AVR32739: AVR32 UC3 Low power software design

Features

- **Pin configurations**
- **Using sleep modes**
- **Using the power manager**
  - **Switching the main clock**
  - **Clock masking**
  - **Clock scaling**
- **Software considerations**

1 Introduction

Reducing the power consumption of a microcontroller in a “Low Power Software Design” demands an in-depth knowledge of the device and the application design.

This application shall give a developer an overview of available features on the UC3A and UC3B devices that help to decrease power consumption. Most parts of the application note are also applicable for other AVR®32 devices.

32-bit AVR® Microcontrollers

Application Note

Rev. 32093B-AVR32-05/08

2 Pin configurations

After a reset all multiplexed pins of the device are configured as inputs. Depending on the design some pins are then configured as outputs, inputs with internal pull-up enabled or dedicated to a peripheral function. This has an impact on the power consumption of the device in the different power save modes.

To easily measure the power consumption due to the pin settings do following measurements:

1. Set all pins to inputs, enable all internal pull-ups and measure the power consumption.
2. Set the pins to the default configuration of the application and measure the power consumption.

This measurement is especially meaningful if it is done in one of the power save modes (sleep modes) because in these states also very low extra power consumption is directly visible.

The set the device into the lowest possible power mode all pins should be configured as inputs with pull-ups enabled followed by entering the “static” mode. More information about the different sleep modes is available in chapter 3.

2.1 Not connected pins

All pins that are not connected externally to pull-ups, pull-downs, ground or power should be left as inputs but with the internal pull-up enabled. This ensures the lowest possible power consumption.

3 Sleep modes

3.1 Introduction

In normal operation, all clock domains are active, allowing software execution and peripheral operation. When the CPU is idle, it is possible to switch off the CPU clock and optionally other clock domains to save power.

3.2 Functional description

3.2.1 Sleep mode instruction

A sleep mode is activated by the sleep instruction, which takes the sleep mode index number as argument. This sleep instruction is defined in the AVR32 architecture document as follows:

SLEEP - Set CPU Activity Mode

Architecture revision:

Architecture revision1 and higher.

Description

Sets the system in the sleep mode specified by the implementation defined Op8 operand. The semantic of Op8 is IMPLEMENTATION DEFINED.

Operation:

I. Set the system in the specified sleep mode.

Syntax:

I. sleep Op8

Operands:

I. Op8 ϵ {0, 1, …, 255}

Status Flags:

Q: Not affected

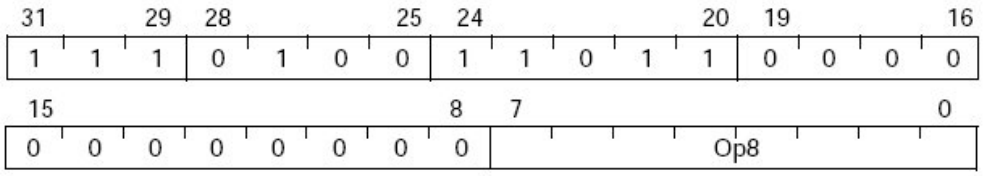
V: Not affected

N: Not affected

Z: Not affected

C: Not affected

Opcode:



Note: The sleep instruction is a privileged instruction, and will trigger a Privilege Violation exception if executed in user mode.

The operand is device specific as not all devices may support the same sleep modes. Take a look at the device datasheet for supported sleep modes and valid operand values.

Entering sleep mode in C-code

The UC3 software framework provides a wrapper for the sleep instruction. Include the power manager driver in you software design and enter a sleep mode with:

SLEEP(AVR32_PM_SMODE_IDLE);

Valid sleep mode macros are in the power manager driver header file.

The wrapper turns the above line into an inline assembly instruction.

__asm__ __volatile__ ("sleep 0");

3.2.2 Entering and exiting sleep modes

The sleep instruction will halt the CPU and all modules belonging to the stopped clock domains. The modules will be halted regardless of the bit settings of the mask registers (see chapter 4.2 for more details about clock masking).

