。 为了识别导致中断的 GPIO 引脚，GpioClx ISR 调用由 GPIO 控制器驱动程序实现 *的 CLIENT_QueryActiveInterrupts* 事件回调函数。 然后，GpioClx ISR 查找分配给此引脚的 GSI，并将此 GSI 传递到硬件抽象层 (HAL) 。 HAL 通过调用为此 GSI 注册的 ISR 生成辅助中断。 此 ISR 属于最初断言中断的外围设备的驱动程序。

有关主要中断和辅助中断的详细信息，请参阅 GPIO 中断。

使用 GPIO 引脚的外围设备的驱动程序

启动时，即插即用 (PnP) 管理器枚举 PnP 设备和非 PnP 设备。 对于与 GPIO 引脚建立固定连接的非 PnP 设备，PnP 管理器会查询平台固件，以确定哪些 GPIO 引脚作为系统管理的硬件资源分配给这些设备。

外围设备的 KMDF 驱动程序在 *EvtDevicePrepareHardware* 回调期间接收其分配的硬件资源。 这些资源可能包括配置为数据输出、数据输入或中断请求输入的 GPIO 引脚。

GPIO I/O 资源是 Windows 8 中的新 Windows 资源类型。 此资源包含一组一个或多个 GPIO 引脚，可用作数据输入或数据输出。 如果外围设备驱动程序打开 GPIO I/O 资源进行读取，则驱动程序会将资源中的所有引脚用作数据输入。 如果驱动程序打开 GPIO I/O 资源进行写入，则驱动程序将使用资源中的所有引脚作为数据输出。 有关显示外围设备驱动程序如何打开与一组 GPIO I/O 引脚的逻辑连接的代码示例，请参阅以下主题：

将 KMDF 驱动程序连接到 GPIO I/O 管脚

配置为中断输入的 GPIO 引脚作为普通 Windows 中断资源分配给驱动程序。 中断资源抽象隐藏了这样一个事实，即中断可能由 GPIO 引脚而不是可编程中断控制器（例如）实现。 因此，驱动程序可以像对待任何其他中断资源一样处理基于 GPIO 的中断资源。

若要访问 GPIO I/O 资源中的 GPIO 引脚，外围设备驱动程序必须打开与引脚的逻辑连接。 KMDF 驱动程序调用 **WdfIoTargetOpen** 方法以打开连接。 通过此连接，驱动程序可以将 I/O 请求发送到 GPIO 引脚。 驱动程序发送 **IOCTL_GPIO_READ_PINS** 请求，以从这些引脚读取数据， (如果他们是输入引脚) 或 **IOCTL_GPIO_WRITE_PINS** 请求将数据写入它们 (输出引脚) 。

若要从中断资源中的 GPIO 引脚接收中断，外围设备驱动程序必须注册其中断服务例程 (ISR) ，以便从此引脚实现的中断资源接收中断。 KMDF 驱动程序调用 **WdfInterruptCreate** 方法将 ISR 连接到中断。

GpioClx I/O 和中断接口

2025/07/18

通常，GPIO 控制器的客户端是连接到 GPIO 引脚的外围设备的驱动程序。 这些驱动程序使用 GPIO 引脚作为低带宽数据通道、设备选择输出和中断请求输入。 外围设备驱动程序打开与配置为数据输入或输出的 GPIO 引脚的逻辑连接。 通过这些连接发送 I/O 请求到这些引脚。 此外，外围设备驱动程序可以在逻辑上将其中断服务例程连接到配置为中断请求输入的 GPIO 引脚。

GPIO 引脚是系统管理的硬件资源。 在外围设备驱动程序启动其设备之前，即插即用（PnP）管理器会为此驱动程序分配硬件资源列表。 此硬件资源列表可能包括：

- GPIO I/O 资源。 此资源是一组配置为数据输入或数据输出的一个或多个 GPIO 引脚。 GPIO I/O 资源是一种新的 Windows 资源类型，从 Windows 8 开始。
- 中断。 此中断资源可以实现为配置为中断输入的 GPIO 引脚，但可以通过可编程中断控制器或处理器包上的专用中断引脚来实现此中断资源。 硬件抽象层 （HAL） 中断抽象隐藏这些实现详细信息，客户端驱动程序可以安全地忽略这些细节。

在外围设备驱动程序可以使用一组 GPIO 引脚作为数据输入或输出之前，驱动程序必须打开与这些引脚的逻辑连接。 例如， 内核模式驱动程序接口（KMDF）驱动程序获取 WDFIOTARGET 句柄以标识连接。 驱动程序使用此句柄将 I/O 请求发送到引脚。 具体而言，客户端驱动程序发送 **IOCTL_GPIO_WRITE_PINS** 和 **IOCTL_GPIO_READ_PINS** I/O 控制请求，以将数据写入输出引脚并从输入引脚读取数据。 有关演示如何连接到一组 GPIO I/O 引脚的代码示例，请参阅以下主题：

将 KMDF 驱动程序连接到 GPIO I/O 引脚

若要使用中断资源来接收中断，外围设备驱动程序必须以逻辑方式将中断服务例程（ISR）连接到中断。 例如，内核模式驱动程序可以通过调用 **WdfInterruptCreate** 方法或 **IoConnectInterruptEx** 例程来建立此连接。 连接后，当外围设备向 GPIO 引脚或中断控制器输入发出中断请求时，驱动程序的 ISR 将运行。 有关中断的详细信息，请参阅 创建中断对象。

GPIO 框架扩展（GpioClx）管理作为其客户端的外围设备驱动程序的 I/O 连接和中断连接。 PnP 管理器可以将 GPIO 控制器设备上的不同 GPIO 引脚组分配给不同的客户端驱动程序。 其中一些引脚配置为数据输入或输出，有些引脚配置为中断请求输入。

当客户端驱动程序收到中断请求或将 I/O 请求发送到 GPIO 引脚时，GpioClx 将调用 GPIO 控制器驱动程序实现的事件回调函数。 这些回调访问 GPIO 控制器设备中的硬件寄存器。 通过这些函数调用，GpioClx 读取数据输入、写入数据输出以及管理中断请求（通过查询、启用、屏蔽、清除等方式，将 GPIO 引脚配置为中断输入）。

GpioClx 执行管理 I/O 和中断客户端打开的连接所需的所有处理。 GPIO 控制器驱动程序将这些连接的管理委托给 GpioClx，仅负责访问 GPIO 控制器设备中硬件寄存器这一相对简单的任务。 GPIO 控制器驱动程序不需要知道为其进行特定访问的客户端驱动程序。

GPIO-Based 硬件资源

2025/07/18

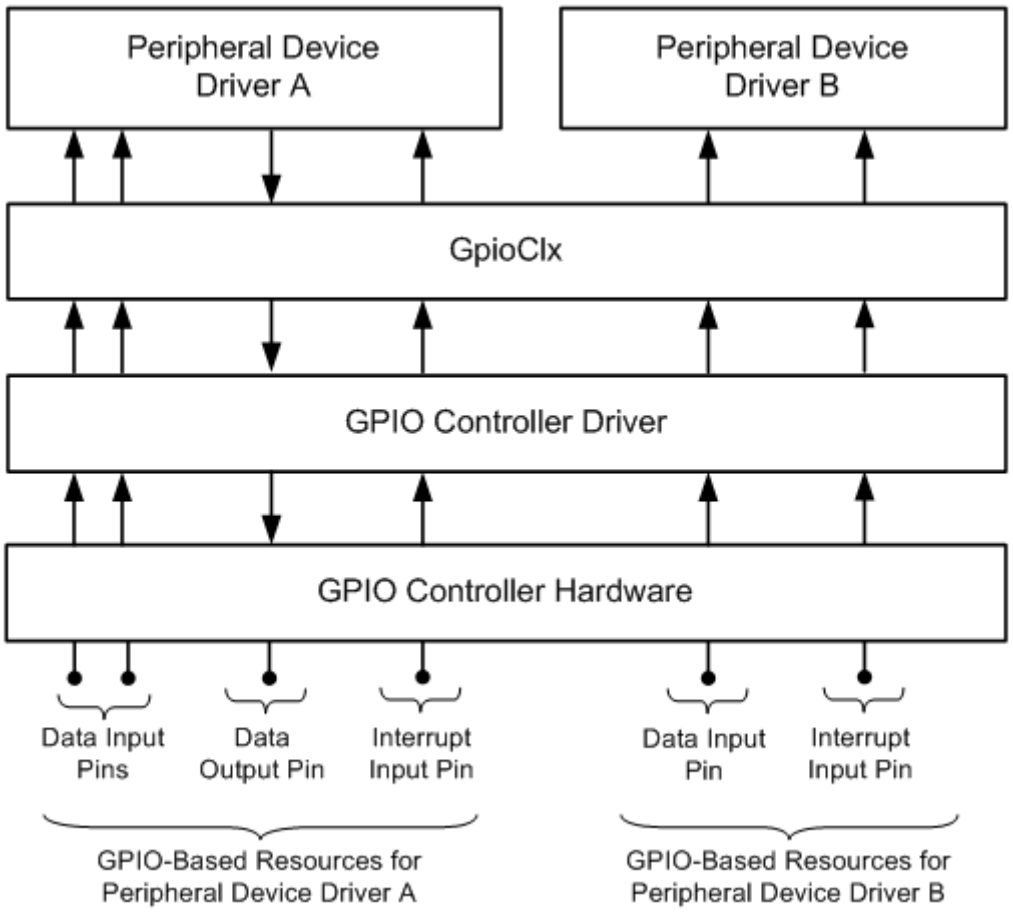
从 Windows 8 开始，由 GPIO 控制器驱动程序控制的常规用途 I/O（GPIO）引脚作为系统管理的硬件资源可供其他驱动程序使用。 GPIO I/O 引脚（配置为数据输入或数据输出的引脚）可用作新的 Windows 资源类型 、*GPIO I/O 资源*。 此外，GPIO 中断引脚（配置为中断请求输入的引脚）可用作普通 Windows 中断资源。

