Getting Started with Events on the tinyAVR 1-series

Getting Started with Events on the tinyAVR® 1-series

Prerequisites

- **Hardware Prerequisites**
  - ATtiny817 Xplained Pro evaluation kit
  - Micro-USB cable (Type-A/Micro-B)
  - A breadboard
  - A compatible potentiometer
  - Three compatible male-to-female wires
  - Internet connection
- **Software Prerequisites**
  - Atmel Studio 7.0
  - Web browser. A list of supported browsers can be found here: http://start.atmel.com/static/help/index.html?GUID-51435BA6-0D59-4458-A413-08A066F6F7CA
- **Estimated Completion Time: 120 minutes**

Introduction

This hands-on training will demonstrate how to develop AVR® applications in Atmel Studio and Atmel START along with the rich user interface and other great development tools that they provide.

Atmel START helps to get started with Microchip microcontroller development. It allows to select MCU, configure software components, drivers, middleware, and example projects to the embedded application in a usable and optimized manner. Once the configuration is complete, the project can be generated in Atmel Studio or another third-party development tool. An IDE is used to develop the code required to extend the functionality of the project, into the final product, as well as compile, program, and debug the downloaded code.

With Atmel START:

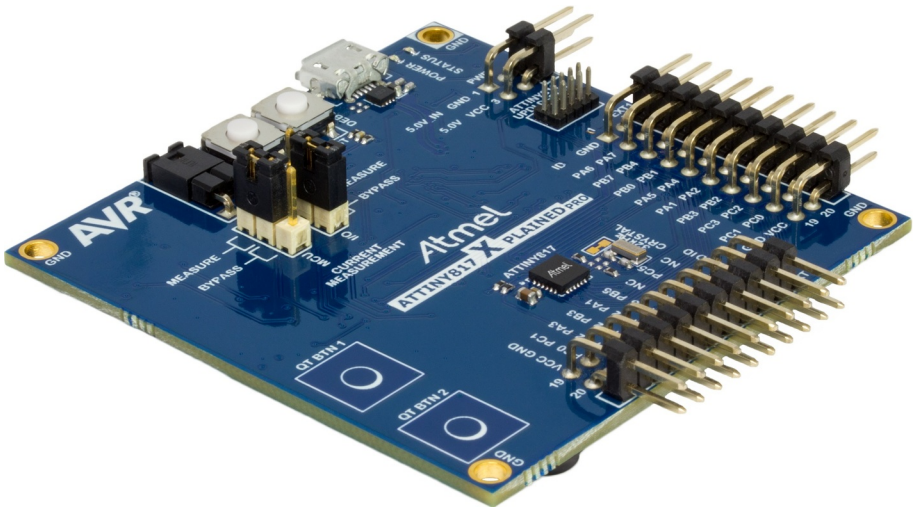
- Get help selecting the MCU based on both software and hardware requirements
- Find and develop examples
- Configure drivers, middleware, and example projects
- Get help with setting up a valid PINMUX layout
- Configure system clock settings

The ATtiny817 Xplained Pro evaluation kit is a hardware platform for evaluating the AVR ATtiny817 microcontroller. A fully integrated embedded debugger is included on the kit, which provides seamless integration with Atmel Studio. Easy access to the features of the ATtiny817 is enabled by the kit, facilitating easy integration of the device in a customer design.

This training demonstrates how to configure the application in Atmel START. Reconfigure the Atmel START project and continue the implementation in Atmel Studio 7.

The peripherals used to create applications are GPIO, RTC, ADC, CPUINT, USART, and Event System.

Figure 1. ATtiny817 Xplained Pro



The following topics are covered:

- Driver configuration in Atmel START
- PINMUX driver configuration
- LED toggling triggered by RTC overflow interrupt
- USART configuration with string print to USART serial-port terminal
- RTC overflow interrupt triggers an ADC conversion and then ADC result-ready interrupt triggers ADC data print to USART terminal
- RTC overflow interrupt triggers an ADC conversion and then an ADC result triggers ADC data print to USART terminal if it is outside a defined window
- RTC overflow event triggers an ADC conversion, then if an ADC result is outside a defined window, waking up CPU with an IRQ
- How to use Data Visualizer in Atmel Studio to view the USART print to the embedded terminal and graph

Table of Contents

1. Icon Key Identifiers

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

Info: Delivers contextual information about a specific topic.

Tip: Highlights useful tips and techniques.

To do: Highlights objectives to be completed.

Result: Highlights the expected result of an assignment step.

Warning: Indicates important information.

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

2. Assignment 1: RTC Overflow Interrupt Triggers LED Toggling

In this assignment an application that triggers LED toggling from the RTC overflow interrupt will be developed.

Atmel | START will be used to configure the RTC, clock, and PINMUX settings. Based on this an Atmel Studio 7 project will be generated.

Code will be developed in Atmel Studio 7 using PINMUX and RTC driver functions generated by *Atmel | START*.

