TRAINING MANUAL

Getting Started with Atmel START on the SAM L21 Xplained Pro B

AN-16642

Prerequisites

- **Hardware Prerequisites**
  - Atmel® | SMART™ SAM L21 Xplained Pro B revision 5 (or newer)
    - Embeds an ATSAML21J18B revision C (or newer)
  - Atmel® IO1 Xplained Pro Extension Board
  - One Micro USB cable (type A/Micro B)

- **Software Prerequisites**
  - Atmel® Studio 7 (Version: 7.0.1006 or higher)
    - Atmel Start (version 1.0.91.0 or higher)
    - Data Visualizer Extension (version 2.6.475 or higher)
  - Internet connection

- **Audience:** Beginner

- **Estimated Completion Time:** 90 min

Introduction

The goal of this hands on is to learn how to use the Atmel Start Web UI but also to get familiar with Atmel Start generated code (ASF version 4).

It will also present Atmel Data Visualizer tool which is an Atmel Studio 7 program used for data processing and visualizing.

Table of Contents

Icon Key Identifiers

ℹ	**INFO**	Delivers contextual information about a specific topic
✎	**TIPS**	Highlights useful tips and techniques
📋	**TO DO**	Highlights objectives to be completed
☑	**RESULT**	Highlights the expected result of an assignment step
!	**WARNING**	Indicates important information
➔	**EXECUTE**	Highlights actions to be executed out of the target when necessary

1. Introduction

The goal of this hands on is to describe and illustrate how to create a project with Atmel Start and get an application up and running.

Atmel Start (http://start.atmel.com) is a tool that will help you to select and configure software components, drivers, middleware and example projects to tailor your embedded application in a usable and optimized manner.

The workflow is quite straight forward: filter MCUs by requirements before starting a project. Next you add components to your project, configure each component, export the project and add it into your favorite IDE for further development.

The hands-on application will retrieve data from both the light and the temperature sensors of the IO1 Xplained Pro extension board.

These data will be sent to the on-board embedded debugger (EDBG) in order to display some graphs using Atmel Studio 7 Data Visualizer tool.

The following drivers will be implemented:

- USART to print debug messages on a Virtual COM port.
- ADC to take samples every second from the light sensor.
- I2C to take samples from temperature sensor.
- SPI to send data to the on-board Embedded Debugger (EDBG).

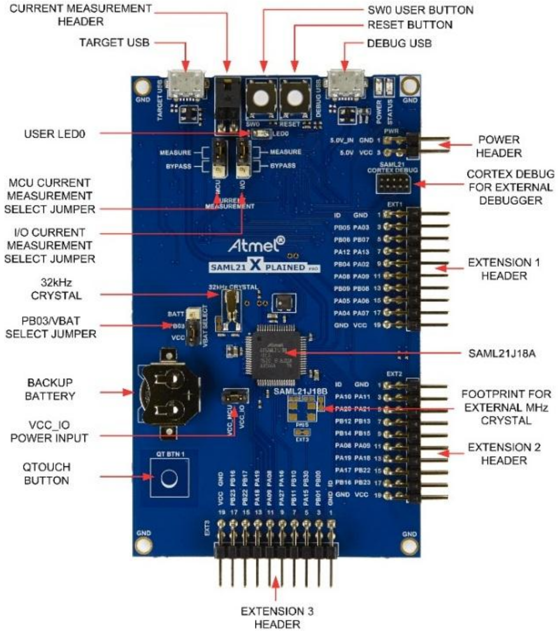
Data Visualization of temperature and light sensor values will be finally done using Atmel Studio 7 Data Visualizer.

1.1 Atmel® | SMART™ SAM L21 Xplained Pro B

The Atmel® SAM L21 Xplained Pro evaluation kit is a hardware platform to evaluate the ATSAML21J18B microcontroller.

Supported by the Atmel Studio integrated development platform, the kit provides easy access to the features of the Atmel ATSAML21J18B and explains how to integrate the device in a custom design.

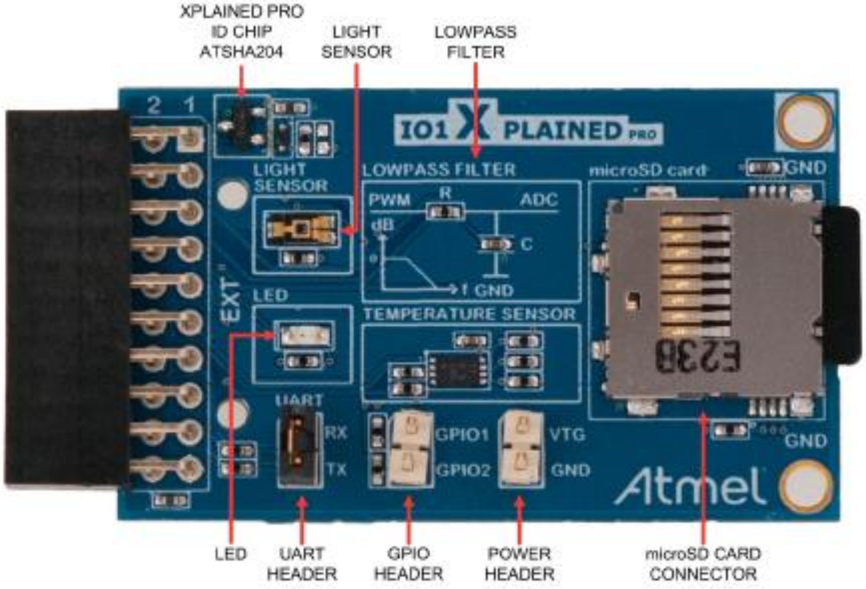
The Xplained Pro MCU series evaluation kits includes an on-board Embedded Debugger so that no external tools are necessary to program or debug the ATSAML21J18B.



1.2 Atmel IO1 Xplained Pro Extension Board

Atmel® I/O1 Xplained Pro is an extension board to the Atmel Xplained Pro evaluation platform.

I/O1 Xplained Pro is designed to give a wide variety of functionality to Xplained Pro MCU boards including a microSD card, a temperature sensor, a light sensor, and more.



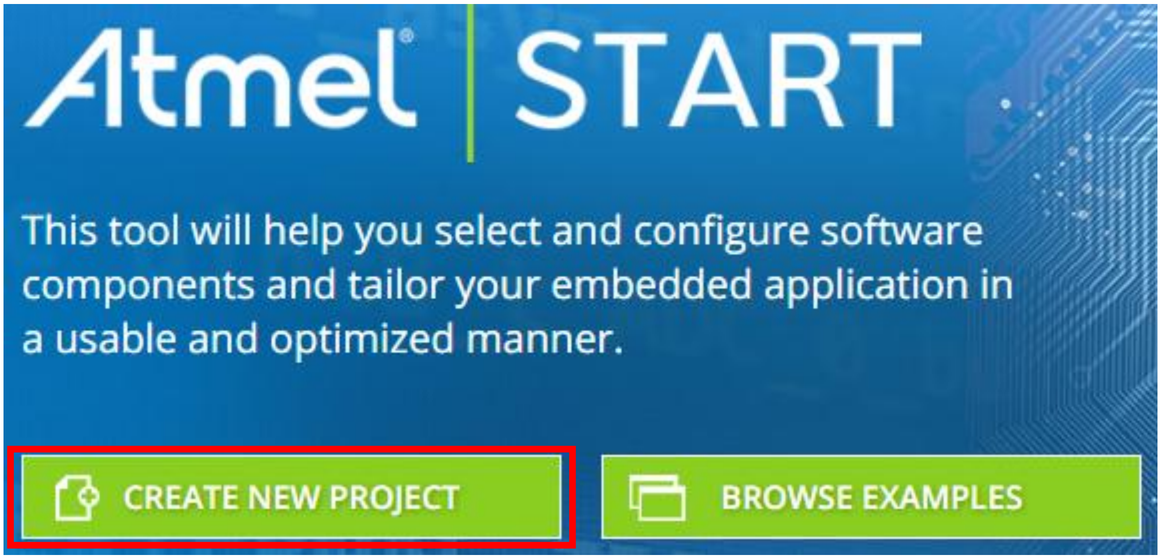
2. Assignment 1: Create and Configure a New Project using Atmel START

In this first assignment, we will create a new project and add the USART driver in order to display debug messages on a serial terminal.

2.1 Project Creation

TO DO Create New Project

- Open a browser and go to http://start.atmel.com
- Select CREATE NEW PROJECT:

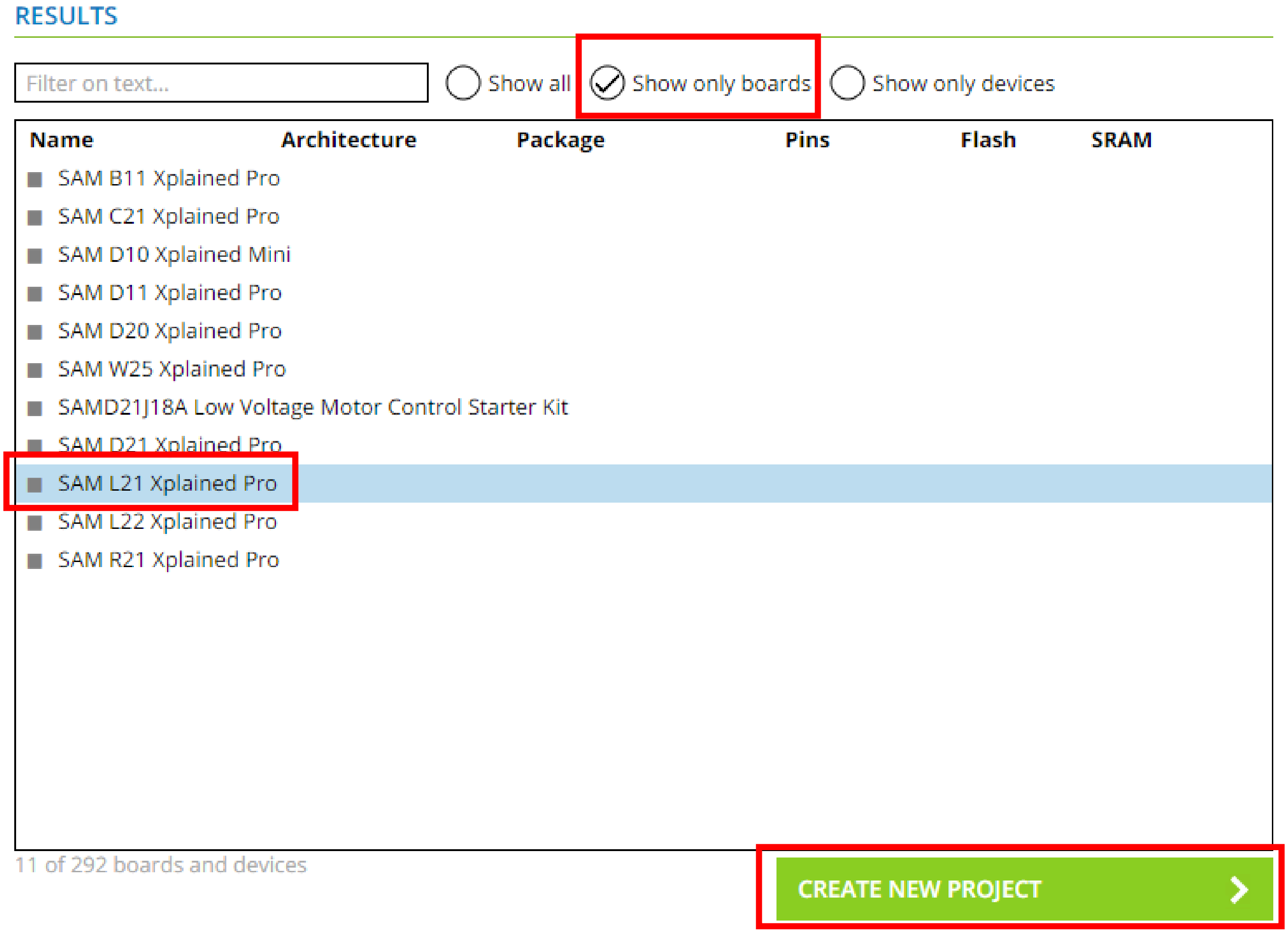


Atmel | START

This tool will help you select and configure software components and tailor your embedded application in a usable and optimized manner.

- Click on “Show only boards” from RESULTS section then select the SAM L21 Xplained Pro.

- Click on CREATE NEW PROJECT to complete the project creation:



RESULTS

Filter on text...   ○ Show all   ✓ Show only boards   ○ Show only devices

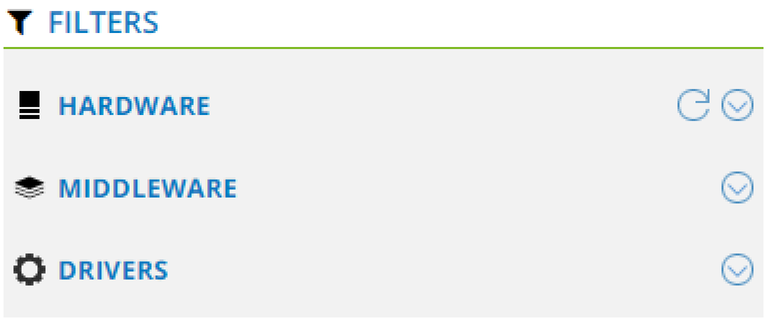
Name	Architecture	Package	Pins	Flash	SRAM
■ SAM B11 Xplained Pro					
■ SAM C21 Xplained Pro					
■ SAM D10 Xplained Mini					
■ SAM D11 Xplained Pro					
■ SAM D20 Xplained Pro					
■ SAM W25 Xplained Pro					
■ SAMD21J18A Low Voltage Motor Control Starter Kit					
■ SAM D21 Xplained Pro					
■ SAM L21 Xplained Pro					
■ SAM L22 Xplained Pro					
■ SAM R21 Xplained Pro					

11 of 292 boards and devices

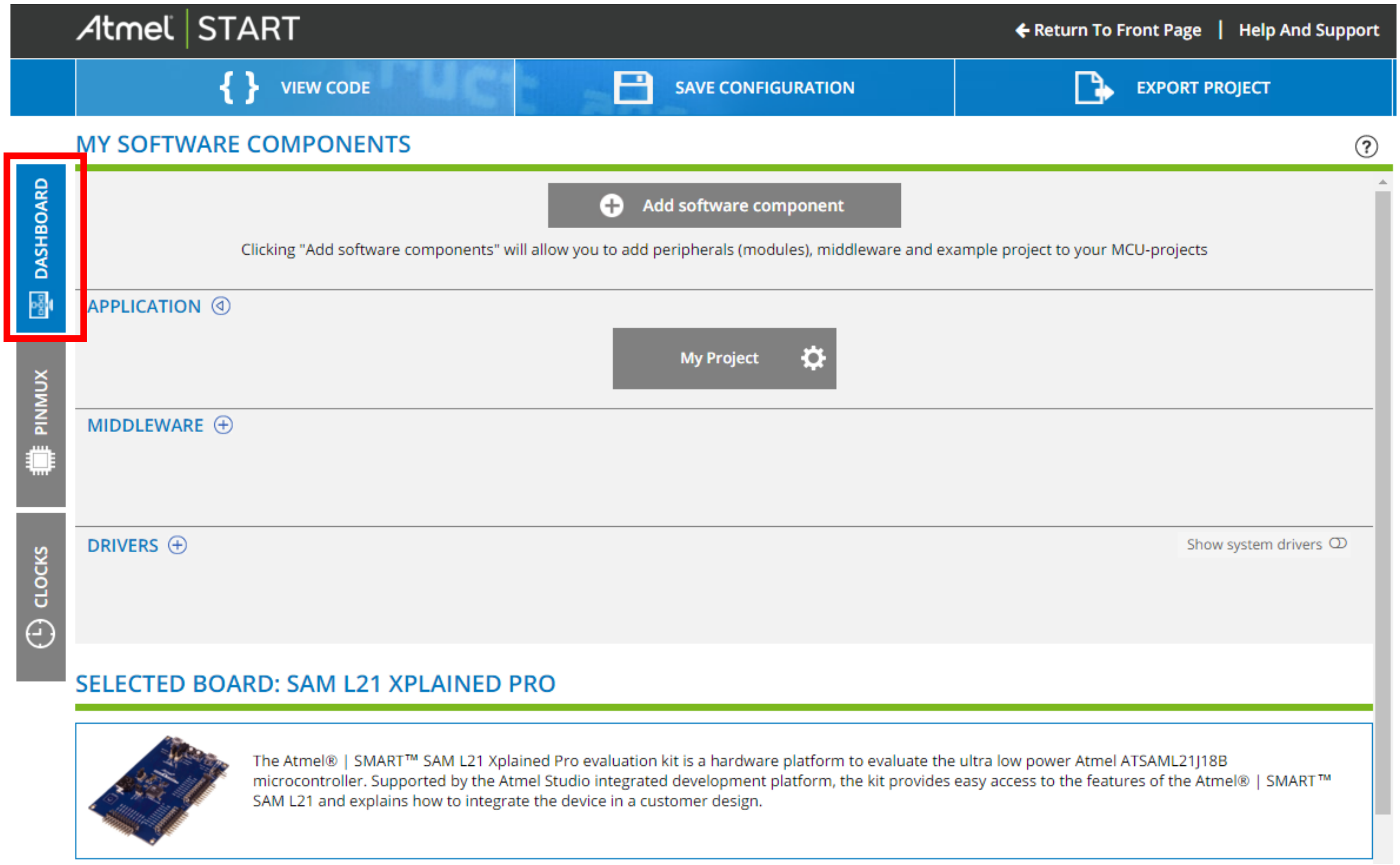
INFO

It is possible to add MIDDLEWARE as DRIVERS before creating the project by selecting them in the FILTERS section.

For this hands-on, we will add them later on.



RESULT The project is created in Atmel START and you have now access to the DASHBOARD view:



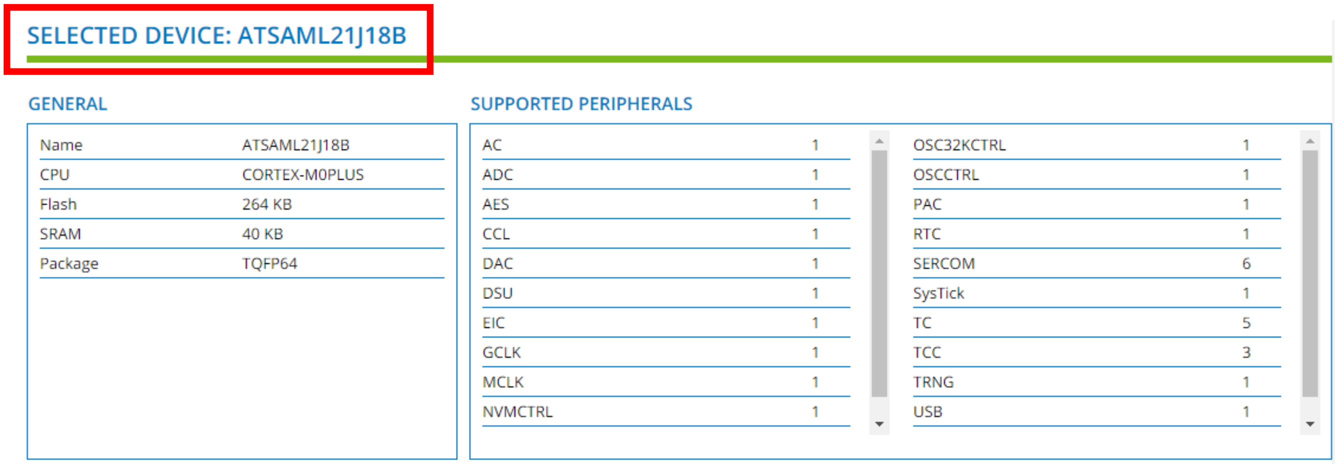
MY SOFTWARE COMPONENTS

Clicking "Add software components" will allow you to add peripherals (modules), middleware and example project to your MCU-projects

SELECTED BOARD: SAM L21 XPLAINED PRO

The Atmel® | SMART™ SAM L21 Xplained Pro evaluation kit is a hardware platform to evaluate the ultra low power Atmel ATSAML21J18B microcontroller. Supported by the Atmel Studio integrated development platform, the kit provides easy access to the features of the Atmel® | SMART™ SAM L21 and explains how to integrate the device in a customer design.

INFO You can check that choosing the SAM L21 Xplained PRO board automatically selects the ATSAML21J18B (TQFP64) as device which is the one mounted on it.



SELECTED DEVICE: ATSAML21J18B

GENERAL

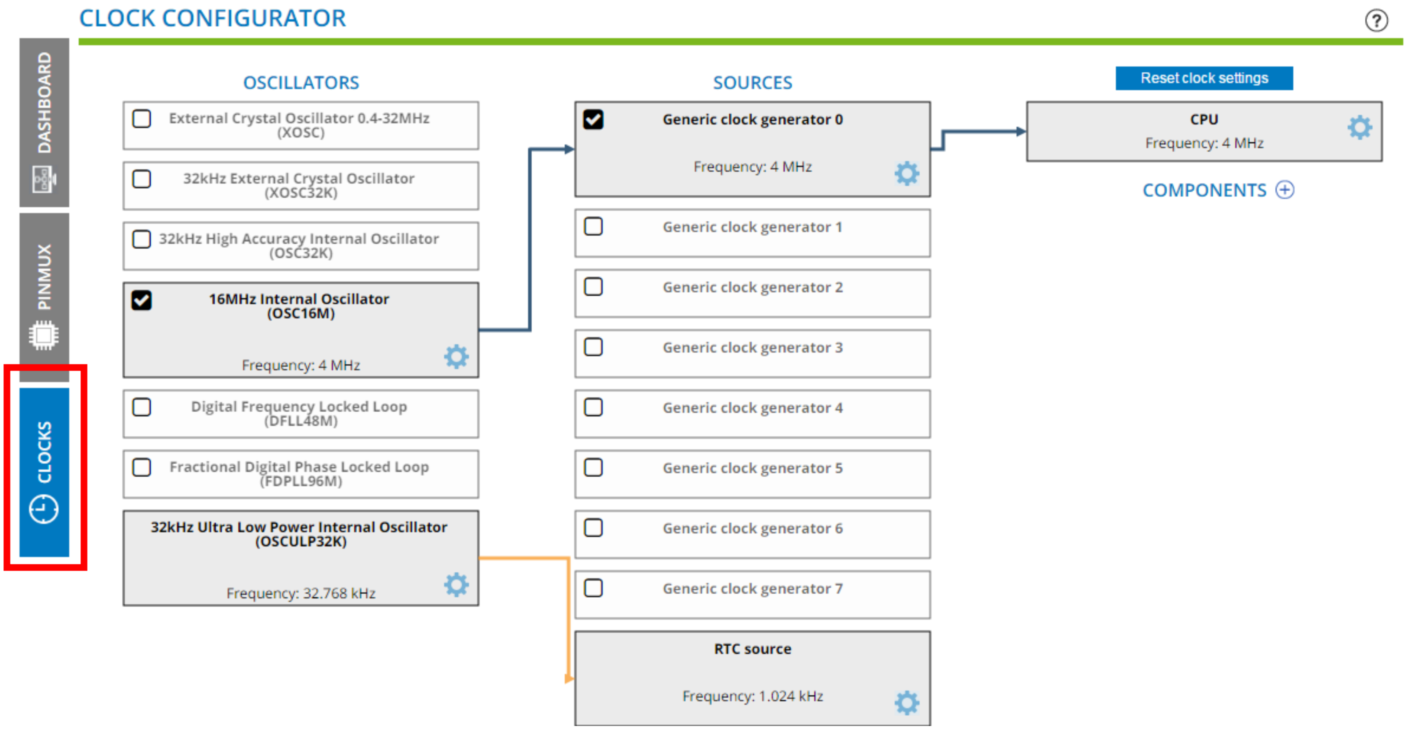
Name	ATSAML21J18B
CPU	CORTEX-M0PLUS
Flash	264 KB
SRAM	40 KB
Package	TQFP64

SUPPORTED PERIPHERALS

AC	1	OSC32KCTRL	1
ADC	1	OSCCTRL	1
AES	1	PAC	1
CCL	1	RTC	1
DAC	1	SERCOM	6
DSU	1	SysTick	1
EIC	1	TC	5
GCLK	1	TCC	3
MCLK	1	TRNG	1
NVMCTRL	1	USB	1

TO DO Review Oscillators and Clocks Configuration using Atmel START Clock Configurator

- Select CLOCKS to access the CLOCK CONFIGURATOR tool:


CLOCK CONFIGURATOR

OSCILLATORS

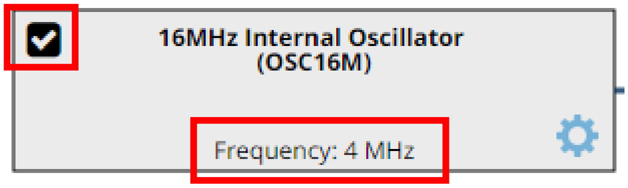
- □ External Crystal Oscillator 0.4-32MHz (XOSC)
- □ 32kHz External Crystal Oscillator (XOSC32K)
- □ 32kHz High Accuracy Internal Oscillator (OSC32K)
- ☑ 16MHz Internal Oscillator (OSC16M) Frequency: 4 MHz
- □ Digital Frequency Locked Loop (DFLL48M)
- □ Fractional Digital Phase Locked Loop (FDPLL96M)
- 32kHz Ultra Low Power Internal Oscillator (OSCULP32K) Frequency: 32.768 kHz

SOURCES

- ☑ Generic clock generator 0 Frequency: 4 MHz
- □ Generic clock generator 1
- □ Generic clock generator 2
- □ Generic clock generator 3
- □ Generic clock generator 4
- □ Generic clock generator 5
- □ Generic clock generator 6
- □ Generic clock generator 7
- RTC source Frequency: 1.024 kHz

CPU Frequency: 4 MHz

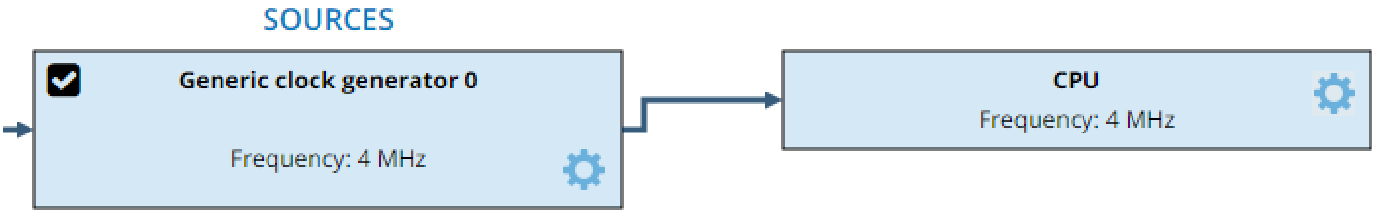
- Check that the internal OSC16M oscillator is enabled and is configured to run at 4MHz:


☑ 16MHz Internal Oscillator (OSC16M)

Frequency: 4 MHz

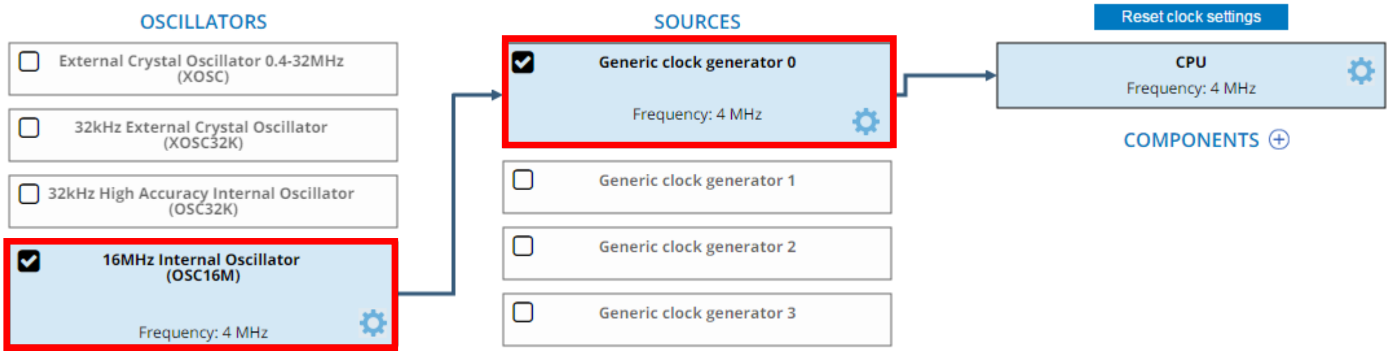
INFO 4MHz is the default SAM L21 operating frequency at power up.

- Click on the CPU block to check that its clock source comes from the Generic Clock Generator 0:



INFO The Generic Clock Generator 0 is always the direct source of the CPU Clock.

- Check then that the oscillator source of the Generic Clock Generator 0 is the internal OSC16M:


OSCILLATORS
☐ External Crystal Oscillator 0.4-32MHz (XOSC)
☐ 32kHz External Crystal Oscillator (XOSC32K)
☐ 32kHz High Accuracy Internal Oscillator (OSC32K)
☑ 16MHz Internal Oscillator (OSC16M)
Frequency: 4 MHz
SOURCES
☑ Generic clock generator 0
Frequency: 4 MHz
☐ Generic clock generator 1
☐ Generic clock generator 2
☐ Generic clock generator 3
Reset clock settings
CPU
Frequency: 4 MHz
COMPONENTS ⊕

INFO After reset, Generic Clock Generator 0 uses the internal OSC16M as default source.

RESULT We have reviewed the default oscillators and clocks project configuration.

2.2 Add USART Driver using Atmel Start

We will use the Virtual COM Port of the SAM L21 Xplained Pro Embedded Debugger (EDBG) as USART communication channel.

INFO The EDBG is a composite USB device with three interfaces; a debugger, Virtual COM Port, and a Data Gateway Interface (DGI) which handles events and data.

The Virtual COM Port is connected to a UART on the ATSAML21J18B and provides an easy way to communicate with the target application through terminal software. It offers variable baud rate, parity, and stop bit settings. Note that the settings on the ATSAML21J18B must match the settings given in the terminal software.

TO DO Get SAM L21 Virtual COM Port Connections

Such info is found in the SAM L21 Xplained Pro User Guide:

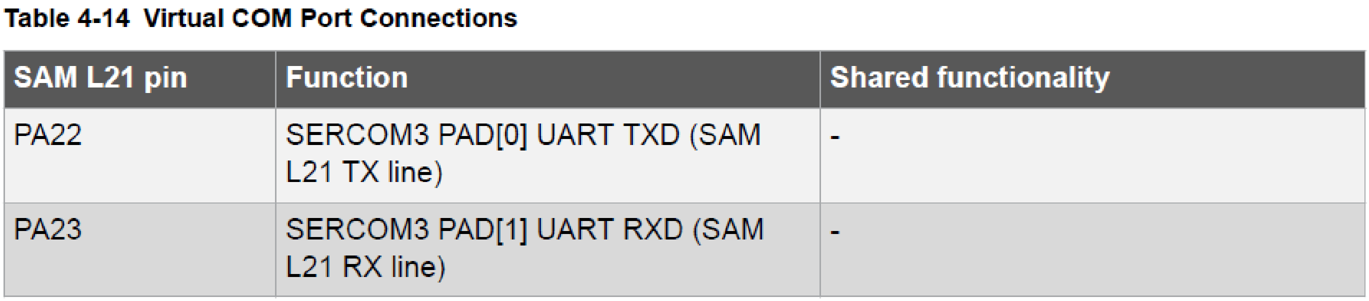


Table 4-14 Virtual COM Port Connections

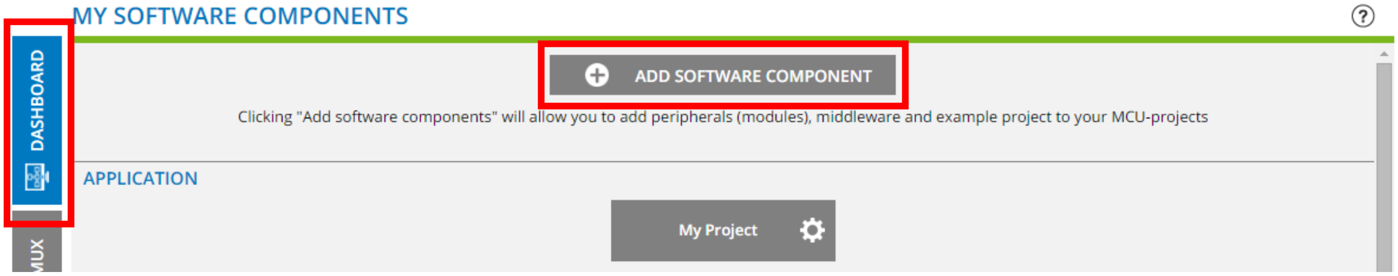
SAM L21 pin	Function	Shared functionality
PA22	SERCOM3 PAD[0] UART TXD (SAM L21 TX line)	-
PA23	SERCOM3 PAD[1] UART RXD (SAM L21 RX line)	-

INFO SAM L21 Xplained Pro B User Guide can be found on at this address: http://www.atmel.com/tools/ATSAML21-XPRO-B.aspx

RESULT SAM L21 UART I/Os belong to SERCOM3 Peripheral.

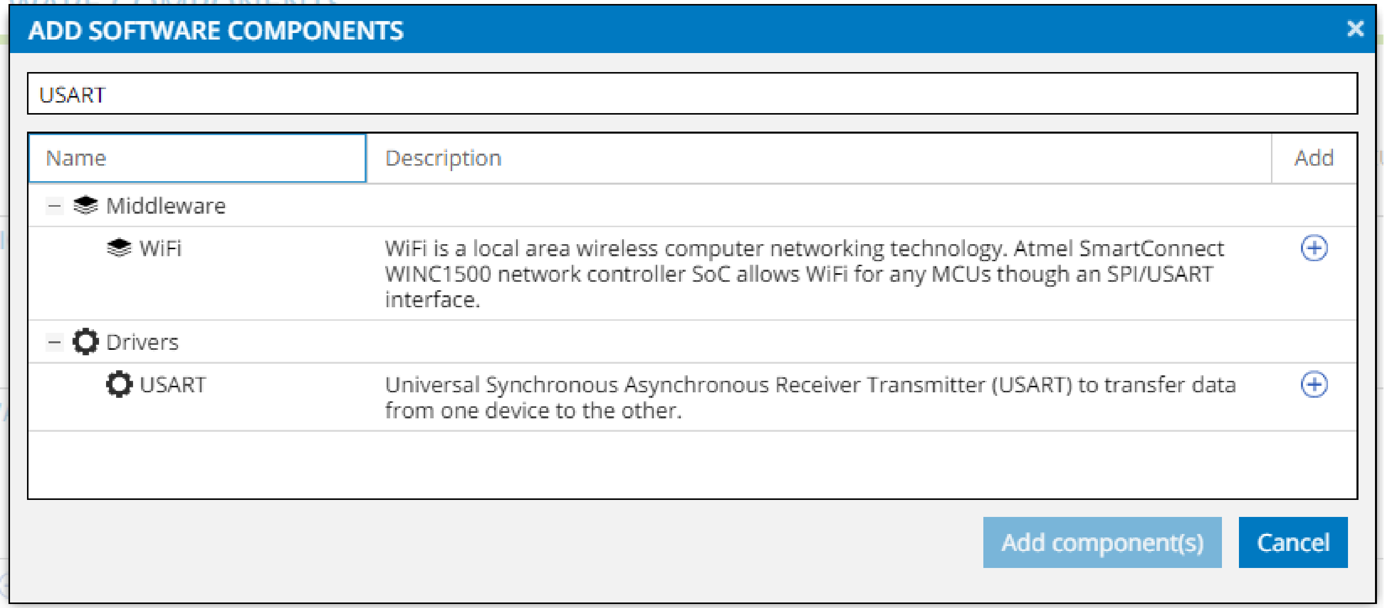
TO DO Add the USART driver

- In Atmel START, select DASHBOARD and click on “ADD SOFTWARE COMPONENT”:



Clicking "Add software components" will allow you to add peripherals (modules), middleware and example project to your MCU-projects

- Type USART in the Filter field, look for the USART driver and add it ⊕:



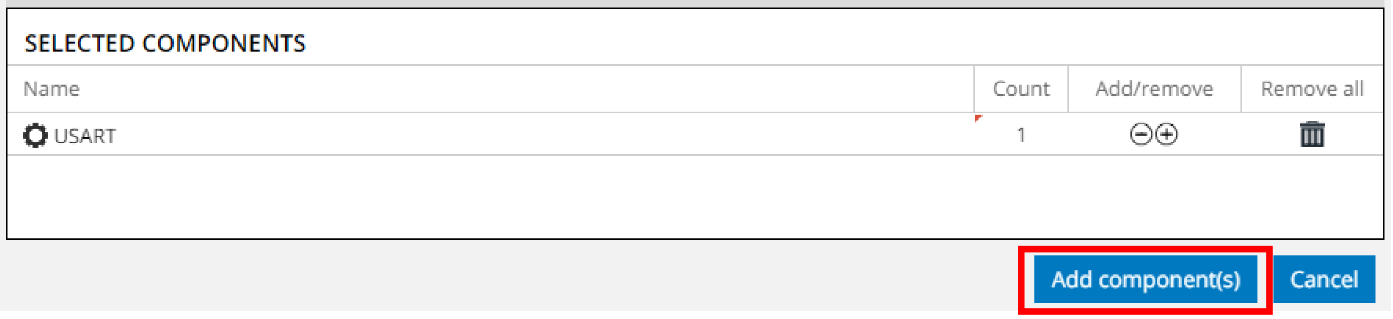
USART

Name	Description	Add
− Middleware		
WiFi	WiFi is a local area wireless computer networking technology. Atmel SmartConnect WINC1500 network controller SoC allows WiFi for any MCUs though an SPI/USART interface.	⊕
− Drivers		
USART	Universal Synchronous Asynchronous Receiver Transmitter (USART) to transfer data from one device to the other.	⊕

INFO You can also directly look for it in the Drivers list.

You will have it displayed in the SELECTED COMPONENTS view:

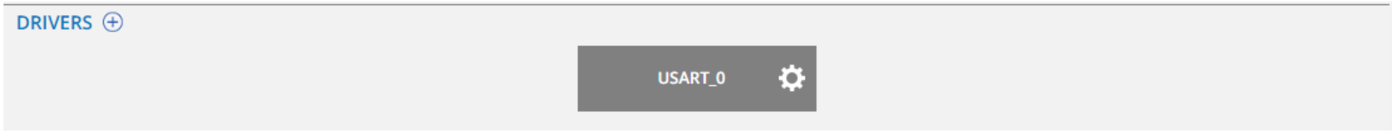
- You can now complete the addition of the USART driver by clicking on Add component(s):



SELECTED COMPONENTS

Name	Count	Add/remove	Remove all
USART	1	⊖⊕	🗑

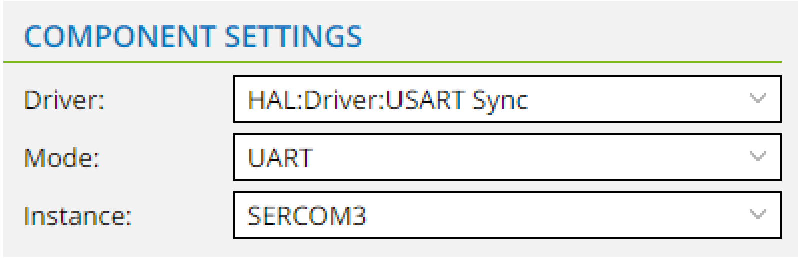
☑ **RESULT** The USART driver is added to the application.



TO DO Configure the USART driver

- Click on USART_0 component block to start its configuration

- Configure USART Component Settings:
  - Driver: USART Sync
  - Mode: UART
  - Instance: SERCOM3

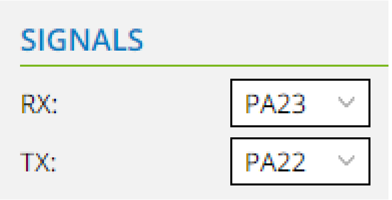


COMPONENT SETTINGS

Driver:	HAL:Driver:USART Sync
Mode:	UART
Instance:	SERCOM3

INFO The USART Sync corresponds to a polling driver implementation contrary to the USART Async which relates to an interrupt driver one.

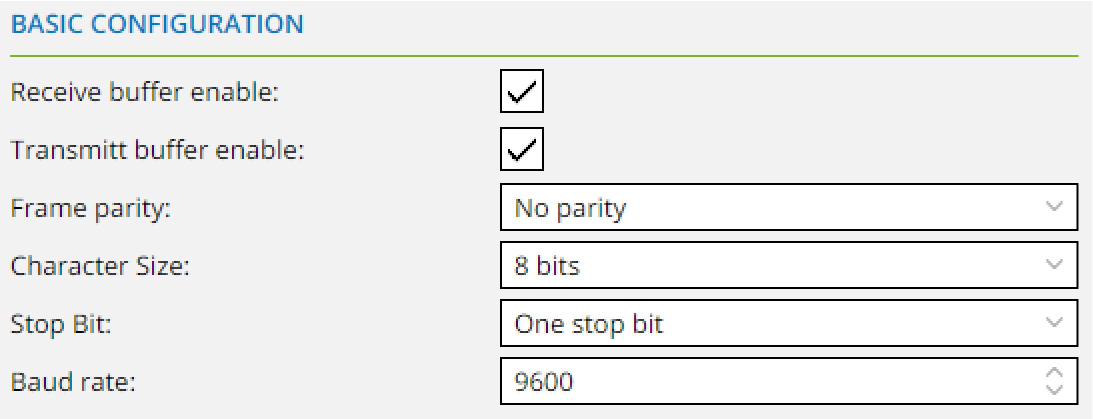
- Configure USART Signals:
  - RX: PA23
  - TX: PA22



SIGNALS

RX:	PA23
TX:	PA22

- Check Virtual COM Port Basic Configuration: 9600 bauds / No parity / 1 Stop Bit



BASIC CONFIGURATION

Receive buffer enable:	☑
Transmitt buffer enable:	☑
Frame parity:	No parity
Character Size:	8 bits
Stop Bit:	One stop bit
Baud rate:	9600

RESULT The USART driver is added and configured.

2.3 Save and Export the Application on an Atmel Studio 7 project

We have now created and configured our Atmel Start based project.

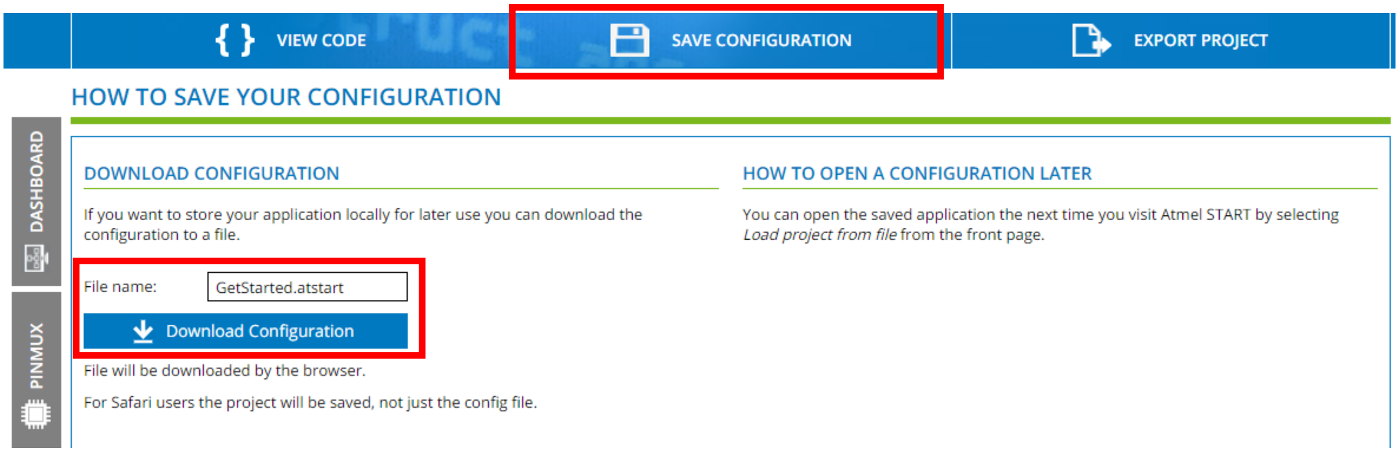
It’s time now to export it as a project for Atmel Studio 7.

But before that, it’s preferable to save the different configurations we did in case we need to come back later on and make some updates.

Indeed, Atmel Start allows restoring any created project using its configuration file (*.atstart file format).

TO DO Save Project Configuration

- Select SAVE CONFIGURATION, give a File Name then Click on DOWNLOAD CONFIGURATION



HOW TO SAVE YOUR CONFIGURATION

DOWNLOAD CONFIGURATION

If you want to store your application locally for later use you can download the configuration to a file.

File name: GetStarted.atstart

Download Configuration

File will be downloaded by the browser.

For Safari users the project will be saved, not just the config file.

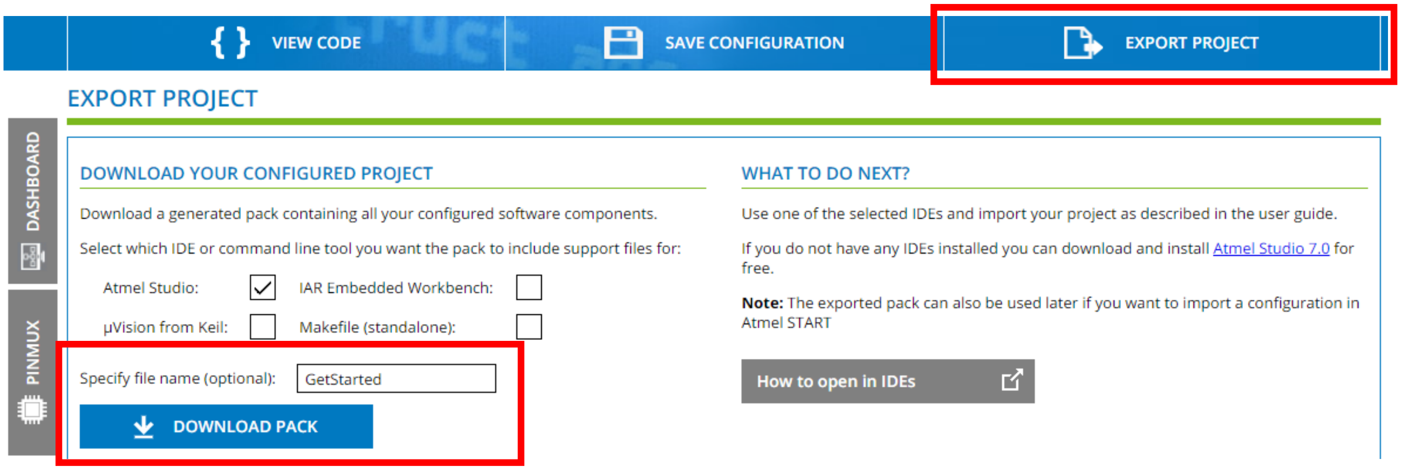
HOW TO OPEN A CONFIGURATION LATER

You can open the saved application the next time you visit Atmel START by selecting *Load project from file* from the front page.

RESULT Your application configuration has been saved in *.atstart file format

TO DO Export Project

- Select EXPORT PROJECT, give a File Name then Click on DOWNLOAD PACK



EXPORT PROJECT

DOWNLOAD YOUR CONFIGURED PROJECT

Download a generated pack containing all your configured software components.

Select which IDE or command line tool you want the pack to include support files for:

Atmel Studio: ☑ IAR Embedded Workbench: □

µVision from Keil: □ Makefile (standalone): □

Specify file name (optional): GetStarted

DOWNLOAD PACK

WHAT TO DO NEXT?

Use one of the selected IDEs and import your project as described in the user guide.

If you do not have any IDEs installed you can download and install Atmel Studio 7.0 for free.

Note: The exported pack can also be used later if you want to import a configuration in Atmel START

How to open in IDEs

RESULT Your application project has been exported for Atmel Studio n a *.atzip file format (standard zip format automatically recognized by Atmel Studio 7)

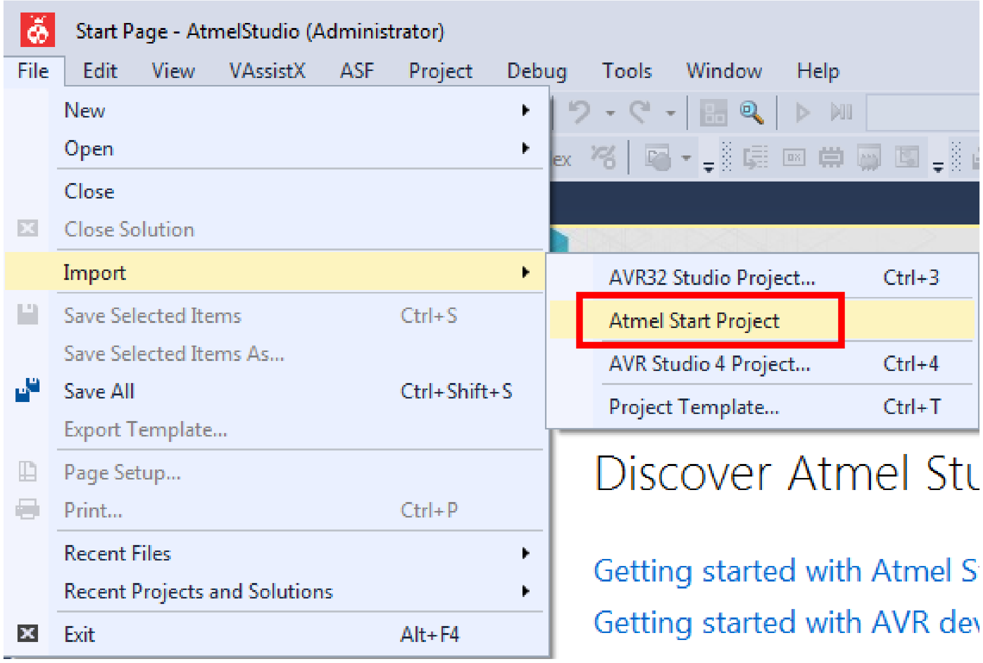
2.4 USART Implementation

We have now finished to create and configure our Atmel Start based project.

