Script for ACPM enabling

Script to automate ACPM enabling by Luis R. Rodriguez. External Link

```
#!/bin/bash
# Copyright (c) 2010 Luis R. Rodriguez <mcgrof@gmail.com>
#
# Permission to use, copy, modify, and/or distribute this software for any
# purpose with or without fee is hereby granted, provided that the above
# copyright notice and this permission notice appear in all copies.
#
# THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
# WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
# MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
# ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
# WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
# ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
# OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.


# ASPM Tuning script
#
# This script lets you enable ASPM on your devices in case your BIOS
# does not have it enabled for some reason. If your BIOS does not have
# it enabled it is usually for a good reason so you should only use this if
# you know what you are doing. Typically you would only need to enable
# ASPM manually when doing development and using a card that typically
# is not present on a laptop, or using the cardbus slot. The BIOS typically
# disables ASPM for foreign cards and on the cardbus slot. Check also
# if you may need to do other things than what is below on your vendor
# documentation.
#
# To use this script You will need for now to at least query your device
# PCI endpoint and root complex addresses using the convention output by
# lspci: [<bus>]:[<slot>].[<func>]
#
# For example:
#
# 03:00.0 Network controller: Atheros Communications Inc. AR9300 Wireless
LAN adaptor (rev 01
# 00:1c.1 PCI bridge: Intel Corporation 82801H (ICH8 Family) PCI Express
Port 2 (rev 03)
#
# The root complex for the endpoint can be found using lspci -t
#
# For more details refer to:
#
# http://wireless.kernel.org/en/users/Documentation/ASPM
```

```
# You just need to modify these three values:

ROOT_COMPLEX="00:1c.1"
ENDPOINT="03:00.0"

# We'll only enable the last 2 bits by using a mask
# of :3 to setpci, this will ensure we keep the existing
# values on the byte.
#
# Hex  Binary  Meaning
# -------------------------
# 0    0b00    L0 only
# 1    0b01    L0s only
# 2    0b10    L1 only
# 3    0b11    L1 and L0s
ASPM_SETTING=3

function aspm_setting_to_string()
{
    case $1 in
    0)
        echo -e "\t${BLUE}L0 only${NORMAL}, ${RED}ASPM disabled${NORMAL}"
        ;;
    1)
        ;;
    2)
        echo -e "\t${GREEN}L1 only${NORMAL}"
        ;;
    3)
        echo -e "\t${GREEN}L1 and L0s${NORMAL}"
        ;;
    *)
        echo -e "\t${RED}Invalid${NORMAL}"
        ;;
    esac
}


#####################################################################
# Do not edit below here unless you are sending me a patch
#####################################################################
#
# TODO: patches are welcomed to me until we submit to to
#       PCI Utilities upstream.
#
# This can be improved by in this order:
#
#   * Accept arguments for endpoint and root complex address, and
#     desired ASPM settings
#   * Look for your ASPM capabilities by quering your
```

```
#     LnkCap register first. Use these values to let you
#     select whether you want to enable only L1 or L1 & L0s
#   * Searching for your root complex for you
#   * Search for your PCI device by using the driver
#   * Disable your driver and ask to reboot ?
#   * Rewrite in C
#   * Write ncurses interface [ wishlist ]
#   * Write GTK/QT interface [ wishlist ]
#   * Submit upstream as aspm.c to the PCI Utilities, which are
#     maintained by Martin Mares <mj@ucw.cz>

# Pretty colors
GREEN="\033[01;32m"
YELLOW="\033[01;33m"
NORMAL="\033[00m"
BLUE="\033[34m"
RED="\033[31m"
PURPLE="\033[35m"
CYAN="\033[36m"
UNDERLINE="\033[02m"

# we can surely read the spec to get a better value
MAX_SEARCH=20
SEARCH_COUNT=1
ASPM_BYTE_ADDRESS="INVALID"

ROOT_PRESENT=$(lspci | grep -c "$ROOT_COMPLEXT")
ENDPOINT_PRESENT=$(lspci | grep -c "$ENDPOINT")

if [[ $(id -u) != 0 ]]; then
    echo "This needs to be run as root"
    exit 1
fi

if [[ $ROOT_PRESENT -eq 0 ]]; then
    echo "Root complex $ROOT_COMPLEX is not present"
    exit
fi

if [[ $ENDPOINT_PRESENT -eq 0 ]]; then
    echo "Endpoint $ENDPOINT is not present"
    exit
fi

# XXX: lspci -s some_device_not_existing does not return positive
# if the device does not exist, fix this upstream
function device_present()
{

    PRESENT=$(lspci | grep -c "$1")
    COMPLAINT="${RED}not present${NORMAL}"
```

```
    if [[ $PRESENT -eq 0 ]]; then
        if [[ $2 != "present" ]]; then
            COMPLAINT="${RED}disappeared${NORMAL}"
        fi

        echo -e "Device ${BLUE}${1}${NORMAL} $COMPLAINT"
        return 1
    fi
    return 0
}

function find_aspm_byte_address()
{
    device_present $ENDPOINT present
    if [[ $? -ne 0 ]]; then
        exit
    fi

    SEARCH=$(setpci -s $1 34.b)
