```
**********************************************************
   79545 Wed Jul  4 02:42:40 2018
new/usr/src/uts/common/io/usb/hcd/xhci/xhci.c
xhci_Quiesce
**********************************************************
```

*_____unchanged_portion_omitted_*

```
1085 static int
1086 xhci_reg_poll(xhci_t *xhcip, xhci_reg_type_t rt, int reg, uint32_t mask,
1087     uint32_t targ, uint_t tries, int delay_ms)
1088 {
1089         uint_t i;

1091         for (i = 0; i < tries; i++) {
1092                 uint32_t val = xhci_get32(xhcip, rt, reg);
1093                 if (!quiesce_active && xhci_check_regs_acc(xhcip)
1094                     != DDI_FM_OK) {
1093                 if (xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1095                         ddi_fm_service_impact(xhcip->xhci_dip,
1096                             DDI_SERVICE_LOST);
1097                         return (EIO);
1098                 }

1100                 if ((val & mask) == targ)
1101                         return (0);

1103                 delay(drv_usectohz(delay_ms * 1000));
1104         }
1105         return (ETIMEDOUT);
1106 }
```

*_____unchanged_portion_omitted_*

```
1571 static int
1572 xhci_controller_stop(xhci_t *xhcip)
1573 {
1574         uint32_t cmdreg;

1576         cmdreg = xhci_get32(xhcip, XHCI_R_OPER, XHCI_USBCMD);
1577         if (!quiesce_active && xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1576         if (xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1578                 xhci_error(xhcip, "failed to read USB Command register: "
1579                     "encountered fatal FM register error");
1580                 ddi_fm_service_impact(xhcip->xhci_dip, DDI_SERVICE_LOST);
1581                 return (EIO);
1582         }

1584         cmdreg &= ~(XHCI_CMD_RS | XHCI_CMD_INTE);
1585         xhci_put32(xhcip, XHCI_R_OPER, XHCI_USBCMD, cmdreg);
1586         if (!quiesce_active && xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1585         if (xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1587                 xhci_error(xhcip, "failed to write USB Command register: "
1588                     "encountered fatal FM register error");
1589                 ddi_fm_service_impact(xhcip->xhci_dip, DDI_SERVICE_LOST);
1590                 return (EIO);
1591         }

1593         /*
1594          * Wait up to 50ms for this to occur. The specification says that this
1595          * should stop within 16ms, but we give ourselves a bit more time just
1596          * in case.
1597          */
1598         return (xhci_reg_poll(xhcip, XHCI_R_OPER, XHCI_USBSTS, XHCI_STS_HCH,
1599             XHCI_STS_HCH, 50, 10));
1600 }

1602 static int
```



```
1603 xhci_controller_reset(xhci_t *xhcip)
1604 {
1605         int ret;
1606         uint32_t cmdreg;

1608         cmdreg = xhci_get32(xhcip, XHCI_R_OPER, XHCI_USBCMD);
1609         if (!quiesce_active && xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1608         if (xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1610                 xhci_error(xhcip, "failed to read USB Command register for "
1611                     "reset: encountered fatal FM register error");
1612                 ddi_fm_service_impact(xhcip->xhci_dip, DDI_SERVICE_LOST);
1613                 return (EIO);
1614         }

1616         cmdreg |= XHCI_CMD_HCRST;
1617         xhci_put32(xhcip, XHCI_R_OPER, XHCI_USBCMD, cmdreg);
1618         if (!quiesce_active && xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1617         if (xhci_check_regs_acc(xhcip) != DDI_FM_OK) {
1619                 xhci_error(xhcip, "failed to write USB Command register for "
1620                     "reset: encountered fatal FM register error");
1621                 ddi_fm_service_impact(xhcip->xhci_dip, DDI_SERVICE_LOST);
1622                 return (EIO);
1623         }

1625         /*
1626          * Some controllers apparently don't want to be touched for at least 1ms
1627          * after we initiate the reset. Therefore give all controllers this
1628          * moment to breathe.
1629          */
1630         delay(drv_usectohz(xhci_reset_delay));

1632         /*
1633          * To tell that the reset has completed we first verify that the reset
1634          * has finished and that the USBCMD register no longer has the reset bit
1635          * asserted. However, once that's done we have to go verify that CNR
1636          * (Controller Not Ready) is no longer asserted.
1637          */
1638         if ((ret = xhci_reg_poll(xhcip, XHCI_R_OPER, XHCI_USBCMD,
1639             XHCI_CMD_HCRST, 0, 500, 10)) != 0)
1640                 return (ret);

1642         return (xhci_reg_poll(xhcip, XHCI_R_OPER, XHCI_USBSTS,
1643             XHCI_STS_CNR, 0, 500, 10));
1644 }
```

