Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Programmers and Debuggers

Power Debugger

USER GUIDE

Table of Contents

1. The Atmel Power Debugger

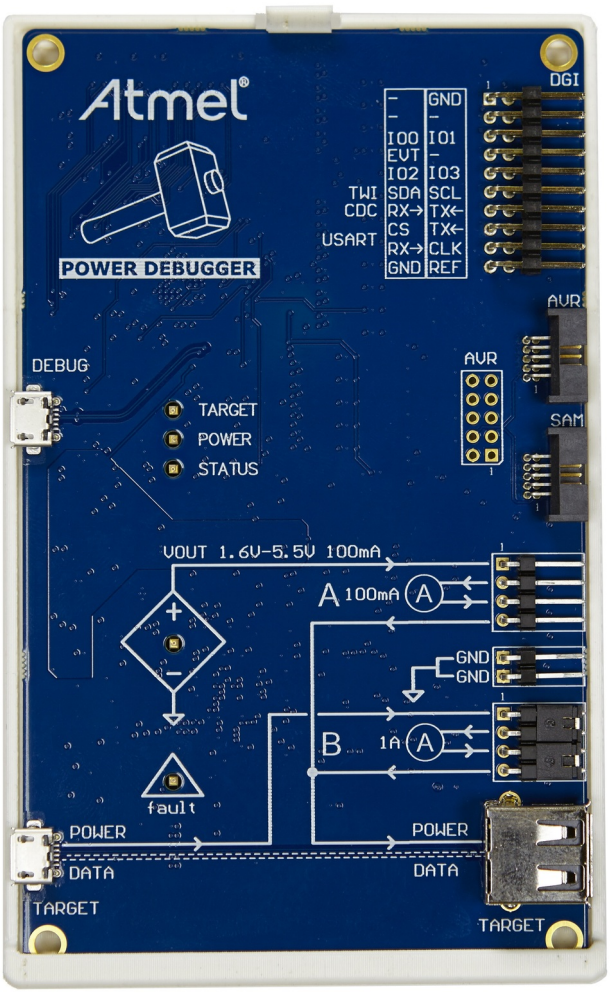
Atmel® Power Debugger is a powerful development tool for debugging and programming ARM® Cortex®-M based Atmel SAM and Atmel AVR® microcontrollers using JTAG, SWD, PDI, UPDI, debugWIRE, aWire, TPI, or SPI target interfaces.

In addition the Atmel Power Debugger has two independent current sensing channels for measuring and optimizing the power consumption of a design.

Atmel Power Debugger also includes a CDC virtual COM port interface as well as Atmel Data Gateway Interface channels for streaming application data to the host computer from a SPI, USART, TWI, or GPIO source.

The Atmel Studio can be downloaded from http://www.atmel.com/tools/atmelstudio.aspx and the Atmel Data Visualizer from https://gallery.atmel.com/Products/Details/2f6059f5-9200-4028-87e1-ba3964e0acc2.

The Power Debugger is a CMSIS-DAP compatible debugger which works with Atmel Studio 7.0 or later, or other front-end software capable of connecting to a generic CMSIS-DAP unit. The Power Debugger streams power measurements and application debug data to Atmel Data Visualizer for real-time analysis.

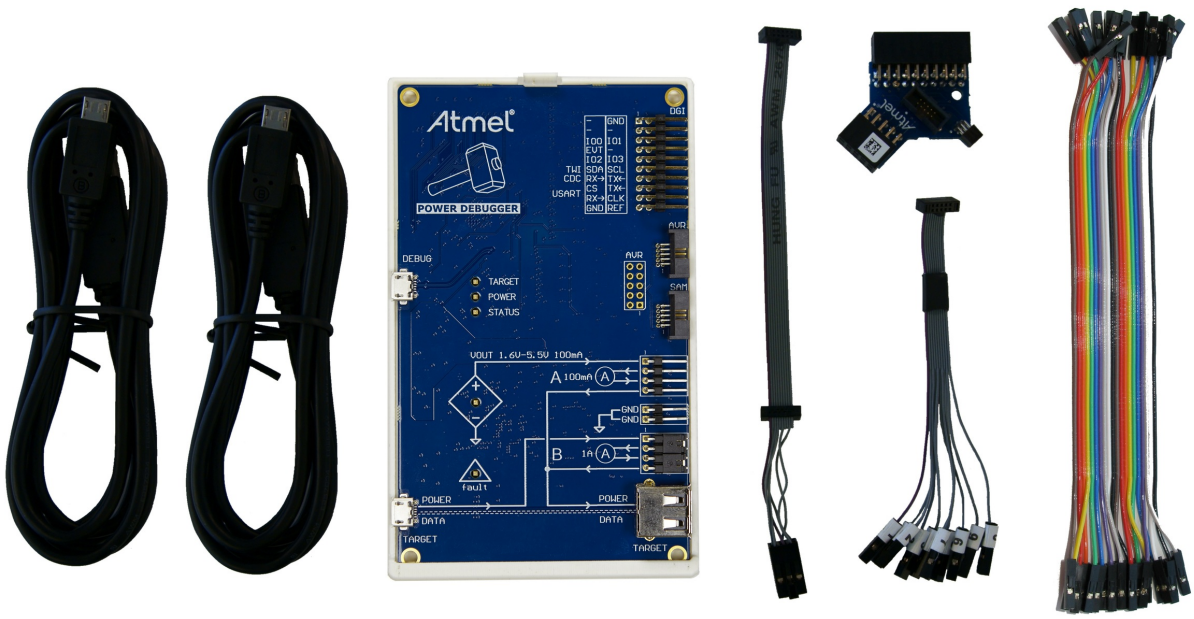


1.1. Kit Contents

The Power Debugger kit contains:

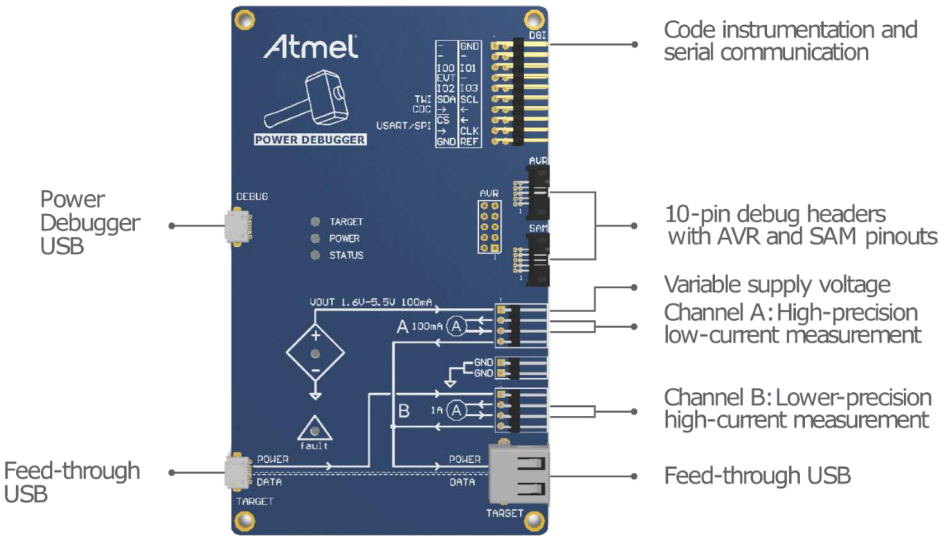
- One Power Debugger unit with plastic back-plate
- Two USB cables (1.5m, high-speed, Micro-B)
- One adapter board containing 50-mil AVR, 100-mil AVR/SAM, and 100-mil 20-pin SAM adapters
- One IDC flat cable with 10-pin 50-mil connector and 6-pin 100-mil connector
- One 50-mil 10-pin mini squid cable with 10 x 100-mil sockets
- 20 x 100-mil separable strap cables

