Programmers and Debuggers

Power Debugger

USER GUIDE

Table of Contents

1. The Atmel Power Debugger

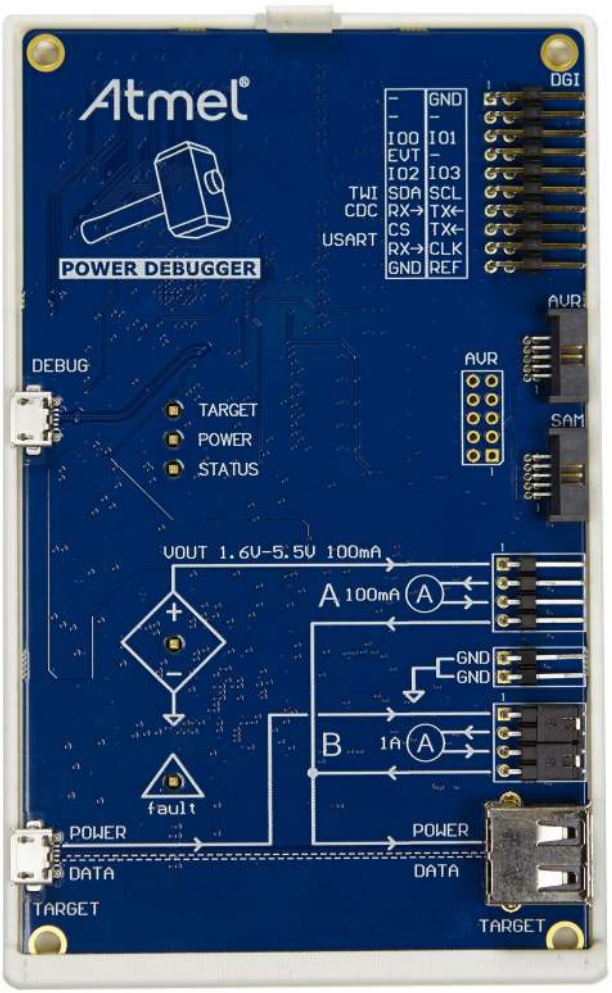
Atmel® Power Debugger is a powerful development tool for debugging and programming ARM® Cortex®-M based Atmel SAM and Atmel AVR® microcontrollers using JTAG, SWD, PDI, UPDI, debugWIRE, aWire, TPI, or SPI target interfaces.

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

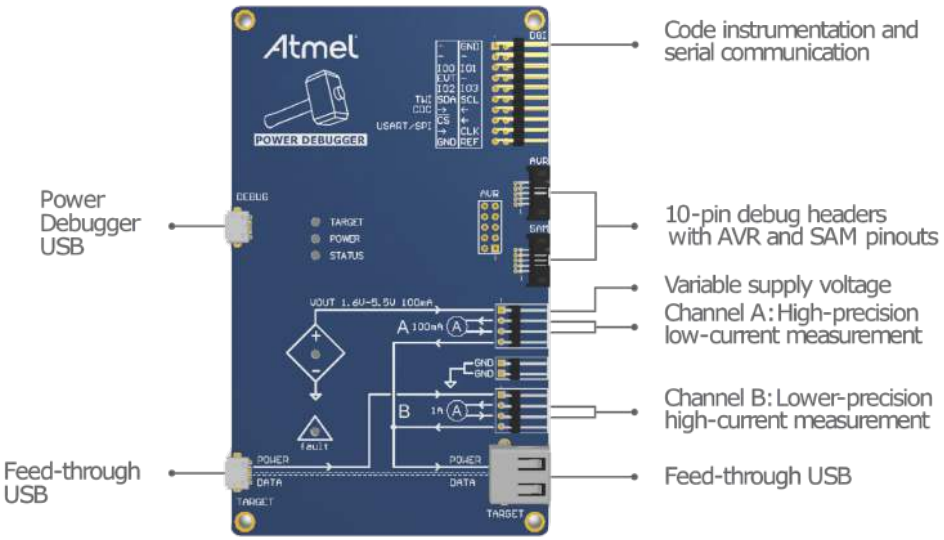
The Power Debugger kit contains:

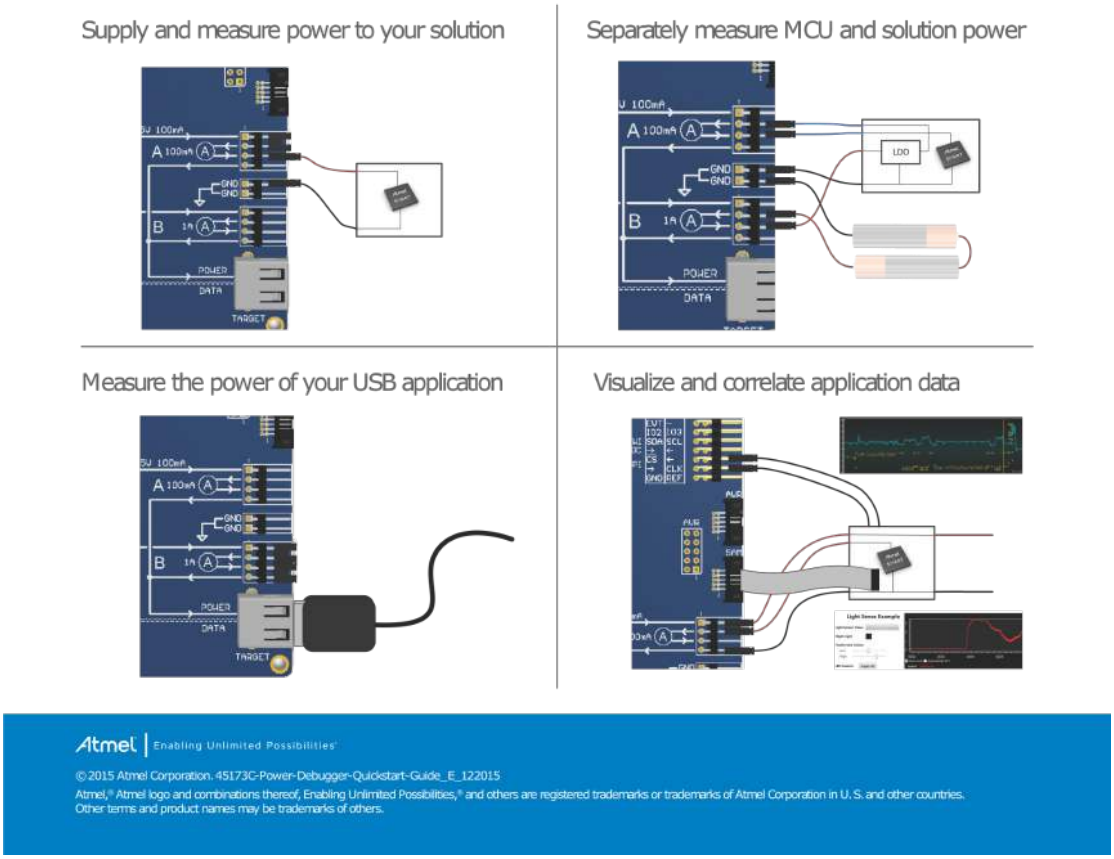
- One Power Debugger unit with plastic back-plate
- Two USB cables (1.5m, high-speed, Micro-B)
- One adapter board containing 50-mil AVR, 100-mil AVR/SAM, and 100-mil 20-pin SAM adapters
- One IDC flat cable with 10-pin 50-mil connector and 6-pin 100-mil connector
- One 50-mil 10-pin mini squid cable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

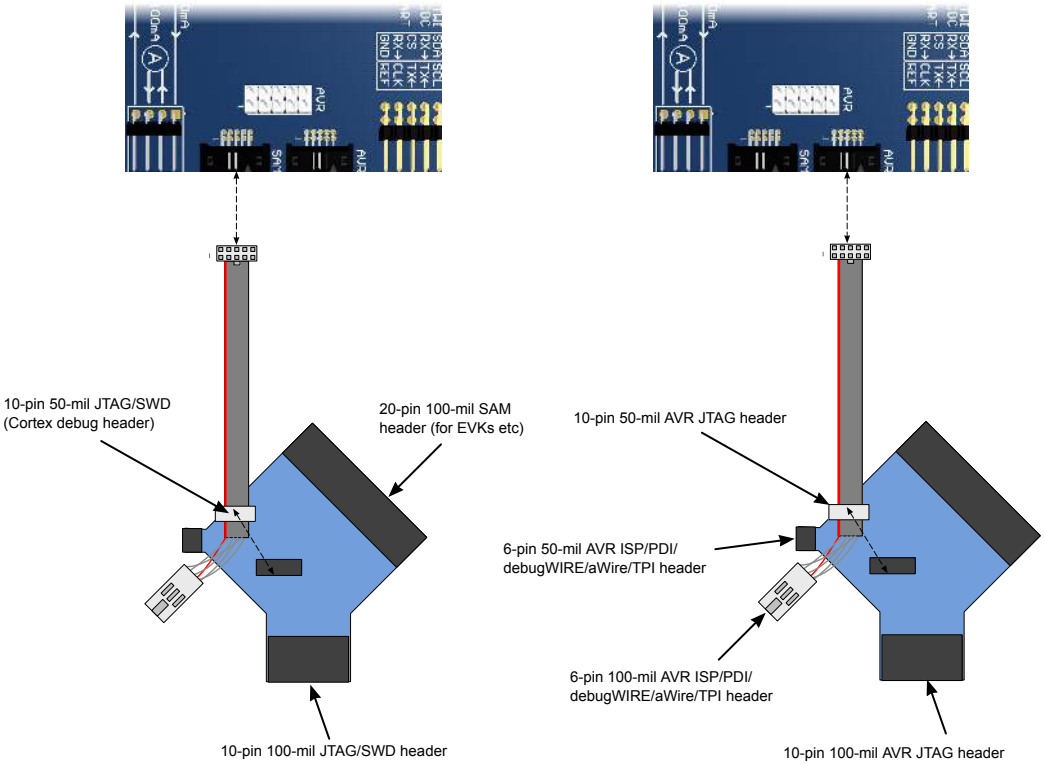
Tip: For common power routing configurations, refer to the printed quick-start guide included in the kit and in the Getting Started section.

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.

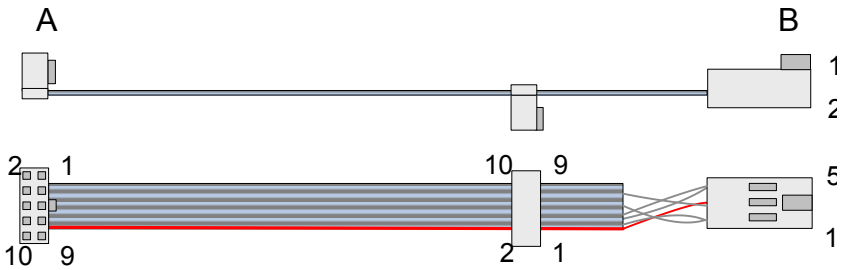
The Power Debugger kit contains all necessary cabling and adapters. An overview of connection options is shown.

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.

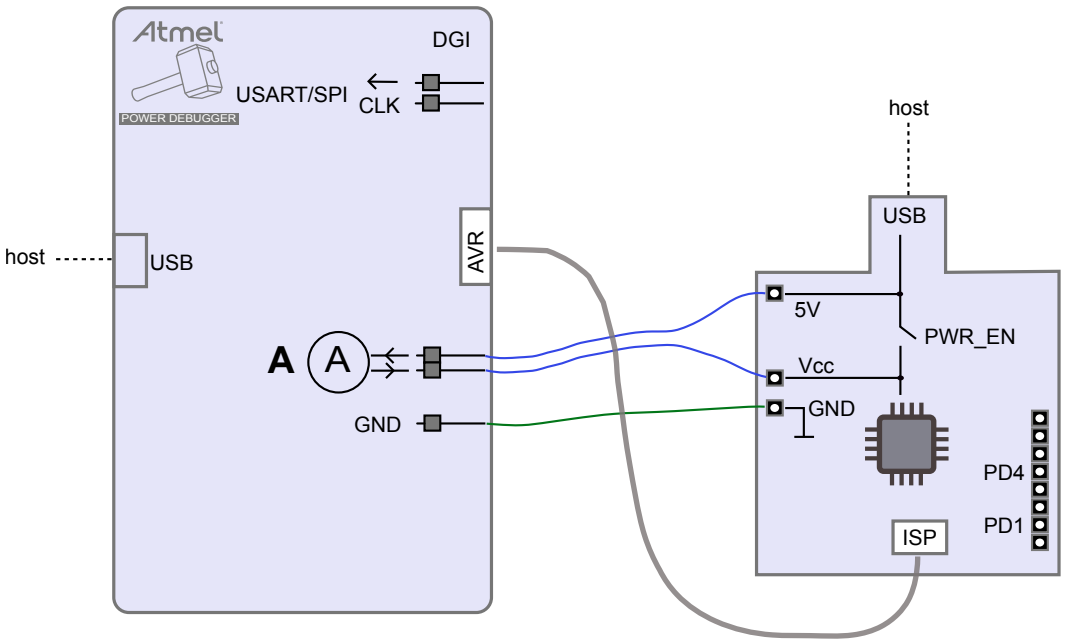
4.1.1. Requirements

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.

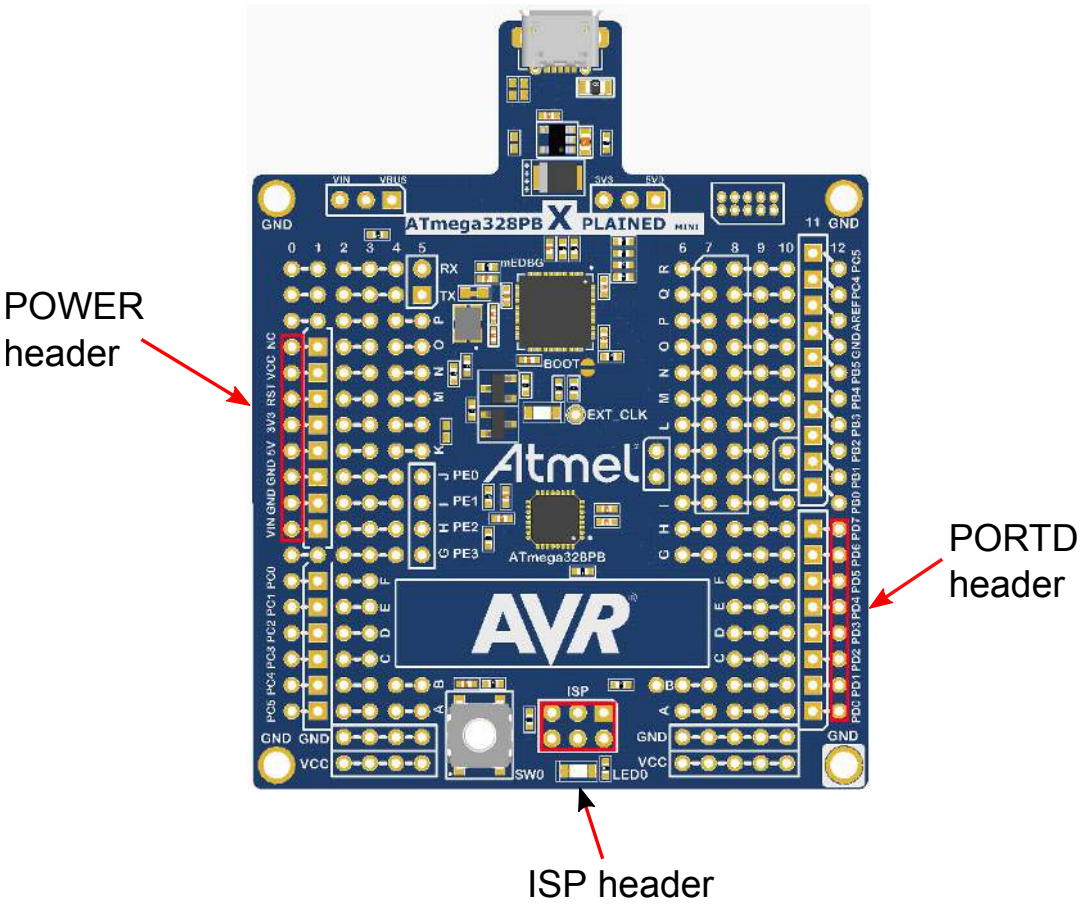
We will make use of code instrumentation to send data from the Xplained Mini board to the Power Debugger DGI. To do this we need access to PORTD features on the target MCU.

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:

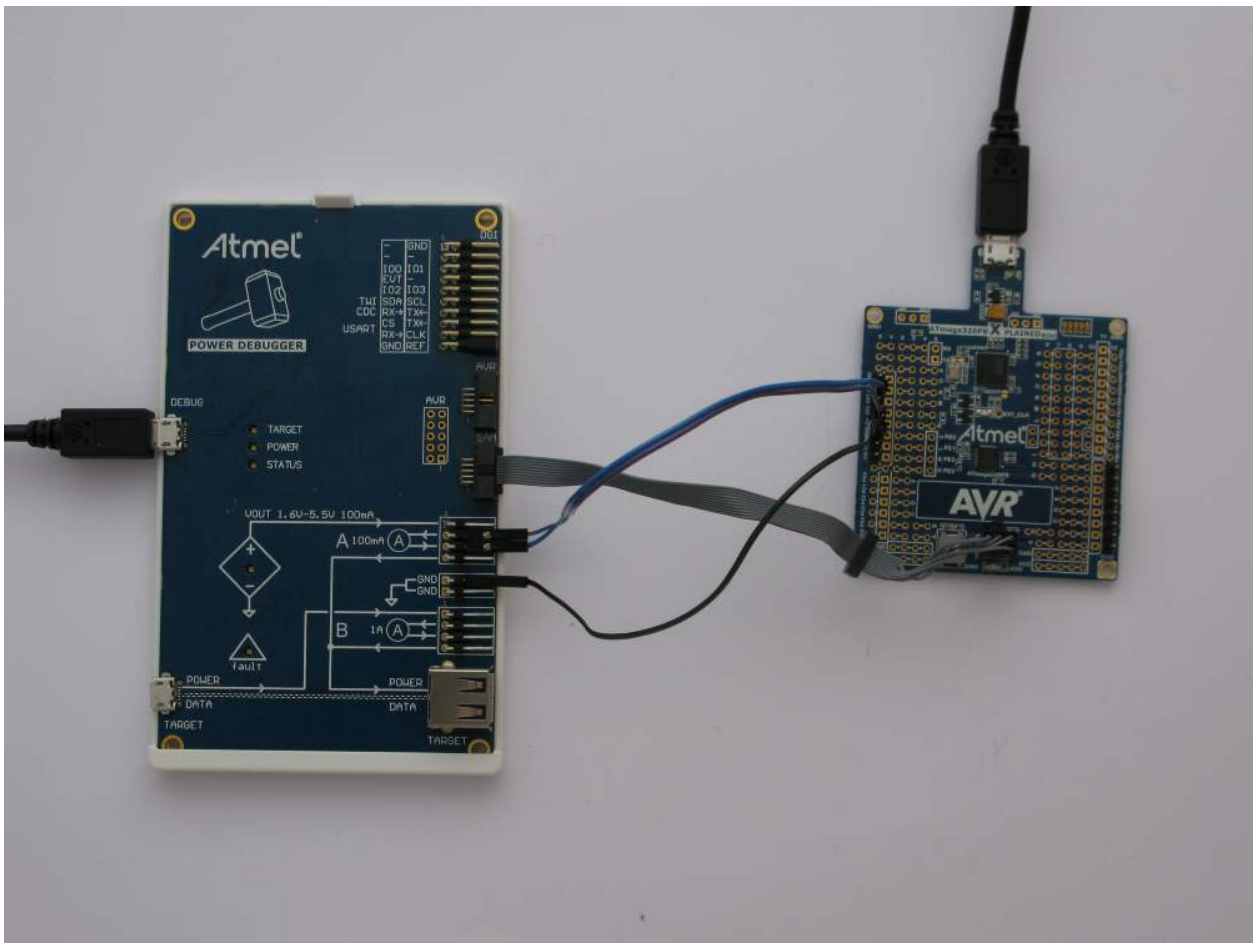
- Connect the 5V pin on the power-header of the Xplained Mini board to the input pin of the A channel on the Power Debugger
- Connect the output pin of the A channel to the VCC pin of the power header on the Xplained Mini

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

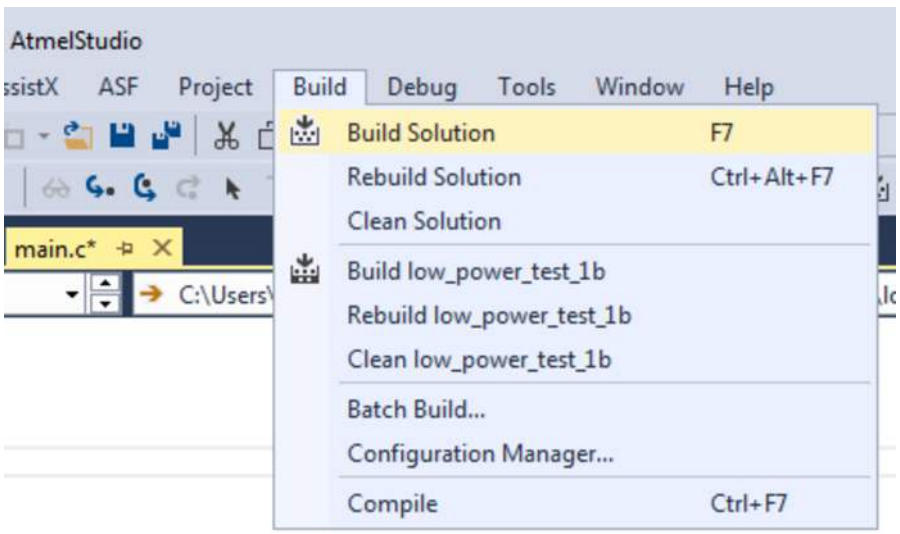
Lets start by writing an application to flash the LED on the Xplained Mini board. We will use a delay loop with a NOP instruction inside a large counter and pulse the LED with about a 1% duty cycle. The code for low_power_101 is shown here.

```
#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");
        }
    }
```