Info: On the ATtiny817 Xplained Pro board, LED0 is connected to pin PB4.

Peripherals used:

- RTC
- GPIO (PB4)

Clock details:

- 3.333 MHz main clock
- 1 kHz RTC clock

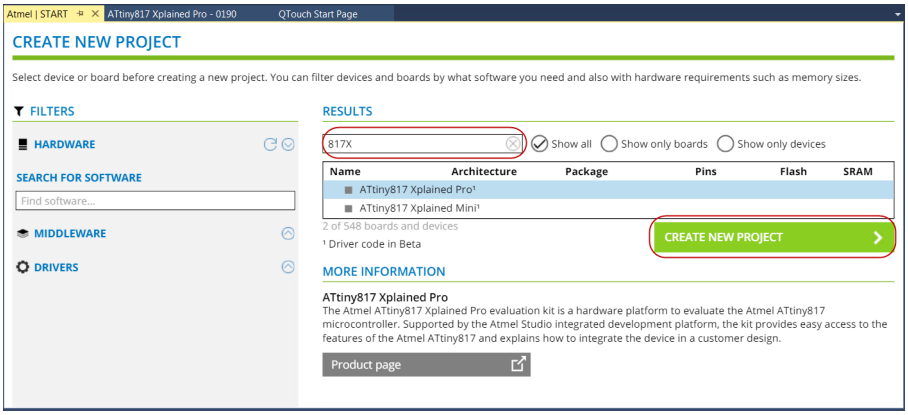
2.1 *Atmel | START* Project Creation

Configure PINMUX driver, RTC driver, and CLOCK in *Atmel | START* and create the project.

To do: Create a new *Atmel | START* Project.

1. Open ***Atmel Studio***.
2. Select *File → New → Atmel Start Project*.
3. The *CREATE NEW PROJECT* window appears within Atmel Studio 7. In the "*Filter on device...*" text box, enter *817X*, then select *ATtiny817 Xplained Pro* from the list and verify that *ATtiny817Xplained Pro* is highlighted, then click *CREATE NEW PROJECT*, as shown below.

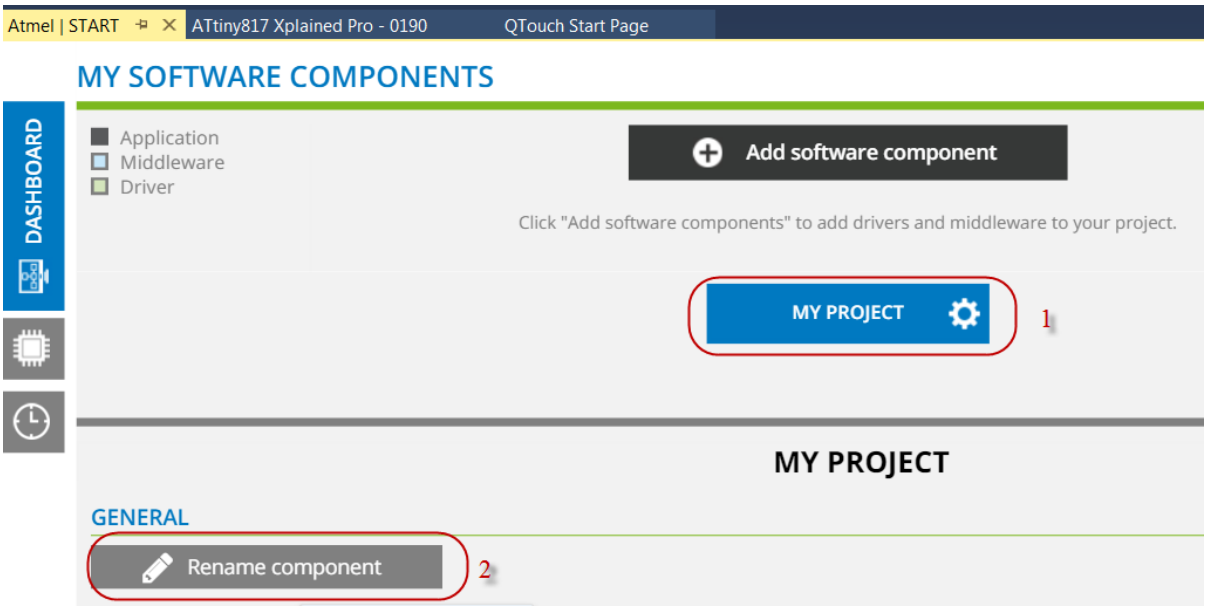
Figure 2-1. CREATE NEW PROJECT



Info: Now the *MY SOFTWARE COMPONENTS* window appears.

4. In the *MY SOFTWARE COMPONENTS* window, rename the project as shown below with 1-2 red markings. First click *MY PROJECT* and then select *Rename Component*.

Figure 2-2. Rename Component



Info: Now *RENAME COMPONENT* window will be displayed.