Oscillators and PLLs can also be switched off to save power. Some of these modules have a relatively long start-up time, and are only switched off when very low power consumption is required.

The CPU and affected modules are restarted when the sleep mode is exited. This occurs when an interrupt triggers. Note that even if an interrupt is enabled in sleep mode, it may not trigger if the source module is not clocked.

3.2.2.1 Supported sleep modes

The behavior of the device upon entering a sleep mode is implementation defined. Therefore a look into the datasheet of the device is necessary to obtain the correct specification. Table 3-1 lists the available sleep modes of the UC3A series and may serve as an overview for all other devices as their behavior is close to this.

Table 3-1 UC3A sleep modes

Sleep Mode	Description	Wake-up source
Idle	The CPU is stopped, the rest of the chip is operating.	Any interrupt
Frozen	CPU and HSB modules are stopped, peripherals are operating.	Any interrupt from PB modules
Standby	All synchronous clocks are stopped, but oscillators and PLLs are running, allowing quick wake-up to normal mode.	RTC or external interrupt
Stop	As Standby, but Oscillator 0 and 1, and the PLLs are stopped. 32 KHz (if enabled) and RC oscillators and RTC/WDT still operate.	RTC, external interrupt or external reset pin
DeepStop	All synchronous clocks, Oscillator 0 and 1 and PLL 0 and 1 are stopped. 32 KHz oscillator can run if enabled. RC oscillator still operates. Bandgap voltage reference and BOD is turned off.	RTC, external interrupt or external reset pin
Static	All oscillators, including 32 KHz and RC oscillator are stopped. Bandgap voltage reference BOD detector is turned off.	External interrupt in asynchronous mode or external reset pin

3.2.2.2 Precautions when entering sleep mode

Modules communicating with external circuits should normally be disabled before entering a sleep mode that will stop the module operation. This prevents erratic behavior when entering or exiting sleep mode. Please refer to the relevant module documentation for recommended actions.

Communication between the synchronous clock domains is disturbed when entering and exiting sleep modes. This means that bus transactions are not allowed between clock domains affected by the sleep mode. The system may hang if the bus clocks are stopped in the middle of a bus transaction.

The CPU is automatically stopped in a safe state to ensure that all CPU bus operations are complete when the sleep mode goes into effect. Thus, when entering Idle mode, no further action is necessary.

When entering a sleep mode (except Idle mode), all HSB masters must be stopped before entering the sleep mode. Also, if there is a chance that any PB write operations are incomplete, the CPU should perform a read operation from any register on the PB bus before executing the sleep instruction. This will stall the CPU while waiting for any pending PB operations to complete.

3.2.3 Wake-up examples

3.2.3.1 Wake-up by an internal interrupt

Depending on the sleep mode an internal interrupt can force the device to leave the sleep mode. This is very useful for the USB module. This module has various interrupts that can be used to wake-up the device. It is possible to wake-up when data is transferred on the USB bus or when device is plugged into the USB connector.

3.2.3.2 Wake-up with the RTC

The RTC has an additional wake-up feature that makes it possible to leave sleep modes where the internal interrupt controller is disabled. The RTC wake-up needs therefore no ISR and after a wake-up the code is executed right after the sleep instruction. By using the RTC a dedicated sleep time is possible. The example application “rtc_wakeup_example” included with this application note shows the usage of the RTC wake-up. More information is available in the source code documentation.

3.2.3.3 Wake-up from Static sleep mode

A wake-up of the device from Static sleep mode is only possible by an asynchronous interrupt on the external interrupt controller or by a reset. The example application in DRIVERS/EIC/EXAMPLE1 in the software framework shows how this can be implemented with an external interrupt. More information is available in the source code documentation.

3.2.4 Conclusion

Using sleep modes instead of polling or endless loops is very effective in reducing power consumption. The different wake-up methods provide a broad spectrum from that a developer can choose a method that fits his design best.