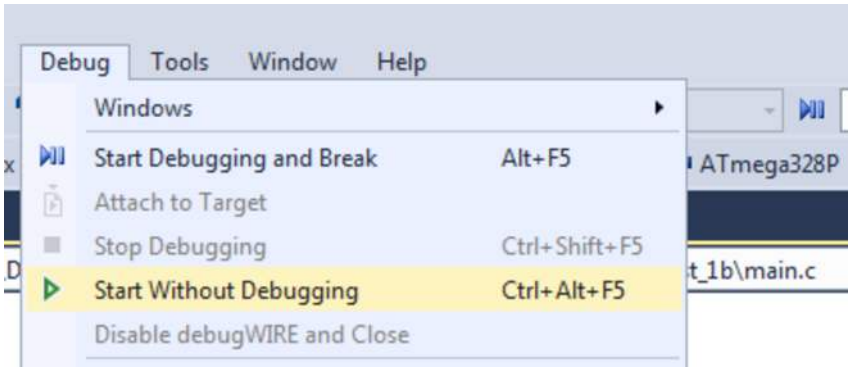
```
}

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.

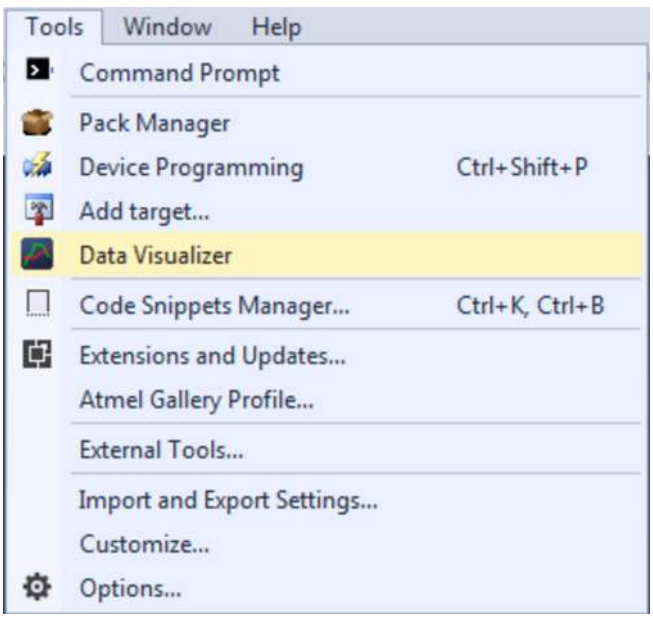
Info: In this example we are going to make use of programming mode only. In most systems running code through a debugger will not yield accurate current measurements. This is because the target device's debug module (OCD) requires a clock source which cannot be disabled while debugging.

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:

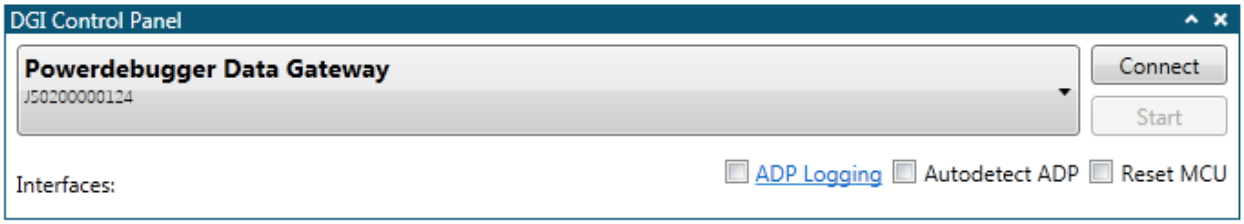


Kits supporting Data Visualizer functionality include a shortcut to the extension on their start page in Atmel Studio.

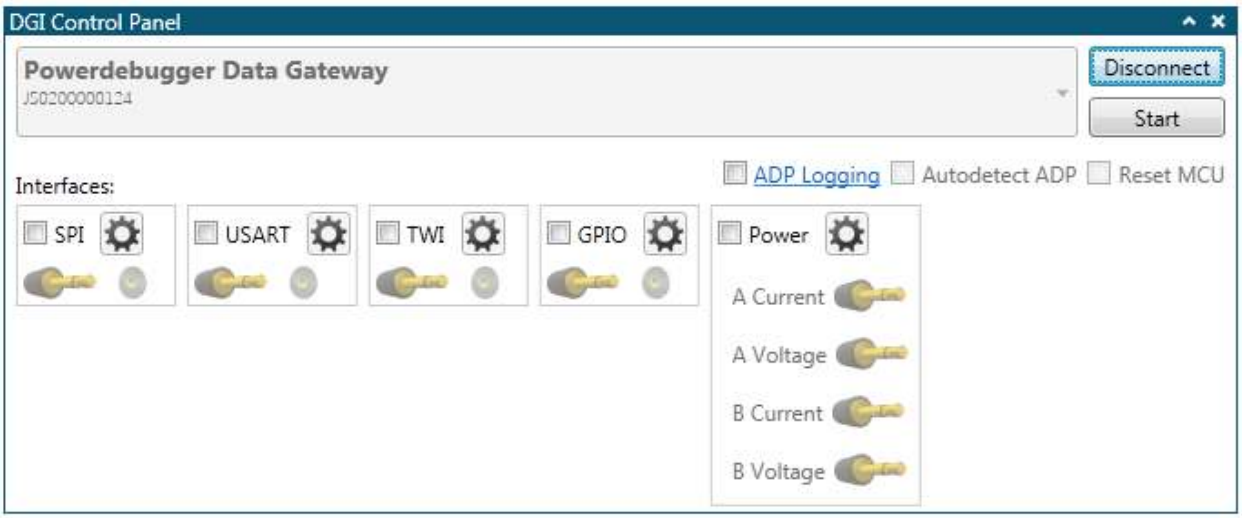
If the standalone version of Atmel Data Visualizer has been installed, look for the shortcut in Windows® start menu. The standalone version is available for download from gallery.atmel.com.

4.1.8. Basic Current Measurement

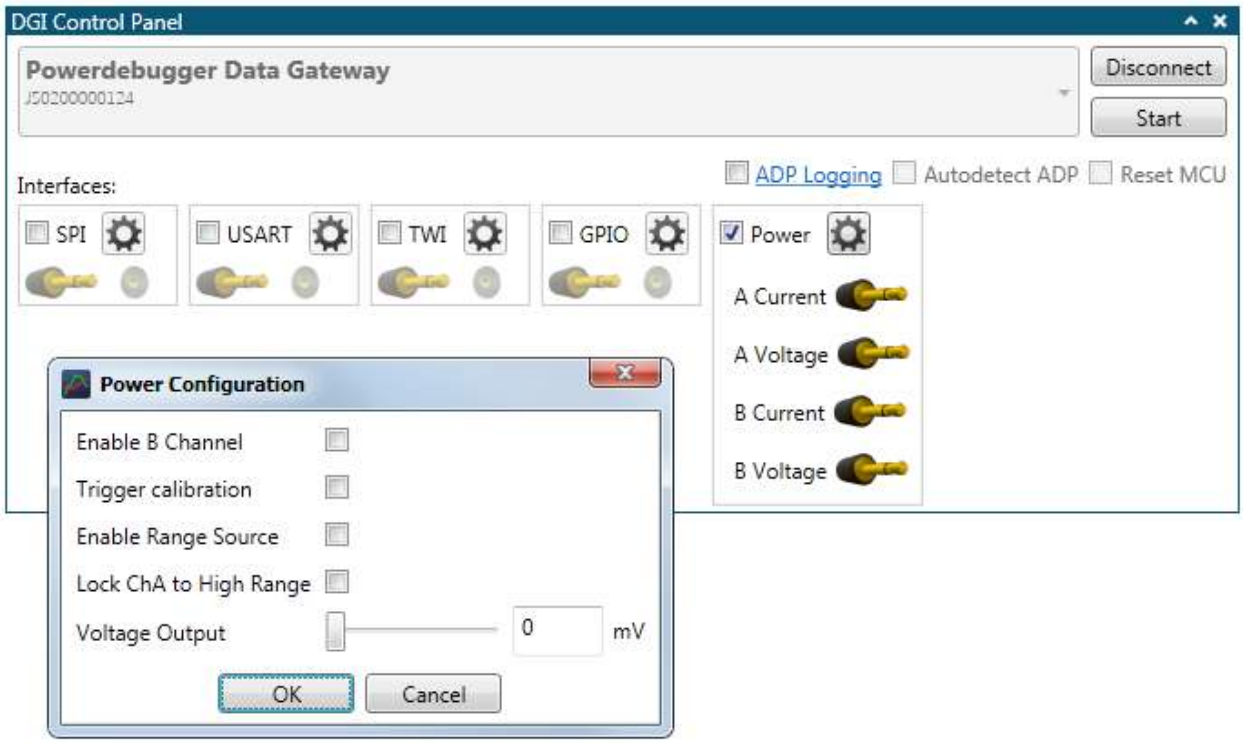
To do: Select correct tool in the **DGI Control Panel**.



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



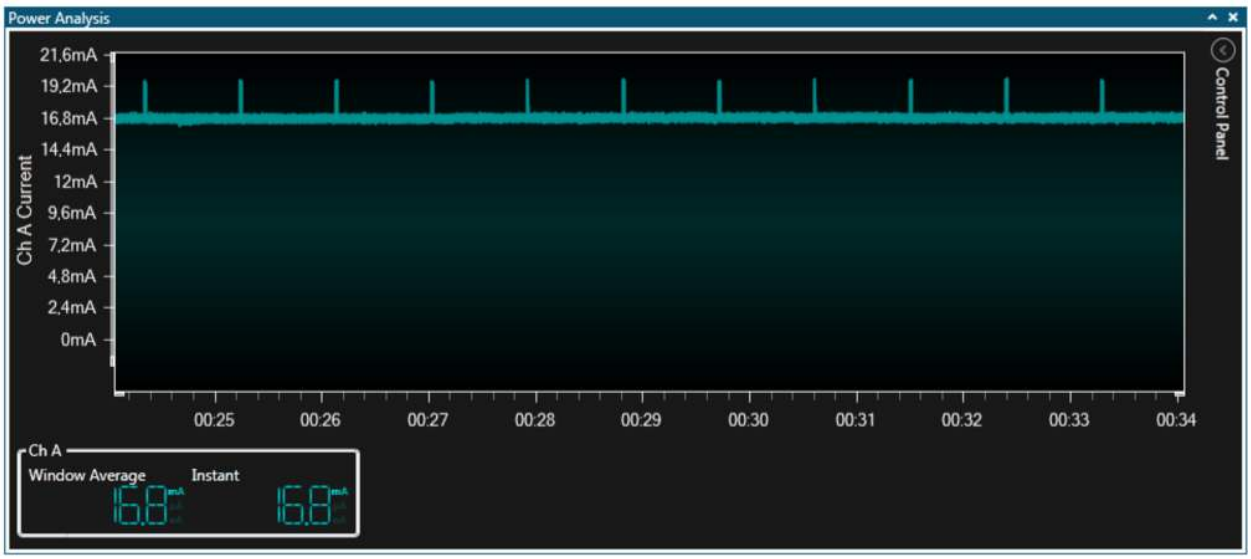
To do: Enable the **Power** interface and modify its settings to monitor the relevant channels.



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

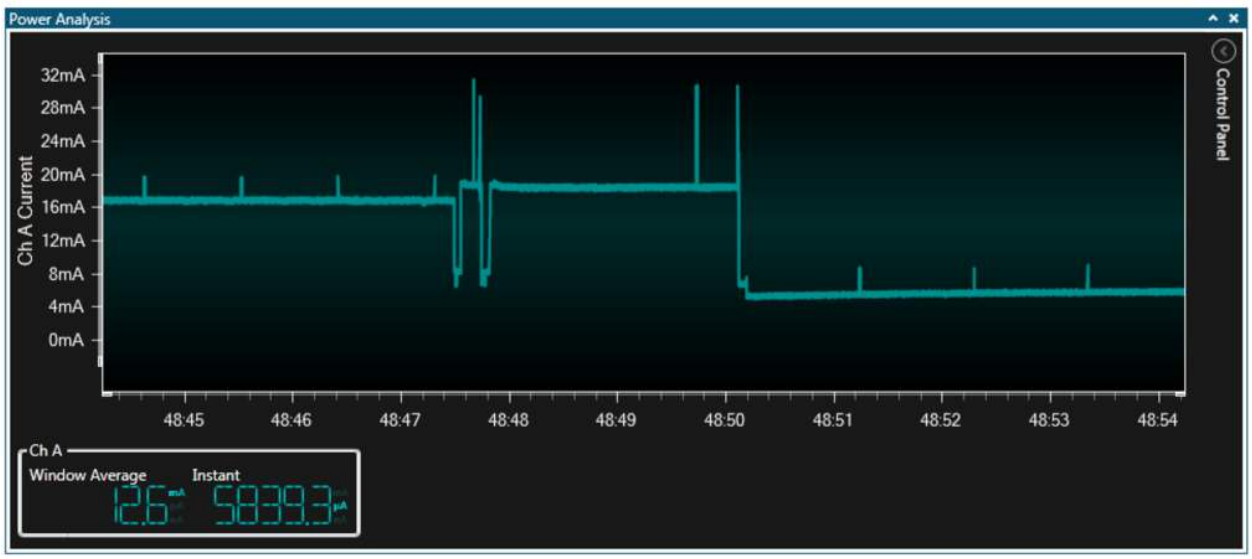
Now let's look at how we can reduce the power consumption of the application, and verify that it is improved.

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:

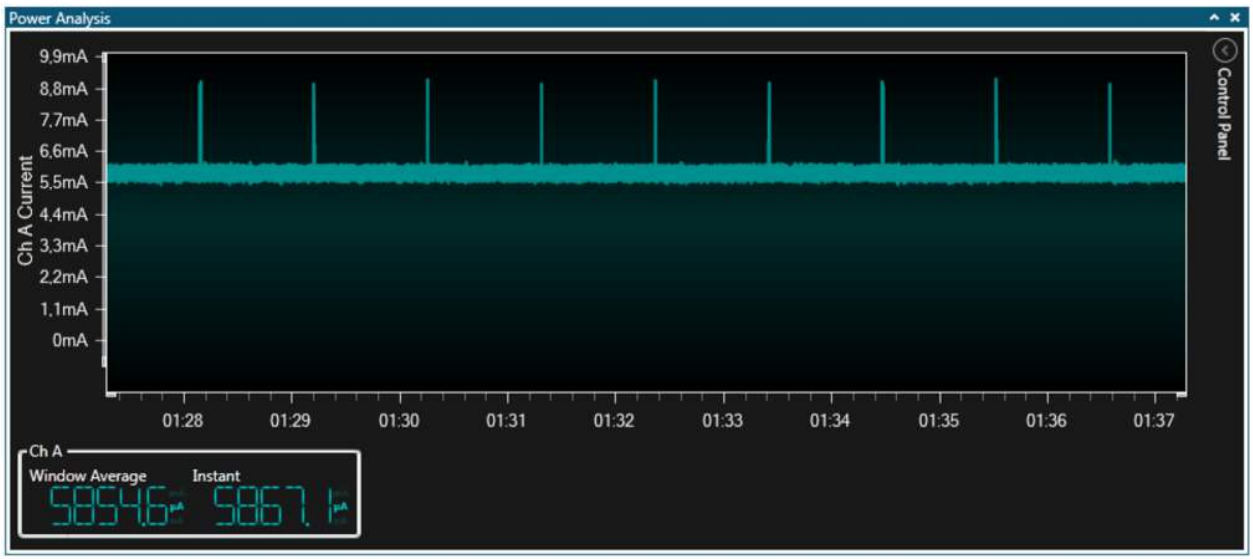
- Build the project/solution (F7)
- Program the application into the target device using Start Without Debugging (Ctrl+Alt +F5)
- Switch to Data Visualizer to see the results



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();
```

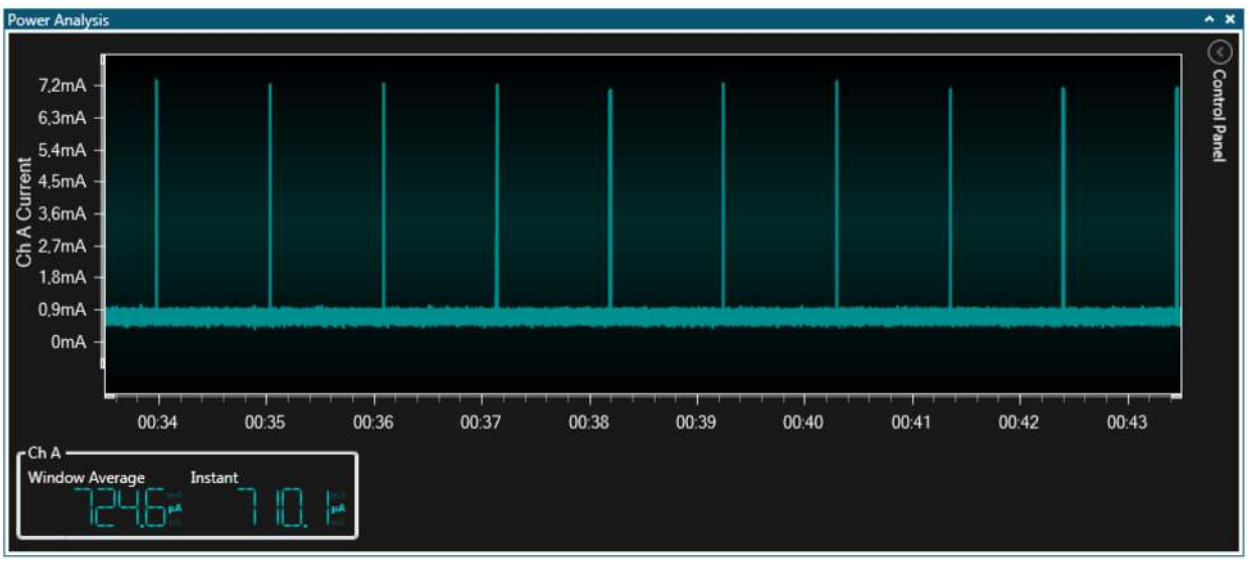
```
    // Change the clock prescaler
    CLKPR = (1 << CLKPCE);
    // Scale by DIV64
    CLKPR = (1 << CLKPS2) | (1 << CLKPS1) | (0 << CLKPS0);
    // Port B5 to output
    DDRB = (1 << 5);
    // Timer2 DIV 1024
    TCCR2B = (1 << CS22) | (1 << CS21) | (1 << CS20);
    // Overflow interrupt enable
    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.

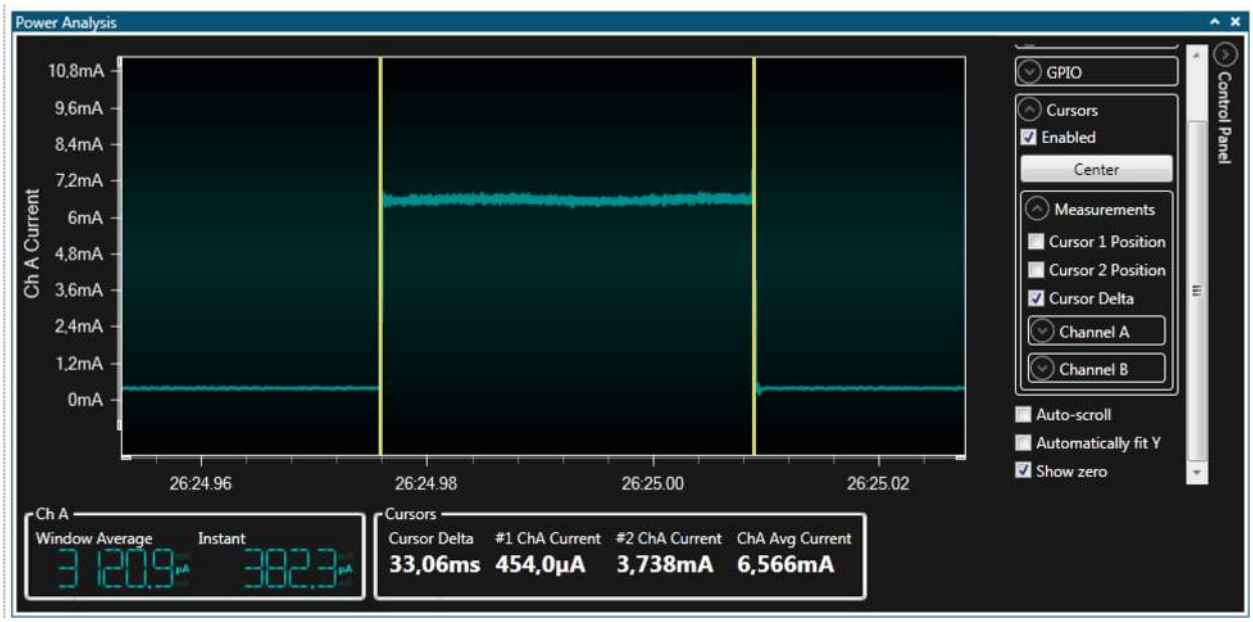
To do:

- 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

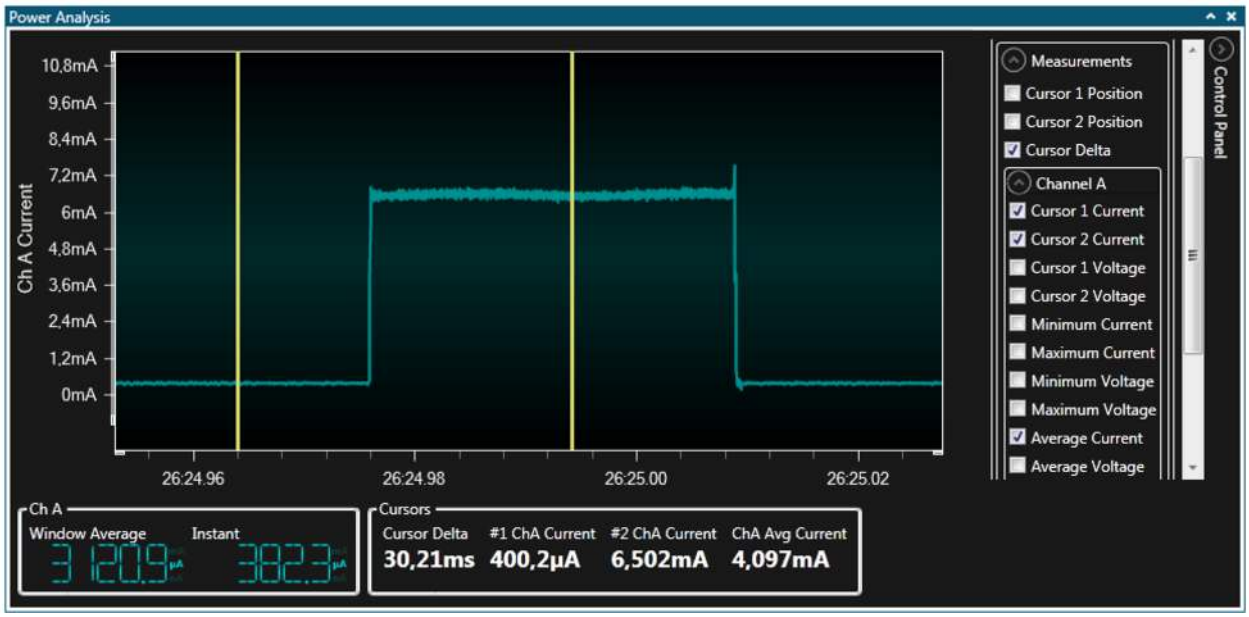
Remember: If the current measurements are still running, make sure to disable **Auto-scroll** before enabling the cursors. Else the graph view will rapidly scroll away from the cursors.

In the plot above we use two cursors to firstly check the delta - confirming that the watchdog timer interrupts approximately every second, as expected.

Zooming further in we can see that the width of the 'on' pulse (shown as Cursor Delta) is a little over 33ms. This equates to a duty-cycle of 3.3% - a touch higher than intended (1%). The main clock is 16MHz at 5V, and we are using DIV64 CLKPR, and DIV32 on TIMER2, giving it a 128µs tick. With overflow every 256 cycles the LED on period is thus 32.7ms, close to our measurement. So to achieve a 1% duty-cycle TIMER2 preload value can be increased from 0x00 to 0xB2.



To get more accurate current measurements from the plots, enable the cursor current measurements as shown here. The cursor values are shown as about 400µA with the LED OFF and 6.5mA with the LED ON.



4.1.13. Using Code Instrumentation

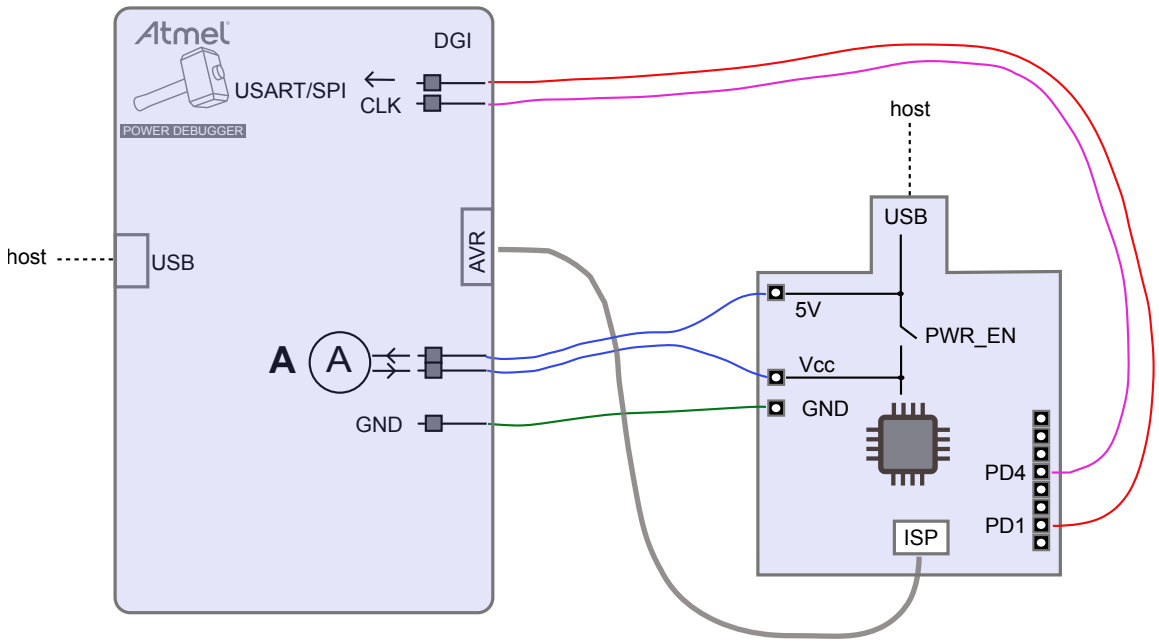
Finally, in this section we will look at some other tricks which the Atmel Data Visualizer is capable of.

Aside from taking current measurements, the Power Debugger has a full Data Gateway Interface (DGI) port for streaming data from the target to the host computer. We will hook this up to the USART for simple console output.

Some hardware wiring needs to be done to link the USART. We use synchronous mode for convenience in baud rate setting.

- PD1 (TXD) of the Xplained Mini is connected to USART ← of the DGI header
- PD4 (XCK) of the Xplained Mini is connected to USART CLK of the DGI header

The updated wiring diagram is shown here:



The example code makes use of the pin-change interrupt connected to the button on the Xplained Mini to toggle a LED and update a ticker. On each button-press the LED toggles and the ticker count is sent using the DGI USART along with a LED status message.

The code below is included in project low_power_105.

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

volatile uint8_t led_on;
volatile uint8_t send_message;
volatile uint8_t ticker = 0;

const char* message_on = "LED ON ";
const char* message_off = "LED OFF ";

ISR (PCINT0_vect)
{
    // Each botton press generates two pin-change interrupt (press, release)
    // Ignore half of these
    if (PINB & (1 << 7))
        return;
    // Update LED
    if (led_on)
        PORTB &= ~(1 << 5);
    else
        PORTB |= (1 << 5);
    // Invert led_on
    led_on = ~led_on;
    // Flag a message send
    send_message = 1;
    // Increment ticker
    ticker++;
    // Reset ticker
    if (ticker >= 10)
        ticker = 0;
}

void usart_send (const char data)
{
    // Send a character to the USART
    UDR0 = data;
    // Wait for the character to be sent
    while (!(UCSR0A & (1 << TXC0)))
        ;
    // Clear the flag
    UCSR0A = (1 << TXC0);
}

int main(void)
{
    // PORTB5 to output
    DDRB = (1 << 5);

    // LED OFF
    PORTB = 0;
    led_on = 0;

    // Enable Pin-change interrupt
    PCICR = (1 << PCIE0);
    PCMSK0 = (1 << PCINT7);

    // USART0
    // Baud trivial in synchronous mode
    UBRR0 = 0x0FFF;
    // Enable XCK for master clock outout
    DDRD |= (1 << 4);
    // Enable USART transmitter
    UCSR0B |= (1 << TXEN0);
    // Synchronous mode
    UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00) | (1 << UMSEL00);

    // Interrupts on
```

```
    sei();

    while(1)  {
        // Sleep
        set_sleep_mode(SLEEP_MODE_IDLE);
        sleep_mode();
        // Woken up
        if(send_message) {
            // Send a message
            const char* pmessage;
            if (led_on)
                pmessage = message_on;
            else
                pmessage = message_off;
            while (*pmessage)
                usart_send(*pmessage++);
            // Send the ticker value
            usart_send(ticker + '0');
            usart_send('\n');
            // Sent
            send_message = 0;
        }
    }
}
```

