APPLICATION NOTE

AT11487: Low Power Consumption Techniques for XMEGA XPLAINED Kits

Atmel AVR XMEGA

Introduction

This application note explains the techniques for achieving the lowest possible power figures for XMEGA® Xplained kits. In this application note XMEGA-A3BU Xplained board is used as a reference kit and Atmel-ICE is used as programmer to program the target device. In addition, the application note provides code examples for entering various sleep modes and provides the value of current consumed by the XMEGA-A3BU Xplained board in different sleep modes.

This application note also explains how current consumption of the device can be reduced while a peripheral (ADC is used as reference) is running.

Table of Contents

1 Factors Affecting Power Consumption

Though many factors like operating voltage, frequency of operation, etc. affect the power consumption, some will naturally affect more than others. Listed below are the important factors with recommendations and considerations.

1.1 Operating Voltage

The power consumption is proportional to the square of the device's supply voltage, which should therefore be kept as low as possible.

A reduction in supply voltage can lower the limit of maximum system clock frequency, thus increasing the time required in ACTIVE mode to execute a given amount of code.

Minimize power consumption by using supply voltage as low as possible.

1.2 Active Mode Operation

In ACTIVE mode, i.e. when sleep modes are not used, the power consumption is proportional to the system clock frequency. This means that if sleep modes are not used, the device should be run at the lowest possible system clock frequency to minimize the power consumption.

Minimize power consumption by keeping the clock frequency as low as possible if sleep modes are not used.

1.3 Clock Prescaling

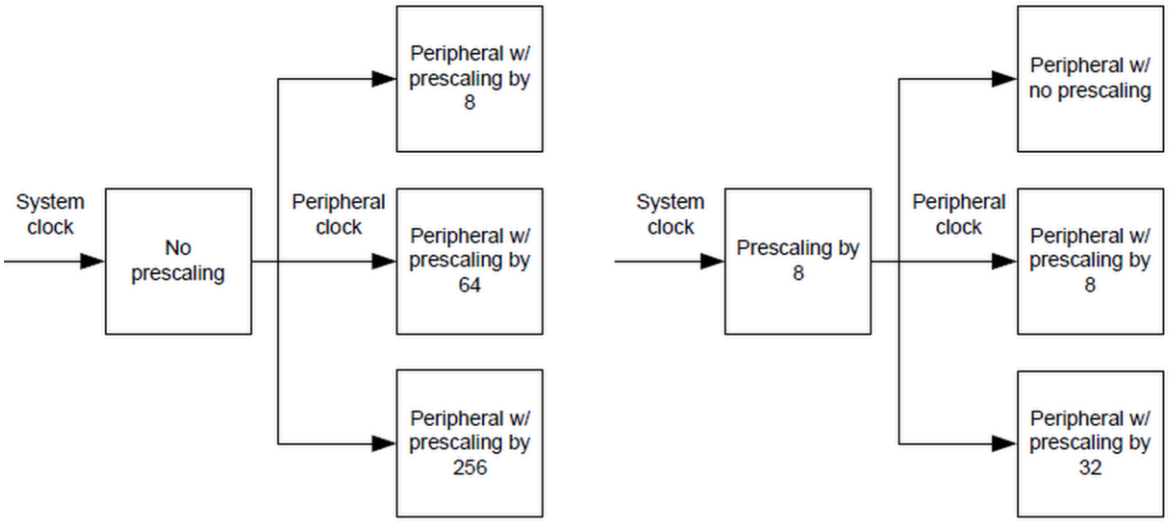
Although it is recommended to run the CPU as fast as possible to minimize the time spent in ACTIVE mode, there are situations where it is better to reduce the clock rate. These situations commonly involve waiting in ACTIVE or IDLE mode for code that takes a fixed amount of time e.g. serial communication. In these cases, one should avoid generating higher CPU and peripheral clock frequencies than are needed for the active peripherals. This can be achieved by using clock prescaler, which can be changed without causing glitches in the clock signal.

If prescaling is done inside the peripherals power consumption can be conserved by prescaling with the largest common factor in the first available prescaler in the clock distribution chain. This principle is illustrated in Figure 1-1.

Note: Since the prescaling also affects the CPU clock, it might not always be desirable to perform this common prescaling in ACTIVE mode because computations will take longer.

Minimize power consumption by effectively using prescaler especially when waiting in ACTIVE or IDLE mode.

Figure 1-1. Peripherals without and with Common Prescaling by Largest Factor



1.4 Clock Source Selection

One should avoid generating higher system clock rates than are actually needed. In the ideal case, prescaling is unnecessary. This can be achieved by selecting appropriate clock sources.

As an example, it is preferable to generate a 16MHz system clock by use of the PLL with the 2MHz RC oscillator as reference, rather than the 32MHz RC oscillator with prescaling to 16MHz. External clock sources may also be a good choice, especially if they are already available in the system and therefore come with no extra “cost”.

The wake-up delay for the device depends on which clock source is used for the system clock. One way to reduce this delay is to switch between clock sources so that the device goes to sleep and wakes up with a fast-responding clock source.

Minimize power consumption by switching clock sources rather than relying on prescaling alone for reducing clock rates.

