

# **Getting Started with Microchip Studio**

### Introduction

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

### **Table of Contents**

Intr	oductic	on	1
1.	Gettin	ng Started	3
	1.1.	Microchip Studio, START, and Software Content	4
	1.2.	AVR <sup>®</sup> and SAM HW Tools and Debuggers	7
	1.3.	Data Visualizer and Power Debugging Demo	8
	1.4.	Installation and Updates	11
	1.5.	Microchip Gallery and Studio Extensions	13
	1.6.	Atmel START Integration	14
	1.7.	Creating a New Project	19
	1.8.	Creating From Arduino <sup>®</sup> Sketch	24
	1.9.	In-System Programming and Kit Connection	
	1.10.	I/O View and Other Bare-Metal Programming References	30
	1.11.	Editor: Writing and Re-Factoring Code (Visual Assist)	43
	1.12.	AVR <sup>®</sup> Simulator Debugging	51
	1.13.	Debugging 1: Break Points, Stepping, and Call Stack	
	1.14.	Debugging 2: Conditional- and Action-Breakpoints	64
	1.15.	Debugging 3: I/O View Memory View and Watch	70
2.	Revis	ion History	77
The	e Micro	chip Website	78
Pro	duct C	hange Notification Service	78
Cu	stomer	Support	78
Mic	rochip	Devices Code Protection Feature	78
Leg	gal Noti	ice	79
Tra	demarl	ks	79
Qu	ality Ma	anagement System	80
Wo	rldwide	e Sales and Service	81

### 1. Getting Started

Getting Started with Microchip Studio - playlist. Note that Atmel Studio has been renamed to Microchip Studio.



#### Prerequisites

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

Hardware prerequisites:

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

Software prerequisites:

- Microchip Studio for AVR and SAM Devices
- avr-gcc toolchain
- Latest Part Pack for tinyAVR<sup>®</sup> devices

Microchip Studio 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.



### 1.1 Microchip Studio, START, and Software Content

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

**Getting Started Topics** 



# AVR<sup>®</sup> & 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

#### 1.1.1 Atmel START

Atmel START is a web-based software configuration tool for various software frameworks, which helps you get 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<sup>®</sup>, Keil<sup>®</sup> µVision<sup>®</sup>, or simply generate a makefile.

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



Figure 1-1. Relation Between START, Software Content, and IDEs

#### 1.1.2 Software Content (Drivers and Middlewares)

#### Advanced Software Framework (ASF)

**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. It represents a complete re-design and -implementation of the whole framework, to improve the memory footprint, code performance, and the integration 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.

microchip.com: ASF Product Page

#### AVR<sup>®</sup> Code

**AVR Code**, 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**<sup>®</sup> MCUs. **AVR Code** is optimized for code-size and -speed, as well as simplicity and readability of code. AVR Code is configured by Atmel START.

#### 1.1.3 Integrated Development Environment (IDE)

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 START supports a range of IDEs, including Microchip Stuido, IAR Embedded Workbench<sup>®</sup>, Keil<sup>®</sup> µVision<sup>®</sup>.

Microchip Stuido is the integrated development platform (IDP) for developing and debugging all AVR and SAM microcontroller applications. The Microchip Studio 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 Microchip Studio has been optimized. Iterative development of START-based projects in Mirochip Studio is supported through re-configure and merge functionality.

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

#### 1.2 AVR<sup>®</sup> and SAM HW Tools and Debuggers

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

**Getting Started Topics** 



#### 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





#### 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 Microchip Studio 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<sup>®</sup> Cortex<sup>®</sup>-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 Microchip Studio, 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.

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



#### 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 pit 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.
11
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;
                                          //picoPower 1: So can run in sleep.
    ADC0.CTRLA |= ADC RUNSTBY bm;
       ADC0.CTRLE = ADC_WINCM_OUTSIDE_gc; //picoPower 3: So can evaluate own sample.
```

```
ADC0.INTCTRL = ADC_WCMP_bm;
       ADC0.WINHT = 200;
       ADCO.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
       USARTO.STATUS = USART TXCIF bm;
    VPORTB.OUT |= PIN6 bm;
       USARTO.TXDATAL = 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++)</pre>
       {
               (&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();
    3
}
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 Microchip Studio for AVR and SAM Devices, 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)



• 4 ×

icc al

#### Notifications 2

Notifications

Updates available for installed device packs Updates available for ATAUTOMOTIVE\_DFP, ATTINY\_DFP, SAMC20\_DFP, SAMC21\_DFP, SAMHA1\_DFP, XMEGAB\_DFP, XMEGAC\_DFP, XMEGAD\_DFP, XMEGAE\_DFP Fri 07.07.2017

New device packs available SAMD51\_DFP, SAME51\_DFP, SAME54\_DFP, UC3A\_DFP, UC3B\_DFP, UC3C\_DFP, UC3D\_DFP, UC3L\_DFP are available Thu 07.20.2017

#### 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 Microchip Studio installer: Microchip Studio
  - The web installer is a small file (<10 MB) and will download specified components as needed
  - The offline installer has all components embedded
- Microchip Studio can be run side-by-side with Atmel Studio 6.2 and older and AVR Studio<sup>®</sup>. Uninstallation of any previous versions is not required. Microchip Studio can not run side-by-side with Atmel Studio 7.
- · 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 USB/Serial hardware devices
- · Double-click the installer executable file and follow the installation wizard
- Once finished, the installer displays an option to **Start Microchip Studio after completion**. If you choose to open, then note that Microchip Studio will launch with administrative privileges since the installer was either launched as the administrator or with elevated privileges.
- If upgrading Microchip Studio from an earlier versions of Atmel Studio 7 or Microchip Studio and running the
  application with a different user account the local application user cache may have to be cleared. The reason for
  this is that the installer only clears the cache for the users it is installed with. This is done by deleting the folder:
  %localappdata%\Atmel\AtmelStudio\7.0 from the Windows File Exporer.
- In Microchip 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 Microchip Studio. To do this, from the *Microchip Studio 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.



### Microchip Studio Getting Started



### 1.5 Microchip Gallery and Studio Extensions

This section describes how Microchip Studio can be extended and updated through the Microchip 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<sup>®</sup> 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

This short video describes the process of adding extensions to Microchip Studio. It covers extensions included by default, what these are used for. Popular extensions are also covered, as well as how to modify Extension Options and Settings.

Website: Microchip Gallery.

#### 1.6 Atmel START Integration

The development experience between Atmel START and Microchip Studio has been optimized. This section demonstrates the iterative development process of START-based projects in Microchip Studio, through the *reconfigure* 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<sup>®</sup> 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 Microchip Studio check-box is checked.
- 3. Click on Download pack. An atmelstart.atzip pack file will be downloaded.

#### Figure 1-2. 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:	$\checkmark$
W µVision from Keil:	$\checkmark$
lAR Embedded Workbench:	$\checkmark$
Somnium DRT. (Atmel Studio plugin):	$\checkmark$
😚 Makefile (standalone):	$\checkmark$
Specify file name (optional): My Project	
<b>DOWNLOAD PACK</b>	



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

- 4. Launch Microchip Studio.
- 5. Select File > Import > Atmel START Project.

#### Figure 1-3. Import Atmel START Project

*	AtmelStudio (Administrato	r)				
File	Edit View ASF Project	Debug Too	ls W	/indow H	lelp	
	New		19	- C" -	🔚 🔍 🕨 M	1
	Open		ex	% 🕒 -	_ 8 G 📼 🛱	
	Close					•
×	Close Solution					
	Import	I	•	AVR32 Stu	udio Project	Ctrl+3
	Save Selected Items	Ctrl+S		AVR Studi	o 4 Project	Ctrl+4
	Save Selected Items As			Atmel Sta	rt Project	
	Save All	Ctrl+Shift+S		Project Te	mplate	Ctrl+T
	Export Template			,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		
	Page Setup					
•	Print	Ctrl+P				
	Recent Files					
	Recent Projects and Solutions					
×	Exit	Alt+F4				

- 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 1-4. Atmel START Project Importer

Atmel Start Importer		×
Import Atmel Start	Project	
Atmel Start Project(.atzip):	C:\Users\m43934\Downloads\My817Pro.atzip	Browse
View project summary (CMS)	IS package information)	
Project Name:	My817Pro1	
Location:	C:\Users\m43934\Documents\Atmel Studio\7.0	Browse
Solution:	Create New Solution	×.
Solution Name:	My817Pro1	
View project import summan	2	
Device Pack Manager	Help OK	Cancel

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





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

- 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 Microchip Studio.