*_____unchanged_portion_omitted_*

```
1975 /* QUIESCE(9E) to support fast reboot */
1976 int
1977 xhci_quiesce(dev_info_t *dip)
1978 {
1979         xhci_t *xhcip;

1981         xhcip = ddi_get_soft_state(xhci_soft_state, ddi_get_instance(dip));

1983         return (xhci_controller_stop(xhcip) == 0 &&
1984             xhci_controller_reset(xhcip) == 0 ? DDI_SUCCESS : DDI_FAILURE);
1985 }

1987 static int
1988 xhci_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
1989 {
1990         int ret, inst, route;
1991         xhci_t *xhcip;

1993         if (cmd != DDI_ATTACH)
1994                 return (DDI_FAILURE);
```

```
1996         inst = ddi_get_instance(dip);
1997         if (ddi_soft_state_zalloc(xhci_soft_state, inst) != 0)
1998                 return (DDI_FAILURE);
1999         xhcip = ddi_get_soft_state(xhci_soft_state, ddi_get_instance(dip));
2000         xhcip->xhci_dip = dip;

2002         xhcip->xhci_regs_capoff = PCI_EINVAL32;
2003         xhcip->xhci_regs_operoff = PCI_EINVAL32;
2004         xhcip->xhci_regs_runoff = PCI_EINVAL32;
2005         xhcip->xhci_regs_dooroff = PCI_EINVAL32;

2007         xhci_fm_init(xhcip);
2008         xhcip->xhci_seq |= XHCI_ATTACH_FM;

2010         if (pci_config_setup(xhcip->xhci_dip, &xhcip->xhci_cfg_handle) !=
2011             DDI_SUCCESS) {
2012                 goto err;
2013         }
2014         xhcip->xhci_seq |= XHCI_ATTACH_PCI_CONFIG;
2015         xhcip->xhci_vendor_id = pci_config_get16(xhcip->xhci_cfg_handle,
2016             PCI_CONF_VENID);
2017         xhcip->xhci_device_id = pci_config_get16(xhcip->xhci_cfg_handle,
2018             PCI_CONF_DEVID);

2020         if (xhci_regs_map(xhcip) == B_FALSE) {
2021                 goto err;
2022         }

2024         xhcip->xhci_seq |= XHCI_ATTACH_REGS_MAP;

2026         if (xhci_regs_init(xhcip) == B_FALSE)
2027                 goto err;

2029         if (xhci_read_params(xhcip) == B_FALSE)
2030                 goto err;

2032         if (xhci_identify(xhcip) == B_FALSE)
2033                 goto err;

2035         if (xhci_alloc_intrs(xhcip) == B_FALSE)
2036                 goto err;
2037         xhcip->xhci_seq |= XHCI_ATTACH_INTR_ALLOC;

2039         if (xhci_add_intr_handler(xhcip) == B_FALSE)
2040                 goto err;
2041         xhcip->xhci_seq |= XHCI_ATTACH_INTR_ADD;

2043         mutex_init(&xhcip->xhci_lock, NULL, MUTEX_DRIVER,
2044             (void *)(uintptr_t)xhcip->xhci_intr_pri);
2045         cv_init(&xhcip->xhci_statecv, NULL, CV_DRIVER, NULL);
2046         xhcip->xhci_seq |= XHCI_ATTACH_SYNCH;

2048         if (xhci_port_count(xhcip) == B_FALSE)
2049                 goto err;

2051         if (xhci_controller_takeover(xhcip) == B_FALSE)
2052                 goto err;

2054         /*
2055          * We don't enable interrupts until after we take over the controller
2056          * from the BIOS. We've observed cases where this can cause spurious
2057          * interrupts.
2058          */
2059         if (xhci_ddi_intr_enable(xhcip) == B_FALSE)
2060                 goto err;
```



```
2061         xhcip->xhci_seq |= XHCI_ATTACH_INTR_ENABLE;

2063         if ((ret = xhci_controller_stop(xhcip)) != 0) {
2064                 xhci_error(xhcip, "failed to stop controller: %s",
