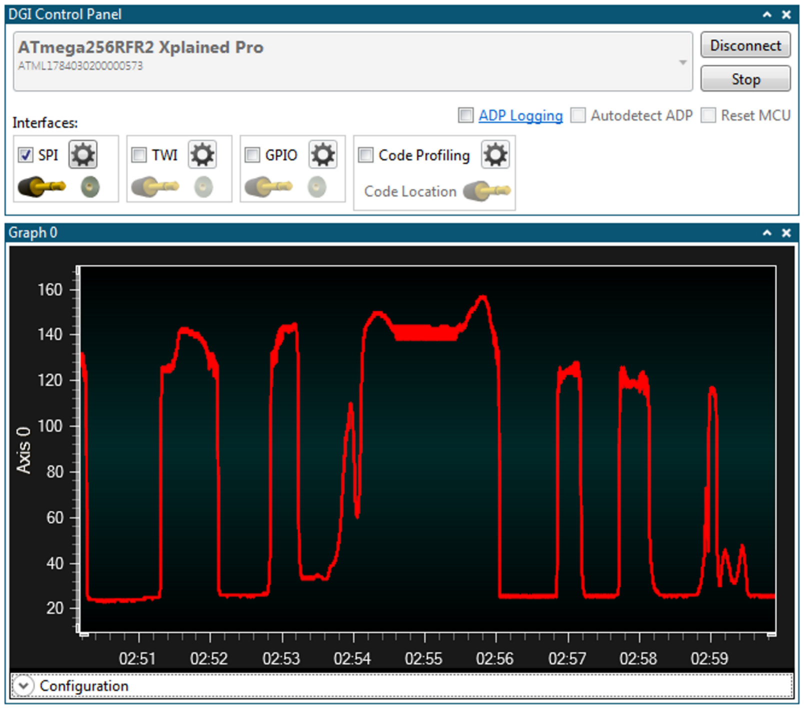
Data Visualizer Software User's Guide

Description



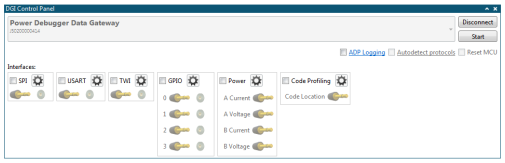
The Data Visualizer is a program to process and visualize data. The Data Visualizer is capable of receiving data from various sources such as the Embedded Debugger Data Gateway Interface (DGI) and COM ports.

Table of Contents

1. Overview

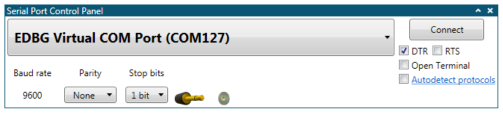
This chapter gives an overview of the main modules/features of the Data Visualizer. Each module is described in a separate chapter with technical details of the module, and includes an example or use case showing how to use the module. As each chapter is self-contained, it is possible for the user to quickly identify and select the chapter/module of interest.

Data Gateway Interface (DGI)



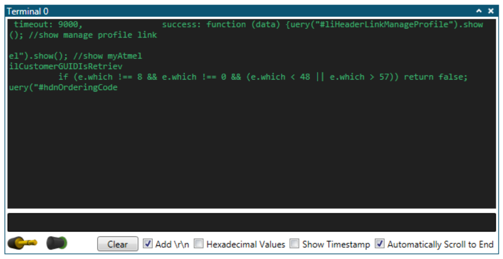
Data Gateway Interface (DGI) enables **bidirectional communication** over **SPI**, **I^{2}C**, and **USART**, in addition to **GPIO monitoring**, **power measurement**, and **code profiling**.

Serial Port



Serial Port communicates with any serial port on the system.

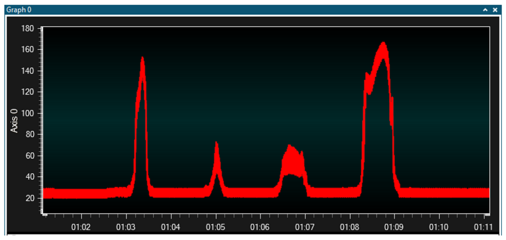
Terminal



```
 timeout: 9000,              success: function (data) {uery("#liHeaderLinkManageProfile").show
(); //show manage profile link

el").show(); //show myAtmel
ilCustomerGUIDIsRetriev
          if (e.which !== 8 && e.which !== 0 && (e.which < 48 || e.which > 57)) return false;
uery("#hdnOrderingCode
```

Terminal display and send simple text or numeric values.

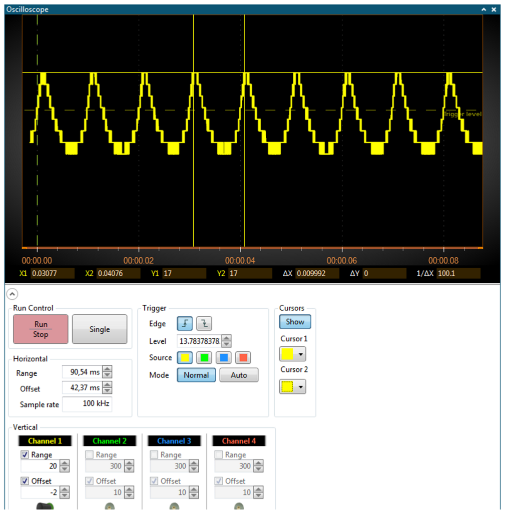
Graph



Graph can be used to plot data source values vs. time.

- **Cursors** (time axis) to measure application timing (e.g., PWM frequency)
- **Horizontal cursor** (data values) to control an application's set point or threshold
- **Band** highlights time periods above customizable thresholds
- **String markers** can be used to add descriptive text to graphed events

Oscilloscope



X1 0.03077 X2 0.04076 Y1 17 Y2 17 ΔX 0.009992 ΔY 0 1/ΔX 100.1

Level 13.78378378

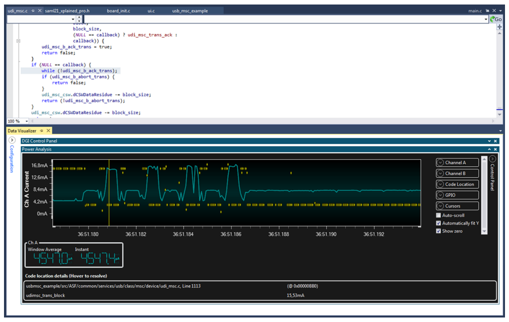
Range 90,54 ms
Offset 42,37 ms
Sample rate 100 kHz

	Channel 1	Channel 2	Channel 3	Channel 4
Range	☑ 20	☐ 300	☐ 300	☐ 300
Offset	☑ -2	☑ 10	☑ 10	☑ 10

Oscilloscope

- **Edge or threshold triggers** on rising or falling edges
- **Run-stop control** for single shot or continuous triggering
- **Cursors** (time axis) to measure application timing (e.g., PWM frequency)

Power Debugging



```
                        block_size,
                        (NULL == callback) ? udi_msc_trans_ack :
                        callback)) {
            udi_msc_b_ack_trans = true;
            return false;
    }
    if (NULL == callback) {
            while (!udi_msc_b_ack_trans);
            if (udi_msc_b_abort_trans) {
                    return false;
            }
            udi_msc_csw.dCSWDataResidue -= block_size;
            return (!udi_msc_b_abort_trans);
    }
    udi_msc_csw.dCSWDataResidue -= block_size;
```

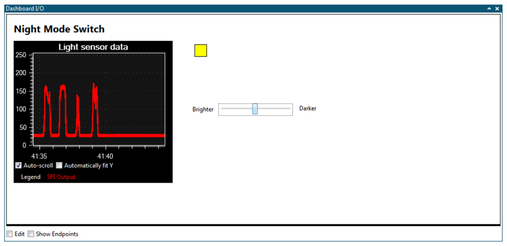
Code location details (Hover to resolve)

```
usbmsc_example/src/ASF/common/services/usb/class/msc/device/udi_msc.c, Line 1113    (@ 0x00000BB0)
udimsc_trans_block                                                                  15,53mA
```

Power Debugging

- Correlation of code execution and power consumption
- Displays current and voltage measured using **Power Debugger** (Embedded debugger on some kits)

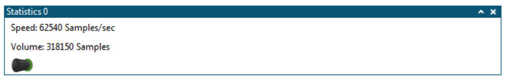
Custom Dashboard



Custom Dashboard

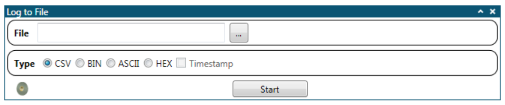
- Build a custom user interface to visualize and control user application using: graph, segment display, binary signals, labels, buttons, linear gauge: Value within defined range. Pie Chart (e.g., for packets lost vs. transmitted in wireless application).

Utilities


Statistics 0

Speed: 62540 Samples/sec

Volume: 318150 Samples


Log to File

File

Type [✓ CSV] BIN ASCII HEX ☐ Timestamp

- **Samplerate Counter** to validate MCU frequencies (e.g., rate of transmitted ADC samples)
- **File Logger** module logs all incoming data to a file of selectable format

1.1 Getting Help

Help can be opened at any time by clicking F1. By selecting a module in the Configuration window and clicking F1, help will be opened at the relevant chapter automatically.

1.2 Key Concepts

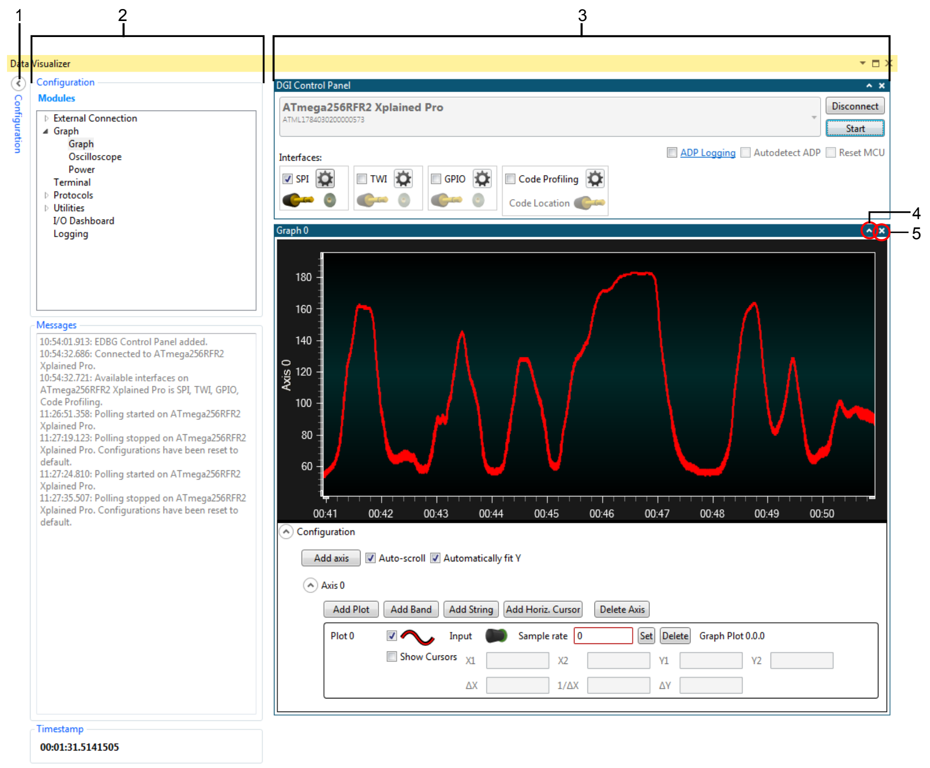
This section describes the key concepts to understand when working with the Data Visualizer.

1.2.1 Workspace

Data Visualizer is made up of several elements such as graphs, interfaces, and controls. All these elements form the *workspace.*

The elements are called *modules*, in which any of them can be added to the workspace.

Figure 1-1. Data Visualizer Workspace



ATmega256RFR2 Xplained Pro
ATML1784030200000573

☑ Auto-scroll ☑ Automatically fit Y

Plot 0 ☑ Input Sample rate 0 Graph Plot 0.0.0 ☐ Show Cursors

Messages

```
10:54:01.913: EDBG Control Panel added.
10:54:32.686: Connected to ATmega256RFR2
Xplained Pro.
10:54:32.721: Available interfaces on
ATmega256RFR2 Xplained Pro is SPI, TWI, GPIO,
Code Profiling.
11:26:51.358: Polling started on ATmega256RFR2
Xplained Pro.
11:27:19.123: Polling stopped on ATmega256RFR2
Xplained Pro. Configurations have been reset to
default.
11:27:24.810: Polling started on ATmega256RFR2
Xplained Pro.
11:27:35.507: Polling stopped on ATmega256RFR2
Xplained Pro. Configurations have been reset to
default.
```

Timestamp

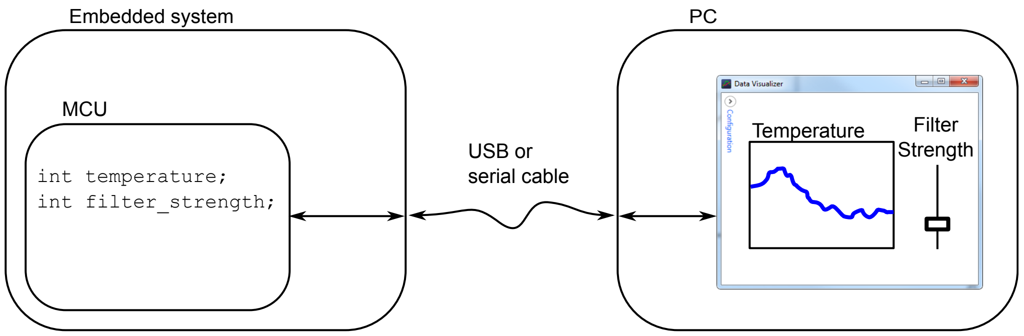
00:01:31.5141505

1. **Expand/Collapse Configuration pane** button. 2. **Configuration and Messages** pane. 3. Active modules. 4. **Minimize module** button. 5. **Remove module** button.

1.2.2 Connection Overview

The Data Visualizer communicates with the firmware running in the MCU of the embedded system. Variables in the firmware can be transferred in both directions. In the following example, the temperature value is sent to the visualizer and plotted in a graph. The filter strength value is set by dragging the slider in the visualizer, and is then sent to the MCU.

Figure 1-2. Data Visualizer Connection Overview

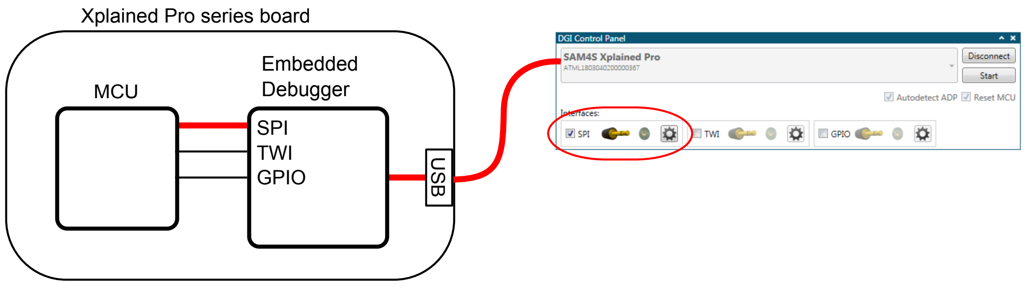


The communication can take place in a serial cable or USB if the embedded system contains an Embedded Debugger. (The Xplained Pro MCU boards contain Embedded Debuggers.)

1.2.3 Embedded Debugger's Data Gateway Interface

The Xplained Pro family of boards contain an *Embedded Debugger* chip. It has a Data Gateway Interface (DGI) that lets the MCU easily communicate with the Data Visualizer through either its SPI or TWI interface, or by GPIO pins.

Figure 1-3. The Data Gateway Interface

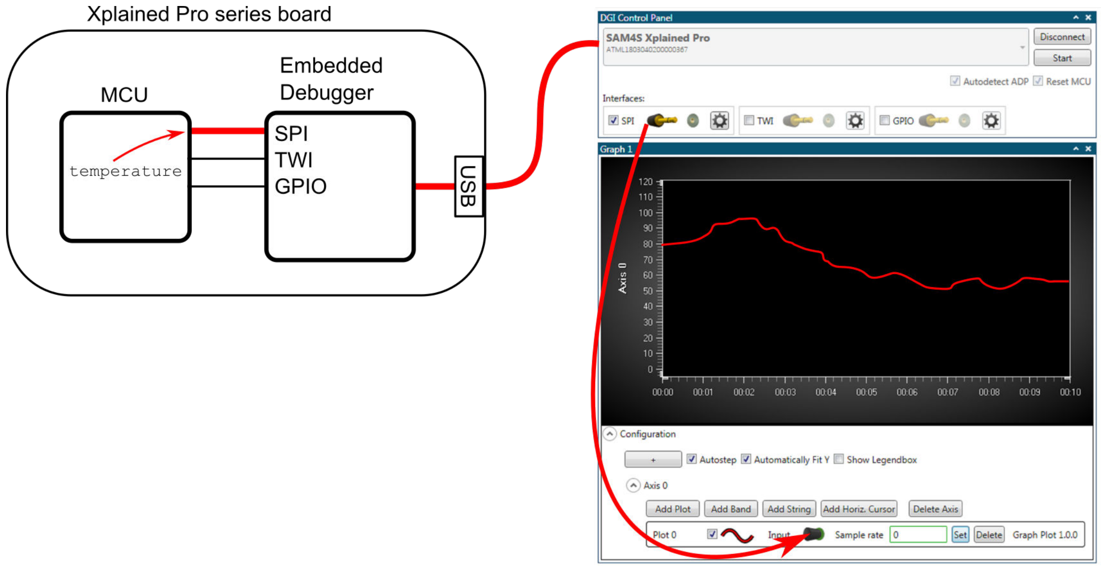


In the Data Visualizer, the **DGI Control Panel** is the module that communicates with the Embedded Debugger's Data Gateway Interface. When the board is connected to the computer with the USB cable, it can be selected in the control panel. A list of available interfaces will appear. Enable one or more of them by checking the boxes. In the figure above, the **SPI** interface is enabled. The MCU can now communicate with the Data Visualizer on its SPI port.

1.2.4 Simple Transfer

Sending a single value from the MCU to the Data Visualizer is quite simple. In the figure below, the MCU sends the *temperature* variable over its SPI interface. In the visualizer, the **SPI** interface on the Embedded Debugger has been enabled. The Embedded Debugger will transmit the SPI data to the visualizer through the **DGI Control Panel**.

Figure 1-4. A Simple DGI Transfer



- [✓] Autodetect ADP [✓] Reset MCU
- [✓] SPI [ ] TWI [ ] GPIO
- [✓] Autostep [✓] Automatically Fit Y [ ] Show Legendbox
- Sample rate: 0

To visualize the temperature data, a **Graph** has been added. The SPI data is routed to the plot by dragging the plug icon from the **SPI** interface in the **DGI Control Panel**, and dropping it in the plot area. This will add a new plot to the **Graph** module.

1.2.5 Endpoints

Data in the Data Visualizer originates from an endpoint and ends in an endpoint. The endpoints are referred to as **sinks** and **sources**. A data **source** sends data to one or more connected **sinks**.

In the workspace, the endpoints are represented by the graphical symbols shown below.

Figure 1-5. Data Source

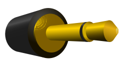


Figure 1-6. Unconnected Data Sink

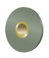
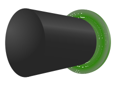


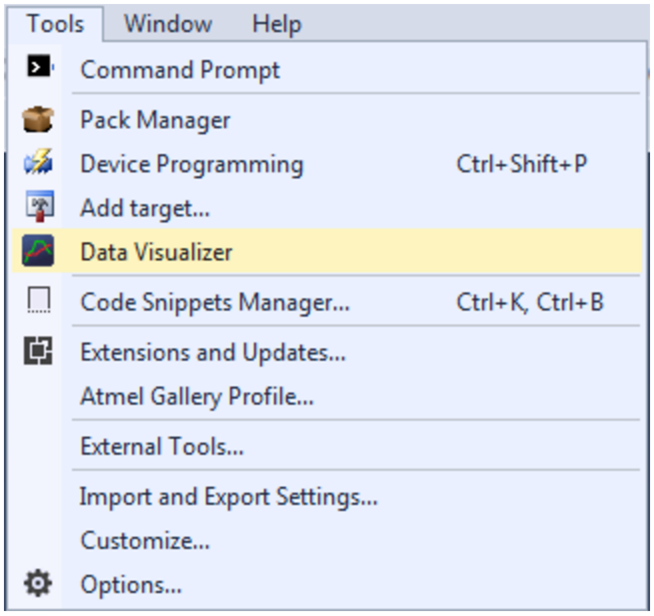
Figure 1-7. Connected Data Sink



1.3 Launching Data Visualizer

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

To run the 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 stand-alone version of the Data Visualizer has been installed, look for the shortcut in the Windows® start menu. The stand-alone version is available for download from gallery.atmel.com.

2. External Connection

All communication to the outside world is handled by modules found under the external connection section.

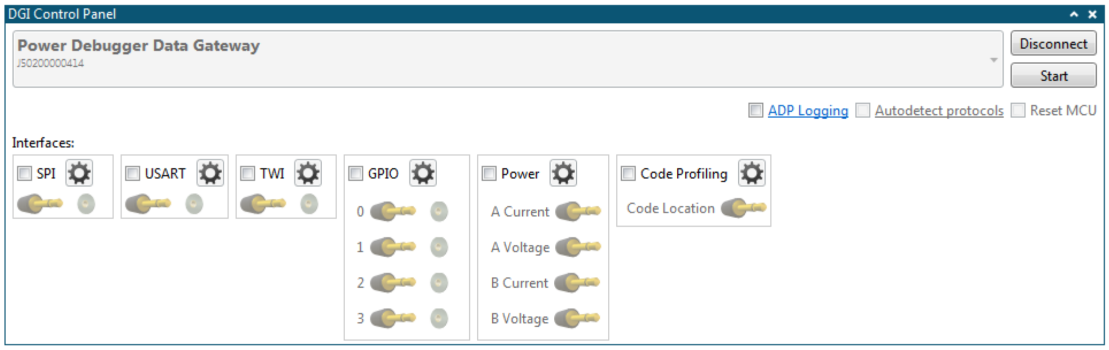
The **DGI Control Panel** communicates with any tool that has the Data Gateway Interface. It is capable of bidirectional communication over SPI, I^{2}C, and USART, in addition to GPIO monitoring, power measurement, and code profiling. The feature set varies by tool.

The **Serial Port Control Panel** communicates with any serial port on the system.

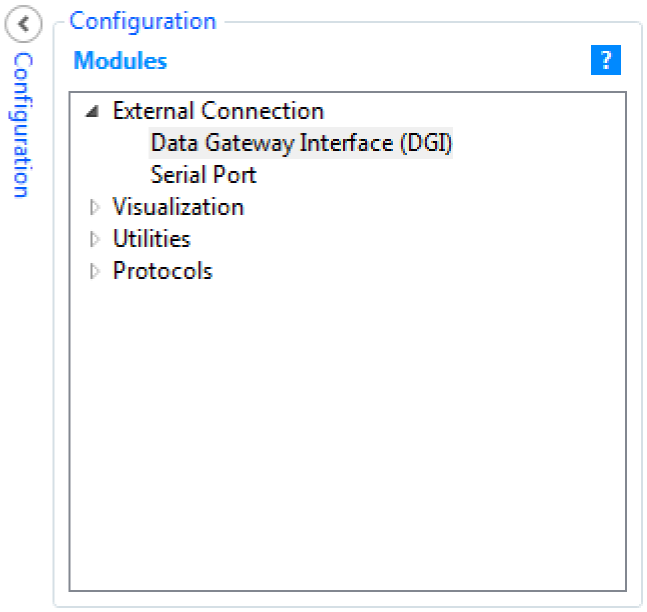
2.1 Data Gateway Interface (DGI)

The Data Gateway Interface is available on most kits with an Embedded Debugger. The DGI control panel can communicate with a DGI device. The figure below shows the DGI control panel module.

Figure 2-1. Data Gateway Interface Control Panel



Tip: A new DGI Control Panel can be opened in **External Connection** in the **Modules** section of the **Configuration** tab in the Data Visualizer.



All detected DGI devices are listed in the drop-down list with the kit name and serial number. Using the **Connect** button will connect to the selected DGI device and query for available interfaces. The available interfaces will be listed under **Interfaces**. To enable an interface, check the box next to the name. When an interface is enabled, the sources and sinks can be connected to other endpoints. The Gear button is used to configure the interface. See the interface-specific sections for an explanation of the configuration fields.

To start polling data from the interfaces, click the **Start** button. The **Reset MCU** check box will cause the MCU to be held in Reset during start.

The Data Visualizer supports two different protocols for Auto-configuration; the Atmel Data Protocol (ADP) and the Data Stream protocol. When using ADP, the configuration resides in the target application code and the target application sends the configuration settings, upon request, from the Data Visualizer. When using the Data Stream protocol, the configuration resides in files stored on the host computer and the target application just sends an ID to identify which configuration files to be loaded by the Data Visualizer. For more information on ADP, see Atmel Data Protocol. For more information on the Data Stream protocol, see Data Stream Protocol.

To enable Auto-configuration the **Autodetect protocols** option must be enabled.

☑ Autodetect protocols

After pushing **Connect** the Data Visualizer will enable all interfaces while it looks for the ADP handshake message or a Data Stream Configuration packet. If an ADP handshake message is received, the Data Visualizer will request configuration information from the target application. If a Data Stream Configuration packet is found, the Data Visualizer searches through the folders in the Auto-Configuration search path looking for configuration files with names matching the detected ID.

Important: To make sure the Data Visualizer detects the Data Stream Configuration packet, it must be sent by the target at least twice per second.

Important: Asynchronous serial protocols (e.g., UART protocols used by DGI USART and CDC Virtual COM port interfaces) use the following baud rates for auto-detection:

Table 2-1. Baud Rates Used on Asynchronous Interfaces for Auto-Detection of Protocols

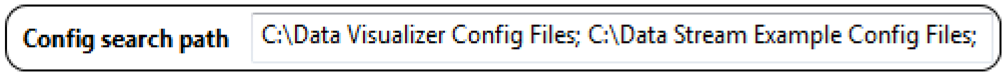
Baud Rate
9600
19200
38400
57600
115200
230400
500000
1000000
2000000

Using any baud rates not in the table will not work for auto-detection of protocols over asynchronous interfaces (DGI UART and Serial port/CDC Virtual COM port).

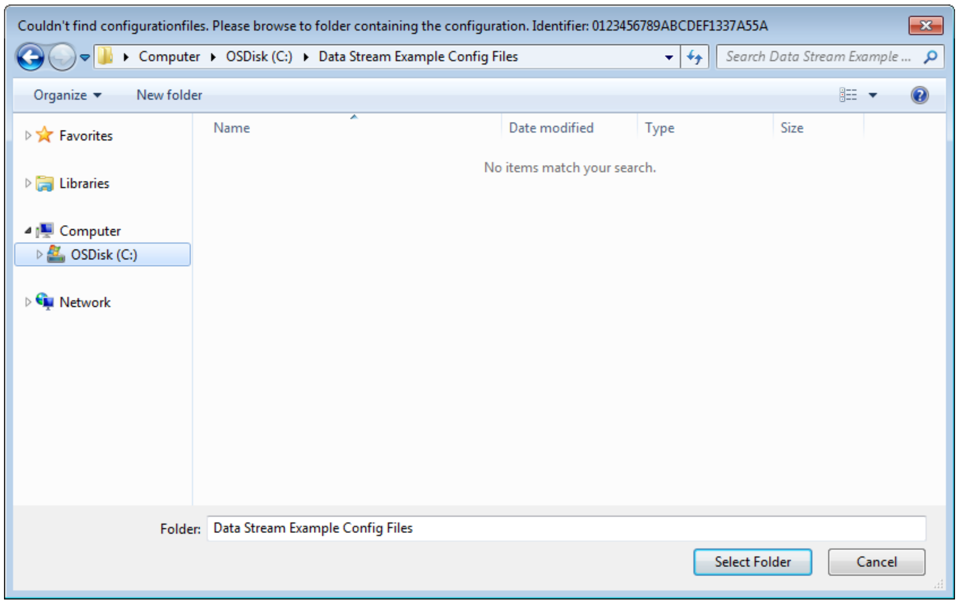
Tip: To see the current search path used by Data Visualizer to look for configuration files, check the **Show Config search path** option.

☑ Show Config search path

The search path is a semicolon separated list of paths. When Data Visualizer detects an Auto-Configuration ID, it will search through the paths in the list looking for configuration files with the correct file names.

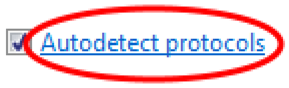

Config search path: C:\Data Visualizer Config Files; C:\Data Stream Example Config Files;

If the Data Visualizer cannot find any valid configuration files it will show a browser dialog window asking for the path to the folder where the correct configuration files reside.


Couldn't find configurationfiles. Please browse to folder containing the configuration. Identifier: 0123456789ABCDEF1337A55A
Computer ▸ OSDisk (C:) ▸ Data Stream Example Config Files
Search Data Stream Example ...
Organize
New folder
Favorites
Libraries
Computer
OSDisk (C:)
Network
Name
Date modified
Type
Size
No items match your search.
Folder: Data Stream Example Config Files
Select Folder
Cancel

After selecting a folder, the folder will be APPENDED to the Auto-Configuration search path.

Tip: To reset the search path and select a new single folder as the search path, click the link on the **Autodetect protocols** option text.



Data Visualizer will then pop up a browser dialog asking for the path to the folder where the configuration files reside. The original search path will be CLEARED and the newly selected folder will be set as search path.

Important: All three configuration files must reside in the same folder.

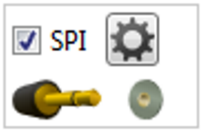
2.1.1 SPI Interface

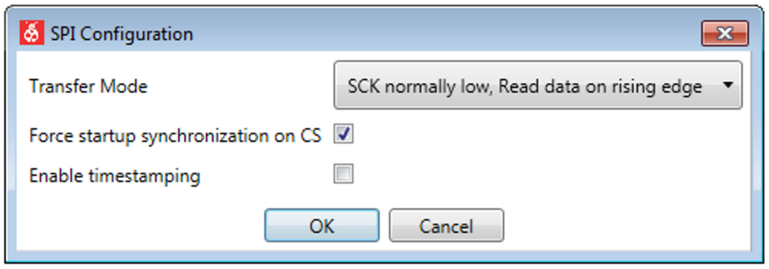
The SPI interface **source** contains the raw values received on the SPI interface. The **sink** sends values received back out on the SPI bus. For further details on the physical part of the SPI interface, see the user guide of the debugging tool to be used to sample the SPI data.

Important: If the SPI **sink** is connected to a **source** with a multibyte type, the byte order may be unpredictable.

Important: The SPI hardware module uses an active-low Chip Select (CS) signal. Any data sent when the CS pin is high will be ignored.

The **SPI Configuration** dialog is opened from the **SPI** interface in the **DGI control panel**.




SPI Configuration

Transfer Mode: SCK normally low, Read data on rising edge

Force startup synchronization on CS [✓]

Enable timestamping [ ]

Table 2-2. Configuration

Field Name	Values	Usage
Transfer Mode	• SCK normally low, Read data on rising edge • SCK normally low, Read data on falling edge • SCK normally high, Read data on falling edge • SCK normally high, Read data on rising edge	SPI mode, controlling clock phase and sampling.
Force synchronization on CS	ON or OFF	The SPI interface is only enabled after the Chip Select line has toggled twice.
Enable timestamping	ON or OFF	Data is timestamped through the DGI timestamp interface (yields a slower transfer rate).

Related Links

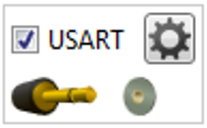
Sink Data Conversion

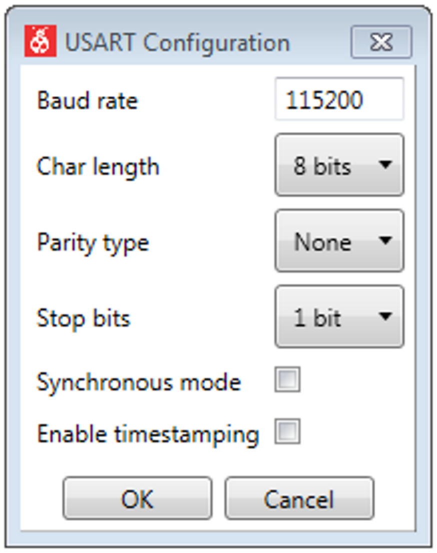
2.1.2 USART Interface

The USART interface source contains the raw values received on the USART interface. The **sink** sends values received back out on the USART interface. For further details on the physical part of the USART interface, see the user guide of the debugging tool to be used to sample the USART data.

Important: If the USART **sink** is connected to a **source** with a multibyte type, the byte order may be unpredictable.

The **USART Configuration** dialog is opened from the **USART** interface in the **DGI Control Panel**.





Baud rate 115200
Char length 8 bits
Parity type None
Stop bits 1 bit
Synchronous mode
Enable timestamping

Table 2-3. Configuration

Field Name	Values	Usage
Baud rate	0-2000000	Baud rate for UART interface in Asynchronous mode
Char length	5, 6, 7, or 8 bits	Number of bits in each transfer
Parity type	None, Even, Odd, Mark, or Space	Parity type used for communication
Stop bits	1, 1.5, or 2 bits	Number of Stop bits
Synchronous mode	ON or OFF	Selecting Synchronous or Asynchronous mode
Enable timestamping	ON or OFF	Data is timestamped through the DGI timestamp interface (yields a slower transfer rate)

Related Links

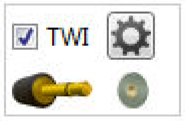
Sink Data Conversion

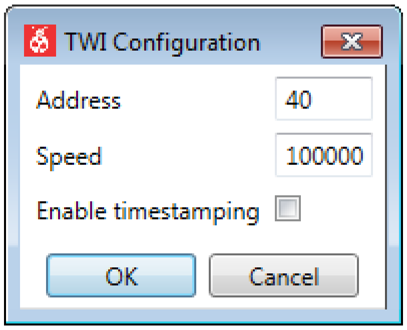
2.1.3 TWI Interface

The TWI interface source contains the raw values received on the TWI interface. The **sink** sends values received back out on the TWI interface. For further details on the physical part of the TWI interface, see the user guide of the debugging tool to be used to sample the TWI data.

Important: If the TWI **sink** is connected to a **source** with a multibyte type, the byte order may be unpredictable.

The **TWI Configuration** dialog is opened from the **TWI** interface in the **DGI Control Panel**.




TWI Configuration

Address 40

Speed 100000

Enable timestamping

Table 2-4. Configuration

Field Name	Values	Usage
Address	0-127	TWI slave address
Speed	100000, 400000	Speed setting for TWI slave. Used for timing.
Enable timestamping	ON, OFF	Data is timestamped through the DGI timestamp interface (yields a slower transfer rate)

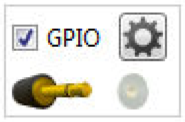
Related Links

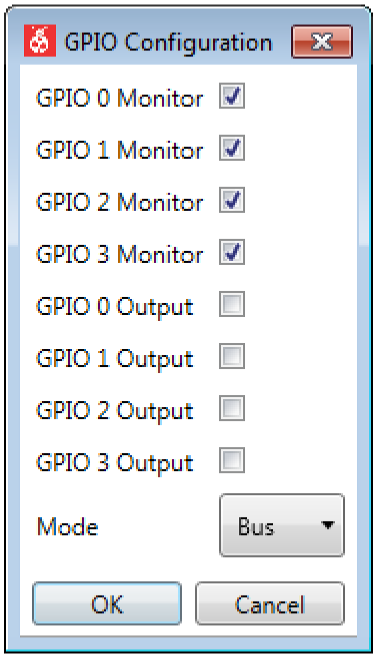
Sink Data Conversion

2.1.4 GPIO Interface

The GPIO interface source is of type uint8, and contains the bit values of the enabled GPIO pins. A packet is transmitted every time a pin toggles. The **sink** sends values received back out to the GPIO pins. For further details on the physical part of the GPIO interface, see the user guide of the debugging tool to be used to sample the GPIO data.

The **GPIO Configuration** dialog is opened from the **GPIO** interface in the **DGI Control Panel**.





GPIO Configuration

GPIO 0 Monitor [✓]

GPIO 1 Monitor [✓]

GPIO 2 Monitor [✓]

GPIO 3 Monitor [✓]

GPIO 0 Output [ ]

GPIO 1 Output [ ]

GPIO 2 Output [ ]

GPIO 3 Output [ ]

Mode: Bus

Table 2-5. Configuration

Field Name	Values	Usage
GPIO 0 Monitor	ON, OFF	Monitor GPIO pin 0
GPIO 1 Monitor	ON, OFF	Monitor GPIO pin 1
GPIO 2 Monitor	ON, OFF	Monitor GPIO pin 2
GPIO 3 Monitor	ON, OFF	Monitor GPIO pin 3
GPIO 0 Output	ON, OFF	Enable GPIO pin 0 output
GPIO 1 Output	ON, OFF	Enable GPIO pin 1 output
GPIO 2 Output	ON, OFF	Enable GPIO pin 2 output
GPIO 3 Output	ON, OFF	Enable GPIO pin 3 output
Mode	Pin, Bus, Latched Bus	GPIO pins as separate pins, a 4-bit bus, or a 3-bit bus that is latched on rising edge of GPIO3

Important: When using any of the bus modes (Bus or Latched Bus) all GPIOs are sampled but only those GPIOs that have monitoring enabled will trigger a sample. For example, if GPIO 0 to GPIO2 all have GPIO Monitor disabled but GPIO 3 has Monitor enabled, then GPIO values will only be sampled when GPIO 3 changes but all four GPIO values will be read when GPIO 3 changes.

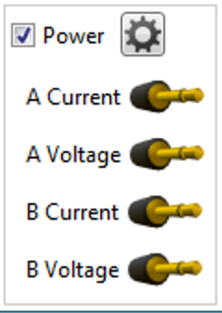
Related Links

Sink Data Conversion

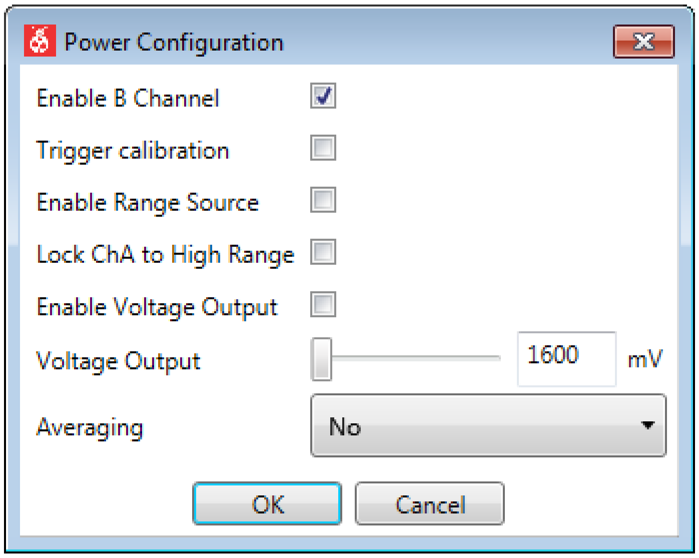
2.1.5 Power Interface

The **Power** interface measures the power consumption of the connected circuitry. For more information on the hardware part of the power interface, see the user guide of the debugging tool to be used for the power measurements.

The **Power Configuration** window is opened from the **Power** interface in the **DGI Control Panel**.



The content of the **Power Configuration** window will vary depending on the capabilities of the connected debugging tool.



Enable B Channel ☑
Trigger calibration ☐
Enable Range Source ☐
Lock ChA to High Range ☐
Enable Voltage Output ☐
Voltage Output 1600 mV
Averaging No

Table 2-6. Power Configuration Options

Field Name	Values	Usage
Enable B Channel	ON, OFF	Enables the second power measurement channel. The A channel is always enabled.
Trigger calibration	ON, OFF	Triggers the calibration procedure of the current measurement circuitry. For further details, see Power Measurement Calibration.
Enable Range Source	ON, OFF	Provides a range **source**, indicating which range is in use for the primary power measurement channel. The physical hardware used to measure power consumption will have different configurations depending on the instantaneous current measured. Each configuration is referred to as a range.
Lock ChA to High Range	ON, OFF	On the Power Debugger, the A channel can be locked to the *high* range to avoid automatic switching to the *low* range. This allows

Field Name	Values	Usage
		detection of short spikes in current consumption without critical samples being lost when switching between the ranges.
Enable Voltage Output	ON, OFF	Enable Power Debugger Voltage Output with the value given by the **Voltage Output** slider.
Voltage Output	0 - 5500 mV	The Power Debugger features an adjustable target supply that can be used to power the target application. This setting controls the output voltage of this supply. The **Enable Voltage Output** option must be enabled for the setting to take effect.

Tip: Any configuration changes will not take effect until clicking **OK** in the Power Configuration window. E.g., to enable the Voltage Output the **Enable Voltage Output** option must be checked, the **Voltage Output** value set and then after pushing **OK** the voltage output will actually be enabled and set according to the slider value.

Tip: The channel A range lock will not force the debugger to return to the high current range if already running in the low range. Either wait for a current high enough to force it to change, or simply **Stop** and **Start** the debugger.

Important: The **Power** interface can only be used with the **Power** module. Neither the **Oscilloscope** module nor the **Graph** module can be used with the **Power** interface.

2.1.6 Code Profiling

The **Code Profiling** interface uses the debug interface of the target device to access internal data like Program Counter and memory locations. It provides timestamped samples of the Program Counter address, allowing an insight in the program execution of the device. The user can also select arbitrary memory addresses to poll and control data variables at those locations. In addition, it is possible to monitor the state of the stack and the Power-Saving/Sleep mode of the target. Finally, it is possible to receive arbitrary data from the target application through a message pipe in the target On-Chip Debug (OCD) system.

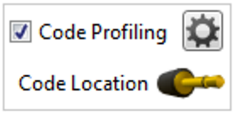
The availability of the above features varies with target device types and more details can be found in the following sections.

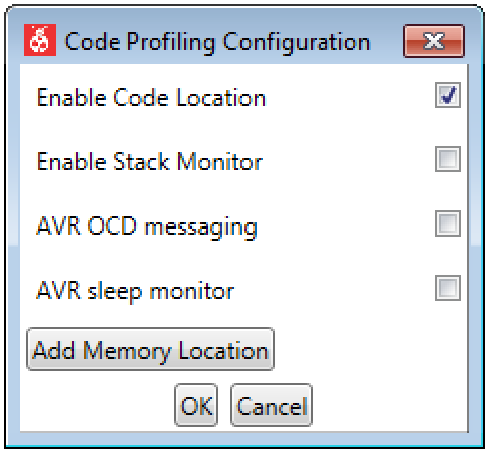
2.1.6.1 Code Profiling Interface

For a couple of examples on how to configure and use the **Code Profiling** interface, see Data Polling Example and Program Counter Polling.

Important: The Code Profiling interface 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 target device through the Atmel Studio debugger backend.

The **Code Profiling Configuration** window can be opened after enabling the **Code Profiling** interface in the **DGI Control Panel**.




Code Profiling Configuration

Enable Code Location	✓
Enable Stack Monitor	☐
AVR OCD messaging	☐
AVR sleep monitor	☐

Table 2-7. Configuration

Field Name	Values	Usage
Enable Code Location	ON, OFF	Controls the state of the Program Counter sampling.
Enable Stack Monitor	ON, OFF	Enables polling of the Stack Pointer to monitor stack usage (AVR® MCU with UPDI only)
AVR MCU OCD messaging	ON, OFF	Enables routing of OCD messages to Data Visualizer rather than Atmel Studio.
AVR MCU Sleep monitor	ON, OFF	Enables monitoring of the Sleep state of the MCU (AVR MCU with UPDI only)
Add Memory Location		Adds a new entry of memory location to poll and control. A text box for entering the address (hexadecimal), selecting data type and a Delete button will appear.

Each configuration option is detailed in the following sections.

2.1.6.2 Code Location

The Code location feature enables the Data Visualizer to sample the Program Counter of the target device. This makes it possible to see what is being executed on the target at various sample points. It is especially useful together with power measurements to correlate code execution with power consumption. The sampled PC values will only show part of the code execution as in most cases it is impossible to read out the PC values as fast as the target is executing instructions. The sampled values are still useful to indicate which code segment is being executed at any point in time.

For an example on how to use the Code location feature, see Program Counter Polling.

Important: The Code location feature is only available on SAM devices and AVR devices featuring the UPDI debugging and programming interface.

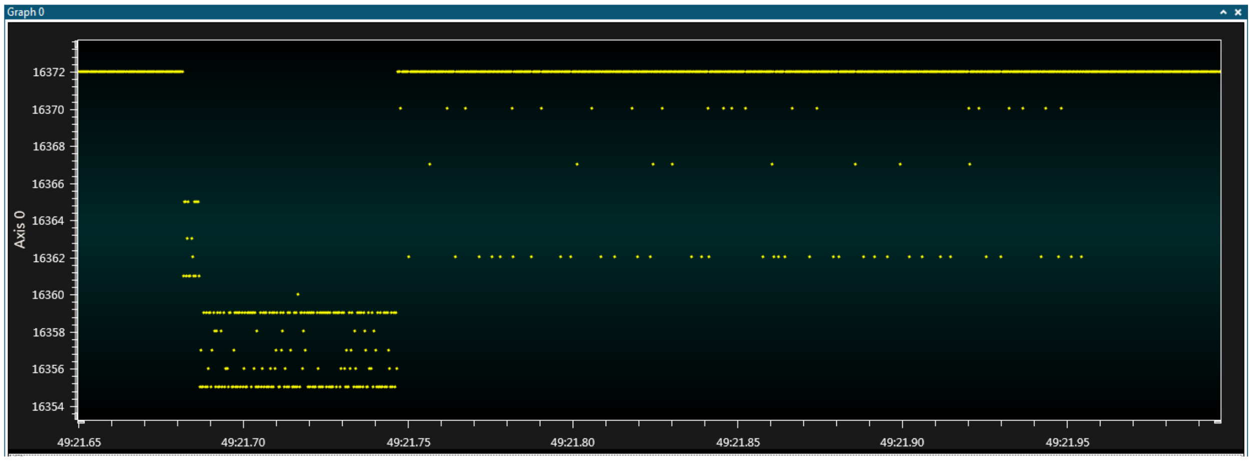
2.1.6.3 Stack Monitor

The Stack Monitor enables developers to monitor the stack usage of their code at run-time. This is done by sampling the Stack Pointer register via the on-chip debug module. Enable the Stack Monitor in the Code Profiling Configuration dialog, then connect the Stack Monitor source to a graph plot sink and start a debug session.

Important: The Stack Monitor feature is only available on AVR devices featuring the UPDI programming and debugging interface.

The Stack Monitor feature is implemented using polling, which means that not all stack levels will be visible. The granularity of the resulting graph is a function of the speed of the device clock, the UPDI clock speed and the nature of the application code. It is recommended to set the UPDI clock to maximum when using the Stack Monitor.

The example shown here is tracing the stack as points (not plot) from an application running on an ATtiny817. The points show samples with the Stack Pointer in "Idle state" in the main loop pointing to address 16372 (0x3FF4) and decrementing as functions are called.



Note: The Data Visualizer has no knowledge of the configuration of the stack on the device, and thus only shows raw samples of the Stack Pointer.

2.1.6.4 AVR MCU OCD Messaging

The AVR MCU OCD messaging system is a side-channel in the on-chip debug module. It is used extensively in some OCD variants to communicate with the core when it is stopped, but is not used by the system during Run mode. It can be used by end-user code to send messages to the debugger at run time. In Run mode, the debugger constantly polls the OCD for run/stop status, and at the same time picks up any messages. AVR MCU OCD messaging is a channel for code instrumentation without using any dedicated pins (other than the debug pins). Messages are single 8-bit values and are by default sent to Atmel Studio and displayed in the Output window as hex values, unless routed to Data Visualizer.

AVR MCU OCD messaging can be used in several ways. The examples below show three examples of various techniques.

- No handshaking, no guaranteed delivery
- With handshaking, blocking transport
- With handshaking, non-blocking transport

There is no standard way to use OCD messaging. The techniques shown in these examples each have advantages and disadvantages, and make use of different resources on the target device. Not all AVR devices support OCD messaging, and not all applications are suited to the use of OCD messaging. It is essentially a side-channel of the on-chip-debug system.

Enable AVR MCU OCD messaging in the Code Profiling Configuration dialog, then connect the AVR MCU OCD messaging source to a graph plot sink and start a debug session. Messages will not appear unless code is instrumented accordingly.

The most typical use-case for AVR MCU OCD messaging is ASCII printf-style debugging displayed on a terminal, as demonstrated in the examples. However, it could be used to transport any 8-bit data values, or even a composite structure. Messages can, for example, be sent from an ADC sample-complete interrupt, writing the 8-bit value of an ADC sample directly to the OCD message register. This can then be plotted directly onto a graph in Data Visualizer.

The 'default' OCD message channel to Atmel Studio operates at a fixed sample rate with 50 ms period. When enabled from Data Visualizer, the polling loop makes use of 'spare' cycles in the debugger to read and transport OCD messages. This leads to a higher throughput, but is also less deterministic in timing.

AVR MCU OCD Messaging Without Handshaking
The simplest form of using AVR MCU OCD messaging is writing directly to the register without any form of handshaking. This might be appropriate when, for example, execution speed is more important than data completeness. A single write to the OCD message register overwrites the previous value, even if it has not been read by the debugger yet. This could also be used for slow-changing data.

The following example shows how to output AVR MCU OCD messages without handshaking on various AVR MCU architectures.

OCD Messaging on AVR UPDI Target Device

```
// Example of OCD message on AVR UPDI target
// No handshaking, no guarantee
#define SYSCFG_OCDM SYSCFG.reserved_0x18
void ocd_putchar (char c)
{
    SYSCFG_OCDM = c;
}
```

OCD Messaging on AVR XMEGA® Target Device

Note: DGI-based OCD messaging is not yet supported on XMEGA targets. The code shown here will push OCD messages to Atmel Studio.

```
// Example of OCD message on AVR XMEGA target
// No handshaking, no guarantee
void ocd_putchar (char c)
{
    OCD.OCDR0 = c;
}
```

OCD Messaging on AVR JTAG Target Device

```
// Example of OCD message on AVR JTAG target
// No handshaking, no guarantee
void ocd_putchar (char c)
{
    OCDR = c;
}
```

OCD Messaging with Handshaking and Blocking
This example will block on each character sent via the OCD messaging system until it is ready to accept a new character. A simple timeout is employed to prevent full lockup of code if the debugger is

disconnected. This example runs on an AVR ATtiny817 using the UPDI interface, but a similar mechanism could be used on other AVR MCU architectures supporting OCD messaging.

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

#define SYSCFG_OCDM SYSCFG.reserved_0x18
#define SYSCFG_OCDMS SYSCFG.reserved_0x19

bool ocd_print_ready (void)
{
    // Has the last character been collected?
    return !(SYSCFG_OCDMS & (1 << 0));
}

bool ocd_print_char (char msg)
{
    // Simple timeout mechanism
    uint8_t timeout = 0xFF;
    while (timeout-- && !ocd_print_ready())
        ;

    // If the debugger fails to collect, continue
    if (!timeout)
        return false;

    // Drop off a message
    SYSCFG_OCDM = msg;
    return true;
}

void ocd_print (char* pmsg)
{
    // Send the message
    while (*pmsg) {
        if (!ocd_print_char(*pmsg++))
            return;
    }
}

int main(void)
{
    // Send an OCD message
    ocd_print ("Hello World\n");
    while (1)
        ;
}
```

Interrupt-Driven Bufferred OCD Messaging

A more complex method of using AVR MCU OCD messaging involves a small I/O buffer into which a printf function can inject data which will be gradually transferred to the debugger. A timer interrupt is used to periodically service the printf buffer. On each interrupt a character will be sent from the buffer, if the message channel is ready and data is available. This example runs on a megaAVR® device with JTAG interface, but a similar mechanism can be employed on other AVR device architectures supporting OCD messaging.

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

// Buffer allocated to OCD messaging
#define OCDR_BUFFER_SIZE 32
static uint8_t ocdr_buffer[OCDR_BUFFER_SIZE];

// Buffer pointers
static uint8_t head;
static uint8_t tail;

// Flag to indicate if a debugger is picking up the messages
static uint8_t debugger_attached = 1;
```

```
// Declarations
static int ocdr_putchar(char c, FILE *stream);
static FILE mystdout = FDEV_SETUP_STREAM(ocdr_putchar, NULL, _FDEV_SETUP_WRITE);

// Puts a char into the stream
static int ocdr_putchar(char c, FILE *stream)
{
    // If the debugger fails to collect, rather just abort
    if (!debugger_attached)
        return 1;

    // Increment head with wrapping
    uint8_t tmphead;
    tmphead = (head + 1 );
    if (tmphead >= OCDR_BUFFER_SIZE)
        tmphead = 0;
    if (tmphead == tail) {
        // Overflow, abort
        debugger_attached = 0;
        return 0;
    }

    // Add data
    ocdr_buffer[tmphead] = c;
    head = tmphead;
    return 1;
}

// Timer interrupt regularly sends data
ISR(TIMER0_OVF_vect)
{
    // If no data, continue
    if (head == tail)
        return;

    // If the previous byte has not been collected, continue
    if (OCDR & 0x80)
        return;

    // Increment tail
    uint8_t tmptail = (tail + 1);
    if (tmptail >= OCDR_BUFFER_SIZE)
        tmptail = 0x00;
    tail = tmptail;

    // Send data to debugger
    OCDR = ocdr_buffer[tmptail];

    // Reset attached flag to allow hot-plugging
    debugger_attached = 1;
}

void ocdr_printf_init (void)
{
    // Zero buffer pointers
    head = 0;
    tail = 0;

    // TC setup. 8Mhz DIV32 gives ~1ms overflow ticks
    TIFR = (1 << TOV0);
    TIMSK = (1 << TOIE0);
    TCCR0 = (1 << CS01) | (1 << CS00);
    sei();
}

int main(void)
{
    // Port init
    DDRB |= 0xFF;
    PORTB = 0x55;

    // Buffer init
    stdout = &mystdout;
    ocdr_printf_init();

    // Demo loop
    uint8_t c = 0;
    while(1)
```

```
        {
            c++;
            PORTB = ~c;
            printf("led %d\n", c);

            // Must delay > ~8ms to guarantee printf delivery
            uint16_t delay = 0x3FFF;
            while (delay--)
                ;
        }
    }
```

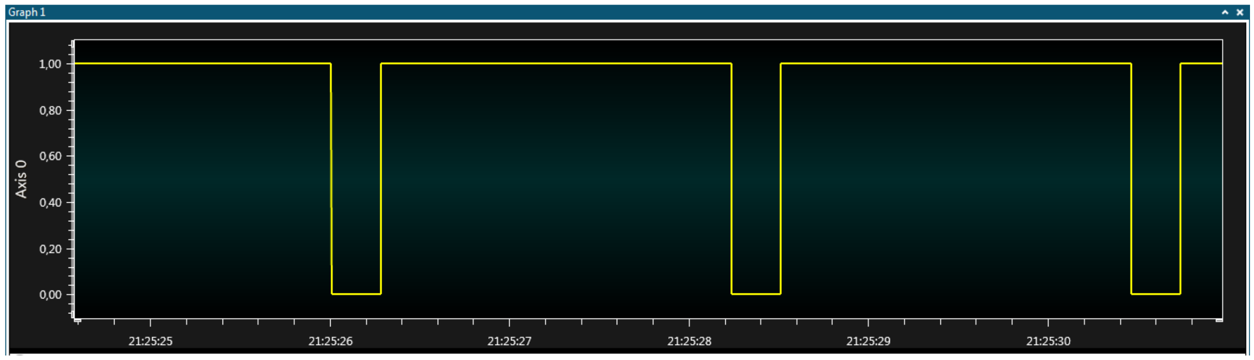
2.1.6.5 AVR MCU Sleep Monitor

The AVR MCU Sleep Monitor enables developers to monitor the Sleep mode state of the AVR MCU CPU at run-time. Sleep mode is a binary representation, and does not indicate which low-power mode is active (idle, power-down, etc.) The AVR MCU Sleep Monitor can be useful for determining the approximate amount of time the CPU spends in Sleep mode. Enable the AVR MCU Sleep Monitor in the Code Profiling Configuration dialog, then connect the AVR MCU Sleep Monitor source to a graph plot sink, and start a debug session.

Important: The AVR MCU Sleep Monitor feature is only available on AVR devices featuring the UPDI programming and debugging interface.

The AVR MCU Sleep Monitor feature is implemented using polling, which means that not ALL Sleep transitions will be visible. The granularity of the resulting graph is a function of the UPDI clock speed and the nature of the application code. It is recommended to set the UPDI clock to maximum when using the AVR MCU Sleep Monitor.

The graph below shows an example of the Sleep Monitor in use. A value of ' 1' indicates that the MCU is in Sleep mode, and '0' means it is running normally. From the plot, one can measure (using cursors) that the MCU is entering and exiting Sleep mode with a period of about 2.2s, and stays 'awake' for about 275 ms on each wake-up cycle.



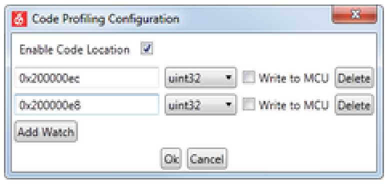
2.1.6.6 Data Polling and Control

The Data Polling and Control feature makes it possible to continuously sample and alter arbitrary memory locations in the target device. For an example on how to use this feature, see Data Polling and Control Example.

Important: The Data Polling and Control feature is only available on SAM devices.

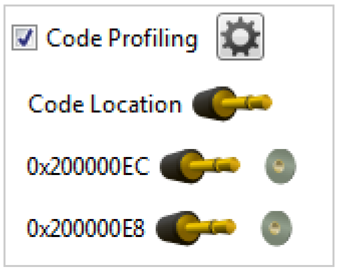
To add a memory location to be polled and/or controlled do the following.

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



0x200000ec uint32
0x200000e8 uint32

There will be one source and one sink for each memory location. Connect the source to any visualization module to monitor the value of the location and connect any data source to the sink to alter the value of the memory location.



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 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.

Data Polling Example

An example on how to use Program Counter sampling for power consumption analysis can be found in Program Counter Polling. The same Mass Storage Class example used in this section is also suited as an example on how to use the data polling and control of data variables features. A SAM L21 Xplained Pro board is connected to a host computer both through Target USB and Debug USB connectors on the kit. The ATSAML21 target device is running the USB Device MSC Example from ASF for SAM L21 Xplained Pro.

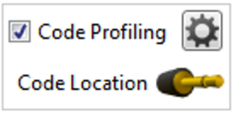
For more information on the hardware setup and target application code used in this example, see Data Polling Example Code.

Although this example makes use of the **Graph** and **Dashboard** modules the principles are the same for using the **Code Profiling** interface with the other modules in the Data Visualizer.

First, a graph will be set up to monitor variables in the target application.

To do:

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

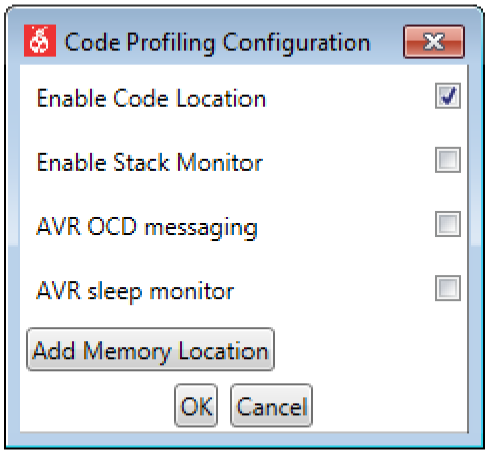


✓ Code Profiling

Code Location

To do:

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



Code Profiling Configuration

Enable Code Location ✓

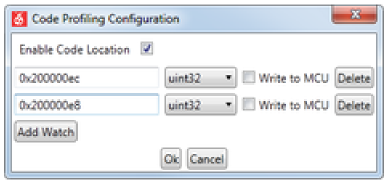
Enable Stack Monitor ☐

AVR OCD messaging ☐

AVR sleep monitor ☐

To do:

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



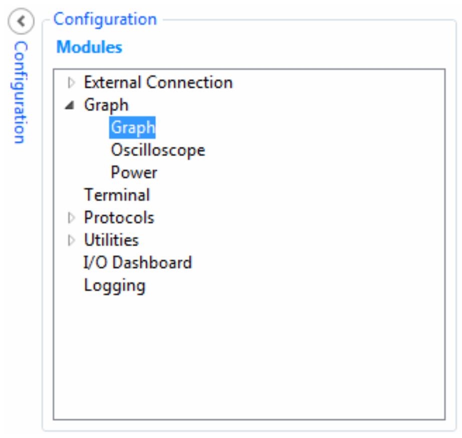
Code Profiling Configuration

Enable Code Location ✓

0x200000ec	uint32	☐ Write to MCU	Delete
0x200000e8	uint32	☐ Write to MCU	Delete

To do:

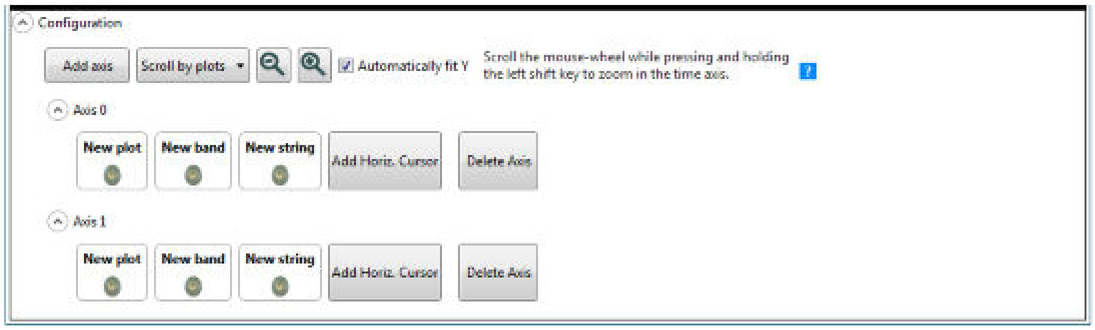
- Open the **Configuration** panel in Data Visualizer
- Add a graph by double-clicking the **Graph** module



A new Graph element will open with one y axis configured. However, there are two unrelated variables to monitor, therefore, two axes are needed.