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 Microchip Studio project.

**Getting Started Topics** 



# **Studio 7: Creating a New Project**



#### Video: Create New Project

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

- 1. Open Microchip Studio.
- 2. In Microchip Studio, go to File  $\rightarrow$  New  $\rightarrow$  Project as depicted in Figure 1-6.

lle	Edit View VAssistX AS	F Project De	bug	Tools Window Help		
	New	•	む	Project	Ctrl+Shift+N	~
	Open	•	*	File	Ctrl+N	-
	Close			Atmel Start Project		
E,	Close Solution			Atmel Start Example Project	ct	
	Import	•	₫	Example Project	Ctrl+Shift+E	
l	Save Selected Items	Ctrl+S	1-			
	Save Selected Items As		El.	1000		
1	Save All	Ctrl+Shift+S	1	~		
	Export Template		E.	/		
	Page Setup		r			
Ê	Print	Ctrl+P		Martin Landard Martin Landard		
	Recent Files	•	tmel	ATtiny817 microcontroller.	vare Supported	
	Recent Projects and Solutions	,	ated	development platform, the	kit	
		Alt. Ed	hed	avise in a sustemer design	anu	

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 barebones executable project. Give the project a Name and click OK. See Figure 1-7.

Figure 1-7. New Project Programming Language and Template Selection

New Project					? ×
▷ Recent	Sort b	by: Default	III 📃		Search Installed Templates (Ctrl+E)
<ul> <li>Installed</li> </ul>	INC. A	AVR XC8 C Application Project	C/C++	1	<b>Type:</b> C/C++
C/C++ Assembler	Juras contra	AVR XC8 C Library Project	C/C++		Creates an AVR 8-bit of AVR/ARM 32-bit C project
AtmelStudio Solut	ion	GCC C Executable Project	C/C++		
	GCC	GCC C Static Library Project	C/C++		#include course
	GCC	GCC C++ Executable Project	C/C++		int main(void)
	GCC	GCC C++ Static Library Project	C/C++		} printf("Hello"
	<u>ک</u>	SAM L11 Secure Solution v1.2	C/C++	-	GCC
<u>N</u> ame:	MyFirstProject				
Location:	c:\users\Documents\At	mel Studio\7.0		•	<u>B</u> rowse
Solution:	Create new solution			•	
Solution name:	MyFirstProject			[	Create directory for solution
					OK Cancel



**Tip:** All Microchip Studio projects belong to a solution, and by default, Microchip 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, Microchip 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
С	AVR XC8 C Appliction Project	Select this template to create an AVR 8-bit project configured to use the MPLAB XC8 compiler.
С	AVR XC8 Library Project	Select this template to create an AVR 8-bit MPLAB XC8 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 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.
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 1-8. 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.

Device Selection						×
Device Family:	All 🔹			817		×
Name	App./Boot Memory (Kbytes	)Data Memory (byte	s)EEPROM (bytes)	Device Info:		
ATtiny817	8	512	128	Device Name: Speed: Vcc: Family: Datasheet (Summary Device Page Supported Tools Atmel-ICE EDBG EDBG MSD UTAGICE3 M mEDBG Power Debugger STK600	ATtiny817 N/A N/A ATtiny	E
				C	<u>o</u> k	<u>C</u> ancel

Figure 1-8.	New Pro	iect Device	Selection
i iguio i o.		1001 001100	0010011011



**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**  $\rightarrow$  **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 ATtiny817 device. The **Solution Explorer** will list the content of the newly generated solution, as depicted in Figure 1-9. If not already open, it can be accessed through **View**  $\rightarrow$  **Solution Explorer** or by pressing Ctrl+Alt+L.

Figure 1-9. Solution Explorer



### 1.8 Creating From Arduino<sup>®</sup> Sketch

This section will outline the process of creating a new Microchip Studio project from an Arduino<sup>®</sup> Sketch. Getting Started Topics



Video: Create from 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** 



#### Video: Kit Connection and In-System Programming



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

ATtiny817 Xplained Pro - 015	)    ≠   ×   Start Page					
MCU board	ATtiny 017 Vicini	and Dro				
ATtiny817 Xplained Pro	ATUNY817 Aprail					
Extension						
	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.					
	S Atmel START example projects using this board New Atmel START project using this board					
	Launch Data Visualizer					
	External Links:					
	Technical Docum	nentation				
	ATtiny817 Device	<u>Datasheet</u>				
	Xplained Pro Har	<u>dware Development Kit (HDK) User Guide</u>				
	Kit Details					
	Serial number	ATML2654041800000150				
	Board name	ATtiny817 Xplained Pro				
	Manufacturer	Atmel				
	Target name	ATtiny817				
	Interfaces	SPI TWI GPIO CDC				
Show page on connect Update board database						

#### Figure 1-10. ATtiny817 Xplained Pro Start Page

- 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  $\rightarrow$  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 1-11. Tool Button

Window Help					
a 🛛 🖉 🗖 🔊 🖉	ebug 🔹	Debug Browse	r +		
🦼 📼 🛱 🐺 🛃 🖕 🕯 🕯	a 🕹   🔯 📮	🗯 ATtiny817	T	No Tool	Ŧ

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 1-12. Select Debugger/Programmer in Project Properties

MyFirstProject* ⇒ ×	ATtiny817 Xplained Pro - 0806 main.c
Build Build Events	Configuration: N/A    Platform: N/A
Toolchain	
Device	Selected debugger/programmer
Tool	
Components Advanced	EDBG ATML2654041800000806 Simulator Custom Programming Tool Erase entire chip • If Preserve EEPROM
	Debug settings
	Keep timers running in stop mode
	Cache all flash memory except



**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.

▲ WARNING 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 Microchip 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 1-13. Start Without Debugging



- When *Microchip Studio* 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.

## Microchip Studio Getting Started

# f

**Info:** If there is new firmware available for the selected tool, the **Firmware Upgrade** dialog will appear, as depicted in Figure 1-14. Click the **Upgrade** button to start the firmware upgrade.

#### Figure 1-14. Firmware Upgrade Dialog

Firmware Upgrade		<b>—</b>				
EDBG firmware must be updated before continuing						
	On Tool	On Disk				
Firmware Version	3.1c	3.1f				
Firmware Upgrade	_					
	Upgra	de Close				

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 *Microchip Studio* to upgrade the firmware for a tool.

#### Figure 1-15. Microchip Studio Available Tools (on view menu)

Available Tools		<b>-</b> ₽ ×
Tools and Simulators		Status
EDBG (ATML265404:	1800000693)	Disconnected
EDBG (ATML265404:		Connected
	LIEV/ICE Programming	
Simulator	Add Target	onnected
Simulator	Add Target Upgrade	onnected

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

This section describes how you would typically write code in *Microchip Studio*, 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

### Microchip Studio Getting Started

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 *Microchip Studio*, though the principles apply to most devices supported in *Microchip Studio*.

Getting Started Topics

#### Studio 7: I/O View & Bare-Metal Prog. Refs. MICROCHIP Filter: AD - 1 In this video: Context: em (EVS Voltage re Turn on LED, then blink with delay. Programming References: FREERUN (How to easily access & what to use each for) CTRLB CTRLC CTRLD Device datasheet CTRLE SAMPCTR MUXPOS Datasheet (from IO view) IO view (debugging) ..... Kit user-guide & schematics Device header files Name: CTRL Offset: 0x00 Reset: 0x00 Editor (Visual Assist) AVR<sup>®</sup>LibC Bit 7 - 8 Atmel START

#### 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 1.15 Debugging 3: I/O View Memory View and Watch as well as 1.11 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; }

Awarning 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 1-16. PORT Block Diagram from the PDF Data Sheet

#### Figure 1-17. 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

Microchip Studio 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.