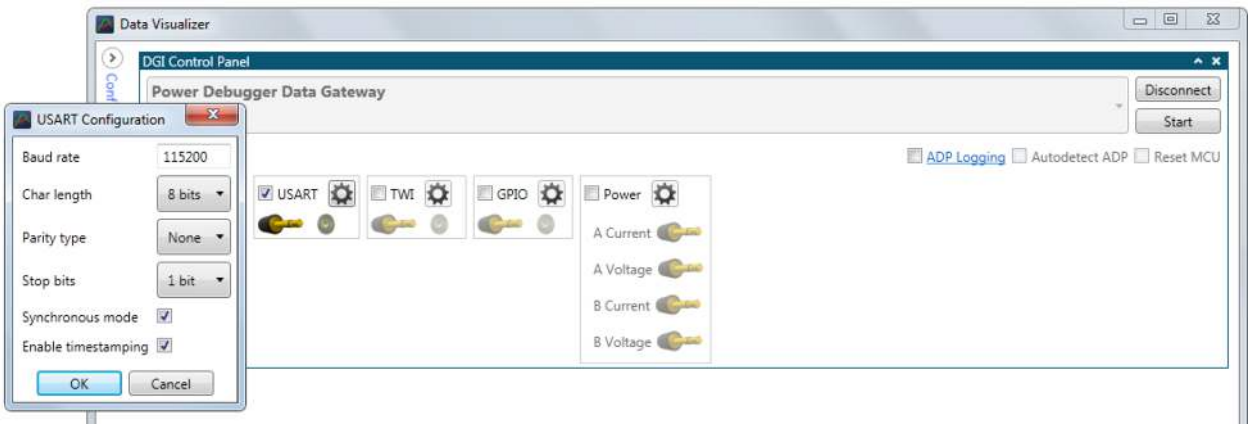
To do:

- 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

4.1.13.1. Enable USART Interface

To do:

- Check the **USART** checkbox
- Open the **USART Configuration** dialog by pushing the gear button next to the **USART** checkbox



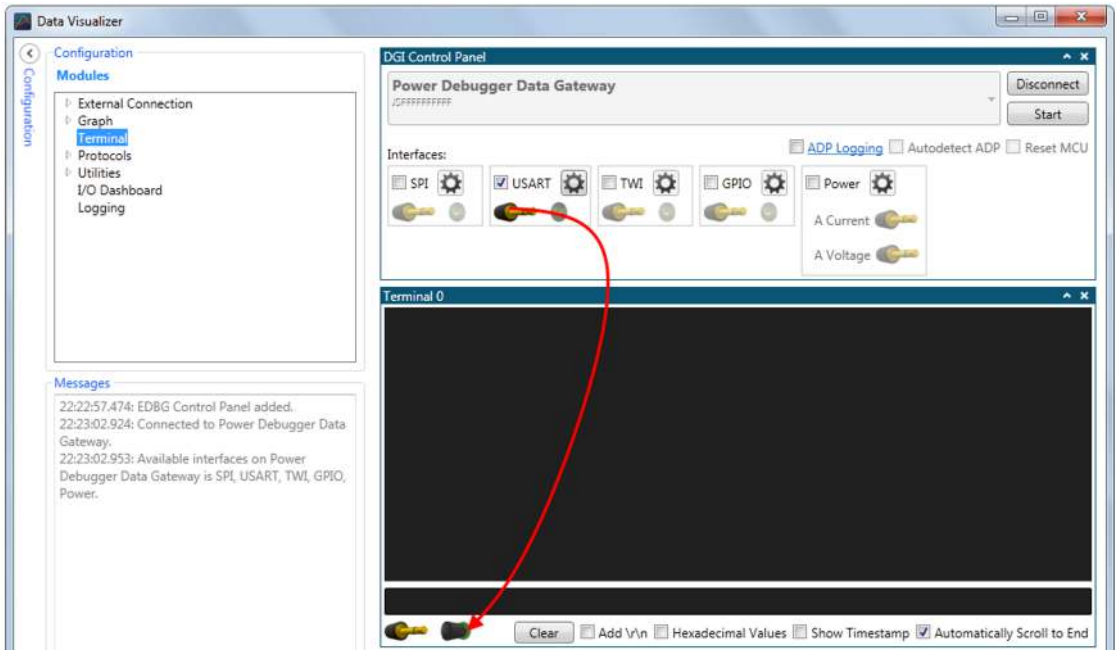
To do:

- Enable synchronous mode with timestamps in the **USART Configuration window**

4.1.13.2. Connect Interface to Terminal

To do:

- Open the configuration panel
- Add a Terminal view to the Visualizer
- Drag the **source** connector from the interface in the **DGI Control Panel** into the **sink** for the Terminal to make a connection

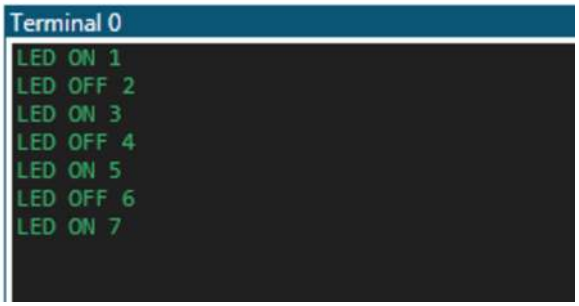


4.1.13.3. Run

To do:

- Start the session
- Press the button on the Xplained Mini to toggle the LED

On each toggle you should receive a message on the Terminal.



Result:
In this use case we have seen:

- Simple power graphing using Power Debugger and Atmel Data Visualizer
- Taking more accurate current and time measurements using cursors
- Simple USART-based code instrumentation

4.2. USB-powered Application

This use case will show you how to make use of the USB "pass-through" connectors on the Power Debugger. We will use a SAM L21 Xplained Pro kit as an example of a USB-powered product, where the designer is interested in both total power consumed over the USB port and that consumed by the target device on the board. The SAM L22 kit could also be used.

4.2.1. Requirements

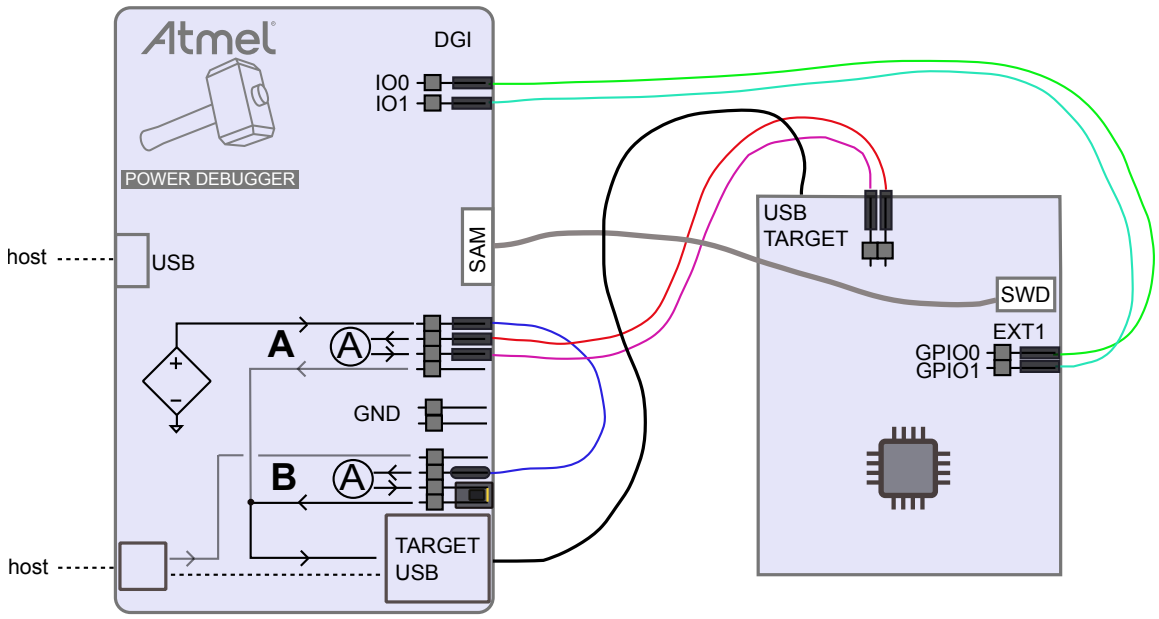
To be able to work through this example, the following is required:

- Host computer with Atmel Studio 7 (or later) installed (Atmel Data Visualizer is included)
- Atmel Power Debugger kit (cabling included)
- Atmel ATSAM L21 Xplained Pro kit, or similar target board with USB device connector, external programming header, and current measurement header

Tip: The SAM L21 Xplained Pro also has integrated on-board current measurement features, but these will not be used in this use case.

4.2.2. Initial Hardware Setup

A block diagram of the initial setup is shown here:



4.2.3. Connections

Connect the kits according to the block diagram. Connections are described here:

To power the target: We will power the target from the variable voltage supply. This enables testing a USB device over the full voltage range of the USB specification.

To do:

- Connect the variable voltage output to the input of the B channel using a strap cable. This enables us to measure the total current drawn by the target.
- Connect the output of the B channel to the POWER input of the USB type-A jack. This is simply done by using a jumper.
- Connect a USB cable from the type-A jack to the TARGET USB micro-jack

To measure the current drawn by the target MCU: the Xplained PRO board has a power 2-pin header next to the TARGET USB connector. Usually a jumper is in place here connecting VCC_TARGET to VCC_MCU. By removing this jumper and routing this circuit through a current measurement channel we can measure the current drawn only by the MCU, and not the whole board.

To do:

- Connect VCC_TARGET on the target board to the input of the A channel
- Connect the output of the A channel to the VCC_MCU pin of the target board

To make use of GPIO instrumentation: The Power Debugger has four GPIO channels, which can be monitored in Atmel Data Visualizer. We can then add pin toggles into the application code and view these events in Data Visualizer.

To do:

- Connect PB07 (GPIO0) on the Xplained PRO EXT1 header to IO0 on the DGI header on the Power Debugger
- Connect PB06 (GPIO1) on the Xplained PRO EXT1 header to IO1 on the DGI header on the Power Debugger

To program and debug the target device: Although the Xplained PRO has debug capability, we will use the Power Debugger for this purpose.

To do:

- Connect the 10-pin debug cable from the SAM header of the Power Debugger to the CORTEX DEBUG header of the Xplained PRO

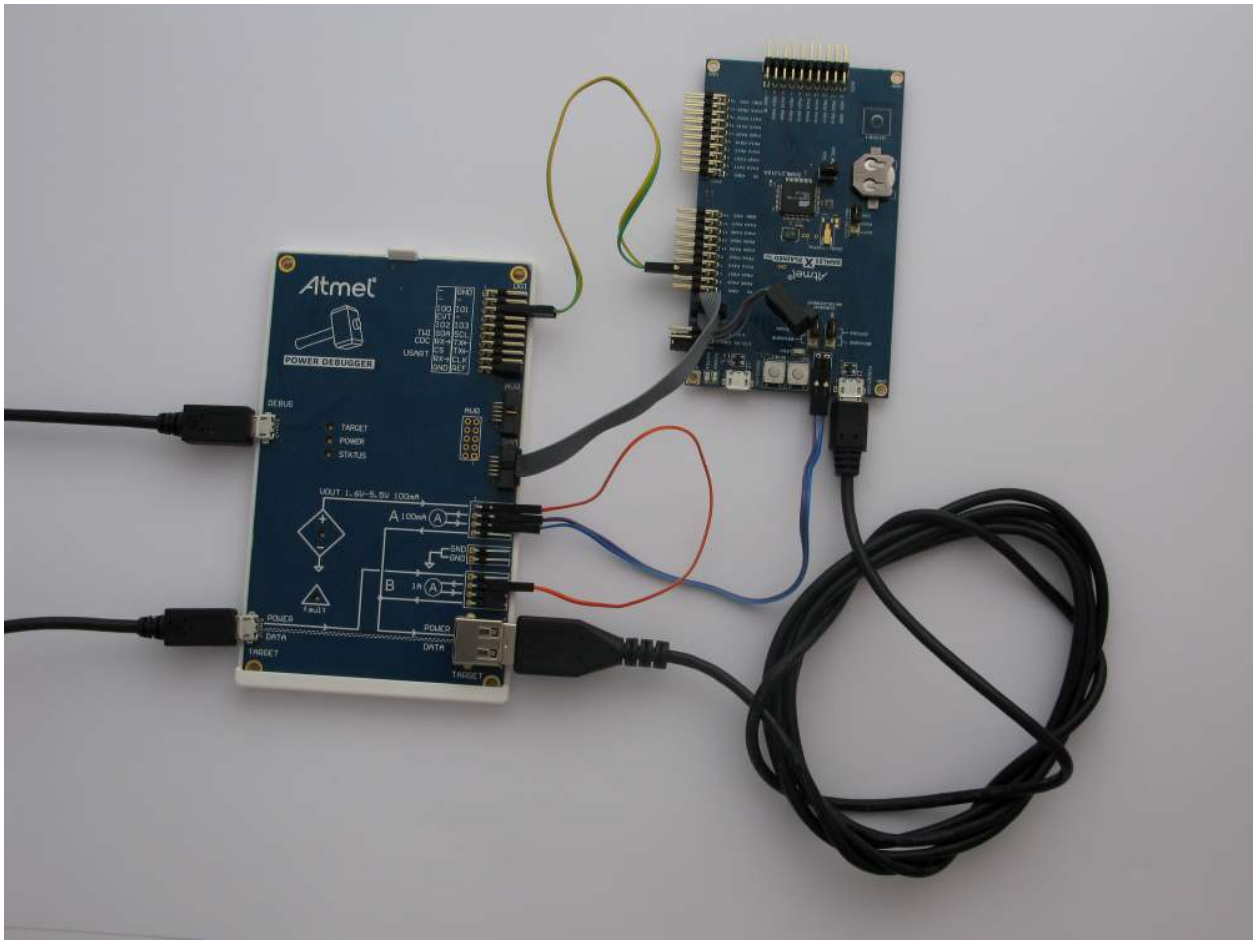
To connect all this to the host computer:

To do:

- Connect a USB cable from the DEBUG connector of the Power Debugger to the host computer
- Connect a USB cable from the TARGET connector of the Power Debugger to the host computer

Tip: GND connections are not necessary when the programming header is connected.

Your setup should look something like this:



Note: The picture is slightly wrong. The debug cable is connected to the SAM debug connector but should have been connected to the AVR debug connector.

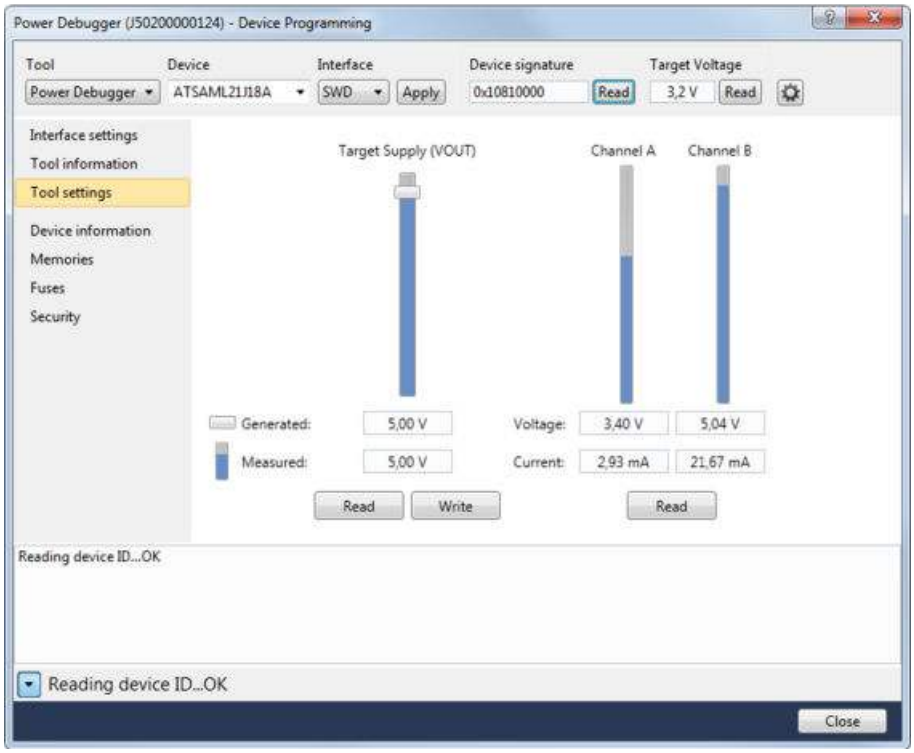
4.2.4. VOUT Target Supply

The Power Debugger can supply voltage to a target in the range of 1.6V to 5.5V, up to 100mA. The voltage will be applied to the pin as shown in the silkscreen, and must be routed to the appropriate load by the user. VOUT can be set in a number of ways.

Important: VOUT setting is stored in volatile memory on the Power Debugger. Toggling power on the tool will result in VOUT being disconnected. This is done to protect the user's hardware.

4.2.4.1. Setting VOUT Using the Programming Dialog

VOUT can be set and read back using the programming dialog in Atmel Studio. This is done using the Tool settings tab, which is available when using the Power Debugger. Use the slider to set the desired voltage and click Write to apply changes. The measured value is automatically read back.



4.2.4.2. Setting VOUT Using atprogram.exe

VOUT can be set using the atprogram command line utility via the parameters command:

```
atprogram.exe -t powerdebugger parameters -ts 5.0
```

VOUT value can be sampled and read back using:

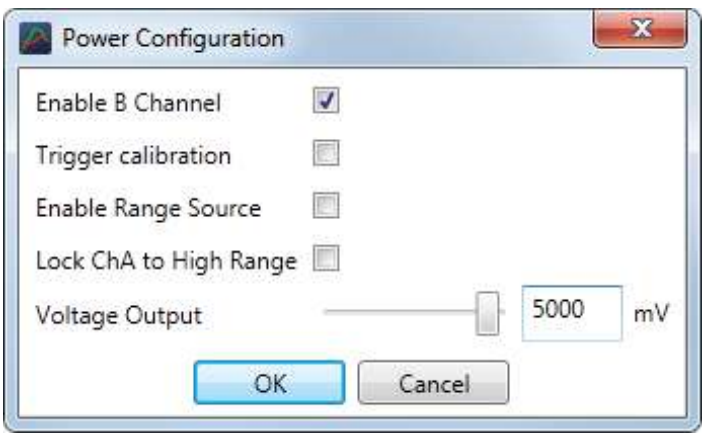
```
atprogram.exe -t powerdebugger parameters -tsout
```

The VOUT set-point can be read back using:

```
atprogram.exe -t powerdebugger parameters -tssp
```

4.2.4.3. Setting VOUT Using Data Visualizer

VOUT can be set using Data Visualizer by opening the configuration for the Power interface.



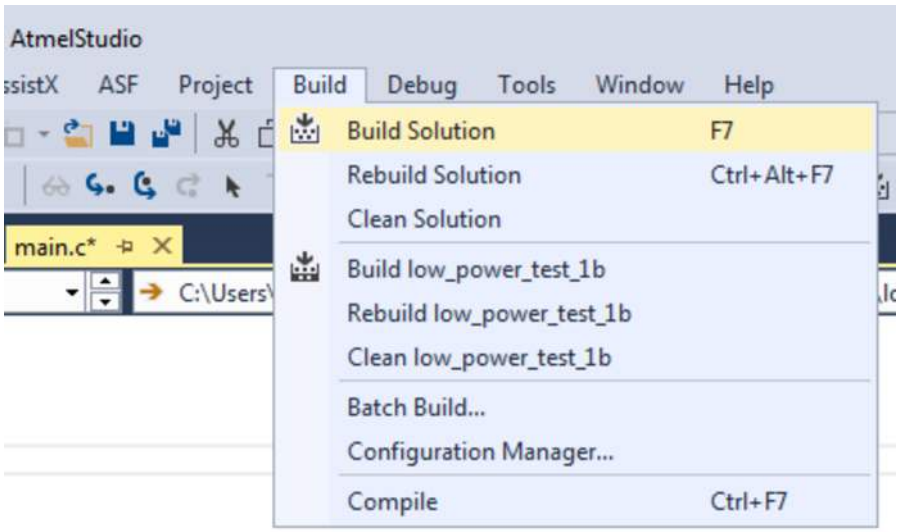
4.2.5. Using Both Measurement Channels

The two current measurement channels of the Power Debugger are useful when analyzing current consumption of two different domains. For example both board-level current draw and MCU-only current draw can be plotted.

To best demonstrate a USB hardware device, we will make use of a Mass Storage Device example from ASF.

To do:

- In Atmel Studio, create a New Example Project
- In the New Example dialog, select the SAM L21 device family (or other relevant device) and filter by the keyword "MSC"
- Select the USB Device MSC Example
- Build the project/solution (F7)

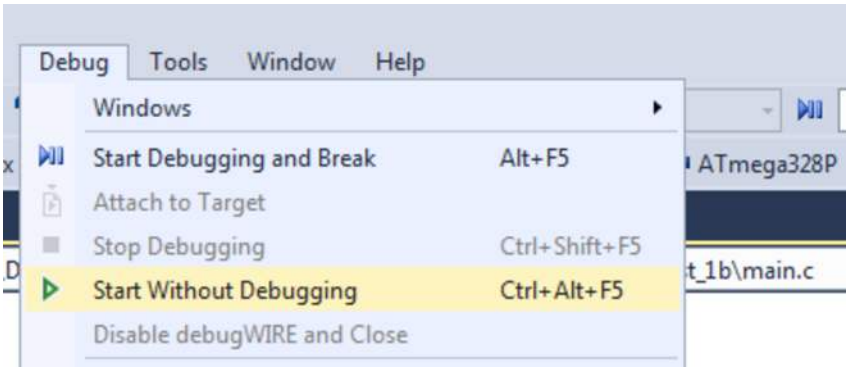


To do:

- Open the project properties (right click the project in the **Solution Explorer** and select **Properties**)
- On the **Tool** tab select the appropriate tool and interface

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

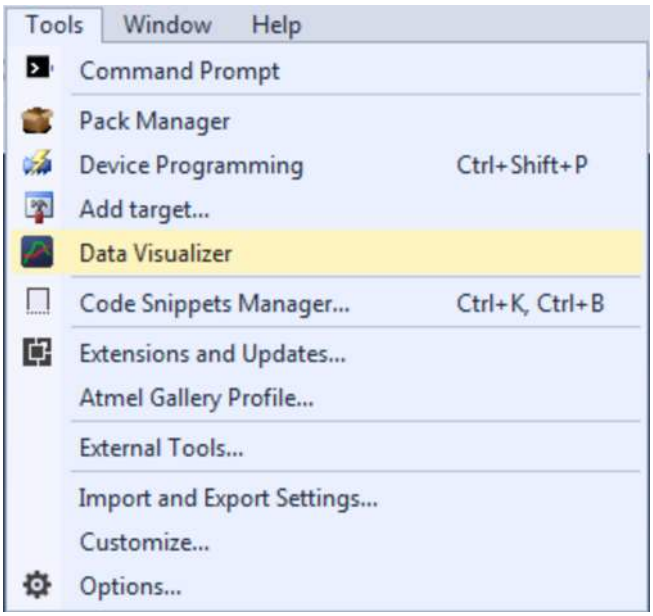
Info: In this part of this example we are going to make use of the programming mode only. In most systems running code through a debugger will not yield accurate current measurements. This is because the target device's debug module (OCD) requires a clock source which cannot be disabled while debugging.



