Working Draft
Do Not Distribute

Cyrix Virtual ACPI Implementation Guide

1 Revisions

Time/Date	By	Comments
5/13/1998	John Kabat	Added new OEM routines Added Compile time defines for ATOMIC IO
4/24/1998	John Kabat	Internal test and control functions (0Fxx) were removed.

2 Files

The files used by VACPI consist of three groups of files. The Primary Files contain the base ACPI code. The OEM files contain chipset specific files. The Modified files are standard VSA files that have been or need to be modified in order to use VACPI.

2.1 Primary Files

There are four files in the Virtual ACPI system. All files reside in the PM directory of VSA (VSA\PM). These files will normally not have to be modified.

2.1.1 VACPI.C

The primary file for VACPI. This file contains the base code for the VACPI implementation.

2.1.2 CACPI.H

This is an "include" file used various VACPI programs to define useful constants.

2.1.3 VACPIA.ASM

This file is used for routines that cannot be written in the 'C' language.

2.1.4 VACPIPM.H

This file contains macros that are used when modifying PM.C

2.2 OEM Files

These files contain chipset specific code. Each file contains routines called by VACPI.C to perform specific functions.

The file name must always be in the form VAxxxxxx.C, where xxxxxx is the type chip or the platform to be used.

2.2.1 Routines

The following routines are called by VACPI to perform specific functions: These routines must be provided in the OEM file. Dummy routines may be specified.

2.2.1.1 oem_acpi_early_init

void oem_acpi_early_init(void)

This routine is called by **acpi_early_init** to perform OEM specific functions that must be done prior to **pm_init**.

2.2.1.2 oem_acpi_late_init

void oem_acpi_late_init(void)

This routine is called by acpi_late_init to perform OEM specific functions that must be done after pm_init.

2.2.1.3 oem_acpi_init

void oem_acpi_init(void)

This function is called by the **acpi_init** routine to perform OEM specific functions that are required when the APCI is enabled by writing **ACPI_ENABLE** to the **SMI_CMD** register.

2.2.1.4 oem_acpi_de_init

void oem_acpi_de_init(void)

This function is called by the **acpi_de_init** routine to perform OEM specific functions that are required when APCI is disabled by writing **ACPI_DISABLE** to the **SMI_CMD** register.

2.2.1.5 oem_acpi_power_off

void oem_acpi_power_off(void)

This routine is called by the **acpi_power_off** routine to soft power off the system. This routine **MUST** power the system off.

2.2.1.6 oem_acpi_gpio_cascade

WORD oem_acpi_gpio_cascade(int gpio)

This routine is call whenever VACPI wishes to check whether the OEM file supports cascading of this GPIO. Returns a zero if not cascaded. Returns a one if this gpio is to be processed by oem_acpi_gpio_event. Called with the GPIO number (0-7).

2.2.1.7 oem_acpi_gpio_event

WORD oem_acpi_gpio_event(int gpio)

This routine is called whenever VACPI detects a GPIO SMI event. The OEM routine will check the GPIO number (0-7) and if it does not want to process this GPIO itself, returns a zero. Otherwise the OEM routine processes the GPIO event, such as recognizing and handling events cascaded from a Super-IO chip, and returns a one to say that it processes the GPIO.

2.2.1.8 oem_acpi_save_to_disk

void oem_acpi_save_to_disk(void)

This routine is called when VACPI receives a S4BIOS command. This routine is responsible for saving the System to Disk, Placing the system in Sleep, and handling the Restore on wakeup.

2.3 Modified VSA files

When adapting VACPI to a VSA that does not already have VACPI, two files must be modified. If the VSA already contains VACPI, this is not required.

2.3.1 VSA\BUILD\MAKEFILE

This file is modified to allow the VACPI portion of VSA to be compiled and to indicate which OEM files should be used.

2.3.1.1 General Modifications

These modifications are to be added to the **makefile**. They will add the ability to use of not use VACPI as desired. With just these modifications, ACPI will not be added. To compile with VACPI see the next section.