Notes: 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 1-18. 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 1.15 Debugging 3: I/O View Memory View and Watch.

**Notes:** 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. Microchip Studio Button Functionality (Programming and Launching Debug Sessions)

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

Figure 1-19. Turning ON/OFF Kit LEDs Through Manipulating I/O View Registers when Debugging

Clicking PORT DIR4 cots PR4	Ind VO Ports (PORTC)     Oris (PORTC)     Oris (PORTC)     Oris (PORTC)     Oris (PORTC)     Discrete (LOCKBIT)     Ind Non-volatile Memory Con			
Clicking FOR I.DIR4 Sets FB4	Name	Address	Value	Bits
to OUTPUT	DIR 🔒	0x420	0x10	00000000
	DIRSET	0x421	0,10	00060000
LED on since LOW is default	DIRCLR	0x422	0x10	
	DIRTGL	0x423	0x10	00080000
C 100 RG 000 0000000000000000000000000000	D OUT	0x424	0%00	00000000
	DUTSET	0x425	0x00	00000000
	OUTCLR	0x426	0,00	00000000
	I OUTTGL	0x427	0,00	00000000
	10 IN	0x428	ONEF	
Terroren and the second s	IN INTELACS	0x429	0,00	00000000
	B D PINOCTIKL	0x430	0%00	0===0000
Ed Hand	H H PINICIKL	0x451	0.00	0
	I II PINZCIKL	0x432	0.00	0 0000
	I D PINISCI RL	0x433	0,00	0
	IS D PENECTRU	00434	0.00	00000
No. That is a second se	THE PINOLIKL	0.435	0.00	000000
	B PINICTRL	0x437	0x00	0===0000
	Processor Status	I/O Sel	ution E	plorer Properties

#### **Downloading Microchip Studio Documentation**

The data sheet can also be downloaded by using the Microchip Studio help system. In this case, a similar functionality will work offline. This is described here: 1.4.2 Downloading Offline Documentation.

#### Microchip Studio Editor (Visual Assist)

The Microchip Studio 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 1-20, while an overview of the code navigation is shown in Figure 1-21. In the next section, 1.11 Editor: Writing and Re-Factoring Code (Visual Assist), the editor features are covered in more detail.



Figure 1-20. Suggestion Functionality in the Microchip Studio Editor for Writing Code

Figure 1-21. Microchip Studio 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.
# Microchip Studio Getting Started

CFile1.bt	ATtiny617 Xplained Pro - 0113 iotn817.h 🗢 🗙 main.c*	Start Page			int	terrupt.h	* × ·
- PORT	ISC_enum • 🕂 🔸 typedef enum PORT_ISC_enum						• CG
1169	/* I/0 Ports */			v pinSctrl	,	x - +	· × 4
1170	Etypedef struct PORT_struct			A			
1171	{   manister 2 + DTP: /* Data Direction */		-	- BA (12 -	<ul> <li>Current Document</li> </ul>	-	
1172	registero_t DIR; /- Data Direction -/	PORTE	. )				
1173	registers_t DIRSET;/* Data Direction Set		DID				
1174	register8_t DIRCLR;/* Data Direction Cle	ear */	Daw				
1176	register8_t DIRTGL;/* Data Direction Tog	gle*/	DIRCLR				
1177	register8 t OUT; /* Output Value */		Parteert				
1178	register8 t OUTSET: /* Output Value Set *	1 9	DIRSET				
1179	registers + OUTCLEA /* Output Value Clear	*/ -	DIDTO				-
1180	registero_t outcik;/ output value clear		DIRTGL				
1181	registers_t OUTIGL;/* Output Value Toggi	Le */	TNI				
1182	register8_t reserved_8x8A;	-	114				
1183	registers_t_reserved_0x00;	A 1	INTELAGS				
1185	registers t reserved even		111110100				
1186	register8 t reserved 0x8E:	9	OUT				
1187	register8 t reserved 0x8F;						
1188	register8_t PINOCTRL; /* Pin 0 Control */	<i>e</i>	OUTCLR				
1189	register8_t PINICTRL; /* Pin 1 Control */	a	OUTSET				
1190	register8_t PIN2CTRL; /* Pin 2 Control */		OUTSET				
1191	register8_t PIN3CTRL; /* Pin 3 Control */	a)	OUTTGI	~			
1192	register8_t PIN4CTRL; /* Pin 4 Control */		OUTTOL				
1193	register8 + PINGCTRL: /* Pin 6 Control */	•	91				
1195	register8 t PIN7CTRL: /* Pin 7 Control */						
1196	register8 t reserved 0x18;						
1197	register8_t reserved_0x19;						
1198	register8_t reserved_0x1A;	- Tv	nina B				
1199	register8_t reserved_0x18;	1 1 1	ршч г		<b>.</b>		
1200	<pre>register8_t reserved_0x1C;</pre>						
1201	register8_t reserved_0x1D;	(a) a	7-6			L	-1
1202	register8_t reserved_0x1E;		≺eteren	ices Pu	JRISI	tru	CI
1203	1 DODT +-	1.000					2.2
1285	1 contract		dovico	hoodor	filo		
1286	/* Input/Sense Configuration select */		uevice	leauer	me)		
1207	Stypedef enum PORT_ISC_enum				100.5		
1208	(						
1209	<pre>PORT_ISC_INTDISABLE_gc = (0x00&lt;&lt;0), /* Iterrupt dis</pre>	abled but input	ouffer enabled */				
1210	PORT_ISC_BOTHEDGES_gc = (0x01<<0), /* Sense Both Ed	ges "/					
1211	PORT_ISC_RISING_gc = (0x02cc0), /* Sense Rising Edg	dan ti					
1212	PORT ISC THRUT DISABLE as a (avalua) /* Disital t	oput Buffer dies	lad #/				
1213	PORT ISC LEVEL or = (0x05((0), /* Serie low Level *	/ burrer disa	1200				
1215	) PORT ISC t:						

Figure 1-22. Suggestion Lists and the MCU Device Header Files

### 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 1-23. How to Find Schematics for a Particular Development Board

EXTENSION 3 HEADER 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 1-24. 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<sup>®</sup> 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 1-25. 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 1-26.

# **Microchip Studio Getting Started**

🗅 avr-libc Modules 🛛 🗙	1-2-2	Contract of the local division of the local	10	and the second se	
← → C ① www.nongnu.org/avr-libc/user-m	anual/modules.html			x 🖬 🐂 🖬 🛛	a, 🗢 O 🔳 🖾
Apps 🗋 Treehouse Learn Wei 🎦 Home - Atmel	Techni 🦞 Training Jira	🗑 npi-trd/ASF · GitHub 🔒	WebHome < Custom	Apps Program Revie	>> Cther bookmark
AVR Libc Home Page			-	interrupts	Development Pages
Main Page User Man	ual Librar	y Reference	FAQ	Example Projects	
Modules					
Here is a list of all modules:					
					[detail level 1 2]
<alloca.n>: Allocate space in the stack</alloca.n>					
Column has Character Operations					
Corpo b>: Sustem Errors					
sinttynes has integer Type conversions					
<math.h>: Mathematics</math.h>					
<setimp.h>: Non-local goto</setimp.h>					
<stdint.h>: Standard Integer Types</stdint.h>					
<stdio.h>: Standard IO facilities</stdio.h>					
<stdlib.h>: General utilities</stdlib.h>					
<string.h>: Strings</string.h>					
<time.h>: Time</time.h>					
<avr boot.h="">: Bootloader Support Utilities</avr>					
<avr cpufunc.h="">: Special AVR CPU functions</avr>					
<avr eeprom.h="">: EEPROM handling</avr>					
<avr fuse.h="">: Fuse Support</avr>					
<avr interrupt.h="">: Interrupts</avr>					
<avr io.h="">: AVR device-specific IO definition:</avr>	5				
<avr lock.h="">: Lockbit Support</avr>					
<avr pgmspace.h="">: Program Space Utilities</avr>					
<avr power.h="">: Power Reduction Management</avr>	nt				

#### AVR Libc Home Page × → C ③ www.nongnu.org/avr-libc/ ← <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 cli() Macros for writing interrupt handler functions #define ISR(vector, attributes) #define SIGNAL(vector) #define EMPTY\_INTERRUPT(vector) #include <avr/interrupt.h> Add header file, #define ISR\_ALIAS(vector, target\_vector) #define reti() Relevant IRQ Vector ISR(ADC\_vect) #define BADISR\_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 Microchip Studio, MPLAB X, 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 1-29. For the ATtiny817 this default value would be: #define F\_CPU 3333333.