4.2.6. 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:



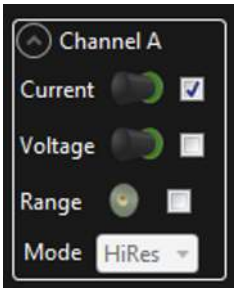
Kits supporting Data Visualizer functionality include a shortcut to the extension on their start page in Atmel Studio.

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.

4.2.7. Two Channel Measurement

When using hardware with two measurement channels, the Data Visualizer will display both in the same graph (unless disabled in the **Power Configuration**).

On the **Control panel** on the right of the module you can show or hide the current and voltage plots as well as range information when available.



By default both channels will be shown in the **Power Analysis** graph but each plot can be moved up or down to separate them as best suited.

4.2.8. Scaling and Scrolling a Graph

Tip: Turn off **Auto-scroll** and **Automatically fit Y** to more closely examine a plot while it is still running.

Use the **scale** and **offset** controls in order to move the plots according to your needs. The mouse scroll-wheel is useful in this regard:

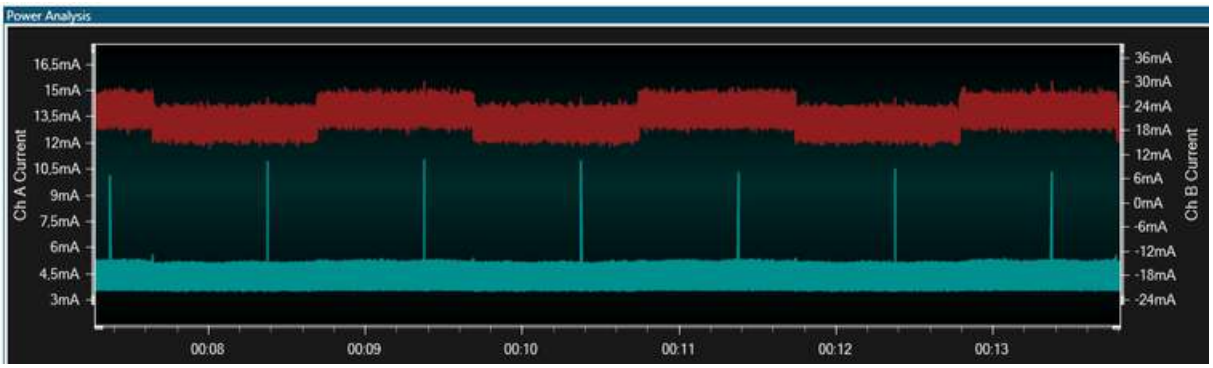
- Shift-scroll on the plot to zoom on the time (X) axis
- Ctrl-scroll on the plot to zoom on the Y axis
- Drag the graph to pan on the time (X) axis and move (offset) the Y axis
- Drag the scales on the left and right to move respective channels in the Y axis (offset)
- Ctrl-scroll on the respective axis scale to zoom that channel
- Right-click and drag to select an area to zoom

4.2.9. Mass Storage Example

To do:

- Program the Mass Storage Class example into the target device using Start Without Debugging (Ctrl+Alt+F5)
- Stop Data Visualizer after capturing a plot

With some scrolling and scaling the plots should look something like this:



The red graph is channel B, and represents the current drawn by the entire board. It appears to toggle between two modes at about 1 second intervals. This is the LED toggling ON and OFF.

The blue graph is the A channel, and represents the current drawn by the target MCU only. It appears to draw about 4.5mA continuously, with pulses of up to 10mA, also at 1 second intervals.

Tip: Of interest: disabling the USB 'disk' in the Windows device manager will stop this pulse from occurring, indicating that this is probably triggered by an event from Windows.

To do: Open the Windows explorer and perform a format of the Mass Storage Device.

Caution: Take care to format the right disk!

On the plot you should now see periods of much higher current consumption on both board and MCU channels, as shown here:

4.2.10. GPIO Instrumentation

Pin toggling is often a simple and useful mechanism for debugging applications. We will now add some simple instrumentation to profile the read and write cycles of the Mass Storage device.

To do: Find the implementation of the ui_start_read () function in the file ui.c This can be easily done by opening ui.h, selecting the function, and using the Goto Implementation option on the context menu.

Add instrumentation as shown here, which toggles GPIO0 for read accesses and GPIO1 for write accesses:

```
void ui_start_read(void)
{
    // GPIO0 high
    port_pin_set_output_level (EXT1_PIN_GPIO_0, true);
}

void ui_stop_read(void)
{
    // GPIO0 low
    port_pin_set_output_level (EXT1_PIN_GPIO_0, false);
}

void ui_start_write(void)
{
    // GPIO1 high
    port_pin_set_output_level (EXT1_PIN_GPIO_1, true);
}

void ui_stop_write(void)
{
    // GPIO1 low
    port_pin_set_output_level (EXT1_PIN_GPIO_1, false);
}
```

Find the board_init.c file and add initialization of GPIO0 and 1. This can be added by copying the LED_0_PIN initialization in system_board_init (void)

```
    // GPIO0 to output
    port_pin_set_config(EXT1_PIN_GPIO_0, &pin_conf);
```

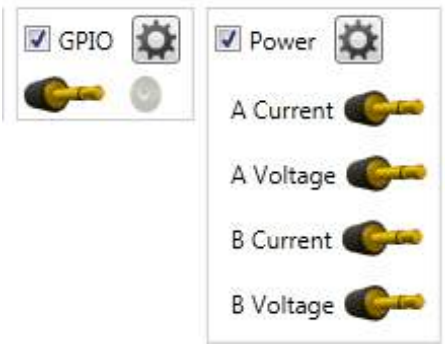
```
    // GPIO1 to output
    port_pin_set_config(EXT1_PIN_GPIO_1, &pin_conf);
```

You can also comment out the LED toggle, as it is not relevant here.

```
void ui_process(uint16_t framenumber)
{
    /*if (0 == framenumber) {
        LED_On(LED_0_PIN);
    }
    if (1000 == framenumber) {
        LED_Off(LED_0_PIN);
    }*/
}
```

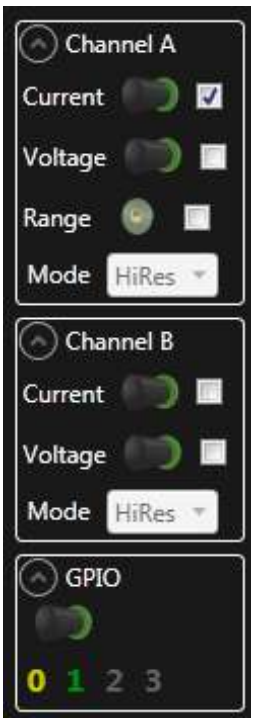
To do:

- 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
- Select both the Power interface and GPIO and start visualizing



To do:

- Open the control panel
- Uncheck channel B
- Uncheck GPIO 2 and 3



The power graph should now look similar to the previous exercise. Now perform a disk format through Windows, and note how the GPIO 0 and 1 signals toggle. Stop the visualizer or disable auto-scrolling and zoom in to an interesting bit. One can clearly see how each read and write access has a GPIO toggle to indicate a start and stop condition.

Since this is a fully compliant disk drive, try a few other operations and see how the read and write access events are affected:

- Create a new text file
- Add content using notepad
- Save the file
- Re-open the file
- Delete the file

Tip: Read caching by the operating system might cause read access not to be executed until the cache is flushed after a write.

4.2.11. Code Correlation

We will now demonstrate how program counter polling can be used to correlate code execution to power consumption.

The Power Debugger can be instructed by Data Visualizer to repeatedly poll the program counter as fast as possible during execution. While this will yield a very low percentage of code lines traced, it can be useful in trapping areas of code which use more power than intended.

4.2.11.1. Enable Program Counter Polling

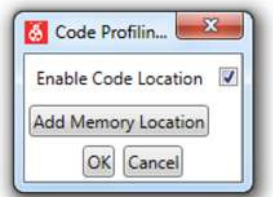
To view program counter samples together with current measurement data both the **Power** interface and the **Code Profiling** interface must be enabled.

To do:

- Enable both **Power** interface and **Code Profiling** interface in **DGI Control Panel**

To do:

- Open the **Code Profiling Configuration** dialog by pushing the gear button on the **Code Profiling** interface
- Select **Enable Code Location**



4.2.11.2. Run

Important: Code correlation is only available when Data Visualizer is run as an extension within Atmel Studio. This is because it needs access to symbolic information only available when the debugger is running.

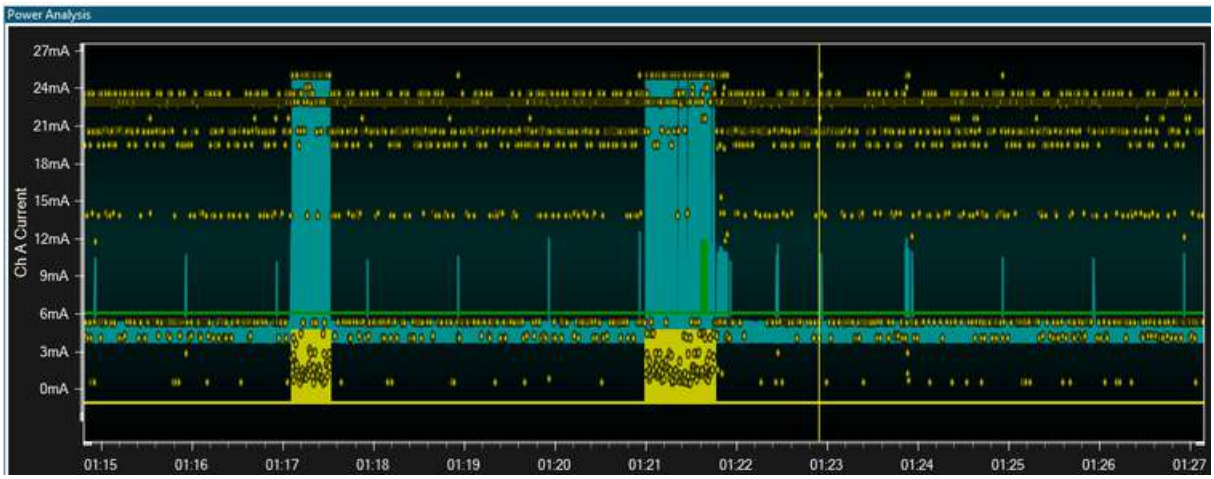
Important: Code correlation requires that the debugger is running, so use **Start Debugging** in Atmel Studio. Just programming the target or using **Start Without Debugging** will not work.

Important: Take care to format the right disk!

To do:

- Start a Data Visualizer session and let it run
- Perform a disk format and capture the resulting plot
- Stop the session or disable auto-scrolling
- Switch off GPIO 2 and 3

Zooming in on the format activity identified previously, you should see a plot like this:



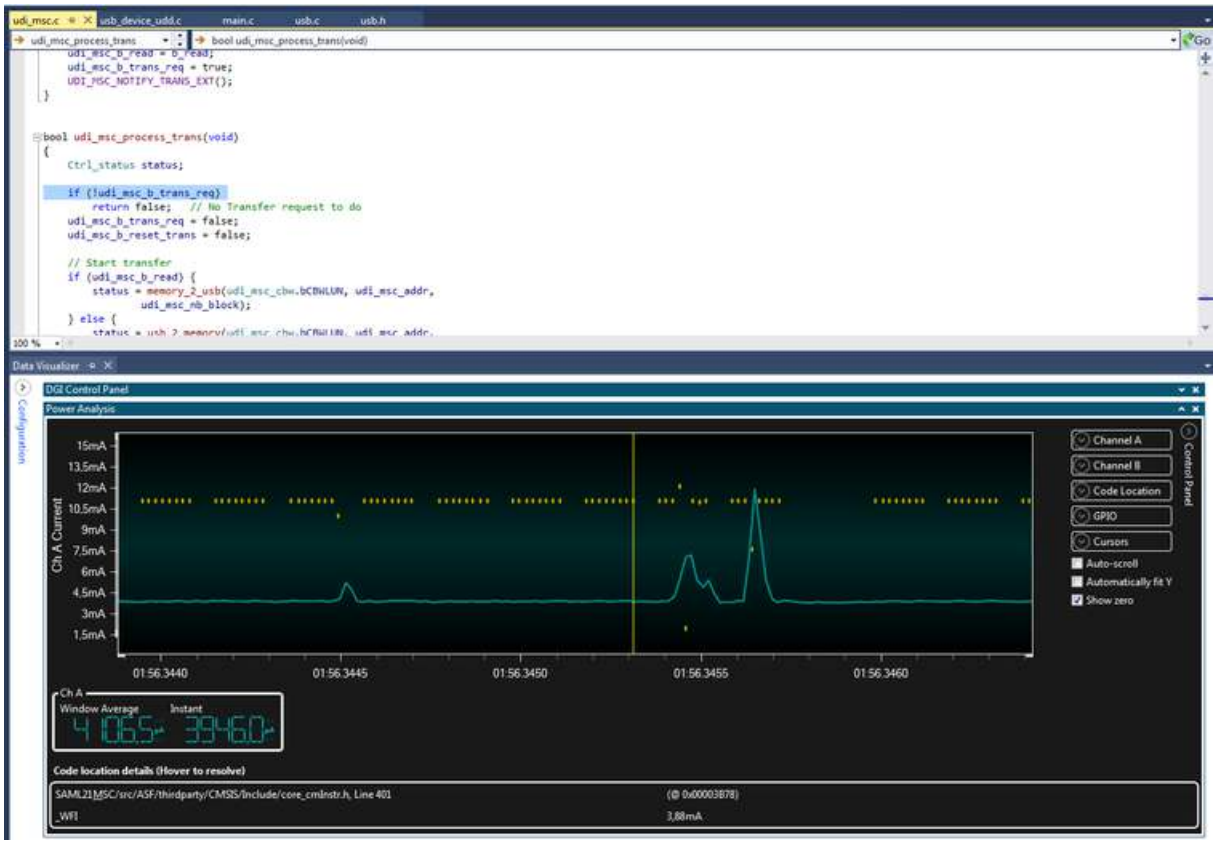
The yellow points plotted on the graph represent polled program counter values. Their location on the y axis is a visual representation of their location in the code-space of the target device. The relative grouping of samples shows us execution of different functions. Patterns can easily be seen using this technique. Notice how the format activity (as seen by GPIO activity) has a different distribution of program counter values compared to the parts with no GPIO activity.

Hovering over one of the samples shows the location of that sample in the **Code Location Details** box below the graph, as well as the value of the current consumption sample at that point.

Zooming in on one of the larger current peaks, we might see something like this:

The smaller current spikes are very regular, and occur every millisecond. Some of these have program counter sample values resolved as usb_device_interrupt_handler, which confirms this is a USB millisecond tick pulse waking the CPU briefly. The interrupt is executed too quickly to be caught on all executions. The larger current spikes are often associated with another USB function (csw_send / receive).

Double clicking on one of the program counter samples will open the editor and highlight the line corresponding to that sample.



Tip: Some samples will be marked as "Could not resolve". This can happen when there is no source available for the location, for example if the sample corresponds to a precompiled library function.

4.2.12. Data Polling

We will now see how Data Visualizer can poll variables from the target and display their values in graphical form.

Important: Data polling is only available when Data Visualizer is run as an extension within Atmel Studio. This is because it needs to access the debug system on the device through the Atmel Studio debugger backend.

First we will add a few lines of code containing variables to poll.

To do: Open ui.c and add two global variables to the top of the file.

```
volatile uint32_t write_count = 0;
volatile uint32_t read_count = 0;
```

Important: Declaring variables you are interested in polling as volatile will ensure that they are placed in SRAM and that their values will not be cached in registers by the compiler. Registers cannot be polled, only SRAM locations.

Tip: Data polling operates on absolute SRAM locations. It is thus advised to use global variables for this purpose so that they are always available at the same location in SRAM. Polling locations in the stack can yield unpredictable results based on the stack context at the time of polling.

To do: Modify the two 'start' functions in ui.c to increment read and write counters on each access started.

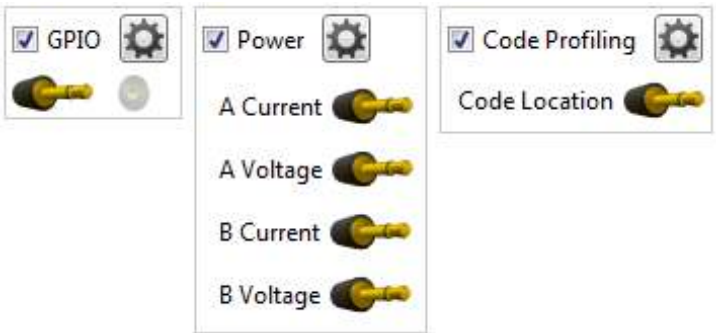
```
void ui_start_read(void)
{
    port_pin_set_output_level(EXT1_PIN_GPIO_0, true);
    read_count++;
}
```

```
void ui_start_write(void)
{
    port_pin_set_output_level(EXT1_PIN_GPIO_1, true);
    write_count++;
}
```

To do:

- Build the project/solution (F7)
- Open Data Visualizer
- Connect

For data polling functionality we need to enable the Code Profiling interface.



To do:

- Start the Data Visualizer session
- Launch the debug session using Start Debugging and Break (Alt + F5)

Data polling operates on SRAM locations, so to find out where variables are located in SRAM we need to use the Atmel Studio watch window.

To do:

- Locate the two global variables we added to ui.c
- Right-click each variable and select Add to Watch
- Examine the type field of each variable in the watch window to find its location

Watch 1

Name	Value	Type
read_count	0	volatile uint32_t(static storage at address 0x200000e8.)
write_count	0	volatile uint32_t(static storage at address 0x200000ec.)

Switch back to the Data Visualizer to set up the **Code Profiling** interface and to connect the two variables to a graph.

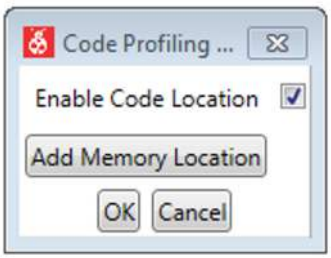
4.2.12.1. Open Code Profiling Configuration Dialog

To do:

- Enable the **Code Profiling** interface by ticking off the check box for the **Code Profiling** interface in the **DGI Control Panel**

To do:

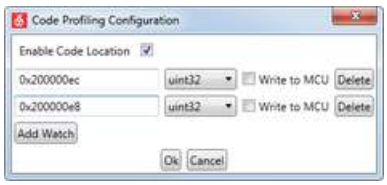
- Open the **Code Profiling Configuration** window by pushing the gear button



4.2.12.2. Add Data Polling Locations

To do:

- Push **Add Memory Location** button for each memory location to be added
- Fill in the address and format of each location

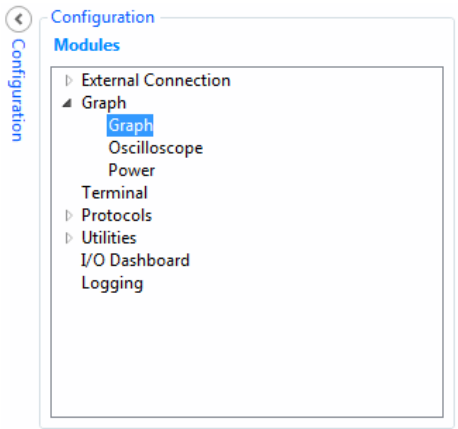


4.2.12.3. Display Variables in Graph

We now have two variables to monitor, and need to display them on a graph.

To do:

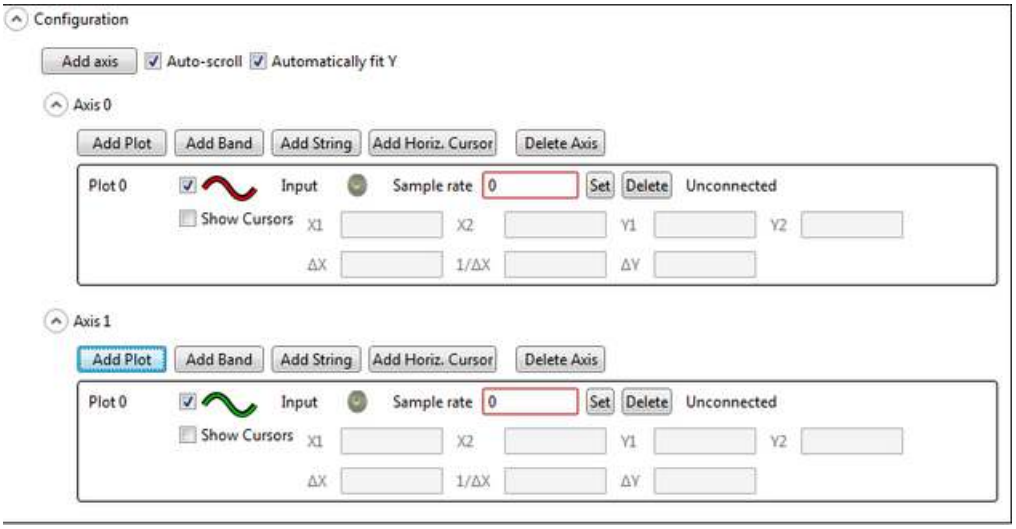
- Open the **Configuration** panel in Data Visualizer
- Add a graph by double clicking the **Graph** module



A new Graph element will open with one y axis configured. However, we have two unrelated variables to monitor, so we need two axes.

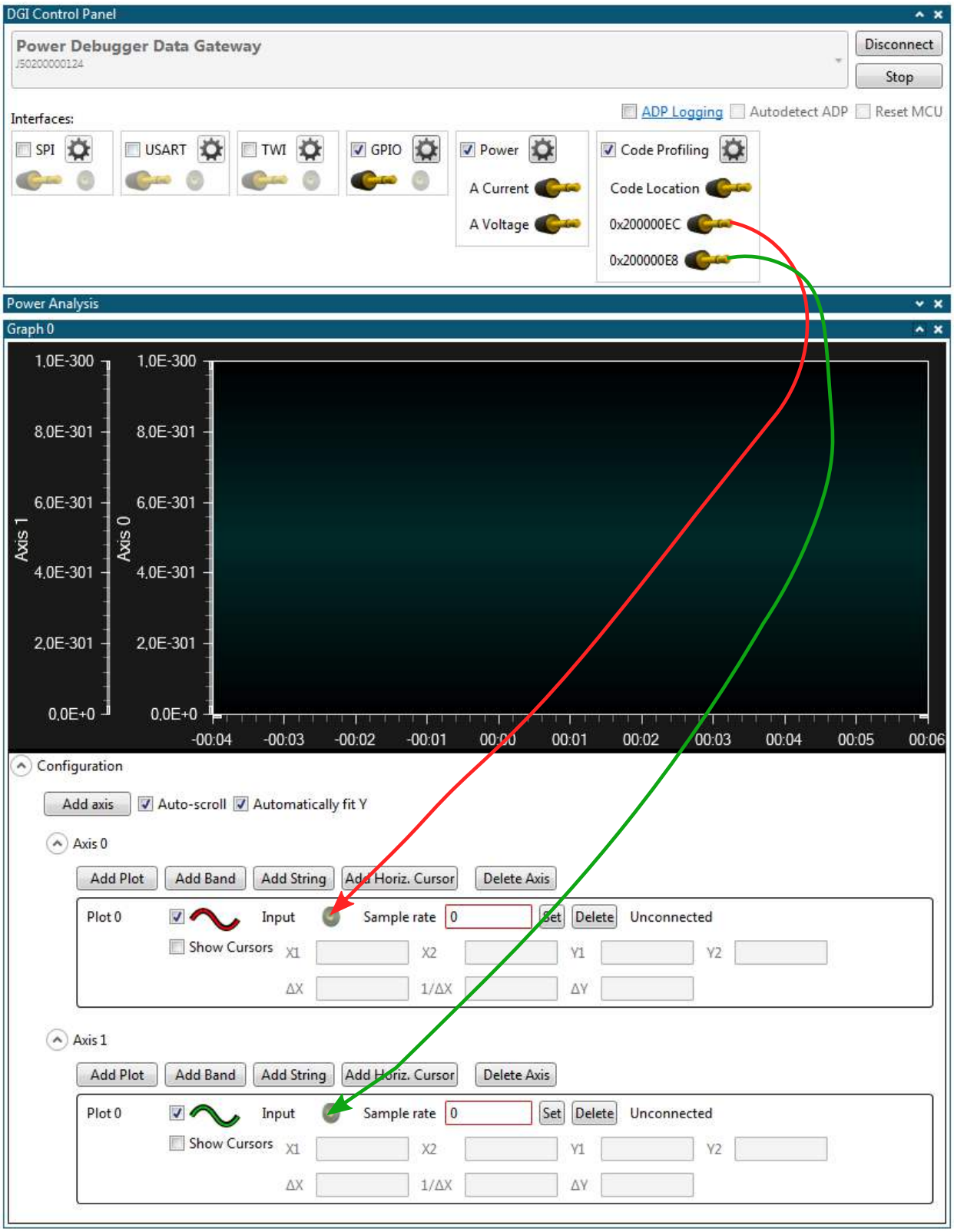
To do:

- Click the **Add axis** button to add an additional axis
- For each of the axes, click the **Add Plot** button



We now have the sources (variables) and sinks (axes), so its just to connect them together.