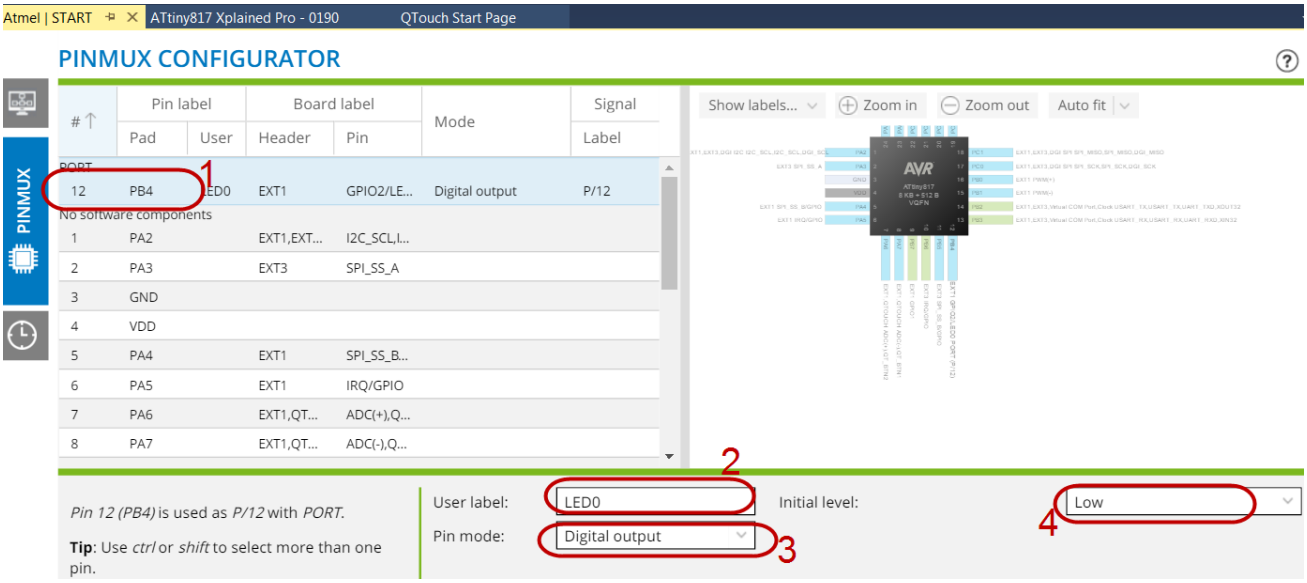
5. In the *RENAME COMPONENT* window specify the new project name as "*ATtiny817_getting_start_events*" and click *Rename*.

6. Open PINMUX configuration by clicking on  in the navigation tab on the left side of the window.

Info: The PINMUX configurator displays an illustration of the device package selected. It shows which pins are currently used by different peripherals. GPIO pins can be configured here.

Info: Here PB4 is configured as LED0 which is available on the ATtiny817 Xplained Pro. Configuration is shown in 1-4 red markings below.

Figure 2-3. PINMUX Configuration LED0



7. Execute the numbered configuration steps 1-4 from the figure as described below:
   - Click "PB4" in the PINMUX list.
   - Enter "User label:" as *LED0*.
   - Select "Pin mode:" *Digital output*.
   - Make sure "Initial level" is set to *Low*.

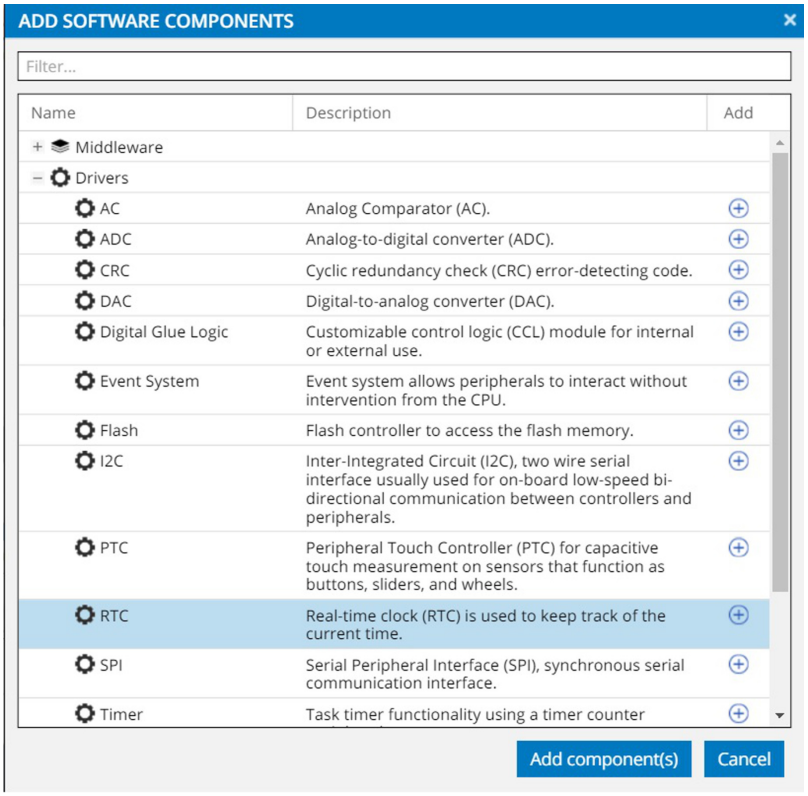
Info: Technical documents related to the ATtiny817 Xplained Pro can be downloaded from the *ATtiny817 Xplained Pro → Technical Documentation* page within Atmel Studio. The *ATtiny817 Xplained Pro* page is displayed when the ATtiny817 Xplained Pro board is connected to the PC.

