AMD

Reset Software for Alchemy™ Au1000™ Processor from AMD

Application Note

Revision: **1.2**
Issue Date: **August 2002**

Contacts

www.amd.com pcs.support@amd.com

Trademarks

1. Introduction

The Au1000 System-On-a-Chip, SOC, contains integrated memory controllers and peripherals. This document outlines how to properly configure the Au1000 resources out of reset.

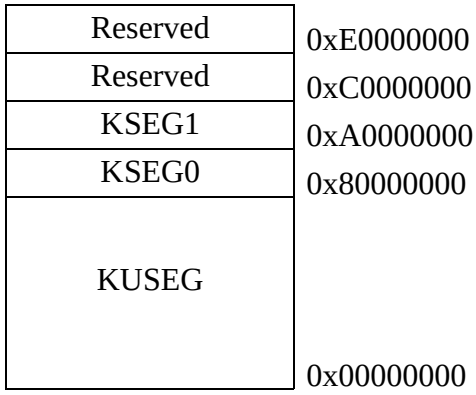
Example source code which demonstrates the initialization of the Au1000 for the Pb1000 reference platform is contained in the file reset_pb1000.S.

2. MIPS Architecture

The basic principles of where to map in a boot ROM are rooted in the MIPS architecture itself. Specifically, the MIPS architecture specifies that upon reset, a MIPS processor must fetch from address 0xBFC00000, the Reset exception vector. [1]

In the MIPS architecture, all addresses (instruction fetches, data loads and data stores) are virtual addresses. As a result, address translation is always performed on program instruction fetches and data accesses. The type of address translation depends upon the upper bits of the program address. The MIPS architecture defines the KUSEG, KSEG0 and KSEG1 regions according to these upper bits of the program's virtual address. The program's 32-bit memory space is thus divided:

Figure 1: MIPS 32-bit Memory Map



The KUSEG region extends from 0x00000000 to 0x7FFFFFFF, a 2GB space which uses translation look-a-side buffers, TLBs, to determine the corresponding physical address. The KUSEG region is accessible while the CPU is in either user mode or kernel mode.

The KSEG0 region extends from 0x80000000 to 0x9FFFFFFF, a 512MB space which has a direct correlation to a physical address. In addition, the KSEG0 region is inherently cacheable; meaning that both instruction and data caching is occuring for references to this area. The KSEG0 region is only accessible while the CPU is in kernel mode.

The KSEG1 region extends from 0xA0000000 to 0xBFFFFFFF, a 512MB space which also has a direct correlation to a physical address. However, the KSEG1 region is inherently non-cacheable; meaning that any instruction or data reference will bypass the cache and directly access physical memory. The KSEG1 region is only accessible while the CPU is in kernel mode.

For the KSEG0 and KSEG1 regions, the corresponding physical address is bits 28:0 of the virtual address with address bits 31:29 zero. That is, KSEG0 and KSEG1 map directly onto the first 512MB of physical memory. For example, KSEG0 address 0x80000000 and KSEG1 address 0xA0000000 both map directly onto physical address 0x00000000. The KSEG0 and KSEG1 regions provide two views of physical memory; one cacheable and one non-cacheable.

Thus, the MIPS architecture Reset exception vector address 0xBFC00000 lies in the KSEG1 region. This provides the MIPS processor a non-cacheable memory space in which to run while memory controllers, caches, TLBs and other system resources are initialized properly before use.

Note: *NOTE: The address translation mechanism of the MIPS architecture always presents a physical address to the memory controllers (and other address decode logic). Thus in the case of the Reset exception vector address 0xBFC00000, the physical address 0x1FC00000 is generated for the first instruction fetch from the boot ROM. Please refer to the separate application note "Mapping a Boot ROM on the Au1000" for additional information [3].*

3. Reset Types

The Au1000 has three forms of reset which all vector thru the MIPS Reset exception vector at address 0xBFC00000. The types are:

- Hardware Reset
- Wake-up from Sleep Mode
- Run-time Reset

Hardware reset is the reset condition that occurs when VDDXOK transitions from low to a high signal value, typically when power is first applied to the Au1000. This is commonly referred as a "cold" reset. At hardware reset, the various registers within the Au1000 are initialized with the default values listed in the Au1000 databook.

