Descripción general de la operación de bajo consumo de AVR® MCU

Descripción general de bajo consumo

El microcontrolador AVR ® de 8 bits proporciona varios modos de reposo y sincronización de reloj controlada por software para adaptar el consumo de energía a los requisitos de la aplicación. Los modos de suspensión permiten que el microcontrolador apague los módulos no utilizados para ahorrar energía. Cuando el dispositivo entra en modo de suspensión, la ejecución del programa se detiene y se utilizan interrupciones o reinicio para reactivar el dispositivo nuevamente. El reloj individual de los periféricos no utilizados se puede detener durante el funcionamiento normal o en suspensión, lo que permite una gestión de energía mucho más precisa que los modos de suspensión por sí solos.

Para llegar a las cifras de potencia más bajas posibles, hay un par de puntos a los que prestar atención. No es solo el modo de suspensión lo que define el consumo de energía, sino también el estado de los pines de E/S, la cantidad de módulos periféricos habilitados, etc.

El consumo de energía es proporcional al voltaje de funcionamiento y, para conservar energía, debe considerar usar el voltaje del sistema más bajo posible. Además, el consumo también es directamente proporcional a la frecuencia del reloj y, si no se utilizan los modos de suspensión, el dispositivo debe funcionar con la frecuencia más baja posible.

Consejos y trucos para reducir la potencia de un AVR®

- Use el **registro de reducción de energía (PRR0)** para detener el reloj en los periféricos individuales no utilizados, lo que reduce el consumo de energía.

Name: PRR0
Offset: 0x64
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	PRTWI0	PRTIM2	PRTIM0	PRUSART1	PRTIM1	PRSPI0	PRUSART0	PRADC
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

(/local--files/8avr:low-power-overview/prp-picture.png)

Bit 7 – PRTWI0: Power Reduction TWI0
Writing a logic one to this bit shuts down the TWI 0 by stopping the clock to the module. When waking up the TWI again, the TWI should be re initialized to ensure proper operation.

Bit 6 – PRTIM2: Power Reduction Timer/Counter2
Writing a logic one to this bit shuts down the Timer/Counter2 module in synchronous mode (AS2 is 0). When the Timer/Counter2 is enabled, operation will continue like before the shutdown.

Bit 5 – PRTIM0: Power Reduction Timer/Counter0
Writing a logic one to this bit shuts down the Timer/Counter0 module. When the Timer/Counter0 is enabled, operation will continue like before the shutdown.

Bit 4 – PRUSART1: Power Reduction USART1
Writing a logic one to this bit shuts down the USART by stopping the clock to the module. When waking up the USART again, the USART should be re initialized to ensure proper operation.

Bit 3 – PRTIM1: Power Reduction Timer/Counter1
Writing a logic one to this bit shuts down the Timer/Counter1 module. When the Timer/Counter1 is enabled, operation will continue like before the shutdown.

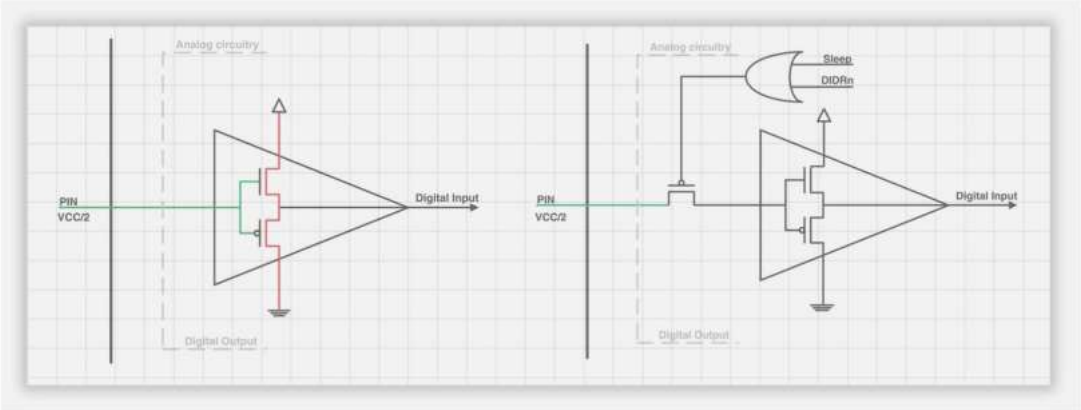
Bit 2 – PRSPI0: Power Reduction Serial Peripheral Interface 0
If using debugWIRE On-chip Debug System, this bit should not be written to one. Writing a logic one to this bit shuts down the Serial Peripheral Interface by stopping the clock to the module. When waking up the SPI again, the SPI should be re initialized to ensure proper operation.

