**ASF ARM Tutorial Introduction**

Created on: 19 April 2016

**Part 1 of the ASF ARM Tutorial**

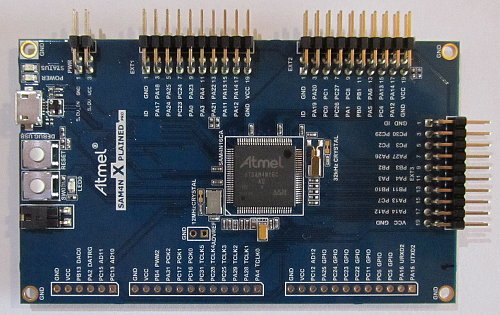
This tutorial shows how to use the ASF or Atmel Software Framework in Atmel Studio 7 for ARM Cortex microcontrollers.

Atmel Software Framework is a collection of software components for Atmel Flash microcontrollers written in the C programming language. ASF includes software drivers, libraries and example projects that make development for Atmel microcontrollers easier. Embedded developers do not need to start a microcontroller software project completely from scratch and create driver functions and other utilities, but can use ASF drivers and utilities which are already written and tested.

**Hardware used in this ASF ARM Tutorial**

Any Atmel ARM Cortex microcontroller board can be used to follow this tutorial, even a custom board. ASF is available for AVR and ARM microcontrollers; [a complete list of supported microcontrollers](http://asf.atmel.com/docs/latest/) can be found on the Atmel website.

All ASF projects in this tutorial series have been developed and tested using Atmel Studio 7 and are run on an [Atmel SAM4N Xplained Pro board (part number ATSAM4N-XPRO)](http://www.atmel.com/tools/atsam4n-xpro.aspx) which contains an [ATSAM4N16C ARM Cortex-M4](http://www.atmel.com/devices/ATSAM4N16C.aspx) based microcontroller. If you are using a different board, you will need to make minor changes to the examples when selecting the board and microcontroller in each project.



**Atmel SAM4N Xplained Pro Evaluation Board**

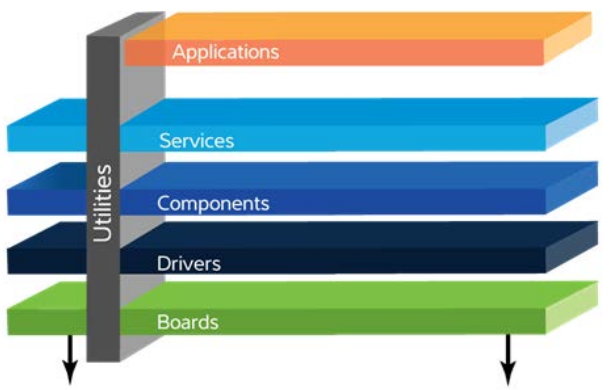
**Getting the Atmel Software Framework**

ASF is installed with [Atmel Studio](http://www.atmel.com/tools/ATMELSTUDIO.aspx), so there is no need to install it separately. This tutorial uses ASF in Atmel Studio only.

[ASF is available as a standalone package](http://www.atmel.com/tools/AVRSOFTWAREFRAMEWORK.aspx), but do not download this package for use with this tutorial, just install Atmel Studio.

**ASF Structure**

The ASF is usually shown arranged in layers as illustrated in the following image.



**ASF Layers Arrangement**

A brief explanation of each layer follows, but will make more sense once they are used in a project.

* **Applications** – user application or example applications that use the underlying ASF modules.
* **Services** – software services such as USB classes, FAT file system, DSP library, etc.
* **Components** – drivers that access **external** hardware.
* **Drivers** – low-level drivers that access on-chip microcontroller hardware.
* **Boards** – hardware definitions for the board being used, e.g. clock crystal frequency, pin definitions for UART, LEDs, etc.
* **Utilities** – linker script files, common files for the build system, general usage defines, macros and functions.

**Next in this Tutorial Series**

In the next part of this tutorial, a simple ASF ARM example project is created in Atmel Studio. The project is a very simple embedded application that blinks a single LED.

# ASF LED Blink Tutorial on Atmel Xplained Pro ARM Board

Created on: 19 April 2016

**Part 2 of the ASF ARM Tutorial**

In the second part of the Atmel Software Framework tutorial, a new ASF project is created for running on a SAM4N Xplained Pro board. Creating an ASF project for an Atmel board is an easy first step in using the ASF because the board hardware is already defined in the new project.

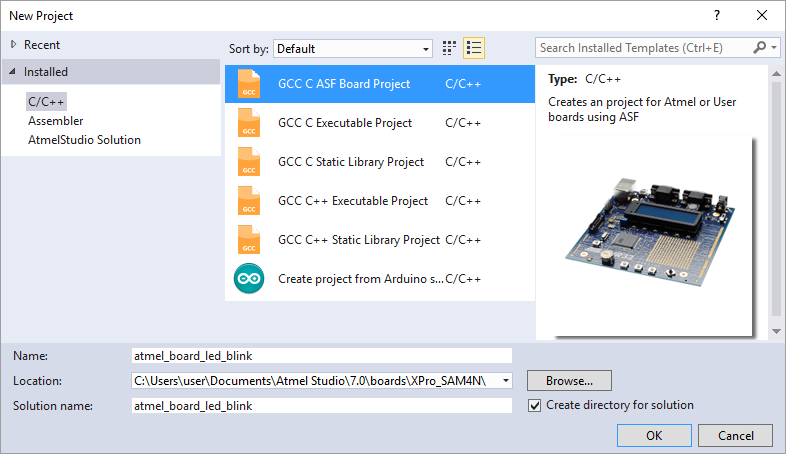
The next part of this tutorial series shows how to use ASF on a custom board with a user board template. In this case the underlying hardware must be defined by the user.

The following tutorial steps show how to create a new Atmel Studio ASF project, add the necessary ASF module and write the code to blink an LED.

## 1. Create a New Atmel Studio ASF Project

Create a new Atmel Studio project by selecting **New Project...** from the Start Page, or by selecting **File → New → Project...** from the top menu.

In the New Project dialog box, select **GCC C ASF Board Project**, select the location for the project and give the project a name. I am calling this project **atmel\_board\_led\_blink**.

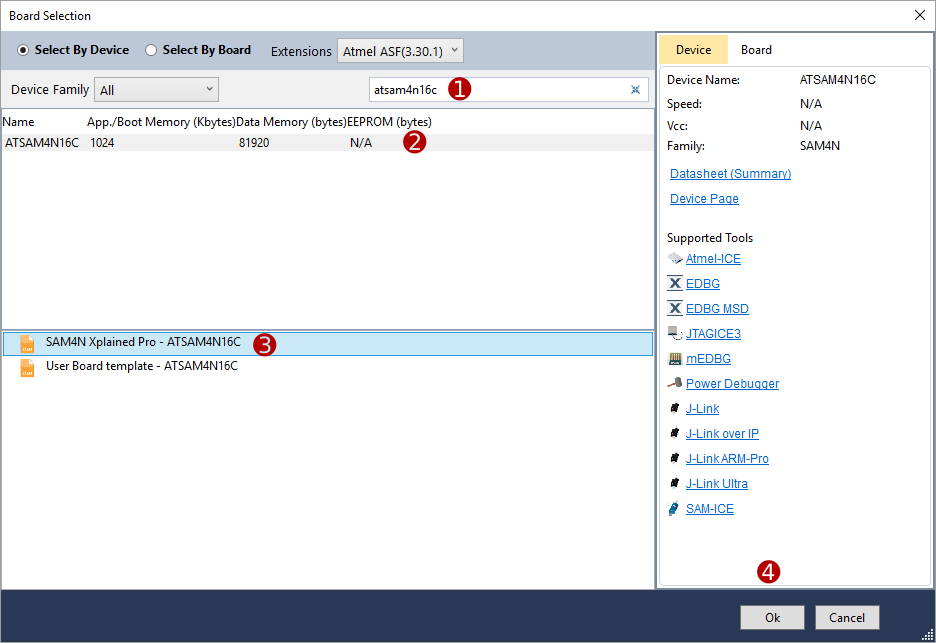
[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/new-asf-board-project.png)

**Creating a New GCC C ASF Board Project**

In the Board Selection dialog box, search for the microcontroller name that is on your board. Once the name has been filtered, it can be selected. Below the filtered device name, select the board for the device. For example, the ATSAM4N16C has **SAM4N Xplained Pro - ATSAM4N16C** as the available board option.

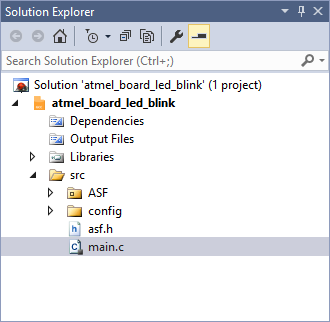
In this part of the tutorial, an Atmel board must be selected and not the "User Board template", which will be covered in the next part of this tutorial.

An alternative method for searching for the board is to select the **Select By Board** radio button at the top of the Board Selection dialog box. Click **Ok** when the board is selected.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/asf-board-selection.png)

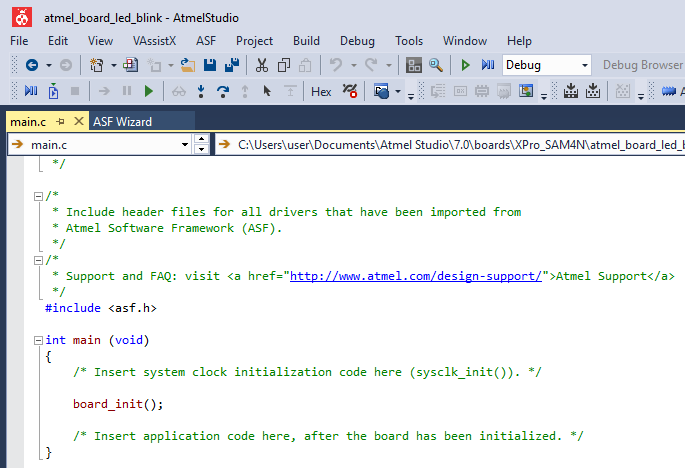
**ASF Project Board Selection**

The new ASF Atmel board project has now been created. If **main.c** can not be seen in the Solution Explorer pane, click **src** to expand it. Double click main.c to open it. If Solution Explorer is not visible, click **View → Solution Explorer** from the top menu.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/atmel-studio-solution-explorer.png)

**Atmel Studio Solution Explorer**

Some skeleton code can be seen inside main.c after scrolling past the comment block at the top of the file. This is where we will write our LED blink application.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/asf-skeleton-code.png)

**ASF Skeleton Code**

## 2. Add the ASF Delay Routines Service

To blink or flash an LED on and off, we need to be able to toggle the LED state and call a delay routine each time the LED is toggled. Default ASF settings allow us to access the on-board LED, so we only need to add support for a delay routine.

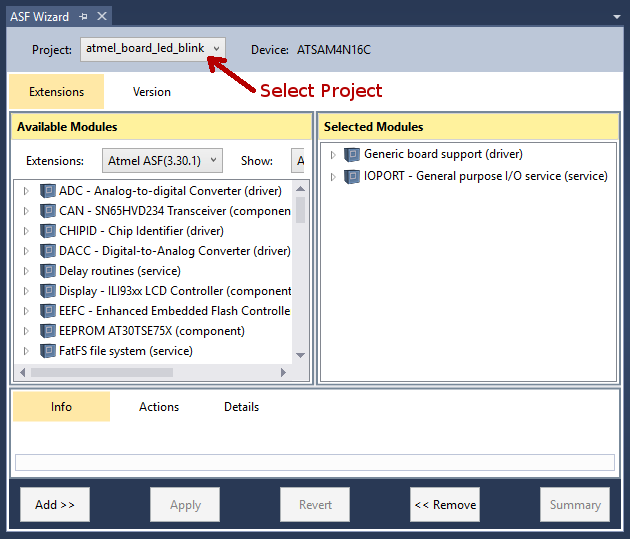
ASF delay routines are available as a module and are added to the Atmel Studio project using the ASF Wizard.

### 2.1 Open the ASF Wizard

Open the ASF Wizard by selecting **ASF → ASF Wizard** from the top menu or by clicking the ASF Wizard icon on the top toolbar.

### 2.2 Select the Project

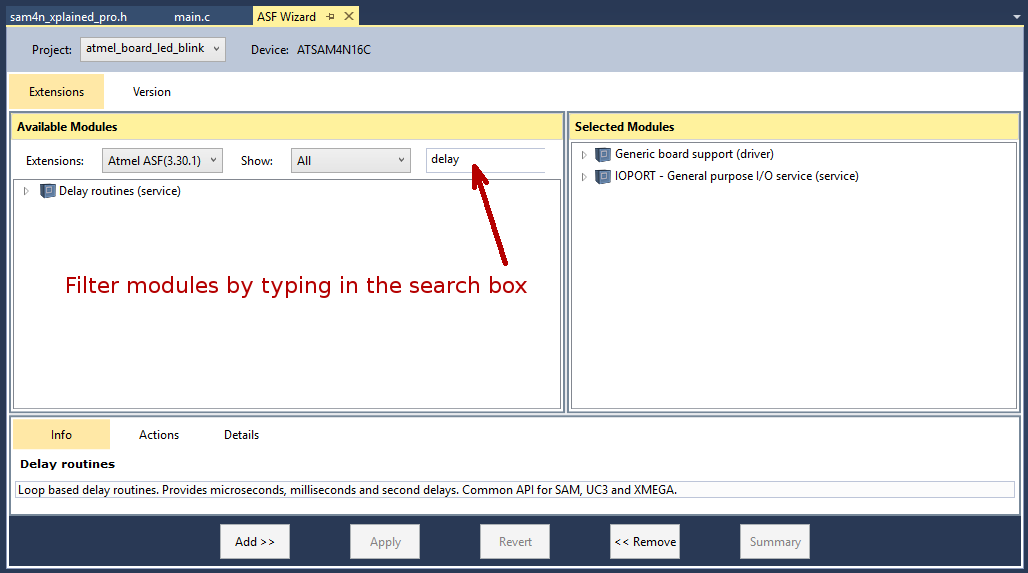
In the ASF Wizard, select the current project in the Project drop-down box if it is not selected by default.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/asf-wizard.png)

**ASF Wizard in Atmel Studio**

### 2.3 Add an ASF Module

Filter the available modules by typing **delay** in the search box in the ASF Wizard under the Available Modules pane on the left. **Delay routines (service)** will now be displayed in the left pane of ASF Wizard.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/filtering-asf-wizard-modules.png)

**Filtering Modules in ASF Wizard**

Click **Delay routines (service)** in the left ASF Wizard pane to select it and then click the **Add >>** button at the bottom.

Now click the **Apply** button to confirm the added module. A dialog box will pop up that contains a summary of the changes. Click **OK** to close the dialog box. The delay routines module has now been added to the project.

## 3. Add the Application C Code to the Project

Open the **main.c** file in Atmel Studio so that the application code can be added to the project.