8. Add the RTC component:

Click DASHBOARD in the navigation tab on the left side of the window. Then click Add software component.

Info: The "ADD SOFTWARE COMPONENTS" window will appear.

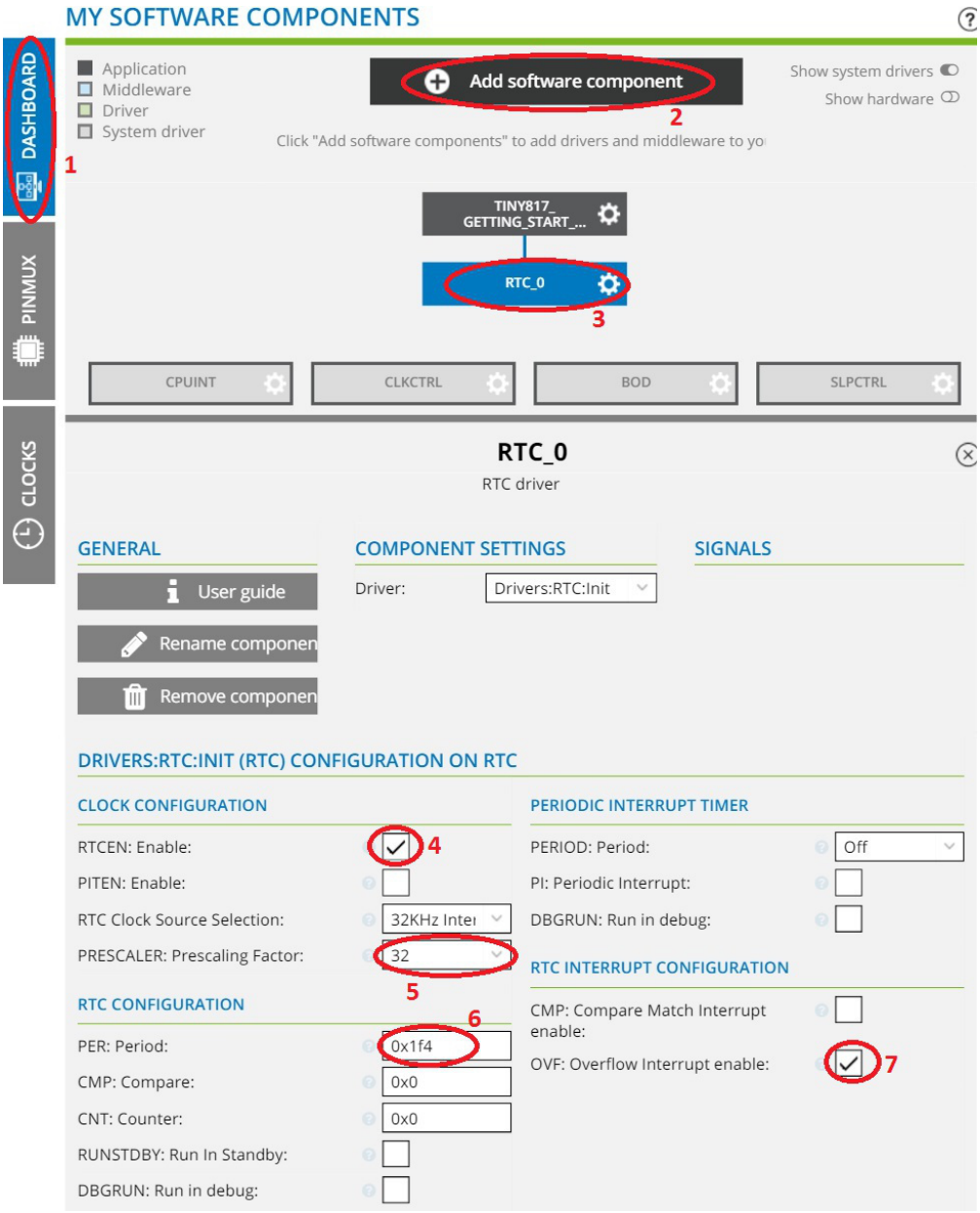
Figure 2-4. Add RTC Component



9. **RTC Configuration:**
   Expand "*Drivers*" in the "*ADD SOFTWARE COMPONENTS*" window as shown above, scroll down to locate "*RTC*", and click to select it. Then click the "*Add component(s)*" button. The "*RTC_0*" module will now be added to the project, as shown below.