Bit 1 – PRUSART0: Power Reduction USART0
Writing a logic one to this bit shuts down the USART by stopping the clock to the module. When waking up the USART again, the USART should be re initialized to ensure proper operation.

Bit 0 – PRADC: Power Reduction ADC
Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot use the ADC input MUX when the ADC is shut down.

(/local--files/8avr:low-power-overview/prt-description.png)

- Utilice el registro de **desactivación de entrada digital (DIDR)** para apagar los búferes de entrada digital no utilizados y detener la corriente de fuga.



(/local--files/8avr:low-power-overview/didr-circuit.png)

Name: DIDR0
Offset: 0x7E
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

(/local--files/8avr:low-power-overview/didr0-picture.png)

Name: DIDR1
Offset: 0x7F
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
							AIN1D	AIN0D
Access							R/W	R/W
Reset							0	0

Bit 1 – AIN1D: AIN1 Digital Input Disable

Bit 0 – AIN0D: AIN0 Digital Input Disable
When this bit is written logic one, the digital input buffer on the AIN1/0 pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the AIN1/0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

(/local--files/8avr:low-power-overview/didr1.png)

Visite la **página (/8avr:low-power-example)** de ejemplo de bajo consumo para ver un código de ejemplo sobre cómo reducir el consumo de energía del AVR.

Proyecto de ejemplo de AVR de baja potencia

◎ Objetivo

Esta página ilustra varios métodos para configurar un MCU AVR ® de 8 bits para que funcione con bajo consumo de energía. Con este ejercicio lograrás:

- Cree un proyecto y agregue un código simple para hacer parpadear un LED
- Ejecutar el proyecto y medir el consumo de energía
- Hacer modificaciones al código para reducir la potencia
- Ejecute el proyecto modificado y observe un consumo de energía reducido

☑ Materiales

Requisitos de Software

		Instaladores			
		ventanas	linux	Mac OS X	
Herramienta	?Sobre				Instrucciones de instalación
Entorno de desarrollo integrado **Atmel®** Studio	(/atstudio:start)	(http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)			(/install:atstudio)

Requisitos de hardware

- ATmega328PB Mini tablero explicado
- Kit depurador de energía
- Dos cables micro USB
- Tres cables hembra a macho y un cable macho a macho

Herramienta		?Sobre	
 (http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)	Mini kit de evaluación **ATmega328PB Xplained**	(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)	(https://www.microchipdirect.com/pro

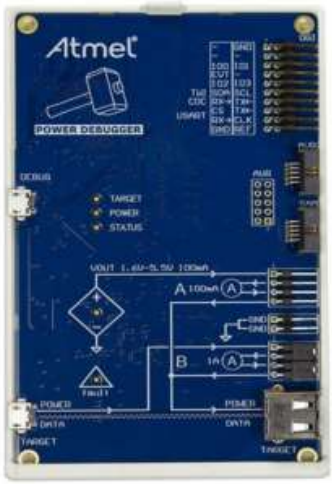
ATmega328PB Tarjeta explicada

The **ATmega328PB Xplained Mini Evaluation Kit** is a hardware platform for evaluating the Atmel ATmega328PB microcontroller. An external debugger is NOT needed to run these exercises. The ATmega328PB has a fully integrated embedded debugger on-board.

(/local--files/8avr:low-power-example/xplained-board.png)

Power Debugger Kit

The Power Debugger is a CMSIS-DAP compatible debugger which works with Atmel Studio v7.0 or later. The power Debugger sends runtime power measurement and application debug data to the Data Visualizer.



(/local--files/8avr:low-power-example/power-debug.png)

The Power Debugger has two independent current sensing channels for measuring and optimizing the power consumption of a design.
The 'A' channel is the recommend channel for accurately measuring low currents.
The 'B' channel is the recommended channel for measuring higher currents with lower resolution.

Hardware Setup

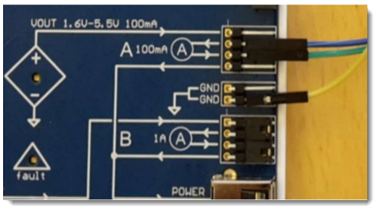
Connecting the Power Debugger to the ATmega328PB Xplained Board.