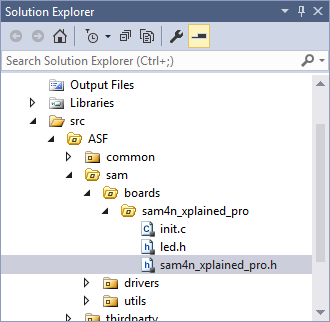
### 3.1 Find the LED Name

Before continuing, we need to find out what the name of the on-board LED has been defined as in the code. The SAM4N Xplained Pro board has an LED marked as **LED0** on the board silkscreen. This is the LED that we want to blink. If you are using a different board, choose an LED on the board that you want to blink and find out what the name of the LED is by looking at the board silkscreen or consulting the user guide for the board.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/led0-sam4n-xplained-board.jpg)

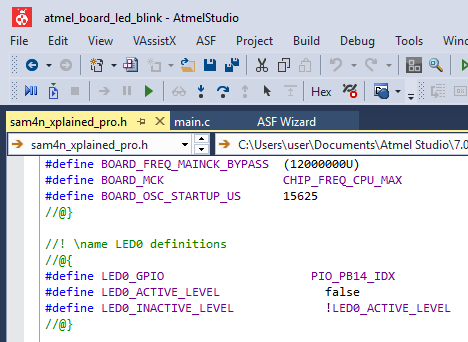
**LED0 on the SAM4N Xplained Pro Board**

Use Solution Explorer in Atmel Studio to find the header file for the board that you are using. The header file can be found in **src\ASF\sam\boards\<board\_name>\<board\_name>.h** For the SAM4N Xplained Pro board, the name of the board folder is **sam4n\_xplained\_pro** and the name of the board file is **sam4n\_xplained\_pro.h**

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/solution-explorer-board-file.png)

**Board Files in Solution Explorer**

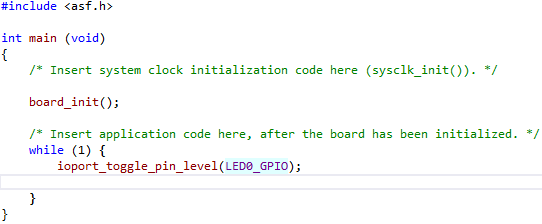
Open the board header file and scroll down to find where the LEDs are defined. As can be seen in the image below, LED0 is defined as **LED0\_GPIO** for the SAM4N Xplained Pro board. Select this name and copy it so that we can use it in main.c.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/board-led-definition.png)

**LED0 Definition in the Board Header File**

### 3.2 Add the ASF Pin Toggle Function

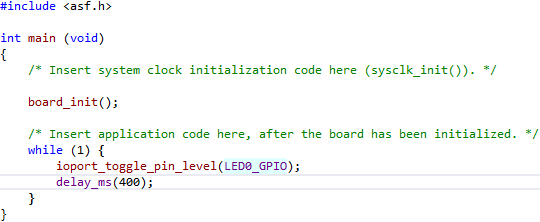
In main.c, create a **while(1)** loop below the board\_init() function. Call a new function in the while(1) loop called **ioport\_toggle\_pin\_level()** and pass the LED name that you copied to it as shown below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/asf-pin-toggle-function.png)

**ASF Pin Toggle Function Called in the Atmel Studio Project**

### 3.3 Add the ASF Delay Routine

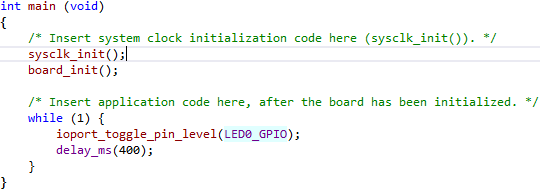
Call the **delay\_ms()** function below the pin toggle function. Pass the delay in milliseconds to the delay\_ms() function. In the code shown below the function is set to generate a 400ms delay.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/asf-delay-function.png)

**ASF delay\_ms() Function Added to the Code**

### 3.4 Initialize the System Clock

The ASF project is nearly finished. There is only one thing to add to make it work properly and that is to call a function that initializes the system clock. Call **sysclk\_init()** before the call to board\_init().

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/asf-system-clock-initialization.png)

**ASF Project System Clock Initialization**

The finished code is presented below in text format for easier copying.

**#include <asf.h>**

**int main (void)**

**{**

**/\* Insert system clock initialization code here (sysclk\_init()). \*/**

**sysclk\_init();**

**board\_init();**

**/\* Insert application code here, after the board has been initialized. \*/**

**while (1) {**

**ioport\_toggle\_pin\_level(LED0\_GPIO);**

**delay\_ms(400);**

**}**

**}**

## 4. Build and Test the Project

The project can now be built and loaded to the board. The LED should toggle state every 400ms as shown in the video below.

Can't see the video? [View on YouTube →](https://youtu.be/bC3gAWYefKw)

## Next in this Tutorial Series

In the next part of this tutorial series an ASF project for a custom or user defined board is created. It is not necessary to have a custom board to follow the tutorial. The same Atmel board can be used in the next part of the tutorial, but when the ASF project is created there won't be any hardware names defined which would be the case with a custom board. The tutorial shows how to use ASF with a custom board.

# ASF Tutorial using ASF on a Custom or User Board

Created on: 20 April 2016

**Part 3 of the ASF ARM Tutorial**

How to create an ASF project in Atmel Studio 7 for a custom or user board.

In the [previous part of this tutorial](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/), we used an Atmel ARM Cortex SAM4N Xplained board and created a new ASF project for this specific board. This is fine for initial evaluation of a microcontroller and the ASF. Development is easy because the hardware on the board is defined in the Atmel Studio ASF project and initialized in the C code.

Most software will be developed using a custom board for a specific project or product rather than for an evaluation board. In this case it is necessary to know how to initialize the hardware and customise an ASF project for use with a custom board. This part of the tutorial shows how to create and use an ASF project in Atmel Studio 7 for a custom board.

## Custom Board Hardware

It is not necessary to actually have a custom board to see how to create an Atmel Studio ASF user board project. The same evaluation board that was used in the previous part of the tutorial can be used again, even with a user board project.

A user board project is created in Atmel Studio instead of a project for a particular Atmel evaluation board or kit. This part of the tutorial will continue to use the Atmel SAM4N Xplained Pro board. The difference is that a user board ASF template will be selected instead of a SAM4N Xplained Pro ASF template during the creation of a new project in Atmel Studio. The hardware in the project will not be pre-defined and will have to be defined by the user.

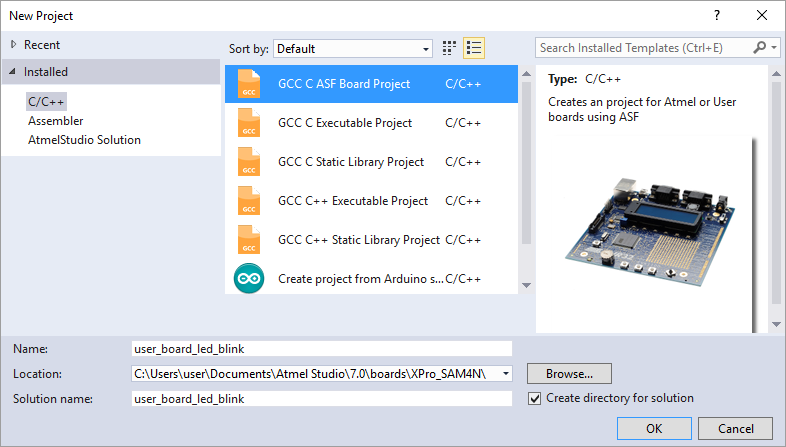
## Creating a User Board ASF Project in Atmel Studio

The following steps show how to create a new user board template ASF project in Atmel Studio, add ASF modules to the project and create a blinking LED C program using ASF functions.

### 1. Create a New ASF Project

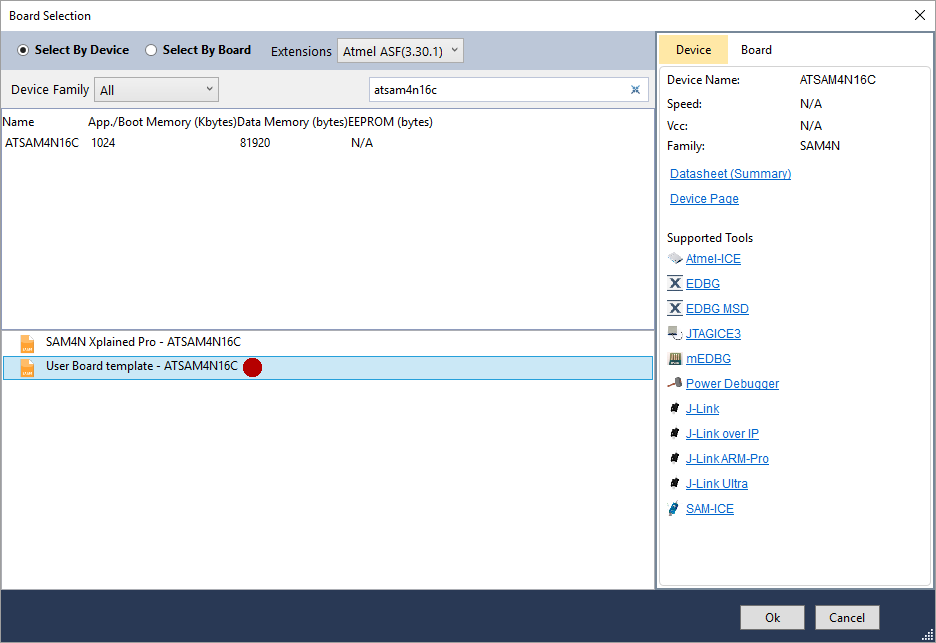
Create a new ASF project as done in the previous part of this tutorial, but select **User Board template** in the Board Selection dialog box. The following images show how to create a user board template for the ATSAM4N16C ARM microcontroller.

Start by creating a new project – I am calling this one **user\_board\_led\_blink**.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/new-asf-project.png)

**Create a New ASF Board Project**

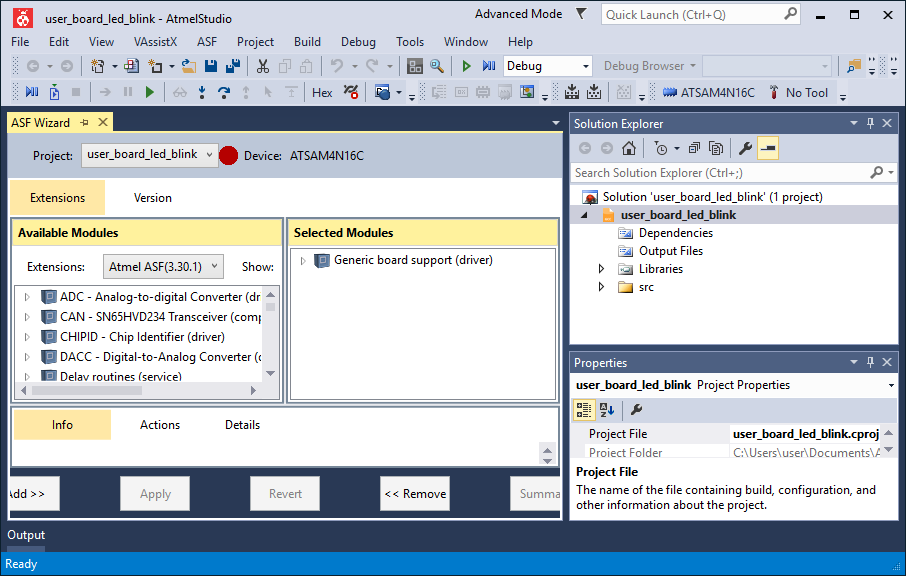
Select **User Board template** after filtering for the particular microcontroller on the board using the search field at the top of the dialog box.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/asf-user-board-template.png)

**Create a New User Board Template ASF Project**

### 2. Add the ASF Modules

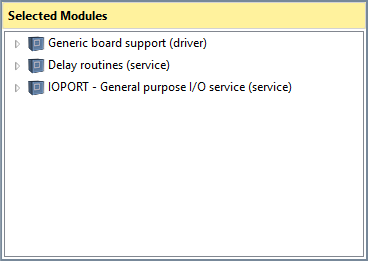
In the ASF Wizard for the new project in Atmel Studio, select the project name at the top of the ASF Wizard dialog box. If ASF Wizard is not open, open it by selecting **ASF → ASF Wizard** from the top menu.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/asf-wizard-select-project.png)

**Select the Current Project in the Atmel Studio ASF Wizard**

Notice that only the Generic board support driver module is automatically included in the user board project. In the Atmel board project in the previous part of this tutorial, the IOPORT module was automatically included.

Add the **IOPORT** service module and the **Delay routines** service module by searching for them in the Available Modules pane of ASF Wizard. The following image shows these two modules added to the project. Remember to click the **Apply** button after adding the two modules.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/asf-modules-added-to-project.png)

**ASF Modules Added to the Project in ASF Wizard**

### 3. Add the C Code

Now that the ASF modules needed by the project have been added, we can write the C application code. Open the **main.c** file using the Solution Explorer pane.

#### 3.1 Write the Application Code

Add the following code to main.c

**#include <asf.h>**

**int main (void)**

**{**

**/\* Insert system clock initialization code here (sysclk\_init()). \*/**

**sysclk\_init();**

**board\_init();**

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**ioport\_set\_pin\_dir(LED0, IOPORT\_DIR\_OUTPUT); // LED pin set as output**

**ioport\_set\_pin\_level(LED0, IOPORT\_PIN\_LEVEL\_HIGH); // switch LED off**

**/\* Insert application code here, after the board has been initialized. \*/**

**while (1) {**

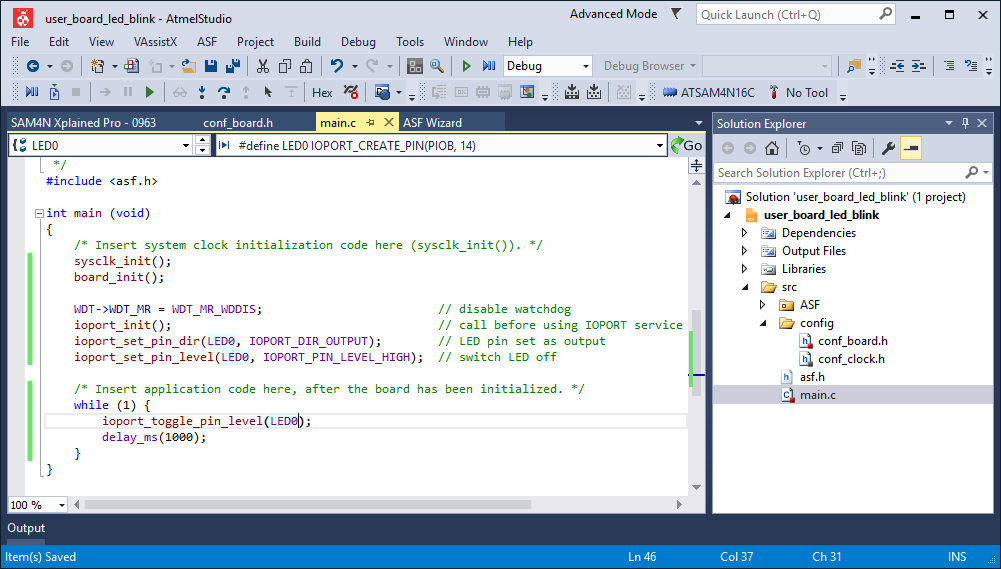
**ioport\_toggle\_pin\_level(LED0);**

**delay\_ms(1000);**

**}**

**}**

The above code is shown here in Atmel Studio.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/asf-project-c-code.png)

**Adding C Application Code to the ASF Project**

The code should mostly look familiar as it flashes an LED in the same way as the code from the previous part of this tutorial did. The difference is the code between the board\_init() function and the top of the while(1) loop.

#### 3.2 Hardware Initialization

In the previous part of this tutorial, the ASF project was created for a specific board, so the hardware definitions for the board and initialization of the hardware were automatically added to the project when it was created. As this project is a user board project, the board hardware is not defined or initialized.

The following code disables the watchdog timer, initializes the IOPORT ASF module, sets the pin that LED 0 is on as an output pin and finally switches the LED off.