Figure 1-27. Using START to creating a new Project for a Relevant boa	Figure 1-27	27. Using START to Cr	eating a New Project	for a Relevant Board
---	-------------	-----------------------	----------------------	----------------------

Apps [] Treehouse Learn Wet [	] Home - Atmel Technic 🏆 1	Iraining Jira ( ) npi-trd/ASF	GitHub 🔟 WebHome	< Custom [] Apps Pi	rogram Revie	20	Other book	kma
1tmel START					🗲 Retur	n To Front P	age   A	٩bo
REATE NEW PROJEC	т							
elect device or board before cre izes.	ating a new project. You c	an filter devices and boa	rds by what software	e you need and also	with hardware requir	ements such a	is memory	y
FILTERS		RESULTS						
HARDWARE	C⊘	817	$\otimes$	Show all	Show only boards	Show only o	levices	
SEARCH FOR SOFTWARE		Name	Architecture	Package	Pins	Flash	SRAM	0
Find software		ATtiny817-MNRES	AVR	VQFN24	24	8 KB	512 B	3
		ATtiny817-MNR	AVR	VQFN24	24	8 KB	512 B	
MIDDLEWARE	$\odot$	ATtiny817-MFR	AVR	VQFN24	24	8 KB	512 B	
+ Bootloader		Tiny817 QTou	uch Moisture Demo					ß
+ Crypto		ATtiny817 Xp	lained Pro					C
O DRIVERS	$\odot$	ATtiny817 Xp	lained Mini					ß
URIVERS	0							
AC	V 0 V m							
AC ADC								
AC ADC CRC								

# Microchip Studio Getting Started

	Atn	nel   STA		ny817					•	Return To Fro	ont Page   Help And Support
		{}	VIEW COD	EUC	t	-	SAVE CONFIGURATIO	DN		D	EXPORT PROJECT
	PINN		GURATO	R							0
aja		Pin label	Boar	d label		Signa	Show labels 🗸 🛞 Zoom i	n E	) Zoom out	Auto fit 🗸	
	# T	Pad User	Header	Pin ADC(+)	L	abel	Instance name     Component name		¥	150W T	
š	8	PA7	EXTLOT_	ADC(-).Q		-	Signal Jahel		AD OL	081,00	
NN	9	PB7	EXT1	GPIO1			Board header		C_SD.	N. Ide	
ā	10	PB6	EXT3	IRQ/GPIO			Board label		SPAJ2	MOSI, A	
	11	PB5	EXT3	SPI_SS_B			0		aPK DGL	Ids Ids	
	12	PB4	EXT1	GPIO2/L							
<b>B</b>	13	PB3	EXT1,EX	USART_R					2 2 2 3	8 8	
$\odot$	14	P82	EXT1,EX	USART_T							
	15	PB1	EXT1	PWM(-)		- 11	I2C_SCLI2C_SCLDGI_SCL	42 T	A\//2	18 PC1	SPI_MIGO.SPI_MISO.DGI_MISO
	16	PBO	EXT1	PWM(+)			SPI_SS_A	A3 ND R	AYR	17 PC0	SPI_SCK.SPI_SCK.DGI_SCK
	17	PCO	EXT1,EX	SPI_SCK,		-11	9	00 4	ATtiny617 8 KB + 512 E	8 15 PB1	PVM(-)
	18	PC1	EXT1,EX	SPI_MIS			SPL_SS_BIGPIO	44 5		14 PB2	USART_TX.UBART_TX.UART_TXD.XOUT32
	19	PC2	EXT1,EX	SPI_MOS		- 11	IRQIGPIO	45 5		13 PB3 = 5	USART_RKUSART_RX UART_RKD,XN32
	20	PC3	EXT1	SPI_SS_A		- 11			9744 Ante 284 284	984	
	21	PC4	DGI SPI	DGI_SS		- 11					
	22	PCS	EXT3	GPIO		- 11					
	23	PAD				-11			ADC(+	GP IO2	
	24	PA1	EXT1,EX	I2C_SDA,					101,8 014		
	4								THI	10	

# Figure 1-28. Showing Board Labels in START as an Alternative to the Kit Schematic





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

The Microchip Studio 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** 



### Video: Microchip Studio Editor (Visual Assist)

1. Starting with the basic functionality from 1.10 I/O View and Other Bare-Metal Programming References, main.c has the following code:

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





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.



 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.

```
PORTB.PIN5CTRL = PORT_PULLUPEN_bm;
```

	Goto Implementation	Alt+G	
	Refactor (VA)	•	
	Surround With (VA)	•	#ifdef #endif
	Insert Snippet	Ctrl+K, Ctrl+X	#if 0 #endif
	Surround With	Ctrl+K, Ctrl+S	#ifndef #endif
	Breakpoint	•	switch () { \$selected\$ }
ŧ	Run To Cursor	Ctrl+F10	if () { }
E	Run Flagged Threads To Cursor		if () { } else { }
5	Cut	Ctrl+X	if () { } else { }
ב	Сору	Ctrl+C	while () { }
ř.	Paste	Ctrl+V	for () { }
	Outlining	•	do { } while ()
)	View Help		- try { } catch {}
_			#ifdet guard in a header
			namespace (VA)
			#ifdef (VA)
			#region (VA)
			{}

(...)

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 */
```

© 2020 Microchip Technology Inc.

```
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.
- 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 1-31. Extract Method

PORTB.OUTCER = PIN4_Dm; /* Turn LED 0		Goto Implementation	Alt+G			
PORTB.OUTSET = PIN4_bm; /* Turn LED o		Refactor (VA)	*	Rename	Shift+Alt+R	
		Surround With (VA)	•	Change Signature		
	1 Insert Snippet		Ctrl+K, Ctrl+X	Encapsulate Field		
	<u>†</u> 1	Surround With	Ctrl+K, Ctrl+S	Create From Usage	Shift+Alt+C	
	Breakpoint			Create Declaration		
	*	Run To Cursor	Ctrl+F10	Create Implementation		
	Run Flagged Threads To Cu       Cut       Copy       Paste       Outlining       X       Add Data Plot	Run Flagged Threads To Cursor		Add Missing Case Statements		
		Cut	Ctrl+X	Add Member		
		Сору	Ctrl+C	Add Similar Member		
		Paste	Ctrl+V	Add Include		
		Outlining	•	Add/Remove Braces		
		Add Data Plot	Add Data Plot			
	W Remove Data Plot			Introduce Variable		
	0	View Help		Implement Interface		

A Extract Method dialog will appear. Name the function 'LED\_on', as indicated in the following figure. Figure 1-32. Extract Method Dialog

🍅 Extract Method		? 💌
New method name:		
LED_on		
Preview of method signature:		
void LED_on()		
	OK	Cancel

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  $LED_off()$ .

# Microchip Studio Getting Started

7. Create a variable for SW0 state, using Refactor  $\rightarrow$  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.

