SSDT Overlays

In order to support ACPI open-ended hardware configurations (e.g. development boards) we need a way to augment the ACPI configuration provided by the firmware image. A common example is connecting sensors on I2C / SPI buses on development boards.

Although this can be accomplished by creating a kernel platform driver or recompiling the firmware image with updated ACPI tables, neither is practical: the former proliferates board specific kernel code while the latter requires access to firmware tools which are often not publicly available.

Because ACPI supports external references in AML code a more practical way to augment firmware ACPI configuration is by dynamically loading user defined SSDT tables that contain the board specific information.

For example, to enumerate a Bosch BMA222E accelerometer on the I2C bus of the Minnowboard MAX development board exposed via the LSE connector [1], the following ASL code can be used:

```
DefinitionBlock ("minnowmax.aml", "SSDT", 1, "Vendor", "Accel", 0x00000003)
{
    External (\_SB.I2C6, DeviceObj)

    Scope (\_SB.I2C6)
    {
        Device (STAC)
        {
            Name (_HID, "BMA222E")
            Name (RBUF, ResourceTemplate ()
            {
                I2cSerialBus (0x0018, ControllerInitiated, 0x00061A80,
                              AddressingMode7Bit, "\\_SB.I2C6", 0x00,
                              ResourceConsumer, ,)
                GpioInt (Edge, ActiveHigh, Exclusive, PullDown, 0x0000,
                         "\\_SB.GPO2", 0x00, ResourceConsumer, , )
                { // Pin list
                    0
                }
            })

            Method (_CRS, 0, Serialized)
            {
                Return (RBUF)
            }
        }
    }
}
```

which can then be compiled to AML binary format:

```
$ iasl minnowmax.asl

Intel ACPI Component Architecture
ASL Optimizing Compiler version 20140214-64 [Mar 29 2014]
Copyright (c) 2000 - 2014 Intel Corporation

ASL Input:     minnomax.asl - 30 lines, 614 bytes, 7 keywords
AML Output:    minnowmax.aml - 165 bytes, 6 named objects, 1 executable opcodes
```

[1] https://www.elinux.org/Minnowboard:MinnowMax#Low_Speed_Expansion_.28Top.29

The resulting AML code can then be loaded by the kernel using one of the methods below.

Loading ACPI SSDTs from initrd

This option allows loading of user defined SSDTs from initrd and it is useful when the system does not support EFI or when there is not enough EFI storage.

It works in a similar way with initrd based ACPI tables override/upgrade: SSDT AML code must be placed in the first, uncompressed, initrd under the "kernel/firmware/acpi" path. Multiple files can be used and this will translate in loading multiple tables. Only SSDT and OEM tables are allowed. See initrd_table_override.txt for more details.

Here is an example:

```
# Add the raw ACPI tables to an uncompressed cpio archive.
# They must be put into a /kernel/firmware/acpi directory inside the
# cpio archive.
# The uncompressed cpio archive must be the first.
# Other, typically compressed cpio archives, must be
# concatenated on top of the uncompressed one.
```

```
mkdir -p kernel/firmware/acpi
cp ssdt.aml kernel/firmware/acpi

# Create the uncompressed cpio archive and concatenate the original initrd
# on top:
find kernel | cpio -H newc --create > /boot/instrumented_initrd
cat /boot/initrd >>/boot/instrumented_initrd
```

Loading ACPI SSDTs from EFI variables

This is the preferred method, when EFI is supported on the platform, because it allows a persistent, OS independent way of storing the user defined SSDTs. There is also work underway to implement EFI support for loading user defined SSDTs and using this method will make it easier to convert to the EFI loading mechanism when that will arrive. To enable it, the CONFIG_EFI_CUSTOM_SSDT_OVERLAYS shoyld be chosen to y.

In order to load SSDTs from an EFI variable the `"efivar_ssdt=..."` kernel command line parameter can be used (the name has a limitation of 16 characters). The argument for the option is the variable name to use. If there are multiple variables with the same name but with different vendor GUIDs, all of them will be loaded.

In order to store the AML code in an EFI variable the efivarfs filesystem can be used. It is enabled and mounted by default in /sys/firmware/efi/efivars in all recent distribution.

Creating a new file in /sys/firmware/efi/efivars will automatically create a new EFI variable. Updating a file in /sys/firmware/efi/efivars will update the EFI variable. Please note that the file name needs to be specially formatted as "Name-GUID" and that the first 4 bytes in the file (little-endian format) represent the attributes of the EFI variable (see EFI_VARIABLE_MASK in include/linux/efi.h). Writing to the file must also be done with one write operation.

For example, you can use the following bash script to create/update an EFI variable with the content from a given file:

```
#!/bin/sh -e

while [ -n "$1" ]; do
        case "$1" in
        "-f") filename="$2"; shift;;
        "-g") guid="$2"; shift;;
        *) name="$1";;
        esac
        shift
done

usage()
{
        echo "Syntax: ${0##*/} -f filename [ -g guid ] name"
        exit 1
}

[ -n "$name" -a -f "$filename" ] || usage

EFIVARFS="/sys/firmware/efi/efivars"

[ -d "$EFIVARFS" ] || exit 2

if stat -tf $EFIVARFS | grep -q -v de5e81e4; then
        mount -t efivarfs none $EFIVARFS
fi

# try to pick up an existing GUID
[ -n "$guid" ] || guid=$(find "$EFIVARFS" -name "$name-*" | head -n1 | cut -f2- -d-)

# use a randomly generated GUID
[ -n "$guid" ] || guid="$(cat /proc/sys/kernel/random/uuid)"

# efivarfs expects all of the data in one write
tmp=$(mktemp)
/bin/echo -ne "\007\000\000\000" | cat - $filename > $tmp
dd if=$tmp of="$EFIVARFS/$name-$guid" bs=$(stat -c %s $tmp)
rm $tmp
```

Loading ACPI SSDTs from configfs

This option allows loading of user defined SSDTs from user space via the configfs interface. The CONFIG_ACPI_CONFIGFS option must be select and configfs must be mounted. In the following examples, we assume that configfs has been mounted in /sys/kernel/config.

New tables can be loading by creating new directories in /sys/kernel/config/acpi/table and writing the SSDT AML code in the aml attribute:

```
cd /sys/kernel/config/acpi/table
```

```
mkdir my_ssdt
cat ~/ssdt.aml > my_ssdt/aml
```