GPIO I/O 资源表示外围设备的驱动程序可以读取或写入的一组或多台 GPIO 引脚。 Windows 隐藏了有关 GPIO I/O 引脚底层实现的详细信息，以便可以编写外围设备驱动程序来操作抽象的 GPIO I/O 资源。 无论实现资源的 GPIO 控制器硬件如何，使用这些抽象资源的外围设备驱动程序都可以跨平台工作。 GPIO I/O 资源由 WDFIOTARGET 句柄表示，该句柄将此资源与拥有基础 GPIO 引脚或引脚的特定 GPIO 控制器驱动程序相关联。

通常，可以针对输入或输出配置 GPIO 控制器上的 I/O 引脚，具体取决于控制器硬件和物理连接到引脚的设备的功能。 因此，驱动程序可以打开与此引脚的逻辑连接，以进行写入或读取操作，但不能同时进行两者。 但是，此约束是由硬件施加的，而不是由 GPIO 框架扩展（GpioClx）施加的。 如果硬件允许为输入和输出配置 I/O 引脚，GpioClx 允许驱动程序为读写作打开与引脚的逻辑连接。

对于配置为中断请求输入的 GPIO 引脚，GPIO 引脚而不是中断控制器或专用中断请求线路实现中断请求的事实被操作系统完全隐藏。 GPIO 中断作为抽象中断资源提供给外围设备驱动程序。 GPIO 驱动程序堆栈和硬件抽象层（HAL）支持这些资源的抽象。 因此，使用中断资源的外围设备驱动程序在很大程度上可以忽略有关这些资源的基础实现的详细信息。 有关详细信息，请参阅 GPIO 中断。

下图显示了将基于 GPIO 的资源分配给两个外围设备驱动程序的示例：



在上图中，为以下三个基于 GPIO 的资源分配了外围设备驱动程序 A：

- 两个数据输入引脚
- 数据输出引脚
- 中断输入引脚

以下两个基于 GPIO 的资源分配给外围设备驱动程序 B：

- 数据输入引脚
- 中断输入引脚

驱动程序 A 和 B 在其 *EvtDevicePrepareHardware* 回调函数中接收分配的资源。 如果驱动程序从资源中获得一组或多个 GPIO I/O 引脚，该驱动程序可以打开一个连接来访问这些引脚。 驱动程序获取 WDFIOTARGET 句柄来标识连接，并将 I/O 请求发送到此句柄以读取或写入这些引脚。

有关演示如何连接到一组 GPIO I/O 引脚并将 I/O 请求发送到此引脚的代码示例，请参阅以下主题：

将 KMDF 驱动程序连接到 GPIO I/O 引脚

在这两个主题中， `IoRoutine` 代码示例中的函数将打开一个 GPIO I/O 引脚资源，用于读取或写入，具体取决于 `ReadOperation` 参数值。 如果为读取打开资源（`DesiredAccess` = GENERIC_READ），则资源中的引脚配置为输入，发送到引脚资源的 IOCTL_GPIO_READ_PINS

请求将读取这些引脚上的输入值。 GpioClx 不允许对一组输入引脚发送 IOCTL_GPIO_WRITE_PINS 请求，并对此类请求以 STATUS_GPIO_OPERATION_DENIED 错误状态完成操作。 同样，如果为写入打开引脚资源（`DesiredAccess` = GENERIC_WRITE），则资源中的引脚配置为输出，并且发送到引脚资源的 **IOCTL_GPIO_WRITE_PINS** 请求将设置驱动这些引脚的输出闩锁中的值。 通常，向一组输出引脚发送 **IOCTL_GPIO_READ_PINS** 请求只是读取写入到输出闩锁的最后一个值。

若要使用中断资源来接收中断，客户端驱动程序必须将中断服务例程（ISR）连接到中断。 通常，驱动程序通过调用 **WdfInterruptCreate** 方法（或 **IoConnectInterruptEx** 例程）来建立此连接。 有关 KMDF 中断的详细信息，请参阅 “创建中断对象”。

与可以动态连接和断开硬件平台的即插即用设备不同，GPIO 控制器设备是永久附加的。 此外，GPIO 引脚与外围设备之间的连接假定为永久连接。 （或者，如果外围设备可以从槽中解拔，则该槽专用于此设备。因此，可用的 GPIO 资源是固定的，可以在平台固件中指定。 同样，假定使用 GPIO 资源的外围设备驱动程序使用专用的 GPIO 资源集。 因此，可以在平台固件中指定这些设备驱动程序的资源要求。

当平台固件将一组 GPIO 引脚指定为 GPIO I/O 资源时，固件指示是否可以为读取、写入或读取和写入打开此资源中的引脚。

如果外围设备驱动程序使用多个 GPIO I/O 资源，则此驱动程序必须知道 PnP 管理器枚举这些资源的顺序。 例如，如果驱动程序使用两个 GPIO I/O 引脚，但这些引脚必须独立访问，并且必须单独访问，则平台固件应将每个引脚描述为单独的 GPIO I/O 资源。 PnP 管理器按平台固件中描述的顺序枚举这些资源，这些资源必须与驱动程序预期的顺序匹配。

外围设备驱动程序打开与 GPIO I/O 资源的连接后， **IOCTL_GPIO_READ_PINS** 或 **IOCTL_GPIO_WRITE_PINS** 请求此驱动程序发送到此连接会访问资源中的所有引脚。 如果驱动程序有时只能访问这些引脚的子集，则必须将此子集作为单独的资源分配给驱动程序。

有关 **IOCTL_GPIO_READ_PINS** 请求的详细信息，包括将数据输入引脚映射到请求输出缓冲区中的位，请参阅 **IOCTL_GPIO_READ_PINS**。 有关 **IOCTL_GPIO_WRITE_PINS** 请求的详细信息，包括请求输入缓冲区中位到数据输出引脚的映射，请参阅 **IOCTL_GPIO_WRITE_PINS**。

将 KMDF 驱动程序连接到 GPIO I/O 管脚

项目 • 2025/03/26

GPIO I/O 资源是一组配置为数据输入或数据输出的一个或多个 GPIO 引脚。 物理连接到这些引脚的外围设备的驱动程序从操作系统获取相应的 GPIO I/O 资源。 外围设备驱动程序会打开与此资源中的 GPIO 引脚的连接，并将 I/O 请求发送到表示此连接的句柄。

下面的代码示例演示了外围设备的内核模式驱动程序框架 (KMDF) 驱动程序如何获取即插即用 (PnP) 管理器分配给驱动程序的 GPIO I/O 资源的说明。

C++

```
NTSTATUS
  EvtDevicePrepareHardware(
    _In_ WDFDEVICE Device,
    _In_ WDFCMRESLIST ResourcesRaw,
    _In_ WDFCMRESLIST ResourcesTranslated
    )
{
    int ResourceCount, Index;
    PCM_PARTIAL_RESOURCE_DESCRIPTOR Descriptor;
    XYZ_DEVICE_CONTEXT *DeviceExtension;

    ...

    DeviceExtension = XyzDrvGetDeviceExtension(Device);
    ResourceCount = WdfCmResourceListGetCount(ResourcesTranslated);
    for (Index = 0; Index < ResourceCount; Index += 1) {
        Descriptor = WdfCmResourceListGetDescriptor(ResourcesTranslated,
Index);
        switch (Descriptor->Type) {

        //
        // GPIO I/O descriptors
        //

        case CmResourceTypeConnection:

            //
            // Check against expected connection type.
            //

            if ((Descriptor->u.Connection.Class ==
CM_RESOURCE_CONNECTION_CLASS_GPIO) &&
                (Descriptor->u.Connection.Type ==
CM_RESOURCE_CONNECTION_TYPE_GPIO_IO)) {

                DeviceExtension->ConnectionId.LowPart = Descriptor-
>u.Connection.IdLowPart;
                DeviceExtension->ConnectionId.HighPart = Descriptor-
```

```
>u.Connection.IdHighPart;

        ...

}
```

在前面的代码示例中 `DeviceExtension` ， 变量是指向外围设备的设备上下文的指针。 用于 `XyzDrvGetDeviceExtension` 检索此设备上下文的函数由外围设备驱动程序实现。 此驱动程序以前通过调用 **WdfDeviceInitSetPnpPowerEventCallbacks** 方法注册了其 *EvtDevicePrepareHardware* 回调函数。