4 Power manager

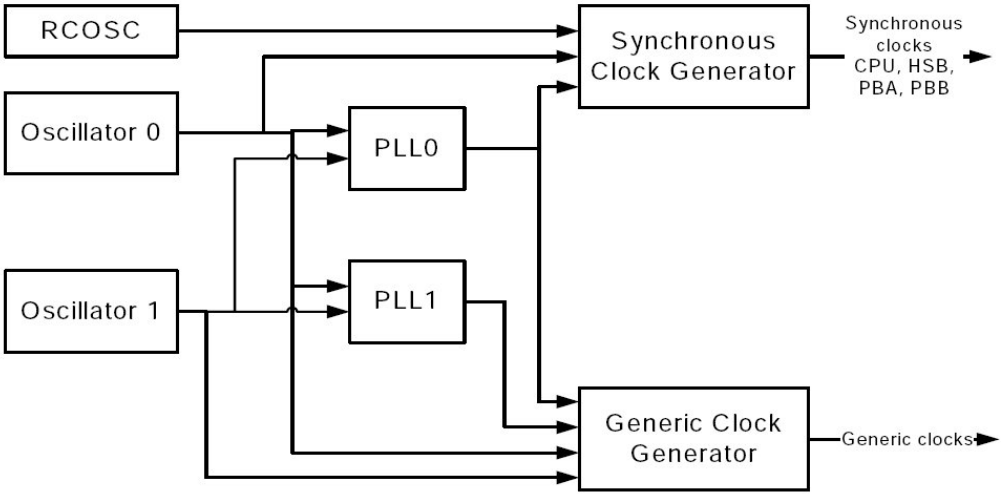
The power manger is an important part in the goal of reducing the power consumption. The use of this module is therefore recommended to gain of several power saving features. The power manager handles among other things the sleeping mechanism described in chapter 2. A driver and some examples for this module is available in the software framework in the directory DRIVERS/PM.

4.1 Switching the main system clock

4.1.1 Introduction

The synchronous clock generator can be clocked from one of three sources as described in Figure 4-1 and this clock is named “main clock”.

Figure 4-1 Input sources for the synchronous clocks



These sources are the internal RC-Oscillator, the internal PLL0 and the external Oscillator 0. By default, the main clock will be connected to the internal RC-Oscillator. It is possible to switch the input source of the synchronous clock generator on-the-fly. This feature makes it possible to adjust the system performance according to the current requirements by switching the input clock. This advance is also a power saving feature.

4.1.2 Functional description

The input clock for the synchronous clock generator is selectable in the Main Clock Control (MCCTRL) register in the MCSEL bit-field. This must only be done after that clock source has been enabled, otherwise a deadlock will occur. Care should also be taken that the new frequency of the synchronous clocks does not exceed the maximum frequency for each clock domain. This data is available in the “Electrical Characteristics” chapter of the datasheet.

Be advised that switching the main clock has also an impact on all modules connected to the peripheral bus. The baud rate of the USART module, for instance, needs to be recalculated upon switching of the bus clock.

4.1.2.1 Cautionary note

When switching to high frequency clocks a wait state for the Flash is needed. The maximum operating frequency for each wait state setting is available in the datasheet in the “Electrical Characteristics” chapter. The software framework provides an example application in the directory DRIVERS/PM/EXAMPLE2 that sets up a PLL and configures a wait state for Flash access.

4.1.3 Using the clock switching feature

The example application in DRIVERS/PM/EXAMPLE1 in the software framework shows how the main clock can be switched. In the example the device starts up with the internal RC oscillator and switches then to the external oscillator. In addition an output is generated from a generic clock that is fed from the same external oscillator.

Another example for clock switching is available in DRIVERS/PM/EXAMPLE2. In this example the system switches first to the external oscillator, then sets up a PLL and switches to it.

Take a look at the source code documentation for more information about the example application.

4.1.4 Conclusion

Switching the main clock of the system is quickly done and but needs extra caution for modules that use this clock to generate a fixed frequency or data rate. Depending on the developers system design it may be better to scale the clock instead of switching it. Clock scaling is described in chapter 4.3.