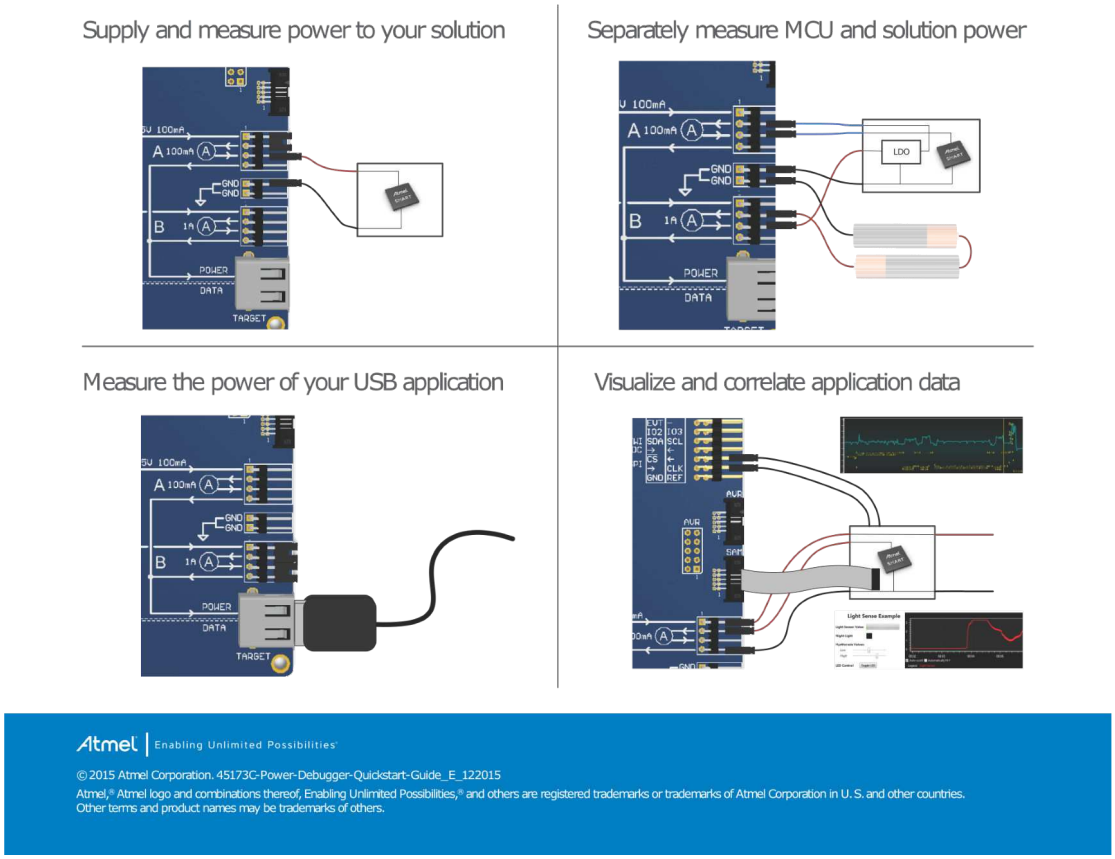
Figure 1-1. Power Debugger Kit Contents



2. Getting Started with the Power Debugger

Getting started with the Power Debugger is simple, especially if one is familiar with the Atmel-ICE programmer and debugger or any of the Atmel Xplained Pro kits.





Tip: The Quick Start Guide shown here is included with the kit for convenience.

The use case section contains descriptions of how the Power Debugger can be put to use analyzing power consumption in a number of scenarios. For a quick and easy introduction to Power Debugging, select the use case which closest matches your current project or available boards.

The Hardware Description section provides technical details of the capabilities of the Power Debugger for users who prefer to get going with custom boards.

Related Links

Hardware Description on page 84

3. Connecting the Power Debugger

The many connection possibilities of the Power Debugger are laid out in a logical manner with the silkscreen serving as a functional reference for most connectors. A connection overview is given here:

USB 'DEBUG' connector: Must be connected to the host computer for debugger power supply and all data transfer.

USB 'TARGET' connectors: Can be optionally used to supply USB power to the target. Data lines are passed through to the corresponding 'TARGET' type A jack. For convenience when debugging USB powered applications, the output to the target USB connector can be easily connected to either 'A' or 'B' channels using a jumper, as indicated by the silkscreen drawing.

DGI 20-pin header rows: Can be optionally connected to target data sources for data streaming to the computer.

AVR and SAM debug headers: To connect to the target device using either AVR 10-pin pinout (or via relevant adapter) or the 10-pin ARM Cortex debug pinout.

VOUT: Optionally provide power from the Power Debugger to the target in the range 1.6V to 5.5V. Up to 100mA can be sourced. Voltage output is configured using the software frontend, and current is sourced from the DEBUG USB connector.

GND: To ensure safe and accurate operation of the Power Debugger, a shared ground is essential.

'A' and 'B' channel current measurement ports: Depicted with ammeter symbols on the silkscreen, these two measurement channels are to be connected in a high-side configuration. This means that the voltage supply is connected to the input of the ammeter, and the load (target) is connected to the output.

Info: The grouping of the 'A' and 'B' channel input and output signals with respect to their neighboring pins is purely for convenience. To route the configurable VOUT voltage source into channel 'A' a jumper can be used. This voltage can be routed to channel 'B' using a wire-strap. The same applies to the voltage provided by the TARGET USB conenctor.

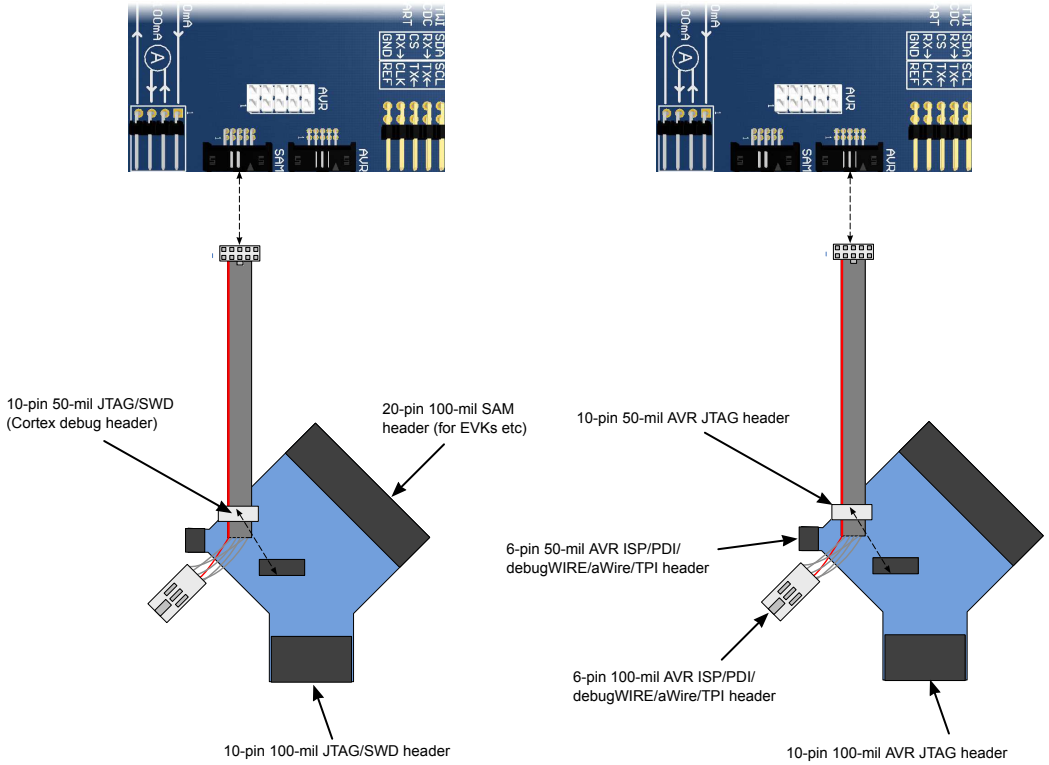
Tip: For common power routing configurations, refer to the printed quick-start guide included in the kit and in the Getting Started section.