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**ioport\_set\_pin\_dir(LED0, IOPORT\_DIR\_OUTPUT); // LED pin set as output**

**ioport\_set\_pin\_level(LED0, IOPORT\_PIN\_LEVEL\_HIGH); // switch LED off**

These tasks and other board initialization were automatically added to the board\_init() function in the previous project, so we could start using the hardware immediately. The above code should also have been put into the board\_init() function for this project, but was left in main() this time to make the project simpler. Currently board\_init() is an empty function in this project.

**Using a different board:**  
If you are using a different board to the SAM4N Xplained Pro, then open the previous project and navigate to and open **src\ASF\sam\boards\<board\_name>\init.c** to find the initialization code for your board in **board\_init()**. Copy and modify the necessary initialization functions from this file for your user board project. If the board is an Atmel ARM Cortex board, then nothing should need to be changed for this simple project, except maybe the LED name which is called LED0, but has not yet been defined for this project.

#### 3.3 Define the Hardware

In the case of a user board project we still have to define the hardware. This is done in the **conf\_board.h**file which can be found using the Atmel Studio Solution Explorer. Use the Solution Explorer to navigate to and open this file found in **src\config\conf\_board.h** and add the following code to the file.

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

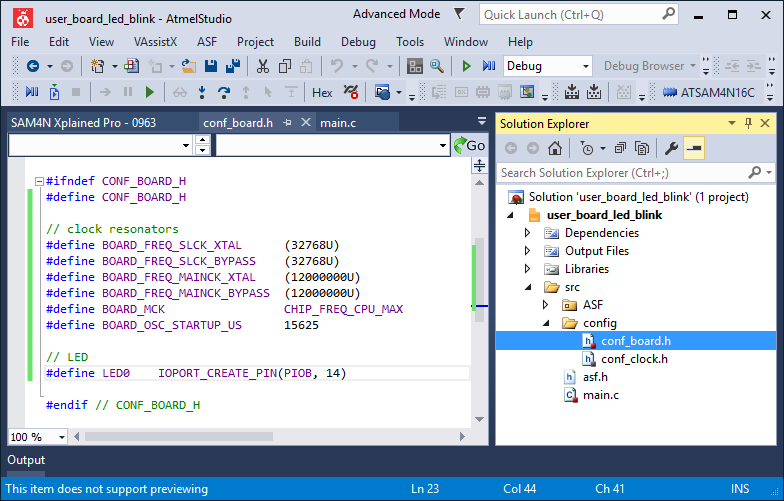
**#define BOARD\_OSC\_STARTUP\_US 15625**

**// LED**

**#define LED0 IOPORT\_CREATE\_PIN(PIOB, 14)**

**#endif // CONF\_BOARD\_H**

The code is shown in the file in Atmel Studio below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/asf-conf-board-file.png)

**ASF Project conf\_board.h File**

The above code defines the clock resonator values and the pin that LED 0 is on. The LED 0 pin is defined as **LED0** which is used in the application code in main().

**Using a different board:**  
If you are using a different board than the SAM4N Xplained Pro, then open the previous project and this file from the previous project: **src\ASF\sam\boards\<board\_name>\<board\_name>.h** to find the clock resonator settings and LED definition for your board.

You will also need to find which port and pin number the LED that you want to use is on. The following line of code shows that the LED is on pin 14 of PORT B.

**#define LED0 IOPORT\_CREATE\_PIN(PIOB, 14)**

### 4. Build and Test the Project

As can be seen from this part of the tutorial, we had to add hardware definitions for the board and add hardware initialization code because this project is based on a user board ASF template.

Once the above code has been added, the project can be built and loaded to the board for testing. It works the same as the previous project, but the LED is toggled at 1 second intervals instead of at 400ms intervals.

## Still to Come in this Tutorial Series

We have seen two ways of creating ASF projects in Atmel Studio. Not much has been said about the ASF modules and functions from these modules. A comparison between Atmel board ASF projects and user board ASF projects is still to be done in this tutorial series. ASF modules and functions all also be explained in more detail.

# Adding ASF to an Existing Atmel Studio Project

Created on: 23 April 2016

**Part 4 of the ASF ARM Tutorial**

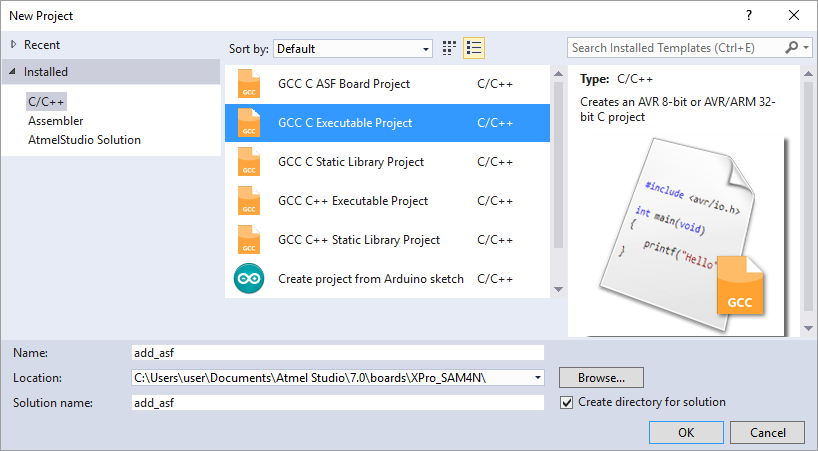
ASF can be added to an existing Atmel Studio project that was not started as an Atmel Software Framework project.

So far in this tutorial series we have looked at [creating a new Atmel board ASF project](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/) and [creating a new user board ASF project](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/). A third scenario is adding ASF to a project that has already been created and has some code in it.

In this part of the ASF tutorial series we will create a new Atmel Studio project without ASF and write some application code for it. We will then change the project into an ASF project. A simple LED blink application will be used again.

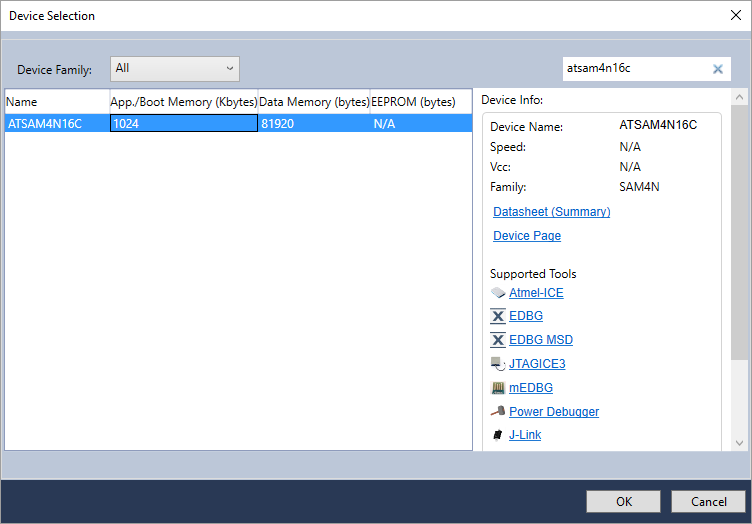
## Example Project without ASF

Start by creating a new Atmel Studio project, but this time make a **GCC C Executable Project** as shown in the image below. I am calling this project **add\_asf**.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/new-gcc-c-executable-project.png)

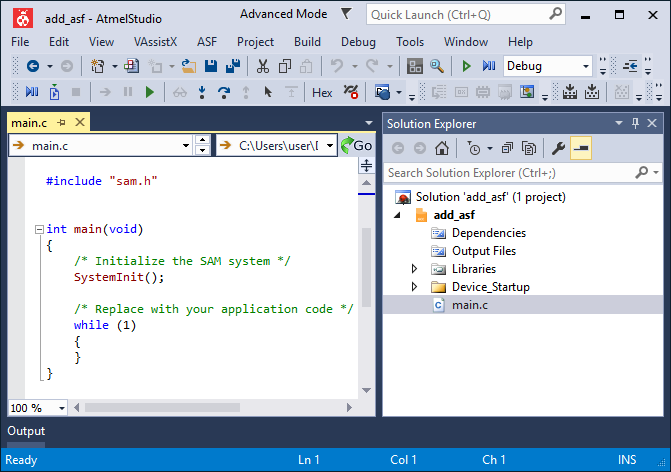
**Create a New GCC C Executable Project**

In the Device Selection dialog box, filter for the device on the board and select the device. This tutorial uses the Atmel SAM4N Xplained Pro again which has an ATSAM4N16C device on the board. This is the device selected in the image below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/device-selection.png)

**Selecting a Device for the New Project**

The new project in Atmel Studio opens with some skeleton code in main.c as shown below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/new-gcc-c-executable-project-atmel-studio.png)

**New Atmel Studio GCC C Executable Project with Skeleton Code in main.c**

Modify the skeleton code in main.c as follows. This is our application code.

**#include "sam.h"**

**int main (void)**

**{**

**SystemInit();**

**REG\_PIOB\_PER = PIO\_PER\_P14; // enable PIO to control pin PB14**

**REG\_PIOB\_OER = PIO\_OER\_P14; // enable PB14 as an output pin**

**while (1) {**

**REG\_PIOB\_CODR = PIO\_CODR\_P14; // switch LED on**

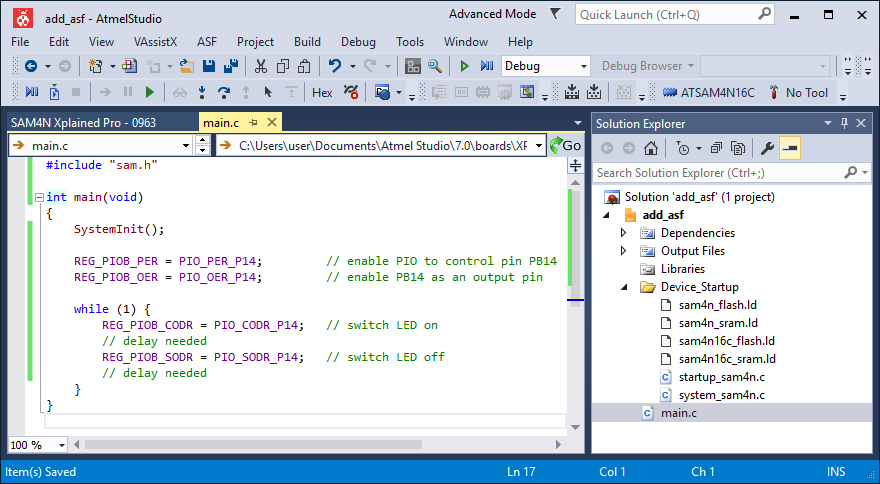
**// delay needed**

**REG\_PIOB\_SODR = PIO\_SODR\_P14; // switch LED off**

**// delay needed**

**}**

**}**

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/add-asf-atmel-studio-application-code.png)

**Application Code for the add\_asf Project in Atmel Studio without ASF**

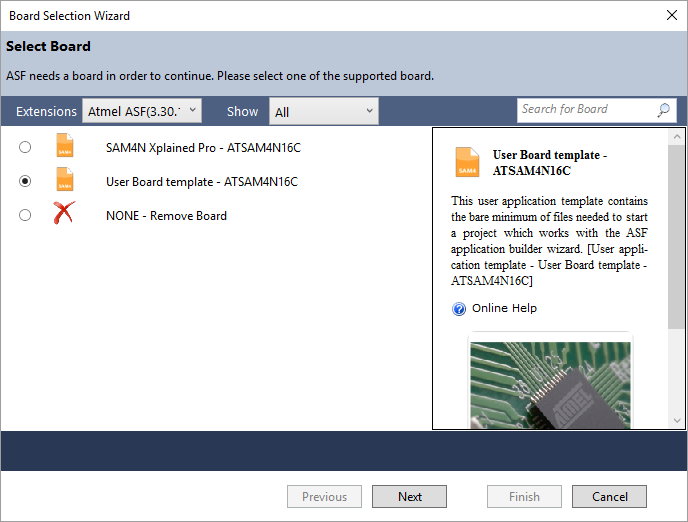
As can be seen in the above application code, pin 14 on PORT B is set as an output pin. This is the pin that LED0 is attached to on the Atmel SAM4N Xplained Pro board. The LED is then switched on and off to blink or flash it, but a delay function is missing from the code. Because we have not created an ASF project, the LED is accessed using the ARM microcontroller registers.

Let's say that we started this project and then at some point decided that we did not want to write our own delay function, but rather reuse some code that was already written to perform the delay. We decide to use the delay function from the ASF library and so need to convert the project to an ASF project.

## Adding ASF to the Project

The Board Selection Wizard is used to add ASF to a project in Atmel Studio. Select **ASF → Board Wizard** from the top menu bar in Atmel Studio to open the Board Selection Wizard dialog box.

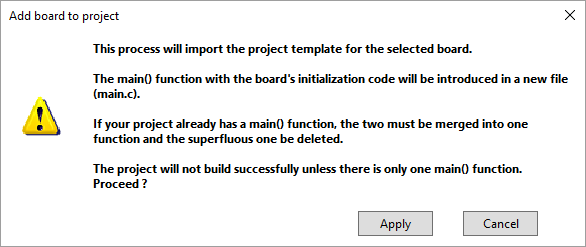
A board can now be selected in the Board Selection Wizard dialog box. For this project and microcontroller, the user board template for the ATSAM4N16C is selected as shown in the image below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/atmel-studio-board-selection-wizard-dialog-box.jpg)

**The Atmel Studio Board Selection Wizard Dialog Box**

After clicking the **Next** button in the dialog box, a summary of the files that will be added to the project is displayed.

A warning dialog box, as shown below, will be displayed after clicking the **Finish** button.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/add-board-warning-dialog-box.png)

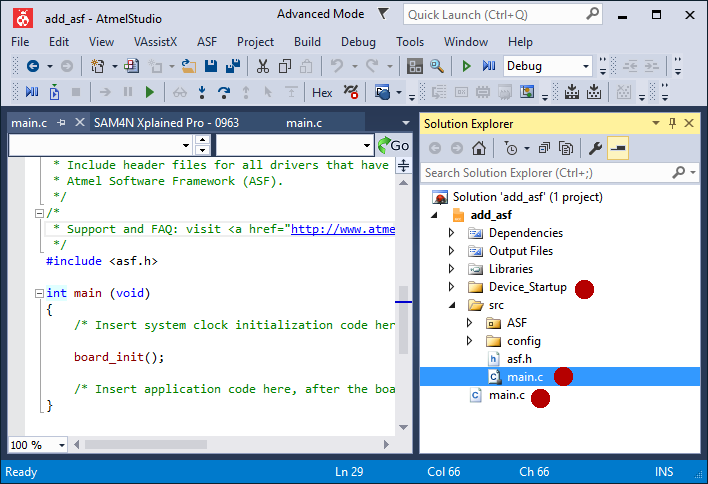
**The Warning Dialog Box Displayed After Adding a Board to the Project**

The dialog box warns that ASF has its own main() function and initialization code. This means that the project will not compile until the project has been modified.

