AVR4013: picoPower Basics

Features

- **Sleep modes**
- **PRR registers**
- **Oscillator calibration**
- **USART**

1 Introduction

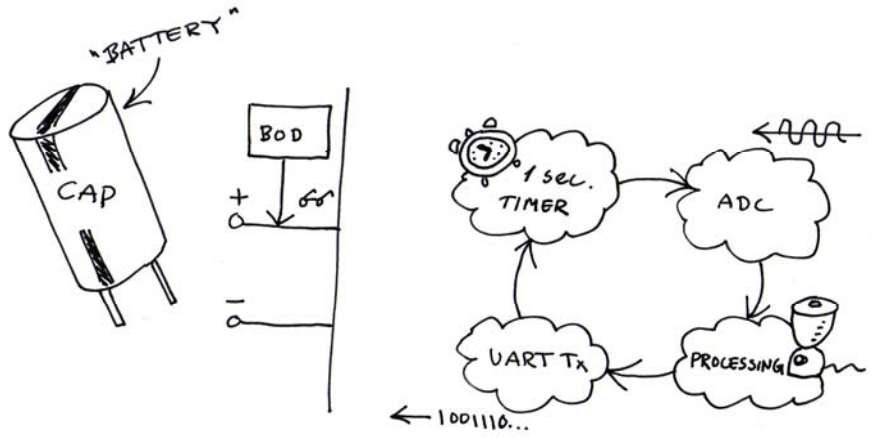
In this demonstration we will show you how to extend the battery life of our application by multiple factors by modifying only the firmware. You will see that while some of the modifications are very simple and only require setting some registers, other modifications will need some rewriting of the code.

We will use a capacitor as a battery in this demonstration because it is easy to recharge with the same amount of energy. We will use the discharge time as a measure of power consumption.

Our demonstration application will do one ADC measurement, perform 1000 cycles worth of processing, and send the string “More oomph to your amps, picoPower!” and the iterations count on the UART. This will be repeated each second. Before reading this application note, you should download the datasheet for the ATmega88PA from

http://www.atmel.com/dyn/resources/prod_documents/doc8271.pdf

Figure 1-1. Demonstration application



8-bit AVR® Microcontrollers

Application Note

Rev. 8349A-AVR-11/10

2 Schematics

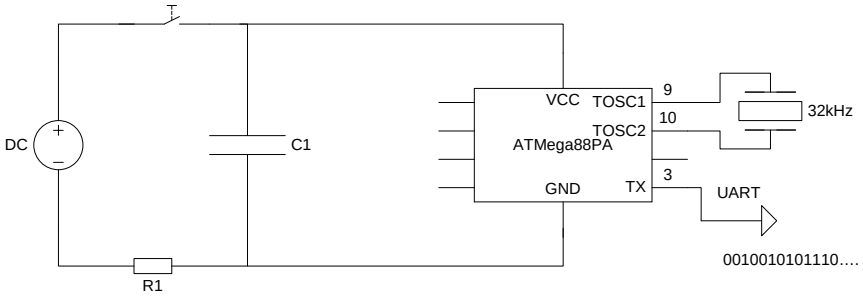


Figure 2-1. picoPower demonstration schematics

From Figure 2-1 we can see that the demonstration circuit is quite simple. We have a DC source set at 5.5V, which is used to charge the capacitor when the switch is pushed. When the switch is released, the ATmega88PA will run off the energy stored in the capacitor, C1. The main clock source for the device is the internal RC oscillator. Timer 2 is connected to an external crystal and will work in asynchronous mode to be able to function in some of the sleep modes which we will make use of in the demonstration.

3 The code

The code is meant to simulate a sensor device that transmits data once a second. After the initialization (which will differ with each optimization method), the code will enter an infinite loop.

This loop is separated into stages:

- ADC conversion
- Simulated 1000 cycles of processing
- Convert number in to ASCII string
- Send data over UART (More oomph to your amps, picoPower! [iteration nr])
- Repeat

The loop will start by doing one ADC conversion to simulate data gathering from some analog device. Then the device will enter a function that simulates 1000 cycles of data processing. In many applications, converting a number to its ASCII representation is preferred, and so we convert the iteration count of the loop to ASCII. Finally, we send the string “More oomph to your amps, picoPower! [iteration count]” over the UART. Then the loop repeats.

4 Source code availability

The code for all five optimization levels is included in a separate .zip file, and has been tested with AVR Studio 4.18 RC2 and the WinAVR-20100110 toolchain.

5 Demonstration stages

The code consists of five levels of optimization, each level offering increased power saving in the application.

- No optimization
- Enable pull-ups on unused I/O pins and disable modules not used
- Pre-scale clock from 8Mhz to 2MHz
- Use power-save sleep mode while waiting for next transmit
- Calibrate oscillator to enable higher baud rate

Before we start the demonstration we must calibrate the internal RC oscillator. This is to make sure that the baud rate generator is able to generate a stable clock source over the entire voltage range.

Oscillator calibration is highly device depended, and referring to the ATmega88PA datasheet, the factory calibration of the internal RC oscillator has an accuracy of ±10% (see section 28.4.1, Calibrated Internal RC Oscillator Accuracy, in the datasheet) and user calibration accuracy is within ±1%.