To do:

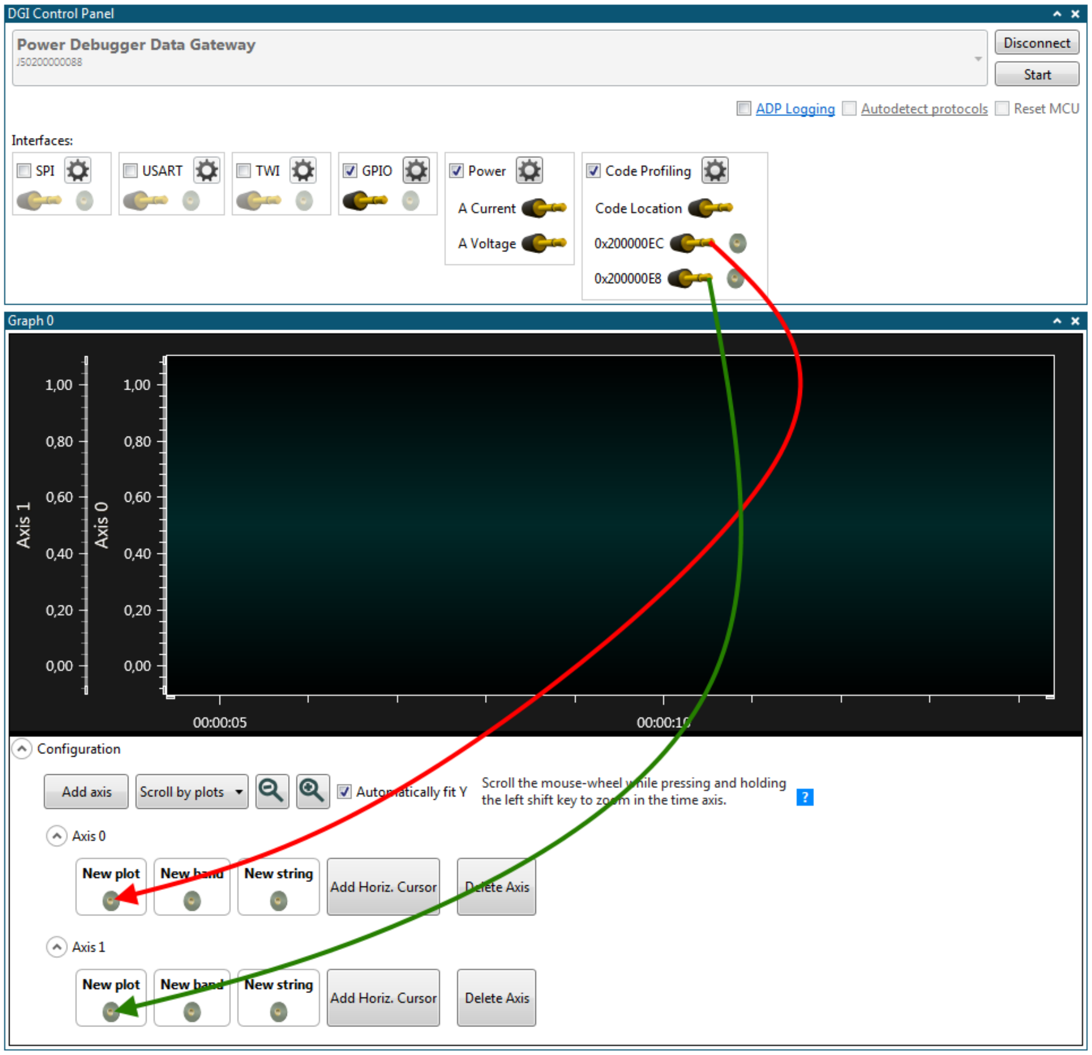
- Click the **Add axis** button to add an additional axis



Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

There are now sources (variables) and sinks (axes), to be connected together.

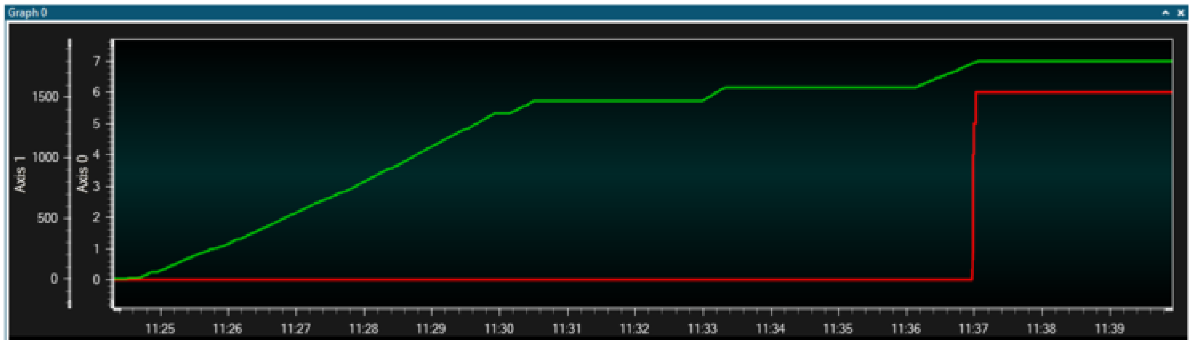
To do: Drag each of the **source** plugs on the **Code Profiling** interface into the **New plot** (**sink**) jack of each axis.



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.

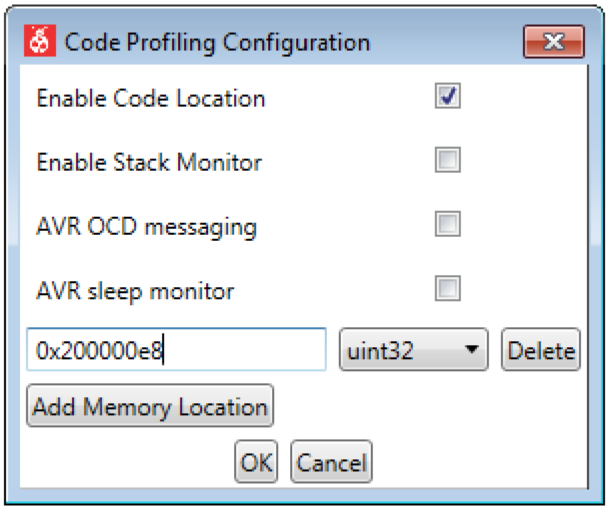
Look at the output in the graph in Data Visualizer. Format the disk and watch how the write cycles counter increments. Both values are plotted on independent axes, so they can be scaled accordingly. The output should look something like this:



The following part of this example shows how to use a dashboard to interact with the target application. For more information on the required code changes in the target application, see Application Interaction using Dashboard Controls.

To do:

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


Code Profiling Configuration

Enable Code Location ☑

Enable Stack Monitor ☐

AVR OCD messaging ☐

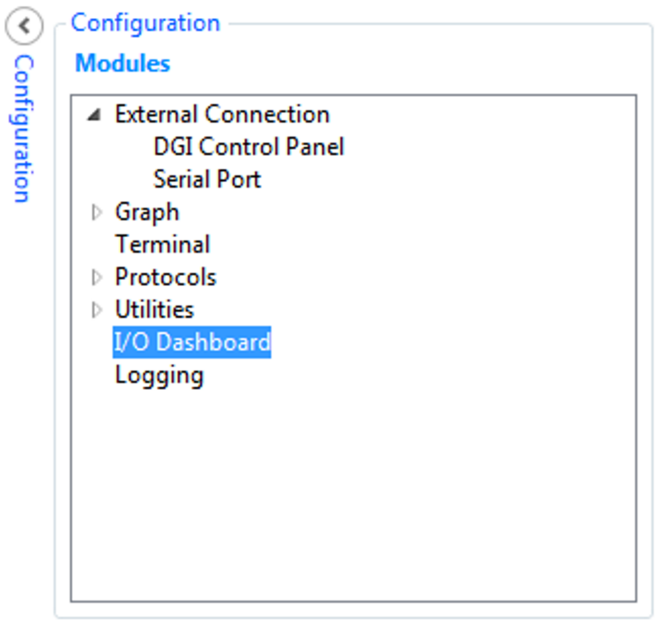
AVR sleep monitor ☐

0x200000e8 uint32

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

To do:

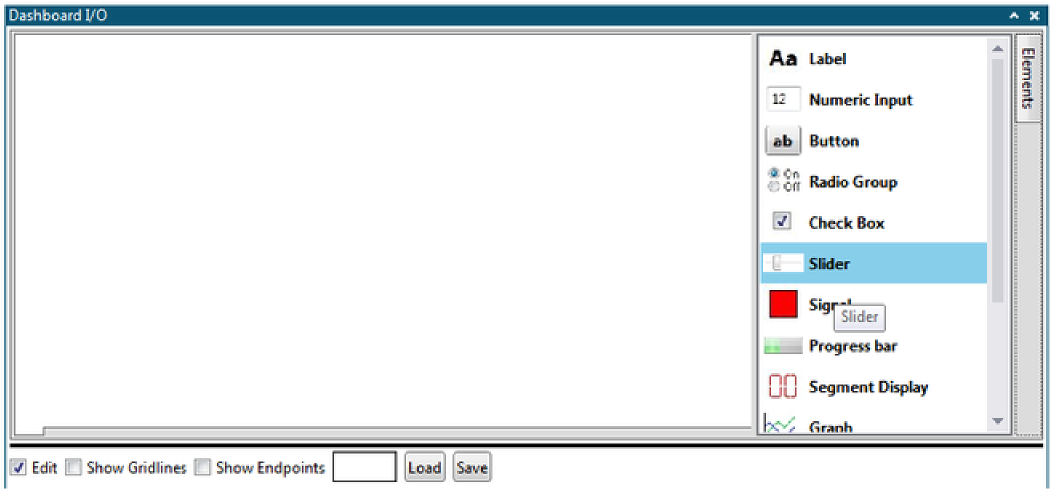
- Open the configuration panel
- Add a new I/O Dashboard component by double-clicking the I/O Dashboard module



A slider control can now be added to the dashboard.

To do:

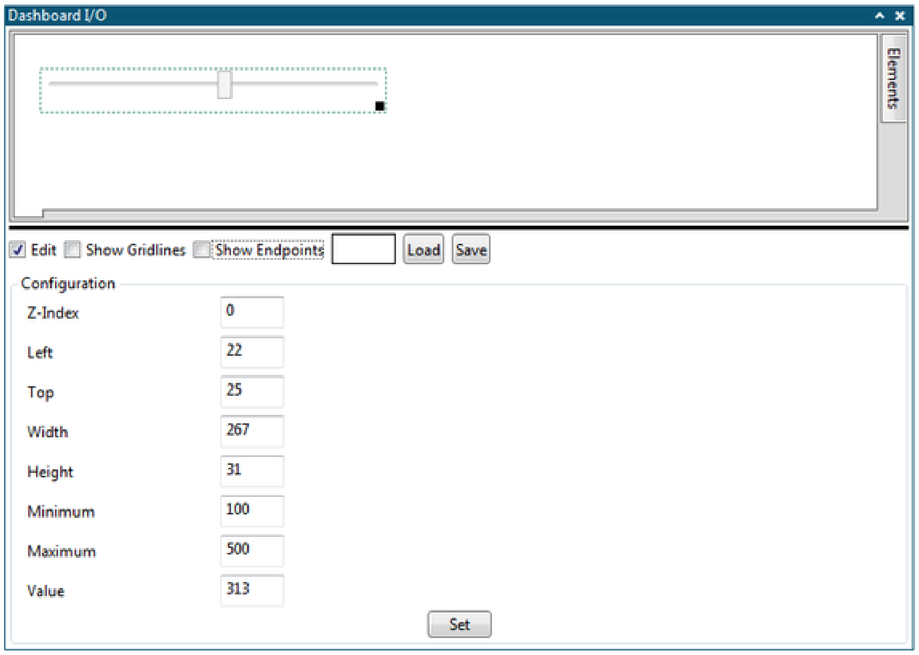
- Select the Edit checkbox
- Open the Elements tab
- Drag a Slider element onto the dashboard



A slider control needs to have some configuration parameters.

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

- Maximum = 500
- Minimum = 100



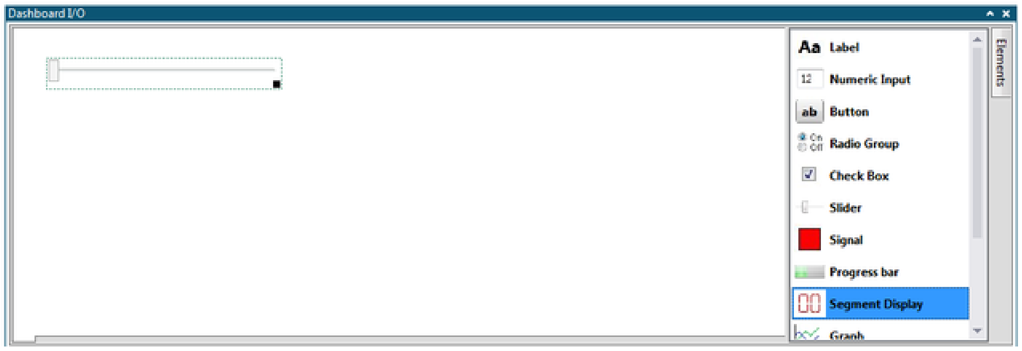
Configuration

Z-Index	0
Left	22
Top	25
Width	267
Height	31
Minimum	100
Maximum	500
Value	313

A segment display control can now be added to the dashboard.

To do:

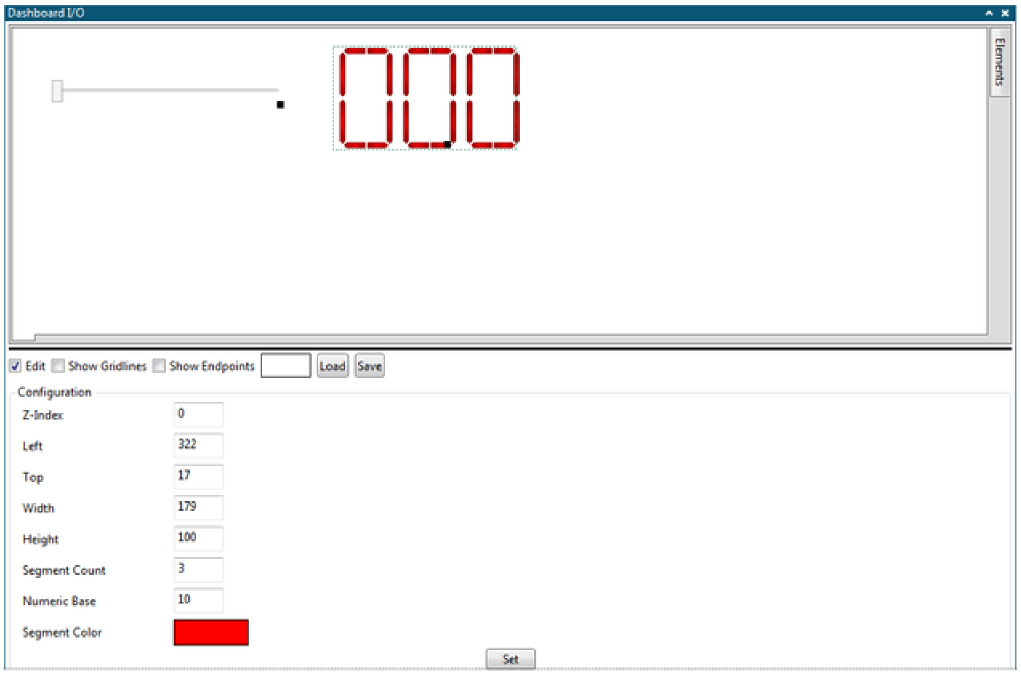
- Select the Edit checkbox
- Open the Elements tab
- Drag a Segment Display element onto the dashboard



A segment display control needs to have some configuration parameters.

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

- Segment Count = 3



Configuration

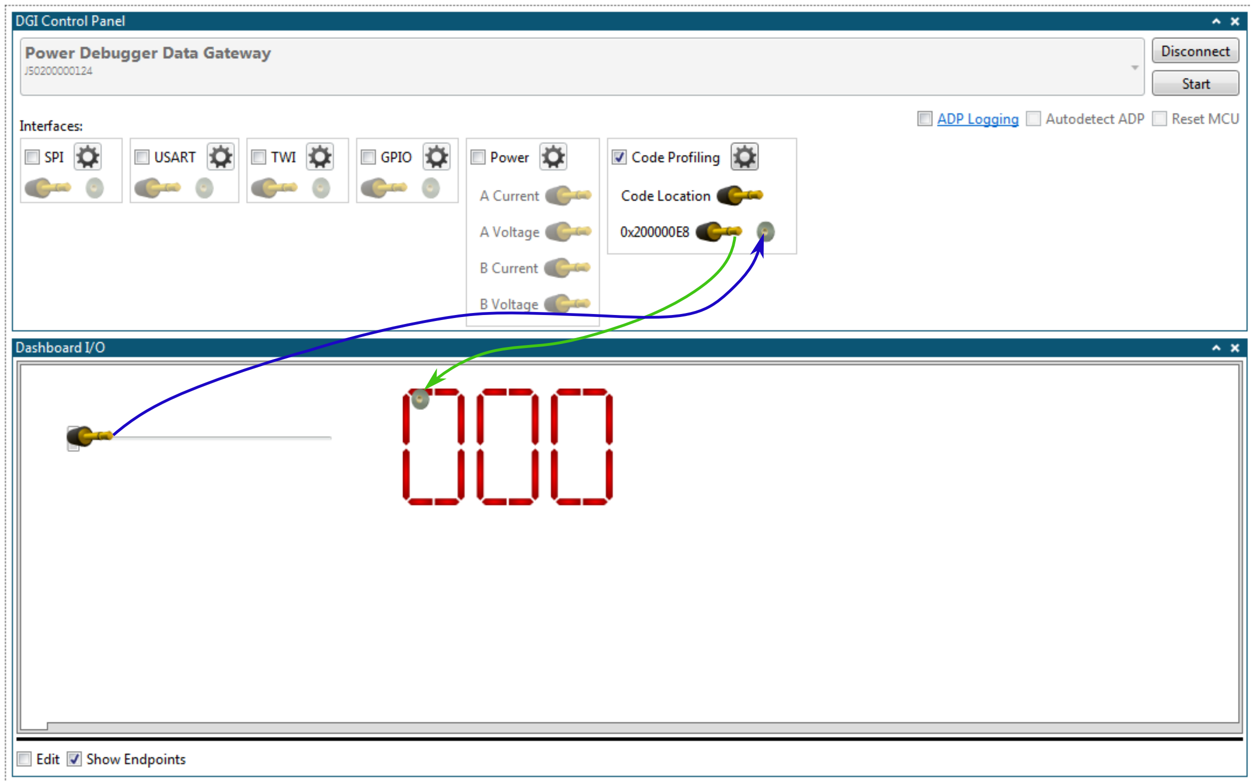
Z-Index	0
Left	322
Top	17
Width	179
Height	100
Segment Count	3
Numeric Base	10
Segment Color	🟥

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. The slider can now be connected to the sink and the segment display to the source.

To do:

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

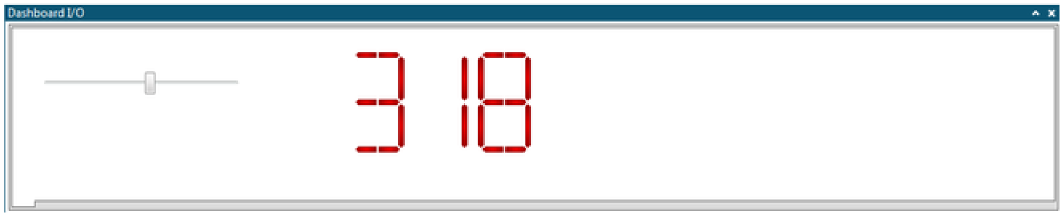


Now that the connections have been made in Data Visualizer, the system can be put into a running state and interaction with the variable can be made through the GUI.

To do:

- Deselect 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 in the variable. At the same time, the value is also read back from the target and displayed on the segment display.



2.1.7 Sink Data Conversion

Since DGI only can handle 8-bit values natively, all values received by DGI are remapped according to the rules in the following table.

Table 2-8. Data Conversion

Data Type	Conversion
Int8	Cast to uint8. 2’s complement value is retained.
Uint8	
Int16	2’s complement value is retained. Split into two uint8 values. Big endian.
Uint16	Split into two uint8 values. Big endian.
Int32	2’s complement value is retained. Split into four uint8 values. Big endian.
Uint32	Split into four uint8 values. Big endian.
Float	Cast to Int32
Double	Cast to Int32
XY8	X-value sent first, then Y-value
XYu8	X-value sent first, then Y-value
XY16	X-value sent first, then Y-value
XYu16	X-value sent first, then Y-value
XY32	X-value sent first, then Y-value
XYu32	X-value sent first, then Y-value
XYFloat	X-value sent first, then Y-value
XYDouble	X-value sent first, then Y-value
String	The ASCII values of each character is sent. A null termination is added.
StringFloat	Sent as a Int32 with the string following
Boolean	False is sent as 0, true as 1

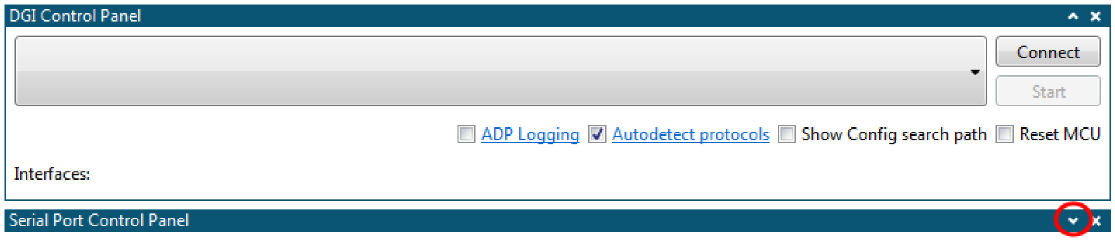
2.1.8 DGI Data Polling

The communication with the Data Gateway Interface (DGI) is done through a separate C++ DLL. When a session is started, it will poll the DGI device for data each 2 ms. However, because the CPU could be busy with other tasks, the polling might happen with a longer interval.

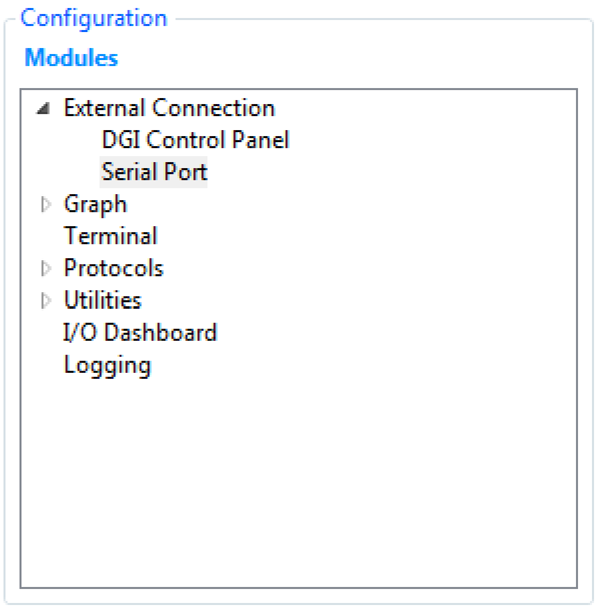
Since the DGI device has a limited buffer, the DLL needs to poll the device regularly to avoid an overflow. Therefore, it is important to keep the CPU usage low during polling sessions. In case of overflow problems, either decrease the transfer rate on the DGI interfaces or decrease the CPU load by shutting down applications.

2.2 Serial Port

The Data Visualizer can be connected to a standard PC serial port. The **Serial Port Control Panel** is by default opened and minimized under the **DGI Control Panel** when starting the Data Visualizer. To expand it, click the down arrow in the right corner of the minimized panel.

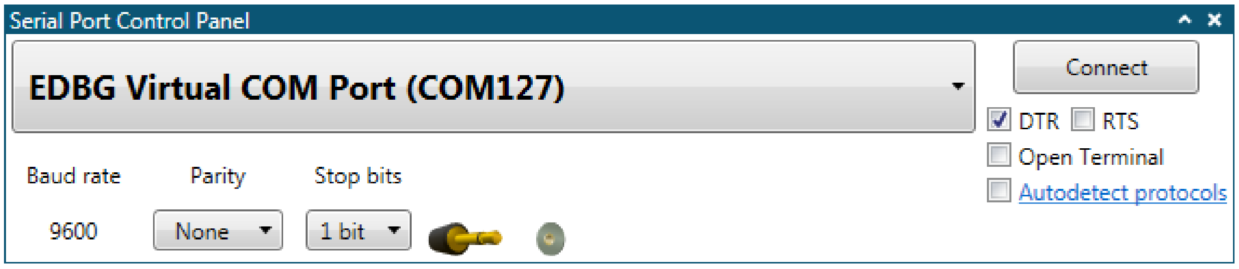


Tip: A new Serial Port Control Panel can be opened in **External Connection** in the **Modules** section of the **Configuration** tab in the Data Visualizer.



Baud rate, Stop bits, and parity must be set to match the required settings for the communication partner. A **sink** and a **source** endpoint is present to represent the outgoing and incoming data for the serial port. The endpoints of the serial port control panel is of uint8 data type, and follows the same conversion rules as the DGI control panel. The Open Terminal check box will cause a terminal module to automatically open and connect the endpoints. When disconnecting from a serial port, the created terminal module will be closed.

Figure 2-2. Serial Port Control Panel



EDBG Virtual COM Port (COM127)

DTR [✓]

Baud rate	Parity	Stop bits
9600	None	1 bit

Table 2-9. Configuration

Field name	Values	Usage
Baud rate	600-2000000	Baud rate of serial interface
Parity	None, Even, Odd, Mark, or Space	Parity type used for communication
Stop bits	1, 1.5, or 2 bits	Number of Stop bits
DTR	ON or OFF	Data Terminal Ready control signal of RS-232 serial communication
RTS	ON or OFF	Request To Send control signal of RS-232 serial communication
Open Terminal	ON or OFF	Opens a terminal upon connection with the **source** and **sink** connections automatically connected between the **Serial Port Control Panel** serial port and the terminal
Autodetect protocols	ON or OFF	Auto-detection of the Atmel Data Protocol or Data Stream protocol Auto-configuration. For more information on the protocols, see Atmel Data Protocol and Data Stream Protocol
Show Config search path	ON or OFF	Only available when **Autodetect protocols** is enabled. Shows the search path for Data Stream Auto-configuration files

The Data Visualizer supports two different protocols for Auto-configuration; the Atmel Data Protocol (ADP) and the Data Stream protocol. When using ADP, the configuration resides in the target application code and the target application sends the configuration settings, upon request, from the Data Visualizer. When using the Data Stream protocol, the configuration resides in files stored on the host computer and the target application just sends an ID to identify which configuration files to be loaded by the Data Visualizer. For more information on ADP, see Atmel Data Protocol. For more information on the Data Stream protocol, see Data Stream Protocol.

To enable Auto-configuration the **Autodetect protocols** option must be enabled.

☑ Autodetect protocols

After pushing **Connect** the Data Visualizer will enable all interfaces while it looks for the ADP handshake message or a Data Stream Configuration packet. If an ADP handshake message is received, the Data Visualizer will request configuration information from the target application. If a Data Stream Configuration packet is found, the Data Visualizer searches through the folders in the Auto-Configuration search path looking for configuration files with names matching the detected ID.

Important: To make sure the Data Visualizer detects the Data Stream Configuration packet, it must be sent by the target at least twice per second.

Important: Asynchronous serial protocols (e.g., UART protocols used by DGI USART and CDC Virtual COM port interfaces) use the following baud rates for auto-detection:

Table 2-10. Baud Rates Used on Asynchronous Interfaces for Auto-Detection of Protocols

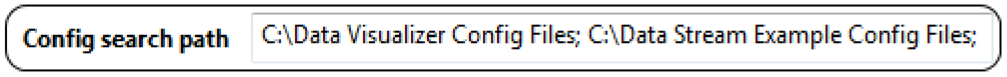
Baud Rate
9600
19200
38400
57600
115200
230400
500000
1000000
2000000

Using any baud rates not in the table will not work for auto-detection of protocols over asynchronous interfaces (DGI UART and Serial port/CDC Virtual COM port).

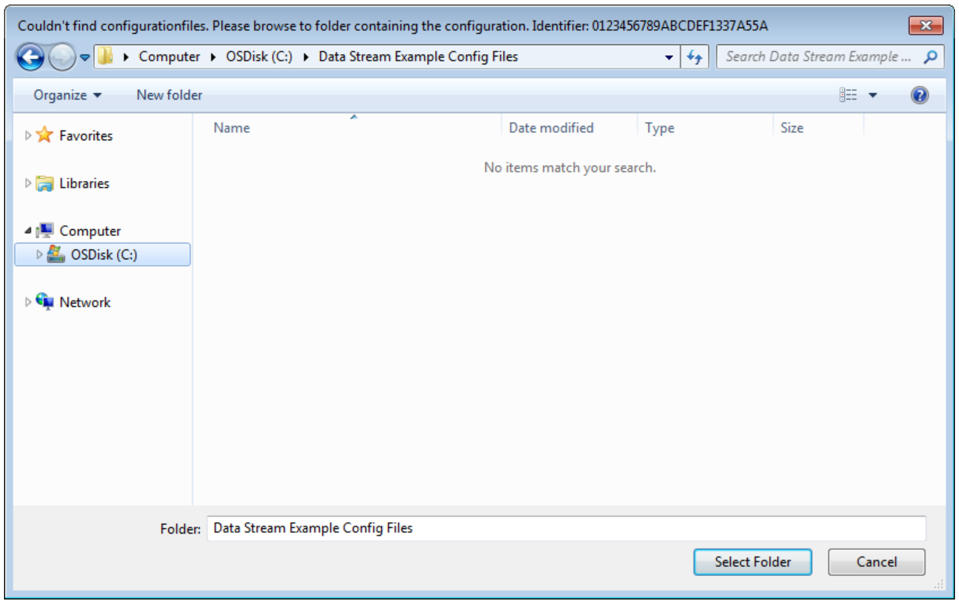
Tip: To see the current search path used by Data Visualizer to look for configuration files, check the **Show Config search path** option.

☑ Show Config search path

The search path is a semicolon separated list of paths. When Data Visualizer detects an Auto-Configuration ID, it will search through the paths in the list looking for configuration files with the correct file names.

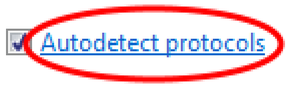

Config search path: C:\Data Visualizer Config Files; C:\Data Stream Example Config Files;

If the Data Visualizer cannot find any valid configuration files it will show a browser dialog window asking for the path to the folder where the correct configuration files reside.


Couldn't find configurationfiles. Please browse to folder containing the configuration. Identifier: 0123456789ABCDEF1337A55A
Computer ▸ OSDisk (C:) ▸ Data Stream Example Config Files
Search Data Stream Example ...
Organize
New folder
Favorites
Libraries
Computer
OSDisk (C:)
Network
Name
Date modified
Type
Size
No items match your search.
Folder: Data Stream Example Config Files
Select Folder
Cancel

After selecting a folder, the folder will be APPENDED to the Auto-Configuration search path.

Tip: To reset the search path and select a new single folder as the search path, click the link on the **Autodetect protocols** option text.



Data Visualizer will then pop up a browser dialog asking for the path to the folder where the configuration files reside. The original search path will be CLEARED and the newly selected folder will be set as search path.

Important: All three configuration files must reside in the same folder.

3. Visualization

Incoming data can be visualized using the modules contained under this section.

The **Terminal** displays data as text, either as raw values or ASCII encoded characters. It is also capable of sending text-based data.

The **Graph** module visualizes incoming data over time as plots, bands, and string flags. Cursor helps analyze the data, and can provide output values for setting thresholds.

The **Oscilloscope** module is helpful for analyzing time-repeating patterns in a data stream.

The **Power Analysis** module is made specifically for analyzing power consumption over time. It can also be used with code profiling to visualize Program Counter samples to get an overview of the program execution versus power consumption.

The **Custom Dashboard** module is a customizable canvas to create user interfaces matching the application. It features the most common user inputs such as buttons, sliders, and check-boxes, in addition to graphing, etc.

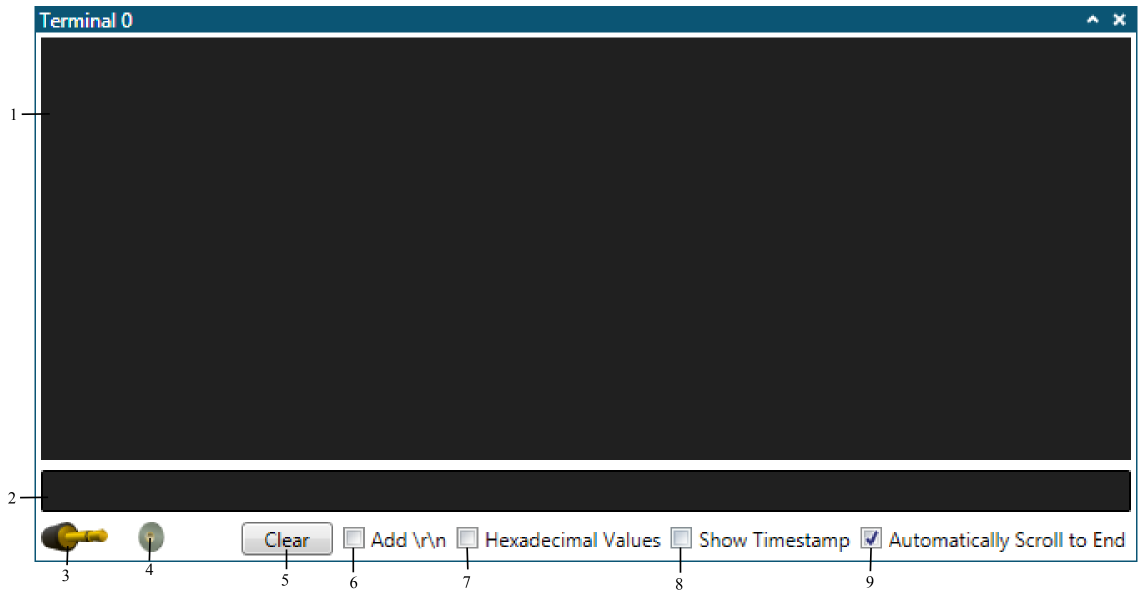
3.1 Terminal

The **Terminal** module is a raw terminal for displaying and sending simple text or numeric values.

3.1.1 Terminal Module

The **Terminal** module is used to display and send simple text or numeric values. For an example on how to configure a terminal, see Terminal Configuration Example.

Figure 3-1. Terminal



1. **Input text box**. 2. **Output text box**. 3. Output **source**. 4. Input **sink**. 5. **Clear** button. 6. **Automatic line feed** checkbox. 7. **Hexadecimal mode** checkbox. 8. **Display timestamp** checkbox. 9. **Autoscroll** checkbox.

3.1.1.1 Connecting the Terminal and Displaying Data

Data streams are connected to the terminal through the **sink** and **source** endpoints. Drop an external **source** onto the terminal **sink**, or drag and drop the terminal **source** onto an external **sink**. Data coming into the terminal's **sink** endpoint will be presented in the **input text box**.

3.1.1.2 Sending Data

When the **source** of the terminal has been connected to a **sink** endpoint, data can be sent by typing data in the **input text box** and pressing enter. Whatever was typed in the text box will be cleared after transmission. The text box supports the use of break characters (e.g. \x55, which will result in the raw value 0x55 being transmitted).

3.1.1.3 Setting Hexadecimal Mode

Data is normally assumed to be an ASCII encoded stream of data. To display the hexadecimal value of the data, select the **Hexadecimal mode** checkbox.

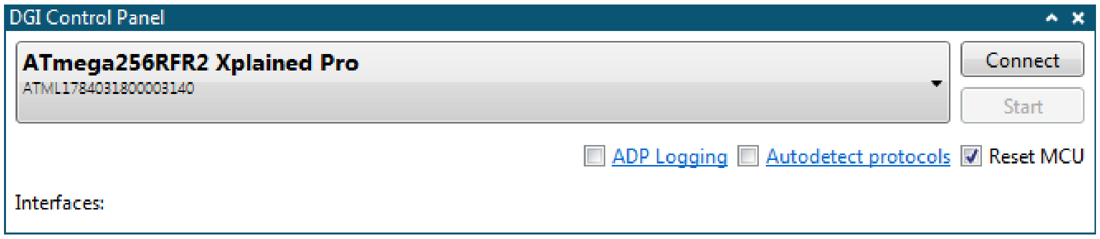
3.1.1.4 Resizing the Input Text Box

The **input text box** is re-sizable by clicking and dragging the lower part of the box.

3.1.2 Terminal Configuration Example

The following example shows how to connect the SPI interface to a terminal. However, the procedure is the same for any of the other available data sources. The target code used in this example can be found in Terminal Example Code.

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

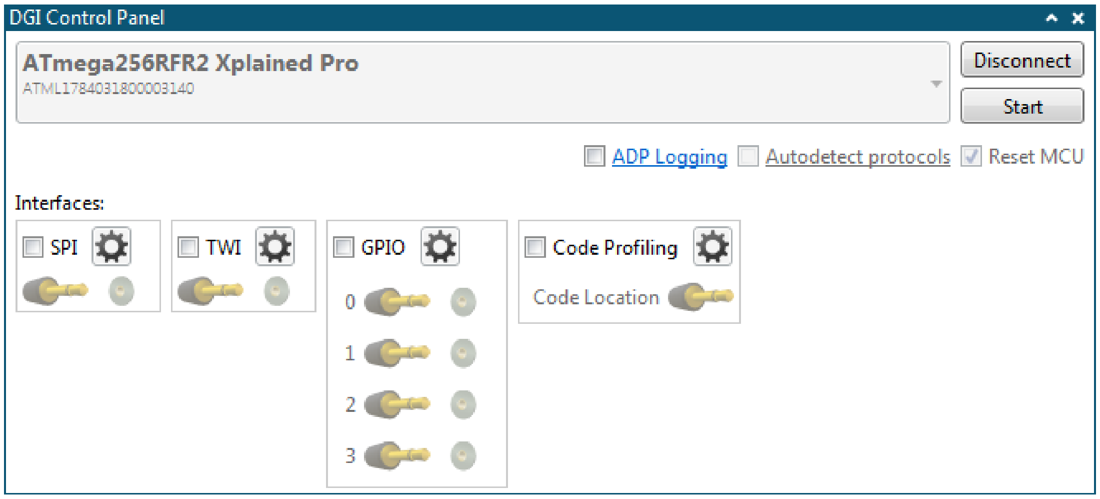

DGI Control Panel

ATmega256RFR2 Xplained Pro
ATML1784031800003140

☐ ADP Logging ☐ Autodetect protocols ☑ Reset MCU

Interfaces:

To do: Click **Connect** to make a connection to the DGI on the selected tool.


DGI Control Panel

ATmega256RFR2 Xplained Pro
ATML1784031800003140

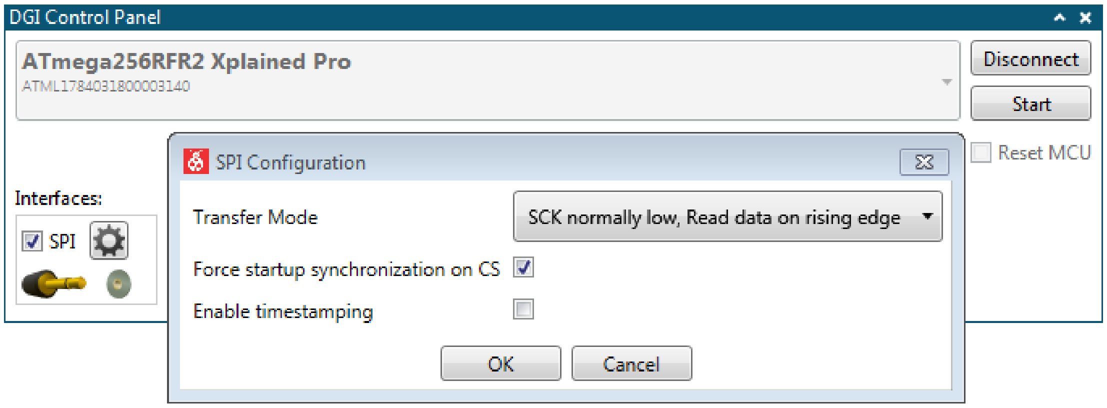
☐ ADP Logging ☐ Autodetect protocols ☑ Reset MCU

Interfaces:

☐ SPI ☐ TWI ☐ GPIO ☐ Code Profiling

To do:

- Click the **SPI** checkbox
- Open the **SPI Configuration** dialog by clicking the Gear button next to the **SPI** checkbox



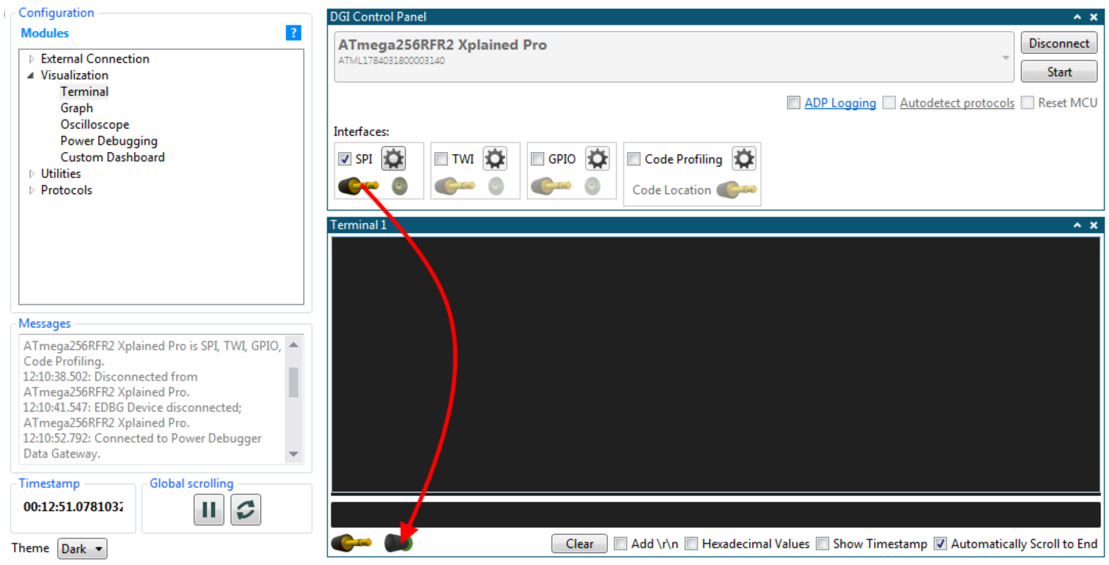
Transfer Mode: SCK normally low, Read data on rising edge
Force startup synchronization on CS: ✓
Enable timestamping: ☐

To do:

- Set **Transfer Mode** to **SCK normally low, Read data on rising edge**
- Enable the **Force start-up synchronization on CS** option

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

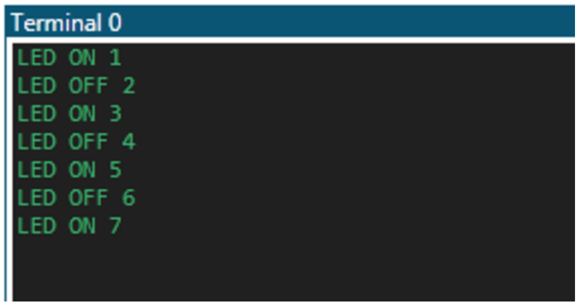


Messages:
```
ATmega256RFR2 Xplained Pro is SPI, TWI, GPIO,
Code Profiling.
12:10:38.502: Disconnected from
ATmega256RFR2 Xplained Pro.
12:10:41.547: EDBG Device disconnected;
ATmega256RFR2 Xplained Pro.
12:10:52.792: Connected to Power Debugger
Data Gateway.
```
Timestamp: 00:12:51.0781032

To do:

- Start the session
- Press the button (SW0) on the Xplained Pro board

On each button press, LED0 on the board should toggle and a message appear on the terminal.



```
LED ON 1
LED OFF 2
LED ON 3
LED OFF 4
LED ON 5
LED OFF 6
LED ON 7
```

Sometimes more than one message appears for each button press. This is an indication that some debouncing algorithm is needed in the button sample routine. It is a lot easier to spot this problem by looking at the terminal output than to watch the LED toggling.

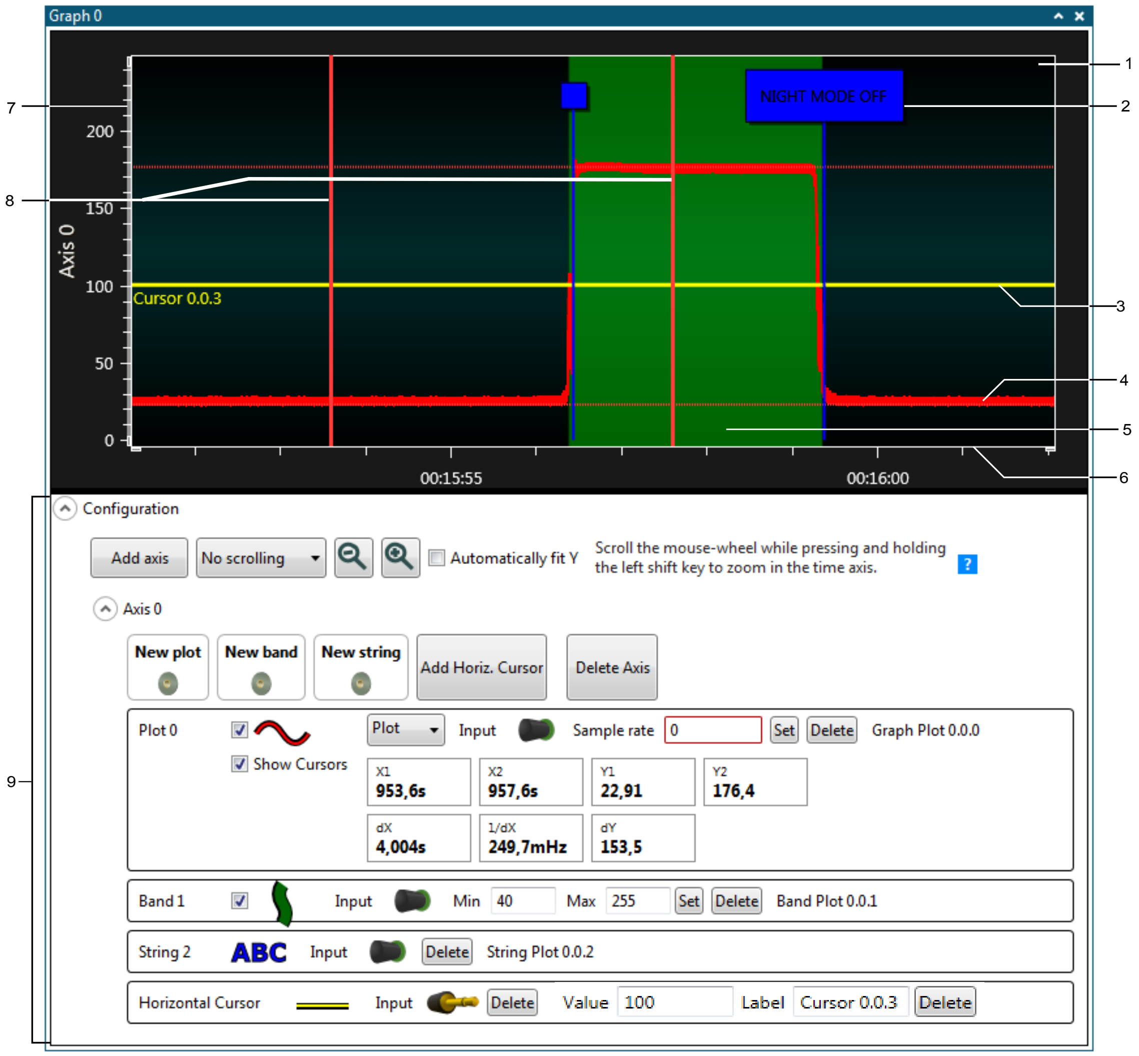
3.2 Graph

The **Graph** module is a versatile graph plotting tool.

3.2.1 Graph Module

The **Graph** module is a versatile graph plotting tool. The large *plot area* has one time axis, and one or more value axes (Y axes). The value axes are stacked on top of each other. For an example on how to configure a graph, see Graph Configuration Example.

Figure 3-2. Graph with a Plot, Band, String Marker, and Cursor



Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

Plot 0 — Plot — Sample rate 0 — Graph Plot 0.0.0

X1	X2	Y1	Y2
953,6s	957,6s	22,91	176,4

dX	1/dX	dY
4,004s	249,7mHz	153,5

Band 1 — Min 40 — Max 255 — Band Plot 0.0.1

String 2 — ABC — String Plot 0.0.2

Horizontal Cursor — Value 100 — Label Cursor 0.0.3

1. Plot area. 2. String marker. 3. Horizontal cursor. 4. Plot. 5. Band. 6. Time axis. 7. Y axis. 8. Plot cursors. 9. Configuration panel.

There are four types of elements that can be added to an Y axis:

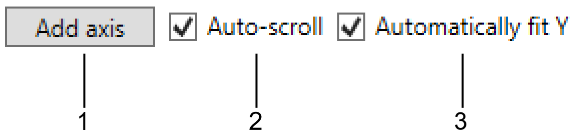
- Plot
- Band
- String marker
- Horizontal cursor

Each of these elements are described in the following sections.

3.2.1.1 Graph Configuration Panel

Through its **Configuration** panel, the **Graph** module is connected to the rest of the system. Here you can add more axes, plots, and other graph elements. Here you will also connect the graph elements by connecting sources and sinks.

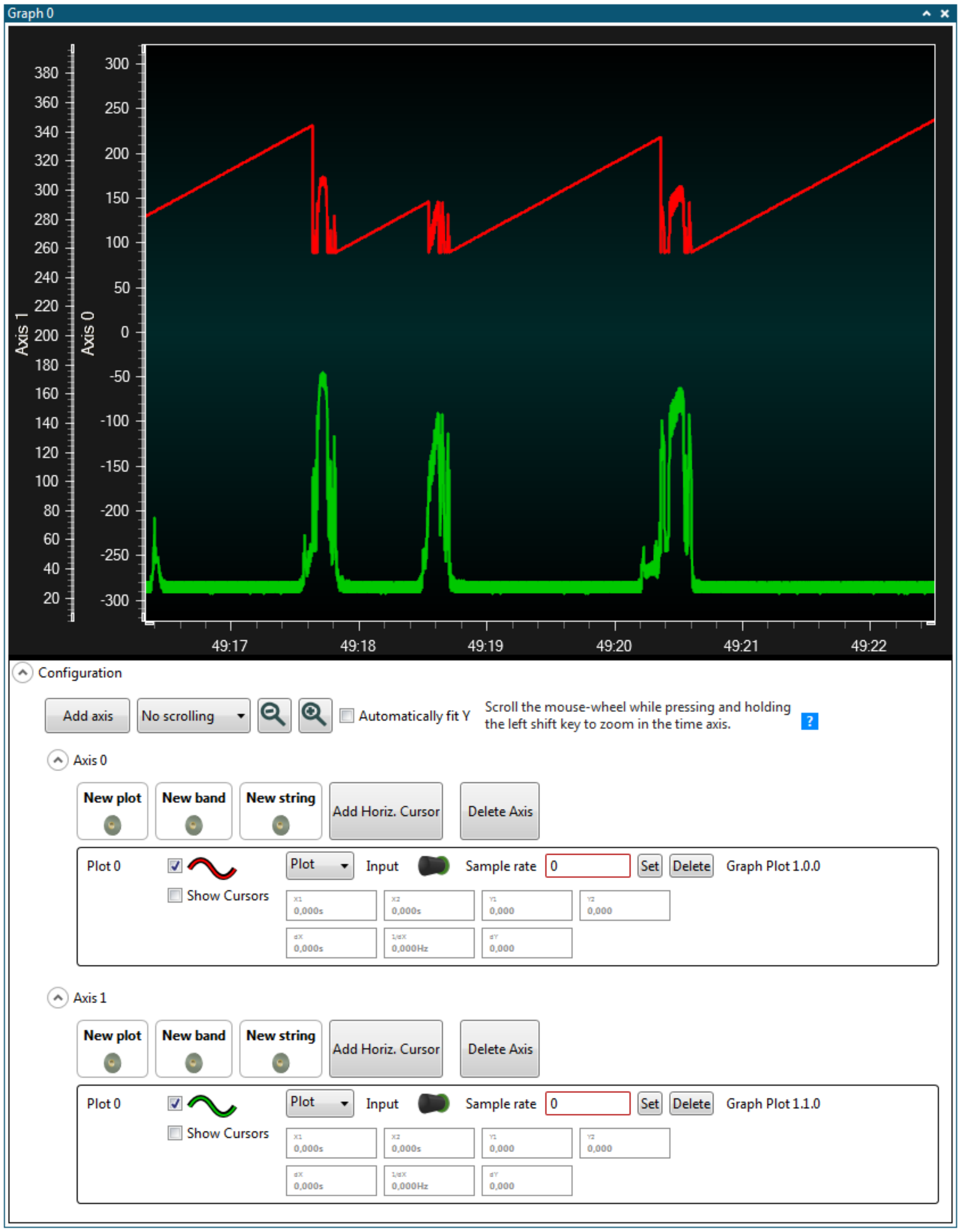
Figure 3-3. Graph Controls



1. **Add axis** button. 2. **Auto-scroll** checkbox. 3. **Automatically fit Y** checkbox.

Add Axes

Figure 3-4. Graph with Two Y Axes



Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

Press the **Add axis** button to add an Y axis to the graph. It will show up in the plot area, and its controls will be added to the bottom of the **Configuration** panel.

Delete Axes

1. If the configuration section for the axis you want to delete is hidden, first expand it by clicking the arrow icon.
2. Delete the axis by pressing its **Delete Axis** button.

Enabling and Disabling Auto-scrolling

Auto-scrolling locks the plot area to include the latest arriving samples. If auto-scrolling is disabled, manually scroll the plot by dragging the time axis with the mouse or with the scroll wheel.

Auto-scrolling is enabled by selecting the **Auto-scroll** check box.

Auto-sizing the Y Axis

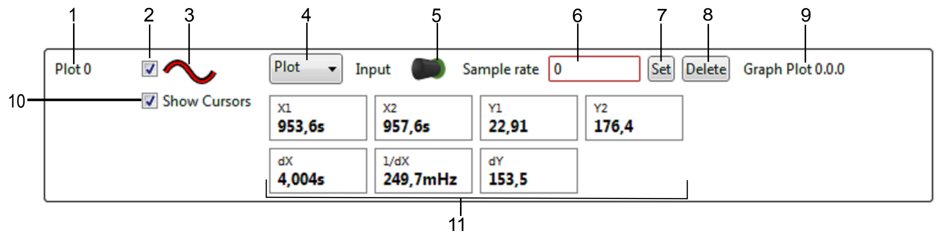
When the **Automatically Fit Y** check box is checked, the Y axis will automatically zoom in or out in order to fit the whole sample range of the plots in that axis.

3.2.1.2 Plot

A *plot* is a curve describing a changing value. The curve is drawn between the data samples it receives from the data **source**. The samples can arrive sporadically, or at a fixed interval. If the data source is known to be sampling at a fixed rate the plot can be set to this sample rate. This way, the curve will be shown correctly even if there are some elasticity in the transmission of the samples. If the samples come at an irregular rate, set the sample rate to 0. This will make the graph position the samples along the time axis according to the sample's timestamp. If there is more than one plot in the graph, each plot will update when new data arrives *for that plot*.

When adding a plot to an axis, the new plot's **Plot control** panel will be placed under that axis in the **Graph configuration** panel.

Figure 3-5. Plot Controls



1. Plot label. 2. **Enable** check box. 3. **Line color** indicator. 4. Plot type selection 5. Data **sink**. 6. **Sample rate** edit box. 7. **Sample rate set** button. 8. **Delete plot** button.1 9. Plot status. 10. **Show Cursors** option. 11. Cursor data.

Adding and Connecting a Plot

To connect a plot to a data **source**, drag the data **source** plug symbol and drop it on the **New plot sink** connector symbol.

Disable a Plot

To stop showing a plot in the graph's plot area, deselect the plot's **Enable** check box.

Change the Plot Color

The plot line's color can be changed:

1. Click on the plot's **line color** indicator.
2. In the dialog box that opens, adjust the color by dragging the **Red**, **Green**, and **Blue** sliders. Press **OK**.
3. The plot line and the **line color** indicator has now changed to the new color.

Change Plot Type
The plot type can be changed between **Plot** and **Points** by changing the selected type in the Plot type selection. The Plot type will show the graph plot as a continuous line while the Points type will show the actual plot samples as dots only.

Plot Data at a Fixed Sample Rate
If the data source sends data to the plot at a fixed rate, the plot's sample rate can be set. Enter the number in the **Sample rate** text box and press the **Set** button.

Plot Timestamped Data
If the data arrives at irregular intervals, the graph will present a more accurate view if the samples are placed using the sample's timestamp.

To plot using timestamps, enter 0 into the **Sample rate** text box and press the **Set** button.

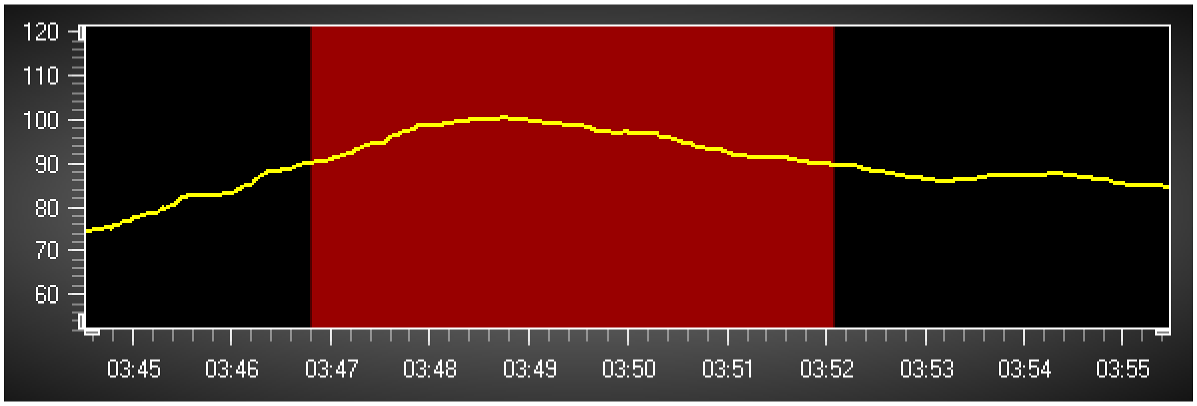
Remove a Plot
To remove a plot from an axis, press the **Delete** button in the plot's control panel.

Cursors
If the **Show Cursors** option is enabled, two vertical cursors will show up in the plot area. The cursors can be moved by the mouse and the **Plot Controls** panel shows data related to the cursors.

3.2.1.3 Band

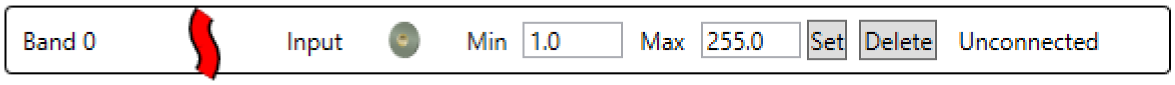
A *band* is a vertical marking in the plot area that highlights the plot background with the band color. For example, on the plot of a temperature reading, a band can be added that highlights portions of the plot where the temperature is above a certain value.

Figure 3-6. Graph Module Showing a Plot and a Band



A band has a minimum and a maximum limit. The band will be active, *on*, if the input to the band is between these two values.

Figure 3-7. Band Controls



Adding and Connecting the Band
To add a new band and connect it to a data **source**, drag the data **source** plug and drop it on the **New band sink** connector.

Setting the Band Color

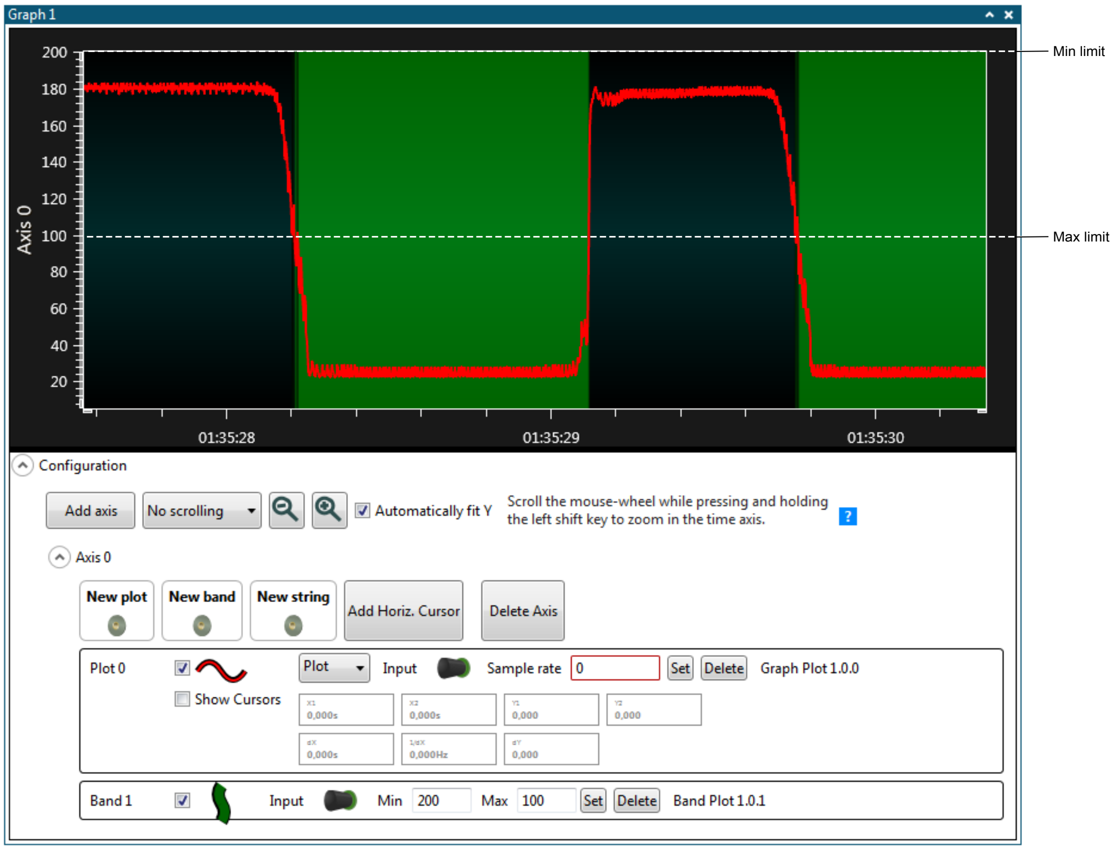
Click the **band color** indicator. A dialog box will open. Change the RGB values, and press OK.