## Modifying the ASF Project Code

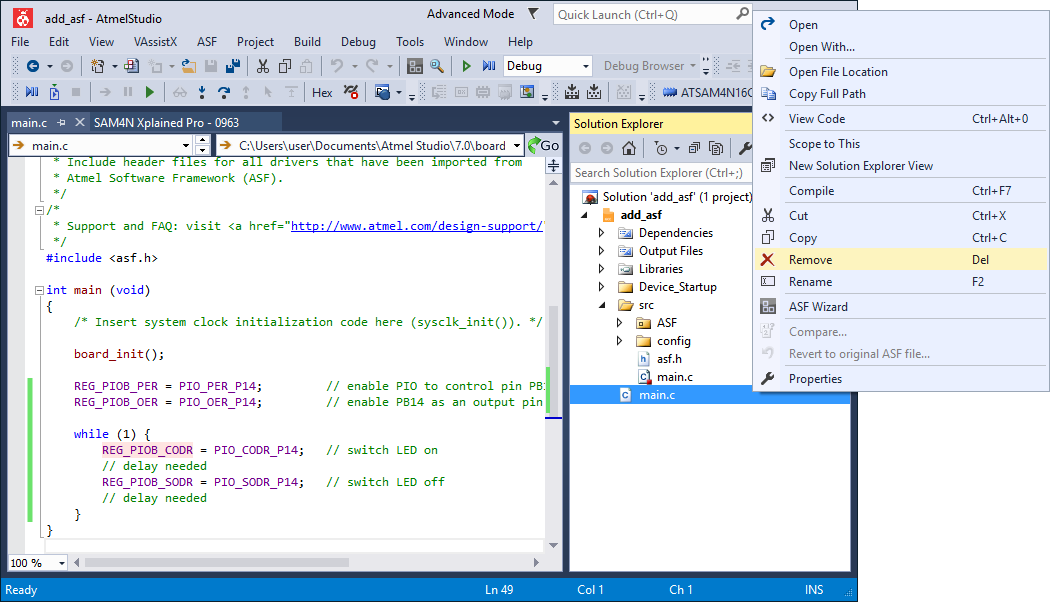
The project now has two main.c files each containing its own main() function. Code from the original main.c file must be copied to the new main.c file. The original main.c file must then be deleted. The original start-up code must also be deleted for the project to compile correctly.

The image below shows the project with ASF added. Marked with a red dot from top to bottom is the original start-up folder, the new main.c file and the old main.c file.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/project-with-asf-added.png)

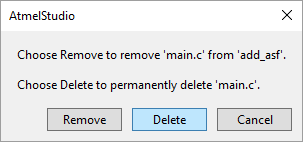
**Project with ASF Added**

Start by copying the code from the old main.c file to the new main.c file so that it looks as follows. Then right-click the original main.c file and delete it by selecting **Remove** from the pop-up menu.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/modify-main-c-code-file.png)

**Modify the Code in the main.c File**

After selecting to remove the file, the following dialog box will pop up.

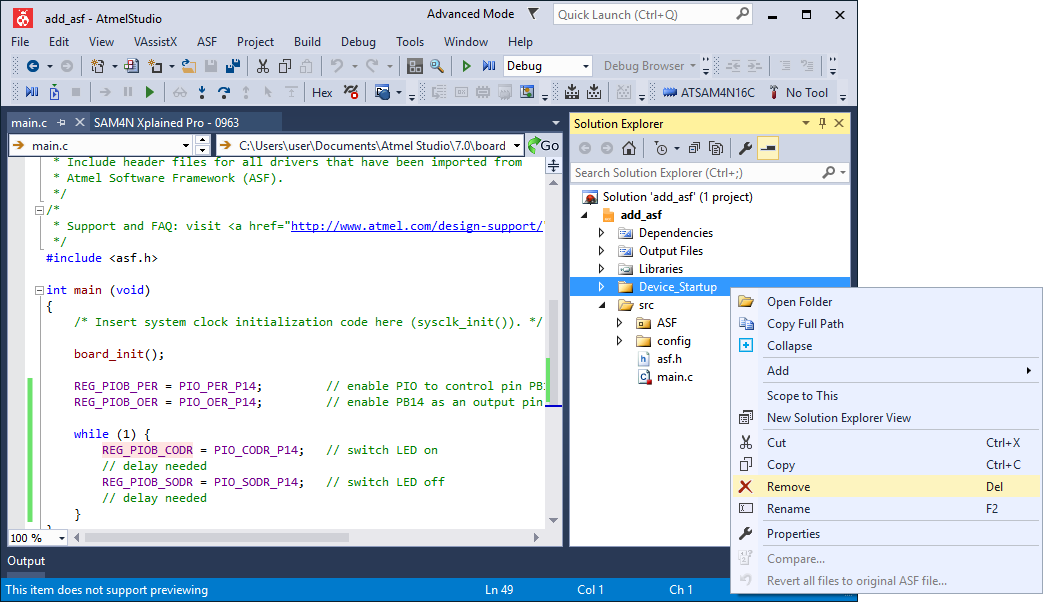
[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/remove-file-dialog-box.png)

**Remove File Dialog Box**

The file can either be removed from the project, but not deleted by clicking the **Remove** button; or the file can be removed from the project and deleted by selecting the **Delete** button.

The Device\_Startup folder and its contents must also be removed from the project for it to compile properly because ASF includes its own start-up functions which will clash with the original ones.

Right-click the **Device\_Startup** folder in Solution Explorer and select **Remove** on the pop-up menu as shown below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/removing-startup-files.png)

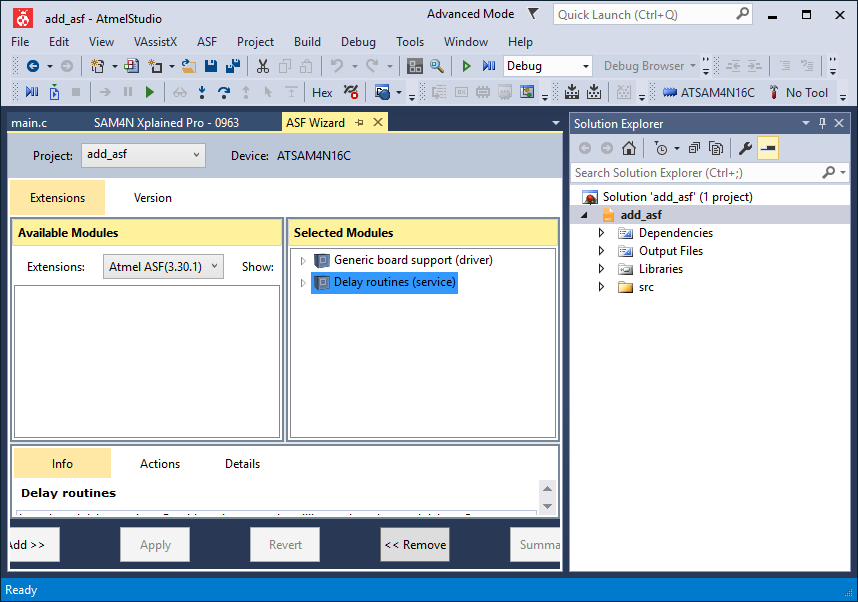
**Removing the Start-up Files by Deleting the Device\_Startup Folder**

At this stage the project should compile successfully, but we still need to add the delay function to the code.

## Adding the ASF Delay Module and Function

To use the delay function, we must first add the ASF delay module to the project using the ASF Wizard. This is done in the same way as was done in the [previous part of this tutorial](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/). In this case we only need to add the delay module and not the IOPORT module because our existing code operates the LED using registers.

Add the **Delay routines (service)** module to the project using the ASF Wizard.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/adding-asf-delay-module.png)

**Adding the ASF Delay Module to the Project**

The **delay\_ms()** function can now be used in the project. Add the timing delays to the code in main() as shown below.

**while (1) {**

**REG\_PIOB\_CODR = PIO\_CODR\_P14; // switch LED on**

**delay\_ms(500);**

**REG\_PIOB\_SODR = PIO\_SODR\_P14; // switch LED off**

**delay\_ms(500);**

**}**

We still need to set up the clock resonator values in **conf\_board.h** and call **sysclk\_init()** in main() in order for the delay function to work. The code is shown below with resonator settings for the SAM4N Xplained Pro board.

**conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

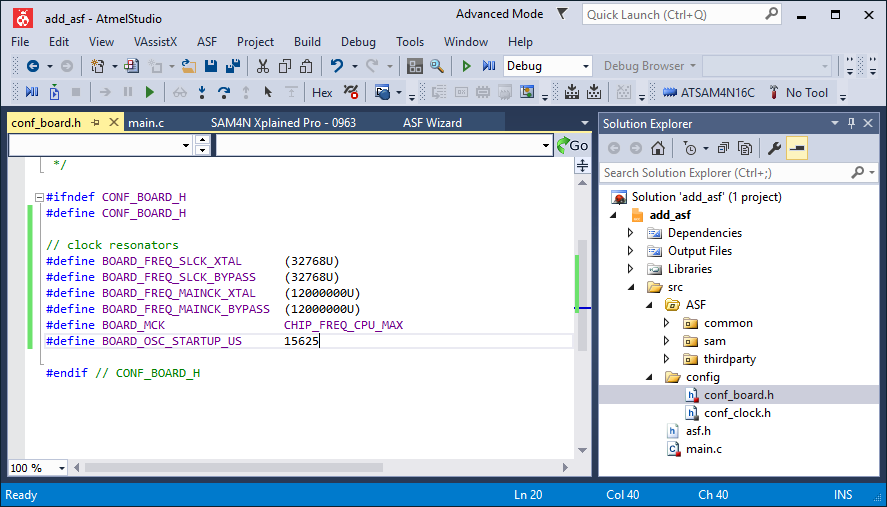
**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/conf-board.png)

**The conf\_board.h File with Clock Resonator Definitions Added**

**main.h**

**#include <asf.h>**

**int main (void)**

**{**

**/\* Insert system clock initialization code here (sysclk\_init()). \*/**

**sysclk\_init();**

**board\_init();**

**WDT->WDT\_MR = WDT\_MR\_WDDIS;**

**REG\_PIOB\_PER = PIO\_PER\_P14; // enable PIO to control pin PB14**

**REG\_PIOB\_OER = PIO\_OER\_P14; // enable PB14 as an output pin**

**while (1) {**

**REG\_PIOB\_CODR = PIO\_CODR\_P14; // switch LED on**

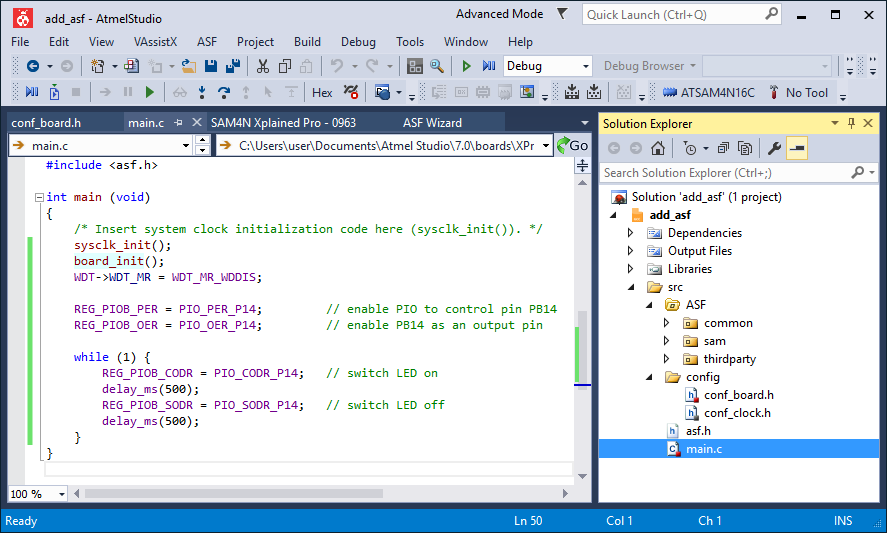
**delay\_ms(500);**

**REG\_PIOB\_SODR = PIO\_SODR\_P14; // switch LED off**

**delay\_ms(500);**

**}**

**}**

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adding-asf-to-project/main-file.png)

**The main.c File with Application Code Added**

The project should now compile and run just as the previous two examples did from the previous two parts of this tutorial.

# Comparing ASF Projects for Atmel Boards and User Boards

Created on: 27 April 2016

**Part 5 of the ASF ARM Tutorial**

Comparison between an ASF project made for an Atmel evaluation board and an ASF project made for a user or custom board.

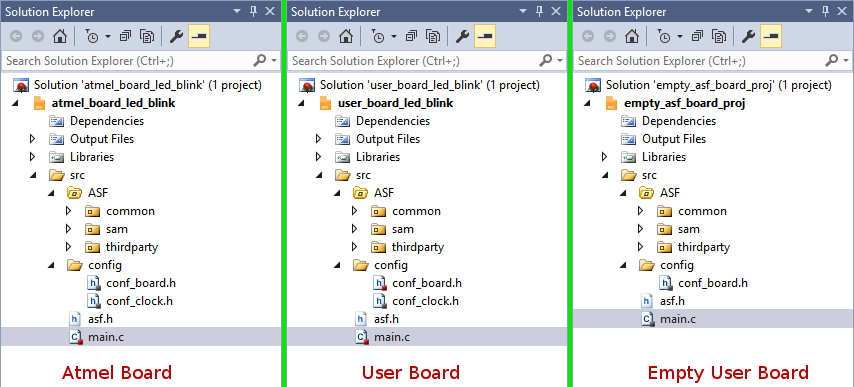
In this part of the Atmel Software Framework tutorial, we look at how the two different ASF project types are structured. It is shown which files need to be edited and where to add code and settings when starting a new ASF project.

By comparing the projects we will answer the following questions:

* What does the red dot or red square next to files in Atmel Studio Solution Explorer mean?
* What are the various folders and files in ASF projects for?
* Which files must be modified in a new ASF project?
* Into which ASF files must which code be placed?

## Basic ASF Files and Project Structure

A top level view of the structure of three different ASF projects is shown in the image below. The image shows the project files and folders in Atmel Studio Solution Explorer for the three projects. On the left is the ASF project made for the Atmel SAM4N Xplained Pro board from [part 2](https://startingelectronics.org/software/atmel/asf-arm-tutorial/atmel-board-blink-led/) of this tutorial series. In the middle is the custom or user board ASF project from [part 3](https://startingelectronics.org/software/atmel/asf-arm-tutorial/user-board-blink-led/) of this tutorial series. On the right is an new empty ASF user board project without any code added.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-project-compare/top-level-asf-compare.png)

**Top Level ASF Projects - Left: Atmel Board, Middle: User Board, Right: Empty User Board**

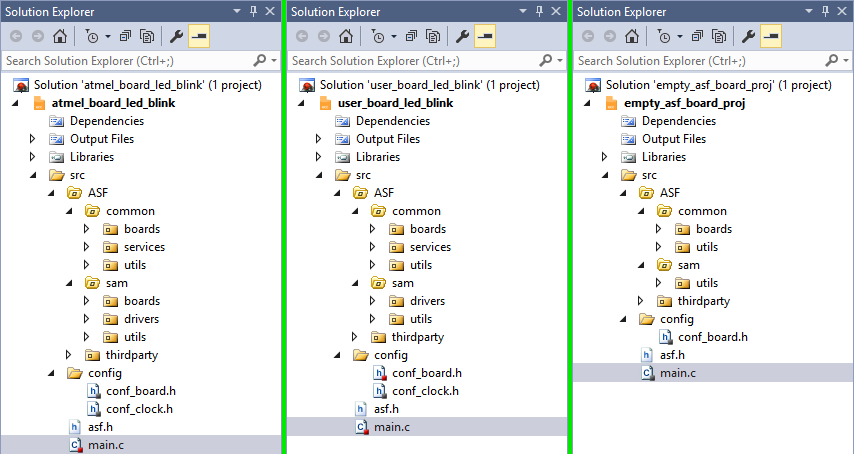
At the top level, the projects are very similar. The Atmel board project and user board project are identical except for the small red squares on some of the file icons. The red square marks files that have been modified by the user. The same small dot or square is grey on files that have not been modified by the user. If any code is added to main.c in the empty ASF user board project, the square on the main.c file icon will change from grey to red.

The first two projects both have a **conf\_clock.h** file, while the empty user board project does not. The reason for this is that the empty user board project has not had any ASF modules added. The conf\_clock.h file is added to the project when the **System Clock Control (service)** is added to the project using the ASF Wizard.