3.1. Connecting to AVR and SAM Target Devices

The Power Debugger is equipped with two 50-mil 10-pin JTAG connectors. Both connectors are directly electrically connected, but conform to two different pinouts; the AVR JTAG header and the ARM Cortex Debug header. The connector should be selected based on the pinout of the target board, and not the target MCU type - for example a SAM device mounted in an AVR STK®600 stack should use the AVR header.

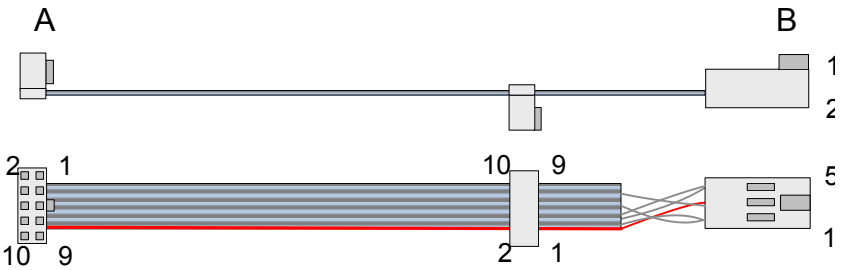
The Power Debugger kit contains all necessary cabling and adapters. An overview of connection options is shown.

Figure 3-1. Power Debugger Connection Options



The red wire marks pin 1 of the 10-pin 50-mil connector. Pin 1 of the 6-pin 100-mil connector is placed to the right of the keying when the connector is seen from the cable. Pin 1 of each connector on the adapter is marked with a white dot. The figure below shows the pinout of the debug cable. The connector marked A plugs into the debugger while the B side plugs into the target board.

Figure 3-2. Debug Cable Pinout



4. Detailed Use Cases

A collection of use cases for the Power Debugger are outlined here. Each use case sets out to solve a given problem in a certain scenario, and then provides a detailed account of how this can be achieved using the Power Debugger.

Low-power (battery) application
Different sleep modes of a basic low-power application are analyzed.

USB application
Using the pass-through USB connection, the overall current consumption of a USB powered board is measured and analyzed.

4.1. Low-power Application

This use case uses a megaAVR® device to introduce you to the Power Debugger. Using the tool with Atmel Studio and Atmel Data Visualizer, we learn how it can be used to measure, analyze, understand, and optimize the power consumption of a typical low-power application. We also look at an example of code instrumentation using the Data Gateway Interface.

The example makes use of the ATmega328PB Xplained Mini board, but could very easily be adapted to suit any simple custom board.

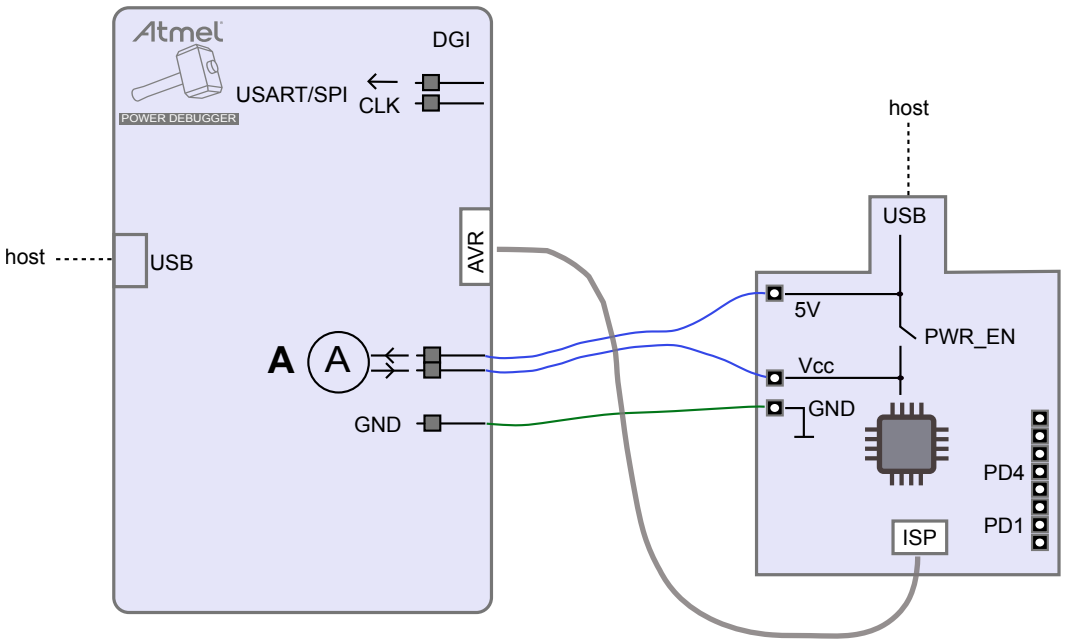
4.1.1. Requirements

To be able to work through this example, the following is required:

- Host computer with Atmel Studio 7 (or later) installed (Atmel Data Visualizer is included)
- Atmel Power Debugger kit (cabling included)
- Atmel ATmega328PB Xplained mini kit, or similar target board
- Pin headers: one 2x3 100-mil ISP header; two 1x8 100-mil headers
- Access to basic soldering equipment

4.1.2. Initial Hardware Setup

A block diagram of the initial setup is shown here.



To get started a few modifications need to be made to the hardware.

In order to measure the current consumed by the target MCU you will need to intercept its power supply and feed it through the measurement circuitry of the Power Debugger. This requires that a power rail header is mounted to the Xplained Mini board.

To do: Mount power rail header on the Xplained Mini board.

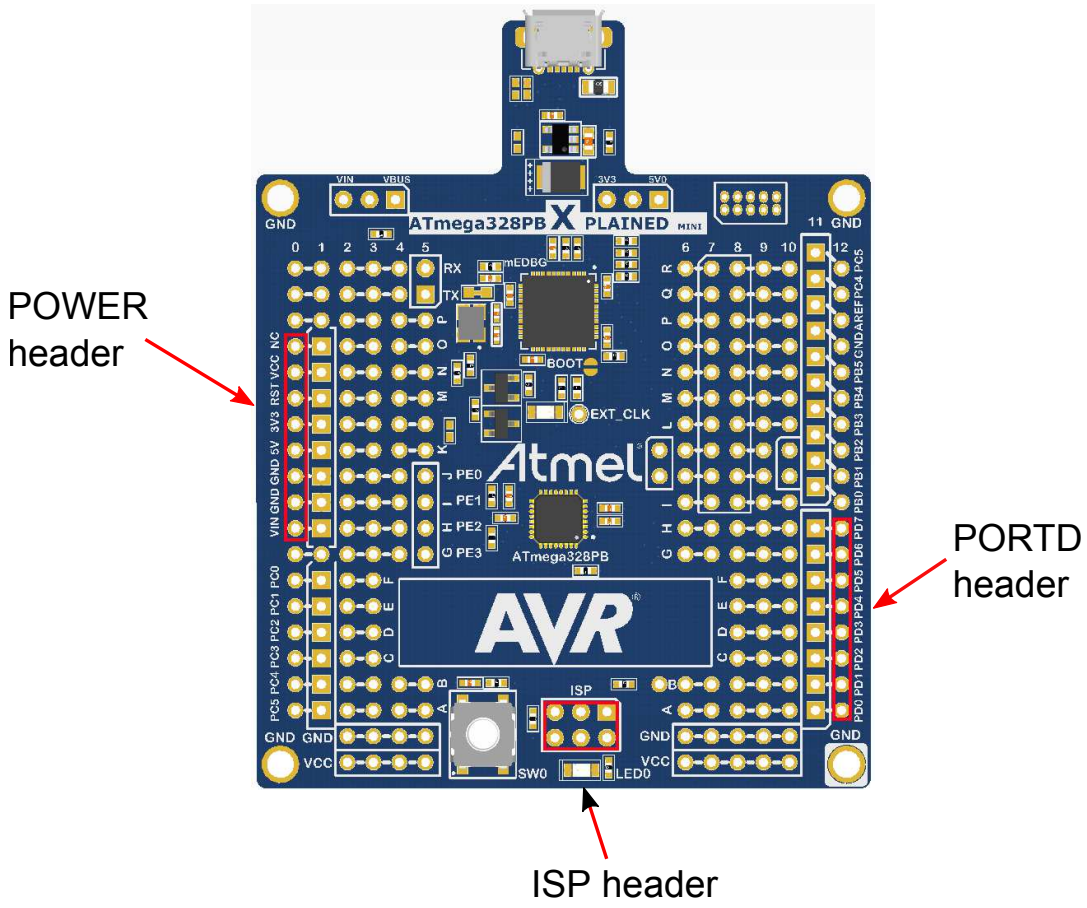
We will make use of code instrumentation to send data from the Xplained Mini board to the Power Debugger DGI. To do this we need access to PORTD features on the target MCU.