The 'A' and 'B' channel current measurement ports on the Power Debugger board are depicted with ammeter symbols on the silkscreen. The voltage supply is connected to the input of the ammeter, and the load (target) is connected to the output. With the following steps, the power debugger measures the consumption on the AVR core.
Table 1-1 Power Debugger and **ATmega328PB Xplained Mini** connection.

Power Debugger	ATmega328PB Xplained Mini
Port A input : Blue wire	5V
Port A output: Green wire	VCC
GND : Yellow Wire	GND

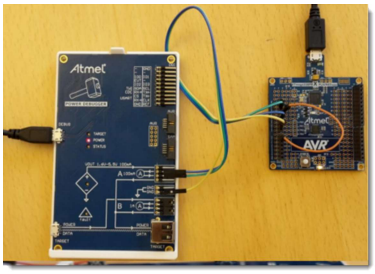
(/local--files/8avr:low-power-example/table.png)

Figure 1-1 Power Debugger and **ATmega328PB Xplained Mini** connection.



(/local--files/8avr:low-power-example/connection.png)

Hardware Configuration



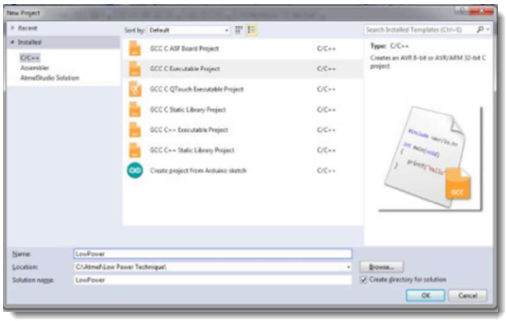
(/local--files/8avr:low-power-example/hw.png)

Measuring the Current in Active Mode

Procedure

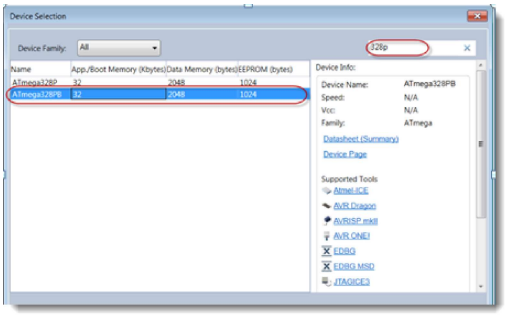
A Project Creation

- Open Atmel Studio 7
- Select **File > New > Project**
- Select **GCC C Executable Project** and give it the name **LowPower.**
- Choose a location to save the project on your computer.



(/local--files/8avr:low-power-example/location.png)

- When the Device Selection window appears, enter 328P into the search window and then select the Atmega328PB. Click OK.



(/local--files/8avr:low-power-example/device-

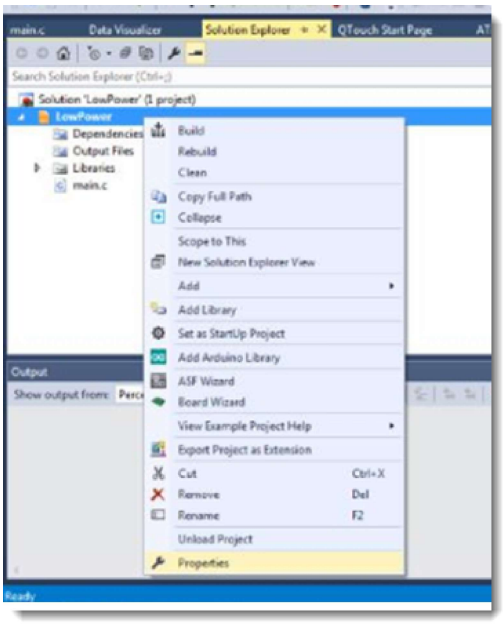
selection.png)

- A project will be created containing an empty `while(1) loop` in `main()`.

```
<font></font>
#include <avr/io.h><font></f
<font></font>
int main(void)<font></font>
{<font></font>
    /* Replace with your app
    while (1) <font></font>
    {<font></font>
    }<font></font>
} <font></font>
<font></font>
```

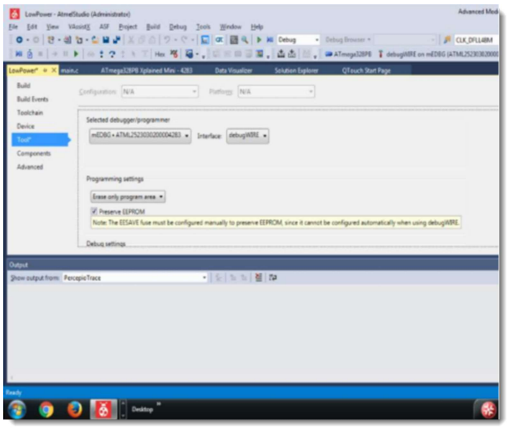
Select **View > Solution Explorer** from the menu bar.