1.5 Wake-Up Delays

When the device wakes up from sleep modes deeper than IDLE (with the exception of the two STANDBY-modes), the system clock source must stabilize before the CPU starts to operate. This introduces a short delay which depends on the selected clock source. If an internal RC oscillator or external clock is used, the start-up delay is 6 cycles. This is in addition to the RC oscillator start-up time. If the XTAL oscillator is used, the start-up delay is configurable. If frequency stability is required it is recommended with start-up delays of 1,000 cycles for ceramic resonators and 16,000 cycles for quartz crystals respectively. This is in addition to the oscillator startup time, which will depend on the resonator and load capacitances.

In addition, there is a 13 cycle minimum delay before an Interrupt Service Routine (ISR) starts executing after wake-up. This is because the program counter is pushed on stack and jump to the ISR.

During the start-up delay the power consumption is close to the power consumption in IDLE, and thus represents “inefficient” power. If possible, it is therefore recommended to wake up as seldom as possible and rather “do more” every time the device wakes up.

To minimize the wake-up delay and conserve power, use an RC oscillator or external clock source, and wake up as seldom as possible.

1.6 Power Reduction Registers

Most peripherals and internal modules can be individually stopped to avoid that these draw power in ACTIVE mode and in IDLE sleep. This is done by setting their respective bits in the Power Reduction Registers (PRR), which causes them to be disconnected from the peripheral clock domain. It is required to disable modules and peripherals via their respective control registers before setting their PRR bit, for the Power Reduction to be effective. Some modules must be reinitialized after clearing their PRR bit. Refer to the sections about the individual PRR-bits in the datasheet manual for more information.

In POWER-SAVE and POWER-DOWN the modules are stopped regardless of the PRRs, since the peripheral clock domain is disabled.

To minimize power consumption, use the PRRs to disable peripherals and modules that are not used.

1.7 RTC Clock Source

One of the reasons for using IDLE, POWER-SAVE, and EXTENDED STANDBY is that the RTC and its clock are active in these sleep modes. The RTC is commonly used to wake the device up at timed intervals.

For most Atmel® AVR® XMEGA®-families, three different oscillators can be used to clock the RTC: An external 32kHz crystal, the internal 32kHz RC oscillator, and the internal 32kHz Ultra Low Power (ULP) oscillator. In all cases, a prescaled 1kHz clock signal is available and should be used for reduced power consumption. For the external 32kHz crystal oscillator, a special low power mode is also available (X32KLPM).

It is recommended to use an external 32kHz crystal with X32KLPM enabled. This gives lower power consumption than the ULP, yet greater accuracy than the internal RC oscillator. This oscillator may also be used as the system clock source, if such a low frequency is acceptable.

Table 1-1. Examples of Accuracy and Current Consumption for RTC with Clock Sources

Oscillator	Accuracy[3]	Current Consumption[1]
ULP	±30%	1µA
RC32k	±1%	30µA
TOSC (32kHz XTAL)[2]	±0.001% (10ppm)	0.6µA

Notes: 1. 3V operating voltage.
2. Depends on e.g. quality of crystal.
3. Refer to the datasheet for exact values and conditions, the values stated here are meant as guideline only.

For Atmel® AVR® XMEGA®-families with the battery backup module and 32-bit RTC, only the 32kHz crystal oscillator may be used as clock source. In these devices, the RTC is left running regardless of sleep.

Minimize power consumption by clocking the RTC at 1kHz with an external crystal in low power mode.

1.8 State of Digital I/O Pins

All digital IO pins are by default configured as input and they are floating to avoid hardware conflicts. All unused pins can be left unconnected with no external components attached. Since these pins have digital input buffers it is important to ensure that the level on an I/O pin is well-defined to avoid sporadic internal switching and leakage. The leakage caused by floating I/O is relatively small and is mainly observable in sleep, but can be minimized by ensuring that the state of the pins is either high or low. Therefore the user should enable internal pull up or pull down on these pins to minimize power consumption.

If a pin is connected to an analog source, the digital input buffer on that pin should be disabled even if it is not configured as an input. This is done by use of the PINnCTRL registers for the individual ports. Refer to the device manual for information on how to do this configuration.

To minimize power consumption, enable pull-up or -down on all unused pins, and disable the digital input buffer on pins that are connected to analog sources.

1.9 Virtual Port Registers

To minimize the time spent in ACTIVE mode, virtual port registers can be used. This allows for single-cycle access with I/O memory specific instructions like IN, OUT and bit manipulation for the registers DIR, IN, OUT, and INTFLAGS for up to four I/O ports.

Use the Virtual Port registers for I/O port access to minimize power consumption.

1.10 General Purpose I/O Registers

Another way to minimize the time spent in ACTIVE mode is to use the GPIO registers for storage of variables. This is also due to the possibility of single-cycle access with I/O memory specific instructions.

Note: The GPIO registers are defined as volatile, so temporary variables should in some cases be used when manipulating variables stored in these registers. Otherwise, the performance gain will be lost.

Use the General Purpose I/O registers for variable storage to minimize power consumption.

1.11 Watchdog

The Watchdog is basically a timer with a separate clock source. It will if enabled contribute to the power consumption in sleep. The Watchdog can only be clocked by the internal 32kHz ultra low power (ULP) oscillator prescaled to 1kHz.

To minimize the power consumption disable the watchdog.

1.12 Brown Out Detector

The purpose of the Brown Out Detector (BOD) is to ensure that the device is not operating at a too low voltage. It is highly recommended to use the internal BOD to ensure that the device always operates within specification.