If you have been following this tutorial series and creating the projects, you may remember that we never added the System Clock Control service ASF module to any of the projects. This module is a dependency of the IOPORT ASF module and the Delay routines ASF module, so it is automatically added to the project if either of these two modules is added.

## ASF Project Structure Next Level

Expanding the **common** and **sam** folders of our three ASF projects shows more differences between the projects as shown in the below image.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-project-compare/level-2-asf-compare.png)

**Next Level into the Folders of ASF Projects**

### The sam Folder

The folder structure separates microcontroller specific low-level code into the **sam** folder for Atmel SAM ARM microcontrollers.

#### sam\boards\

The Atmel board ASF project at the left of the image has a **boards** folder inside the **sam** folder which the user board ASF projects do not have. This **sam\boards\** folder contains board specific initialization code and hardware definitions for the Atmel board chosen when creating the project.

#### sam\drivers\

The **sam\drivers\** folder contains driver code for drivers that were added by the ASF Wizard. This folder is not present in the empty user board ASF project because no ASF modules were added to the project.

#### sam\utils\

The **sam\utils\** folder contains various utility functions, files and definitions including microcontroller hardware register definitions, linker scripts, macros and files for building the project.

### The common Folder

#### common\boards\

The **common\boards\** folder contains a header file used to include the correct header file for the board or microcontroller used. In a user board project it also includes user board initialization code.

#### common\services\

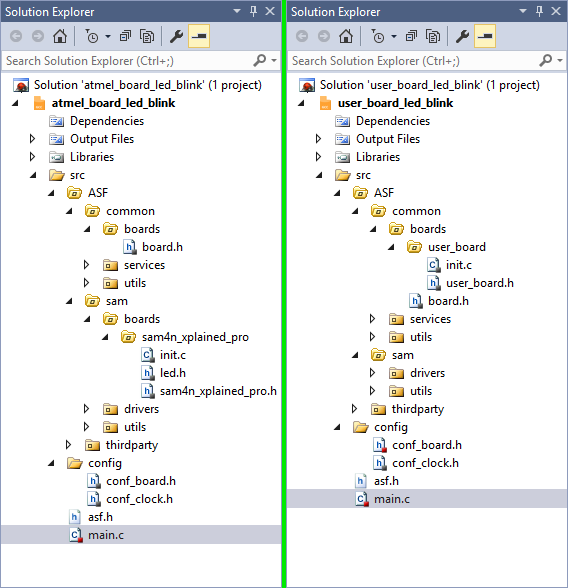
The **common\services\** folder contains ASF code for ASF service modules added by the ASF Wizard. Again this folder is not present in the empty user board ASF project because no ASF modules have been added to the project. Although only ASF service modules were added to the project in the previous parts of this tutorial, these services have driver dependencies. The needed driver(s) are added automatically by the ASF Wizard and the drivers folder created in the sam folder.

#### common\utils\

This folder contains Atmel part identification macros and interrupt controller utility functions.

## ASF Project Structure Configuration and Settings Files

The next level of folders have been expanded for the Atmel board and user board ASF projects to show where the files used for board configuration and settings are found as shown below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-project-compare/level-3-asf-compare.png)

**Atmel Board (left) and User Board (right) ASF Configuration Files**

First observe that the Atmel board project does not have a **user\_board** folder found in **common\boards\user\_board\** in the user board project. Similarly the user board project does not have a **sam\boards\** folder containing a folder named after an Atmel evaluation board.

These two folders are equivalents for the two different ASF project types.

### init.c

The **init.c** file contains the initialization code for the board in the **board\_init()** function which is called at the beginning of main(). This init.c file is found in **common\boards\user\_board\** for a user board ASF project. The same file is found in **\sam\boards\*<board\_name>*\** in Atmel board ASF projects.

In Atmel board projects the code in board\_init() initializes the hardware for the specific Atmel board chosen when creating the project. In a user board project, this code must be added by the user to initialize the specific hardware on the user board.

### Hardware Definition Header File

In Atmel board projects, hardware is defined in the header file in **\sam\boards\*<board\_name>*\** which has the name ***<board\_name>.h*** where board\_name is the name of the Atmel evaluation board selected when the project was created. In a user board project this information must be added to **user\_board.h**found in **common\boards\user\_board\**.

Atmel board projects include the external resenator or crystal settings for the board in this file.

### conf\_board.h

The **conf\_board.h** file found in **config\** is used for board specific configuration definitions.

### conf\_clock.h

This file is only available after adding the **System Clock Control service** ASF module or by adding an ASF module that depends on the system clock control module.

The system clock source, prescaler and PLL settings are changed in this file.

# Use External Crystals on Atmel ARM Cortex Microcontroller ASF Project

Created on: 16 May 2016

**Part 6 of the ASF ARM Tutorial**

Enable the internal oscillators of Atmel ARM Cortex microcontrollers to use external crystals for their timing in an ASF project.

The ATSAM4N16C microcontroller on the Atmel SAM4N Xplained Pro board has two external crystals connected to its oscillator pins. Default Atmel Studio ASF projects use the internal RC oscillators instead of the external crystals. This part of the ARM Cortex ASF tutorial shows how to enable and use the external crystals with the oscillators.

A 12MHz crystal and 32.768kHz crystal are used for the main clock and slow clock respectively. A 100MHz system clock is derived from the internal oscillator using the 12MHz crystal by feeding the 12MHz clock through the PLL. A 32.768kHz slow clock is derived from the internal oscillator using the 32.768kHz crystal.

## Creating the ASF Project

Start by creating a new user board ASF project as done in previous parts of this tutorial. I am calling this project **ext\_xtal**.

In Atmel Studio:

* Select **File → New → Project...** from the top menu.
* In the New Project dialog box choose **GCC C ASF Board Project** and name the new project **ext\_xtal**.
* In the Board Selection dialog box filter for the ARM microcontroller on the board. This is **ATSAM4N16C** for the SAM4N Xplained board.
* Still in the Board Selection dialog box, select the **User Board Template**. For the SAM4N Xplained board it is called **User Board Template - ATSAM4N16C**.
* In the new project, open main.c and add a while(1) loop at the bottom of the main() function.

If the new project is compiled and loaded to the board it will not use the external 12MHz crystal, but rather the internal RC oscillator. This can be verified by probing the crystal pins with an oscilloscope.

## Enabling the External Crystal to be used with the Main Clock Oscillator

The following steps show how to use the 12MHz external crystal instead of the internal RC oscillator for the main clock.

### 1. Add the System Clock Control ASF Module to the Project

Before ASF functions can be used to set up the clock source for a microcontroller, the System Clock Control service ASF module must be added to the project.

* Open the ASF Wizard using **ASF → ASF Wizard** from the top menu.
* Filter the available modules by searching for **clock**.
* Use the ASF Wizard to add the **System Clock Control (service)** module to the project.

### 2. Call sysclk\_init() in main()

The sysclk\_init() function becomes available after adding the System Clock Control ASF module. Call sysclk\_init() at the top of main().

**main.c**

**#include <asf.h>**

**int main (void)**

**{**

**sysclk\_init();**

**board\_init();**

**while(1) {**

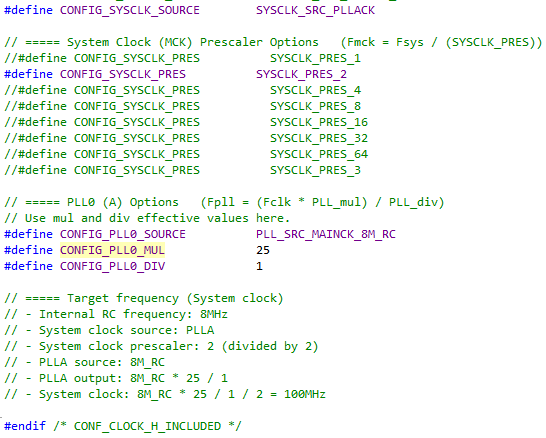
**}**

**}**

### 3. Set the Clock Source and Configuration

After adding the System Clock Control ASF module, **conf\_clock.h** is added to the project in **src\config\**. Open this file for editing.

Current system clock configuration can be found in conf\_clock.h and the current clock settings are shown in the comments at the bottom of the file. As can be seen in the image below, the system clock is set up to run at 100MHz using the internal 8MHz RC oscillator.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/external-crystals/asf-system-clock-settings.png)

**ASF System Clock Settings**

We still want to get the clock source from the PLL and use a prescaler value of two, so leave the first two definitions in conf\_clock.h at their default settings.

**CONFIG\_PLL0\_SOURCE** must change from PLL\_SRC\_MAINCK\_8M\_RC to **OSC\_MAINCK\_XTAL** to change from using the internal RC oscillator to using the crystal attached to the external oscillator pins.

On the SAM4N Xplained Pro, the main crystal is 12MHz. The values of the PLL multiplication and division factors must change in order to derive a 100MHz system clock from the 12MHz crystal. This is done by changing the division to 3 which divides 12MHz by 3 to get 4MHz. A multiplication factor or 50 is then used to change the PLL frequency to 4MHz × 50 = 200MHz. The prescaler division of 2 brings this frequency down to 100MHz.

The modified conf\_clock.h file is shown below with the commented out options removed and the comments at the bottom of the file updated.

**conf\_clock.h**

**#ifndef CONF\_CLOCK\_H\_INCLUDED**

**#define CONF\_CLOCK\_H\_INCLUDED**

**#define CONFIG\_SYSCLK\_SOURCE SYSCLK\_SRC\_PLLACK**

**// ===== System Clock (MCK) Prescaler Options (Fmck = Fsys / (SYSCLK\_PRES))**

**#define CONFIG\_SYSCLK\_PRES SYSCLK\_PRES\_2**

**// ===== PLL0 (A) Options (Fpll = (Fclk \* PLL\_mul) / PLL\_div)**

**// Use mul and div effective values here.**

**#define CONFIG\_PLL0\_SOURCE OSC\_MAINCK\_XTAL**

**#define CONFIG\_PLL0\_MUL 50**

**#define CONFIG\_PLL0\_DIV 3**

**// ===== Target frequency (System clock)**

**// - External Xtal frequency: 12MHz**

**// - System clock source: PLLA**

**// - System clock prescaler: 2 (divided by 2)**

**// - PLLA source: 12MHz**

**// - PLLA output: 12MHz \* 50 / 3**

**// - System clock: 12MHz \* 50 / 3 / 2 = 100MHz**

**#endif /\* CONF\_CLOCK\_H\_INCLUDED \*/**

**Note:** When writing the multiplication factor directly to the PLL configuration register, one would normally be subtracted from this value to make the multiplication work correctly. This is not necessary in the above settings because the function that uses this value subtracts one from it. The multiplication factor for this example can therefore be 50 and not 49 to get the PLL to multiply by 50.

### 4. Define the Crystal Values

Values for the crystals on the board must be defined in **conf\_board.h** as we have done in previous projects in this tutorial series. These values were defined in previous projects to prevent compiler warnings rather than use the external crystals. All previous projects used the internal 8MHz RC oscillator as the clock source which was ramped up to 100MHz by the PLL.

**conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

## Building the Project

The project can now be built and loaded to the board. An oscilloscope can be used to check the pins of the external 12MHz crystal to see that it is being used by the oscillator. A 12MHz clock should be seen on the oscilloscope.

## Enabling the Slow Clock Oscillator to use the External 32.768KHz Crystal

Two lines of code must be added to the project at the top of main() to enable the slow clock oscillator to use the external 32.768kHz crystal. The code is shown below.

**main.c**

**#include <asf.h>**

**int main (void)**

**{**

**pmc\_switch\_sclk\_to\_32kxtal(PMC\_OSC\_XTAL); // enable external 32.768kHz crystal**

**while (!pmc\_osc\_is\_ready\_32kxtal()); // wait until oscillator is ready**

**sysclk\_init();**

**board\_init();**

**while(1) {**

**}**

**}**

The first line of code enables the slow clock to use the 32.768kHz external crystal. The second line of code waits for the slow clock oscillator to stabilize.

The project can be built again and loaded to the board. An oscilloscope can be used to check the pins of the 32.768kHz crystal which should show a 32.768kHz clock pulse.

**Atmel Software Framework Checklist for ARM Cortex ASF C Projects**

Created on: 30 May 2016

**Part 7 of the ASF ARM Tutorial**

This checklist goes through the steps needed to make a new ASF project for Atmel ARM Cortex microcontrollers, which files to modify and how to add ASF modules to a project. The list is intended as a quick reference for starting new ASF projects.

Atmel Software Framework supported devices and latest documentation can be found on the [Atmel ASF website](http://asf.atmel.com/docs/latest/).

The following steps are also available as a handy [downloadable pdf file](https://github.com/startingelectronics/ASF-ARM-checklist/raw/master/ASF-ARM-checklist_v1.pdf) from [startingelectronics on GitHub](https://github.com/startingelectronics" \t "_blank). The pdf file can be downloaded, viewed or even printed and used as a reference whenever a new ASF ARM project is created.

**Create a New ASF Project**

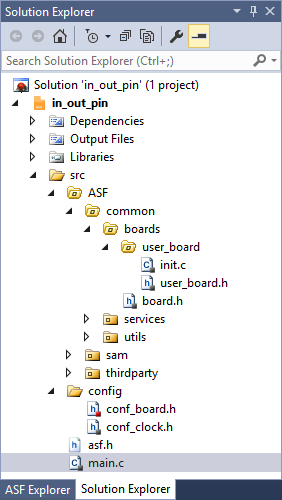
1. Open Atmel Studio and start a new project using **File → New → Project...** from the top menu bar or **Ctrl + Shift + N** on the keyboard.
2. In the New Project dialog box, select **GCC ASF Board Project**, give the project a name and select the project location.
3. In the Board Selection dialog box enter the ARM microcontroller part number in the search field and then select **User Board template** for the desired microcontroller.

**Select Clock Sources and Frequencies**

1. Open the ASF Wizard using **ASF → ASF Wizard** from the top menu bar or **Alt + W** on the keyboard.
2. Search for **clock** under Available Modules in the ASF Wizard. Select **System Clock Control (service)** and then click the **Add >>** button. Click the **Apply** button to add the selected module.
3. Use Solution Explorer (image below) to open **src → config → conf\_board.h** and add definitions for the clock resonators and oscillator start-up time, e.g.
4. **// clock resonators**
5. **#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**
6. **#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**
7. **#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**
8. **#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**
9. **#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**
10. **#define BOARD\_OSC\_STARTUP\_US 15625**

Without these definitions the compiler will issue several warnings.

1. Use Solution Explorer to open **src → config → conf\_clock.h** and comment out and uncomment the definitions to select the desired main clock source and settings.
2. To select an external 32.768kHz crystal for the slow clock source and for an example of changing the main clock to use an external crystal, see <http://startingelectronics.org/software/atmel/asf-arm-tutorial/external-crystals/>

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/ASF-checklist/solution-explorer.png)

**Atmel Studio Solution Explorer**

**Add Hardware Definitions**

1. Open **src → config → conf\_board.h** from Solution Explorer to add hardware definitions, e.g.:
2. **// output pin for LED**
3. **#define LED0 IOPORT\_CREATE\_PIN(PIOB, 14)**
4. **// input pin for switch**
5. **#define SW0 IOPORT\_CREATE\_PIN(PIOA, 30)**

**Add Board Specific Initialization Code and Initialize System Clock**

1. Use Solution Explorer to open **src → ASF → common → boards → user\_board → init.c** and add board specific initialization code in the **board\_init()** function in **init.c**, e.g. code to disable the watchdog timer, code to set the pin direction of GPIO pins defined in conf\_board.h, etc.
2. **WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**
3. **ioport\_init(); // call before using IOPORT service**
4. **ioport\_set\_pin\_dir(LED0, IOPORT\_DIR\_OUTPUT); // LED pin set as output**
5. **ioport\_set\_pin\_dir(SW0, IOPORT\_DIR\_INPUT); // switch pin set as input**

In the above code example, the **IOPORT – General purpose I/O service (service)** ASF module must be added to the project to be able to call functions starting with **ioport\_** in the project.

