

new/usr/src/uts/common/io/usb/hcd/xhci/xhci.c

1

```
*****
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_usecstohz(delay_ms * 1000));
1104         }
1105     }
1106     return (ETIMEDOUT);
_____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
```

new/usr/src/uts/common/io/usb/hcd/xhci/xhci.c

2

```
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 lms
1627              * after we initiate the reset. Therefore give all controllers this
1628              * moment to breathe.
1629              */
1630             delay(drv_usecstohz(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_SYNC;

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 }
2122 }
2123 }
2124 }
2125 }
2126 }
2127 }
2128 }
2129 }
2130 }
2131 }
2132 }
2133 }
2134 }
2135 }
2136 }
2137 }
2138 }
2139 }
2140 }
2141 }
2142 }
2143 }
2144 }
2145 }
2146 }
2147 }
2148 }
2149 }
2150 }
2151 }
2152 }
2153 }
2154 }
2155 }
2156 }
2157 }
2158 }
2159 }
2160 }
2161 }
2162 }
2163 }
2164 }
2165 }
2166 }
2167 }
2168 }
2169 }
2170 }
2171 }
2172 }
2173 }
2174 }
2175 }
2176 }
2177 }
2178 }
2179 }
2180 }
2181 }
2182 }
2183 }
2184 }
2185 }
2186 }
2187 }
2188 }
2189 }
2190 }
2191 }
2192 }
2193 }
2194 }
2195 }
2196 }
2197 }
2198 }
2199 }
2200 }
2201 }
2202 }
2203 }
2204 }
2205 }
2206 }
2207 }
2208 }
2209 }
2210 }
2211 }
2212 }
2213 }
2214 }
2215 }
2216 }
2217 }
2218 }
2219 }
2220 }
2221 }
2222 }
2223 }
2224 }
2225 }
2226 }
2227 }
2228 }
2229 }
2230 }
2231 }
2232 }
2233 }
2234 }
2235 }
2236 }
2237 }
2238 }
2239 }
2240 }
2241 }
2242 }
2243 }
2244 }
2245 }
2246 }
2247 }
2248 }
2249 }
2250 }
2251 }
2252 }
2253 }
2254 }
2255 }
2256 }
2257 }
2258 }
2259 }
2260 }
2261 }
2262 }
2263 }
2264 }
2265 }
2266 }
2267 }
2268 }
2269 }
2270 }
2271 }
2272 }
2273 }
2274 }
2275 }
2276 }
2277 }
2278 }
2279 }
2280 }
2281 }
2282 }
2283 }
2284 }
2285 }
2286 }
2287 }
2288 }
2289 }
2290 }
2291 }
2292 }
2293 }
2294 }
2295 }
2296 }
2297 }
2298 }
2299 }
2300 }
2301 }
2302 }
2303 }
2304 }
2305 }
2306 }
2307 }
2308 }
2309 }
2310 }
2311 }
2312 }
2313 }
2314 }
2315 }
2316 }
2317 }
2318 }
2319 }
2320 }
2321 }
2322 }
2323 }
2324 }
2325 }
2326 }
2327 }
2328 }
2329 }
2330 }
2331 }
2332 }
2333 }
2334 }
2335 }
2336 }
2337 }
2338 }
2339 }
2340 }
2341 }
2342 }
2343 }
2344 }
2345 }
2346 }
2347 }
2348 }
2349 }
2350 }
2351 }
2352 }
2353 }
2354 }
2355 }
2356 }
2357 }
2358 }
2359 }
2360 }
2361 }
2362 }
2363 }
2364 }
2365 }
2366 }
2367 }
2368 }
2369 }
2370 }
2371 }
2372 }
2373 }
2374 }
2375 }
2376 }
2377 }
2378 }
2379 }
2380 }
2381 }
2382 }
2383 }
2384 }
2385 }
2386 }
2387 }
2388 }
2389 }
2390 }
2391 }
2392 }
2393 }
2394 }
2395 }
2396 }
2397 }
2398 }
2399 }
2400 }
2401 }
2402 }
2403 }
2404 }
2405 }
2406 }
2407 }
2408 }
2409 }
2410 }
2411 }
2412 }
2413 }
2414 }
2415 }
2416 }
2417 }
2418 }
2419 }
2420 }
2421 }
2422 }
2423 }
2424 }
2425 }
2426 }
2427 }
2428 }
2429 }
2430 }
2431 }
2432 }
2433 }
2434 }
2435 }
2436 }
2437 }
2438 }
2439 }
2440 }
2441 }
2442 }
2443 }
2444 }
2445 }
2446 }
2447 }
2448 }
2449 }
2450 }
2451 }
2452 }
2453 }
2454 }
2455 }
2456 }
2457 }
2458 }
2459 }
2460 }
2461 }
2462 }
2463 }
2464 }
2465 }
2466 }
2467 }
2468 }
2469 }
2470 }
2471 }
2472 }
2473 }
2474 }
2475 }
2476 }
2477 }
2478 }
2479 }
2480 }
2481 }
2482 }
2483 }
2484 }
2485 }
2486 }
2487 }
2488 }
2489 }
2490 }
2491 }
2492 }
2493 }
2494 }
2495 }
2496 }
2497 }
2498 }
2499 }
2500 }
2501 }
2502 }
2503 }
2504 }
2505 }
2506 }
2507 }
2508 }
2509 }
2510 }
2511 }
2512 }
2513 }
2514 }
2515 }
2516 }
2517 }