Refactor (VA)         Surround With (VA)         Insert Snippet         Surround With         Breakpoint         Run To Cursor         Run Flagged Threads To Cursor	Ctrl+K, Ctrl+X Ctrl+K, Ctrl+S Ctrl+F10		Rename Change Signature Encapsulate Field Create From Usage Create Declaration	Shift+Alt+R Shift+Alt+C
Surround With (VA) Insert Snippet Surround With Breakpoint Run To Cursor Run Flagged Threads To Cursor	Ctrl+K, Ctrl+X Ctrl+K, Ctrl+S Ctrl+F10		Change Signature Encapsulate Field Create From Usage Create Declaration	Shift+Alt+C
Insert Snippet Surround With Breakpoint Run To Cursor Run Flagged Threads To Cursor	Ctrl+K, Ctrl+X Ctrl+K, Ctrl+S Ctrl+F10		Encapsulate Field Create From Usage Create Declaration	Shift+Alt+C
Surround With Breakpoint Run To Cursor Run Flagged Threads To Cursor	Ctrl+K, Ctrl+S Ctrl+F10		Create From Usage Create Declaration	Shift+Alt+C
Breakpoint Run To Cursor Run Flagged Threads To Cursor	► Ctrl+F10		Create Declaration	
Run To Cursor Run Flagged Threads To Cursor	Ctrl+F10	-		
Run Flagged Threads To Cursor			Create Implementation	
			Add Missing Case Statements	
Cut	Ctrl+X		Add Member	
Сору	Ctrl+C		Add Similar Member	
Paste	Ctrl+V		Add Include	
Outlining	•		Add/Remove Braces	
Add Data Plot			Extract Method	
Remove Data Plot			Introduce Variable	
View Help		-	Implement Interface	
newnep			Document Method	
			Create File	
			Move Selection to New File	
			Move Implementation to Source File	
	Paste Outlining Add Data Plot Remove Data Plot View Help	Paste Ctrl+V Outlining Add Data Plot Remove Data Plot View Help	Paste Ctrl+V Outlining Add Data Plot Remove Data Plot View Help	Ctrl+V     Add Include       Outlining     Add/Remove Braces       Add Data Plot     Extract Method       View Help     Introduce Variable       Wiew Help     Occurrent Method       Create File     Move Selection to New File       Move Implementation to Source File     Description

The Introduce Variable dialog will appear, as depicted in Figure 1-34. Name the variable 'uint8\_t SW0\_state'. Figure 1-34. Introduce Variable Dialog

🍅 Introduce Variable		? 💌
Variable signature:		
uint8_t SW0_state		
	ОК	Cancel
		.d



Figure 1-33. Introduce Variable

**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
```



}

- 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. Microchip Studio will detect the argument SW0\_state, as indicated in Figure 1-35.

#### Figure 1-35. Extract Method with Argument

29 30	while(1)	
31	<pre>t uint8_t SW0_state = SW_get_state();</pre>	Extract Method     R     S
33 34	<pre>if (SW0_state) /* Check switch state */ {</pre>	New method name:
35		LED_set_state
36 37 38	LED_on(); } else {	Preview of method signature: void LED set state(uint8 t SW0 state)
39 40 41	<pre>LED_off();</pre>	
42	}	
43 }	}	
45		OK Cancel
		lh.

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. Microchip 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 1-36.

Figure 1-36. Contextual Rename

⊡voi  {	<pre>void LED_set_state(uint8_t SW0_state) {</pre>							
	if (SW0_state) /* Check switch state */							
	{		Goto Implementation	Alt+G				
	LED_o		Refactor (VA)	•		Rename	Shift+Alt+R	
			Surround With (VA)	•		Change Signature		
	} else {	ta	Insert Snippet	Ctrl+K, Ctrl+X		Encapsulate Field		
	LED_o	ta	Surround With	Ctrl+K, Ctrl+S		Create From Usage	Shift+Alt+C	
	}		Breakpoint	•		Create Declaration		
}	,	ŀt.	Run To Cursor	Ctrl+F10		Create Implementation		

The **Rename** dialog will appear, as depicted in Figure 1-37. Rename the SW0\_state variable to 'state'. Microchip 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 1-37. Contextual Renaming Dialog

🗢 Rename 🔹 🔋 💌							
Rename SW0_state (in file) to:							
state	Rename	Cancel					
☑ Display inherited and overridden references	ments and strings						
Search all projects							
<ul> <li>✓ ** c:\users\elizabeth.roy\Documents\Atmel Studio\7.0\debugging_in_atmel_studio\debu</li> <li>✓ LED_set_state (20): void LED_set_state(uint8_t SW0_state)</li> <li>✓ □ LED_set_state (22): if (SW0_state) /* Check switch state */</li> </ul>	gging_in_atmel_studio\	main.c					

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  $SWO\_state$  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)
    -{
```



# 1.12 AVR<sup>®</sup> 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



# Video: AVR Simulator Debugging

The code used in the video above was written in the video: 1.11 Editor: Writing and Re-Factoring Code (Visual Assist).

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

# Microchip Studio Getting Started

cappication1 ~	A mainte A funyour Aplained Pro - 0705 Pending Changes	
Build Build Events	Configuration: N/A    Platform: N/A	
Toolchain		
Device	Selected debugger/programmer	
Tool*		
Components	Simulator	
Advanced	EDBG ATML2654041800000703	
	P Custom Programming Tool	
	Erase entire chip	
	Preserve EEPROM	
	Debug settings	
	V Keep timers running in stop mode	
	Cache all flash memory except	

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 quicklaunch bar and hitting enter (or this can be found under Debug > Windows > Processor Status). Similarly, the **Disassembly** window can also be opened.

Standard Mode 🔻 🕇	pro ×
Most Recently Used (2)	
Debug $\rightarrow$ Windows $\rightarrow$ Processor Status	
Standard Mode 🔻 🕇	dis ×
Most Recently Used (2)	

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.

Name	Value	
Program Counter	0x00000380	<ul> <li>address of the instruction being executed</li> </ul>
Stack Pointer	0x00003821	<ul> <li>current stack pointer value</li> </ul>
X Register	0x0002	
Y Register	0x3FF1	
Z Register	0x3824	
Status Register	ITHSV	) N Z C
Cycle Counter	8186	<ul> <li>cycles elapsed from the simulation's start</li> </ul>
Frequency	1,000 MHz	
Stop Watch	8 186,00 us	<ul> <li>time elapsed based on cycles and frequency</li> </ul>
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.

		Read-modify-writ Elle Edit Yiew V O • O O + O	e (Debugging) - AtmelStudio (Assistă ASF Broject Bui 10 - 20 ■ ■ 1 × 17 0 ► 0 + 2 • 1 × 1	ld Debug Iools Window Help  フ・ワ・    名、 トロ Debug ・ Debug Transer・ Hee 7名 第一: G 回 時 資源 道士 広志 () 二、 AT	<ul> <li>→ # enable sleep iny617 T Simulator =</li> </ul>	ı_mode •   {	ຉ <b>୵ຬ</b> ⊠⊴⊂-
		Data Visualiper	lisassembly # × Read-modify	write main.c Read-modify-write.lss		Processor Status	
		Address: main				- Name	Value
	S ≢include <avr io.h=""> 9 10∎⊖int main(void)</avr>	Wewing Options 00000024 JMP 0x C:\Users\V43 {	00000000 Jump 959\Documents\Atmel Studi	o\7.0\Read-modify-write\Read-modify-write\Debug//./m	dn.c	Stack Pointer     X Register     Y Register     Z Register	Ox60000020     Ox3FFD     Ox6000     Ox3FFF     Ox6000
	11 {	PORTB	.DIR &= ~PIN4 bm;	2		Status Register	
0	13 PORTB.DIR &= ~PIN4 bm;	\$ 00000026	LDI R30,0x20	Load immediate		Cycle Counter	13 Miles
	14 PORTB.DIR  = PIN4_bm;	00000027	LDI R31,0x04	Load immediate		Stop Watch	11.00 µs
	15	00000028	LDD R24, Z+0	Load indirect with displacement		iii Registers	
	16 //PORTB.DIRSET = PIN4_bm; //PORTB_DISCIP = DIMA_bm;	00000029	ANDI R24,0xEF	Logical AND with immediate	Cu	cle Counter	0
	18	0000002A	STD Z+0,R24	Store indirect with displacement		ne counter	
	<pre>19 //VPORTB.DIR &amp;= ~PIN4_bm;</pre>	PORTB	.DIR  = PIN4 bm;	The second residence in the second seco	Fre	quency	1.000 MHz
	<pre>20 //VPORTB.DIR  = PIN4_bm;</pre>	00000028	LDD R24.Z+0	Load indirect with displacement	Sto	p Watch	0.00 µs
	21 22 ace/"non"\1	0000002C	ORI R24,0x10	Logical OR with immediate		100	0.00
	23	0000002D	STD Z+0, R24	Store indirect with displacement		R07	0x00
	24 while (1)	asm("	nop"):			R09	0x00
	25 {	0 0000002F	NOP No ope	eration		R10	0x00
	20 3	0000002F	RIMP PC-0x0000	Relative jump		#11 #12	0x00
		00000017 000	Undefined	,	8	R13	0x00