2065                     ret == EIO ? "encountered FM register error" :
2066                     "timed out while waiting for controller");
2067                 goto err;
2068         }

2070         if ((ret = xhci_controller_reset(xhcip)) != 0) {
2071                 xhci_error(xhcip, "failed to reset controller: %s",
2072                     ret == EIO ? "encountered FM register error" :
2073                     "timed out while waiting for controller");
2074                 goto err;
2075         }

2077         if ((ret = xhci_controller_configure(xhcip)) != 0) {
2078                 xhci_error(xhcip, "failed to configure controller: %d", ret);
2079                 goto err;
2080         }

2082         /*
2083          * Some systems support having ports routed to both an ehci and xhci
2084          * controller. If we support it and the user hasn't requested otherwise
2085          * via a driver.conf tuning, we reroute it now.
2086          */
2087         route = ddi_prop_get_int(DDI_DEV_T_ANY, xhcip->xhci_dip,
2088             DDI_PROP_DONTPASS, "xhci-reroute", XHCI_PROP_REROUTE_DEFAULT);
2089         if (route != XHCI_PROP_REROUTE_DISABLE &&
2090             (xhcip->xhci_quirks & XHCI_QUIRK_INTC_EHCI))
2091                 (void) xhci_reroute_intel(xhcip);

2093         if ((ret = xhci_controller_start(xhcip)) != 0) {
2094                 xhci_log(xhcip, "failed to reset controller: %s",
2095                     ret == EIO ? "encountered FM register error" :
2096                     "timed out while waiting for controller");
2097                 goto err;
2098         }
2099         xhcip->xhci_seq |= XHCI_ATTACH_STARTED;

2101         /*
2102          * Finally, register ourselves with the USB framework itself.
2103          */
2104         if ((ret = xhci_hcd_init(xhcip)) != 0) {
2105                 xhci_error(xhcip, "failed to register hcd with usba");
2106                 goto err;
2107         }
2108         xhcip->xhci_seq |= XHCI_ATTACH_USBA;

2110         if ((ret = xhci_root_hub_init(xhcip)) != 0) {
2111                 xhci_error(xhcip, "failed to load the root hub driver");
2112                 goto err;
2113         }
2114         xhcip->xhci_seq |= XHCI_ATTACH_ROOT_HUB;

2116         return (DDI_SUCCESS);

2118 err:
2119         (void) xhci_cleanup(xhcip);
2120         return (DDI_FAILURE);
2121 }
```

_____***unchanged_portion_omitted_***

```
2187 static struct dev_ops xhci_dev_ops = {
2188         DEVO_REV,                       /* devo_rev */
2189         0,                              /* devo_refcnt */
```

```
2190         xhci_getinfo,                  /* devo_getinfo */
2191         nulldev,                       /* devo_identify */
2192         nulldev,                       /* devo_probe */
2193         xhci_attach,                   /* devo_attach */
2194         xhci_detach,                   /* devo_detach */
2195         nodev,                         /* devo_reset */
2196         &xhci_cb_ops,                  /* devo_cb_ops */
2197         &usba_hubdi_busops,            /* devo_bus_ops */
2198         usba_hubdi_root_hub_power,     /* devo_power */
2199         xhci_quiesce                   /* devo_quiesce */
2186         ddi_quiesce_not_supported      /* devo_quiesce */
2200 };
```

_____unchanged_portion_omitted_