



Atmel Studio 7

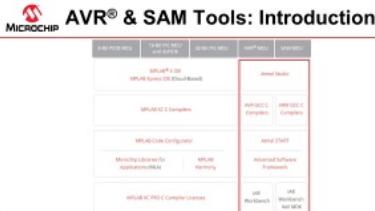
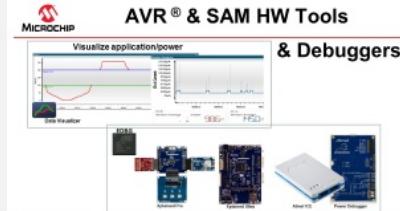
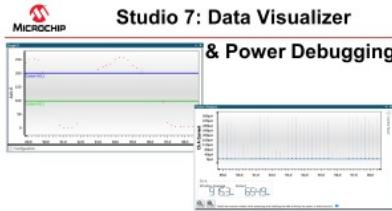
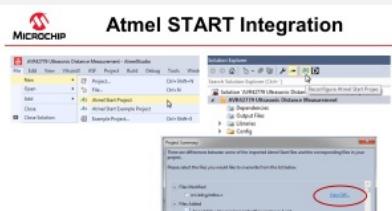
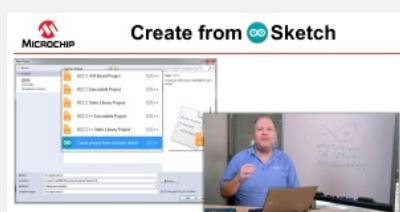
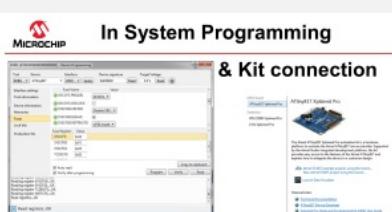
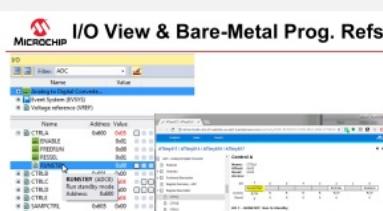
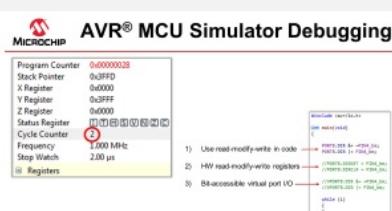
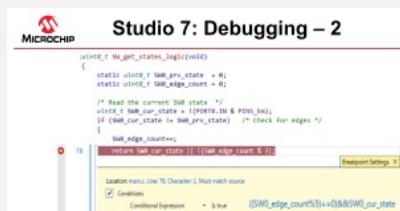
Getting Started with Atmel Studio 7

Table of Contents

1. Getting Started.....	3
1.1. AVR® and SAM Development Tools Overview.....	5
1.2. AVR® and SAM HW Tools and Debuggers.....	6
1.3. Data Visualizer and Power Debugging Demo.....	8
1.4. Installation and Updates.....	10
1.5. Atmel Gallery and Studio Extensions.....	12
1.6. Atmel START Integration.....	13
1.7. Creating a New Project.....	18
1.8. Creating From Arduino Sketch.....	24
1.9. In-System Programming and Kit Connection.....	25
1.10. I/O View and Other Bare-Metal Programming References	32
1.11. Editor: Writing and Re-Factoring Code (Visual Assist).....	44
1.12. AVR Simulator Debugging.....	53
1.13. Debugging 1: Break Points, Stepping, and Call Stack.....	58
1.14. Debugging 2: Conditional- and Action-Breakpoints	68
1.15. Debugging 3: I/O View Memory View and Watch.....	75
2. Revision History.....	83
The Microchip Web Site.....	84
Customer Change Notification Service.....	84
Customer Support.....	84
Microchip Devices Code Protection Feature.....	84
Legal Notice.....	85
Trademarks.....	85
Quality Management System Certified by DNV.....	86
Worldwide Sales and Service.....	87

1. Getting Started

Getting Started Atmel Studio 7 - [playlist](#).

<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Description </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Description </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Demo code </div>
<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>
<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>
<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>
<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>	<div style="border: 1px solid #ccc; padding: 10px; margin-bottom: 10px;">  </div> <div style="display: flex; justify-content: space-around;"> Video Hands-on </div>

This Getting Started training for Atmel Studio 7 will guide you through all the major features of the IDE. It is designed as a video series with accompanying hands-ons. Each section starts with a video, which covers that section.

Prerequisites

Much of the training could be completed by using the editor and simulator, however, in order to cover everything the following is recommended.

Hardware prerequisites:

- ATtiny817 Xplained Pro
- Standard-A to Micro-B USB cable

Software prerequisites:

- Atmel Studio 7.0
- avr-gcc toolchain
- Latest Part Pack for tinyAVR® devices

Atmel Studio 7.0 plugins used:

- Atmel Start 1.0.113.0 or later
- Data Visualizer Extension 2.14.709 or later

Icon Key Identifiers

The following icons are used in this document to identify different assignment sections and to reduce complexity.



Info: Delivers contextual information about a specific topic.



Tip: Highlights useful tips and techniques.



To do: Highlights objectives to be completed.



Result: Highlights the expected result of an assignment step.



Indicates important information.



Execute: Highlights actions to be executed out of the target when necessary.

1.1 AVR® and SAM Development Tools Overview

This section gives an overview of the various pieces in the AVR® and SAM Tools ecosystem and how they relate to each other.

Getting Started Topics



AVR® & SAM Tools: Intro & Overview

In this video:

Context in Microchip Tools Ecosystem

- IDE, Compiler, MCU & SW configurator tools, Firmware Libraries

START, Software Content and IDEs

- How these pieces fit together.
- START-based development
 - START user manual
 - Getting Started projects in START

Atmel Studio 7

- Bare-metal- vs. START-based development
- Build from scratch (bare-metal):
 - Getting Started Atmel Studio 7
 - Getting Started with AVR Tools



Video: AVR and SAM Tools ecosystem overview

Atmel START is a web-based software configuration tool, for various software frameworks, which helps you getting started with MCU development. Starting from either a new project or an example project, Atmel START allows you to select and configure software components (from **ASF4** and **Foundation Services**), such as drivers and middleware to tailor your embedded application in a usable and optimized manner. Once an optimized software configuration is done, you can download the generated code project and open it in the IDE of your choice, including Studio 7, IAR Embedded Workbench®, Keil® µVision®, or simply generate a make file.

Atmel START enables you to:

- Get help with selecting an MCU, based on both software and hardware requirements
- Find and develop examples for your board
- Configure drivers, middleware, and example projects
- Get help with setting up a valid PINMUX layout
- Configure system clock settings



START, Software Content & IDEs

Select MCU on both HW & SW req.

Explore:

- Software components
- Kits, examples & reference solutions
- Microchip & 3rd party software

Configure Device:

- MCU Pin mux, System clock, Logic etc.

Configure Software Content:

- Configure software framework for selected MCU, including peripheral drivers and middleware:
 - AVR: AVR Code (Foundation Services)
 - SAM: ASF4

Develop in IDE:

- Export from START to a range of IDEs
 - Atmel Studio 7, Keil uVision, IAR Embedded Workbench,
 - MPLAB X Coming soon...



ASF, the Advanced Software Framework, provides a rich set of proven drivers and code modules developed by experts to reduce customer design-time. It simplifies the usage of microcontrollers by providing an abstraction to the hardware through drivers and high-value middlewares. ASF is a free and open-source code library designed to be used for evaluation, prototyping, design, and production phases.

ASF4, supporting the SAM product line, is the fourth major generation of ASF. **ASF4** represents a complete re-design and -implementation of the whole framework, to improve the memory footprint, code performance, as well as to better integrate with the Atmel START web user interface. ASF4 must be used in conjunction with Atmel START, which replaces the ASF Wizard of ASF2 and 3.

Foundation Services, supporting the AVR product line, is a simple firmware framework for AVR 8-bit MCUs, equivalent to Foundation Services, which supports 8- and 16-bit **PIC** MCUs. **Foundation Services** is optimized for code-size and -speed, as well as simplicity and readability of code. Foundation Services is configured by Atmel START.

An **IDE** (Integrated Development Environment) is used to develop an application (or further develop an example application) based on the software components, such as drivers and middlewares, configured in and exported from Atmel START.

Atmel Studio 7 is the integrated development platform (IDP) for developing and debugging all AVR and SAM microcontroller applications. The Atmel Studio 7 IDP gives you a seamless and easy-to-use environment to write, build, and debug your applications written in C/C++ or assembly code. It also connects seamlessly to the debuggers, programmers, and development kits that support AVR and SAM devices. The development experience between Atmel START and Studio 7 has been optimized. Iterative development of START-based projects in Studio 7 is supported through re-configure and merge functionality.

1.2 AVR® and SAM HW Tools and Debuggers

This section describes the HW Tools ecosystem for AVR® and SAM MCUs.

Getting Started Topics



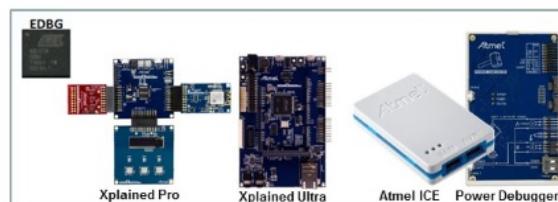
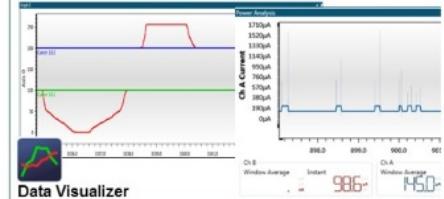
AVR® & SAM HW Tools & Debuggers

In this video:

Debugging Platform & user interface

- Xplained Development kit platform
- In circuit debuggers
 - Atmel ICE / Power Debugger
- Data Visualizer
 - User Interface for debugging platform
 - Visualizes data to give insight to application
 - Analyze and correlate power consumption to code

Visualize application/power



[Video: AVR & SAM HW Tools & Debuggers](#)

Data Visualizer

The Data Visualizer is a program to process and visualize data. The Data Visualizer is capable of receiving data from various sources such as the Embedded Debugger Data Gateway Interface (DGI) and COM ports. Track your application's run-time using a terminal or graph, or analyze the power consumption of your application through correlation of code execution and power consumption, when used together with a supported probe or board. Having full control of your codes' run-time behavior has never been easier.

Both a stand-alone and a plug-in version for Atmel Studio 7 are available at the website link below.

[Website: Data Visualizer.](#)

Atmel-ICE

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

Atmel-ICE is a powerful development tool for debugging and programming ARM Cortex-M based SAM and AVR microcontrollers with on-chip debug capability.

[Website: Atmel-ICE](#)

Power Debugger:

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

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

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

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

For more information, visit the [Online User Guide](#).

Website: [Power Debugger](#)

1.3

Data Visualizer and Power Debugging Demo

This section shows a demo using the Data Visualizer including Power Debugging.

[Getting Started Topics](#)



Studio 7: Data Visualizer & Power Debugging

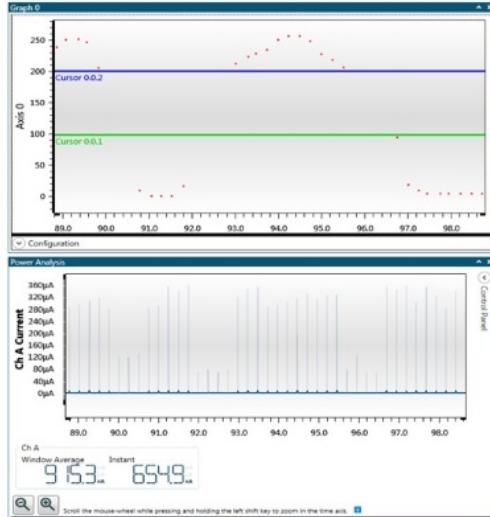
In this video:

Studio 7: Data Visualizer & Power Debugging Context

Low-power demo: RTC periodic timer, starts ADC conversion, via event system. ADC result sent on USART.

Features covered:

- mEDBG: ATtiny817 Xplained Mini
 - Data Input: serial port
 - Visualization: terminal, graph
- EDBG: ATtiny817 Xplained Pro
 - Data Input: Serial + DGI (USART, SPI, I²C, GPIO)
 - Visualization: Graph (serial + DGI GPIO)
- Power Debugger Analog module: ATtiny817 Xplained Pro
 - Power measurement& DGI GPIO graphs
- User-guide
 - Tips for F1 access



[Video: Data Visualizer and Power Debugging Demo](#)

```
/*
 * Power_Demo_ADC_SleepWalking.c
 * Device/board: ATtiny817 Xplained Pro
 * Created: 8/6/2017 3:15:21 PM
 */

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

#define F_CPU (20E6/2)

void sys_init(void)
{
    _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL_PEN_bm | CLKCTRL_PDIV_2X_gc);
}

void rtc坑_init(void)
{
```

```

    RTC.CLKSEL = RTC_CLKSEL_INT1K_gc;
    RTC.PITCTRLA = RTC_PITEN_bm | RTC_PERIOD_CYC256_gc;
}

//picoPower 4: Event system vs. IRQ. Compare to not using IRQ
void evsys_init(void)
{
    EVSYS.ASYNCCH3 = EVSYS_ASYNCCH3_PIT_DIV128_gc;
    EVSYS.ASYNCUSER1 = EVSYS_ASYNCUSER1_ASYNCCH3_gc;
}

//picoPower 3: Evaluate own sample, e.g. window mode.
//              Significantly reduce awake time.

void adc_init(void)
{
    ADC0.CTRLC = ADC_PRESC_DIV8_gc | ADC_REFSEL_VDDREF_gc;
    ADC0.CTRLA = ADC_ENABLE_bm | ADC_RESSEL_8BIT_gc;
    ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc;

    ADC0.CTRLA |= ADC_RUNSTBY_bm;           //picoPower 1: So can run in sleep.
    ADC0.CTRLA = ADC_WINCM_OUTSIDE_gc;      //picoPower 3: So can evaluate own sample.
    ADC0.INTCTRL = ADC_WCMP_bm;
    ADC0.WINHT = 200;
    ADC0.WINLT = 100;

    ADC0.EVCTRL = ADC_STARTEI_bm;          //picoPower 4: So event can trigger conversion
}

uint8_t adc_get_result(void)
{
    return ADC0.RESL;
}

//picoPower 5: Send quickly, then back to sleep: compare 9600, 115200, 1250000 baud rates
//note only sending 1 byte
#define BAUD_RATE 57600
void usart_init()
{
    USART0.CTRLB = USART_TXEN_bm;
    USART0.BAUD = (F_CPU * 64.0) / (BAUD_RATE * 16.0);
}
void usart_put_c(uint8_t c)
{
    VPORTB.DIR |= PIN2_bm | PIN6_bm; //picoPower 2b: see Disable Tx below
    USART0.STATUS = USART_TXCIF_bm;

    VPORTB.OUT |= PIN6_bm;
    USART0.TXDATA = c;
    while(!(USART0.STATUS & USART_TXCIF_bm));
    VPORTB.OUT &= ~PIN6_bm;
    VPORTB.DIR &= ~PIN2_bm | PIN6_bm;
    //picoPower 2b: Disable Tx pin in-between transmissions
}

//picoPower 2: Disable unused GPIO
//              compare: Nothing, PORT_ISC_INPUT_DISABLE_gc, PORT_PULLUPEN_bp

void io_init(void)
{
    for (uint8_t pin=0; pin < 8; pin++)
    {
        (&PORTA.PIN0CTRL)[pin] = PORT_ISC_INPUT_DISABLE_gc;
        (&PORTB.PIN0CTRL)[pin] = PORT_ISC_INPUT_DISABLE_gc;
        (&PORTC.PIN0CTRL)[pin] = PORT_ISC_INPUT_DISABLE_gc;
    }
}

int main(void)
{
    sys_init();
    rtc_pit_init();
    evsys_init();
    adc_init();
    io_init();
    usart_init();
}

```

```
VPORTB.DIR |= PIN6_bm;
VPORTB.OUT &= ~PIN6_bm;
sei();

//picoPower 1: Go to sleep. Compare with no sleep, IDLE and STANDBY
set_sleep_mode(SLEEP_MODE_STANDBY);

while (1)
{
    sleep_mode();
}

ISR(ADC0_WCOMP_vect)          //picoPower 3: Only called if relevant sample
{
    ADC0.INTFLAGS = ADC_WCMP_bm;
    usart_put_c(adc_get_result());
}
```

1.4 Installation and Updates

This section describes the process of installing Atmel Studio 7, installing updates for Studio or plugins, as well as adding support for new devices.

Getting Started Topics



Studio 7: Installation & Updates

In this video:

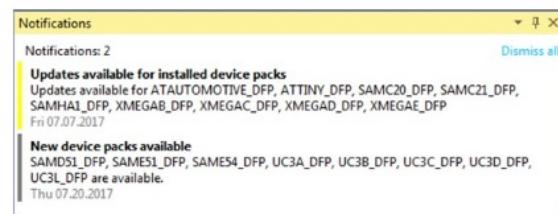
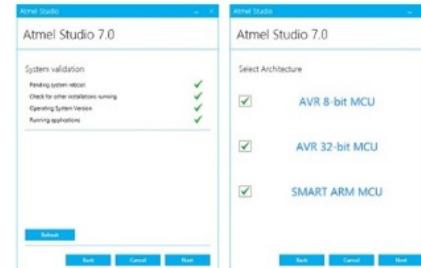
Studio 7 installation experience

Installation choices:

- AVR® 8-bit MCU, AVR 32-bit MCU, SAM MCU
- Atmel Software Framework and example projects

Updating Studio 7:

- Update notifications
- Installing support for latest devices (pack manager)



Video: Installation and Updates

1.4.1 Installation

Supported Operating Systems

- Windows 7 Service Pack 1 or higher
- Windows Server 2008 R2 Service Pack 1 or higher
- Windows 8/8.1

- Windows Server 2012 and Windows Server 2012 R2
- Windows 10

Supported Architectures

- 32-bit (x86)
- 64-bit (x64)

Hardware Requirements

- A computer that has a 1.6 GHz or faster processor
- RAM
 - 1 GB RAM for x86
 - 2 GB RAM for x64
 - An additional 512 MB RAM if running in a Virtual Machine
- 6 GB available hard disk space

Downloading and Installing

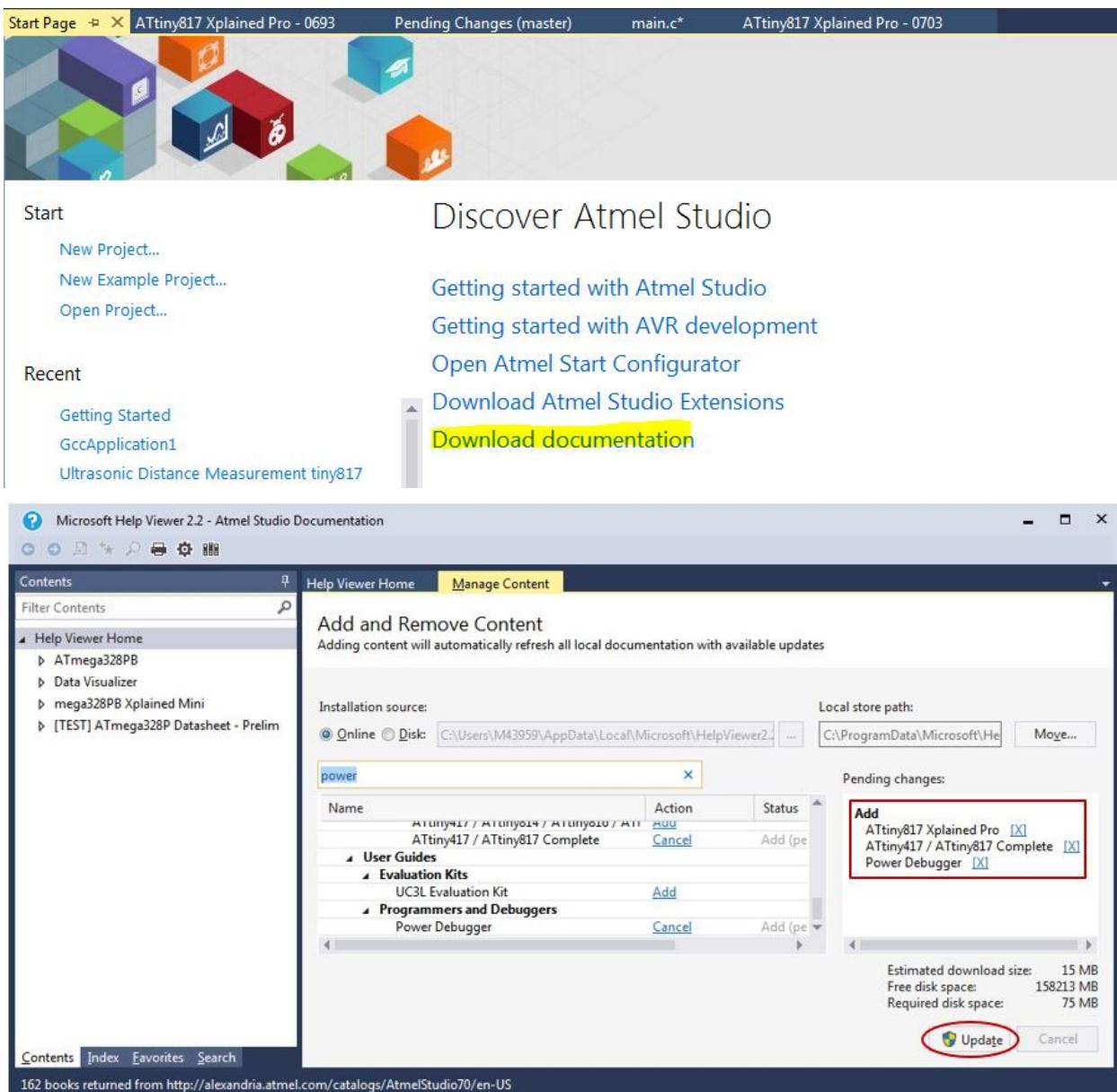
- Download the latest Atmel Studio installer: [Atmel Studio 7](#)
 - The web installer is a small file (<10 MB) and will download specified components as needed.
 - The offline installer has all components embedded
- Atmel Studio can be run side-by-side with older versions of Atmel Studio and AVR Studio®. Uninstallation of any previous versions is not required.
- Verify the hardware and software requirements from the "System Requirements" section
- Make sure your user has local administrator privileges
- Save all your work before starting. The installation might prompt you to restart if required.
- Disconnect all Atmel USB/Serial hardware devices
- Double-click the installer executable file and follow the installation wizard
- Once finished, the installer displays an option to **Start Atmel Studio after completion**. If you choose to open, then note that Atmel Studio will launch with administrative privileges, since the installer was either launched as administrator or with elevated privileges.
- In Atmel Studio you may see an update notification (flag symbol) next to the Quick Launch field in the title bar. Here you may select and install updated components or device support.

1.4.2 Downloading Offline Documentation

If you would like to work offline, it would be advisable to use the offline documentation for Studio 7. To do this, from the *Studio 7 Start Page*, click on *Download documentation*. When the help viewer pops up, first click the *Online* button and search for documentation of interest, such as *data sheets*, *user manuals*, and *application notes* (wait for the available documents to show up).

In the example below, we are choosing to download the *Power Debugger user manual*, the *ATtiny817 Xplained Pro user manual*, as well as the *ATtiny817 Complete data sheet*. Clicking update will then initiate the download.





1.5 Atmel Gallery and Studio Extensions

This section describes how Atmel Studio can be extended and updated through the Atmel Gallery. Some of the most useful and popular extensions are described.

[Getting Started Topics](#)



Studio 7: Gallery & Extensions

In this video: How to add extensions

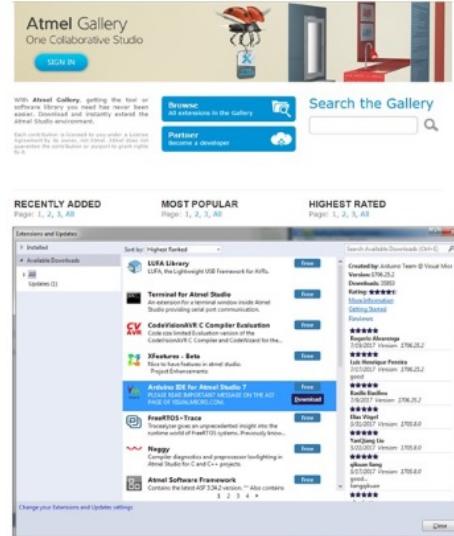
- Tools -> Gallery Profile

Extensions:

- Part of Studio 7:** Visual Assist, Atmel START, Data Visualizer, Toolchain
- Popular:** Arduino® IDE for Studio 7, LUFA Library, ASF (Naggy)
- Used in series:** Doxygen integrator, Git Source Control Provider,

Extension options/settings

- Tools → Options



[Video: Gallery, Studio Extensions and Updates](#)

Add Extensions

Included Extensions

Popular Extensions

Extensions Used in Series

Extension Options/Settings

1.6 Atmel START Integration

The development experience between Atmel START and Studio 7 has been optimized. This section demonstrates the iterative development process of START-based projects in Studio 7, through the *re-configure* and *merge* functionality.

[Getting Started Topics](#)



Studio 7: Atmel START Integration

In this video:

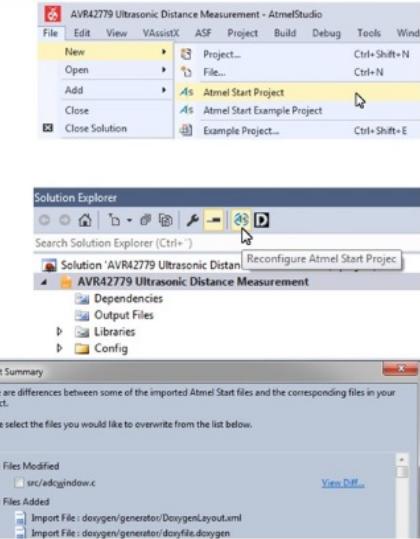
START-based dev. in Studio 7

Creating:

- New Atmel START project
- New Atmel START example project
 - Open: Ultrasonic distance measurement example

Iterative development

- Re-configure Atmel START project
- Handling Diff/Merge
- AVR® code project documentation



[Video: Atmel START Integration](#)



To do: Exporting the Project from Atmel START.