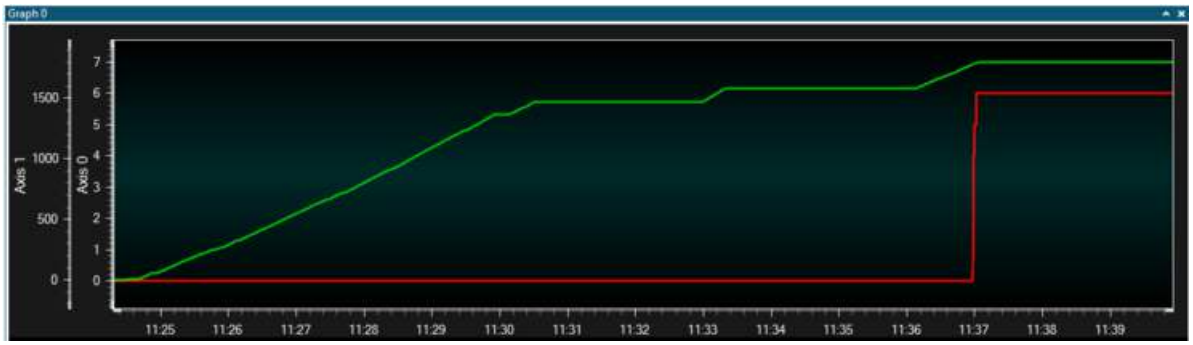
To do: Drag each of the **source** plugs on the **Code Profiling** interface into the Input (**sink**) jack of the plot.


DGI Control Panel
Power Debugger Data Gateway
/50200000124
Disconnect
Stop
Interfaces:
ADP Logging
Autodetect ADP
Reset MCU
SPI
USART
TWI
GPIO
Power
A Current
A Voltage
Code Profiling
Code Location
0x200000EC
0x200000E8
Power Analysis
Graph 0
1,0E-300
8,0E-301
6,0E-301
4,0E-301
2,0E-301
0,0E+0
Axis 1
Axis 0
-00:04
-00:03
-00:02
-00:01
00:00
00:01
00:02
00:03
00:04
00:05
00:06
Configuration
Add axis
Auto-scroll
Automatically fit Y
Axis 0
Add Plot
Add Band
Add String
Add Horiz. Cursor
Delete Axis
Plot 0
Input
Sample rate
0
Set
Delete
Unconnected
Show Cursors
X1
X2
Y1
Y2
ΔX
1/ΔX
ΔY
Axis 1
Add Plot
Add Band
Add String
Add Horiz. Cursor
Delete Axis
Plot 0
Input
Sample rate
0
Set
Delete
Unconnected
Show Cursors
X1
X2
Y1
Y2
ΔX
1/ΔX
ΔY

To do: In Atmel Studio click Continue (F5) to resume execution.

Tip: A USB device in the HALT state no longer responds to Windows events, and may be disconnected from the bus if held in this state for too long. To remedy this simply reset execution in Atmel Studio.

Look at the output in the graph in Data Visualizer. Format the disk and watch how the write cycles counter increments. Both values are plotted on independent axes, so they can be scaled accordingly. The output should look something like this:



4.2.13. Application Interaction Using Dashboard Controls

We will now see how components placed on a dashboard in Data Visualizer can be hooked up to variables in the application, and how the dashboard can thus interact with the application at run-time.

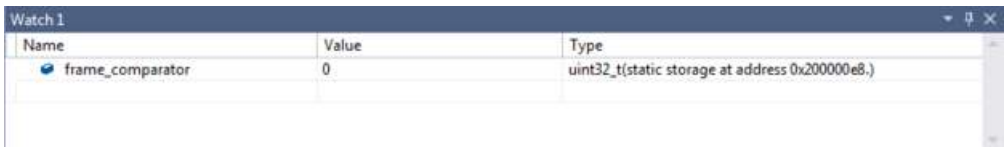
Instead of a predefined interval of 1000 USB sync pulses (1 second), we will add a variable compare reference to the original code.

To do: Modify ui.c to include a LED blinker in the ui_process() handler as shown here.

```
volatile uint32_t frame_comparator = 100;
volatile uint32_t frames_received = 0;
void ui_process(uint16_t framenumber)
{
    frames_received++;
    if (frames_received >= frame_comparator) {
        LED_Toggle(LED_0_PIN);
        frames_received = 0;
    }
}
```

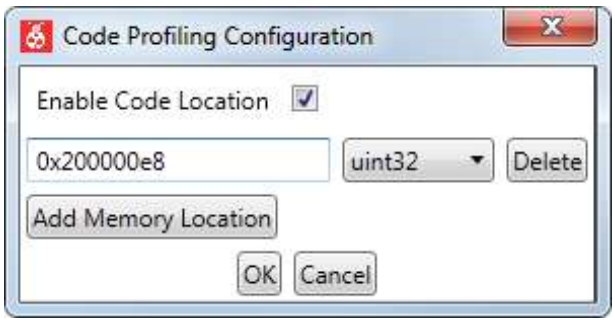
To do:

- Build the project/solution (F7)
- Launch a debug session using Start Debugging and Break (Alt + F5)
- Find the location of the variable uint32_t frame_comparator



To do:

- Open Data Visualizer
- Connect
- Add the location of the frame_comparator in the Code Profiling Configuration window

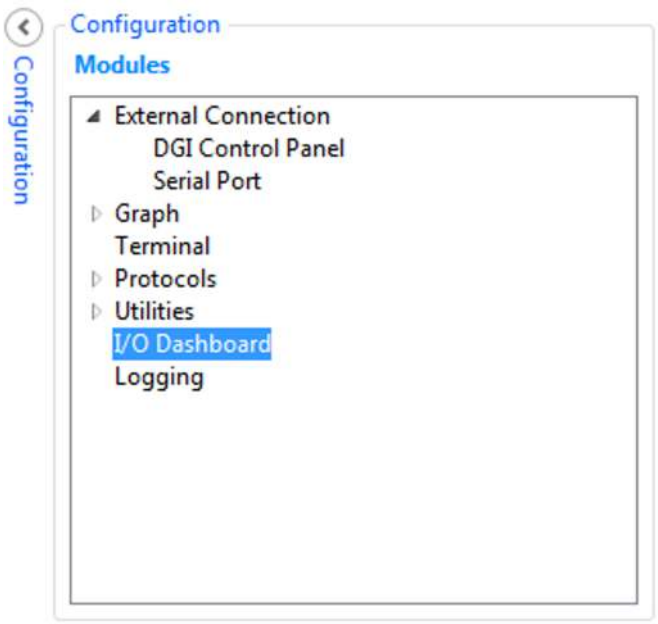


A Data Visualizer dashboard can now be made with controls which manipulate the value of this variable.

4.2.13.1. Add I/O Dashboard

To do:

- Open the configuration panel
- Add a new I/O Dashboard component by double clicking the I/O Dashboard module

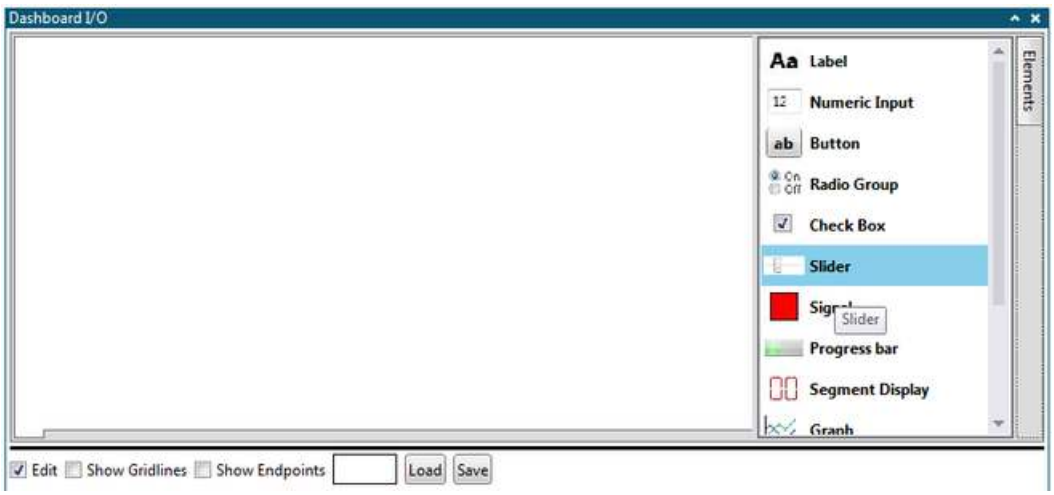


4.2.13.2. Configure Dashboard

We can now add a slider control to the dashboard.

To do:

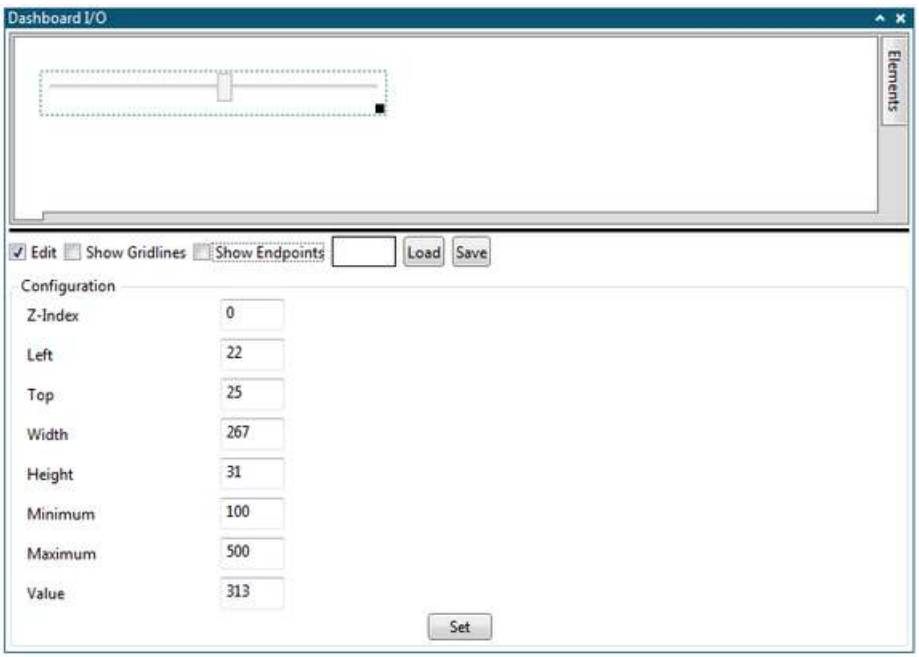
- Check the Edit checkbox
- Open the Elements tab
- Drag a Slider element onto the dashboard



A slider control needs to have some configuration parameters.

To do: Select the slider element and set its properties:

- Maximum = 500
- Minimum = 100



We can now add a segment display control to the dashboard.

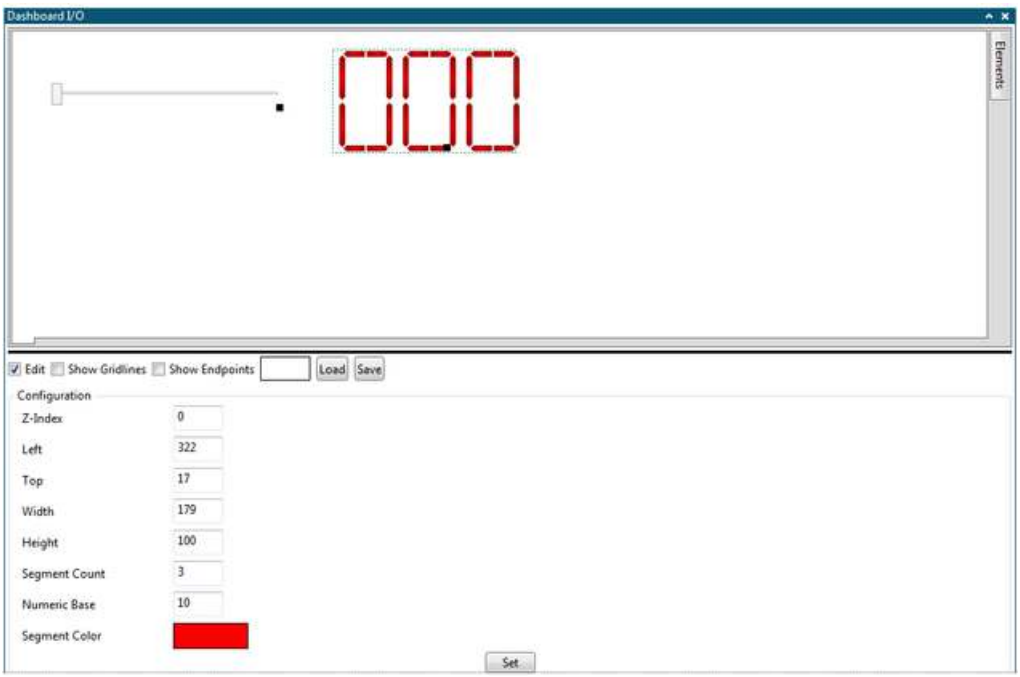
To do:

- Check the Edit checkbox
- Open the Elements tab
- Drag a Segment Display element onto the dashboard

A segment display control needs to have some configuration parameters.

To do: Select the segment display element and set its properties:

- Segment Count = 3

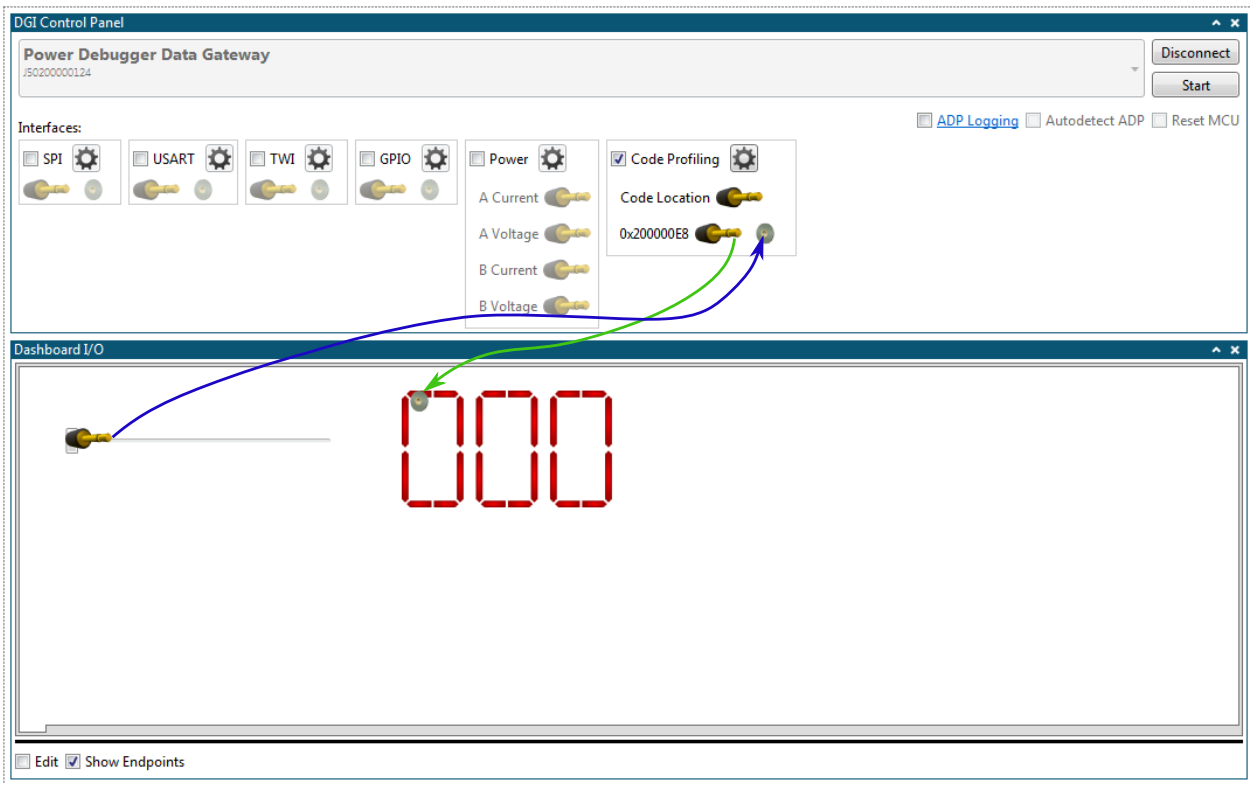


The slider control can now be used as a source which can be connected to any relevant sink in Data Visualizer. The segment display can similarly be used as a sink to connect any relevant source to.

The Code Profiling data polling interface provides both a source of data and a sink of data. We can now connect the slider to the sink and the segment display to the source.

To do:

- Uncheck the Edit checkbox
- Check the Show Endpoints checkbox
- Connect sources to sinks by dragging each **source** plug and drop it on a **sink**



4.2.13.3. Run

Now that the connections have been made in Data Visualizer, we can put the system into a running state and interact with the variable through the GUI.

To do:

- Uncheck the Show Endpoints checkbox
- Start Data Visualizer
- Resume execution in Atmel Studio (F5)

The slider is now in control of the frame_comparator variable in the application code. Drag the slider, and notice that the LED blink frequency changes. Any change in the slider position will be sent to the target device through the debug interface, and a new value stored to the variable. At the same time the value is also read back from the target, and displayed on the segment display.


Dashboard I/O
3 18

5. On-chip Debugging

5.1. Introduction

On-chip Debugging
An on-chip debug module is a system allowing a developer to monitor and control execution on a device from an external development platform, usually through a device known as a *debugger* or *debug adapter*.

With an OCD system the application can be executed whilst maintaining exact electrical and timing characteristics in the target system, while being able to stop execution conditionally or manually and inspect program flow and memory.

Run Mode
When in Run mode, the execution of code is completely independent of the Power Debugger. The Power Debugger will continuously monitor the target device to see if a break condition has occurred. When this happens the OCD system will interrogate the device through its debug interface, allowing the user to view the internal state of the device.

Stopped Mode
When a breakpoint is reached, the program execution is halted, but some I/O may continue to run as if no breakpoint had occurred. For example, assume that a USART transmit has just been initiated when a breakpoint is reached. In this case the USART continues to run at full speed completing the transmission, even though the core is in stopped mode.

Hardware Breakpoints
The target OCD module contains a number of program counter comparators implemented in the hardware. When the program counter matches the value stored in one of the comparator registers, the OCD enters stopped mode. Since hardware breakpoints require dedicated hardware on the OCD module, the number of breakpoints available depends upon the size of the OCD module implemented on the target. Usually one such hardware comparator is ‘reserved’ by the debugger for internal use.

Software Breakpoints
A software breakpoint is a BREAK instruction placed in program memory on the target device. When this instruction is loaded, program execution will break and the OCD enters stopped mode. To continue execution a "start" command has to be given from the OCD. Not all Atmel devices have OCD modules supporting the BREAK instruction.

5.2. SAM Devices with JTAG/SWD

All SAM devices feature the SWD interface for programming and debugging. In addition, some SAM devices feature a JTAG interface with identical functionality. Check the device datasheet for supported interfaces of that device.

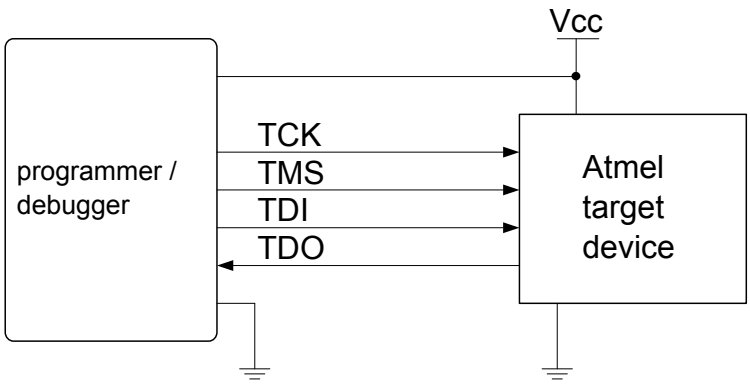
5.2.1. ARM CoreSight Components

Atmel ARM Cortex-M based microcontrollers implement CoreSight compliant OCD components. The features of these components can vary from device to device. For further information consult the device's datasheet as well as CoreSight documentation provided by ARM.

5.2.2. JTAG Physical Interface

The JTAG interface consists of a 4-wire Test Access Port (TAP) controller that is compliant with the IEEE® 1149.1 standard. The IEEE standard was developed to provide an industry-standard way to efficiently test circuit board connectivity (Boundary Scan). Atmel AVR and SAM devices have extended this functionality to include full Programming and On-chip Debugging support.

Figure 5-1. JTAG Interface Basics



5.2.2.1. SAM JTAG Pinout (Cortex-M debug connector)

When designing an application PCB which includes an Atmel SAM with the JTAG interface, it is recommended to use the pinout as shown in the figure below. Both 100-mil and 50-mil variants of this pinout are supported, depending on the cabling and adapters included with the particular kit.

Figure 5-2. SAM JTAG Header Pinout

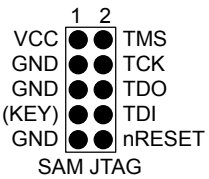


Table 5-1. SAM JTAG Pin Description

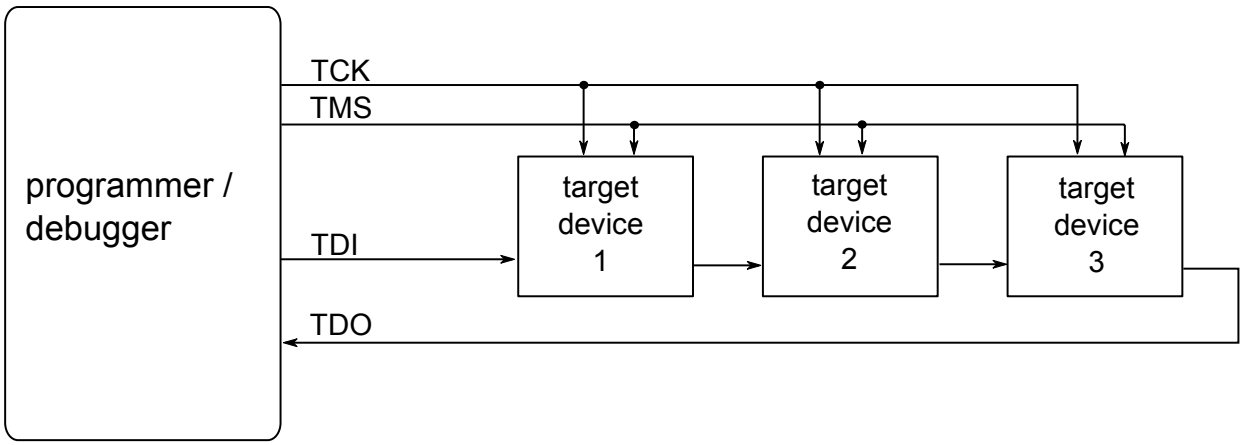
Name	Pin	Description
TCK	4	Test Clock (clock signal from the Power Debugger into the target device).
TMS	2	Test Mode Select (control signal from the Power Debugger into the target device).
TDI	8	Test Data In (data transmitted from the Power Debugger into the target device).
TDO	6	Test Data Out (data transmitted from the target device into the Power Debugger).
nRESET	10	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the Power Debugger to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
VTG	1	Target voltage reference. The Power Debugger samples the target voltage on this pin in order to power the level converters correctly. The Power Debugger draws less than 1mA from this pin in this mode.
GND	3, 5, 9	Ground. All must be connected to ensure that the Power Debugger and the target device share the same ground reference.
KEY	7	Connected internally to the TRST pin on the AVR connector. Recommended as not connected.

Tip: Remember to include a decoupling capacitor between pin 1 and GND.

5.2.2.2. JTAG Daisy Chaining

The JTAG interface allows for several devices to be connected to a single interface in a daisy chain configuration. The target devices must all be powered by the same supply voltage, share a common ground node, and must be connected as shown in the figure below.

Figure 5-3. JTAG Daisy Chain



When connecting devices in a daisy chain, the following points must be considered:

- All devices must share a common ground, connected to GND on the Power Debugger probe
- All devices must be operating on the same target voltage. VTG on the Power Debugger must be connected to this voltage.
- TMS and TCK are connected in parallel; TDI and TDO are connected in a serial chain.
- nSRST on the Power Debugger probe must be connected to RESET on the devices if any of the devices in the chain disables its JTAG port
- "Devices before" refers to the number of JTAG devices that the TDI signal has to pass through in the daisy chain before reaching the target device. Similarly "devices after" is the number of devices that the signal has to pass through after the target device before reaching the Power Debugger TDO pin.
- "Instruction bits "before" and "after" refers to the total sum of all JTAG devices' instruction register lengths, which are connected before and after the target device in the daisy chain
- The total IR length (instruction bits before + Atmel target device IR length + instruction bits after) is limited to a maximum of 256 bits. The number of devices in the chain is limited to 15 before and 15 after.

Tip:
Daisy chaining example: TDI → ATmega1280 → ATxmega128A1 → ATUC3A0512 → TDO.