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

	Image: Sear - modely-write (Debugging) - Atmetibilitie       Eine Edit View VAssing? ASF Briject Build Debug Iools Window Help       Image: Sear - Sea	기 anable_sloep_mode · 디가 / 다 / 아이 등 24 프 ·	
	Data Visualizer Disassembly & X Read-modify-write main.c Read-modify-write.lss	Processor Status	
<pre>8 #include <avr io.h=""> 9 10 ⊜int main(void) 11 { 12 13 //PORTB.DIR &amp;= ~PIN4_bm; 14 //PORTB.DIRSET = PIN4_bm; 15 16 //PORTB.DIRSET = PIN4_bm; 17 //PORTB.DIRCLR = PIN4_bm; 18 19 VPORTB.DIR &amp;= ~PIN4_bm;</avr></pre>	Address main Viewing Options 00000024 JWP 0:00000000 Jump C:UUsers/VH39959Documents/Ltteel Studio\7.0\Read-modify-write\Read-modify-write\Debug//./main.c //PORTB.DIR &= ~PIN4_bm; //PORTB.DIRSET = PIN4_bm; //PORTB.DIR &= ~PIN4_bm; VPORTB.DIR &= ~PIN4_bm;	Name         Value           Program Counter         0x00000028           Stack Pointer         0x3FFD           X Register         0x0000           Y Register         0x0000           Status Register         0x0000           Status Register         0x000           Cycle Counter         2           Frequency         1.000 MHz           Stop Watch         2.00 µs	
<pre>20 20 20 21 22 23 23 24 while (1) 25 { 26 } 27 }</pre>	00000026     CBI 0x04,4     Clear bit in I/O register       VPORTB.DIR  = PIN4_bm;     00000027       00000027     SBI 0x04,4     Set bit in I/O register       asm("nop");     Set bit in I/O register       00000028     NOP       00000029     RJMP PC-0x0000       Relative jump       00000029     NOP       00000029     RJMP PC-0x0000       Relative jump       00000029     Undefined       00000036     NOP       Undefined	Registers           805         0.60           807         0.60           808         0.60           809         0.60           810         0.60           811         0.60           813         0.60	

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	•	1	
Name	Valu	le	
+ VO Ports (PORTB)			
Virtual Ports (VPORTB)			
Name	Address	Value	Bits
DIR DIR	0x420	0x00	00000000
DIRSET	0x421	0x00	00000000
DIRCLR	0x422	0x00	00000000
DIRTGL	0x423	0x00	00000000
OUT	0x424	0x00	00000000
OUTSET	0x425	0x00	00000000
OUTCLR	0x426	0x00	00000000
OUTTGL	0x427	0x00	00000000
IN IN	0x428	0x00	00000000
😑 🔣 INTFLAGS	0x429	0x20	
🛃 INT		0x20	
PINOCTRL	0x430	0x00	
🗉 🗎 PIN1CTRL	0x431	0x00	0 Bit 5 1000
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)		😑 🕄 INTFLAGS	0x429	0x20	
	77	{		INT 💽		0x20	
0	78	<pre>uint8 t intflags = PORTB.INTFLAGS;</pre>		PINOCTRL	0x430	0x00	
	79	PORTB.INTELAGS = intflags:	Internet in the second	🗉 🗎 PIN1CTRL	0x431	0x00	
	80			🗉 🗎 PIN2CTRL	0x432	0x00	
	01	hool SW state - SW get state():		E PIN3CTRL	0x433	0x00	
	01	boor Sw_state = Sw_get_state();		🗉 📄 PIN4CTRL	0x434	0x00	
	82	LED_set_state(Sw_state);	and High ()	PIN5CTRL	0x435	0x09	
	83			INVEN		0x00	
	84	[}	(* (4,44) (* (4,45)	ISC ISC		0x01	
	85		Haunty	PULLUPEN		0x01	
			and other lands	I PIN6CTRL	0x436	0x00	
			antinacioni mui	🗉 🗎 PIN7CTRL	0x437	0x00	
			-	Accuracy and a second second			

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 **Microchip Studio** 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.

```
#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);
}
```

Code Used to Demonstrate AVR® Simulator (Written for ATtiny187)

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

This section will introduce the debugging capabilities of Microchip Studio, 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**



# Video: Microchip Studio Debugging-1

The same code as the one created in section 1.11 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.

Set a breakpoint on the line getting the switch state, as indicated in Figure 1-38.
 Figure 1-38. Placing a Breakpoint



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 1-39.

Figure 1-39. Execution Halting when a Breakpoint is Hit





**Tip:** If a breakpoint is hit in a file that is not currently open, Microchip 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 1-40. Microchip Studio Buttons for Code Stepping



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

Button	Functionality	Keyboard Shortcut
*	Step Into Function Call	F11
3	Step Over	F10
1	Step Out of Function Call	Shift + F11
•	Run to Cursor	Ctrl + F10
Ī	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* **Call** *Can* be used to move to the next line after returning from the function. Pressing *Step Over Call* 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 1-41. Checking Value of SW\_state Using Mouse Hover



**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 Microchip Studio user interface menus. This is demonstrated below, by comparing the two figures Figure 1-42 and Figure 1-43. Note that each of the hits in the Quick Launch bar is from '*break*' related entries in the *Debug* menu.



Windows	*	•	Breakpoints	Alt+F9
Start Debugging and Break	Alt+F5		Data Breakpoints	
Attach to Target			Processor Status	
Stop Debugging	Ctrl+Shift+F5			
Start Without Debugging	Ctrl+Alt+F5	1		
Disable debugWIRE and Close				
Continue	F5			
Execute Stimulifile				
Set Stimulifile				
Restart				
Break All	Ctrl+F5			
QuickWatch	Shift+F9	L		
Step Into	F11			
Step Over	F10	L		
Step Out	Shift+F11			
Run To Cursor	Ctrl+F10	L		
Reset	Shift+F5			
Toggle Breakpoint	F9			
New Breakpoint	15		New Data <mark>Break</mark> point	Ctrl+Shift+D, B
Delete All <mark>Breakpoin</mark> ts	Ctrl+Shift+F9		Function Breakpoint	Ctrl+B
Disable All Breakpoints		-		
Clear All DataTips		F		
Export DataTips				
Import DataTips				
Save Dump As				
Ontions				

# Figure 1-43. 'Break' Hits in Debug Menu

Open the Breakpoints Window by clicking on the top result (**Debug**  $\rightarrow$  **Windows**  $\rightarrow$  **Breakpoints**). The Breakpoints Window lists all the breakpoints in the project, along with the current hit count, as depicted in Figure 1-44.