1. On the Atmel START website, create a new project (Example or Board).
2. Click on the Export Software Component button. Make sure the Atmel Studio check-box is checked.
3. Click on Download pack. An atmelstart.atzip pack file will be downloaded.

Figure 1-1. Download Your Configured Project

DOWNLOAD YOUR CONFIGURED PROJECT

Download a generated pack containing all your configured software components.

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

 Atmel Studio:	<input checked="" type="checkbox"/>
---	-------------------------------------

 μVision from Keil:	<input checked="" type="checkbox"/>
--	-------------------------------------

 IAR Embedded Workbench:	<input checked="" type="checkbox"/>
---	-------------------------------------

 Somnium DRT (Atmel Studio plugin):	<input checked="" type="checkbox"/>
--	-------------------------------------

 Makefile (standalone):	<input checked="" type="checkbox"/>
--	-------------------------------------

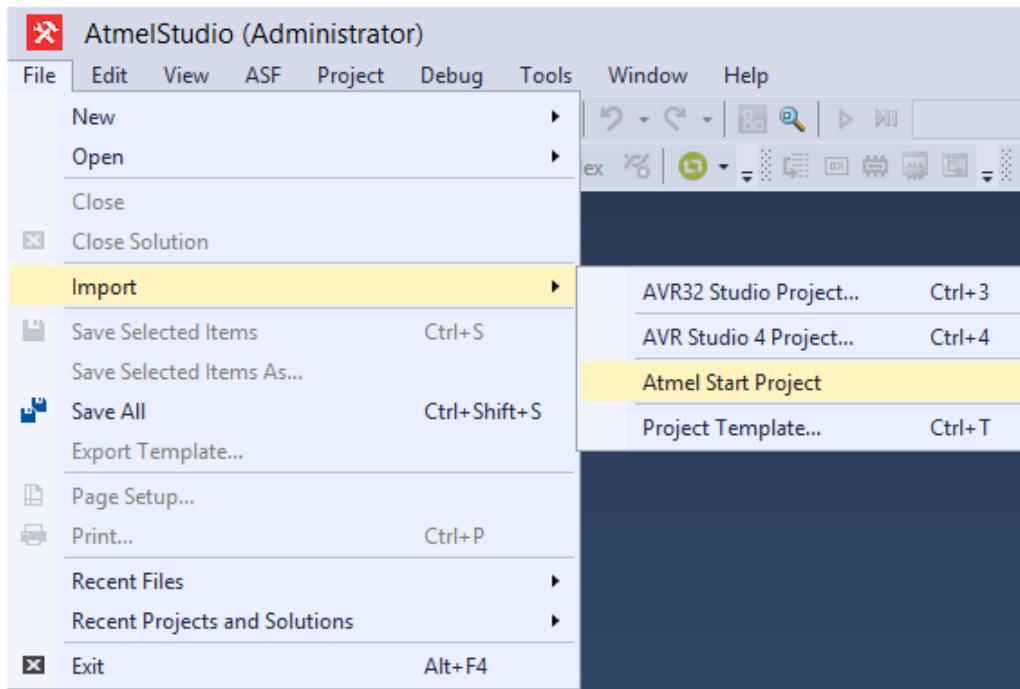
Specify file name (optional):

 DOWNLOAD PACK

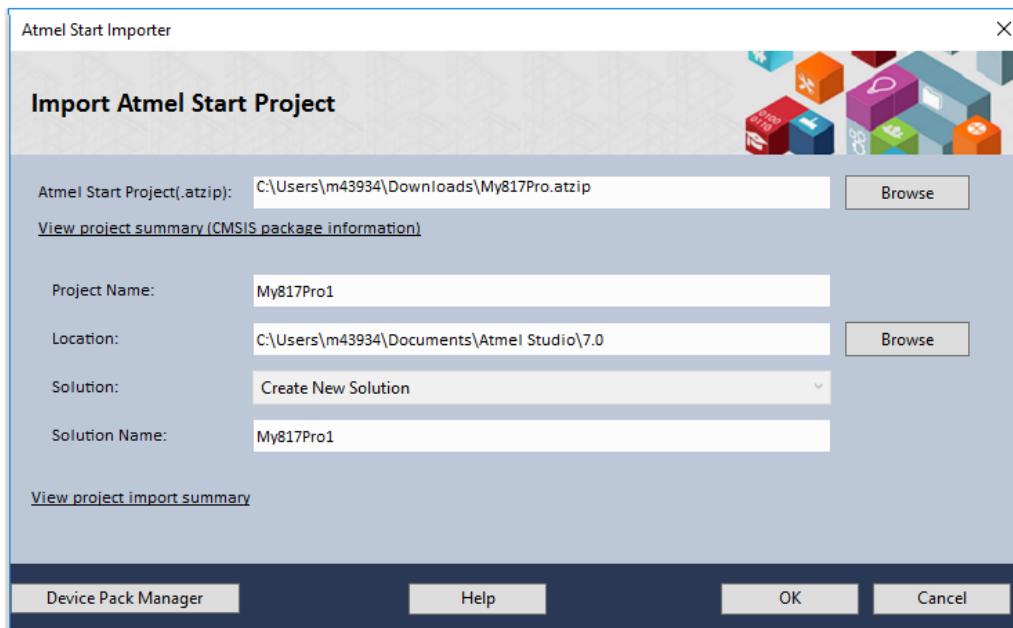


To do: Import the Atmel START Output into Atmel Studio.

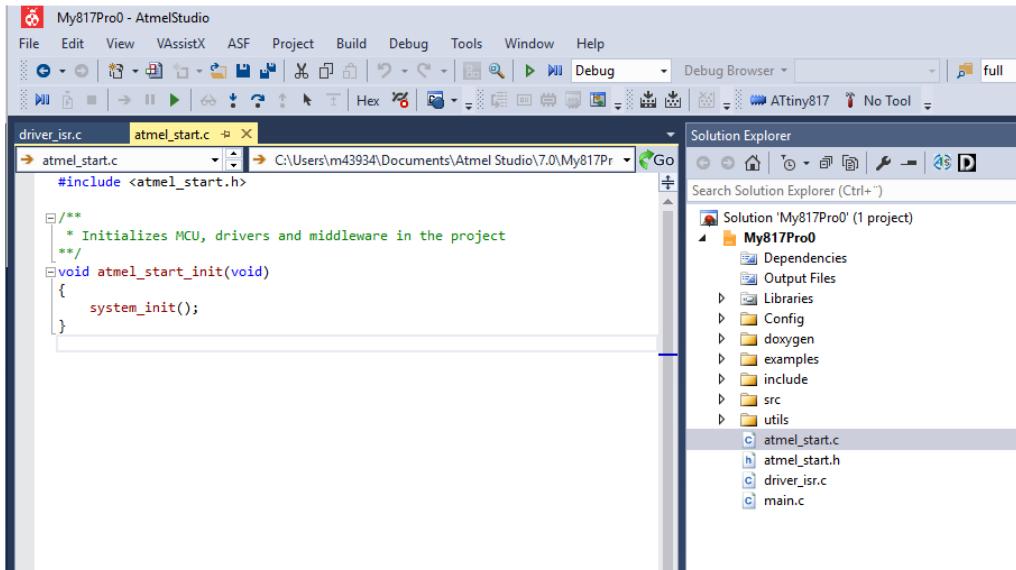
4. Launch Atmel Studio.
5. Select File > Import > Atmel Start Project.

Figure 1-2. Import Atmel START Project

6. Browse and select the downloaded atmelstart.atzip file.
7. The Atmel Start Importer dialog box will open. Enter the project details as Project name, Location, and Solution name. Click OK.

Figure 1-3. START Project Importer

-
8. A new Atmel Studio project will be created and the files will be imported.



To do: Import the Atmel START Output into Atmel Studio.

9. Some projects contain documentation formatted for Doxygen.

Note: Doxygen must be downloaded from <http://www.doxygen.org> and installed. You will be asked to configure Studio to locate Doxygen executable, this defaults to C:\Program Files\doxygen\bin\doxygen.exe.

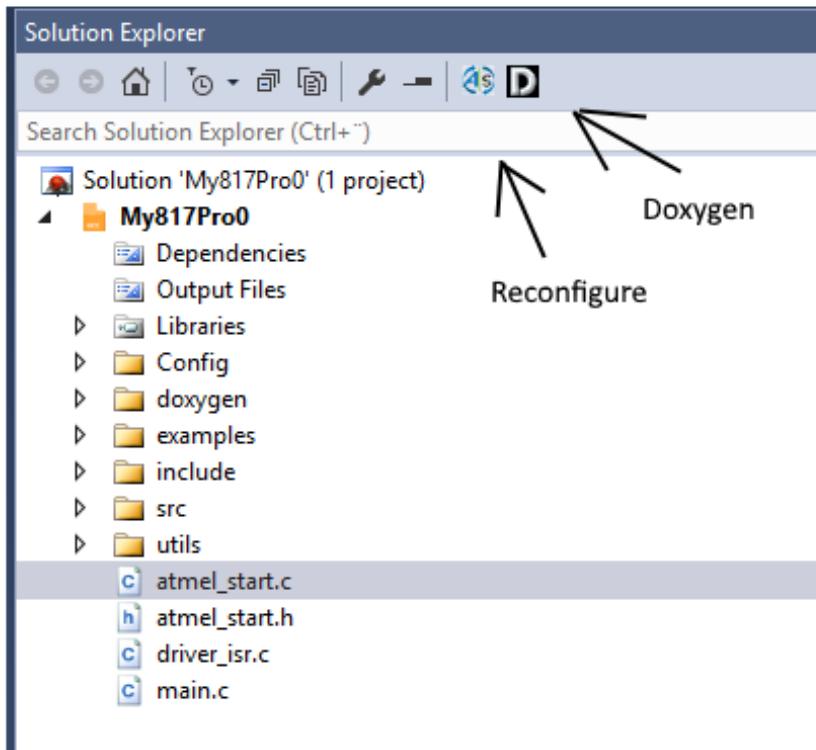
10. Click on the Doxygen button to generate the documentation. Doxygen will run and the generated documentation will open in a new window.
-



To do: Reconfigure the project using Atmel START.

11. Click on the Reconfigure button or right-click on the project node in the Solution Explorer, and, from the menu, select Reconfigure Atmel Start Project.
12. Atmel Start will open in a window inside Atmel Studio.

Figure 1-4. Reconfigure START Project and Doxygen Buttons



13. Do the necessary changes to the project. Click the GENERATE PROJECT button at the bottom of the Atmel Start window.

1.7 Creating a New Project

This section will outline the process of creating a new Atmel Studio project.

[Getting Started Topics](#)



Studio 7: Creating a New Project

In this video:

Create new project:

Selecting the right project type

- GCC C/C++ Executable Project
- GCC C/C++ Static Library Project
- Create project from Arduino® Sketch

ASF3 Projects

- GCC ASF Board Project
- ASF Example Project

ASF4/AVR® Code Projects:

- Atmel START project
- Atmel START example project



ATtiny817 Xplained Pro

The Atmel ATtiny817 Xplained Pro evaluation kit is a hardware platform to evaluate the Atmel ATtiny817 microcontroller. Supported by the Atmel Xplained Pro software development platform, it provides easy access to the features of the Atmel ATtiny817 and explains how to integrate the device in a customer design.

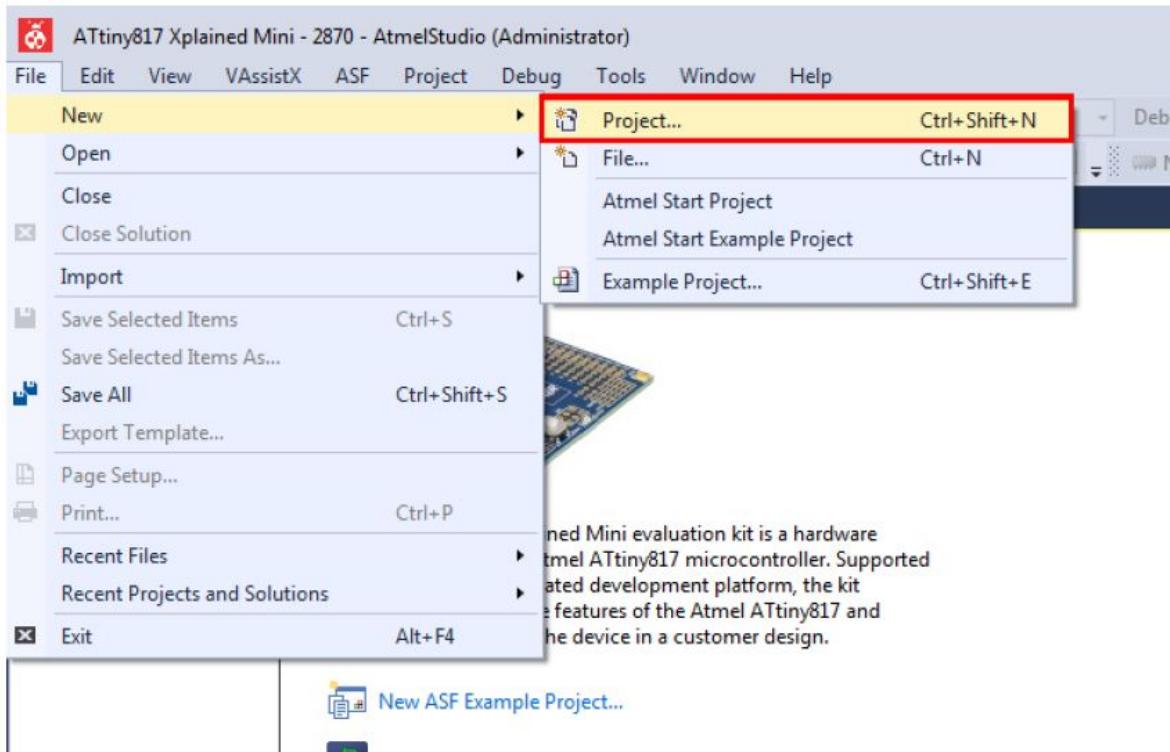
AS Atmel START example projects using this board...
New Atmel START project using this board...

[Video: Create New Project](#)



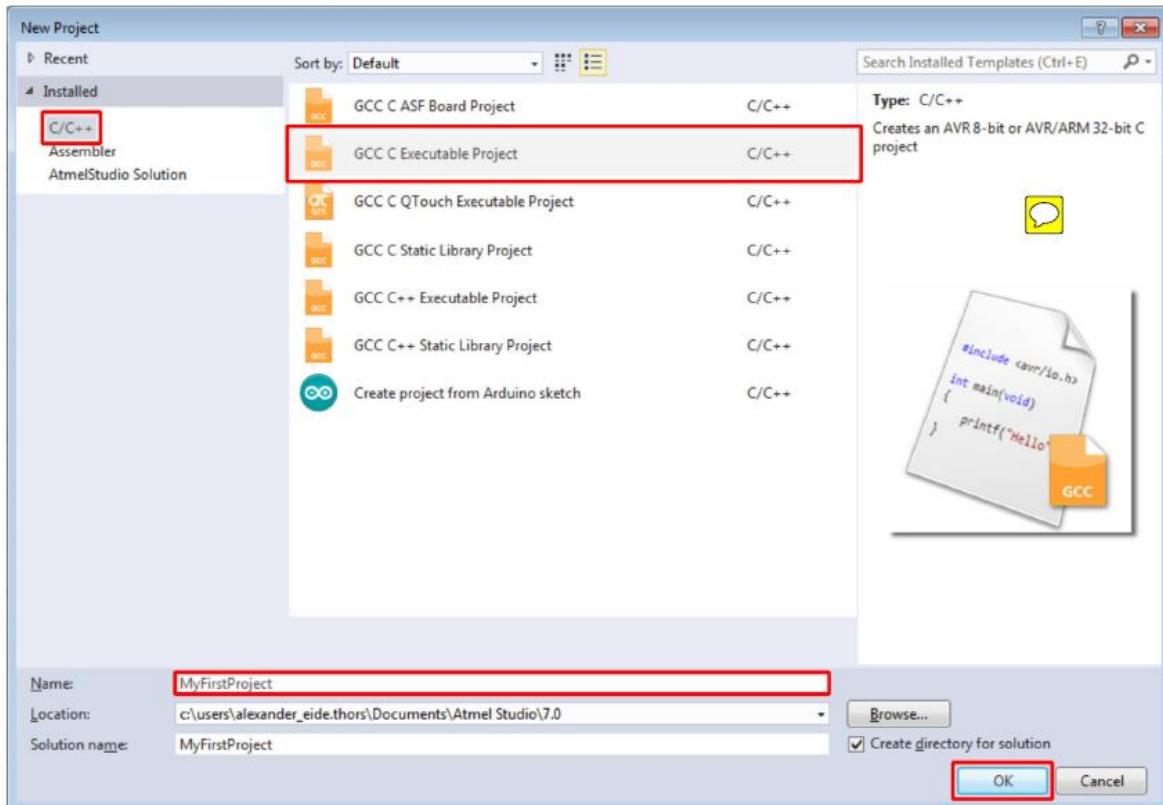
To do: Create a new bare-metal GCC C Executable project for the AVR ATtiny817 device.

1. Open **Atmel Studio**.
2. In **Atmel Studio**, go to **File → New → Project** as depicted in Figure 1-5.

Figure 1-5. Creating a New Project in Atmel Studio

3. The project generation wizard will appear. This dialog provides the option to specify the programming language and project template to be used. This project will use C, so make sure **C/C+** is selected in the upper left corner. Select the **GCC C Executable Project** option from the template list to generate a bare-bones executable project. Give the project a **Name** and click **OK**. See [Figure 1-6](#).

Figure 1-6. New Project Programming Language and Template Selection



Tip: All Atmel Studio projects belong to a solution, and by default, Atmel studio will use the same name for both the newly created solution and the project. The solution name field can be used to manually specify the solution name.



Tip: The *create directory for solution* check-box is checked by default. When this box is ticked, Atmel Studio will generate a new folder with the specified solution name at the location specified by the Location field.

About Project Types

Table 1-1. Project Types

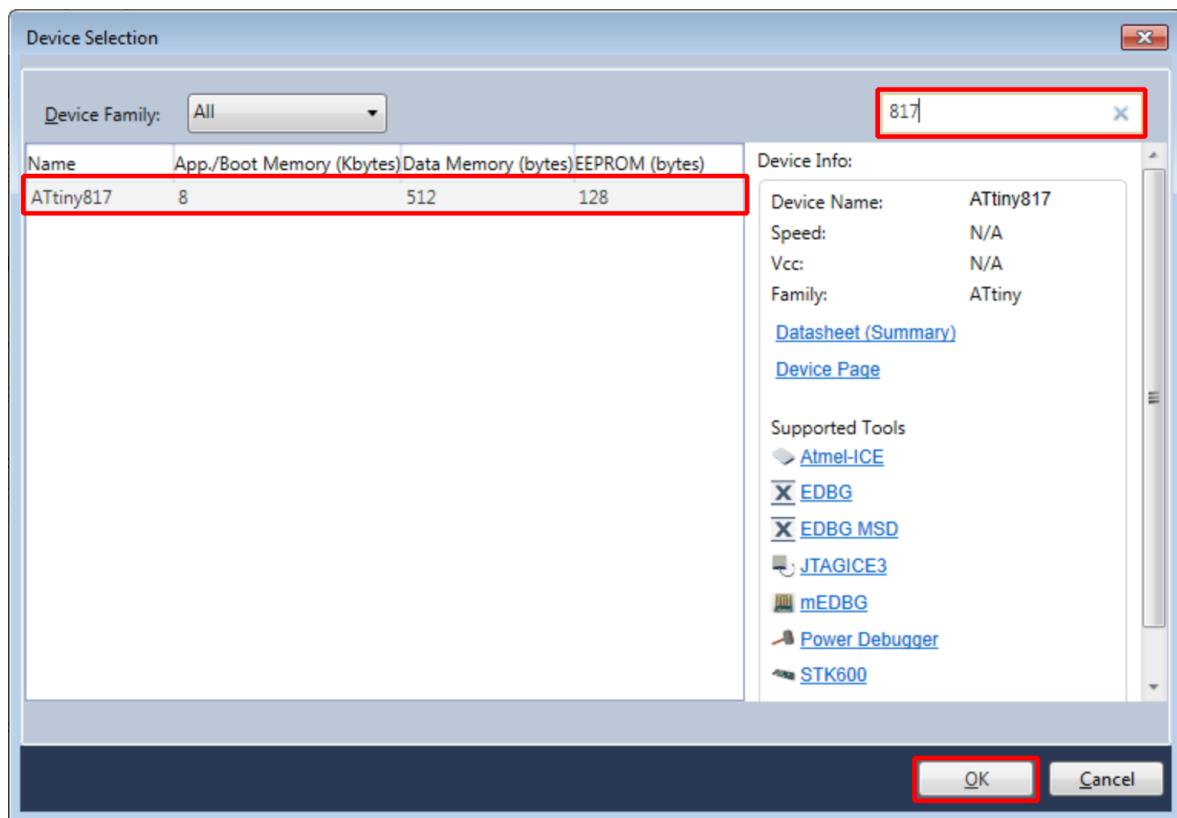
Category	Project templates	Description
C/C++	GCC C ASF Board Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit ASF3 Board project. Choose between the different boards supported by ASF3.
C/C++	GCC C Executable Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit GCC project.

Category	Project templates	Description
C/C++	GCC C Static Library Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit GCC static library(LIB) project. This pre-compiled library (.a) can be used to link to other projects (closed source) or referenced from applications that need the same functionality (code reuse).
C/C++	GCC C++ Executable Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit C++ project.
C/C++	GCC C++ Static Library Project	Select this template to create an AVR 8-bit or AVR/ARM 32-bit C++ static library (LIB) project. This pre-compiled library (.a) can be used to link to other projects (closed source) or referenced from applications that need the same functionality (code reuse).
Assembler	Assembler Project	Select this template to create an AVR 8-bit Assembler project.
Category	Project Templates	Description



Attention: This table only lists the default project types. Other project types may be added by extensions.

4. Next, it is necessary to specify which device the project will be developed for. A list of devices will be presented in the *Device Selection* dialog, which can be scrolled through, as depicted in [Figure 1-7](#). It is possible to narrow the search by using the *Device Family* drop-down menu or by using the search box. This project will be developed for the ATtiny817 AVR device, so enter "817" in the search box in the top right corner. Select the **ATtiny817** entry in the device list and confirm the device selection by clicking **OK**.

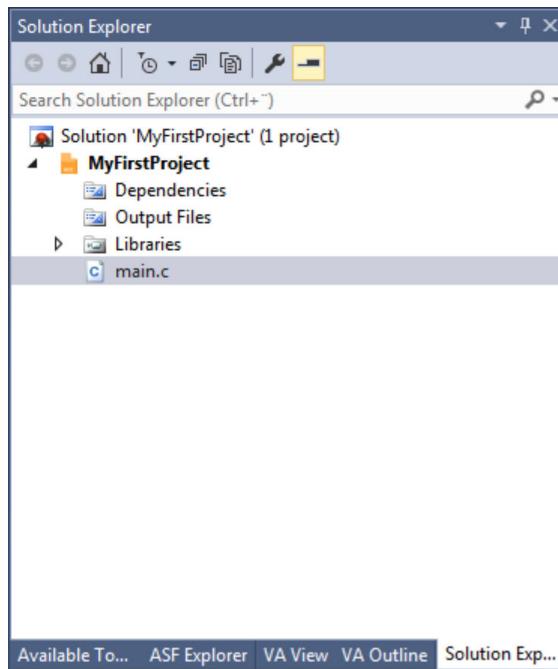
Figure 1-7. New Project Device Selection

Tip: A search for "tiny" will provide a list of all supported ATtiny devices. A search for "mega" will provide a list of all supported ATmega devices. **Tools → Device Pack Manager** can be used to install support for additional devices.



Result: A new GCC C Executable project has now been created for the AVR ATtiny817 device. The **Solution Explorer** will list the contents of the newly generated solution, as depicted in [Figure 1-8](#). If not already open, it can be accessed through **View → Solution Explorer** or by pressing **Ctrl+Alt+L**.

Figure 1-8. Solution Explorer



1.8 Creating From Arduino Sketch

This section will outline the process of creating a new Atmel Studio project from an Arduino Sketch.

[Getting Started Topics](#)

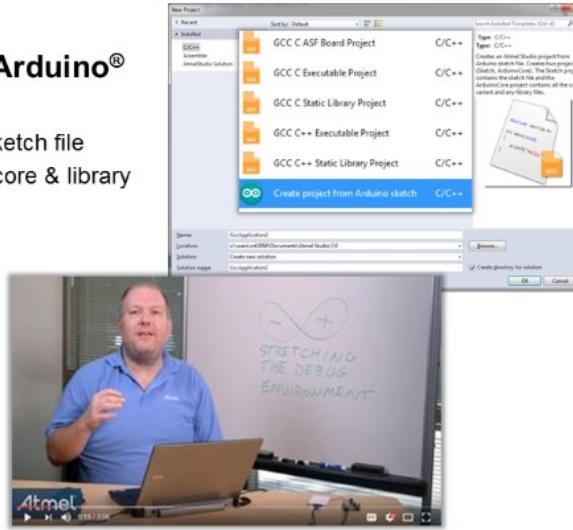


Studio 7: Create from Sketch

In this video:

Create project from Arduino® sketch file

- Sketch project, with sketch file
- Arduino core project, core & library files



[Video: Create from Arduino Sketch](#)



To do: Create a new project from an Arduino Sketch.

1.9

In-System Programming and Kit Connection

This video gives an overview of the Device Programming dialog box, to check the kit connection. The ATtiny817 Xplained Pro kit has an on-board embedded debugger (EDBG) which eliminates the need for a dedicated programmer/debugger. This section will also go through the process of associating the EDBG with your project.

[Getting Started Topics](#)



Studio 7: In System Programming

In this video:

Kit Autodetection

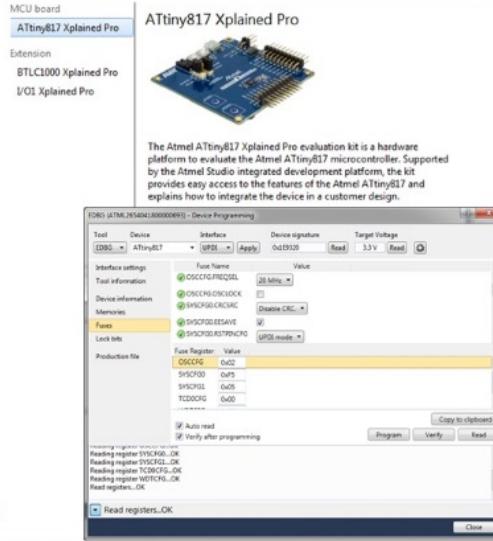
- Xplained Pro MCU & Extension Boards
- Key links

Device Programming Dialog

- Device signature & target voltage
- Tool/device information
 - Device silicon version
- Memories
 - Flash, EEPROM
- Fuses (Config bits equivalent)
- Project output files

AVR, SAM and PIC Differences

- (in terms of) Memory programming



Video: Kit Connection and In-System Programming



To do: Associate the EDBG on your ATtiny817 Xplained Pro kit with your project.