Just before the line:

```
CODEC_C_SRCS    = codec.c mixer.c codspec.c
```

Add the lines:

```
# added for acpi
!if "$(ACPI)"=="VACPI"
ACPI_C_SRCS = vacpi.c va$(ACPI_CHIP).c
ACPI_ASM_SRCS = vacpia.asm
DACPI= /DACPI=$(ACPI)
!else
ACPI_C_SRCS =
ACPI_ASM_SRCS =
DACPI=
!endif
```

This added block defines the various MAKEFILE macros for compiling VACPI.

Change the lines reading:

```
PM_C_SRCS   = pm.c pmapm.c pmdevice.c pmutils.c
PM_ASM_SRCS = pma.asm
```

To: (added code is underlined)

```
PM_C_SRCS   = pm.c pmapm.c pmdevice.c pmutils.c $(ACPI_C_SRCS)
PM_ASM_SRCS = pma.asm $(ACPI_ASM_SRCS)
```

In addition, change to lines reading: (added code is underlined)
(NOTE: The following lines are split into two lines for formatting. In the actual file, each is a single line).

```
AS_OPTS = $(BRIDG) $(SB) $(VGA) $(SMM) $(OEM) $(GPIO) $(S2D) $(CPU) /c
      /Cx /Sa /W3 $(ALIST)
CC_OPTS = $(BRIDG) $(SB) $(VGA) $(SMM) $(OEM) $(GPIO) $(S2D) $(CPU) /c
      /AS /FPi87 /G3fs /W3 $(COPTS_OPT) $(CLIST)
ICC_OPTS = $(BRIDG) $(SB) $(VGA) $(SMM) $(OEM) $(GPIO) $(S2D) $(CPU) /c
      /AC /FPi87 /G3fs /W3 $(COPTS_OPT) $(CLIST)
```

To:

```
AS_OPTS = $(BRIDG) $(SB) $(VGA) $(SMM) $(OEM) $(GPIO) $(S2D) $(CPU)
      $(DACPI) /c /Cx /Sa /W3 $(ALIST)
CC_OPTS = $(BRIDG) $(SB) $(VGA) $(SMM) $(OEM) $(GPIO) $(S2D) $(CPU)
      $(DACPI) /c /AS /FPi87 /G3fs /W3 $(COPTS_OPT) $(CLIST)
ICC_OPTS = $(BRIDG) $(SB) $(VGA) $(SMM) $(OEM) $(GPIO) $(S2D) $(CPU)
      $(DACPI) /c /AC /FPi87 /G3fs /W3 $(COPTS_OPT) $(CLIST)
```

This will add the ability to add VACPI to the PM module.

2.3.1.2 Platform Specific Modifications

In order to enable VACPI to be compiled into VSA, for a specific platform, the following modifications must be made:

To enable an OEM file to be added an entry of the form: **ACPI_CHIP=xxxxxx**, where **xxxxxx** is the same as the **xxxxxx** in the OEM file to use, must be added to the line used to compile for a specific platform.

To enable VACP the entry **ACPI=VACPI** must be added. If this entry is not present, not VACPI module will be generated.

This is an example from the makefile that will add VACPI to a Kestrel system that uses the SMC FDC37C932 chip.

```
# Cyrix's Kestrel board
kes:  begin last.all
  $(MAKE) OEM=/DOEM_BA CODEC=ak4532 ACPI=VACPI ACPI_CHIP=smc932 \
            install.exe gxvsa.rom
```

2.3.2 VSA\PM\PM.C

This file is modified to allow VACPI to hook into the system. The hook macros are written so that if VSA is to be built without VACPI, no calls to VACPI will be made.

All versions of ACPI now have a correctly modified PM.C file.

3 Configuration of VACPI

3.1 Compile Time Configuration

By careful modification of the CAPCI.H, file defaults for VACPI may be modified.

3.1.1 SCI IRQ Number

Change the constant DEFAULT_ACPI_SCI from 9 to the desired value. Take care to use a free IRQ as VAPCI can **NOT** share an interrupt.

3.1.2 ACPI Fixed Register Base Address

Change the constant ACPI_FIXED_BASE from 0xac00 to the desire ACPI Fixed Register Base. The base used must be a free block of 32 IO addresses.