Note:

When changing the band color, the change will not affect band regions already in the graph. Only new band regions will have the new color.

Setting Inverted Band Limits

Figure 3-8. Band with Inverted Behavior



Automatically fit Y ✓

Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

Plot 0 ✓ Plot Input Sample rate 0 Graph Plot 1.0.0

Band 1 ✓ Input Min 200 Max 100 Band Plot 1.0.1

If the maximum limit is set to a value less than the minimum value, the band will behave in an inverted manner. Now, the band will be active when the input value is less than the maximum limit, or if the input value is greater than the minimum limit.

- Enter the minimum and maximum values, and make sure the maximum value is less than the minimum value. Press the **Set** button.

Setting the Band Color

Click the **band color** indicator. A dialog box will open. Change the RGB values, and press OK.

Note:

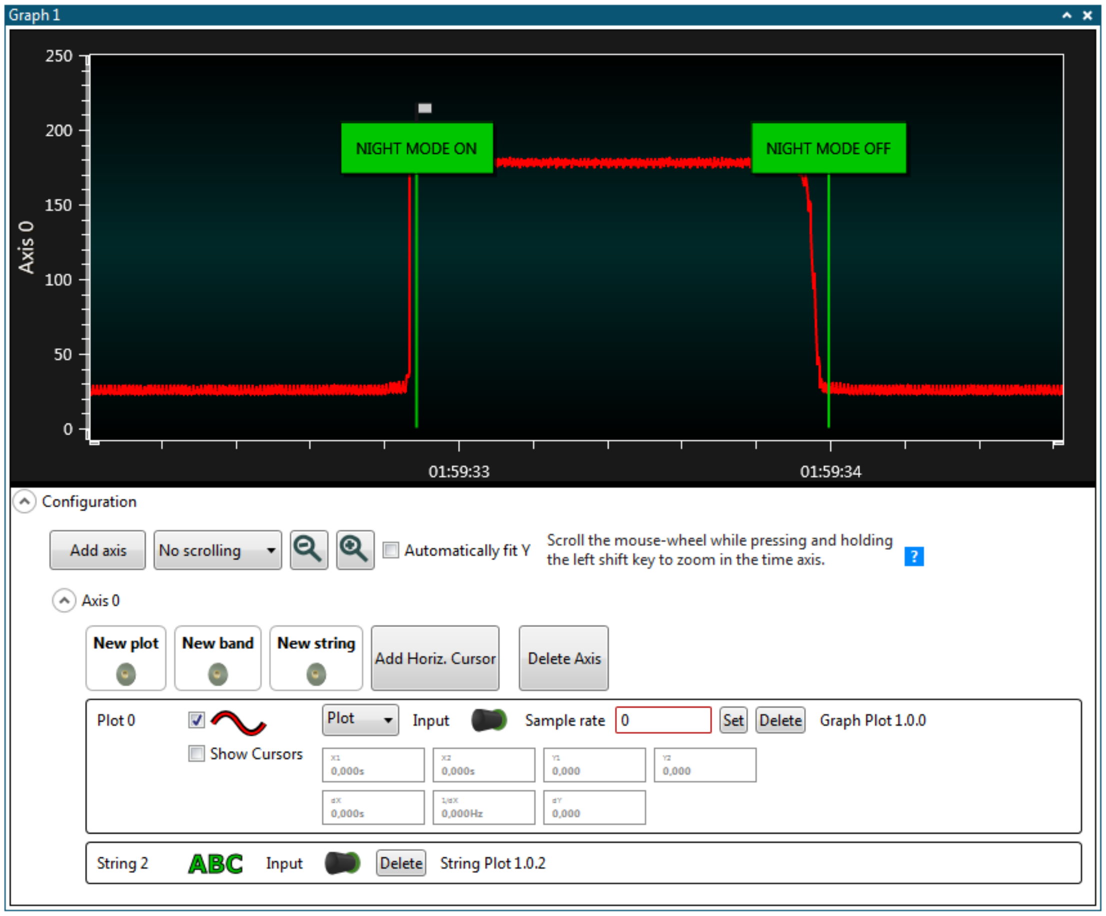
When changing the band color, the change will not affect band regions already in the graph. Only new band regions will have the new color.

Remove a Band
To remove a band, press the **Delete** button in the band's control panel.

3.2.1.4 String Markers

When the **source** sends a string, the *string marker* will attach these short messages to the graph. These markers will be placed according to the timestamp of the sample.

Figure 3-9. Graph with Plot and Two String Markers



Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

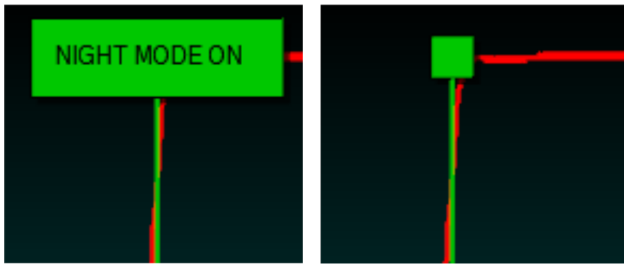
Adding and Connecting a String Marker
To add and connect a string marker to a data **source**, drag the data **source** plug and drop it on the **New string sink** connector.

Setting the String Marker Color
Click on the **string color** indicator in the **String control** panel. Change the RGB values, and press OK.

Note:
When changing the string marker color, the change will not affect string markers already in the graph. Only new string markers will have the new color.

Expanding and Collapsing String Markers
When large strings are sent to a string marker, the marker will collapse into a small box to reduce the space it occupies in the plot area.

Figure 3-10. Expanded and Collapsed String Marker



To see the text, it must be expanded.

- Expand and collapse a string marker by double-clicking the marker

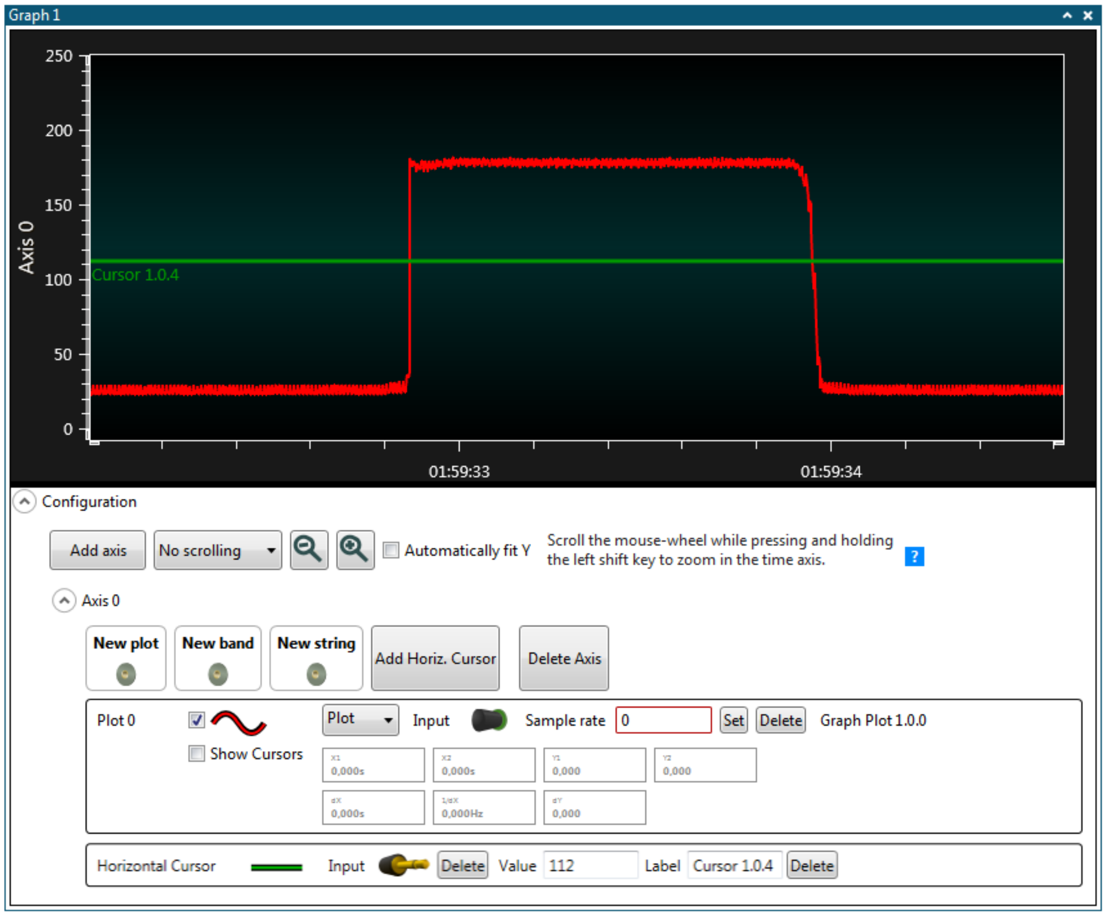
Remove a String Marker
To remove a string marker, press the **Delete** button in the string's control panel.

3.2.1.5 Horizontal Cursor

The **Horizontal cursor** is a horizontal line in the graph that, when dragged up or down, outputs a value that can be used as a source.

Tip: Use the **Horizontal cursor** to control an application's setpoint or threshold.

Figure 3-11. Graph Plot and Cursor



Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

Sample rate 0
Graph Plot 1.0.0
x1 0,000s
x2 0,000s
Y1 0,000
Y2 0,000
dX 0,000s
1/dX 0,000Hz
dY 0,000
Value 112
Label Cursor 1.0.4

Connecting the Cursor
To connect a cursor to a data **sink**, drag the cursor's data **source** plug and drop it on the target's data **sink** connector.

Changing the Cursor Value
To change the cursor value, position the mouse over the cursor line. The mouse cursor will change into a handle. Click and drag the cursor to its new position.

Alternatively, the cursor value can be changed by typing in a new value in the **Value** field in the Horizontal Cursor configuration. Note that the change won't take effect until the text box is deactivated by clicking with the mouse outside the text box.

Changing the Cursor Label
To change the label of the cursor type in a new label in the **Label** field in the Horizontal Cursor Configuration. Note that the change won't take effect until the text box is deactivated by clicking with the mouse outside the text box.

Setting the Cursor Color
Click the **Cursor color** indicator in the **Cursor control** panel. A dialog box will open. Change the RGB values, and press OK.

Remove a Cursor
To remove a cursor, press the **Delete** button in the **cursor control** panel.

3.2.1.6 Zooming and Panning

When the **Auto-scroll** and **Automatically fit Y** check boxes are checked, the last samples will be shown and the Y axis will be zoomed such that all values will be visible.

For manually zooming in or out or inspecting a region in more detail, disable these options and zoom and pan using the mouse.

Zooming the X Axis
The X axis can be zoomed in two different ways:

- Using the mouse scroll wheel
  1.1. Click somewhere inside the plot area.
  1.2. Press and hold the SHIFT key on the keyboard.
  1.3. Scroll the mouse wheel in either direction.

  The X axis will zoom in or out (depending on which way you turned the mouse wheel), centered around the mouse cursor.
- Dragging the X axis resize markers
  2.1. Position the mouse cursor over one of the X axis' resize markers. The mouse cursor will change into horizontal resizing arrows.
  2.2. Click and drag horizontally.

Zooming the Y Axis
The Y axis can be zoomed in two different ways:

- Using the mouse scroll wheel
  1.1. Click somewhere inside the plot area.
  1.2. Press and hold the CTRL key on the keyboard.
  1.3. Scroll the mouse wheel in either direction.

The Y axis will zoom in or out (depending on which way the mouse wheel is turned), centered around the mouse cursor.

- Dragging the X axis resize markers
    2.1. Position the mouse cursor over one of the X axis' resize markers. The mouse cursor will change into vertical resizing arrows.
    2.2. Click and drag vertically.

Panning

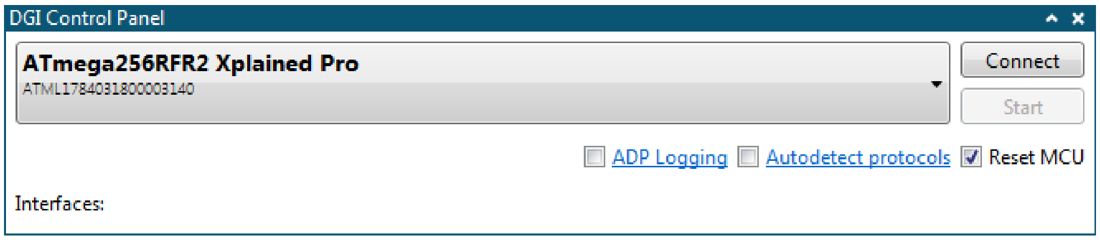
Panning around the graph can be done in two ways:

- Dragging the plot area
    1.1. Position the mouse cursor inside the plot area.
    1.2. Click and hold the left mouse button.
    1.3. Drag the mouse.
- Dragging the axes
    2.1. Position the mouse cursor over one of the axes. The cursor will change into a pointing hand.
    2.2. Click and drag the axis.

3.2.2 Graph Configuration Example

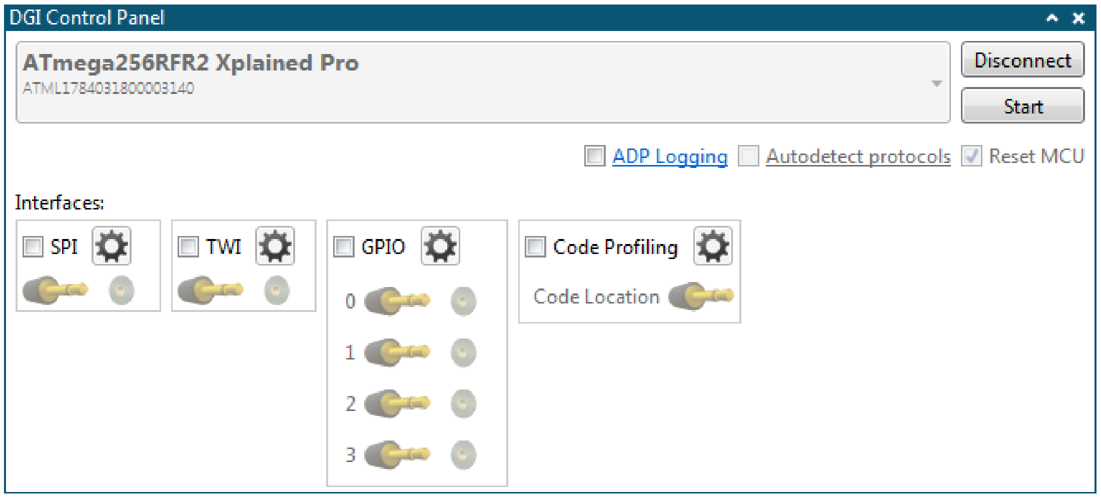
This chapter gives an example on how to configure the **Graph** module to be used with a target application implementing a Night mode switch with a light sensor. Although this example utilizes only some of the data sources available in the Data Visualizer, the procedure will be the same for all data sources. The target code used in this example and a description of the hardware setup can be found in the Graph Example Code chapter. The first part of the configuration example uses the code found in the first subsection of the Graph Example Code chapter (Basic Graph). When changes to the target application code are required as the example progress a link to the corresponding code listing will be provided.

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


DGI Control Panel

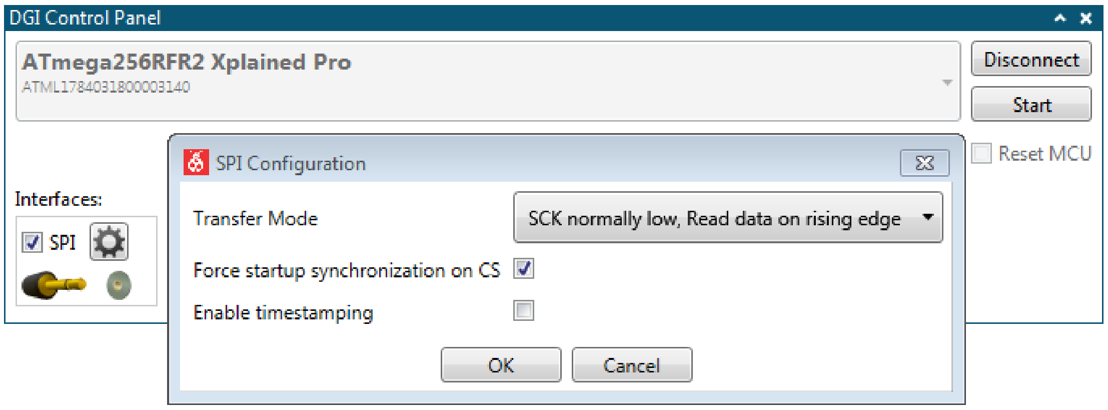
ATmega256RFR2 Xplained Pro
ATML1784031800003140

To do: Click **Connect** to make a connection to the DGI on the selected tool.



To do:

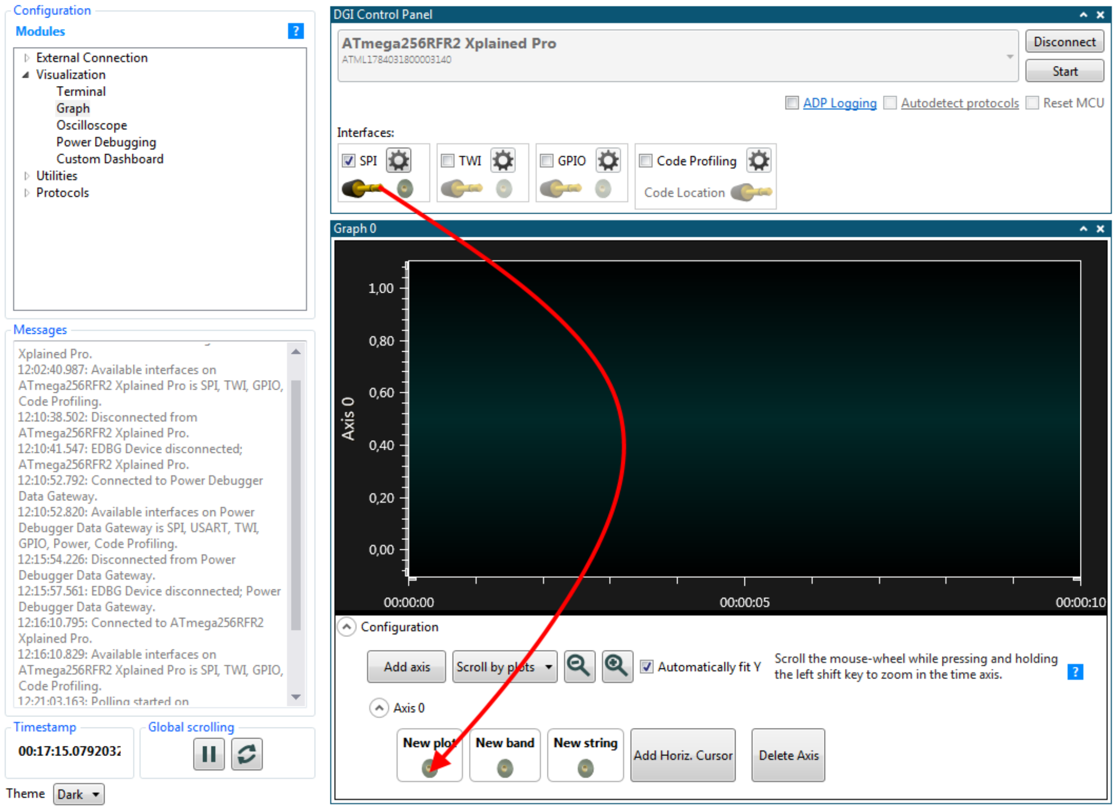
- Click the **SPI** checkbox
- Open the **SPI Configuration** dialog by clicking the Gear button next to the **SPI** checkbox



Transfer Mode: SCK normally low, Read data on rising edge
Force startup synchronization on CS ☑
Enable timestamping ☐

To do:

- Open the configuration panel
- Add a **Graph** module to the Data Visualizer
- Drag the **source** connector from the interface in the **DGI Control Panel** into the **sink** marked **New plot** to make a connection to a new plot



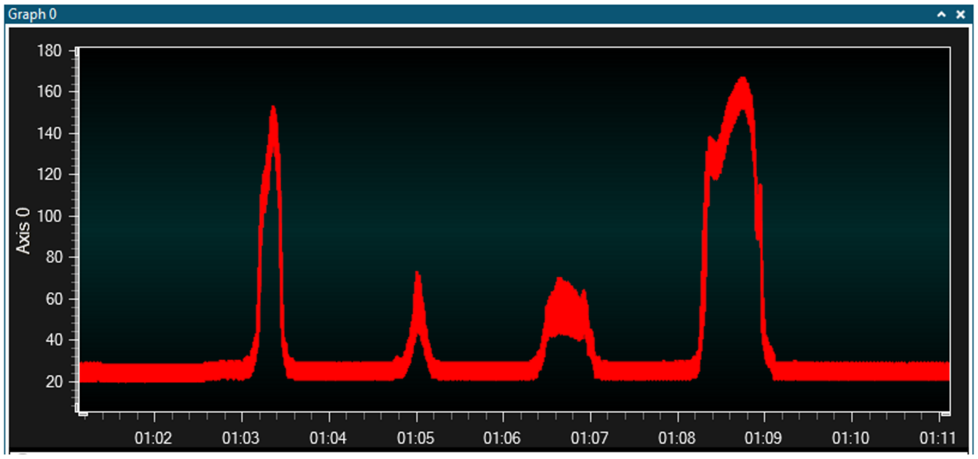
```
Xplained Pro.
12:02:40.987: Available interfaces on
ATmega256RFR2 Xplained Pro is SPI, TWI, GPIO,
Code Profiling.
12:10:38.502: Disconnected from
ATmega256RFR2 Xplained Pro.
12:10:41.547: EDBG Device disconnected;
ATmega256RFR2 Xplained Pro.
12:10:52.792: Connected to Power Debugger
Data Gateway.
12:10:52.820: Available interfaces on Power
Debugger Data Gateway is SPI, USART, TWI,
GPIO, Power, Code Profiling.
12:15:54.226: Disconnected from Power
Debugger Data Gateway.
12:15:57.561: EDBG Device disconnected; Power
Debugger Data Gateway.
12:16:10.795: Connected to ATmega256RFR2
Xplained Pro.
12:16:10.829: Available interfaces on
ATmega256RFR2 Xplained Pro is SPI, TWI, GPIO,
Code Profiling.
12:21:03.163: Polling started on
```

Timestamp: 00:17:15.0792032

To do:

- Push **Start** in **DGI Control Panel**

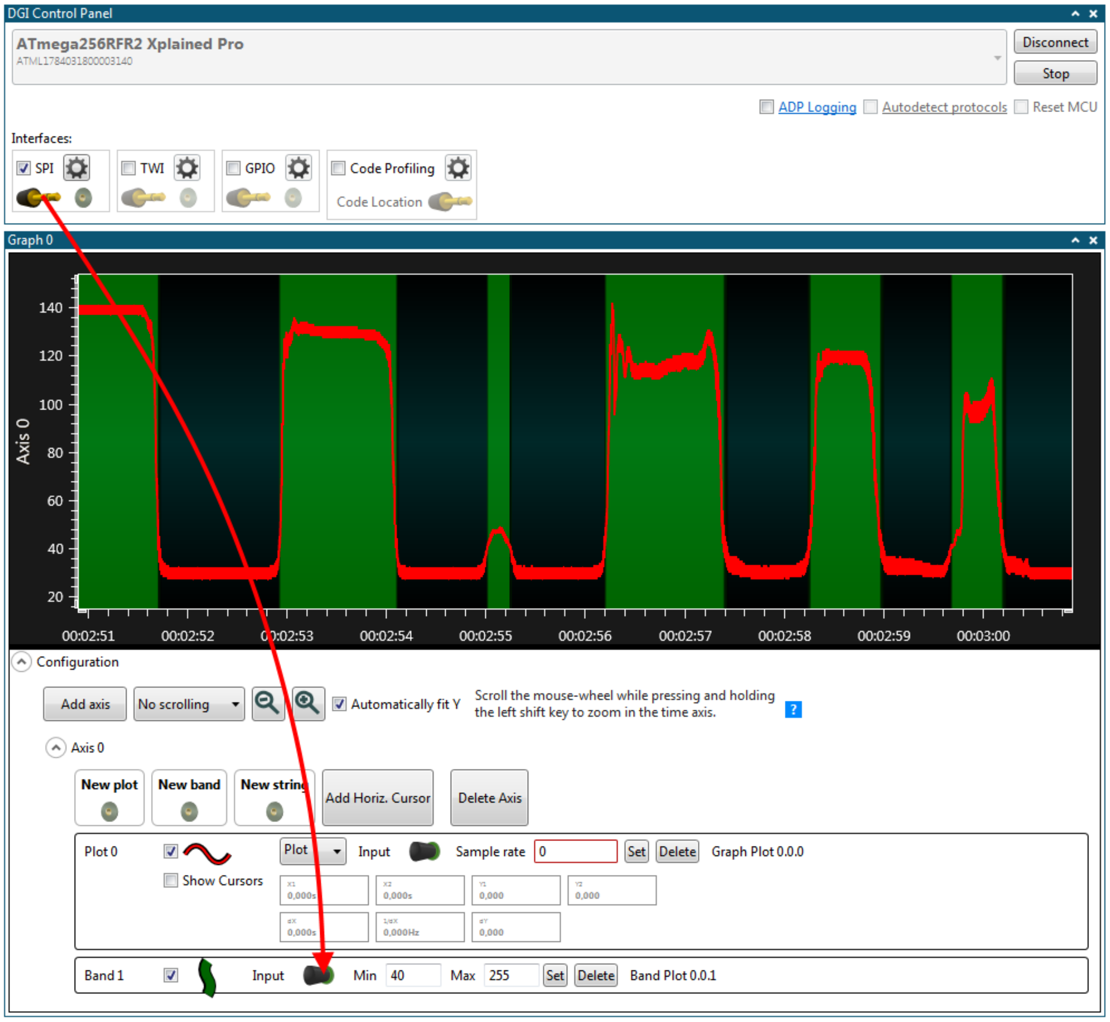
The data will be plotted in the Graph module. It could look something like the picture below when hovering a hand above the light sensor.



The light sensor data can be used to switch between Day and Night mode. For the Night mode switch to be useful, the threshold when switching between the modes are important. The **Graph** module contains a useful feature called **Band** to mark when the plot data is within a certain range. This can be used to simplify the selection of the mode switch threshold.

To do:

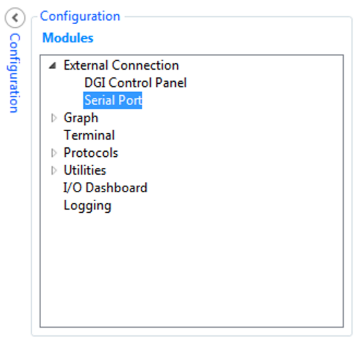
- Drag the interface **source** to the **New band sink**



To see that the Night mode switch is actually working and switching at the right threshold, the string marker feature of the **Graph** module is useful. In this example, the CDC USART interface of the target board is used to send a string each time the mode is switched. These messages can then be shown in the graph as string markers. The target application source code for this part of the configuration example can be found in Adding String Markers.

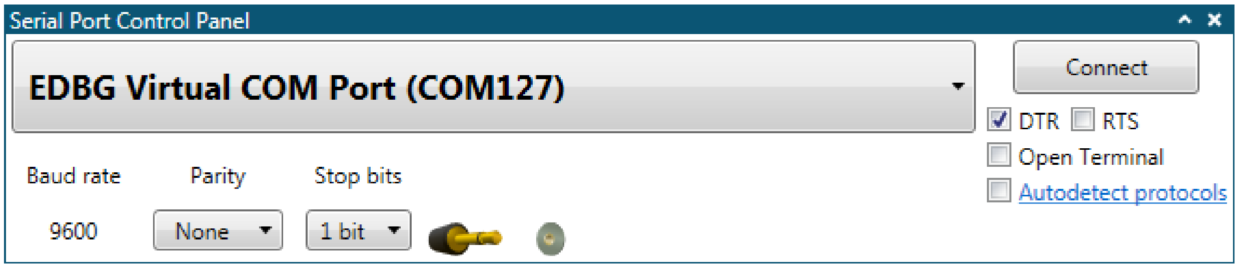
To do:

- Open the **Serial Port Control** panel found under **External Connection** in the **Modules** section of the **Configuration** tab in Data Visualizer



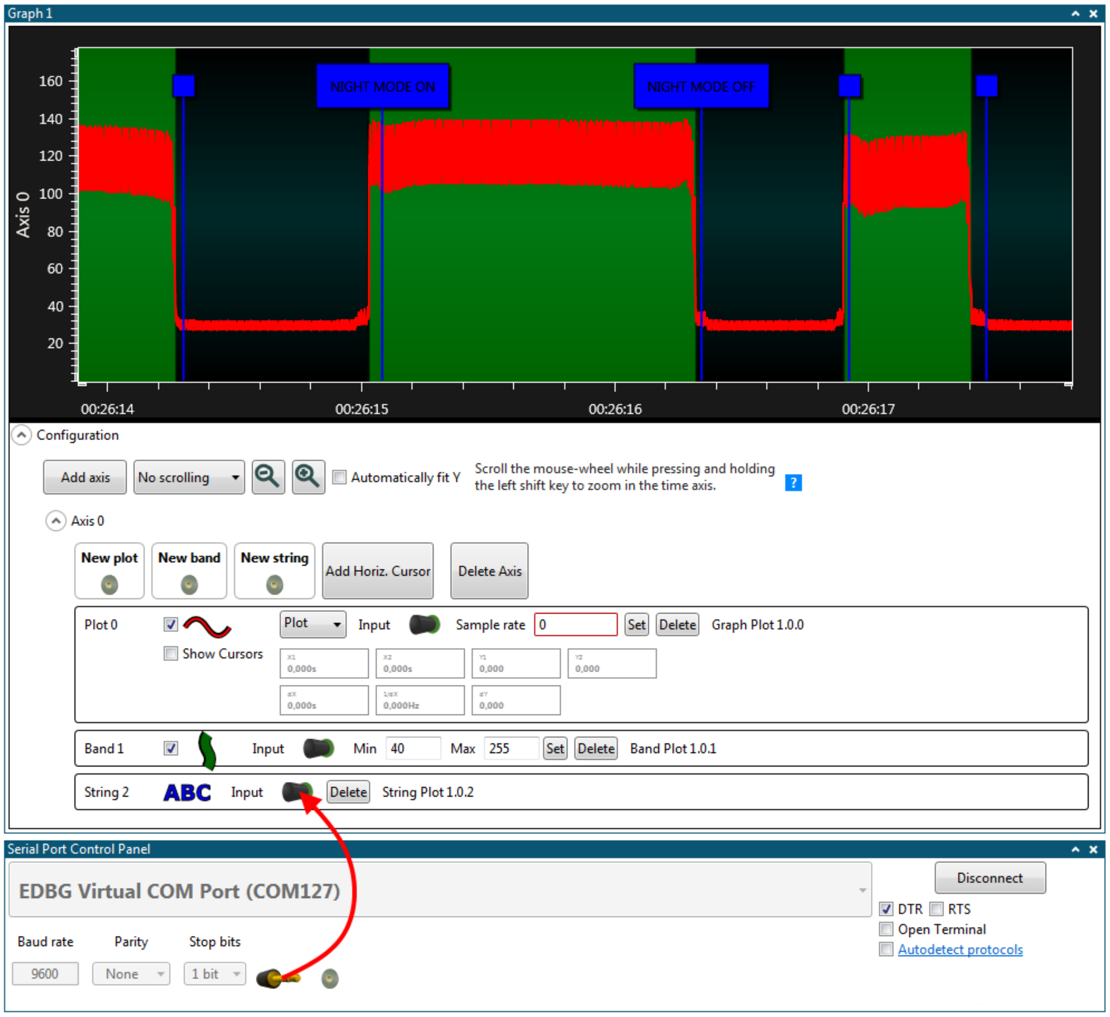
To do:

- Select the correct COM port corresponding to the connected kit
- Set the serial port parameters according to the application code
- Make sure the **Open Terminal** option is not checked



To do:

- Drag the serial port **source** to the **New string sink**
- Click **Connect** in the **Serial Port Control Panel**



Scroll the mouse-wheel while pressing and holding the left shift key to zoom in the time axis.

Plot 0 ✓ Plot Input Sample rate 0 Graph Plot 1.0.0

Band 1 ✓ Input Min 40 Max 255 Band Plot 1.0.1

String 2 Input String Plot 1.0.2

EDBG Virtual COM Port (COM127)

Baud rate 9600 Parity None Stop bits 1 bit

✓ DTR ☐ RTS

String markers will appear as vertical lines with a square on top. By double-clicking the square the string text will be shown. Note that there will naturally be some delay from the ADC data values crosses the threshold until the string message reaches the host computer. In addition, the timestamping of the data is added on the host computer and the two serial interfaces are not synchronized. This results in a misalignment of the string markers compared to the ADC values. DGI includes timestamping functionality on the EDBG on the Xplained Pro and this can be enabled in the **DGI Control Panel** at a performance cost, but CDC includes no time stamping functionality.

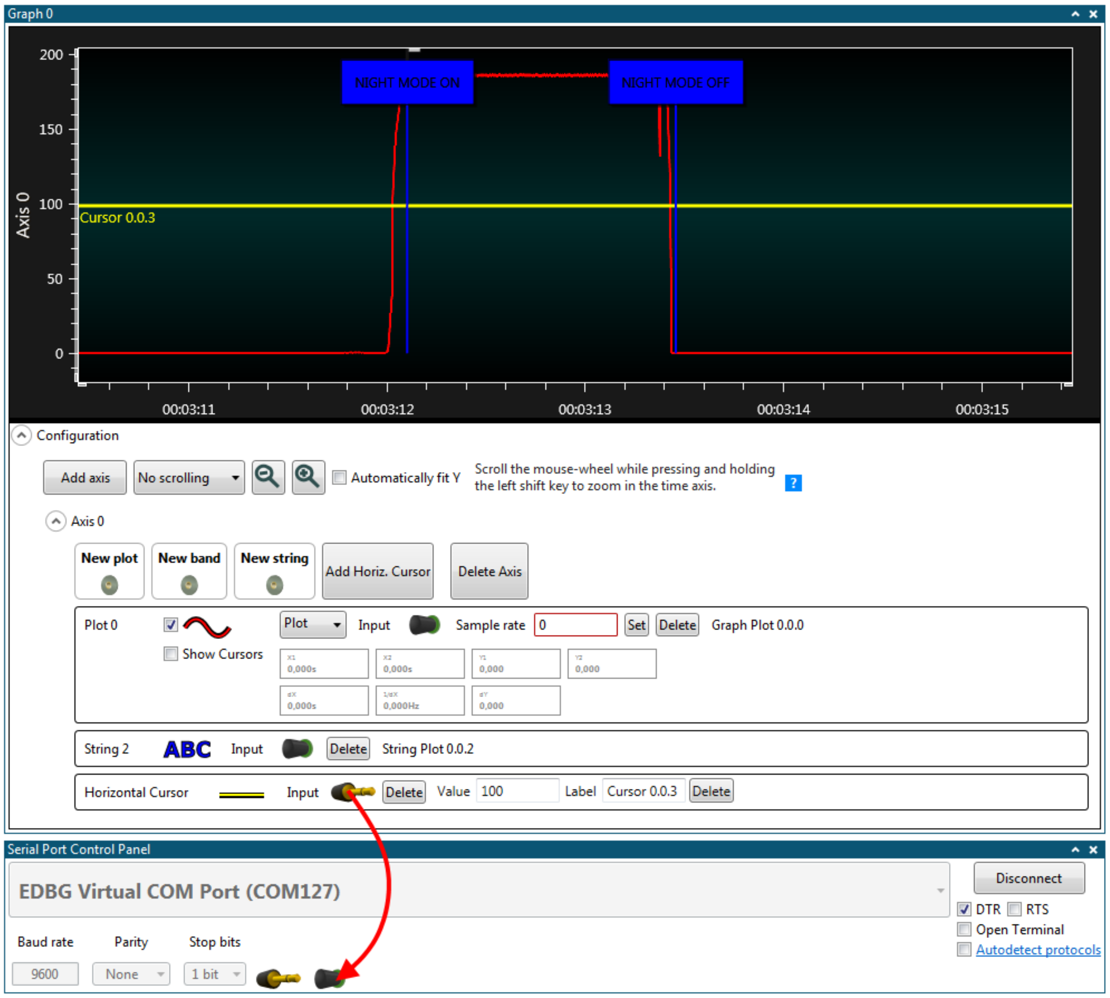
Tip: In this example, a separate serial interface was used for the string marker data. If the number of serial interfaces available are constrained, the same interface could be used to stream both the ADC data and the string marker data by using the Atmel Data Protocol (ADP). For more information, see the Atmel Data Protocol.

So far, the **Graph** module of the Data Visualizer has been used to show the data generated by the light sensor and to show when the Night mode switch toggles between the two modes. The **Graph** module can also be used to interact with the target application while it is running. In this example, the Night mode threshold can be adjusted dynamically by using a horizontal cursor.

The target application source code for this part of the configuration example can be found in Using Horizontal Cursor Code.

To do:

- First, remove the band from the graph as it is of no use when the Night mode threshold is dynamic
- Click **Add Horiz. Cursor** to add a horizontal cursor to the graph
- Drag the **Horizontal Cursor source** to the **sink** in the **Serial Port Control Panel**



To move the horizontal cursor either drag it or change the value by typing a new value in the **Value** field in the configuration. Note that the value will not be updated until the **Value** text box is not in focus, i.e. click somewhere else in the GUI after typing in a value. Every time the cursor is moved the Data Visualizer sends a new float value to the serial port the cursor is connected to.

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

3.3 Oscilloscope

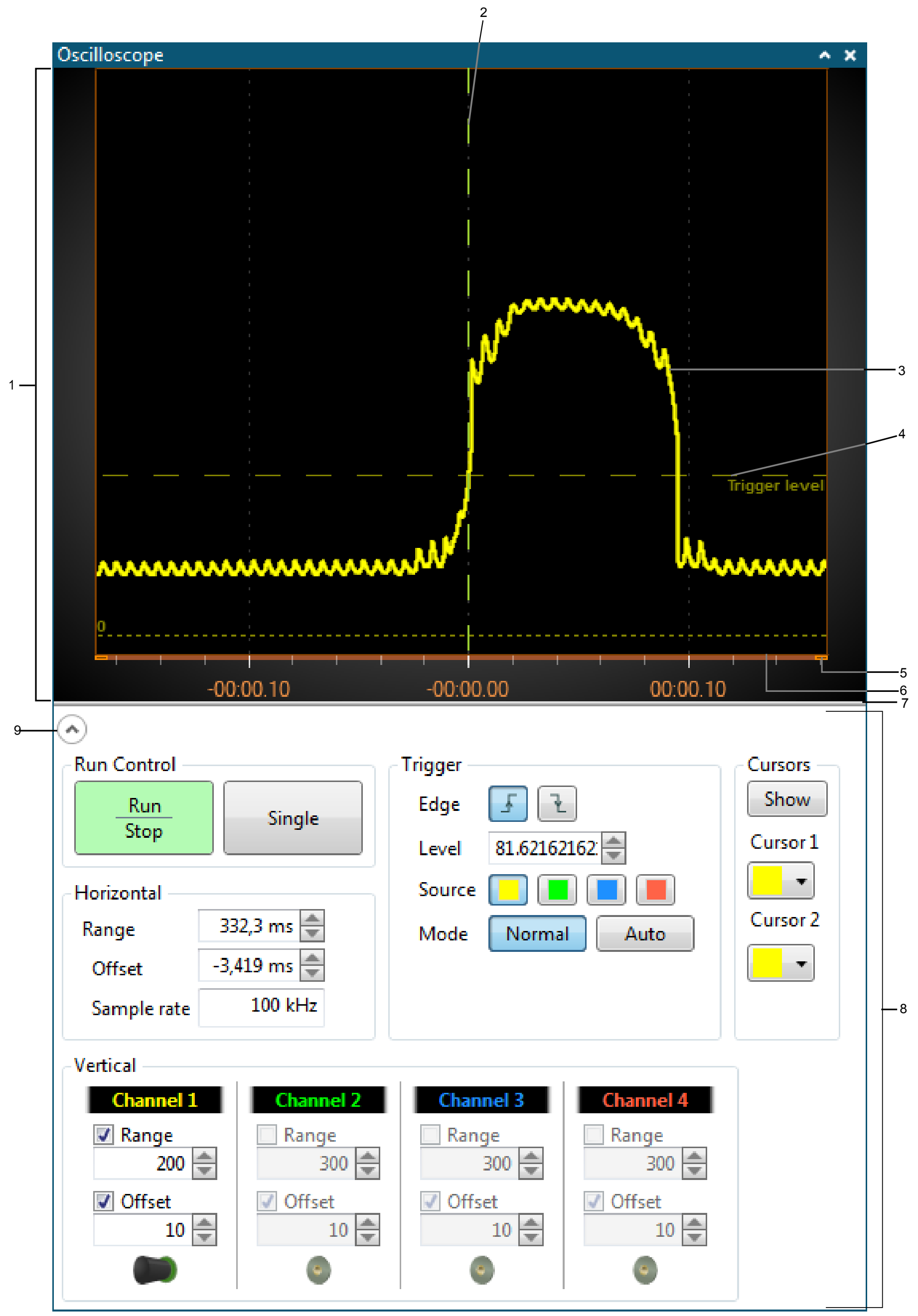
The **Oscilloscope** module visualizes data values in real time. The oscilloscope features a trigger sub-module to capture repeating signals or rare events. The oscilloscope also has a cursor system to measure various properties of the data streams.

3.3.1 Oscilloscope Module

The **Oscilloscope** module visualizes data values in real time. It has four channels for monitoring four different data streams at the same time. Each channel's data stream is visualized as a graph in the plot area, each with a different color. The vertical position and amplitude of each channel can be modified. For repeating signals, or for capturing rare events, the oscilloscope has a trigger sub-module. The oscilloscope also has a cursor system to measure various properties of the data streams.

For an example on how to configure an oscilloscope, see Oscilloscope Configuration Example.

Figure 3-12. The Oscilloscope Module

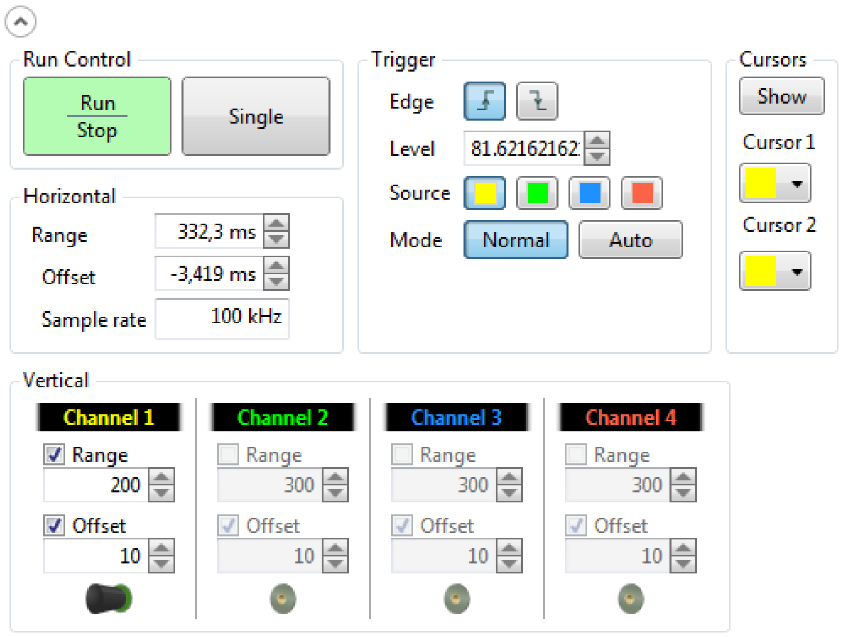


1. Plot area. 2. Zero-line. 3. Plot. 4. **Trigger level** indicator line. 5. Time axis scale handle. 6. Time axis. 7. Plot area resize handle. 8. **Control** panel. 9. Show/hide control area arrow.

3.3.1.1 Oscilloscope Control Panel

The **Oscilloscope control panel** is where the oscilloscope is configured and connected to the rest of the system. The control panel has five sections, which are described in detail in the following sections.

Figure 3-13. Oscilloscope Control Panel



Vertical Controls
The vertical control section has four sub-sections, one for each of the four oscilloscope channels. The channel controls are disabled until a **source** is connected to the channel **sink**.

Connecting the Oscilloscope
Signals or data streams are connected to the oscilloscope through the channel **sink** endpoints. Drop an external **source** onto the **sink**. When connected, the rest of the channel controls will be enabled.

Adjusting the Channel Amplitude
When a channel is displayed in the plot area, the signal's height is determined by the **channel amplitude** setting.

The amplitude can be adjusted in three different ways:

- Enter an amplitude value into the text box. Deselect the text box to let the new value take effect.
- Click on the up/down arrows located to the right of the text box
- With the mouse cursor positioned over the text box, scroll the mouse wheel to increase/decrease the amplitude setting

Show and Hide a Channel
When a channel is in use, the plot can be hidden from the plot area by clearing the **Amplitude** check box. Click it to show the plot again.

Adjusting the Channel Offset
The channel's vertical position in the plot area can be adjusted with the **Offset** setting. The offset value is the channel's zero-point's distance above the bottom of the plot area.

There are four ways to adjust the offset:

- Enter an offset value into the text box. Deselect the text box to let the new value take effect.
- Click on the up/down arrows located to the right of the text box
- With the mouse cursor positioned over the text box, scroll the mouse wheel to increase/decrease the offset setting
- If the channel's zero line is enabled in the plot area, drag it to a new position

Show and Hide the Zero Line
A zero line is, by default, shown when a channel is enabled by connecting it to a source. The zero line is a horizontal line shown in the plot with the same color as the channel's color. It also has a 0 label on the left end.

The zero line can be shown/hidden by clicking/clearing the **Offset** check box.

Customize the Channel Name
When the oscilloscope module is added to the workspace, the four channels are labeled *Channel 1* to *Channel 4*. The label can be changed, as a reminder of what signal is connected to that channel.

- Click inside the label and type in the new name

Run Control
The **Run/Stop** and **Single** buttons are the run control. These buttons control if the plots are updated or not.

There are three operating modes:

- Stop (the **Run/Stop** button is red)
- Single (the **Single** button is yellow)
- Run (the **Run/Stop** button is green)

Enter Run Mode
When the *stop* or *single* operating mode is active (red or yellow light), enter the *run* mode by clicking the **Run/Stop** button. The button will turn green, and the plots will continuously update according to trigger settings.

Enter Single Mode
When the *stop* or *run* operating mode is active (red or green light), enter the *single* mode by clicking the **Single** button. The button will turn yellow, and the plots will trigger and update only once.

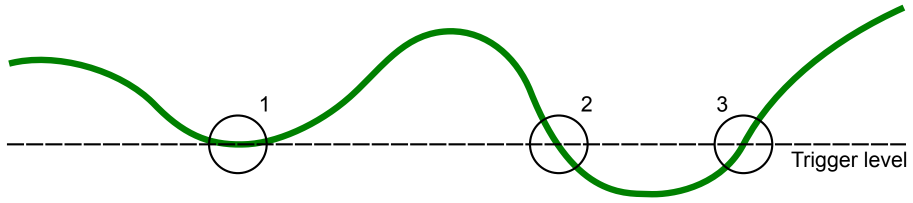
Enter Stop Mode
When the *single* or *run* operating mode is active (yellow or green light), enter the *stop* mode by clicking the **Run/Stop** button. The button will turn red, and the plots will freeze.

Trigger Controls
The **Oscilloscope trigger** sub-module helps to identify and lock on to only the portion of the input signal desired. Depending on the operating mode set by the run controls, the trigger can:

- Lock on to a periodic signal and constantly update the plot
- Only update the plot when the signal exceeds some level

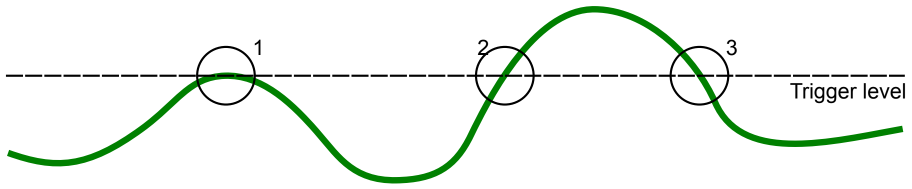
Edge Triggering
The edge triggering mechanism is looking for the signal to *cross* the *trigger level*. For a positive edge trigger, the signal must go from below the trigger level, to above the trigger level.

Figure 3-14. Positive Edge Trigger



1. No trigger – the line must cross. 2. No trigger – wrong direction. 3. Trigger point.

Figure 3-15. Negative Edge Trigger



1. No trigger – the line must cross. 2. No trigger – wrong direction. 3. Trigger point.

Set the Edge Trigger Type

The trigger mechanism has two edge trigger modes: Positive and Negative Edge Trigger.

- To set the Positive Edge Trigger mode, click the button. The button will be highlighted when activated.
- To set the Negative Edge Trigger mode, click the button. The button will be highlighted when activated.

Set the Trigger Level

The trigger level can be adjusted in three different ways:

- Enter a trigger level value into the text box. Press the TAB keyboard button or click the mouse outside the text box to let the new value take effect.
- Click on the up/down arrows located to the right of the text box
- With the mouse cursor positioned over the text box, scroll the mouse wheel to increase/decrease the trigger level setting
- Drag the trigger level line in the plot area to a new position

Select the Trigger Source

The **Oscilloscope trigger** sub-module uses one of the channel signals when looking for the trigger condition.

- Click the colored **Trigger source** button corresponding to the channel chosen for use as a trigger source. The active **Trigger source** button will be highlighted.

Set the Trigger Mode

The **Oscilloscope** module supports both Triggered and Free Running mode.

- Click **Normal** to enable Triggered mode. The plot will only be updated when the trigger condition is satisfied.
- Click **Auto** to enable Free Running mode. The plot will be updated continuously and the trigger conditions will be ignored.

Horizontal Control

The oscilloscope draws the plot lines at a constant speed. The X axis is the time axis. The axis labels show time relative to the trigger point. For the labels to display correctly, the oscilloscope needs to know the sample rate of the source.

Set the Sample Rate

In the sample rate text box, enter the source's sample rate.

Note:

All sources connected to the oscilloscope must have the same sample rate. If not, the plot lines will not be synchronous with the time axis.

Set the Horizontal Resolution

The horizontal resolution determines the time axis range, or what time-span is visible in the plot. It can be adjusted in four different ways:

- Position the mouse cursor inside the plot area. Use the mouse wheel to zoom in or out.
- Enter a resolution value into the text box. Deselect the text box to let the new value take effect.
- Click on the up/down arrows located to the right of the text box
- With the mouse cursor positioned over the text box, scroll the mouse wheel to increase/decrease the resolution setting
- Drag the time axis scale handles to change the resolution

Set the Horizontal Offset

The horizontal offset is the trigger point's position relative to the center of the plot area. Typically, the offset is changed in order to inspect the plot on either side of the trigger point. There are five different ways of changing the offset:

- Position the mouse cursor inside the plot area. Make sure it does not touch any of the trigger line, zero line, or cursor lines. Then, click and drag the mouse horizontally to change the offset.
- Position the mouse cursor on the time axis. Then click and drag the mouse horizontally to change the offset.
- Enter an offset value into the text box. Deselect the text box to let the new value take effect.
- Click on the up/down arrows located to the right of the text box
- With the mouse cursor positioned over the text box, scroll the mouse wheel to increase/decrease the offset setting

3.3.1.2 Cursors

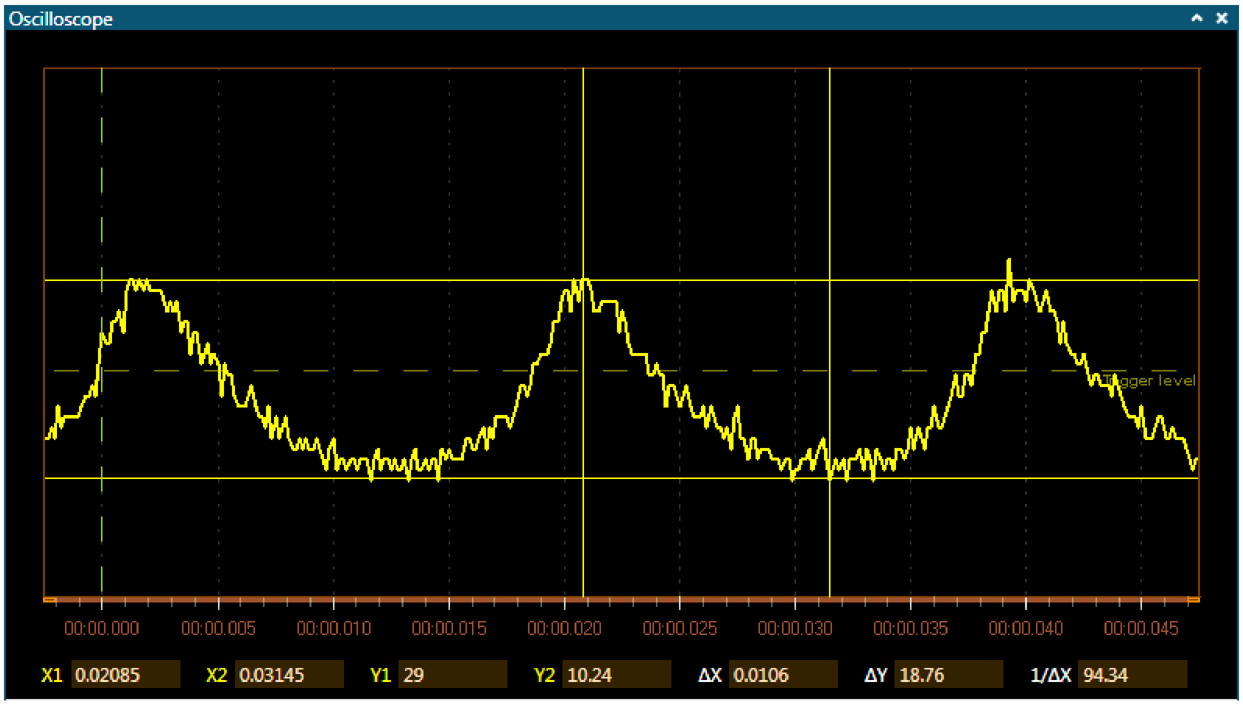
The oscilloscope has two cursors that can be used to inspect the plots. The cursors simplify measurements such as pulse widths, amplitudes, frequencies, and so on.

Each cursor is displayed in the plot area as two lines, one vertical and one horizontal. When the vertical cursor line is moved, the horizontal line will follow so that the plot line, vertical and horizontal cursor lines intersect in the same point.

You can set which channel is the source for each of the cursors.

At the bottom of the plot area is the data line. It displays the X and Y values for each of the cursors. In addition, ΔX, ΔY, and 1/ΔX is calculated and displayed.

Figure 3-16. Oscilloscope Cursors



X1 0.02085 X2 0.03145 Y1 29 Y2 10.24 ΔX 0.0106 ΔY 18.76 1/ΔX 94.34

Show and Hide the Cursors
In the **Cursor** area in the **Oscilloscope control** panel, toggle the **Show** button to show or hide the cursors and the cursor data line in the plot area.

Select Cursor Source Channel
In the **Cursor** group in the oscilloscope's control panel, click on the **Cursor 1** and **Cursor 2** drop-down list boxes to select the channel to use as the source for that channel. Pick the color matching the channel chosen for use.

The cursor's X and Y labels in the data line will change color to match the color of the channel selected.

Move a Cursor
Only the vertical line (the X value) of a cursor can be moved. The horizontal line (the Y value) will follow.

- Position the cursor over the vertical cursor line. The mouse cursor will change into a left/right cursor. Click and drag the cursor to its new position.

After repositioning a cursor, the readouts in the data line are updated.

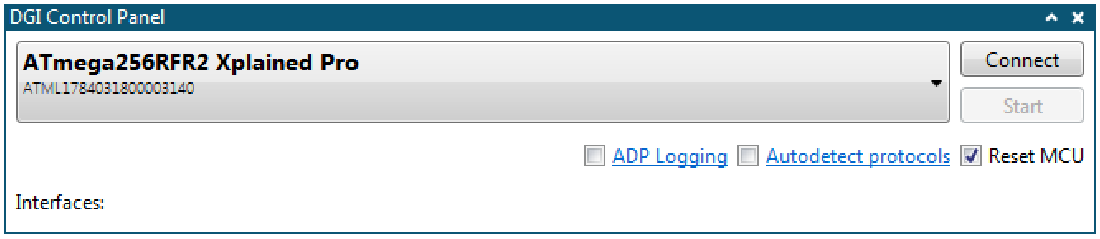
Bring a Cursor Into View
After some zooming and panning, a cursor can end up far outside the visible region. It is easy to bring it back into view:

- Right-click on the **X1** or **Y1** labels in the data line. From the pop-up menu, select **Bring into view**.

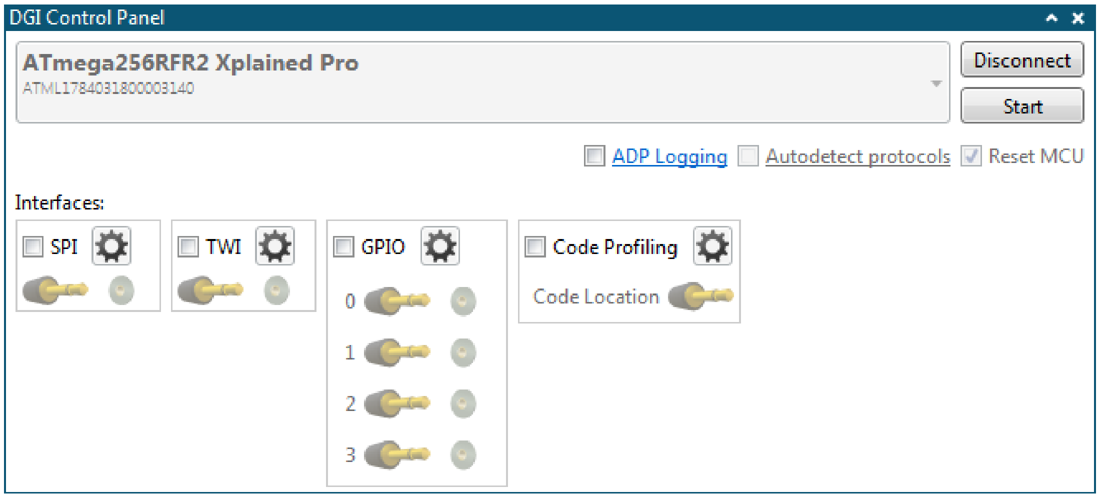
3.3.2 Oscilloscope Configuration Example

This chapter gives an example on how to configure the **Oscilloscope** module to be used with a target application implementing a Night mode switch with a light sensor. Although this example only utilizes the SPI interface as data source, the procedure will be the same for all data sources. The target code used in this example and a description of the hardware setup can be found in Oscilloscope Example Code.

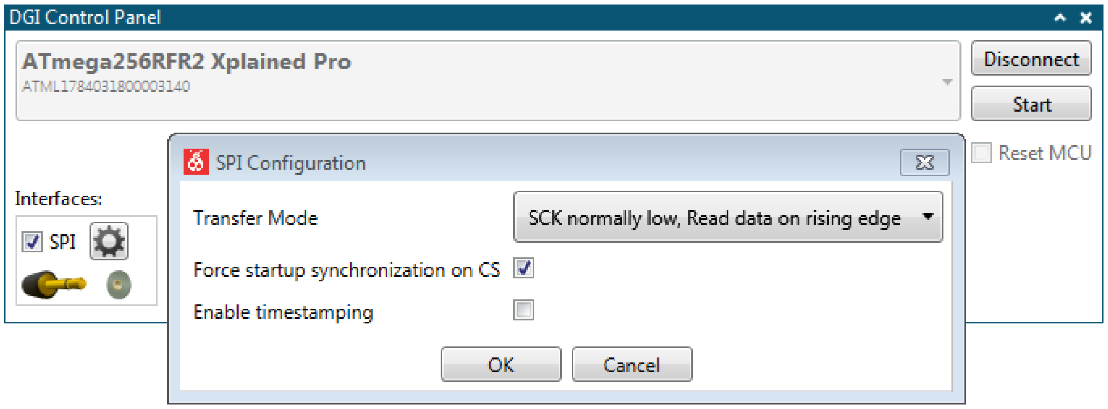
To do: Select correct tool in the **DGI Control Panel**.



To do: Click **Connect** to make a connection to the DGI on the selected tool.