However, during sleep the device is “not operating” or rather, it is not executing code. For this reason, the BOD can be configured separately for ACTIVE/IDLE and sleep modes. This allows for the BOD to be enabled only in ACTIVE and IDLE mode. All configuration of the BOD is done with the device fuses.

To further reduce power consumption, the BOD may be run in sampled mode. The sample rate is approximately 1kHz, as it is clocked from the prescaled ULP oscillator. The BOD cannot detect voltage dips between samples in this mode, so it should only be used in applications with slowly varying operating voltages, such as battery powered ones.

Disable the BOD - or better, disable it while in sleep - to reduce power consumption. Use sampled mode if only slow changes in operating voltage are likely.

1.13 JTAG Interface and On-Chip Debugging

The JTAG interface is used for programming and debugging, but has no function during operation of an end-product. It is clocked and active during sleep if the On-chip Debugging (OCD) feature is enabled. The OCD and JTAG interface should therefore be disabled if it is not needed.

The OCD can be disabled in fuses, while the JTAG interface can be disabled both in fuses and in software. Disabling the JTAG in software ensures that the device can be reprogrammed because the JTAG interface is re-enabled upon RESET.

Alternatively, the PDI interface can be used for programming and debugging. In this case the JTAG interface may not be required at all, and may be disabled by fuses. The PDI interface also works in all sleep modes.

To minimize power consumption, disable the OCD and the JTAG interface.

1.14 Flash and EEPROM Power Reduction Modes

With the Atmel® AVR® XMEGA® NVM (Non-Volatile Memory) controller, it is possible to enable power reduction modes for the EEPROM and Flash. In these modes, the EEPROM and the currently unused section of Flash (i.e., application or boot section) are powered down in ACTIVE mode, just as they are in any sleep modes. These power reduction modes will not affect power consumption in sleep.

If the CPU attempts to access a non-volatile memory with power reduction mode on, the CPU is halted for a time interval corresponding to wake-up from IDLE sleep while the memory is re-activated.

Note: There is an erratum regarding Flash power reduction mode and sleep. For the affected devices, the workaround is to disable the Flash power reduction mode before entering sleep, then enabling it again on wake-up. Power consumption in sleep is not affected by this.

Enable power reduction mode for EEPROM and Flash to reduce power consumption in ACTIVE mode.

1.15 Writing to EEPROM

If more than one byte is to be written to EEPROM, one should make use of the EEPROM page buffer rather than doing byte-wise writes. This is because it takes just as long to write one byte as it takes to write an entire page to EEPROM. If, e.g., two bytes are to be written, byte-wise writing will take twice as long as necessary. Since the current consumption also increases during EEPROM writing, this gives a “double penalty”.

To minimize power consumption, use page-wise writing to EEPROM rather than byte wise.

1.16 Sleep Modes

In most applications it is desirable to minimize the power consumption, but not to reduce the system clock frequency – mainly to ensure fast processing and quick response of the system/product. In such applications one can use the “sleep modes” of the Atmel® AVR® XMEGA® to put the device in a low power state when there is nothing to process. The main principle is then to run as fast as possible, and sleep as much as possible. Running as fast as possible reduces the effect of static power consumption (i.e. independent of clock frequency), e.g. due to non-volatile memory being enabled in ACTIVE mode.

The power consumption and operation of peripherals in sleep depends on which sleep mode is used. Table 1-2 shows the characteristics of the different sleep modes available for XMEGA devices. An application may switch between sleep modes during operation, depending on which mode is the most suitable at the time.

Table 1-2. Available Sleep Modes for XMEGA

	Active Clock Domain			Oscillators		Wake-up Sources			
Sleep Mode	**CPU Clock**	**Peripheral Clock**	**RTC Clock**	**System clock source**	**RTC clock source**	**Asynchronous Port Interrupt**	**TWI address match interrupt**	**Real Time Clock interrupt**	**All interrupt**

	Active Clock Domain			Oscillators		Wake-up Sources			
IDLE		X	X	X	X	X	X	X	X
POWER DOWN						X	X		
POWER SAVE			X		X	X	X	X	
STANDBY				X		X	X		
EXTENDED STANDBY			X	X	X	X	X	X	

2 Reducing Power Consumption for XMEGA Xplained Kits

Chapter 2 describes the procedure to reduce power consumption of the XMEGA-A3BU Xplained kit by making the device (ATxmega256A3BU) to enter into sleep mode. This chapter also provides the code example to enter into various sleep modes (IDLE, Power Down, Power Save, Standby, Extended Standby) and the current consumption values that were obtained in respective sleep modes under fixed operating condition.

2.1 Operating Condition and Measurement Setup

Operating Condition

ATxmega256A3BU on the XMEGA-A3BU Xplained kit runs at 2MHz frequency from the internal 2MHz RC oscillator and at a supply voltage of 3.3V. Since this condition is different from the datasheet condition there may be difference in power consumption numbers between the values mentioned in datasheet and those that are measured in this application note. So these numbers (the current values highlighted in bold in upcoming sections) are only a reference to show that power consumption reduces on entering different sleep modes.

Since XMEGA-A3BU Xplained kit does not have the provision to be clocked by external source we are using the default internal 2MHz RC oscillator.

2.2 Getting Started with Firmware

The application note is associated with a firmware zip package which contains code examples for low power techniques. This section explains the usage of the code examples. The zip package consists of one Atmel Studio solution with three projects (Sleep_modes, Active_mode_with_ADC and Idle_mode_with_ADC). To measure the current consumption in various sleep modes (from Section 2.3 to 2.8) select the “Sleep_modes” project as the Start up project as shown below. Then compile the solution and burn the hex file into the XMEG-A3BU Xplained kit using Atmel-ICE and measure the power.