To do: Mount PORTD 0..7 header on the Xplained Mini board.

Although the ATmega328PB Xplained Mini board already contains a debugger (mEDBG), it is not going to be used in this example. Instead we would rather make use of the Power Debugger for programming purposes.

To do: Mount the 6-pin 100-mil ISP programming header on the Xplained Mini board.

The headers to be mounted are shown here.



4.1.3. Connections

Once the hardware modifications have been made we are ready to connect the Power Debugger to the target.

To measure the current on the ATmega328PB device you will have to route its supply through the Power Debugger. With the FET switch set to OFF, the ATmega328PB device has no target power source. To route the 5V Xplained Mini supply voltage through the A channel and into the ATmega328PB device's supply rail:

To do:

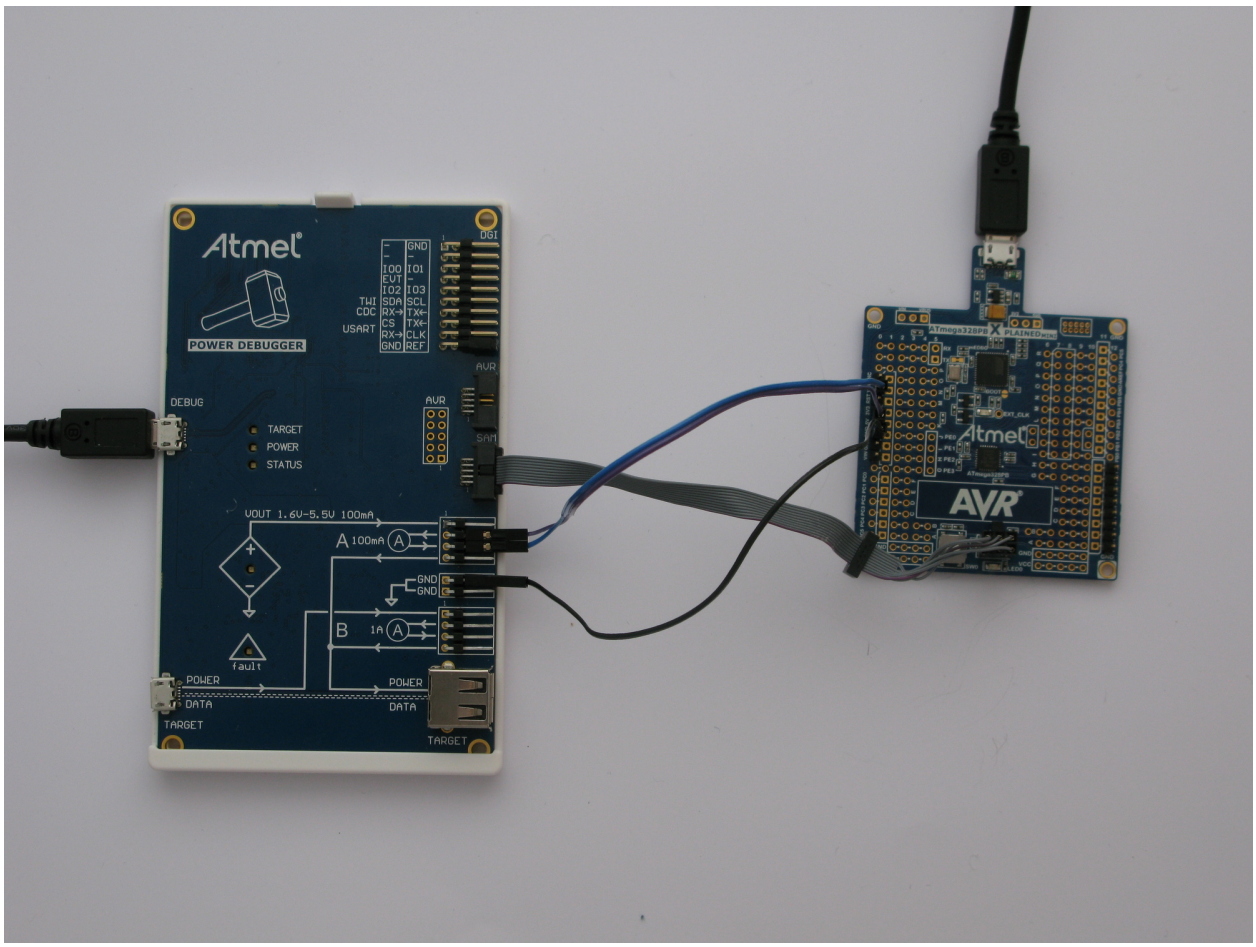
- Connect the 5V pin on the power-header of the Xplained Mini board to the input pin of the A channel on the Power Debugger
- Connect the output pin of the A channel to the VCC pin of the power header on the Xplained Mini

Connect the USB and programming cables:

To do:

- Connect the debug cable from the AVR output of the Power Debugger to the ISP header of the Xplained Mini board
- Plug both Power Debugger and Xplained Mini into USB ports on the host computer

Your setup should look something like this:



If the debugWIRE interface on the target device is enabled, it will need to be disabled.

Tip: If you are able to read the device signature using ISP after a power toggle, then debugWIRE is successfully disabled and DWEN is cleared.

4.1.4. Disabling the debugWIRE Interface

Disabling debugWIRE can be done either by:

- Selecting "Disable DebugWIRE and close" from the Debug menu during a debug session, or
- Using Atmel Studio command line programming utility: atprogram.exe

```
atprogram.exe -t powerdebugger -i debugwire -d atmega328pb dwdisable
```

and then using ISP to clear DWEN, either in the Atmel Studio programming dialog, or using atprogram.exe:

- Read the high fuse (offset 1)

```
atprogram.exe -t powerdebugger -i isp -d atmega328pb read -fs -s 1 -o 1 --format
hex
```

- OR the output value with 0x40 (DWEN is bit 6)
- Write the value back to the high fuse (offset 1)

```
atprogram.exe -t powerdebugger -i isp -d atmega328pb write -fs -o 1 --values
<new_fuse_value>
```

Caution: Take extreme care when writing fuses on the target device. Modifying the wrong fuse can result in the Xplained Mini kit being permanently unusable.

4.1.5. Disabling On-board Power Supply on the Xplained Mini

The mEDBG debugger on the Xplained Mini board has control of the target device's V_{CC} so that it can toggle its power. By opening the switch the user can reroute power to the target device through an external current measuring probe. By default on power-up the switch is closed (power is on).