4.2 Peripheral clock masking

4.2.1 Introduction

By default, the clocks for all modules are enabled, regardless of which modules are actually being used. Because of that it is essential to disable a module clock if it is not used in order to reduce power consumption. A list of current consumption for each module is available in the “Electrical Characteristics” chapter under the title “Power Consumption” in the device datasheet. The table in that chapter describes the power consumption of a peripheral in active mode on a per MHz basis. This data can be used in power consumption estimations.

4.2.2 Functional description

It is possible to disable the clock for a module in the CPU, HSB, PBA, or PBB clock domain by writing the corresponding bit in the Clock Mask register (CPU/HSB/PBA/PBB) to 0. The register content is implementation defined because its content depends on the available on-chip modules. When a module is not clocked, it will cease operation, and its registers cannot be read or written. The module can be re-enabled later by writing the corresponding mask bit to 1. A module may be connected to several clock domains, in which case it will have several mask bits. A good overview over which module is clocked from which clock domain (or in other words, which module is connected to which bus) is available in the device block diagram in the datasheet.

4.2.2.1 Cautionary note

Note that clocks should only be switched off if it is certain that the module will not be used. Switching off the clock for the internal RAM will cause a problem if the stack is mapped there. Switching off the clock to the Power Manager (PM), which contains the mask registers, or the corresponding PBx bridge, will make it impossible to write the mask registers again. In this case, they can only be re-enabled by a system reset.

4.2.2.2 Clock synchronization delay

Due to synchronization in the clock generator, there is a slight delay from a mask register is written until the new mask setting goes into effect. When clearing mask bits, this delay can usually be ignored. However, when setting mask bits, the registers in the corresponding module must not be written until the clock has actually be re-enabled. The status flag MSKRDY in ISR provides the required mask status information. When writing either mask register with any value, this bit is cleared. The bit is set when the clocks have been enabled and disabled according to the new mask

setting. Optionally, the Power Manager interrupt can be enabled by writing the MSKRDY bit in IER.

4.2.3 Using the clock masking feature

The clock masks for the different clock domains on the AVR32 UC3A devices look as described in Table 4-1. For other devices the modules listed in their according clock masks may vary depending on the available on-chip modules. Some modules are connected to two clock domains. This is visible in Table 4-1 for the USBB module. Therefore two clocks must be disable to disable the module completely. For a detailed description about the different clocks take a look at the module specific chapter in the datasheet.

Table 4-1 Maskable module clocks in the AT32UC3A

Bit	CPUMASK	HSBMASK	PBAMASK	PBBMASK
0	OCD Clock	FLASHC	INTC	HMATRIX
1	OCD	PBA bridge	GPIO	USBB
2	-	PBB bridge	PDCA	FLASHC
3	-	USBB	PM/RTC/EIC	MACB
4	-	MACB	ADC	SMC
5	-	PDCA	SPI0	SDRAMC
6	-	EBI	SPI1	-
7	-	-	USART0	-
8	-	-	USART1	-
9	-	-	USART2	-
10	-	-	USART3	-
11	-	-	PWM	-
12	-	-	SSC	-
13	-	-	TC	-
31:15	-	-	-	-

In order to disable a module the according clock mask bit in the register CPU/HSB/PBA/PBBMASK must be cleared. To disable USART0, for instance, the bit USART0 has to be cleared in the register PBAMASK. The on-chip debug system can be disabled to reduce power consumption if the need for debugging the device is not necessary any more. This is the case when a product is ready for production.

The example application “clock_masking_example” included with this application note shows the use of the clock masking feature on the EVK1100 board. Use the tree buttons on the board to enable/disable module clocks. A detailed description is included in the source code documentation.

4.2.4 Conclusion

The clock masking is a powerful feature to reduce power consumption. The developer should consider disabling all on-chip modules that are not being used in an application. This may be done in advance to disable modules that are not used at all or during runtime to disable a module in periods where it is not needed. Modules can also be disabled by the sleep instruction regardless of the mask settings. This is described in chapter 2.