While measuring power consumptions following steps should be followed:

- Before programming the hex file, ensure that the BODACT and BODPD fuses are disabled in the programming window.
- After programming the hex file using Atmel-ICE disconnect the Atmel-ICE header connected to the XMEGA-A3BU Xplained kit.
- Now power toggle the XMEGA-A3BU Xplained kit, then connect the multimeter as shown in the Figure 2-1 to measure the current consumption values. The multimeter must be connected only during power measurement. It should be disconnected during programming.

Figure 2-1. Measurement Setup

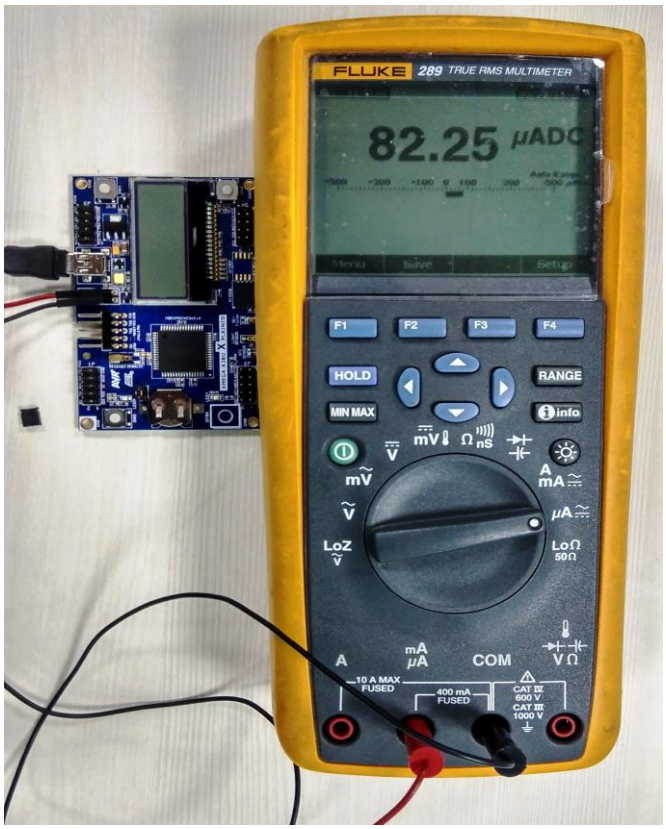
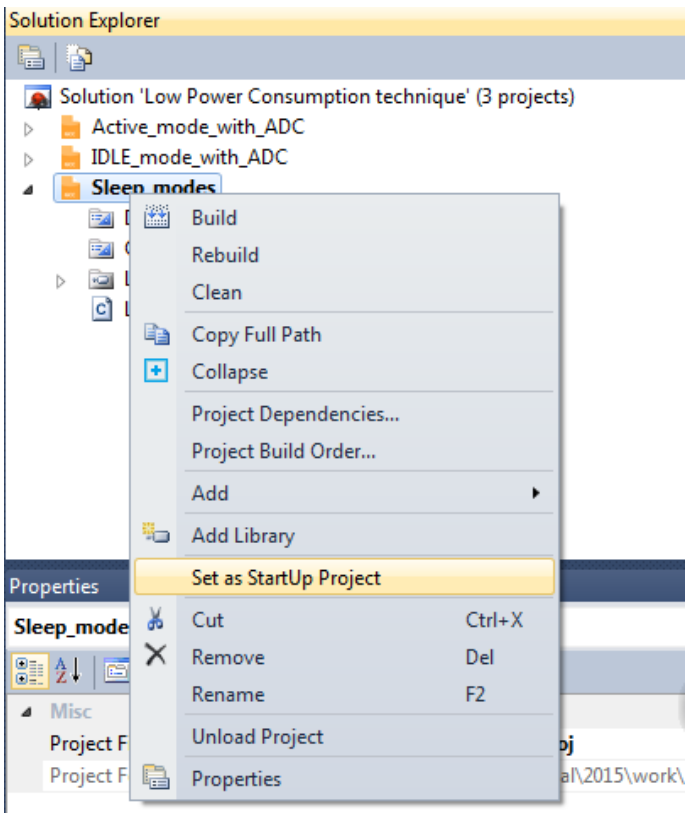


Figure 2-2. Selecting “Sleep_modes” as Startup Project



2.3 Current Consumption in Active Mode

In **Active mode** i.e. when the XMEGA-A3BU Xplained kit is powered via USB and none of the sleep modes are enabled, the current consumption of the kit is high (approximately 3mA).This current can be reduced by applying the techniques mentioned in Chapter 1 Factors Affecting Power Consumption. The power consumption is reduced by using PRR (Power Reduction Register) to disable peripherals and modules that are not used. In this example none of the peripherals are used hence we set PRR bits for all the peripherals.