1. Connect the **ATtiny817 Xplained Pro** board to the computer using the provided Micro-USB cable. The kit page should be present in Atmel Studio as in the figure below.

Figure 1-9. ATtiny817 Xplained Pro Start Page

The screenshot shows the 'Start Page' for the ATtiny817 Xplained Pro board in Atmel Studio 7. The top navigation bar includes tabs for 'ATtiny817 Xplained Pro - 0150' and 'Start Page'. On the left, a sidebar lists 'MCU board' (selected) and 'Extension'. The main content area features a large image of the blue ATtiny817 Xplained Pro evaluation kit. Below the image is a descriptive text block: 'The Atmel ATtiny817 Xplained Pro evaluation kit is a hardware platform to evaluate the Atmel ATtiny817 microcontroller. Supported by the Atmel Studio integrated development platform, the kit provides easy access to the features of the Atmel ATtiny817 and explains how to integrate the device in a customer design.' There are two blue links: 'Atmel START example projects using this board...' and 'New Atmel START project using this board...'. A 'Launch Data Visualizer' button is also present. Under 'External Links', there are three blue links: 'Technical Documentation', 'ATtiny817 Device Datasheet', and 'Xplained Pro Hardware Development Kit (HDK) User Guide'. A section titled 'Kit Details' contains the following table:

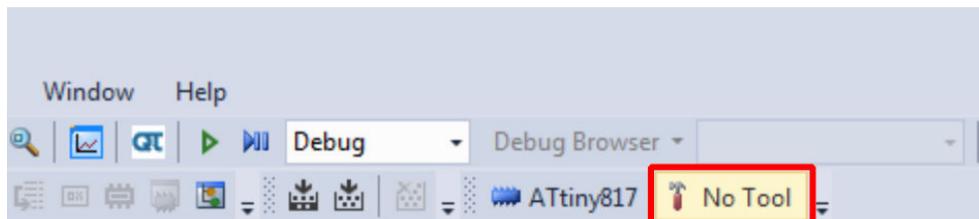
Serial number	ATML2654041800000150
Board name	ATtiny817 Xplained Pro
Manufacturer	Atmel
Target name	ATtiny817
Interfaces	SPI TWI GPIO CDC

At the bottom left, there is a checkbox labeled 'Show page on connect' with the value 'Update board database'.

- 1.1. There are links to documentation for the board and data sheet for the device.
- 1.2. It is possible to create an Atmel START project for the board. Clicking on the Atmel START links project links will bring you into Atmel START where you get options for this specific board.
2. Opening the **Programming Dialog** by Tools → Device Programming.
 - 2.1. Select EDBG Tool and assure that Device = ATtiny817, then you may read Device Signature and Target Voltage.

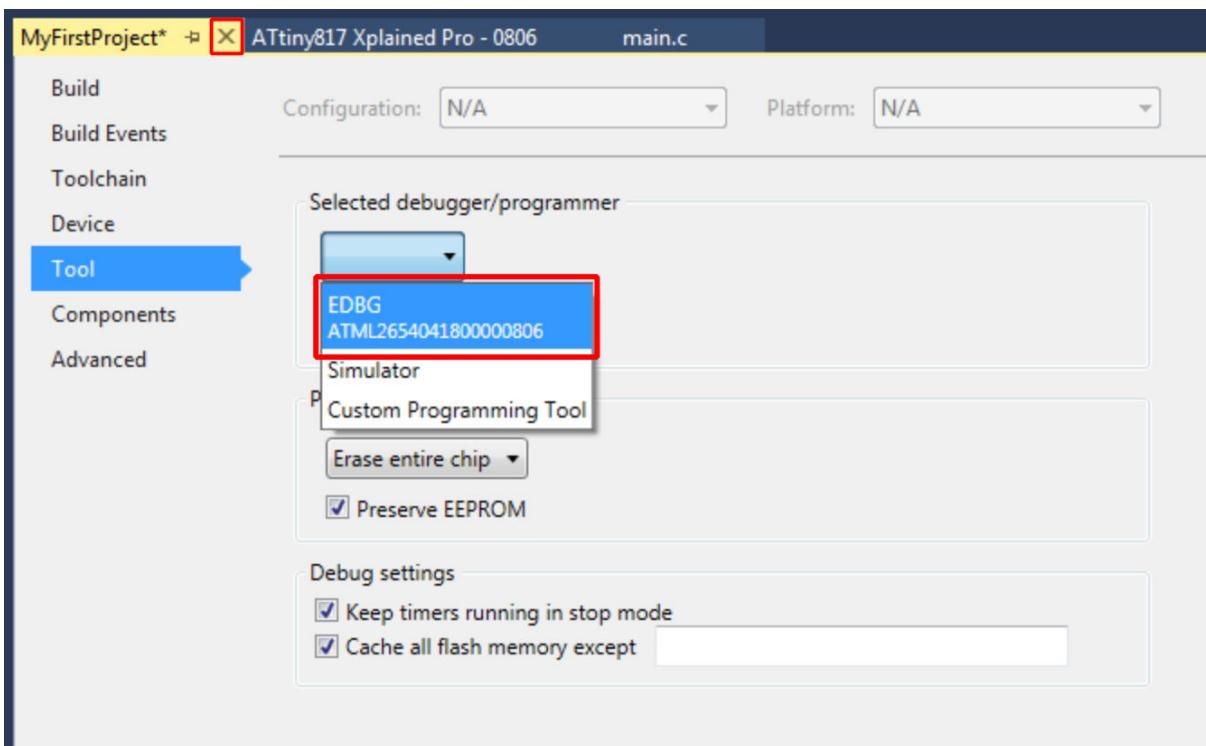
- 2.2. Interface settings: You may see and change the interface clock frequency.
 - 2.3. Tool information: Shows information about the EDBG tool.
 - 2.4. Device information: Shows information about the device. Note that you can also see the silicon revision of the device. This may be useful in customer support cases.
 - 2.5. Memories: May program the flash, EEPROM, and user signature separately from the files.
 - 2.6. Fuses: Read and set fuses, for instance, oscillator frequency (16 or 20 MHz), brown-out voltage detection etc.
 - 2.7. Lock bits: Lock memory.
 - 2.8. Production file: Program the device using a production file to program flash, EEPROM, and user signatures.
 - 2.9. Note that AVR has flash in the HEX file and EEPROM in the EEP files, while PIC has everything, even fuses, in a HEX file.
 - 2.10. For instance, SAML21J devices don't have EEPROM (may be emulated in flash). It also has a security bit option to lock the device.
3. **Create a new project** by selecting File → New project, select for instance C executable project, select the device by filtering on the device name. Different project types are discussed in another Getting Started video.
 4. If a project is selected, click the **Tool** button located in the top menu bar to open the tool dialog as indicated in the figure below.

Figure 1-10. Tool Button



5. The **Tool** tab of the **Project Properties** will open. In the drop-down menu, select the **EDBG** tool, as indicated in the figure below. The interface should automatically initiate to UPDI (Unified Programming Debugging Interface).

Figure 1-11. Select Debugger/Programmer in Project Properties



Tip: The serial number of the tool will accompany its name in the drop-down menu. This serial number is printed on the backside of each tool, allowing differentiation when more than one is connected.



Tip: These steps can always be repeated if a different tool should be used for the next debug/program session.



On the ATtiny817 Xplained Pro, the EDBG is permanently connected to the target MCU, but for a custom hardware solution it is necessary to ensure the target device is powered and properly connected before a debug session can be launched.



Result: The tool to be used by Atmel Studio when a debug/programming session is launched, has now been specified.

1.9.1 Settings Verification

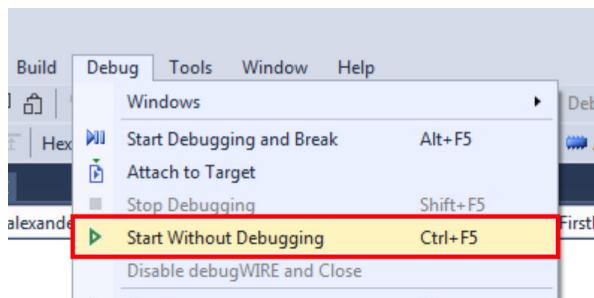
This section is a guide to verifying the tool and project configuration setup by compiling the empty project and writing it to the ATtiny817.



To do: Verify the tool and project configuration setup done in the previous sections.

1. Click the **Start Without Debugging** button located in the **Debug** menu, as shown in the figure below. This will compile the project and write it to the specified target MCU using the configured tool.

Figure 1-12. Start Without Debugging

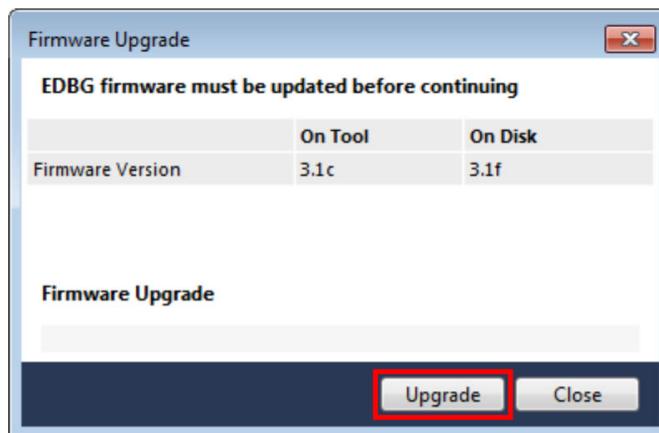


2. When *Atmel Studio 7* builds the project (automatically done when pressing **Start Without Debugging**), several **generated output files** will show up in the Solution Explorer window. The following output files are generated:
 - 2.1. EEP file: EEPROM content written to the device.
 - 2.2. ELF file: Contains everything written to the device, including program, EEPROM, and fuses.
 - 2.3. HEX file: Flash content written to the device.
 - 2.4. LSS file: Disassembled ELF file.
 - 2.5. MAP file: Linker info, what did the linker do, decisions about where to put things.
 - 2.6. SREC file: Same as HEX but in Motorola format.



Info: If there is new firmware available for the selected tool, the **Firmware Upgrade** dialog will appear, as depicted in [Figure 1-13](#). Click the **Upgrade** button to start the firmware upgrade.

Figure 1-13. Firmware Upgrade Dialog



Depending on the state of the connected tool and the actual firmware upgrade, the upgrade may fail on the first attempt. This is normal and can be resolved by disconnecting and reconnecting the kit before clicking **Upgrade** again. After the upgrade has completed, the dialog should say "EDBG Firmware Successfully Upgraded". **Close** the dialog box and make a new attempt at programming the kit by clicking the **Start Without Debugging** button again.

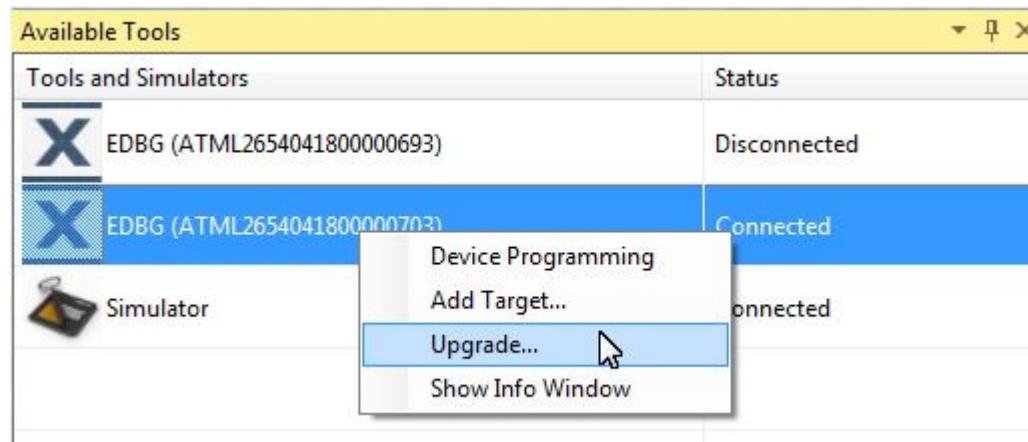


Result: By compiling the empty project and writing it to the ATtiny817 the following has been verified:

- The project is configured for the correct MCU
- The correct tool has been selected
- The tool's firmware is up-to-date

Under *View > Available Tools* you are able to see a list of available or recently used Tools. Here you can specifically ask *Atmel Studio 7* to upgrade the firmware for a tool.

Figure 1-14. Atmel Studio 7 Available Tools (on view menu)



1.10 I/O View and Other Bare-Metal Programming References

This section describes how you would typically write code in *Studio 7*, independent of a software configuration tool or framework, i.e. bare-metal. This is covered both as video (linked below) and hands-on document. The main focus is on each of the relevant programming references, how each is accessed, and what each is used for. The project context is to turn ON an LED, then blink with a delay. Although the ATtiny817 Xplained Pro is used the principles are general enough to use with any kit in *Studio 7*, though the principles apply to most devices supported in *Studio 7*.

[Getting Started Topics](#)

In this video:
Context:

- Turn on LED, then blink with delay.

Programming References:
(How to easily access & what to use each for)

- Device datasheet
- Datasheet (from IO view)
- IO view (debugging)
- Kit user-guide & schematics
- Device header files
- Editor (Visual Assist)
- AVR® LibC
- Atmel START

The screenshot shows the Atmel Studio 7 interface with the "I/O View & Bare-Metal Prog. Refs." tab selected. On the left, there's a sidebar with the Microchip logo and a video thumbnail titled "In this video: Context". The main area has two panes: one for "Programming References" listing the items above, and another for "I/O View" showing a table of registers and their bit fields. The "I/O View" pane shows the following table for the ADC register:

Name	Address	Value	Bits
CTRLA	0x600	0x05	0:05
ENABLE		0x01	0:01
FREERUN		0x00	0:00
REFSEL		0x01	0:01
RUNSTBY		0x00	0:00
CTRLB		0x04	0:04
CTRLC		0x00	0:00
CTRLD		0x00	0:00
CTRLE		0x04	0:04
SAMPCTRL		0x00	0:00
MUXPOS		0x05	0:05

Below the table, a detailed description of the RUNSTBY bit is provided:

Bit 7 - RUNSTBY: Run in Standby
This bit determines whether the ADC needs to run when the chip is in standby sleep mode.

[Video: I/O View and Bare-metal programming references](#)

The list below is an overview of the programming references which are typically used. Particular emphasis is placed on I/O View, which provides a way to navigate data sheet register descriptions when editing or debugging, as well as to understand the current configuration when debugging. This second use of I/O view when debugging is also used to test new register configurations.

This topic is closely related to both [Debugging 3: I/O View Memory View and Watch](#) as well as [Editor: Writing and Re-Factoring Code \(Visual Assist\)](#).

- Device data sheet
- Data sheet (from I/O view)
- Kit user guide and schematics
- I/O View (debugging)
- Editor (Visual Assist)
- Device header files
- AVR Libc (AVR specific)
- Atmel START: ATtiny817 project

In the process the following code is written. Although the code is simple, the decision process, using the list of programming references above, is described.

```
#include <avr/io.h>
#define F_CPU 3333333
#include <util/delay.h>

int main(void)
{
    PORTB.DIR = PIN4_bm;

    while (1)
    {
        delay_ms(500);
        PORTB.OUTTGL = PIN4_bm;
    }
}
```



Be sure to keep the `#include <avr/io.h>` line at the top of *main.c*. This header file will include the correct register map for the selected device, and without this statement, the compiler will not recognize any of the macros referenced in the code above.

Device Data Sheet (PDF)

Although I/O View allows easy access to navigate the data sheet at a register level, the PDF version still has a role. The device data sheet, in PDF format, tends to be used at least to get an understanding of the peripheral, through the **block diagram** and **functional description**. For example, to understand the PORT peripheral of the ATtiny817, we consulted the *PORT Block Diagram* and *Functional Description > Principle of operation* sections of the data sheet. These two sections together, connecting the description to the diagram, give a basic understanding of the PORT peripheral.

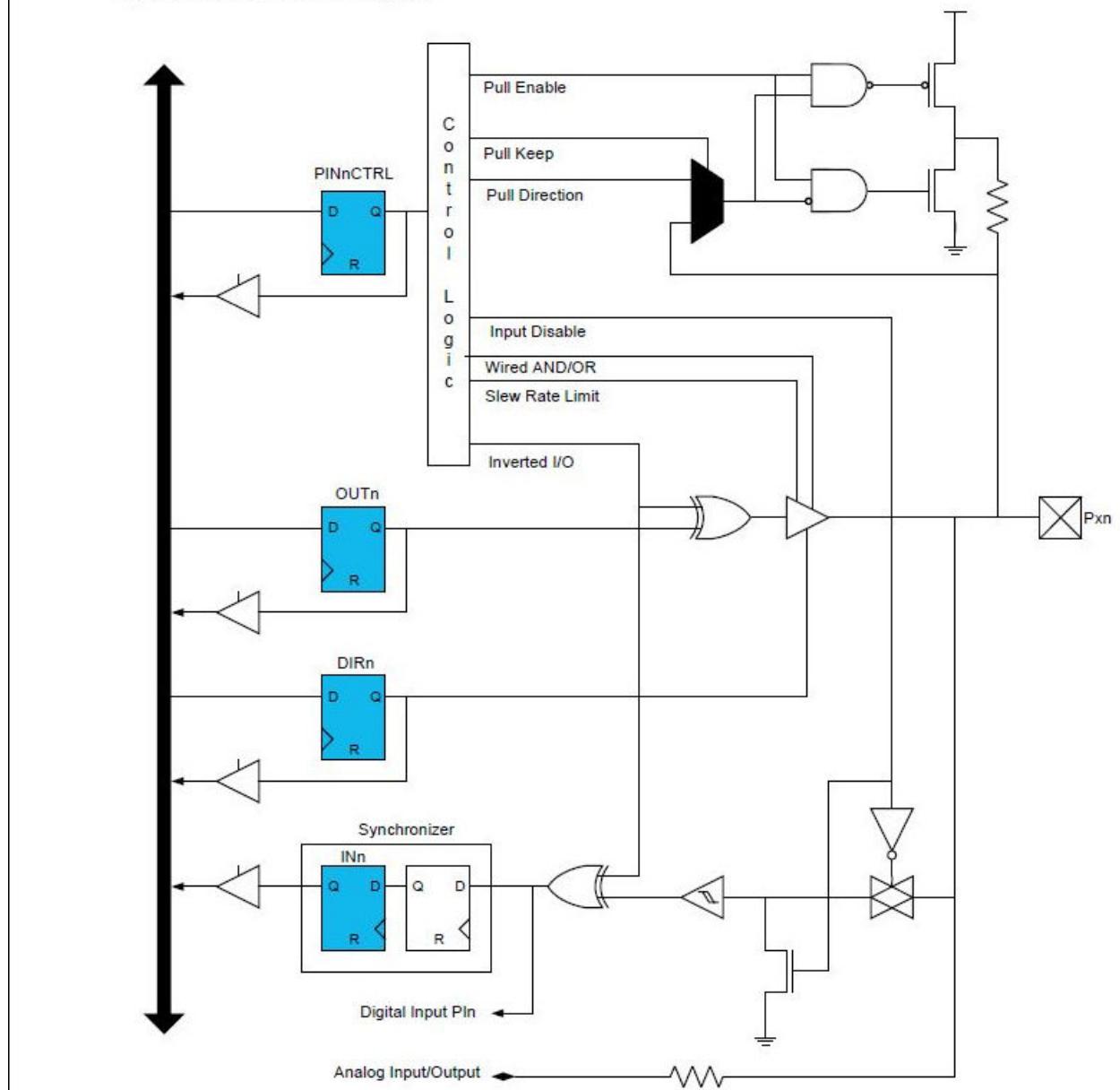
Figure 1-15. PORT Block Diagram from the PDF Data Sheet**17.3. Block Diagram (ATTiny817 data sheet extract: PORT - I/O Pin Controller chapter)****Figure 17-1. PORT Block Diagram**

Figure 1-16. Principle of Operation from the PDF Data Sheet of ATtiny817**17.6. Functional Description (ATtiny817 data sheet extract: PORT - I/O Pin Controller chapter)****17.6.1. Principle of Operation**

The I/O pins of the device are controlled by PORT peripheral registers. Each of the port pins has a corresponding bit in the **Data Direction (PORT.DIR)** and **Data Output Value (PORT.OUT)** registers to enable that pin as an output and to define the output state. For example, pin PB3 is controlled by DIR[3] and OUT[3] of the PORTB instance.

The direction (input or output) of each pin in a pin group is configured by the PORT.DIR register.

When the direction is set as output, the corresponding bit in the PORT.OUT register will select the level of the pin. If bit n in PORT.OUT is written to '1', pin n is driven HIGH. If bit n in PORT.OUT is written to '0', pin n is driven LOW. Pin configuration can be set by writing to the Pin n Control registers (PORT_PINnCTRL) with n=0..7 representing the bit position.

The Data Input Value (PORT.IN) is set as the input value of a PORT pin with resynchronization to the Main Clock. To reduce power consumption, these input synchronizers are clocked only when the value of the Input Sense Configuration bit field (ISC) in PORT.PINnCTRL is not INPUT_DISABLE. The value of the pin can always be read, whether the pin is configured as input or output.

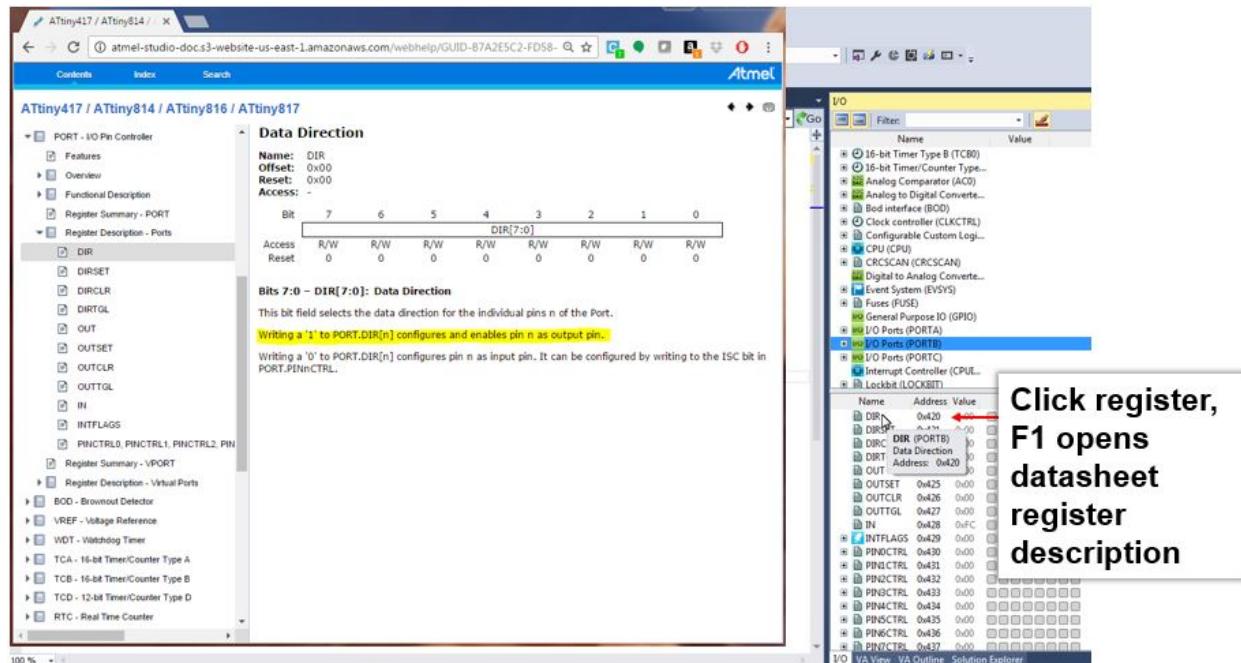
Note: We used the **device data sheet** for the **peripheral block diagram**, as well as a description of the **PORT DIR and OUT registers**.

I/O View Data Sheet

Studio 7 allows to easily access the data sheet register descriptions by clicking F1 on the relevant register description. The HTML version of the data sheet opens online (by default). The data sheet will open in the context of the relevant register description.

Note: In this way we use the **Data sheet from I/O View** to understand that:

1. Writing a '1' to PORT.DIR[n] configures and enables pin n as an output pin.
2. If OUT[n] is written to '0', pin n is driven low.

Figure 1-17. Opening an Online Data Sheet from I/O View

I/O View (Debugging)

This functionality can directly be tested by starting a debug session, using *Start Debugging and Break*. So we are now able to begin testing functionality, as shown in the image below.

I/O View is covered in more detail in [Debugging 3: I/O View Memory View and Watch](#).

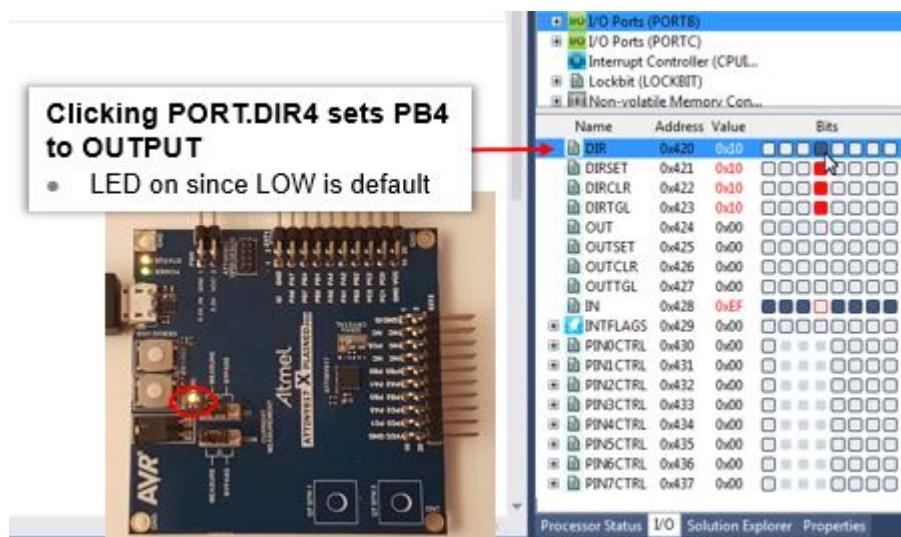
Note: I/O View when debugging is used to:

1. Verify that writing a '1' to PORT.DIR4, sets pin as OUTPUT, LOW by default to LED turns ON.
2. Verify that writing a '1' to PORT.OUT4, turns OFF the LED.

