# Reset controller API

## Introduction

Reset controllers are central units that control the reset signals to multiple peripherals. The reset controller API is split into two parts: the consumer driver interface (API reference), which allows peripheral drivers to request control over their reset input signals, and the reset controller driver interface (API reference), which is used by drivers for reset controller devices to register their reset controls to provide them to the consumers.

While some reset controller hardware units also implement system restart functionality, restart handlers are out of scope for the reset controller API.

### Glossary

The reset controller API uses these terms with a specific meaning:

Reset line

> Physical reset line carrying a reset signal from a reset controller hardware unit to a peripheral module.

Reset control

> Control method that determines the state of one or multiple reset lines. Most commonly this is a single bit in reset controller register space that either allows direct control over the physical state of the reset line, or is self-clearing and can be used to trigger a predetermined pulse on the reset line. In more complicated reset controls, a single trigger action can launch a carefully timed sequence of pulses on multiple reset lines.

Reset controller

> A hardware module that provides a number of reset controls to control a number of reset lines.

Reset consumer

> Peripheral module or external IC that is put into reset by the signal on a reset line.

## Consumer driver interface

This interface provides an API that is similar to the kernel clock framework. Consumer drivers use get and put operations to acquire and release reset controls. Functions are provided to assert and deassert the controlled reset lines, trigger reset pulses, or to query reset line status.

When requesting reset controls, consumers can use symbolic names for their reset inputs, which are mapped to an actual reset control on an existing reset controller device by the core.

A stub version of this API is provided when the reset controller framework is not in use in order to minimize the need to use ifdefs.

### Shared and exclusive resets

The reset controller API provides either reference counted deassertion and assertion or direct, exclusive control. The distinction between shared and exclusive reset controls is made at the time the reset control is requested, either via devm_reset_control_get_shared() or via devm_reset_control_get_exclusive(). This choice determines the behavior of the API calls made with the reset control.

Shared resets behave similarly to clocks in the kernel clock framework. They provide reference counted deassertion, where only the first deassert, which increments the deassertion reference count to one, and the last assert which decrements the deassertion reference count back to zero, have a physical effect on the reset line.

Exclusive resets on the other hand guarantee direct control. That is, an assert causes the reset line to be asserted immediately, and a deassert causes the reset line to be deasserted immediately.

### Assertion and deassertion

Consumer drivers use the reset_control_assert() and reset_control_deassert() functions to assert and deassert reset lines. For shared reset controls, calls to the two functions must be balanced.

Note that since multiple consumers may be using a shared reset control, there is no guarantee that calling reset_control_assert() on a shared reset control will actually cause the reset line to be asserted. Consumer drivers using shared reset controls should assume that the reset line may be kept deasserted at all times. The API only guarantees that the reset line can not be asserted as long as any consumer has requested it to be deasserted.

### Triggering

Consumer drivers use reset_control_reset() to trigger a reset pulse on a self-deasserting reset control. In general, these resets can not be shared between multiple consumers, since requesting a pulse from any consumer driver will reset all connected peripherals.

The reset controller API allows requesting self-deasserting reset controls as shared, but for those only the first trigger request causes an actual pulse to be issued on the reset line. All further calls to this function have no effect until all consumers have called reset_control_rearm(). For shared reset controls, calls to the two functions must be balanced. This allows devices that only require an initial reset at any point before the driver is probed or resumed to share a pulsed reset line.

### Querying

Only some reset controllers support querying the current status of a reset line, via reset_control_status(). If supported, this function returns a positive non-zero value if the given reset line is asserted. The reset_control_status() function does not accept a reset control array handle as its input parameter.

### Optional resets

Often peripherals require a reset line on some platforms but not on others. For this, reset controls can be requested as optional using devm_reset_control_get_optional_exclusive() or devm_reset_control_get_optional_shared(). These functions return a NULL pointer instead of an error when the requested reset control is not specified in the device tree. Passing a NULL pointer to the reset_control functions causes them to return quietly without an error.

### Reset control arrays

Some drivers need to assert a bunch of reset lines in no particular order. devm_reset_control_array_get() returns an opaque reset control handle that can be used to assert, deassert, or trigger all specified reset controls at once. The reset control API does not guarantee the order in which the individual controls therein are handled.

## Reset controller driver interface

Drivers for reset controller modules provide the functionality necessary to assert or deassert reset signals, to trigger a reset pulse on a reset line, or to query its current state. All functions are optional.

### Initialization

Drivers fill a struct :c:type:`reset_controller_dev` and register it with reset_controller_register() in their probe function. The actual functionality is implemented in callback functions via a struct :c:type:`reset_control_ops`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 166); *backlink***
>
> Unknown interpreted text role "c:type".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 166); *backlink***
>
> Unknown interpreted text role "c:type".

## API reference

The reset controller API is documented here in two parts: the reset consumer API and the reset controller driver API.

### Reset consumer API

Reset consumers can control a reset line using an opaque reset control handle, which can be obtained from devm_reset_control_get_exclusive() or devm_reset_control_get_shared(). Given the reset control, consumers can call reset_control_assert() and reset_control_deassert(), trigger a reset pulse using reset_control_reset(), or query the reset line status using reset_control_status().

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 188)**
>
> Unknown directive type "kernel-doc".
>
> ```
> .. kernel-doc:: include/linux/reset.h
> ```

```
   :internal:
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 191)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/reset/core.c
   :functions: reset_control_reset
               reset_control_assert
               reset_control_deassert
               reset_control_status
               reset_control_acquire
               reset_control_release
               reset_control_rearm
               reset_control_put
               of_reset_control_get_count
               of_reset_control_array_get
               devm_reset_control_array_get
               reset_control_get_count
```

## Reset controller driver API

Reset controller drivers are supposed to implement the necessary functions in a static constant structure :c:type:`reset_control_ops`, allocate and fill out a struct :c:type:`reset_controller_dev`, and register it using devm_reset_controller_register().

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 208); *backlink***

Unknown interpreted text role "c:type".

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 208); *backlink***

Unknown interpreted text role "c:type".

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 213)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/reset-controller.h
   :internal:
```

**System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\[linux-master][Documentation][driver-api]reset.rst, line 216)**

Unknown directive type "kernel-doc".

```
.. kernel-doc:: drivers/reset/core.c
   :functions: of_reset_simple_xlate
               reset_controller_register
               reset_controller_unregister
               devm_reset_controller_register
               reset_controller_add_lookup
```