下面的代码示例演示了外围设备驱动程序如何使用它在上一代码示例中获取的 GPIO 资源说明打开驱动程序 GPIO I/O 资源的 WDFIOTARGET 句柄。

C++

```
NTSTATUS IoRoutine(WDFDEVICE Device, BOOLEAN ReadOperation)
{
    WDFIOTARGET IoTarget;
    XYZ_DEVICE_CONTEXT *DeviceExtension;
    UNICODE_STRING ReadString;
    WCHAR ReadStringBuffer[100];
    BOOL DesiredAccess;
    NTSTATUS Status;
    WDF_OBJECT_ATTRIBUTES ObjectAttributes;
    WDF_IO_TARGET_OPEN_PARAMS OpenParams

    DeviceExtension = XyzDrvGetDeviceExtension(Device);
    RtlInitEmptyUnicodeString(&ReadString,
                              ReadStringBuffer,
                              sizeof(ReadStringBuffer));

    Status = RESOURCE_HUB_CREATE_PATH_FROM_ID(&ReadString,
                                              DeviceExtension-
>ConnectionId.LowPart,
                                              DeviceExtension-
>ConnectionId.HighPart);

    NT_ASSERT(NT_SUCCESS(Status));

    WDF_OBJECT_ATTRIBUTES_INIT(&ObjectAttributes);
    ObjectAttributes.ParentObject = Device;

    Status = WdfIoTargetCreate(Device, &ObjectAttributes, &IoTarget);
    if (!NT_SUCCESS(Status)) {
        goto IoErrorEnd;
    }

    if (ReadOperation != FALSE) {
        DesiredAccess = GENERIC_READ;
    } else {
```

```
        DesiredAccess = GENERIC_WRITE;
    }

    WDF_IO_TARGET_OPEN_PARAMS_INIT_OPEN_BY_NAME(&OpenParams, ReadString,
DesiredAccess);

    Status = WdfIoTargetOpen(IoTarget, &OpenParams);
    if (!NT_SUCCESS(Status)) {
        goto IoErrorEnd;
    }
    ...
```

在前面的代码示例中 `Device` ，变量是外围设备的框架设备对象的 WDFDEVICE 句柄。 **RESOURCE_HUB_CREATE_PATH_FROM_ID** 函数创建一个字符串，其中包含 GPIO I/O 资源的名称。 代码示例使用此字符串按名称打开 GPIO I/O 资源。

外围设备驱动程序获取 GPIO I/O 资源的句柄后，此驱动程序可以发送 I/O 控制请求，以从 GPIO 引脚读取数据或将数据写入数据。 为读取打开 GPIO I/O 资源的驱动程序使用 IOCTL_GPIO_READ_PINS I/O 控制请求从资源中的引脚读取数据。 打开 GPIO I/O 资源进行写入的驱动程序使用 IOCTL_GPIO_WRITE_PINS I/O 控制请求将数据写入资源中的引脚。 下面的代码示例演示如何执行 GPIO 读取或写入操作。

C++

```
    WDF_OBJECT_ATTRIBUTES RequestAttributes;
    WDF_OBJECT_ATTRIBUTES Attributes;
    WDF_REQUEST_SEND_OPTIONS SendOptions;
    WDFREQUEST IoctlRequest;
    WDFIOTARGET IoTarget;
    WDFMEMORY WdfMemory;
    NTSTATUS Status;

    WDF_OBJECT_ATTRIBUTES_INIT(&RequestAttributes);
    Status = WdfRequestCreate(&RequestAttributes, IoTarget, &IoctlRequest);
    if (!NT_SUCCESS(Status)) {
        goto RwErrorExit;
    }

    //
    // Set up a WDF memory object for the IOCTL request.
    //

    WDF_OBJECT_ATTRIBUTES_INIT(&Attributes);
    Attributes.ParentObject = IoctlRequest;
    Status = WdfMemoryCreatePreallocated(&Attributes, Data, Size,
&WdfMemory);
    if (!NT_SUCCESS(Status)) {
        goto RwErrorExit;
    }

    //
```

```
    // Format the request.
    //

    if (ReadOperation != FALSE) {
        Status = WdfIoTargetFormatRequestForIoctl(IoTarget,
                                                  IoctlRequest,
                                                  IOCTL_GPIO_READ_PINS,
                                                  NULL,
                                                  0,
                                                  WdfMemory,
                                                  0);

    } else {
        Status = WdfIoTargetFormatRequestForIoctl(IoTarget,
                                                  IoctlRequest,
                                                  IOCTL_GPIO_WRITE_PINS,
                                                  WdfMemory,
                                                  0,
                                                  WdfMemory,
                                                  0);
    }

    if (!NT_SUCCESS(Status)) {
        goto RwErrorExit;
    }

    //
    // Send the request synchronously (with a 60-second time-out).
    //

    WDF_REQUEST_SEND_OPTIONS_INIT(&SendOptions,
                                  WDF_REQUEST_SEND_OPTION_SYNCHRONOUS);
    WDF_REQUEST_SEND_OPTIONS_SET_TIMEOUT(&SendOptions,
                                         WDF_REL_TIMEOUT_IN_SEC(60));

    Status = WdfRequestAllocateTimer(IoctlRequest);
    if (!NT_SUCCESS(Status)) {
        goto RwErrorExit;
    }

    if (!WdfRequestSend(IoctlRequest, IoTarget, &SendOptions)) {
        Status = WdfRequestGetStatus(IoctlRequest);
    }

    ...
```

在前面的代码示例中， `Data` 是指向数据缓冲区的指针， `Size` 是此数据缓冲区的大小（以字节为单位），并 `ReadOperation` 指示请求的操作是读取 (**TRUE**) 还是写入 (**FALSE**)。

更多信息

有关 IOCTL_GPIO_READ_PINS 请求的详细信息，包括将数据输入引脚映射到请求输出缓冲区中的位，请参阅 IOCTL_GPIO_READ_PINS。 有关 IOCTL_GPIO_WRITE_PINS 请求的详细信息，包括将请求输入缓冲区中的位映射到数据输出引脚，请参阅 IOCTL_GPIO_WRITE_PINS。

有关演示如何编写在内核模式下运行的 GPIO 外围驱动程序的示例驱动程序，请参阅 GitHub 上的 GPIO 示例驱动程序 集合中的 SimDevice 示例驱动程序。

反馈

此页面是否有帮助? 是 否

提供产品反馈 | 在 Microsoft Q&A 获取帮助

GPIO 中断

2025/07/18

某些通用 I/O （GPIO）控制器设备可以将其 GPIO 引脚配置为中断请求输入。 这些中断请求输入由物理连接到 GPIO 引脚的外围设备驱动。 这些 GPIO 控制器的驱动程序可以在单个 GPIO 引脚上启用、禁用、屏蔽、取消屏蔽和清除中断请求。

对 GPIO 中断的支持是可选的。 GPIO 框架扩展（GpioClx）不需要 GPIO 控制器来支持 GPIO 中断。

本部分内容

展开表

主题	DESCRIPTION
主要中断和次要中断	GPIO 中断处理本质上是一个两阶段的过程。 常规用途 I/O （GPIO） 控制器触发的中断，会导致 GPIO 框架扩展 （GpioClx） 的中断服务例程 （ISR） 运行，这种中断被称为*主要中断*。 此 ISR 将中断的 GPIO 引脚映射到全局系统中断（GSI），并将此 GSI 传递到硬件抽象层（HAL）。 HAL 生成*辅助中断*以运行第二个 ISR，该 ISR 在逻辑上通过此 GSI 连接到 GPIO 引脚。 此流程显示在 GPIO 驱动程序支持概述中的关系图中。
GPIO-Based 中断资源	向通用 I/O（GPIO）引脚发送中断的外围设备驱动程序将 GPIO 中断作为抽象的 Windows 中断资源获取。 内核模式驱动程序框架 （KMDF） 驱动程序通过其 *EvtDevicePrepareHardware* 事件回调函数接收这些资源。
Passive-Level ISRs	从 Windows 8 开始，内核模式驱动程序框架（KMDF）和用户模式驱动程序框架（UMDF）驱动程序可以作为选项注册其中断服务例程（ISR）以在被动级别运行。
Interrupt-Related 回调	作为选项，常规用途 I/O （GPIO） 控制器的驱动程序可以为 GPIO 中断提供支持。 为了支持 GPIO 中断，GPIO 控制器驱动程序实现一组回调函数来管理这些中断。 驱动程序在将自身注册为 GPIO 框架扩展客户端（GpioClx）时，在注册数据包中包含指向这些回调函数的指针。
GPIO 控制器驱动程序的中断同步	GPIO 控制器驱动程序可以调用 **GPIO_CLX_AcquireInterruptLock** 和

主题	DESCRIPTION
	GPIO_CLX_ReleaseInterruptLock 方法来获取和释放 GPIO 框架扩展（GpioClx）在内部实现的中断锁。 在 IRQL = PASSIVE_LEVEL 运行的驱动程序代码可以调用这些方法以同步到 GpioClx 中的中断服务例程（ISR）。 GpioClx 为 GPIO 控制器中的每个引脚库指定单独的中断锁。
启用和禁用共享 GPIO 中断	在某些情况下，来自两个或多个外围设备的中断请求行可能会连接到同一物理通用输入输出（GPIO）引脚。 共享中断线的 GPIO 引脚通常被配置为电平触发的中断。
GPIO 中断掩码	配置为中断输入的通用 I/O（GPIO）引脚除了可以启用和禁用外，还可以屏蔽和取消屏蔽。

主要中断和辅助中断

2025/07/18

GPIO 中断处理本质上是一个两阶段的过程。 常规用途 I/O （GPIO） 控制器触发的中断，会导致 GPIO 框架扩展 （GpioClx） 的中断服务例程 （ISR） 运行，这种中断被称为*主要中断*。 此 ISR 将中断的 GPIO 引脚映射到全局系统中断（GSI），并将此 GSI 传递到硬件抽象层（HAL）。 HAL 生成 *辅助中断* 以运行第二个 ISR，该 ISR 在逻辑上通过此 GSI 连接到 GPIO 引脚。 此流程显示在 GPIO 驱动程序支持概述中的关系图中。