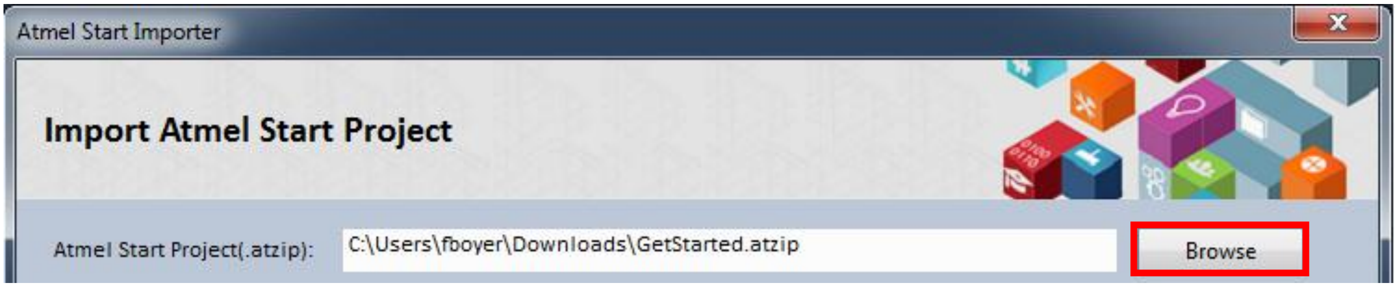
It's time now to export it as a project for Atmel Studio 7.

TO DO Import Atmel Start Project

- Open Atmel Studio 7
- Select File > Import > Atmel Start Project:


Start Page - AtmelStudio (Administrator)
File Edit View VAssistX ASF Project Debug Tools Window Help
New
Open
Close
Close Solution
Import
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Page Setup...
Print... Ctrl+P
Recent Files
Recent Projects and Solutions
Exit Alt+F4
AVR32 Studio Project... Ctrl+3
Atmel Start Project
AVR Studio 4 Project... Ctrl+4
Project Template... Ctrl+T
Discover Atmel Stu
Getting started with Atmel S
Getting started with AVR dev

- Select the project (*.atzip) you have exported from Atmel Start and click OK:

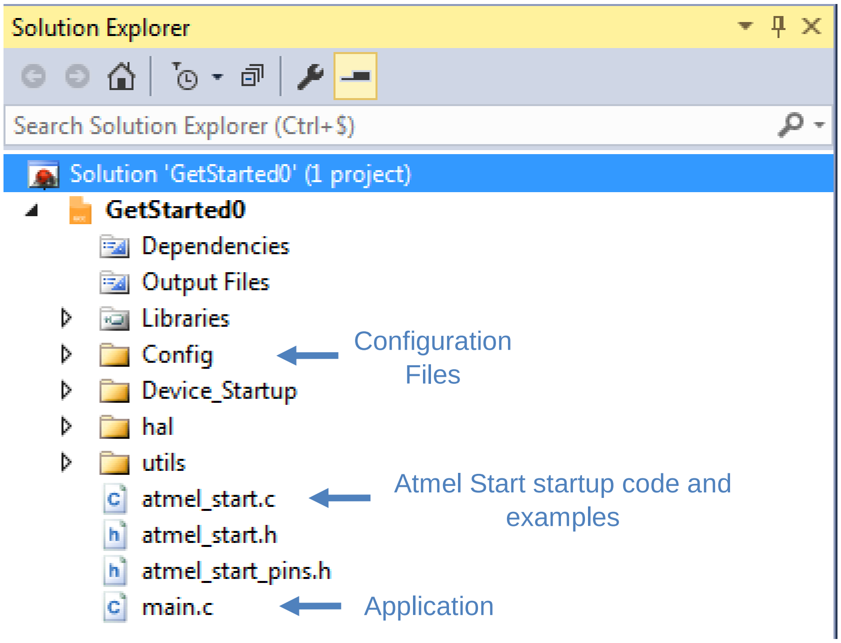


Atmel Start Project(.atzip): C:\Users\fboyer\Downloads\GetStarted.atzip

RESULT The project is created.

TO DO Review Reference Project

Any Atmel Start-based project adds some useful examples to get started on the different peripherals' initialization: `atmel_start.c` is the file which includes all of them.



Configuration Files

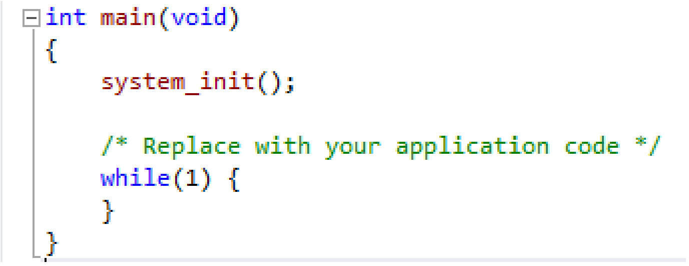
Atmel Start startup code and examples

Application

INFO In our application, we will only find USART related functions as this is the only peripheral we have selected in Atmel Start.

So, when we will develop the application in the `main.c` file, we will call and adapt (if required), some of the functions provided in `atmel_start.c` to build very quickly our project.

- Open `main.c` file. You will see that the only function called in the current project is the `system_init()` function:



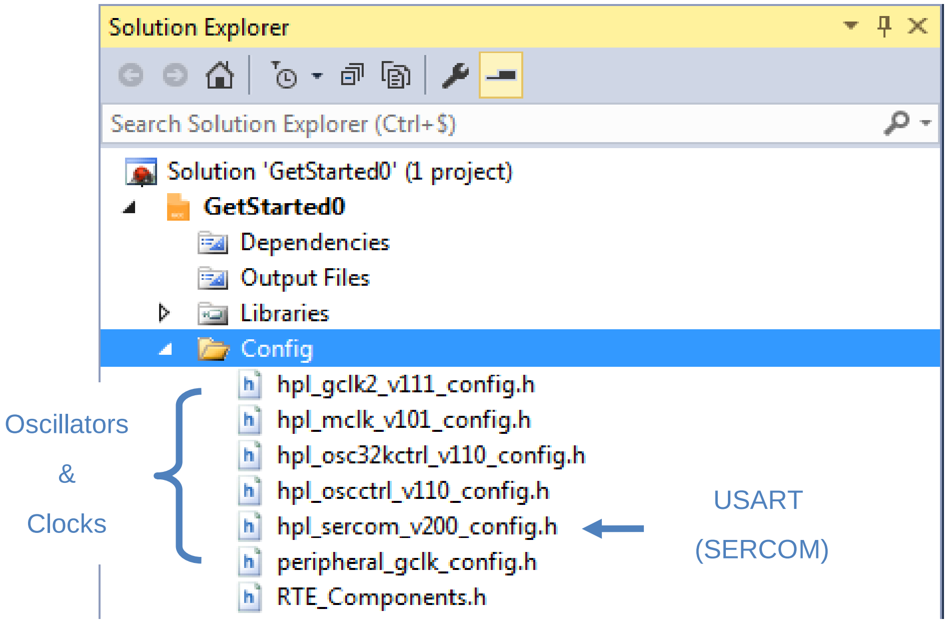
```
int main(void)
{
    system_init();

    /* Replace with your application code */
    while(1) {
    }
}
```

`system_init()` function is implemented in `atmel_start.c` and:

- Initializes the MCU (oscillators, clocks, flash wait states…)
  - Using `init_mcu()` function.
- Initializes the peripherals which have been selected:
  - The USART in our case using USART_0_init() function.

```
void system_init(void)
{
        init_mcu();

        USART_0_init();
}
```

The different initialization functions which are called in `system_init()`, use the configuration's parameters that the user has selected during the Atmel Start configuration process.



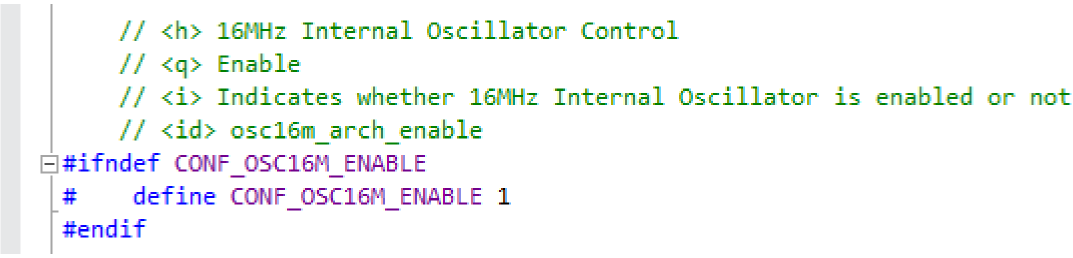
Solution Explorer

Search Solution Explorer (Ctrl+$)

- Solution 'GetStarted0' (1 project)
  - GetStarted0
    - Dependencies
    - Output Files
    - Libraries
    - Config
      - hpl_gclk2_v111_config.h
      - hpl_mclk_v101_config.h
      - hpl_osc32kctrl_v110_config.h
      - hpl_oscctrl_v110_config.h
      - hpl_sercom_v200_config.h
      - peripheral_gclk_config.h
      - RTE_Components.h

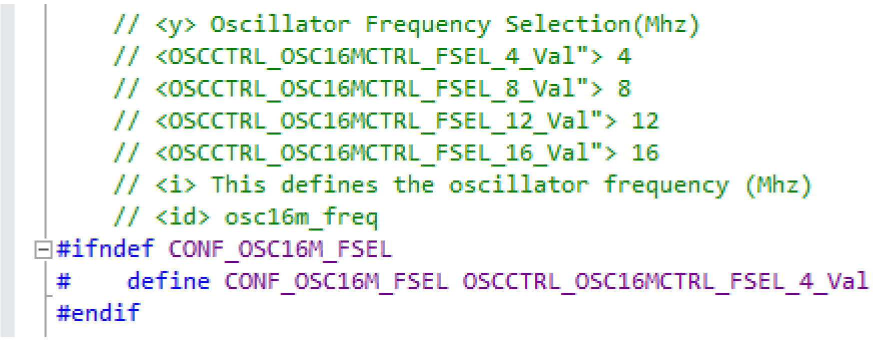
INFO You can retrieve these configurations in the `Config` folder.

As an example, if you open the `hpl_oscctrl_vxxx_config.h` file, you can check that the only oscillator enabled is the OSC16M:



```
    // <h> 16MHz Internal Oscillator Control
    // <q> Enable
    // <i> Indicates whether 16MHz Internal Oscillator is enabled or not
    // <id> osc16m_arch_enable
#ifndef CONF_OSC16M_ENABLE
#    define CONF_OSC16M_ENABLE 1
#endif
```

And that the selected frequency for the OSC16M oscillator is 4MHz:



```
    // <y> Oscillator Frequency Selection(Mhz)
    // <OSCCTRL_OSC16MCTRL_FSEL_4_Val"> 4
    // <OSCCTRL_OSC16MCTRL_FSEL_8_Val"> 8
    // <OSCCTRL_OSC16MCTRL_FSEL_12_Val"> 12
    // <OSCCTRL_OSC16MCTRL_FSEL_16_Val"> 16
    // <i> This defines the oscillator frequency (Mhz)
    // <id> osc16m_freq
#ifndef CONF_OSC16M_FSEL
#    define CONF_OSC16M_FSEL OSCCTRL_OSC16MCTRL_FSEL_4_Val
#endif
```

✓ **RESULT** The main components of the project have been reviewed.

TO DO Implement USART to send Debug Messages

As mentioned above, the examples in `atmel_start.c` file will be used to help us get started.

- Open `atmel_start.c` and copy `USART_0_example` function
- Paste it above `main()` function from `main.c` file
- Rename it as `UART_EDBG_init`

```
void UART_EDBG_init(void)
{
        struct io_descriptor *io;
        usart_sync_get_io_descriptor(&USART_0, &io);
        usart_sync_enable(&USART_0);

        io_write(io, (uint8_t *)"Hello World!", 12);
}

int main(void)
{
        system_init();

        while(1) {
        }
}
```

We will use `io_write()` function to send debug messages to the serial terminal.

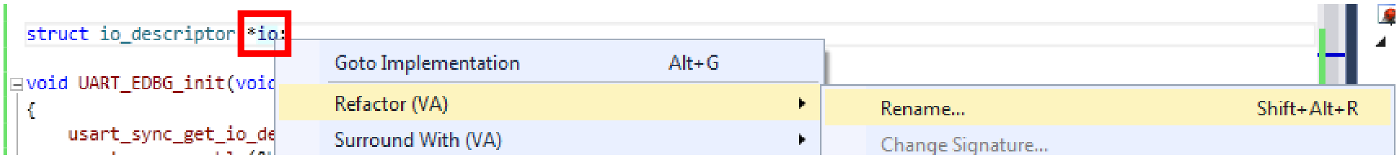
That function, such as `io_read()`, relies on a descriptor called `io_descriptor.`

So, we need to define that descriptor as global in order to be able to use it outside this example function:

- Move/Cut the `struct io_descriptor *io;` line outside of the function

```
struct io_descriptor *io;

void UART_EDBG_init(void)
{
        usart_sync_get_io_descriptor(&USART_0, &io);
        usart_sync_enable(&USART_0);

        io_write(io, (uint8_t *)"Hello World!", 12);
}
```

- Right click on “io” and select Refactor (VA) -> Rename... which allows to rename all found references of a searched instance

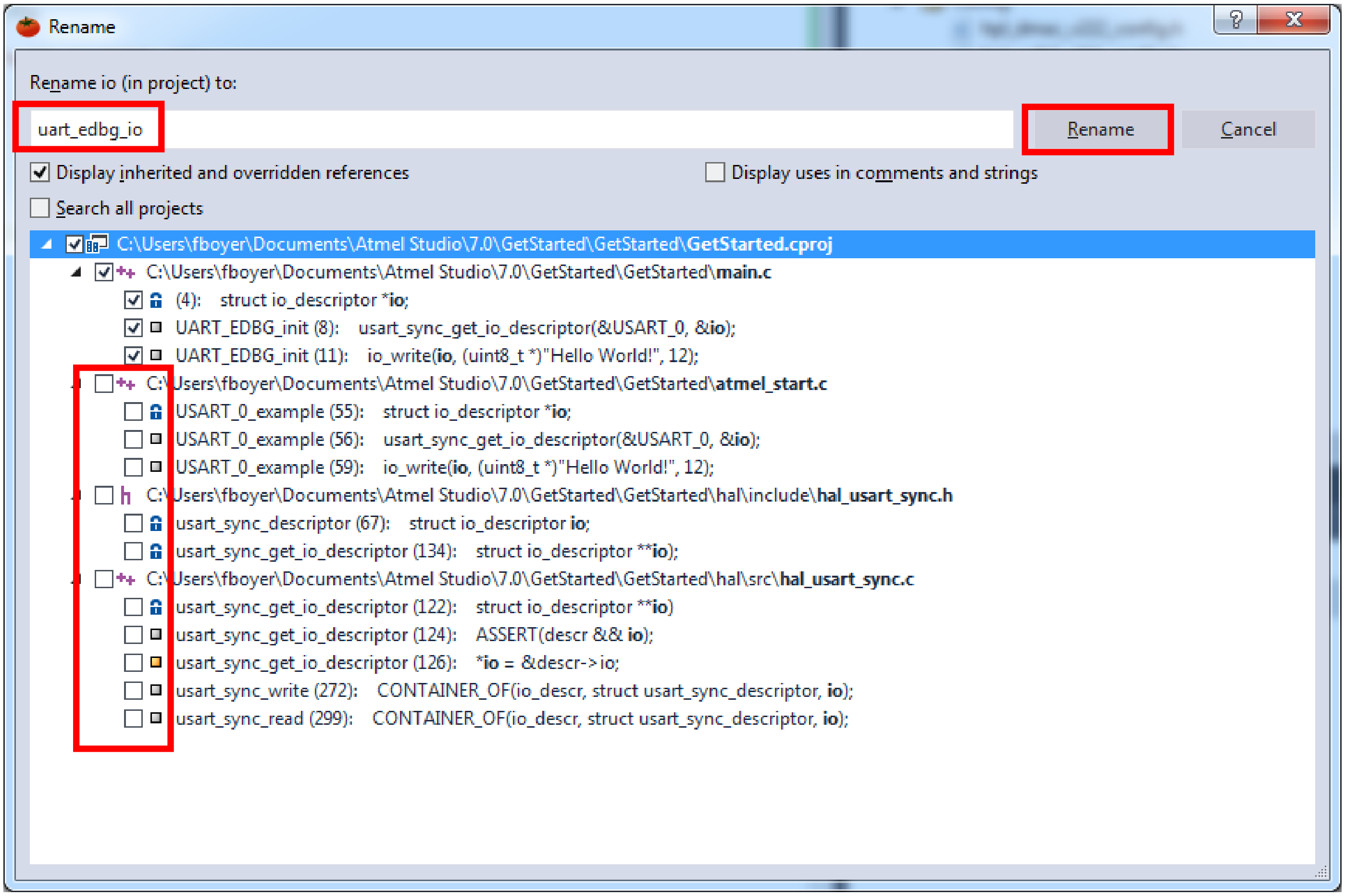


```
struct io_descriptor *io;

void UART_EDBG_init(void
{
    usart_sync_get_io_de
```

- Deselect ALL the lines which do not correspond to main.c file

- Rename it from “io” to “uart_edbg_io”:

WARNING Make sure that only the instances in main.c are renamed or you may modify the usart driver (hal_usart_sync.c) itself or other components.



Rename

Rename io (in project) to:

uart_edbg_io

☑ Display inherited and overridden references ☐ Display uses in comments and strings

☐ Search all projects

- ☑ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\GetStarted.cproj
  - ☑ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\main.c
    - ☑ (4): struct io_descriptor ***io**;
    - ☑ UART_EDBG_init (8): usart_sync_get_io_descriptor(&USART_0, &**io**);
    - ☑ UART_EDBG_init (11): io_write(**io**, (uint8_t *)"Hello World!", 12);
  - ☐ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\atmel_start.c
    - ☐ USART_0_example (55): struct io_descriptor ***io**;
    - ☐ USART_0_example (56): usart_sync_get_io_descriptor(&USART_0, &**io**);
    - ☐ USART_0_example (59): io_write(**io**, (uint8_t *)"Hello World!", 12);
  - ☐ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\hal\include\hal_usart_sync.h
    - ☐ usart_sync_descriptor (67): struct io_descriptor **io**;
    - ☐ usart_sync_get_io_descriptor (134): struct io_descriptor ****io**);
  - ☐ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\hal\src\hal_usart_sync.c
    - ☐ usart_sync_get_io_descriptor (122): struct io_descriptor ****io**)
    - ☐ usart_sync_get_io_descriptor (124): ASSERT(descr && **io**);
    - ☐ usart_sync_get_io_descriptor (126): ***io** = &descr->io;
    - ☐ usart_sync_write (272): CONTAINER_OF(io_descr, struct usart_sync_descriptor, **io**);
    - ☐ usart_sync_read (299): CONTAINER_OF(io_descr, struct usart_sync_descriptor, **io**);

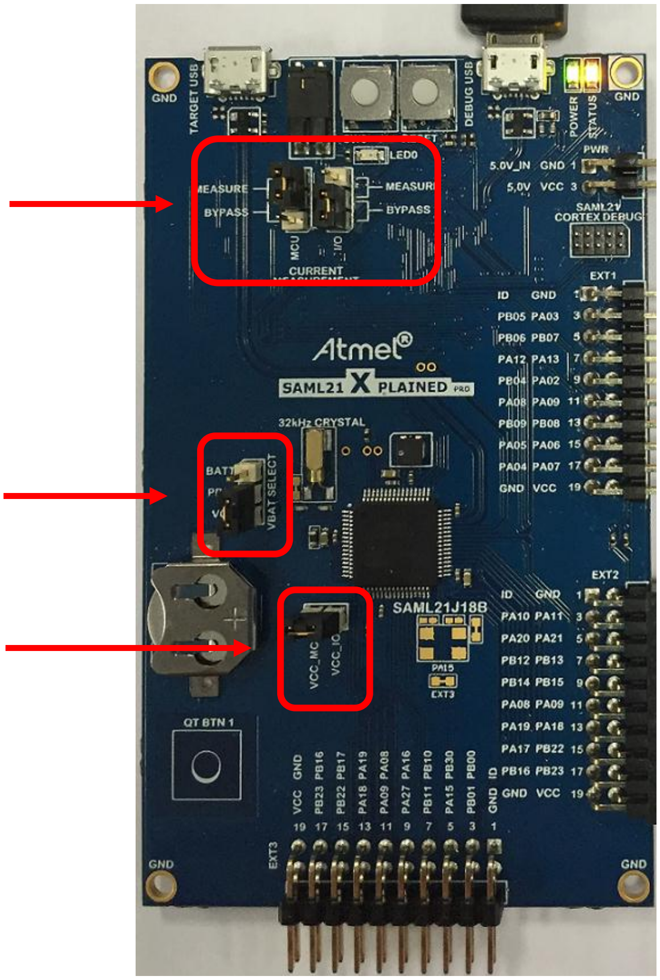
- Call UART_EDBG_init(); after system_init();

RESULT USART Implementation is completed:

```
struct io_descriptor *uart_edbg_io;

void UART_EDBG_init(void)
{
	usart_sync_get_io_descriptor(&USART_0, &uart_edbg_io);
	usart_sync_enable(&USART_0);

	io_write(uart_edbg_io, (uint8_t *)"Hello World!", 12);
}

int main(void)
{
	system_init();

	UART_EDBG_init();

	while(1) {
	}
}
```

TO DO Hardware Setup

- Check the board’s jumpers are correctly set:



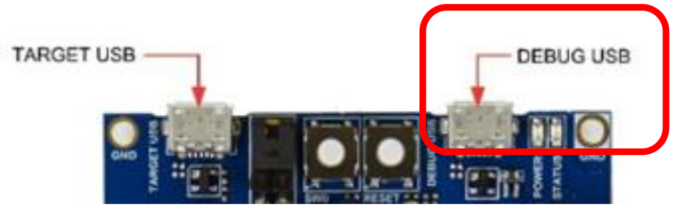
MCU to MEASURE

I/O to BYPASS

VBAT SELECT to PB03-VCC

VCC_MCU VCC_IO set

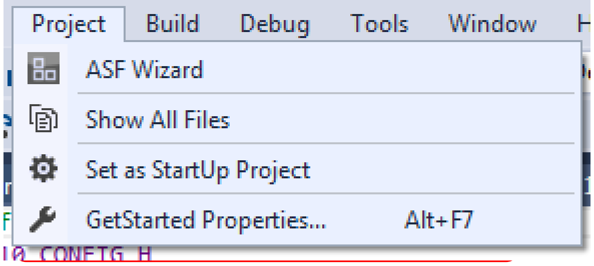
- Power-up your SAM L21 Xplained Pro B using DEBUG USB Connector:



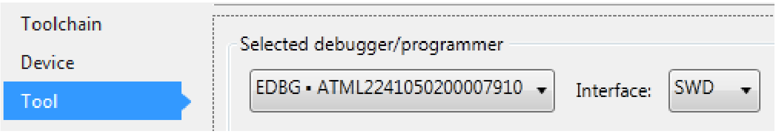
RESULT Hardware Setup is completed.

TO DO Compile and Program the Project

- Select SAM L21 XPRO Debugger/Programmer:
  - Click on Project > Properties



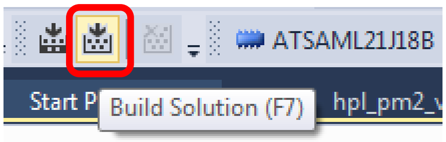
  - Select Tool > EDBG as debugger/programmer and SWD as Interface:



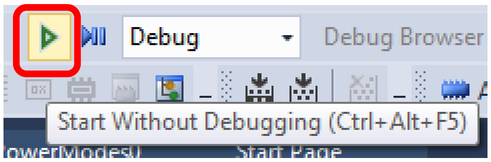
Selected debugger/programmer: EDBG • ATML2241050200007910

Interface: SWD

- Compile the project by clicking on the Build Solution icon or by typing ‘F7’.