The following function disables the clock to all unused peripherals by writing to the power reduction registers:

```
void disable_peripherals(void)
{
        PR.PRGEN = PR_USB_bm | PR_AES_bm | PR_EBI_bm | PR_RTC_bm | PR_EVSYS_bm | PR_DMA_bm ; // stop the clock to USB,AES,EBI,RTC,Event system and DMA controller.
        PR.PRPA = PR_DAC_bm | PR_ADC_bm | PR_AC_bm; // Stops the clock to DAC, ADC and Analog Comparator.

        PR.PRPB = PR_DAC_bm | PR_ADC_bm | PR_AC_bm;

        PR.PRPC = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm | PR_TC1_bm | PR_TC0_bm ; // Stops the clock to Two wire interface, USART1, USART0,SPI,Timer counter 0/1.

        PR.PRPD = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm | PR_TC1_bm | PR_TC0_bm ;


        PR.PRPE = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm | PR_TC1_bm | PR_TC0_bm ;

        PR.PRPF = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm | PR_TC1_bm | PR_TC0_bm ;

}
```

On executing the above code, clocks to all the peripherals are stopped and hence the current consumption of the XMEGA-A3BU Xplained kit is reduced (approximately **2.5mA**).

To further minimize power consumption, disable the JTAG interface. The JTAG interface is used for programming and debugging, but has no function during operation of an end-product. The OCD and JTAG interface should therefore be disabled if it is not needed. The OCD can be disabled in fuses, while the JTAG interface can be disabled both in fuses and in software.

Disabling the JTAG in software ensures that the device can be reprogrammed because the JTAG interface is re-enabled upon RESET.

Alternatively, the PDI interface can be used for programming and debugging. In this case the JTAG interface may not be required at all, and may be disabled by fuses. The PDI interface works in all sleep modes.

The following code is used to disable JTAG interface from software:

```
void disable_JTAG(void)(1)
{
CCP        = CCP_IOREG_gc;
MCU.MCUCR = MCU_JTAGD_bm;
}
```

One disabling the JTAG interface the current consumption is significantly reduced (approximately **1.8mA**).

Note: When JTAGD bit is set from software further programming is blocked by Atmel Studio in normal mode. So, enable the option ‘Use external reset’ in device programming window or in tool tab in project setting for programming/debugging.

Note: (1) Make sure that the optimization level is greater than –O0.

2.4 Current Consumption in IDLE Mode

In **IDLE** Mode, all the peripherals are active, only the Atmel AVR CPU core and non-volatile memories (Flash and EEPROM) are stopped. The DMA controller and Event system are still active in this mode, allowing for DMA data transfers and event triggers e.g. ADC conversion complete triggering DMA to transfer data from ADC result register to USART DATA register while the CPU is not running. The device can be woken up by all interrupts. The device enters into IDLE sleep mode by executing the below code:

```
#include <avr/io.h>
#include <avr/sleep.h>

int main(void)
{
    set_sleep_mode(CURRENT_SLEEP_MODE);
    sleep_enable();
    sleep_cpu();
    while(1)
    {

    }
}
```

Note: If additional power reduction is needed then the functions `disable_peripherals ()` and `disable_JTAG ();` explained above have to be included in the application and called before entering into sleep mode.

In the example code, to put the device into IDLE sleep mode, the line selecting Active mode has to be commented out and the line selecting IDLE mode has to be uncommented as shown below:

```
/* Choose the sleep mode that the user wishes, by default no sleep mode is selected*/

//#define CURRENT_SLEEP_MODE      SLEEP_MODE_ACTIVE
#define CURRENT_SLEEP_MODE        SLEEP_MODE_IDLE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_STANDBY
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_EXT_STANDBY
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_PWR_DOWN
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_PWR_SAVE
```

Since in the IDLE mode the AVR CPU core and non-volatile memories are stopped the current consumption (approximately **1.4mA**) is lower than that obtained in Active mode. As mentioned above this power can further be reduced by disabling the peripherals and the JTAG interface.

The current consumption comes down approximately to **1mA** when unused peripherals are disabled by calling the `disable_peripherals ()` function before entering into IDLE sleep mode inside main () function.

This current consumption is further reduced approximately to **0.8mA** by disabling the JTAG interface from software by calling `disable_JTAG ();` function before entering into IDLE sleep mode.

2.5 Current Consumption in POWER DOWN Mode

POWER DOWN is the deepest sleep mode. In this mode most of the device peripherals are stopped. The device is unable to wake itself up from this mode since both the peripheral clock and RTC clock are disabled. This mode therefore relies on external signals to wake up the device for example asynchronous pin interrupts or TWI address match interrupt. However, XMEGAs with battery backup module can wake the device up from

power down sleep mode using the 32-bit RTC module as the RTC in these devices will run regardless of the sleep modes.

The device enters into the POWER DOWN mode by executing the below code:

```
#include <avr/io.h>
#include <avr/sleep.h>
int main(void)

{
    set_sleep_mode(CURRENT_SLEEP_MODE);
    sleep_enable();
    sleep_cpu();
    while(1)
    {

    }
}
```

Note: If additional power reduction is needed then the functions `disable_peripherals ()` and `disable_JTAG ();` explained above have to be included in the application and called before entering into sleep mode.

In this application note to put the device into POWER DOWN mode, the line selecting Power-Down Mode has to be uncommented.

```
/* Choose the sleep mode that the user wishes, by default no sleep mode is selected*/

//#define CURRENT_SLEEP_MODE      SLEEP_MODE_ACTIVE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_IDLE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_STANDBY
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_EXT_STANDBY
  #define CURRENT_SLEEP_MODE      SLEEP_MODE_PWR_DOWN
 //#define CURRENT_SLEEP_MODE     SLEEP_MODE_PWR_SAVE
```

In POWER-DOWN mode XMEGA-A3BU Xplained kit's current consumption is approximately **0.8mA**. This can be significantly reduced approximately to **0.8µA** by disabling the JTAG interface and disabling the unused peripherals.