GpioClx 实现 ISR 来服务 GPIO 控制器通过配置为中断输入的 GPIO 引脚接收的中断请求。 当外围设备在 GPIO 引脚上触发中断，并且在 GPIO 控制器中启用且取消屏蔽中断时，GPIO 控制器硬件会触发对处理器的中断。 为了响应此中断，GpioClx 中的 ISR 查询 GPIO 控制器，以确定生成中断的 GPIO 引脚，然后确定分配给此引脚的 GSI。 GpioClx ISR 将此 GSI 传递给 HAL，HAL 调用逻辑上连接到 GSI 的 ISR。

通常，第二个 ISR 属于触发该 GPIO 引脚中断的外围设备的驱动程序。 有关外围设备驱动程序如何以逻辑方式将其 ISR 连接到 GPIO 中断引脚的信息，请参阅 GPIO-Based 中断资源。

GPIO-Based 中断资源

2025/07/18

向通用 I/O（GPIO）引脚发送中断的外围设备驱动程序将 GPIO 中断作为抽象的 Windows 中断资源获取。 内核模式驱动程序框架 （KMDF） 驱动程序和 用户模式驱动程序框架 （UMDF） 驱动程序通过其 *EvtDevicePrepareHardware* 事件回调函数接收这些资源。

使用基于 GPIO 的中断资源的外围设备驱动程序可以忽略低级别实现详细信息，例如，中断是由 GPIO 引脚而不是由中断控制器或处理器芯片上的中断引脚生成的。

基于 GPIO 的中断是类型为 **CmResourceTypeInterrupt** 的资源。 此中断的配置参数包含在描述中断资源的**CM_PARTIAL_RESOURCE_DESCRIPTOR**结构的 **u.Interrupt** 成员中。 若要将中断服务例程（ISR）连接到中断，UMDF 或 KMDF 驱动程序会将中断资源的 "原始描述" 和 "翻译描述" 提供给中断创建方法。

外围设备的 KMDF 或 UMDF 驱动程序调用 **WdfInterruptCreate** 方法，将 ISR 与设备的中断关联。 此方法的输入参数之一是指向包含中断配置信息的 **WDF_INTERRUPT_CONFIG** 结构的指针。

如果外围设备驱动程序使用多个 GPIO 中断资源，则此驱动程序必须知道这些资源在原始和已翻译的资源列表中出现的顺序，这些资源作为输入参数提供给 *EvtDevicePrepareHardware* 函数或 **OnPrepareHardware** 方法。 这些列表中的资源按平台固件中描述的顺序显示，这些固件必须与驱动程序预期的顺序匹配。

Passive-Level ISRs

2025/07/18

从 Windows 8 开始，内核模式驱动程序框架（KMDF）和用户模式驱动程序框架（UMDF）驱动程序可以作为选项注册其中断服务例程（ISR）以在被动级别运行。

有关 KMDF 和 UMDF 驱动程序的被动级别 ISR 的详细信息，请参阅以下主题：

- 支持 Passive-Level 中断
- 访问硬件和处理中断

如果外围设备使用常规用途 I/O （GPIO） 引脚将中断请求中继到处理器，Windows 中断抽象可方便地使此设备的驱动程序忽略此引脚所属的 GPIO 控制器的硬件特定详细信息。 当内核陷阱处理程序在响应来自设备的 GPIO 中继中断请求时，此处理程序会根据需要自动清除或屏蔽 GPIO 硬件寄存器中的中断。 此外，内核陷阱处理程序可以直接调用设备的 ISR，或将此 ISR 调度在另一个线程中运行。

通常，GPIO 硬件寄存器是内存映射的，在这种情况下，内核陷阱处理程序可以直接在 DIRQL 上访问它们。 但是，外围设备的硬件寄存器可能不是内存映射的，在这种情况下，外围设备驱动程序必须使用 I/O 请求来访问它们。 如果是这样，外围设备驱动程序的 ISR 必须在 IRQL = PASSIVE_LEVEL运行，以便它可以使用 I/O 请求来静默中断或执行中断的初始服务。 被动级别的 ISR 可以同步发送 I/O 请求，并在必要时阻止请求，直到请求完成。

若要为生成边缘触发的中断请求信号的外围设备支持被动级别的 ISR，内核陷阱处理程序会清除 GPIO 引脚上的挂起中断，然后计划 ISR 在被动级内核线程中运行。

若要为生成级别触发的中断请求信号的外围设备支持被动级别的 ISR，内核陷阱处理程序会屏蔽 GPIO 引脚上的挂起中断，然后计划 ISR 在被动级内核线程中运行。 ISR 负责清除外围设备中的中断请求。 ISR 返回后，内核线程会在 GPIO 引脚上取消屏蔽中断。

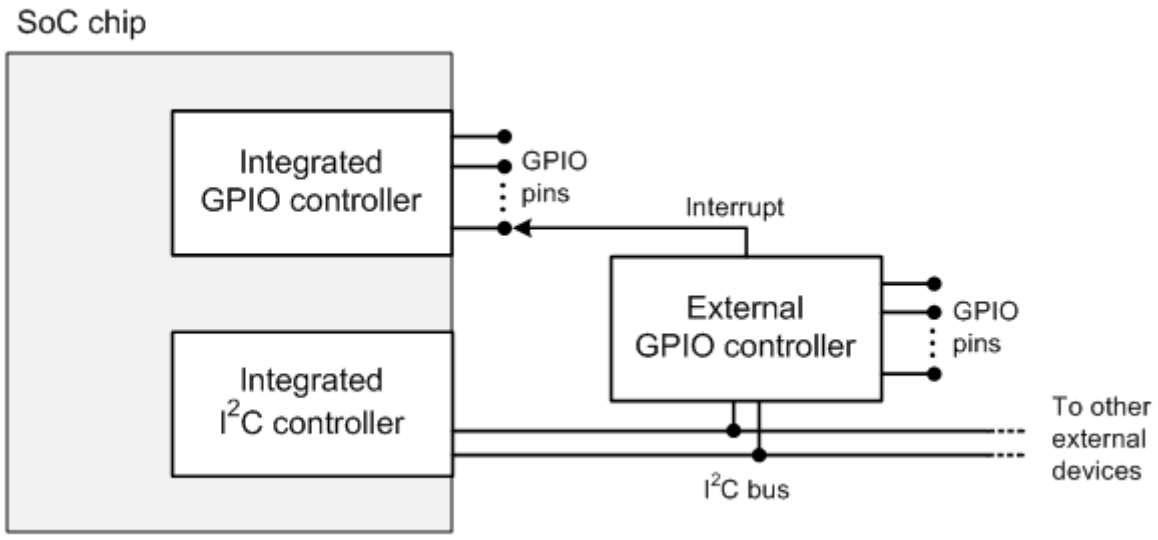
由于中断在 ISR 返回之前保持屏蔽状态，因此设备的被动级别 ISR 应仅执行中断的初始服务，然后返回以避免延迟其他设备的被动级 ISR。 通常，驱动程序应将其他与中断相关的处理推迟到中断工作线程，该线程的优先级低于 ISR。

Interrupt-Related 回调

2025/07/18

作为选项，常规用途 I/O （GPIO） 控制器的驱动程序可以为 GPIO 中断提供支持。 为了支持 GPIO 中断，GPIO 控制器驱动程序实现一组回调函数来管理这些中断。 驱动程序在将自身注册为 GPIO 框架扩展客户端（GpioClx）时，在注册数据包中包含指向这些回调函数的指针。 有关此注册数据包的详细信息，请参阅 **GPIO_CLIENT_REGISTRATION_PACKET**。

通常情况下，作为芯片系统 (SoC) 集成部分的 GPIO 控制器具有内存映射的硬件寄存器，可以被 SoC 芯片中的处理器直接访问。 但是，单独的 GPIO 控制器设备可以通过串行总线在外部连接到 SoC 芯片，如下图所示。



在此图中，外部 GPIO 控制器已连接到 I IoC 总线。 此总线由属于 SoC 芯片的集成部分的 I IoC 总线控制器控制。 来自外部 GPIO 控制器的中断请求行连接到集成 GPIO 控制器上的引脚。 在此示例中， GpioClx DDI 可以同时容纳集成的 GPIO 控制器和外部 GPIO 控制器。

如果 GPIO 控制器设备是内存映射的，GPIO 控制器驱动程序可以直接在 DIRQL 上访问控制器的硬件寄存器。 但是，如果 GPIO 控制器是串行连接的，则 GPIO 控制器驱动程序只能访问硬件注册在 IRQL = PASSIVE_LEVEL，如 Passive-Level ISR 中所述。

具有内存映射硬件寄存器的 GPIO 控制器的驱动程序应在驱动程序提供给 GpioClx 的设备信息中设置 **MemoryMappedController** 标志位。 否则，GpioClx 假定这些硬件寄存器不是内存映射的，并且驱动程序只能在 IRQL = PASSIVE_LEVEL 访问这些寄存器。 有关此标志位的详细信息，请参阅 **CONTROLLER_ATTRIBUTE_FLAGS**。

GpioClx 实现中断服务例程（ISR），以服务来自 GPIO 控制器的中断请求。 此 ISR 调用以下与中断相关的回调函数：

CLIENT_ClearActiveInterruptsCLIENT_MaskInterruptsCLIENT_QueryActiveInterruptsCLIENT_QueryEnabledInterruptsCLIENT_UnmaskInterrupt 这些函数根据 GpioClx 中的 ISR 是在 DIRQL 还是在