In the Solution Explorer, right click the project and select **Properties.**



(/local--files/8avr:low-power-example/se.png)

Under **Tool**, Select **mEDBG** and **debugWIRE**



(/local--files/8avr:low-power-example/debugwire.png)

B Add Code to the Project

- Add the following two statements in `main()` to call `TMR0_init` and set an I/O pin as an output:

```
<font></font>
TMR0_init();<font></font>
<font></font>
DDRB |= 1<<DDRB5;   // Direction o
<font></font>
```

Add the Timer 0 initialization routine

```
<font></font>                                    ?
/**********************************
                                        TM
**********************************
void TMR0_init( void ){<font></fon
<font></font>
    // enable timer overflow inter
    TIMSK0=(1<<TOIE0) <font></font
    // set timer0 counter initial
    TCNT0=0x00;<font></font>
    // start timer0 with /1024 pre
    TCCR0B = (1<<CS02) | (1<<CS00)
<font></font>
    // enable interrupts<font></fo
    sei(); <font></font>
<font></font>
}<font></font>
<font></font>
```

- Provide the TMR0 interrupt service routine to the project, and make the LED blink at about 1 Hz.

```
<font></font>                                    ?
uint8_t    count;<font></font>
ISR(TIMER0_OVF_vect){<font></font>
<font></font>
    count++;<font></font>
    if(count==20){<font></font>
        count=0;<font></font>
        // Toogle LED<font></font>
        PORTB=PORTB ^ 0x20;<font></f
    }<font></font>
}<font></font>
<font></font>
```

The complete program is as follow,

```
<font></font>                                    ?
#include <avr/io.h><font></font>
#include "avr/interrupt.h"<font></
<font></font>
    uint8_t    count;<font></font>
/* Timer 0 Interrupt */<font></fon
ISR(TIMER0_OVF_vect){<font></font>
<font></font>
    count++;<font></font>
    if(count==20){<font></font>
        count=0;<font></font>
        // Toogle LED<font></font>
        PORTB=PORTB ^ 0x20;<font><
    }<font></font>
}<font></font>
<font></font>
int main(void)<font></font>
{<font></font>
    TMR0_init();<font></font>
<font></font>
    DDRB |= 1<<DDRB5;   // Directi
    /* Replace with your applicati
    while (1) <font></font>
    {<font></font>
    }<font></font>
}<font></font>
<font></font>
/**********************************
                                        TM
**********************************
void TMR0_init( void ){<font></fon
<font></font>
    // enable timer overflow inter
    TIMSK0=(1 <<TOIE0) ;<font></fo
<font></font>
    // set timer0 counter initial
    TCNT0=0x00;<font></font>
<font></font>
    // start timer0 with /1024 pre
    TCCR0B = (1 <<CS02) | (1<<CS00
<font></font>
    // enable interrupts<font></fo
    sei(); <font></font>
<font></font>
}<font></font>
<font></font>
```

C Verify the Project Runs

- Build the project and program the device by selecting **Debug > Continue.**

> The LED will blink if the project is working correctly

- Ensure debugging for the project is terminated by selecting **Debug > Disable debugWire** then **Close.**

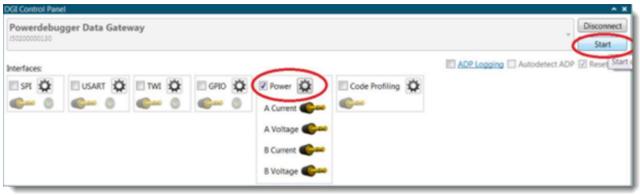
D Measure the Power Consumption in Active Mode