Info: In the figure below, red markings 1-2 refer to what has been completed above. Steps 3-7 describes the configuration of the RTC.

Figure 2-5. RTC Configuration



10. Execute the numbered configuration steps 3 through 7 from the figure above as:
    - Click "*RTC_0*" to open the RTC driver configuration page
    - Tick the "*RTCEN*" checkbox to configure the RTC to be enabled in the RTC initialization routine that will be generated by *Atmel | START*
    - Click the "*PRESCALER*" dropdown menu and select "*32*" to configure the RTC clock prescaler to 32

Info: This will configure the RTC count frequency to 1 kHz as the default RTC clock source is the 32 kHz Internal Ultra Low Power Oscillator.

- Continue by entering decimal number 500, equivalent to the hexadecimal value 0x1f4, in the "*PER*" text box to define the RTC period
- Tick the "*OVF*" checkbox to enable the RTC overflow interrupt

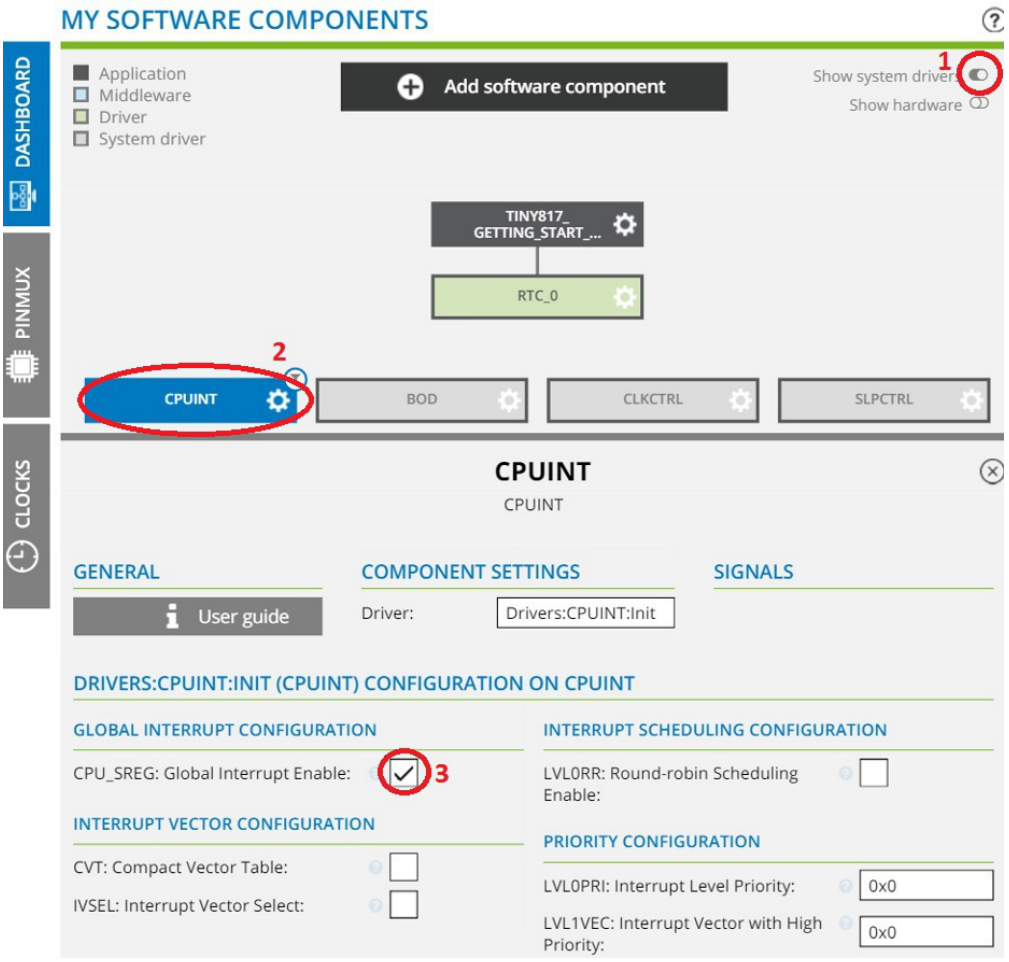
Info: The RTC overflow interrupt is configured to trigger approximately twice per second, as the period is set to 500 with a RTC clock frequency of 1 kHz.

11. **Enable CPUINT Global Interrupt:**

Info: The three steps involved in configuring the CPUINT module are described below and illustrated in Figure 2-6.

- Make sure the "*Show system drivers*" slider in the top right corner of the "*DASHBOARD*"-view is switched ON
- Click "*CPUINIT*" to open the CPUINT configuration view
- Tick the "*CPU_SREG: Global Interrupt Enable*" checkbox such that the initialization routine generated by *Atmel | START* will enable global interrupts

Figure 2-6. CPUINT Configuration



Info: The RTC and CPUINIT are now configured. The RTC will now be able to trigger its overflow interrupt.