To open the switch, use the Atmel Studio command line utility atprogram.exe.

```
atprogram.exe -t medbg parameters -psoff
```

Important: The clock source from the mEDBG to the target device will remain active, so the target device may be partially powered through I/O leakage.

To close the switch again, execute:

```
atprogram.exe -t medbg parameters -pson
```

Tip: Older mEDBG firmware does not support power-on requests. In that case, toggle the USB power to restore the target power.

4.1.6. Starting with Simple Current Measurements

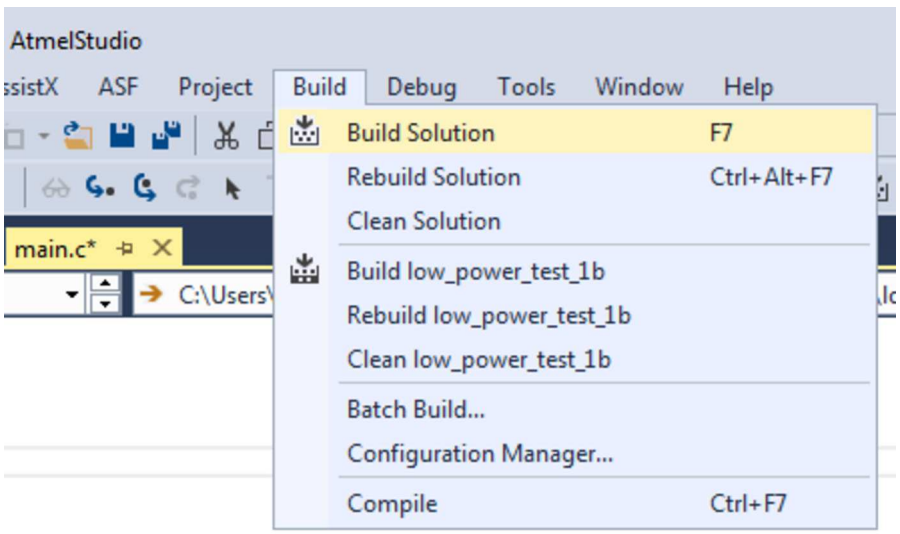
Lets start by writing an application to flash the LED on the Xplained Mini board. We will use a delay loop with a NOP instruction inside a large counter and pulse the LED with about a 1% duty cycle. The code for low_power_101 is shown here.

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

void delay (uint16_t length)
{
    // Simple delay loop
    for (uint16_t i=0; i<length; i++) {
        for (uint8_t j=0; j<255; j++) {
            asm volatile("nop");
        }
    }
```

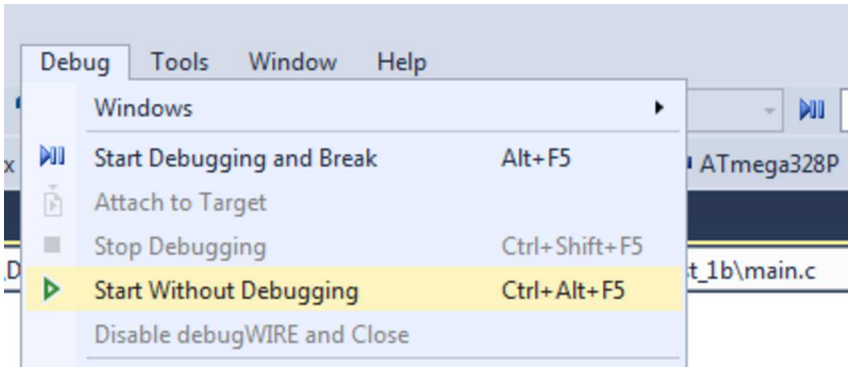
```
}