PASSIVE_LEVEL 运行，可以在 DIRQL 或 PASSIVE_LEVEL 调用。 如果 **MemoryMappedController** = 1，则 ISR 在 DIRQL 级别调用这些函数；如果 **MemoryMappedController** = 0，则在 PASSIVE_LEVEL 级别调用。 在任一情况下，ISR 都会自动序列化其回调，以便对其中一个函数的调用不会在调用其中另一个函数的中间发生。

GPIO 框架扩展仅在 PASSIVE_LEVEL 调用以下与中断有关的回调函数，而不考虑是否设置了 **MemoryMappedController** 标志：

CLIENT_DisableInterruptCLIENT_EnableInterrupt 如果未设置 **MemoryMappedController** 标志，则会在PASSIVE_LEVEL调用与中断相关的所有回调函数。 GpioClx 自动序列化对这些函数的调用，以便调用其中一个函数不会在调用其中另一个函数时发生。

但是，如果设置了 **MemoryMappedController** 标志， *则CLIENT_EnableInterrupt* 和 *CLIENT_DisableInterrupt* 函数必须将中断启用和禁用作显式同步到 GpioClx ISR，后者在 DIRQL 上调用其他四个与中断相关的回调函数。

通常，其他 *CLIENT*_Xxx 回调函数（其名称不包含"*中断*"）不执行与中断相关的处理，因此不需要同步到 GpioClx ISR。 但是，如果在PASSIVE_LEVEL调用这些函数中的任何一个，并且代码包含访问由DIRQL层中断相关函数所访问的中断设置，则必须将此代码与ISR同步。

为了支持中断同步，GpioClx 实现一组中断锁。 在PASSIVE_LEVEL运行的回调函数可以调用 **GPIO_CLX_AcquireInterruptLock** 方法来获取中断锁，并调用 **GPIO_CLX_ReleaseInterruptLock** 方法来释放锁。 当函数保存中断锁时，GpioClx ISR 无法运行，并且此 ISR 无法调用任何与中断相关的回调函数。 若要使 GPIO 中断能够及时处理，驱动程序应将中断锁保留的时间不超过必要时间。

有关详细信息，请参阅 GPIO 控制器驱动程序的中断同步。

GPIO 控制器驱动程序的中断同步

项目 • 2023/06/15

GPIO 控制器驱动程序可以调用 **GPIO_CLX_AcquireInterruptLock** 和 **GPIO_CLX_ReleaseInterruptLock** 方法来获取和释放 GPIO 框架扩展 (GpioClx) 在内部实现的中断锁。 在 IRQL = PASSIVE_LEVEL 运行的驱动程序代码可以调用这些方法，以同步到 gpioClx 中 (ISR) 中断服务例程。 GpioClx 将单独的中断锁专用于 GPIO 控制器中的每个引脚组。

如果 GPIO 控制器的硬件寄存器是内存映射的，则 GpioClx 中的 ISR 在 DIRQL 上调用某些驱动程序实现的事件回调函数;GpioClx 在PASSIVE_LEVEL调用其余回调函数。 访问寄存器库的被动级回调函数可能需要使用中断锁来同步到在 DIRQL 上运行且访问相同寄存器的回调函数。

例如，被动级别 *CLIENT_EnableInterrupt* 和 *CLIENT_DisableInterrupt* 回调函数修改影响在 DIRQL 上运行的其他与中断相关的回调例程操作的硬件设置。 *CLIENT_EnableInterrupt*和 *CLIENT_DisableInterrupt*函数通常使用库中断锁来同步其寄存器访问。

GpioClx 自动序列化 DIRQL 中发生的与中断相关的回调和 I/O 相关回调。 GpioClx 在 DIRQL 调用回调函数之前获取目标库的中断锁，并在函数返回后释放锁。 在 DIRQL 中调用的回调函数尝试通过调用 GPIO_CLX_AcquireInterruptLock 重新获取银行中断锁是 **错误的。**

同样，GpioClx 会自动序列化PASSIVE_LEVEL发生的回调。 GpioClx 在内部实现每个库的等待锁。 GpioClx 在PASSIVE_LEVEL调用回调函数之前获取目标库的等待锁，并在函数返回时释放该锁。 对于内存映射 GPIO 控制器，GpioClx 代表驱动程序管理银行等待锁，但不允许驱动程序显式获取和释放锁。

但是，对于非内存映射 GPIO 控制器， **GPIO_CLX_AcquireInterruptLock** 和 **GPIO_CLX_ReleaseInterruptLock** 获取并释放等待锁而不是中断锁。 GpioClx 为 GPIO 控制器中的每个引脚组实现单独的等待锁。 由于寄存器不是内存映射的，因此所有与中断相关的和 I/O 相关的回调函数在PASSIVE_LEVEL调用，以便它们可以使用 I/O 请求通过串行总线（如 I^2C）访问寄存器。 GpioClx 在调用其中一个回调函数之前获取目标库的等待锁，并在函数返回后释放锁。

非内存映射控制器的回调函数尝试通过调用 GPIO_CLX_AcquireInterruptLock 重新获取银行等待锁是 **错误的。** 但是，回调函数外部的被动级别驱动程序代码可以调用 **GPIO_CLX_*Xxx*InterruptLock** 方法以同步到回调函数。 由于 GpioClx 在 PASSIVE_LEVEL 调用所有与中断相关的和 I/O 相关的回调函数，因此库等待锁有效地取代了非内存映射控制器的库中断锁。

非内存映射控制器的另一个选项是让控制器驱动程序实现一组等待锁。 与 GpioClx 实现的等待锁相比，这些等待锁可能使回调例程能够对共享资源执行更精细的锁定和解锁。

在调用 *CLIENT_QueryControllerBasicInformation* 回调例程期间，GPIO 控制器驱动程序向 GpioClx 报告控制器寄存器是否为内存映射。 有关详细信息，请参阅 **CLIENT_CONTROLLER_BASIC_INFORMATION** 中的 **MemoryMappedController** 标志的说明。

有关中断锁和等待锁的详细信息，请参阅 使用框架锁。

下表提供了有关在 DIRQL（而不是PASSIVE_LEVEL寄存器是否为内存映射）调用的回调函数的更多详细信息。 表后面的说明说明了被动级别回调函数何时应使用中断锁。

- 与中断相关的回调函数
- 与 I/O 相关的回调函数
- GPIO 初始化和与设置相关的回调函数
- GPIO 电源管理相关的回调函数
- 其他回调函数

与中断相关的回调函数

为了支持配置为中断输入的 GPIO 引脚，GPIO 控制器驱动程序实现了一组事件回调函数，以通过这些引脚管理中断请求。 在下表中，中间列指示如果 GPIO 控制器的硬件寄存器是内存映射的，则调用函数的 IRQL。 最右边的列指示在寄存器未进行内存映射且必须通过串行总线访问的情况下调用函数的 IRQL。

回调函数	**如果内存映射 (MemoryMappedController = 1)，则为 IRQL**	**如果串行访问的 IRQL (MemoryMappedController = 0)**
CLIENT_EnableInterrupt	PASSIVE_LEVEL	PASSIVE_LEVEL
CLIENT_DisableInterrupt	(请参阅注释 1.)	(请参阅注释 2.)
CLIENT_ClearActiveInterrupts	DIRQL	PASSIVE_LEVEL
CLIENT_MaskInterrupts	(请参阅注释 3.)	(请参阅注释 4.)
CLIENT_QueryActiveInterrupts		
CLIENT_QueryEnabledInterrupts		
CLIENT_ReconfigureInterrupt		
CLIENT_UnmaskInterrupt		

回调函数	如果内存映射 (MemoryMappedController = 1)，则为 IRQL	如果串行访问的 IRQL (MemoryMappedController = 0)
CLIENT_PreProcessControllerInterrupt	DIRQL (请参阅注释 5.)	DIRQL (请参阅注释 6.)

备注

1. GpioClx 在调用此回调函数之前不会获取库中断锁。 如有必要，回调函数可以获取库中断锁，以同步与在 DIRQL 上运行的回调函数共享的寄存器的访问。
2. GpioClx 使用PASSIVE_LEVEL调用的其他与中断相关的和 I/O 相关的回调函数序列化对此回调函数的调用。 因此，回调函数不应尝试获取银行等待锁。
3. GpioClx 在调用此回调函数之前获取库中断锁，并在函数返回后释放锁。 因此，回调函数不应尝试获取库中断锁。
4. GpioClx 使用PASSIVE_LEVEL调用的其他与中断相关的和 I/O 相关的回调函数序列化对此回调函数的调用。 因此，回调函数不应尝试获取银行等待锁。
5. GpioClx 在调用此回调函数之前获取库中断锁，并在函数返回后释放锁。 因此，回调函数不应尝试获取库中断锁。
6. GpioClx 在调用此回调函数之前不会获取库中断锁。 GPIO 控制器驱动程序负责提供可能需要的任何同步。

与 I/O 相关的回调函数

为了支持配置为数据 I/O 引脚的 GPIO 引脚，GPIO 控制器驱动程序实现一组事件回调函数，以通过这些引脚管理 I/O 操作。 在下表中，中间列指示如果 GPIO 控制器的硬件寄存器是内存映射的，则调用函数的 IRQL。 最右边的列指示在寄存器未进行内存映射且必须通过串行总线访问的情况下调用函数的 IRQL。

回调函数	如果内存映射 (MemoryMappedController = 1)，则为 IRQL	如果串行访问的 IRQL (MemoryMappedController = 0)
CLIENT_ConnectIoPins *CLIENT_DisconnectIoPins*	PASSIVE_LEVEL (请参阅注释 1.)	PASSIVE_LEVEL (请参阅注释 2.)