**Tip:** A breakpoint can temporarily be 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 1-45 section.

#### Figure 1-44. Breakpoints Window



## Figure 1-45. Disassembly View

•	main.c	▼	ts\/ 👻 ኛGo	Address:vector_4
	66	SK(PORTB_PORT_VECC)	+ (	Viewing Options
	67 68 69	<pre>uint8_t intflags =     PORTB.INTFLAGS; PORTB.INTFLAGS = intflags;</pre>	<ul> <li>- Constant - Statement - Stat</li></ul>	<pre>00000054 LDD R24,Z+9 Load indirect with disp PORTB.INTFLAGS = intflags; 00000055 STD Z+9,R24 Store indirect with dis  C:\Users\Y443959\Documents\Atmel Studio\7.0\Gett bool SW_state = SW_get_state();</pre>
•	70 71	<pre>bool SW_state = SW_get_state();</pre>	name never	<ul> <li>00000056 CALL 0x00000039 Call subroutine LED_set_state(SW_state);</li> </ul>
0	72	<pre>LED_set_state(SW_state);</pre>	1	00000058 CALL 0x0000002F Call subroutine }
	13	21	interpret and the prover of	0000005A POP R31 Pop register from stack
	74	}	No. Laboration Co.	0000005B POP R30 Pop register from stack

ToDo: 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**  $\rightarrow$  **Windows**  $\rightarrow$  **Call Stack**, as represented in Figure 1-46.

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

#### Figure 1-46. 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 1-47); this is because of compiler optimization.

## Figure 1-47. Call Stack with Optimization

Call Stack	
Name	
C Getting	3 Started.elf! LED_on Line: 39
Getting	3 Started.elf!vector_4 Line: 74
Call Stack	Breakpoints Command Window Immediate Windo
Stopped	

**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 *PORTB\_PORT\_vect* and look in the context field for the definition, as shown in Figure 1-48.

#### Figure 1-48. \_\_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 1-49.
- 6. Under AVR/GNU C Compiler → Optimization, set the Optimization Level to None (-O0) using the dropdown menu.

#### Figure 1-49. Disabling Compiler Optimizations

Build Build Events	Configuration: Active (Debug)	Platform: Active (AVR)
Toolchain* Device	Configuration Manager	
Tool Components Advanced	AVR/GNU Common     General     Output Files     AVR/GNU C Compiler     General     Preprocessor     Symbols     Directories     Optimization     Debugging     Warnings     AVR/GNU Linker     General     Libraries	AVR/GNU C Compiler ⇒ Optimization         Optimization Level:       Optimize (-O1)         Other optimization flags:       Optimize (-O1)         Image: I



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 shown in Figure 1-50.

```
Figure 1-50. Call Stack Without Optimization
```





**Tip:** Microchip 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.
 SWO is connected directly to GND and to PB5 through a current limiting resistor.
 SWO does not have an external pull-up resistor.
 SWO 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 Microchip Studio both as video (linked below) and handson document. The main topics are how to modify variables in the code, conditional- and action-breakpoints, as well as memory view.

**Getting Started Topics** 



#### Video: Debugging - 2

**ToDo:** Use Microchip 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 1.11 Editor: Writing and Re-Factoring 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 1-51.

## Figure 1-51. 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, Microchip 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 1-52.

#### Figure 1-52. Add Variable to Watch Window

main.c* + ×						
{\$ SW_get_state.SW0_edge	e_cc 🔻 🕂 🤗	static uint8_t SW0_edge_count = 0 -				
<pre>* multiple of 3 */ return SW0_cur_state    !(SW0_edge_count % 3); } 86 % *</pre>						
Name	Value	Type				
SW0_edge_count 26		uint8_t{data}@0x3e01				
Autos Locals Watch 1						
Stopped						

- 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 1-53.
  - Double-click a variable value in the Watch window
  - Type in the desired new value of the variable
  - Press Enter to confirm

#### Figure 1-53. Newly Updated Variable Value in the Watch Window

Watch 1		
Name	Value	Туре
SW0_edge_count	3	uint8_t{data}@0x3e01

**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 SWO edge count, disable all breakpoints and continue the

debug session by clicking P 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 Microchip 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 'a' symbol must be used in this case to obtain a pointer to the variable.

Result: Microchip 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 Microchip 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.

**ToDo:** Place a conditional breakpoint inside  $SW_get_state()$  to halt execution for debugging at every 5<sup>th</sup> 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 1-54.
- 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 1-54. Conditional Breakpoint Expression Example

Con	ditions	1, Must match sourc	e		
	Conditional Expression	• Is true	- (	(SW0_edge_count % 5) == 0) && SW0_cur_state	0
	Conditional Expression	- Is true	- (	(SW0_edge_count % 5) == 0) && SW0_cur_state	

- 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, Microchip 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: Microchip 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 Microchip 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.

**ToDo:** 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 1-55.
- 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 1-55. Action Breakpoint Example

Settings ×
Settings ×
J. J. J.
<ol> <li>Cancel</li> </ol>

- 5. Press Enter to confirm.
- 6. Start a debug session.
- 7. Open the Debug Output window by going to **Debug**  $\rightarrow$  **Windows**  $\rightarrow$  **Output**. It should list the variable contents as in Figure 1-56. If SW0 is pushed on the kit, the content is updated.

Figure 1-56. Debug Output Window Showing Variable Contents

Output	
Show output from: Debug	-   🖆   🖆 🞽   🏜
Prv state:0, Cur_state:0, Edge count:0 Prv state:0, Cur_state:0, Edge count:0 Prv state:0, Cur_state:0, Edge count:0 Prv state:0, Cur_state:0, Edge count:0 Prv state:0, Cur_state:0, Edge count:0	
1	
Autos Locals Watch1 Call Stack Breakpoints Output Error List	

WARNING When using action breakpoints, Microchip Studio will temporarily halt code execution 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 to log data only if a condition is satisfied.