Table 1-2. Atmel Studio Button Functionality (Programming and Launching Debug Sessions)

Button	Functionality	Keyboard Shortcut
	Start Debugging and Break	Alt + F5
	Attach to Target	
	Start Debugging	F5
	Break All	Ctrl + Alt + Break
	Start Without Debugging	Ctrl + F5

Figure 1-18. Turning ON/OFF Kit LEDs Through Manipulating I/O View Registers when Debugging



Downloading Studio 7 Documentation

The data sheet can also be downloaded by using the Studio 7 help system. In this case, a similar functionality will work offline. This is described here: [Downloading Offline Documentation](#).

Atmel Studio 7 Editor (Visual Assist)

The Studio 7 Editor, powered by [Visual Assist](#) has powerful features to help you write and refactor code, as well as easily navigate large projects. Suggestion functionality is shown in [Figure 1-19](#), while an overview of the code navigation is shown in [Figure 1-20](#). In the next section, [Editor: Writing and Refactoring Code \(Visual Assist\)](#), the editor features are covered in more detail.

Figure 1-19. Suggestion Functionality in the Studio 7 Editor for Writing Code

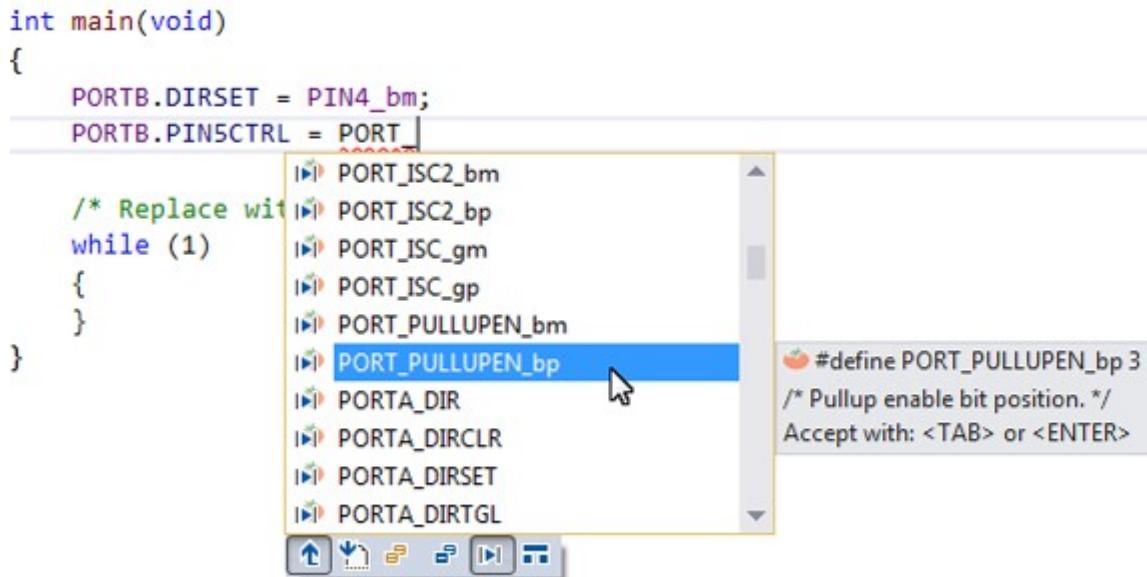


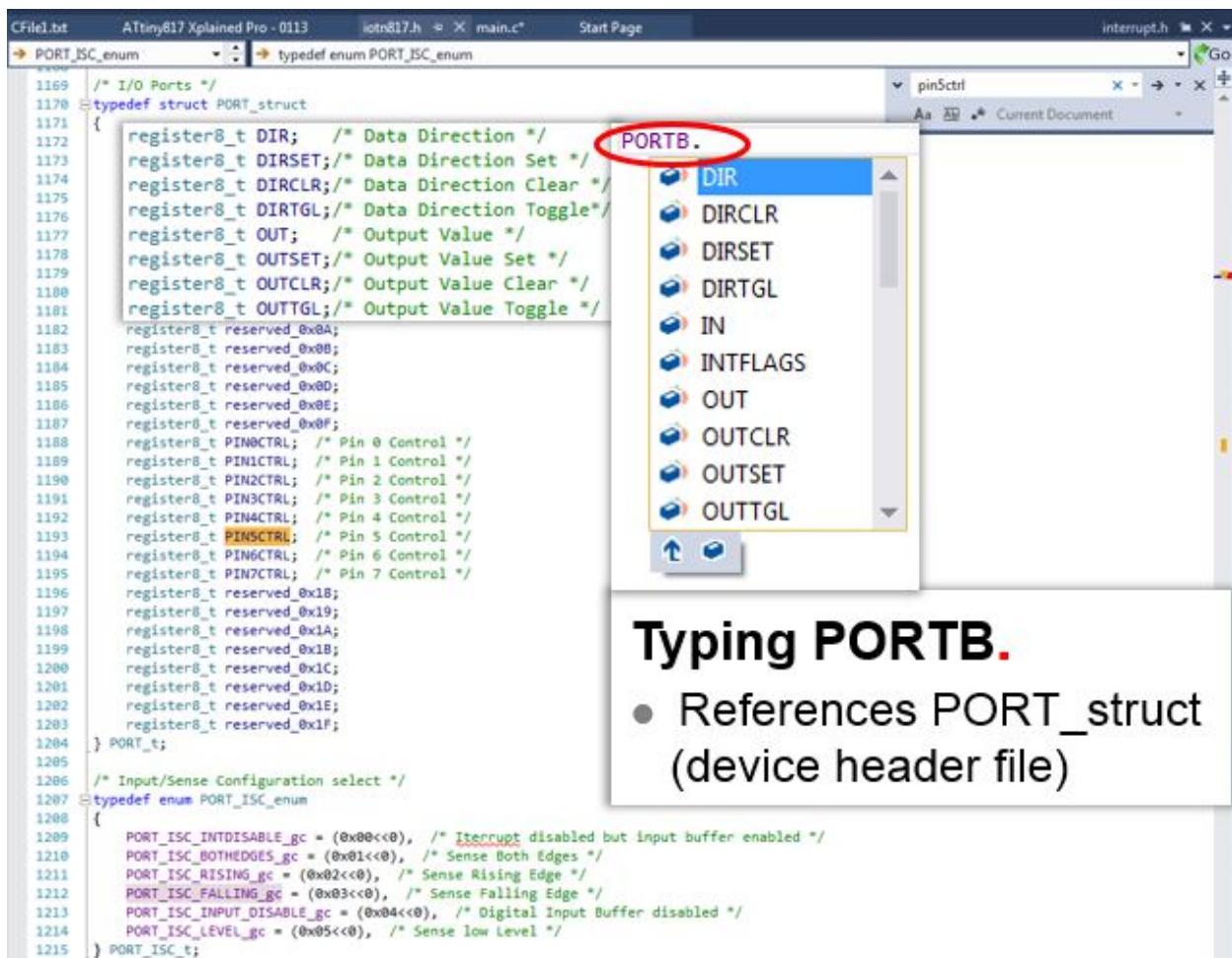
Figure 1-20. Atmel Studio 7 Editor Navigation Overview



Specifically in the video related to this section, the editor is used for the following.

Device Header Files

Through the *Goto Definition* functionality of the editor, it is easy to access the MCU device header files, i.e. by clicking on any register and then clicking on the goto button, or typing Alt+G. Writing PORTB. gives a suggestion list of potential registers, from the PORT structure, shown in figure [Suggestion lists and the MCU device header files](#). For more information about how the AVR header files are structured, see [AVR1000](#) for more information.

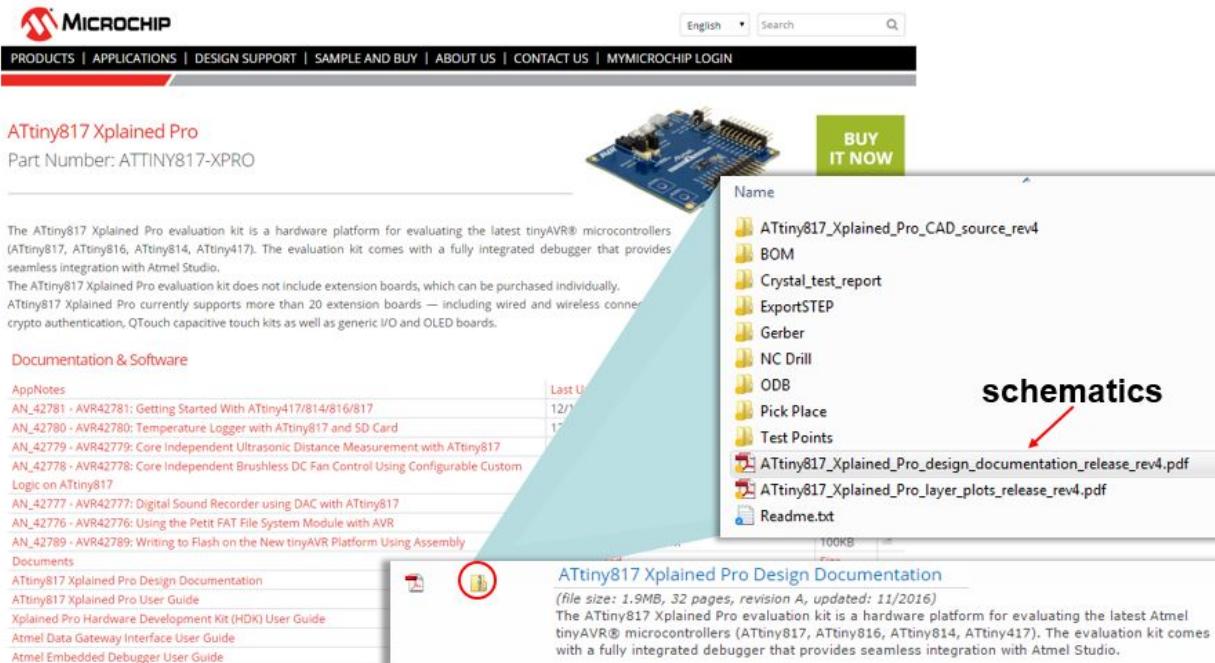
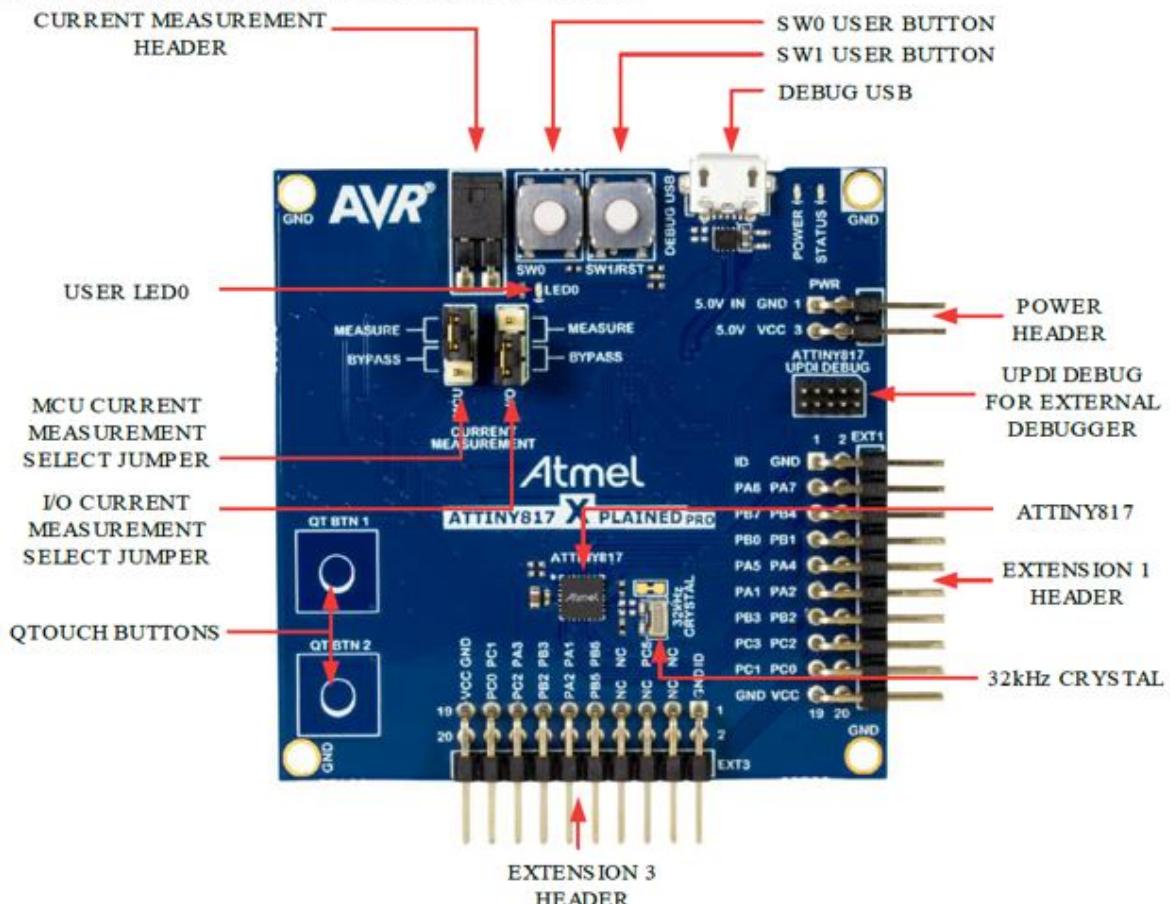
Figure 1-21. Suggestion lists and the MCU device header files

Typing PORTB.

- References PORT_struct (device header file)

Kit Schematics and User Guide

The kit schematics and user guide are useful to understand the MCU pin connections on the kit. Full schematics and kit design files, such as Gerbers, are available on www.microchip.com, on the kit's product page.

Figure 1-22. How to Find Schematics for a Particular Development Board**Figure 1-1. ATtiny817 Xplained Pro Evaluation Kit Overview**

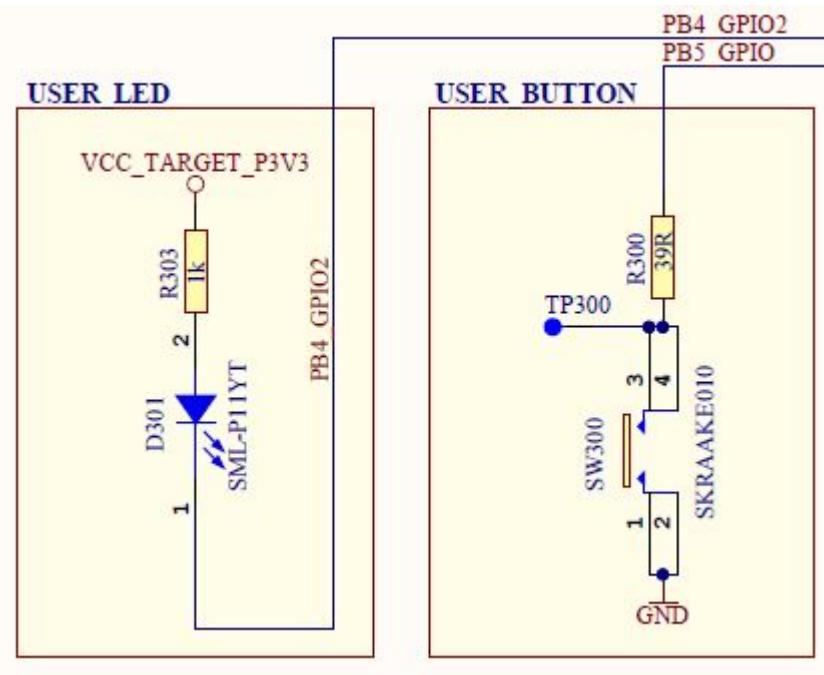
The LED and button are connected to the pins as per the table below, from the [ATtiny817 Xplained Pro User Guide](#).

Table 1-3. ATtiny817 Xplained Pro GPIO Connections

Silkscreen Text	ATtiny817 GPIO Pin
LED0	PB4
SW0	PB5

The ATtiny817 Xplained Pro design documentation schematic shows the connections for the LED and button, as in the figure below.

Figure 1-23. ATtiny827 Xplained Pro GPIO Connection Schematics



From the schematics, it is concluded that:

- The LED can be turned ON by driving PB4 low.
- SW0 is connected directly to GND and to PB5 through a current limiting resistor.
- SW0 does not have an external pull-up resistor.
- SW0 will be read as '0' when pushed and as '1' when released, if the ATtiny817 internal pull-up is enabled.

AVR Libc

All the references covered to this point are just as relevant for SAM as for AVR, however, as the name suggests, this one is specific to AVR. [AVR Libc](#) is a Free Software project whose goal is to provide a high-quality C library for use with GCC on AVR microcontrollers. Together, avr-binutils, avr-gcc, and avr-libc form the heart of the Free Software toolchain for the AVR microcontrollers. Further, they are accompanied by projects for in-system programming software ([avrdude](#)), simulation ([simulavr](#)), and debugging ([avr-gdb](#), [AVaRICE](#)).

The [library reference](#) is usually a quick interface into AVR Libc, as shown in [Figure 1-24](#). One can quickly search the page for a relevant library. Relevant header files, which should be added to the project, are

indicated in the module name. For example searching for "interrupts", the relevant include will be #include <avr/interrupt.h>. Clicking into the module, a list of available functions and relevant interrupt callbacks can be found, as shown in Figure 1-25.

Figure 1-24. AVR Libc Library Reference

The screenshot shows a web browser displaying the AVR Libc Library Reference. The URL in the address bar is www.nongnu.org/avr-libc/user-manual/modules.html. The page title is "AVR Libc Home Page". The navigation menu includes "Main Page", "User Manual", "Library Reference" (which is circled in red), "FAQ", and "Example Projects". Below the menu, there is a section titled "Modules" with the sub-section "Modules". A list of C header files is provided, each with a brief description. The header file "avr/interrupt.h" is highlighted with a yellow background and has the word "Interrupts" underlined. The list includes:

- <alloca.h>: Allocate space in the stack
- <assert.h>: Diagnostics
- <ctype.h>: Character Operations
- <errno.h>: System Errors
- <inttypes.h>: Integer Type conversions
- <math.h>: Mathematics
- <setjmp.h>: Non-local goto
- <stdint.h>: Standard Integer Types
- <stdio.h>: Standard IO facilities
- <stdlib.h>: General utilities
- <string.h>: Strings
- <time.h>: Time
- <avr/boot.h>: Bootloader Support Utilities
- <avr/cpufunc.h>: Special AVR CPU functions
- <avr/eeprom.h>: EEPROM handling
- <avr/fuse.h>: Fuse Support
- <avr/interrupt.h>: **Interrupts**
- <avr/io.h>: AVR device-specific IO definitions
- <avr/lock.h>: Lockbit Support
- <avr/pgmspace.h>: Program Space Utilities
- <avr/power.h>: Power Reduction Management

Figure 1-25. Using Interrupts with AVR Libc

The screenshot shows a web browser window displaying the AVR Libc Home Page from www.nongnu.org/avr-libc/. The page content is as follows:

- <avr/interrupt.h>: Interrupts**
- Global manipulation of the interrupt flag**
 - The global interrupt flag is maintained in the I bit of the status register (SREG).
 - Handling interrupts frequently requires attention regarding atomic access to objects that could be altered by code running within an interrupt context, see `<util/atomic.h>`.
 - Frequently, interrupts are being disabled for periods of time in order to perform certain operations without being disturbed; see [Problems with reordering code](#) for things to be taken into account with respect to compiler optimizations.
- #define sei()** Enables interrupts by setting the global interrupt mask.
`#define sei()`
- Macros for writing interrupt handler functions**
 - `#define ISR(vector, attributes)`
 - `#define SIGNAL(vector)`
 - `#define EMPTY_INTERRUPT(vector)`
 - `#define ISR_ALIAS(vector, target_vector)`
 - `#define reti()`
 - `#define BADISR_vect`
- Add header file, Relevant IRQ Vector** (highlighted with a callout box)
 - `#include <avr/interrupt.h>`
 - `ISR(ADC_vect)`
 - `{`
 - `// user code here`
 - `}`

Atmel START

[Atmel START](#) is a web-based software configuration tool, for various software frameworks, which helps you getting started with MCU development. Starting from either a new project or an example project, Atmel START allows you to select and configure software components (from **ASF4** and **AVR Code**), such as drivers and middleware to tailor your embedded application in a usable and optimized manner. Once an optimized software configuration is done, you can download the generated code project and open it in the IDE of your choice, including Studio 7, IAR Embedded Workbench, Keil µVision, or simply generate a make-file.

Although Atmel START is a tool for MCU and software configuration, it can still be useful even in bare-metal development, i.e. writing code from scratch using the list of programming references described in this section. Creating a new project for the kit you are using, can be a useful alternative to the board schematic, using the PINMUX view. In addition, the CLOCKS view can be useful to check the default clocks of a device. Furthermore, viewing the configuration code, useful pieces can be pasted back into your project. For example, the AVR Libc delay functions require that the clock frequency is defined, as shown in [Figure 1-28](#). For the ATtiny817 this default value would be: `#define F_CPU 3333333`.

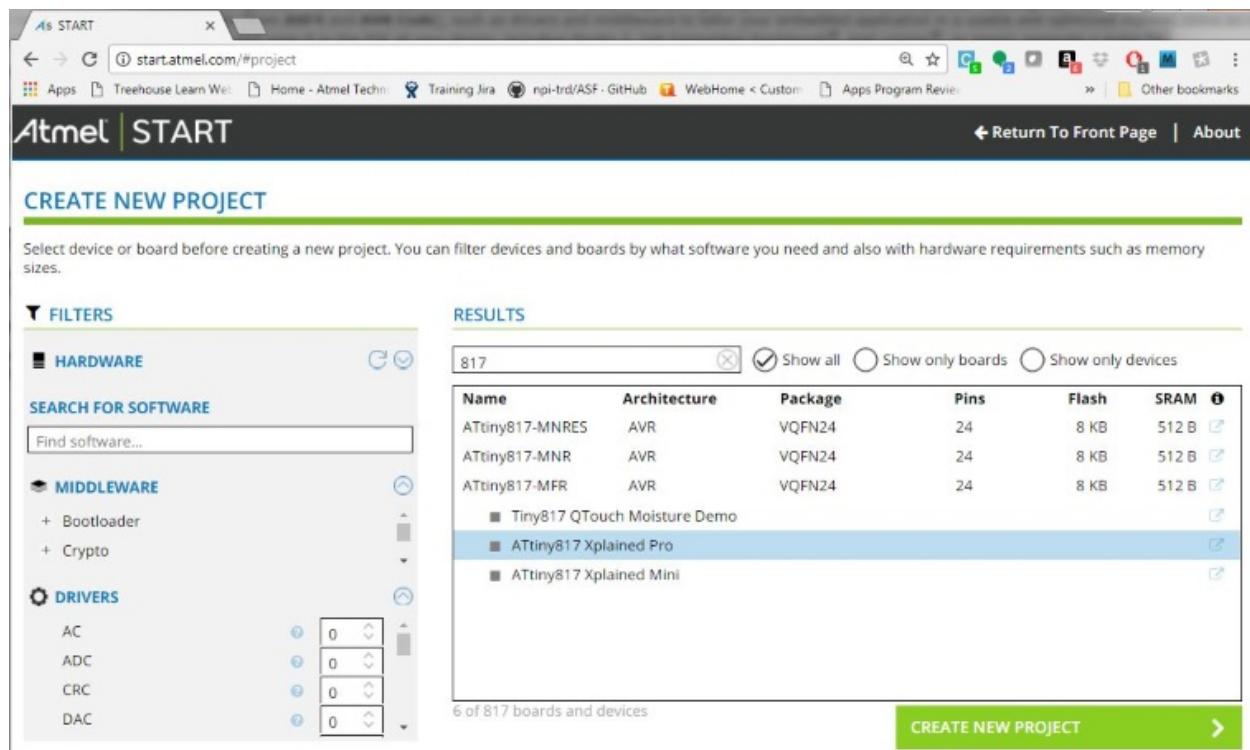
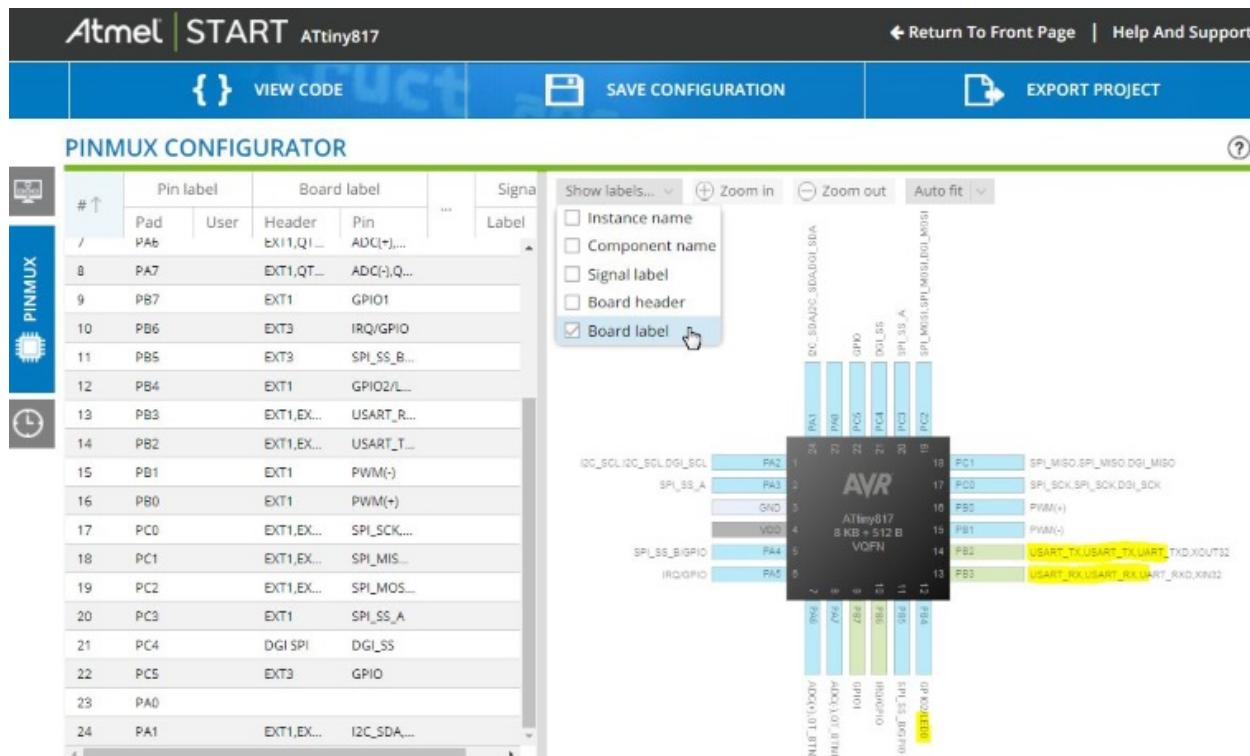
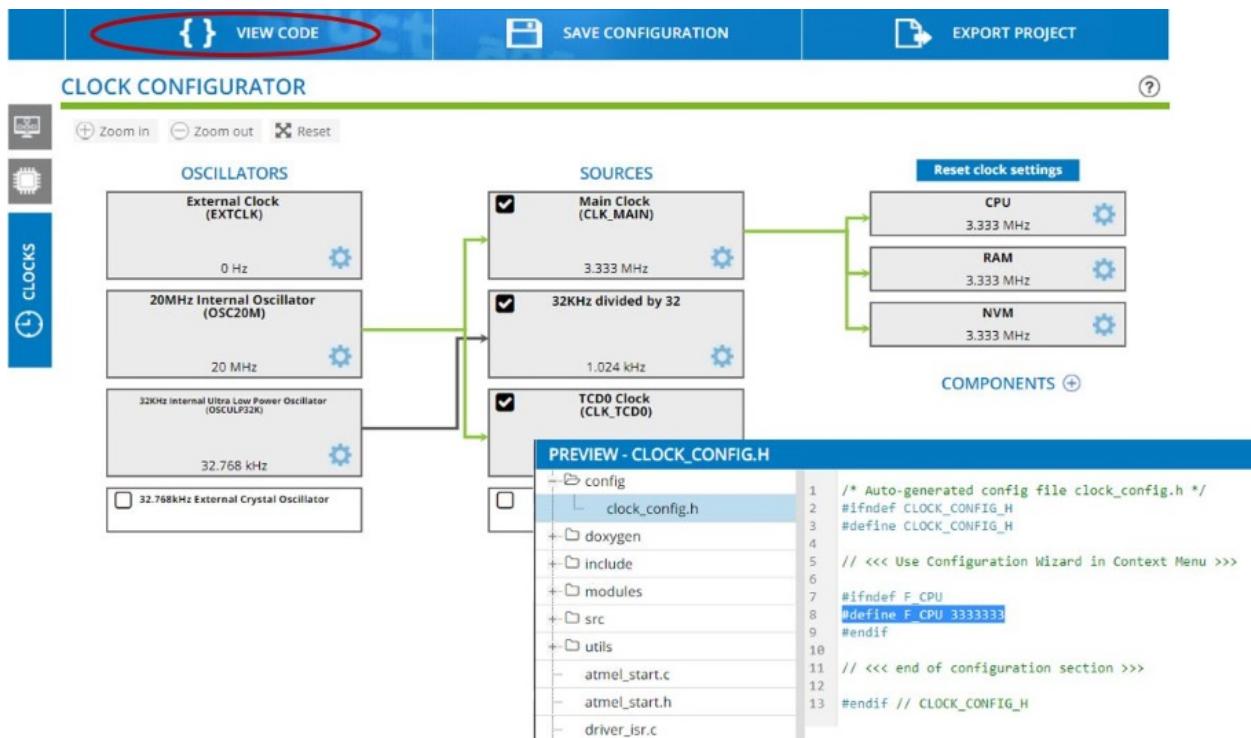
Figure 1-26. Using START to Creating a New Project for a Relevant Board**Figure 1-27. Showing Board Labels in START as an Alternative to the Kit Schematic**