int main(void)
{
    // PORTB5 to output
    DDRB = (1 << 5);
    // Do forever:
    while (1) {
        // PORTB5 on
        PORTB = (1 << 5);
        // Short delay
        delay(50);
        // PORB5 off
        PORTB = 0x00;
        // Long delay
        delay(5000);
    }
}
```

Important: For this example set the optimisation level to None (-O0) in the project options under Toolchain → AVR/GNU C Compiler → Optimization.

To do: Build the project/solution (F7).



To do: Program the application into the target by selecting Start Without Debugging (Ctrl+Alt +F5).



LED0 on the mEDBG kit should now start to blink.

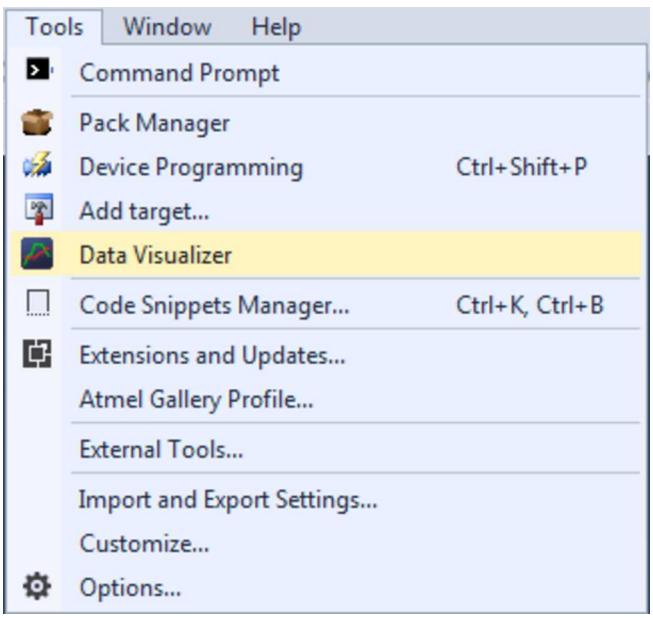
Info: In this example we are going to make use of programming mode only. In most systems running code through a debugger will not yield accurate current measurements. This is because the target device's debug module (OCD) requires a clock source which cannot be disabled while debugging.

Important: Remember to disable on-board power on the Xplained Mini.

4.1.7. Launching Atmel Data Visualizer

Atmel Data Visualizer is included as part of the Atmel Studio installer, and can be run either as a Studio extension or in standalone mode.

To run Atmel Data Visualizer as an extension inside Atmel Studio, select it in the **Tools** menu:

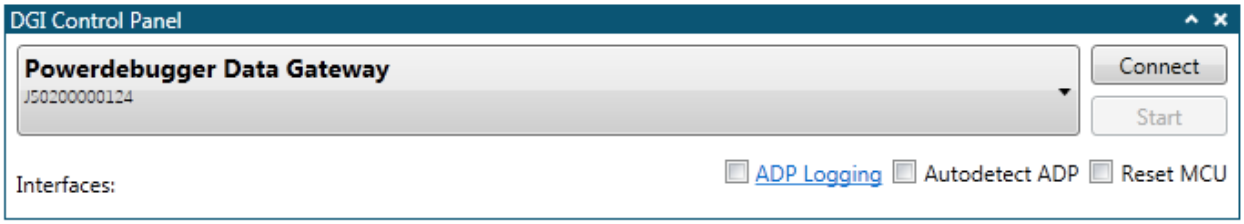


Kits supporting Data Visualizer functionality include a shortcut to the extension on their start page in Atmel Studio.

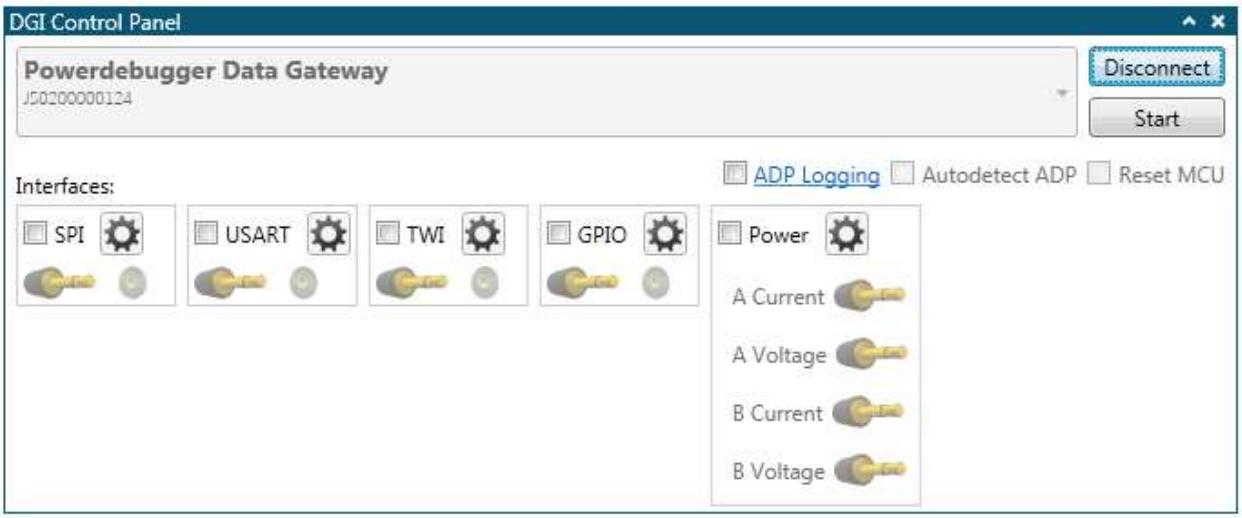
If the standalone version of Atmel Data Visualizer has been installed, look for the shortcut in Windows® start menu. The standalone version is available for download from gallery.atmel.com.

4.1.8. Basic Current Measurement

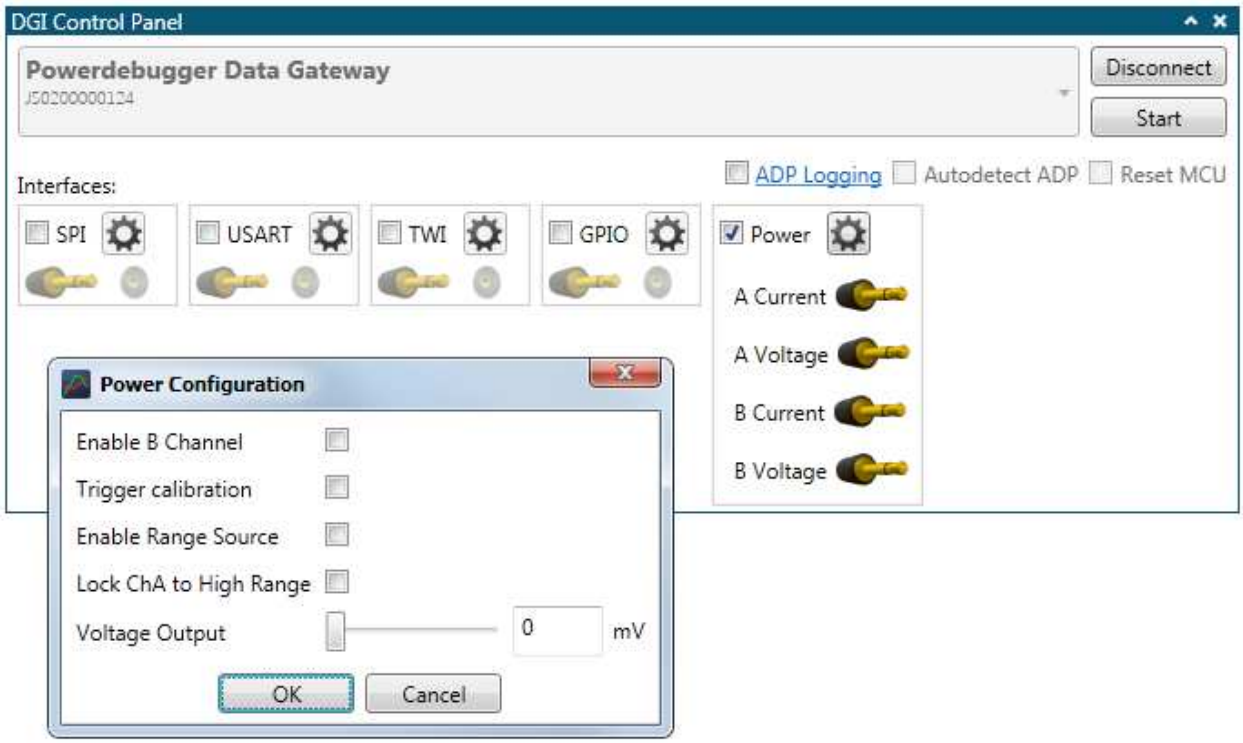
To do: Select correct tool in the **DGI Control Panel**.



To do: **Connect** to the DGI on the selected tool.



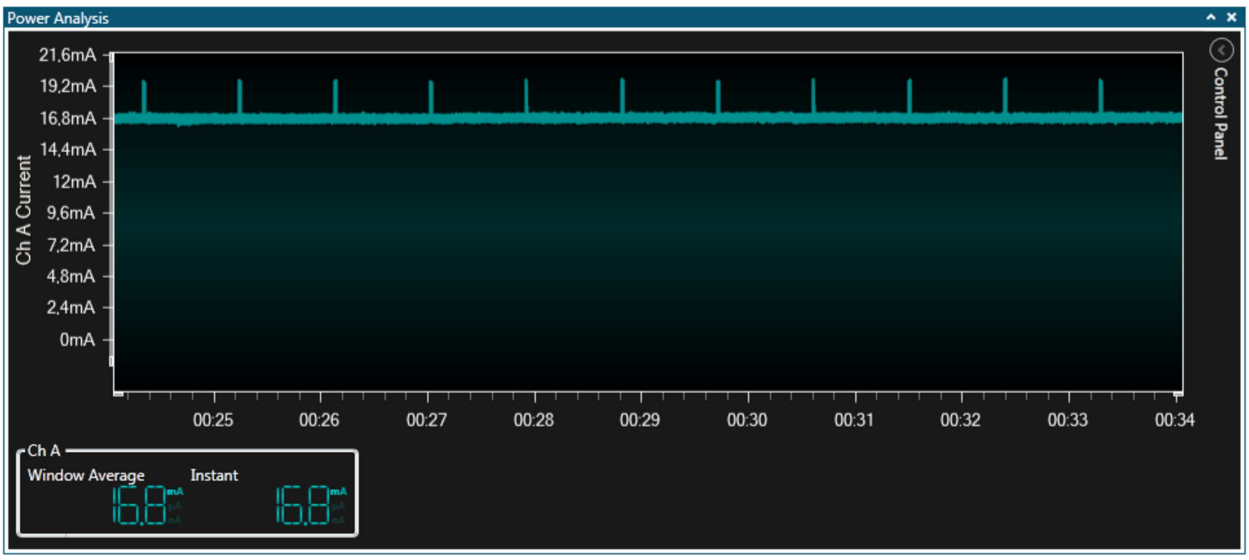
To do: Enable the **Power** interface and modify its settings to monitor the relevant channels.



To do: **Start** the Data Visualizer session.

4.1.9. LED Blinking

Running Data Visualizer for a few seconds should give you a plot similar to this one:



What can we see from this plot?

- The kit draws about 17mA with the LED OFF
- Current draw increases to about 20mA when the LED is pulsed ON

- The 1% duty cycle seems approximately correct

If your plot does not look like this, go back and check your setup for:

- Power supply to Xplained Mini (USB cable)
- Straps from VCC header to 'A' channel
- Common GND connection
- Is on-board power disabled on the Xplained Mini?
- Is the LED flashing? Has programming succeeded?
- Is debugWIRE (DWEN) disabled?
- If the 'fault' LED on the Power Debugger is ON, check your wiring and soldering one more time

4.1.10. Reducing the Clock Frequency

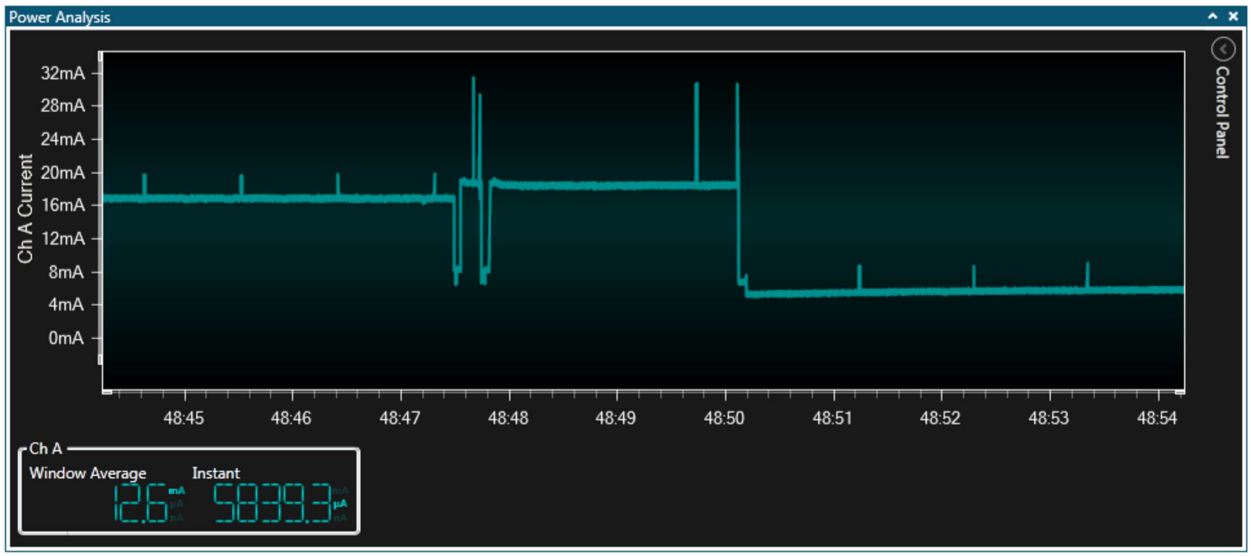
Now let's look at how we can reduce the power consumption of the application, and verify that it is improved.

A first thought might be to get rid of the delay loop and run the LED blinker of a timer interrupt. In addition, a simple example like this doesn't need to run at high-speed, so we can use the clock prescaler to run slower. The code below is included in project low_power_102.

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

// Timer 0 ISR
ISR (TIMER0_OVF_vect)
{
    if (PORTB == 0x00) {
        // LED is OFF, turn it on
        PORTB = (1 << 5);
        // Shortened timeout for on cycle
        TCNT0 = 0x102;
    }
    else {
        // LED is ON, turn it off
        PORTB = 0x00;
    }
}

int main(void)
{
    // Change the clock prescaler
    CLKPR = (1 << CLKPCE);
    // Scale by DIV64
    CLKPR = (1 << CLKPS2) | (1 << CLKPS1) | (0 << CLKPS0);
    // Port B5 to output
    DDRB = (1 << 5);
    // Timer0 DIV 1024
    TCCR0B = (1 << CS02) | (1 << CS00);
    // Overflow interrupt enable
    TIMSK0 = (1 << TOIE0);
    // Interrupts on
    sei();
    // Do nothing
    while (1)
        ;
}
```