Result: Microchip 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
    {
```



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

This section covers more advanced debugging topics with Microchip Studio both as video (linked below) and handson 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** 



# 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.

# ToDo: 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**  $\rightarrow$  **Windows**  $\rightarrow$  **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 1-57. 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 1-57. Manipulate Bit Value in Register Using I/O View

I/O				
	Filter:			-   🚄
±	Na E Fuses (FUS General Pu	me 6E) Irpose IO (G	PIO)	Value
. E	I/O Ports (	PORTA)		
+	VO I/O Ports (	PORTE)		
	I/O POILS (	FORTCJ		
	Name	Address	Value	Bits
	DIR	0x420	0x10	
	DIRSET	0x421	0x10	
	DIRCLR	0x422	0x10	
	DIRTGL	0x423	0x10	
	DUT	0x424	0x00	
	OUTSET	0x425	0x00	
	OUTCLR	0x426	0x00	
	OUTTGL	0x427	0x00	
	IN IN	0x428	0xEC	
•		0x429	0x00	
		0x430	00x00	
		0x431	00x00	
		0:432	00x00	
		0x433	0.00	
		0x454	0,08	
Ē		0x435	0,00	
Đ	PIN7CTRL	0x430	0x00	

**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 ensure 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 to recompile the project and program the device.
- Halt code execution by clicking . Examine the clock settings in I/O view, depicted in Figure 1-58. 8. Figure 1-58. Clock Settings in I/O View Remain Unchanged

🗉 🕘 Clock controller (	CLKCTRL
------------------------	---------

Clock select (MCLKCTRLA)	20MHz internal oscillator	0x00 🔻
Prescaler divition (MCLKCTRLB)	6X	0x08 🔻
Crystal startup time (XOSC32KCTR	1K cycles	0x00 🔻

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

Select the MCLKCTRLB register in I/O view, as indicated in Figure 1-59. 9.

# Figure 1-59. Select MCLKCTRLB in I/O View

Name	Address Value	Bits
MCLKCTRLA	0x60 0x00	
MCLKCTRLB	0x61 0x11	
PDIV	0x08	
PEN	0x01	
MCLKLOCK	0x62 0x00	
MCLKSTATUS	0x63 0x10	
OSC20MCTRLA	0x70 0x00	
RUNSTDBY	0x00	
OSC20MCALIBA	0x71 0x9C	
CALSEL20M	0x02	
CAL20M	0x1C	

- 10. 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.
- 11. 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.

12. 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 ensure 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 1-60.

#### Figure 1-60. Clock Settings in I/O View Changed Successfully

Clock controller (CLKCTRL)

Clock select (MCLKCTRLA)	20MHz internal oscillator	0x00 🔻
Prescaler divition (MCLKCTRLB)	2X	0x00 🔻
Crystal startup time (XOSC32KCTR	1K cycles	0x00 ▼



**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**  $\rightarrow$  **Windows**  $\rightarrow$  **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

**ToDo:** 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 1-61.

#### Figure 1-61. Breakpoint to Halt for Checking EEPROM



- 4. Start a new debug session 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**  $\rightarrow$  **Windows**  $\rightarrow$  **Memory**  $\rightarrow$  **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 1-62 at each interval respectively.

Figure 1-62. Memory View Updating After EEPROM Writes

Memory 1				<b>-</b> ↓ ×
Memory: data	EEPROM	Address: 0x1400, data	•	Columns: Auto -
data 0x1400 data 0x141A data 0x1434 data 0x144E data 0x144E data 0x1468 data 0x1482	48         65         6c         6f         20         77         6           ff         ff <td>f 72 Gc 64 21 00 ff f ff ff ff ff ff ff ff f ff ff ff ff</td> <td>ff       ff       <td< td=""><td><pre>ff ff Hello world!.9999999999999 * ff ff 999999999999999999</pre></td></td<></td>	f 72 Gc 64 21 00 ff f ff ff ff ff ff ff ff f ff ff ff ff	ff       ff <td< td=""><td><pre>ff ff Hello world!.9999999999999 * ff ff 999999999999999999</pre></td></td<>	<pre>ff ff Hello world!.9999999999999 * ff ff 999999999999999999</pre>
data 0x149C Memory 1	11 11 11 11 11 11 11 1		** ** ** ** ** ** ** ** ** **	+ ₽×
Memory: data	EEPROM	Address: 0x1400,data	•	Columns: Auto -
data 0x1400 data 0x141A data 0x1434 data 0x144E data 0x1468	41 56 52 20 53 61 79 7 ff ff ff ff ff ff ff ff ff ff ff ff ff	3 20 68 69 21 00 ff f ff ff ff ff ff ff ff f ff ff ff ff	ff         ff<	<pre>ff ff AVR Says hi! 999999999999 = ff ff f yyyyyyyyyyyyyyyyyyyyyyyyyyyyy</pre>



**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 <code>eeprom\_write\_block()</code>. 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 1.14 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 Microchip 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 'a' symbol must be used in this case to obtain a pointer to the variable.

**Result:** Microchip 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 SWO 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 (SWO 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 SWO cur state = ! (PORTB.IN & PIN5 bm); /* Read the current SWO state */
    if (SW0_cur_state != SW0_prv_state)
                                                   /* Check for edges
                                                                                    */
        SW0_edge_count++;
                                                   /* Keep track of previous state */
    SW0 prv state = SW0 cur 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
В	01/2021	Renaming to Microchip Studio
А	2018	Initial version

### The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 (FAQs), technical support requests, online discussion groups, Microchip design partner 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

### **Product Change Notification Service**

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

## Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: 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 specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- · Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code
  protection does not mean that we are guaranteeing the product is "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 is provided for the sole purpose of designing with and using Microchip products. Information 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.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". 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 ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. 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, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, WinPath, and ZL 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, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, 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.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks 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.

© 2020, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-7362-6

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

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.



# **Worldwide Sales and Service**

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office	Australia - Sydney	India - Bangalore	Austria - Wels
2355 West Chandler Blvd.	Tel: 61-2-9868-6733	Tel: 91-80-3090-4444	Tel: 43-7242-2244-39
Chandler, AZ 85224-6199	China - Beijing	India - New Delhi	Fax: 43-7242-2244-393
Tel: 480-792-7200	Tel: 86-10-8569-7000	Tel: 91-11-4160-8631	Denmark - Copenhagen
Fax: 480-792-7277	China - Chenɑdu	India - Pune	Tel: 45-4485-5910
Technical Support:	Tel: 86-28-8665-5511	Tel: 91-20-4121-0141	Fax: 45-4485-2829
www.microchip.com/support	China - Chongging	Japan - Osaka	Finland - Espoo
Web Address:	Tel: 86-23-8980-9588	Tel: 81-6-6152-7160	Tel: 358-9-4520-820
www.microchip.com	China - Dongguan	Japan - Tokyo	France - Paris
Atlanta	Tel: 86-769-8702-9880	Tel: 81-3-6880- 3770	Tel: 33-1-69-53-63-20
Duluth, GA	China - Guangzhou	Korea - Daegu	Fax: 33-1-69-30-90-79
Tel: 678-957-9614	Tel: 86-20-8755-8029	Tel: 82-53-744-4301	Germany - Garching
Fax: 678-957-1455	China - Hangzhou	Korea - Seoul	Tel: 49-8931-9700
Austin, TX	Tel: 86-571-8792-8115	Tel: 82-2-554-7200	Germany - Haan
Tel: 512-257-3370	China - Hong Kong SAR	Malaysia - Kuala Lumpur	Tel: 49-2129-3766400
Boston	Tel: 852-2943-5100	Tel: 60-3-7651-7906	Germany - Heilbronn
Westborough, MA	China - Nanjing	Malaysia - Penang	Tel: 49-7131-72400
Tel: 774-760-0087	Tel: 86-25-8473-2460	Tel: 60-4-227-8870	Germany - Karlsruhe
Fax: 774-760-0088	China - Qingdao	Philippines - Manila	Tel: 49-721-625370
Chicago	Tel: 86-532-8502-7355	Tel: 63-2-634-9065	Germany - Munich
Itasca. IL	China - Shanghai	Singapore	Tel: 49-89-627-144-0
Tel: 630-285-0071	Tel: 86-21-3326-8000	Tel: 65-6334-8870	Fax: 49-89-627-144-44
Fax: 630-285-0075	China - Shenvang	Taiwan - Hsin Chu	Germany - Rosenheim
Dallas	Tel: 86-24-2334-2829	Tel: 886-3-577-8366	Tel: 49-8031-354-560
Addison, TX	China - Shenzhen	Taiwan - Kaohsiung	Israel - Ra'anana
Tel: 972-818-7423	Tel: 86-755-8864-2200	Tel: 886-7-213-7830	Tel: 972-9-744-7705
Fax: 972-818-2924	China - Suzhou	Taiwan - Taipei	Italy - Milan
Detroit	Tel: 86-186-6233-1526	Tel: 886-2-2508-8600	Tel: 39-0331-742611
Novi. MI	China - Wuhan	Thailand - Bangkok	Fax: 39-0331-466781
Tel: 248-848-4000	Tel: 86-27-5980-5300	Tel: 66-2-694-1351	Italy - Padova
Houston, TX	China - Xian	Vietnam - Ho Chi Minh	Tel: 39-049-7625286
Tel: 281-894-5983	Tel: 86-29-8833-7252	Tel: 84-28-5448-2100	Netherlands - Drunen
Indianapolis	China - Xiamen		Tel: 31-416-690399
Noblesville. IN	Tel: 86-592-2388138		Fax: 31-416-690340
Tel: 317-773-8323	China - Zhuhai		Norway - Trondheim
Fax: 317-773-5453	Tel: 86-756-3210040		Tel: 47-72884388
Tel: 317-536-2380			Poland - Warsaw
Los Angeles			Tel: 48-22-3325737
Mission Viejo, CA			Romania - Bucharest
Tel: 949-462-9523			Tel: 40-21-407-87-50
Fax: 949-462-9608			Spain - Madrid
Tel: 951-273-7800			Tel: 34-91-708-08-90
Raleigh, NC			Fax: 34-91-708-08-91
Tel: 919-844-7510			Sweden - Gothenberg
New York, NY			Tel: 46-31-704-60-40
Tel: 631-435-6000			Sweden - Stockholm
San Jose. CA			Tel: 46-8-5090-4654
Tel: 408-735-9110			UK - Wokingham
Tel: 408-436-4270			Tel: 44-118-921-5800
Canada - Toronto			Fax: 44-118-921-5820
Tel: 905-695-1980			1 dx. ++-110-02 1-0020
Fax: 905-695-2078			
1 a. 000-000-2010			