Figure 1-28. Checking Default Clock Configuration and Using VIEW CODE to Find F_CPU Define

1.11 Editor: Writing and Re-Factoring Code (Visual Assist)

The Studio 7 Editor is powered by an extension called *Visual Assist*, a productivity tool for re-factoring, reading, writing, and navigating C and C++ code.

[Getting Started Topics](#)



Studio 7: Editor (Visual Assist)

In this video:

Studio 7 Editor...

Context:

- Turn on LED, when switch pressed
- Polled, then with pin change IRQ

Writing Code

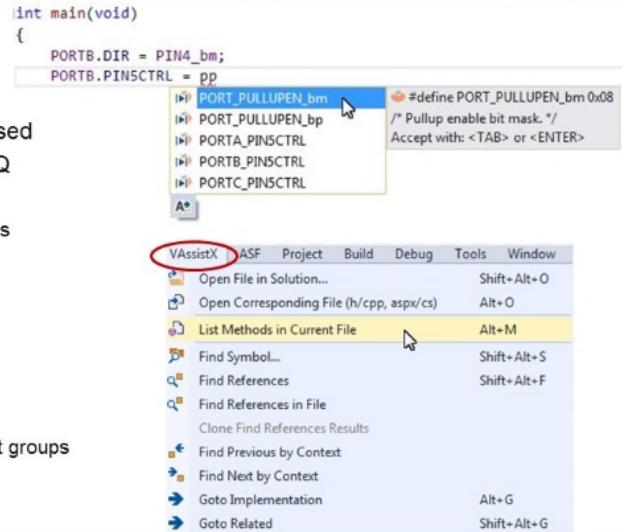
- Suggestion lists, enhanced list boxes
- Visual assist code snippets

Refactoring Code

- Extract method
- Introduce variable
- Contextual rename

Header File Navigation

- Finding enumerators to configure bit groups



Video: Studio 7 Editor (Visual Assist)

1. Starting with the basic functionality from [I/O View and Other Bare-Metal Programming References](#), main.c has the following code:

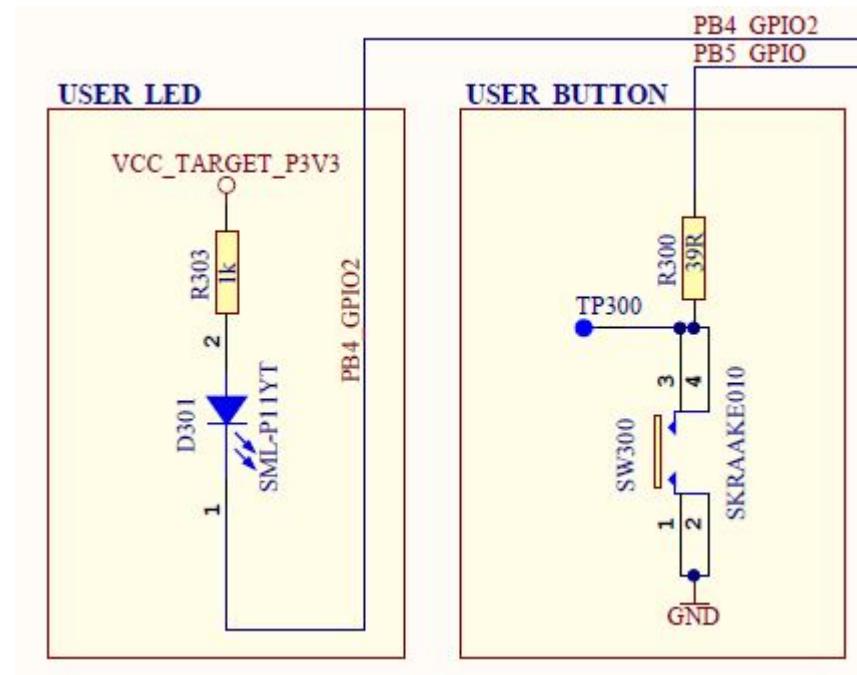
```
#include <avr/io.h>

int main(void)
{
    PORTB.DIR = PIN4_bm;

    while (1)
    {
    }
}
```

The ATtiny817 Xplained Pro design documentation schematic shows the connections for the LED and button, as in the figure below.

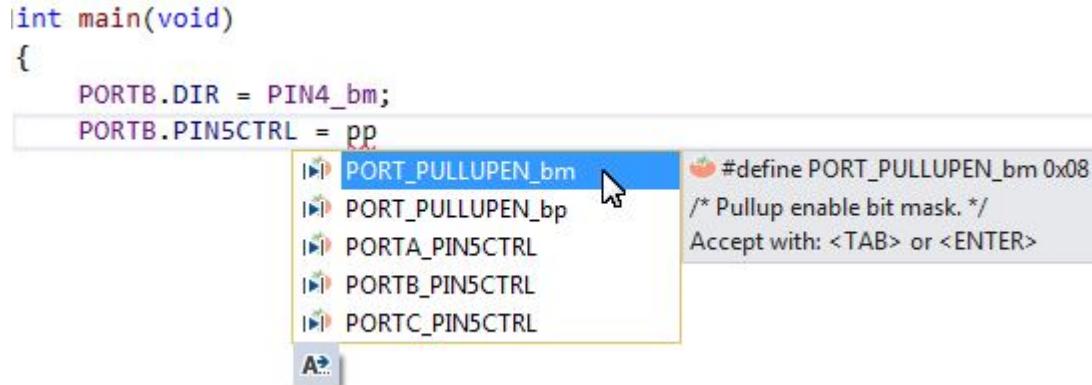
Figure 1-29. ATtiny827 Xplained Pro GPIO Connection Schematics



From the schematics, it is concluded that:

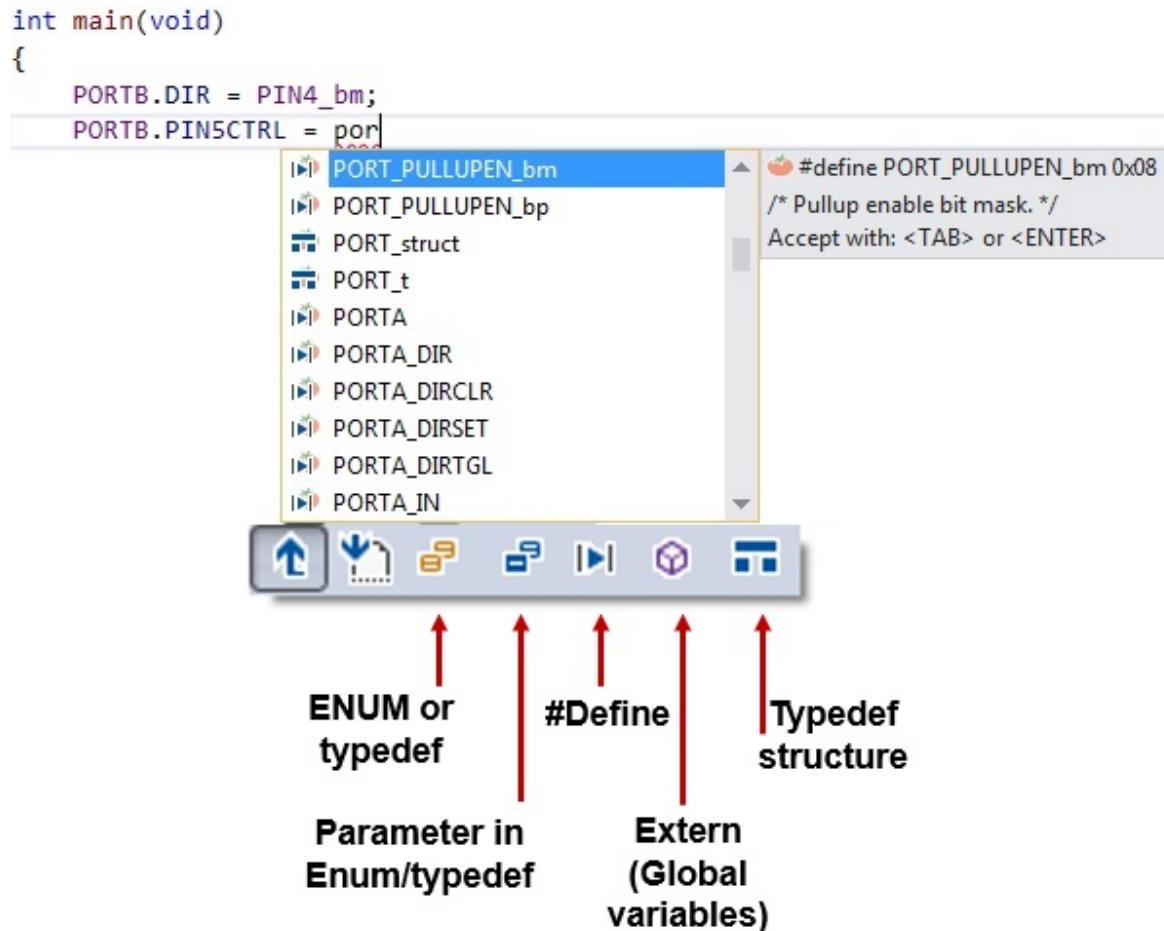
- The LED can be turned ON by driving PB4 low.
- SW0 is connected directly to GND and to PB5 through a current limiting resistor.
- SW0 does not have an external pull-up resistor.
- SW0 will be read as '0' when pushed and as '1' when released, if the ATtiny817 internal pull-up is enabled.

1. Enable the pull-up on PORTB5, **using suggestion list and enhanced list box**. Note that suggestion lists support acronyms, so typing "pp" PORT_PULLUPEN is the top suggestion.



2. However, before hitting enter, first type "POR", then hit CTRL+SPACE. This will bring up the Enhanced Listbox with all possible options.

Now it is possible to filter suggestions by type, as indicated in the picture below.

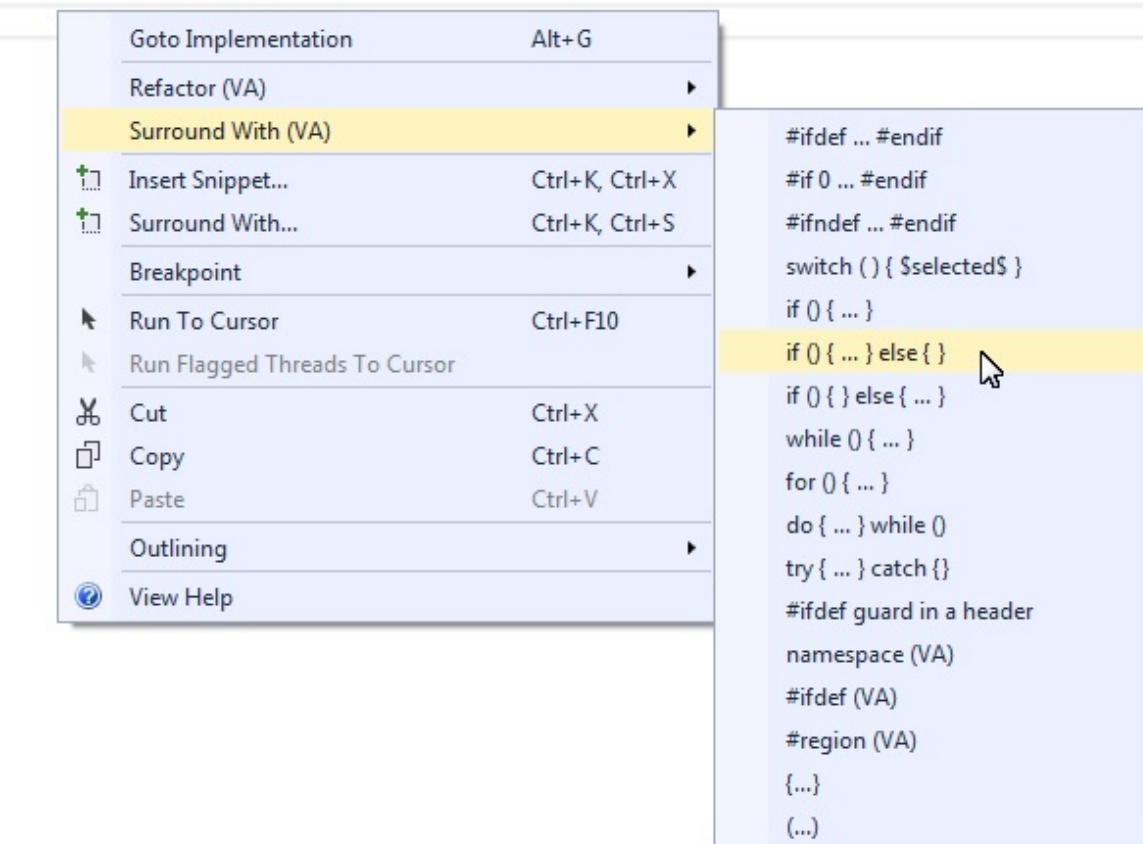


3. Test if SW0 is pressed, using **if(){...}else{...}** visual assist code snippet.
Simply typing "if" will bring up the option. Or, you could *R-click* and choose **Surround With (VA)**, which gives a full list of snippets. This is an editable list, so you can add your own snippets.

```
int main(void)
{
    PORTB.DIR = PIN4_bm;
    PORTB.PIN5CTRL = PORT_PULLUPEN_bm;

    if
        if
        if 0 { ... }
        if 0 { ... } else {} VA Snippet [ Edit ]
        Accept with: <TAB> or <ENTER>
    A+ A-
```

```
int main(void)
{
    PORTB.DIR = PIN4_bm;
    PORTB.PIN5CTRL = PORT_PULLUPEN_bm;
```



4. Test if the switch is pressed, as the `if(){...}else{...}` condition, turn the LED ON if pressed and OFF if not. `main.c` should now look as follows:

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

int main(void)
{
    PORTB.DIRSET = PIN4_bm; /* Configure LED Pin as output */
    PORTB.PIN5CTRL = PORT_PULLUPEN_bm; /* Enable pull-up for SW0 pin */
```

```

while(1)
{
    if (! (PORTB.IN & PIN5_bm)) /* Check switch state */
    {
        PORTB.OUTCLR = PIN4_bm; /* Turn LED off */
    }
    else
    {
        PORTB.OUTSET = PIN4_bm; /* Turn LED on */
    }
}

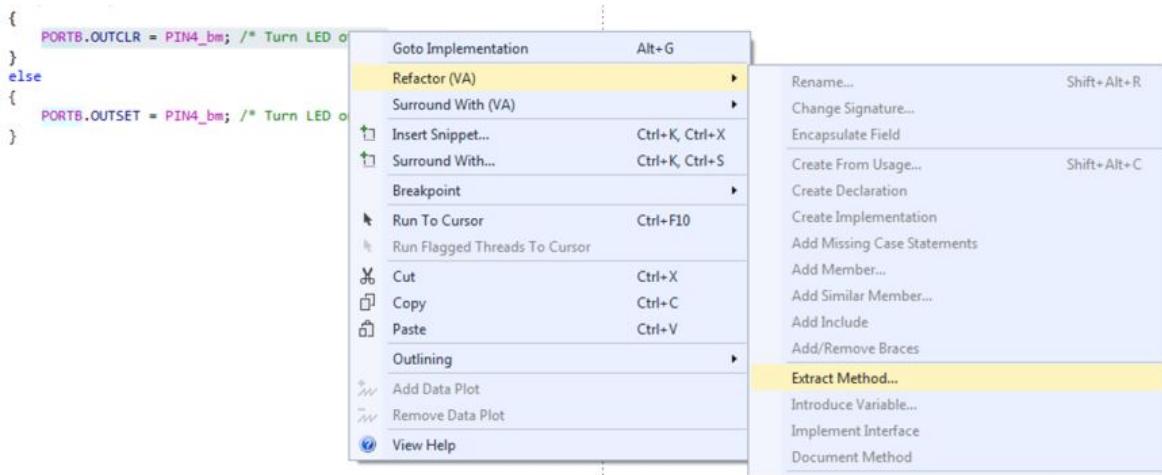
```

5. **Verify that LED0 lights up when pushing SW0.** Run the code by clicking *Start Without Debugging* (Ctrl+Alt+F5), to verify that LED0 lights up when pushing SW0 on the ATtiny817 Xplained Pro kit.

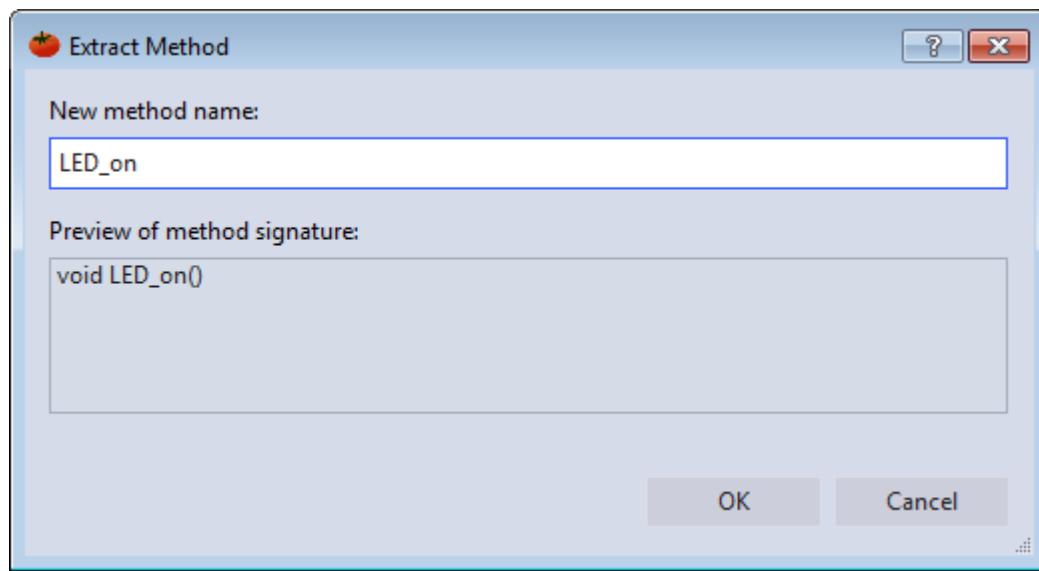
Now that the basic functionality is in place, let's refactor the code to make it more readable.

6. **Create functions LED_on() and LED_off() using Refactor → Extract Method** The line of code to turn the LED ON is executed when SW0 is pressed. Highlight this line of code, right-click and go to it, as indicated in the figure below.

Figure 1-30. Extract Method

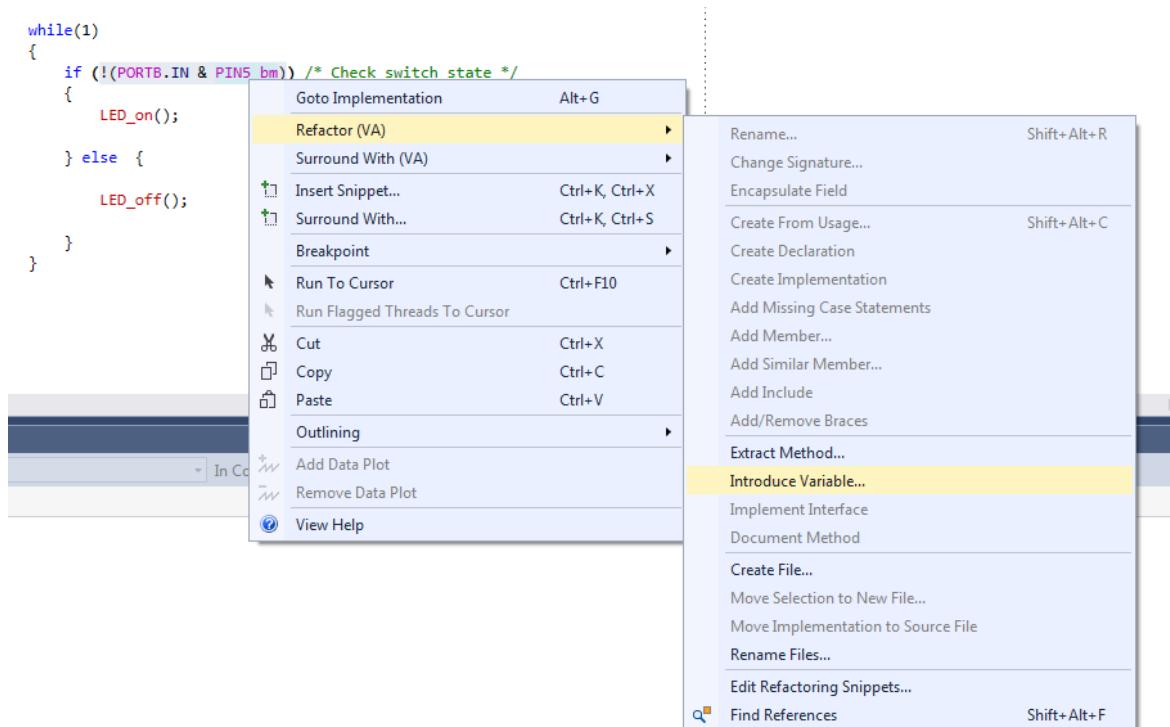


A **Extract Method** dialog will appear. Name the function "LED_on", as indicated in the following figure.

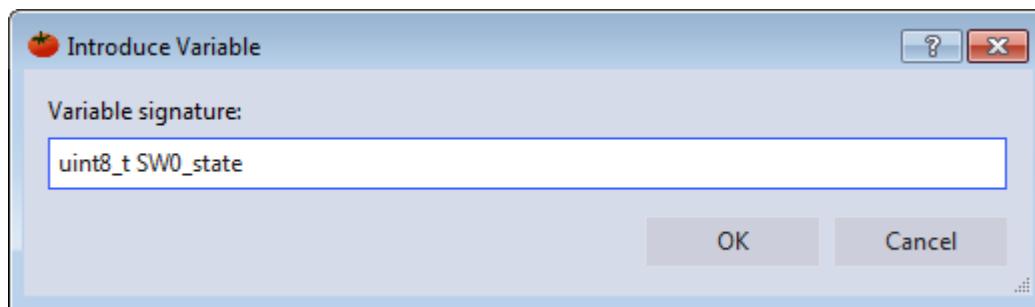
Figure 1-31. Extract Method Dialog

Click **OK**, and the code should change. A new function called `LED_on()` should appear at the top of the file, with a function call where the line of code used to be. Use the same method to implement `LED_off()`.

7. **Create a variable for SW0 state, using Refactor → Introduce Variable.** Next, it is necessary to create a variable for the SW0 state. Highlight the condition inside the `if()` in the `main()` `while(1)` loop. Right-click and go to it, as indicated in the figure below.

Figure 1-32. Introduce Variable

The **Introduce Variable** dialog will appear, as depicted in [Figure 1-33](#). Name the variable "uint8_t SW0_state".

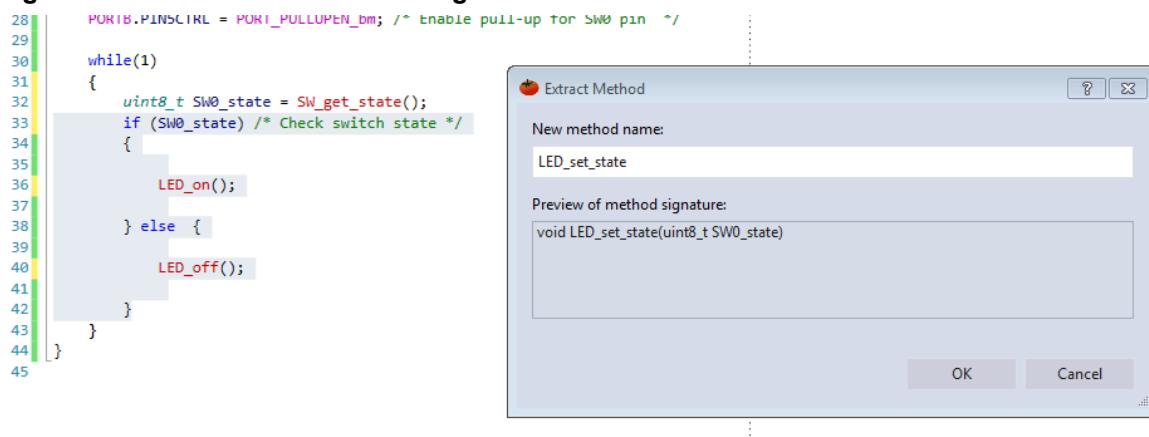
Figure 1-33. Introduce Variable Dialog

tip: Change the automatically generated `bool` return value to `uint8_t` to avoid having to include an extra header to deal with Boolean values.

Click **OK** and the code should change. The condition inside the `if()` statement should now reference a variable assigned to the variable on the line above it, as shown in the code block below.