1. Call **sysclk\_init()** in main.c
2. **int main (void)**
3. **{**
4. **sysclk\_init();**
5. **board\_init();**
6. **while (1) {**
7. **}**
8. **}**

Finally the application code for the project can be written.

# Reading and Writing ARM Cortex Pins using ASF

Created on: 31 May 2016

**Part 8 of the ASF ARM Tutorial**

How to read the state of a microcontroller pin and write to a microcontroller pin using ASF.

In this part of the ASF ARM tutorial, the IOPORT ASF module is used to read the state of a single microcontroller pin. The same module is used to write to a pin to set the voltage level of the pin to high or low.

## Which ASF Module to Use for Pin I/O

When searching for **IO** in the ASF Wizard in Atmel Studio to find an input/output module for controlling and reading pins, three possible candidates appear in the results, namely, IOPORT, GPIO and PIO. **IOPORT** is the correct module to use in most cases.

The GPIO ASF module is marked as deprecated in the ASF documentation and should not be used in any projects. Use IOPORT instead.

PIO is classed as a driver in ASF and so is lower level than the IOPORT module which is classed as an ASF service. In fact PIO is a dependency of IOPORT and is therefore automatically installed in a project when the IOPORT module is added to the project.

## Creating a Pin Read and Write ASF Project

The Atmel Studio ASF project that follows demonstrates how to read a single microcontroller pin and write to a single microcontroller pin. This is done by reading the on-board push-button switch attached to pin 30 of port A and writing to the on-board LED on pin 14 of port B of the SAM4N Xplained Pro board. If you are using a different board, make the necessary adjustments to the pin definitions in the project.

Follow the steps from the [ASF project checklist](https://startingelectronics.org/software/atmel/asf-arm-tutorial/ASF-checklist/) from the previous part of this tutorial series to create the **in\_out\_pin** ASF project. Add the ASF IOPORT module to the project using the ASF Wizard.

Pins are defined as I/O pins in conf\_board.h.

**conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**// output pin for LED**

**#define LED0 IOPORT\_CREATE\_PIN(PIOB, 14)**

**// input pin for switch**

**#define SW0 IOPORT\_CREATE\_PIN(PIOA, 30)**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

The **IOPORT\_CREATE\_PIN()** macro is used to define pin names in ASF and set their pin and port numbers.

Although the crystals on the board are defined in conf\_board.h to prevent compiler warnings, conf\_clock.h was not modified to use the external 12MHz crystal. The internal R/C oscillator and PLL is used in the project which is the default setting.

Disable the watchdog timer, initialize the IOPORT module and set the pin directions in init.c.

**init.c**

**#include <asf.h>**

**#include <board.h>**

**#include <conf\_board.h>**

**void board\_init(void)**

**{**

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**ioport\_set\_pin\_dir(LED0, IOPORT\_DIR\_OUTPUT); // LED pin set as output**

**ioport\_set\_pin\_dir(SW0, IOPORT\_DIR\_INPUT); // switch pin set as input**

**}**

**ioport\_init()** must be called before using functions from the IOPORT module to enable the IO module.

**ioport\_set\_pin\_dir()** is used to specify whether the pins defined using IOPORT\_CREATE\_PIN() are to be input pins or output pins. The first parameter passed to this function is the pin name. Pin direction is specified by passing either **IOPORT\_DIR\_OUTPUT** for an output pin or **IOPORT\_DIR\_INPUT** for an input pin to this function.

Application code in main.c calls the necessary initialization functions and then reads the pin level of the input pin and writes this level to the output pin.

**main.c**

**#include <asf.h>**

**int main (void)**

**{**

**bool pin\_level;**

**sysclk\_init();**

**board\_init();**

**while (1) {**

**pin\_level = ioport\_get\_pin\_level(SW0);**

**ioport\_set\_pin\_level(LED0, pin\_level);**

**}**

**}**

**ioport\_get\_pin\_level()** reads the state of the push-button switch on the input pin and saves it to the pin\_level variable.

**ioport\_set\_pin\_level()** writes the state of the input pin stored in pin\_level to the output pin which switches the LED.

The video below shows the edited files in Atmel Studio and the program running on the board.

Can't see the video? [View on YouTube →](https://youtu.be/QrzscrQMNQ8)

# Write to a Port on an ARM Microcontroller using ASF

Created on: 3 June 2016

**Part 9 of the ASF ARM Tutorial**

How to write to a port or group of pins on an Atmel ARM Cortex microcontroller using ASF in Atmel Studio.

So far in this tutorial series we have looked at how to enable single microcontroller pins as outputs. ASF has functions that can be used to write to an entire port or to selected pins on a port using a mask to select the pins.

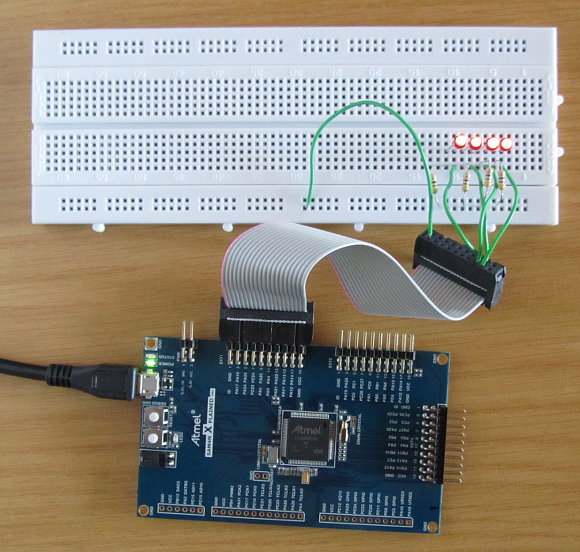
In this part of the ASF ARM tutorial four consecutive pins on a port are configured as outputs and LEDs are connected to them. A count value is written to the LEDs by calling an ASF module function.

The following ASF IOPORT module functions are used to configure and write to the port:

* **ioport\_enable\_port()** – enables the selected port or group of port pins determined by a mask value.
* **ioport\_set\_port\_dir()** – sets the direction of the port or group of port pins as either inputs or outputs.
* **ioport\_set\_port\_level()** – is used to write a value to a port or group of port pins.

## Hardware Configuration

Code is run on an Atmel SAM4N Xplained Pro board and four LEDs with series resistors are connected to PORTA pins on the EXT1 connector as shown in the image below. If you are using a different board, make the necessary changes to your hardware and software if needed.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/port-write/SAM4N-ASF-port-write-LEDs.jpg)

**Hardware for Writing to a Port with ASF on the SAM4N Xplained Pro Board**

Pins from the 20-pin EXT1 header are connected to the breadboard using a [flat ribbon cable with two female IDC connectors](https://startingelectronics.org/articles/IDC-ribbon-cable/) and single core jumper wires.

The following pins are used on the pin header:

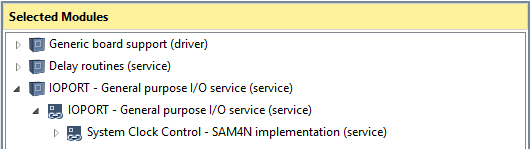
* **GND** – pin 2 of the header is used for common GND on the breadboard power rail.
* **PA11** – EXT1 header pin 15
* **PA12** – EXT1 header pin 17
* **PA13** – EXT1 header pin 16
* **PA14** – EXT1 header pin 18

Each of the four pins is connected to the anode of its own LED and 470Ω series resistor. The other side of each series resistor connects back to the common GND rail on the breadboard. In other words the LEDs are wired in a [current sourcing](https://startingelectronics.org/articles/current-sourcing-sinking/) configuration.

## Creating the ASF Port Write Project

Create a new ASF project called **out\_port** in Atmel Studio as explained in the [ASF quick start checklist](https://startingelectronics.org/software/atmel/asf-arm-tutorial/ASF-checklist/).

Add the **IOPORT - General purpose I/O service (service)** module and the **Delay routines (service)**module to the project using the ASF Wizard. The System Clock Control service module does not have to be added as it is a dependency of the IOPORT module and so is added automatically as shown in the image from the ASF Wizard below. Expanding the IOPORT module in ASF Wizard reveals that it is dependent on the System Clock Control module. It therefore does not matter if you add this module or not.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/port-write/ASF-modules.png)

**ASF Modules Added to the Out Port Project**

## Writing the Port Write Application Code using ASF

The application is written by modifying the following files in the out\_port Atmel Studio project.

### User Hardware Configuration

Hardware for the board is defined in conf\_board.h as shown below. A mask value for the four LEDs is defined and an offset value to shift the mask to the correct position on the port.

**\src\config\conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**// 4 LEDs on pins PA11, PA12, PA13 and PA14**

**// LEDs starting at pin 11 of PORTA**

**#define EXT\_LED\_PORTA\_OFFSET 11**

**// 4 bit wide mask for LEDs on PORTA**

**#define EXT\_LED\_PORTA\_MASK (0x0F << EXT\_LED\_PORTA\_OFFSET)**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

### Hardware Initialization

Hardware initialization functions are called in board\_init() which is found in the init.c file.

**\src\ASF\common\boards\user\_board\init.c**

**#include <asf.h>**

**#include <board.h>**

**#include <conf\_board.h>**

**void board\_init(void)**

**{**

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**// enable PORTA and set direction of port as output**

**ioport\_enable\_port(IOPORT\_PIOA, EXT\_LED\_PORTA\_MASK);**

**ioport\_set\_port\_dir(IOPORT\_PIOA, EXT\_LED\_PORTA\_MASK, IOPORT\_DIR\_OUTPUT);**

**}**

**ioport\_enable\_port()** is called to enable the selected port. In this case PORTA or PIOA is enabled by passing **IOPORT\_PIOA** as the first parameter to this function. The LED mask for the four LEDs defined in conf\_board.h is passed as the second parameter to this function.

**ioport\_set\_port\_dir()** is called to set the direction of the selected port. The port name is passed as the first parameter and the LED mask value as the second parameter. **IOPORT\_DIR\_OUTPUT**, the third parameter passed to this function, sets the masked LED pins as outputs on the selected port.

### Port Write Application Code

Finally the main application code can be added to main.c. The application listed below simply writes a count to the four LEDs. Because the LEDs are not connected to the bottom four pins of the port, the count must be shifted up to the LEDs on the port.

**\src\main.c**

**#include <asf.h>**

**int main (void)**

**{**

**uint32\_t count = 0;**

**sysclk\_init();**

**board\_init();**

**while (1) {**

**ioport\_set\_port\_level(IOPORT\_PIOA, EXT\_LED\_PORTA\_MASK, (count << EXT\_LED\_PORTA\_OFFSET));**

**count++;**

**delay\_ms(200);**

**}**

**}**

**ioport\_set\_port\_level()** takes the port name as the first parameter, LED or pin mask as the second parameter and the shifted count value as the third parameter. When this function is called, it displays the bottom four bits of the count on the LEDs.

The following video shows the code running on the SAM4N Xplained Pro board.

Can't see the video? [View on YouTube →](https://youtu.be/6_cAnHD2JLQ)

# ASF Port Read Write

Created on: 5 June 2016

**Part 10 of the ASF ARM Tutorial**

How to read and write a port or group of port pins using Atmel Software Framework.

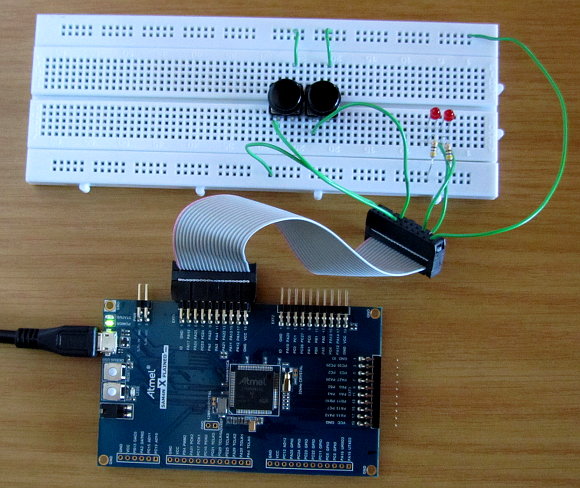
ASF is used to read two switches on a port and write the switch values to two LEDs on a port of an Atmel ARM Cortex microcontroller. The port pins for the read and write operations are accessed using ASF functions from the IOPORT module that access the port pins as a group of pins rather than individual input and output pins.

The following ASF IOPORT module functions are used in this part of the ASF ARM tutorial:

* **ioport\_enable\_port()** – enables the selected port or group of port pins determined by a mask value.
* **ioport\_set\_port\_dir()** – sets the direction of the port or group of port pins as either inputs or outputs.
* **ioport\_set\_port\_mode()** – sets the mode of the port such as enabling pull-up or pull-down resistors on the port pins.
* **ioport\_get\_port\_level()** – gets or reads the state of a port or group of port pins determined by a mask value.
* **ioport\_set\_port\_level()** – is used to write a value to a port or group of port pins.

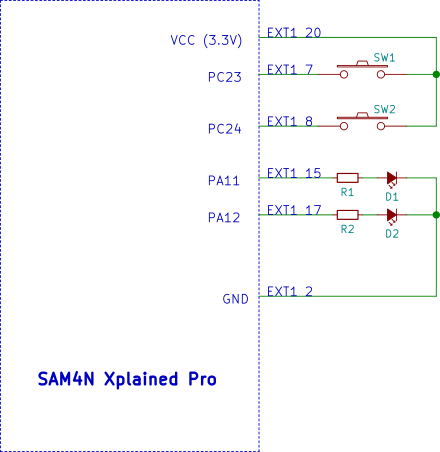
## Hardware Configuration and Circuit Diagram

Two switches are connected to two port pins PC23 and PC24. Two LEDs and series resistors are connected to port pins PA11 and PA12. These pins are taken from the EXT1 header of a SAM4N Xplained Pro board and the circuit built on a breadboard as shown in the image below. Make the necessary changes to the hardware and port definitions in the software if you are using a different board.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/port-read-write/asf-port-read-write-breadboard-circuit.jpg)

**ASF Port Read Write Breadboard Circuit**

Switch and LED connections to the board are shown in the circuit diagram below.

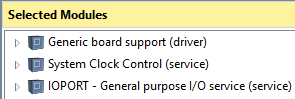
[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/port-read-write/in-out-port-circuit.png)

**Circuit Diagram for the In Out Port Project**

## Creating the ASF Port Read Write Project

Create a new ASF project called **in\_out\_port** in Atmel Studio as explained in the [ASF quick start checklist](https://startingelectronics.org/software/atmel/asf-arm-tutorial/ASF-checklist/).

Add the **IOPORT - General purpose I/O service (service)** module to the project using the ASF Wizard. Although the System Clock Control service is shown in the modules added to the project in the image below, it is a dependency of the IOPORT module so will be added automatically.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/port-read-write/in-out-port-ASF-modules.png)

**In Out Port Project ASF Modules**

## In Out Port Application Code using ASF

Modify the following files in the in\_out\_port Atmel Studio project.

### Hardware Definitions

Mask and offset values are defined in conf\_board.h as shown in the code listing below. The offset value is used to shift the mask to the correct bit position on the LED and switch ports.