- In Atmel Studio 7, open the menu **Tools > Data Visualizer**

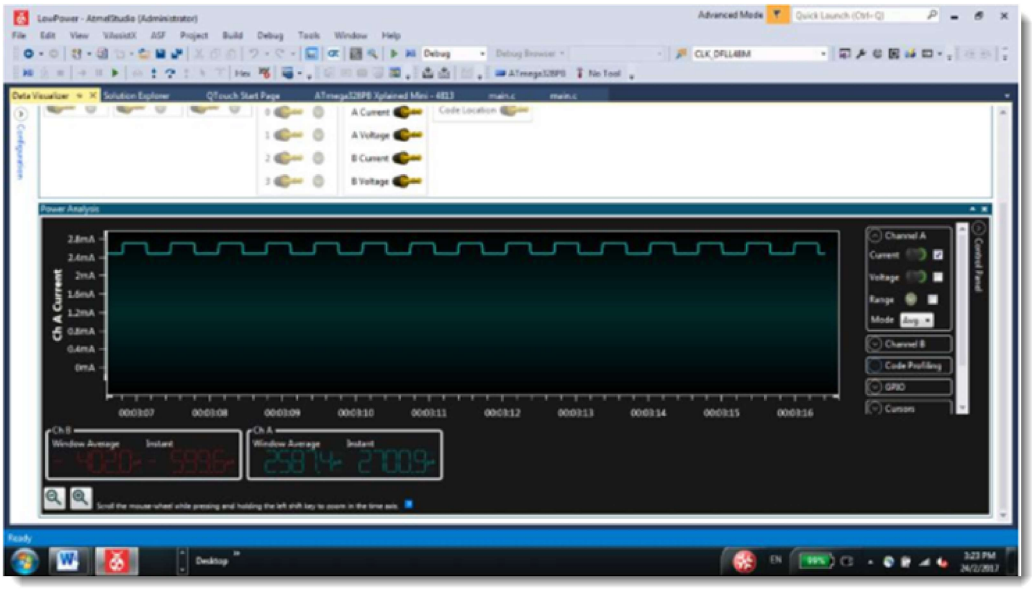
Power Debugger Data Gateway should be selected by default in DGI Control Panel, select it if not. Click on **Connect.**

(/local--files/8avr:low-power-example/connect.png)

- Select **Power** and **Start**



(/local--files/8avr:low-power-example/power-and-start.png)



(/local--files/8avr:low-power-example/power-monitor.png)

Turn off the target Power Supply to Xplained board and enable power measurement

(/local--files/8avr:low-power-example/enable-power.png)

- Close the Device Programming
- Verify that the average current consumption is about 2.6 mA

Reducing the power consumption

There are several methods you can use to reduce the power consumption of an AVR® microcontroller. This example reduces power by:

- turning off unused peripherals
- Stopping leakage current on the digital I/O pins
- Using sleep mode. Power consumption optimization techniques are implementing in the function `Optimize_power_consumption()` which is called from `main()`.

Disable digital input buffers and Analog comparator

For analog input pins, the digital input buffer should be disabled.

The digital input buffer measures the voltage on the pin and represents the value as a logical one or zero. An analog signal level close to VCC/2 on an input pin can cause significant additional current consumption. If the pin is used as analog pin there is no need to know if the analog signal's digital value would be a one or zero; these digital pins should be disabled.

Turn off unused peripherals

Disable the peripherals not used in the application. The **Power Reduction Register** (**PRR0**) and (**PRR1**) can stop the clock to individual peripherals to reduce power consumption. Resources used by the peripheral remain occupied when the clock is stopped. In most instances, peripherals should be disabled before the clock is halted.

1. Include the <power.h> file in the main program
#include <avr/power.h>
2. Add below code to `optimize_power_consumption()`.

```
<font></font>
/* Power shutdown to unused periph
PPR0 = 0xDF;<font></font>
PPR1 = 0x3F;<font></font>
<font></font>
```

3. Add and #include for <wdt.h> to main.c.
#include <avr/wdt.h>
4. Add the following code in `optimize_power_consumption()` to turn off watch dog timer.

```
<font></font>                    ?
/* Disable interrupts*/<font></fon
Cli();<font></font>
Wdt_reset();<font></font>
MCUSR&= ~(1<<WDRF);<font></font>
<font></font>
```

Apply pull-up resistors

Unused and unconnected pins consume power. The un-needed power load of floating pins can be avoided by using the AVR's internal pull-up resistors on the unused pins. Port pins can be set to input pull-ups by setting the **DDxn** bit to 0 and **PORTxn** bit to 1. (where **x** is PORT B,C,D,E and **n** is 0 to 7)

```
1  <font></font>                ?
2      /* Unused pins set as inpu
3          DDRB  &= 0xE0;<font></
4          DDRC &= 0xC9;<font></f
5          DDRD &= 0x03;<font></f
6          DDRE &= 0xF3;<font></f
7  <font></font>
8          PORTB |= ~(0xE0);<font
9          PORTC |= ~(0xC9);<font
10         PORTD |= ~(0x03);<font
11         PORTE |= ~(0xF3);<font
12 <font></font>
```

Use Sleep function

Sleep mode allows the application to save power by shutting down unused modules. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application’s requirements.