回调函数	如果内存映射 (MemoryMappedController = 1)，则为 IRQL	如果串行访问的 IRQL (MemoryMappedController = 0)
CLIENT_ReadGpioPins	DIRQL	PASSIVE_LEVEL
CLIENT_ReadGpioPinsUsingMask	(请参阅注释 3.)	(请参阅注释 4.)
CLIENT_WriteGpioPins		
CLIENT_WriteGpioPinsUsingMask		

备注

1. GpioClx 在调用此回调函数之前不会获取库中断锁。 如有必要，回调函数可以获取中断锁，以同步与在 DIRQL 上运行的回调函数共享的寄存器的访问。
2. GpioClx 通过PASSIVE_LEVEL调用的其他与中断相关的和 I/O 相关的回调函数序列化对此回调函数的调用。 因此，回调函数不应尝试获取银行等待锁。
3. GpioClx 在调用此回调函数之前获取库中断锁，并在函数返回后释放锁。 因此，回调函数不应尝试获取库中断锁。
4. GpioClx 通过PASSIVE_LEVEL调用的其他与中断相关的和 I/O 相关的回调函数序列化对此回调函数的调用。 因此，回调函数不应尝试获取银行等待锁。

GPIO 初始化和与设置相关的回调函数

若要设置 GPIO 控制器以执行 I/O 和中断操作，GPIO 控制器驱动程序实现一组事件回调函数来初始化控制器。 在下表中，中间列指示如果 GPIO 控制器的硬件寄存器是内存映射的，则调用函数的 IRQL。 最右侧的列指示在寄存器未进行内存映射且必须通过串行总线访问时调用函数的 IRQL。

回调函数	如果内存映射的 IRQL (MemoryMappedController = 1)	IRQL（如果以串行方式访问） (MemoryMappedController = 0)

回调函数	如果内存映射的 IRQL (MemoryMappedController = 1)	IRQL（如果以串行方式访问) (MemoryMappedController = 0)
CLIENT_PrepareController *CLIENT_ReleaseController* *CLIENT_StartController* *CLIENT_StopController* *CLIENT_QueryControllerBasicInformation* *CLIENT_QuerySetControllerInformation*	PASSIVE_LEVEL (请参阅注释 1.)	PASSIVE_LEVEL (请参阅注释 2.)

备注

1. 当 GpioClx 调用任何这些回调函数时，库中断锁不可用。 因此，这些回调函数不应尝试获取库中断锁。
2. 调用这些回调函数时，GpioClx 库等待锁不可用。 因此，驱动程序不应尝试获取银行等待锁以同步到这些回调函数。

GPIO 电源管理相关的回调函数

为了使 GPIO 控制器能够更改设备电源状态，GPIO 控制器驱动程序实现了一组事件回调函数，以便在这些更改期间保存和还原硬件设置。 在下表中，中间列指示如果 GPIO 控制器的硬件寄存器是内存映射的，则调用函数的 IRQL。 最右侧的列指示在寄存器未进行内存映射且必须通过串行总线访问时调用函数的 IRQL。

回调函数	如果内存映射的 IRQL (MemoryMappedController = 1)	IRQL（如果以串行方式访问) (MemoryMappedController = 0)
CLIENT_RestoreBankHardwareContext *CLIENT_SaveBankHardwareContext*	DIRQL 或 HIGH_LEVEL (请参阅 Notes.)	不支持。

备注

- 对于常规 F 状态转换：保存/还原回调函数是使用 GPioClx 在 DIRQL 中持有的库中断锁调用的。 因此，这两个回调函数都不应尝试获取库中断锁。

- 对于关键的 F 状态转换：调用电源引擎插件 (PEP) 来保存和还原 GPIO 状态时，将调用保存/还原回调。 保存/还原回调函数在HIGH_LEVEL最后一个处理器处于空闲状态的上下文中调用，这发生在平台深度空闲转换序列的后期。 因此，这两个回调函数都不应尝试获取库中断锁。

有关 F 状态的详细信息，请参阅 组件级电源管理。 有关 PEP 的详细信息，请参阅 **PoFxPowerControl**。

其他回调函数

若要使 GPIO 控制器支持特定于控制器的操作，GPIO 控制器驱动程序实现 *CLIENT_ControllerSpecificFunction* 事件回调函数。 在下表中，中间列指示如果 GPIO 控制器的硬件寄存器是内存映射的，则调用函数的 IRQL。 最右侧的列指示在寄存器未进行内存映射且必须通过串行总线访问时调用函数的 IRQL。

回调函数	**如果内存映射的 IRQL (MemoryMappedController = 1)**	**IRQL（如果以串行方式访问) (MemoryMappedController = 0)**
CLIENT_ControllerSpecificFunction	PASSIVE_LEVEL (请参阅注释 1.)	PASSIVE_LEVEL (请参阅注释 2.)

备注

1. 在调用此回调函数之前，GpioClx 不会获取库中断锁。 如有必要，回调函数可以获取库中断锁，以同步与在 DIRQL 运行的回调函数共享的寄存器的访问。

2. GpioClx 通过PASSIVE_LEVEL调用的其他与中断相关的和 I/O 相关的回调函数序列化对此回调函数的调用。 因此，回调函数不应尝试获取银行等待锁。

启用和禁用共享 GPIO 中断

项目 • 2023/06/15

在某些情况下，来自两个或多个外围设备的中断请求线路可能会连接到同一物理通用 I/O (GPIO) 引脚。 共享中断线的 GPIO 引脚通常配置为级别触发的中断。

如果这些设备的驱动程序注册其中断服务例程 (ISR) 在此 GPIO 引脚上断言中断时触发，则仅当第一个驱动程序注册此中断时，GPIO 框架扩展 (GpioClx) 调用 *CLIENT_EnableInterrupt* 回调函数。 当其他驱动程序注册为使用已启用的 GPIO 中断时，GpioClx 会在内部跟踪这些注册，但不会冗余地调用 *CLIENT_EnableInterrupt* 回调函数来启用此中断。 同样，仅当最后一个注册的驱动程序释放中断时，GpioClx 才会调用 *CLIENT_DisableInterrupt* 回调函数。

GPIO 中断掩码

2025/07/18

配置为中断输入的通用 I/O（GPIO）引脚除了可以启用和禁用外，还可以屏蔽和取消屏蔽。

如果外围设备的级别触发中断已启用且处于活动状态，但内核陷阱处理程序无法立即运行设备的中断服务例程（ISR），以清除中断，处理程序会屏蔽 GPIO 引脚上的中断，以防止该引脚重复导致更多中断。 稍后，在 ISR 完成运行并清除中断之后，可以安全地取消屏蔽中断。

屏蔽中断不会清除或禁用中断。 如果启用了 GPIO 中断，并且该中断处于活动状态且已屏蔽，那么取消屏蔽此中断会导致 GPIO 控制器设备向处理器发出中断请求。

GPIO 中断掩码位在禁用 GPIO 中断时不起作用。 *CLIENT_EnableInterrupt*回调函数将中断的掩码位设置为零;也就是说，中断最初在启用后取消屏蔽。

掩码和禁用 GPIO 中断引脚之间的一个重要区别在于，掩码会保留引脚的中断配置设置，而禁用引脚不会。 当 GPIO 中断引脚被屏蔽时，它将保留其以前编程的中断模式（边沿触发或电平触发）、极性（高电平有效、低电平有效或双边有效）和消抖设置。 这些设置在中断解除屏蔽后立即再次生效。 但是，当中断被禁用时，所有引脚的中断配置设置都会丢失。 启用引脚后，必须按照所需的中断配置重新对其进行编程。

某些 GPIO 控制器在硬件中实现中断掩码寄存器，这些寄存器与中断使能寄存器是独立且不同的。

然而，其他 GPIO 控制器提供一组硬件寄存器，用于合并中断掩码和中断启用功能。 这些控制器的驱动程序模拟软件中的单独的中断掩码和中断启用寄存器。 为此，这些驱动程序跟踪中断启用位和中断掩码位的逻辑状态，并操作硬件寄存器中的相应位，以准确反映每个 GPIO 中断的组合逻辑中断启用位和中断掩码位的行为。

GpioClx DDI

项目 • 2025/04/16

常规用途 I/O (GPIO) 控制器驱动程序通过 GpioClx 设备驱动程序接口 (DDI) 与 GPIO 框架扩展 (GpioClx) 通信。 此 DDI 在 Gpioclx.h 头文件中定义，并在常规用途 I/O (GPIO) 中介绍。 作为此 DDI 的一部分，GpioClx 实现多个驱动程序支持方法，这些方法由 GPIO 控制器驱动程序调用。 此驱动程序实现了一组事件回调函数，这些函数由 GpioClx 调用。 GpioClx 使用这些回调来管理已配置为中断输入的 GPIO 引脚提供的中断请求，并将数据传输到已配置为数据输入和输出的 GPIO 引脚，或者从其传输出来。