Under no circumstances, change the order of registers within this block.

3.1.3 SMI_CMD values

You may change the values of the constants ACPI_ENABLE, ACPI_DISABLE, and S4BIOS_REQ if it is desired to use other values. This may be done to emulate another ACPI chip.

3.1.4 Sleep State values

You may change the values of the constants SLEEP_S1, SLEEP_S2, SLEEP_S4 and SLEEP_S5 if it is desired to use other values. This may be done to emulate another ACPI chip.

3.1.5 Selecting Optional Components.

The standard VACPI file is shipped with several option components turned off. At the beginning of the VACPI.C file there is a section with several #defines commented out. Uncomment the define for the component that you wish to activate.

3.1.5.1 Activating ATOMIC IO

ATOMIC IO is disabled by default. To enable ATOMIC IO un-comment the line "#define USE_ATOMIC_IO 1" in the VACPI.C file.

3.2 Load time Early/late Init

These routines are called when PM in VSA is initialized. These routines are modified at compile time and executed at run time.
Tasks done in these routines typically include configuring GPIO pins, configuring SIO GP pins if needed, and assigning GPIO pins to emulated ACPI Fixed Registers. The choice of **oem_acpi_early_init()** or **oem_acpi_late_init()** is somewhat arbitrary. The choice normally depends on if it is desired to override something that was done by **pm_init()**.

3.2.1 GPIO Routing

Use the routine **acpi_set_gpio()** to route the 5520 GPIO pins. The fist parameter is the GPIO to use, the second is the bit in either the PM1A_STS or the GPE0_STS registers. The last parameter is a group of flags that control the triggering and special functions of the GPIO pin. The constants used are defined in the CACPI.H file.

Example:
acpi_setup_gpio(ASETUP_GPIO_STS1,ASETUP_PM1A_STS_8,RISING_EDGE | IS_PWRBTN);

This will set GPIO 1 to be routed to the PWRBTN_STS in the PM1A_STS register, trigger on the rising edge, and enable the 4-second power button override.
Alternatively, the GPIO can be routed by a write to the ACPI setup index and data registers.

3.2.2 GPIO Direction

Use **acpi_gpio_set_dir** to set the direction of a GIPO pin. This is only required when a pin is desired to be an output for control purposes. The first parameter is the GPIO to set the direction for. The second parameter is a **zero** for input and a **one** for output.

Example:
acpi_gpio_set_dir(1,1);

This will set GPIO1 to an output.

3.2.3 GPIO State

Use **acpi_gpio_set_state** to set the state of a GIPO pin. This is only required when a GPIO is configured as an output and the state must be controlled. The first parameter is the GPIO to set the state of. The second parameter is a **zero** for off and a **one** for on.

Example:
acpi_gpio_set_state(1,1);

This will set GPIO1 on.

3.3 Run Time BIOS configuration

If the user desires, configuration may be done by the BIOS using IO to the ACPI SETUP_INDEX and SETUP_DATA registers.

3.3.1 0x00 – No operation

3.3.2 0x10 – 0x17 GPIO Mapping

By default, there is no mapping of GPIOs to bits in the _STS registers. Use this command to perform such mapping.

3.3.3 0x30 – 0x3F IRQ wakeup enable

3.3.4 0x40 Generate GBL_STS

A write of 0x40 to SETUP_INDEX will set the GBL_STS bit in the PM1A_STS register and generate a SCI if GBL_EN is set.

3.3.5 0x41 Configure SCI IRQ

By default the SCI is routed to the interrupt specified by DEFAULT_ACPI_SCI (normally IRQ 9). To change the SCI from the default, write 0x41 to the SETUP_INDEX register and then write the desired SCI to SETUP_DATA (0-15).

3.3.6 0x42 Enable Reads of ACPI Fixed Registers

When using the CYRIX 5520, a write of 0x42 to the SETUP_INDEX register is required to allow reads of the VACPI fixed registers. Prior to this command, no reads on the VACPI registers will succeed.