A wake-up from Sleep mode reset occurs when software has placed the Au1000 core into the power-savings Sleep mode, and the resulting interrupting event has occured to "wake-up" the Au1000 from Sleep mode. During Sleep mode, only the TOY continues to operate, all other peripherals and the Au1 core cease to operate in order to conserve power.

A run-time reset occurs when the $\overline{\text{RESETIN}}$ signal is asserted some time after a hardware reset occured. This is commonly referred as a "warm" reset. Many registers within the Au1000 are not affected by a run-time reset, while others obtain initial values again.

In all three cases, the Au1000 begins to fetch instructions from the MIPS Reset exception vector at KSEG1 address 0xBFC00000.

Note: *NOTE: The Au1000 can be configured to boot from either the static memory controller (via $\overline{RCE0}$) or the synchronous memory controller (via $\overline{SDCS0}$). The choice is determined by the signals ROMSEL and ROMSIZE at reset. Most of the following discussion applies when booting from synchronous memory devices connected to the synchronous memory controller, though the discussion focuses on booting via static memory controller $\overline{RCE0}$.*

4. Reset Exception Vector

Upon reset, the Status[BEV] bit is set, thus the MIPS exception vector table is located in the KSEG1 region starting at address 0xBFC00000. When Status[BEV]=1, the exception vector table is such:

Table 1. Exception Vector Table with Status[BEV]=1

Exception Type	Virtual Address
Reset	0xBFC00000
TLB refill	0xBFC00200
All others	0xBFC00380
Interrupt	0xBFC00400 (Cause[IV]=1)

The TLB exception vector is at address 0xBFC00200 which permits only 128 instructions for the Reset exception vector. Boot code tends to be larger than 128 instructions, thus, the software designer must choose one of two approaches to overcome this limitation:

- Boot code is as large as necessary and located entirely at 0xBFC00000, but the KSEG1 exception vector table is un-usable, or
- Boot code at 0xBFC00000 performs a jump to the real boot code located outside the MIPS exception vector table, permitting the KSEG1 exception vector table to be usable.

The decision is primarily based in whether the application will run from ROM or RAM. If the application must run from ROM, then it is likely that a valid exception vector table must be located in KSEG1 (i.e. in ROM, and Status[BEV]=1). If the application will run from RAM, then it is likely that the application will create an exception vector table in KSEG0 (i.e. in RAM, and Status[BEV]=0) and therefore the exception vector table located in KSEG1 will not be used except for the Reset exception vector. Most applications run primarily in RAM, but a few applications run entirely from ROM, a debug monitor, for example.

If the application will run from RAM (either the application is copied from ROM into RAM, or it is loaded into memory), then the first approach of permitting the boot code to be as large as necessary and exist at 0xBFC00000 is normally preferred. The only caveat is that the boot code must guarantee that it will not generate an exception of any kind else the processor may fetch exception vector instructions which are in the middle of the boot code and likely result in failure.

If the application is to run from ROM (e.g. a debug monitor), then the second approach of performing a jump to the real boot code located outside the exception vector table is normally preferred. The

main precaution to be aware of is that the Au1000 initially has a 256K window of ROM visible, so the boot code must be contained within this first 256K (i.e. the bootcode is normally placed immediately following the vector table). And again, the boot code must be careful to not generate exceptions unless it is prepared to handle exceptions.

Of course some combination of the two approaches is always possible, where a minimum amount of boot code is located at the Reset exception vector, and then the remainder of the boot code is located outside the Reset exception vector to complete the boot process. Or, for example, only the interrupt exception may be useful in a ROM-based vector table, so the software designer permits the boot code to be located at the Reset exception vector, but limits its size so as not to overlap with the interrupt exception vector.

In most situations, the software designer chooses to link the ROM contents to the KSEG0 region to take advantage of caching. However, when the processor enters the Reset exception vector, it is in the non-cachable KSEG1 region, to allow the boot code to properly initialize the caches before using the caches. In this situation, historically the boot code must take care so that any symbol reference, whether that be a branch or jump target address or a data address must be converted to a KSEG1 address prior to using the address (otherwise the address is a KSEG0, cachable address) and the reference will result in incorrect instructions and/or data since the caches are not ready for use. Fortunately, the Au1000 caches are ready for use out of reset, so the boot code need not take such precautions.