In order to connect to the Atmel AVR XMEGA® device, the daisy chain settings are:

- Devices before: 1
- Devices after: 1

- Instruction bits before: 4 (8-bit AVR devices have 4 IR bits)
- Instruction bits after: 5 (32-bit AVR devices have 5 IR bits)

Table 5-2. IR Lengths of Atmel MCUs

Device type	IR length
AVR 8-bit	4 bits
AVR 32-bit	5 bits
SAM	4 bits

5.2.3. Connecting to a JTAG Target

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 STK600 stack should use the AVR header.

In addition the Power Debugger also has an unpopulated 100-mil 10-pin JTAG connector in the AVR pinout on its PCB. This is wired directly to the 50-mil AVR header.

The recommended pinout for the 10-pin AVR JTAG connector is shown in Figure 5-6.

The recommended pinout for the 10-pin ARM Cortex Debug connector is shown in Figure 5-2.

Direct connection to a standard 10-pin 50-mil header

Use the 50-mil 10-pin flat cable (included in some kits) to connect directly to a board supporting this header type. Use the AVR connector port on the Power Debugger for headers with the AVR pinout, and the SAM connector port for headers complying with the ARM Cortex Debug header pinout.

The pinouts for both 10-pin connector ports are shown below.

Connection to a standard 10-pin 100-mil header

Use a standard 50-mil to 100-mil adapter to connect to 100-mil headers. An adapter board (included in some kits) can be used for this purpose, or alternatively the JTAGICE3 adapter can be used for AVR targets.

Important:
The JTAGICE3 100-mil adapter cannot be used with the SAM connector port, since pins 2 and 10 (AVR GND) on the adapter are connected.

Connection to a custom 100-mil header

If your target board does not have a compliant 10-pin JTAG header in 50- or 100-mil, you can map to a custom pinout using the 10-pin "mini-squid" cable (included in some kits), which gives access to ten individual 100-mil sockets.

Connection to a 20-pin 100-mil header

Use the adapter board (included in some kits) to connect to targets with a 20-pin 100-mil header.

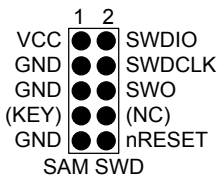
Table 5-3. Power Debugger JTAG Pin Description

Name	AVR port pin	SAM port pin	Description
TCK	1	4	Test Clock (clock signal from the Power Debugger into the target device).
TMS	5	2	Test Mode Select (control signal from the Power Debugger into the target device).
TDI	9	8	Test Data In (data transmitted from the Power Debugger into the target device).
TDO	3	6	Test Data Out (data transmitted from the target device into the Power Debugger).
nTRST	8	-	Test Reset (optional, only on some AVR devices). Used to reset the JTAG TAP controller.
nSRST	6	10	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the Power Debugger to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
VTG	4	1	Target voltage reference. The Power Debugger samples the target voltage on this pin in order to power the level converters correctly. The Power Debugger draws less than 3mA from this pin in debugWIRE mode and less than 1mA in other modes.
GND	2, 10	3, 5, 9	Ground. All must be connected to ensure that the Power Debugger and the target device share the same ground reference.

5.2.4. SWD Physical Interface

The ARM SWD interface is a subset of the JTAG interface, making use of TCK and TMS pins. The ARM JTAG and AVR JTAG connectors are, however, not pin-compatible, so when designing an application PCB, which uses a SAM device with SWD or JTAG interface, it is recommended to use the ARM pinout shown in the figure below. The SAM connector port on the Power Debugger can connect directly to this pinout.

Figure 5-4. Recommended ARM SWD/JTAG Header Pinout



The Power Debugger is capable of streaming UART-format ITM trace to the host computer. Trace is captured on the TRACE/SWO pin of the 10-pin header (JTAG TDO pin). Data is buffered internally on the Power Debugger and is sent over the HID interface to the host computer. The maximum reliable data rate is about 3MB/s.

5.2.5. Connecting to an SWD Target

The ARM SWD interface is a subset of the JTAG interface, making use of the TCK and TMS pins, which means that when connecting to an SWD device, the 10-pin JTAG connector can technically be used. The ARM JTAG and AVR JTAG connectors are, however, not pin-compatible, so this depends upon the layout of the target board in use. When using an STK600 or a board making use of the AVR JTAG pinout, the

AVR connector port on the Power Debugger must be used. When connecting to a board, which makes use of the ARM JTAG pinout, the SAM connector port on the Power Debugger must be used.

The recommended pinout for the 10-pin Cortex Debug connector is shown in Figure 5-4.

Connection to a 10-pin 50-mil Cortex header

Use the flat cable (included in some kits) to connect to a standard 50-mil Cortex header.

Connection to a 10-pin 100-mil Cortex-layout header

Use the adapter board (included in some kits) to connect to a 100-mil Cortex-pinout header.

Connection to a 20-pin 100-mil SAM header

Use the adapter board (included in some kits) to connect to a 20-pin 100-mil SAM header.

Connection to a custom 100-mil header

The 10-pin mini-squid cable should be used to connect between the Power Debugger AVR or SAM connector port and the target board. Six connections are required, as described in the table below.

Table 5-4. Power Debugger SWD Pin Mapping

Name	AVR port pin	SAM port pin	Description
SWDCLK	1	4	Serial Wire Debug Clock.
SWDIO	5	2	Serial Wire Debug Data Input/Output.
SWO	3	6	Serial Wire Output (optional- not implemented on all devices).
nSRST	6	10	Reset.
VTG	4	1	Target voltage reference.
GND	2, 10	3, 5, 9	Ground.

5.2.6. Special Considerations

ERASE pin

Some SAM devices include an ERASE pin which is asserted to perform a complete chip erase and unlock devices on which the security bit is set. This feature is coupled to the device itself as well as the flash controller and is not part of the ARM core.

The ERASE pin is NOT part of any debug header, and the Power Debugger is thus unable to assert this signal to unlock a device. In such cases the user should perform the erase manually before starting a debug session.

Physical interfaces

JTAG interface

The RESET line should always be connected so that the Power Debugger can enable the JTAG interface.

SWD interface

The RESET line should always be connected so that the Power Debugger can enable the SWD interface.

5.3. AVR UC3 Devices with JTAG/aWire

All AVR UC3 devices feature the JTAG interface for programming and debugging. In addition, some AVR UC3 devices feature the aWire interface with identical functionality using a single wire. Check the device datasheet for supported interfaces of that device.

5.3.1. Atmel AVR UC3 On-chip Debug System

The Atmel AVR UC3 OCD system is designed in accordance with the Nexus 2.0 standard (IEEE-ISTO 5001™-2003), which is a highly flexible and powerful open on-chip debug standard for 32-bit microcontrollers. It supports the following features:

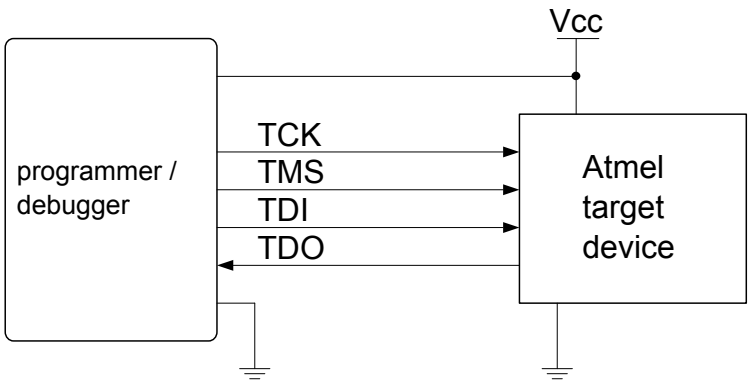
- Nexus compliant debug solution
- OCD supports any CPU speed
- Six program counter hardware breakpoints
- Two data breakpoints
- Breakpoints can be configured as watchpoints
- Hardware breakpoints can be combined to give break on ranges
- Unlimited number of user program breakpoints (using BREAK)
- Real-time program counter branch tracing, data trace, process trace (supported only by debuggers with parallel trace capture port)

For more information regarding the AVR UC3 OCD system, consult the AVR32UC Technical Reference Manuals, located on www.atmel.com/uc3.

5.3.2. JTAG Physical Interface

The JTAG interface consists of a 4-wire Test Access Port (TAP) controller that is compliant with the IEEE® 1149.1 standard. The IEEE standard was developed to provide an industry-standard way to efficiently test circuit board connectivity (Boundary Scan). Atmel AVR and SAM devices have extended this functionality to include full Programming and On-chip Debugging support.

Figure 5-5. JTAG Interface Basics



5.3.2.1. AVR JTAG Pinout

When designing an application PCB, which includes an Atmel AVR with the JTAG interface, it is recommended to use the pinout as shown in the figure below. Both 100-mil and 50-mil variants of this pinout are supported, depending on the cabling and adapters included with the particular kit.

Figure 5-6. AVR JTAG Header Pinout

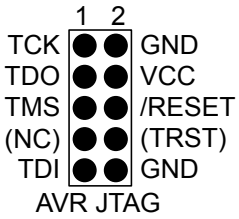


Table 5-5. AVR JTAG Pin Description

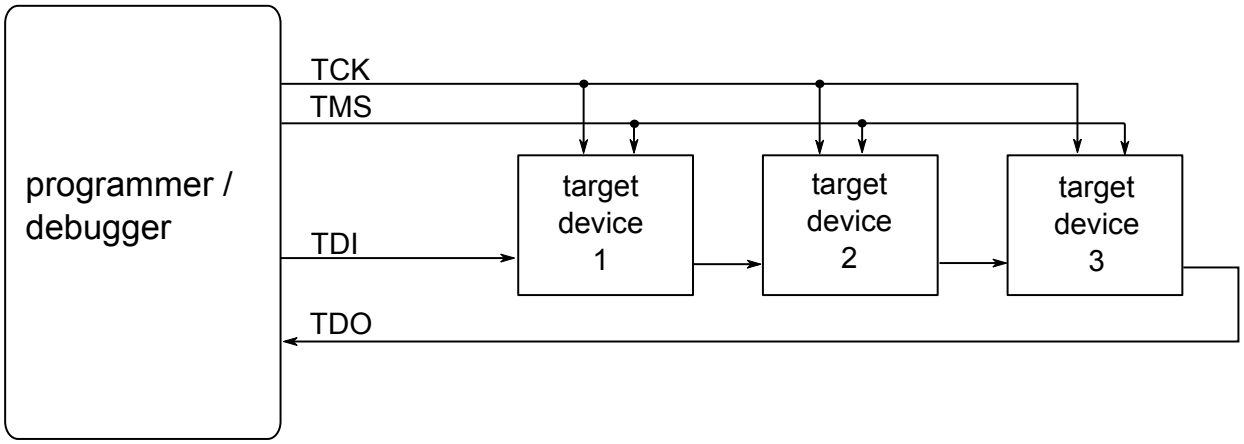
Name	Pin	Description
TCK	1	Test Clock (clock signal from the Power Debugger into the target device).
TMS	5	Test Mode Select (control signal from the Power Debugger into the target device).
TDI	9	Test Data In (data transmitted from the Power Debugger into the target device).
TDO	3	Test Data Out (data transmitted from the target device into the Power Debugger).
nTRST	8	Test Reset (optional, only on some AVR devices). Used to reset the JTAG TAP controller.
nSRST	6	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the Power Debugger to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
VTG	4	Target voltage reference. The Power Debugger samples the target voltage on this pin in order to power the level converters correctly. The Power Debugger draws less than 3mA from this pin in debugWIRE mode and less than 1mA in other modes.
GND	2, 10	Ground. Both must be connected to ensure that the Power Debugger and the target device share the same ground reference.

Tip: Remember to include a decoupling capacitor between pin 4 and GND.

5.3.2.2. JTAG Daisy Chaining

The JTAG interface allows for several devices to be connected to a single interface in a daisy chain configuration. The target devices must all be powered by the same supply voltage, share a common ground node, and must be connected as shown in the figure below.

Figure 5-7. JTAG Daisy Chain



When connecting devices in a daisy chain, the following points must be considered:

- All devices must share a common ground, connected to GND on the Power Debugger probe
- All devices must be operating on the same target voltage. VTG on the Power Debugger must be connected to this voltage.
- TMS and TCK are connected in parallel; TDI and TDO are connected in a serial chain.
- nSRST on the Power Debugger probe must be connected to RESET on the devices if any of the devices in the chain disables its JTAG port
- "Devices before" refers to the number of JTAG devices that the TDI signal has to pass through in the daisy chain before reaching the target device. Similarly "devices after" is the number of devices that the signal has to pass through after the target device before reaching the Power Debugger TDO pin.
- "Instruction bits "before" and "after" refers to the total sum of all JTAG devices' instruction register lengths, which are connected before and after the target device in the daisy chain
- The total IR length (instruction bits before + Atmel target device IR length + instruction bits after) is limited to a maximum of 256 bits. The number of devices in the chain is limited to 15 before and 15 after.

Tip:
Daisy chaining example: TDI → ATmega1280 → ATxmega128A1 → ATUC3A0512 → TDO.

In order to connect to the Atmel AVR XMEGA® device, the daisy chain settings are:

- Devices before: 1
- Devices after: 1
- Instruction bits before: 4 (8-bit AVR devices have 4 IR bits)
- Instruction bits after: 5 (32-bit AVR devices have 5 IR bits)

Table 5-6. IR Lengths of Atmel MCUs

Device type	IR length
AVR 8-bit	4 bits
AVR 32-bit	5 bits
SAM	4 bits

5.3.3. Connecting to a JTAG Target

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 STK600 stack should use the AVR header.

In addition the Power Debugger also has an unpopulated 100-mil 10-pin JTAG connector in the AVR pinout on its PCB. This is wired directly to the 50-mil AVR header.

The recommended pinout for the 10-pin AVR JTAG connector is shown in Figure 5-6.

The recommended pinout for the 10-pin ARM Cortex Debug connector is shown in Figure 5-2.

Direct connection to a standard 10-pin 50-mil header

Use the 50-mil 10-pin flat cable (included in some kits) to connect directly to a board supporting this header type. Use the AVR connector port on the Power Debugger for headers with the AVR pinout, and the SAM connector port for headers complying with the ARM Cortex Debug header pinout.

The pinouts for both 10-pin connector ports are shown below.

Connection to a standard 10-pin 100-mil header

Use a standard 50-mil to 100-mil adapter to connect to 100-mil headers. An adapter board (included in some kits) can be used for this purpose, or alternatively the JTAGICE3 adapter can be used for AVR targets.

Important:
The JTAGICE3 100-mil adapter cannot be used with the SAM connector port, since pins 2 and 10 (AVR GND) on the adapter are connected.

Connection to a custom 100-mil header

If your target board does not have a compliant 10-pin JTAG header in 50- or 100-mil, you can map to a custom pinout using the 10-pin "mini-squid" cable (included in some kits), which gives access to ten individual 100-mil sockets.

Connection to a 20-pin 100-mil header

Use the adapter board (included in some kits) to connect to targets with a 20-pin 100-mil header.

Table 5-7. Power Debugger JTAG Pin Description

Name	AVR port pin	SAM port pin	Description
TCK	1	4	Test Clock (clock signal from the Power Debugger into the target device).
TMS	5	2	Test Mode Select (control signal from the Power Debugger into the target device).
TDI	9	8	Test Data In (data transmitted from the Power Debugger into the target device).

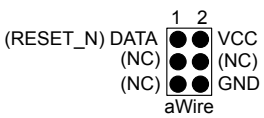
Name	AVR port pin	SAM port pin	Description
TDO	3	6	Test Data Out (data transmitted from the target device into the Power Debugger).
nTRST	8	-	Test Reset (optional, only on some AVR devices). Used to reset the JTAG TAP controller.
nSRST	6	10	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the Power Debugger to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
VTG	4	1	Target voltage reference. The Power Debugger samples the target voltage on this pin in order to power the level converters correctly. The Power Debugger draws less than 3mA from this pin in debugWIRE mode and less than 1mA in other modes.
GND	2, 10	3, 5, 9	Ground. All must be connected to ensure that the Power Debugger and the target device share the same ground reference.

5.3.4. aWire Physical Interface

The aWire interface makes use of the RESET wire of the AVR device to allow programming and debugging functions. A special enable sequence is transmitted by the Power Debugger, which disables the default RESET functionality of the pin.

When designing an application PCB, which includes an Atmel AVR with the aWire interface, it is recommended to use the pinout as shown in Figure 5-8. Both 100-mil and 50-mil variants of this pinout are supported, depending on the cabling and adapters included with the particular kit.

Figure 5-8. aWire Header Pinout



Tip:
Since aWire is a half-duplex interface, a pull-up resistor on the RESET line in the order of 47kΩ is recommended to avoid false start-bit detection when changing direction.

The aWire interface can be used as both a programming and debugging interface. All features of the OCD system available through the 10-pin JTAG interface can also be accessed using aWire.

5.3.5. Connecting to an aWire Target

The aWire interface requires only one data line in addition to V_{CC} and GND. On the target this line is the nRESET line, although the debugger uses the JTAG TDO line as the data line.

The recommended pinout for the 6-pin aWire connector is shown in Figure 5-8.

Connection to a 6-pin 100-mil aWire header

Use the 6-pin 100-mil tap on the flat cable (included in some kits) to connect to a standard 100-mil aWire header.

Connection to a 6-pin 50-mil aWire header

Use the adapter board (included in some kits) to connect to a standard 50-mil aWire header.

Connection to a custom 100-mil header

The 10-pin mini-squid cable should be used to connect between the Power Debugger AVR connector port and the target board. Three connections are required, as described in the table below.

Table 5-8. Power Debugger aWire Pin Mapping

Power Debugger AVR port pins	Target pins	Mini-squid pin	aWire pinout
Pin 1 (TCK)		1	
Pin 2 (GND)	GND	2	6
Pin 3 (TDO)	DATA	3	1
Pin 4 (VTG)	VTG	4	2
Pin 5 (TMS)		5	
Pin 6 (nSRST)		6	
Pin 7 (Not connected)		7	
Pin 8 (nTRST)		8	
Pin 9 (TDI)		9	
Pin 10 (GND)		0	

5.3.6. Special Considerations

JTAG interface

On some Atmel AVR UC3 devices the JTAG port is not enabled by default. When using these devices it is essential to connect the RESET line so that the Power Debugger can enable the JTAG interface.

aWire interface

The baud rate of aWire communications depends upon the frequency of the system clock, since data must be synchronized between these two domains. The Power Debugger will automatically detect that the system clock has been lowered, and re-calibrate its baud rate accordingly. The automatic calibration only works down to a system clock frequency of 8kHz. Switching to a lower system clock during a debug session may cause contact with the target to be lost.

If required, the aWire baud rate can be restricted by setting the aWire clock parameter. Automatic detection will still work, but a ceiling value will be imposed on the results.

Any stabilizing capacitor connected to the RESET pin must be disconnected when using aWire since it will interfere with correct operation of the interface. A weak external pullup (10kΩ or higher) on this line is recommended.

Shutdown sleep mode

Some AVR UC3 devices have an internal regulator that can be used in 3.3V supply mode with 1.8V regulated I/O lines. This means that the internal regulator powers both the core and most of the I/O. Only Atmel AVR ONE! debugger supports debugging while using sleep modes where this regulator is shut off.

5.3.7. EVTI / EVTO Usage

The EVTI and EVTO pins are not accessible on the Power Debugger. However, they can still be used in conjunction with other external equipment.

EVTI can be used for the following purposes:

- The target can be forced to stop execution in response to an external event. If the Event In Control (EIC) bits in the DC register are written to `0b01`, high-to-low transition on the EVTI pin will generate a breakpoint condition. EVTI must remain low for one CPU clock cycle to guarantee that a breakpoint is triggered. The External Breakpoint bit (EXB) in DS is set when this occurs.
- Generating trace synchronization messages. Not used by the Power Debugger.

EVTO can be used for the following purposes:

- Indicating that the CPU has entered debug mode. Setting the EOS bits in DC to `0b01` causes the EVTO pin to be pulled low for one CPU clock cycle when the target device enters debug mode. This signal can be used as a trigger source for an external oscilloscope.
- Indicating that the CPU has reached a breakpoint or watchpoint. By setting the EOC bit in a corresponding Breakpoint/Watchpoint Control Register, the breakpoint or watchpoint status is indicated on the EVTO pin. The EOS bits in DC must be set to 0xb10 to enable this feature. The EVTO pin can then be connected to an external oscilloscope in order to examine watchpoint timing.
- Generating trace timing signals. Not used by the Power Debugger.

5.4. tinyAVR, megaAVR, and XMEGA Devices

AVR devices feature various programming and debugging interfaces. Check the device datasheet for supported interfaces of that device.

- Some tinyAVR® devices have a TPI interface. TPI can be used for programming the device only, and these devices do not have on-chip debug capability at all.
- Some tinyAVR devices and some megaAVR devices have the debugWIRE interface, which connects to an on-chip debug system known as tinyOCD. All devices with debugWIRE also have the SPI interface for in-system programming.
- Some megaAVR devices have a JTAG interface for programming and debugging, with an on-chip debug system also known as megaOCD. All devices with JTAG also feature the SPI interface as an alternative interface for in-system programming.
- All AVR XMEGA devices have the PDI interface for programming and debugging. Some AVR XMEGA devices also have a JTAG interface with identical functionality.
- New tinyAVR devices have a UPDI interface, which is used for programming and debugging

Table 5-9. Programming and Debugging Interfaces Summary

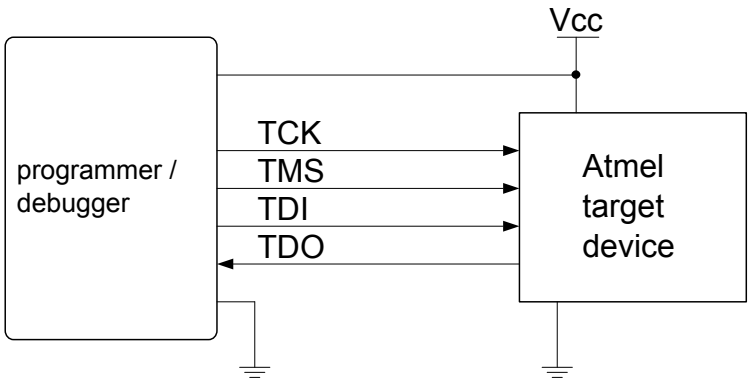
	UPDI	TPI	SPI	debugWIRE	JTAG	PDI	aWire	SWD
tinyAVR	New devices	Some devices	Some devices	Some devices				
megaAVR			All devices	Some devices	Some devices			
AVR XMEGA					Some devices	All devices		

	UPDI	TPI	SPI	debugWIRE	JTAG	PDI	aWire	SWD
AVR UC					All devices		Some devices	
SAM					Some devices			All devices

5.4.1. JTAG Physical Interface

The JTAG interface consists of a 4-wire Test Access Port (TAP) controller that is compliant with the IEEE® 1149.1 standard. The IEEE standard was developed to provide an industry-standard way to efficiently test circuit board connectivity (Boundary Scan). Atmel AVR and SAM devices have extended this functionality to include full Programming and On-chip Debugging support.

Figure 5-9. JTAG Interface Basics



5.4.2. Connecting to a JTAG Target

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 STK600 stack should use the AVR header.

In addition the Power Debugger also has an unpopulated 100-mil 10-pin JTAG connector in the AVR pinout on its PCB. This is wired directly to the 50-mil AVR header.

The recommended pinout for the 10-pin AVR JTAG connector is shown in Figure 5-6.

The recommended pinout for the 10-pin ARM Cortex Debug connector is shown in Figure 5-2.

Direct connection to a standard 10-pin 50-mil header

Use the 50-mil 10-pin flat cable (included in some kits) to connect directly to a board supporting this header type. Use the AVR connector port on the Power Debugger for headers with the AVR pinout, and the SAM connector port for headers complying with the ARM Cortex Debug header pinout.

The pinouts for both 10-pin connector ports are shown below.

Connection to a standard 10-pin 100-mil header

Use a standard 50-mil to 100-mil adapter to connect to 100-mil headers. An adapter board (included in some kits) can be used for this purpose, or alternatively the JTAGICE3 adapter can be used for AVR targets.

Important:
The JTAGICE3 100-mil adapter cannot be used with the SAM connector port, since pins 2 and 10 (AVR GND) on the adapter are connected.

Connection to a custom 100-mil header

If your target board does not have a compliant 10-pin JTAG header in 50- or 100-mil, you can map to a custom pinout using the 10-pin "mini-squid" cable (included in some kits), which gives access to ten individual 100-mil sockets.

Connection to a 20-pin 100-mil header

Use the adapter board (included in some kits) to connect to targets with a 20-pin 100-mil header.