To do:
- Click the **SPI** checkbox
- Open the **SPI Configuration** dialog by clicking the Gear button next to the **SPI** checkbox



SPI Configuration

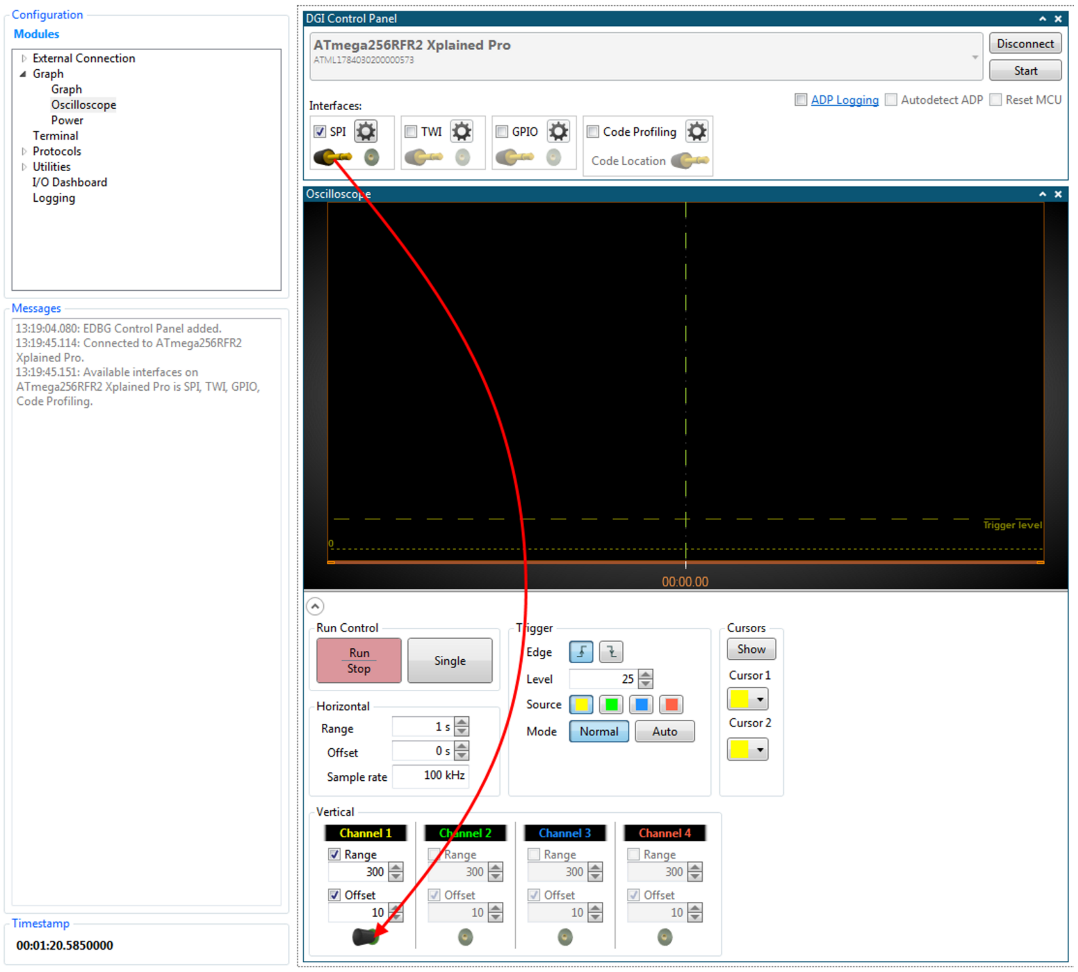
Transfer Mode: SCK normally low, Read data on rising edge

Force startup synchronization on CS: ✓

Enable timestamping: ☐

To do:

- Open the configuration panel
- Add an **Oscilloscope** module to the Data Visualizer
- Drag the **source** connector from the interface in the **DGI Control Panel** into the **sink** for the oscilloscope channel to make a connection



13:19:04.080: EDBG Control Panel added.
13:19:45.114: Connected to ATmega256RFR2 Xplained Pro.
13:19:45.151: Available interfaces on ATmega256RFR2 Xplained Pro is SPI, TWI, GPIO, Code Profiling.

Timestamp: 00:01:20.5850000

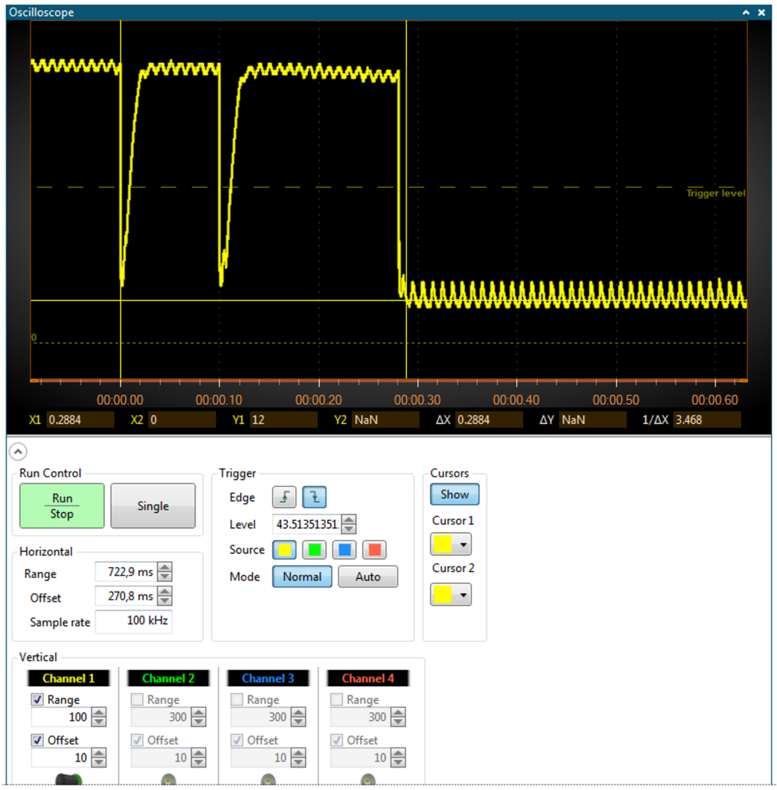
Range: 1 s
Offset: 0 s
Sample rate: 100 kHz
Level: 25
Range: 300
Offset: 10

The **Oscilloscope** module can now be used to analyze the data acquired from the light sensor when toggling a desk lamp ON and OFF above the I/O1 Xplained Pro.

To do:

- Set sample rate to 100 kHz
- Enable **Trigger** on falling **Edge** and set **Mode** to **Normal**
- Push **Start** in the **DGI Control Panel**
- Push the **Run-Stop** button in the **Oscilloscope** module

After some adjustments of the trigger level by dragging it with the mouse in the oscilloscope plot area, and zooming in on the plot by adjusting the **Horizontal** and **Vertical** range, a lamp switch on event could look something like the picture below.



X1 0.2884 X2 0 Y1 12 Y2 NaN ΔX 0.2884 ΔY NaN 1/ΔX 3.468

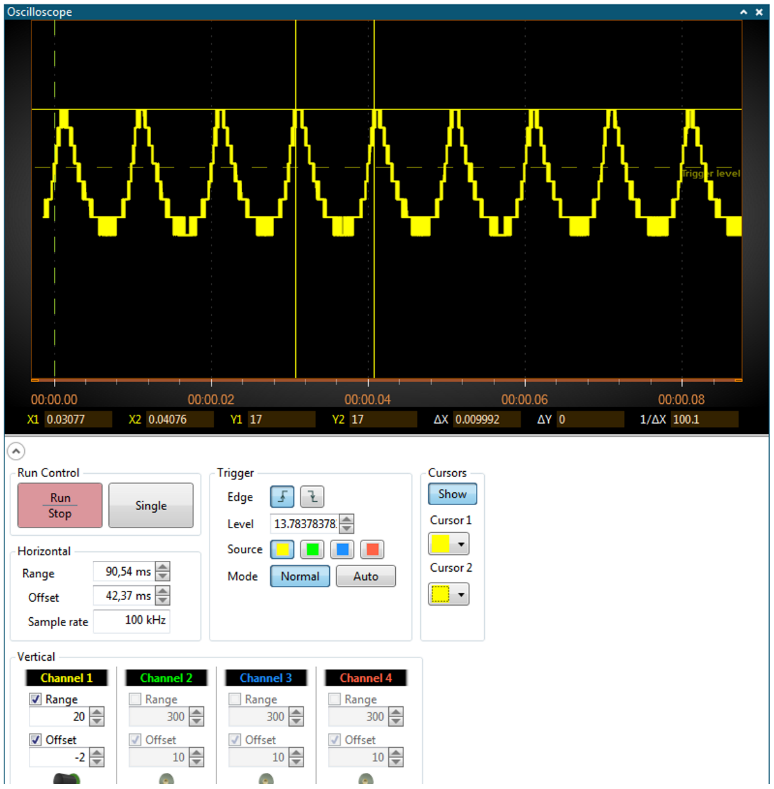
Horizontal: Range 722,9 ms; Offset 270,8 ms; Sample rate 100 kHz

Trigger: Edge; Level 43.51351351; Source; Mode Normal / Auto

Cursor 1; Cursor 2

	Channel 1	Channel 2	Channel 3	Channel 4
Range	☑ 100	☐ 300	☐ 300	☐ 300
Offset	☑ 10	☑ 10	☑ 10	☑ 10

By turning on the **Cursors** it is possible to measure the time it takes for the lamp to settle in the ON state. In this case, it took about 300 ms (**ΔX** in the plot area). Zooming further in on the plot makes it possible to use the cursors to measure the frequency of the light flickering. The **1/ΔX** field in the plot area shows that the frequency is about 100 Hz, which matches well with the 50 Hz AC power of the lamp (the power switches polarity 100 times per second).



X1 0.03077 X2 0.04076 Y1 17 Y2 17 ΔX 0.009992 ΔY 0 1/ΔX 100.1

Run Control

Horizontal
Range 90,54 ms
Offset 42,37 ms
Sample rate 100 kHz

Trigger
Edge
Level 13.78378378
Source 🟨 🟩 🟦 🟥
Mode [✓ Normal] Auto

Cursors
Cursor 1
Cursor 2

Vertical

Channel 1	Channel 2	Channel 3	Channel 4
☑ Range	☐ Range	☐ Range	☐ Range
20	300	300	300
☑ Offset	☑ Offset	☑ Offset	☑ Offset
-2	10	10	10

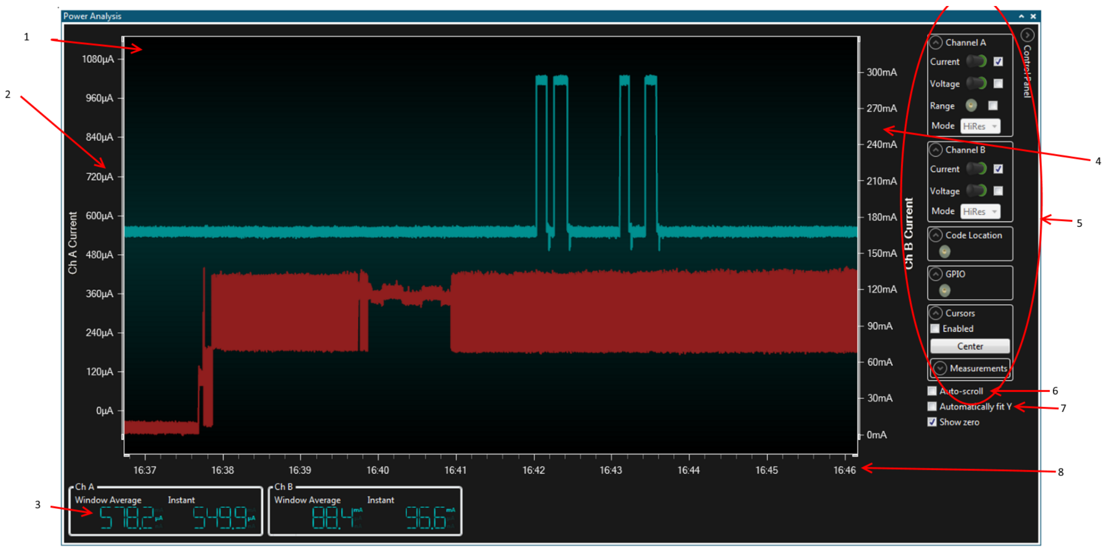
3.4 Power Debugging

The **Power Debugging** module displays current and voltage measurements (commonly referred to as power measurements) generated by the **Power** interface in the **DGI Control Panel**. The power measurements can be combined with various other interfaces like **GPIO** and **Code Profiling** in the same graph to correlate code execution on the target MCU and power consumption of the target application.

3.4.1 Power Debugging Module

The **Power Debugging** module displays the current consumption of a connected kit. To get started with basic current measurements, see the Basic Current Measurement chapter. For an example on how to use cursors, see Power Analysis using Cursors. For examples on how to correlate current consumption with code execution, see Code Correlation.

Figure 3-17. Power Debugging



1. Current consumption graph. 2. Y-axis of channel A. 3. Channel A average and instant values. 4. Y-axis of channel B. 5. **Control Panel**. 6. **Auto-scroll** checkbox. 7. **Automatically fit Y** checkbox. 8. X-axis (time), unit is [minutes]:[seconds].

Important: The **Power** module can only be used with the **Power** interface.

3.4.1.1 Scaling and Scrolling a Graph

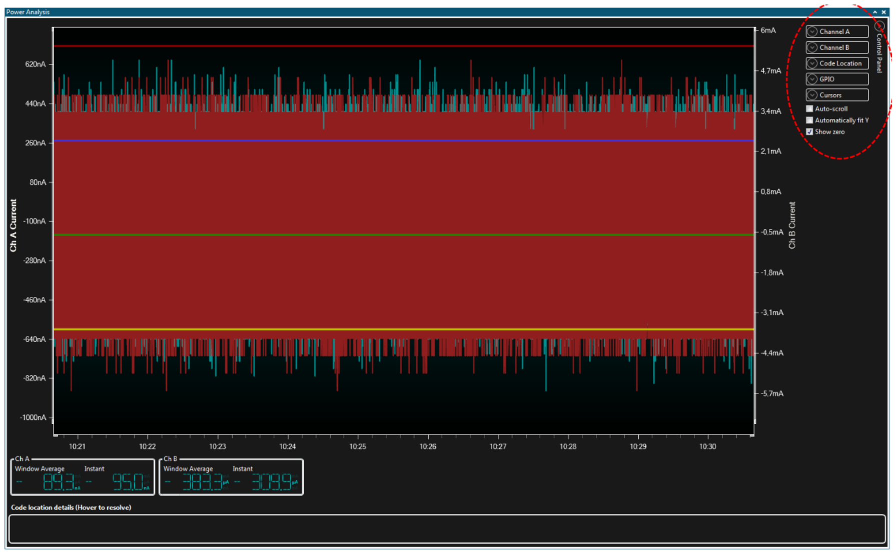
Tip: Turn off **Auto-scroll** and **Automatically fit Y** to more closely examine a plot while it is still running.

Use the **scale** and **offset** controls in order to move the plots as needed. 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

3.4.1.2 Power Debugging Module Control Panel

The **Power Debugging** module **Control Panel** is placed in the upper right corner of the module.



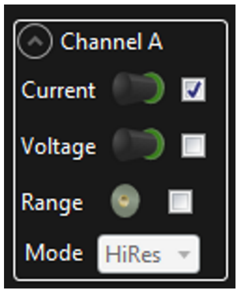
Notice: Not all configuration options in the control panel are available on all tools. For example, only the Power Debugger has both an A channel and a B channel. All options will be visible for all tools, but will have no effect unless the tool supports them.

The **Auto-scroll** option controls the scrolling in the X-axis direction (time axis). To zoom in on and examine the graphs in detail, disable this option.

The **Automatically fit Y** option controls whether the Data Visualizer will automatically adjust the range of the Y axis according to the graph content or not. If this option is enabled, any manual adjustments of the Y axis will be overridden.

The **Show zero** option controls whether the zero-point of the Y axis should always be visible when **Automatically fit Y** is enabled.

Channel Configuration
For each power measurement channel there is a **Channel configuration** section in the **Control Panel** of the **Power Analysis** module.


Channel A
Current [✓]
Voltage [ ]
Range [ ]
Mode HiRes

The channel section allows the user to enable/disable the current and voltage graphs in the **Power Analysis** module.

Notice: If the **Enable B Channel** option in the **Power Configuration** of the **DGI Control Panel** (see Power Interface) is not selected, the B channel will not be available even though the tool has a B channel. But the B channel configuration will still be visible in the **Control Panel**.

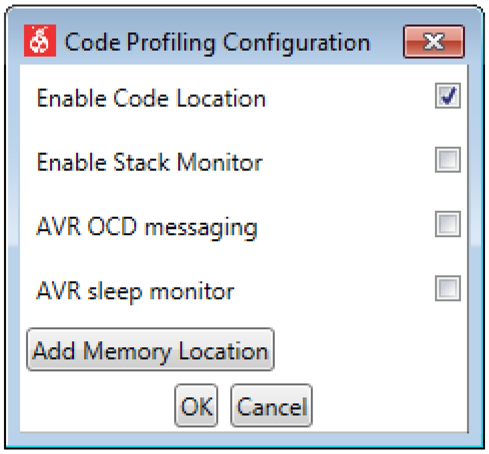
The **Range** setting enables the measurement range data for the current measurement channel. To cover the full range of current values supported by the current measurement channel, most tools have two or more hardware configurations for each channel. The number of ranges for a channel varies with the connected tool. The switching between the hardware configurations is done automatically based on the instant current measured.

Notice: The range graph will only be visible if the **Enable Range Source** option in the **Power Configuration** of the **DGI Control Panel** is selected.

The **Mode** option allows for different averaging algorithms to be used for the display of data if this is enabled for the current tool.

Code Location

The **Code Location** section contains no options, just the source connection. To enable code locations in the **Power Analysis** graph the **Code Profiling** interface in the **DGI control panel** must be enabled and the **Enable Code Location** option in the **Code Profiling Configuration** of the **DGI Control Panel** must be enabled.


Code Profiling Configuration

Enable Code Location [✓]

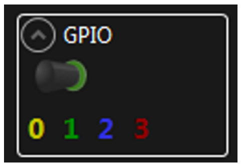
Enable Stack Monitor [ ]

AVR OCD messaging [ ]

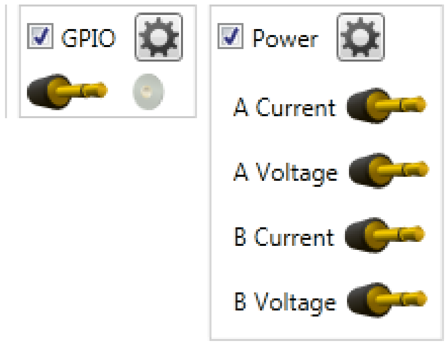
AVR sleep monitor [ ]

GPIO

Each of the GPIO sources can be switched ON or OFF in the **GPIO** section of the **Control Panel** of the **Power Analysis** module.

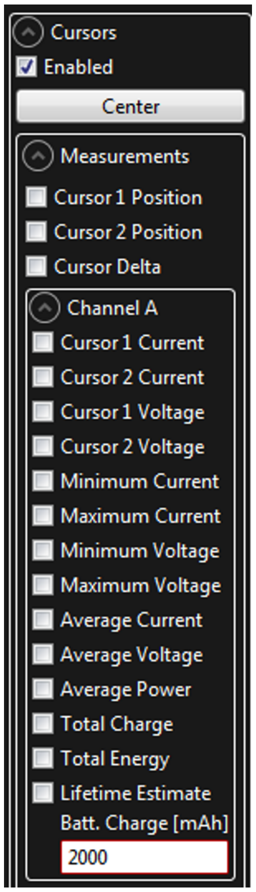


For GPIO data to be available for the **Power Analysis** module the **GPIO** interface has to be enabled in the **DGI Control Panel**.



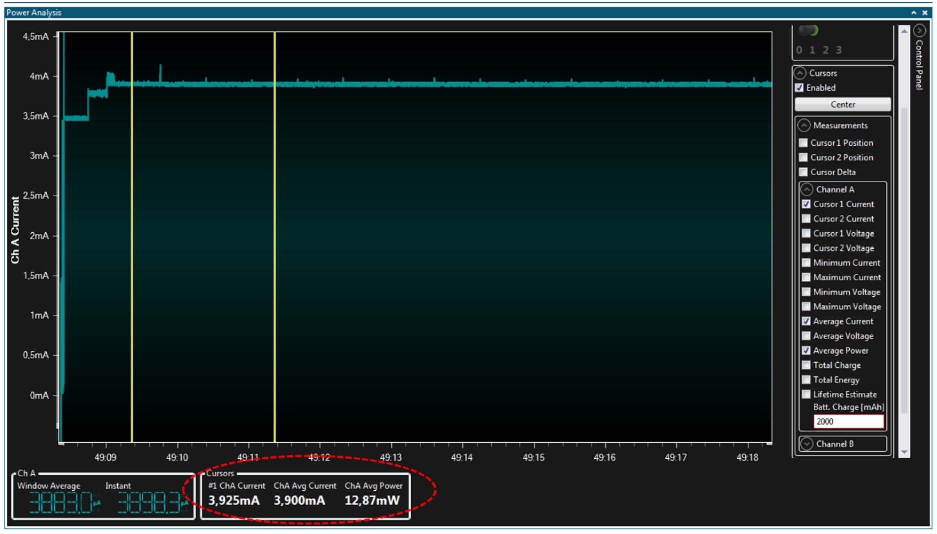
Cursors

The Cursors section in the **Power Analysis** module **Control Panel** allows the user to enable two vertical cursors in the graph by checking the **Enabled** box.



The cursors can be moved by using the mouse pointer to drag them along the X-axis or they can be centered by pushing the **Center** button.

When the cursors are enabled the section of the graph between the cursors can be used for various measurements. The measurements will be shown in the **Cursors** section below the graph.



Cursors: Enabled [✓]

Measurements: Cursor 1 Position [ ], Cursor 2 Position [ ], Cursor Delta [ ]

Channel A: Cursor 1 Current [✓], Cursor 2 Current [ ], Cursor 1 Voltage [ ], Cursor 2 Voltage [ ], Minimum Current [ ], Maximum Current [ ], Minimum Voltage [ ], Maximum Voltage [ ], Average Current [✓], Average Voltage [ ], Average Power [✓], Total Charge [ ], Total Energy [ ], Lifetime Estimate [ ]

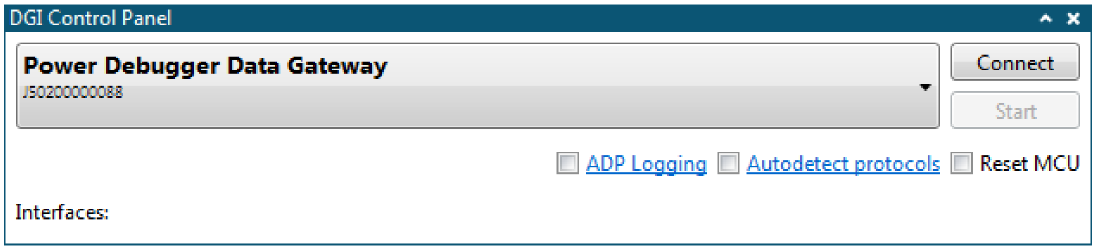
Batt. Charge [mAh]: 2000

#1 ChA Current	ChA Avg Current	ChA Avg Power
3,925mA	3,900mA	12,87mW

Which measurements to be shown can be selected in the **Measurements** sub-section of the **Cursors** section in the **Power Analysis** module **Control Panel**.

3.4.2 Basic Current Measurement

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

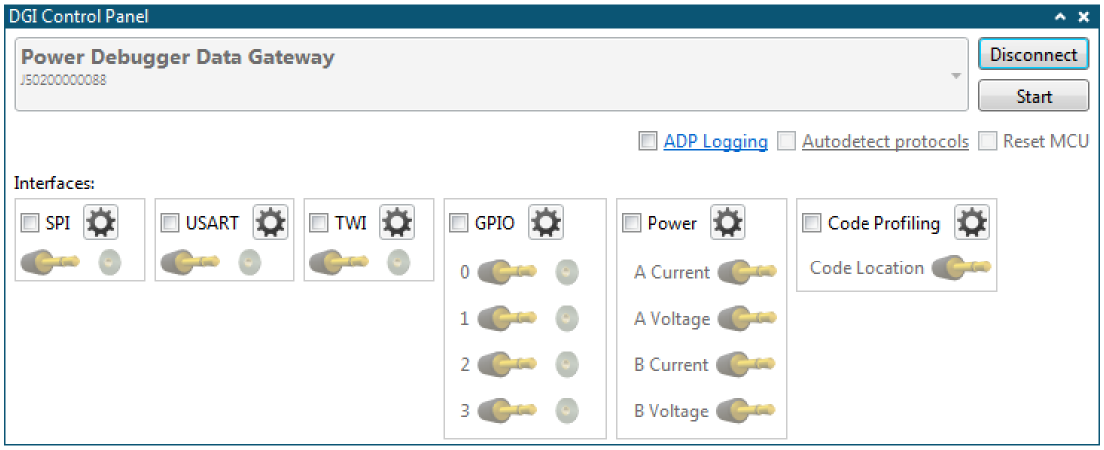

DGI Control Panel

Power Debugger Data Gateway
J50200000088

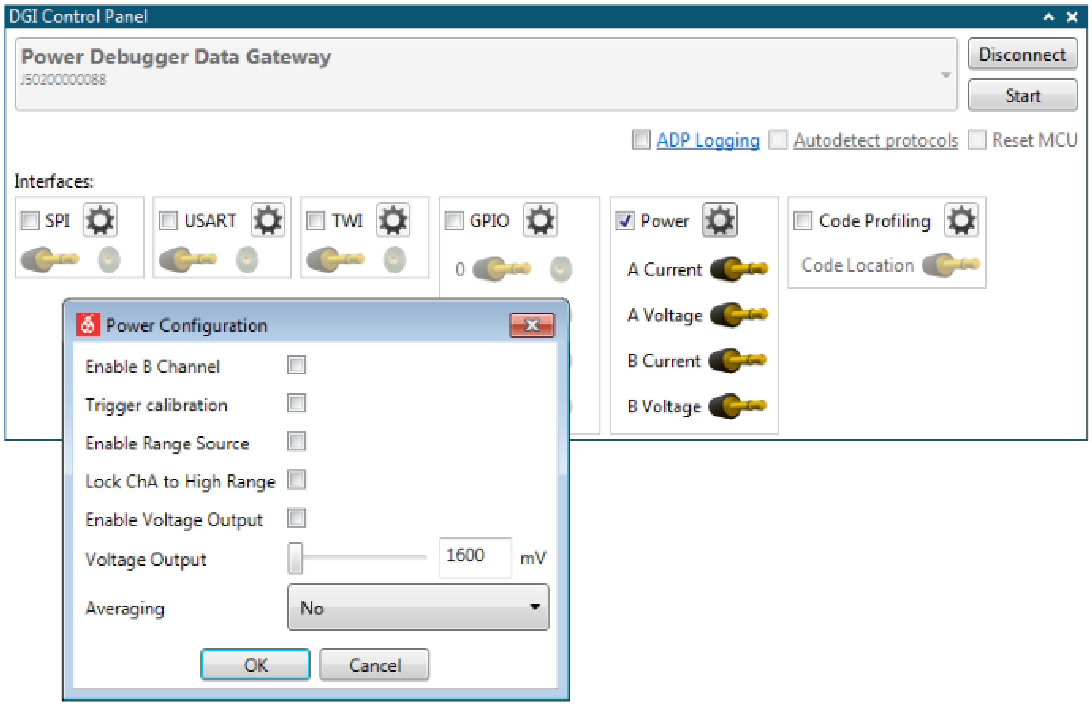
[ ] ADP Logging [ ] Autodetect protocols [ ] Reset MCU

Interfaces:

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.

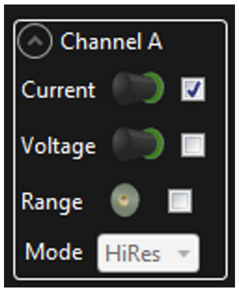


To do: **Start** the Data Visualizer session.

3.4.2.1 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 the user 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.

3.4.2.2 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 as needed. 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

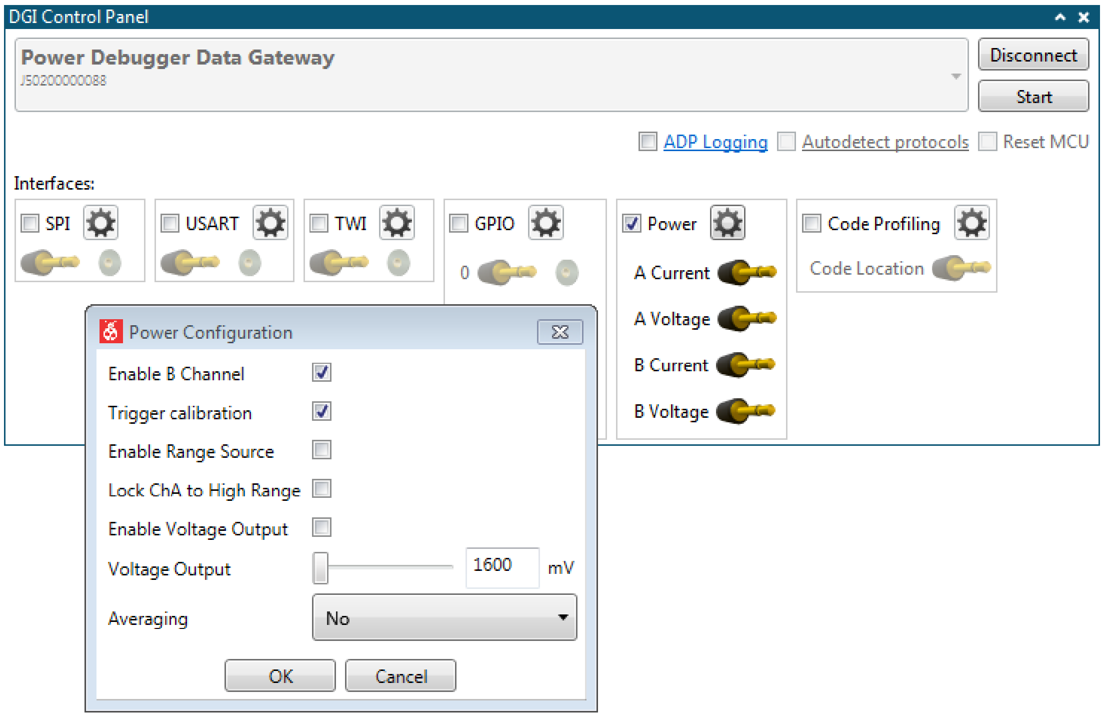
3.4.2.3 Current Averaging

The **Power Analysis** module displays two averaged values per channel. One shows the instantaneous current value, while the other shows the average of the samples visible in the graph view.

3.4.2.4 Power Measurement Calibration

To achieve full measurement accuracy on the current measurement hardware, it should be calibrated before running the measurements. The calibration procedure is started through the **Power Configuration** window in the **Power** interface in the **DGI Control Panel**.

Figure 3-18. Triggering Power Measurement Calibration from the Power Configuration Window



Enable B Channel ✓
Trigger calibration ✓
Enable Range Source ☐
Lock ChA to High Range ☐
Enable Voltage Output ☐
Voltage Output 1600 mV
Averaging No

To start the calibration procedure, select **Trigger calibration** and press OK. Then follow the instructions to complete the calibration procedure for the tool.

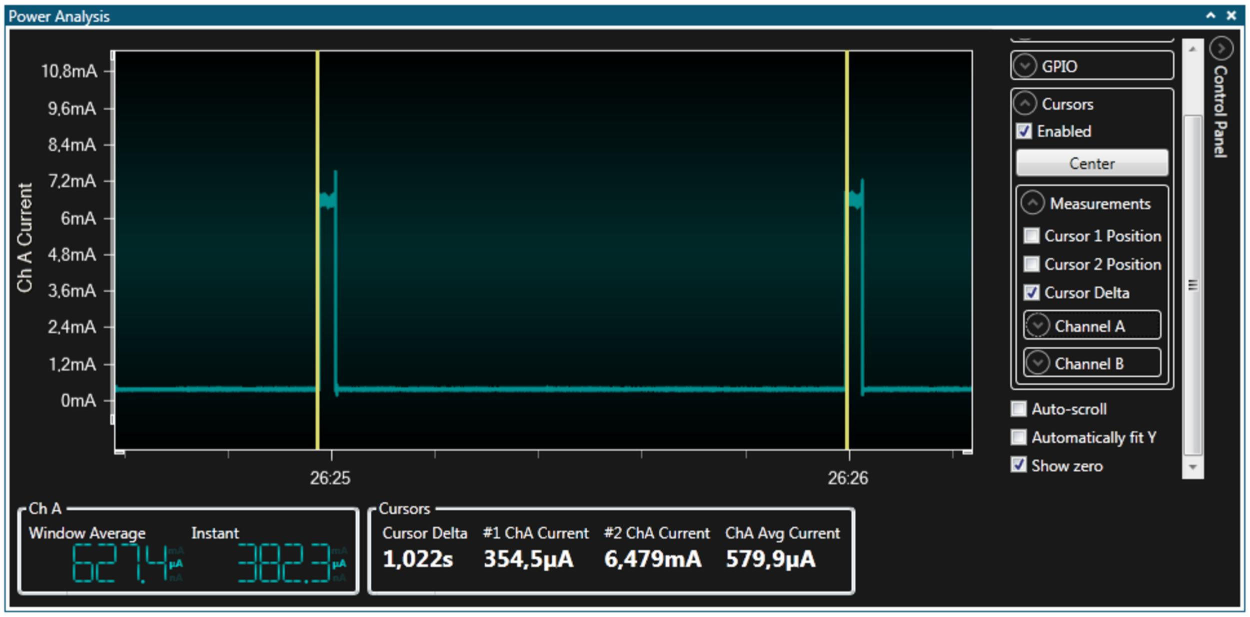
3.4.3 Power Analysis using Cursors

In order to analyze the current more closely, the cursor feature of the **Power Analysis** module is useful.

To do:

- Open the **Control Panel** in the upper right corner of the **Power Analysis** module
- Expand the **Cursors** section
- Click 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, or else the graph view will rapidly scroll away from the cursors.



Cursors

Cursor Delta	#1 ChA Current	#2 ChA Current	ChA Avg Current
1,022s	354,5µA	6,479mA	579,9µA

The example above shows the current consumption of a target board with a LED that toggles ON and OFF regularly.

The cursor data at the bottom of the module shows that the current consumption when the LED is OFF is about 354 µA, while the current consumption when the LED is ON is about 6.5 mA. The average current consumption during one period of the LED toggling is about 580 µA. As the current measurement channel is also monitoring voltage, you can measure the power consumption directly. Enable this by setting the corresponding options in the **Measurements** section of the **Cursors** section in the **Control Panel** of the **Power Analysis** module.

3.4.4 Code Correlation

To optimize current consumption, current measurements must be correlated to the code execution of the application. The Data Visualizer enables code correlation by the use of GPIO instrumentation or program counter sample readout. Crucial to both these methods is the ability to show the code execution related events with the same time base and in the same graph as the current consumption.

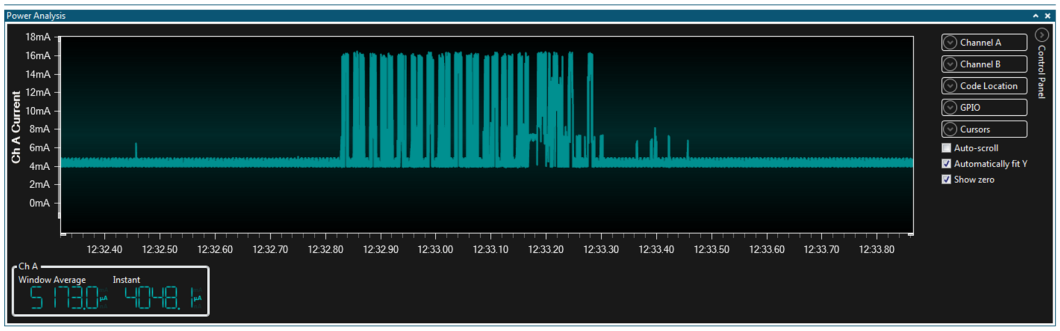
3.4.4.1 GPIO Instrumentation

By inserting simple GPIO toggling in the application code, the user can generate common reference points between the measured current and the code execution. The Data Visualizer is capable of showing the GPIO events in the same graph as the current measurements.

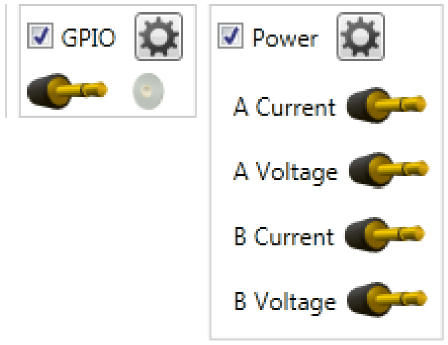
A mass storage application will be used to demonstrate the use of GPIO instrumentation.

Both Target USB and Debug USB connectors of a SAM L21 Xplained Pro board is connected to a host computer. The ATSAML21 target device is running the USB Device MSC Example from ASF for SAM L21 Xplained Pro (in Atmel Studio select File→New→Example Project and search for “MSC”). The Current Measurement jumpers on the kit are set to measure MCU current and bypass I/O current.

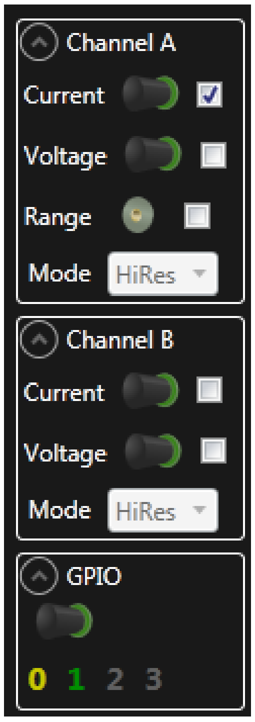
The current graph after running a format of the mass storage device:



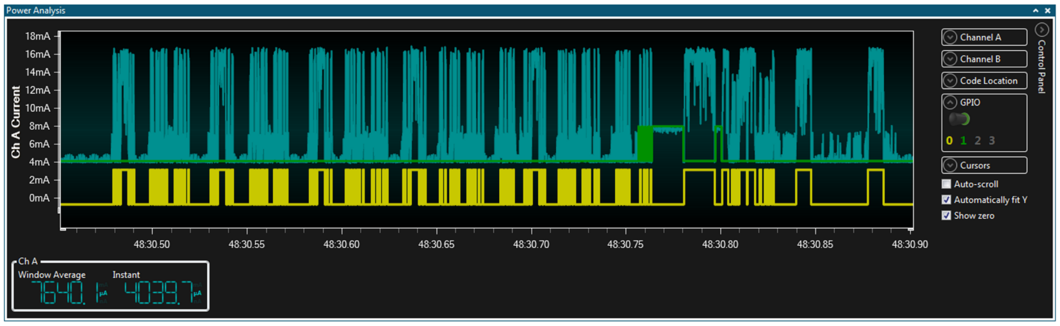
A disk format operation consists of both read and write operations, but with just the current it is difficult to tell *what* is going on *when*. To be able to separate the read and write operations, the application code is modified to set GPIO0 (PB01 on the ATSAML21) *high* on the start of a read operation and set it *low* at the end of the read operation. GPIO1 (PA16 on the ATSAML21) is similarly toggled for write operations. Both the **GPIO** interface and the **Power** interface must be enabled in the **DGI Control Panel** of the Data Visualizer as shown below.



In the **Control Panel** of the **Power Analysis** module disable **GPIO2** and **GPIO3** as shown below.



With the GPIO signals enabled, the user can distinguish the current consumption of the read and write operations. The yellow signal is GPIO0 which signals the read operations and the green signal is GPIO1 showing the write operations.



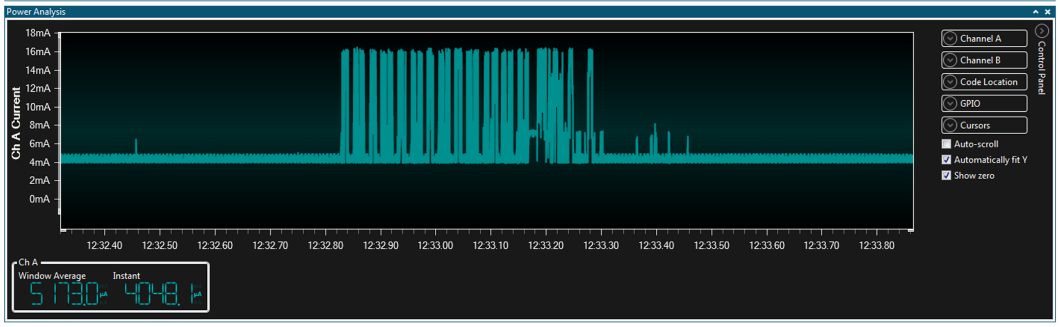
3.4.4.2 Program Counter Polling

Each time the Program Counter (PC) is read out from the target, we get the exact information on the address of the code location currently being executed. The Data Visualizer can show PC values with current measurements in the same graph. This allows the user to see what is being executed by the target CPU at the various sample points of the current consumption graph. The sampled PC values will only show part of the code execution, as in most cases it is impossible to read out the PC values as fast as the target is executing instructions. The sampled values are still useful to indicate which code segment is being executed at any point in time.

A SAM L21 Xplained Pro board running a Mass Storage Class example will be used to demonstrate PC polling.

Both Target USB and Debug USB connectors of a SAM L21 Xplained Pro board is connected to a host computer. The ATSAML21 target device is running the USB Device MSC Example from ASF for SAM L21 Xplained Pro (in Atmel Studio select File→New→Example Project and search for “MSC”). The Current Measurement jumpers on the kit are set to measure MCU current and bypass I/O current.

The current graph after running a format of the mass storage device:

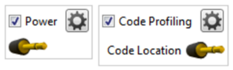


A disk format operation consists of both read and write operations, but from the current graph it is difficult to see *what* is going on *when*. To get more information on what is going on in the target at the various points in the current graph, the Program Counter sampling feature will be useful.

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

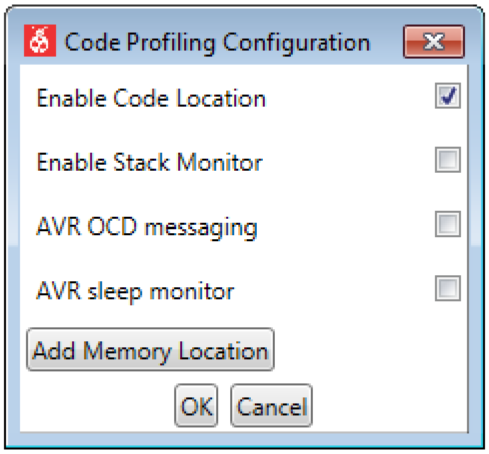
To do:

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



To do:

- Open the **Code Profiling Configuration** dialog by clicking the Gear button on the **Code Profiling** interface
- Select **Enable Code Location**


Code Profiling Configuration

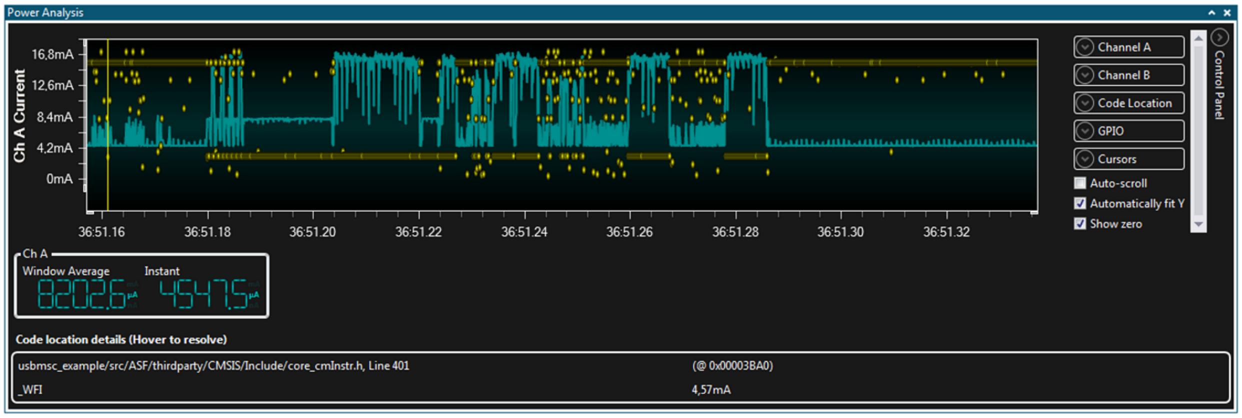
Enable Code Location ☑

Enable Stack Monitor ☐

AVR OCD messaging ☐

AVR sleep monitor ☐

A typical current graph with Program Counter sampling enabled during a format operation is shown below.

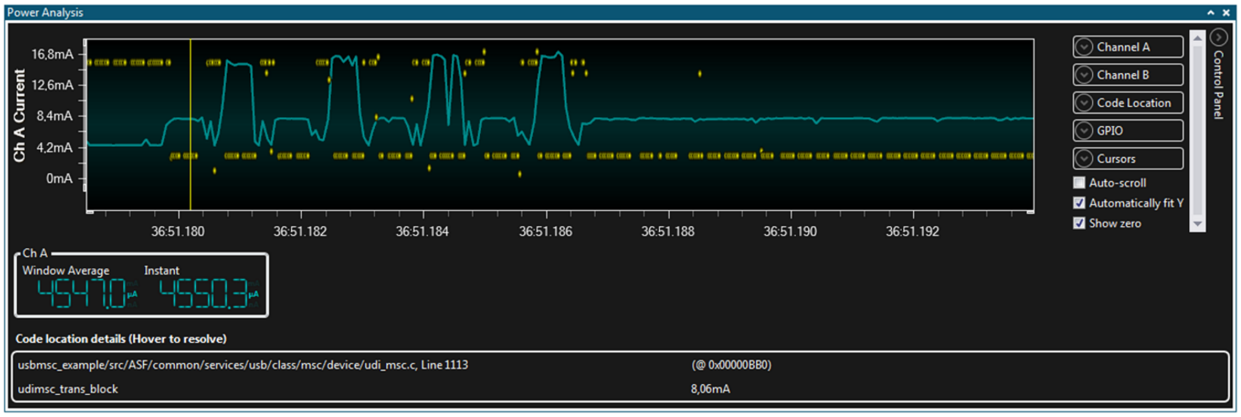


Ch A
Window Average 8202.6 µA Instant 4547.5 µA

Code location details (Hover to resolve)

```
usbmsc_example/src/ASF/thirdparty/CMSIS/Include/core_cmInstr.h, Line 401    (@ 0x00003BA0)
_WFI                                                                         4,57mA
```

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 the execution of different functions. Patterns can easily be seen using this technique. 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 sample at that point.



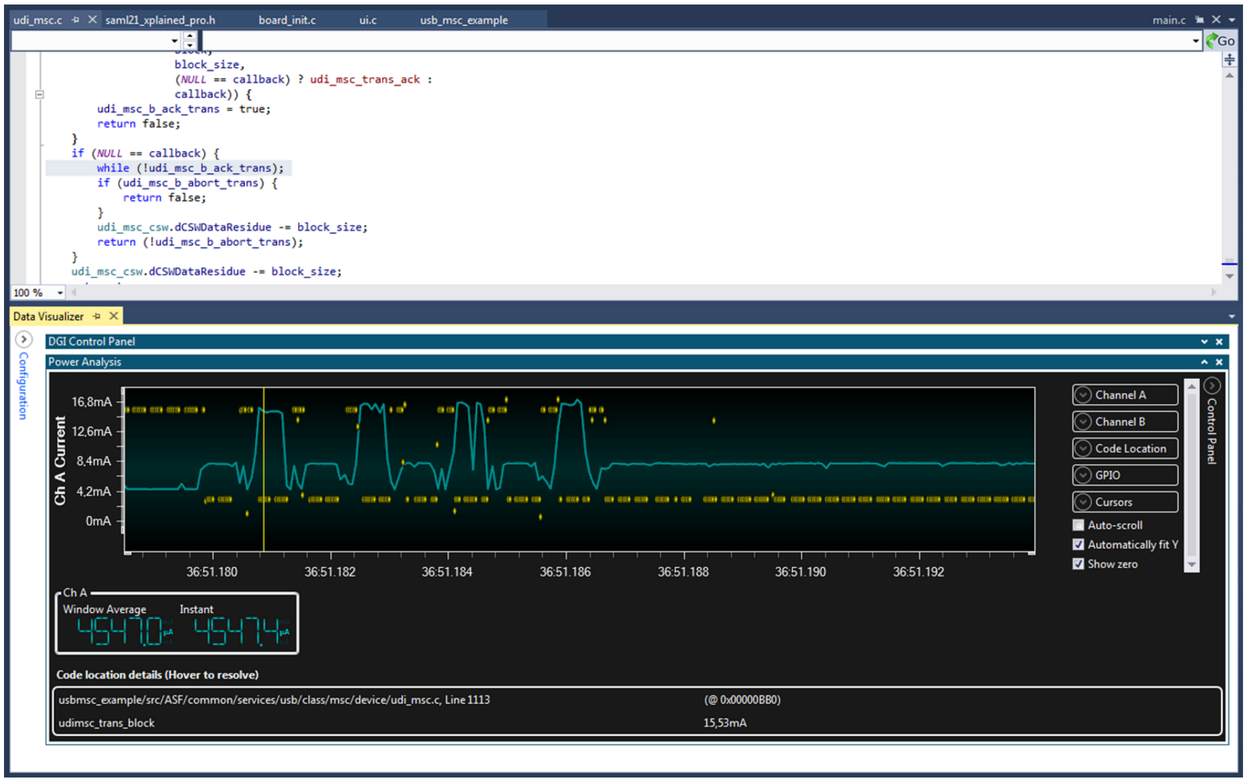
Auto-scroll ☐
Automatically fit Y ✓
Show zero ✓

Ch A
Window Average 4547,0 µA
Instant 4550,3 µA

Code location details (Hover to resolve)

usbmsc_example/src/ASF/common/services/usb/class/msc/device/udi_msc.c, Line 1113	(@ 0x00000BB0)
udimsc_trans_block	8,06mA

Double-clicking on one of the samples will open the editor and highlight the corresponding line of code.



```
                    block_size,
                    (NULL == callback) ? udi_msc_trans_ack :
                    callback)) {
            udi_msc_b_ack_trans = true;
            return false;
        }
        if (NULL == callback) {
            while (!udi_msc_b_ack_trans);
            if (udi_msc_b_abort_trans) {
                return false;
            }
            udi_msc_csw.dCSWDataResidue -= block_size;
            return (!udi_msc_b_abort_trans);
        }
        udi_msc_csw.dCSWDataResidue -= block_size;
```

Auto-scroll ☐
Automatically fit Y ✓
Show zero ✓

Ch A
Window Average 4547,0 µA
Instant 4547,4 µA

Code location details (Hover to resolve)

usbmsc_example/src/ASF/common/services/usb/class/msc/device/udi_msc.c, Line 1113	(@ 0x00000BB0)
udimsc_trans_block	15,53mA

The highlighted sample is located in a function called udi_msc_trans_block. This function transfers data from RAM to USB. From the graph it can be seen that the current spike at the marker is generated by the execution of this function as all Program Counter samples are from the same location during this spike.

3.5 Custom Dashboard

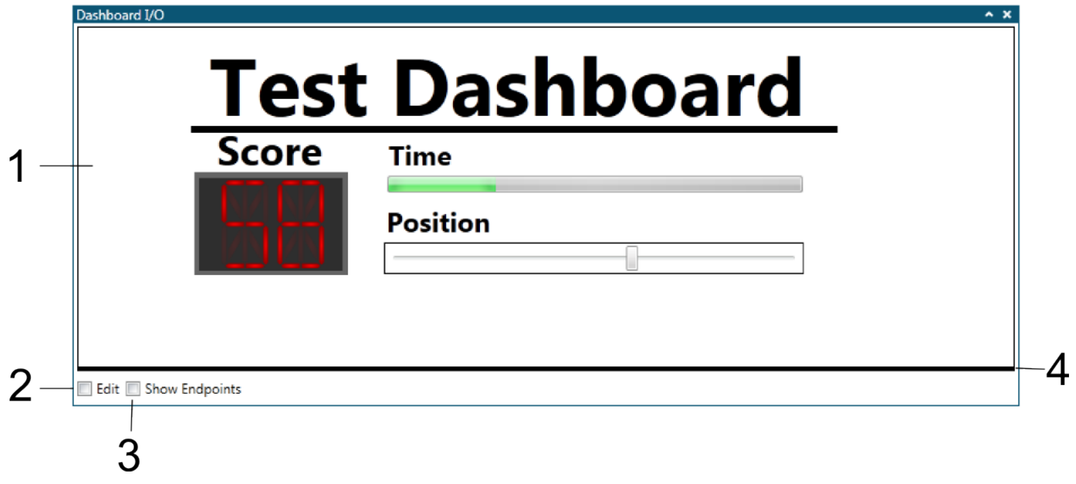
The **Dashboard** module is a customizable Graphical User Interface (GUI) panel. It can be used to control and display parameters from the target application.

3.5.1 Dashboard Module

The **Dashboard** module is a customizable Graphical User Interface (GUI) panel. It can be used to control and display parameters from the application firmware. **Elements** (button, label, slider, etc.) are placed in the dashboard area to form the GUI. Each element can have an endpoint associated with it to send or receive values. A slider, for example, has a source that outputs the slider position as a numeric value. Endpoints are shown when the **Show Endpoints** option is selected.

For an example on how to configure a dashboard, see Dashboard Configuration Example.

Figure 3-19. Dashboard

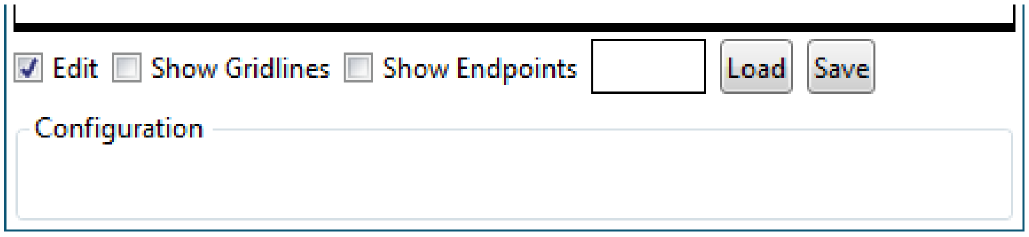


1. Dashboard area. 2. **Edit** checkbox. 3. **Show Endpoints** checkbox. 4. **Height adjustment** tab.

3.5.1.1 Edit Panel

When going into Edit mode (by selecting the **Edit** box), the **Edit** panel will become visible. Here the user can customize the dashboard.

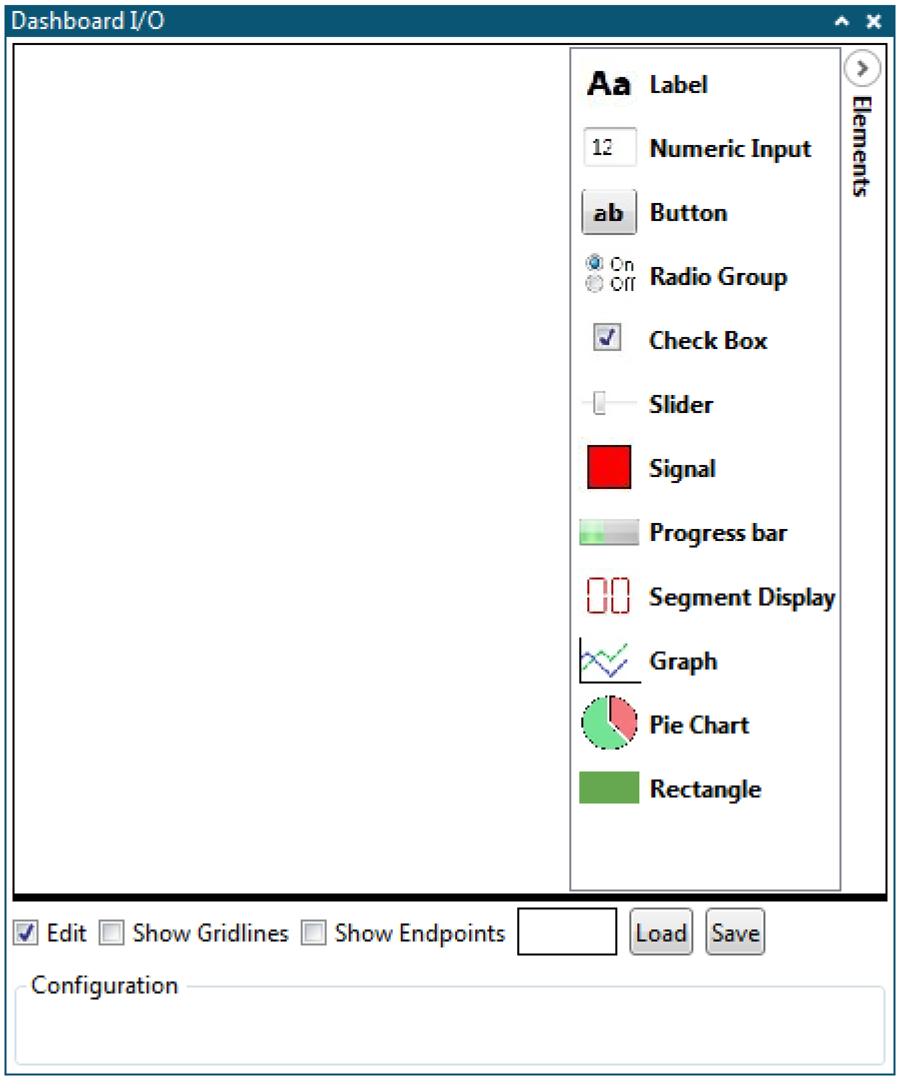
Figure 3-20. Edit Panel



Placing Elements on the Dashboard

By default, the dashboard area is empty. Elements can be placed on the dashboard by following the procedure below.

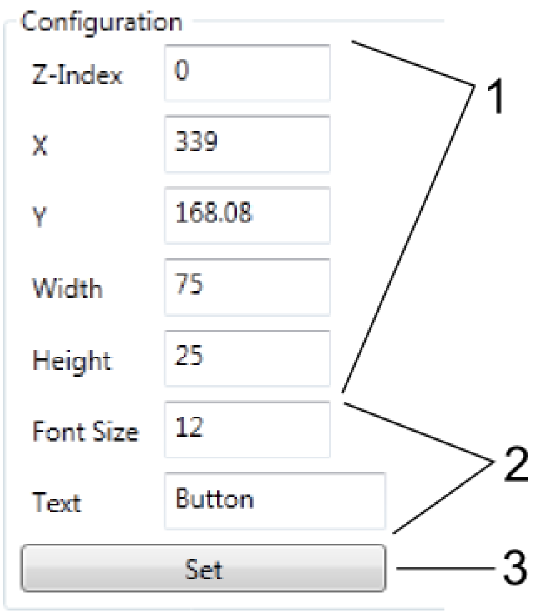
- Click the **Edit** checkbox
- Open the **Elements** panel in the upper right corner of the dashboard area
- Click and hold the element
- Drag the mouse over the dashboard area
- Drop the element in the dashboard area on the desired location



Configuring Dashboard Elements

All dashboard elements can be configured when in Edit mode. The parameters will vary depending on element type, but the procedure for changing them is the same.

Figure 3-21. Element Configuration



1. Common parameters. 2. Element-specific parameters. 3. **Set** button.

- Click the **Edit** checkbox
- Select the element to configure by clicking it. The Configuration window will list the configurable parameters for the selected element.
- Change the parameters to the desired value
- Click the **Set** button

Moving Elements

All parameters related to position and size are available in the element configurations. Elements can also be moved by dragging them around in the dashboard area in Edit mode. Resizing can be done by dragging the black tab in the corner after selecting an element.

Deleting Elements

To delete unwanted elements, simply select the element by left-clicking it, and then right-click it to delete.

Important: This action is permanent, and all configuration is lost after deletion.

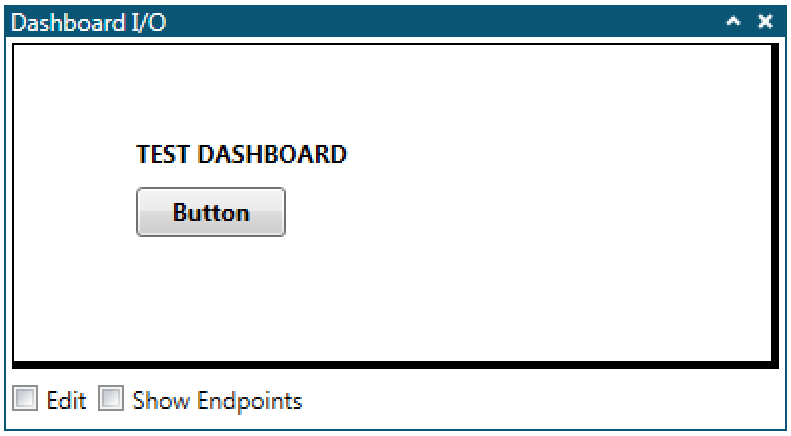
Loading and Saving

The dashboard can be saved by clicking the **Save** button in Edit mode. All elements and configuration parameters, in addition to dashboard background color, will be stored.

To load a dashboard, click the **Load** button and browse to a valid dashboard save file.

The saved file is a text file but could have any file extension containing all configuration parameters for each dashboard element enclosed in curly brackets {} and separated by a semicolon. Each line corresponds to one configuration parameter and the format of each parameter is a list of decimal byte values separated by commas. Each configuration parameter is given by the Least Significant Byte first. The order of the configuration parameters are the same as the order of the configuration parameters in the Configuration window when the **Edit** option for the dashboard is selected. Comments are marked by double forward slashes "//" and the rest of the line is ignored by the parser when encountering double slashes.

A simple example of a saved dashboard configuration is given below. A more complex example can be found in Auto-Configuration Example.



The file content of the saved configuration for this dashboard is given below.

```
{
0,
'\0',
0, 255, 255, 255,
158, 0,
};

{
0, // Dashboard ID
0, // Element ID
DB_TYPE_LABEL, // Element Type
0, // Z-Index (GUI stack order)
61, 0, // X-coordinate
46, 0, // Y-coordinate
122, 0, // Width
17, 0, // Height
12, // Font Size
1,
0, // Horizontal Alignment
0, // Vertical Alignment
0, 255, 255, 255, // Background Color
255, 0, 0, 0, // Foreground Color
'T', 'E', 'S', 'T', ' ', 'D', 'A', 'S', 'H', 'B', 'O', 'A', 'R', 'D', '\0', // Text
};

{
0, // Dashboard ID
1, // Element ID
DB_TYPE_BUTTON, // Element Type
0, // Z-Index (GUI stack order)
61, 0, // X-coordinate
70, 0, // Y-coordinate
75, 0, // Width
25, 0, // Height
12, // Font Size
'B', 'u', 't', 't', 'o', 'n', '\0', // Text
0,
};
```

The first element in the file is the dashboard itself. The first line defines the Dashboard ID (0). Then follows the Title of the dashboard (empty string), the background color of the dashboard (Alpha = 0x00, Red = 0xFF, Green = 0xFF and Blue = 0xFF), and the height of the dashboard (two byte value, LSB first; 152, 0 => 152 = 0x0098 pixels).