1. **Include <avr/sleep.h>** header file from AVR library at the top of file.
2. Set the desired sleep mode in the `optimize_power_consumption ()` function by configuring the microcontroller to use the sleep mode **SLEEP_MODE_PWR_DOWN** for minimum power consumption.

```
<font></font>                    ?
              /* Set sleep mode */<
            set_sleep_mode(SLEEP_MOD
<font></font>
```

3. Call the `function sleep_mode()` from `main()` to enter sleep mode.

```
1  <font></font>                ?
2                           while
3
4                              sle
5          }<font></font>
6  <font></font>
```

Using Pin Change Interrupt

To wake up the microcontroller from **SLEEP_MODE_PWR_DOWN** the **Pin Change Interrupt** is used. The **ATmega328PB Xplained Mini** board's switch (SW0) is connected to **PB7**. Pin **PB7** is the source for the pin change interrupt. Make the following additions to `main()`:

#include avr/interrupt.h

Configuring bit PCINT7 and PCICR register:

```
<font></font>                    ?
 PCMSK0 |= (1<<PCINT7); //Enable P
    PCICR |= (1<<PCIE0); //Enable
    sei();<font></font>
<font></font>
```

Add the interrupt service routine:

In order to enable the ISR routines, the vector name for the PCINT7 is ISR (PCINT0_vect), defined in device header file.

```
<font></font>                    ?
   ISR (PCINT0_vect)<font></font>
{<font></font>
     if (!(PINB & (1<<PINB7))) //
    {<font></font>
        PORTB |= (1 <<PORTB5); //
    }<font></font>
    else<font></font>
    {<font></font>
        PORTB &= ~(1<<PORTB5); //
    } <font></font>
}        }<font></font>
<font></font>
```