- Program the application by clicking on the Start Without Debugging icon



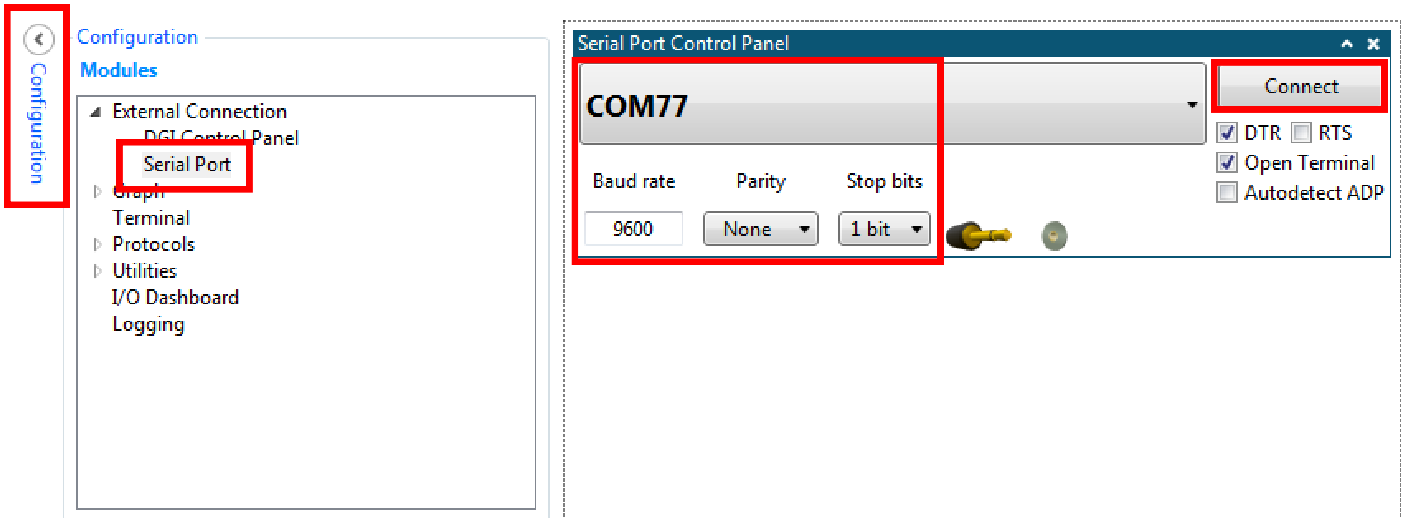
WARNING You may be asked to update the Embedded Debugger Firmware of the board (EDBG) if this was not initially done.

In that case, please have a look at section 8 Appendix: Upgrade Embedded Debugger (EDBG) Firmware to get the procedure.

RESULT The application is programmed and runs out of the target.

TO DO Test USART Implementation

- Open Data Visualizer by clicking on Tools > Data Visualizer.
- Click on Configuration > External Connection > Serial Port to open Serial Port Control Panel.



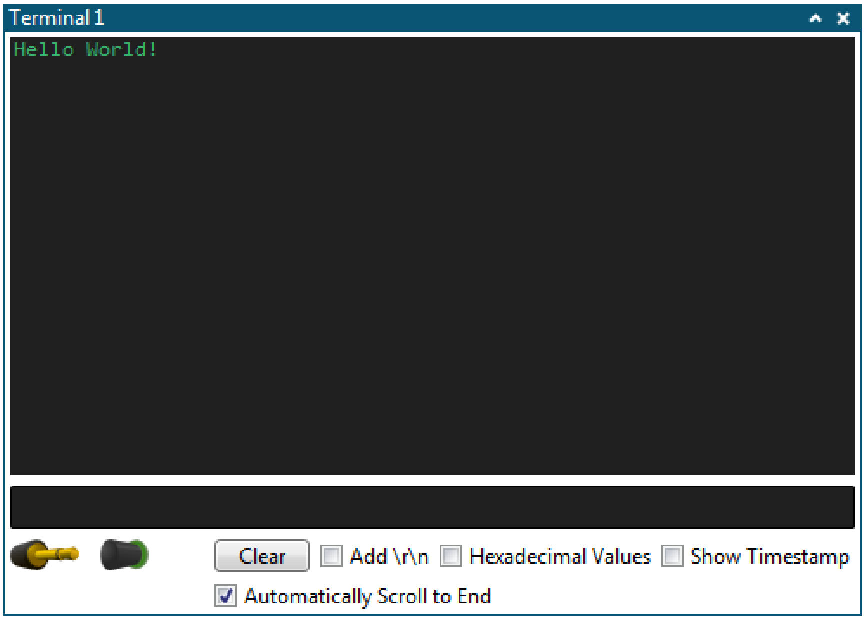
COM77

Baud rate: 9600
Parity: None
Stop bits: 1 bit

☑ DTR □ RTS
☑ Open Terminal
□ Autodetect ADP

- Choose right Virtual COM Port (you can find it using Windows OS Device Manager)
- Set Baud rate to 9600, Parity to None and Stop bits to 1 and Click on Connect:
- Press RESET button of the SAM L21 Xplained Pro

RESULT The debug message is displayed on the Serial Terminal:



```
Hello World!
```

□ Add \r\n □ Hexadecimal Values □ Show Timestamp
☑ Automatically Scroll to End

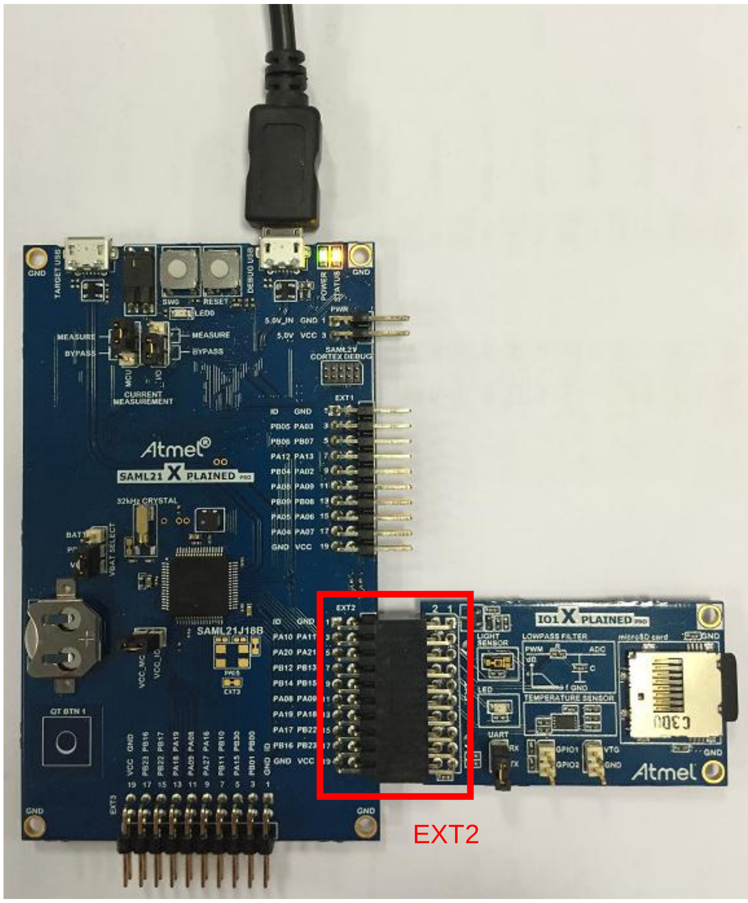
3. Assignment 2: Application Implementation

In this second assignment, we will add to the existing project the ADC and I2C drivers using Atmel Start then implement the different functions to:

- Get samples every second from the light sensor (ADC peripheral).
- Get samples the from temperature sensor (SERCOM I2C peripheral).

INFO Both sensors are embedded on the Atmel IO1 Xplained Pro extension board.

So, we will connect the IO1 Xplained Pro extension board on the SAM L21 Xplained Pro EXT2 Connector

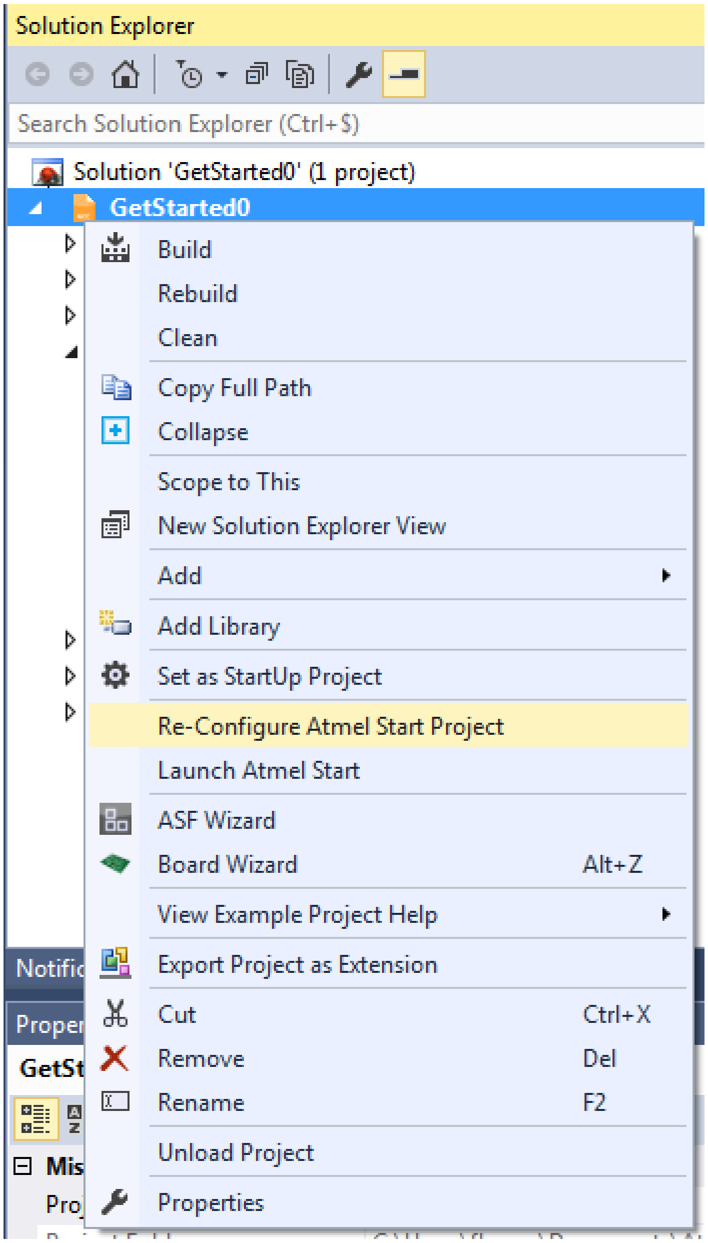


3.1 Recover Atmel Start Project

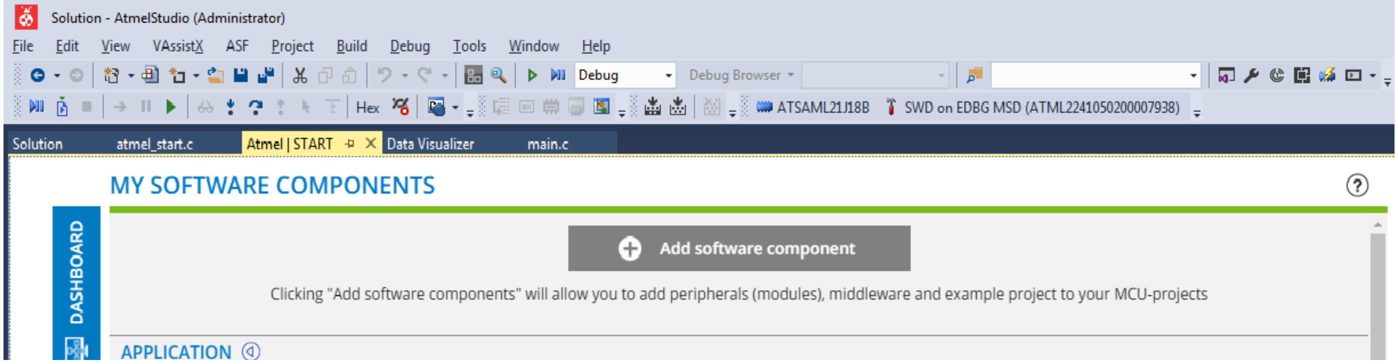
Atmel Studio 7 allows to simply update an existing project by adding new drivers from Atmel Start.

TO DO Re-Configure Atmel Start Project

- Right click on the Project and Select Re-Configure Atmel Start Project:



- The Atmel START Web page will open directly in Atmel STUDIO:



Clicking "Add software components" will allow you to add peripherals (modules), middleware and example project to your MCU-projects

✓ **RESULT** It is now possible to add new drivers to the project using Atmel START.

3.2 Add ADC Driver using Atmel Start

I/O1 Xplained Pro features a TEMT6000 light sensor from Vishay.

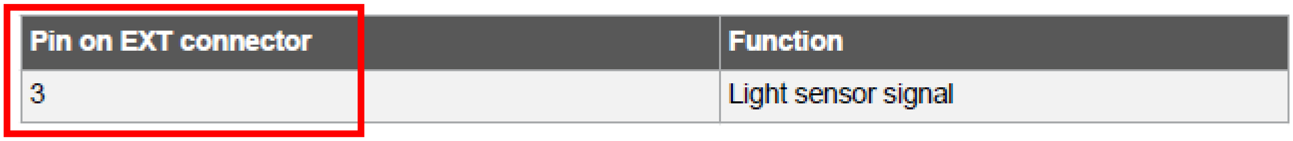
The sensor data can be read by an ADC pin on any Xplained Pro MCU board.

Let’s determine which pin from the SAM L21 Xplained Pro interfaces with it.

TO DO Determine which SAM L21 I/O interfaces with the ADC Light Sensor

- Get ADC Light Sensor pin from **Atmel IO1 Extension Board User Guide:**

Table 4-10 Light Sensor Connections

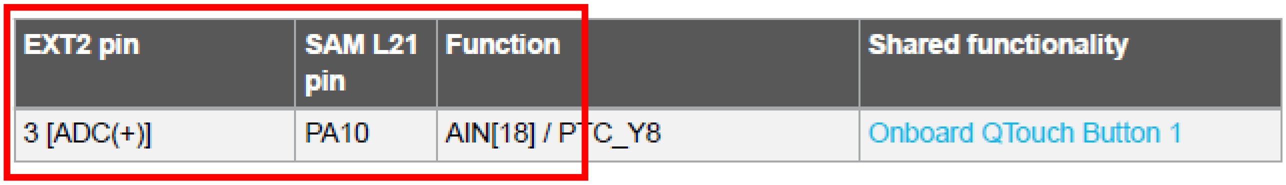


Pin on EXT connector	Function
3	Light sensor signal

INFO IO1 Xplained Pro User Guide can be found on at the address:
http://www.atmel.com/tools/ATIO1-XPRO.aspx

- Correlate ADC Light Sensor pins with **Atmel SAM L21 User Guide:**

Table 4-2 Extension Header EXT2



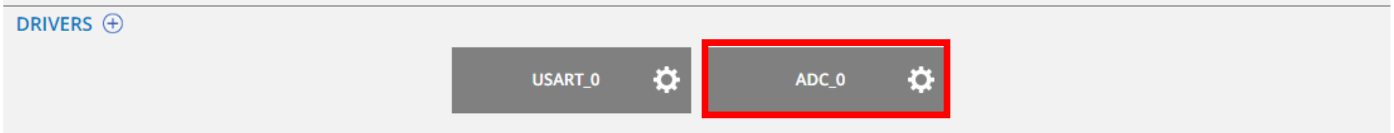
EXT2 pin	SAM L21 pin	Function	Shared functionality
3 [ADC(+)]	PA10	AIN[18] / PTC_Y8	Onboard QTouch Button 1

RESULT ADC Light Sensor is connected to SAM L21 PA10 (ADC+ positive input)

TO DO Add the ADC driver

- Select DASHBOARD and click on “ADD SOFTWARE COMPONENT”.
- Look for the ADC driver and add it.
- You can now complete the addition of the ADC driver by clicking on Add component(s).

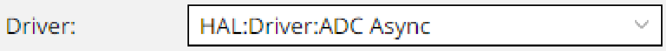
RESULT The ADC driver is added to the application.



TO DO Configure the ADC driver

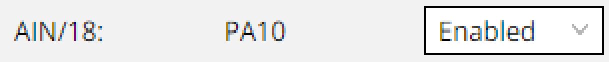
- Click on ADC_0 component block to start its configuration

- Configure ADC Component Settings:
  - Driver: ADC **<u>Async</u>**



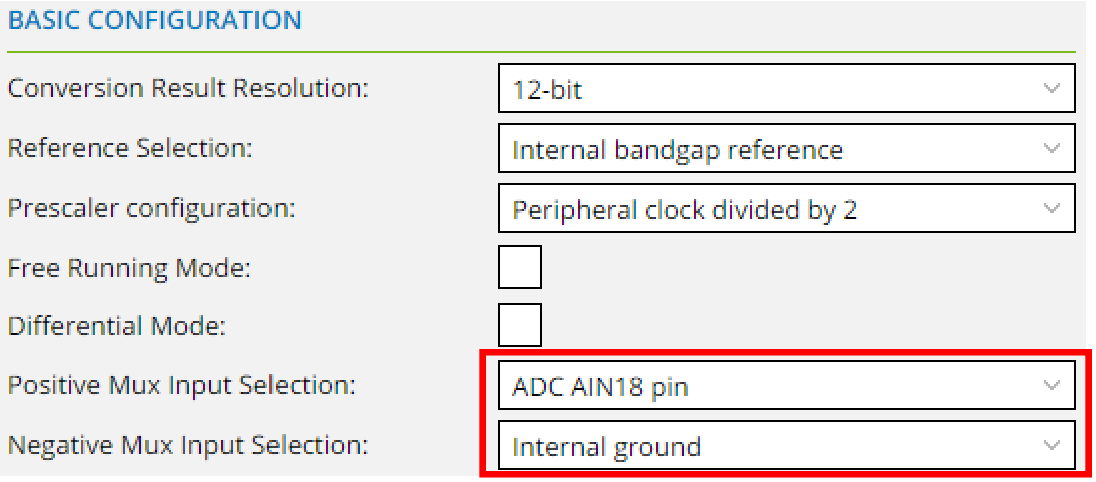
Driver: HAL:Driver:ADC Async

- Configure ADC Signals:
  - AIN/18 - PA10: Enabled



AIN/18: PA10 Enabled

- Configure ADC Basic Configuration:
  - Positive Mux Input Selection: ADC AIN18 pin
  - Negative Mux Input Selection: Internal ground



BASIC CONFIGURATION

Conversion Result Resolution:	12-bit
Reference Selection:	Internal bandgap reference
Prescaler configuration:	Peripheral clock divided by 2
Free Running Mode:	□
Differential Mode:	□
Positive Mux Input Selection:	ADC AIN18 pin
Negative Mux Input Selection:	Internal ground

RESULT The ADC driver is configured

3.3 Add Temperature Sensor Middleware using Atmel Start

I/O1 Xplained Pro extension board features an Atmel AT30TSE758 temperature sensor chip with an 8kb serial EEPROM inside.

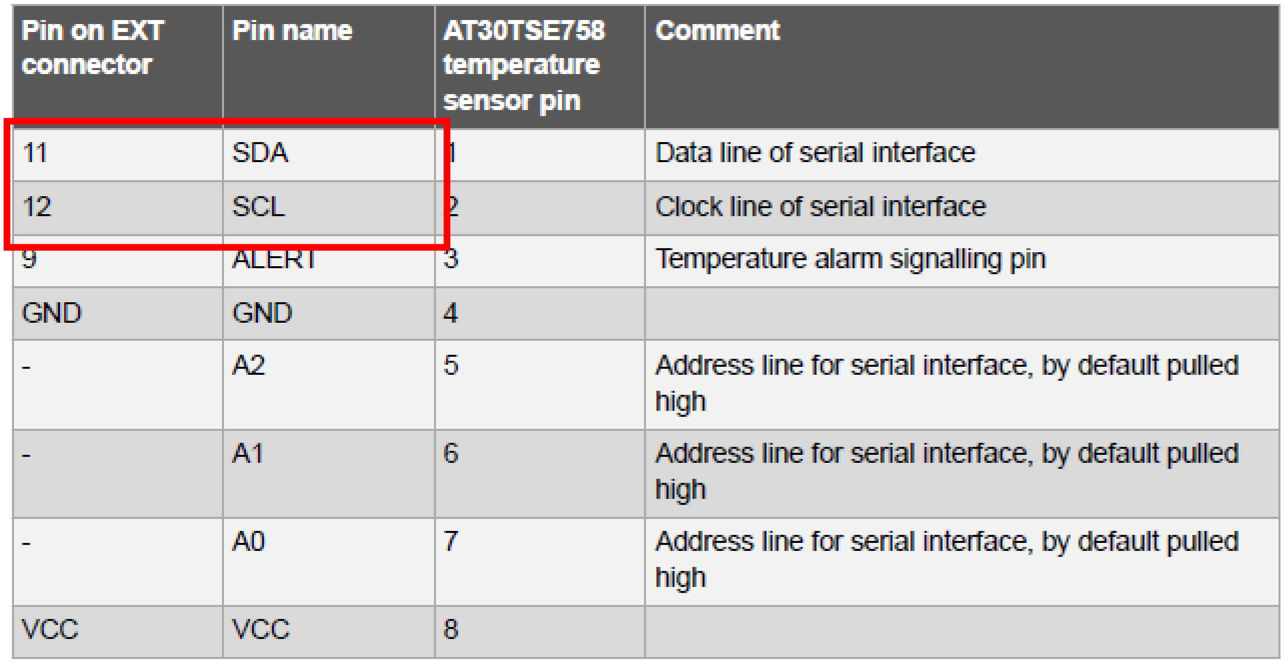
The sensor includes programmable high and low temperature alarms, user selectable temperature resolution up to 12 bits, and an I2C/SMBus™ compatible serial interface.

The temperature sensor is controlled using an I2C peripheral. So, let’s determine which pins from the SAM L21 Xplained Pro interface with it.