```
while (1)
{
    uint8_t SW0_state = !(PORTB.IN & PIN5_bm);
    if (SW0_state)
    {
        LED_on();
    }
    else
    {
        LED_off();
    }
}
```

8. **Create a function `sw_get_state`, using Refactor → Extract Method.** Select the right side of the `SW0_state` assignment and extract a method for `sw_get_state`.
9. **Implement a function `void LED_set_state(uint8_t state)`.** Extract the method. Atmel Studio will detect the argument `SW0_state`, as indicated in [Figure 1-34](#).

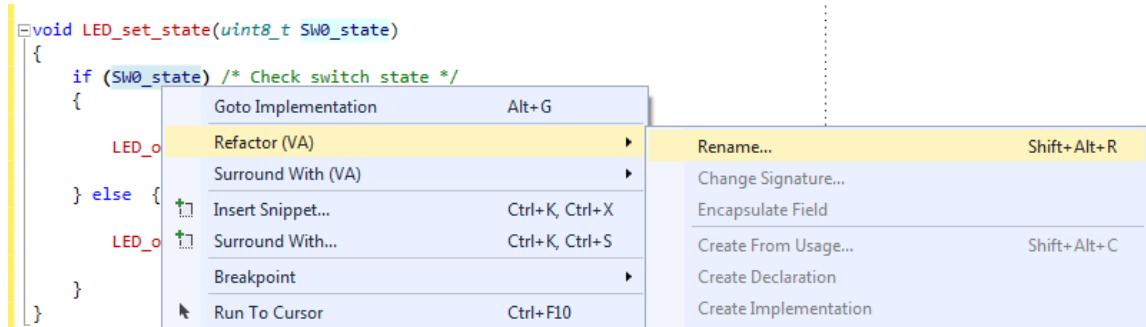
Figure 1-34. Extract Method with Argument

Click **OK** and the code should change. Now, there is a separate method for setting the LED state.

10. **In function `void LED_set_state(uint8_t state)` rename `SW0_state` to `state` using Refactor → Rename.** In a larger application, this function may be used for setting the LED state in a context

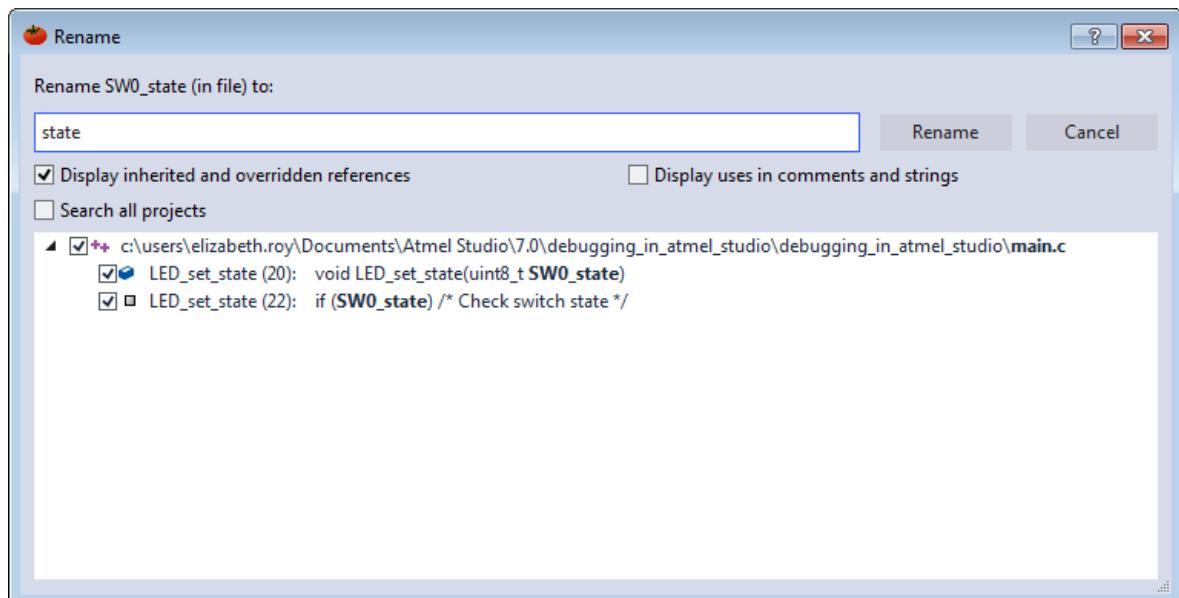
that is irrelevant to the SW0 state. Atmel Studio is capable of contextual renaming, so this feature can be used to easily rename the argument and avoid confusion. Inside the `LED_set_state()` function, right-click on the `SW0_state` variable and go to **Refactor → Rename**, as indicated in Figure 1-35.

Figure 1-35. Contextual Rename



The **Rename** dialog will appear, as depicted in Figure 1-36. Rename the `SW0_state` variable to "state". Atmel Studio will detect all occurrences of the variable with the same context as the one which has been selected, and which are presented in a list and able to be individually selected or deselected.

Figure 1-36. Contextual Renaming Dialog



Click **Rename** and the code should change. Observe that the argument of `LED_set_state()` and all of its references inside the function have been renamed, but the references to `SW0_state` in `main()` have remained the same.

11. Create function definitions, moving created functions below `main()`.
`main.c` should now look as follows:

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

void LED_on(void);
void LED_off(void);
void LED_set_state(uint8_t state);
uint8_t SW_get_state(void);
```

```
int main(void)
{
    PORTB.DIRSET = PIN4_bm; /* Configure LED Pin as output */
    PORTB.PIN5CTRL = PORT_PULLUPEN_bm; /* Enable pull-up for SW0 pin */

    while(1)
    {
        uint8_t SW0_state = SW_get_state(); /* Read switch state */
        LED_set_state(SW0_state); /* Set LED state */
    }
}

uint8_t SW_get_state(void)
{
    return !(PORTB.IN & PIN5_bm); /* Read switch state */
}

void LED_off(void)
{
    PORTB.OUTSET = PIN4_bm; /* Turn LED off */
}

void LED_on(void)
{
    PORTB.OUTCLR = PIN4_bm; /* Turn LED on */
}

void LED_set_state(uint8_t state)
{
    if (state)
    {
        LED_on();
    }
    else
    {
        LED_off();
    }
}
```

1.12 AVR Simulator Debugging

This section will demonstrate the use of the AVR Simulator key features, such as: Cycle Counter, Stopwatch (only available in the simulator), and basic debugging (setting breakpoints and stepping through code). We will also show how to simulate interrupts.

Getting Started Topics



Studio 7: AVR® MCU Simulator Debugging

In this video:

Studio 7: AVR MCU Simulator

Project Setup:

Modify project from Studio 7 Editor video

- Basic debugging: set breakpoint, step, ...
- **Processor view:** Demonstrate use of cycle counter & stop watch
- **Disassembly view:** difference how code compiled
- Simulate IRQ (IO view)

Program Counter	0x00000028
Stack Pointer	0x3FFD
X Register	0x0000
Y Register	0x3FFF
Z Register	0x0000
Status Register	0x0000
Cycle Counter	2
Frequency	1.000 MHz
Stop Watch	2.00 µs
Registers	

Context:

- Set up 3 options to clear, then set register bit
- LED on when switch pressed (using pin change IRQ).

1) Use read-modify-write in code

2) HW read-modify-write registers

3) Bit-accessible virtual port I/O

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

int main(void)
{
    PORTB.DIR &= ~PIN4_bm;
    PORTB.DIR |= PIN4_bm;

    //PORTB.DIRSET = PIN4_bm;
    //PORTB.DIRCLR = PIN4_bm;

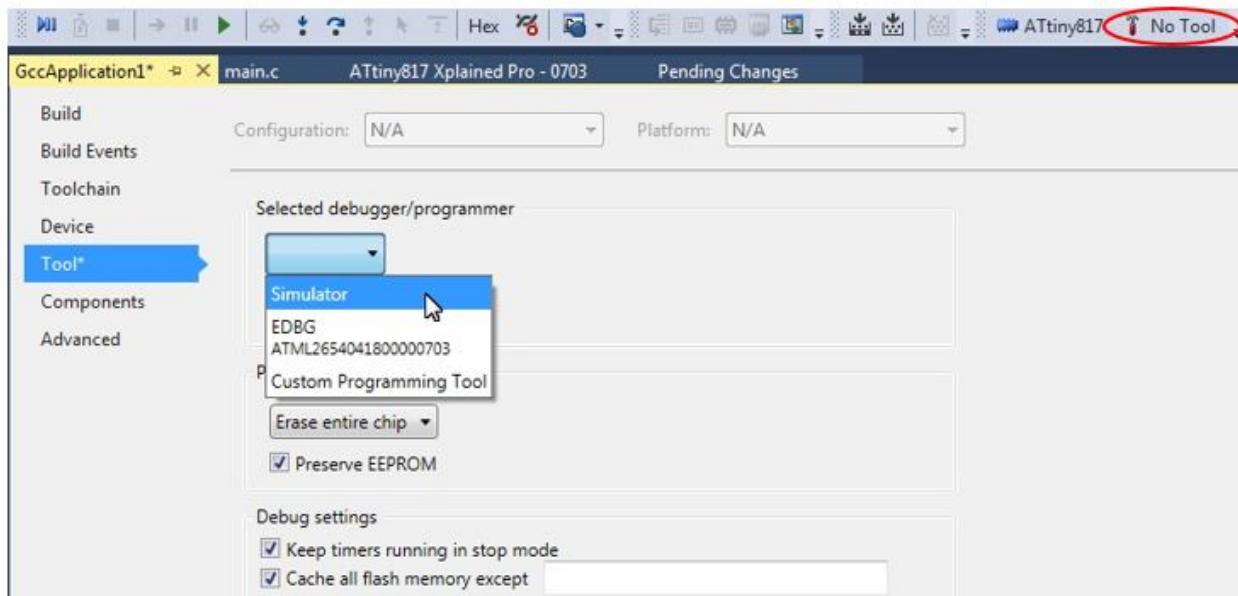
    //VPORTB.DIR &= ~PIN4_bm;
    //VPORTB.DIR |= PIN4_bm;

    while (1)
    {
    }
}
```

Video: AVR Simulator Debugging

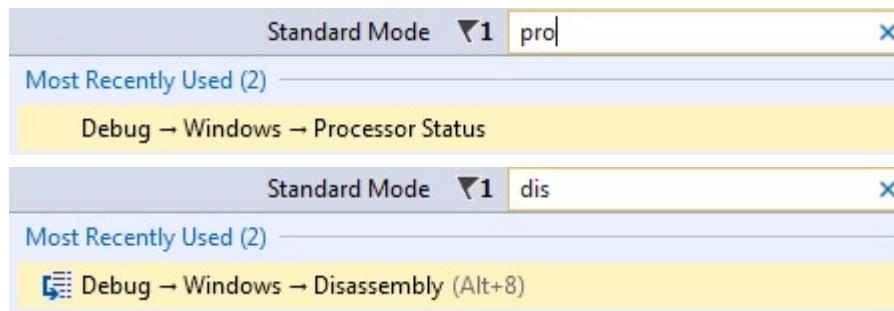
The [code](#) used in the video above was written in the video: [Editor: Writing and Re-Factoring Code \(Visual Assist\)](#).

To associate the simulator with the project, click on the Tool icon , then select Simulator.



The **Cycle Counter** and **Stopwatch** is only available with the simulator. To use these, first, click **Start Debugging and Break** to start a debug session and then open the **Processor Status** window by

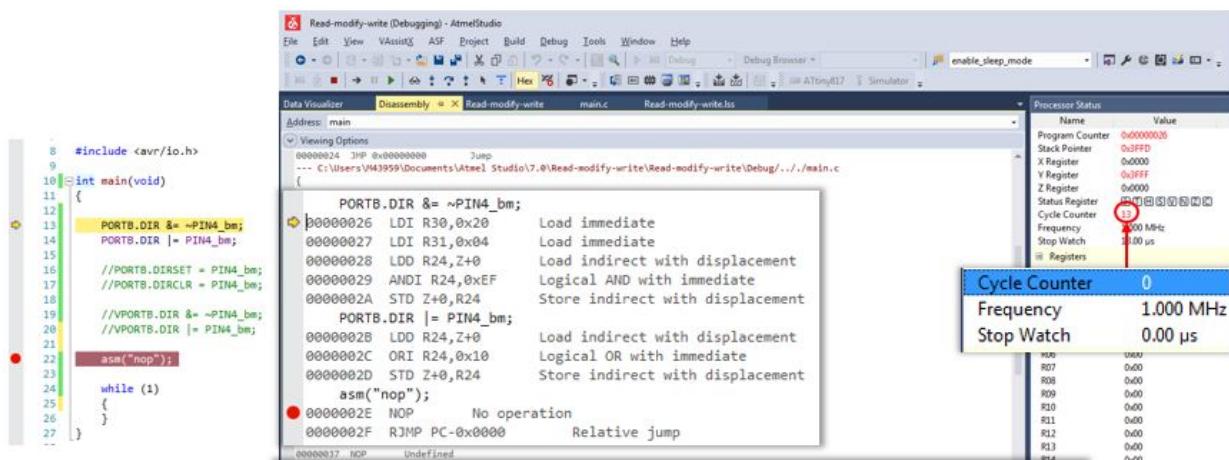
typing "Processor" into the quick-launch bar and hitting enter (or this can be found under Debug > Windows > Processor Status). Similarly, the **Disassembly** window can also be opened.



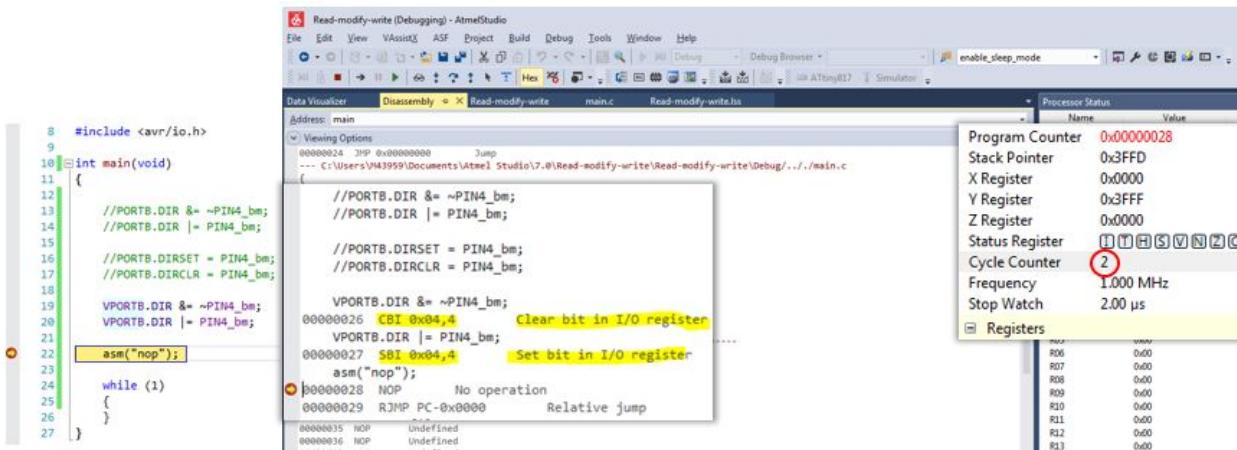
The AVR Simulator is using models based on the same RTL code used to make the real device. This makes the **Cycle Counter** both bug and delay accurately. Note that the **Stop Watch** is related to the **Frequency**, which you can set by double-clicking on the value and entering the clock frequency you would like to use.

Processor	
Name	Value
Program Counter	0x000000380 ← address of the instruction being executed
Stack Pointer	0x00003821 ← current stack pointer value
X Register	0x0002
Y Register	0x3FF1
Z Register	0x38214
Status Register	I T H S V N Z C
Cycle Counter	8186 ← cycles elapsed from the simulation's start
Frequency	1,000 MHz
Stop Watch	8 186,00 us ← time elapsed based on cycles and frequency
+ Registers	

The **Cycle Counter** can be reset by clicking on the value and entering 0. Values in the Processor Status window are updated every time the program breaks, similar to the I/O view. Then running to a breakpoint.



Note the difference in generated assembly code between the SW read-modify-write (above) and the virtual port registers (see below).



The result of comparing these three methods are summarized in the table below:

Method	Cycles	Comments
SW read-modify-write	10	
HW read-modify-write reg.	5	Atomic instruction (IRQ safe)
Bit-accessible virtual port I/O	2	Atomic instruction (IRQ safe), really fast

Next, we would like to simulate a pin change IRQ. We can do this by setting the relevant IRQ flag in the I/O view when debugging.

I/O

Filter: portB

Name	Value		
+ I/O Ports (PORTB)			
+ Virtual Ports (VPORTB)			
Name	Address	Value	Bits
DIR	0x420	0x00	████████████████
DIRSET	0x421	0x00	████████████████
DIRCLR	0x422	0x00	████████████████
DIRTGL	0x423	0x00	████████████████
OUT	0x424	0x00	████████████████
OUTSET	0x425	0x00	████████████████
OUTCLR	0x426	0x00	████████████████
OUTTGL	0x427	0x00	████████████████
IN	0x428	0x00	████████████████
- INTFLAGS	0x429	0x20	███████ ■ ████████
INT	0x20	0x00	████████████████
PIN0CTRL	0x430	0x00	████████████████
PIN1CTRL	0x431	0x00	████████████████
PIN2CTRL	0x432	0x00	████████████████

As shown below the ISR is hit. Note that the INTERRUPT still needs to be enabled, as shown in the write to PORTB.PIN5CTRL in the code below.

```

76 ISR(PORTB_PORT_vect)
77 {
78     uint8_t intflags = PORTB.INTFLAGS;
79     PORTB.INTFLAGS = intflags;
80
81     bool SW_state = SW_get_state();
82     LED_set_state(SW_state);
83 }
84
85

```

The pin change IRQ could also have been triggered by writing to the Port Input register in the I/O view. Writing a bit in the Port Input register is the same as applying that value to the physical pin of the device package. The internal Port logic will then trigger the interrupt if it is configured accordingly.

Most of the standard debugging features of **Studio 7** are available when using the simulator, and those features will also be available on devices that lack on-chip debugging capabilities and cannot be debugged using hardware debuggers. See the debugging sections of this Getting Started guide.

Code Used to Demonstrate AVR Simulator (Written for ATtiny187)

```

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

void LED_on();
void LED_off();
bool SW_get_state();
void LED_set_state(bool SW_state);

```

```
int main(void)
{
    PORTB.DIR &= ~PIN4_bm;
    PORTB.DIR |= PIN4_bm;

    PORTB.DIRCLR = PIN4_bm;
    PORTB.DIRSET = PIN4_bm;

    VPORTB.DIR &= ~PIN4_bm;
    VPORTB.DIR |= PIN4_bm;

    PORTB.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;
    sei();

    while (1)
    {
    }
}

#pragma region LED_functions
void LED_on()
{
    PORTB.OUTCLR = PIN4_bm; //LED on
}

void LED_off()
{
    PORTB.OUTSET = PIN4_bm; //LED off
}

void LED_set_state(bool SW_state)
{
    if (SW_state)
    {
        LED_on();
    }
    else
    {
        LED_off();
    }
}
#pragma endregion LED_functions

bool SW_get_state()
{
    return !(PORTB.IN & PIN5_bm);
}

ISR(PORTB_PORT_vect)
{
    uint8_t intflags = PORTB.INTFLAGS;
    PORTB.INTFLAGS = intflags;

    bool SW_state = SW_get_state();
    LED_set_state(SW_state);
}
```

1.13 Debugging 1: Break Points, Stepping, and Call Stack

This section will introduce the debugging capabilities of Studio 7, both as video (linked below) and hands-on document. The main topics are breakpoints, basic code stepping using the Breakpoint, and Callstack-Windows, as well as adjusting project compiler optimization settings.

[Getting Started Topics](#)



Studio 7: Debugging – 1

In this video:

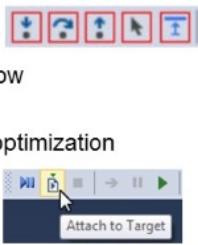
Studio 7: Debugging 1

Context:

Debug project from Studio 7 Editor video

Features Covered:

- Basic break points
- Code stepping
- Breakpoints window
- Call stack
- Project compiler optimization
- Attach to target



Launch a debug session on the selected target
without RESET or uploading a new application.

```

60     bool SW_get_state()
61     {
62         return !(PORTB.IN & PINS_bm);
63     }
64
65     ISR(PORTB_PORT_vect)
66     {
67         uint8_t intflags = PORTB.INTFLAGS;
68         PORTB.INTFLAGS = intflags;
69
70         bool SW_state = SW_get_state();
71         LED_set_state(SW_state);
72
73     }
74

```

Call Stack

Name
Getting Started.elf! LED_on Line: 39
Getting Started.elf! LED_set_state Line: 57
Getting Started.elf! _vector_4 Line: 74

Video: Studio 7 Debugging-1

The same code as the one created in section [Editor: Writing and Re-Factoring Code \(Visual Assist\)](#), is used.



To do: Place a breakpoint and inspect a list of all breakpoints in the project.

1. Set a breakpoint on the line getting the switch state, as indicated in [Figure 1-37](#).

Figure 1-37. Placing a Breakpoint

A screenshot of the Atmel Studio 7 code editor. The cursor is hovering over line 70 of the file 'main.c'. A red circle highlights the breakpoint icon (a gear with a circle) on the left margin. The line of code is highlighted in red, and a tooltip shows 'Location: main.c, line 70 character 1'. The code in the snippet is identical to the one shown in Figure 1-36.