2518 }
2519 }
2520 }
2521 }
2522 }
2523 }
2524 }
2525 }
2526 }
2527 }
2528 }
2529 }
2530 }
2531 }
2532 }
2533 }
2534 }
2535 }
2536 }
2537 }
2538 }
2539 }
2540 }
2541 }
2542 }
2543 }
2544 }
2545 }
2546 }
2547 }
2548 }
2549 }
2550 }
2551 }
2552 }
2553 }
2554 }
2555 }
2556 }
2557 }
2558 }
2559 }
2560 }
2561 }
2562 }
2563 }
2564 }
2565 }
2566 }
2567 }
2568 }
2569 }
2570 }
2571 }
2572 }
2573 }
2574 }
2575 }
2576 }
2577 }
2578 }
2579 }
2580 }
2581 }
2582 }
2583 }
2584 }
2585 }
2586 }
2587 }
2588 }
2589 }
2590 }
2591 }
2592 }
2593 }
2594 }
2595 }
2596 }
2597 }
2598 }
2599 }
2600 }
2601 }
2602 }
2603 }
2604 }
2605 }
2606 }
2607 }
2608 }
2609 }
2610 }
2611 }
2612 }
2613 }
2614 }
2615 }
2616 }
2617 }
2618 }
2619 }
2620 }
2621 }
2622 }
2623 }
2624 }
2625 }
2626 }
2627 }
2628 }
2629 }
2630 }
2631 }
2632 }
2633 }
2634 }
2635 }
2636 }
2637 }
2638 }
2639 }
2640 }
2641 }
2642 }
2643 }
2644 }
2645 }
2646 }
2647 }
2648 }
2649 }
2650 }
2651 }
2652 }
2653 }
2654 }
2655 }
2656 }
2657 }
2658 }
2659 }
2660 }
2661 }
2662 }
2663 }
2664 }
2665 }
2666 }
2667 }
2668 }
2669 }
2670 }
2671 }
2672 }
2673 }
2674 }
2675 }
2676 }
2677 }
2678 }
2679 }
2680 }
2681 }
2682 }
2683 }
2684 }
2685 }
2686 }
2687 }
2688 }
2689 }
2690 }
2691 }
2692 }
2693 }
2694 }
2695 }
2696 }
2697 }
2698 }
2699 }
2700 }
2701 }
2702 }
2703 }
2704 }
2705 }
2706 }
2707 }
2708 }
2709 }
2710 }
2711 }
2712 }
2713 }
2714 }
2715 }
2716 }
2717 }
2718 }
2719 }
2720 }
2721 }
2722 }
2723 }
2724 }
2725 }
2726 }
2727 }
2728 }
2729 }
2730 }
2731 }
2732 }
2733 }
2734 }
2735 }
2736 }
2737 }
2738 }
2739 }
2740 }
2741 }
2742 }
2743 }
2744 }
2745 }
2746 }
2747 }
2748 }
2749 }
2750 }
2751 }
2752 }
2753 }
2754 }
2755 }
2756 }
2757 }
2758 }
2759 }
2760 }
2761 }
2762 }
2763 }
2764 }
2765 }
2766 }
2767 }
2768 }
2769 }
2770 }
2771 }
2772 }
2773 }
2774 }
2775 }
2776 }
2777 }
2778 }
2779 }
2780 }
2781 }
2782 }
2783 }
2784 }
2785 }
2786 }
2787 }
2788 }
2789 }
2790 }
2791 }
2792 }
2793 }
2794 }
2795 }
2796 }
2797 }
2798 }
2799 }
2800 }
2801 }
2802 }
2803 }
2804 }
2805 }
2806 }
2807 }
2808 }
2809 }
2810 }
2811 }
2812 }
2813 }
2814 }
2815 }
2816 }
2817 }
2818 }
2819 }
2820 }
2821 }
2822 }
2823 }
2824 }
2825 }
2826 }
2827 }
2828 }
2829 }
2830 }
2831 }
2832 }
2833 }
2834 }
2835 }
2836 }
2837 }
2838 }
2839 }
2840 }
2841 }
2842 }
2843 }
2844 }
2845 }
2846 }
2847 }
2848 }
2849 }
2850 }
2851 }
2852 }
2853 }
2854 }
2855 }
2856 }
2857 }
2858 }
2859 }
2860 }
2861 }
2862 }
2863 }
2864 }
2865 }
2866 }
2867 }
2868 }
2869 }
2870 }
2871 }
2872 }
2873 }
2874 }
2875 }
2876 }
2877 }
2878 }
2879 }
2880 }
2881 }
2882 }
2883 }
2884 }
2885 }
2886 }
2887 }
2888 }
2889 }
2890 }
2891 }
2892 }
2893 }
2894 }
2895 }
2896 }
2897 }
2898 }
2899 }
2900 }
2901 }
2902 }
2903 }
2904 }
2905 }
2906 }
2907 }
2908 }
2909 }
2910 }
2911 }
2912 }
2913 }
2914 }
2915 }
2916 }
2917 }
2918 }
2919 }
2920 }
2921 }
2922 }
2923 }
2924 }
2925 }
2926 }
2927 }
2928 }
2929 }
2930 }
2931 }
2932 }
2933 }
2934 }
2935 }
2936 }
2937 }
2938 }
2939 }
2940 }
2941 }
2942 }
2943 }
2944 }
2945 }
2946 }
2947 }
2948 }
2949 }
2950 }
2951 }
2952 }
2953 }
2954 }
2955 }
2956 }
2957 }
2958 }
2959 }
2960 }
2961 }
2962 }
2963 }
2964 }
2965 }
2966 }
2967 }
2968 }
2969 }
2970 }
2971 }
2972 }
2973 }
2974 }
2975 }
2976 }
2977 }
2978 }
2979 }
2980 }
2981 }
2982 }
2983 }
2984 }
2985 }
2986 }
2987 }
2988 }
2989 }
2990 }
2991 }
2992 }
2993 }
2994 }
2995 }
2996 }
2997 }
2998 }
2999 }
3000 }

```

```
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