4.3 Clock scaling

4.3.1 Introduction

Modules on the peripheral busses may not need their bus to run at the maximum speed. This is the case when a low data bandwidth is expected on the bus or some peripherals connected to the bus are not used at all. This depends of course on the design the developer has in mind with the device. If the bus bandwidth utilization is low it is possible to decrease the bus clock and thus saving power.

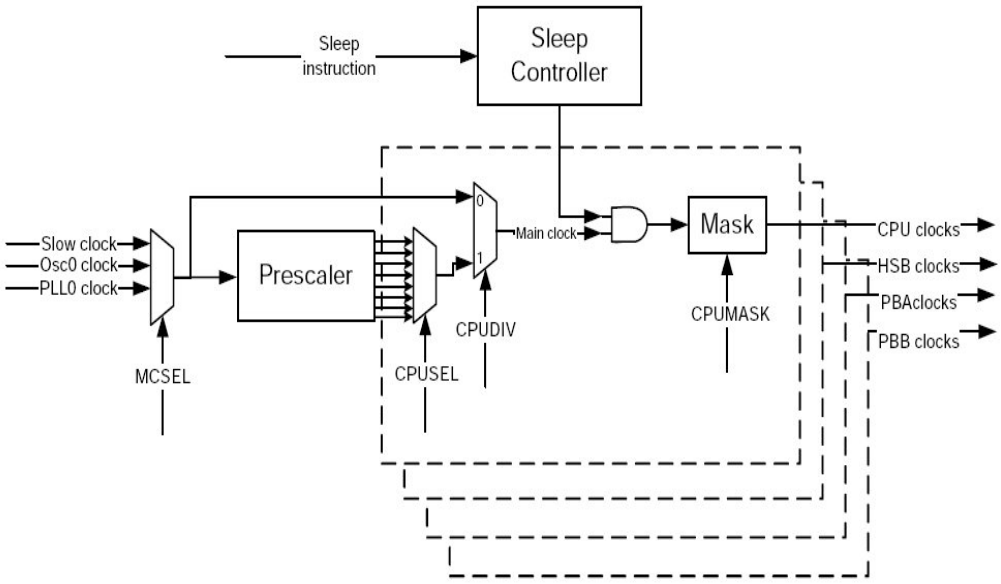
In contradiction to the above mentioned circumstances it is often the case that peripherals do a lot of data transfers without much CPU intervention or processing. An example for such a case is when peripherals make use of the DMA controller. As the CPU does not need to do much work it is possible to decrease its clock speed and thus reduces the power consumption.

4.3.2 Functional description

Adjusting clock speeds and thus the bandwidth of the busses or the performance of the CPU is possible in the power manager. In the register CLKSEL a prescaler is configurable for each of the four synchronous clocks CPU, HSB, PBA and PBB. Actually only three of them are configurable because the HSB clock is coupled to the CPU clock on the UC3 series. By default, the synchronous clocks run on the undivided main clock.

The main clock can be divided by an 8-bit prescaler, and each of these four synchronous clocks can run from any tapping of this prescaler, or the undivided main clock, as long as $f_{CPU} \geq f_{PBA,B}$,. Figure 4-2 shows an overview of the synchronous clock generator.

Figure 4-2 Synchronous clock generation



CKSEL can be written without halting or disabling peripheral modules. Writing CKSEL allows a new clock setting to be written to all synchronous clocks at the same time. It is possible to keep one or more clocks unchanged by writing the same value a before to the xxxDIV and xxxSEL bitfields. This way, it is possible to e.g. scale CPU and

HSB speed according to the required performance, while keeping the PBA and PBB frequency constant.

4.3.2.1 Clock ready delay

There is a slight delay from the moment CKSEL is written and the new clock setting becomes effective. During this interval, the Clock Ready (CKRDY) flag in ISR will read as 0. If IER:CKRDY is written to 1, the Power Manager interrupt can be triggered when the new clock setting is effective. CKSEL must not be re-written while CKRDY is 0, or the system may become unstable or hang.