3.3.7 0x43 Do Atomic IO sequence

This optional command allows a sequence of IO operations to be done with no interruption. Certain Super-IO chips must receive unlock codes with NO intervening IO. In addition, other Super-IO chips do not allow IO to devices while in configuration mode. This command will insure that IO operations are completed without interruption. The address of a sequence of IO commands is places in the SETUP_DATA register. The command sequence will then be processed immediately.

The block address written to SETUP_DATA is a physical linear 32-bit address. **It must NOT be in paged memory.**

The IO command sequence consists of two parts: the signature and length block and the IO block. There is only one signature/length block. There may be one or more IO blocks.

The signature block consists of four double words:

Byte offset	Value
0	Signature - Always 0x00000070
4	The length of the entire buffer, including the signature block in bytes.
8	Reserved - set to 0
12	Reserved - set to 0

The IO block consists of four bytes followed by three double words:

Byte offset	Description
0	BYTE - Operation Type 1 - Read 2 - Write 3 - Read/And/Or/Write 4 - Define index and data ports In addition, values may be ORed in to the upper 2 bits of this byte to indicate that special functions are desired: 0x80 - Do not perform this operation (convert to NO-OP) 0x40 - This is an index operation.
1	BYTE - Reserved set to 0
2	BYTE - IO length - Determines whether a BYTE, WORD or DWORD operation is performed 1 - Byte operation

Byte offset	Description
	2 - Word operation 3 - DWORD operation If byte 0 is a 4 then this field is used to indicate the size of the index write.
3	BYTE - Reserved set to 0
4	DWORD - IO address - This is the address in the IO space to be used. It is always a WORD value. If this is a define index/data port operation, this DWORD contains the IO address of the index port. If this is an index operation, other than define, this DWORD contains the value to be written to the index port.
8	DWORD - IO data - The meaning depends on the Operation type: Read - This is where the data read from the IO port will be placed Write- This is the data to write to the IO port. Read/And/Or/Write - This data will be ANDed with the data read from the IO port. Define index/data port - This DWORD contains the IO address of the data port.
12	DWORD - Or Data - This field is only used in a Read/And/Or/Write operation. It contains the data that will be Ored after the data read was ANDed with the previous field. After the OR is done, the data will be re-written to the IO port.

In all cases, if the data called for is shorter than the field, the data will be stored or retrieved from the least significant portion of the DWORD.

3.3.8 0x50 Video Power Control

This command will control the power to the SOFTVGA. If SETUP_DATA is written with a 0, power will be turned off. If a 1 is written, power will be turned on.

3.3.9 0x60 - 0x63 Audio Soft SMI Emulation

Arbitrary registers cannot be set in ASL code before issuing a soft SMI. This command provides a IO interface to allow AUDIO Soft SMIs to be emulated.

SOFT SMI AX	SETUP_INDEX	SETUP_DATA
0x6000	0x60	BP register value
0x6001	0x61	BP register value
0x6002	0x62	BX register value
0x6003	0x63	BX register value

3.3.10 0x64 Audio Power Control

This command will allow control of power to the audio CODEC as well as control of amplifier muting.

SETUP_DATA	Action
0	Power CODEC off and Mute output
1	Power CODEC off, do not mute (allows CD to play)
2	Power CODEC on and un-mute output
3	Power CODEC on only

4 Provided OEM FILES

Files for these chipsets and/or Super IO chips have been written.

4.1 VASMC932.C

This file contains code for the SMC FDC37C932 chip as used on a KESTREL board using a Cyrix 5520. The board must be modified to the latest power control revisions.

POWERBUTTON is routed to 5520 GPIO 1. Power off control is on SMC932 SIOGP15.

Below are descriptions of each routine for this configuration.

4.1.1 oem_acpi_early_init

This routine is called by **acpi_early_init** to perform OEM specific functions that must be done prior to **pm_init**.

For this platform, this routine insures that the SIO GP15 is set to an output, and that SIO GP15 is set **high,** VACPI is configured so that GPIO1 is routed to PWRBTN_STS, triggered on a rising edge and is the power button (for 4 second fail safe delay).

Setting SIO GP15 high on this platform insures that the soft power button will be active without software intervention when ACPI is disabled.