TO DO Determine which SAM L21 I/Os interface with the I2C Temperature Sensor

- Get I2C Temperature Sensor pins from **Atmel IO1 Extension Board User Guide:**

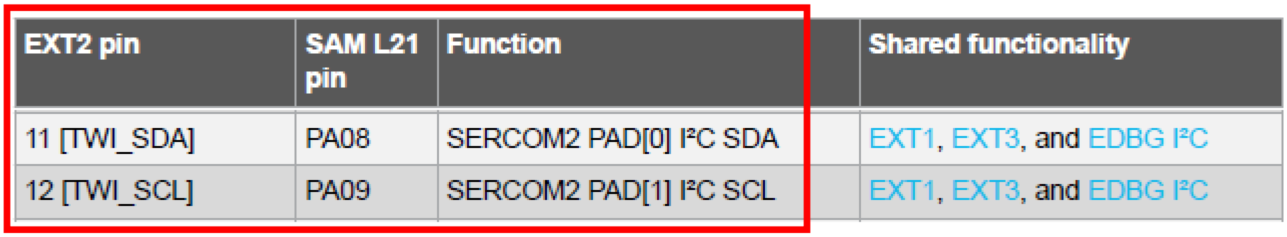
Table 4-8 Temperature Sensor Connections



Pin on EXT connector	Pin name	AT30TSE758 temperature sensor pin	Comment
11	SDA	1	Data line of serial interface
12	SCL	2	Clock line of serial interface
9	ALERT	3	Temperature alarm signalling pin
GND	GND	4	
-	A2	5	Address line for serial interface, by default pulled high
-	A1	6	Address line for serial interface, by default pulled high
-	A0	7	Address line for serial interface, by default pulled high
VCC	VCC	8	

- Correlate I2C Temperature Sensor pins with **Atmel SAM L21 User Guide:**

Table 4-2 Extension Header EXT2



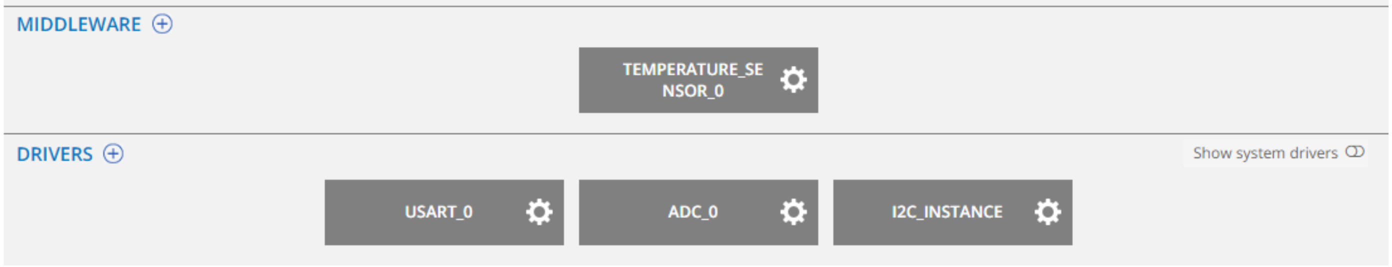
EXT2 pin	SAM L21 pin	Function		Shared functionality
11 [TWI_SDA]	PA08	SERCOM2 PAD[0] I²C SDA		EXT1, EXT3, and EDBG I²C
12 [TWI_SCL]	PA09	SERCOM2 PAD[1] I²C SCL		EXT1, EXT3, and EDBG I²C

RESULT I2C Temperature Sensor is connected to SAM L21 SERCOM2 (PA08 / PA09)

TO DO Add the Temperature Sensor Middleware

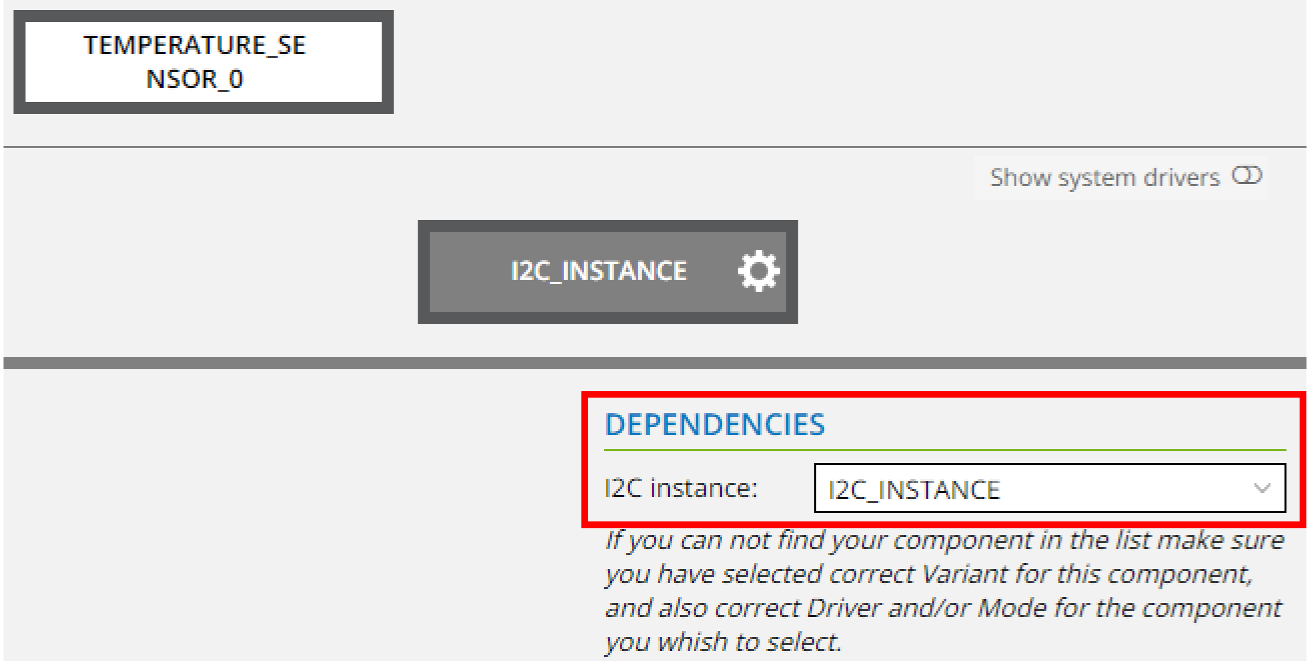
- Select DASHBOARD and click on "ADD SOFTWARE COMPONENT".
- Look for the Temperature Sensor **middleware** and add it.
- You can now complete the addition of the middleware by clicking on Add component(s).

You can check that adding the temperature sensor middleware will automatically add an I2C driver which is required to interface the sensor:



RESULT The Temperature Sensor middleware is added to the application.

INFO If you click on the Temperature Sensor component block, you will see the dependencie(s) of that middleware:



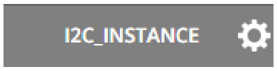
DEPENDENCIES

I2C instance: I2C_INSTANCE

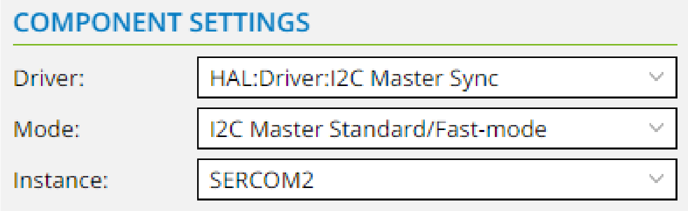
If you can not find your component in the list make sure you have selected correct Variant for this component, and also correct Driver and/or Mode for the component you whish to select.

TO DO Configure the I2C driver

- Click on I2C_INSTANCE component block to start its configuration



- Configure I2C Component Settings:
    - Driver: I2C Master Sync
    - Mode: I2C Master Standard/Fast-mode
    - SERCOM2



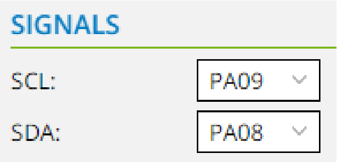
COMPONENT SETTINGS

Driver: HAL:Driver:I2C Master Sync

Mode: I2C Master Standard/Fast-mode

Instance: SERCOM2

- Configure I2C Signals:
    - SCL: PA09
    - SDA: PA08

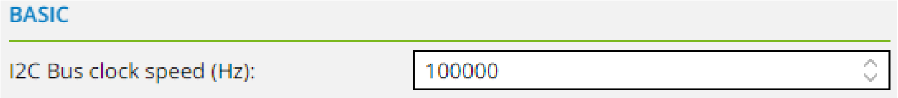


SIGNALS

SCL: PA09

SDA: PA08

- Configure I2C Basic Configuration: I2C Bus Clock speed to 100kHz



BASIC

I2C Bus clock speed (Hz): 100000

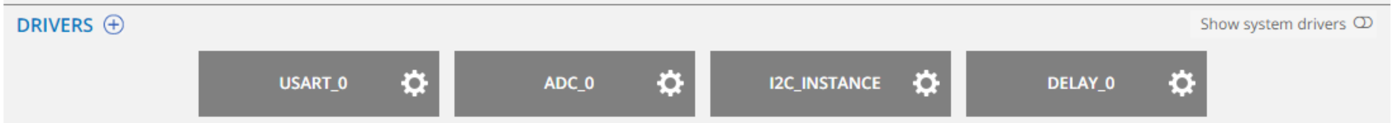
RESULT The I2C driver is configured

3.4 Add Delay Driver using Atmel Start

This driver provides functions which allow to add delays in us or ms using the Cortex-M0+ systick timer.

TO DO Add the Delay driver

- Select DASHBOARD and click on “ADD SOFTWARE COMPONENT”.
- Look for the Delay driver and add it.
- You can now complete the addition of the Delay driver by clicking on Add component(s):



RESULT The Delay driver is added to the application.

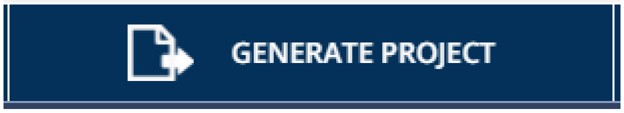
3.5 Update the Application on the existing Atmel Studio 7 project

We have now finished to update our Atmel Start configuration project.

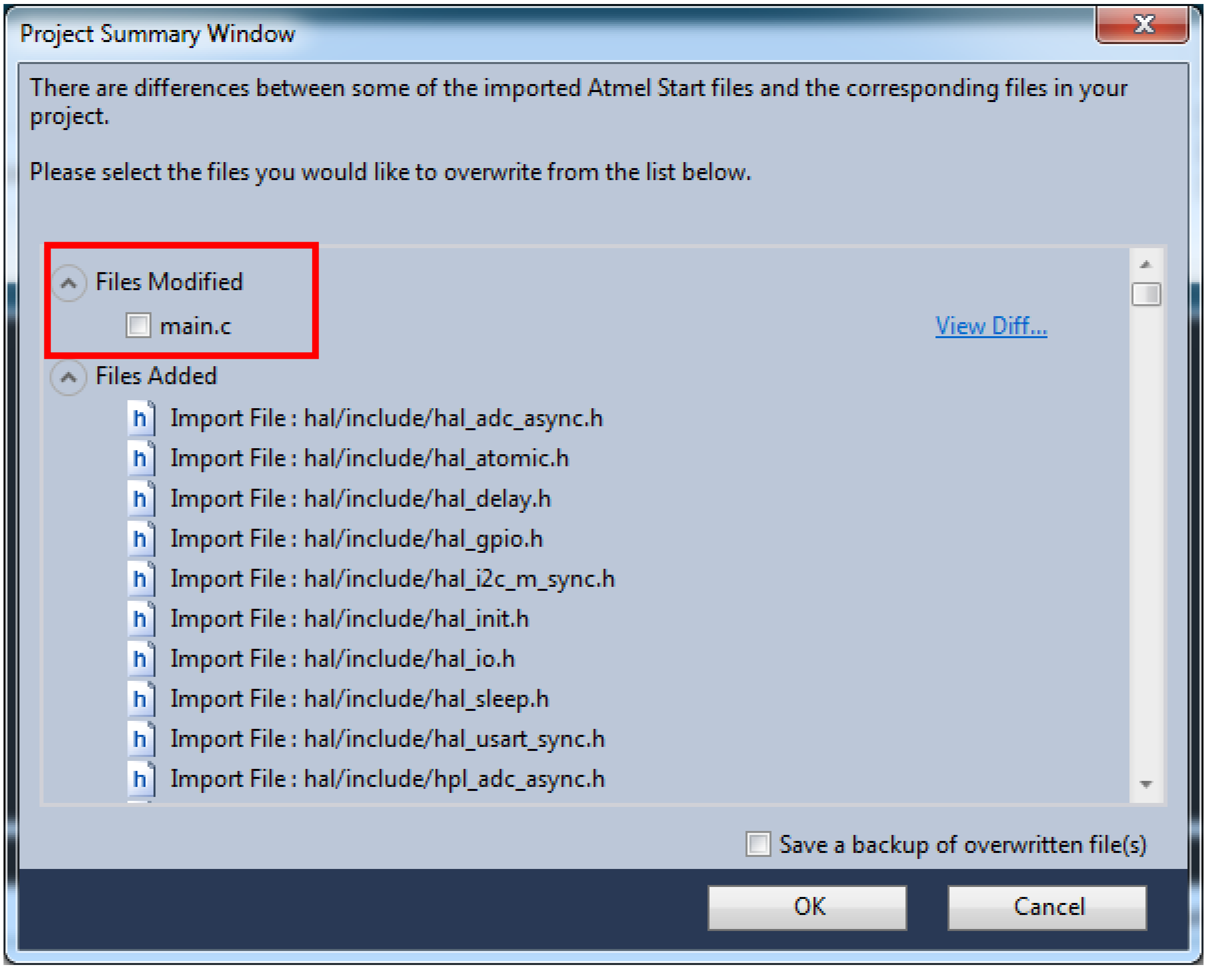
It's time now to update the existing Atmel Studio 7 project we have started to implement.

TO DO Generate and Update Project

- Select GENERATE PROJECT to download the new content:



- Take care to have the main.c file unselected in order to preserve previous implementations made (default selection) and Click OK:



Project Summary Window

There are differences between some of the imported Atmel Start files and the corresponding files in your project.

Please select the files you would like to overwrite from the list below.

Files Modified

□ main.c

View Diff...

Files Added

Import File : hal/include/hal_adc_async.h

Import File : hal/include/hal_atomic.h

Import File : hal/include/hal_delay.h

Import File : hal/include/hal_gpio.h

Import File : hal/include/hal_i2c_m_sync.h

Import File : hal/include/hal_init.h

Import File : hal/include/hal_io.h

Import File : hal/include/hal_sleep.h

Import File : hal/include/hal_usart_sync.h

Import File : hal/include/hpl_adc_async.h

□ Save a backup of overwritten file(s)

RESULT Your application project has been updated within Atmel Studio 7

3.6 ADC Light Sensor Implementation

TO DO Implement the ADC to retrieve Light Sensor Data

Once again, the examples in `atmel_start.c` file will be used to help us get started.

The example for the ADC is different from the USART example because for the ADC, the Async driver option was chosen. This means that the ADC will do measurements and then generate an interrupt.

The ADC interrupt will then call a callback function that will have to be implemented.

- Open `atmel_start.c` and copy both `convert_cb_ADC_0` and `ADC_0_example` functions
- Paste it above `main()` function from `main.c` file
- Rename `ADC_0_example()` function as `ADC_light_init()`
- Rename `convert_cb_ADC_0()` function as `convert_cb_ADC()`
- Update `convert_cb_ADC` callback call in `ADC_light_init()`

The code should look like this:

```
static void convert_cb_ADC(const struct adc_async_descriptor *const descr)
{
}

void ADC_light_init(void)
{
        adc_async_register_callback(&ADC_0, ADC_ASYNC_CONVERT_CB, convert_cb_ADC);
        adc_async_enable(&ADC_0);
        adc_async_start_conversion(&ADC_0);
}
```

- Call `ADC_light_init` after `UART_EDBG_init` and add a debug message as provided below:

```
int main(void)
{
        system_init();

        UART_EDBG_init();

        ADC_light_init();
        io_write(uart_edbg_io, (uint8_t *)"ADC Init\n", 9);

        while(1) {
        }
}
```

We want to start an ADC conversion periodically. To do this, a simple loop with a delay will be used.

- Add a delay of 1 second in the while(1) loop:

```
while(1) {
    delay_ms(1000);
}
```

After one second, we start an ADC conversion:

- Move/Cut the `adc_async_start_conversion()` function from `ADC_light_init()` after the delay:

```
void ADC_light_init(void)
{
  adc_async_register_callback(&ADC_0, ADC_ASYNC_CONVERT_CB, convert_cb_ADC);
  adc_async_enable(&ADC_0);
  ~~adc_async_start_conversion(&ADC_0);~~
}

while(1) {
    delay_ms(1000);

    adc_async_start_conversion(&ADC_0);
}
```

A boolean will also be used to know if the conversion is completed or not.

- Add boolean:

```
struct io_descriptor *uart_edbg_io;
volatile bool conversion_done = false;
```

- Update ADC callback:

```
static void convert_cb_ADC(const struct adc_async_descriptor *const descr)
{
        conversion_done = true;
}
```

- Wait for conversion is done:

```
while(1) {
  delay_ms(1000);

  adc_async_start_conversion(&ADC_0);

  while(!conversion_done);
  conversion_done = false;
}
```

Let’s read now the converted value which consists in a 12-bit value (light sensor resolution):

- Declare a two-byte buffer to store the ADC 12-bit converted data:

```
struct io_descriptor *uart_edbg_io;
volatile bool conversion_done = false;
uint8_t ADC_buffer[2];
```

- Call `adc_async_read` function once data is converted to read the 12-bit value:

```
while(!conversion_done);
conversion_done = false;

adc_async_read(&ADC_0, ADC_buffer, 2);
```

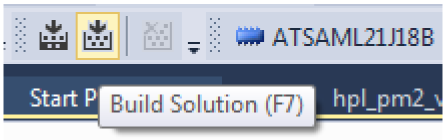
INFO ADC driver functions can be found in the hal/src folder: hal_adc_async.c

RESULT ADC Light Sensor Implementation is completed:

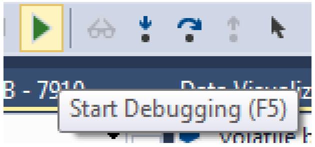
```
#include "atmel_start.h"
#include "atmel_start_pins.h"

struct io_descriptor *uart_edbg_io;
volatile bool conversion_done = false;
uint8_t ADC_buffer[2];

void UART_EDBG_init(void)
{
	usart_sync_get_io_descriptor(&USART_0, &uart_edbg_io);
	usart_sync_enable(&USART_0);

	io_write(uart_edbg_io, (uint8_t *)"Hello World!", 12);
}

static void convert_cb_ADC(const struct adc_async_descriptor *const descr)
{
	conversion_done = true;
}

void ADC_light_init(void)
{
	adc_async_register_callback(&ADC_0, ADC_ASYNC_CONVERT_CB, convert_cb_ADC);
	adc_async_enable(&ADC_0);
}

int main(void)
{
	system_init();

	UART_EDBG_init();

	ADC_light_init();
	io_write(uart_edbg_io, (uint8_t *)"ADC Init\n", 9);

	while(1) {
		delay_ms(1000);

		adc_async_start_conversion(&ADC_0);

		while(!conversion_done);
		conversion_done = false;

		adc_async_read(&ADC_0, ADC_buffer, 2);
	}
}
```

TO DO Test Implementation

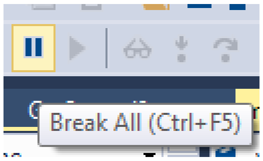
- Compile the project by clicking on the Build Solution icon or by typing ‘F7’.



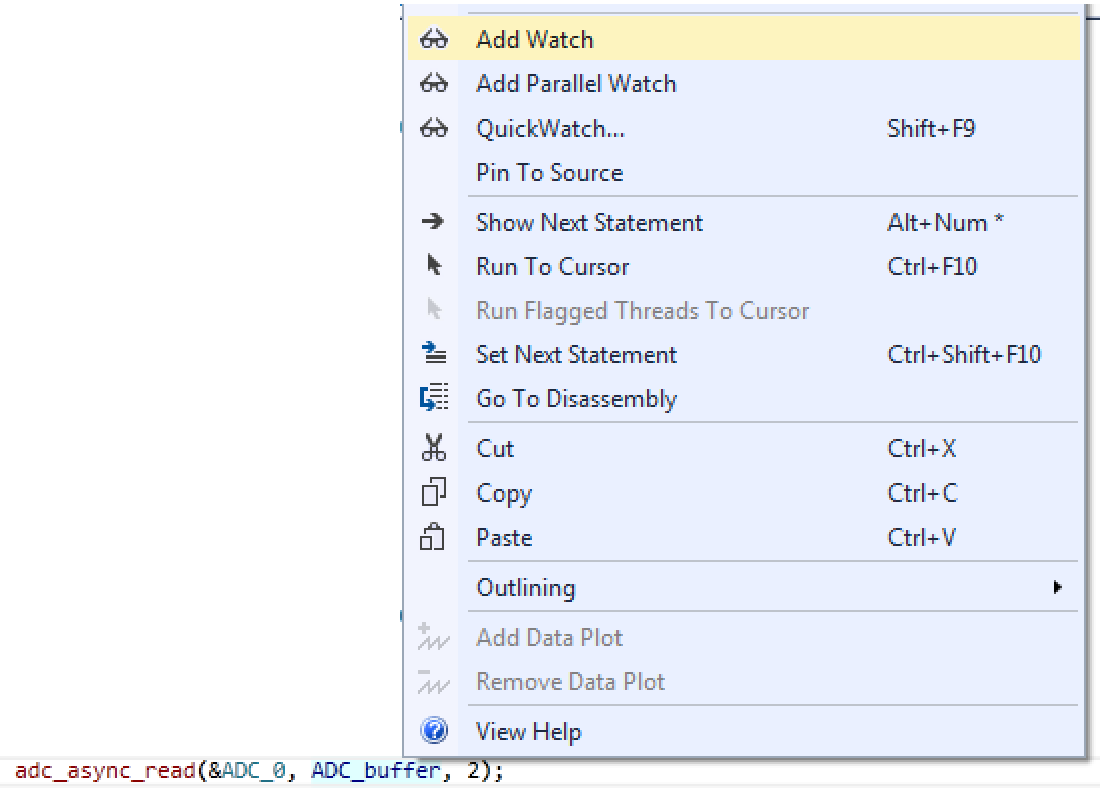
- Launch Debugger by clicking on Start Debugging button:



- Break the application by clicking on Break All button:

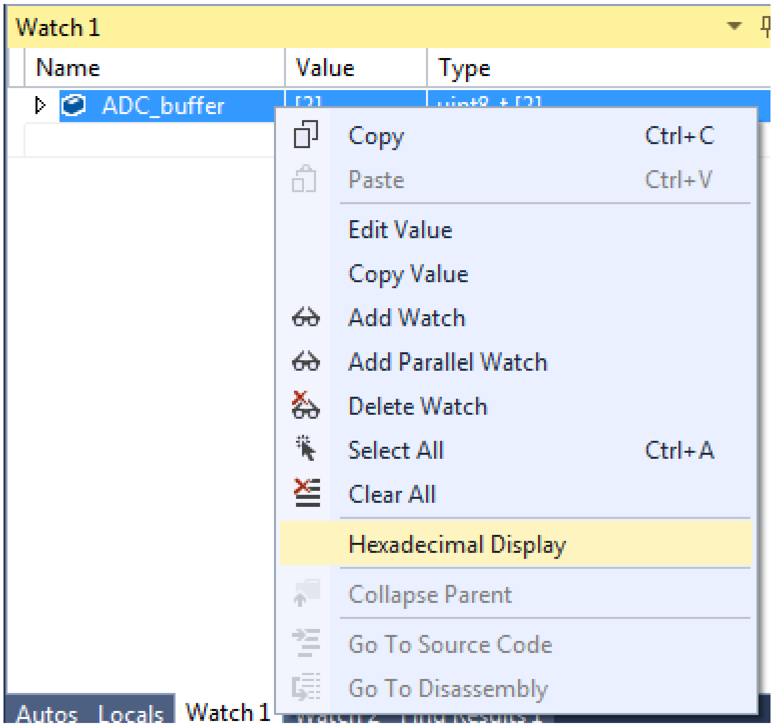


- On the main.c file, right click on `ADC_buffer` variable then Add Watch:

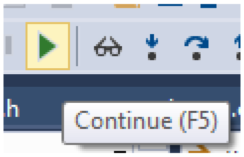


```
adc_async_read(&ADC_0, ADC_buffer, 2);
```

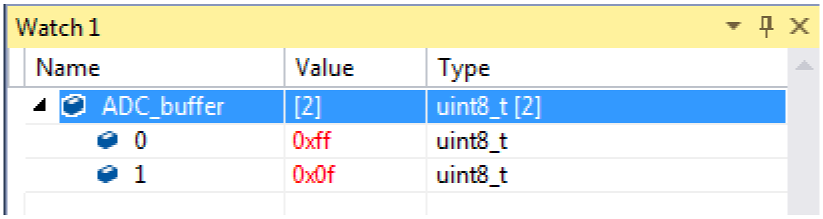
- Watch1 window will appear with your selected variable. Select Hexadecimal Display:



- Click on Continue to execute the application



- If you hide the light sensor then click on Break All button to break the application, you can check the buffer value should approach 0x0FFF:



Name	Value	Type
ADC_buffer	[2]	uint8_t [2]
0	0xff	uint8_t
1	0x0f	uint8_t

INFO ADC Light Sensor values range is between 0x000 and 0xFFF (12-bit value).

RESULT ADC Light Sensor Implementation works as expected.

3.7 I2C Temperature Sensor Implementation

TO DO Implement the I2C to retrieve Temperature Sensor Data

- Add the temperature_sensor_main.h include file:

```
#include "atmel_start.h"
#include "atmel_start_pins.h"
#include "temperature_sensor_main.h"
```

- Call `temperature_sensors_init` after ADC Init and add a debug message as provided below:

```
int main(void)
{
        system_init();

        UART_EDBG_init();

        ADC_light_init();
        io_write(uart_edbg_io, (uint8_t *)"ADC Init\n", 9);

        temperature_sensors_init();
        io_write(uart_edbg_io, (uint8_t *)"I2C Init\n", 9);
```

INFO Temperature sensor initialization source code can be found in the `temperature_sensor_main.c` file.

A global variable will also be used to get temperature data.

- Add temperature global variable:

```
struct io_descriptor *uart_edbg_io;
volatile bool conversion_done = false;
uint8_t ADC_buffer[2];
float temperature;
```

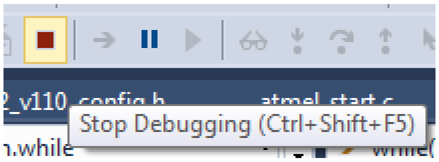
- Get temperature using `at30tse75x_read` function:

```
while(1) {
        delay_ms(1000);

        temperature = at30tse75x_read(TEMPERATURE_SENSOR_0);

        adc_async_start_conversion(&ADC_0);

        while(!conversion_done);
        conversion_done = false;

        adc_async_read(&ADC_0, ADC_buffer, 2);
```

RESULT I2C Temperature Sensor Implementation is completed.