```

65     ISR(PORTB_PORT_vect)
66     {
67         uint8_t intflags = PORTB.INTFLAGS;
68         PORTB.INTFLAGS = intflags;
69
70         bool SW_state = SW_get_state();
71         LED_set_state(SW_state);
72
73     }

```



Info: A breakpoint can be placed at a line of code by:

- Clicking the gray bar on the left edge of the editor window.
- In the top menu bar, go to **Debug** → **Toggle Breakpoint**.
- By pressing F9 on the keyboard.

2. Launch a debug session . The breakpoint will be hit when the switch (SW0) on the Xplained Pro kit is pressed. Observe that execution is halted when the breakpoint is hit, and the execution arrow indicates that the line of code where the breakpoint is placed is about to execute. See [Figure 1-38](#).

Figure 1-38. Execution Halting when a Breakpoint is Hit

```

65     ISR(PORTB_PORT_vect)
66     {
67         uint8_t intflags = PORTB.INTFLAGS;
68         PORTB.INTFLAGS = intflags;
69
70         bool SW_state = SW_get_state();
71         LED_set_state(SW_state);
72
73     }

```



tip: If a breakpoint is hit in a file that is not currently open, Atmel Studio will open the file in a temporary pane. A file containing a breakpoint that is hit in a debug session will always be brought to focus.

3. Since most of the logic of the program is handled only when an ISR is processed, it is now possible to check the logical flow of the program. If the switch is pressed and then released when the ISR is hit - what will be the state of the switch that the function returns? The assumption is that since pressing the switch triggered the interrupt, that switch will be set as *pressed*, and the LED will thus be turned ON.

Code stepping can be used to check this assumption. The key buttons used for code stepping are illustrated in the table below, found in the top menu bar or in the **Debug** menu. The corresponding functionality and keyboard shortcuts are outlined in the figure below.

Figure 1-39. Atmel Studio Buttons for Code Stepping



Table 1-4. Atmel Studio Button Functionality (Code Stepping)

Button	Functionality	Keyboard Shortcut
	Step Into Function Call	F11
	Step Over	F10

Button	Functionality	Keyboard Shortcut
	Step Out of Function Call	Shift + F11
	Run to Cursor	Ctrl + F10
	Issue System Reset	



To do: Find out what state is returned if the switch is pressed and then released when the ISR is hit. Is our assumption correct that since pressing the switch triggered the interrupt, it will be set as *pressed*, and the LED will thus be turned ON?

The *Step Into Function Call* can be used first. To go into the `SW_get_state()` function, the *Step Out of Function Call* can be used to move to the next line after returning from the function. Pressing *Step Over* from the breakpoint would land us at this same point directly. Note that we could step further into the function `LED_set_state(SW_state)` to determine if the LED is turned ON or not. However, we could simply hover the mouse pointer over the `SW_state` variable to see that it is now set to 0, i.e. the LED will be turned OFF. Verify this by stepping further.

Figure 1-40. Checking Value of `SW_state` Using Mouse Hover

```

60     bool SW_get_state()
61     {
62         return !(PORTB.IN & PIN5_bm);
63     }
64
65     ISR(PORTB_PORT_vect)
66     {
67         uint8_t intflags = PORTB.INTFLAGS;
68         PORTB.INTFLAGS = intflags;
69
70         bool SW_state = SW_get_state();
71         SW_state // tooltip: SW_state[0]
72         LED_set_state(SW_state);
73     }
74 }
```



Info: Although the breakpoint was triggered by the falling edge by pressing the switch, only when calling the `SW_get_state()` function the switch state is recorded. Verify that `SW_state` will read 1 if the switch is held down when stepping over this line.

1. A window or view to keep track of the breakpoints in a program is needed. The Quick Launch bar performs a search of the Studio 7 user interface menus. This is demonstrated below, by comparing the two figures [Figure 1-41](#), and [Figure 1-42](#). Note that each of the hits in the Quick Launch bar are from "break" related entries in the *Debug* menu.

Figure 1-41. "Break" Search in the Quick Launch Bar

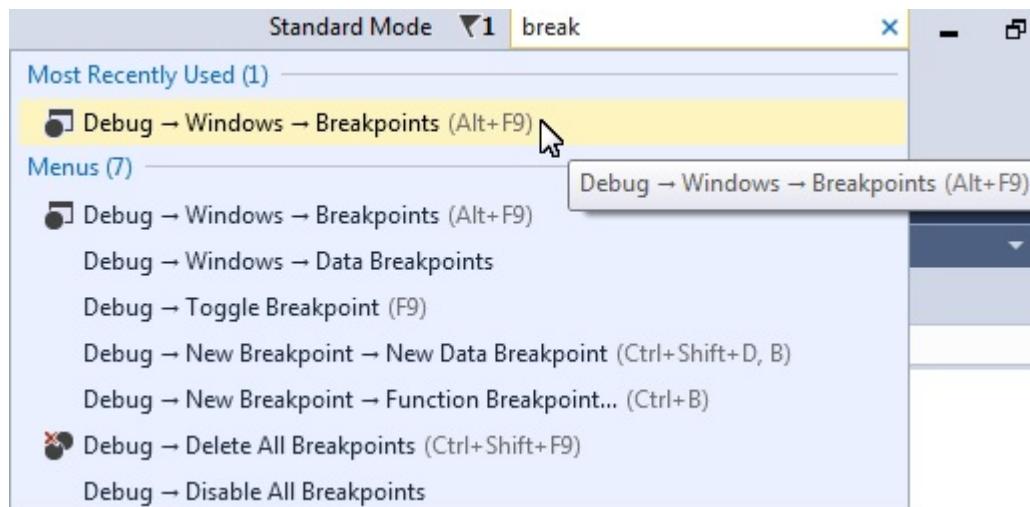
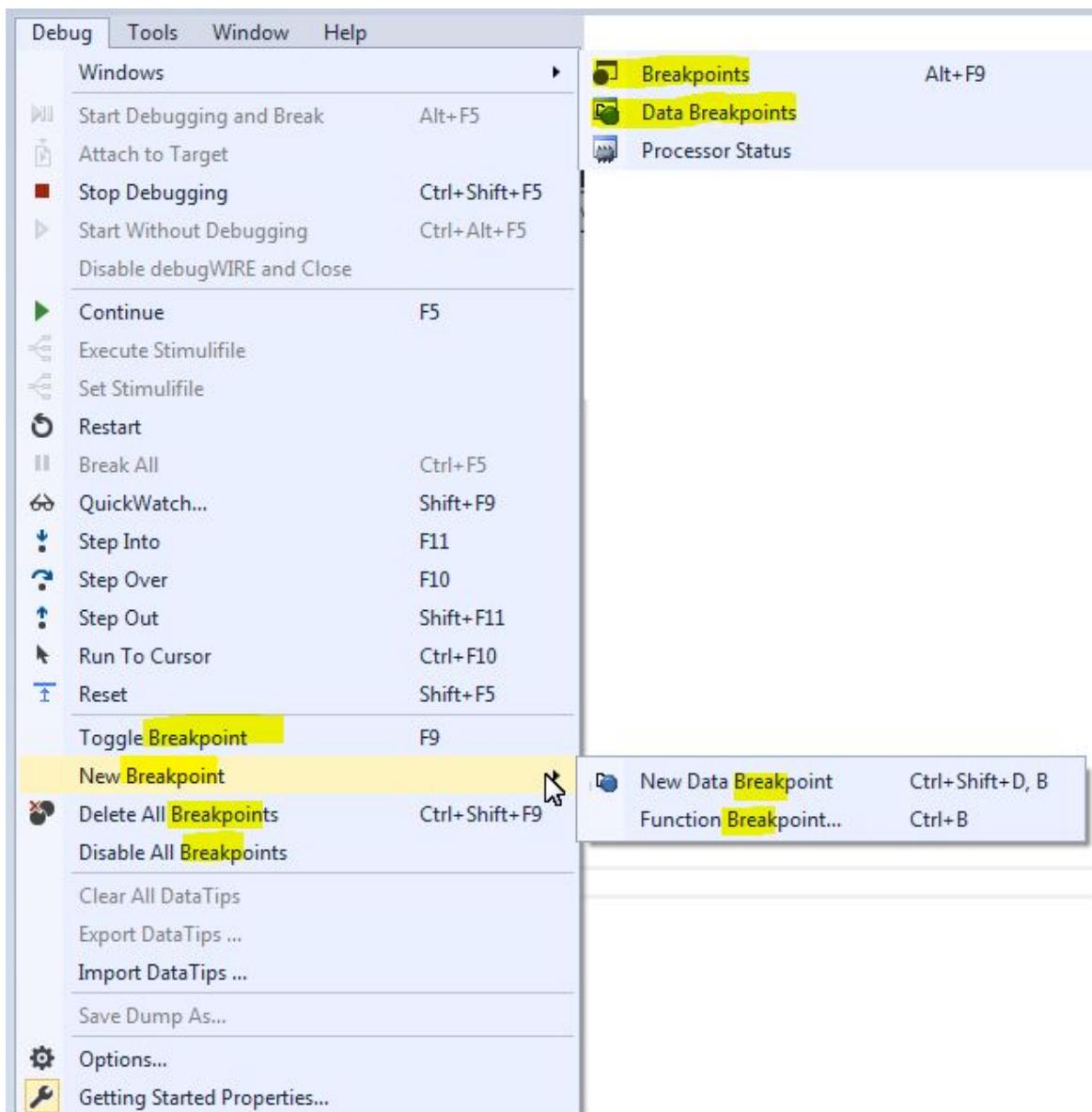


Figure 1-42. "Break" Hits in Debug Menu

Open the Breakpoints Window by clicking on the top result (**Debug → Windows → Breakpoints**). The Breakpoints Window lists all the breakpoints in the project, along with the current hit count, as depicted in [Figure 1-43](#).



tip: A breakpoint can be temporarily disabled by unchecking the checkbox next to a breakpoint in the list.



tip: The Disassembly view can be conveniently displayed alongside the source code, as demonstrated in the [Figure 1-44](#) section.

Figure 1-43. Breakpoints Window

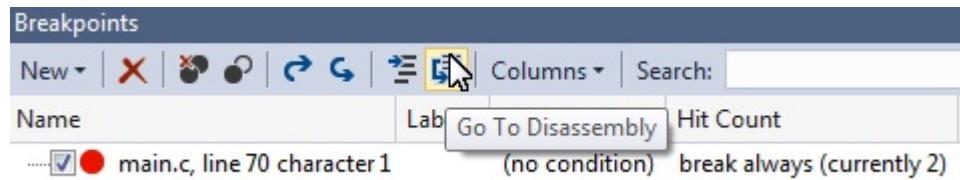
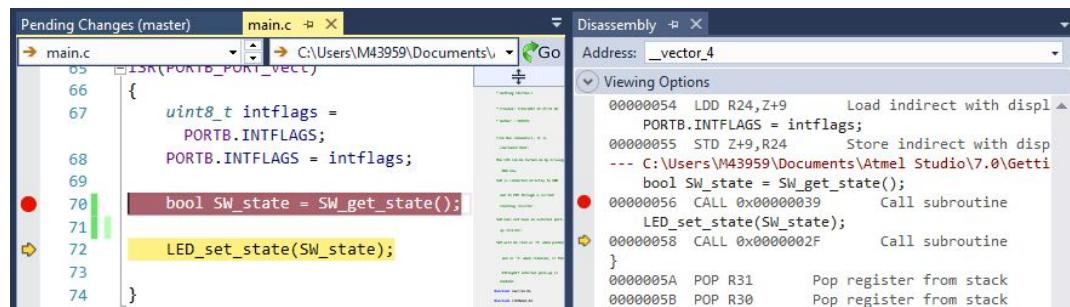


Figure 1-44. Disassembly View

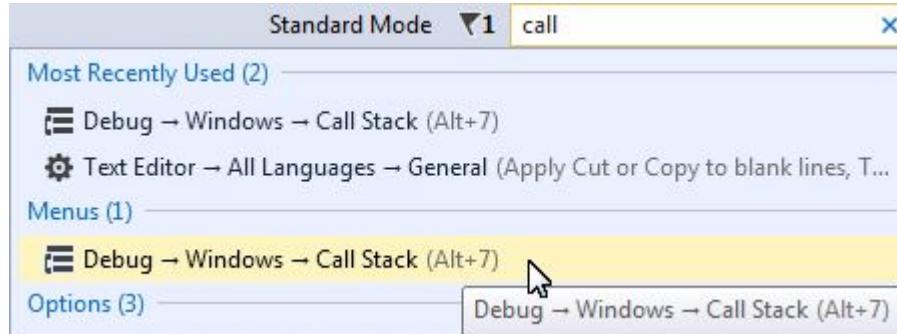


To do: Examine the Call Stack and the effect on it when optimizations are disabled.

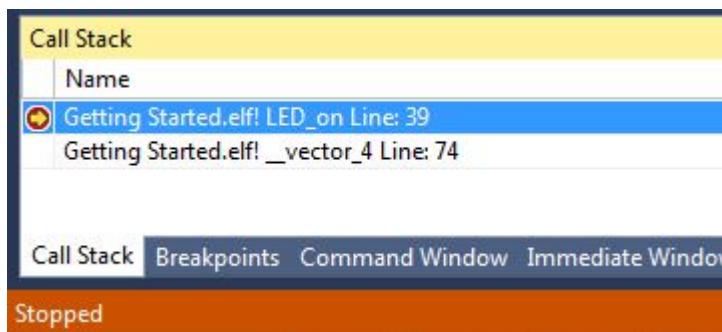
1. Following from the previous section, set a breakpoint on the `LED_on()` function, then trigger the breakpoint so that it is hit.
2. Open the Call Stack window by typing "Call" in the Quick Launch bar, selecting **Debug** → **Windows** → **Call Stack**, as represented in [Figure 1-45](#).

Note: A debug session needs to be active to open this window.

Figure 1-45. Open the Call Stack Window

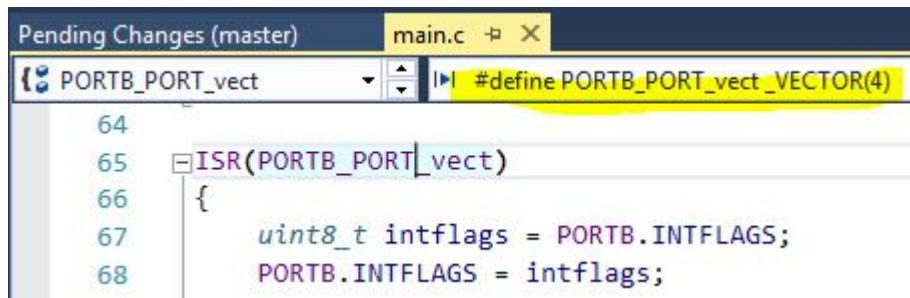


3. It would be expected that the Call Stack shows `LED_set_state()` as the caller of `LED_on()`, since that's how the code is written. However, in the Call Stack window, `_vector_4` is listed as the caller (as in [Figure 1-46](#)); this is because of compiler optimization.

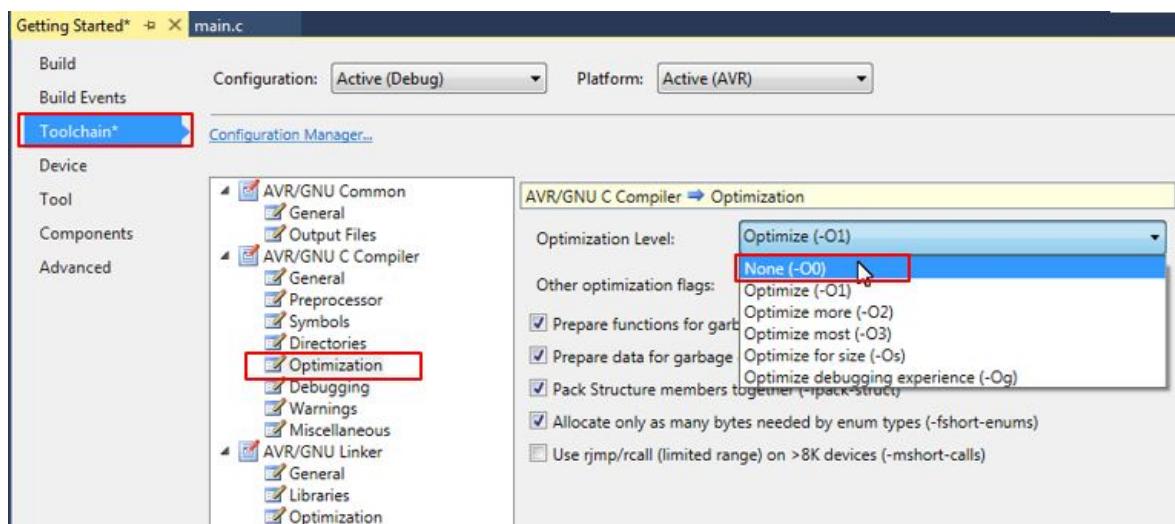
Figure 1-46. Call Stack with Optimization

Info: The call order is different because of the compiler optimization. This code is relatively simple to follow and it is possible to understand what is going on even though the compiler has optimized and made subtle changes to what is expected. In a more complex project, it can sometimes be helpful to disable the compiler optimization to track down a bug.

Note: To see why the Call Stack shows that it comes from `_vector_4` initially, click on `PORTE_PORT_vect` and look in the context field for the definition, as shown in [Figure 1-47](#).

Figure 1-47. `_vector_4` is the PORTB ISR Vector

4. Stop debugging by clicking the Stop Debugging button or pressing Shift + F5.
5. Open the project settings by going to **Project** → <project_name> properties or pressing Alt + F7. Go to the **Toolchain** tab on the left menu, as in [Figure 1-48](#).
6. Under **AVR/GNU C Compiler** → **Optimization**, set the **Optimization Level** to **None (-O0)** using the drop-down menu.

Figure 1-48. Disabling Compiler Optimizations

Disabling compiler optimization will result in increased memory consumption and can result in changes in execution timing. This can be important to consider when debugging time is a critical code.

7. Launch a new debug session and break code execution inside `LED_on()`.
8. Observe the Call Stack. It should now adhere to how the code is actually written and list `LED_set_state()` as the caller of `LED_on()`, as in [Figure 1-49](#).

Figure 1-49. Call Stack Without Optimization

tip: Atmel Studio will try to link the compiled code to the source-code as best it can, but the compiler optimization can make this challenging. Disabling compiler optimization can help if breakpoints seem to be ignored during debugging, or if the execution flow is hard to follow during code stepping.



Result: The call stack has now been examined both with and without optimization enabled.

Code used for Debugging 1

```
/*
 LED is turned on when switch is pressed, LED is turned on (via a pin change interrupt).
```

```
MY_mistake() written to demonstrate Attach to Target, is commented out, to avoid hanging  
project unintentionally.
```

```
From the schematics, it is concluded that:  
The LED can be turned on by driving PB4 low.  
SW0 is connected directly to GND and to PB5 through a current limiting resistor.  
SW0 does not have an external pull-up resistor.  
SW0 will be read as '0' when pushed and as '1' when released, if the ATTiny817 internal pull-  
up is enabled.  
*/  
  
#include <avr/io.h>  
#include <stdbool.h>  
#include <avr/interrupt.h>  
  
void LED_on();  
void LED_off();  
bool SW_get_state();  
void LED_set_state(bool SW_state);  
  
int main(void)  
{  
    PORTB.DIRSET = PIN4_bm;  
    PORTB.OUTSET = PIN4_bm;  
    PORTB.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;  
    sei();  
  
    while (1)  
    {  
    }  
}  
  
#pragma region LED_functions  
void LED_on()  
{  
    PORTB.OUTCLR = PIN4_bm; //LED on  
}  
  
void LED_off()  
{  
    PORTB.OUTSET = PIN4_bm; //LED off  
}  
  
void LED_set_state(bool SW_state)  
{  
    if (SW_state)  
    {  
        LED_on();  
    }  
    else  
    {  
        LED_off();  
    }  
}  
#pragma endregion LED_functions  
  
bool SW_get_state()  
{  
    return !(PORTB.IN & PIN5_bm);  
}  
  
/*  
void My_mistake()  
{  
    while(1)  
    {  
        asm("nop");  
    }  
}  
*/  
  
ISR(PORTB_PORT_vect)  
{  
    uint8_t intflags = PORTB.INTFLAGS;  
    PORTB.INTFLAGS = intflags;  
    //My_mistake();  
}
```

```

        bool SW_state = SW_get_state();

        LED_set_state(SW_state);

    }

```

1.14 Debugging 2: Conditional- and Action-Breakpoints

This section covers more advanced debugging topics with Studio 7 both as video (linked below) and hands-on document. The main topics are how to modify variables in the code, conditional- and action-breakpoints, as well as memory view.

[Getting Started Topics](#)



Studio 7: Debugging – 2

In this video:

Studio 7: Debugging 2 Context"

Project from Studio 7 Editor video (polled),
added logic to SW_get_state():

- [SW_get_state\(\) -> SW_get_states_logic\(\)](#)

Features Covered:

- **Watch:** view & modify variables
- **Conditional Breakpoints**
 **To do:** In SW_get_states_logic()
break only every 5th edge count, &
if edge was rising;
- **Action Breakpoints**
 **To do:** Log various state variables
to output window.

```

uint8_t SW_get_states_logic(void)
{
    static uint8_t SW0_prv_state = 0;
    static uint8_t SW0_edge_count = 0;

    /* Read the current SW0 state */
    uint8_t SW0_cur_state = !(PORTB.IN & PIN5_bm);
    if (SW0_cur_state != SW0_prv_state) /* Check for edges */
    {
        SW0_edge_count++;

```

Location: main.c, Line: 78, Character: 1. Must match source
 Conditions
Conditional Expression: ((SW0_edge_count%5)==0)&&SW0_cur_state
Add condition

Output
Show output from: Debug
Prv state:0, Cur_state:0, Edge count:0
Prv state:0, Cur_state:0, Edge count:0

[Video: Debugging - 2](#)



To do: Use Atmel Studio to inspect and modify the contents of variables in the code.

1. The code (see below) used is the same as the one developed in section [Editor: Writing and Refactoring Code \(Visual Assist\)](#). The SW_get_state() function has just been replaced with the following code (note also the change in return value type):

```

uint8_t SW_get_state(void)
{
    static uint8_t SW0_prv_state = 0;
    static uint8_t SW0_edge_count = 0;

    uint8_t SW0_cur_state = !(PORTB.IN & PIN5_bm); /* Read the current SW0 state */
    if (SW0_cur_state != SW0_prv_state) /* Check for edges */
    {
        SW0_edge_count++;

```

```
        }
        SW0_prv_state = SW0_cur_state;           /* Keep track of previous state */

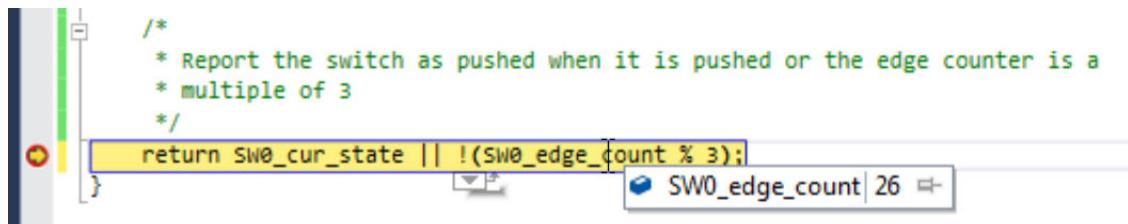
        /*
         * Report the switch as pushed when it is pushed or the edge counter is a
         * multiple of 3
         */
        return SW0_cur_state || !(SW0_edge_count % 3);
    }
```



Info: This code will count how many times the SW0 push button has been pressed or released. The return statement has also been modified to always report the button as pushed if the `SW0_edge_count` variable is a multiple of three.

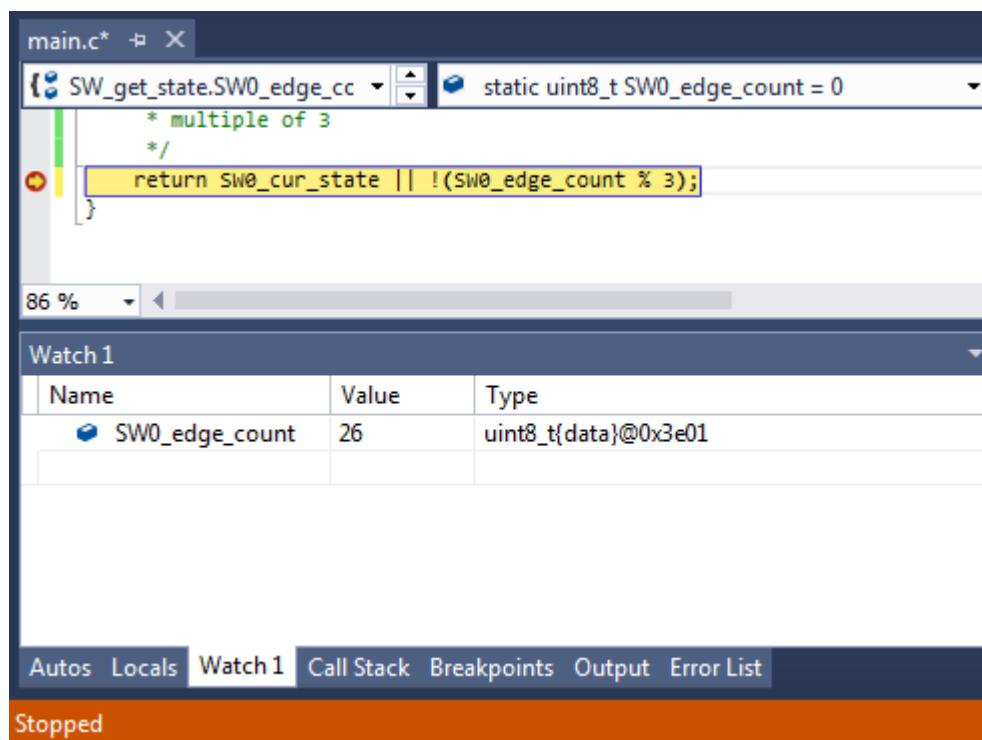
2. Go to **Debug → Disable All Breakpoints** to disable all breakpoints. This should be reflected by all the checkboxes becoming unchecked in the Breakpoints window.
3. Launch a new debug session by clicking the Start Debugging button .
4. Push SW0 on the kit several times and observe how the changes to the code have affected the LED's behavior.
5. Break execution by placing a breakpoint at the return line of the `SW_get_state` function.
6. Hover over the `SW0_edge_count` variable to observe the current value, as indicated in [Figure 1-50](#).

Figure 1-50. Hover Over Variable to See Current Value



Info: When the cursor hovers over a variable that is in scope at the point where execution is halted, Atmel Studio will present the content of the variable in a pop-up.

7. Right-click the `SW0_edge_count` variable and select **Add Watch** from the context menu to add the variable to the data Watch window. The Watch window should appear, with the `SW0_edge_count` variable listed, with the variable value, data type, and memory address, as in [Figure 1-51](#).

Figure 1-51. Add Variable to Watch Window

8. Modify the contents of a **Watch Window** variable, using the process described below. Assign the value '3' to the `SW0_edge_count` variable. The value will reflect as updated by turning red, as indicated in [Figure 1-52](#).
 - Double-click a variable value in the Watch window
 - Type in the desired new value of the variable
 - Press Enter to confirm

Figure 1-52. Newly Updated Variable Value in the Watch Window

Watch 1

Name	Value	Type
SW0_edge_count	3	uint8_t{data}@0x3e01

i **Info:** The Value column in the Watch window can be displayed in hex by right-clicking in the Watch window and selecting **Hexadecimal Display** from the context menu.

9. To have the device evaluate the new value of `SW0_edge_count`, disable all breakpoints and continue the debug session by clicking or pressing F5. Observe how the LED stays ON as a result of the change made to `SW0_edge_count`.

**Info:**

A variable can also be added to the Watch window by clicking on an empty field name and typing the variable name. This way, it is even possible to cast a variable to a different data type for better readability in the Watch window. This is especially useful if it is required to look at an array that is passed to a function as a pointer.

For example, if an array is passed to a function, it will be passed to the function as a pointer. This makes it impossible for Atmel Studio to know the length of the array. If the length of the array is known, and it needs to be examined in the Watch window, the pointer can be cast to an array using the following cast:

```
* (uint8_t (*)[<n>])<name_of_array_pointer>
```

Where *<n>* is the number of elements in the array and *<name_of_array_pointer>* is the name of the array to be examined.

This can be tested on the *SW0_edge_count* variable by entering the following in an empty name field in the Watch window:

```
* (uint8_t (*)[5])&SW0_edge_count
```

Note that the '*&*' symbol must be used in this case to obtain a pointer to the variable.



Result: Atmel Studio has now been used to inspect and modify the contents of variables in the code.

1.14.1 Conditional Breakpoints

This section is a guide to using Atmel Studio to place conditional breakpoints.

Conditional breakpoints are those which will only halt code execution if a specified condition is met, and can be useful if it is required to break if certain variables have given values. Conditional breakpoints can also be used to halt code execution according to the number of times a breakpoint has been hit.



To do: Place a conditional breakpoint inside *SW_get_state()* to halt execution for debugging at every 5th edge count, but only if the edge was rising, and check its functionality.

1. Clear all breakpoints from the project using the Breakpoints window.
2. Place a breakpoint at the return line of *SW_get_state()*, as in [Figure 1-53](#).
3. Right-click the breakpoint and select **Conditions...** from the context menu.
4. Enter the following in the condition textbox:

```
((SW0_edge_count % 5) == 0) && SW0_cur_state
```

Figure 1-53. Conditional Breakpoint Expression Example

5. Press Enter to confirm the break condition.
6. Continue/Start a new debug session by clicking the button or pressing F5.
7. Push SW0 on the kit several times and observe how code execution is halted when the condition is fulfilled.
8. Verify that the condition is met by double-checking the variable values in the Watch window.



Even though code execution is completely halted only if the specified break condition is met, Atmel Studio temporarily breaks code execution each time the breakpoint is hit to read the variable content and determine if the break condition is met. Conditional breakpoints will, therefore, have an impact on execution timing, even if the actual break condition is never met.



tip: Use the **Hit Count** condition if execution needs to break based on how many times a breakpoint has been hit.



Result: Atmel Studio has been used to halt execution when the specified break condition is satisfied.

1.14.2 Action Breakpoints

This section is a guide to using Atmel Studio to place action breakpoints.

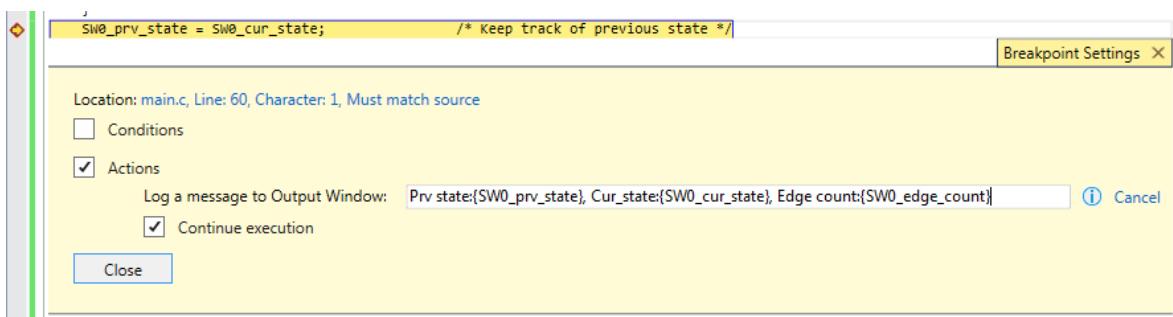
Action breakpoints can be useful if variable contents or execution flow needs to be logged without having to halt code execution and manually record the required data.



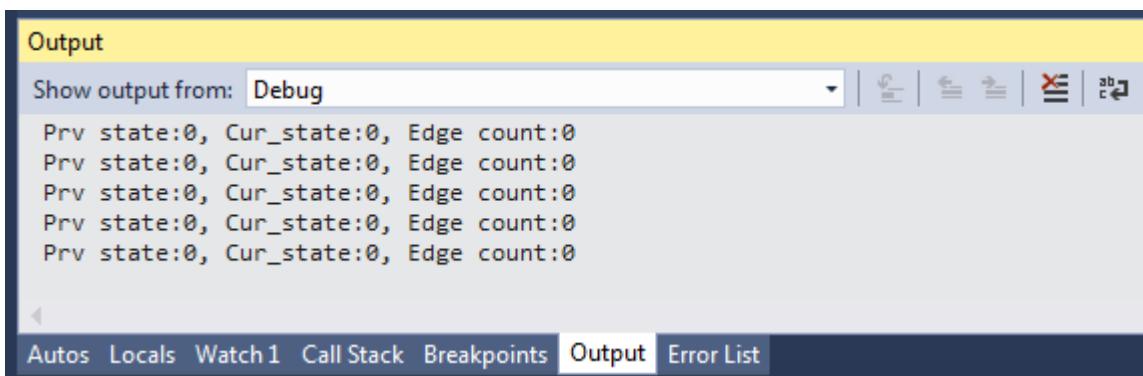
To do: Place an action breakpoint to log `SW0_cur_state`, `SW0_prv_state` and `SW0_edge_count`, and check the output for the relevant variable states.

1. Stop the ongoing debug session and clear all the breakpoints from the Breakpoints window.
2. Place a breakpoint at the `SW0_prv_state = SW0_cur_state;` line, as in [Figure 1-54](#).
3. Right-click the breakpoint and select **Actions...** from the context menu.
4. Enter the following in the output message text box:

```
Prv state:{SW0_prv_state}, Cur_state:{SW0_cur_state}, Edge count:{SW0_edge_count}
```

Figure 1-54. Action Breakpoint Example

5. Press Enter to confirm.
6. Start a debug session.
7. Open the Debug Output window by going to **Debug → Windows → Output**. It should list the variable contents as in [Figure 1-55](#). If SW0 is pushed on the kit, the content is updated.

Figure 1-55. Debug Output Window Showing Variable Contents

When using action breakpoints, Atmel Studio will temporarily halt code execution in order to read out variable content. As a result, execution timing will be affected. A less intrusive approach would be to place the action breakpoint at the `SW0_edge_count++` line, which is only executed upon SW0 edge detection. This will cause a temporary halt only when SW0 is pressed, but will also cause the debug window output to be delayed by one line of code.



tip: Action and Conditional breakpoints can be used together in order to log data only if a condition is satisfied.



Result: Atmel Studio has been used to log variable data using an action breakpoint.

1.14.3 Code used (for ATtiny817 Xplained Pro)

Code used for conditional- and action-breakpoints.

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

```

void LED_on();
void LED_off();
uint8_t SW_get_state();
void LED_set_state(uint8_t SW_state);

int main(void)
{
    PORTB.DIRSET = PIN4_bm;
    PORTB.OUTSET = PIN4_bm;
    PORTB.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;
    sei();

    while (1)
    {
    }
}

#pragma region LED_functions
void LED_on()
{
    PORTB.OUTCLR = PIN4_bm; //LED on
}

void LED_off()
{
    PORTB.OUTSET = PIN4_bm; //LED off
}

void LED_set_state(uint8_t SW_state)
{
    if (SW_state)
    {
        LED_on();
    }
    else
    {
        LED_off();
    }
}
#pragma endregion LED_functions

uint8_t SW_get_state(void)
{
    static uint8_t SW0_prv_state = 0;
    static uint8_t SW0_edge_count = 0;

    uint8_t SW0_cur_state = !(PORTB.IN & PIN5_bm); /* Read the current SW0 state */
    if (SW0_cur_state != SW0_prv_state) /* Check for edges */
    {
        SW0_edge_count++;
    }
    SW0_prv_state = SW0_cur_state; /* Keep track of previous state */

    /*
     * Report the switch as pushed when it is pushed or the edge counter is a
     * multiple of 3
     */
    return SW0_cur_state || !(SW0_edge_count % 3);
}

ISR(PORTB_PORT_vect)
{
    uint8_t intflags = PORTB.INTFLAGS;
    PORTB.INTFLAGS = intflags;

    uint8_t SW_state = SW_get_state();

    LED_set_state(SW_state);
}

```

1.15 Debugging 3: I/O View Memory View and Watch

This section covers more advanced debugging topics with Studio 7 both as video (linked below) and hands-on document. The main topics are using I/O View to work with Configuration Change Protected (CCP) registers, Memory View to validate EEPROM writes, as well as using the Watch window to cast pointers as an array.

[Getting Started Topics](#)



Studio 7: Debugging – 3

In this video:

Studio 7: Debugging 3

Context:

Project from Debugging 2, add function to save data to eeprom. Change clock freq to 10 MHz.

Features Covered:

- **I/O View:**
 - Configuration Change Protect (CCP) registers
- **Memory view:**
 - EEPROM Write (AVR® LibC)
- **Watch:**
 - Cast pointer to array of specified size, so can view in Watch Window

The screenshot shows the Atmel Studio 7 interface with three main windows:

- I/O View:** Displays the memory map of the device. It shows registers like CLKCTRL, MCLKCTRLA, MCLKCTRLB, and PDIV. A specific line of code is highlighted: `_PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_2X_gc | CLKCTRL_PEN_bm);`
- Memory View:** Shows the EEPROM data. It displays memory starting at address 0x1400 with values corresponding to the string "Hello World!". Below it, a hex dump of the EEPROM data is shown.
- Watch View:** Shows a table of variables and their values. The table includes:

Name	Type	Value
[uint8_t ()[30]]to_save	uint8_t [30]@0x3fe2	0x3fe2
[0]	uint8_t @0x3fe2	72
[1]	uint8_t @0x3fe3	101
[2]	uint8_t @0x3fe4	108
[3]	uint8_t @0x3fe5	108

[Video: Debugging - 3](#)

1.15.1 I/O View

The I/O view provides a graphical view of the I/O memory map of the device associated with the active project. This debug tool will display the actual register content when debugging, allowing verification of peripheral configurations. It can also be used to modify the content of a register without having to recompile.



To do: Use I/O view to:

- Get an overview of the device memory map.
- Check current peripheral configurations.
- Modify peripheral configurations.
- Validate configuration changes.

1. Remove all breakpoints and start a new debug session.
2. Break code execution by pressing the Break All button
3. Open the I/O view from the top menu bar by going to **Debug → Windows → I/O**.

4. Scroll through the list of peripherals and select **I/O Ports (PORTB)**. Find the **OUT** register and click on **Bit 4** in the **Bits** column, so the corresponding square changes color, as depicted in [Figure 1-56](#). Observe that clicking Bit 4 in the PORTB.OUT register toggles the output level on GPIO pin PB4, which controls the LED on the ATtiny817 Xplained Pro.

Figure 1-56. Manipulate Bit Value in Register Using I/O View

Name	Address	Value	Bits
DIR	0x420	0x10	0 0 0 0 1 0 0 0 0
DIRSET	0x421	0x10	0 0 0 0 0 1 0 0 0
DIRCLR	0x422	0x10	0 0 0 0 1 0 0 0 0
DIRTGL	0x423	0x10	0 0 0 0 0 1 0 0 0
OUT	0x424	0x00	0 0 0 0 0 0 1 0 0
OUTSET	0x425	0x00	0 0 0 0 0 0 0 1 0
OUTCLR	0x426	0x00	0 0 0 0 0 0 0 0 1
OUTTGL	0x427	0x00	0 0 0 0 0 0 0 0 0
IN	0x428	0xEC	1 1 1 0 0 1 0 0 0
INTFLAGS	0x429	0x00	0 0 0 0 0 0 0 0 0
PIN0CTRL	0x430	0x00	0 0 0 0 0 0 0 0 0
PIN1CTRL	0x431	0x00	0 0 0 0 0 0 0 0 0
PIN2CTRL	0x432	0x00	0 0 0 0 0 0 0 0 0
PIN3CTRL	0x433	0x00	0 0 0 0 0 0 0 0 0
PIN4CTRL	0x434	0x00	0 0 0 0 0 0 0 0 0
PIN5CTRL	0x435	0x08	0 0 0 0 0 0 1 0 0
PIN6CTRL	0x436	0x00	0 0 0 0 0 0 0 0 0
PIN7CTRL	0x437	0x00	0 0 0 0 0 0 0 0 0



Info: The I/O view is refreshed after any register has been modified, and all detected changes are highlighted in red.



tip: Multiple bits can be modified simultaneously by double-clicking the value field and typing in the desired value to be assigned to the register.

5. Expand the Clock controller (CLKCTRL) in the I/O view, and answer the following questions:

-
- What is the currently selected clock source (Clock select)?
 - What is the configured prescaler value (Prescaler division)?
 - Is the main clock prescaler enabled (MCLKCTRLB.PEN)?
-



Result: The Clock controller should be configured with the ATtiny817 default clock settings; the main clock is running from the internal RC oscillator with prescaler enabled and a division factor of six.



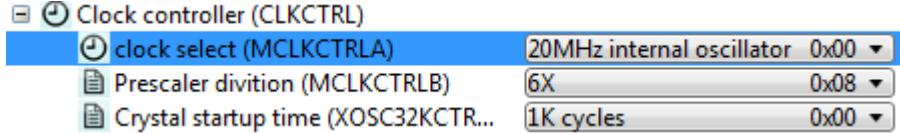
Info: The default clock configuration guarantees that the device will execute code reliably over the entire supported operating voltage range, 1.8V to 5.5V. The Xplained Pro kit powers the ATtiny817 at 3.3V. According to the “General Operating Ratings” section in the device data sheet, the device can be safely run at 10 MHz with a 3.3V supply.

6. The code will now be changed to run the ATtiny817 at 10 MHz. Modify the start of `main()` as below:

```
int main(void)
{
    /*
     * Set the Main clock division factor to 2X,
     * and keep the Main clock prescaler enabled.
     */
    CLKCTRL.MCLKCTRLB = CLKCTRL_PDIV_2X_gc | CLKCTRL_PEN_bm;
```

7. Start a new debug session in order to recompile the project and program the device.
8. Halt code execution by clicking . Examine the clock settings in I/O view, depicted in [Figure 1-57](#).

Figure 1-57. Clock Settings in I/O View Remain Unchanged



Result: There is a problem! The prescaler remains unchanged.

9. Select the **MCLKCTRLB** register in I/O view, as indicated in [Figure 1-58](#).

Figure 1-58. Select MCLKCTRLB in I/O View

Name	Address	Value	Bits
MCLKCTRLA	0x60	0x00	<input type="checkbox"/>
MCLKCTRLB	0x61	0x11	<input checked="" type="checkbox"/>
PDIV		0x08	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
PEN		0x01	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>
MCLKLOCK	0x62	0x00	<input type="checkbox"/>
MCLKSTATUS	0x63	0x10	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
OSC20MCTRLA	0x70	0x00	<input type="checkbox"/>
RUNSTDBY		0x00	<input type="checkbox"/>
OSC20MCALIBA	0x71	0x9C	<input checked="" type="checkbox"/> <input type="checkbox"/>
CALSEL20M		0x02	<input checked="" type="checkbox"/> <input type="checkbox"/>
CAL20M		0x1C	<input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

- Push F1 on the keyboard to bring up a web-based register description.



Info: Internet access is required to use the web-based register description. Refer to an offline version of the ATtiny817 data sheet if internet access is not available.

- Find out if any access restrictions apply to the MCLKCTRLB register.



Result: The register is protected by the **Configuration Change Protection (CCP)** mechanism. Critical registers are configuration change protected to prevent unintended changes. These registers can only be modified if the correct unlock sequence is followed, as described in the data sheet.

- Replace the line of code which was just added with the following:

```
_PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_2X_gc | CLKCTRL_PEN_bm);
```



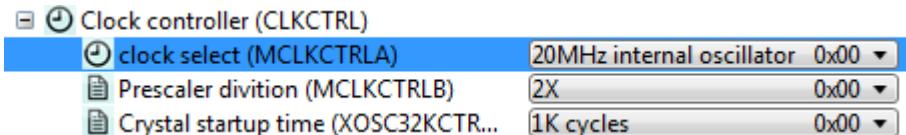
Info: `_PROTECTED_WRITE()` is an assembly macro that guarantees timing requirements for unlocking protected registers are met. It is recommended to use this macro when modifying protected registers.



tip: Right-click the macro name in the code and select **Goto Implementation** to navigate to the implementation of the macro. This is also possible by placing the cursor at the macro name in the code and pressing Alt+G on the keyboard. The same process can also be used for variable declarations and function implementations.

13. Stop the previous debug session and launch a new session to program the device with the changes.
14. Break code execution and use the I/O view to verify that the prescaler is now successfully set to 2X, as indicated in [Figure 1-59](#).

Figure 1-59. Clock Settings in I/O View Changed Successfully



 **tip:** The Processor Status window is the register view tool for the AVR Core. This tool can be opened from the top menu bar by going to **Debug** → **Windows** → **Processor Status**. This window will provide a detailed view of the status of the internal AVR Core registers. This view can be used to check if global interrupts are enabled; look for the I-bit in the status register.

 **Result:** The capabilities of the I/O view have been used to find and fix a bug in the project.

1.15.2 Memory View

 **To do:** Write two strings to the beginning of the ATtiny817 EEPROM and use Memory view to verify the EEPROM contents.

1. Add `#include <avr/eeprom.h>` after the `#include <avr/io.h>` line.
2. Add the following code before the `while(1)` loop in `main()`:

```
uint8_t hello[] = "Hello World";
eeprom_write_block(hello, (void *)0, sizeof(hello));
uint8_t hi[] = "AVR says hi";
eeprom_write_block(hi, (void *)0, sizeof(hi));
```

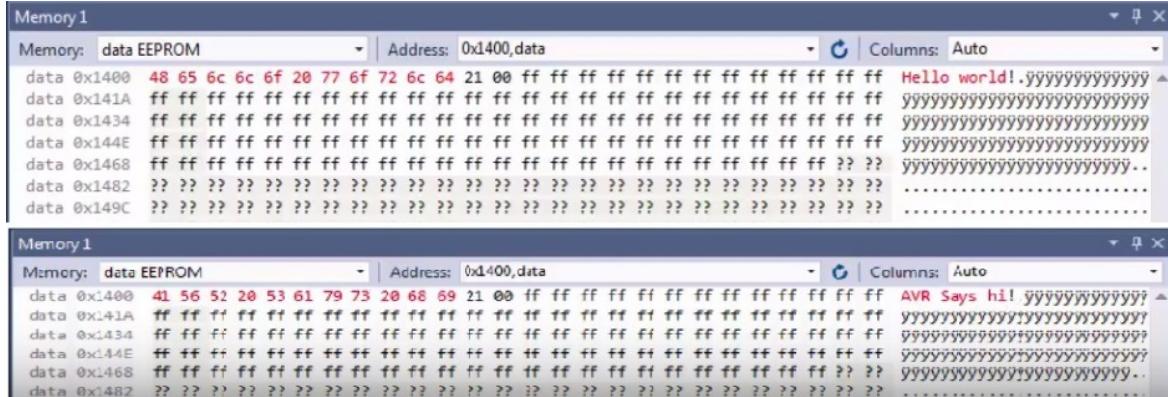
3. Place a breakpoint next to the first call to `eeprom_write_block()` as in [Figure 1-60](#).

Figure 1-60. Breakpoint to Halt for Checking EEPROM



4. Start a new debug session in order to program the device with the updated code.
5. After the breakpoint has been hit, open the memory window from the top menu bar by going to **Debug** → **Windows** → **Memory** → **Memory 1**. Look at the current content of the EEPROM.

6. Push F10 on the keyboard to step over the `eeprom_write_block()` call and verify the EEPROM write.
7. Allow the ATtiny817 to execute the next EEPROM write before verifying the write using the Memory view. The view should appear as in [Figure 1-61](#) at each interval respectively.

Figure 1-61. Memory View Updating After EEPROM Writes

tip: The Memory view tool can also be used to investigate the contents of other AVR memory sections, including the program memory. This can be useful when debugging bootloaders.



Result: The content of the EEPROM is updated after each call to `eeprom_write_block()`. The updated content is highlighted in red, and the ASCII interpretation of the EEPROM content matches the written strings. Therefore, the contents of EEPROM after writing to it have been verified using Memory view.

1.15.3 Watch Window

This is covered in more detail in section [Debugging 2: Conditional- and Action-Breakpoints](#), however, the note on how to cast pointers as an array in the Watch window is repeated here.



Info: A variable can also be added to the Watch window by clicking on an empty field name and typing the variable name. This way, it is even possible to cast a variable to a different data type for better readability in the Watch window. This is especially useful if it is required to look at an array that is passed to a function as a pointer.

For example, if an array is passed to a function, it will be passed to the function as a pointer. This makes it impossible for Atmel Studio to know the length of the array. If the length of the array is known, and it needs to be examined in the Watch window, the pointer can be cast to an array using the following cast:

```
* (uint8_t (*)[<n>])<name_of_array_pointer>
```

Where `<n>` is the number of elements in the array and `<name_of_array_pointer>` is the name of the array to be examined.

This can be tested on the `SW0_edge_count` variable by entering the following in an empty name field in the Watch window:

```
* (uint8_t (*)[5])&SW0_edge_count
```

Note that the '`&`' symbol must be used in this case to obtain a pointer to the variable.



Result: Atmel Studio has now been used to inspect and modify the contents of variables in the code.

Code used for Debugging 3

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

void LED_on(void);
void LED_off(void);
void LED_set_state(uint8_t state);
uint8_t SW_get_state(void);
uint8_t SW_get_state_logic(void);

int main(void)
{
    PORTB.DIRSET = PIN4_bm; /* Configure LED Pin as output */
    PORTB.PIN5CTRL = PORT_PULLUPEN_bm; /* Enable pull-up for SW0 pin */

    _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_2X_gc | CLKCTRL_PEN_bm);

    uint8_t Hello[] = "Hello World!";
    save(Hello, sizeof(Hello));
    uint8_t Hi[] = "AVR says hi!";
    save(Hi, sizeof(Hi));

    while(1)
    {
        uint8_t SW0_state = SW_get_state_logic(); /* Read switch state */
        LED_set_state(SW0_state); /* Set LED state */
    }
}

void save(const uint8_t* to_save, uint8_t size)
{
    eeprom_write_block(to_save, (void*)0, size);
}

uint8_t SW_get_state()
```

```
{  
    return !(PORTB.IN & PIN5_bm);  
}  
  
uint8_t SW_get_state_logic(void)  
{  
    static uint8_t SW0_prv_state = 0;  
    static uint8_t SW0_edge_count = 0;  
  
    uint8_t SW0_cur_state = !(PORTB.IN & PIN5_bm); /* Read the current SW0 state */  
    if (SW0_cur_state != SW0_prv_state) /* Check for edges */  
    {  
        SW0_edge_count++;  
    }  
    SW0_prv_state = SW0_cur_state; /* Keep track of previous state */  
  
    /*  
     * Report the switch as pushed when it is pushed or the edge counter is a  
     * multiple of 3  
     */  
    return SW0_cur_state || !(SW0_edge_count % 3);  
}  
  
void LED_off(void)  
{  
    PORTB.OUTSET = PIN4_bm; /* Turn LED off */  
}  
  
void LED_on(void)  
{  
    PORTB.OUTCLR = PIN4_bm; /* Turn LED on */  
}  
  
void LED_set_state(uint8_t state)  
{  
    if (state)  
    {  
        LED_on();  
    }  
    else  
    {  
        LED_off();  
    }  
}
```

2. Revision History

Doc. Rev.	Date	Comments
A	01/2018	Initial document release.

The Microchip Web Site

Microchip provides online support via our web site at <http://www.microchip.com/>. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at <http://www.microchip.com/>. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: <http://www.microchip.com/support>

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. **MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE.**

Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BeaconThings, BitCloud, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, KeeLoq logo, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, RightTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, chipKIT, chipKIT logo, CodeGuard, CryptoAuthentication, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscent Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PureSilicon, QMatrix, RightTouch logo, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2018, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-2529-8

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



MICROCHIP

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 Austin, TX Tel: 512-257-3370 Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 Detroit Novi, MI Tel: 248-848-4000 Houston, TX Tel: 281-894-5983 Indianapolis Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 Raleigh, NC Tel: 919-844-7510 New York, NY Tel: 631-435-6000 San Jose, CA Tel: 408-735-9110 Tel: 408-436-4270 Canada - Toronto Tel: 905-695-1980 Fax: 905-695-2078	Australia - Sydney Tel: 61-2-9868-6733 China - Beijing Tel: 86-10-8569-7000 China - Chengdu Tel: 86-28-8665-5511 China - Chongqing Tel: 86-23-8980-9588 China - Dongguan Tel: 86-769-8702-9880 China - Guangzhou Tel: 86-20-8755-8029 China - Hangzhou Tel: 86-571-8792-8115 China - Hong Kong SAR Tel: 852-2943-5100 China - Nanjing Tel: 86-25-8473-2460 China - Qingdao Tel: 86-532-8502-7355 China - Shanghai Tel: 86-21-3326-8000 China - Shenyang Tel: 86-24-2334-2829 China - Shenzhen Tel: 86-755-8864-2200 China - Suzhou Tel: 86-186-6233-1526 China - Wuhan Tel: 86-27-5980-5300 China - Xian Tel: 86-29-8833-7252 China - Xiamen Tel: 86-592-2388138 China - Zhuhai Tel: 86-756-3210040	India - Bangalore Tel: 91-80-3090-4444 India - New Delhi Tel: 91-11-4160-8631 India - Pune Tel: 91-20-4121-0141 Japan - Osaka Tel: 81-6-6152-7160 Japan - Tokyo Tel: 81-3-6880- 3770 Korea - Daegu Tel: 82-53-744-4301 Korea - Seoul Tel: 82-2-554-7200 Malaysia - Kuala Lumpur Tel: 60-3-7651-7906 Malaysia - Penang Tel: 60-4-227-8870 Philippines - Manila Tel: 63-2-634-9065 Singapore Tel: 65-6334-8870 Taiwan - Hsin Chu Tel: 886-3-577-8366 Taiwan - Kaohsiung Tel: 886-7-213-7830 Taiwan - Taipei Tel: 886-2-2508-8600 Thailand - Bangkok Tel: 66-2-694-1351 Vietnam - Ho Chi Minh Tel: 84-28-5448-2100	Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 Denmark - Copenhagen Tel: 45-4450-2828 Fax: 45-4485-2829 Finland - Espoo Tel: 358-9-4520-820 France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 Germany - Garching Tel: 49-8931-9700 Germany - Haan Tel: 49-2129-3766400 Germany - Heilbronn Tel: 49-7131-67-3636 Germany - Karlsruhe Tel: 49-721-625370 Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 Germany - Rosenheim Tel: 49-8031-354-560 Israel - Ra'anana Tel: 972-9-744-7705 Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781 Italy - Padova Tel: 39-049-7625286 Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340 Norway - Trondheim Tel: 47-7289-7561 Poland - Warsaw Tel: 48-22-3325737 Romania - Bucharest Tel: 40-21-407-87-50 Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 Sweden - Gothenberg Tel: 46-31-704-60-40 Sweden - Stockholm Tel: 46-8-5090-4654 UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820