Completed Code

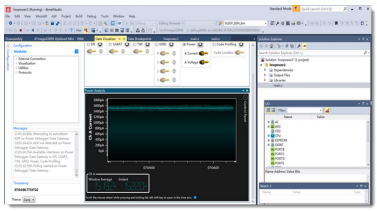
After making the previous changes the complete code should like the following:

```
<font></font>                    ?
/*<font></font>
 * lowpower2.c<font></font>
 */ <font></font>
<font></font>
//#include <avr/io.h><font></font>
<font></font>
#include <avr/io.h><font></font>
#include "avr/interrupt.h"<font></
#include "avr/wdt.h"<font></font>
#include "avr/sleep.h"<font></font
<font></font>
#include <avr/power.h><font></font
#include <avr/interrupt.h><font></
//#include <util/delay.h><font></f
<font></font>
void    optimize_power_consumption
<font></font>
    /* Disable digital input buffe
    DIDR0 = (1 << ADC5D) | (1 << A
    DIDR0 |= 0xC0 ;   /*ADC7D and
<font></font>
    /* Disable digital input buffe
    DIDR1 |= (1 << AIN1D) | (1 <<
    /* Disable Analog Comparator *
    ACSR |= (1 << ACD);<font></fon
<font></font>
    /*Power shutdown to unused per
    PRR0 = 0xDF;<font></font>
    PRR1 = 0x3F;<font></font>
    /*Unused pins set as input pul
    DDRB &= 0xE0;<font></font>
    DDRC &= 0xC9;<font></font>
    DDRD &= 0x03;<font></font>
    DDRE &=0xF3;<font></font>
<font></font>
    PORTB |=~(0xE0);<font></font>
    PORTC |=~(0xC9);<font></font>
    PORTD |=~(0x03);<font></font>
    PORTE |=~(0xf3);<font></font>
<font></font>
    /*Watchdog Timer OFF*/<font></
<font></font>
    /* Disable interrupts */<font>
    cli();<font></font>
    /* Reset watchdog timer */<fon
    wdt_reset();<font></font>
    /* Clear WDRF in MCUSR */<font
    MCUSR &= ~(1<<WDRF);<font></fo
    /* Turn off WDT */<font></font
    WDTCSR = 0x00;<font></font>
<font></font>
    /* Set sleep mode */<font></fo
    set_sleep_mode(SLEEP_MODE_PWR_
<font></font>
}<font></font>
<font></font>
/*********************************
                                TM
*********************************
 void TMR0_init( void ){<font></fo
<font></font>
    // enable timer overflow inter
    TIMSK0=(1<<TOIE0) ;<font></fon
<font></font>
    // set timer0 counter initial
    TCNT0=0x00;<font></font>
```

```
<font></font>
    // start timer0 with /1024 pre
    TCCR0B = (1<<CS02) | (1<<CS00)
<font></font>
    // enable interrupts<font></fo
    sei(); <font></font>
<font></font>
}  <font></font>
     uint8_t    count;<font></font
/* Timer 0 Interrupt */<font></fon
 ISR(TIMER0_OVF_vect){<font></font
<font></font>
    count++;<font></font>
    if(count==20){<font></font>
        count=0;<font></font>
        // Toogle LED<font></font>
        PORTB=PORTB ^ 0x20;<font><
    }<font></font>
}<font></font>
<font></font>
ISR (PCINT0_vect)<font></font>
{<font></font>
     if (!(PINB & (1<<PINB7))) //
    {<font></font>
        PORTB |= (1<<PORTB5); // T
    }<font></font>
    else<font></font>
    {<font></font>
        PORTB &= ~(1<<PORTB5); //
    } <font></font>
}<font></font>
<font></font>
int main(void)<font></font>
{<font></font>
    TMR0_init();<font></font>
<font></font>
    DDRB |= 1<<DDRB5;   // Directi
    DDRB &= ~(1<<DDB7); //Set PORT
<font></font>
    optimize_power_consumption();<
<font></font>
    PCMSK0 |= (1<<PCINT7); //Enabl
    PCICR |= (1<<PCIE0); //Enable
    sei();<font></font>
<font></font>
    //set_sleep_mode(SLEEP_MODE_PW
    //sleep_enable();<font></font>
    //sleep_cpu();<font></font>
<font></font>
    /* Replace with your applicati
    while (1) <font></font>
    {<font></font>
        sleep_mode();<font></font>
    }<font></font>
}<font></font>
<font></font>
```

Program the application and measure power consumption.

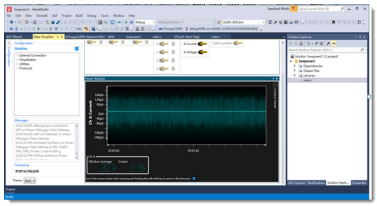
1. Program the code by selecting **Debug > Continue.**
2. Wait until the application is programmed and the message at the bottom of the window appears as **Running.**
3. Exit the debugger by selecting **Debug > Disable debugWire** then **Close.**
4. Open the **DataVisualizer window** and check the power consumption as shown below.



(/local--files/8avr:low-power-example/one-five.png)

Current consumption observed should be down to around 1.52 mA.

- Set target power switch off: **Tools > Device Programming > Tools** setting



(/local--files/8avr:low-power-example/threema.png)

Current consumption should have been lowered to approximately 20.7 µA.

> Note that the Timer does not wake the device from SLEEP mode. The application is set to exit sleep mode when the **Pin Change Interrupt** occurs.

Conclusions

This example demonstrated several different techniques to lower the power consumption of an AVR application. The power consumption can be significantly reduced by:

- Intelligent design
- Using sleep modes
- Switching off unused peripherals

This example used the **Atmel Studio 7 Data Visualizer** and the **Power Debugger Board** to measure the power.

Capacitación sobre ADC y optimización de energía para microcontroladores tinyAVR® y megaAVR®

Introducción:

Este tutorial contiene cinco aplicaciones prácticas para la conversión de datos ADC, con el consumo de corriente medido para cada aplicación. El tutorial comienza con una sencilla aplicación de conversión de ADC. En las siguientes aplicaciones, se introducen diferentes técnicas para demostrar cómo se puede reducir el consumo de corriente en los microcontroladores de las series **tinyAVR** ® 0 y 1 y **megaAVR** ® 0.

Este tutorial también demuestra cómo usar Atmel START para comenzar con el desarrollo de aplicaciones ADC de dispositivos AVR® . Las aplicaciones ADC se han desarrollado paso a paso en Atmel Studio. Este tutorial se ha desarrollado en el kit de evaluación ATtiny817 Xplained Pro, pero debería ser aplicable para todos los dispositivos tinyAVR 0- y 1-series, y megaAVR 0-series.