**\src\config\conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**// 2 LEDs on pins PA11 and PA12**

**// LEDs starting at pin 11 of PORTA**

**#define EXT\_LED\_PORTA\_OFFSET 11**

**// 2 bit wide mask for LEDs on PORTA**

**#define EXT\_LED\_PORTA\_MASK (0x03 << EXT\_LED\_PORTA\_OFFSET)**

**// 2 switches on pins PC23 and PC24**

**#define EXT\_SW\_PORTC\_OFFSET 23**

**#define EXT\_SW\_PORTC\_MASK (0x03 << EXT\_SW\_PORTC\_OFFSET)**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

### Hardware Initialization

Pins on port A are set up as outputs and pins on port C are set up as inputs with pull-down resistors in the board\_init() function in init.c as shown in the listing below.

**\src\ASF\common\boards\user\_board\init.c**

**#include <asf.h>**

**#include <board.h>**

**#include <conf\_board.h>**

**void board\_init(void)**

**{**

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**// enable PORTA and set direction of port as output**

**ioport\_enable\_port(IOPORT\_PIOA, EXT\_LED\_PORTA\_MASK);**

**ioport\_set\_port\_dir(IOPORT\_PIOA, EXT\_LED\_PORTA\_MASK, IOPORT\_DIR\_OUTPUT);**

**// enable PORTC and set direction of port as input**

**ioport\_enable\_port(IOPORT\_PIOC, EXT\_SW\_PORTC\_MASK);**

**ioport\_set\_port\_dir(IOPORT\_PIOC, EXT\_SW\_PORTC\_MASK, IOPORT\_DIR\_INPUT);**

**// enable pull-down resistors on port**

**ioport\_set\_port\_mode(IOPORT\_PIOC, EXT\_SW\_PORTC\_MASK, IOPORT\_MODE\_PULLDOWN);**

**}**

### Main Application code

In the main application code in main.c the state of the switches on port C are read into the port\_pins variable. These values are then shifted to get them to the bottom of this variable.

The switch states that are stored in port\_pins are then written to the two LEDs on PORT A. It is necessary to shift these bits to the correct bit positions to line up with the LEDs connected to PORT A.

**\src\main.c**

**#include <asf.h>**

**int main (void)**

**{**

**uint32\_t port\_pins;**

**sysclk\_init();**

**board\_init();**

**while (1) {**

**port\_pins = ioport\_get\_port\_level(IOPORT\_PIOC, EXT\_SW\_PORTC\_MASK);**

**port\_pins >>= EXT\_SW\_PORTC\_OFFSET;**

**ioport\_set\_port\_level(IOPORT\_PIOA, EXT\_LED\_PORTA\_MASK, (port\_pins << EXT\_LED\_PORTA\_OFFSET));**

**}**

**}**

The following video shows the port read write / in out application running on a SAM4N Xplained Pro board.

Can't see the video? [View on YouTube →](https://youtu.be/YOpIUYOJkts)

# Standard I/O using a UART and ASF

Created on: 8 June 2016

**Part 11 of the ASF ARM Tutorial**

Standard Input/Output (I/O) is configured to use a UART in an Atmel Studio ASF project so that C stdio functions such as printf() and scanf() use the serial port UART as the standard input/output device.

An ASF module called the Standard serial I/O (stdio)(driver) is used to set up the UART as the stdio device. Strings printed using printf() are then sent out of the UART and scanf() receives input from the UART.

On the Atmel SAM4N Xplained Pro ARM Cortex board, UART0 is connected to the EDBG on-board embedded debugger. One of the functions of the EDBG, besides being used as a debugger, is to act as a virtual COM port when the board is plugged into a PC using the board's USB connector. The virtual COM port can be opened in a terminal emulator running on the PC and data can then be sent and received between the terminal emulator and microcontroller UART.

## Standard I/O UART Project Hardware and Software

### Hardware

This project simply uses the SAM4N Xplained Pro board and USB cable to connect to a PC. The virtual COM port of the EDBG then automatically appears on the PC and should be found by terminal emulator software.

If you are using a different board to the SAM4N, then check the board's user guide to see if it has an EDBG and which UART from the microcontroller is connected to the virtual COM port. The user guide for the SAM4N board shows that pins PA10 and PA9 are used for virtual COM port connections. UART0 is shown to connect to these pins in the ATSAM4N16C datasheet.

### Terminal Emulator Software

[Terminal emulator software such as Tera Term](https://en.osdn.jp/projects/ttssh2/releases/) for Windows will be needed to test this project and communicate over the virtual COM port. The zipped version of Tera Term can be downloaded and opened and the folder inside extracted to a convenient location on the PC. Inside the folder find and run **ttermppro.exe** to start the terminal emulator.

The video at the end of this tutorial shows how to use Tera Term to connect to the virtual COM port and how to change the COM port settings.

## Creating the ASF Standard I/O UART Project

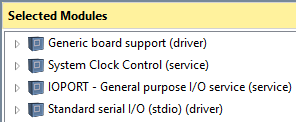
This project basically provides the same functionality as the **Serial Standard I/O (stdio) Example** ASF example project which can be found in Atmel Studio by selecting **File → New → Example Project...** from the top menu bar. The difference is that this project is built from scratch using a user board ASF template and the steps to build the project are described below.

### Start a New ASF Project in Atmel Studio

Create a new ASF project called **stdio\_uart** in Atmel Studio as explained in the [new ASF project quick start checklist](https://startingelectronics.org/software/atmel/asf-arm-tutorial/ASF-checklist/).

### Adding ASF Modules

Add the **IOPORT** service ASF module and the **Standard serial I/O (stdio)** driver module to the project. It is necessary to add the IOPORT module to be able to configure the microcontroller pins to use the UART.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/stdio-uart/stdio-uart-ASF-modules.png)

**ASF Modules used in the Stdio UART Project**

### Hardware Definitions and Configuration (conf\_board.h and conf\_uart\_serial.h)

Add the definition in conf\_board.h for the UART that will be used for stdio. This is UART0 on the SAM4N Xplained Pro board.

**\src\config\conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**#define CONSOLE\_UART UART0**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

The conf\_uart\_serial.h file is added to the project when the Standard serial I/O (stdio) ASF driver module is added to the project. Uncomment the USART settings in this file and change them to the desired values. Be sure to set the UART name (CONF\_UART) to point to the correct UART for the board.

**\src\config\conf\_uart\_serial.h**

**#ifndef CONF\_USART\_SERIAL\_H**

**#define CONF\_USART\_SERIAL\_H**

**/\* A reference setting for USART \*/**

**/\*\* USART Interface \*/**

**#define CONF\_UART CONSOLE\_UART**

**/\*\* Baudrate setting \*/**

**#define CONF\_UART\_BAUDRATE 115200**

**/\*\* Character length setting \*/**

**#define CONF\_UART\_CHAR\_LENGTH US\_MR\_CHRL\_8\_BIT**

**/\*\* Parity setting \*/**

**#define CONF\_UART\_PARITY US\_MR\_PAR\_NO**

**/\*\* Stop bits setting \*/**

**#define CONF\_UART\_STOP\_BITS US\_MR\_NBSTOP\_1\_BIT**

**#endif/\* CONF\_USART\_SERIAL\_H\_INCLUDED \*/**

### Hardware Initialization (init.c)

Pins PA9 and PA10 are set to use internal peripheral A which is UART0 by calling ioport\_set\_port\_mode(). These two UART pins are then disabled as I/O pins by calling ioport\_disable\_port() which dedicates them to the UART.

**\src\ASF\common\boards\user\_board\init.c**

**#include <asf.h>**

**#include <board.h>**

**#include <conf\_board.h>**

**void board\_init(void)**

**{**

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**// configure UART pins**

**ioport\_set\_port\_mode(IOPORT\_PIOA, PIO\_PA9A\_URXD0 | PIO\_PA10A\_UTXD0, IOPORT\_MODE\_MUX\_A);**

**ioport\_disable\_port(IOPORT\_PIOA, PIO\_PA9A\_URXD0 | PIO\_PA10A\_UTXD0);**

**}**

### Application Code (main.c)

A **usart\_serial\_options\_t** structure is set up in main() which is actually a USART initialization structure but can be used with the UART. The structure is initialized with the UART settings defined in conf\_uart\_serial.h.

**stdio\_serial\_init()** is called to initialize the UART and to set it up as the standard I/O device. Standard I/O functions such as printf() and scanf() can then be used in the project as shown in the code in main().

An initial message is written to the terminal emulator. scanf() is then used to check for characters from the terminal emulator which are echoed back to the terminal emulator using printf().

The video below the code listing demonstrates the application being used with Tera Term.

**\src\main.c**

**#include <asf.h>**

**int main (void)**

**{**

**char in\_char;**

**const usart\_serial\_options\_t usart\_serial\_options = {**

**.baudrate = CONF\_UART\_BAUDRATE,**

**.charlength = CONF\_UART\_CHAR\_LENGTH,**

**.paritytype = CONF\_UART\_PARITY,**

**.stopbits = CONF\_UART\_STOP\_BITS**

**};**

**sysclk\_init();**

**board\_init();**

**stdio\_serial\_init(CONF\_UART, &usart\_serial\_options);**

**printf("\r\nHello, world!\r\n");**

**while (1) {**

**scanf("%c", &in\_char);**

**if (in\_char) {**

**printf("%c\r\n", in\_char);**

**}**

**}**

**}**

Using the above application with Tera Term on a Windows 10 PC:

Can't see the video? [View on YouTube →](https://youtu.be/Rn0NzPelj6Q)

# ASF USART Serial Interface for UART Tx and Rx

Created on: 14 June 2016

**Part 12 of the ASF ARM Tutorial**

Communicating between a microcontroller and terminal window using the ASF USART serial interface service module. How to send and receive characters and/or packets of data using a UART or USART with Atmel Software Framework and demonstrated on an ARM Cortex board.

In this part of the ASF tutorial series a UART is used to transmit and receive characters and packets of data using the ASF USART serial interface module. This module provides functions for initializing a UART or USART for serial communications and provides functions for sending and receiving individual characters or packets of data.

Code is used to send and receive data over the virtual COM port which is part of the EDBG unit on an Atmel evaluation board. The same code works on an RS-232 serial port.

## ASF USART Serial Interface Hardware and Software

### Hardware

A SAM4N Xplained Pro board is used which communicates with a PC over a USB cable. Hardware is the same as used in the [previous part of this tutorial](https://startingelectronics.org/software/atmel/asf-arm-tutorial/stdio-uart/) series.

### Software

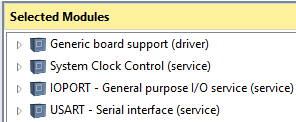
As with the previous part of this tutorial series, a terminal emulator will be needed such as [Tera Term](https://en.osdn.jp/projects/ttssh2/releases/" \t "_blank) for Windows.

## Creating the ASF USART Serial Interface Project

Create a new ASF project called **uart** in Atmel Studio as explained in the [new ASF project quick start checklist](https://startingelectronics.org/software/atmel/asf-arm-tutorial/ASF-checklist/).

### ASF Modules

The ASF module that adds the UART and USART functionality to the project is the **USART - Serial interface (service)** module. Also add the **IOPORT** module which is needed to configure the UART or USART pins.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/usart-serial/uart-asf-modules.png)

**ASF Modules used in the UART Project**

### Hardware Definitions and Configuration (conf\_board.h and conf\_uart\_serial.h)

UART0 is defined as the console UART as it is connected to the EDBG virtual COM port on the SAM4N Xplained Pro board.

**\src\config\conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**#define CONSOLE\_UART (Usart \*)UART0**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

The conf\_uart\_serial.h file is added to the project when the USART Serial interface service module is added to the project. Uncomment the USART settings in this file and change them to the desired values. Be sure to set the UART name (CONF\_UART) to point to the correct UART for the board.

**\src\config\conf\_uart\_serial.h**

**#ifndef CONF\_USART\_SERIAL\_H**

**#define CONF\_USART\_SERIAL\_H**

**/\* A reference setting for UART \*/**

**/\*\* UART Interface \*/**

**//#define CONF\_UART CONSOLE\_UART**

**/\*\* Baudrate setting \*/**

**//#define CONF\_UART\_BAUDRATE 115200**

**/\*\* Parity setting \*/**

**//#define CONF\_UART\_PARITY UART\_MR\_PAR\_NO**

**/\* A reference setting for USART \*/**

**/\*\* USART Interface \*/**

**#define CONF\_UART CONSOLE\_UART**

**/\*\* Baudrate setting \*/**

**#define CONF\_UART\_BAUDRATE 115200**

**/\*\* Character length setting \*/**

**#define CONF\_UART\_CHAR\_LENGTH US\_MR\_CHRL\_8\_BIT**

**/\*\* Parity setting \*/**

**#define CONF\_UART\_PARITY US\_MR\_PAR\_NO**

**/\*\* Stop bits setting \*/**

**#define CONF\_UART\_STOP\_BITS US\_MR\_NBSTOP\_1\_BIT**

**#endif/\* CONF\_USART\_SERIAL\_H\_INCLUDED \*/**

### Hardware Initialization (init.c)

Pins PA9 and PA10 are set to use internal peripheral A which is UART0 by calling ioport\_set\_port\_mode(). These two UART pins are then disabled as I/O pins by calling ioport\_disable\_port() which dedicates them to the UART.

**\src\ASF\common\boards\user\_board\init.c**

**#include <asf.h>**

**#include <board.h>**

**#include <conf\_board.h>**

**void board\_init(void)**

**{**

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**// configure UART pins**

**ioport\_set\_port\_mode(IOPORT\_PIOA, PIO\_PA9A\_URXD0 | PIO\_PA10A\_UTXD0, IOPORT\_MODE\_MUX\_A);**

**ioport\_disable\_port(IOPORT\_PIOA, PIO\_PA9A\_URXD0 | PIO\_PA10A\_UTXD0);**

**}**

### Application Code (main.c)

The application demonstrates some of the UART transmit and receive functions available in the USART - Serial interface service module. Although these functions are more suited to applications where devices or systems may be communicating serially with each other, they are demonstrated using strings and characters in this example.

Using strings and characters makes it easier to demonstrate the functions by using a simple terminal emulator program running on a PC.

**\src\main.c**

**#include <asf.h>**

**int main (void)**

**{**

**const char str1[] = "Type 'a' to continue...\r\n";**

**uint8\_t rx\_char = 0;**

**static usart\_serial\_options\_t usart\_options = {**

**.baudrate = CONF\_UART\_BAUDRATE,**

**.charlength = CONF\_UART\_CHAR\_LENGTH,**

**.paritytype = CONF\_UART\_PARITY,**

**.stopbits = CONF\_UART\_STOP\_BITS**

**};**

**sysclk\_init();**

**board\_init();**

**// initialize the UART**

**usart\_serial\_init(CONF\_UART, &usart\_options);**

**// send a string using the write packet function**

**usart\_serial\_write\_packet(CONF\_UART, (const uint8\_t\*)str1, sizeof(str1) - 1);**

**do {**

**// get a single character**

**usart\_serial\_getchar(CONF\_UART, &rx\_char);**

**} while (rx\_char != 'a');**

**// send a single character**

**usart\_serial\_putchar(CONF\_UART, 'A');**

**while (1) {**

**}**

**}**

The main application code sends a string out of the UART using the **usart\_serial\_write\_packet()**function. This function would typically be used to send some communication data packet from one system to another.

**usart\_serial\_getchar()** is used to receive a single character from the UART serial port.

**usart\_serial\_putchar()** sends a single character to the terminal program using the UART.

A function called **usart\_serial\_read\_packet()** is also available as part of the USART service module. This function is not used in the example program. It simply receives a packet of data of a certain length from the UART.

The following video demonstrates the UART application running.