12. **Clock configuration**: Open the clock configuration view by clicking on  in the navigation tab on the left side of the window.

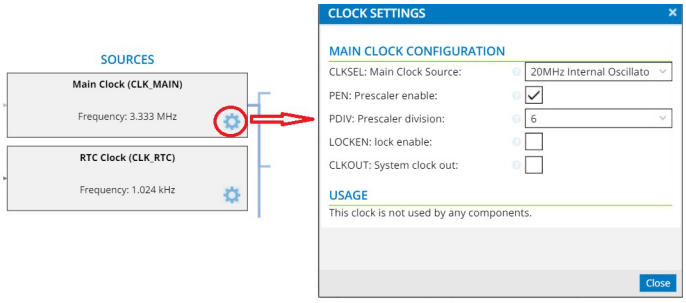
Info: The *CLOCK CONFIGURATOR* view will now be displayed. It contains oscillators and clock sources of different types. Required clock sources can be selected and configured. The corresponding output frequency is then calculated and presented. The settings dialog for each element in the clock configuration view can be opened by clicking the associated cog-wheel.

- The *Oscillators* section displays the oscillators available for the selected device
- The *Source* section is used to configure clock frequency by selecting input signal and prescaling factor

13. Open the "*Clock Settings*"- dialog for "*Main Clock*" by clicking its corresponding cog-wheel to view the default clock settings as shown below.

Info: *CLOCK SETTINGS* window will be displayed.

Figure 2-7. MAIN CLOCK CONFIGURATION

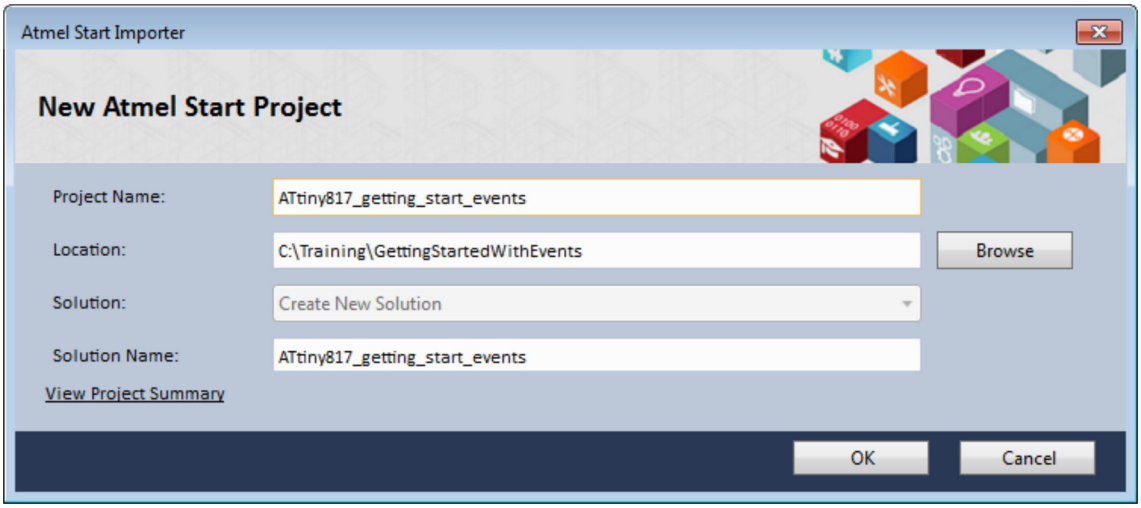


Info: For this application the default clock settings are kept as is. Here, the Main Clock source is the 20 MHz Internal Oscillator with prescaler division set to 6. The resulting CPU clock frequency is 3.33 MHz. Clicking the question mark next to each configuration will provide the data sheet description of the individual bit setting. The RTC clock was configured to 1 kHz earlier and this is also shown in the figure.

14. Click *Close* to close the *CLOCK SETTING* window.
15. Now, click the *GENERATE PROJECT* button 

.
16. Select the desired path where the project should be stored, as shown below, and then click *OK*.

Figure 2-8. Project Importer Window



Result: The *Atmel* | *START* project has been created in Atmel Studio.

2.2 *Atmel* | *START* Project Overview In Atmel Studio

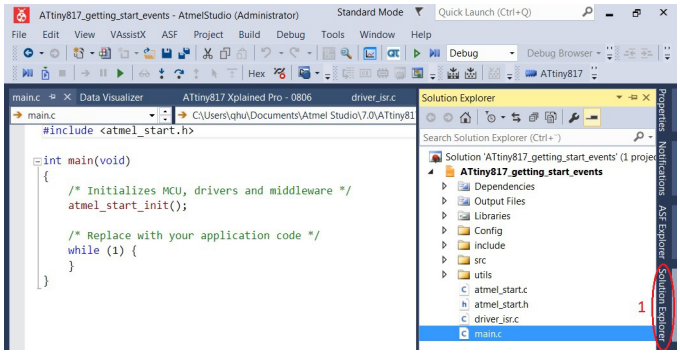
The *Atmel* | *START* project generates peripheral driver functions and files, as well as a `main()` function that initializes all drivers.