Los proyectos de solución para cada una de las asignaciones están disponibles en el navegador de ejemplo Atmel START (http://start.atmel.com/#examples) .
En la categoría 'Primeros pasos', busque ADC y solución de optimización de energía (1-5).
Los enlaces directos a los proyectos de ejemplo relevantes se proporcionan en las descripciones de tareas a continuación.

- ADC y optimización de energía: manual tutorial práctico (http://ww1.microchip.com/downloads/en/DeviceDoc/40002008A.pdf)

Requisitos previos de hardware

- Kit de evaluación ATtiny817 Xplained Pro
- Cable micro-USB (Tipo-A/Micro-B)
- un potenciometro
- Tres cables macho a hembra
- conexión a Internet

Requisitos previos del software

- Estudio Atmel 7.0
- Navegador web. La lista de navegadores compatibles se puede encontrar aquí: START navegadores compatibles (http://start.atmel.com/static/help/index.html?GUID-51435BA6-0D59-4458-A413-08A066F6F7CA)

Tiempo estimado de finalización: 120 minutos

Tarea 1: Conversión de ADC con la aplicación de impresión USART (http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN1/redirect)

En esta tarea, se utiliza Atmel Studio para desarrollar una aplicación utilizando controladores ADC y USART de Atmel START. El ADC está configurado para funcionar en modo de conversión única y se conecta un potenciómetro al pin de entrada del ADC para estudiar la funcionalidad del ADC. Los datos del ADC se envían a través de USART al terminal integrado en el visualizador de datos de Atmel Studio. En Data Visualizer, el consumo actual de la aplicación se analiza utilizando el Power Analyzer incorporado.

Solución de la tarea (http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3A%3AApplication%3AADC_and_Power_Optimization_Solutio

Assignment 2: RTC Interrupts Triggers ADC and USART Print (http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN2/redirect)

In this assignment, the Real Time Counter (RTC) module is used. The RTC overflow interrupt is used to trigger an ADC conversion every half second. ADC Result Ready (RESRDY) interrupt triggers a print of the ADC result to the USART terminal. When RTC overflow interrupt is not triggered, the device is kept in Sleep Standby mode in order to reduce the power consumption. Atmel START is used to add the RTC module and to configure the RTC, ADC, CPUINIT, and SLEEPCTRL drivers. An Atmel Studio project is regenerated afterward.

Assignment 2 (http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3A%3AApplication%3AADC_and_Power_Optimization_Solutio

---

Assignment 3: Power Optimization on I/O Pins (http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN3/redirect)

In this assignment, the digital input buffer on the I/O pins is disabled in order to reduce the current consumption. The current consumption is further reduced when the USART TX pin is configured as a high impedance pin during no data transmission period. The same drivers and configurations from the previous assignment is used here. Atmel Studio is used to further develop the code.

Assignment 3 (http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3A%3AApplication%3AADC_and_Power_Optimization_Solutio

---

Assignment 4: ADC Conversion Using Window Compare Mode (http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN4/redirect)

In this assignment, the ADC result ready interrupt is replaced by the ADC WCMP interrupt, to trigger a USART transmission. In this case, the ADC result, which is below ADC window threshold value, triggers USART transmission. The ADC results, which are above the window threshold value, is ignored and not trigger any USART transmission. Atmel START is used to reconfigure the ADC module and the Atmel Studio project is updated with the new configuration.

Assignment 4 (http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3A%3AApplication%3AADC_and_Power_Optimization_Solutio

---

Assignment 5: Event System (EVSYS) Used to Replace the RTC Interrupt Handler (http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN5/redirect)

In this assignment, the event system with the RTC overflow event signal, instead of the RTC overflow interrupt, is used to trigger an ADC conversion. The Event System enables direct peripheral-to-peripheral signaling. It allows a change in one peripheral (the Event Generator) to trigger actions in other peripherals (the Event Users) through Event channels without using the CPU. A channel path can be either asynchronous or synchronous to the main clock.

Assignment 5 (http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3A%3AApplication%3AADC_and_Power_Optimization_Solutio

---

Summary

This tutorial contains five hands-on applications doing ADC data conversion, with current consumption measured for each application. It starts with a simple ADC conversion application and different techniques are introduced in order to demonstrate how the current consumption can be reduced. This is a useful foundation for developing future ADC applications with specific current consumption requirements.