    # We know on the first search $SEARCH will not be
    # 10 but this simplifies the implementation.
    while [[ $SEARCH != 10 && $SEARCH_COUNT -le $MAX_SEARCH ]]; do
        END_SEARCH=$(setpci -s $1 ${SEARCH}.b)

        # Convert hex digits to uppercase for bc
        SEARCH_UPPER=$(printf "%X" 0x${SEARCH})

        if [[ $END_SEARCH = 10 ]]; then
            ASPM_BYTE_ADDRESS=$(echo "obase=16; ibase=16; $SEARCH_UPPER +
10" | bc)
            break
        fi

        SEARCH=$(echo "obase=16; ibase=16; $SEARCH + 1" | bc)
        SEARCH=$(setpci -s $1 ${SEARCH}.b)

        let SEARCH_COUNT=$SEARCH_COUNT+1
    done

    if [[ $SEARCH_COUNT -ge $MAX_SEARCH ]]; then
        echo -e "Long loop while looking for ASPM word for $1"
        return 1
    fi
    return 0
}

function enable_aspm_byte()
{
    device_present $1 present
    if [[ $? -ne 0 ]]; then
```

```
        exit
    fi

    find_aspm_byte_address $1
    if [[ $? -ne 0 ]]; then
        return 1
    fi

    ASPM_BYTE_HEX=$(setpci -s $1 ${ASPM_BYTE_ADDRESS}.b)
    ASPM_BYTE_HEX=$(printf "%X" 0x${ASPM_BYTE_HEX})
    # setpci doesn't support a mask on the query yet, only on the set,
    # so to verify a setting on a mask we have no other optoin but
    # to do do this stuff ourselves.
    DESIRED_ASPM_BYTE_HEX=$(printf "%X" $(( (0x${ASPM_BYTE_HEX} & ~0x7)
|0x${ASPM_SETTING})))

    if [[ $ASPM_BYTE_ADDRESS = "INVALID" ]]; then
        echo -e "No ASPM byte could be found for $(lspci -s $1)"
        return
    fi

    echo -e "$(lspci -s $1)"
    echo -en "\t${YELLOW}0x${ASPM_BYTE_ADDRESS}${NORMAL} :
${CYAN}0x${ASPM_BYTE_HEX}${GREEN} -->
${BLUE}0x${DESIRED_ASPM_BYTE_HEX}${NORMAL} ... "

    device_present $1 present
    if [[ $? -ne 0 ]]; then
        exit
    fi

    # Avoid setting if already set
    if [[ $ASPM_BYTE_HEX = $DESIRED_ASPM_BYTE_HEX ]]; then
        echo -e "[${GREEN}SUCESS${NORMAL}] (${GREEN}already set${NORMAL})"
        aspm_setting_to_string $ASPM_SETTING
        return 0
    fi

    # This only writes the last 3 bits
    setpci -s $1 ${ASPM_BYTE_ADDRESS}.b=${ASPM_SETTING}:3

    sleep 3

    ACTUAL_ASPM_BYTE_HEX=$(setpci -s $1 ${ASPM_BYTE_ADDRESS}.b)
    ACTUAL_ASPM_BYTE_HEX=$(printf "%X" 0x${ACTUAL_ASPM_BYTE_HEX})

    # Do not retry this if it failed, if it failed to set.
    # Likey if it failed its a good reason and you should look
    # into that.
    if [[ $ACTUAL_ASPM_BYTE_HEX != $DESIRED_ASPM_BYTE_HEX ]]; then
        echo -e "\t[${RED}FAIL${NORMAL}] (0x${ACTUAL_ASPM_BYTE_HEX})"
```

```
        return 1
    fi

    echo -e "\t[${GREEN}SUCCESS]${NORMAL}]"
    aspm_setting_to_string $ASPM_SETTING

    return 0
}

device_present $ENDPOINT not_sure
if [[ $? -ne 0 ]]; then
    exit
fi

echo -e "${CYAN}Root complex${NORMAL}:"
enable_aspm_byte $ROOT_COMPLEX
echo

echo -e "${CYAN}Endpoint${NORMAL}:"
enable_aspm_byte $ENDPOINT
echo
```