# XHCI初始化

本文本并不暗示以下操作的特定顺序，但应在将USBCMD寄存器运行/停止（R/S）位设置为“1”之前完成这些操作

在CONFIG寄存器（5.4.7）中编程启用的最大设备插槽数（MaxSlotsEn）字段，以启用系统软件将要使用的设备插槽。

将设备上下文基址阵列指针（DCBAAP）寄存器（5.4.6）编程为64位地址，该地址指向设备上下文基址阵列所在的位置。

通过使用指向命令环第一个TRB起始地址的64位地址对命令环控制寄存器（5.4.5）进行编程，定义命令环出列指针。

初始化中断：

分配并初始化设置消息地址和消息数据的MSI-X消息表，并启用矢量。

至少，表向量条目0应被初始化并启用。有关详细信息，请参阅PCI规范。

分配并初始化MSI-X挂起位阵列（PBA，5.2.8.4）。

将MSI-X能力结构中的表偏移量和PBA偏移量分别指向MSI-X消息控制表和未决位阵列。

初始化MSI-X能力结构的消息控制寄存器（5.2.8.3）。

通过以下方式初始化每个活动中断器：

定义事件环：（有关事件环管理的讨论，请参阅第4.9.4节。）

分配并初始化事件环段。

分配事件环段表（ERST）（第6.5节）。初始化ERST表条目，以指向并定义相应事件环段的大小（以TRB为单位）。

将中断器事件环段表大小（ERSTSZ）寄存器（5.5.2.3.1）编程为事件环段表所述的段数。

使用事件环段表中描述的第一段的起始地址，对中断器事件环出列指针（ERDP）寄存器（5.5.2.3.3）进行编程。

使用指向事件环段表所在位置的64位地址指针，对中断器事件环段表基址（ERSTBA）寄存器（5.5.2.3.2）进行编程。

请注意，写入ERSTBA将启用事件环。有关事件环寄存器及其初始化的更多信息，请参见第4.9.4节。

定义中断

通过设置MSI-X能力结构消息控制寄存器中的MSI-X启用标志，启用MSI-X中断机制（5.2.8.3）。

使用目标中断调整率初始化中断调整寄存器（5.5.2.2）的间隔字段。

通过向USBCMD寄存器的中断器使能（INTE）标志写入“1”来使能系统总线中断生成（5.4.1）。

通过向中断器管理寄存器（5.5.2.1）的“启用”（IE）字段写入“1”，启用中断器。

写入USBCMD（5.4.1），通过将运行/停止（R/S）位设置为“1”来打开主机控制器。此操作允许xHC开始接受门铃参考。

static int usb\_register\_bus(struct usb\_bus \*bus)

[ 0.728474] xhci\_hcd 0000:06:00.2: new USB bus registered, assigned bus number 3

int usb\_add\_hcd(struct usb\_hcd \*hcd,

    if (hcd->driver->reset) {

        retval = hcd->driver->reset(hcd);

        if (retval < 0) {

            dev\_err(hcd->self.controller, "can't setup: %d\n",

                    retval);

            goto err\_hcd\_driver\_setup;

        }

    }

}

[ 10.728498] xhci\_hcd 0000:06:00.2: can't setup: -110

[ 10.728504] xhci\_hcd 0000:06:00.2: USB bus 3 deregistered

[ 10.728586] xhci\_hcd 0000:06:00.2: init 0000:06:00.2 fail, -110

static const struct hc\_driver xhci\_hc\_driver = {

.reset =        NULL, /\* set in xhci\_init\_driver() \*/

}

void xhci\_init\_driver(struct hc\_driver \*drv,

              const struct xhci\_driver\_overrides \*over)

{

    BUG\_ON(!over);

    /\* Copy the generic table to drv then apply the overrides \*/

    \*drv = xhci\_hc\_driver;

    if (over) {

        drv->hcd\_priv\_size += over->extra\_priv\_size;

        if (over->reset)

            drv->reset = over->reset;

        if (over->start)

            drv->start = over->start;

        if (over->add\_endpoint)

            drv->add\_endpoint = over->add\_endpoint;

        if (over->drop\_endpoint)

            drv->drop\_endpoint = over->drop\_endpoint;

        if (over->check\_bandwidth)

            drv->check\_bandwidth = over->check\_bandwidth;

        if (over->reset\_bandwidth)

            drv->reset\_bandwidth = over->reset\_bandwidth;

    }

}

static int \_\_init xhci\_plat\_init(void)

{

    xhci\_init\_driver(&xhci\_plat\_hc\_driver, &xhci\_plat\_overrides);

    return platform\_driver\_register(&usb\_xhci\_driver);

}

static int xhci\_pci\_setup(struct usb\_hcd \*hcd)

{

    struct xhci\_hcd     \*xhci;

    struct pci\_dev      \*pdev = to\_pci\_dev(hcd->self.controller);

    int         retval;

    xhci = hcd\_to\_xhci(hcd);

    if (!xhci->sbrn)

        pci\_read\_config\_byte(pdev, XHCI\_SBRN\_OFFSET, &xhci->sbrn);

    /\* imod\_interval is the interrupt moderation value in nanoseconds. \*/

    xhci->imod\_interval = 40000;

    retval = xhci\_gen\_setup(hcd, xhci\_pci\_quirks);

    if (retval)

        return retval;

    if (!usb\_hcd\_is\_primary\_hcd(hcd))

        return 0;

    xhci\_dbg(xhci, "Got SBRN %u\n", (unsigned int) xhci->sbrn);

    /\* Find any debug ports \*/

    return xhci\_pci\_reinit(xhci, pdev);

}