本部分内容

展开表

主题	DESCRIPTION
GpioClx DDI 中的驱动程序支持方法	从 Windows 8 开始，提供 GPIO 框架扩展 (GpioClx)。 GpioClx-DDI 中系统提供的方法在 GpioClx 内核模式驱动程序 Msgpioclx.sys 中实现。 此驱动程序导出 GpioClx 驱动程序支持方法的入口点。 从 Windows 8 开始，Msgpioclx.sys 是操作系统的标准组件。
可选和必需的 GPIO 回调函数	常规用途 I/O (GPIO) 控制器驱动程序调用 GPIO_CLX_RegisterClient 方法注册为 GPIO 框架扩展 (GpioClx) 的客户端。 在此调用期间，驱动程序将一个注册包传递给 GpioClx，该包指定了由驱动程序实现的事件回调函数列表。 GpioClx 调用这些回调函数来配置 GPIO 控制器硬件、执行 I/O 操作和管理中断。 GpioClx 需要 GPIO 控制器驱动程序来实现某些回调函数，但支持其他回调函数是可选的。
GPIO 设备上下文	常规用途 I/O (GPIO) 控制器设备由框架设备对象表示。 GPIO 控制器驱动程序可以将设备上下文与此设备对象相关联。 驱动程序使用此设备上下文来持久存储有关 GPIO 控制器设备状态的信息。
将 GPIO 控制器分区为管脚库	驱动程序开发人员可以选择将常规用途 I/O (GPIO) 控制器设备划分为两组或更多组 GPIO 引脚。 例如，具有 64 个 GPIO 引脚的 GPIO 控制器设备可以被 GPIO 控制器驱动器描述为两个组，每个组有 32 个 GPIO 引脚。
GPIO 控制器驱动程序的实现问题	GPIO 框架扩展 (GpioClx) 提供了一个灵活的设备驱动程序接口 (DDI)。 此 DDI 使开发人员能够在备选回调接口中进行选择。 驱动程序开发人员应实现最

主题	DESCRIPTION
	适合目标 GPIO 控制器设备硬件体系结构的事件回调函数集。

GpioClx DDI 中的驱动程序支持方法

2025/07/18

从 Windows 8 开始，提供 GPIO 框架扩展 (GpioClx)。 GpioClx-DDI 中系统提供的方法在 GpioClx 内核模式驱动程序 Msgpioclx.sys 中实现。 此驱动程序导出 GpioClx 驱动程序支持方法的入口点。 从 Windows 8 开始，Msgpioclx.sys 是操作系统的标准组件。

在生成时，GPIO 控制器驱动程序静态链接到 GpioClx 存根库中的 DDI 入口点 Msgpioclxstub.lib。 在运行时，此库执行必要的驱动程序版本协商，以动态链接到 Msgpioclx.sys中的相应入口点。

需要特定版本的 Msgpioclx.sys 的 GPIO 控制器驱动程序可以安全地链接到版本号较高的 Msgpioclx.sys 版本。 但是，此驱动程序无法链接到版本号较低的 Msgpioclx.sys 版本。

驱动程序注册

若要注册为 GpioClx 的客户端，GPIO 控制器驱动程序调用 **GPIO_CLX_RegisterClient** 方法。 通常，驱动程序从其 **DriverEntry** 例程调用此方法。 在此调用期间，驱动程序会将注册数据包传递给该方法。 此数据包包含指向一组驱动程序实现的事件回调函数的指针。 这些函数访问 GPIO 控制器设备中的硬件寄存器。 GpioClx 调用这些函数来处理 I/O 请求并管理中断。

GPIO 控制器驱动程序调用 **GPIO_CLX_UnregisterClient** 方法以取消其与 GpioClx 的注册。 通常，驱动程序从其 *EvtDriverUnload* 事件回调函数调用此方法。

设备对象初始化

若要初始化 GpioClx，GPIO 控制器驱动程序必须从其 *EvtDriverDeviceAdd* 回调函数调用两个 GpioClx 方法。 第一个方法 （**GPIO_CLX_ProcessAddDevicePreDeviceCreate**）必须在调用 **WdfDeviceCreate** 方法之前调用，该方法将创建设备对象。 第二种方法 （**GPIO_CLX_ProcessAddDevicePostDeviceCreate**）必须在 **WdfDeviceCreate** 调用后调用。

中断锁

大多数驱动程序实现的事件回调函数仅由 GpioClx 在 IRQL = PASSIVE_LEVEL 调用。 但是，以下列表中的回调函数在 PASSIVE_LEVEL 或 DIRQL 上调用，具体取决于 *CLIENT_QueryControllerBasicInformation* 回调函数提供给 GpioClx 的设备信息：

- *CLIENT_ClearActiveInterrupts*
- *CLIENT_MaskInterrupts*
- *CLIENT_QueryActiveInterrupts*

- *CLIENT_QueryEnabledInterrupts*
- *CLIENT_UnmaskInterrupt*

这些函数是从 GpioClx 中的中断服务例程（ISR）调用的，该例程在 DIRQL 或PASSIVE_LEVEL运行，具体取决于 GPIO 控制器的硬件寄存器是内存映射的。

*CLIENT_QueryControllerBasicInformation*函数以**CLIENT_CONTROLLER_BASIC_INFORMATION**结构的形式提供设备信息。 如果在此结构的 **Flags** 成员中设置了 **MemoryMappedController** 标志位，GpioClx ISR 会在 DIRQL 上调用上述列表中的回调函数。 否则，ISR 会在 PASSIVE_LEVEL 调用所有由驱动程序实现的回调函数。 有关此标志位的详细信息，请参阅 Interrupt-Related 回调。

GpioClx 自动同步对驱动程序实现的回调函数的调用，这些函数在PASSIVE_LEVEL运行，并且不会从 GpioClx ISR 调用。 因此，一次只能运行其中一个函数。 但是，GpioClx 不会自动将这些PASSIVE_LEVEL回调与 GpioClx 从其 ISR 发出的回调同步。 如果需要，GPIO 控制器驱动程序必须显式提供此类同步。

为了避免潜在的同步错误，GpioClx 实现 GPIO 控制器驱动程序可以获取和释放的 *中断锁* 。 中断锁主要用于驱动程序 *的CLIENT_EnableInterrupt* 和 *CLIENT_DisableInterrupt* 回调函数。 驱动程序调用 **GPIO_CLX_AcquireInterruptLock** 方法以获取锁，并调用 **GPIO_CLX_ReleaseInterruptLock** 方法来释放锁。 驱动程序从在 PASSIVE_LEVEL 调用的回调函数中调用这些方法，并且不会从 GpioClx 的 ISR 中调用。 当驱动程序持有锁时，GpioClx ISR 无法运行。 驱动程序应仅短暂且仅在必须与 ISR 同步的关键作期间保留锁。

如果 GpioClx ISR 调用驱动程序实现的回调函数，则此函数不需要获取中断锁（或释放），因为 ISR 已持有锁（并将释放它）。 此函数对 **GPIO_CLX_AcquireInterruptLock** 和 **GPIO_CLX_ReleaseInterruptLock** 方法的调用不起作用，但不会被视为错误。

可选和必需的 GPIO 回调函数

2025/07/18

常规用途 I/O (GPIO) 控制器驱动程序调用 **GPIO_CLX_RegisterClient** 方法注册为 GPIO 框架扩展 (GpioClx) 的客户端。 在此调用期间，驱动程序将一个注册包传递给 GpioClx，该包指定了由驱动程序实现的事件回调函数列表。 GpioClx 调用这些回调函数来配置 GPIO 控制器硬件、执行 I/O 操作和管理中断。 GpioClx 需要 GPIO 控制器驱动程序来实现某些回调函数，但支持其他回调函数是可选的。

注册数据包是一种 **GPIO_CLIENT_REGISTRATION_PACKET** 结构。 如果 GPIO 控制器驱动程序实现特定的回调函数，它将函数指针写入到该回调函数的相应成员中。 或者，若要指示不支持特定的回调函数，驱动程序会将 NULL 写入相应的成员。

必须在注册数据包中包含以下回调函数：

CLIENT_PrepareControllerCLIENT_QueryControllerBasicInformationCLIENT_StartControllerCLIENT_StopControllerCLIENT_ReleaseController 如果上面的列表中缺少任何回调函数（即注册数据包中的相应函数指针为 NULL），则**GPIO_CLX_RegisterClient**方法将会失败。

GPIO 控制器驱动程序不需要支持从 GPIO I/O 引脚读取或写入，这些引脚是配置为数据输入或数据输出的引脚。 （没有 I/O 引脚的 GPIO 控制器仍可以中继来自外围设备的中断请求。但是，如果注册数据包包括以下与 I/O 相关的回调函数之一，则数据包必须包括以下两个回调函数：

CLIENT_ConnectIoPinsCLIENT_DisconnectIoPins 此外，如果注册数据包包含上述列表中的两个回调函数，驱动程序必须额外支持从 GPIO I/O 引脚读取、写入 GPIO I/O 引脚或两者。 具体而言，注册数据包必须在以下列表中至少包含一个回调函数：

*CLIENT_ReadGpioPins*或*CLIENT_ReadGpioPinsUsingMask*、*CLIENT_WriteGpioPins*或*CLIENT_WriteGpioPinsUsingMask*。支持读取的驱动程序必须实现上述列表中的 *CLIENT_ReadGpioPins* Xxx 回调函数之一。 支持写入的驱动程序必须实现上述列表中的两个 *CLIENT_WriteGpioPins*Xxx 回调函数之一。

实现*CLIENT_ReadGpioPinsUsingMask*、*CLIENT_WriteGpioPinsUsingMask*或两者兼有的驱动程序必须在*CLIENT_QueryControllerBasicInformation*回调函数提供的设备信息中设置 **FormatIoRequestsAsMasks** 标志位。 实现 *CLIENT_ReadGpioPins*、 *CLIENT_WriteGpioPins*或两者都的驱动程序不得设置此标志位。 有关详细信息，请参阅 **CLIENT_CONTROLLER_BASIC_INFORMATION** 中的 **Flags** 成员的说明。