The following elements are the Label element and the Button element.

Note that strings are null terminated (\0).

Checkboxes are usually grouped and only one bit per checkbox is used to indicate the checkbox state. For example, for the Label element there are two checkboxes following each other in the Edit window, one named **Bold** and one **Italic**. These are combined into one Configuration byte with bit 0 giving the state of the **Bold** checkbox and bit 1 giving the state of the **Italic** checkbox. In the example above, this is the 1 between Font size and Horizontal alignment. The value 1 indicates that the Label text format should be bold but not italic.

Drop-down boxes are given as single byte values with a number corresponding to the selected option. The topmost option in the list corresponds to a configuration value of 0. For example, for the Label element the **Horizontal Alignment** can be either **Left**=0, **Center**=1, or **Right**=2.

Setting Background Color

The square next to the **Load** button is the **Background color** selector. Clicking the selector will bring up the **Color selector** dialog. Use the sliders to select the desired color, then press OK.

3.5.1.2 Element Types

The various dashboard element types are presented in this section. All element types have some common parameters. These are listed in the table below. The following sections will list only the parameters specific to each element type.

Table 3-1. Common Element Parameters

Parameter	Type	Usage
Z-index	Numeric	Order index, 0 places the element the farthest to the back
Left	Numeric	Horizontal placement, unit pixels
Top	Numeric	Vertical placement, unit pixels
Width	Numeric	Width of element in pixels
Height	Numeric	Height of element in pixels

Label

The **Label** element displays a text string.

Figure 3-22. Label

Endpoints

The **Label** element has a sink endpoint that accepts all types of sources. Any data sent to the label will be converted to a string and displayed as text.

Configuration

Table 3-2. Label Specific Parameters

Parameter	Type	Usage
Font Size	Numeric	Adjusts the size of the font
Bold	Checkbox	Sets bold style of the font

Parameter	Type	Usage
Italic	Checkbox	Sets italic style of the font
Horizontal Alignment	Dropdown box	Selects the alignment of the text within the specified width
Vertical Alignment	Dropdown box	Selects the alignment of the text within the specified height
Background Color	Color	Sets the background color of the label
Foreground Color	Color	Sets the color of the text
Text	String	Sets the label text

Numeric Input

The **Numeric Input** element enables input of numeric values to the dashboard.

Figure 3-23. Numerical Input

Endpoints

The **Numeric Input** has a source endpoint of type int32. Each time the numerical input value is changed a packet with the value is sent.

Configuration

Table 3-3. Numerical Input Specific Parameters

Parameter	Type	Usage
Minimum	Numeric	Minimum value of input
Maximum	Numeric	Maximum value of input
Value	Numeric	Initial value

Button

The **Button** element will send an event each time it is pressed. The button can either be configured as a normal push button or as a toggle button. The button can have a static text to indicate its functionality. When it is configured as a toggle button the text will be replaced by ON or OFF depending on the state of the button. To replace the ON/OFF text by something else the **Text** parameter must be given as a '/' delimited text with the first part of the text being the ON state text and the second part the OFF state text.

Figure 3-24. Button

Endpoints

The **Button** element has a source endpoint of type uint8. Each time the button is pressed a packet is sent. The value of the packet will always be 0 for a normal button and 0 for a toggle button in its OFF state and a 1 for a toggle button in its ON state.

Configuration

Table 3-4. Button Parameters

Parameter	Type	Usage
Font Size	Numeric	Sets the font size of the button tag
Text	String	Sets the button tag text. If the button is configured as a toggle button the test should be delimited by '/'. The first part of the text will then be the ON state text while the second part will be the OFF state text.
Toggle Button	Checkbox	Configures the button to be a ON/OFF toggle switch.

Radio Group

The **Radio Group** element is a group of radio buttons where only one option can be selected at any time. Initially the first option is selected.

Figure 3-25. Radio Group

Endpoints

The **Radio Group** element has a source endpoint of type uint16. Each time the state of the element is changed a message is sent with the index of the currently active option.

Configuration

Table 3-5. Radio Group Specific Parameters

Parameter	Type	Usage
Font Size	Numeric	Font size of the button text
Number of Radio Buttons	Numeric	Number of buttons in the group
Orientation	Numeric	0 = Horizontal 1 = Vertical
Text	String	'/' delimited text with the text for each button starting with the text for button with index 0

Check Box

The **Check Box** element will send an event each time its state is changed.

Figure 3-26. Check Box

Endpoints

The **Check Box** element has a source endpoint of type uint8. Every time the state of the element is changed a message is sent. When the box is checked a 1 is sent and when it is unchecked a 0 is sent.

Configuration

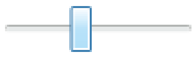
Table 3-6. Check Box Specific Parameters

Parameter	Type	Usage
Font Size	Numeric	Font size for the text
Text	String	Sets the **Check Box** tag text

Slider

The **Slider** element is a linear bar with a movable marker. The marker can be moved to adjust the value of the slider.

Figure 3-27. Slider



Endpoints

The **Slider** element has a source endpoint of type double. When the slider value is changed a packet with the value is sent.

Configuration

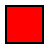
Table 3-7. Slider Parameters

Parameter	Type	Usage
Minimum	Numeric	Sets the minimum value of the slider
Maximum	Numeric	Sets the maximum value of the slider
Value	Numeric	Sets the value of the slider

Signal

The **Signal** element is a simple color-based ON/OFF indicator.

Figure 3-28. Signal



Endpoints

The **Signal** element has a sink endpoint that accepts all data types, but ignores strings and multi-dimensional values. The color of the signal is decided by a boolean evaluation, if the incoming value is a number it is true if it is greater than 0.

Configuration

Table 3-8. Signal Parameters

Parameter	Type	Usage
Color On	Color	Selects the color used when the signal is ON
Color Off	Color	Selects the color used when the signal is OFF

Progress Bar

The **Progress bar** element is a linear bar that shows the position of a value between a min. and max. value.

Figure 3-29. Progress Bar

Endpoints

The **Progress bar** element has a sink endpoint that accepts all numeric data types. When a value is received, it will update the amount of colored area of the progress bar depending on the min. and max. values.

Configuration

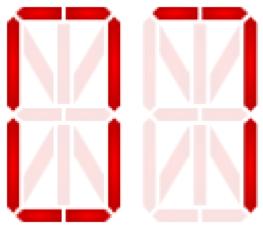
Table 3-9. Progress Bar Parameters

Parameter	Type	Usage
Minimum	Numeric	Sets the minimum value of the progress bar
Maximum	Numeric	Sets the maximum value of the progress bar
Value	Numeric	Sets the value of the progress bar
Color	Color	Sets the color of the progress bar

Segment Display

The **Segment display** element simulates a hex-digit LED display.

Figure 3-30. Segment Display



Endpoints

The **Segment display** element has a sink endpoint that accepts all numeric data types. The value received is displayed.

Configuration

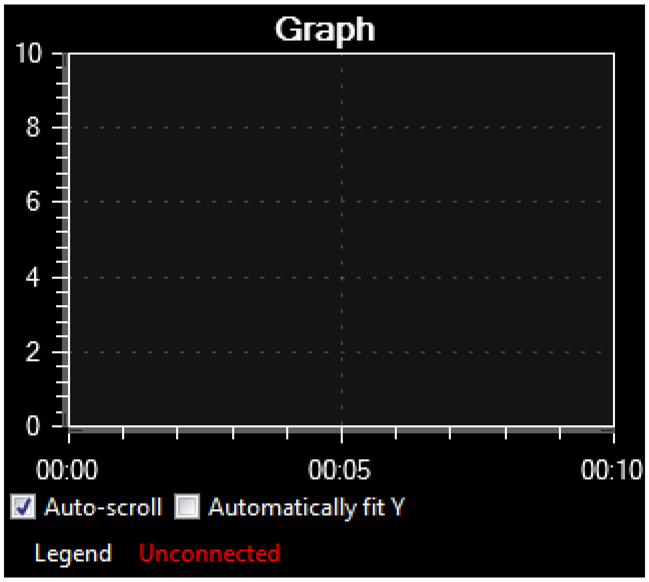
Table 3-10. Segment Display Parameters

Parameter	Type	Usage
Segment Count	Numeric	Number of hex-segments to display
Numeric Base	Numeric	Sets the base used for displaying numbers
Segment Color	Color	Sets the color of the segment display

Graph

The **Graph** element plots the incoming data streams in a two-dimensional graph. The graph can be configured to accept zooming and scrolling by mouse interaction or to be static ignoring any mouse interaction.

Figure 3-31. Graph

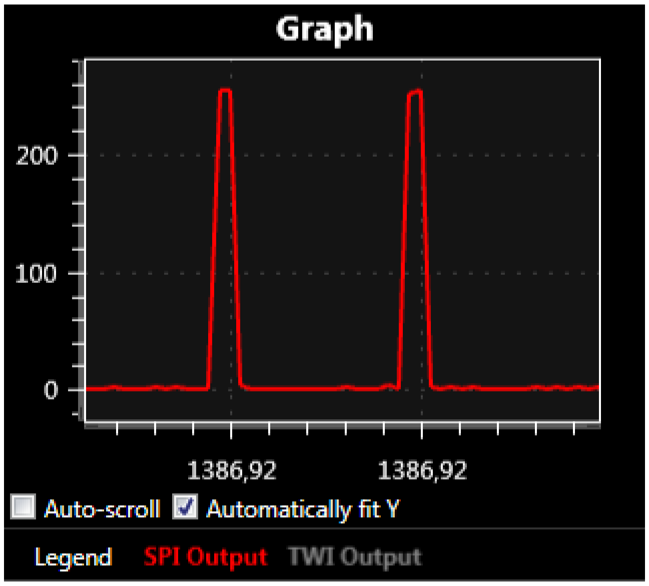


Endpoints

The **Graph** element has one sink endpoint for each plot. The endpoints accepts all numerical data types.

Each plot in the **Graph** can be shown or hidden dynamically by clicking the legend corresponding to the plot at the bottom of the **Graph** element. Hidden plots have a gray legend compared to visible plots having the same color on the legend as the plot itself.

Figure 3-32. Graph with Visible SPI Output Plot and Hidden TWI Output Plot



In the graph above the plot SPI Output is visible while the plot TWI Output is hidden.

Configuration

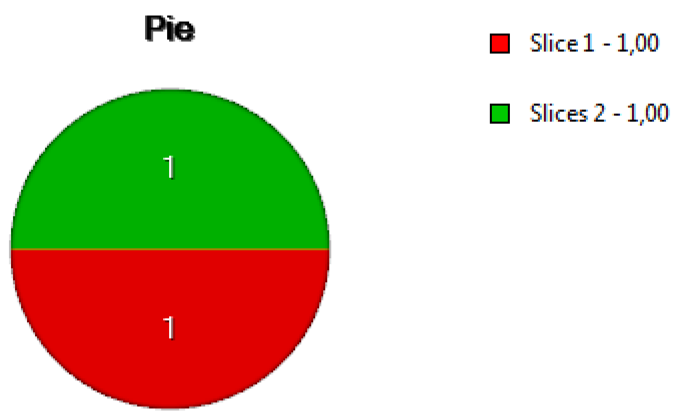
Table 3-11. Graph Specific Parameters

Parameter	Type	Usage
Title color	Color	Selects the color of the title text
Background color	Color	Selects the color of the complete **Graph** element background
Graph background color	Color	Selects the color of the graph plot area background
Title	String	Title of the graph
Number of plots	Numeric	Number of plots to display in the graph. Each plot will have its own sink endpoint.
X Minimum	Numeric	Minimum value of X axis
X Maximum	Numeric	Maximum value of X axis
Y Minimum	Numeric	Minimum value of Y axis
Y Maximum	Numeric	Maximum value of Y axis
Mouse Interaction	Checkbox	Enable mouse interaction with the **Graph** element
Fit to right	Checkbox	Expand the **Graph** element to the right edge of the dashboard
Autoscale	Checkbox	Automatically scale Y axis accoriding to plot data
Scroll by time	Checkbox	Scroll X axis by time. If not checked the X axis will scroll by incoming plot samples.
Show plot	Checkbox	View continuous graph plot (sample points interconnected)
Show points	Checkbox	Show single samples as dots

Pie Chart

The **Pie Chart** element displays the value of the incoming streams as slices in a pie chart.

Figure 3-33. Pie Chart



Endpoints

The **Pie Chart** element has one sink endpoint for each slice in the pie chart. The sink endpoints accepts all numerical data types.

Configuration

Table 3-12. Pie Chart Specific Parameters

Parameter	Type	Usage
Title color	Color	Selects the color of the title text
Background color	Color	Selects the element background color
Title	String	Title of the element
Number of plots	Numeric	Number of slizes in the pie chart

Rectangle

The **Rectangle** element sends a packet each time it is clicked by the mouse.

Figure 3-34. Rectangle

Endpoints

The **Rectangle** element has a source endpoint of type uint32. Each time the element is clicked by the mouse pointer a packet with value 0 is sent.

Configuration

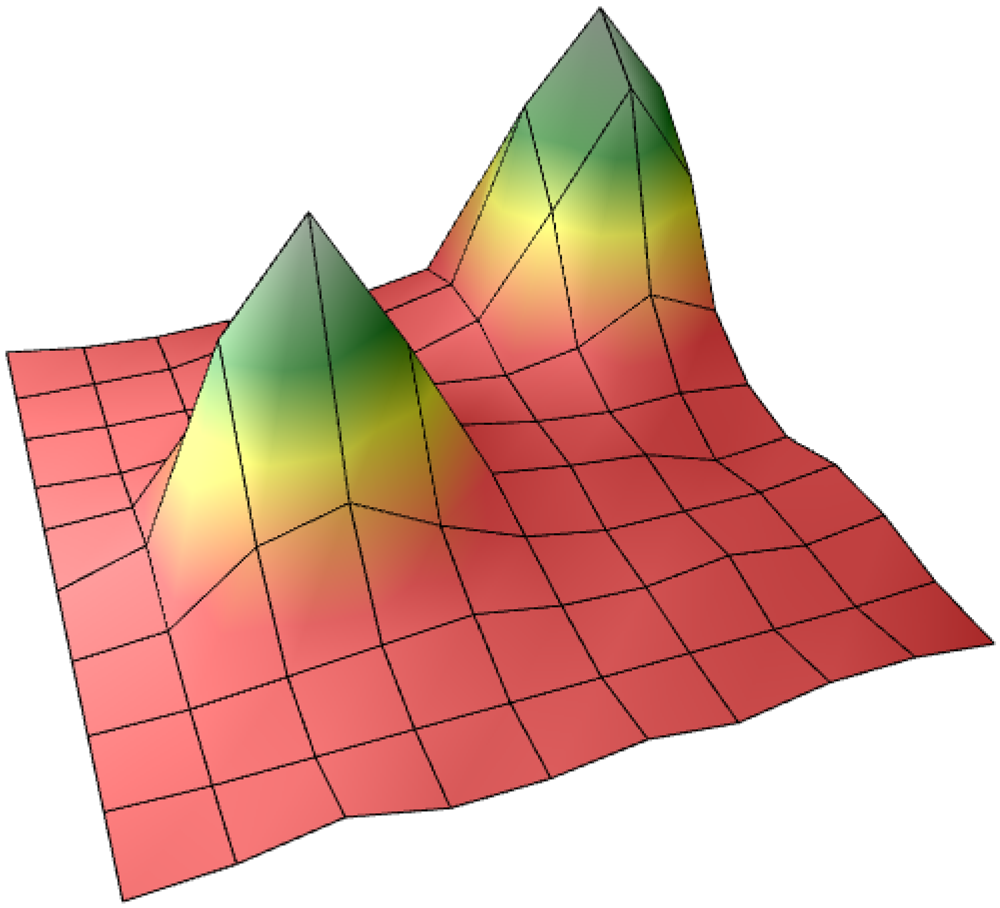
Table 3-13. Rectangle Specific Parameters

Parameter	Type	Usage
Background color	Color	Selects the color of the fill of the rectangle
Foreground color	Color	Selects the color of the frame of the rectangle

Surface

The **Surface** element displays grid data as a surface in 3D space.

Figure 3-35. Surface



Endpoints

The **Surface** element has one endpoint accepting any source of a grid type.

Configuration

Table 3-14. Surface Specific Parameters

Parameter	Type	Usage
Fill color	Color	Selects the color of the surface fill
Mesh color	Color	Selects the color of the surface mesh
Background color	Color	Selects the color of the background
Background gradient color	Color	Selects the color of the background gradient
Axis color	Color	Selects the color of the axes
Tick color	Color	Selects the color of the tick labels
X Rotation	Numeric	Sets rotation of view around X
Y Rotation	Numeric	Sets rotation of view around Y
Z Rotation	Numeric	Sets rotation of view around Z

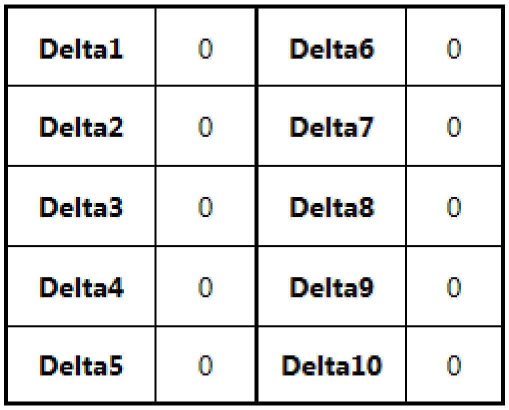
Parameter	Type	Usage
Show X-axis	Checkbox	Sets visibility of X-axis
Show Y-axis	Checkbox	Sets visibility of Y-axis
Show Z-axis	Checkbox	Sets visibility of Z-axis
Show fill	Checkbox	Sets visibility of surface fill
Show mesh	Checkbox	Sets visibility of surface mesh
Use palette coloring	Checkbox	Sets usage of palette coloring (red-yellow-green-white)
Scaling mode	Drop-down box	Selects mode of Y-axis auto-scaling
Axis minimum	Numeric	Sets minimum axis value for Y
Axis maximum	Numeric	Sets maximum axis value for Y

Table

The **Table** element displays one or more data sources in a table. Two modes are supported, Auto Labels and Manual Labels. In the Auto Labels mode, each cell is split into two fields, the field to the left is a label with the name of the data stream and the field to the right is the actual data of the stream. In the Manual Labels mode each cell can be manually configured to either be a Label cell or a Data cell. The mode is selected by the checkbox named **Auto Labels** in the configuration, see Configuration.

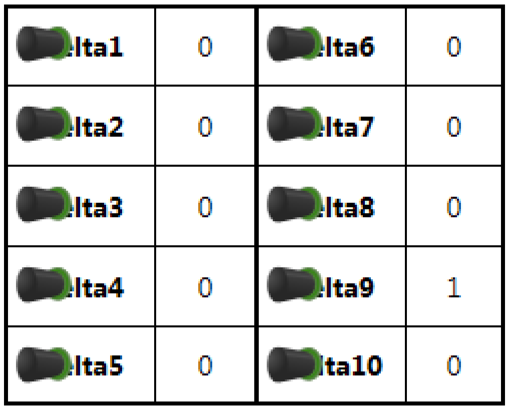
Auto Labels

When using the Auto Labels mode each cell is associated with one data source and the name of the data source is shown to the left in the cell and the actual data to the right. The source name is automatically fetched from the source connected to the sink endpoint.



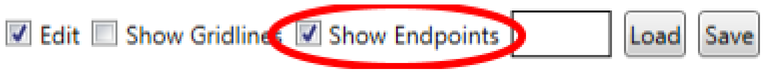
Delta1	0	Delta6	0
Delta2	0	Delta7	0
Delta3	0	Delta8	0
Delta4	0	Delta9	0
Delta5	0	Delta10	0

The **Table** element has one endpoint per table cell accepting any data source. The data will be converted to a string and displayed as text.



lta1	0	lta6	0
lta2	0	lta7	0
lta3	0	lta8	0
lta4	0	lta9	1
lta5	0	lta10	0

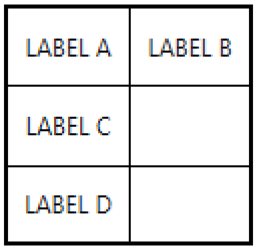
Endpoints are shown when the **Show Endpoints** option is selected.



☑ Edit ☐ Show Gridlines ☑ Show Endpoints Load Save

Manual Labels

When using the Manual Labels mode each cell either is a Label cell or a Data cell. By default all cells are Data cells. Label cells can be configured by setting the **Label Configuration** string, see Configuration. The **Label Configuration** string configures which cells are Labels by giving a semicolon separated list of Labels. Each Label is given by the format <column>:<row>:<label>. The upper left cell is column 0, row 0. An example is given below.

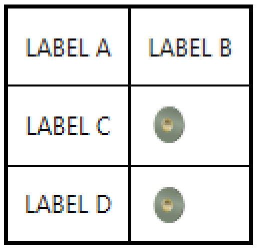


LABEL A	LABEL B
LABEL C	
LABEL D	

The **Label Configuration** for this table is:

```
0:0:LABEL A;1:0:LABEL B;0:1:LABEL C;0:2:LABEL D
```

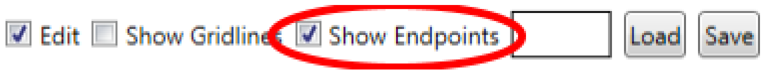
Only the Data cells have endpoints.



LABEL A	LABEL B
LABEL C	●
LABEL D	●

The label text should not contain colons or semicolons to avoid confusing the **Label Configuration** parser.

Endpoints are shown when the **Show Endpoints** option is selected.



☑ Edit ☐ Show Gridlines ☑ Show Endpoints Load Save

Configuration

Table 3-15. Table Specific Parameters

Parameter	Type	Usage
Data Font Size	Numeric	Sets the size of the font in the data part of a cell
Label Font Size	Numeric	Sets the size of the font in the label part of a cell
Data Column Width	Numeric	Width of the data part of each cell. Note that changing this width will change the total width of the table.
Label Column Width	Numeric	Width of the label part of each cell. Note that changing this width will change the total width of the table.
Row Height	Numeric	Height if each row in the table. Note that changing this height will change the total height of the table.
Number of Rows	Numeric	Number of rows in the table
Number of Columns	Numeric	Number of columns in the table
Auto Labels	Checkbox	Enables the Auto Labels mode. If disabled labels must be configured manually
Label Configuration	String	String configuring the labels when using Manual Labels mode (Auto Labels option disabled). Format is <column>:<row>:<label>;<column>:<row>:<label>...
Data Bold	Checkbox	Sets bold style of the font in the data part of each cell
Data Italic	Checkbox	Sets italic style of the font in the data part of each cell
Label Bold	Checkbox	Sets bold style of the font in the label part of each cell
Label Italic	Checkbox	Sets italic style of the font in the label part of each cell
Background Color	Color	Sets the background color of the table
Foreground Color	Color	Sets the color of the table grid and the data and label text
Label Horizontal Alignment	Drop-down box	Selects the placement of the text in the label part of each cell (Left, Center, or Right)
Data Horizontal Alignment	Drop-down box	Selects the placement of the text in the data part of each cell (Left, Center, or Right)

3.5.2 Dashboard Configuration Example

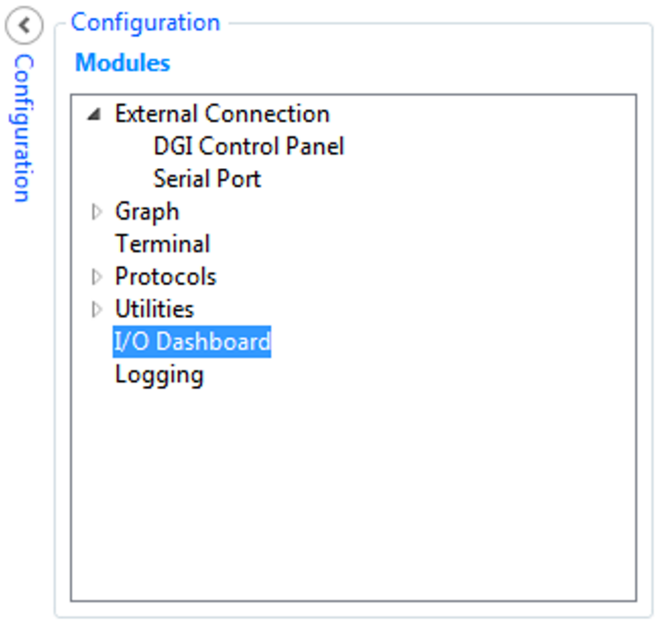
This section gives an example on how to configure the **Dashboard** module. Although the example utilizes only a subset of the available dashboard elements data sources available in the Data Visualizer, the basic principles are applicable to all elements and data sources.

This example uses manual configuration of the **Dashboard** module, but it is also possible to use the Atmel Data Protocol (ADP) to set up a dashboard automatically. For more information on ADP and an example of a automatically configured dashboard, see Atmel Data Protocol.

The target application code used in this example and a description of the hardware setup can be found in Dashboard Example Code.

To do:

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

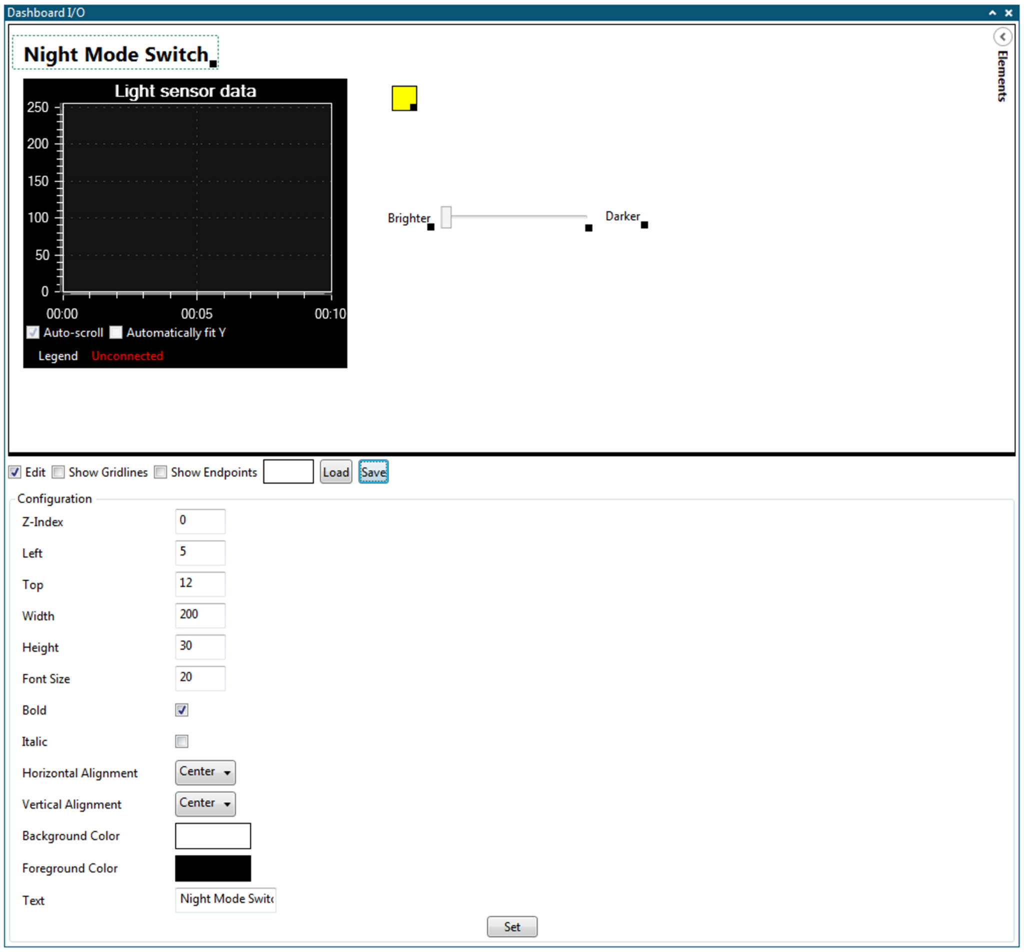


To do:

- Enable editing the dashboard by clicking the **Edit** option in the lower left corner of the **Dashboard I/O** module
- Open the **Elements** panel in the upper right corner of the dashboard and drag elements onto the dashboard.

Tip: To remove an element from the dashboard, select it by left-clicking it, and then right-click the element.

Tip: Changing the parameters in the **Configuration** section will not take effect until the **Set** button is clicked.



Configuration

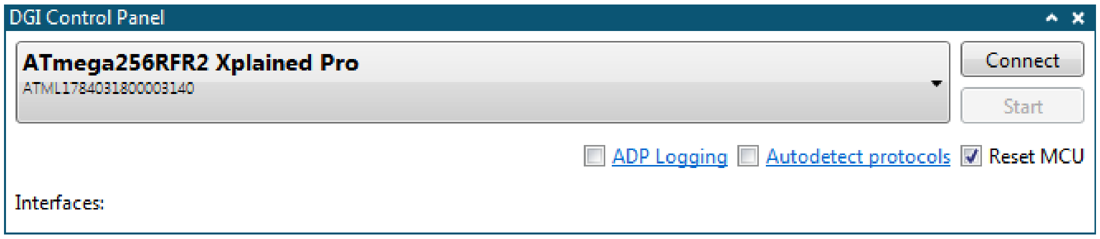
Z-Index	0
Left	5
Top	12
Width	200
Height	30
Font Size	20
Bold	✓
Italic	☐
Horizontal Alignment	Center
Vertical Alignment	Center
Background Color	⬜
Foreground Color	⬛
Text	Night Mode Switc

In this example, three **Label** elements are added, one as a title for the dashboard and the two others as help text for the slider. A **Graph** element with one plot was added to be used for the light sensor data. The **Y Minimum** and **Y Maximum** values were set to 0 and 255, respectively. A **Signal** element was added to be able to see which mode is active. When the Night mode is active the signal turns dark blue (**Color On**) and when the Night mode is inactive the signal turns yellow (**Color Off**). Finally, a slider was added to make it possible to adjust the Night mode threshold. The **Minimum** was set to 0 and the **Maximum** was set to 255. Moving the slider to the left lowers the threshold and results in the Night mode being active at brighter light levels.

When the dashboard has been set up it is time to connect the dashboard to the serial interfaces to enable communication with the target application.

Before the endpoints in the dashboard can be hooked up, the interfaces between the target board and the host computer must be configured. This example uses the DGI SPI interface and the CDC USART interface. The CDC interface will appear on the host computer as an ordinary serial COM port.

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


DGI Control Panel

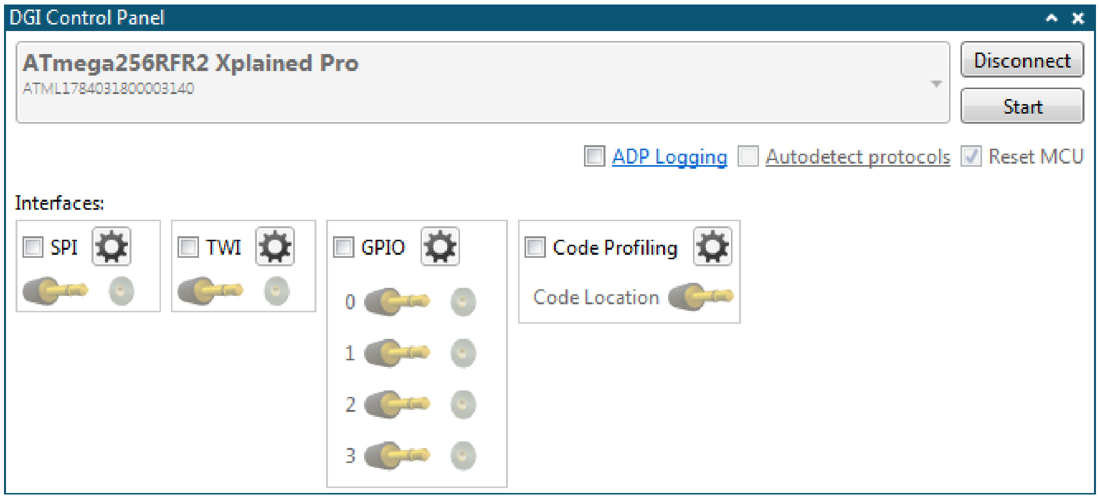
ATmega256RFR2 Xplained Pro
ATML1784031800003140

Connect
Start

☐ ADP Logging ☐ Autodetect protocols ☑ Reset MCU

Interfaces:

To do: Click **Connect** to make a connection to the DGI on the selected tool.


DGI Control Panel

ATmega256RFR2 Xplained Pro
ATML1784031800003140

Disconnect
Start

☐ ADP Logging ☐ Autodetect protocols ☑ Reset MCU

Interfaces:

☐ SPI

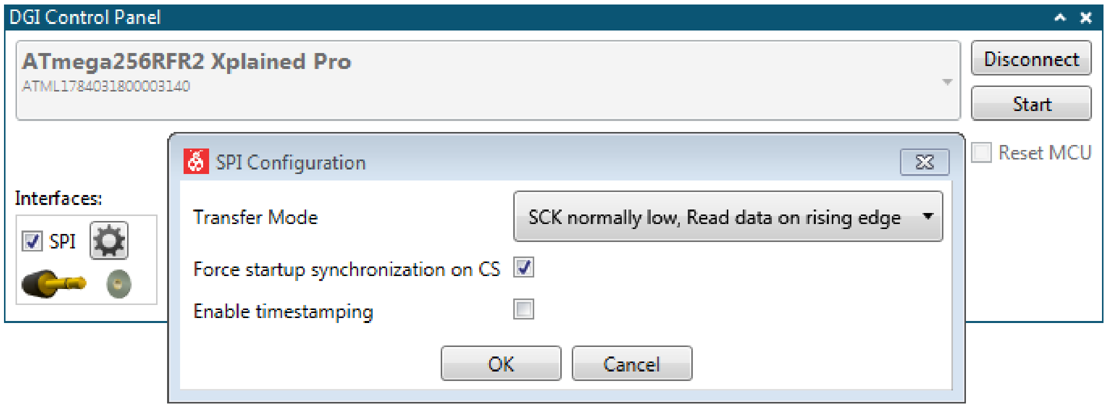
☐ TWI

☐ GPIO
0
1
2
3

☐ Code Profiling
Code Location

To do:

- Click the **SPI** checkbox
- Open the **SPI Configuration** dialog by clicking the Gear button next to the **SPI** checkbox


DGI Control Panel

ATmega256RFR2 Xplained Pro
ATML1784031800003140

Disconnect
Start

☐ Reset MCU

Interfaces:

☑ SPI

SPI Configuration

Transfer Mode: SCK normally low, Read data on rising edge

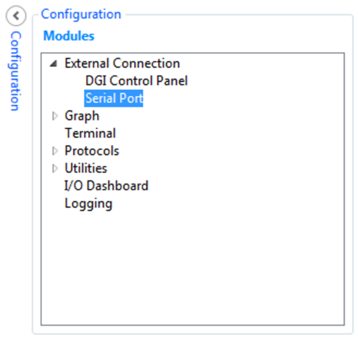
Force startup synchronization on CS ☑

Enable timestamping ☐

OK
Cancel

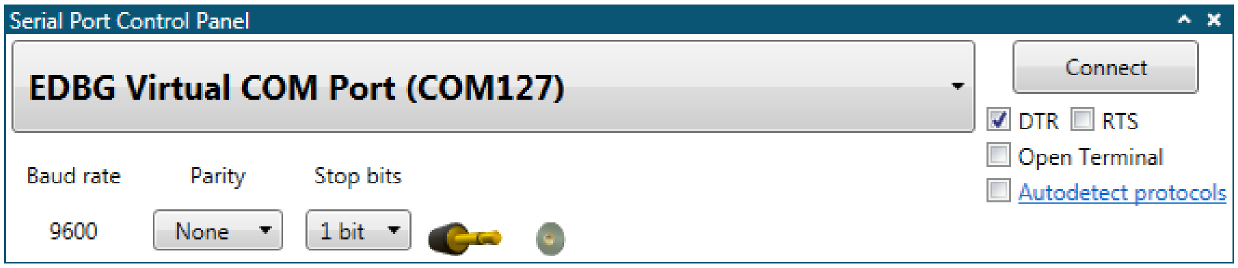
To do:

- Open the **Serial Port Control** panel found under **External Connection** in the **Modules** section of the **Configuration** tab in Data Visualizer



To do:

- Select the correct COM port corresponding to the connected kit
- Set the serial port parameters according to the application code
- Make sure the **Open Terminal** option is not checked


Serial Port Control Panel

EDBG Virtual COM Port (COM127)

Connect

☑ DTR ☐ RTS

☐ Open Terminal

☐ Autodetect protocols

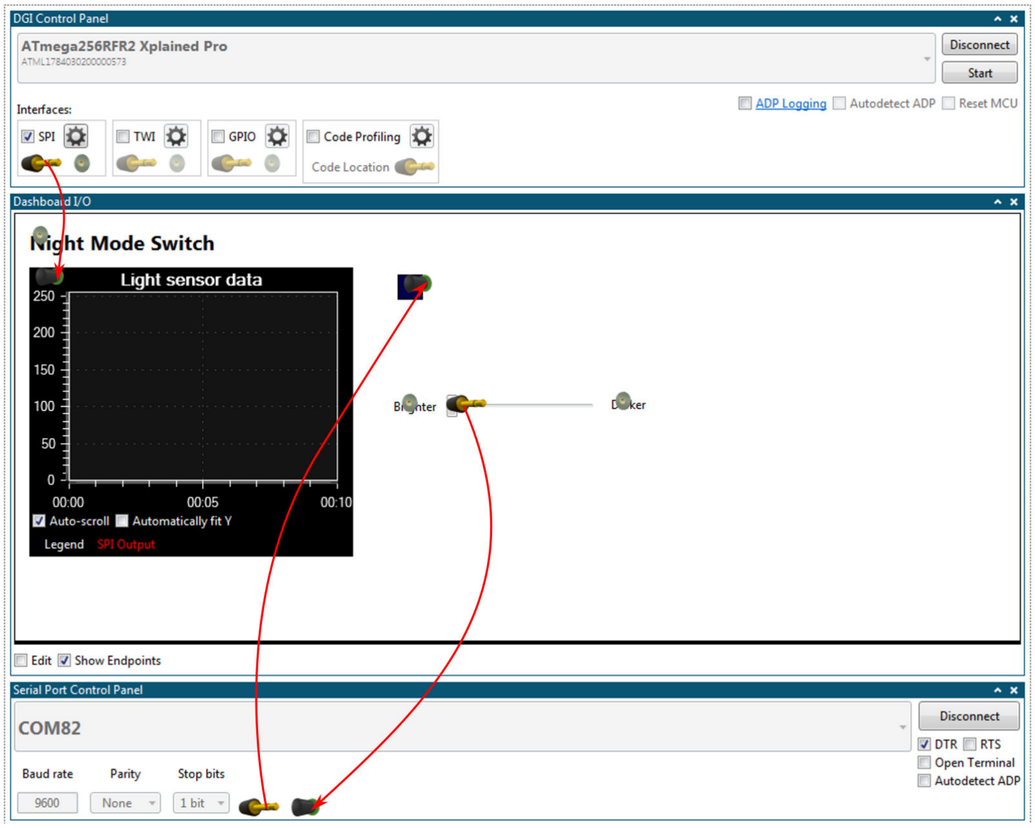
Baud rate: 9600

Parity: None

Stop bits: 1 bit

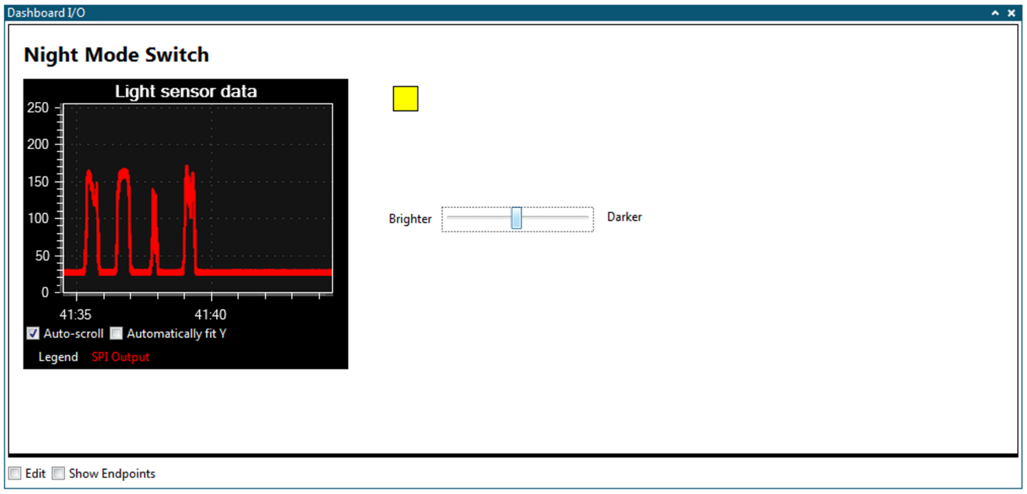
To do:

- Deselect the **Edit** option
- Click the **Show Endpoints** option
- Drag the SPI **source** to the graph **sink**
- Drag the serial port **source** to the signal **sink**
- Drag the slider **source** to the serial port **sink**



Now the dashboard is fully configured and can be used to interact with the Night mode switch application.

To do:
- Deselect the **Show Endpoints** option
- Click **Start** in the **DGI Control Panel**
- Click **Connect** in the **Serial Port Control Panel**



Now the ADC raw values are shown in the graph and the slider can be adjusted to a suitable threshold for the Night mode switch. The signal element shows the state of the switch.

4. Utilities

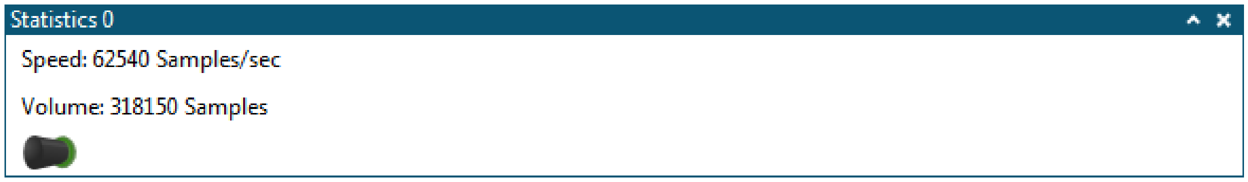
Utilities are modules that do not fit in the other categories, but are still helpful for analyzing data.

The **Samplerate Counter** provides a measure of how much data is being transmitted.

The **File Logger** module stores incoming data in a file of selectable format. The file contents can then be analyzed in another application.

4.1 Samplerate Counter

The samplerate counter module takes an incoming data stream and measures the amount of incoming samples in the stream.



Statistics 0

Speed: 62540 Samples/sec

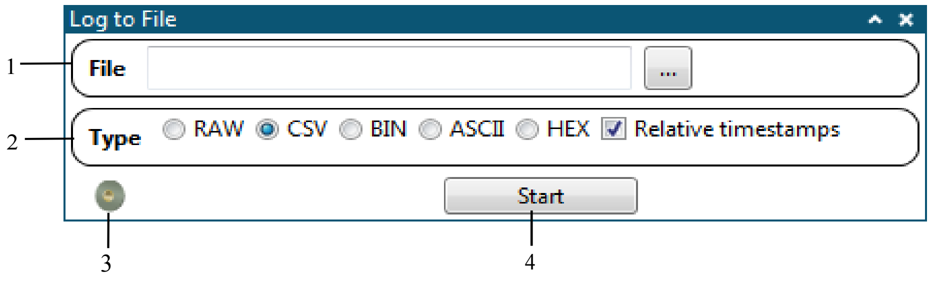
Volume: 318150 Samples

To use the samplerate counter simply connect a **source** to the **sink** of the samplerate counter module and start the data stream. The samplerate counter can be used with streams of any data type.

4.2 File Logger

The **File Logger** module logs all incoming data to a file of selectable format.

Figure 4-1. File Logging



1. **Output file** selection box. 2. **Output format** selection box. 3. Input **sink**. 4. **Start/Stop** button.

4.2.1 Logging to a Binary File

- Select output format “BIN” in the **Output format** selection box
- Set the output file by pressing the “...” button in the **Output file** selection box and selecting a path and name
- Connect an external **source** to the input **sink**
- Press the **Start** button to begin logging. The button will be replaced by the **Stop** button.
- Press the **Stop** button to close the file and end logging before inspecting the logged data

5. Protocols

Most communication interfaces use streams of bytes to transfer data. This is enough for single data values of 8-bit precision, but when multiple values are required to be transmitted over the same interface, data must be packed in a protocol. The Data Visualizer supports two such protocols.

The Data Stream protocol is using a light-weight framing format to pack several numerical values over one interface. It is only capable of handling incoming data and it only supports synchronous streams (i.e., every data packet must contain one sample from each data stream).

The Atmel Data Protocol (ADP) enables streaming of data in both directions. ADP is, in general, more flexible than the Data Stream protocol and it also supports asynchronous streams (i.e., data samples from each data stream can be sent independently).

Both protocols can be automatically detected and used to automatically configure the Data Visualizer by defining data streams, configuring a Dashboard and connect the data streams to the Dashboard elements. The main difference is that for the Data Stream protocol, the configuration settings reside in configuration files on the host computer while for ADP the configuration settings are sent from the target application to the host computer. In addition, ADP can configure Graph and Terminal views. The Graph element in the Dashboard view is supported by both protocols.

Table 5-1. Protocol Comparison

	Data Stream Protocol	Atmel Data Protocol (ADP)	Comment
Complexity	Simple/Lightweight	Complex/Flexible	
Start/Stop individual streams	-	X	
Data input	X	X	Data transmission from target application to host computer
Data output	-	X	Data transmission from host computer to target application
Support String data type	-	X	Various length strings
Configuration settings stored on host computer	X	-	Configuration of Data Visualizer GUI
Configuration settings stored in target application	-	X	Configuration of Data Visualizer GUI
Dashboard View configuration	X	X	
Graph View configuration	-	X	
Terminal View configuration	-	X	

	Data Stream Protocol	Atmel Data Protocol (ADP)	Comment
Data bursts	-	X	More than one sample of a data stream can be sent in one package
Asynchronous data streams	-	X	Samples from different data streams can be sent independently

X = Supported

- = Not supported

5.1 Data Stream Protocol

The data stream module takes an incoming raw data stream and splits it into multiple data streams. The data stream format is specified by a configuration file provided by the user.

5.1.1 Configuration Format

The configuration file is a comma-delimited text file that specifies one data variable per line. Each line starts by specifying the data format of the variable by one of the tags presented in the table below. The position of the variable in the output grid is then given by two coordinates starting at index 1. The final parameter assigns a text string to the variable.

Table 5-2. Data Stream Types

Type	Size	Tag	Example
Unsigned byte	1	B	B,1,1,Light
Signed byte	1	-B	-B,1,1,Encoder
Unsigned short	2	D	D,1,1,ADC
Signed short	2	-D	-D,1,1,ADC
Unsigned word	4	W	W,1,1,Transfer rate
Signed word	4	-W	-W,1,1,Status code
Floating point	4	F	F,1,1,Temperature
Double-precision floating point	8	DF	DF,1,1,Measurement
Grid of unsigned bytes	1 * W * D	GB<WxD>	GB<10x10>,1,1,Surface
Grid of signed bytes	1 * W * D	-GB<WxD>	-GB<10x10>,1,1,Surface
Grid of unsigned short	2 * W * D	GD<WxD>	GD<10x10>,1,1,Surface
Grid of signed short	2 * W * D	-GD<WxD>	-GD<10x10>,1,1,Surface
Grid of unsigned word	4 * W * D	GW<WxD>	GW<10x10>,1,1,Surface
Grid of signed word	4 * W * D	-GW<WxD>	-GW<10x10>,1,1,Surface

Type	Size	Tag	Example
Grid of floating point	8 * W * D	GF<WxD>	GF<10x10>,1,1,Surface
Grid of double-precision floating point	8 * W * D	GDF<WxD>	GDF<10x10>,1,1,Surface

This is an example configuration:

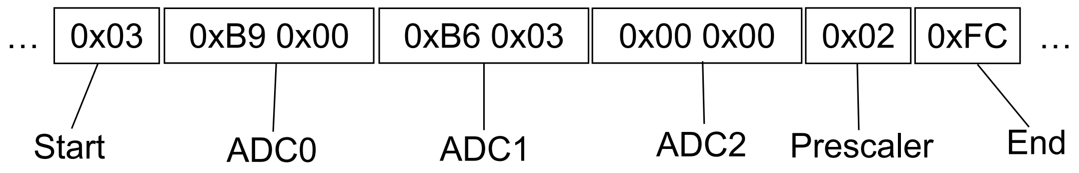
```
D,1,1,ADC0
D,1,2,ADC1
D,1,3,ADC2
B,2,1,Prescaler
```

5.1.2 Stream Format

The data stream is processed in the same order as the configuration file specifies. All data must be given as little endian values, meaning that the lowest byte must be sent first. Additionally, a wrapper consisting of one byte before and one byte after the data stream variables must be added. This wrapper is used by the interpreter to synchronize to the data stream. The start byte can be of an arbitrary value except 0x5F, which is reserved for Auto-Configuration, but the end byte must be the inverse of the Start byte. For more information on the Auto-Configuration feature see Auto-Configuration and Auto-Configuration Example. The configuration file shall not define the start and end bytes.

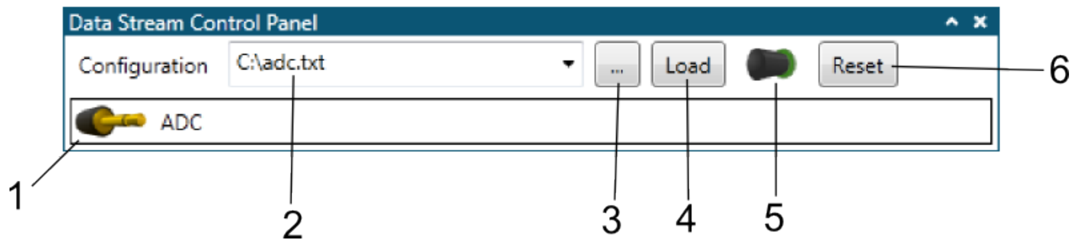
Consider the example configuration given in the previous section. The figure below gives an example raw data transmission where ADC0 is 185, ADC1 is 950, ADC2 is 0, and Prescaler is 2.

Figure 5-1. Data Streamer



5.1.3 Basic Usage

Figure 5-2. Data Streamer



- Press the Open file button (3)
- Select the configuration file
- Click the Load configuration button (4)
- Connect the input sink (5)
- Connect one or more output source to a desired sink (1)

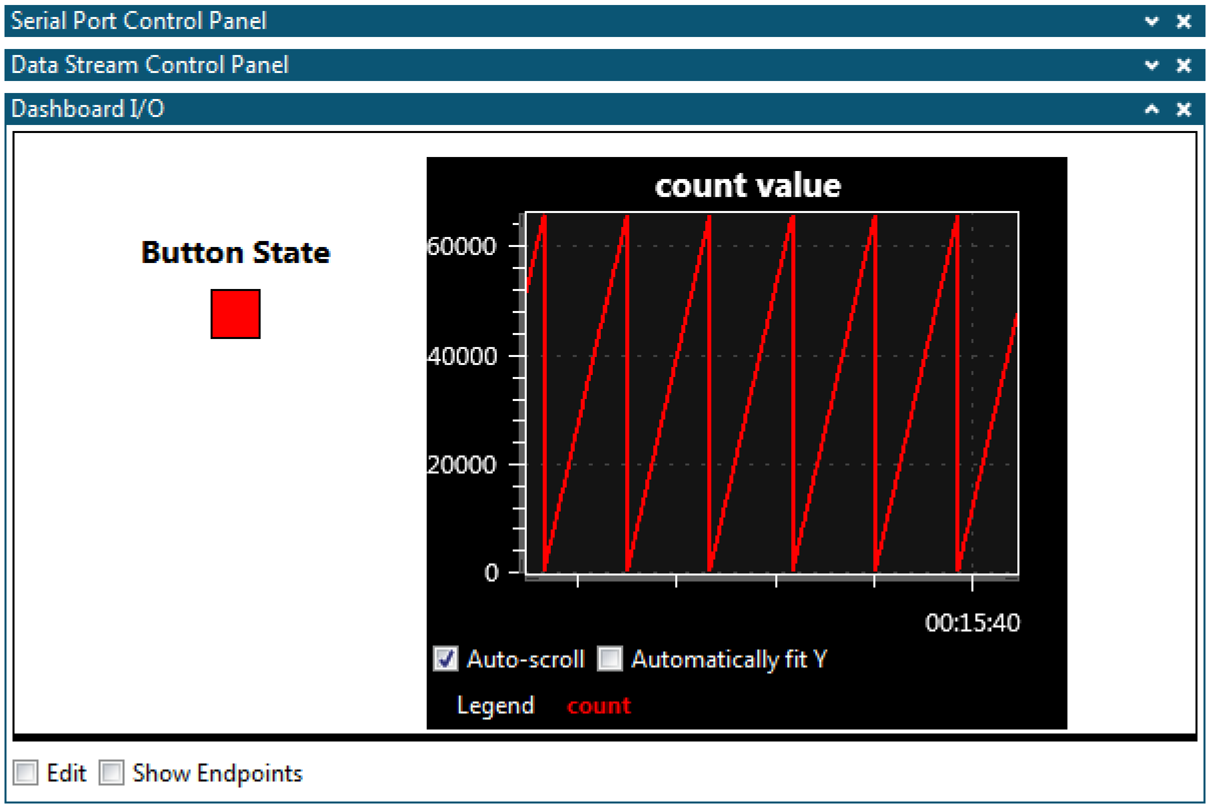
5.1.4 Auto-Configuration

The Data Stream format can be used to automatically configure the Data Visualizer based on some pre-defined configuration files. This differs from the Atmel Data Protocol (ADP) auto-configuration where the configuration settings are stored in the target application and sent to the host upon request. The Data

Stream auto-configuration is limited to setting up a data stream, configure a Dashboard View and connect the data stream output sources to the dashboard elements.

5.1.5 Auto-Configuration Example

The purpose of this example is to show how to set up the configuration files required to automatically configure a dashboard with a signal showing the state of a pushbutton on the target hardware and a graph showing the value of a counter in the target application. The screenshot below shows the final dashboard.



The target code used in this example can be found in Auto-Configuration Example Code.

Three files are required by the Data Visualizer to enable Auto-Configuration, a .ds file describing the data stream content, a .db file describing the dashboard and a .sc file describing the connections between the data stream content and the dashboard elements. All files should be named "C0FFEEC0FFEEC0FFEEC0FFEE" before the extension to match the Auto Configuration ID sent by the application code.

The example application generates a data stream with two stream components, an unsigned 16-bit value and an unsigned 8-bit value.

To do:

- Create a file called "C0FFEEC0FFEEC0FFEEC0FFEE.ds" and add the content below

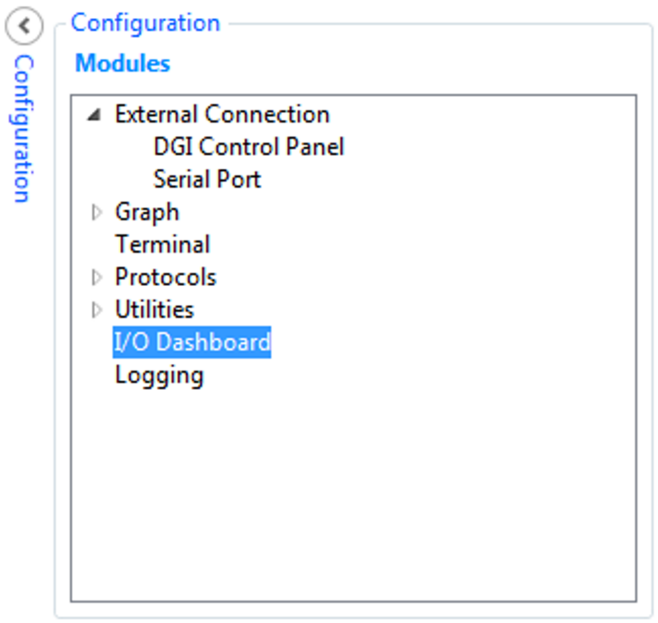
```
D,1,1,count
B,2,1,button
```

The .ds file will create a table with two sources from the data stream. Column 1 row 1 will contain a source with unsigned 16-bit values named "count" and column 2, row 1 will contain a source with unsigned 8-bit values named "button". For more information on the .ds file format see Configuration Format.

The easiest way to set up a dashboard for the Auto-Configuration is to draw it in the Data Visualizer GUI and save the configuration to file.

To do:

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



To do:

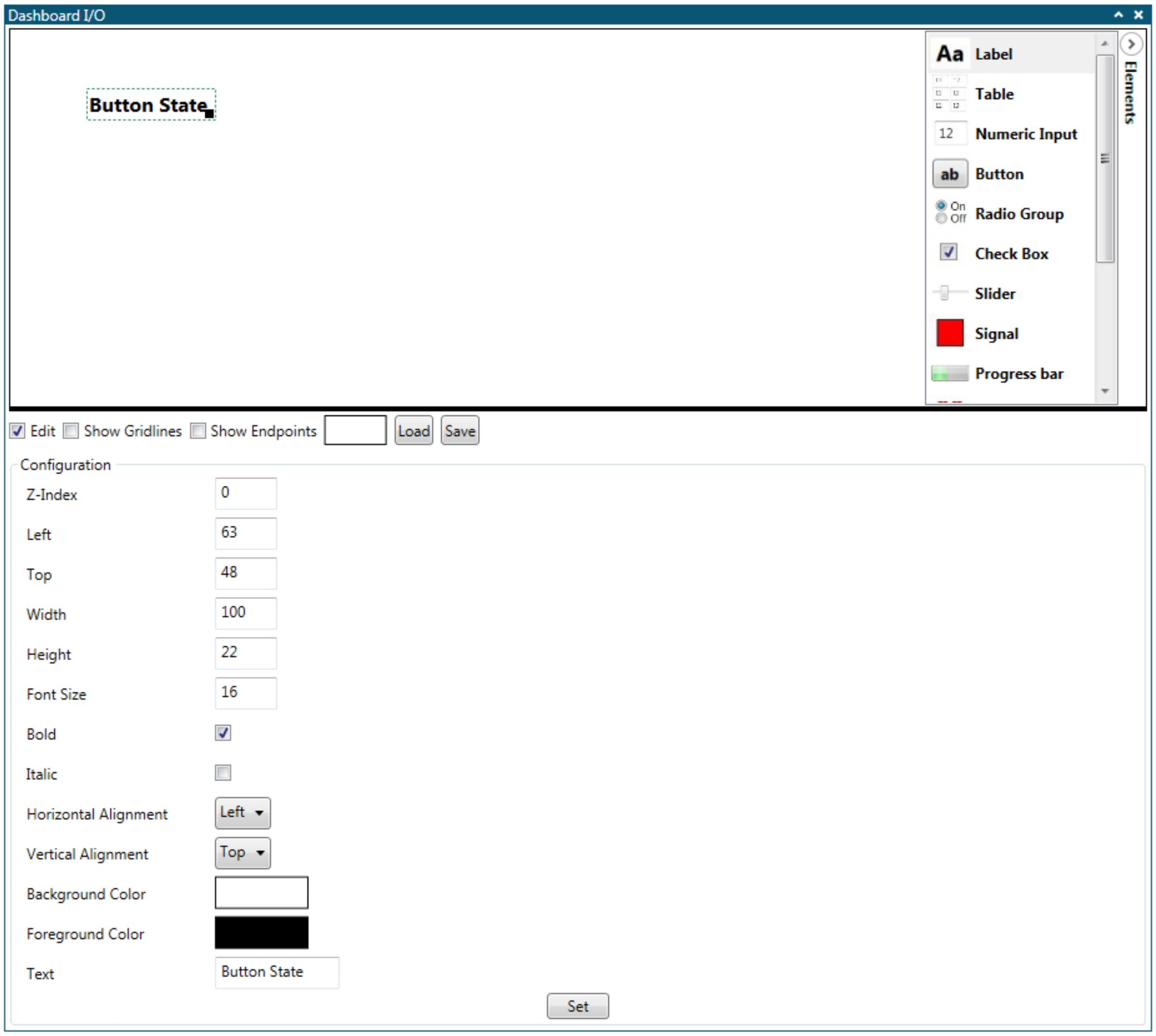
- Enable editing the dashboard by clicking the **Edit** option in the lower left corner of the **Dashboard I/O** module
- Open the **Elements** panel in the upper right corner of the dashboard and drag elements onto the dashboard.

Tip: To remove an element from the dashboard, select it by left-clicking it, and then right-click the element.

Tip: Changing the parameters in the **Configuration** section will not take effect until the **Set** button is clicked.