2.6 Current Consumption in POWER SAVE Mode

POWER SAVE mode is identical to power down, with one exception. If the real-time counter (RTC) is enabled, it will keep running during sleep and the device can wake up from either an RTC overflow or compare match interrupt. Because the system clock source is stopped in this sleep mode, wake-up takes a bit longer than for IDLE mode as the system clock must stabilize before operation.

The device enters into the POWER SAVE mode by executing the below code:

```
#include <avr/io.h>
#include <avr/sleep.h>
int main(void)
{
    set_sleep_mode(CURRENT_SLEEP_MODE);
    sleep_enable();
    sleep_cpu();
    while(1)
    {

    }
}
```

Note: If additional power reduction is needed then the functions `disable_peripherals ()` and `disable_JTAG ();` explained above have to be included in the application and called before entering into sleep mode.

In this application note to put the device into POWER SAVE mode, the line selecting POWER SAVE mode has to be uncommented.

```
/* Choose the sleep mode that the user wishes, by default no sleep mode is selected*/

//#define CURRENT_SLEEP_MODE      SLEEP_MODE_ACTIVE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_IDLE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_STANDBY
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_EXT_STANDBY
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_PWR_DOWN
  #define CURRENT_SLEEP_MODE      SLEEP_MODE_PWR_SAVE
```

In POWER SAVE mode XMEGA-A3BU Xplained kit's current consumption is around **0.8mA**.This can be significantly reduced approximately to **0.8µA** by disabling the JTAG interface and disabling the unused peripherals.

Note: For XMEGAs with battery backup module there is no difference between the POWER DOWN and POWER SAVE sleep modes.

2.7 Current Consumption in STANDBY Mode

STANDBY mode is identical to POWER DOWN, with the exception that the enabled system clock sources are kept running while the CPU, peripherals, and RTC clocks are stopped. This reduces the wake-up time. Therefore it is useful when short wake-up time is needed.

The device enters into the STANDBY mode by executing the below code:

```
#include <avr/io.h>
#include <avr/sleep.h>
int main(void)
{
    set_sleep_mode(CURRENT_SLEEP_MODE);
    sleep_enable();
    sleep_cpu();
    while(1)
    {

    }
}
```

In this application note to put the device into STANDBY mode, the line selecting STANDBY mode has to be uncommented.

```
/* Choose the sleep mode that the user wishes, by default no sleep mode is selected*/

//#define CURRENT_SLEEP_MODE      SLEEP_MODE_ACTIVE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_IDLE
  #define CURRENT_SLEEP_MODE      SLEEP_MODE_STANDBY
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_EXT_STANDBY
 //#define CURRENT_SLEEP_MODE     SLEEP_MODE_PWR_DOWN
 //#define CURRENT_SLEEP_MODE     SLEEP_MODE_PWR_SAVE
```

In STANDBY mode XMEGA-A3BU Xplained kit's current consumption is approximately **0.9mA**. This can be significantly reduced to around **82µA** by disabling the JTAG interface and disabling the unused peripherals.

2.8 Current Consumption in EXTENDED STANDBY Mode

EXTENDED STANDBY mode is identical to POWER SAVE mode, with the exception that the enabled system clock sources are kept running while the CPU and peripheral clocks are stopped. This reduces the wake-up time.

The device enters into the EXTENDED STANDBY mode by executing the below code:

```
#include <avr/io.h>
#include <avr/sleep.h>
int main(void)
{
    set_sleep_mode(CURRENT_SLEEP_MODE);
    sleep_enable();
    sleep_cpu();
    while(1)
    {

    }
}
```

In this application note to put the device into EXTENDED STANDBY mode, the line selecting EXTENDED STANDBY mode has to be uncommented.

```
/* Choose the sleep mode that the user wishes, by default no sleep mode is selected*/

//#define CURRENT_SLEEP_MODE      SLEEP_MODE_ACTIVE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_IDLE
//#define CURRENT_SLEEP_MODE      SLEEP_MODE_STANDBY
  #define CURRENT_SLEEP_MODE      SLEEP_MODE_EXT_STANDBY
 //#define CURRENT_SLEEP_MODE     SLEEP_MODE_PWR_DOWN
 //#define CURRENT_SLEEP_MODE     SLEEP_MODE_PWR_SAVE
```

In EXTENDED STANDBY mode XMEGA-A3BU Xplained kit's current consumption is approximately **0.9mA**. This can be significantly reduced to approximately **82µA** by disabling the JTAG interface and disabling the unused peripherals.

Table 2-1. Power Consumption of XMEGA-A3BU Xplained Kit in Various Sleep Modes

	Power Consumption	Power[1] Consumption with JTAG Disabled
IDLE	~2.1mA	~800µA
POWER-DOWN	~800µA	~0.8µA
POWER-SAVE	~800µA	~0.8µA
STANDBY	~900µA	~82µA
EXTENDED STANDBY	~900µA	~82µA

Note: 1. Since the measurement condition in this application notes is different from the datasheet condition there may be difference in power consumption numbers between the values mentioned in datasheet and those that are mentioned in this application note.

Vcc = 3.3V, Clock source= 2MHz internal RC oscillator.