To do this, you can use an oscilloscope and enable the CKOUT fuse (see section 27.2, Fuse Bits, in the datasheet) to find the frequency closest to 8MHz.

You will use the OSCCAL register to calibrate the oscillator (see OSCCAL – Oscillator Calibration Register in the datasheet), and refer to application note AVR053 for a good way to calibrate the oscillator.

On the ATmega88PA used in this example, we found that the value 0x66 in the OSCCAL register gave the best frequency.

5.1 No optimization

We start by running the code with no optimization to create a reference point. We execute the code with the device set at 8MHz and the baud rate generator set to 19200 baud.

While waiting for the next transmit, the controller will poll the TIFR2 Timer/Counter 2 Interrupt Flag Register to check if a second has elapsed since the last transmit started. This approach will run the code in active mode all the time

Runtime: 6 seconds.

5.2 Enable pull-ups on unused I/O pins and disable modules not used

To remove some power consumption, we will enable pull-ups on unused I/O pins to get a defined logical level and avoid unnecessary switching. To enable pull-ups on the I/O pins, we set the pins to input in the Port Data Direction Register and set the Port Data Register to high level (see section 13.2.1, Configuring the Pin, in the datasheet).

Code:

```
DDRA = 0x00;  // Set direction to input on all pins
PORTA = 0xFF; // Enable pull-ups on pins
```

Another power-saving feature is to disable unused on-chip modules in the PRR Power Reduction Register (see section 9.10, Minimizing Power Consumption, in the datasheet).

We are not using the TWI, Timer/Counter 0, Timer/Counter 1, and Serial Peripheral Interface modules, and will therefore disable these modules.

Code:

```
void enable_prr()
{
    power_spi_disable();
    power_timer0_disable();
    power_timer1_disable();
    power_twi_disable();
}
```

Runtime is increased to 9 seconds.

5.3 Pre-scale clock from 8MHz to 2MHz

Because we have been using the device at 8MHz, we have to configure the BOD (brown out detection) level at 2.7V. The ATmega88PA is not rated to run at lower voltages with the clock set at 8MHz (see section 28.3, Speed Grades, in the datasheet). Therefore, we will halt the execution at 2.7V, which means that a lot of energy will be left in the capacitor which we will not be able to utilize with this setting.

We do not need the processing capacity of a device running at 8Mhz, and should instead decrease the system frequency to 2MHz, which will lower overall power consumption and allow us to change the BOD setting to 1.8V.

We will still be able to generate a stable baud rate of 19200. To pre-scale the RC oscillator output with 4 to get 2MHz (8MHz / 4 = 2MHz) we use the CLKPR Clock Prescale Register.

The AVR toolchain includes a macro for this that is found in the power.h header file.

Code:

```
clock_prescale_set(clock_div_4); // This divides the clock by 4
```

Runtime is now increased to 40 seconds.

5.4 Use power-save sleep mode while waiting for next transmit

As the ATmega88PA is not processing any data while waiting for its next UART data transmission, it's a good idea to put the device into sleep mode to further reduce power consumption

In the power-save sleep mode (see section 9.1, Sleep Modes, in the datasheet) we can use Timer 2 as our wakeup source.

Brown out detection is not required in the power-save sleep mode because no data can be corrupted. It will be automatically re-enabled on wakeup from sleep (see section 9.2, BOD Disable, in the datasheet), so we choose to disable the BOD while sleeping

The AVR toolchain includes a header file, sleep.h, which has functions for setting sleep mode, disabling BOD, and putting the device into sleep. These functions are called:

- set_sleep_mode() – selects which sleep mode to use when the sleep instruction is executed
- sleep_enable() – enables sleep modes
- sleep_bod_disable() – disables BOD while sleeping
- sleep_cpu() – executes the sleep instruction

Runtime is now 198 seconds.

5.5 Calibrate oscillator to enable higher baud rate

Another approach to reduce power consumption is to reduce the time spent in active mode.

In this application, the CPU is active while waiting for the UART transmission to end. We will calibrate the oscillator to a frequency that allows a higher baud rate on the UART to reduce the time spent in active mode.

We need a frequency close to 7.3728MHz to generate a stable baud rate of 115.2kbps (see section 19.10, Examples of Baud Rate Setting, in the datasheet). You will need to set the frequency a bit higher than 7.3728MHz to be able to utilize the whole voltage range. You will also want to divide the clock by 4 to get 1.8432MHz to still be able to run the device with the 1.8V BOD setting.

We change the OSCCAL value to 0x5e.

Runtime is now increased to 217 seconds.

5.6 Conclusion

By structuring your application and understanding the power saving techniques mentioned in this document, you can reduce energy consumption without doing any changes to your PCB.

In this example, the runtime was increased from 6 to 217 seconds running off the same power source.

6 References

[1]: ATmega88PA datasheet:
http://www.atmel.com/dyn/resources/prod_documents/8271S.pdf
[2]: Application note AVR053:
http://www.atmel.com/dyn/resources/prod_documents/doc2555.pdf

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

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

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