To do:

- Drag a **Label** element to the dashboard
- Select the **Label** element and set **Font Size** to 16, Tick the **Bold** option, set **Text** field to "Button State" and set **Width** to 100.
- Push **Set**

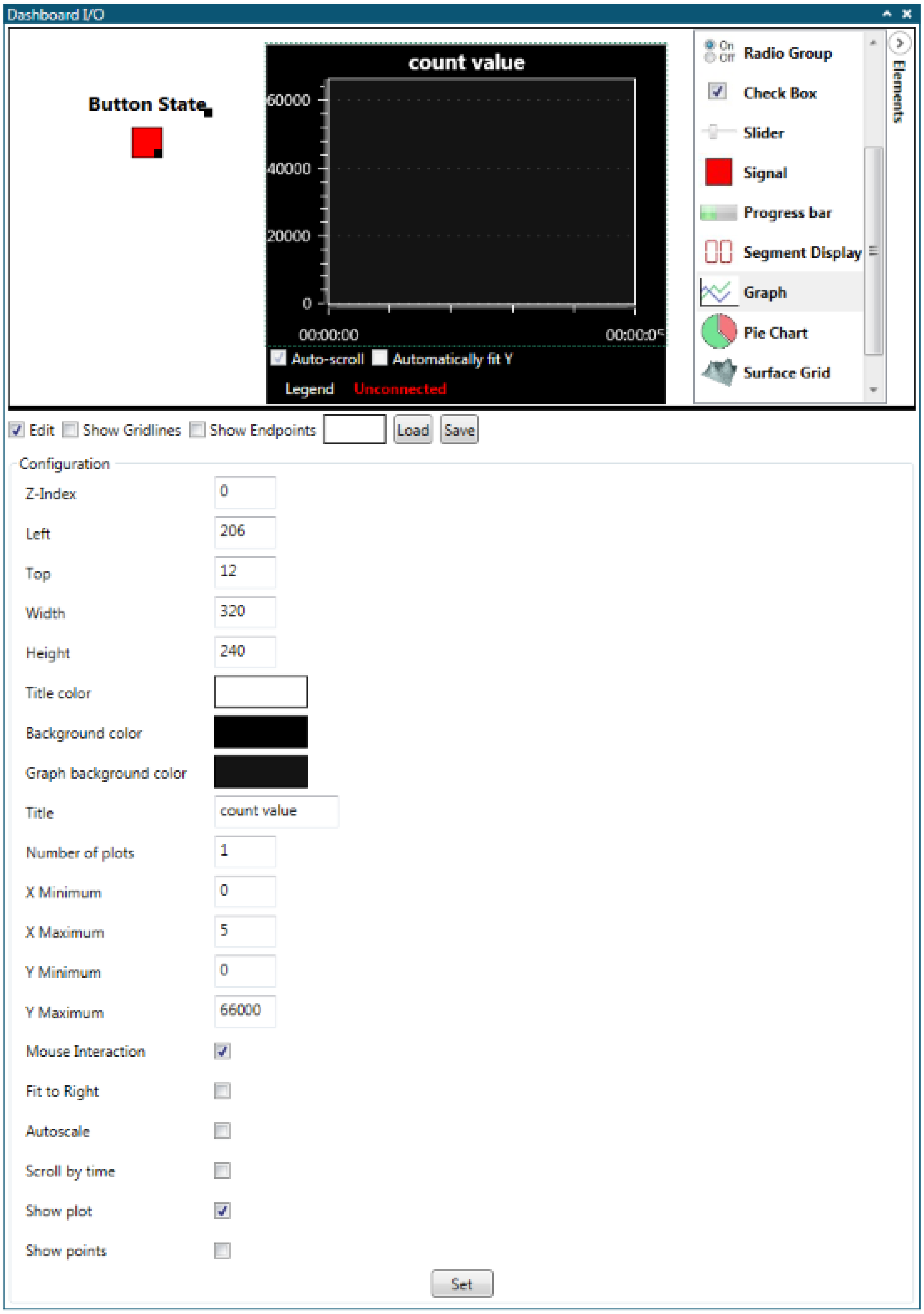


Configuration

Z-Index	0
Left	63
Top	48
Width	100
Height	22
Font Size	16
Bold	✓
Italic	
Horizontal Alignment	Left
Vertical Alignment	Top
Background Color	⬜
Foreground Color	⬛
Text	Button State

To do:

- Drag a **Signal** element to the Dashboard and put it below the Button State label
- Drag a **Graph** element to the Dashboard and put it to the right of the Button State label and signal
- Select the **Graph** element and set **Title** to “count value”, **X Maximum** to 5, **Y Maximum** to 66000
- Click **Set**

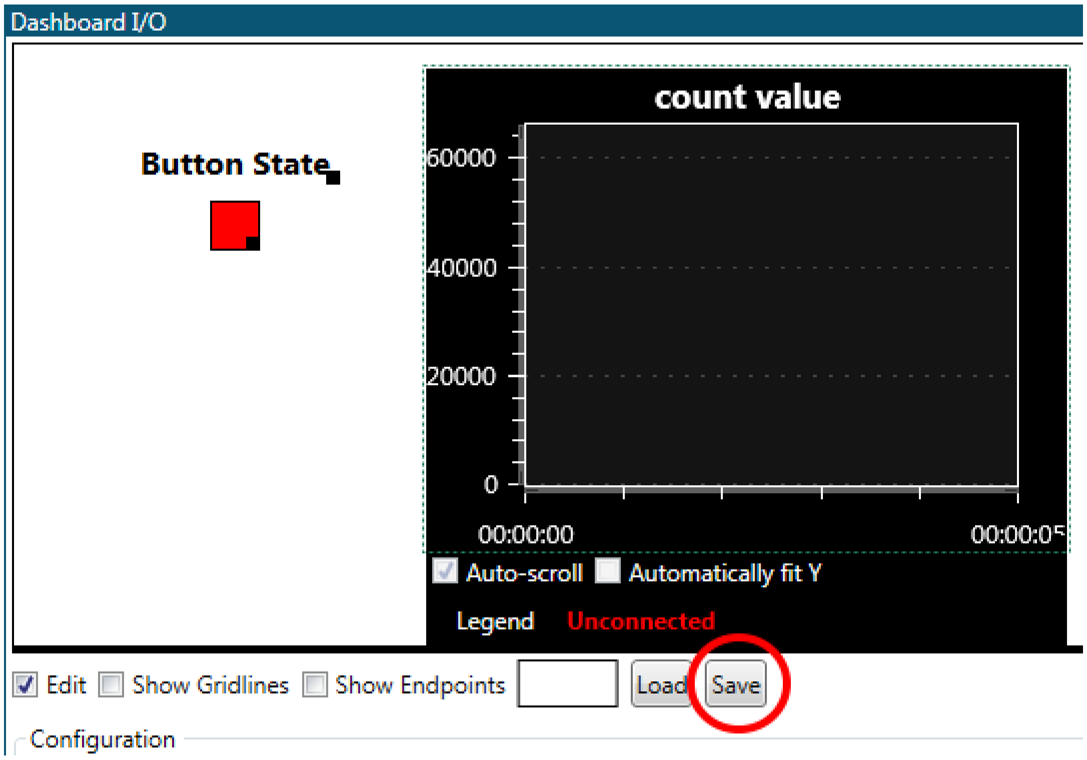


Configuration

Z-Index	0
Left	206
Top	12
Width	320
Height	240
Title color	⬜
Background color	⬛
Graph background color	⬛
Title	count value
Number of plots	1
X Minimum	0
X Maximum	5
Y Minimum	0
Y Maximum	66000
Mouse Interaction	✓
Fit to Right	
Autoscale	
Scroll by time	
Show plot	✓
Show points	

To do:

- Click **Save**



To do:

- Save the Dashboard configuration as “C0FFEEC0FFEEC0FFEEC0FFEE.db” in the same folder as the .ds file

The .db file content should have content similar to:

```
{
0,
'\0',
0, 255, 255, 255,
44, 1,
};

{
0, // Dashboard ID
0, // Element ID
DB_TYPE_LABEL, // Element Type
0, // Z-Index (GUI stack order)
63, 0, // X-coordinate
48, 0, // Y-coordinate
100, 0, // Width
22, 0, // Height
16, // Font Size
1,
0, // Horizontal Alignment
0, // Vertical Alignment
0, 255, 255, 255, // Background Color
255, 0, 0, 0, // Foreground Color
'B', 'u', 't', 't', 'o', 'n', ' ', 'S', 't', 'a', 't', 'e', '\0', // Text
};

{
0, // Dashboard ID
1, // Element ID
DB_TYPE_SIGNAL, // Element Type
0, // Z-Index (GUI stack order)
98, 0, // X-coordinate
78, 0, // Y-coordinate
25, 0, // Width
25, 0, // Height
255, 0, 255, 0, // Color On
255, 255, 0, 0, // Color Off
};

{
0, // Dashboard ID
2, // Element ID
DB_TYPE_GRAPH, // Element Type
0, // Z-Index (GUI stack order)
206, 0, // X-coordinate
12, 0, // Y-coordinate
64, 1, // Width
240, 0, // Height
255, 255, 255, // Title color
0, 0, 0, // Background color
20, 20, 20, // Graph background color
'c', 'o', 'u', 'n', 't', ' ', 'v', 'a', 'l', 'u', 'e', '\0', // Title
1, // Number of plots
0,0,0,0, // X Minimum
0,0,160,64, // X Maximum
0,0,0,0, // Y Minimum
0,232,128,71, // Y Maximum
1,
1,
};
```

For more information on the Dashboard elements, see Element Types.

Finally, the configuration file to connect the data stream sources defined in the .ds file to the Dashboard elements defined in the .db file must be made.

To do:

- Create a file called “C0FFEEC0FFEEC0FFEEC0FFEE.sc” in the same folder as the other config files and add the content below:

```
button,1
count,2
```

The .sc file will connect the stream source named “button” to the Dashboard element with ID 1 (the Signal element) and the stream source named “count” to the Dashboard element with ID 2 (the Graph element).

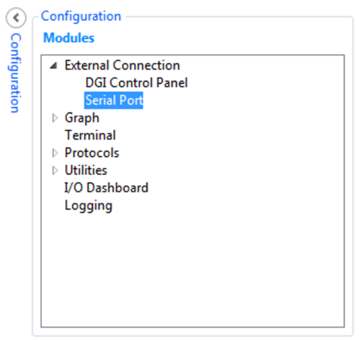
Then it is time to run the example.

To do:

- Close the Dashboard panel

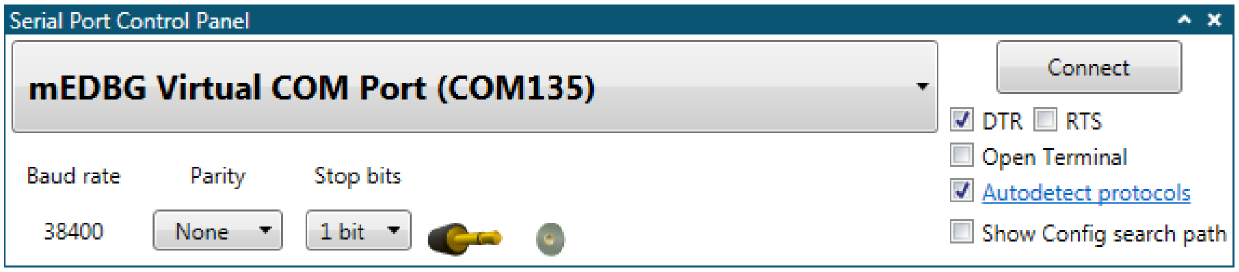
To do:

- Open the **Serial Port Control** panel found under **External Connection** in the **Modules** section of the **Configuration** tab in Data Visualizer



To do:

- Select the correct COM port corresponding to the connected kit
- Make sure **Autodetect protocols** option is checked and the **Parity** and **Stop bits** configurations are set according to the target application. The **Baud rate** will be detected automatically.



mEDBG Virtual COM Port (COM135)

Baud rate 38400

Parity None

Stop bits 1 bit

[✓] DTR [ ] RTS

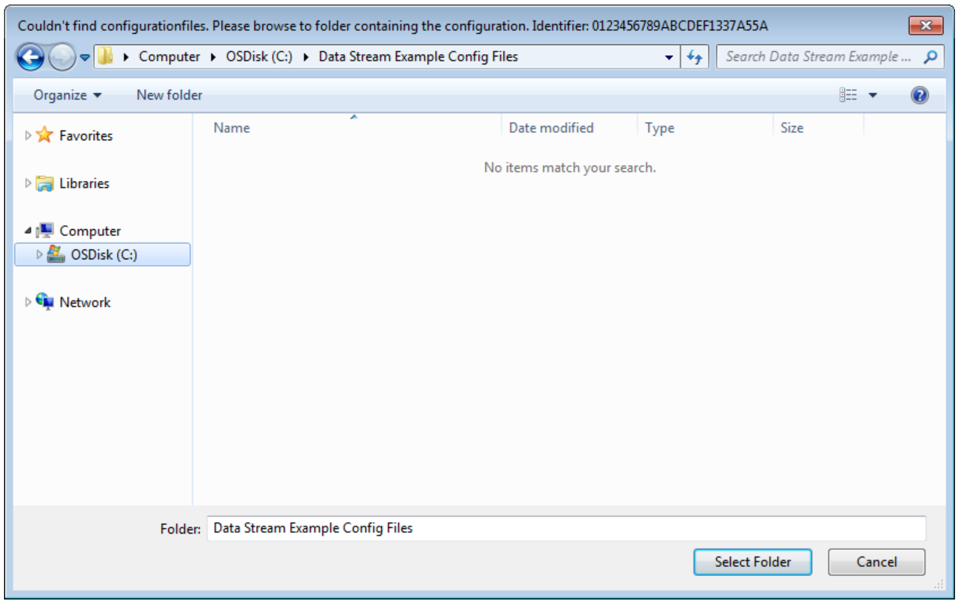
[ ] Open Terminal

[✓] Autodetect protocols

[ ] Show Config search path

To do:

- Click **Connect**
- If Data Visualizer cannot find the configuration files for the detected Auto-Configuration ID it will show a pop-up asking for the path to the configuration files. Browse to the folder where the configuration files reside and click **OK**

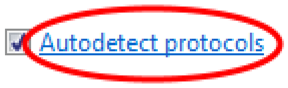

Couldn't find configurationfiles. Please browse to folder containing the configuration. Identifier: 0123456789ABCDEF1337A55A

Computer ▸ OSDisk (C:) ▸ Data Stream Example Config Files
Search Data Stream Example ...
Organize
New folder
Favorites
Libraries
Computer
OSDisk (C:)
Network
Name
Date modified
Type
Size
No items match your search.
Select Folder
Cancel

Folder: Data Stream Example Config Files

After selecting a folder, the folder will be APPENDED to the Auto-Configuration search path.

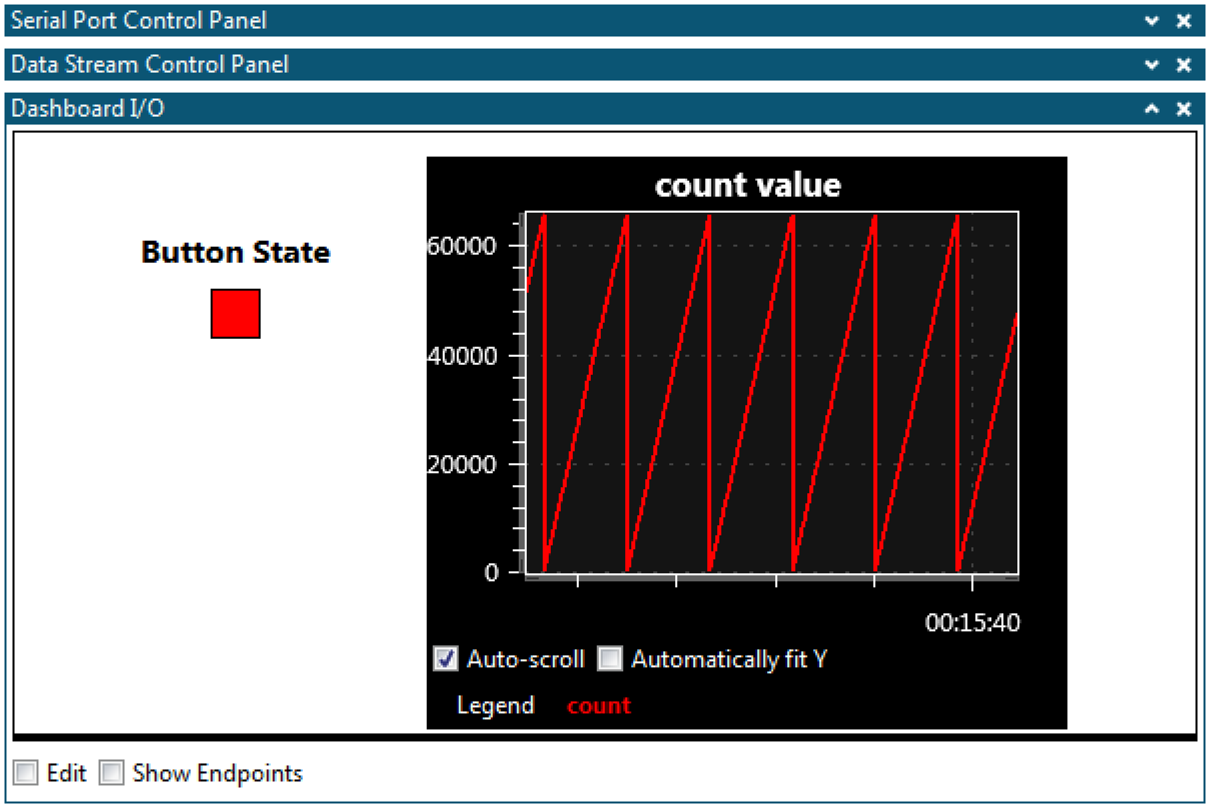
Tip: To reset the search path and select a new single folder as the search path, click the link on the **Autodetect protocols** option text.



Data Visualizer will then pop up a browser dialog asking for the path to the folder where the configuration files reside. The original search path will be CLEARED and the newly selected folder will be set as search path.

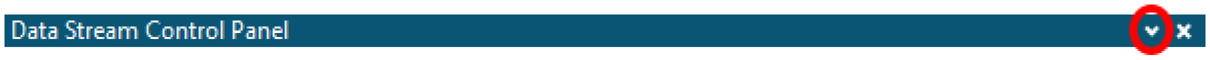
Important: All three configuration files must reside in the same folder.

After connecting and detecting the Auto-Configuration ID the Data Visualizer should create a Data Stream Control Panel and a Dashboard I/O looking something like the image below.

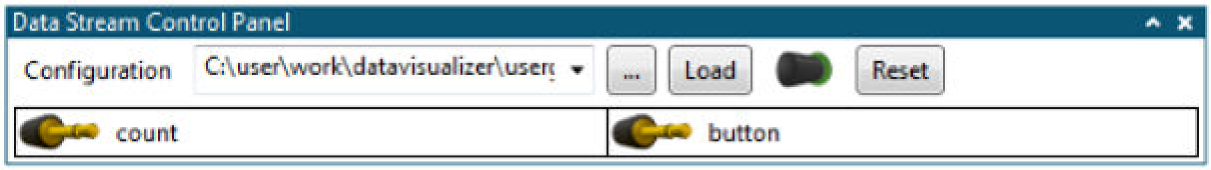


The Graph element shows a running sawtooth signal which represents the counter continuously counting up until it wraps. The Signal element shows the state of the push button on the ATtiny104 Xplained Nano board. Pushing the button changes the color of the Signal element from red to green.

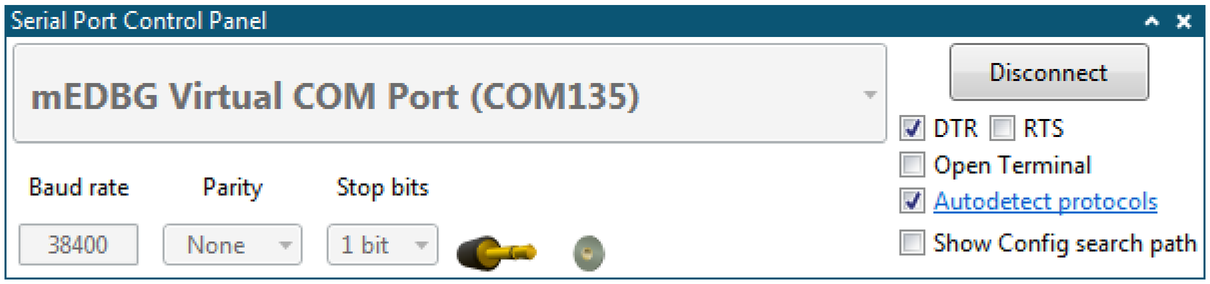
Expanding the Data Stream Control Panel by clicking the down arrow to the right in the panel shows the content of the automatically configured Data Stream.



The stream has two sources, one for the counter value and one for the button state.



Expanding the Serial Port Control Panel shows that Data Visualizer detected the baud rate to be 38400.



5.1.6 Auto-Configuration Format

If the start byte of a Data Stream packet is 0x5F then this packet is a special Configuration packet.

Table 5-3. Configuration Packet Format

Field	Size	Values	Description
Start token	1 byte	0x5F	Start token reserved for configuration packets.
Checksum format	4 bytes	0xB4 0x00 0x86 0x4A	Specifies the checksum format to be used. Currently only LRC8 is supported.
Configuration identifier	12 bytes	Any value	Unique identifier for the configuration.
Checksum	1 byte	Checksum according to Checksum format	Currently only LRC8 checksum format is supported. This is the XOR sum of the packet excluding the start token, the checksum itself and the end token.
End token	1 byte	0xA0	Following the Data Stream format the end token is the inverse of the start token.

The identifier given in the Configuration packet is used by Data Visualizer to look-up the corresponding configuration files used to configure the Data Visualizer. Three configuration files are needed:

- A .ds file defining the Data Stream. This is a normal Data Stream format file and follows the format given in Configuration Format.
- A .db file defining the Dashboard. This file follows the format of the files generated when saving a Dashboard to file, see Edit Panel.
- A .sc file defining the connections between the Data Stream components defined in the .ds file and the elements of the Dashboard defined in the .db file. The format is defined in Signal Connections File Format.

All three configuration files should have a name equal to the hex values of each Configuration identifier byte. As an example, a Configuration identifier of [0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0xDE, 0xF0, 0x12, 0x34, 0x56, 0x78] corresponds to configuration files named 123456789ABCDEF012345678.sc, 123456789ABCDEF012345678.db and 123456789ABCDEF012345678.ds.

The Data Streamer Auto-Configuration feature is available both in the DGI Control Panel for all DGI serial interfaces and in the Serial Port Control Panel for COM ports and Virtual COM ports (CDC interface)

To enable Auto-configuration the **Autodetect protocols** option must be enabled.

☑ Autodetect protocols

After pushing **Connect** the Data Visualizer will enable all interfaces while it looks for the ADP handshake message or a Data Stream Configuration packet. If an ADP handshake message is received, the Data Visualizer will request configuration information from the target application. If a Data Stream Configuration packet is found, the Data Visualizer searches through the folders in the Auto-Configuration search path looking for configuration files with names matching the detected ID.

Important: To make sure the Data Visualizer detects the Data Stream Configuration packet, it must be sent by the target at least twice per second.

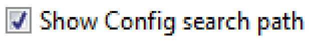
Important: Asynchronous serial protocols (e.g., UART protocols used by DGI USART and CDC Virtual COM port interfaces) use the following baud rates for auto-detection:

Table 5-4. Baud Rates Used on Asynchronous Interfaces for Auto-Detection of Protocols

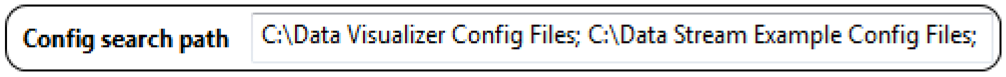
Baud Rate
9600
19200
38400
57600
115200
230400
500000
1000000
2000000

Using any baud rates not in the table will not work for auto-detection of protocols over asynchronous interfaces (DGI UART and Serial port/CDC Virtual COM port).

Tip: To see the current search path used by Data Visualizer to look for configuration files, check the **Show Config search path** option.


☑ Show Config search path

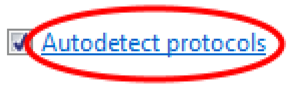
The search path is a semicolon separated list of paths. When Data Visualizer detects an Auto-Configuration ID, it will search through the paths in the list looking for configuration files with the correct file names.


Config search path: C:\Data Visualizer Config Files; C:\Data Stream Example Config Files;

If the Data Visualizer cannot find any valid configuration files it will show a browser dialog window asking for the path to the folder where the correct configuration files reside.

After selecting a folder, the folder will be APPENDED to the Auto-Configuration search path.

Tip: To reset the search path and select a new single folder as the search path, click the link on the **Autodetect protocols** option text.


☑ Autodetect protocols

Data Visualizer will then pop up a browser dialog asking for the path to the folder where the configuration files reside. The original search path will be CLEARED and the newly selected folder will be set as search path.

Important: All three configuration files must reside in the same folder.

5.1.7 Signal Connections File Format

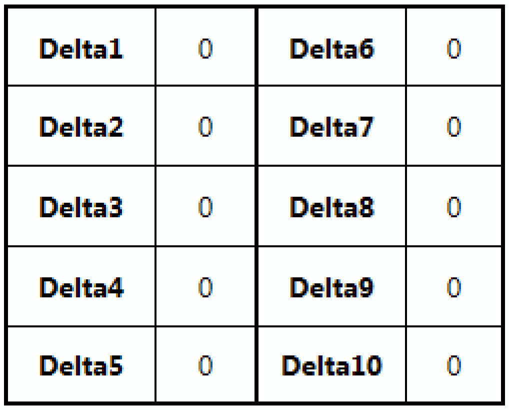
A signal connections file has the file extension .sc and it specifies the connections between the data stream sources defined in a .ds file and the GUI elements in the Dashboard defined in a .db file. The .sc file format is a comma-delimited text file specifying one connection per line. Each line follows the format <Stream name>, <Element ID>. The Stream name is defined in the .ds file and is the text string assigned to each data variable. The Element ID is defined in the .db file for each Dashboard Element.

An example of a .sc file content:

```
Plane,0
Delta1,2
Delta2,2
```

A stream called Plane is connected to an Element with ID 0 and both streams Delta1 and Delta2 are connected to an Element with ID 2. For a full auto-configuration example, see Auto-Configuration Example.

The Table Element (see Table) has some extra parameters in addition to the Stream name and Element ID. The column and row of the cell to connect the signal to is given by appending (Column:<column number>;Row:<row number>) to the lines in the .sc file. The upper left cell is specified by column 0 and row 0. As an example the table



Delta1	0	Delta6	0
Delta2	0	Delta7	0
Delta3	0	Delta8	0
Delta4	0	Delta9	0
Delta5	0	Delta10	0

is connected to sources by the following .sc file content:

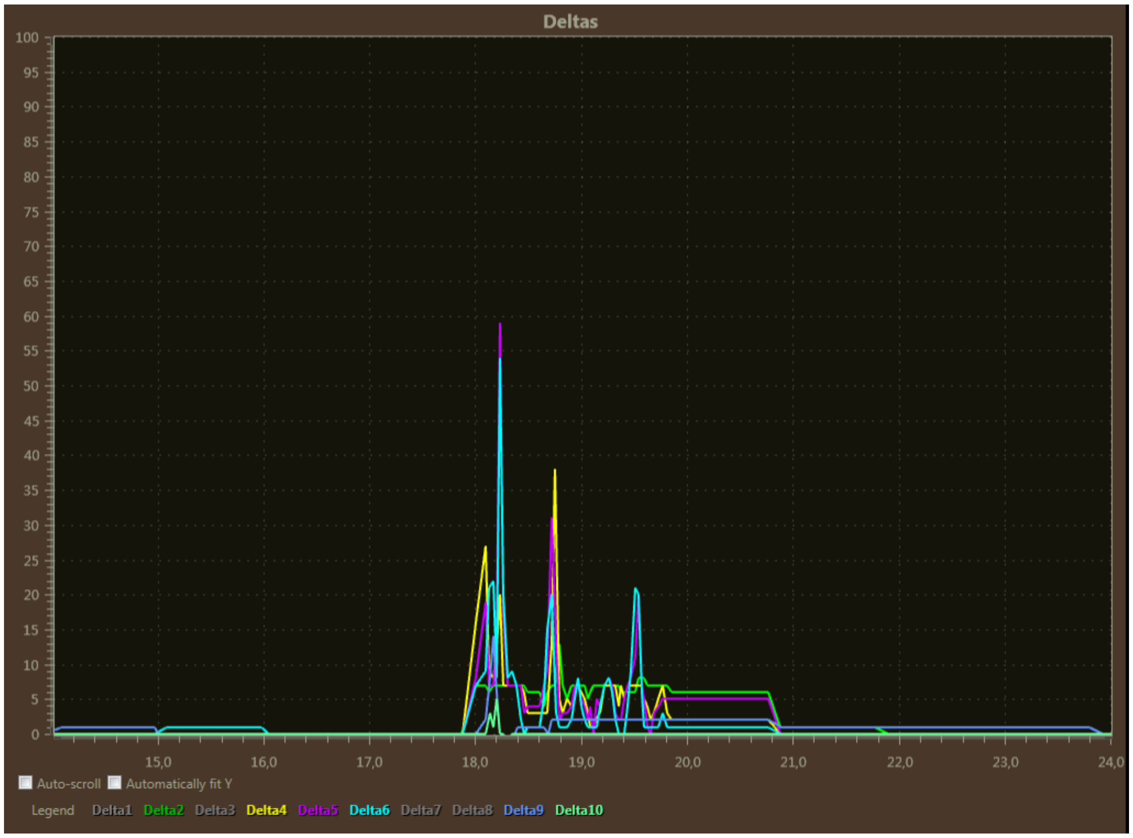
```
Delta1,2(Column:1;Row:1)
Delta2,2(Column:1;Row:2)
Delta3,2(Column:1;Row:3)
Delta4,2(Column:1;Row:4)
Delta5,2(Column:1;Row:5)
Delta6,2(Column:1;Row:6)
Delta7,2(Column:1;Row:7)
Delta8,2(Column:1;Row:8)
Delta9,2(Column:1;Row:9)
Delta10,2(Column:1;Row:10)
```

Note that in the example the Table element is in Auto-Labels mode so each cell has two fields; a label to the left and the actual data to the right. For more information, see Auto Labels.

The Graph Element (see Graph) also supports an extra parameter in addition to the Stream name and Element ID. By default, all plots are visible in the Graph element but they can be hidden or shown by the user clicking the legend in the Graph view corresponding to a plot. The extra parameter supported makes it possible to change the default behavior to hide plots when doing auto-configuration. This is done by appending (visible:0) to the .sc file. As an example, see the following .sc file

```
Delta1,2(visible:0)
Delta2,2
Delta3,2(visible:0)
Delta4,2
Delta5,2
Delta6,2
Delta7,2(visible:0)
Delta8,2(visible:0)
Delta9,2
Delta10,2
```

results in the graph element looking like this:



In the graph the plots named Delta1, Delta3, Delta7, and Delta8 are all hidden and the legends are gray. The user can enable them by clicking the legends. In the same way the visible plots can be hidden by the user clicking the corresponding legends.

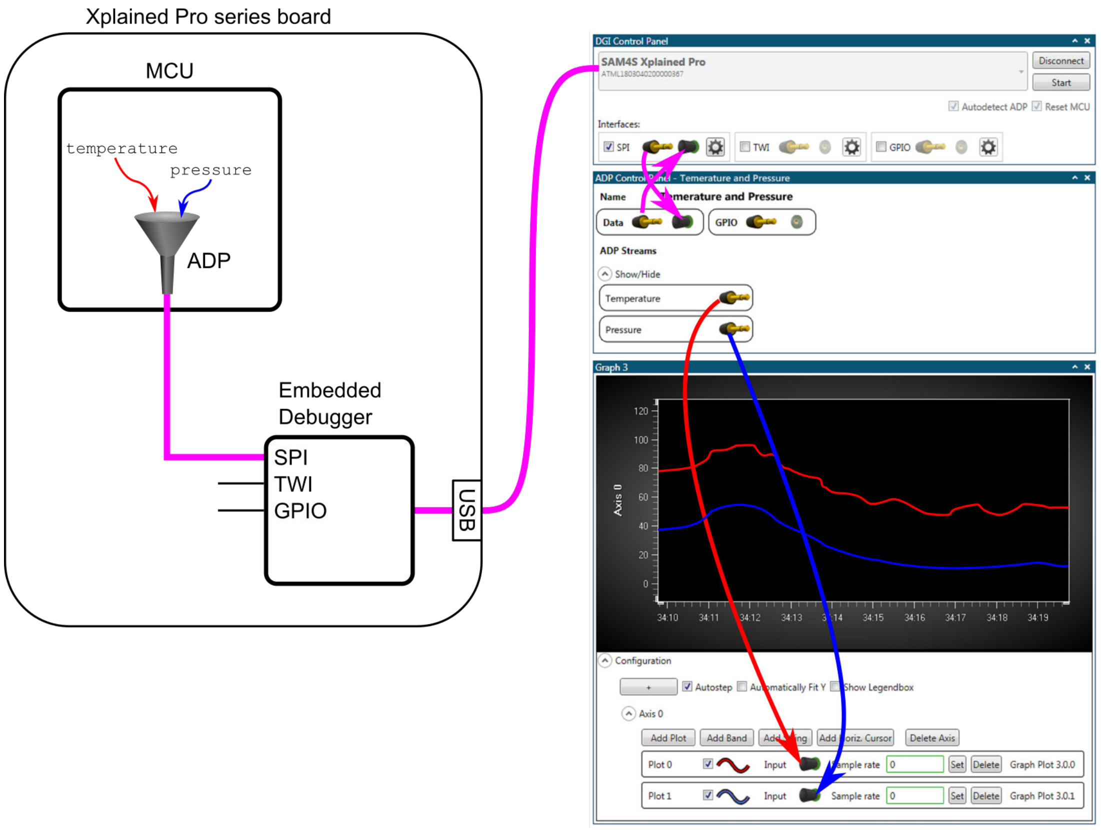
5.2 Atmel Data Protocol

5.2.1 Transfer using Atmel Data Protocol

The Atmel Data Protocol (ADP) is a content independent protocol intended for transferring data from a target MCU to a host PC through an EDBG-based debugger (EDBG, Atmel-ICE, Power Debugger) using the Data Gateway Interface (DGI, see Embedded Debugger’s Data Gateway Interface) or directly to the host computer using a serial port. ADP is content independent and the transfer through the debugger is transparent, meaning that the content is not interpreted by the debugger.

Transferring a single value is quite simple. But to transfer more than one value, they have to be wrapped in some kind of protocol that both the sender and receiver understands. ADP is such a protocol. If the MCU wraps all its data into an ADP packet, it can be decoded in the Data Visualizer and split into separate data streams.

Figure 5-3. An ADP Transfer



In the figure above, the MCU packs a *temperature* and a *pressure* variable inside an ADP packet. In the Data Visualizer, the **SPI** endpoints in the **DGI Control Panel** are now connected to the Data endpoints of an **ADP Control Panel**. The **ADP Control Panel** will decode the packets into separate temperature and pressure data streams. They can then be connected to two plot lines in the **Graph** module.

The ADP protocol supports data transfer in both directions. In addition, the MCU can send configuration packets describing what modules should be opened in the Data Visualizer, and how to connect them. When the target board is connected to the host computer everything will be configured automatically.

5.2.2 ADP Example

For an example of ADP protocol usage, the ADP example application for SAM D21 Xplained Pro can be used. This example can be found in Atmel Software Framework (ASF) in Atmel Studio. It uses a SAM D21 Xplained Pro together with an I/O Xplained Pro board.

This example uses the Data Gateway Interface (DGI), see the Embedded Debugger's Data Gateway Interface on the Embedded Debugger (EDBG), but any serial port is sufficient.

5.2.2.1 Requirements

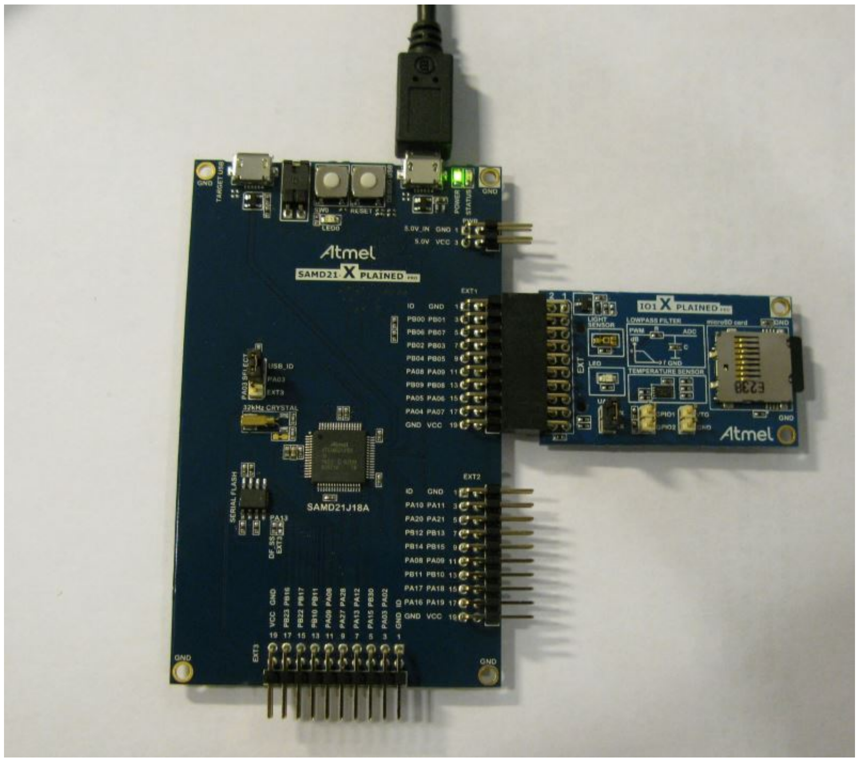
- Host computer with Atmel Studio 7 (or later) installed (Data Visualizer is included)
- SAM D21 Xplained Pro kit
- I/O1 Xplained Pro extension

5.2.2.2 Hardware Setup

To run the example the following hardware setup is required:

- I/O1 Xplained Pro extension connected to SAM D21 Xplained Pro EXT1 connector
- USB cable connected from host PC to DEBUG USB connector on SAM D21 Xplained Pro

A picture of the setup is shown below.



5.2.2.3 Run Example

To run the ADP example follow the steps below.

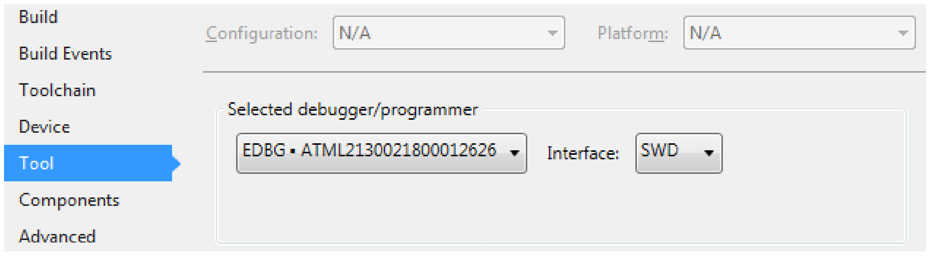
To do:

- Open Atmel Studio
- Select File → New → Example Project
- Browse to, or search for the *ADP example application - SAM D21 Xplained Pro* and select it
- Choose the preferred folder and give the project a name, then click **OK** to create the project

The project will be generated, then it is just a matter of compiling it and programming it into the target board.

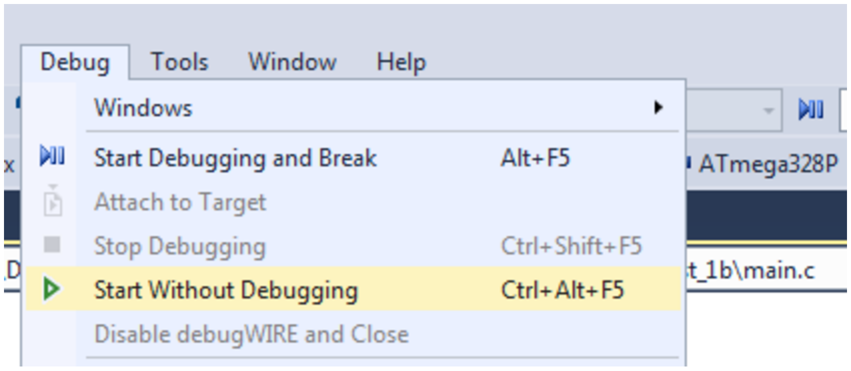
To do:

- 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

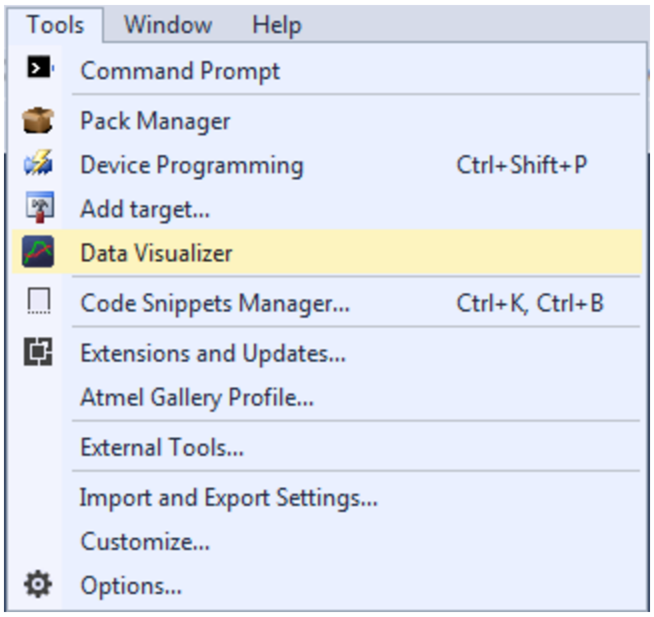


To do:

- Click **Start Without Debugging** (Debug → Start Without Debugging)

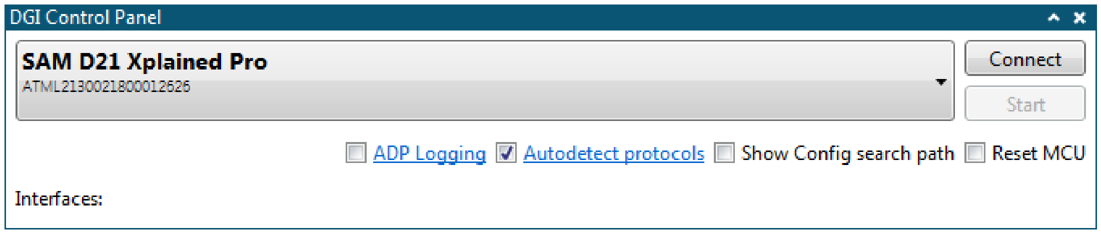

Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Ctrl+Shift+F5
Start Without Debugging Ctrl+Alt+F5
Disable debugWIRE and Close
ATmega328P
t_1b\main.c

To do: Open the Data Visualizer as an extension inside Atmel Studio by selecting it in the **Tools** menu.



To do:

- In the DGI Control Panel, select *SAM D21 Xplained Pro*
- Select the **Autodetect protocols** box
- Click **Connect**

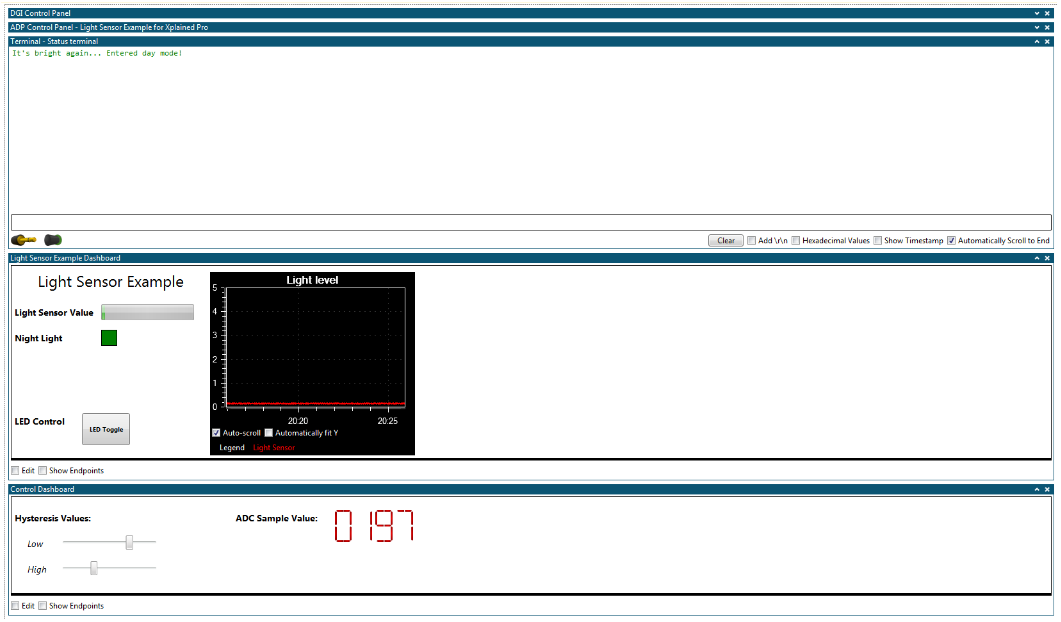

DGI Control Panel

SAM D21 Xplained Pro
ATML2130021800012626

☐ ADP Logging ☑ Autodetect protocols ☐ Show Config search path ☐ Reset MCU

Interfaces:

You should see something like the screenshot below in the Data Visualizer.



```
It's bright again... Entered day mode!
```

```
0 197
```

5.2.2.4 How it Works

As the code for the ADP example is quite extensive, it will not make sense to list it or describe all the details. Especially, details on how to set up the required peripherals on the ATSAMD21 will be left out. The ADP messages required to create the ADP example dashboard will be detailed in the following chapters. Note that, after each message sent to the computer, the target (the SAM D21 device) waits for a MSG_CONF_ACK (MSG_CONF_ACK) before sending the next message.

Tip: This example includes full automatic configuration of a Dashboard. However, the ADP could be used to configure a set of streams to be connected manually to various modules in the Data Visualizer like **Graph**, **Oscilloscope**, or **Terminal**. The **ADP Control Panel** shows the available sinks and sources for the current ADP instance. These sinks and sources can be used in the same way as the sources and sinks in the **DGI Control Panel** and the **Serial Port Control Panel**. For more information, see Atmel Data Protocol.

Serial Interface

The ADP example uses an SPI interface to stream the ADP data from the SAM D21 to the embedded debugger (EDBG) on the SAM D21 Xplained Pro board. The EDBG uses the Data Gateway Interface (DGI) to stream the data over USB to the host computer. If the target board did not contain a device with DGI capability, ADP could have been streamed directly to the computer over a serial interface. If this was the case the **Serial Port Control Panel** (Serial Port) would have been used in the Data Visualizer instead of the **DGI Control Panel** (Data Gateway Interface (DGI)).

Initialization

After setting up the hardware (e.g., initializing the serial interface, setting up the ADC and I/O ports), the application is ready to start sending the ADP messages. The first message sent is a MSG_REQ_HANDSHAKE.

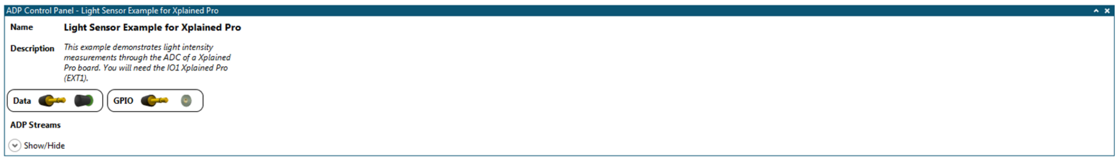
Table 5-5. MSG_REQ_HANDSHAKE

Field	Values	Description
Token	0xFF	
Message ID	0x00	
Data length	10	
Protocol version	0x01	The ADP example uses ADP protocol version 1
Options	0x01	GPIO is in use in this application
Token	{0x58, 0x99, 0xAB, 0xC9, 0x0F, 0xE2, 0xF7, 0xAA}	Token used to verify ADP protocol

This message is repeated until a MSG_RES_HANDSHAKE (MSG_RES_HANDSHAKE) is received, indicating the host is ready to receive messages.

ADP Control Panel
The ADP example configures the ADP Control Panel to look something like the screenshot below.

Notice: The appearance of screenshots will vary with operating system version and configuration.


ADP Control Panel - Light Sensor Example for Xplained Pro

Name: Light Sensor Example for Xplained Pro

Description: *This example demonstrates light intensity measurements through the ADC of a Xplained Pro board. You will need the IO1 Xplained Pro (EXT1).*

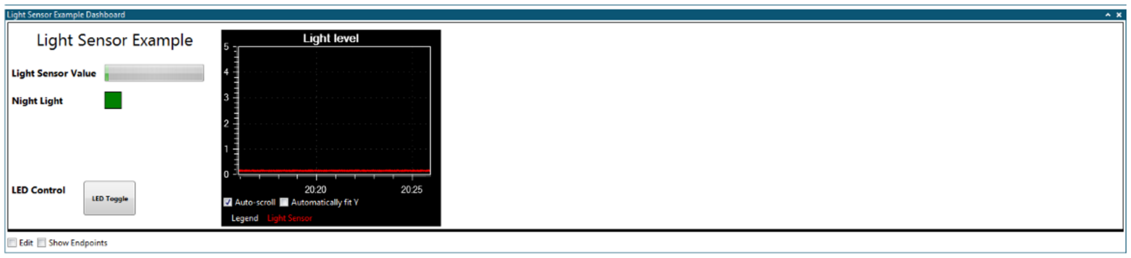
The **ADP Control Panel** is configured by the message detailed in the table below.

Table 5-6. MSG_CONF_INFO

Field	Values	Description
Token	0xFF	
Message ID	0x28	
Data length	177	
Title	“Light Sensor Example for Xplained Pro\0”	
Description	“This example demonstrates light intensity measurements through the ADC of a Xplained Pro board. You will need the I/O1 Xplained Pro (EXT1)\0”	Short description of the application

Light Sensor Dashboard
The ADP example sets up a dashboard for the I/O1 Xplained Pro light sensor and LED that looks something like the screenshot below.

Notice: The appearance of screenshots will vary with operating system version and configuration.



The light sensor dashboard is configured by the messages detailed in the tables below.

First, the dashboard itself must be set up.

Table 5-7. MSG_CONF_DASHBOARD

Field	Values	Description
Token	0xFF	
Message ID	0x2A	
Data length	38	
ID	0x0000	Dashboard ID
Label	"Light Sensor Example Dashboard\0"	Dashboard label
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	Background color of dashboard
Height	300	Height (in pixels) of dashboard

Next, **Label** elements are added to the dashboard.

Table 5-8. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	47	Depending on element type
Dashboard ID	0x0000	ID of light sensor dashboard
Element ID	0x0000	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	5	X-coordinate of element location. 0 is topmost position on dashboard.

Field	Values	Description
Y-coordinate	5	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	300	Width of element (pixels)
Height	35	Height of element (pixels)
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	24	
Attribute	0x00	Bold = OFF, Italic = OFF
Horizontal alignment	1	Center
Vertical alignment	1	Center
Background transparency	0	
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Foreground color	0x000000	RGB color of background
Label text	“Light Sensor Example\0”	

Table 5-9. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	45	Depending on element type
Dashboard ID	0x0000	ID of light sensor dashboard
Element ID	0x0001	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	5	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	60	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	129	Width of element (pixels)

Field	Values	Description
Height	25	Height of element (pixels)
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	14	
Attribute	0x01	Bold = ON, Italic = OFF
Horizontal alignment	0	Left
Vertical alignment	1	Center
Background transparency	0	
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Foreground color	0x000000	RGB color of background
Label text	“Light Sensor Value\0”	

Table 5-10. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	38	Depending on element type
Dashboard ID	0x0000	ID of light sensor dashboard
Element ID	0x0002	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	5	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	100	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	82	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	14	

Field	Values	Description
Attribute	0x01	Bold = ON, Italic = OFF
Horizontal alignment	0	Left
Vertical alignment	1	Center
Background transparency	0	
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Foreground color	0x000000	RGB color of background
Label text	"Night Light\0"	

Table 5-11. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	38	Depending on element type
Dashboard ID	0x0000	ID of light sensor dashboard
Element ID	0x0003	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	5	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	230	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	80	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	14	
Attribute	0x01	Bold = ON, Italic = OFF
Horizontal alignment	0	Left
Vertical alignment	1	Center

Field	Values	Description
Background transparency	0	
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Foreground color	0x000000	RGB color of background
Label text	“LED Control\0”	

A stream needs to be set up to receive light sensor data.

Table 5-12. MSG_CONF_STREAM

Field	Values	Description
Token	0xFF	
Message ID	0x20	
Data length	18	
ID	0x0001	ID of the light sensor data stream
Type	12	Stream type float
Mode	2	Out from target
State	0	Stream state ON
Label	“Light sensor\0”	Label of the data stream

And a **Progress bar** to show the light sensor data is added.

Table 5-13. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_PROGRESS

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	29	Depending on element type
Dashboard ID	0x0000	ID of the light sensor dashboard
Element ID	0x0004	Unique ID of the progress bar element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	140	X-coordinate of element location. 0 is topmost position on dashboard.

Field	Values	Description
Y-coordinate	60	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	145	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x03	ELEMENT_TYPE_PROGRESS
Minimum value	0	
Maximum value	4	
Initial value	0	
Color	0x008000	RGB color of progress bar

Eventually, the light sensor data stream is connected to the **Progress bar** element.

Table 5-14. MSG_CONF_ADD_STREAM_TO_ELEMENT

Field	Values	Description
Message ID	0x2C	
Data length	6	
Dashboard ID	0x0000	ID of the light sensor dashboard
Element ID	0x0004	ID of the progress bar element
Stream ID	0x0001	ID of the light sensor data stream

Next, a **Graph** element is added to the dashboard.

Table 5-15. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_GRAPH

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	53	Depending on element type
Dashboard ID	0x0000	ID of the light sensor dashboard
Element ID	0x0007	Unique ID of graph element
Title color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of title
Background color	0x000000	RGB color of graph frame
Graph background color	0x000000	RGB color of graph
Title text	“Light level\0”	
Plot count	1	
Xmin	0	

Field	Values	Description
Xmax	10	
Ymin	0	
Ymax	5	
Mode	0x00	Mouse interaction OFF Fit graph to right edge of canvas OFF

And the light sensor data stream is connected to the **Graph**.

Table 5-16. MSG_CONF_ADD_STREAM_TO_ELEMENT

Field	Values	Description
Message ID	0x2C	
Data length	6	
Dashboard ID	0x0000	ID of the light sensor dashboard
Element ID	0x0007	ID of the graph element
Stream ID	0x0001	ID of the light sensor data stream

A separate stream is set up to signal Night mode.

Table 5-17. MSG_CONF_STREAM

Field	Values	Description
Token	0xFF	
Message ID	0x20	
Data length	16	
ID	0x0029	ID of the Night mode stream
Type	2	Stream type uint_8
Mode	2	Out from target
State	0	Stream state ON
Label	“Night Mode\0”	Label of the data stream

A **Signal** element is added to the dashboard for the Night mode signal.

Table 5-18. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_SIGNAL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	22	Depending on element type
Dashboard ID	0x0000	ID of the light sensor dashboard

Field	Values	Description
Element ID	0x0005	Unique ID of the signal element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	140	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	100	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	25	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x04	ELEMENT_TYPE_SIGNAL
On transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
On color	0x008000	RGB color for ON state
Off transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Off color	0x000000	RGB color for OFF state

And the Night mode stream is connected to the **Signal** element.

Table 5-19. MSG_CONF_ADD_STREAM_TO_ELEMENT

Field	Values	Description
Message ID	0x2C	
Data length	6	
Dashboard ID	0x0000	ID of the light sensor dashboard
Element ID	0x0005	ID of the signal element
Stream ID	0x0029	ID of the night mode stream

Next, a incoming stream (in to target) is set up to transfer the **Button** status to the target.

Table 5-20. MSG_CONF_STREAM

Field	Values	Description
Token	0xFF	
Message ID	0x20	
Data length	16	
ID	0x0030	ID of the button stream

Field	Values	Description
Type	2	Stream type uint_8
Mode	0	In to target
State	0	Stream state ON
Label	“LED Toggle\0”	Label of the stream

A **Button** is added to the dashboard to toggle the target LED.

Table 5-21. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_BUTTON

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	26	Depending on element type
Dashboard ID	0x0000	ID of the light sensor dashboard
Element ID	0x0006	Unique ID of the signal element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	110	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	230	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	75	Width of element (pixels)
Height	50	Height of element (pixels)
Element type	0x01	ELEMENT_TYPE_BUTTON
Font size	10	
Button text	“LED Toggle\0”	
Toggle button	0x00 = Normal button 0x01 = Toggle button	Normal button

And the button stream is connected to the **Button** element.

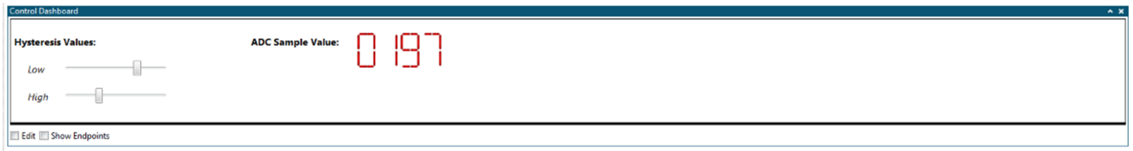
Table 5-22. MSG_CONF_ADD_STREAM_TO_ELEMENT

Field	Values	Description
Message ID	0x2C	
Data length	6	
Dashboard ID	0x0000	ID of the light sensor dashboard
Element ID	0x0006	ID of the button element
Stream ID	0x0030	ID of the button stream

Control Dashboard

The ADP example sets up a dashboard to control the ADC. The Control dashboard will look something like the screenshot below.

Notice: The appearance of screenshots will vary with operating system version and configuration.



The Control dashboard is configured by the messages detailed in the tables below.

First, the dashboard itself is set up:

Table 5-23. MSG_CONF_DASHBOARD

Field	Values	Description
Token	0xFF	
Message ID	0x2A	
Data length	25	
ID	0x0001	Dashboard ID
Label	"Control Dashboard\0"	Dashboard label
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	Background color of dashboard
Height	150	Height (in pixels) of dashboard

Next, a few labels are added to the dashboard.

Table 5-24. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	45	
Dashboard ID	0x0001	ID of control dashboard
Element ID	0x0008	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	5	X-coordinate of element location. 0 is topmost position on dashboard.

Field	Values	Description
Y-coordinate	20	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	128	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	14	
Attribute	0x01	Bold = ON, Italic = OFF
Horizontal alignment	0	Left
Vertical alignment	1	Center
Background transparency	0	
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Foreground color	0x000000	RGB color of background
Label text	“Hysteresis Values\0”	

Table 5-25. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	31	
Dashboard ID	0x0001	ID of control dashboard
Element ID	0x000A	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	25	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	100	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	30	Width of element (pixels)
Height	25	Height of element (pixels)

Field	Values	Description
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	14	
Attribute	0x02	Bold = OFF, Italic = ON
Horizontal alignment	0	Left
Vertical alignment	1	Center
Background transparency	0	
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Foreground color	0x000000	RGB color of background
Label text	“High\0”	

Table 5-26. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	30	
Dashboard ID	0x0001	ID of control dashboard
Element ID	0x0009	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	25	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	60	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	30	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	14	
Attribute	0x02	Bold = OFF, Italic = ON

Field	Values	Description
Horizontal alignment	0	Left
Vertical alignment	1	Center
Background transparency	0	
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255	
Foreground color	0x000000	RGB color of background
Label text	“Low\0”	

Table 5-27. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_LABEL

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	44	
Dashboard ID	0x0001	ID of control dashboard
Element ID	0x000B	Unique ID of label element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	350	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	20	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	130	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x00	ELEMENT_TYPE_LABEL
Font size	14	
Attribute	0x01	Bold = ON, Italic = OFF
Horizontal alignment	0	Left
Vertical alignment	1	Center
Background transparency	0	

Field	Values	Description
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of background
Foreground transparency	255 (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	
Foreground color	0x000000	RGB color of background
Label text	“ADC Sample Value\0”	

Next, a stream is set up to set the high value of the hysteresis.

Table 5-28. MSG_CONF_STREAM

Field	Values	Description
Token	0xFF	
Message ID	0x20	
Data length	16	
ID	0x0010	ID of the hysteresis high value stream
Type	4	Stream type uint_16
Mode	0	In to target
State	0	Stream state ON
Label	“Hyst. High\0”	Label of the data stream

A **Slider** is added to be able to adjust the high value of the hysteresis from the dashboard.

Table 5-29. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_SLIDER

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	26	
Dashboard ID	0x0001	ID of the control dashboard
Element ID	0x000D	Unique ID of the slider element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	75	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	100	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	156	Width of element (pixels)
Height	25	Height of element (pixels)

Field	Values	Description
Element type	0x02	ELEMENT_TYPE_SLIDER
Minimum value	2500	
Maximum value	4000	
Initial value	3000	

And the hysteresis high value stream is connected to the **Slider** element.

Table 5-30. MSG_CONF_ADD_STREAM_TO_ELEMENT

Field	Values	Description
Message ID	0x2C	
Data length	6	
Dashboard ID	0x0001	ID of the control dashboard
Element ID	0x000D	ID of the hysteresis high slider element
Stream ID	0x0010	ID of the hysteresis high stream

A stream for hysteresis low values is created and added to a **Slider** element in the same way as for the hysteresis high value above.

Table 5-31. MSG_CONF_STREAM

Field	Values	Description
Token	0xFF	
Message ID	0x20	
Data length	15	
ID	0x0011	ID of the hysteresis low value stream
Type	4	Stream type uint_16
Mode	0	In to target
State	0	Stream state ON
Label	“Hyst. Low\0”	Label of the data stream

Table 5-32. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_SLIDER

Field	Values	Description
Token	0xFF	
Message ID	0x2B	
Data length	26	
Dashboard ID	0x0001	ID of the control dashboard
Element ID	0x000C	Unique ID of the slider element
Z-Index	0	Order index, 0 places the element the farthest to the back

Field	Values	Description
X-coordinate	75	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	60	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	156	Width of element (pixels)
Height	25	Height of element (pixels)
Element type	0x02	ELEMENT_TYPE_SLIDER
Minimum value	1000	
Maximum value	2400	
Initial value	2000	

Table 5-33. MSG_CONF_ADD_STREAM_TO_ELEMENT

Field	Values	Description
Message ID	0x2C	
Data length	6	
Dashboard ID	0x0001	ID of the control dashboard
Element ID	0x000C	ID of the hysteresis low slider element
Stream ID	0x0011	ID of the hysteresis low stream

A separate stream for the light sensor ADC values to be fed to the **Segment display** is set up.

Table 5-34. MSG_CONF_STREAM

Field	Values	Description
Token	0xFF	
Message ID	0x20	
Data length	22	
ID	0x0002	ID of the adc value stream
Type	4	Stream type uint_16
Mode	2	Out from target
State	0	Stream state ON
Label	“Light Sensor ADC\0”	Label of the data stream

Next, a **Segment display** with four segments is added to the dashboard.