4.1.2 oem_acpi_late_init

This routine is called by **acpi_late_init** to perform OEM specific functions that must be done after **pm_init**.

For this platform, this is a dummy routine that does nothing.

4.1.3 oem_acpi_init

This function is called by the **acpi_init** routine to perform OEM specific functions that are required when the APCI is enabled by writing **ACPI_ENABLE** to the **SMI_CMD** register.

For this platform, this routine insures that the SIO GP15 is set to an output, and that SIO GP15 is set **low**, VACPI is configured so that GPIO1 is routed to PWRBTN_STS, triggered on a rising edge and is the power button (for 4 second fail safe delay).

Setting SIO GP15 low on this platform insures that the soft power button will NOT directly control the Power State. When the power button is pressed, VAPCI will set PWRBTN_STS to inform the OS that the power button has been pressed. The OS will then request SOFT OFF (Sleep State S5) which will actually power off the system.

4.1.4 oem_acpi_de_init

This function is called by the **acpi_de_init** routine to perform OEM specific functions that are required when APCI is disabled by writing **ACPI_DISABLE** to the **SMI_CMD** register.

For this platform, this routine insures that the SIO GP15 is set to an output, and that SIO GP15 is set **high,**

Setting SIO GP15 high on this platform insures that the soft power button will be active without software intervention when ACPI is disabled.

4.1.5 oem_acpi_power_off

This routine is called by the **acpi_power_off** routine to soft power off the system. This routine **MUST** power the system off.

This routine will set SIO GP15 **high**, and then toggle 5520 GPOIO1. This will turn power to the platform off when using an ATX power supply.

4.1.6 oem_acpi_gpio_cascade

This is a dummy routine that always returns zero.

4.1.7 oem_acpi_gpio_event

This is a dummy routine that always returns zero.

4.1.8 oem_acpi_save_to_disk

This is a dummy routine.

4.2 VAMAR317.C

This file contains code for the NS97317 chip as used on a MARMOT board using a Cyrix 5530.

POWERBUTTON is routed to 317 -POWERBUTTON Input.. Power off control is on the 317 -ONCTL line. The 317 -POR Line is connected to the 5530 GPIO0 pin.

Below are descriptions of each routine for this configuration.

4.2.1 oem_acpi_early_init

This routine is called by **acpi_early_init** to perform OEM specific functions that must be done prior to **pm_init**.

For this platform, this is a dummy routine that does nothing.

4.2.2 oem_acpi_late_init

This routine is called by **acpi_late_init** to perform OEM specific functions that must be done after **pm_init**.

This routine establishes the addressability of the 317's ACPI registers by setting the addresses to be used into Logical Device 8 of the 317.

4.2.3 oem_acpi_init

This function is called by the **acpi_init** routine to perform OEM specific functions that are required when the APCI is enabled by writing **ACPI_ENABLE** to the **SMI_CMD** register.

This routine insures that 317 events activate the –POR line.

4.2.4 oem_acpi_de_init

This function is called by the **acpi_de_init** routine to perform OEM specific functions that are required when APCI is disabled by writing **ACPI_DISABLE** to the **SMI_CMD** register.

This routine restores the 317 to its normal state for non-ACPI operation.

4.2.5 oem_acpi_power_off

This routine is called by the **acpi_power_off** routine to soft power off the system. This routine **MUST** power the system off.

This routine will power down the system by issuing a software off command to Logical Device 2 APCR1 register in the 317.

4.2.6 oem_acpi_gpio_cascade

This routine will return a one to GPIO 0 to indicate that it is cascaded to the 317.

4.2.7 oem_acpi_gpio_event

For GPIOs other than 0, a zero is returned to indicate that this routine is not interested in that GPIO.

For GPIO 0 the 317 is polled to see what triggered -POR

For the power button – The fail-safe timer is reset and the PWRBTN_STS bit is set in VACPI's PM1A_STS register.

For the PME line from the PCI bus – VACPI GPE0 Bit 0 is set.

4.2.8 oem_acpi_save_to_disk

This is a dummy routine.