About folders and files generated by *Atmel* | *START*:

- Header files and source files for the peripheral drivers generated by START are located in the folders *src* and *include*
- The *atmel_start_pins.h* file in the *include* folder contains useful PINMUX driver functions
- The *utils* folder contain files that define some functions that can be commonly used by the drivers and the application itself
- The *atmel_start.c* file contains the `atmel_start_init()` function, which initializes the MCU, drivers, and middleware defined for the project
- The *driver_isr.c* file contains generated ISRs if interrupts have been enabled in *Atmel* | *START*

To do: Get an overview of the *Atmel* | *START* project.

Figure 2-9. Project Overview



1. The *Solution Explorer* sub-window for the generated *ATtiny817_getting_started_events* Studio project should be present, as shown in the figure above. If not, click on the *Solution Explorer* sub-window as indicated in the figure to unhide it. It can also be opened by clicking *View* followed by *Solution Explorer*. Double-click the *main.c* file in the *Solution Explorer* sub-window to open it.
2. Right-click on the `atmel_start_init()` function and then click *Goto Implementation*. Repeat the procedure for the `system_init()` function, which is called by `atmel_start_init()` and observe that `system_init()` calls the initialization functions for the generated peripheral drivers.

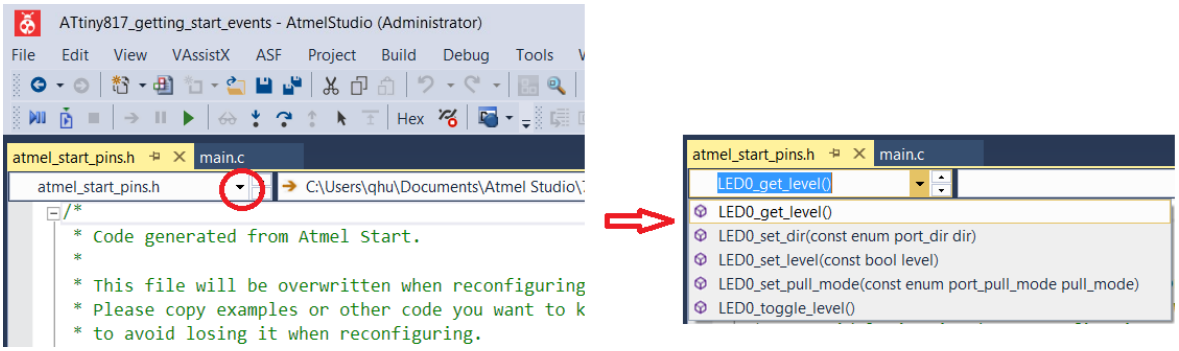
Info: The `mcu_init()` function enables the internal pull-up resistor on all pins to reduce power consumption. All driver initialization functions are called from the `system_init()` function, in addition to configuring the port pin associated with LED0 to output mode with a defined initial level.

3. Go to implementation of `CLKCTRL_init()` and observe that the default clock settings are commented out.

Info: If the non-default clock settings are selected in START, it will be reflected in `CLKCTRL_init()`.

4. Open the *atmel_start_pins.h* file in the *Solution Explorer* sub-window and then click on the down arrow, as shown in the figure below, to open the list of functions defined in this file. Observe that many useful GPIO functions have been generated.

Figure 2-10. atmel_start_pins.h Functions



Result: The *Atmel | START* project overview is completed.

2.3 Code Development

Here the RTC overflow interrupt will be set up to trigger LED0 toggling on the ATtiny817 Xplained Pro development kit.

To do: Write code to toggle LED0 in the RTC interrupt handler.

1. Open the *driver_isr.c* file in the *ATtiny817_getting_started_events* project.

   To do: Edit the *driver_isr.c* file.

2. Insert the code `LED0_toggle_level();` in the `ISR(RTC_CNT_vect)` interrupt handler, as shown below. This will, as the function name reveals, toggle the level on the pin connected to LED0 each time the RTC overflow interrupt is executed.

```
ISR(RTC_CNT_vect)
{
    /* Insert your RTC Overflow interrupt handling code */
    LED0_toggle_level();

    /* Overflow interrupt flag has to be cleared manually */
    RTC.INTFLAGS = RTC_OVF_bm;
}
```

   Info: Since the RTC clock runs at 1 kHz and the RTC period is set to 500, the RTC overflow interrupt will be executed two times per second. LED0 will consequently toggle two times per second.