Table 5-35. MSG_CONF_DASHBOARD_ELEMENT, ELEMENT_TYPE_SEGMENT

Field	Values	Description
Token	0xFF	
Message ID	0x2B	

Field	Values	Description
Data length	20	
Dashboard ID	0x0001	ID of the control dashboard
Element ID	0x000D	Unique ID of the segment display element
Z-Index	0	Order index, 0 places the element the farthest to the back
X-coordinate	500	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	20	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	150	Width of element (pixels)
Height	50	Height of element (pixels)
Element type	0x05	ELEMENT_TYPE_SEGMENT
Count	4	Four segments
Base	10	Ordinary decimal base
Transparency	0xFF (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	0 - 255
Color	0xFF0000 (transmitted as 0xFFFF0000 as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB color of display

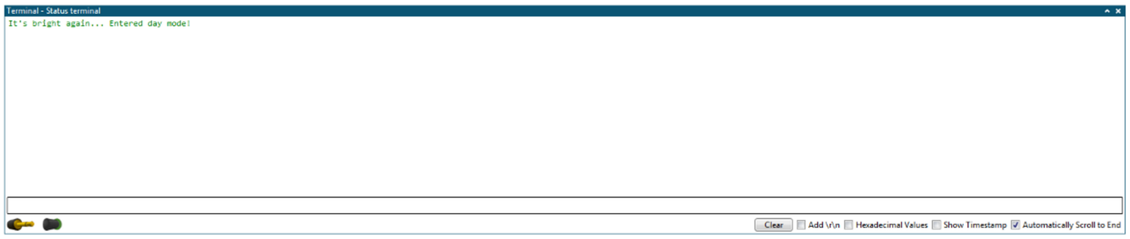
And the ADC sample value stream is connected to the **Segment display**.

Table 5-36. MSG_CONF_ADD_STREAM_TO_ELEMENT

Field	Values	Description
Message ID	0x2C	
Data length	6	
Dashboard ID	0x0001	ID of the control dashboard
Element ID	0x000D	ID of the segment display element
Stream ID	0x0002	ID of the ADC sample data stream

Terminal
The ADP example sets up a terminal module that looks something like the screenshot below.

Notice: The appearance of screenshots will vary with operating system version and configuration.



```
It's bright again... Entered day mode!
```

The terminal module is configured by the messages detailed in the tables below.

First, a stream is set up to send terminal data to the host computer.

Table 5-37. MSG_CONF_STREAM

Field	Values	Description
Token	0xFF	
Message ID	0x20	
Data length	21	
ID	0x0000	
Type	2	UINT_8 stream
Mode	2	Outgoing stream (out from target)
State	0	Stream state ON
Label	“Status messages\0”	Label of the data stream

Next, the terminal itself is configured.

Table 5-38. MSG_CONF_TERMINAL

Field	Values	Description
Token	0xFF	
Message ID	0x26	
Data length	26	
ID	0x0000	ID of terminal
Label	“Status terminal\0”	Terminal label
Width	80	Number of characters wide
Height	50	Number of lines high

Field	Values	Description
Background color	0xFFFFFF (transmitted as 0xFFFFFFFFFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	RGB background color
Foreground color	0x008000	RGB foreground color

Finally, the data stream is connected to the terminal module.

Table 5-39. MSG_CONF_ADD_TO_TERMINAL

Field	Values	Description
Token	0xFF	
Message ID	0x27	
Data length	27	
Terminal ID	0x0000	ID of terminal
Stream ID	0x0000	ID of stream
Mode	0xFF (transmitted as 0xFFFF as each 0xFF character must be transmitted as 0xFFFF, see Message Format)	Implicit newline in incoming text = ON
Text color	0x000000	RGB color of the text stream received
Tag text	“Status messages\0”	
Tag text color	0x000000	RGB color of the tag text

Data Transmission

When the terminal module and the two dashboards have been set up as described in the previous sections, the ADP example goes into a mode where it is continuously sending data to the host computer and receiving data from the host computer according to the configured streams. Below are examples of data messages being transmitted from the ATSAMD21 target to the host computer.

Table 5-40. Light Sensor ADC Stream

Field	Values	Description
Token	0xFF	
Message ID	0x40	
Data length	6	
Number of streams (N)	1	
Stream ID	0x0002	ID of ADC value stream
Num bytes (Xn)	2	Number of bytes from the stream
Stream X data sample M	634 (0x027A)	The data of the stream (uint_16)

Table 5-41. Night Mode Stream

Field	Values	Description
Token	0xFF	
Message ID	0x40	
Data length	5	
Number of streams (N)	1	
Stream ID	0x0029	ID of Night mode stream
Num bytes (Xn)	1	Number of bytes from the stream
Stream X data sample M	0x01 (Bright light, day mode)	The data of the stream (uint_8)

When the Night mode changes, the example also changes the background color of the terminal module by sending another MSG_CONF_TERMINAL.

Table 5-42. MSG_CONF_TERMINAL to update Terminal Background Color to White

Field	Values	Description
Token	0xFF	
Message ID	0x26	
Data length	26	
ID	0x0000	ID of terminal
Label	“Status terminal\0”	Terminal label
Width	80	Number of characters wide
Height	50	Number of lines high
Background color	0xFFFFFF	RGB background color
Foreground color	0x008000	RGB foreground color

Table 5-43. Status Message Stream

Field	Values	Description
Token	0xFF	
Message ID	0x40	
Data length	44	
Number of streams (N)	1	
Stream ID	0x0000	ID of status message stream
Num bytes (Xn)	40	Number of bytes from the stream
Stream X data samples	“It's bright again... Entered day mode!\r\n”	The data of the stream (uint_8)

Field	Values	Description
	{0x49, 0x74, 0x27, 0x73, 0x20, 0x62, 0x72, 0x69, 0x67, 0x68, 0x74, 0x20, 0x61, 0x67, 0x61, 0x69, 0x6E, 0x2E, 0x2E, 0x2E, 0x20, 0x45, 0x6E, 0x74, 0x65, 0x72, 0x65, 0x64, 0x20, 0x64, 0x61, 0x79, 0x20, 0x6D, 0x6F, 0x64, 0x65, 0x21, 0x0D, 0x0A}	

Examples of data messages for the various streams from the computer to the target can be found in the tables below.

Table 5-44. Hysteresis Low Value Stream Data Message

Field	Values	Description
Token	0xFF	
Message ID	0x14	MSG_RES_DATA
Data length	5	
Stream ID	0x0011	ID of hysteresis low value stream
Bytes sent	2	Number of bytes in the data payload
Data bytes	0x07C6	The data (uint_16)

Table 5-45. Hysteresis High Value Stream Data Message

Field	Values	Description
Token	0xFF	
Message ID	0x14	MSG_RES_DATA
Data length	5	
Stream ID	0x0002	ID of hysteresis high value stream
Bytes sent	2	Number of bytes in the data payload
Data bytes	0x0BB7	The data (uint_16)

Table 5-46. LED Toggle Stream Data Message

Field	Values	Description
Token	0xFF	
Message ID	0x14	MSG_RES_DATA
Data length	5	
Stream ID	0x0030	ID of LED toggle stream
Bytes sent	1	Number of bytes in the data payload
Data bytes	0x00	The data (uint_8)

5.2.3 Message Flow

The target is the master in the system, whereas the host computer is the slave. The target will initiate communication, and the computer will respond. However, the computer will transmit data to the target MCU as soon as data is generated on the computer.

Before any data can be exchanged between the target and the computer, the connection must be established using handshake messages.

5.2.4 Message Format

The ADP protocol uses a common message format for both directions of communication and all message types.

Table 5-47. ADP Message Format

Field	Size	Values	Description
Token	1 byte	0xFF	Start token for ADP data
Message ID	1 byte	0x00-0xFE	Identifies the type of message being sent
Data length	2 bytes	0x0000-0xFFFF	Length of data packet (bytes)
Data	N bytes	...	Data content of the message

Token
The value 0xFF followed by a value other than 0xFF (0xFF is not a valid Message ID), is used to indicate the start of the message. This means that 0xFF must be sent between each message.

If the value 0xFF is to be transmitted as part of data or data length, a new 0xFF should be inserted after it. When receiving messages, two 0xFF should be decoded as a single 0xFF. The extra 0xFF bytes are not contributing to the Data length field. For example, a color field with the value 0xFFFFFF will have to be transmitted as 0xFFFFFFFFFFFF, but only contributes to the data length by three bytes.

The value 0xFF is not allowed to be used as a message ID. When polling for data over SPI, the 0xFF token must be used as a dummy character to not trigger a command unintentionally.

Endianness
All message data is ordered using little endian.

5.2.5 Message Types

There are three main message types; Request messages, Configuration messages, and Data messages.

5.2.5.1 Request Messages

Request messages are used by the target to request information/status from the host PC. These messages are pre-fixed with MSG_REQ. The PC should always respond with the corresponding response message, pre-fixed MSG_RES.

Table 5-48. Request Messages and Responses

Message Type	ID	Description
MSG_REQ_HANDSHAKE	0x00	Request handshake
MSG_RES_HANDSHAKE	0x10	Respond to handshake
MSG_REQ_STATUS	0x02	Request status from PC

Message Type	ID	Description
MSG_RES_STATUS	0x12	Respond to status
MSG_RES_DATA	0x14	Raw data from PC to target

5.2.5.2 Configuration Messages

Used by target to send configuration settings to the PC. These messages are pre-fixed with MSG_CONF. The PC should respond to these messages with an acknowledge message (MSG_ACK).

Table 5-49. Configuration Messages, Target to PC

Message type	ID	Description
MSG_CONF_STREAM	0x20	Create a new stream
Reserved	0x21	Reserved for future use
MSG_CONF_GRAPH	0x22	Create new graph or reconfigure existing one
MSG_CONF_ADD_STREAM_TO_AXIS	0x23	Add stream to axis in graph
MSG_CONF_CURSOR_TO_GRAPH	0x24	Add parameter cursor to graph
Reserved	0x25	Reserved for future use
MSG_CONF_TERMINAL	0x26	Create new terminal or reconfigure existing one
MSG_CONF_ADD_TO_TERMINAL	0x27	Add stream to terminal
MSG_CONF_INFO	0x28	Info about the application
MSG_CONF_AXIS	0x29	Create new axis or reconfigure existing one
MSG_CONF_DASHBOARD	0x2A	Add dashboard container
MSG_CONF_DASHBOARD_ELEMENT	0x2B	Add element to dashboard container
MSG_CONF_ADD_STREAM_TO_ELEMENT	0x2C	Tie an already configured stream to an already configured dashboard element.

Table 5-50. Configuration Messages, PC to Target

Message type	ID	Description
MSG_CONF_ACK	0x30	Status of last received configuration message

5.2.5.3 Data Messages

Used by a target to send data to the PC. Prefixed with MSG_DATA. These messages should not be responded too.

Table 5-51. Data Messages

Message Type	ID	Description
MSG_DATA_STREAM	0x40	Send data from one or more streams

5.2.5.4 Request Message Details

MSG_REQ_HANDSHAKE
Before any data can be exchanged between the target and the PC, the connection must be established using handshake messages.

Important: To make sure the handshake message is detected by the Data Visualizer during Auto-Detection, the handshake message should be sent at least twice per second.

Field	Size	Values	Description
Message ID	1 byte	0x00	
Data length	2 bytes	10	
Protocol version	1 byte	0x01-0xFF	Version of protocol on target
Options	1 byte	0xXX	Reserved for future use
Token	8 bytes	{0x58, 0x99, 0xAB, 0xC9, 0x0F, 0xE2, 0xF7, 0xAA}	Token used to verify ADP protocol

MSG_RES_HANDSHAKE
The PC should respond to a handshake request from the target with this packet. The PC should always communicate with the target using the protocol version stated in the targets handshake request message. If the PC for some reason is unable to do this, the handshake request must be rejected.

Field	Size	Values	Description
Message ID	1 byte	0x10	
Data length	2 bytes	1	
Handshake status	1 byte	0x00: Handshake accepted 0x01: Handshake rejected. Invalid protocol version. 0x02: Handshake rejected. Other reason.	

MSG_REQ_STATUS
Message used by target to request status from PC. It is good practice to ask for status each time a new configuration message is sent. While sending raw data, it is good practice to ask for status... occasionally.

Field	Size	Values	Description
Message ID	1 byte	0x02	
Data length	2 bytes	0	

This message has no data fields.

MSG_RES_STATUS
Status message from PC to target. Once a status is requested from the target and this packet is sent, all status flags should be cleared on the PC.

Field	Size	Values	Description
Message ID	1 byte	0x12	
Data length	2 bytes	2	
Status code	2 bytes	0x0000	Reserved for future use

MSG_RES_DATA
Data packet from PC to target MCU. This message can come asynchronously from the PC (i.e., without the target MCU having requested it).

Field	Size	Values	Description
Message ID	1 byte	0x14	
Data length	2 bytes	3 + Data bytes	
Stream ID	2 bytes		ID of stream
Bytes sent	1 byte		Number of bytes in the data payload. If the target has requested data from an unknown stream, or if the stream has no data to send, this field should be set to 0 and the appropriate status flag should be set.
Data bytes	N bytes		The data

5.2.5.5 Configuration Message Details

MSG_CONF_STREAM
Used to create a new stream. The type of the stream can be either EVENT, TEXT, or DATA. Each stream must be given a unique ID.

Field	Size	Values	Description
Message ID	1 byte	0x20	
Data length	2 bytes	5 + label length (N) + parameter length (M)	

Field	Size	Values	Description
ID	2 bytes	0x0000-0xFFFF	ID of stream
Type	1 byte	STREAM_DATA_TYPE	Stream type
Mode	1 byte	0 = Incoming (normal) 1 = Incoming (single value) 2 = Outgoing	Stream mode/direction Direction is defined seen from target
State	1 byte	0 = ON 1= OFF	Stream state
Label	N bytes	Null-terminated string	Label of the data stream
Parameters	M bytes	Byte array	Parameters specific to stream type

Table 5-52. STREAM_DATA_TYPE

Type code	Data type	Parameters	Data size
0	EVENT	None	0 bytes
1	STRING	None	N bytes
2	UINT_8	None	1 byte
3	INT_8	None	1 byte
4	UINT_16	None	2 bytes
5	INT_16	None	2 bytes
6	UINT_32	None	4 bytes
7	INT_32	None	4 bytes
8	XY_8	None	2 bytes
9	XY_16	None	4 bytes
10	XY_32	None	8 bytes
11	BOOL	None	1 byte
12	Float	None	4 bytes
13	Double	None	8 bytes
20	Grid	Base data type (1 byte, e.g. 6 for UINT_32) Width of grid (1 byte) Depth of grid (1 byte)	Size of base data type * Width * Depth

The XY data types are combos of X and Y coordinates. If the format is XY_8 the data will contain one byte of X-coordinate and one byte of Y-coordinate. For XY_16 the data will contain two bytes of X-coordinate and two bytes of Y-coordinates and for XY_32 each coordinate will be four bytes long.

MSG_CONF_GRAPH
Used to create a new or reconfigure an existing graph view. The graph view requires an unique ID. Values for range, labels, and background color can also be set.

Field	Size	Values	Description
Message ID	1 byte	0x22	
Data length	2 bytes	23 + label length (N) + Xlabel length (M)	
ID	2 bytes	0x0000-0xFFFF	ID of new graph
Label	N bytes	Null-terminated string	Graph label
Xmin	4 bytes		Range Xmin value
Xmax	4 bytes		Range Xmax value
Xlabel	M bytes	Null-terminated string	X label
Xscale numerator	4 bytes		X range scale value. Set to 0 to enable auto-range.
Xscale denumerator	4 bytes		X range scale value. Set to 0 to enable auto-range.
Scale mode	1 byte	0 = scaling off 1 = auto-scale	Vertical scaling
Background color	3 bytes	0xRRGGBB	RGB background color
Scroll mode	1 byte	0 = no scrolling 1 = stepping 2 = scroll 3 = circular/sweep	Horizontal scrolling

MSG_CONF_ADD_STREAM_TO_AXIS
Used to add a stream to the specified graph view.

Field	Size	Values	Description
Message ID	1 byte	0x23	
Data length	2 bytes	32	
Graph ID	2 bytes	0x0000-0xFFFF	ID of graph
Axis ID	2 bytes	0x0000-0xFFFF	ID of axis
Stream ID	2 bytes	0x0000-0xFFFF	ID of stream

Field	Size	Values	Description
Sample rate numerator	4 bytes		Sample rate of stream, set to 0 if not applicable
Sample rate denominator	4 bytes		Sample rate of stream, set to 0 in not applicable
Yscale numerator	4 bytes		Y range scale value. Set to 0 to enable auto-range.
Yscale denumerator	4 bytes		Y range scale value. Set to 0 to enable auto-range.
Yoffset	4 bytes		Offset of values
Transparency	1 byte	0 - 255	Adjust the transparency
Mode	1 byte	For graphs: bit 0 = line ON/OFF bit 1 = points ON/OFF For text: 0 = flag 1 = text	
Thickness	1 byte	0 - 255	Thickness of line
Color	3 bytes	0xRRGGBB	RGB color of line

MSG_CONF_CURSOR_TO_GRAPH
Used to add a parameter cursor to a graph.

Field	Size	Values	Description
Message ID	1 byte	0x24	
Data length	2 bytes	35 + label length (N)	
Stream ID	2 bytes	0x0000-0xFFFF	ID of stream
Graph ID	2 bytes	0x0000-0xFFFF	ID of graph
Axis ID	2 bytes	0x0000-0xFFFF	ID of axis
Label	N bytes	Null-terminated string	Label of cursor
Thickness	1 byte	0 - 255	Thickness of line
Color	3 bytes	0xRRGGBB	RGB color of cursor
Initial value	4 bytes		Starting point of cursor
Minimum	4 bytes		Minimum allowed value
Maximum	4 byte		Maximum allowed value

Field	Size	Values	Description
Scale numerator	4 bytes		Numerator of scaling value
Scale denominator	4 bytes		Denominator of scaling value
Scale offset	4 bytes		Offset of scaling value
Style	1 byte	0xXX	Reserved for future use

MSG_CONF_TERMINAL
Used to create a new or reconfigure an existing terminal. The terminal requires a unique ID. Values for label and background color must also be set.

Field	Size	Values	Description
Message ID	1 byte	0x26	
Data length	2 bytes	10 + label length (N)	
ID	2 bytes	0x0000-0xFFFF	ID of terminal
Label	N bytes	Null-terminated string	Terminal label
Width	1 byte	0 - 255	Number of characters wide
Height	1 byte	0 - 255	Number of lines high
Background color	3 bytes	0xRRGGBB	RGB background color
Foreground color	3 byte	0xRRGGBB	RGB foreground color

MSG_CONF_ADD_TO_TERMINAL
Used to add a stream to the specified terminal.

Field	Size	Values	Description
Message ID	1 byte	0x27	
Data length	2 bytes	11 + tag length (N)	
Terminal ID	2 bytes	0x0000-0xFFFF	ID of terminal
Stream ID	2 bytes	0x0000-0xFFFF	ID of stream
Mode	1 byte	0bXXXNXXXX	N = implicit newline in incoming text
Text color	3 bytes	0xRRGGBB	RGB color of the text stream received
Tag text	N bytes	Null-terminated string	Descriptive tag
Tag text color	3 bytes	0xRRGGBB	RGB color of the tag text

MSG_CONF_INFO
Used to send info about the application. For example, a text string describing the example application.

Field	Size	Values	Description
Message ID	1 byte	0x28	
Data length	2 bytes	Title length (N) + Description length (M)	
Title	N bytes	Null-terminated string	Application title
Description	M bytes	Null-terminated string	Short description of the application

MSG_CONF_AXIS
Used to create a new or reconfigure an existing axis of a graph view. The axis requires a unique ID. Values for range, label, and axis color must also be set.

Field	Size	Values	Description
Message ID	1 byte	0x29	
Data length	2 bytes	24 + label length (N)	
Axis ID	2 bytes	0x0000-0xFFFF	ID of new axis
Graph ID	2 bytes	0x0000-0xFFFF	ID of graph containing the axis
Label	N bytes	Null-terminated string	Axis label
Ymin	4 bytes		Range Ymin value
Ymax	4 bytes		Range Ymax value
Yscale numerator	4 bytes		Y range scale value. Set to 0 to enable auto-range.
Yscale denominator	4 bytes		Y range scale value. Set to 0 to enable auto-range.
Scale mode	1 byte	0 = scaling off 1 = auto-scale	Vertical scaling
Axis color	3 bytes	0xRRGGBB	RGB color

MSG_CONF_DASHBOARD
Add a dashboard container where dashboard elements can be placed.

Field	Size	Values	Description
Message ID	1 byte	0x2A	
Data length	2 bytes	7 + label length (N)	
ID	2 bytes	0x0000-0xFFFF	Dashboard ID
Label	N bytes	Null-terminated string	Dashboard label

Field	Size	Values	Description
Background color	3 bytes	0xRRGGBB	Background color of dashboard
Height	2 bytes		Height (in pixels) of dashboard

MSG_CONF_DASHBOARD_ELEMENT
Configure a dashboard element and add it to the specified dashboard. The table shows common fields for all dashboard element types. Additional fields must be added, depending on element type. See below.

Field	Size	Values	Description
Message ID	1 byte	0x2B	
Data length	2 bytes	14 + element specific length	Depending on element type
Dashboard ID	2 bytes	0x0000-0xFFFF	ID of dashboard to add the element to
Element ID	2 bytes	0x0000-0xFFFF	Unique ID of element
Z-Index	1 byte		Order index, 0 places the element the farthest to the back
X-coordinate	2 bytes	Coordinate value in pixels	X-coordinate of element location. 0 is topmost position on dashboard.
Y-coordinate	2 bytes	Coordinate value in pixels	Y-coordinate of element location. 0 is topmost position on dashboard.
Width	2 bytes	Width in pixels	Width of element
Height	2 bytes	Height in pixels	Height of element
Element type	1 byte	ELEMENT_TYPE	See each element type below

ELEMENT_TYPE_LABEL
The tables below describe the ELEMENT_TYPE_LABEL specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message.

Table 5-53. ELEMENT_TYPE_LABEL Parameters

Parameter	Value
ELEMENT_TYPE	0x00
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_LABEL	26 + length of default text (N)

Table 5-54. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT

Field	Size	Values	Description
Font size	1 byte	1-255	
Attribute	1 byte	Bit 0: Bold ON/OFF Bit 1: Italic ON/OFF	
Horizontal alignment	1 byte	0 = Left 1 = Center 2 = Right	
Vertical alignment	1 byte	0 = Top 1 = Center 2 = Bottom	
Background transparency	1 byte	0 - 255	
Background color	3 bytes	0xRRGGBB	RGB color of background
Foreground transparency	1 byte	0 - 255	
Foreground color	3 bytes	0xRRGGBB	RGB color of background
Label text	N bytes	Null-terminated string max. 100 bytes	

Example
The picture below shows an example of two labels with the corresponding parameters given in the tables below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

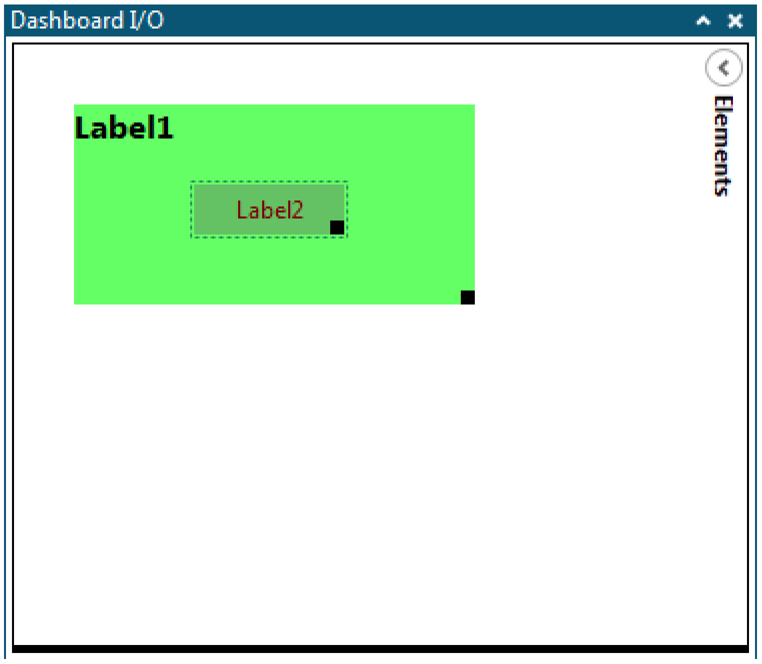


Table 5-55. Label1

Field	Value
Z-index	0
X-coordinate	30
Y-coordinate	30
Width	200
Height	100
Element type	0x00
Font size	16
Attribute	0x01
Horizontal alignment	0
Vertical alignment	0
Background transparency	255
Background color	0x64FF64
Foreground transparency	255
Foreground color	0x000000
Label text	“Label1\0”

Table 5-56. Label2

Field	Value
Z-index	1
X-coordinate	90
Y-coordinate	70
Width	75
Height	25
Element type	0x00
Font size	12
Attribute	0x00
Horizontal alignment	1
Vertical alignment	1
Background transparency	100
Background color	0x646464
Foreground transparency	255
Foreground color	0x7C0000
Label text	"Label2\0"

ELEMENT_TYPE_BUTTON

The tables below describe the ELEMENT_TYPE_BUTTON specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message.

Table 5-57. ELEMENT_TYPE_BUTTON Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x01
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_BUTTON	16 + length of button text (N)

Table 5-58. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_BUTTON

Field	Size	Values	Description
Font size	1 byte	0-255	
Button text	N bytes	Null-terminated string, max. 20 bytes	For toggle button text is selected by '/' delimited text (<off text>/<on text>)
Toggle button	1 byte	0x00 = Normal button 0x01 = Toggle button	Change mode to toggle button. Button text is selected by '/' delimited text field.

Example

The picture below shows an example of two buttons; one normal button and one toggle button. The element data fields for the example are shown in the tables below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

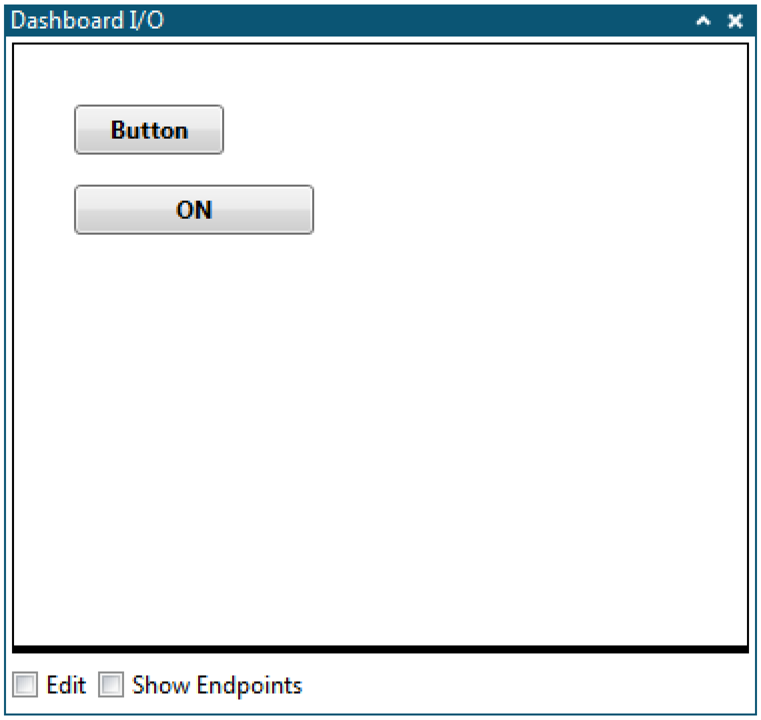


Table 5-59. Button

Field	Value
Z-index	0
X-coordinate	30
Y-coordinate	30
Width	75
Height	25
Element type	0x01
Font size	12
Button text	“Button\0”
Toggle button	0x00

Table 5-60. Toggle Button

Field	Value
Z-index	0
X-coordinate	30
Y-coordinate	70
Width	120
Height	25
Element type	0x01
Font size	12
Button text	“ON/OFF\0”
Toggle button	0x01

ELEMENT_TYPE_SLIDER

The tables below describe the ELEMENT_TYPE_SLIDER specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message.

Table 5-61. ELEMENT_TYPE_SLIDER Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x02
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_SLIDER	26

Table 5-62. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_SLIDER

Field	Size	Values	Description
Minimum value	4 bytes		
Maximum value	4 bytes		
Initial value	4 byte		

Example

The picture below shows an example of a slider element with a range from 0 to 50 and an initial value of 20. The corresponding element data fields are given in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

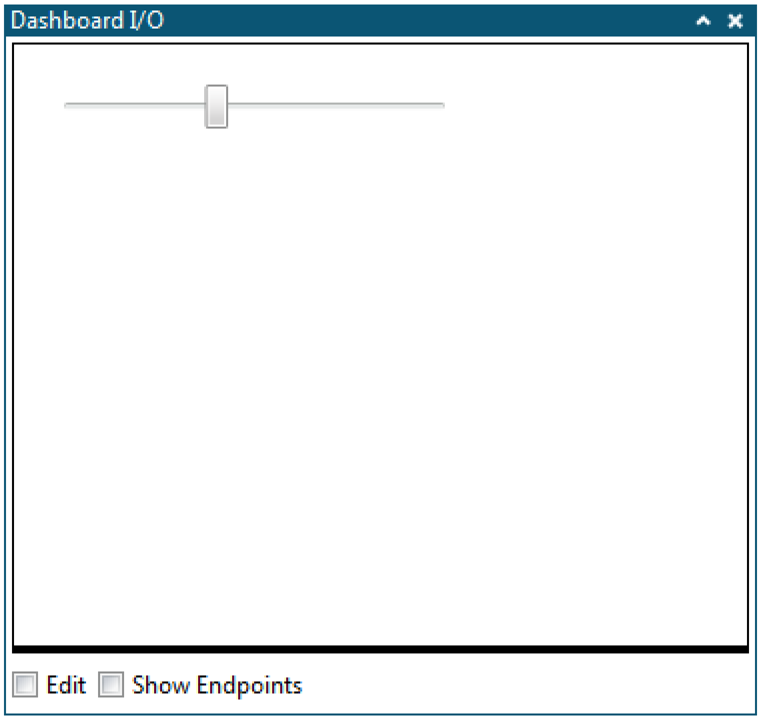


Table 5-63. Slider

Field	Value
Z-index	0
X-coordinate	20
Y-coordinate	20
Width	200
Height	30
Element type	0x02
Minimum value	0
Maximum value	50
Initial value	20

ELEMENT_TYPE_PROGRESS

The tables below describe the ELEMENT_TYPE_PROGRESS specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message.

Table 5-64. ELEMENT_TYPE_PROGRESS Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x03
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_PROGRESS	29

Table 5-65. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_PROGRESS

Field	Size	Values	Description
Minimum value	4 bytes		
Maximum value	4 bytes		
Initial value	4 byte		
Color	3 bytes	0xRRGGBB	RGB color of progress bar

Example

The picture below shows an example of a progress bar element. The corresponding element data fields are given in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

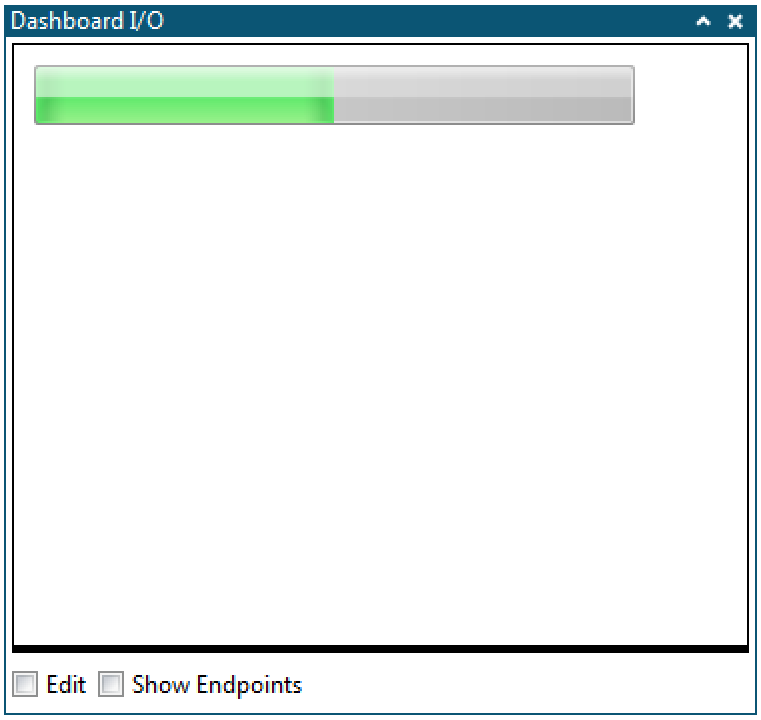


Table 5-66. Progress Bar

Field	Value
Z-index	0
X-coordinate	10
Y-coordinate	10
Width	100

Field	Value
Height	30
Element type	0x03
Minimum value	0
Maximum value	100
Initial value	50
Color	0x01D328

ELEMENT_TYPE_SIGNAL

The tables below describe the ELEMENT_TYPE_SIGNAL specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message.

Table 5-67. ELEMENT_TYPE_SIGNAL Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x04
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_SIGNAL	22

Table 5-68. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_SIGNAL

Field	Size	Values	Description
On transparency	1 byte	0 - 255	
On color	3 bytes	0xRRGGBB	RGB color for on state
Off transparency	1 byte		
Off color	3 bytes	0xRRGGBB	RGB color for off state

Example

The picture below shows an example of a signal element. The corresponding element data fields are given in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

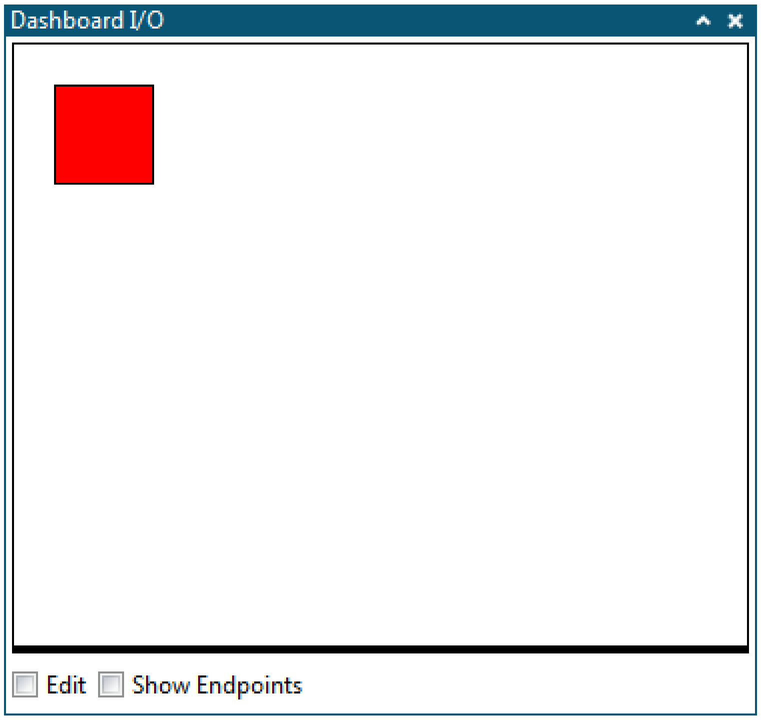


Table 5-69. Signal

Field	Value
Z-index	0
X-coordinate	20
Y-coordinate	20
Width	50
Height	50
Element type	0x04
On transparency	255
On color	0x00FF00
Off transparency	255
Off Color	0xFF0000

ELEMENT_TYPE_SEGMENT

The tables below describe the ELEMENT_TYPE_SEGMENT specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message.

Table 5-70. ELEMENT_TYPE_SEGMENT Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x05
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_SEGMENT	20

Table 5-71. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_SEGMENT

Field	Size	Values	Description
Count	1 byte	1-20	Number of displays
Base	1 byte	2-16	Numeric base
Transparency	1 byte	0 - 255	
Color	3 bytes	0xRRGGBB	RGB color of display

Example

The picture below shows an example of a segment element with two digits. The corresponding element data fields are given in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

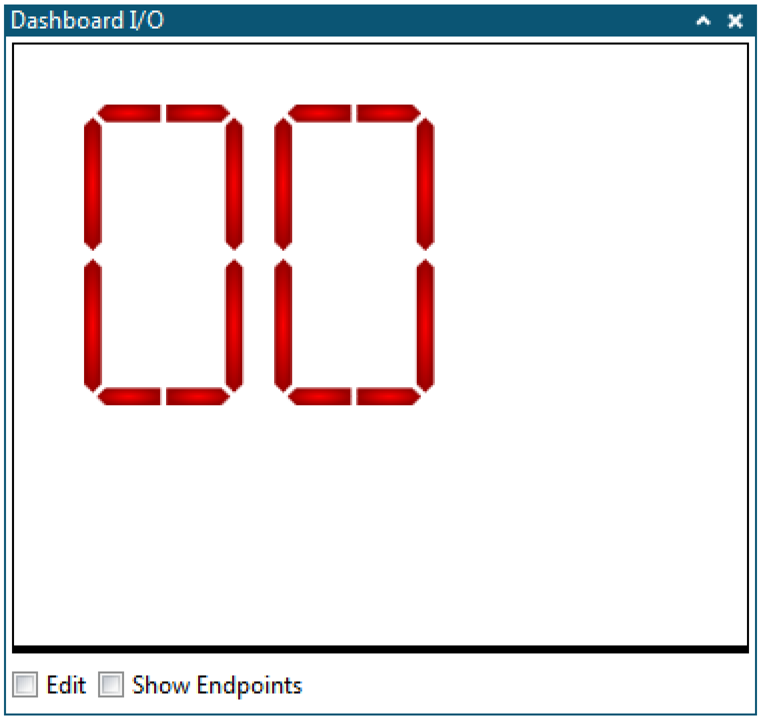


Table 5-72. Signal

Field	Value
Z-index	0
X-coordinate	30
Y-coordinate	30
Width	174

Field	Value
Height	150
Element type	0x05
Count	2
Base	10
Transparency	255
Color	0xFD0000

ELEMENT_TYPE_GRAPH

The tables below describe the ELEMENT_TYPE_GRAPH specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message. This element generates a graph that visualizes the data from the target. There will be one input stream for each plot.

Table 5-73. ELEMENT_TYPE_GRAPH Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x06
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_GRAPH	41 + length of title (N)

Table 5-74. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_GRAPH

Field	Size	Values	Description
Title color	3 bytes	0xRRGGBB	RGB color of title
Background color	3 bytes	0xRRGGBB	RGB color of graph frame
Graph background color	3 bytes	0xRRGGBB	RGB color of graph
Title text	N bytes	Null-terminated string max. 20 bytes	
Plot count	1 byte	1-10	Number of plots
Xmin	4 bytes	Floating point, seconds	Will be converted to <minutes>:<seconds> format
Xmax	4 bytes	Floating point, seconds	Will be converted to <minutes>:<seconds> format
Ymin	4 bytes	Floating point	
Ymax	4 bytes	Floating point	
Mode	1 byte	Bit 0: Mouse interaction ON/OFF Bit 1: Fit graph to right edge of canvas	

Example

The picture below shows an example of a graph element with tree input plot inputs. The element data fields for the example is shown in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

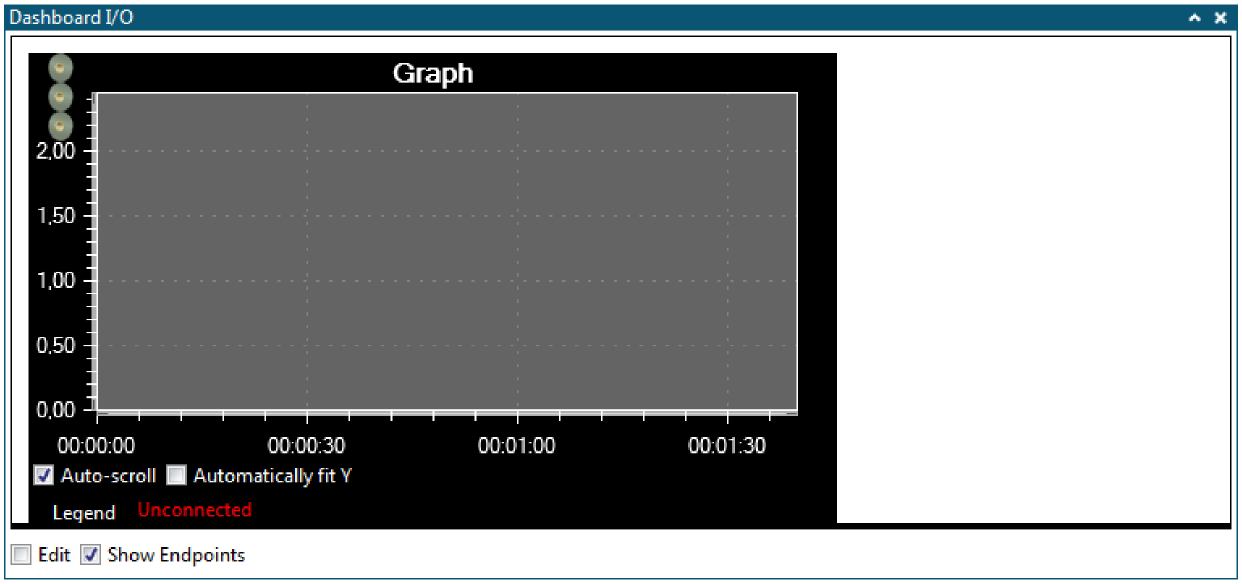


Table 5-75. Graph

Field	Value
Z-index	0
X-coordinate	10
Y-coordinate	10
Width	500
Height	250
Element type	0x06
Title color	0xFFFFFF
Background color	0x000000
Graph background color	0x646464
Title text	“Graph\0”
Plot count	3
Xmin	0
Xmax	100
Ymin	0

Field	Value
Ymax	2.45
Mode	0x01

ELEMENT_TYPE_NUMERICAL_INPUT

The tables below describe the ELEMENT_TYPE_NUMERICAL_INPUT specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message. This element type is used to output numerical values to the target from the PC. The element is editable in the dashboard.

Table 5-76. ELEMENT_TYPE_NUMERICAL_INPUT Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x07
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_NUMERICAL_INPUT	26

Table 5-77. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_NUMERICAL_INPUT

Field	Size	Values	Description
Minimum	4 bytes	Signed 32-bit integer	
Maximum	4 bytes	Signed 32-bit integer	
Value	4 bytes	Signed 32-bit integer	Initial value

Example

The picture below shows an example of a numerical input element with a range of valid values from -100 to 100 and a default value of 30. The element data fields for the example is shown in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

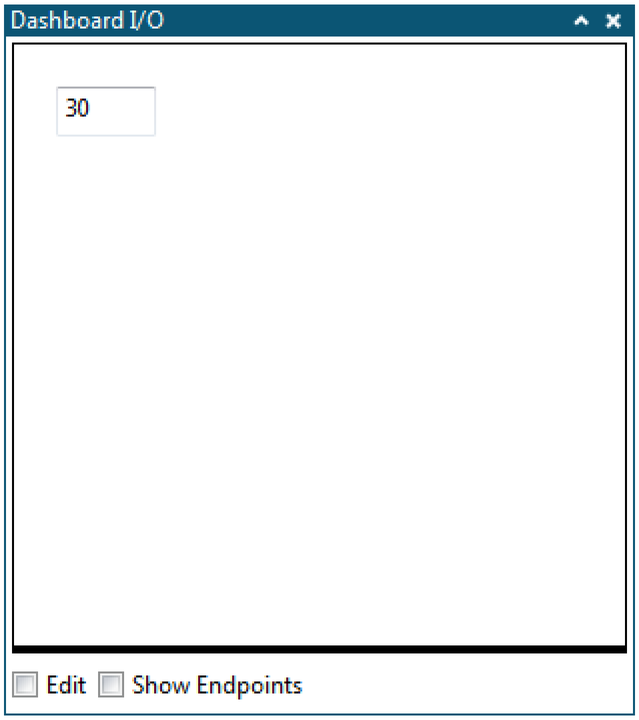


Table 5-78. Numerical Input

Field	Value
Z-index	0
X-coordinate	20
Y-coordinate	20
Width	50
Height	25
Element type	0x07
Minimum	-100
Maximum	100
Value	30

ELEMENT_TYPE_RADIO

The tables below describe the ELEMENT_TYPE_RADIO specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message. This element generates a group of radio buttons where only one option can be selected. Initially, the first option is selected.

Table 5-79. ELEMENT_TYPE_RADIO Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x08
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_RADIO	17 + length of text fields (N)

Table 5-80. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_RADIO

Field	Size	Values	Description
Font size	1 byte	0-100	
Number of items	1 byte	1-10	
Orientation	1 byte	0 = Horizontal 1 = Vertical	
Text fields	N bytes	Null-terminated string max. 20 bytes	'/' separated option list

Example
The picture below shows an example of radio button group with three buttons. The corresponding element data fields are given in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

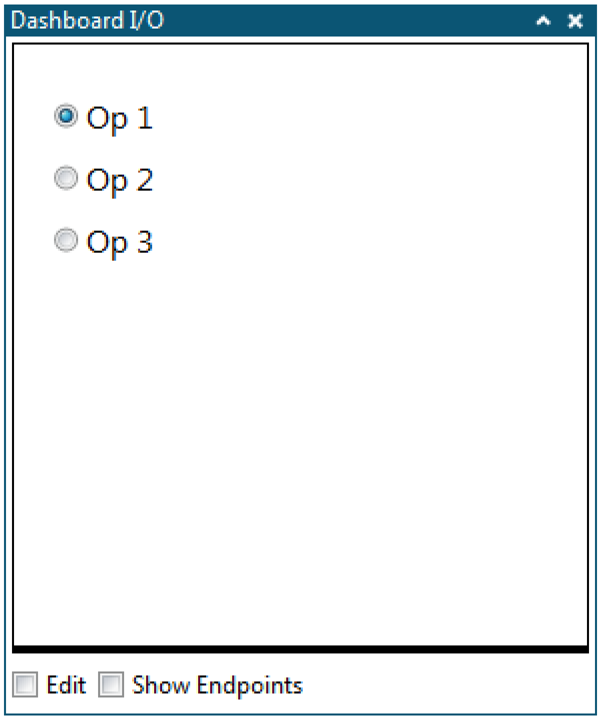


Table 5-81. Radio Button Group

Field	Value
Z-index	0
X-coordinate	20
Y-coordinate	20

Field	Value
Width	100
Height	100
Element type	0x08
Font size	16
Number of radio buttons	3
Orientation	1
Text fields	“Op 1/Op 2/Op 3\0”

ELEMENT_TYPE_PIE

The tables below describe the ELEMENT_TYPE_PIE specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message. This element generates a pie chart. The size of the slices are updated based on the data from the target to the PC. There will be one data stream input for each slice.

Table 5-82. ELEMENT_TYPE_PIE Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x09
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_PIE	21 + length of title (N)

Table 5-83. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_PIE

Field	Size	Values	Description
Background color	3 bytes	0xRRGGBB	RGB color of background
Title color	3 bytes	0xRRGGBB	RGB color of title
Title	N bytes	Null-terminated string max. 20 bytes	
Number of slices	1 byte	1-10	

Example

The picture below shows an example of a pie element with three slices. The corresponding element data fields are given in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

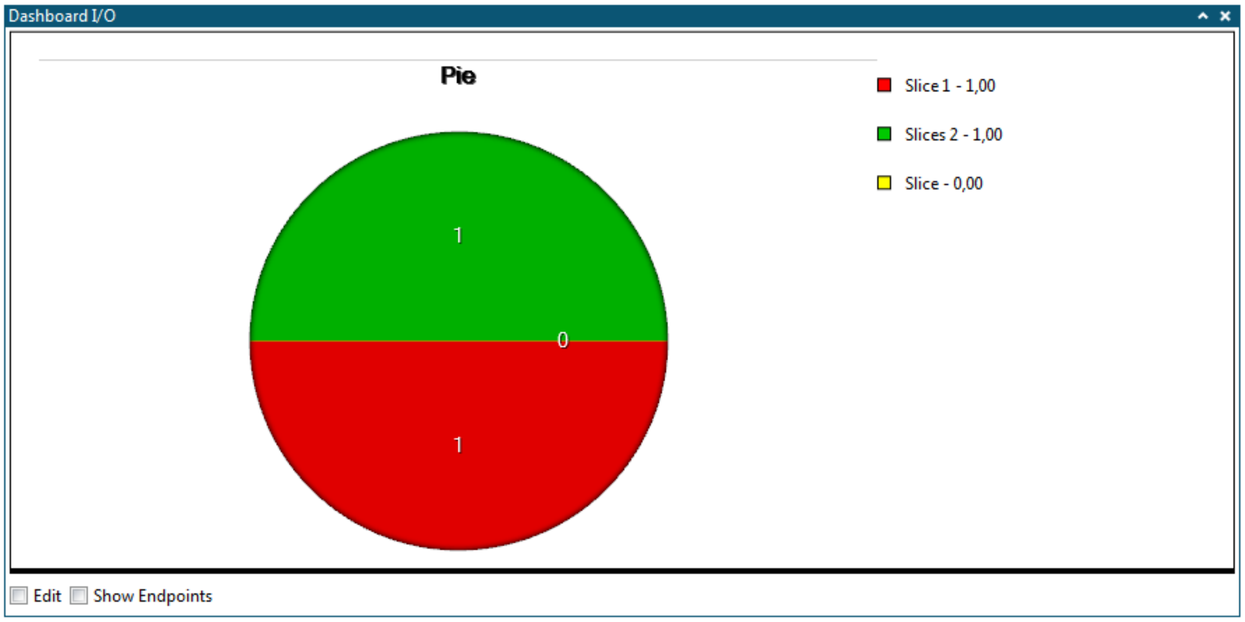


Table 5-84. Numerical Input

Field	Value
Z-index	0
X-coordinate	20
Y-coordinate	20
Width	600
Height	400
Element type	0x09
Background color	0xFFFFFF
Title color	0x000000
Title	“Pie\0”
Number of slices	3

ELEMENT_TYPE_SURFACE

The tables below describe the ELEMENT_TYPE_SURFACE specific parameters and additional fields for the MSG_CONF_DASHBOARD_ELEMENT message. This element generates a 3D surface plot that visualizes the grid of data points from the target. There is one input stream that accepts grid type data.

Table 5-85. ELEMENT_TYPE_SURFACE Specific Parameters

Parameter	Value
ELEMENT_TYPE	0x0D
Total data length of MSG_CONF_DASHBOARD_ELEMENT when using ELEMENT_TYPE_SURFACE	47

Table 5-86. Additional Data Fields to MSG_CONF_DASHBOARD_ELEMENT for ELEMENT_TYPE_SURFACE

Field	Size	Values	Description
Fill color	3 bytes	0xRRGGBB	RGB color of surface fill
Mesh color	3 bytes	0xRRGGBB	RGB color of surface mesh
Background color	4 bytes	0xAARRGGBB	RGB color of background
Background gradient color	4 bytes	0xAARRGGBB	RGB color of background gradient
Axis color	3 bytes	0xRRGGBB	RGB color of axes
Tick color	3 bytes	0xRRGGBB	RGB color of ticks
X Rotation	1 byte	-90-90	Rotation of view around X axis in degrees
Y Rotation	1 byte	-90-90	Rotation of view around Y axis in degrees
Z Rotation	1 byte	-90-90	Rotation of view around Z axis in degrees
Attributes	1 byte	Bit 0: Show X axis Bit 1: Show Y axis Bit 2: Show Z axis Bit 3: Show fill Bit 4: Show mesh Bit 5: Use palette coloring	
Scaling mode	1 byte	0 = Static 1 = Scale roof and floor 2 = Scale roof 3 = Scale floor 4 = Sticky scale roof and floor 5 = Sticky scale roof 6 = Sticky scale floor	
Axis minimum	4 bytes	Floating point	Minimum value (floor) of Y axis
Axis maximum	4 bytes	Floating point	Maximum value (roof) of Y axis

Example

The picture below shows an example of a surface element. The element data fields for the example are shown in the table below the picture.

Notice: The appearance of elements will vary with operating system version and configuration.

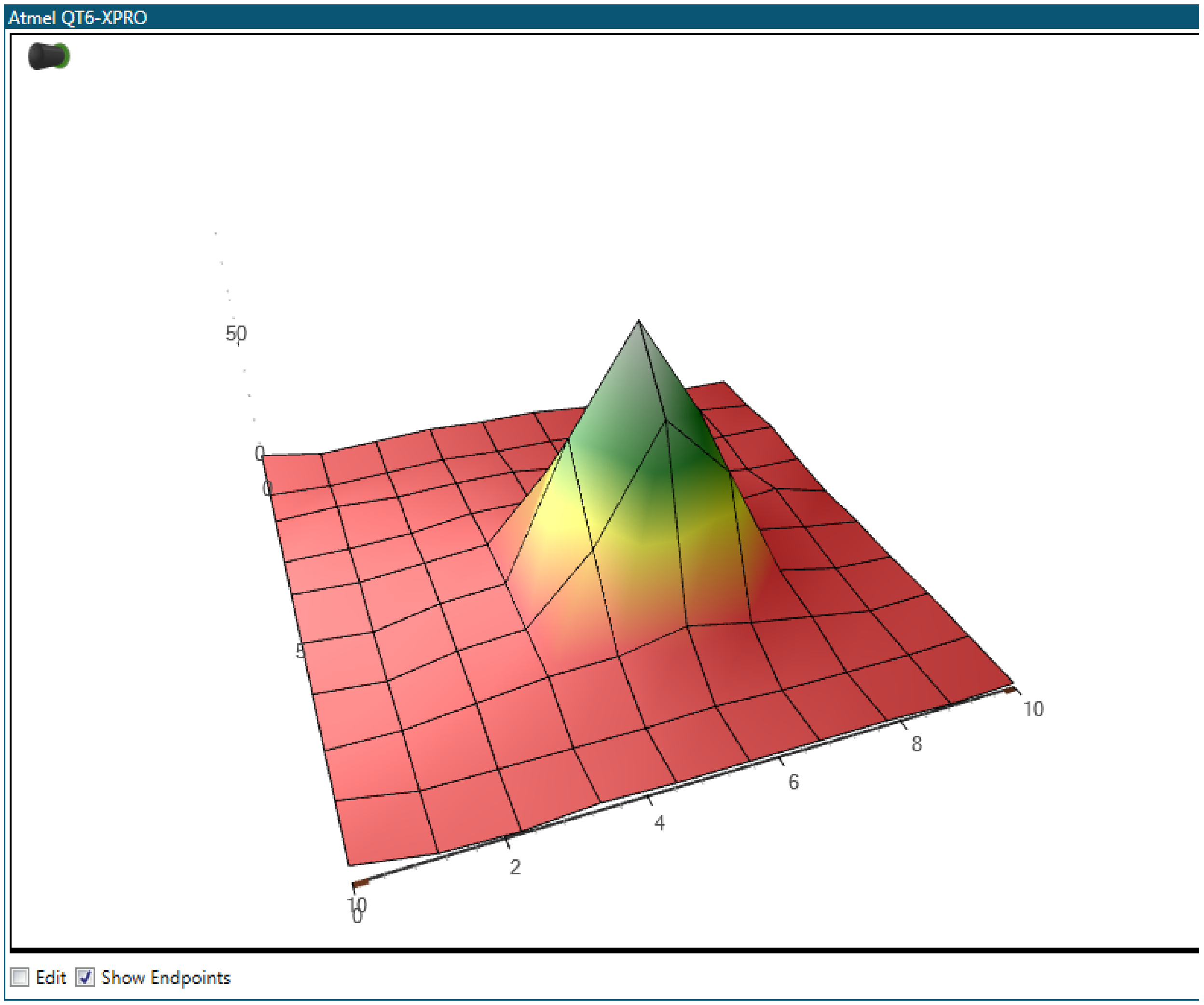


Table 5-87. Surface

Field	Value
Z-index	0
X-coordinate	5
Y-coordinate	5
Width	750
Height	590
Element type	0x0D
Fill color	0x2F4F4F
Mesh color	0x000000
Background color	0x00000000
Axis color	0x505050
Tick color	0x505050

Field	Value
X Rotation	35
Y Rotation	-70
Z Rotation	0
Attributes	0b00111111
Scaling mode	5
Axis minimum	0
Axis maximum	10

MSG_CONF_ADD_STREAM_TO_ELEMENT
This message is used to tie an already configured stream to an already configured dashboard element.

Field	Size	Values	Description
Message ID	1 byte	0x2C	
Data length	2 bytes	6	
Dashboard ID	2 bytes	0x0000-0xFFFF	ID of dashboard of given element
Element ID	2 bytes	0x0000-0xFFFF	ID of element
Stream ID	2 bytes	0x0000-0xFFFF	ID of stream for this element

MSG_CONF_ACK
Sent by PC to target to verify whether the last received configuration message was valid or not.

Field	Size	Values	Description
Message ID	1 byte	0x30	
Data length	2 bytes	1	
Status	1 byte	0 = Not OK 1 = OK	OK = Last configuration was OK and applied Not OK = Last configuration was invalid and got discarded

5.2.5.6 Data Message Details

MSG_DATA_STREAM
This message is used to send data from all enabled streams to the PC. It is possible to send one or multiple samples of data from all streams in one single message. Only data from the enabled streams will be sent.

Field	Size	Values	Description
Message ID	1 byte	0x40	
Data length	2 bytes	1 + number of streams (N) * (2 + num bytes for each stream (Xn) + length of each data sample in the stream (Xd))	
Number of streams (N)	1 byte	1 - 255	
Stream ID	2 bytes	0x0000-0xFFFF	ID of stream X
Num bytes (Xn)	1 byte	1-255	Number of bytes from stream X
Stream X data sample M	Dependent on data type of the stream (Xd)		The data of stream X

Note that the last row repeats for each sample in the stream and the three last rows repeat for each stream in the message.

MSG_DATA_STREAM_SINGLE
This message is used to send data from a single enabled stream to the PC. Only data from enabled streams will be sent.

Field	Size	Values	Description
Message ID	1 byte	0x41	
Data length	2 bytes	2 + length of one data sample of the stream (Xd)	
Stream ID	2 bytes	0x0000-0xFFFF	ID of stream
Stream data sample	Dependent on data type of the stream (Xd)		The data of stream

6. Example Code Snippets

This chapter contains the code snippets used in the examples in this user guide.

6.1 Data Polling Example Code

A Mass Storage Class example is used as an example on how to use the data polling and control of data variables features. A SAM L21 Xplained Pro board is connected to a host computer trough both the Target USB and Debug USB connectors on the kit. The ATSAML21 target device is running the USB Device MSC Example from ASF for SAM L21 Xplained Pro.

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

- Host computer with Atmel Studio 7 (or later) installed (Data Visualizer is included)
- ATSAML21 Xplained Pro kit

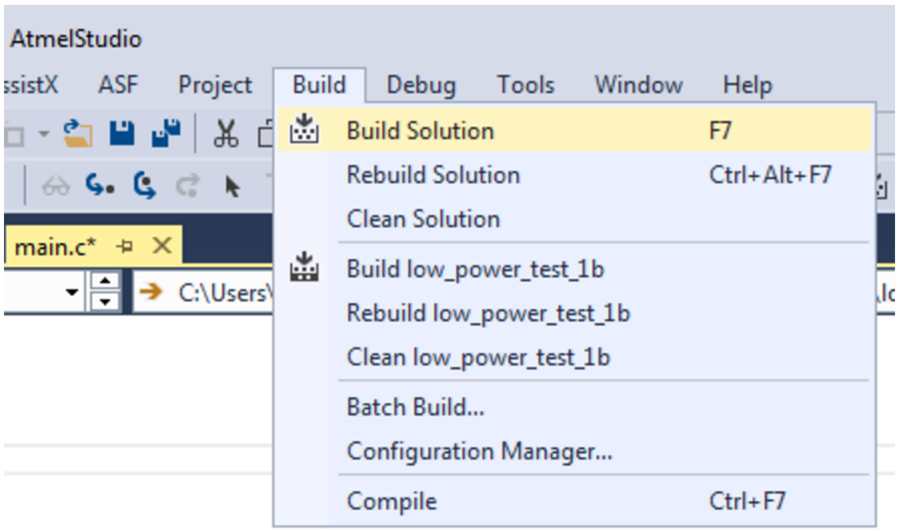
To do:

- Connect both the Target USB and Debug USB connectors on the SAM L21 Xplained Pro board

This example makes use of the **USB Device MSC Example** from ASF for SAM L21 Xplained Pro.

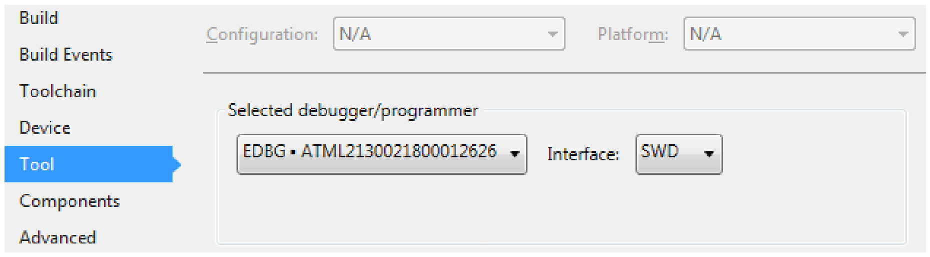
To do:

- 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



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, 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.