Also, note that a “j target” instruction has a 256MB range, and with respect to the boot code, this means that the upper 4 bits of the address remain unchanged and therefore the boot code stays within the same KSEG0 or KSEG1 region in which is was operating prior to the jump. In otherwords, a “j target” instruction can not be used to crossover between KSEG0 and KSEG1 regions (but a “jr” instruction can).

Nonetheless, whatever the Reset exception vector strategy, the following steps are taken to configure the Au1000 resources out of reset.

5. Common Reset Boot Code Activities

The recommended startup activities for the Au1000 are listed below, in the order given. These activities are common to all three types of reset.

NOTE: If booting from a 16-bit device, then special startup steps are needed. Please see the applications note “Au1000 Bus Operation” for additional details [4].

5.1 Establish CPU Endian Mode

In most applications, it is important to establish the CPU endian mode very early in the boot process, and usually as the first activity. The endian mode on the Au1000 is software selectable, and defaults to big endian upon reset. In most applications, the endian mode is determined in one of the following ways:

- The endian mode is selected by a switch/jumper that is readable by the CPU, and then software changes endian accordingly.
- The endian mode is pre-determined by the software designer, and the application sets the endian mode upon boot.

Regardless of the method, the endian mode affects the interpretation of the contents of the boot ROM. 32-bit words (e.g. instructions) are interpreted the same whether the CPU is in little or big endian, but 8- and 16-bit values in ROM are affected by the endian mode. For this reason, it is important that the endian mode be established before accessing data items. The application should be compiled using the desired endian mode (dual-endian support requires both big- and little-endian versions of the application).

The Au1000 defaults to big-endian operation upon reset. The Au1000 can be changed to little-endian by executing the following code sequence, as early as possible in the boot sequence:

```
li   t0, 0xB1900000
li   t1, 1
sw   t1, 0x0038(t0) # sys_endian register
mfc0 t2, CP0_Config
mtc0 t2, CP0_Config # endian change takes effect
nop
nop
```

For additional information on endian support, including 16-bit boot devices, see the application note “Au1x00 Bus Operation” [4].

5.2 Establish the Status Register

The Status register should be written to establish the operating mode of the processor. Typically, the Status[BEV] is set (to keep the exception vector table located in KSEG1), Status[ERL] and Status[EXL] are for “normal” operation, and Status[IM] and Status[IE] are cleared to ensure interrupts are disabled during boot-up. Writing the value 0x00400000 to CP0 Status accomplishes this.

Typically once the application has created an exception vector table at KSEG0 0x80000xxx, the Status[BEV] bit is then cleared to permit the processor to fetch exception vector instructions from KSEG0.

5.3 Establish Config0

The K0 field in the CP0 Config0 register determines the cache coherency attributes for the KSEG0 region. This field defaults to CCA 3, cachable, coherent, but as a matter of good practice it should be set to the desired value, as appropriate for the application. (The value of CCA 3 for the K0 field is both historical and usually the correct value to use for most applications.)

Also, to overcome some issues with early Au1000 silicon, the Config0[OD] bit must be set. See the “Au1000 Specification Update” for details [5]. NOTE: In early version of the Au1000 silicon, the

Config0[OD] bit is a write-only bit, so any subsequent changes by software to the Config0 register must also set Config0[OD].

5.4 Disable the Watch Facilities

The Watch and IWatch facilities should be disabled upon boot to prevent unwanted watchpoint exceptions. Writing the value zero to both WatchLo and IWatchLo CP0 registers disables the watchpoint resources.

5.5 Disable the Performance Counters

The application may choose to disable the performance counters, especially if the application is not being debugged or profiled for performance. By disabling the performance counters, power consumption is further reduced.

5.6 Establish the EJTAG Debug Register

The EJTAG Debug register should be written to ensure proper operation of the processor. Typically this register is written with a zero to effect normal operation, but the application may choose another value during the debug phase of the application.

5.7 Establish the Cause Register

The Cause register is typically written so that Cause[IV] is set to permit the interrupt exception vector to be located at its own, unique exception vector address. With Cause[IV] set, the processor fetches from KSEG0 0x80000200 or KSEG1 0xBFC00400, depending upon the setting of Status[BEV]. Otherwise, with Cause[IV] cleared, the interrupt exception is grouped with the other exceptions and must be decoded via the Cause[ExcCode] field.

Obviously, setting Cause[IV] to permit the processor to fetch from an interrupt exception reduces the amount of time necessary for the application to determine the interrupt cause and service it [2].

5.8 Initialize the Instruction and Data Caches

The Au1000 caches are ready for use upon reset. It is good practice to ensure caches are invalidated should any data and/or instructions be left over from a previous execution of the application (i.e. a run-time reset).

5.9 Initialize the TLB

The Au1000 TLB is not invalidated out of reset, so software must ensure the TLB is invalidated to prevent the TLB from containing any valid mappings. A random valid mapping could create quite an elusive software bug.

5.10 Establish the CPU Core Operating Frequency

The sys_cpupll register is to be programmed with the multiplier to achieve the desired operating frequency of the Au1000 CPU core. Setting the core frequency early is important since many memory controller values are derived from the CPU frequency.

NOTE: The CPU core frequency defaults to 192MHz at reset (assuming the recommended 12MHz input oscillator).

NOTE: A CPU write to the sys_cpupll register stalls the CPU until the PLL regains lock, a maximum of 20us.

5.11 Establish the System Bus Frequency

The system bus, SBUS, frequency is derived from the CPU core frequency. The register sys_powerctrl contains the divisor for the system bus operating frequency. Setting the system bus frequency divisor early is important as many memory controller and peripheral timings are derived from the system bus frequency.

NOTE: The system bus frequency defaults to the CPU core frequency divided by two.

5.12 Establish the Auxiliary PLL Frequency

The auxiliary PLL contains the frequency generators that are used for many peripherals. The register sys_auxpll contains the multiplier that determines the main clock reference for many integrated peripherals. The auxiliary PLL needs time to gain PLL lock.

5.13 Start the 32kHz Oscillator

The 32kHz oscillator which is used for the RTC/TOY needs to be enabled. It is best to enable the oscillator early in the boot sequence to give the oscillator time to start before it is needed.

5.14 Initialize the Static Memory Controller

The Au1000 static memory controller must be initialized to establish the memory map and more importantly correctly place and size the boot ROM (see “Mapping a Boot ROM on the Au1000” applications note [3]). The static memory controller does not retain its value across the various resets, so it must be initialized everytime upon reset.

NOTE: The SDRAM may be in a Sleep mode, so the synchronous memory controller is not initialized until the reset cause has been determined.

5.15 Initialize Integrated Resources to Known State

Out of reset, set the integrated resources of the Au1000 to a known, usually disabled and/or safe state. This prevents peripherals from operating undesirably (if enabled prior to the run-time reset) and

consuming power un-necessarily. All integrated resources should be set to a known state as a matter of good practice.

- Frequency Generators (sys_freqctrl0, sys_freqctrl1, sys_clksrc)
- GPIO (sys_pininputen)
- AC97 (ac97_enable)
- USB Host (usbh_enable)
- USB Device (usbd_enable)
- IrDA (ir_enable)
- UARTs (uart_enable)
- MACs (macen_enable)
- I2S (i2s_enable)
- SSIs (ssi_enable)

Generally these peripheral modules should be set to a known state, usually disabled. In particular, the clocks to these modules should be disabled so that power consumption is reduced.

5.16 Determine the Cause of Reset

The type of reset is recorded in the sys_wakesrc register. Depending upon the type of reset, the boot code should follow either an Initial Power-on Reset code path, a Wake from Sleep Mode code path, or a Run-time Reset code path.

6. Hardware Reset Code Path

When an hardware reset is detected, the boot code should perform the following tasks, in the order given.

6.1 Clear the sys_wakesrc Register

The sys_wakesrc[SW,IP] bits indicate the reset type and must be cleared by software. Once cleared by software, the next reset event type is successfully detected by software and the appropriate code path followed.

NOTE: This step can be ommitted if the application needs to know the reset type as well. The application must read and then clear the sys_wakesrc register.

6.2 Initialize the SDRAM Controller

The SDRAM must now be initialized for correct operation. The values used to initialize the SDRAM controller are dependent upon the SDRAM devices in use.

NOTE: The initialization sequence for SDRAM is different for power-on reset than it is for resuming from sleep mode.

6.3 Invoke the Application

At this point, with the Au1000 resources set to a known state and the memory controller configured, the application should be invoked. Typically the application will copy data (and/or itself) from ROM to RAM, initialize external peripherals to known states, and begin the application activities.