Can't see the video? [View on YouTube →](https://youtu.be/iClxcc-j0gk)

# ADC ASF Driver Tutorial

Created on: 4 July 2016

**Part 13 of the ASF ARM Tutorial**

How to use the Analog to Digital Converter (ADC) Atmel Software Framework (ASF) driver to initialize and use the ADC on an Atmel ARM Cortex microcontroller.

In this part of the ASF ARM tutorial, the ASF ADC driver is used to initialize the ADC and read the analog value from one of the ADC's channels. The ADC is polled by software to get the channel value.

The result of each ADC conversion is sent out of the serial port of the ARM microcontroller and displayed in a terminal window.

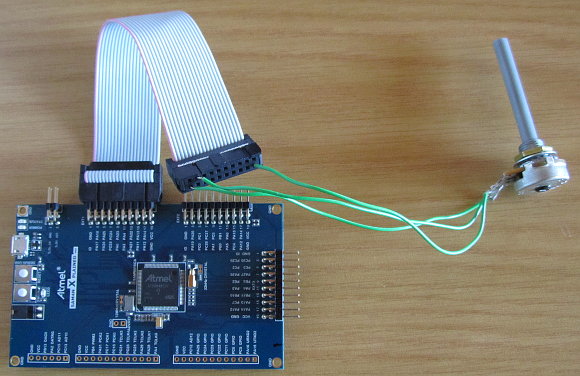
## ADC ASF Tutorial Hardware and Software

### Hardware

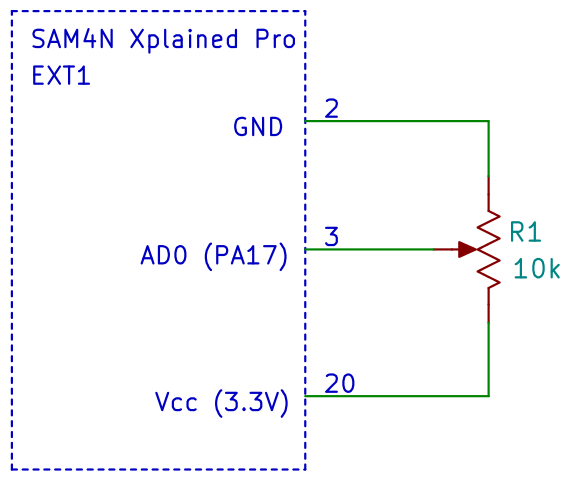
An Atmel SAM4N Xplained Pro board with ATSAM4N16C microcontroller is used in this tutorial. Make the necessary changes to the software and hardware connections if you are using a different board.

A 10k potentiometer is connected to pin header EXT1 of the board with one of the outside pins of the pot. connected to GND and the other outside pin connected to Vcc (3.3V). The centre wiper of the pot. is connected to PA17 which is AD0 of the ADC and pin 3 of the EXT1 header.

The images below show the physical connection of the potentiometer to the SAM4N board and the circuit diagram of the pot. connections to header EXT1.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adc/adc-sam4n-potentiometer-connection.jpg)

**ADC Potentiometer Connection to SAM4N Board**

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adc/SAM4N-potentiometer-circuit.png)

**SAM4N Potentiometer Circuit Connected to EXT1**

### Software

As with the previous part of this tutorial series, a terminal emulator will be needed such as [Tera Term](https://en.osdn.jp/projects/ttssh2/releases/" \t "_blank) for Windows.

## Creating the ADC ASF Tutorial Project

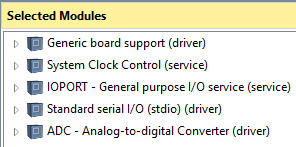
Create a new ASF project called **adc** in Atmel Studio as explained in the [new ASF project quick start checklist](https://startingelectronics.org/software/atmel/asf-arm-tutorial/ASF-checklist/).

This project uses printf() to output text to the terminal window over the serial port, so the project is based on [part 11 of this tutorial series which uses a UART for stdio functions](https://startingelectronics.org/software/atmel/asf-arm-tutorial/stdio-uart/).

### ASF Modules

Add the following modules to the project using the ASF Wizard.

* **System Clock Control (service)**
* **IOPORT - General purpose I/O service (service)**
* **Standard serial I/O (stdio)(driver)**
* **ADC - Analog-to-digital Converter (driver)**

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/adc/adc-asf-modules.png)

**ASF Modules used in the ADC Project**

The ADC driver ASF module adds the ADC functionality to the project. IOPORT and Standard serial I/O are used to set up the UART for stdio functions.

### Hardware Definitions and Configuration (conf\_board.h and conf\_uart\_serial.h)

UART0 is defined as the console UART as it is connected to the EDBG virtual COM port on the SAM4N Xplained Pro board.

**\src\config\conf\_board.h**

**#ifndef CONF\_BOARD\_H**

**#define CONF\_BOARD\_H**

**#define CONSOLE\_UART UART0**

**// clock resonators**

**#define BOARD\_FREQ\_SLCK\_XTAL (32768U)**

**#define BOARD\_FREQ\_SLCK\_BYPASS (32768U)**

**#define BOARD\_FREQ\_MAINCK\_XTAL (12000000U)**

**#define BOARD\_FREQ\_MAINCK\_BYPASS (12000000U)**

**#define BOARD\_MCK CHIP\_FREQ\_CPU\_MAX**

**#define BOARD\_OSC\_STARTUP\_US 15625**

**#endif // CONF\_BOARD\_H**

Uncomment the USART settings in conf\_uart\_serial.h as shown in the code listing below. Be sure to set the UART name (CONF\_UART) to point to the correct UART for the board.

**\src\config\conf\_uart\_serial.h**

**#ifndef CONF\_USART\_SERIAL\_H**

**#define CONF\_USART\_SERIAL\_H**

**/\* A reference setting for UART \*/**

**/\*\* UART Interface \*/**

**//#define CONF\_UART CONSOLE\_UART**

**/\*\* Baudrate setting \*/**

**//#define CONF\_UART\_BAUDRATE 115200**

**/\*\* Parity setting \*/**

**//#define CONF\_UART\_PARITY UART\_MR\_PAR\_NO**

**/\* A reference setting for USART \*/**

**/\*\* USART Interface \*/**

**#define CONF\_UART CONSOLE\_UART**

**/\*\* Baudrate setting \*/**

**#define CONF\_UART\_BAUDRATE 115200**

**/\*\* Character length setting \*/**

**#define CONF\_UART\_CHAR\_LENGTH US\_MR\_CHRL\_8\_BIT**

**/\*\* Parity setting \*/**

**#define CONF\_UART\_PARITY US\_MR\_PAR\_NO**

**/\*\* Stop bits setting \*/**

**#define CONF\_UART\_STOP\_BITS US\_MR\_NBSTOP\_1\_BIT**

**#endif /\* CONF\_USART\_SERIAL\_H\_INCLUDED \*/**

### Hardware Initialization (init.c)

The UART pins are configured in init.c.

**\src\ASF\common\boards\user\_board\init.c**

**#include <asf.h>**

**#include <board.h>**

**#include <conf\_board.h>**

**void board\_init(void)**

**{**

**WDT->WDT\_MR = WDT\_MR\_WDDIS; // disable watchdog**

**ioport\_init(); // call before using IOPORT service**

**// configure UART pins**

**ioport\_set\_port\_mode(IOPORT\_PIOA, PIO\_PA9A\_URXD0 | PIO\_PA10A\_UTXD0, IOPORT\_MODE\_MUX\_A);**

**ioport\_disable\_port(IOPORT\_PIOA, PIO\_PA9A\_URXD0 | PIO\_PA10A\_UTXD0);**

**}**

### Application Code (main.c)

The main application code sets up stdio functions to use the UART. ADC channel 0 is then enabled and polled in the main loop. Values from the ADC are sent out of the UART serial port and displayed in a terminal window.

**\src\main.c**

**#include <asf.h>**

**int main (void)**

**{**

**// structure used to initialize UART for use with stdio functions**

**const usart\_serial\_options\_t usart\_serial\_options = {**

**.baudrate = CONF\_UART\_BAUDRATE,**

**.charlength = CONF\_UART\_CHAR\_LENGTH,**

**.paritytype = CONF\_UART\_PARITY,**

**.stopbits = CONF\_UART\_STOP\_BITS**

**};**

**struct adc\_config adc\_conf; // struct for configuring ADC**

**uint32\_t adc\_val; // holds ADC value read from ADC channel**

**sysclk\_init();**

**board\_init();**

**// initialize stdio functions so printf() uses the UART**

**stdio\_serial\_init(CONF\_UART, &usart\_serial\_options);**

**// configure and enable the ADC**

**adc\_enable(); // enable ADC**

**adc\_get\_config\_defaults(&adc\_conf); // read ADC default values**

**adc\_init(ADC, &adc\_conf); // initialize the ADC with default values**

**adc\_set\_trigger(ADC, ADC\_TRIG\_SW); // set ADC to trigger with software for polling**

**adc\_channel\_enable(ADC, ADC\_CHANNEL\_0); // enable ADC channel 0 (AD0, pin PA17 on ATSAM4N16C)**

**while (1) {**

**// start the ADC conversion**

**adc\_start\_software\_conversion(ADC);**

**// wait until the conversion complete bit for AD0 is set**

**while (!(adc\_get\_interrupt\_status(ADC) & (1 << ADC\_CHANNEL\_0)));**

**// read the value from the ADC for channel 0**

**adc\_val = adc\_channel\_get\_value(ADC, ADC\_CHANNEL\_0);**

**// display the ADC value in decimal and hexadecimal**

**printf("ADC value: %04u, 0x%04x\r\n", (unsigned)adc\_val, (unsigned)adc\_val);**

**}**

**}**

In the while(1) loop, the ADC conversion is started. A bit is then polled in the ADC\_ISR register which will be set when the conversion for AD0 is complete. Polling of the register is done by calling adc\_get\_interrupt\_status(). When the conversion is complete, the ADC value for channel 0 is read from the ADC and sent to the terminal window.

The video below shows the above ADC program running.

Can't see the video? [View on YouTube →](https://youtu.be/G9Ozbu_Ozao)

# ASF Documentation and Help

Created on: 12 January 2017

**Part 14 of the ASF ARM Tutorial**

How to find ASF documentation and help on ASF modules and functions in Atmel Studio 7.

This part of the Atmel ASF ARM tutorial shows how to find documentation and help on using ASF modules in Atmel Studio 7.

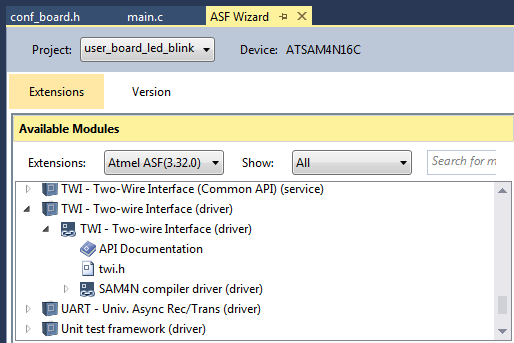
So far this tutorial series has shown how to use some of the modules available in ASF. In order to use other ASF modules it is necessary to find documentation and examples of the use of these modules in Atmel Studio.

## Finding ASF Module Documentation before Adding Modules

Documentation for a specific ASF module can easily be found and read before adding the module to a project in Atmel Studio. This enables the module to be evaluated to see that it fits the required purpose before adding the module to the project.

The documentation can be found in the ASF Wizard by selecting **ASF → ASF Wizard** from the top menu bar or by using the keyboard shortcut **Alt + W**.

Available modules in the left pane of the ASF Wizard can be expanded to find their documentation as shown in the image below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-documentation/asf-wizard-available-modules-documentation.png)

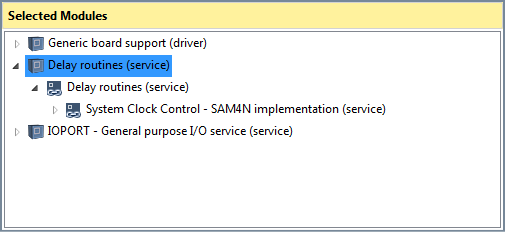
**Available Modules Documentation in ASF Wizard**

Double-clicking **API Documentation** opens the Atmel Software Framework documentation for the module in the default web browser, e.g. for the TWI module in the above image.

Double-clicking the module header file opens the header file for the module in Atmel Studio, e.g. twi.h in the above example. The header file for a module contains data structures and functions prototypes for the module with comments which can help provide information on usage of the module.

## Finding ASF Module Documentation after Adding Modules

Once a module has been added to a project, the documentation for the module can no longer be found in the ASF Wizard as shown in the image below.

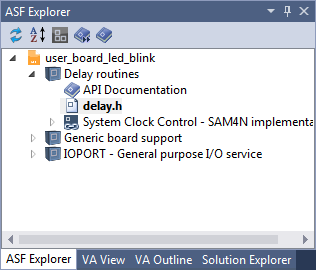
[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-documentation/asf-wizard-right-panel.png)

**ASF Wizard Right Panel Selected Modules**

In the above image, the **Delay Routines** ASF module that was added to the project appears in the ASF Wizard right pane. When the module is expanded, no documentation links can be found.

To find documentation for modules already added to the project, select **ASF → ASF Explorer** from the top menu or use the keyboard shortcut **Alt + A**. ASF Explorer will appear in the top right pane of Atmel Studio.

The image below shows ASF Explorer with the Delay Routines ASF Module expanded. ASF documentation can now be accessed the same way as was done in the ASF Wizard.

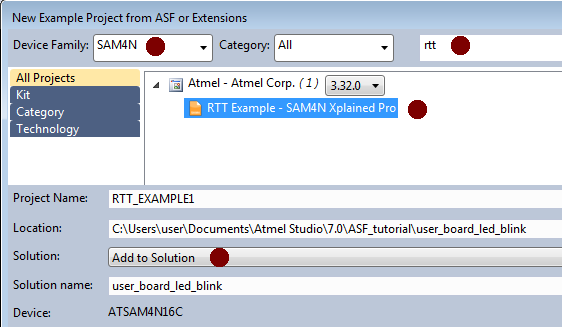
[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-documentation/asf-explorer.png)

**ASF Explorer Showing the Delay Routines Module Expanded**

## Finding ASF Examples

ASF examples that show the usage of various ASF modules can be found by starting a new example project. This is done by selecting **File → New → Example Project...** from the top menu or using the **Ctrl + Shift + E** keyboard shortcut.

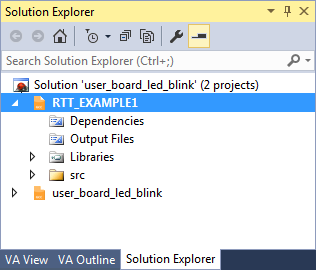
In the new example project dialog box that opens, the device family can be filtered for as well as category and keyword using the top filter boxes in the dialog box as shown in the image below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-documentation/new-asf-example-project.png)

**New ASF Example Project**

The image shows that the SAM4N family device was filtered for and that the term "rtt" was used for further filtering. The RTT Example was selected and added to the current solution by selecting **Add to Solution**near the bottom of the dialog box.

When an example project is added to a solution, both the example project and the currently opened project appear in the same solution as shown below.

[](https://startingelectronics.org/software/atmel/asf-arm-tutorial/asf-documentation/atmel-studio-solution-explorer.png)

**Atmel Studio Solution Explorer with Two Projects Open in the Solution**

In the above image two projects are open in the same solution, namely **RTT\_EXAMPLE1** and **user\_board\_led\_blink**. This allows easy access to the example project so that code can be copied and pasted to the user project as needed without having to open another instance of Atmel Studio.

In this way any example project can be added to the solution that a current user project is in. If code on how to use the USART is needed, search for and add a USART example project to the solution. See how the code works and add the necessary code to the user project.