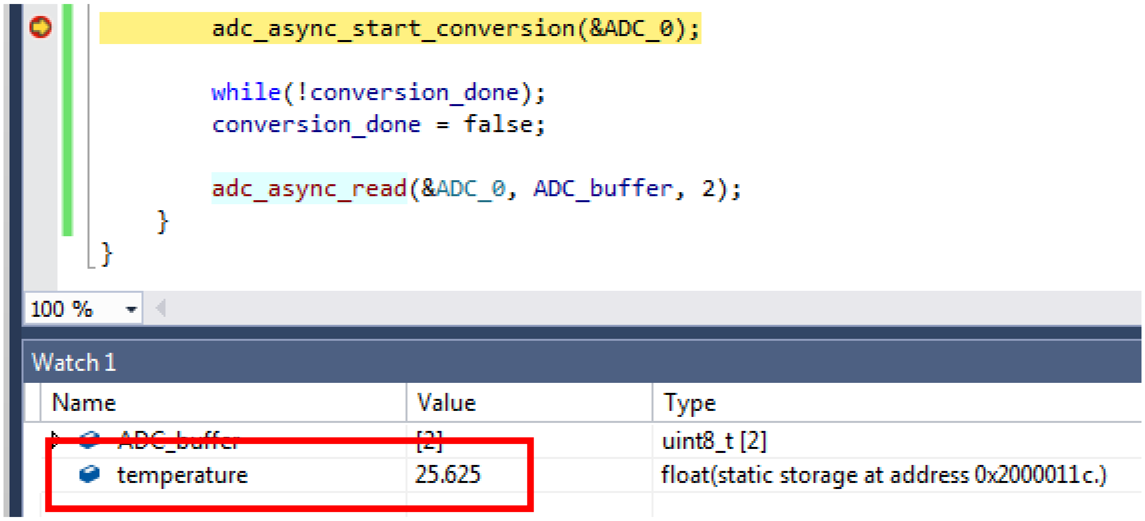
TO DO Test Implementation

- Stop the Debugger



Stop Debugging (Ctrl+Shift+F5)

- Compile the project by clicking on the Build Solution icon or by typing ‘F7’.
- Launch Debugger by clicking on Start Debugging button then break the application by clicking on Break All button
- Right click on `temperature` variable then Add Watch (Unselect Hexadecimal Display)
- Put a breakpoint after `at30tse_read_temperature` call and execute the application to get the temperature value:



```
        adc_async_start_conversion(&ADC_0);

        while(!conversion_done);
        conversion_done = false;

        adc_async_read(&ADC_0, ADC_buffer, 2);
    }
}
```

Watch 1

Name	Value	Type
ADC_buffer	[2]	uint8_t [2]
temperature	25.625	float(static storage at address 0x2000011c.)

RESULT I2C Temperature Sensor Implementation works as expected.

4. Assignment 3: Sensors Data Visualization using Atmel Data Visualizer

We will use Atmel Data Visualizer tool to display the light and temperature sensors data on Atmel Studio.

Atmel Data Visualizer is a program used for processing and visualizing data which can communicate with any Xplained Pro boards using a specific interface called Data Gateway Interface (DGI).

Atmel Data Gateway Interface (DGI) can be accessed using an SPI communication channel.

We will then add the SPI driver using Atmel Start to our application in order to use the DGI to communicate with Atmel Data Visualizer tool.

4.1 Recover Atmel Start Project

TO DO Re-Configure Atmel Start Project

- Right click on the Project and Select Re-Configure Atmel Start Project

RESULT It is now possible to add the SPI driver to the project using Atmel START.

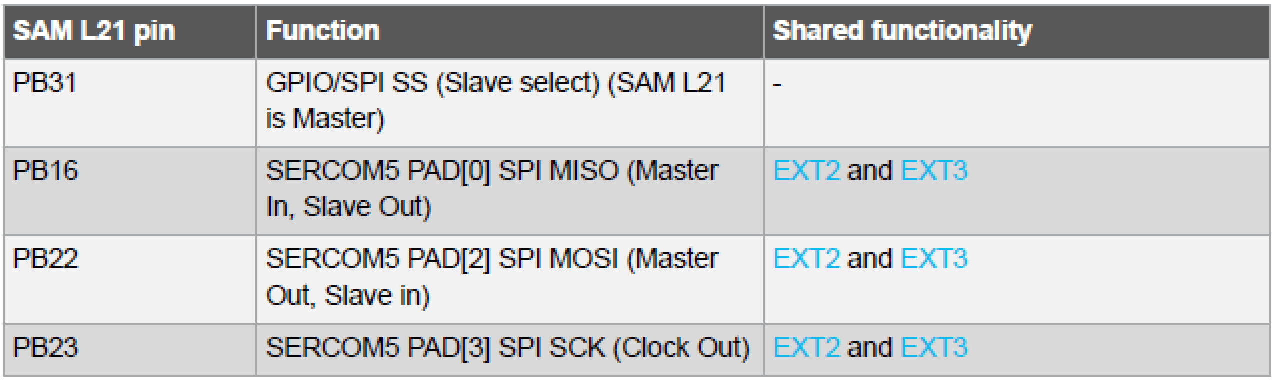
4.2 Add SPI Driver using Atmel Start

Firstly, let’s determine which I/Os need to be configured for the SPI in our project.

TO DO Determine which SAM L21 I/Os interface with the SPI DGI

- Get DGI Interface Connections from **Atmel SAM L21 Xplained Pro User Guide:**

Table 4-15 DGI Interface Connections When Using SPI

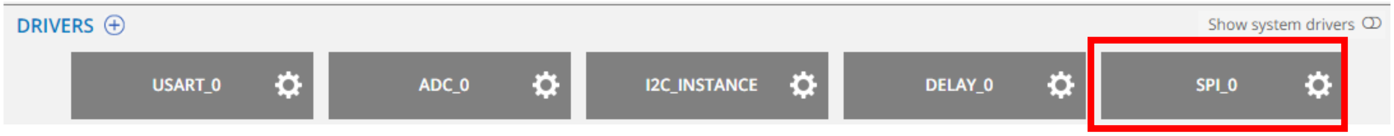


SAM L21 pin	Function	Shared functionality
PB31	GPIO/SPI SS (Slave select) (SAM L21 is Master)	-
PB16	SERCOM5 PAD[0] SPI MISO (Master In, Slave Out)	EXT2 and EXT3
PB22	SERCOM5 PAD[2] SPI MOSI (Master Out, Slave in)	EXT2 and EXT3
PB23	SERCOM5 PAD[3] SPI SCK (Clock Out)	EXT2 and EXT3

RESULT SPI is connected to SAM L21 SERCOM5 (PB31 / PB16 / PB22 / PB23)

TO DO Add the SPI driver

- Select DASHBOARD and click on “ADD SOFTWARE COMPONENT”.
- Look for the SPI driver and add it.
- You can now complete the addition of the SPI driver by clicking on Add component(s):

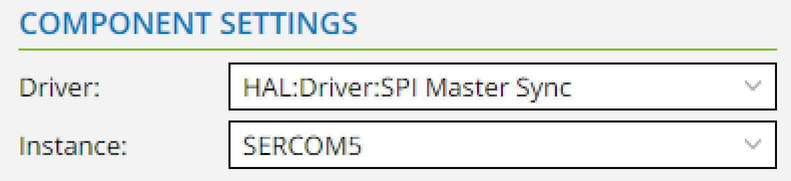


RESULT The SPI driver is added to the application.

TO DO Configure the SPI driver

- Click on SPI_0 component block to start its configuration

- Configure SPI Component Settings:
  - Driver: SPI Master Sync
  - SERCOM5

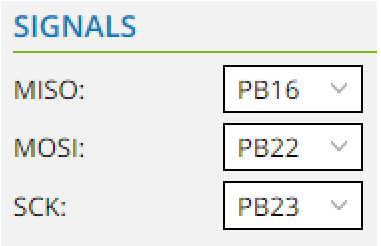


COMPONENT SETTINGS

Driver: HAL:Driver:SPI Master Sync

Instance: SERCOM5

- Configure SPI Signals:
  - MISO: PB16
  - MOSI: PB22
  - SCK: PB23



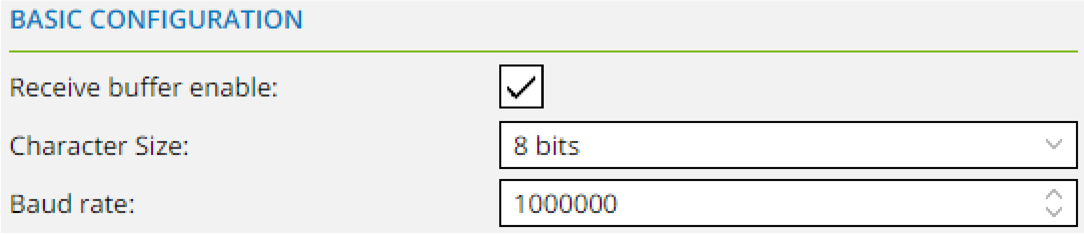
SIGNALS

MISO: PB16

MOSI: PB22

SCK: PB23

- Configure SPI Basic Configuration: SPI Baud rate to 1Mbit

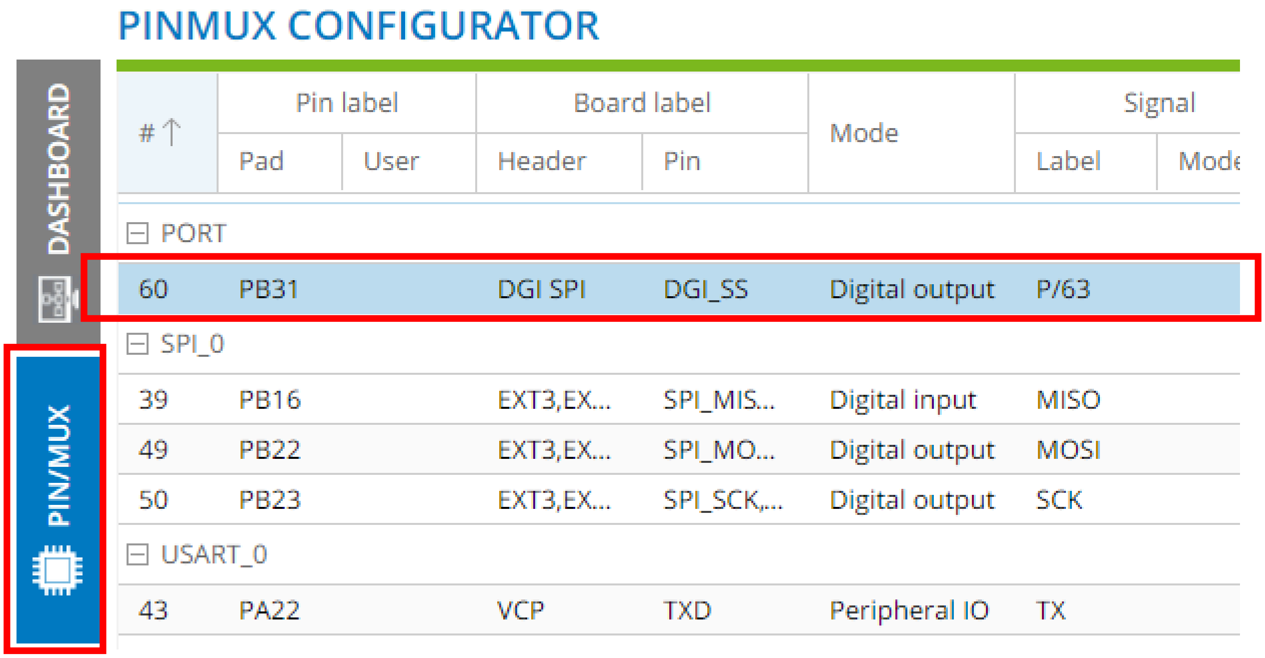


BASIC CONFIGURATION

Receive buffer enable:	☑
Character Size:	8 bits
Baud rate:	1000000

In addition to the SERCOM5, we will also need to configure an I/O (PB31) as Slave Select pin.

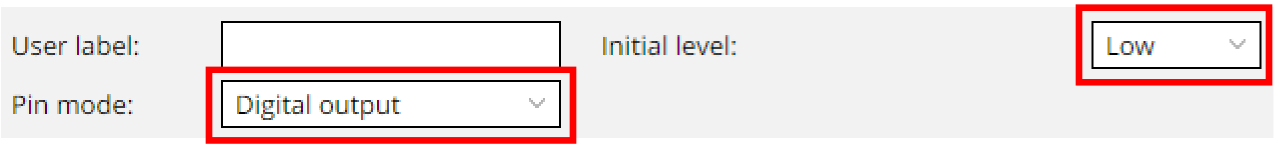
- In Atmel START, select PINMUX and click on PB31 which is listed as the DGI_SS pin:



PINMUX CONFIGURATOR

# ↑	Pin label		Board label		Mode	Signal	
	Pad	User	Header	Pin		Label	Mode
⊟ PORT							
60	PB31		DGI SPI	DGI_SS	Digital output	P/63	
⊟ SPI_0							
39	PB16		EXT3,EX...	SPI_MIS...	Digital input	MISO	
49	PB22		EXT3,EX...	SPI_MO...	Digital output	MOSI	
50	PB23		EXT3,EX...	SPI_SCK,...	Digital output	SCK	
⊟ USART_0							
43	PA22		VCP	TXD	Peripheral IO	TX	

- Update PB31 I/O configuration:
  - Select Digital output as Pin mode
  - Select Low as Initial level



User label:		Initial level:	Low
Pin mode:	Digital output		

☑ **RESULT** The SPI driver is configured

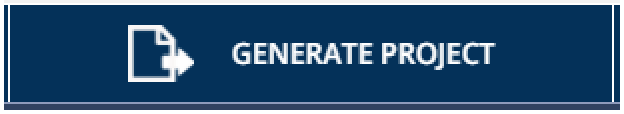
4.3 Update the Application on the existing Atmel Studio 7 project

We have now finished to update our Atmel Start configuration project.

It’s time now to update the existing Atmel Studio 7 project we have started to implement.

TO DO Generate and Update Project

- Select GENERATE PROJECT to download the new content:



RESULT Your application project has been updated within Atmel Studio 7

4.4 SPI Implementation

ADC and I2C Sensors Data will be sent using SPI to the Embedded Debugger DGI Interface to be displayed on Atmel Data Visualizer.

TO DO Implement the SPI to send sensors data to the Embedded Debugger DGI Interface

Once again, the examples in `atmel_start.c` file will be used to help us get started.

- Close Debug session
- Open `atmel_start.c` and copy `SPI_0_example` function
- Paste it above `main()` function from `main.c` file
- Rename `SPI_0_example()` function as `SPI_DGI_init()`
- Update `io_write` call message: `io_write(uart_edbg_io, (uint8_t *)"\nSPI Init\n", 10);`

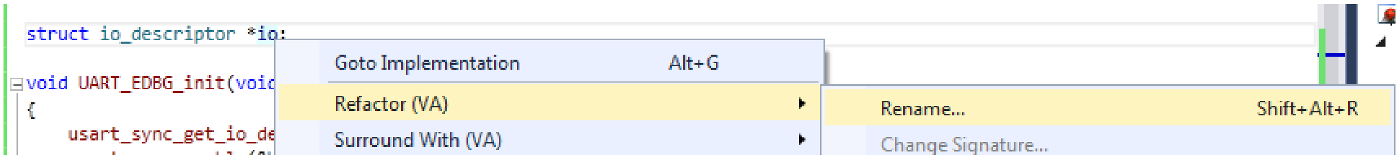
The code should look like this:

```
void SPI_DGI_init(void)
{
        struct io_descriptor *io;
        spi_m_sync_get_io_descriptor(&SPI_0, &io);

        spi_m_sync_enable(&SPI_0);
        io_write(uart_edbg_io, (uint8_t *)"\nSPI Init\n", 10);
}
```

- Move/Cut the `struct io_descriptor *io;` line outside of this function

```
struct io_descriptor *uart_edbg_io;
struct io_descriptor *io;
volatile bool conversion_done = false;
uint8_t ADC_buffer[2];
float temperature ;


void SPI_DGI_init(void)
{
     ~~struct io_descriptor *io;~~
     spi_m_sync_get_io_descriptor(&SPI_0, &io);

     spi_m_sync_enable(&SPI_0);
     io_write(uart_edbg_io, (uint8_t *)"\nSPI Init\n", 10);
}
```

- Right click on “io” and select Refactor (VA) -> Rename... which allows to rename all found references of a searched instance.

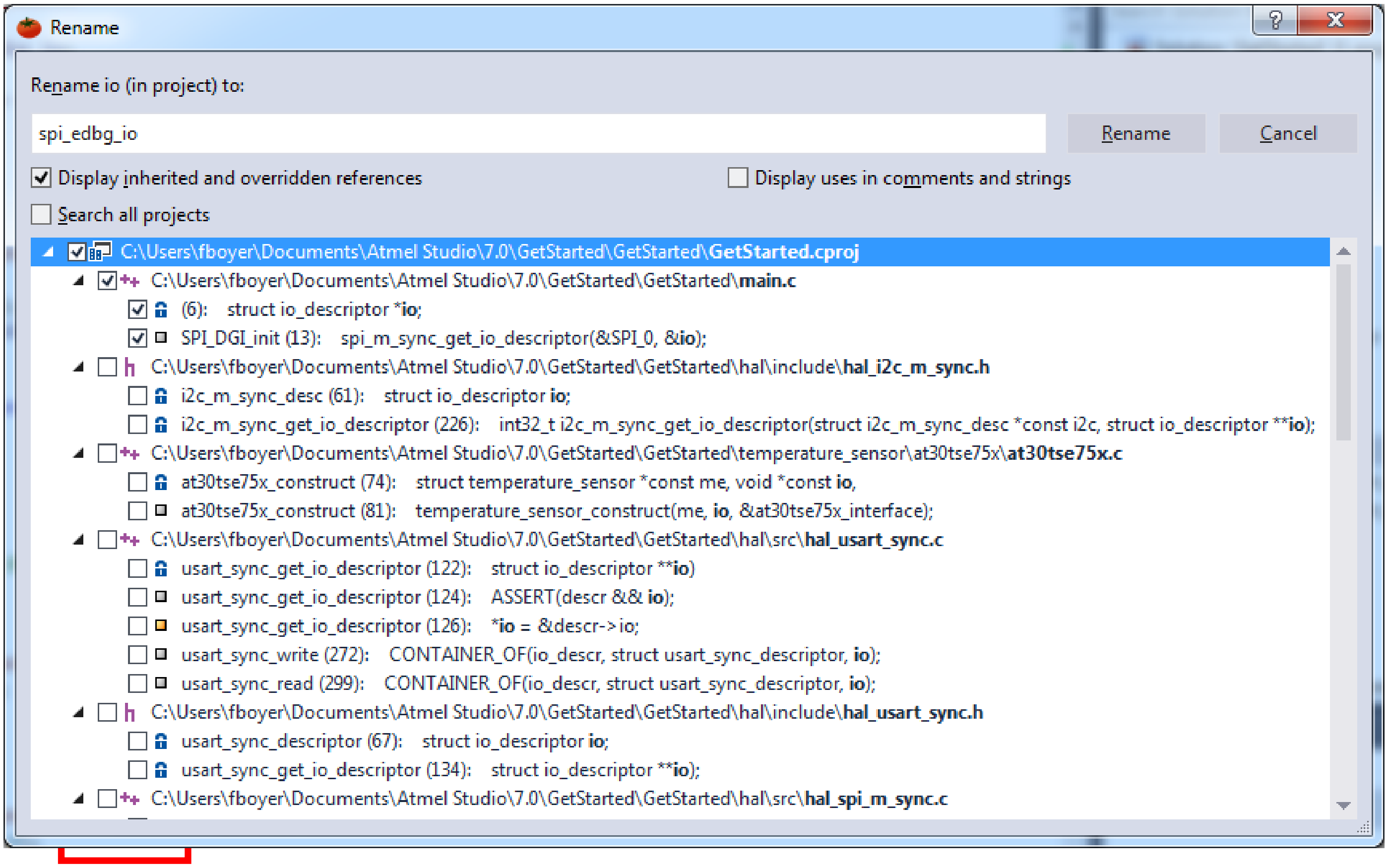


```
struct io_descriptor *io;

void UART_EDBG_init(void
{
    usart_sync_get_io_de
```

- Deselect ALL the lines which do not correspond to main.c file

- Rename it from “io” to “spi_edbg_io” and Click on Rename:

WARNING Make sure that only the instances in main.c are renamed or you may modify the different drivers.



Rename

Rename io (in project) to:

spi_edbg_io

☑ Display inherited and overridden references □ Display uses in comments and strings

□ Search all projects

- ☑ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\GetStarted.cproj
  - ☑ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\main.c
    - ☑ (6): struct io_descriptor *io;
    - ☑ SPI_DGI_init (13): spi_m_sync_get_io_descriptor(&SPI_0, &io);
  - □ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\hal\include\hal_i2c_m_sync.h
    - □ i2c_m_sync_desc (61): struct io_descriptor io;
    - □ i2c_m_sync_get_io_descriptor (226): int32_t i2c_m_sync_get_io_descriptor(struct i2c_m_sync_desc *const i2c, struct io_descriptor **io);
  - □ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\temperature_sensor\at30tse75x\at30tse75x.c
    - □ at30tse75x_construct (74): struct temperature_sensor *const me, void *const io,
    - □ at30tse75x_construct (81): temperature_sensor_construct(me, io, &at30tse75x_interface);
  - □ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\hal\src\hal_usart_sync.c
    - □ usart_sync_get_io_descriptor (122): struct io_descriptor **io)
    - □ usart_sync_get_io_descriptor (124): ASSERT(descr && io);
    - □ usart_sync_get_io_descriptor (126): *io = &descr->io;
    - □ usart_sync_write (272): CONTAINER_OF(io_descr, struct usart_sync_descriptor, io);
    - □ usart_sync_read (299): CONTAINER_OF(io_descr, struct usart_sync_descriptor, io);
  - □ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\hal\include\hal_usart_sync.h
    - □ usart_sync_descriptor (67): struct io_descriptor io;
    - □ usart_sync_get_io_descriptor (134): struct io_descriptor **io);
  - □ C:\Users\fboyer\Documents\Atmel Studio\7.0\GetStarted\GetStarted\hal\src\hal_spi_m_sync.c

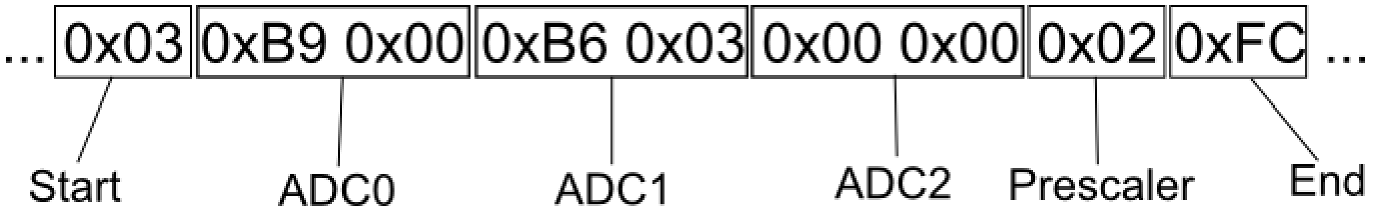
- Call SPI_DGI_init(); after UART_EDBG_init();

```
int main(void)
{
      system_init();

      UART_EDBG_init();

      SPI_DGI_init();
```

The protocol supported by the DGI interface and that we will use to display the sensors data is called Data Stream protocol.

The data stream protocol 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.

Here is a stream format example of a raw data transmission of four different data where ADC0 value is 0x00B9, ADC1 value is 0x3B6, ADC2 value is 0x0000 and Prescaler value is 2:



INFO More info here: http://www.atmel.com/webdoc/dv/dv.Modules.DataStreamer.html

For our application, we need two bytes for the 12-bit ADC value and we will use one byte for the temperature data (we will cast it as a uint8_t to remove temperature decimals).

So three bytes are required for our data steam in addition to the Start and End patterns:

- Create a global 8-bit array that is five elements long and has Start and End patterns on top of `main.c` file:

```
float temperature ;
uint8_t DataStream_buf[5] = {0x03, 0x00, 0x00, 0x00, 0xFC};
```

Data values will contain the light and temperature data.

The light data has 12-bit resolution which requires then two data bytes

The temperature is read out as a float but to avoid having to handle the decimals in the protocol, it will be cast to a uint8_t integer.

The data for light and temperature must be updated in the while loop.

- Update while(1) loop to prepare DataSteam buffer and send its content to the DGI interface using a SPI write:

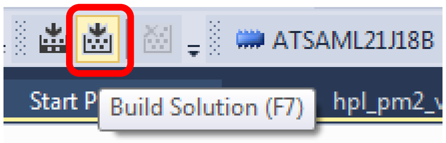
```
adc_async_read(&ADC_0, ADC_buffer, 2);

DataStream_buf[1] = (uint8_t)temperature;
DataStream_buf[2] = ADC_buffer[0];
DataStream_buf[3] = ADC_buffer[1];

io_write(spi_edbg_io, DataStream_buf, 5);
```

RESULT SPI DGI implementation is completed.