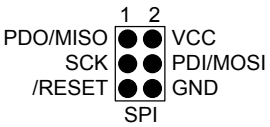
Table 5-10. Power Debugger JTAG Pin Description

Name	AVR port pin	SAM port pin	Description
TCK	1	4	Test Clock (clock signal from the Power Debugger into the target device).
TMS	5	2	Test Mode Select (control signal from the Power Debugger into the target device).
TDI	9	8	Test Data In (data transmitted from the Power Debugger into the target device).
TDO	3	6	Test Data Out (data transmitted from the target device into the Power Debugger).
nTRST	8	-	Test Reset (optional, only on some AVR devices). Used to reset the JTAG TAP controller.
nSRST	6	10	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the Power Debugger to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
VTG	4	1	Target voltage reference. The Power Debugger samples the target voltage on this pin in order to power the level converters correctly. The Power Debugger draws less than 3mA from this pin in debugWIRE mode and less than 1mA in other modes.
GND	2, 10	3, 5, 9	Ground. All must be connected to ensure that the Power Debugger and the target device share the same ground reference.

5.4.3. SPI Physical Interface

In-System Programming uses the target Atmel AVR's internal SPI (Serial Peripheral Interface) to download code into the flash and EEPROM memories. It is not a debugging interface. When designing an application PCB, which includes an AVR with the SPI interface, the pinout as shown in the figure below should be used.

Figure 5-10. SPI Header Pinout



5.4.4. Connecting to an SPI Target

The recommended pinout for the 6-pin SPI connector is shown in Figure 5-10.

Connection to a 6-pin 100-mil SPI header

Use the 6-pin 100-mil tap on the flat cable (included in some kits) to connect to a standard 100-mil SPI header.

Connection to a 6-pin 50-mil SPI header

Use the adapter board (included in some kits) to connect to a standard 50-mil SPI header.

Connection to a custom 100-mil header

The 10-pin mini-squid cable should be used to connect between the Power Debugger AVR connector port and the target board. Six connections are required, as described in the table below.

Important:
The SPI interface is effectively disabled when the debugWIRE enable fuse (DWEN) is programmed, even if SPIEN fuse is also programmed. To re-enable the SPI interface, the 'disable debugWIRE' command must be issued while in a debugWIRE debugging session. Disabling debugWIRE in this manner requires that the SPIEN fuse is already programmed. If Atmel Studio fails to disable debugWIRE, it is probable because the SPIEN fuse is NOT programmed. If this is the case, it is necessary to use a high-voltage programming interface to program the SPIEN fuse.

Info:
The SPI interface is often referred to as "ISP", since it was the first In System Programming interface on Atmel AVR products. Other interfaces are now available for In System Programming.

Table 5-11. Power Debugger SPI Pin Mapping

Power Debugger AVR port pins	Target pins	Mini-squid pin	SPI pinout
Pin 1 (TCK)	SCK	1	3
Pin 2 (GND)	GND	2	6
Pin 3 (TDO)	MISO	3	1
Pin 4 (VTG)	VTG	4	2
Pin 5 (TMS)		5	
Pin 6 (nSRST)	/RESET	6	5
Pin 7 (not connected)		7	

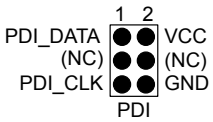
Power Debugger AVR port pins	Target pins	Mini-squid pin	SPI pinout
Pin 8 (nTRST)		8	
Pin 9 (TDI)	MOSI	9	4
Pin 10 (GND)		0	

5.4.5. PDI

The Program and Debug Interface (PDI) is an Atmel proprietary interface for external programming and on-chip debugging of a device. PDI Physical is a 2-pin interface providing a bi-directional half-duplex synchronous communication with the target device.

When designing an application PCB, which includes an Atmel AVR with the PDI interface, the pinout shown in the figure below should be used. One of the 6-pin adapters provided with the Power Debugger kit can then be used to connect the Power Debugger probe to the application PCB.

Figure 5-11. PDI Header Pinout



5.4.6. Connecting to a PDI Target

The recommended pinout for the 6-pin PDI connector is shown in Figure 5-11.

Connection to a 6-pin 100-mil PDI header

Use the 6-pin 100-mil tap on the flat cable (included in some kits) to connect to a standard 100-mil PDI header.

Connection to a 6-pin 50-mil PDI header

Use the adapter board (included in some kits) to connect to a standard 50-mil PDI header.

Connection to a custom 100-mil header

The 10-pin mini-squid cable should be used to connect between the Power Debugger AVR connector port and the target board. Four connections are required, as described in the table below.

Important:
The pinout required is different from the JTAGICE mkII JTAG probe, where PDI_DATA is connected to pin 9. The Power Debugger is compatible with the pinout used by the Atmel-ICE, JTAGICE3, AVR ONE!, and AVR Dragon™ products.

Table 5-12. Power Debugger PDI Pin Mapping

Power Debugger AVR port pin	Target pins	Mini-squid pin	Atmel STK600 PDI pinout
Pin 1 (TCK)		1	
Pin 2 (GND)	GND	2	6
Pin 3 (TDO)	PDI_DATA	3	1
Pin 4 (VTG)	VTG	4	2

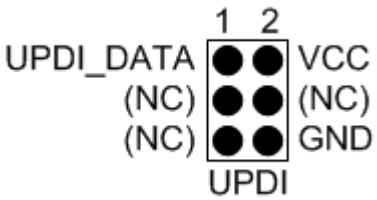
Power Debugger AVR port pin	Target pins	Mini-squid pin	Atmel STK600 PDI pinout
Pin 5 (TMS)		5	
Pin 6 (nSRST)	PDI_CLK	6	5
Pin 7 (not connected)		7	
Pin 8 (nTRST)		8	
Pin 9 (TDI)		9	
Pin 10 (GND)		0	

5.4.7. UPDI Physical Interface

The Unified Program and Debug Interface (UPDI) is an Atmel proprietary interface for external programming and on-chip debugging of a device. It is a successor to the PDI 2-wire physical interface, which is found on all AVR XMEGA devices. UPDI is a single-wire interface providing a bi-directional half-duplex asynchronous communication with the target device for purposes of programming and debugging.

When designing an application PCB, which includes an Atmel AVR with the UPDI interface, the pinout shown below should be used. One of the 6-pin adapters provided with the Power Debugger kit can then be used to connect the Power Debugger probe to the application PCB.

Figure 5-12. UPDI Header Pinout



5.4.7.1. UPDI and /RESET

The UPDI one-wire interface can be a dedicated pin or a shared pin, depending on the target AVR device. Consult the device datasheet for further information.

When the UPDI interface is on a shared pin, the pin can be configured to be either UPDI, /RESET, or GPIO by setting the RSTPINCFG[1:0] fuses.

The RSTPINCFG[1:0] fuses have the following configurations, as described in the datasheet. The practical implications of each choice are given here.

Table 5-13. RSTPINCFG[1:0] Fuse Configuration

RSTPINCFG[1:0]	Configuration	Usage
00	GPIO	General purpose I/O pin. In order to access UPDI, a 12V pulse must be applied to this pin. No external reset source is available.
01	UPDI	Dedicated programming and debugging pin. No external reset source is available.
10	Reset	Reset signal input. In order to access UPDI, a 12V pulse must be applied to this pin.
11	Reserved	NA

Note: Older AVR devices have a programming interface, known as "High-Voltage Programming" (both serial and parallel variants exist.) In general this interface requires 12V to be applied to the /RESET pin for the duration of the programming session. The UPDI interface is an entirely different interface. The UPDI pin is primarily a programming and debugging pin, which can be fused to have an alternative function (/RESET or GPIO). If the alternative function is selected then a 12V pulse is required on that pin in order to re-activate the UPDI functionality.

Note: If a design requires the sharing of the UPDI signal due to pin constraints, steps must be taken in order to ensure that the device can be programmed. To ensure that the UPDI signal can function correctly, as well as to avoid damage to external components from the 12V pulse, it is recommended to disconnect any components on this pin when attempting to debug or program the device. This can be done using a 0Ω resistor, which is mounted by default and removed or replaced by a pin header while debugging. This configuration effectively means that programming should be done before mounting the device.

5.4.8. Connecting to a UPDI Target

The recommended pinout for the 6-pin UPDI connector is shown in Figure 5-12.

Connection to a 6-pin 100-mil UPDI header

Use the 6-pin 100-mil tap on the flat cable (included in some kits) to connect to a standard 100-mil UPDI header.

Connection to a 6-pin 50-mil UPDI header

Use the adapter board (included in some kits) to connect to a standard 50-mil UPDI header.

Connection to a custom 100-mil header

The 10-pin mini-squid cable should be used to connect between the Power Debugger AVR connector port and the target board. Three connections are required, as described in the table below.

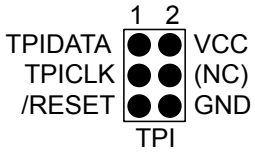
Table 5-14. Power Debugger UPDI Pin Mapping

Power Debugger AVR port pin	Target pins	Mini-squid pin	Atmel STK600 UPDI pinout
Pin 1 (TCK)		1	
Pin 2 (GND)	GND	2	6
Pin 3 (TDO)	UPDI_DATA	3	1
Pin 4 (VTG)	VTG	4	2
Pin 5 (TMS)		5	
Pin 6 (nSRST)	[/RESET sense]	6	5
Pin 7 (Not connected)		7	
Pin 8 (nTRST)		8	
Pin 9 (TDI)		9	
Pin 10 (GND)		0	

5.4.9. TPI Physical Interface

TPI is a programming-only interface for some AVR ATtiny devices. It is not a debugging interface, and these devices do not have OCD capability. When designing an application PCB which includes an AVR with the TPI interface, the pinout shown in the figure below should be used.

Figure 5-13. TPI Header Pinout



5.4.10. Connecting to a TPI Target

The recommended pinout for the 6-pin TPI connector is shown in Figure 5-13.

Connection to a 6-pin 100-mil TPI header

Use the 6-pin 100-mil tap on the flat cable (included in some kits) to connect to a standard 100-mil TPI header.

Connection to a 6-pin 50-mil TPI header

Use the adapter board (included in some kits) to connect to a standard 50-mil TPI header.

Connection to a custom 100-mil header

The 10-pin mini-squid cable should be used to connect between the Power Debugger AVR connector port and the target board. Six connections are required, as described in the table below.

Table 5-15. Power Debugger TPI Pin Mapping

Power Debugger AVR port pins	Target pins	Mini-squid pin	TPI pinout
Pin 1 (TCK)	CLOCK	1	3
Pin 2 (GND)	GND	2	6
Pin 3 (TDO)	DATA	3	1
Pin 4 (VTG)	VTG	4	2
Pin 5 (TMS)		5	
Pin 6 (nSRST)	/RESET	6	5
Pin 7 (not connected)		7	
Pin 8 (nTRST)		8	
Pin 9 (TDI)		9	
Pin 10 (GND)		0	

5.4.11. Advanced Debugging (AVR JTAG /debugWIRE devices)

I/O Peripherals

Most I/O peripherals will continue to run even though the program execution is stopped by a breakpoint. Example: If a breakpoint is reached during a UART transmission, the transmission will be completed and corresponding bits set. The TXC (transmit complete) flag will be set and be available on the next single step of the code even though it normally would happen later in an actual device.

All I/O modules will continue to run in stopped mode with the following two exceptions:

- Timer/Counters (configurable using the software front-end)
- Watchdog Timer (always stopped to prevent resets during debugging)

Single Stepping I/O access

Since the I/O continues to run in stopped mode, care should be taken to avoid certain timing issues. For example, the code:

```
OUT PORTB, 0xAA
IN TEMP, PINB
```

When running this code normally, the TEMP register would not read back 0xAA because the data would not yet have been latched physically to the pin by the time it is sampled by the IN operation. A NOP instruction must be placed between the OUT and the IN instruction to ensure that the correct value is present in the PIN register.

However, when single stepping this function through the OCD, this code will always give 0xAA in the PIN register since the I/O is running at full speed even when the core is stopped during the single stepping.

Single stepping and timing

Certain registers need to be read or written within a given number of cycles after enabling a control signal. Since the I/O clock and peripherals continue to run at full speed in stopped mode, single stepping through such code will not meet the timing requirements. Between two single steps, the I/O clock may have run millions of cycles. To successfully read or write registers with such timing requirements, the whole read or write sequence should be performed as an atomic operation running the device at full speed. This can be done by using a macro or a function call to execute the code, or use the run-to-cursor function in the debugging environment.

Accessing 16-bit registers

The Atmel AVR peripherals typically contain several 16-bit registers that can be accessed via the 8-bit data bus (e.g.: TCNTn of a 16-bit timer). The 16-bit register must be byte accessed using two read or write operations. Breaking in the middle of a 16-bit access or single stepping through this situation may result in erroneous values.

Restricted I/O register access

Certain registers cannot be read without affecting their contents. Such registers include those which contain flags which are cleared by reading, or buffered data registers (e.g.: UDR). The software front-end will prevent reading these registers when in stopped mode to preserve the intended non-intrusive nature of OCD debugging. In addition, some registers cannot safely be written without side-effects occurring - these registers are read-only. For example:

- Flag registers, where a flag is cleared by writing '1' to any bit. These registers are read-only.
- UDR and SPDR registers cannot be read without affecting the state of the module. These registers are not accessible.

5.4.12. megaAVR Special Considerations

Software breakpoints

Since it contains an early version of the OCD module, ATmega128[A] does not support the use of the BREAK instruction for software breakpoints.

JTAG clock

The target clock frequency must be accurately specified in the software front-end before starting a debug session. For synchronization reasons, the JTAG TCK signal must be less than one fourth of the target clock frequency for reliable debugging. When programming via the JTAG interface, the TCK frequency is

limited by the maximum frequency rating of the target device, and not the actual clock frequency being used.

When using the internal RC oscillator, be aware that the frequency may vary from device to device and is affected by temperature and V_{CC} changes. Be conservative when specifying the target clock frequency.

JTAGEN and OCDEN fuses

The JTAG interface is enabled using the JTAGEN fuse, which is programmed by default. This allows access to the JTAG programming interface. Through this mechanism, the OCDEN fuse can be programmed (by default OCDEN is un-programmed). This allows access to the OCD in order to facilitate debugging the device. The software front-end will always ensure that the OCDEN fuse is left un-programmed when terminating a session, thereby restricting unnecessary power consumption by the OCD module. If the JTAGEN fuse is unintentionally disabled, it can only be re-enabled using SPI or High Voltage programming methods.

If the JTAGEN fuse is programmed, the JTAG interface can still be disabled in firmware by setting the JTD bit. This will render code un-debuggable, and should not be done when attempting a debug session. If such code is already executing on the Atmel AVR device when starting a debug session, the Power Debugger will assert the RESET line while connecting. If this line is wired correctly, it will force the target AVR device into reset, thereby allowing a JTAG connection.

If the JTAG interface is enabled, the JTAG pins cannot be used for alternative pin functions. They will remain dedicated JTAG pins until either the JTAG interface is disabled by setting the JTD bit from the program code, or by clearing the JTAGEN fuse through a programming interface.

Tip:
Be sure to check the "use external reset" checkbox in both the programming dialog and debug options dialog in order to allow the Power Debugger to assert the RESET line and re-enable the JTAG interface on devices which are running code which disables the JTAG interface by setting the JTD bit.

IDR/OCDR events

The IDR (In-out Data Register) is also known as the OCDR (On Chip Debug Register), and is used extensively by the debugger to read and write information to the MCU when in stopped mode during a debug session. When the application program in run mode writes a byte of data to the OCDR register of the AVR device being debugged, the Power Debugger reads this value out and displays it in the message window of the software front-end. The OCDR register is polled every 50ms, so writing to it at a higher frequency will NOT yield reliable results. When the AVR device loses power while it is being debugged, spurious OCDR events may be reported. This happens because the Power Debugger may still poll the device as the target voltage drops below the AVR's minimum operating voltage.

5.4.13. AVR XMEGA Special Considerations

OCD and clocking

When the MCU enters stopped mode, the OCD clock is used as MCU clock. The OCD clock is either the JTAG TCK if the JTAG interface is being used, or the PDI_CLK if the PDI interface is being used.

I/O modules in stopped mode
In contrast to earlier Atmel megaAVR devices, in XMEGA the I/O modules are stopped in stop mode. This means that USART transmissions will be interrupted, timers (and PWM) will be stopped.

Hardware breakpoints
There are four hardware breakpoint comparators - two address comparators and two value comparators. They have certain restrictions:

- All breakpoints must be of the same type (program or data)
- All data breakpoints must be in the same memory area (I/O, SRAM, or XRAM)
- There can only be one breakpoint if address range is used

Here are the different combinations that can be set:

- Two single data or program address breakpoints
- One data or program address range breakpoint
- Two single data address breakpoints with single value compare
- One data breakpoint with address range, value range, or both

Atmel Studio will tell you if the breakpoint can't be set, and why. Data breakpoints have priority over program breakpoints, if software breakpoints are available.

External reset and PDI physical
The PDI physical interface uses the reset line as clock. While debugging, the reset pullup should be 10k or more or be removed. Any reset capacitors should be removed. Other external reset sources should be disconnected.

Debugging with sleep for ATxmegaA1 rev H and earlier

A bug existed on early versions of ATxmegaA1 devices that prevented the OCD from being enabled while the device was in certain sleep modes. There are two workarounds to re-enable OCD:

- Go into the Power Debugger. Options in the Tools menu and enable "Always activate external reset when reprogramming device".
- Perform a chip erase

The sleep modes that trigger this bug are:

- Power-down
- Power-save
- Standby
- Extended standby

5.4.14. debugWIRE Special Considerations

The debugWIRE communication pin (dW) is physically located on the same pin as the external reset (RESET). An external reset source is therefore not supported when the debugWIRE interface is enabled.

The debugWIRE Enable fuse (DWEN) must be set on the target device in order for the debugWIRE interface to function. This fuse is by default un-programmed when the Atmel AVR device is shipped from the factory. The debugWIRE interface itself cannot be used to set this fuse. In order to set the DWEN fuse, the SPI mode must be used. The software front-end handles this automatically provided that the necessary SPI pins are connected. It can also be set using SPI programming from the Atmel Studio programming dialog.

Either: Attempt to start a debug session on the debugWIRE part. If the debugWIRE interface is not enabled, Atmel Studio will offer to retry, or attempt to enable debugWIRE using SPI programming. If you have the full SPI header connected, debugWIRE will be enabled, and you will be asked to toggle power on the target. This is required for the fuse changes to be effective.

Or: Open the programming dialog in SPI mode, and verify that the signature matches the correct device. Check the DWEN fuse to enable debugWIRE.

Important:
It is important to leave the SPIEN fuse programmed, the RSTDISBL fuse un-programmed! Not doing this will render the device stuck in debugWIRE mode, and High Voltage programming will be required to revert the DWEN setting.

To disable the debugWIRE interface, use High Voltage programming to un-program the DWEN fuse. Alternately, use the debugWIRE interface itself to temporarily disable itself, which will allow SPI programming to take place, provided that the SPIEN fuse is set.

Important:
If the SPIEN fuse was NOT left programmed, Atmel Studio will not be able to complete this operation, and High Voltage programming must be used.

During a debug session, select the 'Disable debugWIRE and Close' menu option from the 'Debug' menu. DebugWIRE will be temporarily disabled, and Atmel Studio will use SPI programming to un-program the DWEN fuse.

Having the DWEN fuse programmed enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption of the AVR while in sleep modes. The DWEN Fuse should therefore always be disabled when debugWIRE is not used.

When designing a target application PCB where debugWIRE will be used, the following considerations must be made for correct operation:

- Pull-up resistors on the dW/(RESET) line must not be smaller (stronger) than 10kΩ. The pull-up resistor is not required for debugWIRE functionality, since the debugger tool provides this.
- Any stabilizing capacitor connected to the RESET pin must be disconnected when using debugWIRE, since they will interfere with correct operation of the interface
- All external reset sources or other active drivers on the RESET line must be disconnected, since they may interfere with the correct operation of the interface

Never program the lock-bits on the target device. The debugWIRE interface requires that lock-bits are cleared in order to function correctly.

5.4.15. debugWIRE Software Breakpoints

The debugWIRE OCD is drastically scaled down when compared to the Atmel megaAVR (JTAG) OCD. This means that it does not have any program counter breakpoint comparators available to the user for debugging purposes. One such comparator does exist for purposes of run-to-cursor and single-stepping operations, but additional user breakpoints are not supported in hardware.

Instead, the debugger must make use of the AVR BREAK instruction. This instruction can be placed in FLASH, and when it is loaded for execution it will cause the AVR CPU to enter stopped mode. To support breakpoints during debugging, the debugger must insert a BREAK instruction into FLASH at the point at which the users requests a breakpoint. The original instruction must be cached for later replacement. When single stepping over a BREAK instruction, the debugger has to execute the original cached instruction in order to preserve program behavior. In extreme cases, the BREAK has to be removed from FLASH and replaced later. All these scenarios can cause apparent delays when single stepping from breakpoints, which will be exacerbated when the target clock frequency is very low.

It is thus recommended to observe the following guidelines, where possible:

- Always run the target at as high a frequency as possible during debugging. The debugWIRE physical interface is clocked from the target clock.
- Try to minimize on the number of breakpoint additions and removals, as each one require a FLASH page to be replaced on the target
- Try to add or remove a small number of breakpoints at a time, to minimize the number of FLASH page write operations
- If possible, avoid placing breakpoints on double-word instructions

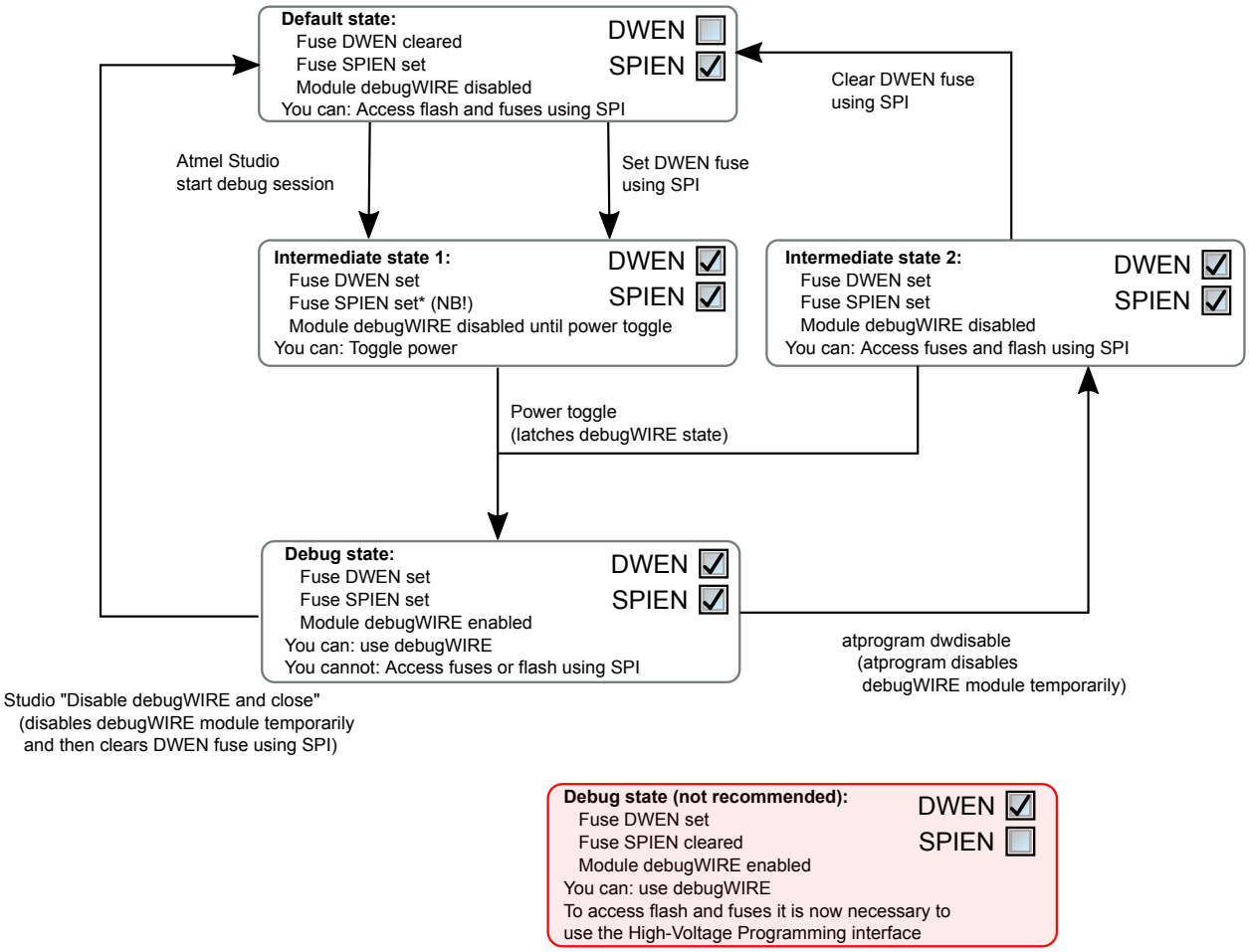
5.4.16. Understanding debugWIRE and the DWEN Fuse

When enabled, the debugWIRE interface takes control of the device's /RESET pin, which makes it mutually exclusive to the SPI interface, which also needs this pin. When enabling and disabling the debugWIRE module, follow one of these two approaches:

- Let Atmel Studio take care of things (recommended)
- Set and clear DWEN manually (exercise caution, advanced users only!)

Important: When manipulating DWEN manually, it is important that the SPIEN fuse remains set to avoid having to use High-Voltage programming.