To do:

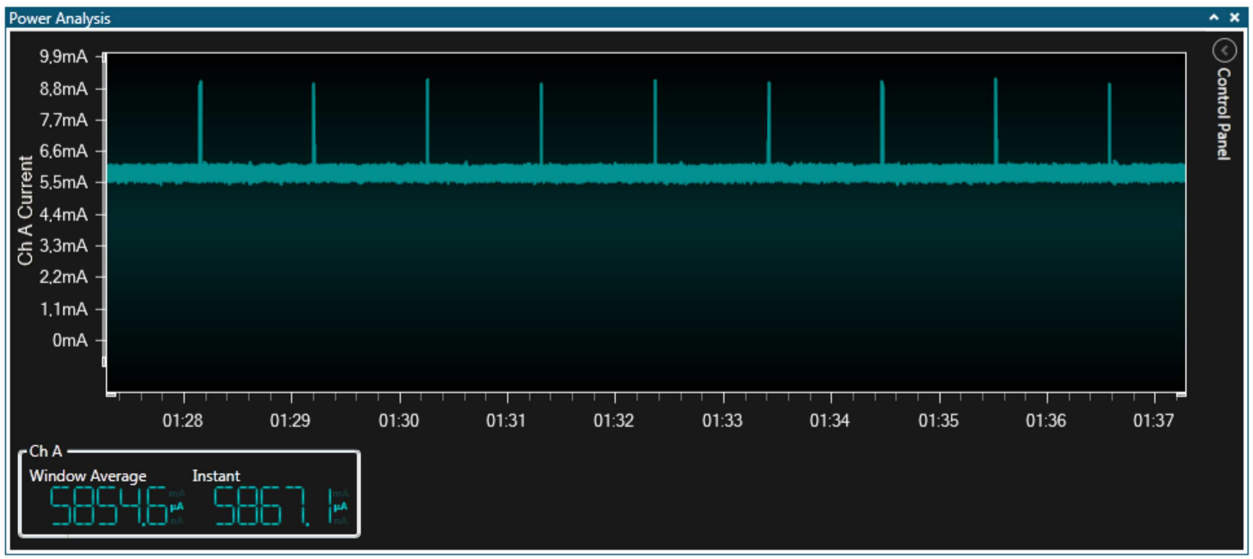
- Build the project/solution (F7)
- Program the application into the target device using Start Without Debugging (Ctrl+Alt +F5)
- Switch to Data Visualizer to see the results



The plot above was captured while reprogramming the target. What can we see from this plot?

- Two negative power pulses are seen (the device is pulled into reset)
- Four positive pulses are seen (read ID, erase, program flash, and verify flash)
- The new application starts to execute

Leaving the application to run for a few seconds should give you a plot similar to this one:



What can we see from this plot?

- The kit now draws about 6mA with the LED OFF

- Current draw increases to about 9mA when the LED is pulsed ON
- The 1% duty cycle still seems approximately correct

Result: Power consumption has been significantly improved by clocking the device slower.

Important: Because this example prescales the clock to 8MHz/64, the ISP programming clock must be set to less than 32kHz to be below 1/4 of the main clock when attempting to reprogram the device!

4.1.11. Using Sleep Mode

Power consumption has been reduced, but the CPU is still actively doing nothing. To benefit from an interrupt-driven model, the CPU can be put to sleep while waiting for interrupts. In addition we will power down all peripherals not used to further reduce power consumption. The code below is included in project low_power_103.