3 ADC Power Measurements

Chapter 3 explains how current consumption of the device can be reduced while a peripheral (ADC is taken as example in this application note) is running. The operating condition and measurement setup is similar to that mentioned under the Section 2.1. The ADC is enabled with the following basic configuration.

The ADC is configured in single ended unsigned mode with 12 bit resolution (right adjusted) and high current limitation. The internal 2MHz clock source is prescaled by 2 with main clock prescaler and the ADC prescaler further divides the clock by 4. The reference is selected as internal Vcc/1.6. The channel 0 conversion complete interrupt is enabled with high priority.

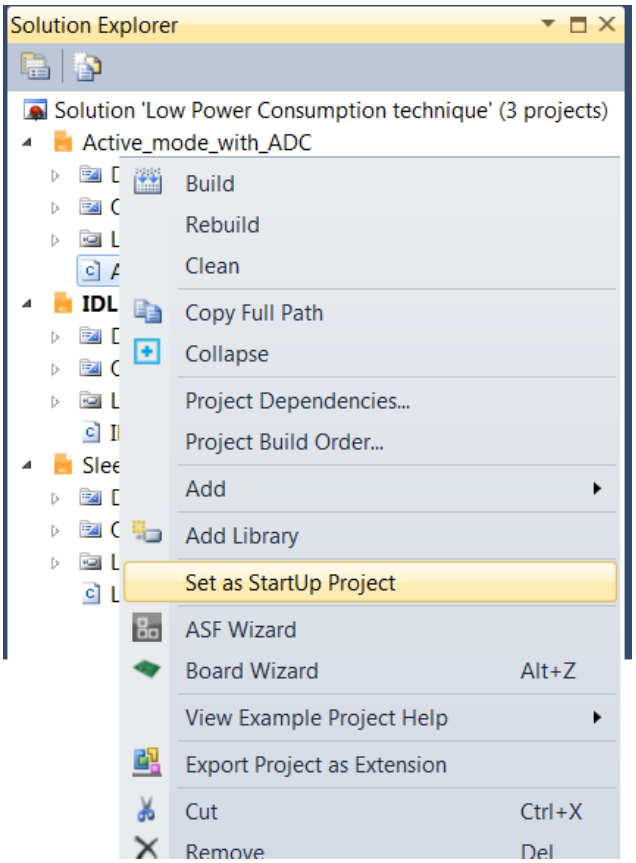
```
void ADC_initialize(void)
{
CCP = 0xD8;
//prescaler divides the clock source by 2
CLK.PSCTRL = CLK_PSADIV_2_gc;
//12bit ADC resolution(right adjusted) & high current limitation
ADCA.CTRLB = ADC_CURRLIMIT_HIGH_gc;
//ADC ref setting Vcc/1.6
ADCA.REFCTRL = ADC_REFSEL_INTVCC_gc;
//ADC clock source Fosc/4
ADCA.PRESCALER = ADC_PRESCALER_DIV4_gc;
//ADC channel selection as single ended
ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;
//CH0 ROOTED TO PB1
ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN0_gc;
//enable the conversion completes interrupt and defines it as high priority interrupt.
ADCA.CH0.INTCTRL = ADC_CH_INTMODE_COMPLETE_gc | ADC_CH_INTLVL_HI_gc;
//enable the ADC
ADCA.CTRLA = ADC_ENABLE_bm;
}
```

3.1 Current Consumption in Active Mode with ADC Running

Section 3.1 provides the code example for achieving minimum current while running ADC under active mode. Disabling the JTAG interface and unused peripherals will reduce the power consumption, as mentioned under Section 2.3. Hence, disabling the unused peripherals (except ADC) and JTAG interface that reduces the power consumption.

In this application note the code example for ADC running is available in the project Active_mode_with_ADC, so this project has to be set as “Startup project” and then the solution should be compiled.

Figure 3-1. Selecting “Active_mode_with_ADC” as Startup Project




```
#include <avr/io.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>

uint16_t adc_results =0;

void disable_peripherals(void)
{
	PR.PRGEN = PR_USB_bm | PR_AES_bm | PR_EBI_bm | PR_RTC_bm | PR_EVSYS_bm | PR_DMA_bm;
// stop the clock to USB,AES,EBI,RTC,Event system and DMA controller.
	PR.PRPA = PR_DAC_bm | PR_AC_bm;  // Stops the clock to DAC and Analog Comparator
	PR.PRPB = PR_DAC_bm | PR_AC_bm;

	PR.PRPC = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ; // Stops the clock to Two wire interface, USART1, USART0, SPI, Ti-
mer counter 0/1.

	PR.PRPD = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ;

	PR.PRPE = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ;

	PR.PRPF = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ;
```

```
}

void disable_JTAG (void)
{
    CCP = CCP_IOREG_gc;
    MCU.MCUCR = MCU_JTAGD_bm; // disable JTAG.
}

void ADC_initialize(void)
{
CCP = 0xD8;
//prescaler divides the clock source by 2
CLK.PSCTRL = CLK_PSADIV_2_gc;
//12bit ADC resolution(right adjusted) & high current limitation
ADCA.CTRLB = ADC_CURRLIMIT_HIGH_gc;
//ADC ref setting Vcc/1.6
ADCA.REFCTRL = ADC_REFSEL_INTVCC_gc;
//ADC clock source Fosc/4
ADCA.PRESCALER = ADC_PRESCALER_DIV4_gc;
//ADC channel selection as single ended
ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;
//CH0 ROOTED TO PB1
ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN0_gc;
//enable the conversion completes interrupt and defines it as high priority interrupt.
ADCA.CH0.INTCTRL = ADC_CH_INTMODE_COMPLETE_gc | ADC_CH_INTLVL_HI_gc;
//enable the ADC//
ADCA.CTRLA = ADC_ENABLE_bm;
}

int main(void)
{


       disable_peripherals();
       disable_JTAG();
       ADC_initialize();
       //start conversion for the ADC channel 0
       ADCA.CH0.CTRL = ADC_CH_START_bm;
       PMIC.CTRL = PMIC_HILVLEN_bm;
       sei();
       while(1)
       {

       }
}

ISR(ADCA_CH0_vect)
{
       //ADC results
       adc_results = ADCA_CH0RES;
       ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;
       //start conversion for the ADC channel 0
       ADCA.CH0.CTRL = ADC_CH_START_bm;

}
```