4.3.3 Using the clock scaling

The example application “cpu_frequ_scaling_example” included with this application note shows the usage of the clock scaling. It adjusts the CPU clock (and HSB clock as it is coupled to the CPU clock) according to the buttons the user presses. More detailed information is available in the source code documentation of the example application.

4.3.4 Conclusion

As described in the previous chapters, the clock scaling may be used to adjust performance to the system requirements. The needed scaling can be configured as a static setup or on-the-fly to follow changing performance needs. Thus a system that is using this feature consumes only that much energy as it really needs.

5 Software considerations

5.1 Introduction

Power saving features of a device are useless, or even the cause of malfunctions, unless the user makes proper use of them. It makes a big difference in the achieved power savings when a developer implements the right power saving features at the appropriate places in his software design.

5.2 Using sleep modes: Tips & Tricks

Listed below are some tips and tricks regarding the usage of sleep modes in an application. Take a look at them and reconsider where they could be useful in your application.

- In order to make use of the sleep modes the development system must be event driven. Basically this means do not use polling wherever this is possible because it prevents you from going to sleep. Use interrupts instead of polling to be able to sleep until an event occurs.
- The easiest way to go to sleep (easy because no additional actions to the sleep instruction have to be taken) is to use the Idle sleep mode. This will only turn off the CPU but lets the rest of the system running. Use this sleep mode extensively whenever your application is idle and has to wait for an event. After a wake-up from an event the according interrupt service routine is executed(if available for the event as it is possible for external interrupts to wake-up the application without running a ISR) and after that the code after the sleep instruction is executed.
- Often it is the case that developers use some kind of counting loop to delay code execution. Instead of using such a loop the use of the sleep instruction in

combination with a timer would it make possible to let the CPU sleep instead of counting.

- Entering the other sleep modes takes more caution because the system busses are affected and it needs to be made sure that no current data transfer is ongoing or will occur during the sleep mode. The benefit of the “deeper” sleep modes is of course a lower power consumption and makes this effort worth while. A save way to enter a sleep mode “below” the Idle mode can be done as follows:
  1. Stop all HSB masters to avoid data transfers during sleep on the bus
  2. Read out any register on the PB bus to make sure that no write operation is ongoing on the bus because the CPU will stall while waiting for any pending PB operations to complete.
  3. When disabling clocks for modules that communicate with external circuits, the module itself should be disabled first. This prevents erratic behavior when entering or exiting sleep mode.

  After these steps it is save to enter the other sleep modes. These sleep modes are adequate for “longer periods” of sleeping.

5.3 Interrupts

Keeping interrupts as short as possible results in shorter times intervals in which the system is awake. Only the most important things should be implemented in an interrupt service routine and all other processing should be done in the main code. Data processing that needs to be done after an interrupt service routine has occurred can be done at a later time point and maybe also at another CPU speed.

5.4 Adjusting system performance

As described in chapter 4.3 the system performance can be adjusted to the current needs. This can be done either from start, according to performance estimations, or on-the-fly on a running system to adjust the performance to the current condition.

The clock speed of the peripheral busses should be estimated during prototyping by calculating the needed bandwidth and testing the settings on the hardware. The adjustment of the PB on-the-fly is not recommended as all clock settings for the modules on the bus need to be re-calculated. For instance the SPI and USART modules derive their communication speed from the PB clock und will therefore not work upon a bus clock change without a new configuration.

The CPU and the HSB are better candidates for on-the-fly clock adjustments to meet the current performance needs. Use this feature when sleeping is not an option and not the full performance is needed either.

6 References

UC3A datasheet:

http://www.atmel.com/dyn/resources/prod_documents/doc32058.pdf

UC3B datasheet:

http://www.atmel.com/dyn/resources/prod_documents/doc32059.pdf

AVR32 architecture manual:

http://www.atmel.com/dyn/resources/prod_documents/doc32000.pdf

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
Avr32@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature