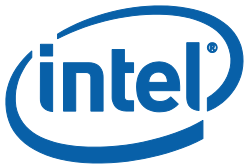

Rapid Start Technology

Driver Interface Specification

November 2011

Revision 0.70

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
N/A	0.70	Added information regarding requirement to release driver handle as soon as possible	November 2010
N/A	0.52	Changed product name to Rapid Start Technology	June 2010
N/A	0.51	Change GUID_AOAC_INTERFACE to GUID_IFFS_INTERFACE	May 2010
N/A	0.5	Initial release	May 2010

§

1 Introduction

This document specifies the user mode interface with the Intel® Rapid Start Technology driver.

The Intel® Rapid Start Technology driver is a Windows Driver Model (WDM) driver. The driver is designed to allow user-mode access the ACPI methods exposed by the Intel® Rapid Start Technology virtual ACPI device. The Intel® Rapid Start Technology virtual device is specified by the ACPI _HID "INT3392".

Communication with the driver from user mode is performed through I/O Control Codes (IOCTL's) using the "DeviceIoControl" Windows API function. The definition of the supported IOCTL's is contained herein.

The details of processing within the driver are outside the scope of this document.

1.1 Terminology

Term	Description
ACPI	Advanced Configuration and Power Interface
API	Application Programming Interface
GUID	Globally Unique IDentifier
IOCTL	I/O Control Code used to send control commands to drivers
WDM	Windows Driver Model

1.2 Reference Documents

Document	Document No./Location
Advance Configuration and Power Interface Specification (Latest rev.)	

§

2 Driver Interface

2.1 Device Interface GUID

The driver exports a device interface for access by user-mode applications. The interface is identified by the system using the following device interface class GUID:

```
static const GUID GUID_IFFS_INTERFACE =
{0xe7ab5e0c,0x11e8,0x4802,{0xbf,0x90,0x31,0x80,0x61,0xb5,0x7d,0x1}};
```

The GUID_IFFS_INTERFACE GUID may be used to obtain a handle to the device for communication through the supported IOCTL's. (See Appendix A for reference code.)

Warning: Only one handle to the driver may be open by system software at any given time. There is a low probability that multiple pieces of software will attempt to access the driver at the same time. If a handle is held by one piece of software, any attempts by other software to obtain a handle will fail. Software must take this possibility into account when accessing the driver. The following recommendations are made:

1. Software should release the handle to the driver immediately after use and not hold the handle indefinitely.
2. Software should retry obtaining the handle multiple times with a short delay between attempts. (200 attempts with a 1ms sleep time is recommended.)

This requirement is due to the characteristics of the ACPI device object exposed by the ACPI driver.

2.2 Driver IOCTL Interface

The IOCTL's supported by the driver interface are listed in Table 1.

Table 1. Supported IOCTL's

IOCTL NAME	USAGE	IOCTL DECLARATION
IOCTL_IFFS_GFFS	Returns the function status from the GFFS ACPI method	CTL_CODE(0x8086, 0x800, METHOD_BUFFERED, FILE_ANY_ACCESS)
IOCTL_IFFS_SFFS	Sets the function status through the SFFS ACPI method	CTL_CODE(0x8086, 0x801, METHOD_BUFFERED, FILE_ANY_ACCESS)
IOCTL_IFFS_GFTV	Returns the "Enter After" timer value from the GFTV ACPI method	CTL_CODE(0x8086, 0x802, METHOD_BUFFERED, FILE_ANY_ACCESS)
IOCTL_IFFS_SFTV	Sets the "Enter After" timer value through the SFFS ACPI method	CTL_CODE(0x8086, 0x803, METHOD_BUFFERED, FILE_ANY_ACCESS)

2.2.1 IOCTL_IFFS_GFFS

The IOCTL_IFFS_GFFS IOCTL returns the current Intel® Rapid Start Technology status by evaluating the GFFS ACPI method of the Intel® Rapid Start Technology ACPI device.

Data is returned from the IOCTL using the IFFS_GFFS_DATA structure shown in Figure 1. The caller must provide a sufficient buffer for the data returned.

The driver will return an appropriate error status if the IOCTL is unsuccessful.

Figure 1. IFFS_GFFS_DATA Structure

```
typedef union _IFFS_GFFS_DATA {
    struct {
        ULONG entryOnRtcWakeEnable:1;
        ULONG entryOnCritBattWakeEnable:1;
    } u;
    ULONG asUlong;
} IFFS_GFFS_DATA, *PIFFS_GFFS_DATA;
```

entryOnRtcWakeEnable	Intel® Rapid Start Technology entry on RTC Wake: 0 = Disabled 1 = Enabled
entryOnCritBattWakeEnable	Intel® Rapid Start Technology entry on Critical Battery Wake: 0 = Disabled 1 = Enabled

2.2.2 IOCTL_IFFS_SFFS

The IOCTL_IFFS_SFFS IOCTL is used to set the current Intel® Rapid Start Technology status by calling the SFFS ACPI method of the Intel® Rapid Start Technology ACPI device.

The state is set through the IOCTL using the IFFS_GFFS_DATA structure shown in Figure 1.

The driver will return an appropriate error status if the IOCTL is unsuccessful.

2.2.3 IOCTL_IFFS_GFTV

The IOCTL_IFFS_GFTV IOCTL returns the BIOS Intel® Rapid Start Technology "Entry After" timer setting by evaluating the GFTV ACPI method of the Intel® Rapid Start Technology ACPI device.

Data is returned from the IOCTL using the IFFS_GFTV_DATA structure shown in Figure 2. The caller must provide a sufficient buffer for the data returned.

The driver will return an appropriate error status if the IOCTL is unsuccessful.

Figure 2. IFFS_GFFS_DATA Structure

```
typedef struct _IFFS_GFTV_DATA {
    ULONG timerValue;
} IFFS_GFTV_DATA, *PIFFS_GFTV_DATA;
```

timerValue	Intel® Rapid Start Technology timer value in minutes

2.2.4 IOCTL_IFFS_SFTV

The IOCTL_IFFS_SFTV IOCTL is used to set the Intel® Rapid Start Technology timeout value by calling the SFTV ACPI method of the Intel® Rapid Start Technology ACPI device.

The state is set through the IOCTL using the IFFS_GFTV_DATA structure shown in Figure 2.

The driver will return an appropriate error status if the IOCTL is unsuccessful.

3 Appendix A

The following code may be used to obtain a device path to the Intel® Rapid Start Technology driver. The path is required to send IOCTL's to the driver using the DeviceIoControl API function.

The "Setupapi.lib" library must be linked to the program.

The function returns TRUE on success, and FALSE otherwise. If successful, the path to interface is returned by pointer.

NOTE: The code provided is one possible methodology which may be used to obtain the device path from the device interface GUILD. Other solutions are possible and some changes may be required; dependent upon the development environment used.

```
#include <setupapi.h>

BOOL GetInterfaceAlias(
    GUID *pGuid,
    CString* pAliasString
  )
{
    BOOL                            status = FALSE;
    BOOL                            bRet;
    HDEVINFO                        hDevInfo = INVALID_HANDLE_VALUE;
    ULONG                           detailSize = 0;
    SP_DEVICE_INTERFACE_DATA   deviceInterfaceData;
    PSP_DEVICE_INTERFACE_DETAIL_DATA pDeviceInterfaceDetailData = NULL;

    //
    // Get a device information set of the devices supporting the
    //required interface
    //
    hDevInfo = SetupDiGetClassDevs(pGuid,
                                   NULL,
                                   NULL,
                                   DIGCF_DEVICEINTERFACE | DIGCF_PRESENT);
    if (hDevInfo == INVALID_HANDLE_VALUE) {
        goto exit;
    }

    //
    // Get the first interface from the set
    //
    deviceInterfaceData.cbSize = sizeof(deviceInterfaceData);
    bRet = SetupDiEnumDeviceInterfaces(hDevInfo,
                                       NULL,
                                       pGuid,
                                       0,     // Just get first instance
                                       &deviceInterfaceData);
    if (!bRet) {
        goto exit;
```

```
    }

    //
    // Determine the buffer size needed for the path
    //
    bRet = SetupDiGetDeviceInterfaceDetail(hDevInfo,
                                           &deviceInterfaceData,
                                           NULL,
                                           0,
                                           &detailSize,
                                           NULL);
    if (!bRet) {
        if (GetLastError() != ERROR_INSUFFICIENT_BUFFER) {
            goto exit;
        }
    }

    pDeviceInterfaceDetailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA) new
char[detailSize];
    if(pDeviceInterfaceDetailData == NULL) {
        goto exit;
    }
    pDeviceInterfaceDetailData->cbSize =
sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);
    bRet = SetupDiGetDeviceInterfaceDetail(hDevInfo,
                                           &deviceInterfaceData,
                                           pDeviceInterfaceDetailData,
                                           detailSize,
                                           NULL,
                                           NULL);
    if (!bRet) {
        goto exit;
    }

    pAliasString->SetString(pDeviceInterfaceDetailData->DevicePath);
    status = TRUE;

  exit:
    if(pDeviceInterfaceDetailData != NULL) {
        delete [] pDeviceInterfaceDetailData;
    }
    return status;
}
```

§