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

ISR (TIMER2_OVF_vect)
{
    if (PORTB == 0x00) {
        // LED is OFF, turn it on
        PORTB = (1 << 5);
        // Shortened timeout for on cycle
        TCNT2 = 0x102;
    }
    else {
        // LED is ON, turn it off
        PORTB = 0x00;
    }
}

int main(void)
{
    // Disable digital input buffer on ADC pins
    DIDR0 = (1 << ADC5D) | (1 << ADC4D) | (1 << ADC3D)
    | (1 << ADC2D) | (1 << ADC1D) | (1 << ADC0D);
    // Disable digital input buffer on Analog comparator pins
    DIDR1 |= (1 << AIN1D) | (1 << AIN0D);
    // Disable Analog Comparator interrupt
    ACSR &= ~(1 << ACIE);
    // Disable Analog Comparator
    ACSR |= (1 << ACD);
    // Disable unused peripherals to save power
    // Disable ADC (ADC must be disabled before shutdown)
    ADCSRA &= ~(1 << ADEN);
    // Shut down the ADC
    power_adc_disable();
    // Disable SPI
    power_spi_disable();
    // Disable TWI
    power_twi_disable();
    // Disable the USART 0 module
    power_usart0_disable();
    // Disable the Timer 1 module
    power_timer1_disable();
    // Disable the Timer 0 module
    power_timer0_disable();
```

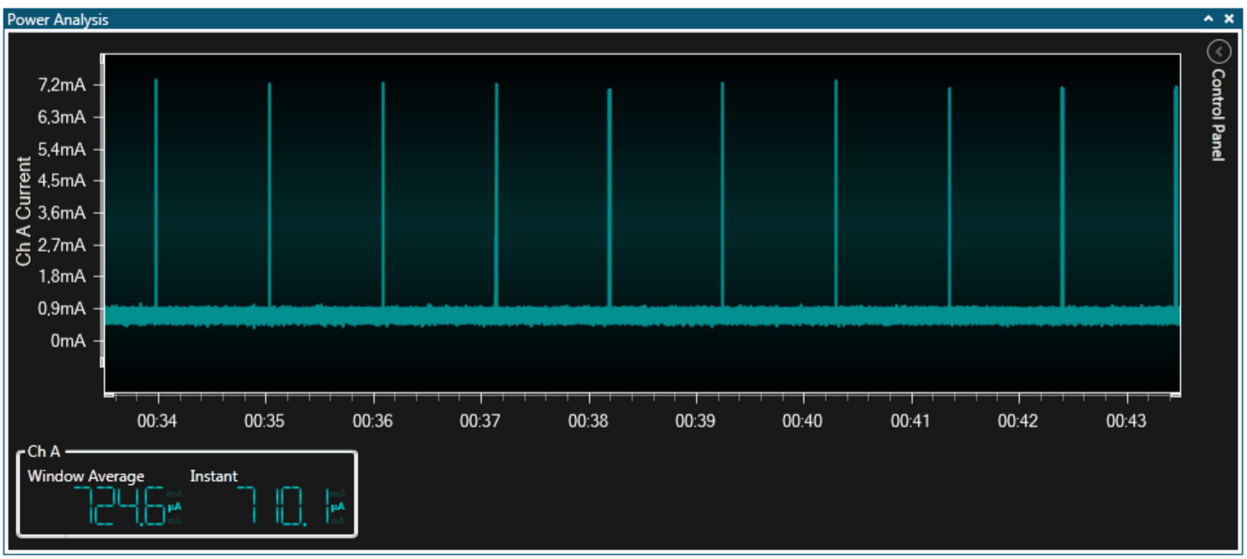
```
    // Change the clock prescaler
    CLKPR = (1 << CLKPCE);
    // Scale by DIV64
    CLKPR = (1 << CLKPS2) | (1 << CLKPS1) | (0 << CLKPS0);
    // Port B5 to output
    DDRB = (1 << 5);
    // Timer2 DIV 1024
    TCCR2B = (1 << CS22) | (1 << CS21) | (1 << CS20);
    // Overflow interrupt enable
    TIMSK2 = (1 << TOIE2);
    // Interrupts on
    sei();
    while (1) {
        set_sleep_mode(SLEEP_MODE_PWR_SAVE);
        sleep_mode();
    }
}
```

To do:

- Build the project/solution (F7)
- Program the application into the target device using Start Without Debugging (Ctrl+Alt +F5)
- Switch to Data Visualizer to see the results

Important: Because the previous example prescaled the clock to 8MHz/64, the ISP programming clock must be set to less than 32kHz to be below 1/4 of the main clock.

Important: Remember to disable on-board power on the Xplained Mini.



What can we see from this plot?

- The kit now draws less than 1mA with the LED OFF
- Current draw increases to about 7mA when the LED is pulsed ON
- The 1% duty cycle still seems approximately correct

Result: Power consumption has again been improved by using sleep mode and disabling unused peripherals.

4.1.12. Using Power Down Mode

Again our power consumption has been reduced, but the CPU is still in a "lighter" sleep mode than it needs to be. To go into Power Down mode we have to switch to the watchdog timer as a wake-up source. When the watchdog wakes triggers an interrupt, we light the LED and go into IDLE mode until we switch it off again. The code below is included in project low_power_104.

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

volatile uint8_t deep_sleep = 0;

ISR (WDT_vect)
{
    // LED is OFF, turn it on
    PORTB = (1 << 5);
    // Flag a lighter sleep state
    deep_sleep = 0;
}

ISR (TIMER2_OVF_vect)
{
    // LED is ON, turn it off
    PORTB = 0x00;
    // Flag a deep-sleep state
    deep_sleep = 1;
}

int main(void)
{
    // Disable digital input buffer on ADC pins
    DIDR0 = (1 << ADC5D) | (1 << ADC4D) | (1 << ADC3D)
    | (1 << ADC2D) | (1 << ADC1D) | (1 << ADC0D);
    // Disable digital input buffer on Analog comparator pins
    DIDR1 |= (1 << AIN1D) | (1 << AIN0D);
    // Disable Analog Comparator interrupt
    ACSR &= ~(1 << ACIE);
    // Disable Analog Comparator
    ACSR |= (1 << ACD);
    // Disable unused peripherals to save power
    // Disable ADC (ADC must be disabled before shutdown)
    ADCSRA &= ~(1 << ADEN);
    // Shut down the ADC
    power_adc_disable();
    // Disable SPI
    power_spi_disable();
    // Disable TWI
    power_twi_disable();
    // Disable the USART 0 module
    power_usart0_disable();
    // Disable the Timer 1 module
    power_timer1_disable();
    // Disable the Timer 0 and 2 modules
    power_timer0_disable();

    // Timer 2 needs to stay on
```

```
    //power_timer2_disable();

    // Change the clock prescaler
    CLKPR = (1 << CLKPCE);
    // Scale by DIV64
    CLKPR = (1 << CLKPS2) | (1 << CLKPS1) | (0 << CLKPS0);
    // Port B5 to output
    DDRB = (1 << 5);
    // Watchdog reset
    wdt_reset();
    // Start timed sequence
    WDTCSR |= (1<<WDCE) | (1<<WDE);
    // Set new prescaler(time-out) value = 64K cycles (~0.5s)
    WDTCSR = (1<<WDIE) | (1<<WDP2) | (1<<WDP1);

    // Timer2 DIV 32
    TCCR2B = (0 << CS22) | (1 << CS21) | (1 << CS20);
    // Overflow interrupt enable
    TIMSK2 = (1 << TOIE2);

    // Interrupts on
    sei();

    while (1)
    {
        // If deep sleep, then power down
        if (deep_sleep)
            set_sleep_mode(SLEEP_MODE_PWR_DOWN);
        else {
            // Shorter sleep in idle with LED on
            TCNT2 = 0;
            set_sleep_mode(SLEEP_MODE_IDLE);
        }
        sleep_mode();
    }
}
```

To do:

- Build the project/solution (F7)
- Program the application into the target device using Start Without Debugging (Ctrl+Alt +F5)
- Switch to Data Visualizer to see the results

Important: Because the previous example prescaled the clock to 8MHz/64, the ISP programming clock must be set to less than 32kHz to be below 1/4 of the main clock.

Important: Remember to disable on-board power on the Xplained Mini.

What can we see from this plot?

- Even lower power consumption with the LED OFF

Result: Using the watchdog timer as wake-up source allows us to use the lowest power sleep mode in this example.

Now let's take a brief and closer look at the details of our plots. Enabling cursors allows us to take more accurate measurements from the plot.

To do:

- Open the **Control Panel** in the upper right corner of the **Power Analysis** module
- Expand the **Cursors** section
- Check the **Enabled** box to turn the cursors on

Remember: If the current measurements are still running, make sure to disable **Auto-scroll** before enabling the cursors. Else the graph view will rapidly scroll away from the cursors.