Figure 5-14. Understanding debugWIRE and the DWEN Fuse



5.4.17. TinyX-OCD (UPDI) Special Considerations

The UPDI data pin (UPDI_DATA) can be a dedicated pin or a shared pin, depending on the target AVR device. A shared UPDI pin is 12V tolerant, and can be configured to be used as /RESET or GPIO. For further details on how to use the pin in these configurations, see UPDI Physical Interface.

On devices which include the CRCSCAN module (Cyclic Redundancy Check Memory Scan) this module should not be used in continuous background mode while debugging. The OCD module has limited hardware breakpoint comparator resources, so BREAK instructions may be inserted into flash (software breakpoints) when more breakpoints are required, or even during source-level code stepping. The CRC module could incorrectly detect this breakpoint as a corruption of flash memory contents.

The CRCSCAN module can also be configured to perform a CRC scan before boot. In the case of a CRC mismatch, the device will not boot, and appear to be in a locked state. The only way to recover the device from this state is to perform a full chip erase and either program a valid flash image or disable the pre-boot CRCSCAN. (A simple chip erase will result in a blank flash with invalid CRC, and the part will thus still not boot.) Atmel Studio will automatically disable the CRCSCAN fuses when chip erasing a device in this state.

When designing a target application PCB where UPDI interface will be used, the following considerations must be made for correct operation:

- Pull-up resistors on the UPDI line must not be smaller (stronger) than 10kΩ. A pull-down resistor should not be used, or it should be removed when using UPDI. The UPDI physical is push-pull

capable, so only a weak pull-up resistor is required to prevent false start bit triggering when the line is idle.

- If the UPDI pin is to be used as a RESET pin, any stabilizing capacitor must be disconnected when using UPDI, since it will interfere with correct operation of the interface
- If the UPDI pin is used as RESET or GPIO pin, all external drivers on the line must be disconnected during programming or debugging since they may interfere with the correct operation of the interface

6. Hardware Description

6.1. Overview

The Atmel Power Debugger hardware consists of three main sections; the debugger, analog frontend, and data gateway. The logical construction of the tool is shown here.

Figure 6-1. Logical Construction of the Power Debugger

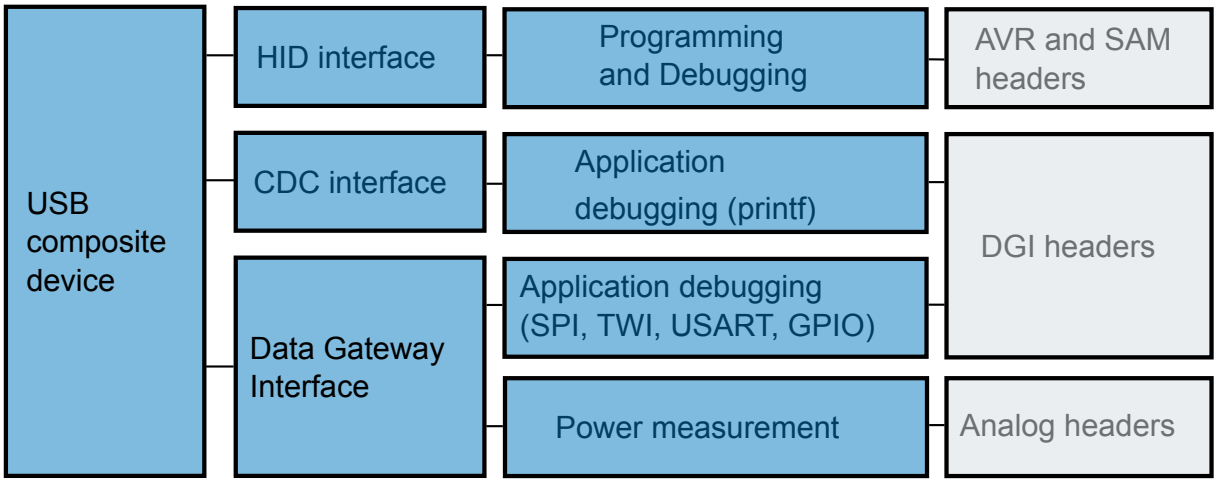
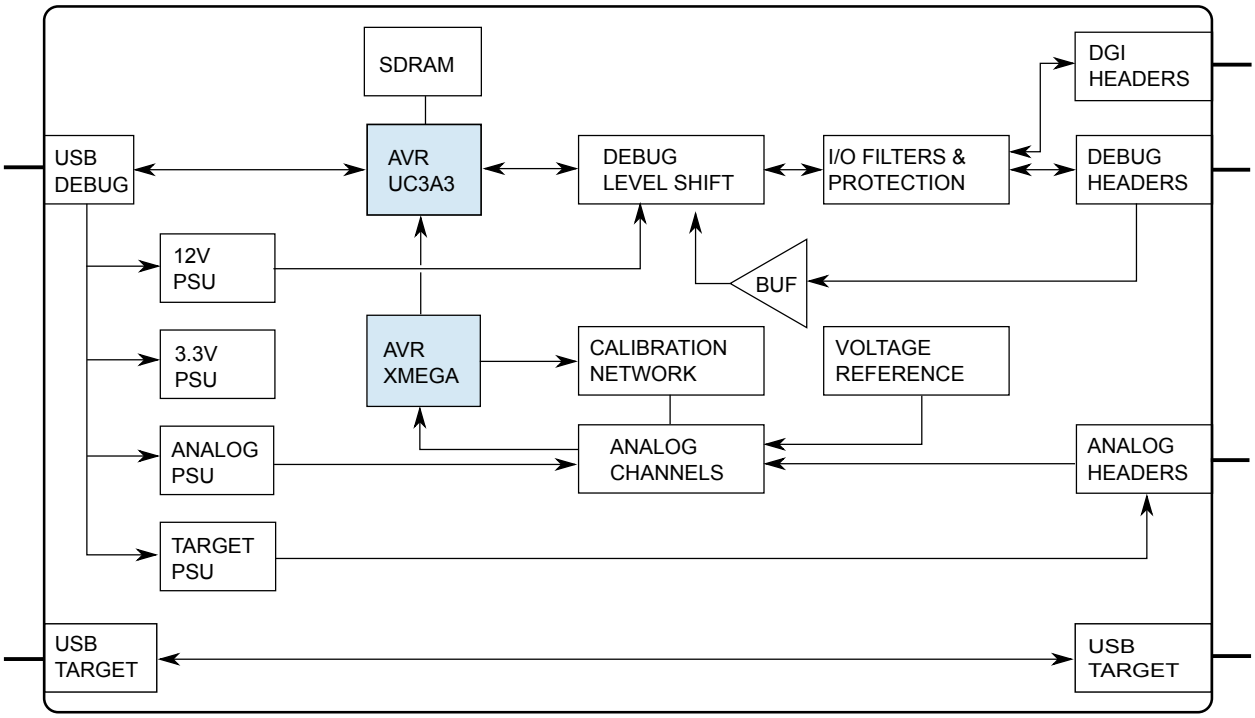


Figure 6-2. Hardware Block Diagram



Power is supplied to the Power Debugger from the USB bus connected to the DEBUG port, regulated to 3.3V by a step-down switch-mode regulator. The VTG pins are used as a reference input only, and a separate power supply feeds the variable-voltage side of the on-board level converters. At the heart of the Power Debugger main board is the Atmel AVR UC3 microcontroller AT32UC3A3256, which runs at up to

84MHz depending on the tasks being processed. The microcontroller includes an on-chip USB 2.0 high-speed module, allowing high data throughput to and from the tool.

6.2. Programming and Debugging

The debugger section of the Power Debugger closely resembles the Atmel-ICE hardware. At the heart of the debugger is the Atmel AVR UC3 microcontroller, which implements a USB HID interface for its programming and debugging API.

Communication between the Power Debugger and the programming and debugging interface on the target device is done through a bank of level converters that shift signals between the target's operating voltage and the internal voltage level on the Power Debugger. Also in the signal path are PTC fuses, zener over-voltage protection diodes to ground, series termination resistors, inductive filters, and ESD protection diodes. All signal channels can be operated in the range 1.62 to 5.5V, although the Power Debugger hardware itself can not drive out a higher voltage than 5.0V. Maximum operating frequency varies according to the target interface in use.

The Power Debugger includes three electrically connected output ports for convenience. The two 50-mil horizontal headers are for use with AVR 10-pin pinout and the ARM Cortex debug headers respectively. These ports are identical to those found on the Atmel-ICE. In addition the Power Debugger includes an unpopulated 100-mil header in the AVR pinout for custom connections.

Important: Soldering onto this header is done at the user's risk, and ESD precautions must be taken.

6.3. Analog Hardware

The Power Debugger analog front-end contains two channels, referred to as the 'A' channel and the 'B' channel. Although they operate on similar mechanisms, the two channels are not symmetrical and should be used for different purposes.

Both channels are fed into independent ADC channels on the AVR XMEGA128A1U microcontroller. This ADC module allows for the 'A' channel to operate totally independently at maximum sampling rate while the 'B' channel ADC is shared with both channels' voltage measurement, which are sampled less frequently.

The 'A' channel is the recommended channel for accurately measuring low currents. It features two shunt stages which through bypass switches provide a circuit capable of measuring from 100mA on the top-end down to under 1µA. Range switching is done automatically, and it is possible to lock sampling into the high-range only should this be needed.

1. 'A' channel high range: 100mA - 500µA, ~3µA resolution.
2. 'A' channel low-range: 1mA - 1µA, ~30nA resolution.

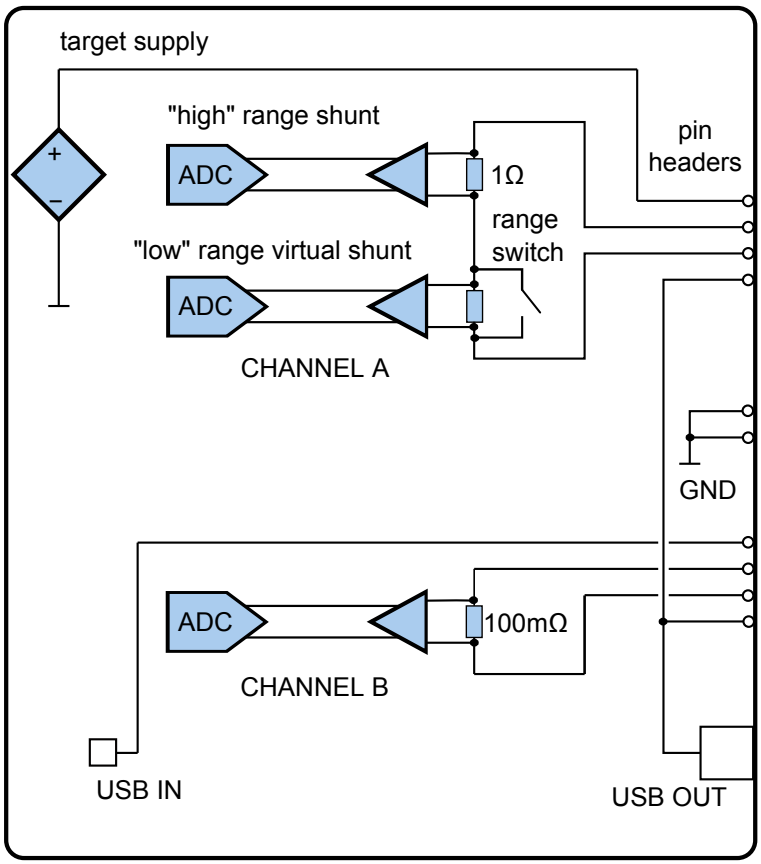
The sampling rate of the 'A' channel is 62.5kHz and data is sent in 16-bit frames to the host computer. A calibrated 'A' channel has accuracy no worse than 3% down to around 1µA.

The 'B' channel is the recommended channel for measuring higher currents with lower resolution than the 'A' channel. It is based on a single shunt resistor allowing measurement of current up to 1A and down to 1mA.

1. 'B' channel single range: 1A, ~500µA resolution.

Data is sampled at 62.5kHz and transferred to the computer in 12-bit frames.

Figure 6-3. Analog Block Diagram



6.3.1. Analog Hardware Calibration

Both Power Debugger analog channels requires calibration to achieve full measurement accuracy. The hardware is calibrated during manufacturing but it is also possible to do a re-calibration to get the most accurate measurements. The calibration procedure itself is automatic but before the calibration procedure can be started it is necessary to disconnect all external wires or jumpers from the analog channel pins and the voltage supply pin (the pins next to the ammeter symbols, and the pin next to the voltage source symbol on the Power Debugger). The calibration is triggered through the software front end used to interface the Power Debugger.

6.4. Target Voltage Supply (VOUT)

The Power Debugger has an on-board voltage supply capable of providing up to 100mA in the range 1.6V to 5.5V from the USB DEBUG connector.

Tip: The voltage supply will immediately switch off if an over-current is detected on the A channel. This will protect the target circuit *if* the supply is connected through the A channel.

6.5. Data Gateway Interface

The Power Debugger includes a full Data Gateway Interface (DGI). Functionality is identical to the one found on Atmel Xplained Pro kits powered by the Atmel EDBG device.

The Data Gateway Interface is an interface for streaming data from the target device to a computer. This is meant as an aid in application debugging as well as for demonstration of features in the application running on the target device.

DGI consists of multiple channels for data streaming. The Power Debugger supports the following modes:

- USART
- SPI
- TWI
- GPIO[0..3]

All signals on the DGI header are level shifted to operate on the target device's power domain. Also, in the signal path are series PTC fuses, zener over-voltage protection diodes to ground, series termination resistors, inductive filters, and ESD protection diodes. All signal channels can be operated in the range 1.62 to 5.5V, although the Power Debugger hardware itself can not output a higher voltage than 5.0V.

Important: The arrows in the silk screen on the USART/SPI interface shows data directions. The arrow pointing to the left is a receiver channel to which the target device's TX line should be connected for USART usage and MOSI for SPI usage. Similarly, the arrow pointing to the right is a transmitter channel and should be connected to the target device's RX line for USART usage and MISO for SPI usage. Remember that for SPI usage the Power Debugger is the slave.

Important: The DGI uses the same reference voltage as the debugger section. This means that the REF pins of the DGI header and the debug headers are electrically connected.

Important: SPI and USART mode can not be used simultaneously.

Important: An I^2C write transaction (could be of length zero) must take place before an I^2C read transaction is acknowledged by the EDBG.

6.6. CDC Interface

The Power Debugger includes a standard CDC virtual COM port interface. The CDC header is part of the 20-pin DGI header, although logically it is a separate interface.

Important: The arrows on the silkscreen shows data directions. The arrow pointing to the left is a receiver channel to which the user connects the target's TX. Similarly the arrow pointing to the right is a transmitter channel to which the target device's RX is connected.

6.7. USB Connectors

The Power Debugger contains a USB "feed-through" section across its lower side. This is a convenience feature enabling a quick and easy connection to USB powered target boards.

The USB feed through section has two USB connectors:

- The left side "TARGET" connector is a USB Micro-B jack. Connect this to the host computer using the cable included in the kit.
- The right side "TARGET" connector is a USB Type A jack. Connect this to the board being debugged.

Figure 6-4. USB Feed Through

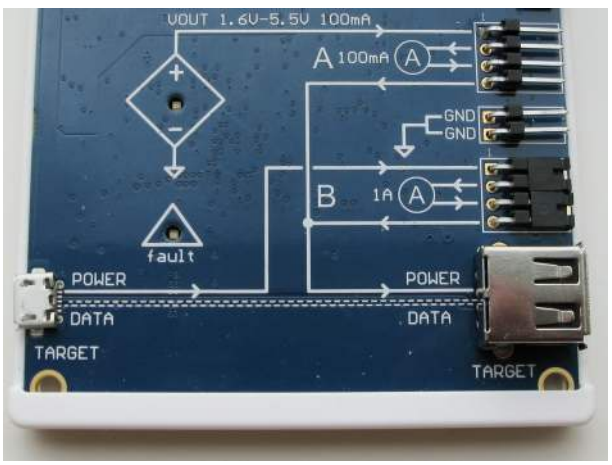
The routing of the USB feed-through section is clearly indicated by the silkscreen on the board:

- DATA lines are passed through untouched from the Micro-B jack to the A jack
- GND is connected to the board ground
- SHIELD is connected to board ground through a 1MΩ resistor
- POWER (VBUS) is routed from the Micro-B jack to pin 1 on the B channel header
- POWER (VBUS) is routed from pin 4 on both A and B channel headers to the A jack

The routing of the POWER line enables the user to easily connect up one of the following configurations:

- Routing USB power from the TARGET USB connector through the B channel for measurement. Mount both jumpers on the B channel header as shown here.

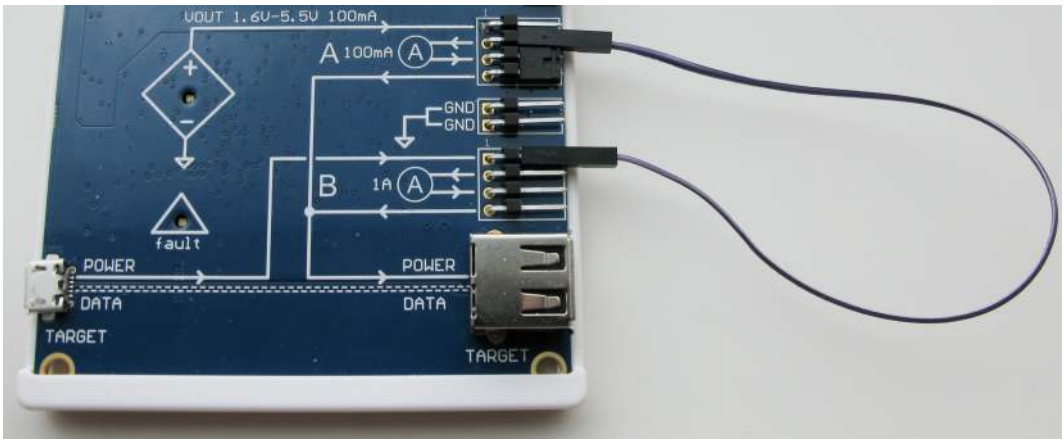
  Figure 6-5. Jumper Configuration



- Routing USB power from the TARGET USB connector through the A channel for measurement

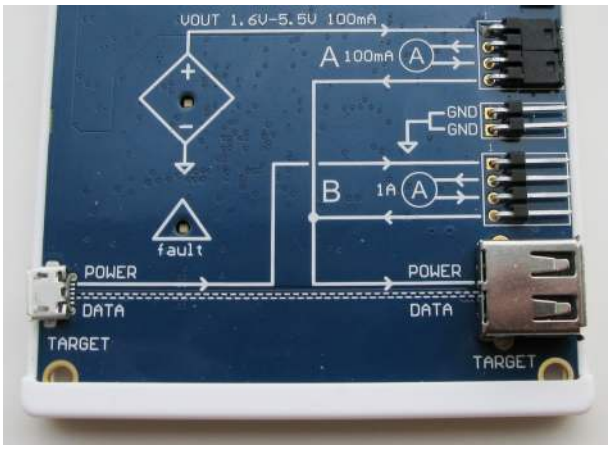
Use a single wire to connect POWER to the A channel and a jumper to connect the output as shown here.

Figure 6-6. Jumper Configuration

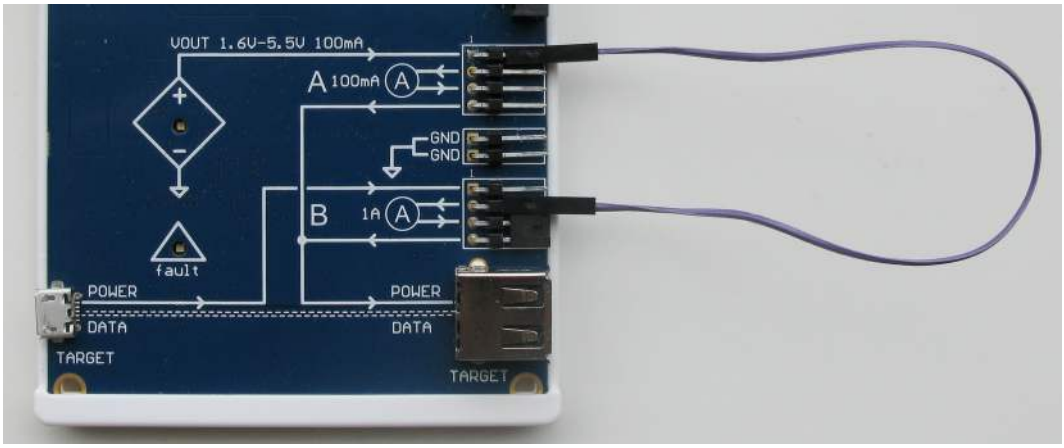


- Sourcing USB power from the DEBUG USB connector through the variable voltage source through the A channel
  Mount both jumpers on the A channel as shown here.

Figure 6-7. Jumper Configuration



- Sourcing USB power from the DEBUG USB connector through the variable voltage source through the B channel
  Use a single wire to connect POWER to the B channel and a jumper to connect the output as shown here.



6.8. LEDs

The Power Debugger has five LEDs, which indicate the status of the tool. The debug LEDs are identical in function to those on Atmel-ICE and previous debugger hardware. For those familiar with the Atmel-ICE debugger, the entire central section of the Power Debugger resembles the Atmel-ICE.

Figure 6-8. Debugger LEDs

Figure 6-9. VOUT LED, used to Indicate the Status of the Target Voltage Supply

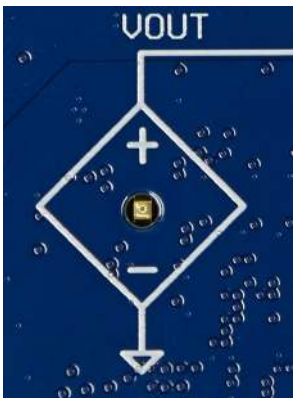


Figure 6-10. Fault LED, used to Indicate the Status of the Analog Sampling Circuitry

Table 6-1. LEDs

LED	Function	Description
TARGET	Target power	GREEN when target power is OK.
POWER	Main power	RED when main-board power is OK.
STATUS	Status	Flashing GREEN when the target is running/stepping. OFF when target is stopped.
VOUT	Target voltage	GREEN when target output voltage is ON. Flashing when an over-current is detected in the A channel.
fault	Analog fault	RED when a voltage or current outside recommended operating conditions is detected on the analog pins.

7. Product Compliance

7.1. RoHS and WEEE

The Power Debugger and all accessories are manufactured in accordance to both the RoHS Directive (2002/95/EC) and the WEEE Directive (2002/96/EC).

7.2. CE and FCC

The Power Debugger unit has been tested in accordance to the essential requirements and other relevant provisions of Directives:

- Directive 2004/108/EC (class B)
- FCC part 15 subpart B
- 2002/95/EC (RoHS, WEEE)

The following standards are used for evaluation:

- EN 61000-6-1 (2007)
- EN 61000-6-3 (2007) + A1(2011)
- FCC CFR 47 Part 15 (2013)

The Technical Construction File is located at:

```
Atmel Norway
Vestre Rosten 79
7075 Tiller
Norway
```

Every effort has been made to minimise electromagnetic emissions from this product. However, under certain conditions, the system (this product connected to a target application circuit) may emit individual electromagnetic component frequencies which exceed the maximum values allowed by the abovementioned standards. The frequency and magnitude of the emissions will be determined by several factors, including layout and routing of the target application with which the product is used.

8. Firmware Release History and Known Issues

8.1. Firmware Release History

Table 8-1. Public Firmware Revisions

Firmware version (decimal)	Date	Relevant changes
1.45	29.09.2016	Added support for UPDI interface (tinyX devices) Made USB endpoint size configurable General bug fixes
1.18	18.11.2015	Improved CDC functionality LED flasher fix
1.16	22.09.2015	First release of Power Debugger tool

8.2. Known Issues

- If a host or power source is connected to the TARGET Micro-B USB connector while no host is connected to the DEBUG USB connector then the Power Debugger will be partly powered through the TARGET Micro-B connection and the POWER LED will flash. It is always recommended to plug a host to the DEBUG USB connector before plugging anything to the TARGET Micro-B USB connector.
- If a voltage is applied to the EVT pin on the 20-pin DGI header while no host is connected to the DEBUG USB connector a current of up to 25mA might flow into the EVT pin. It is always recommended to plug a host to the DEBUG USB connector before applying any voltage to any of the Power Debugger pin header pins.
- Running current measurements in Data Visualizer while programming or debugging at low interface frequencies/baud rates might result in Data Visualizer disconnecting from the Power Debugger. The lower limit of the interface speed varies depending on target type, flash size, and interface type, but is typically in the range 100-300kHz. In general it is not recommended to measure current while debugging as the current consumption will be affected by the on-chip debug system on the target.

9. Revision History

Doc. Rev.	Date	Comments
42696D	10/2016	Added UPDI interface and updated Firmware Release History
42696C	03/2016	• Updated quick start guide • Added debug cable pinout • Reworked firmware revision history table • Several minor updates and corrections
42696B	03/2015	Internal version not published
42696A		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**