7. Run-time Reset Code Path

Since the system has already performed all the common activities in “Common Reset Boot Code Activities” on page 6, the additional run-time reset activities are quite minimal. The remaining activities are listed below.

The run-time reset sequence can be optimized, if necessary, since some resources do retain values across the reset.

7.1 Clear the sys_wakesrc Register

The sys_wakesrc[SW,IP] bits indicate the reset type and must be cleared by software. Once cleared by software, the next reset event type is successfully detected by software and the appropriate code path followed.

NOTE: This step can be ommitted if the application needs to know the reset type as well. The application must read and then clear the sys_wakesrc register.

7.2 Initialize the SDRAM Controller

The SDRAM must now be initialized for correct operation.

NOTE: The initialization sequence for SDRAM is different for power-on reset/run-time reset than it is for resuming from sleep mode.

7.3 Invoke the Application

At this point, with the Au1000 resources set to a known state and the memory controller configured, the application should be invoked. Typically the application will copy data (and/or itself) from ROM to RAM, initialize external peripherals to known states, and begin the application activities.

8. Wake-up from Sleep Mode Code Path

The activities for the wake-up from Sleep mode are interesting in that they prepare the system to resume after where it previously went into Sleep mode. Typically the SDRAM already contains the application and/or data and must be re-started to resumed where it left off.

Generally, the steps needed to place the system into Sleep mode are:

- Gracefully shut-down appropriate external peripherals.

- Gracefully shut-down all internal peripherals (only the TOY continues to operate in Sleep mode).
- Save the CPU general purpose register state (optional, but usually necessary)
- Setup the event which allows the Au1000 to exit Sleep mode (GPIO or M20).
- Save the interrupt controller settings into SDRAM (optional)
- Store a pointer to the resume-from-Sleep startup code into either sys_scratch0.
- Issue SDRAM “auto refresh” command.
- Issue SDRAM “precharge all” command.
- Place the remainder of this sequence into the instruction cache
- Issue another SDRAM “auto refresh” command.
- Issue SDRAM “self-refresh” command; SDRAM is now no longer accessible
- Prepare Au1000 for Sleep mode by writing sys_slppwr
- Enter Sleep mode by writing sys_sleep.

In many ways, the wake-up code must “undo” the steps taken to put the system into sleep mode.

8.1 Clear the sys_wakesrc Register

The sys_wakesrc[SW,IP] bits indicate the reset type and must be cleared by software. Once cleared by software, the next reset event type is successfully detected by software and the appropriate code path followed.

NOTE: This step can be ommitted if the application needs to know the reset type as well. The application must read and then clear the sys_wakesrc register.

8.2 Wake-up the SDRAM

The SDRAM is placed into its own Sleep mode prior to the Au1000 entering Sleep mode. The SDRAM controller must be initialized to bring the SDRAM out of Sleep. The sequence for placing SDRAM into its sleep mode, and for bringing it out of its sleep mode is device dependent. Consult the datasheets for the SDRAM used.

8.3 Invoke the Pointer to the Resume Code

Now that SDRAM is enabled and usable again, jump to the code which is pointed by the contents of the sys_scratch0. Then, this code should perform the following tasks:

- Restore interrupt controller settings (optional)
- Re-start any external peripherals
- Re-start all appropriate internal peripherals
- Restore the CPU general purpose register state
- Resume application activities

Obviously the sequence is application specific, but the above steps are a general outline of the activities necessary to enter and exit Au1000 Sleep mode.

9. Miscellaneous

The following are possibilities with the Au1000 and the reset software.

- To facilitate dynamic frequency changing, the desired CPU core frequency could be contained in the unused sys_scratch1 register, and the reset software could examine this register to determine the new operating frequency. For example, the application can write the desired value to sys_scratch1 and then toggle a GPIO pin to assert $\overline{\text{RESETIN}}$ or setup, enter and exit sleep to change to the new desired operating frequency.

10. References

[1] “The Alchemy™ Au1000™ Processor from AMD Data Book”, AMD, 2002.

[2] “MIPS32™ Architecture for Programmers”, MIPS Technologies, Inc., 2001.

[3] “Mapping a Boot ROM on the Au1000”, AMD, 2002.

[4] “Au1000 Bus Operation”, AMD, 2002.

[5] “Au1000 Specification Update”, AMD, 2002.