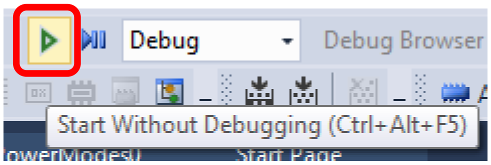
4.5 Configure Atmel Data Visualizer

TO DO Compile and Program the Project

- Compile the project by clicking on the Build Solution icon or by typing ‘F7’.



- Program the application by clicking on the Start Without Debugging icon



RESULT The application is programmed and runs out of the target.

The data stream protocol relies upon a configuration text file that we have to create in order to interpret the string of data being sent to it.

INFO More info here: http://www.atmel.com/webdoc/dv/dv.Modules.DataStreamer.html

TO DO Create DataStream Configuration File

- Open a text editor in order to create a .txt file
- Paste the following text to that file:

```
B,1,1,Temperature
D,1,2,Light_level
```

INFO B means one unsigned byte, D means two unsigned bytes.
1,1 and 1,2 represents their index in the DataStream buffer sent
Temperature and Light_level are labels displayed in Data Visualizer

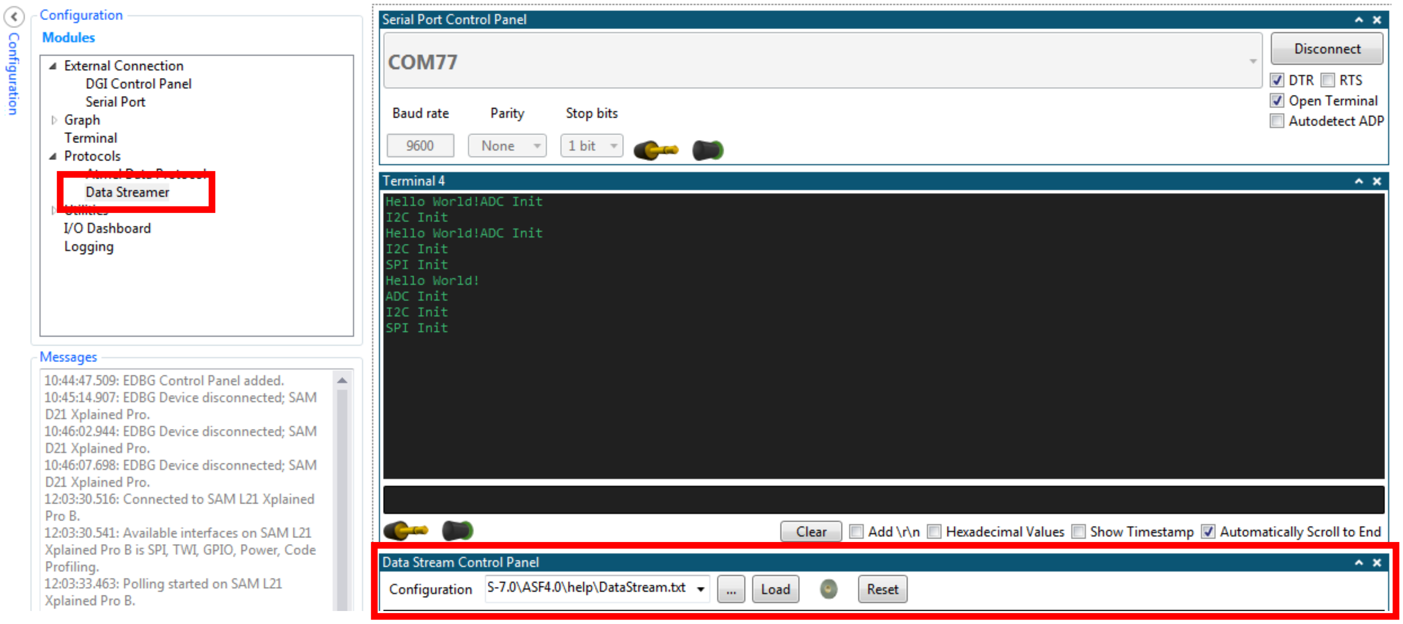
- Save the file as DataStream.txt (as an example)

RESULT DataStream Configuration File is created

TO DO Configure Atmel Data Visualizer

- Open Data Visualizer by clicking on Tools > Data Visualizer.

- Click on Configuration > Protocols > Data Streamer.



```
10:44:47.509: EDBG Control Panel added.
10:45:14.907: EDBG Device disconnected; SAM D21 Xplained Pro.
10:46:02.944: EDBG Device disconnected; SAM D21 Xplained Pro.
10:46:07.698: EDBG Device disconnected; SAM D21 Xplained Pro.
12:03:30.516: Connected to SAM L21 Xplained Pro B.
12:03:30.541: Available interfaces on SAM L21 Xplained Pro B is SPI, TWI, GPIO, Power, Code Profiling.
12:03:33.463: Polling started on SAM L21 Xplained Pro B.
```

COM77

☑ DTR ☐ RTS
☑ Open Terminal
☐ Autodetect ADP

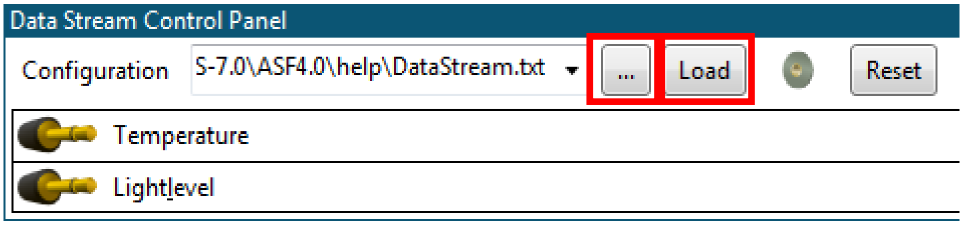
Baud rate: 9600 Parity: None Stop bits: 1 bit

```
Hello World!ADC Init
I2C Init
Hello World!ADC Init
I2C Init
SPI Init
Hello World!
ADC Init
I2C Init
SPI Init
```

☐ Add \r\n ☐ Hexadecimal Values ☐ Show Timestamp ☑ Automatically Scroll to End

Configuration: S-7.0\ASF4.0\help\DataStream.txt

- Search for the DataStream.txt then click on Load:



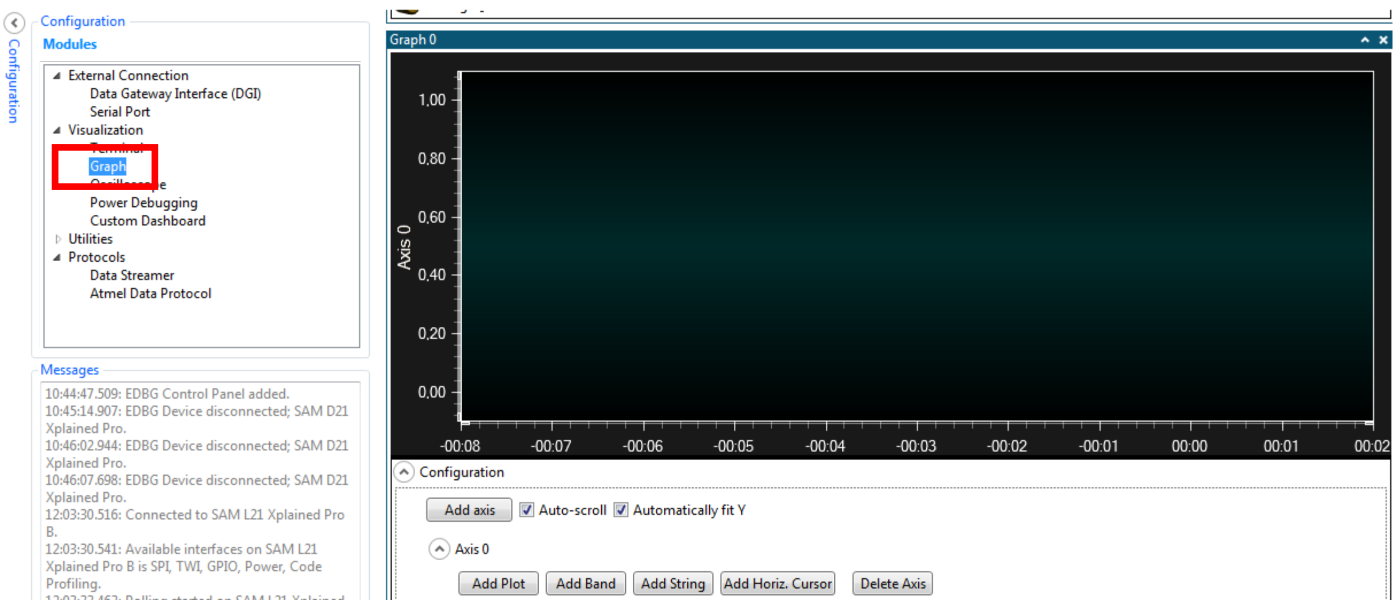
Configuration: S-7.0\ASF4.0\help\DataStream.txt

Temperature

Lightlevel

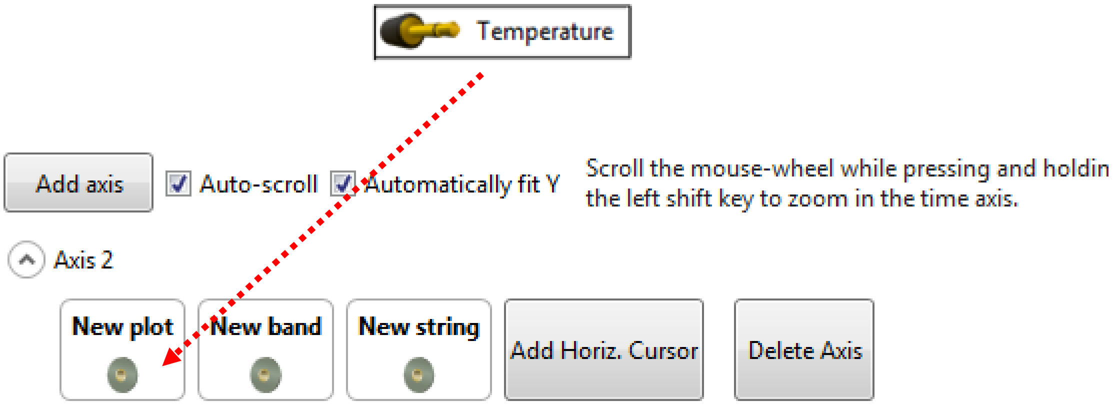
INFO You will get Temperature and Light_level labels you defined in DataStream.txt file.

- Double Click on Visualization > Graph to open the Graph Window:



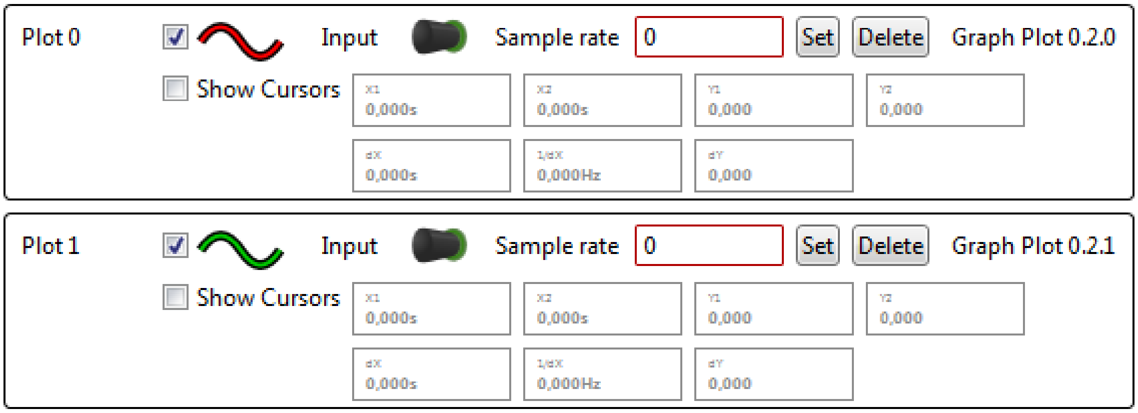
```
10:44:47.509: EDBG Control Panel added.
10:45:14.907: EDBG Device disconnected; SAM D21 Xplained Pro.
10:46:02.944: EDBG Device disconnected; SAM D21 Xplained Pro.
10:46:07.698: EDBG Device disconnected; SAM D21 Xplained Pro.
12:03:30.516: Connected to SAM L21 Xplained Pro B.
12:03:30.541: Available interfaces on SAM L21 Xplained Pro B is SPI, TWI, GPIO, Power, Code Profiling.
```

- Drag and Drop the Temperature connector to the “New plot” receptacle:



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

- Do the same operation for the Lightlevel connector. You will have two plots created:



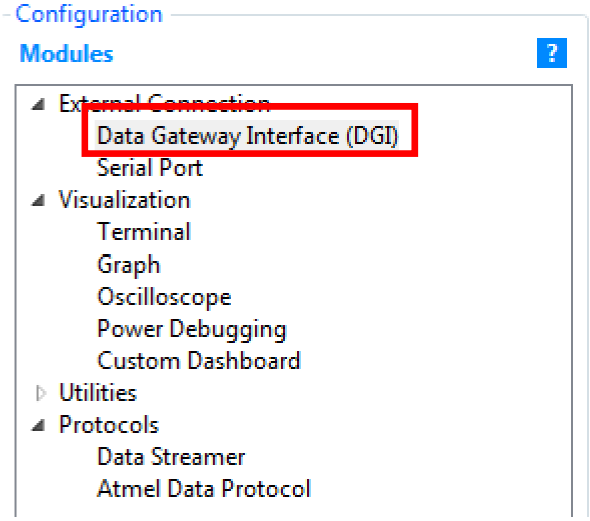
Plot 0 ☑ Input Sample rate 0 Graph Plot 0.2.0

□ Show Cursors X1 0,000s X2 0,000s Y1 0,000 Y2 0,000 dX 0,000s 1/dX 0,000Hz dY 0,000

Plot 1 ☑ Input Sample rate 0 Graph Plot 0.2.1

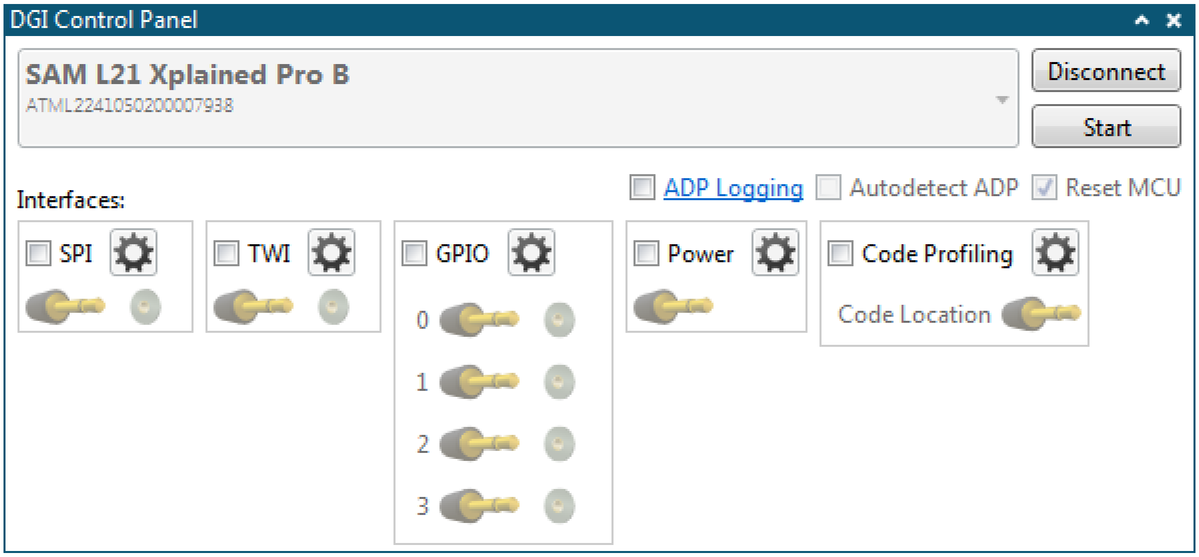
□ Show Cursors X1 0,000s X2 0,000s Y1 0,000 Y2 0,000 dX 0,000s 1/dX 0,000Hz dY 0,000

- Double Click on External Connection > Data Gateway Interface (DGI) to open the DGI Control Panel:


Configuration
Modules

- External Connection
  - Data Gateway Interface (DGI)
  - Serial Port
- Visualization
  - Terminal
  - Graph
  - Oscilloscope
  - Power Debugging
  - Custom Dashboard
- Utilities
- Protocols
  - Data Streamer
  - Atmel Data Protocol

- Click on Connect:

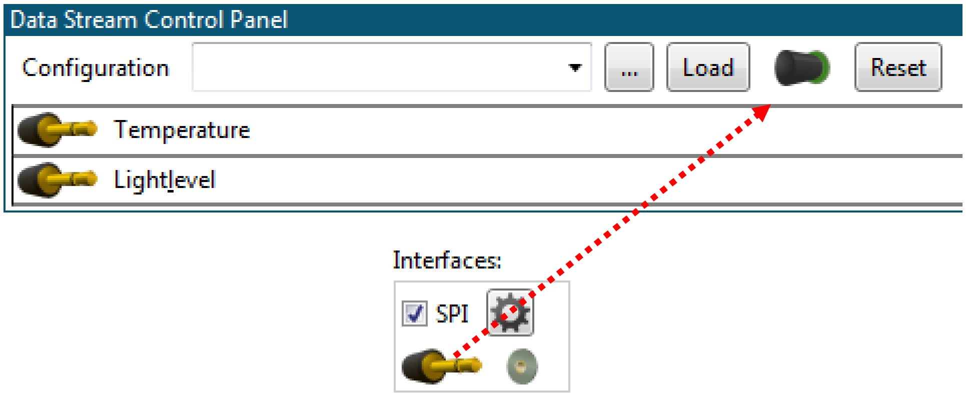

DGI Control Panel

SAM L21 Xplained Pro B
ATML2241050200007938

□ ADP Logging □ Autodetect ADP ☑ Reset MCU

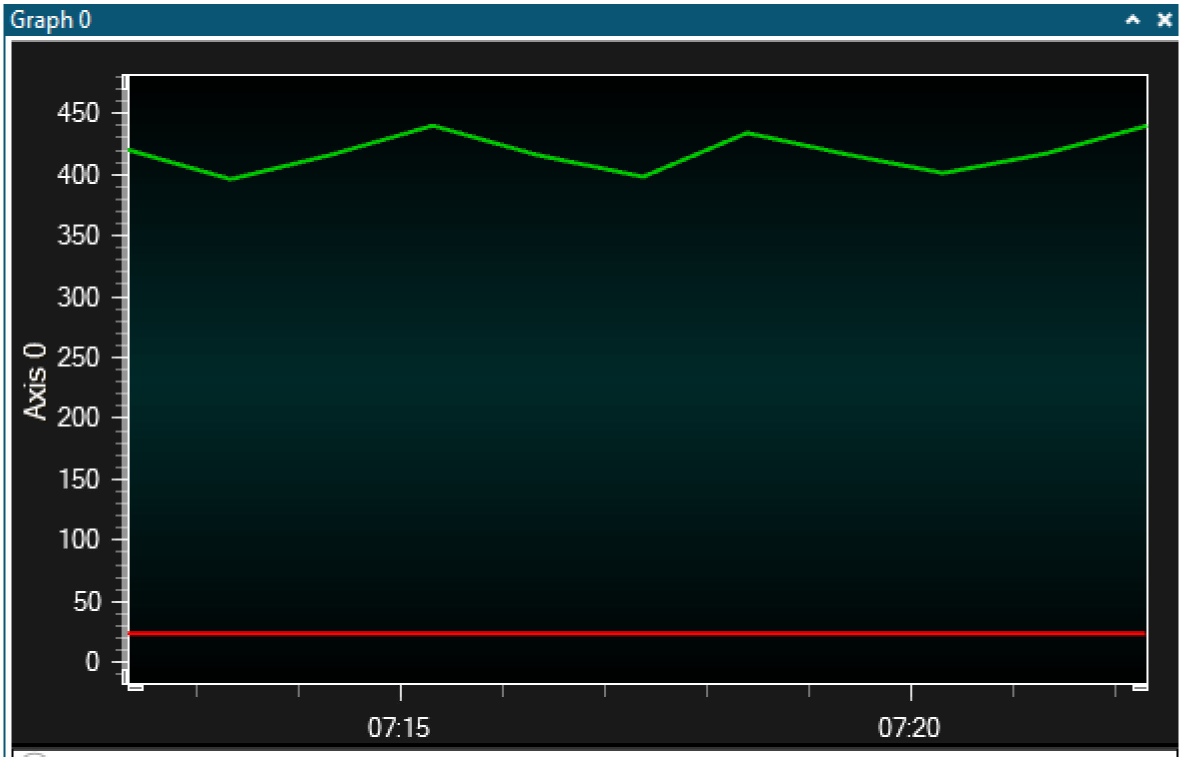
Interfaces:
□ SPI □ TWI □ GPIO □ Power □ Code Profiling

- Select SPI as DGI Interface and drag and drop the plug to Data Steam Protocol receptacle, then Click on Start:

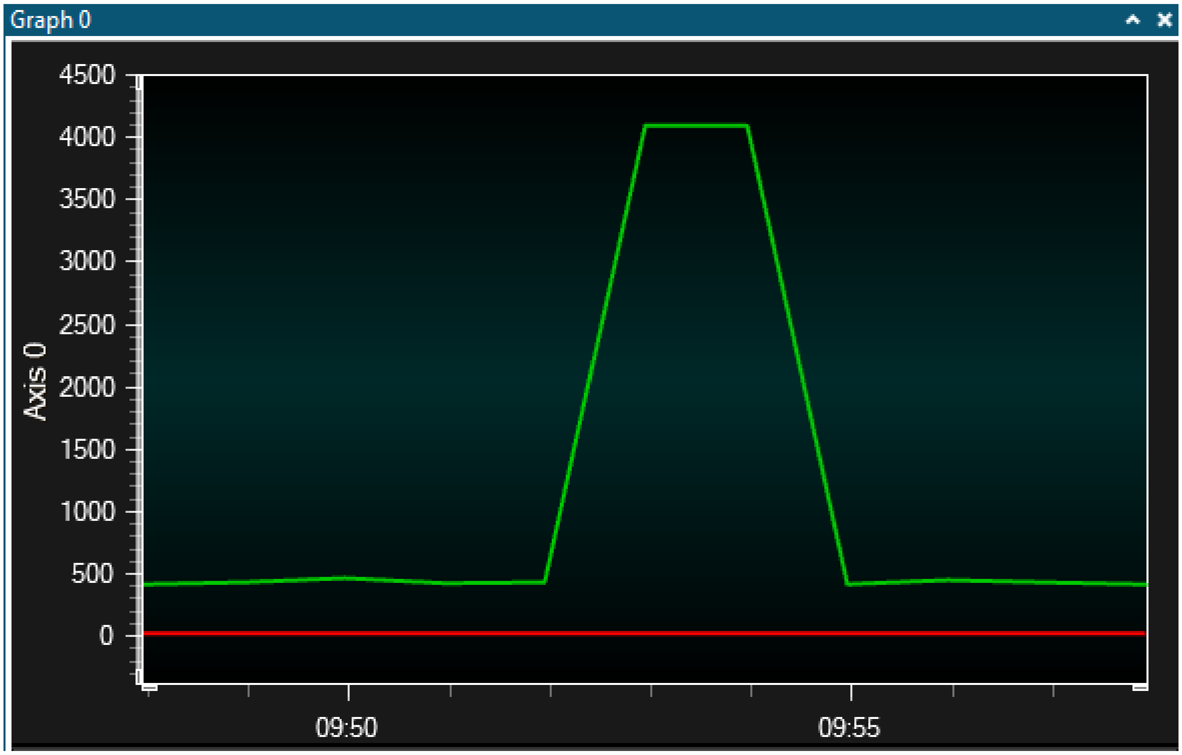

Data Stream Control Panel

Interfaces:
☑ SPI

☑ **RESULT** The Temperature and Light Sensor values should be displayed on the graphs.



Depending on your luminosity, if you hide the light sensor, you should see a peak around 4000 (max value is 4095) which is in the range of the data you measured during debugging (0xFF0)



5. Conclusion

In this hands-on, we have described and illustrated how to create from scratch with Atmel Start a project and get an application up and running in Atmel Studio IDE.

We have also demonstrated different Atmel Studio 7 features such as the Atmel Data Visualizer tool which can be used to display graphs of user data using an easy to use interface.