支持 GPIO 中断不需要 GPIO 控制器驱动程序。 但是，如果注册数据包包含以下任何与中断相关的回调函数，则数据包必须包括以下所有回调函数：

CLIENT_EnableInterruptCLIENT_DisableInterruptCLIENT_MaskInterruptsCLIENT_QueryActiveInterruptsCLIENT_UnmaskInterrupt 一个支持中断掩码的驱动程序必须实现 *CLIENT_MaskInterrupts* 回调函数。 支持查询活动中断的驱动程序必须实现 *CLIENT_QueryActiveInterrupts* 回调函数。

*CLIENT_ClearActiveInterrupts*回调函数是一种特殊情况。 如果 GPIO 控制器硬件在读取时自动清除活动中断，则不需要 *CLIENT_ClearActiveInterrupts* 函数，注册数据包中的相应函数指针应设置为 NULL。 但是，如果在读取活动中断时，它们不会被自动清除，并且如果注册数据包中提供了上述列表中的中断相关回调函数，则必须在数据包中包含 *CLIENT_ClearActiveInterrupts* 函数。 若要指示硬件在读取时自动清除活动中断，驱动程序会在由 *CLIENT_QueryControllerBasicInformation*回调函数提供的设备信息中设置 **ActiveInterruptsAutoClearOnRead**标志位。 有关详细信息，请参阅 **CLIENT_CONTROLLER_BASIC_INFORMATION** 中的 **Flags** 成员的说明。

如果 GPIO 控制器驱动程序支持 GPIO 中断，则注册数据包可以选择包括以下回调函数：

CLIENT_QueryEnabledInterrupts GpioClx 支持从 Windows 8.1 开始 *CLIENT_QueryEnabledInterrupts* 函数。

支持 组件级电源管理的 驱动程序必须实现以下两个回调函数：

*CLIENT_RestoreBankHardwareContextCLIENT_SaveBankHardwareContext*若要指示硬件支持组件级电源管理，驱动程序在*CLIENT_QueryControllerBasicInformation*回调函数提供的设备信息中设置 **BankIdlePowerMgmtSupported** 标志位。 有关详细信息，请参阅 **CLIENT_CONTROLLER_BASIC_INFORMATION** 中的 **Flags** 成员的说明。

CLIENT_PreProcessControllerInterrupt、*CLIENT_ReconfigureInterrupt*和 *CLIENT_ControllerSpecificFunction*回调函数是可选的，GpioClx 支持解决某些 GPIO 控制器实现中特定于硬件的问题。 只有具有特殊要求的 GPIO 控制器驱动程序才能实现这些功能。

GPIO 设备上下文

2025/07/18

常规用途 I/O (GPIO) 控制器设备由框架设备对象表示。 GPIO 控制器驱动程序可以将设备上下文与此设备对象相关联。 驱动程序使用此设备上下文来持久存储有关 GPIO 控制器设备状态的信息。

当 GPIO 框架扩展 （GpioClx） 调用驱动程序实现的事件回调函数时，GpioClx 会将设备上下文作为参数传递给此函数。 回调函数检查设备上下文以确定设备的当前状态。 如果函数更改此状态，它将相应地更新设备上下文。

GpioClx 为设备对象分配存储。 如果 GPIO 控制器驱动程序有多个设备对象，则其中每个对象的设备上下文大小相同。 在 **DriverEntry** 例程中，驱动程序调用 **GPIO_CLX_RegisterClient** 方法来注册其回调函数，并指定其所需的设备上下文大小。 稍后，在 *EvtDriverDeviceAdd* 回调例程期间，驱动程序调用 **GPIO_CLX_ProcessAddDevicePostDeviceCreate** 方法将新设备对象传递给 GpioClx，GpioClx 将为此对象分配设备上下文。 此后，当 GpioClx 调用驱动程序实现的回调函数时，此设备上下文将作为参数传递给函数。

将 GPIO 控制器分区为管脚库

项目 • 2023/06/15

驱动程序开发人员可以选择将常规用途 I/O (GPIO) 控制器设备分区为两个或更多组 GPIO 引脚。 例如，具有 64 个 GPIO 引脚的 GPIO 控制器设备可由 GPIO 控制器驱动程序描述为两个库，每个库都有 32 个 GPIO 引脚。 开发人员可以提供单个驱动程序来管理 GPIO 控制器设备中的所有库，此驱动程序通常使用一个设备对象来表示整个设备。 但是，设备中的部分或所有库可以独立于设备中的其他库进行管理。

通常，GPIO 控制器驱动程序会出于以下原因之一选择将 GPIO 控制器分区为两个或多个库：

- 可以独立于其他库中的引脚管理一个库中的 GPIO 引脚的电源状态。
- GPIO 控制器中的引脚总数大于 64。

GPIO 框架扩展 (GpioClx) 支持的最大库大小为 64 个引脚。 包含超过 64 个引脚的 GPIO 控制器设备必须由驱动程序分区为两个或更多个库，每个库包含不超过 64 个引脚。

为了确定 GPIO 控制器如何分区为库，GpioClx 调用 *CLIENT_QueryControllerBasicInformation* 事件回调函数。 此函数由 GPIO 控制器驱动程序实现，它提供描述 GPIO 控制器的属性和功能的 **CLIENT_CONTROLLER_BASIC_INFORMATION** 结构。 此结构的两个成员 **TotalPins** 和 **NumberOfPinsPerBank** 指定如何将 GPIO 控制器中的引脚分区为库。 **TotalPins** 指定 GPIO 控制器中的引脚总数， **NumberOfPinsPerBank** 指定每个组的引脚数。 如果 N 是控制器中的库数，则库的编号从 0 到 N–1。 除最后一个银行 (，即银行编号 N-1) 必须包含 **NumberOfPinsPerBank** 成员中指定的引脚数。 最后一个库可以有任意数量的引脚，从 1 到 **NumberOfPinsPerBank**。

GpioClx 根据 **TotalPins** 和**NumberOfPinsPerBank** 成员的值确定 GPIO 控制器中的库总数。 GpioClx 使用以下整数公式来计算总库数：

(**TotalPins** + **NumberOfPinsPerBank** – 1) / **NumberOfPinsPerBank** 在某些 GPIO 控制器设备中，设备中的一组引脚可以独立于同一设备中的其他库打开或切换到低功耗状态。 因此，当某个特定库处于空闲状态时，可以将此库切换到低功耗状态以降低功耗。 为了适应此类设备，GpioClx 支持 组件级电源管理。 GpioClx 定义了两种组件级电源状态：F0 (完全打开) 和 F1 (低功耗或关闭) 。

为了确定一组 GPIO 引脚是否支持组件级电源管理，GpioClx 调用 *CLIENT_QuerySetControllerInformation* 事件回调函数。 此函数的 *InputBuffer* 参数是指向 **CLIENT_CONTROLLER_QUERY_SET_INFORMATION_INPUT** 结构的指针。 若要请求电源管理信息，调用方将此结构的 **RequestType** 成员设置为 **QueryBankPowerInformation**。

如果 GPIO 库支持组件级电源管理，则 GpioClx 允许在库空闲时转换为 F1 电源状态。 在银行进入 F1 状态之前，GpioClx 调用 *CLIENT_SaveBankHardwareContext* 事件回调函数，告知驱动程序保存硬件上下文 (主要是库的寄存器内容) 。 稍后，在银行进入 F0 状态后，GpioClx 调用

CLIENT_RestoreBankHardwareContext 事件回调函数，告知驱动程序还原以前保存的硬件上下文。

GPIO 控制器驱动程序的实现问题

2025/07/18

GPIO 框架扩展 (GpioClx) 提供了一个灵活的设备驱动程序接口 (DDI)。 此 DDI 使开发人员能够在备选回调接口中进行选择。 驱动程序开发人员应实现最适合目标 GPIO 控制器设备硬件体系结构的事件回调函数集。

例如，如果 GPIO 控制器驱动程序支持从 GPIO I/O 引脚读取和写入，开发人员可以选择实现以下回调函数对之一：

*CLIENT_ReadGpioPins*和*CLIENT_WriteGpioPins*、*CLIENT_ReadGpioPinsUsingMask*和*CLIENT_WriteGpioPinsUsingMask*函数接收一个银行编号、一个GPIO引脚编号数组，以及一个用于读取或写入这些引脚的位值的数据缓冲区。 如果在读取或写入过程中通常只访问少量 GPIO 引脚，这一对回调可能会实现最佳效果。 此实现通常用于硬件寄存器未进行内存映射的 GPIO 控制器。 但是，如果在读取或写入作期间可能会访问多个 GPIO 引脚，或者 GPIO 控制器硬件可以有效地并行访问多个 GPIO 引脚，则另一对回调函数可能会产生更好的实现。

*CLIENT_ReadGpioPinsUsingMask*和*CLIENT_WriteGpioPinsUsingMask*回调函数可以在一次调用中读取或写入最多 64 个引脚的库。 *CLIENT_ReadGpioPinsUsingMask*函数将 GPIO 引脚值读入 64 位掩码。 *CLIENT_WriteGpioPinsUsingMask*函数使用两个 64 位掩码。 一个掩码指示要设置的 GPIO 引脚，另一个掩码指示要清除的 GPIO 引脚。 此实现通常用于内存映射 GPIO 控制器。