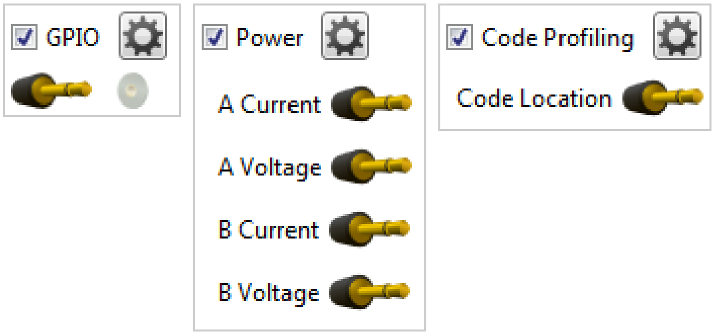
To do: Modify the two 'start' functions in ui.c to increment read and write counters on each access started.

```
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

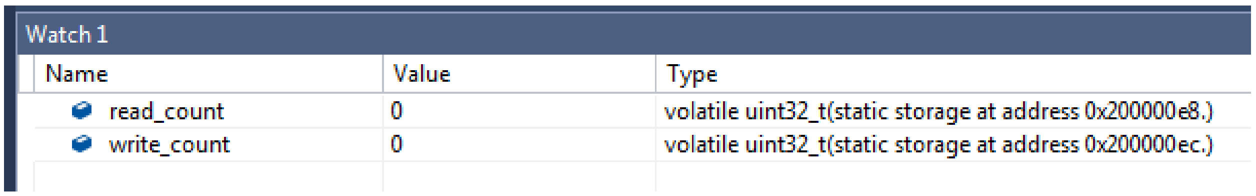
For data polling functionality, enable the Code Profiling interface.



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 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.

6.1.1 Application Interaction using Dashboard Controls

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.

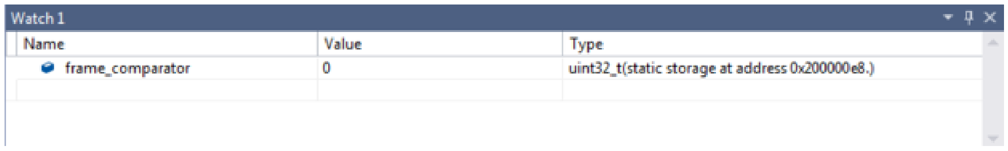
Instead of a predefined interval of 1000 USB sync pulses (1 second), add a variable compare reference to the original code.

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


Watch 1

Name	Value	Type
frame_comparator	0	uint32_t(static storage at address 0x200000e8.)

6.2 Terminal Example Code

A typical use of the **Terminal** module is print-type debugging. A serial interface is used to print debug messages from the target device to the terminal. In the following example an SPI interface will be used but the procedure will be the same for any serial interface.

The target device is an ATmega256RFR2 on an ATmega256RFR2 Xplained Pro kit. As the SPI interface is already wired internally on the board, the only connection needed is the USB cable between the host computer and the Xplained Pro board.

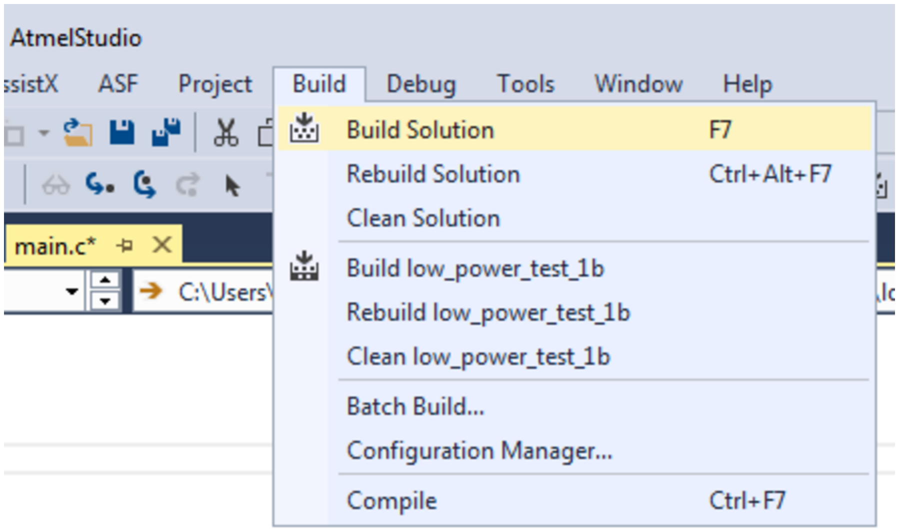
To do:

- Make a new project in Atmel Studio (**File** → **New** → **Project** → **GCC C Executable Project**)
- Replace the content of the automatically generated main.c file with the code below

```
#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 (INT4_vect)
{
    // Simple debounce
    if (PINE & (1 << 4))
    return;
    // Update LED
    if (led_on)
        PORTB |= (1 << 4);
    else
        PORTB &= ~(1 << 4);
    // 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 spi_send (const char data)
{
    PORTB &= ~(1 << PINB0);
    // Send a character to the USART
    SPDR = data;
    // Wait for the character to be sent
    while (!(SPSR & (1 << SPIF)))
    ;
    PORTB |= (1 << PINB0);
}
int main(void){
    // PORTB4 to output
    DDRB = (1 << PINB4);
    // LED OFF
    PORTB |= (1 << PINB4);
    led_on = 0;

    // Enable pullup on button pin to avoid floating line
    PORTE |= (1<<PINE4);
    // Enable falling edge interrupt for button pin
    EIMSK = (1 << INT4);
    EICRB = (1 << ISC41);

    // SPI
    // MOSI, SCK and /SS as output
    DDRB |= (1 << PINB2) | (1 << PINB1) | (1 << PINB0);
    // Set /SS high
    PORTB |= (1 << PINB0);
    // Enable SPI, Master, set clock rate fck/16
    SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);

    // Interrupts on
    sei();

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

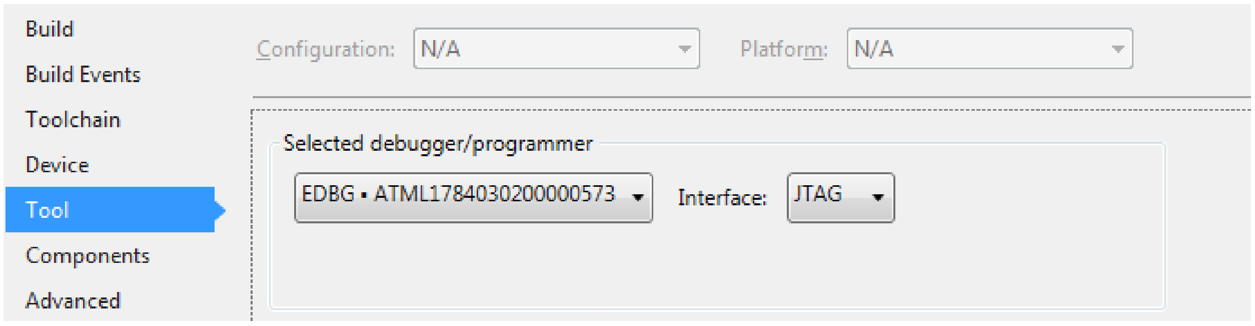
The code triggers an interrupt when the switch (SW0) on the Xplained Pro board is pushed and toggles the LED0 at each push. Each time the switch is pushed a message is printed on the SPI interface.

To do: 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

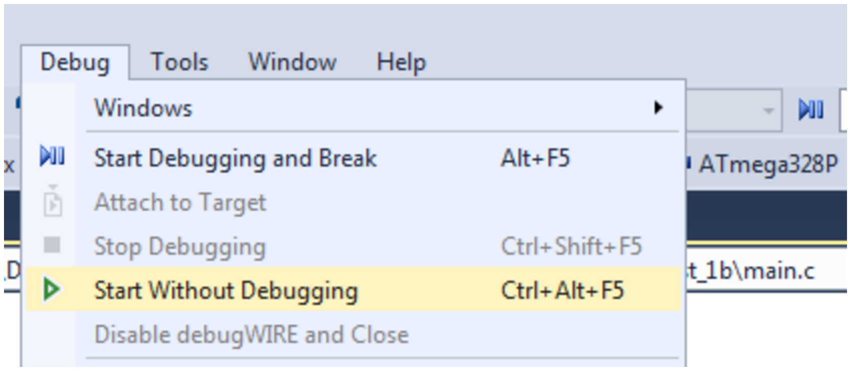


Configuration: N/A   Platform: N/A

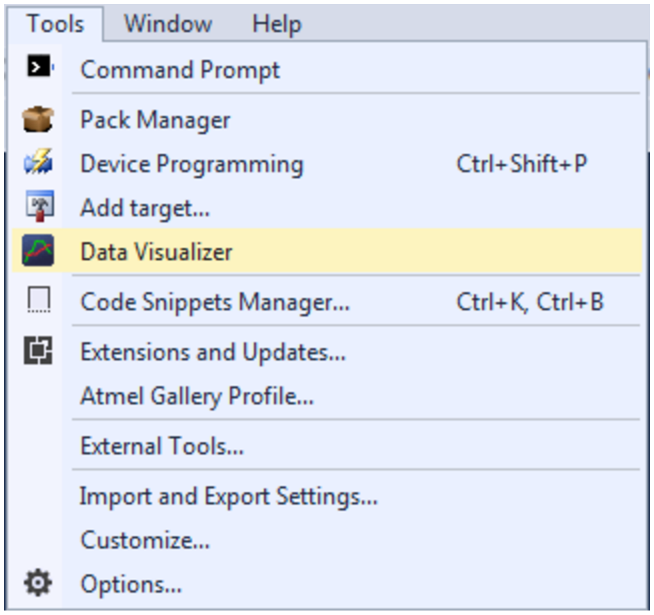
Selected debugger/programmer

EDBG • ATML1784030200000573   Interface: JTAG

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


Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Ctrl+Shift+F5
Start Without Debugging Ctrl+Alt+F5
Disable debugWIRE and Close
ATmega328P
t_1b\main.c

To do: Open the Data Visualizer as an extension inside Atmel Studio by selecting it in the **Tools** menu.



6.3 Graph Example Code

To demonstrate how to use the **Graph** module an example with a Night mode switch will be used. An ATmega256RFR2 Xplained Pro with an I/O1 Xplained Pro extension is suitable as target hardware. The I/O1 extension board features a light sensor that can be used to detect whether it is night or day. This information can be used, for example, to switch a lamp on when the surroundings turn dark.

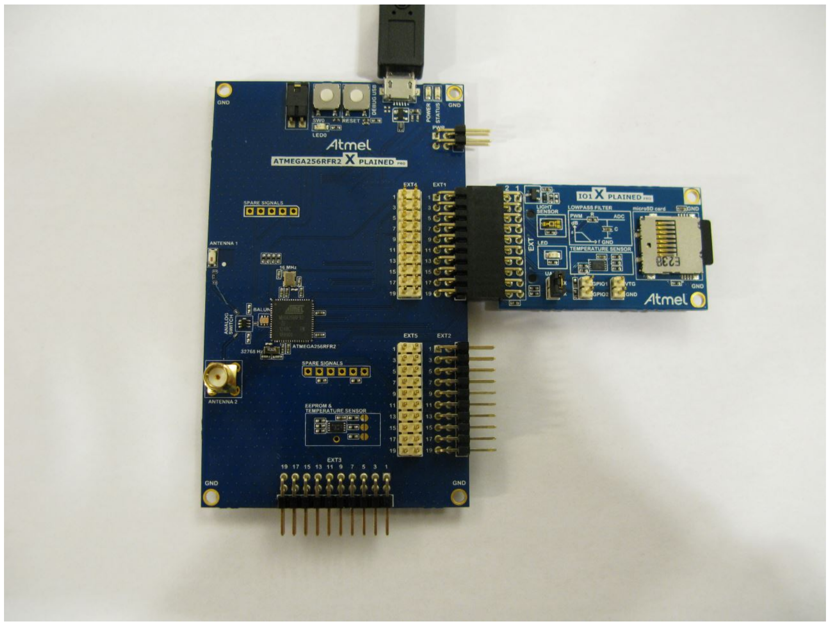
The example requires the following equipment and software:

- Host computer with Atmel Studio 7 or later installed (Data Visualizer is included)
- ATmega256RFR2 Xplained Pro kit
- I/O1 Xplained Pro extension

To run the example, the following hardware setup is required:

- I/O1 Xplained Pro extension connected to ATmega256RFR2 Xplained Pro EXT1 connector
- USB cable connected from host computer to ATmega256RFR2 Xplained Pro

A picture of the setup is shown below.



6.3.1 Basic Graph

To start with, implement a sampling of the light sensor and stream the data to the host computer to be able to view the data as a plot in a graph.

To do:

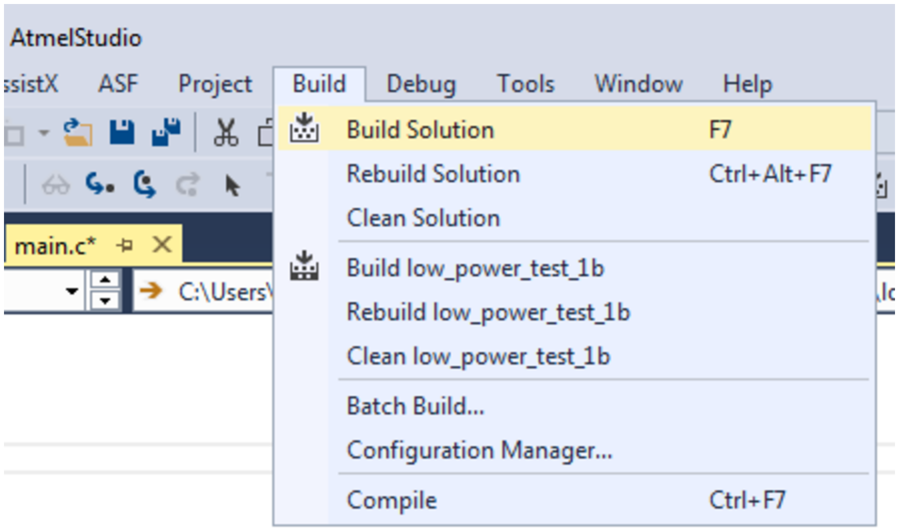
- Make a new project in Atmel Studio (**File** → **New** → **Project** → **GCC C Executable Project**)
- Replace the content of the automatically generated main.c file with the code below

```
#include <avr/io.h>
uint16_t adc_value = 0;
void adc_init(void){
    // Internal 1.5V reference, ADC0 as single ended input
    ADMUX = (1 << REFS1);
    // Enable the ADC,
    ADCSRA |= (1<<ADEN);
    // Check that the reference is OK
    while (0x00 == (ADCSRB & (1 << REFOK)));
}
uint16_t adc_sample(void){
    // Trigger an ADC conversion
    ADCSRA |= (1<<ADSC);
    // Wait for conversion to finish
    while (0x00 == (ADCSRA & (1 << ADIF)));
    // Clear the interrupt flag
    ADCSRA |= (1 << ADIF);
    return (ADC);
}
void spi_init(void){
    // Slave select (PB0), MOSI (PB2) and SCK (PB1) as output
    DDRB |= (1<<PINB0) | (1<<PINB2) | (1<<PINB1);
    //Slave select high (inactive)
    PORTB |= (1<<PINB0);
    // Master mode, enable SPI module.
    // Clock polarity and phase is kept at default (Sample on rising edge)
    SPCR = (1<<SPE) | (1<<MSTR);
}
void spi_send(uint8_t data){
    // Slave select low
```

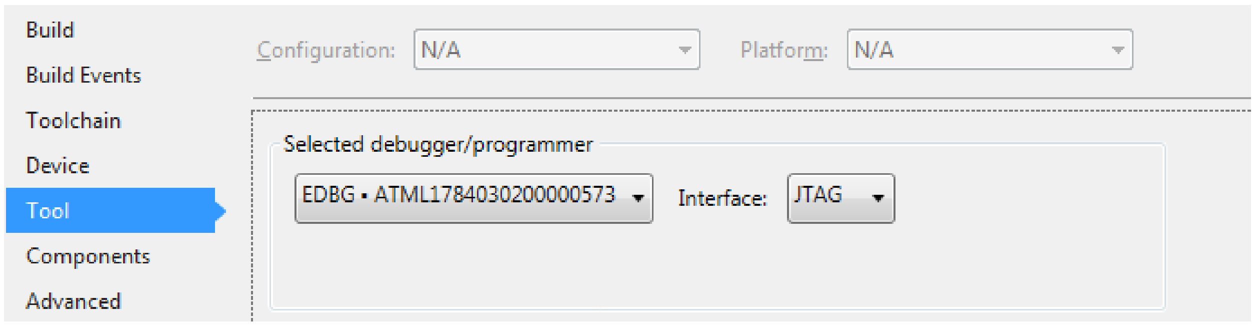
```
     PORTB &= ~(1<<PINB0);
     // Write data to shift register
     SPDR = data;
     // Wait for the transmission to complete
     while (0x00 == (SPSR & (1<<SPIF)));
     // Slave select high
     PORTB |= (1<<PINB0);
}
int main(void){
     adc_init();
     spi_init();
     while (1){
          adc_value = adc_sample();
          // Send the ADC value over SPI to the host
          // Only the 8 lower bits contain useful data
          spi_send(adc_value & 0xFF);
     }
}
```

The code samples the ADC continuously and sends the data over the SPI interface to the EDBG (Embedded Debugger) on the ATmega256RFR2 Xplained Pro board. The EDBG then sends the SPI data over DGI to the host computer. The ATmega256RFR2 ADC is 10-bit but only the lower 8 bits contain useful data in this example.

To do: 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

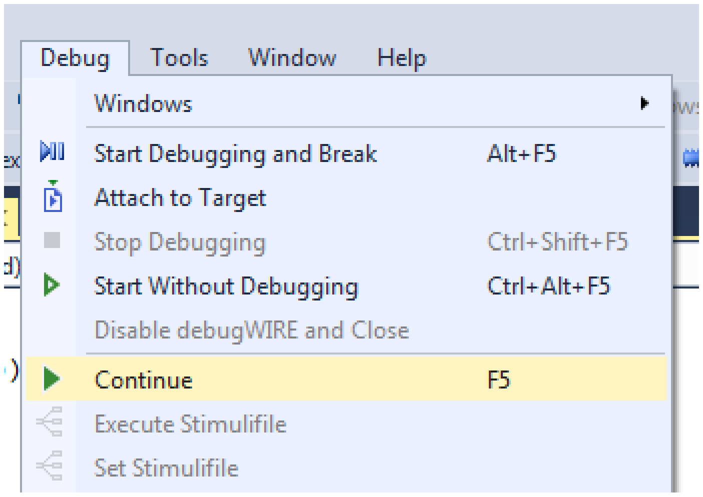


Build
Build Events
Toolchain
Device
Tool
Components
Advanced

Configuration: N/A
Platform: N/A

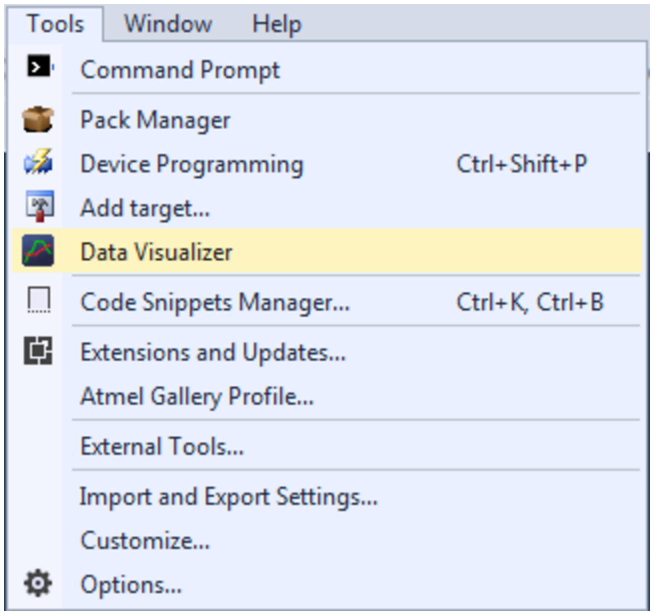
Selected debugger/programmer
EDBG • ATML1784030200000573
Interface: JTAG

To do: Program the application into the target and start the debugging by selecting **Continue** (F5).



Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Ctrl+Shift+F5
Start Without Debugging Ctrl+Alt+F5
Disable debugWIRE and Close
Continue F5
Execute Stimulifile
Set Stimulifile

To do: Open the Data Visualizer as an extension inside Atmel Studio by selecting it in the **Tools** menu.



6.3.2 Adding String Markers

It is time to implement the mode switch. To check that the switch is actually working and switching at the right threshold, the string marker feature of the **Graph** module is useful. By setting up the CDC USART interface of the ATmega256RFR2 Xplained Pro to send a string each time the mode is switched. These messages can then be shown in the graph as string markers. First, extend the code as shown below.

Important: The code will only work if the target CPU is running at 8 MHz. Use the Atmel Studio Programming dialog to set the fuses correctly (Tools → Device Programming). The clock should be set to internal RC oscillator and the CKDIV8 fuse should not be set.

```
#include <avr/io.h>
#include <avr/interrupt.h>
const char* message_on = "NIGHT MODE ON";
const char* message_off = "NIGHT MODE OFF";
uint16_t adc_value = 0;
uint8_t nightmode_threshold = 40;
uint8_t nightmode_active = 0;
void adc_init(void){
    // Internal 1.5V reference, ADC0 as single ended input
    ADMUX = (1 << REFS1);
    // Enable the ADC,
    ADCSRA |= (1<<ADEN);
    // Check that the reference is OK
    while (0x00 == (ADCSRB & (1 << REFOK)));
}
uint16_t adc_sample(void){
    // Trigger an ADC conversion
    ADCSRA |= (1<<ADSC);
    // Wait for conversion to finish
    while (0x00 == (ADCSRA & (1 << ADIF)));
    // Clear the interrupt flag
    ADCSRA |= (1 << ADIF);
    return (ADC);
}
void spi_init(void){
    // Slave select (PB0), MOSI (PB2) and SCK (PB1) as output
    DDRB |= (1<<PINB0) | (1<<PINB2) | (1<<PINB1);
    //Slave select high (inactive)
    PORTB |= (1<<PINB0);
    // Master mode, enable SPI module.
    // Clock polarity and phase is kept at default (sample on rising edge)
    SPCR = (1<<SPE) | (1<<MSTR);
}
void spi_send(uint8_t data){
    // Slave select low
    PORTB &= ~(1<<PINB0);
    // Write data to shift register
    SPDR = data;
    // Wait for the transmission to complete
    while (0x00 == (SPSR & (1<<SPIF)));
    // Slave select high
    PORTB |= (1<<PINB0);
}
void cdc_init(void){
    // Baud rate 9600 based on 8 MHz CPU clock
    UBRR1 = 51;
    // Enable the transmitter and receiver, 8 bit character size,
    // receive interrupts enabled
    UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1);
}
void cdc_send(const char data){
    // Wait for transmitter to be ready for more data
    while (0x00 == (UCSR1A & (1<<UDRE1)));
    // Send the data
    UDR1 = data;
}
void send_message(const char* message){
    while (*message)
    cdc_send(*message++);
    // String markers requires Null-termination
    cdc_send(0);
}
int main(void){
    adc_init();
    spi_init();
    cdc_init();
    while (1){
        adc_value = adc_sample();
        // Send the ADC value over SPI to the host
        // Only the 8 lower bits contain useful data
        spi_send(adc_value & 0xFF);
        //  higher adc value == less light
        if (adc_value > nightmode_threshold){
            if (0x00 == nightmode_active){
                // Changing from night mode inactive to active
                nightmode_active = 0x01;
                send_message(message_on);
            }
            } else {
            if (0x01 == nightmode_active){
                // Changing from night mode active to inactive
                nightmode_active = 0x00;
                send_message(message_off);
            }
        }
    }
}
```

To do:

- Build the project, program and run the application by simply selecting **Continue** (F5) in the **Debug** menu of Atmel Studio

6.3.3 Using Horizontal Cursor Code

So far, the **Graph** module of the Data Visualizer has been used to show the data generated by the light sensor and to show when the Night mode switch toggles between the two modes. The **Graph** module can also be used to interact with the target application while it is running. In this example, the Night mode threshold can be adjusted dynamically by using a horizontal cursor.

First, the code must be extended to accept incoming data on the CDC USART. The output of the horizontal cursor is a 4-byte float value and will be sent over the CDC interface to the target application. This float value will be used as the threshold for the Night mode switch.

```
#include <avr/io.h>
#include <avr/interrupt.h>
const char* message_on = "NIGHT MODE ON";
const char* message_off = "NIGHT MODE OFF";
union u_float{
    float flt;
    char data[4];
    };
uint16_t adc_value = 0;
uint8_t nightmode_threshold;
uint8_t nightmode_active = 0;
union u_float cdc_received_data;
uint8_t cdc_read_index=0;
ISR (USART1_RX_vect){
    // A byte is received on the CDC UART, MSB first
    cdc_received_data.data[cdc_read_index] = UDR1 & 0xFF;
    if (3 == cdc_read_index){
        // A complete float value is received
        nightmode_threshold = (uint8_t) cdc_received_data.flt;
        cdc_read_index = 0;
    }
    else {
        cdc_read_index++;
    }
}
void adc_init(void){
    // Internal 1.5V reference, ADC0 as single ended input
    ADMUX = (1 << REFS1);
    // Enable the ADC,
    ADCSRA |= (1<<ADEN);
    // Check that the reference is OK
    while (0x00 == (ADCSRB & (1 << REFOK)));
}
uint16_t adc_sample(void){
    // Trigger an ADC conversion
    ADCSRA |= (1<<ADSC);
    // Wait for conversion to finish
    while (0x00 == (ADCSRA & (1 << ADIF)));
    // Clear the interrupt flag
    ADCSRA |= (1 << ADIF);
    return (ADC);
}
void spi_init(void){
    // Slave select (PB0), MOSI (PB2) and SCK (PB1) as output
    DDRB |= (1<<PINB0) | (1<<PINB2) | (1<<PINB1);
    //Slave select high (inactive)
    PORTB |= (1<<PINB0);
    // Master mode, enable SPI module.
    // Clock polarity and phase is kept at default (sample on rising edge)
    SPCR = (1<<SPE) | (1<<MSTR);
}
void spi_send(uint8_t data){
    // Slave select low
    PORTB &= ~(1<<PINB0);
    // Write data to shift register
    SPDR = data;
    // Wait for the transmission to complete
    while (0x00 == (SPSR & (1<<SPIF)));
    // Slave select high
    PORTB |= (1<<PINB0);
}
void cdc_init(void){
    // Baud rate 9600 based on 8 MHz CPU clock
    UBRR1 = 51;
    // Enable the transmitter and receiver, 8 bit character size,
    // receive interrupts enabled
    UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1);
}
void cdc_send(const char data){
    // Wait for transmitter to be ready for more data
    while (0x00 == (UCSR1A & (1<<UDRE1)));
    // Send the data
    UDR1 = data;
}
void send_message(const char* message){
    while (*message)
        cdc_send(*message++);
    // String markers requires Null-termination
    cdc_send(0);
}
int main(void){
    adc_init();
    spi_init();
    cdc_init();
    // Interrupts on
    sei();
       while (1){
        adc_value = adc_sample();
        // Send the ADC value over SPI to the host
        // Only the 8 lower bits contain useful data
        spi_send(adc_value & 0xFF);
        //  higher adc value == less light
        if (adc_value > nightmode_threshold){
            if (0x00 == nightmode_active){
                // Changing from nightmode inactive to active
                nightmode_active = 0x01;
                send_message(message_on);
            }
        } else {
            if (0x01 == nightmode_active){
                // Changing from nightmode active to inactive
                nightmode_active = 0x00;
                send_message(message_off);
            }
        }
    }
}
```

To do:

- Build the project, program and run the application by simply selecting **Continue** (F5) in the **Debug** menu of Atmel Studio

6.4 Oscilloscope Example Code

To demonstrate how to use the **Oscilloscope** module a light sensor target application will be used. The light sensor example is a rather generic data source where an Analog-to-Digital Converter is used to sample the sensor, and the example applies to a wide range of other data sources.

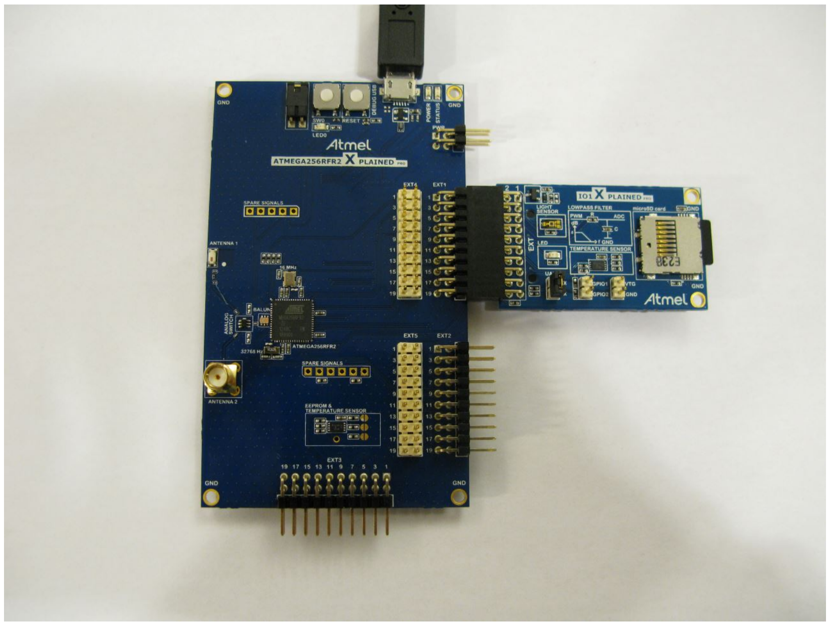
The example requires the following equipment and software:

- Host computer with Atmel Studio 7 or later installed (Data Visualizer is included)
- ATmega256RFR2 Xplained Pro kit
- I/O1 Xplained Pro extension

To run the example, the following hardware setup is required:

- I/O1 Xplained Pro extension connected to ATmega256RFR2 Xplained Pro EXT1 connector
- USB cable connected from host computer to ATmega256RFR2 Xplained Pro

A picture of the setup is shown below.



To be able to view the light sensor data in the **Oscilloscope** module, the ATmega256RFR2 target has to be programmed with code that samples the light sensor and sends the data to the Embedded Debugger (EDBG) on the ATmega256RFR2 Xplained Pro over a serial interface. The EDBG then uses the Data Gateway Interface (DGI) to send the data to the host computer.

First, a new project for the target application code has to be set up in Atmel Studio.

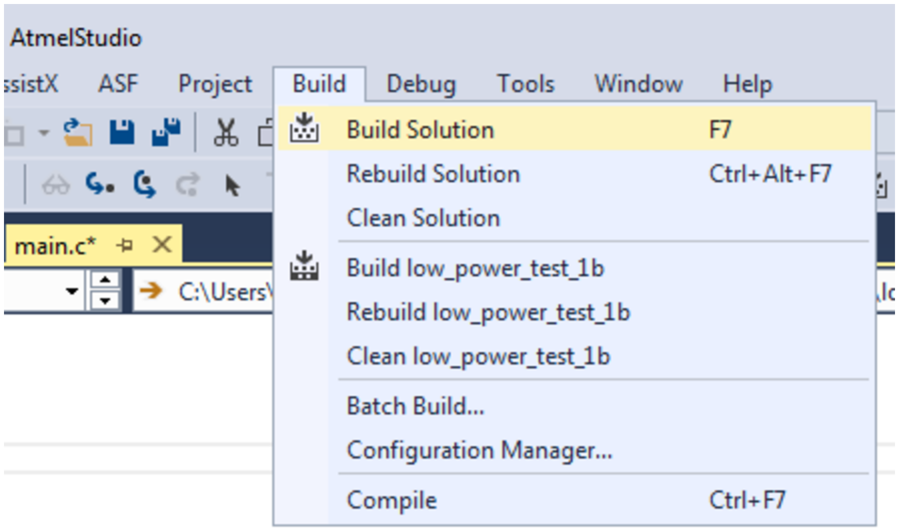
To do:

- Make a new project in Atmel Studio (**File** → **New** → **Project** → **GCC C Executable Project**)
- Replace the content of the automatically generated main.c file with the code below

```
#include <avr/io.h>
#include <avr/interrupt.h>
uint16_t adc_value = 0;
volatile uint8_t send_data = 0;
void adc_init(void){
    // Internal 1.5V reference, ADC0 as single ended input
    ADMUX = (1 << REFS1);
    // Enable the ADC, auto triggered mode, interrupt on conversion finished
    ADCSRA |= (1<<ADEN) | (1<<ADATE) | (1<<ADIE);
    // Timer/Counter compare match A as trigger of ADC conversion
    ADCSRB |= (1<<ADTS1) | (1<<ADTS0);
    // Check that the reference is OK
    while (0x00 == (ADCSRB & (1 << REFOK)));
}
void spi_init(void){
    // Slave select (PB0), MOSI (PB2) and SCK (PB1) as output
    DDRB |= (1<<PINB0) | (1<<PINB2) | (1<<PINB1);
    //Slave select high (inactive)
    PORTB |= (1<<PINB0);
    // 2X mode, 4MHz SPI clock when CPU clock is 8MHz
    SPSR |= (1<<SPI2X);
    // Master mode, enable SPI module.
    // Clock polarity and phase is kept at default (Sample on rising edge)
    SPCR = (1<<SPE) | (1<<MSTR);
}
void spi_send(uint8_t data){
    // Slave select low
    PORTB &= ~(1<<PINB0);
    // Write data to shift register
    SPDR = data;
    // Wait for the transmission to complete
    while (0x00 == (SPSR & (1<<SPIF)));
    // Slave select high
    PORTB |= (1<<PINB0);
}
void timer_init(){
    // Set TOP value for timer (output compare A value)
    OCR0A = 80;
    // Clear timer on compare match mode
    TCCR0A = (1<<WGM01);
    // Timer clocked by CPU clock, no prescaler
    TCCR0B = (1<<CS00);
}
ISR (ADC_vect){
    // Store the light sensor sample
    adc_value = (ADC);
    // Clear timer interrupt flag to enable the next sample
    TIFR0 |= (1<<OCF0A);
    // Flag sending of data to host
    send_data = 1;
}
int main(void){
    timer_init();
    adc_init();
    spi_init();
    // Interrupts on
    sei();
    while (1){
        if (1 == send_data){
            send_data = 0;
            // Send the ADC value over SPI to the host
            // Only the 8 lower bits contain useful data
            spi_send(adc_value & 0xFF);
        }
    }
}
```

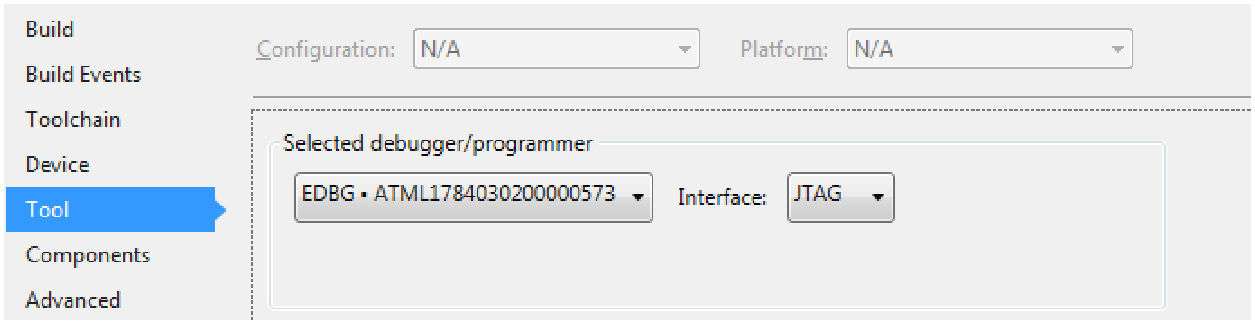
The code configures the ADC to take a new sample every 10th µs giving a sample rate of 100 kHz. This is achieved by using a timer that counts up to 80 before resetting. The code is based on the target CPU running on the internal 16 MHz clock with a clock prescaler of 2 (default) and the CKDIV8 fuse not set. The data samples are sent to the EDBG over the DGI SPI interface. The SPI interface is running at 4 MHz. The ATmega256RFR2 ADC is 10-bit but only the lower 8 bits contain useful data in this example.

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



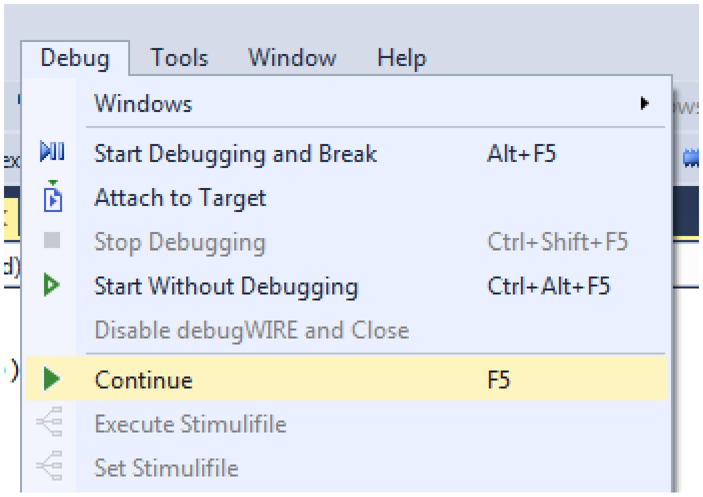
To do:

- 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

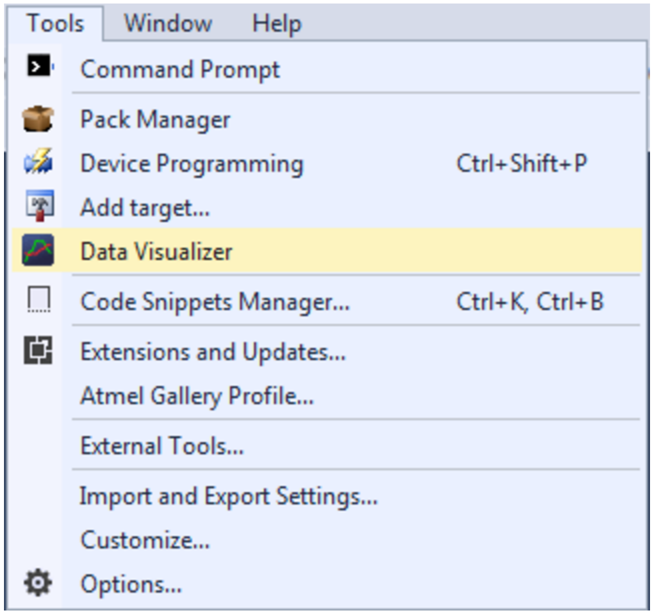


Configuration: N/A
Platform: N/A
Selected debugger/programmer
EDBG • ATML1784030200000573
Interface: JTAG

To do: Program the application into the target and start the debugging by selecting **Continue** (F5).


Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Ctrl+Shift+F5
Start Without Debugging Ctrl+Alt+F5
Disable debugWIRE and Close
Continue F5
Execute Stimulifile
Set Stimulifile

To do: Open the Data Visualizer as an extension inside Atmel Studio by selecting it in the **Tools** menu.



6.5 Dashboard Example Code

To demonstrate how to use the **Dashboard** module, an example with a Night mode switch will be used. An ATmega256RFR2 Xplained Pro with an I/O1 Xplained Pro extension is suitable as target hardware. The I/O1 extension board features a light sensor that can be used to detect whether it is night or day. This information can be used, for example, to switch a lamp on when the surroundings turn dark.

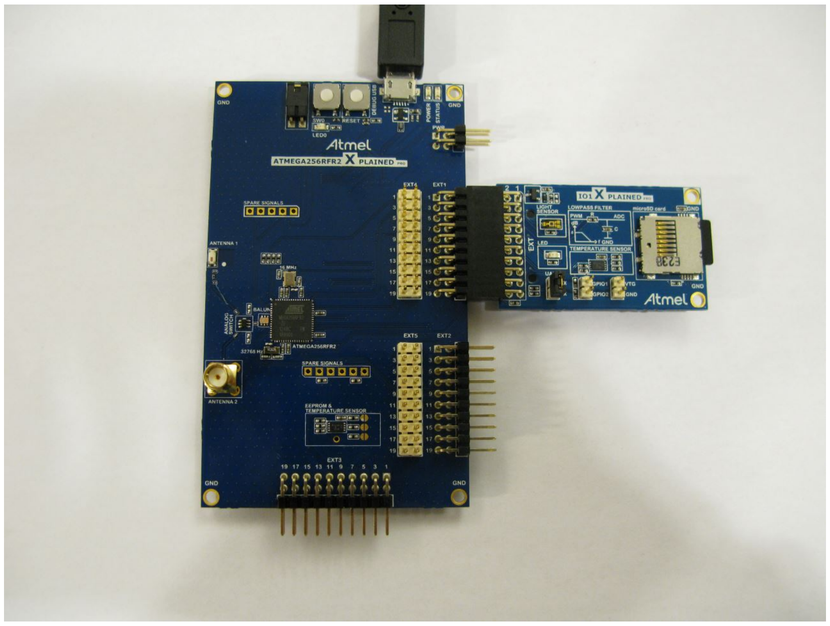
The example requires the following equipment and software:

- Host computer with Atmel Studio 7 or later installed (Data Visualizer is included)
- ATmega256RFR2 Xplained Pro kit
- I/O1 Xplained Pro extension

To run the example, the following hardware setup is required:

- I/O1 Xplained Pro extension connected to ATmega256RFR2 Xplained Pro EXT1 connector
- USB cable connected from host computer to ATmega256RFR2 Xplained Pro

A picture of the setup is shown below.



The ATmega256RFR2 target on the Xplained Pro must be programmed with code that implements the Night mode switch.

To do:

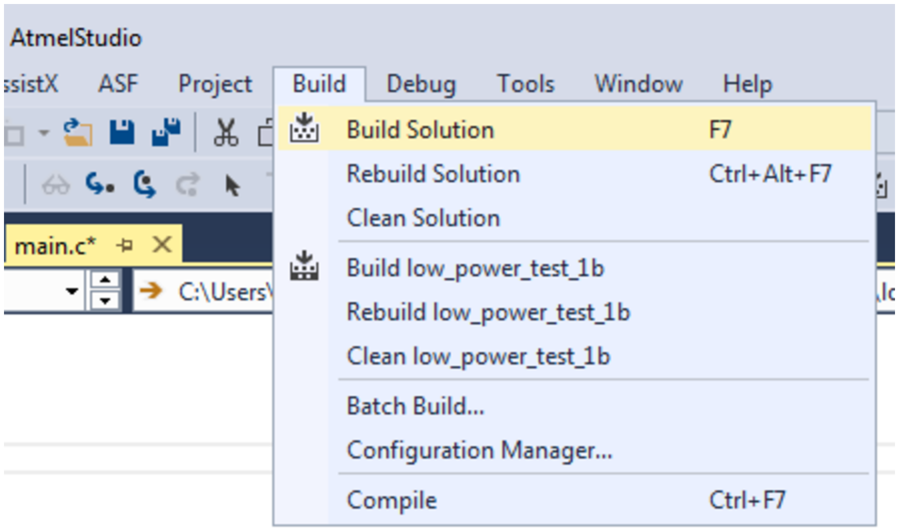
- Make a new project in Atmel Studio (**File** → **New** → **Project** → **GCC C Executable Project**)
- Replace the content of the automatically generated main.c file with the code below

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

union u_double{
    double dbl;
    char data[8];
};

uint16_t adc_value = 0;
uint8_t nightmode_threshold;
uint8_t nightmode_active = 0;
union u_double cdc_received_data;
uint8_t cdc_read_index=0;

ISR (USART1_RX_vect){
    // A byte is received on the CDC UART, MSB first
    cdc_received_data.data[cdc_read_index] = UDR1 & 0xFF;
    if (7 == cdc_read_index){
        // A complete double value is received
        nightmode_threshold = (uint8_t) cdc_received_data.dbl;
        cdc_read_index = 0;
    }
    else {
        cdc_read_index++;
    }
}
void adc_init(void){
    // Internal 1.5V reference, ADC0 as single ended input
    ADMUX = (1 << REFS1);
    // Enable the ADC,
    ADCSRA |= (1<<ADEN);
    // Check that the reference is OK
    while (0x00 == (ADCSRB & (1 << REFOK)));
}
uint16_t adc_sample(void){
    // Trigger an ADC conversion
    ADCSRA |= (1<<ADSC);
    // Wait for conversion to finish
    while (0x00 == (ADCSRA & (1 << ADIF)));
    // Clear the interrupt flag
    ADCSRA |= (1 << ADIF);
    return (ADC);
}
void spi_init(void){
    // Slave select (PB0), MOSI (PB2) and SCK (PB1) as output
    DDRB |= (1<<PINB0) | (1<<PINB2) | (1<<PINB1);
    //Slave select high (inactive)
    PORTB |= (1<<PINB0);
    // Master mode, enable SPI module. Clock polarity and phase is kept at default (Rising
edge is leading edge and sample on leading edge)
    SPCR = (1<<SPE) | (1<<MSTR);
}
void spi_send(uint8_t data){
    // Slave select low
    PORTB &= ~(1<<PINB0);
    // Write data to shift register
    SPDR = data;
    // Wait for the transmission to complete
    while (0x00 == (SPSR & (1<<SPIF)));
    // Slave select high
    PORTB |= (1<<PINB0);
}
void cdc_init(void){
    // Baud rate 9600 based on 8 MHz CPU clock
    UBRR1 = 51;
    // Enable the transmitter and receiver, 8 bit character size, receive interrupts enabled
    UCSR1B = (1<<RXEN1) | (1<<TXEN1) | (1<<RXCIE1);
}
void cdc_send(const char data){
    // Wait for transmitter to be ready for more data
    while (0x00 == (UCSR1A & (1<<UDRE1)));
    // Send the data
    UDR1 = data;
}
int main(void){
    adc_init();
    spi_init();
    cdc_init();
    // Interrupts on
    sei();
    while (1){
        adc_value = adc_sample();
        // Send the ADC value over SPI to the host
        // Only the 8 lower bits contain useful data
        spi_send(adc_value & 0xFF);
        //  higher adc value == less light
        if (adc_value > nightmode_threshold){
            if (0x00 == nightmode_active){
                // Changing from nightmode inactive to active
                nightmode_active = 0x01;
            }
        } else {
            if (0x01 == nightmode_active){
                // Changing from nightmode active to inactive
                nightmode_active = 0x00;
            }
        }
        cdc_send(nightmode_active);
    }
}
```

The code samples the ADC continuously and sends the data over the SPI interface to the EDBG (Embedded Debugger) on the ATmega256RFR2 Xplained Pro board. The EDBG then sends the SPI data over DGI to the host computer. The ATmega256RFR2 ADC is 10-bit but only the lower 8 bits contain useful data in this example.

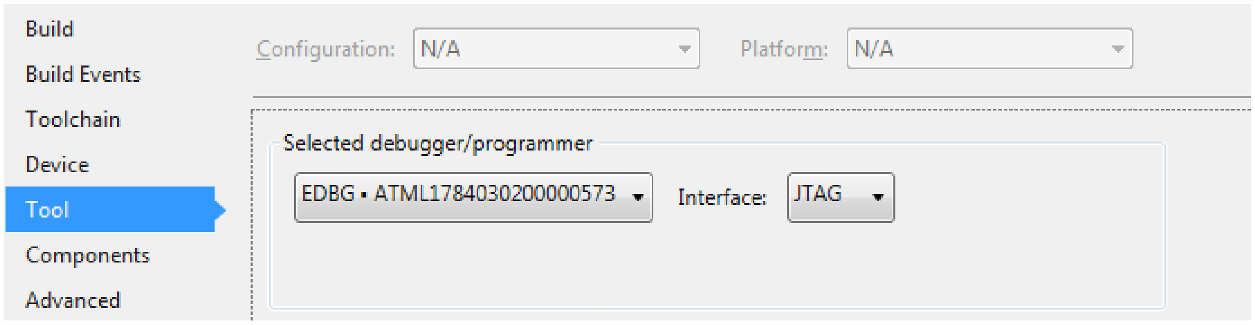
In addition, the code sets up the CDC USART and sends the state of the Night mode switch as a single byte. The received data on the CDC USART is parsed as a double value and is used as threshold for the Night mode switch.

To do: 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

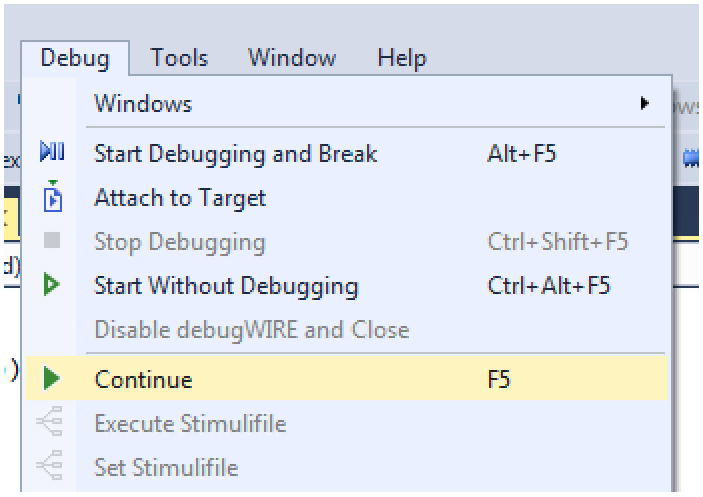


Configuration: N/A Platform: N/A

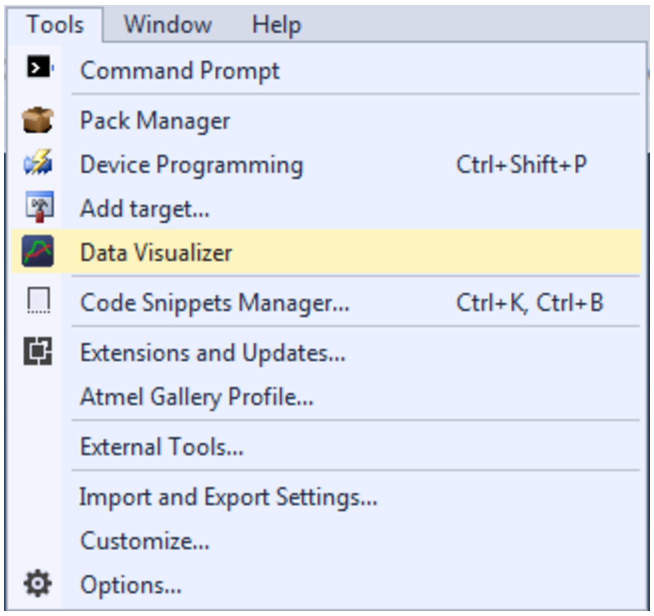
Selected debugger/programmer

EDBG • ATML1784030200000573 Interface: JTAG

To do: Program the application into the target and start the debugging by selecting **Continue** (F5).

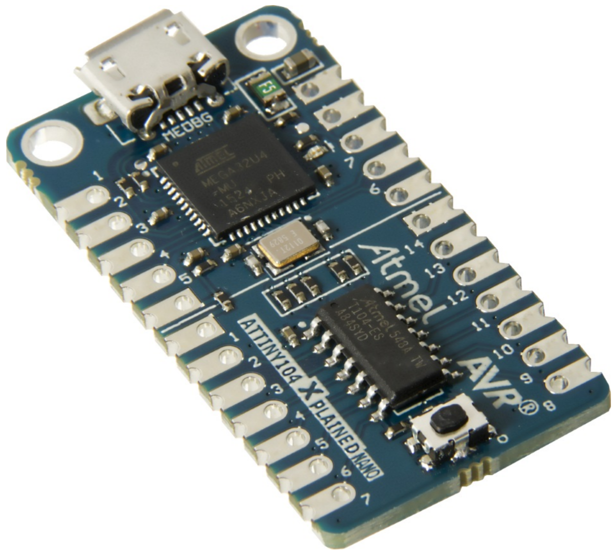

Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Ctrl+Shift+F5
Start Without Debugging Ctrl+Alt+F5
Disable debugWIRE and Close
Continue F5
Execute Stimulifile
Set Stimulifile

To do: Open the Data Visualizer as an extension inside Atmel Studio by selecting it in the **Tools** menu.



6.6 Auto-Configuration Example Code

The Auto-Configuration feature of the Data Stream protocol can be used over any DGI serial interface or Serial Port. In this example, the Virtual COM port of an ATtiny104 Xplained Nano board will be used. This board has a built-in debugger (mEDBG) that can be used to program the target ATtiny104 device, and the emEDBG provides a Virtual COM port over USB that is connected to the UART pins of the ATtiny104.



The only hardware connection required to run this example is to connect a USB cable between the host computer and the Xplained Nano board.

The Data Stream protocol will be used to send the state of the button on the Xplained Nano and the value of a 16-bit counter to the host computer.

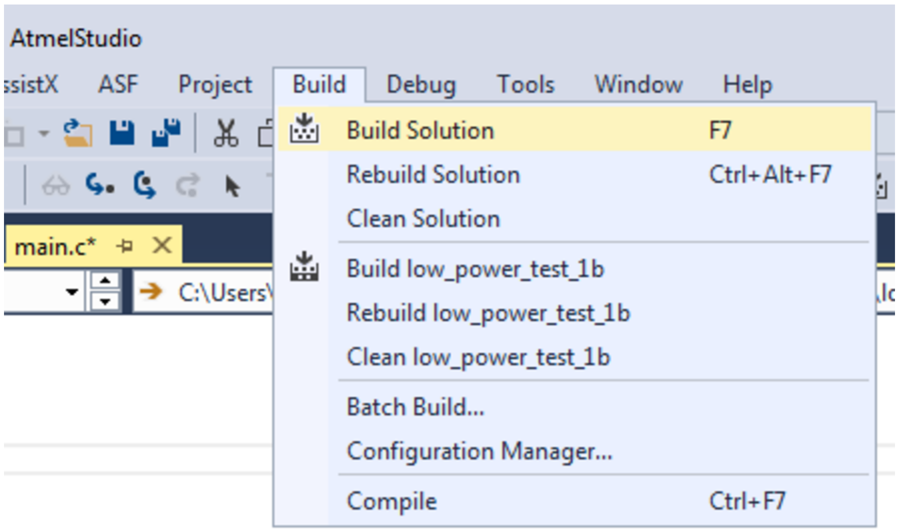
To do:

- Make a new project in Atmel Studio (**File** → **New** → **Project** → **GCC C Executable Project**)
- Replace the content of the automatically generated main.c file with the code below

```
#include <avr/io.h>
uint8_t start_token = 0xAB;
uint8_t config_id_packet[] = {
    /* Token */
    0x5F,
    /* Specify checksum LRC8 */
    0xB4, 0x00, 0x86, 0x4A,
    /* Configuration ID */
    0xC0, 0xFF, 0xEE, 0xC0, 0xFF, 0xEE, 0xC0, 0xFF, 0xEE, 0xC0, 0xFF, 0xEE,
    /* The actual checksum */
    0x78,
    /* Inverse of Token */
    0xA0
};
uint16_t count = 0;
uint8_t send_id = 0;
void uart_send(uint8_t byte){
   /* Wait for empty transmit buffer */
   while ( !( UCSRA & (1<<UDRE)) );
   /* Put data into buffer, sends the data */
   UDR = byte;
}
int main(void){
    /* Set clock prescaler to 1 to get a 8MHz main clock */
    CCP = 0xD8; // Change protection register
    CLKPSR = 0x00;
    /* Set baud rate to 38400: UBR = (8000000/(16*38400))-1 = 12 */
    UBRR = 12;
    /* Enable UART transmitter */
    UCSRB = (1<<TXEN);
    /* Enable pull-up on button pin*/
    PUEB |= (1<<PINB1);
    while (1) {
        /* Send configuration ID */
        if (send_id == 0){
            for (int i = 0; i < sizeof(config_id_packet);i++)
                uart_send(config_id_packet[i]);
            send_id = 100;
        }
        /* Send data */
        uart_send(start_token);
        uart_send(count & 0xFF);
        uart_send(count >> 8);
        uart_send (!(PINB & (1<<PINB1)));
        uart_send(~start_token);
        count +=100;
        send_id--;
    }
}
```

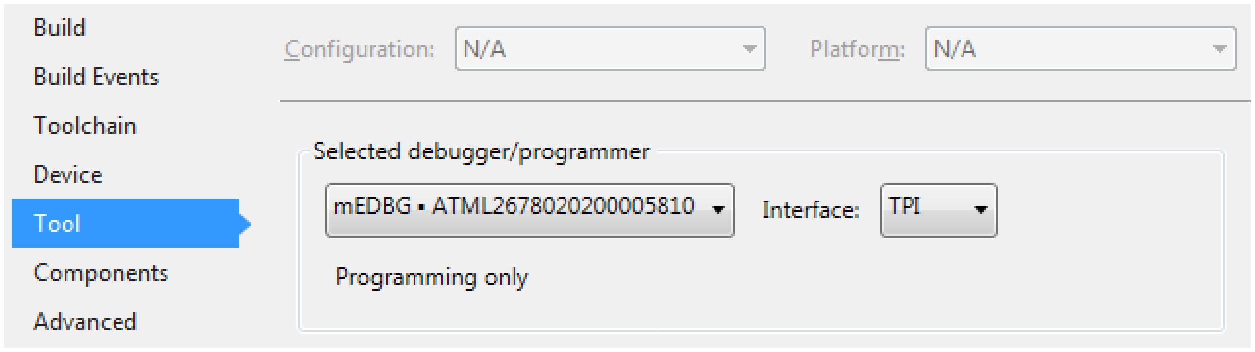
The code continuously sends the value of a 16-bit counter that is incremented by 100 for each iteration. In addition, the state of the ATtiny104 Xplained Nano push button (PINB1) is sent. Note that the push button pin is low when the button is pushed. Every 100th loop iteration the Auto-Configuration ID packet is sent. The Auto-Configuration ID is 0xC0, 0xFF, 0xEE, 0xC0, 0xFF, 0xEE, 0xC0, 0xFF, 0xEE, 0xC0, 0xFF, 0xEE, so the configuration files must be named “C0FFEEC0FFEEC0FFEEC0FFEE”. The Auto-Configuration ID packet format is described in Auto-Configuration and the format of the data packet is described in Stream Format. The target ATtiny104 device is clocked at 8 MHz internal RC oscillator and the UART for the Virtual COM port is run at a baud rate of 38400 with 8-bit character width, one Stop bit and no parity.

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



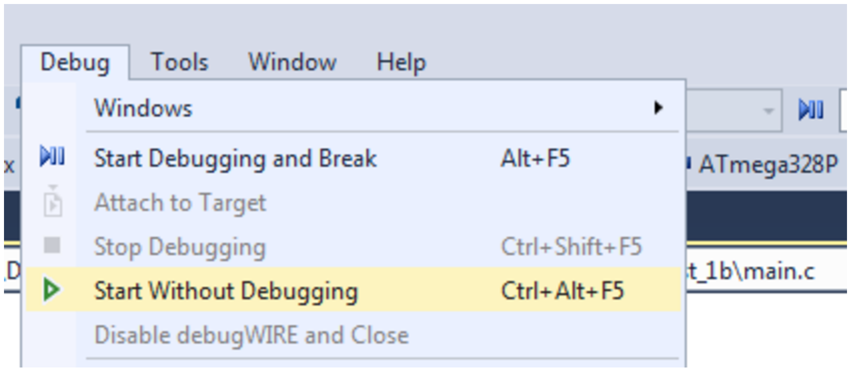
To do:

- 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

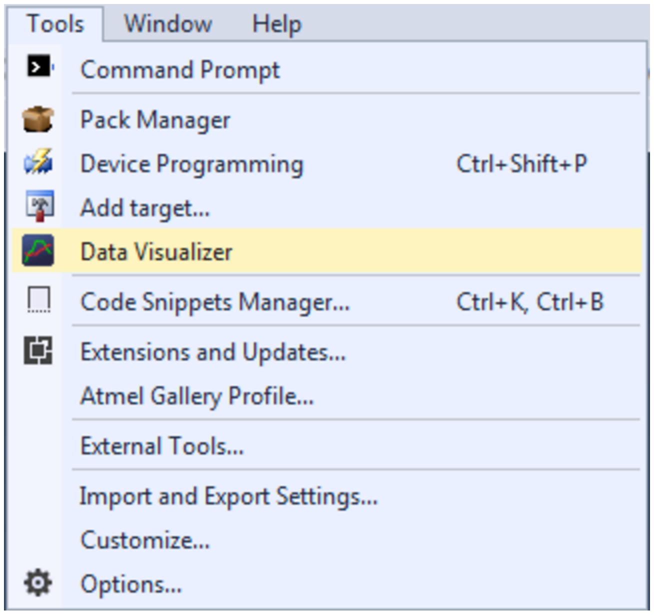


Configuration: N/A
Platform: N/A
Selected debugger/programmer
mEDBG • ATML2678020200005810
Interface: TPI
Programming only

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


Debug
Tools
Window
Help
Windows
Start Debugging and Break Alt+F5
Attach to Target
Stop Debugging Ctrl+Shift+F5
Start Without Debugging Ctrl+Alt+F5
Disable debugWIRE and Close
ATmega328P
t_1b\main.c

To do: Open the Data Visualizer as an extension inside Atmel Studio by selecting it in the **Tools** menu.



7. Known Issues

Generally, Data Visualizer requires all Windows updates to be installed in order to function properly. In particular, the following optional KBs (Knowledge Base) are essential for graphs to be shown correctly:

Windows Version	Important KBs
Windows 8.1	• KB2975719, https://support.microsoft.com/en-us/help/2975719 • KB2978092, https://support.microsoft.com/en-us/help/2978092
Windows 8	• KB2975331, https://support.microsoft.com/en-us/help/2975331 • KB2978092, https://support.microsoft.com/en-us/help/2978092
Windows 7	• KB2670838, https://support.microsoft.com/en-us/help/2670838 • KB2978092, https://support.microsoft.com/en-us/help/2978092

8. Document Revision History

Document Revision	Date	Comment
40001903B	09/2017	Updated Horizontal Cursor in Graph Module
40001903A	06/2017	• Microchip version DS40001903A replaces Atmel version 42730B • Added auto-hide plot option for auto-configuration of Graph elements • Added known issue regarding required KBs for Data Visualizer to work. • Added Data Stream auto-configuration section • Updated DGI Control Panel and Serial Port Control Panel descriptions with the latest Auto-detect protocols functionality • Updated Power Interface Configuration regarding Power Debugger Vout control • Added Stack Monitor, AVR OCD messages and AVR sleep monitor features to Code Profiling DGI interface • Updated Table element in Dashboard View according to implementation changes • Updated Graph element in Dashboard View according to added configuration options • General screenshot updates
42730B	01/2017	Added Surface and Table elements to Custom Dashboard view. New document template.
42730A	05/2016	Initial document release.

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

Legal Notice

Trademarks

ISBN: 978-1-5224-2151-1

Quality Management System Certified by DNV

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820