After completing this hands-on, you should now:

- Have a clear picture of an Atmel Start project creation.
- Know how to import and update an Atmel Start project in Atmel Studio 7.
- Know how to print debug messages on a Virtual COM Port using the Atmel Xplained Pro Embedded Debugger Virtual COM Port interface.
- Know how to display graphs on Atmel Data Visualizer using the Atmel Xplained Pro Embedded Debugger DGI Interface.

6. Appendix: Project Solution

```c
#include "atmel_start.h"
#include "atmel_start_pins.h"
#include "temperature_sensor_main.h"

struct io_descriptor *uart_edbg_io;
struct io_descriptor *spi_edbg_io;
volatile bool conversion_done = false;
uint8_t ADC_buffer[2];
float temperature;
uint8_t DataStream_buf[5] = {0x03, 0x00, 0x00, 0x00, 0xFC};

void UART_EDBG_init(void)
{
        usart_sync_get_io_descriptor(&USART_0, &uart_edbg_io);
        usart_sync_enable(&USART_0);

        io_write(uart_edbg_io, (uint8_t *)"Hello World!", 12);
}

static void convert_cb_ADC(const struct adc_async_descriptor *const descr)
{
        conversion_done = true;
}

/**
 * Example of using ADC_0 to generate waveform.
 */
void ADC_light_init(void)
{
        adc_async_register_callback(&ADC_0, ADC_ASYNC_CONVERT_CB, convert_cb_ADC);
        adc_async_enable(&ADC_0);
}

void SPI_DGI_init(void)
{
        spi_m_sync_get_io_descriptor(&SPI_0, &spi_edbg_io);

        spi_m_sync_enable(&SPI_0);
        io_write(uart_edbg_io, (uint8_t *)"\nSPI Init\n", 10);
}

int main(void)
{
        system_init();

        UART_EDBG_init();

        SPI_DGI_init();

        ADC_light_init();
        io_write(uart_edbg_io, (uint8_t *)"ADC Init\n",9);

        temperature_sensors_init();
        io_write(uart_edbg_io, (uint8_t *)"I2C Init\n", 9);
        /* Replace with your application code */
        while(1) {
                delay_ms(1000);

                temperature = at30tse75x_read(TEMPERATURE_SENSOR_0);

                adc_async_start_conversion(&ADC_0);

                while(!conversion_done);
                conversion_done = false;

                adc_async_read(&ADC_0, ADC_buffer, 2);

                DataStream_buf[1] = (uint8_t)temperature;
                DataStream_buf[2] = ADC_buffer[0];
                DataStream_buf[3] = ADC_buffer[1];

                io_write(spi_edbg_io, DataStream_buf, 5);
        }
}
```

7. Appendix: Atmel Studio 7 Help Viewer

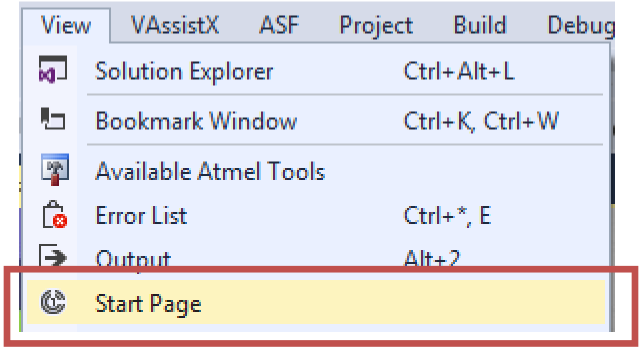
We will see some of the new Atmel Studio 7 features which will be helpful while developing the application.

It is possible to download the microcontroller data sheet /user guides /application notes within Atmel Studio 7 and also it is possible to check the relative information, register information while implementing the application.

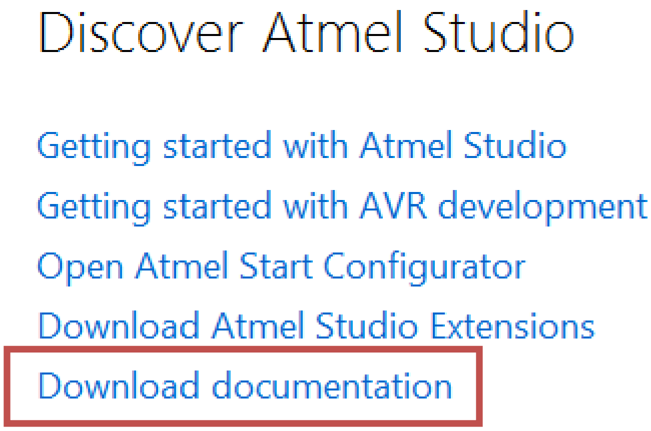
This feature makes it very easy to take a quick overview of datasheet /Application notes/user guides topics.

TO DO Get SAM L21 Documentation

- From the menu select View -> Start page.



- Click on ‘Download documentation’ to launch the Atmel Studio 7 Help Viewer:



Discover Atmel Studio

Getting started with Atmel Studio

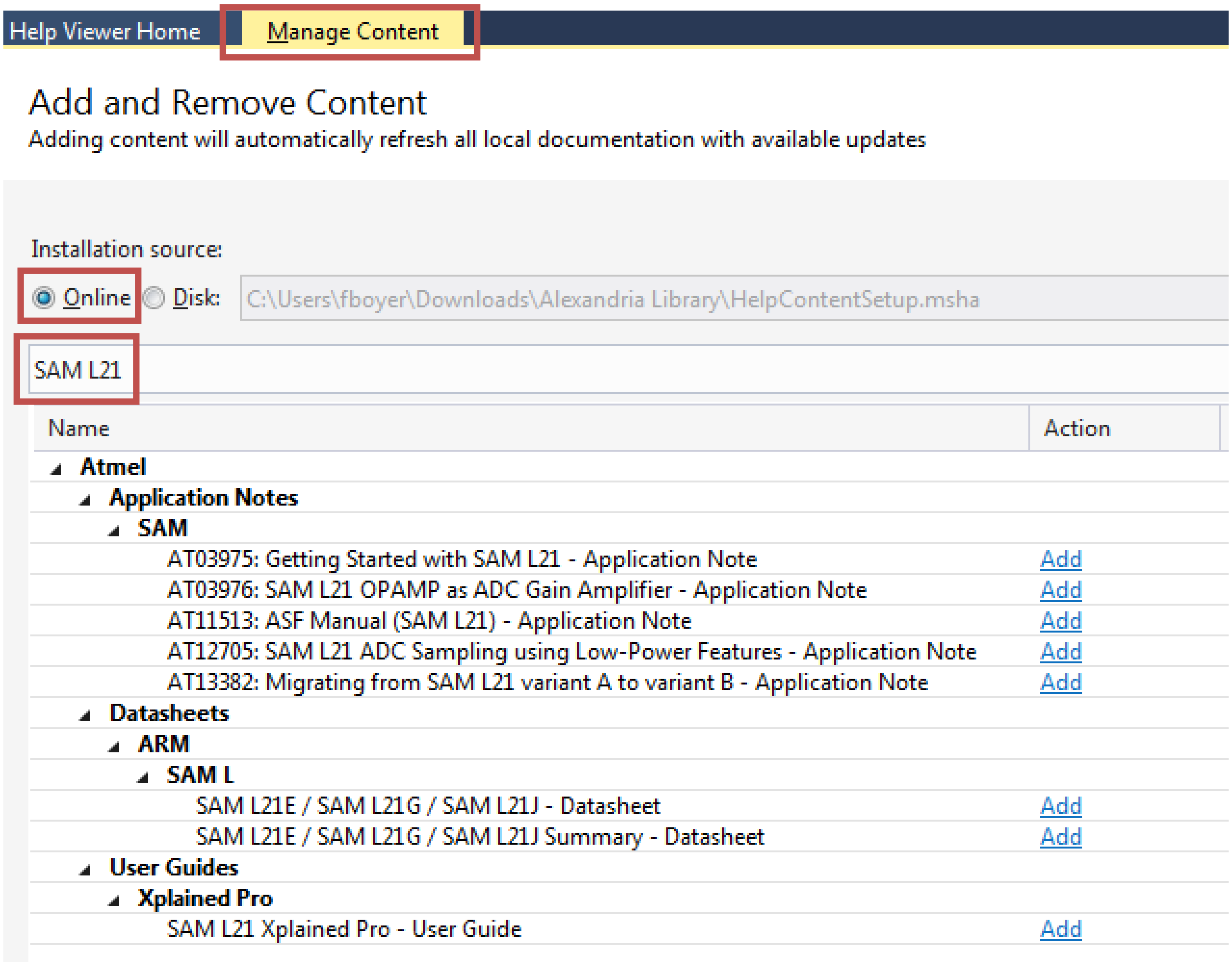
Getting started with AVR development

Open Atmel Start Configurator

Download Atmel Studio Extensions

Download documentation

- In the ‘Manage Content’ tab, select ‘Online’ and type ‘SAM L21’ in the search box:



Add and Remove Content

Adding content will automatically refresh all local documentation with available updates

Installation source:

● Online ○ Disk: C:\Users\fboyer\Downloads\Alexandria Library\HelpContentSetup.msha

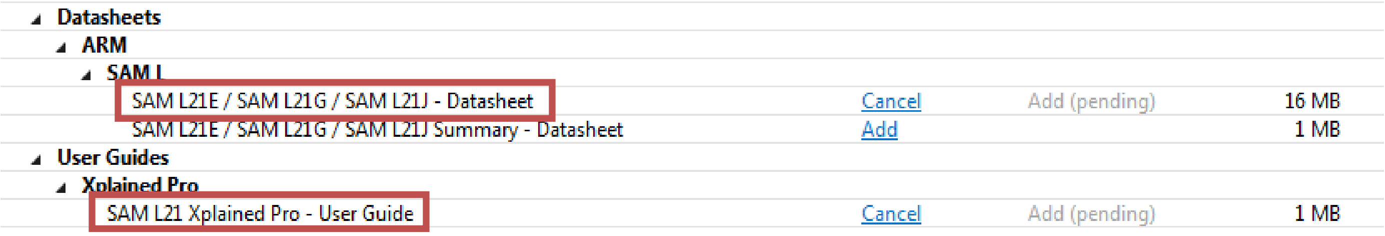
SAM L21

Name	Action
Atmel	
Application Notes	
SAM	
AT03975: Getting Started with SAM L21 - Application Note	Add
AT03976: SAM L21 OPAMP as ADC Gain Amplifier - Application Note	Add
AT11513: ASF Manual (SAM L21) - Application Note	Add
AT12705: SAM L21 ADC Sampling using Low-Power Features - Application Note	Add
AT13382: Migrating from SAM L21 variant A to variant B - Application Note	Add
Datasheets	
ARM	
SAM L	
SAM L21E / SAM L21G / SAM L21J - Datasheet	Add
SAM L21E / SAM L21G / SAM L21J Summary - Datasheet	Add
User Guides	
Xplained Pro	
SAM L21 Xplained Pro - User Guide	Add

☑ **RESULT** All the Atmel SAM L21 documentation is listed.

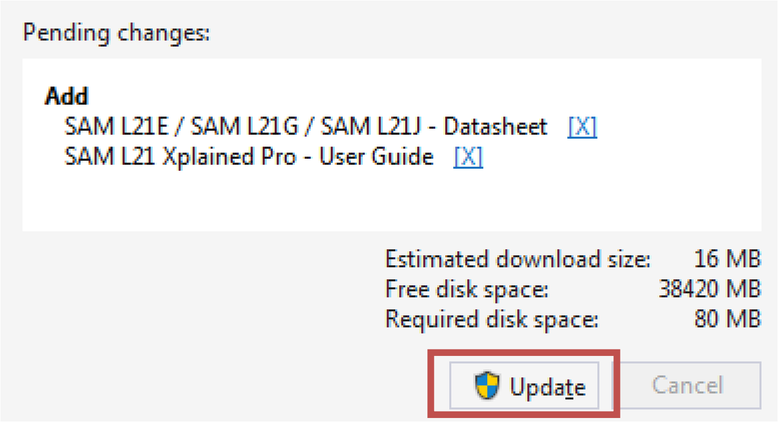
TO DO Download the SAM L21 Datasheet / User Guide

- Select SAM L21 Datasheet and User Guide documents by clicking on their ‘Add’ link:



◢ Datasheets			
◢ ARM			
◢ SAM L			
SAM L21E / SAM L21G / SAM L21J - Datasheet	Cancel	Add (pending)	16 MB
SAM L21E / SAM L21G / SAM L21J Summary - Datasheet	Add		1 MB
◢ User Guides			
◢ Xplained Pro			
SAM L21 Xplained Pro - User Guide	Cancel	Add (pending)	1 MB

- Click on the ‘Update’ button



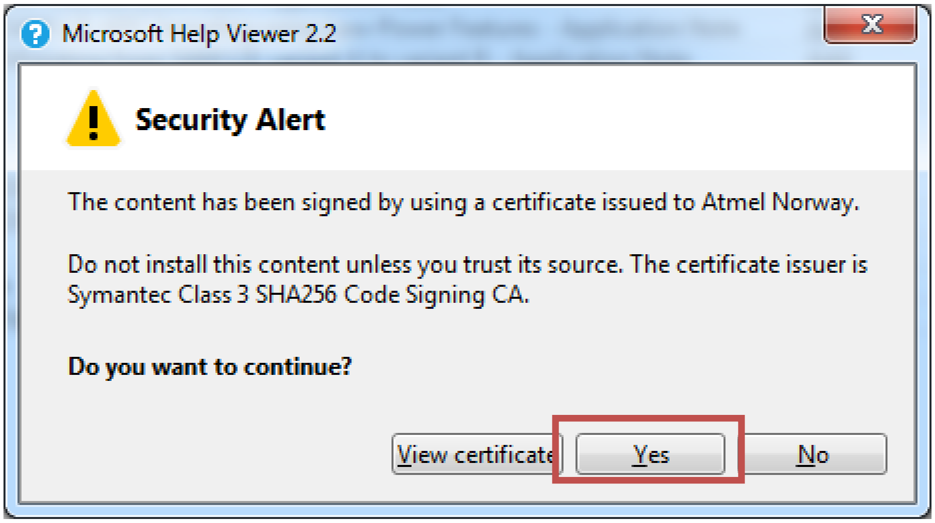
Pending changes:

Add
SAM L21E / SAM L21G / SAM L21J - Datasheet [X]
SAM L21 Xplained Pro - User Guide [X]

Estimated download size: 16 MB
Free disk space: 38420 MB
Required disk space: 80 MB

INFO Downloading is started and download completion is indicated by the progress bar at the right bottom corner.

- ‘Security Alert’ message box will be displayed. Click ‘Yes’



Microsoft Help Viewer 2.2

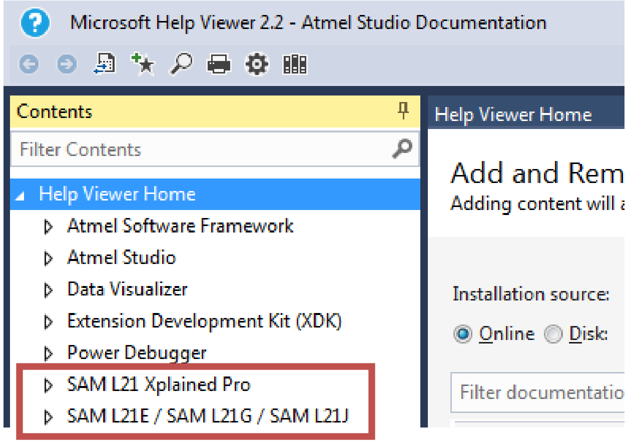
Security Alert

The content has been signed by using a certificate issued to Atmel Norway.

Do not install this content unless you trust its source. The certificate issuer is Symantec Class 3 SHA256 Code Signing CA.

Do you want to continue?

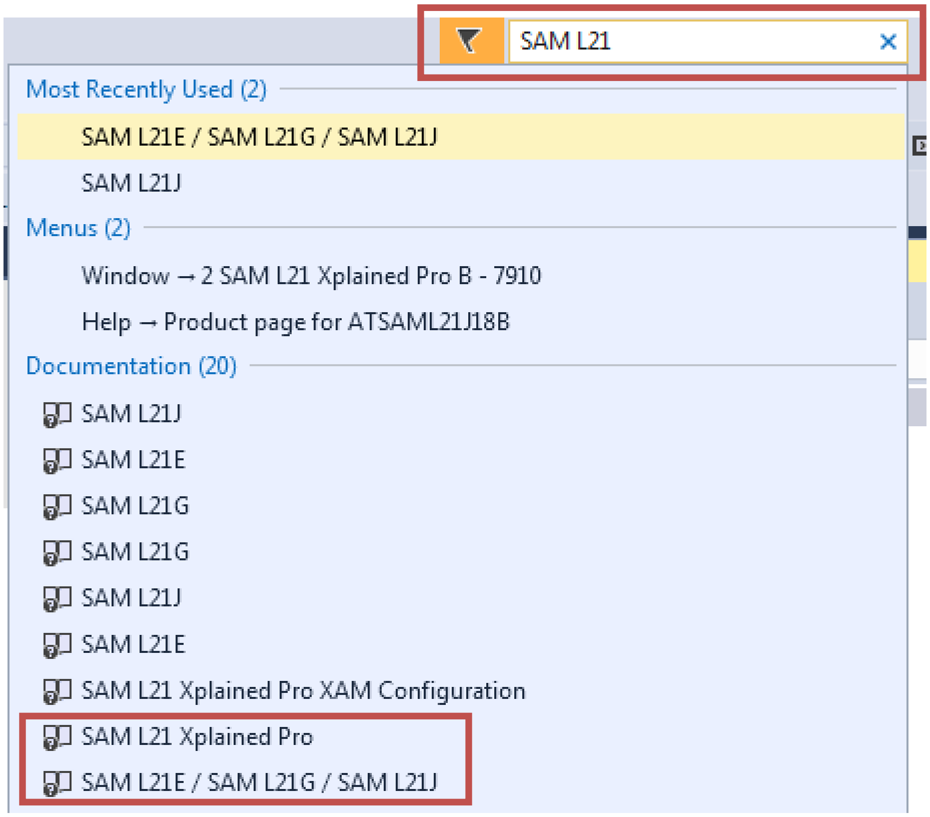
✓ **RESULT** The downloaded documentations are listed under ‘Contents’ tab



TO DO Use Offline Documentation

There are different ways to quickly access the downloaded documentation:

- Click on Help -> View Help or ‘Ctrl+F1’
- You can also use the ‘QuickLaunch’ box in Atmel Studio 7 at the right up corner:
  - Type ‘SAM L21’ in QuickLaunch search box
  - You can now select the desired documentation:



SAM L21

Most Recently Used (2)

SAM L21E / SAM L21G / SAM L21J

SAM L21J

Menus (2)

Window → 2 SAM L21 Xplained Pro B - 7910

Help → Product page for ATSAML21J18B

Documentation (20)

SAM L21J

SAM L21E

SAM L21G

SAM L21G

SAM L21J

SAM L21E

SAM L21 Xplained Pro XAM Configuration

SAM L21 Xplained Pro

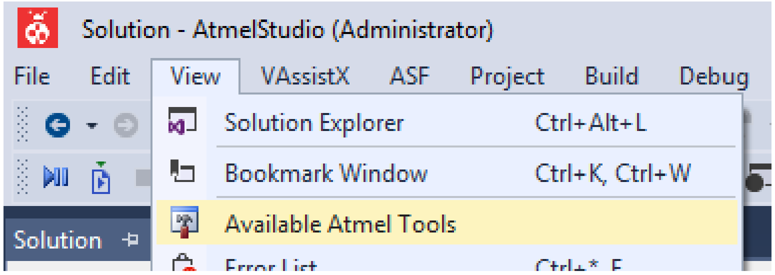
SAM L21E / SAM L21G / SAM L21J

RESULT You can now use the opened documentation

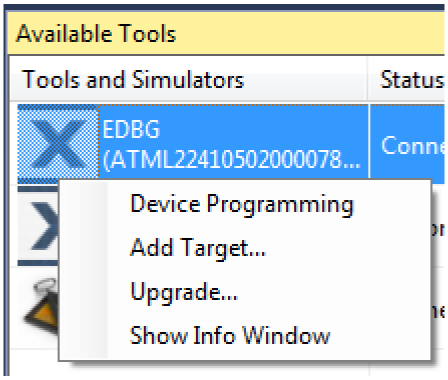
8. Appendix: Upgrade Embedded Debugger (EDBG) Firmware

TO DO Check & Upgrade EDBG Firmware Update

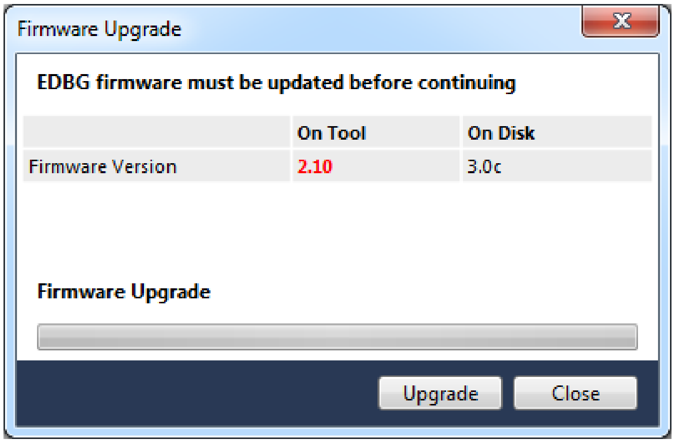
- Open Atmel Studio 7
- Connect your board to your computer using dedicated DEBUG USB connector
- Select View > Available Atmel Tools


Solution - AtmelStudio (Administrator)
File
Edit
View
VAssistX
ASF
Project
Build
Debug
Solution Explorer Ctrl+Alt+L
Bookmark Window Ctrl+K, Ctrl+W
Available Atmel Tools
Solution
Error List Ctrl+*, E

- Select your EDBG firmware and right click on it to select Upgrade…



- If the EDBG firmware is NOT up to date, please click on Upgrade



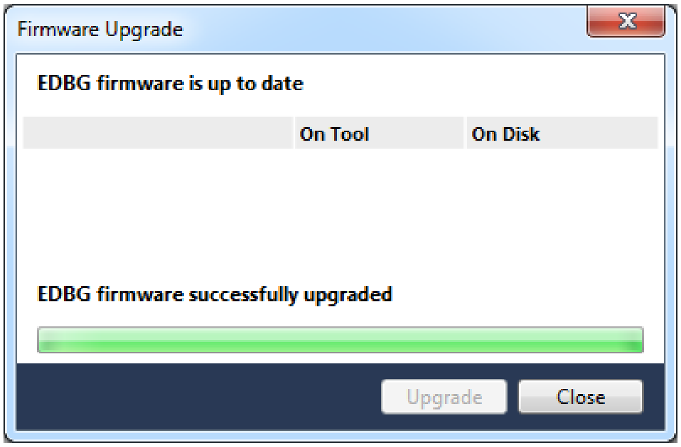
EDBG firmware must be updated before continuing

	On Tool	On Disk
Firmware Version	2.10	3.0c

Firmware Upgrade

RESULT Your EDBG Firmware is upgraded

OK



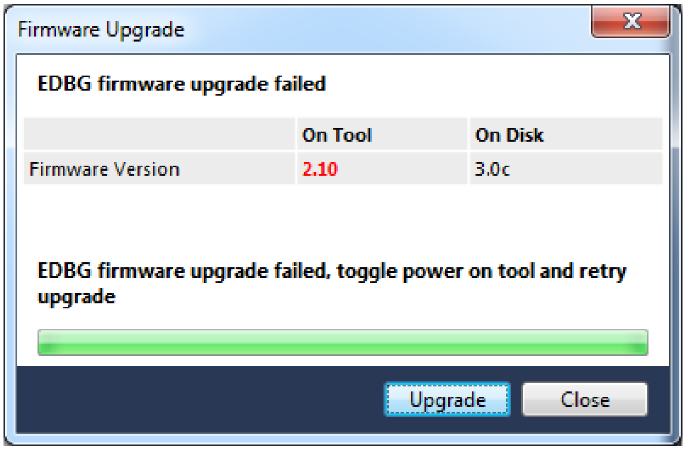
EDBG firmware is up to date

	On Tool	On Disk

EDBG firmware successfully upgraded

WARNING Check you have the above message as it may happen a power down/up of the board may be required so that the process completes.

NOT OK



EDBG firmware upgrade failed

	On Tool	On Disk
Firmware Version	2.10	3.0c

EDBG firmware upgrade failed, toggle power on tool and retry upgrade

9. Revision History

Doc. Rev.	Date	Comments
-	07/2016	Updates to comply with latest Atmel Studio/Start improvements (build 1006)
-	06/2016	Updates to comply with latest Atmel Start improvements (June 2016 release)
-	03/2016	Minor Updates following first trainings / Update Appendix (Help Viewer)
-	02/2016	Initial document release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA T: (+1)(408) 441.0311 F: (+1)(408) 436.4200 | **www.atmel.com**

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death ("Safety-Critical Applications") without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.