The current consumed on executing the above project is approximately **3.5mA**. The ADC conversion result is stored in the variable adc_results which is defined globally. As mentioned under the Section 1.16, this power can further be reduced by enabling sleep modes.

3.2 Current Consumption in IDLE Mode with ADC Running

The power consumed by XMEGA-A3BU Xplained kit, under IDLE mode with ADC enabled is given as follows (only IDLE mode is discussed because ADC can be used only in IDLE mode). Section 3.2 provides the code example for achieving minimum current while running ADC under idle sleep mode. In this project the sleep mode is enabled and the power reduction techniques are also applied.

Now 'IDLE_mode_with_ADC' project that is available along with the attached solution should be set as "Startup project" and then the solution should be compiled.

```
#include <avr/io.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>

uint16_t adc_results =0;

void disable_peripherals(void)
{
       PR.PRGEN = PR_USB_bm | PR_AES_bm | PR_EBI_bm | PR_RTC_bm | PR_EVSYS_bm | PR_DMA_bm ;
// stop the clock to USB,AES,EBI,RTC,Event system and DMA controller.
       PR.PRPA = PR_DAC_bm | PR_AC_bm; // Stops the clock to DAC and Analog Comparator
       PR.PRPB = PR_DAC_bm | PR_AC_bm;

       PR.PRPC = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ; // Stops the clock to Two wire interface, USART1, USART0, SPI, Ti-
mer counter 0/1.

       PR.PRPD = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ;

       PR.PRPE = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ;

       PR.PRPF = PR_TWI_bm | PR_USART1_bm | PR_USART0_bm | PR_SPI_bm | PR_HIRES_bm |
PR_TC1_bm | PR_TC0_bm ;

}

void disable_JTAG (void)
{
    CCP = CCP_IOREG_gc;
    MCU.MCUCR = MCU_JTAGD_bm; // disable JTAG.
}

void ADC_initialize(void)
{
CCP = 0xD8;
//prescalerA divides the clock source by 2
CLK.PSCTRL = CLK_PSADIV_2_gc;
//12bit ADC resolution(right adjusted) & high current limitation
ADCA.CTRLB = ADC_CURRLIMIT_HIGH_gc;
//ADC ref setting Vcc/1.6
ADCA.REFCTRL = ADC_REFSEL_INTVCC_gc;
//ADC clock source Fosc/4
```

```
ADCA.PRESCALER = ADC_PRESCALER_DIV4_gc;
//ADC channel selection as single ended
ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;
//CH0 ROOTED TO PB1
ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN0_gc;
//enable the conversion completes interrupt and defines it as high priority interrupt.
ADCA.CH0.INTCTRL = ADC_CH_INTMODE_COMPLETE_gc | ADC_CH_INTLVL_HI_gc;
//enable the ADC//
ADCA.CTRLA = ADC_ENABLE_bm;
}

int main(void)
{

        disable_peripherals();
        disable_JTAG();
        ADC_initialize();
        set_sleep_mode(SLEEP_MODE_IDLE); // select IDLE mode
        sleep_enable();
        //start conversion for the ADC channel 0
        ADCA.CH0.CTRL = ADC_CH_START_bm;
        PMIC.CTRL = PMIC_HILVLEN_bm;
        sei();
        while(1)
        {
                //enable sleep
                sleep_cpu();
        }

}

ISR(ADCA_CH0_vect)
{
        //ADC results
        adc_results = ADCA_CH0RES;
        ADCA.CH0.INTFLAGS = ADC_CH_CHIF_bm;
        //start conversion for the ADC channel 0
        ADCA.CH0.CTRL = ADC_CH_START_bm;
}
```

On executing the above code the power consumed by XMEGA-A3BU kit is approximately about **2.8mA**. Under active mode, the current consumed when the ADC is running (with basic conditions as explained in the code snippet ADC_initialize ()) is approximately about **3.9mA** which is considerably reduced to **2.8mA** by applying various power consumption techniques.

4 References

[1] Atmel AVR XMEGA AU Manual

[2] ATxmegaA3BU complete datasheet

[3] Atmel AVR1935: XMEGA A3BU Xplained Getting Started Guide

[4] Atmel AVR1923: XMEGA-A3BU Xplained Hardware User Guide

[5] Atmel AVR1934: XMEGA A3BU Xplained Software User Guide

[6] AVR1010: Minimizing the power consumption of XMEGA devices

5 Revision History

Doc Rev.	Date	Comments
42456A	05/2014	Initial document release.

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**