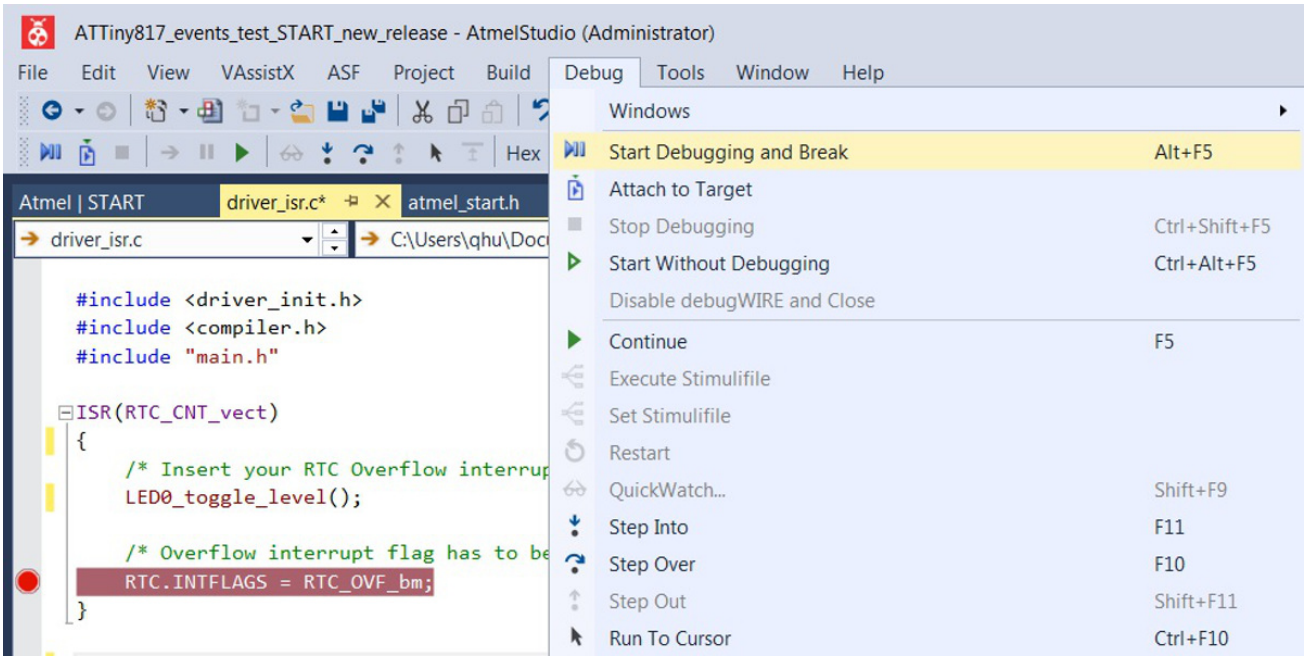
3. The *main.c* file is kept as is. Click *Build → Build Solution* in the top toolbar or press F7 in Atmel Studio to build the solution.
   Note: The build should finish successfully with no errors.

2.4 Debug the Application

To do: Debug the application where the RTC overflow interrupt triggers LED0 toggling.

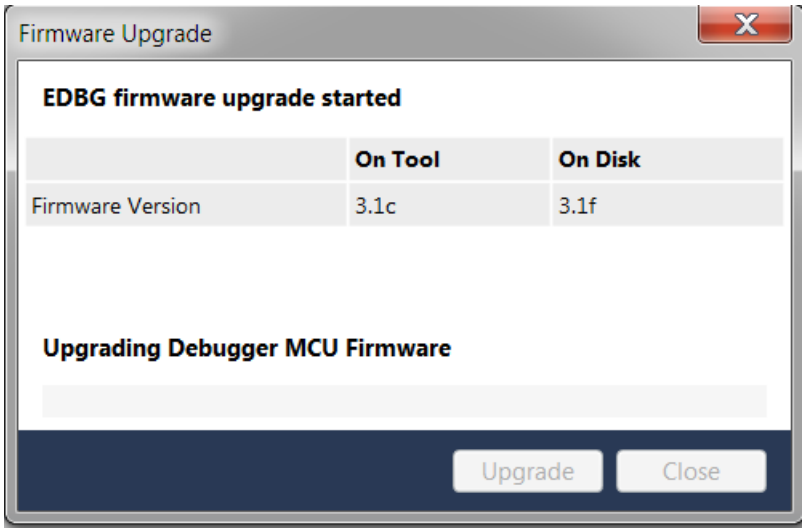
1. Power the ATtiny817 Xplained Pro kit by connecting a Micro-USB cable to the computer.
2. In Studio, click *Debug* → *Start Debugging and Break* to program the kit and start debugging, as shown below. This can also be done by pressing *Alt+F5*.

Figure 2-11. Start Debugging and Break



Info: If the embedded debugger firmware version on is lower than the one in the Atmel Studio installation, a firmware upgrade dialog will be opened.

Figure 2-12. Firmware Upgrade



Select *Upgrade* and when the progress bar is complete, select *Close*. Now start debugging by clicking *Debug* → *Start Debugging and Break* or by pressing Alt+F5.

Result: The kit is programmed with the application and a debugging session is initiated, starting with the code in *main.c*.

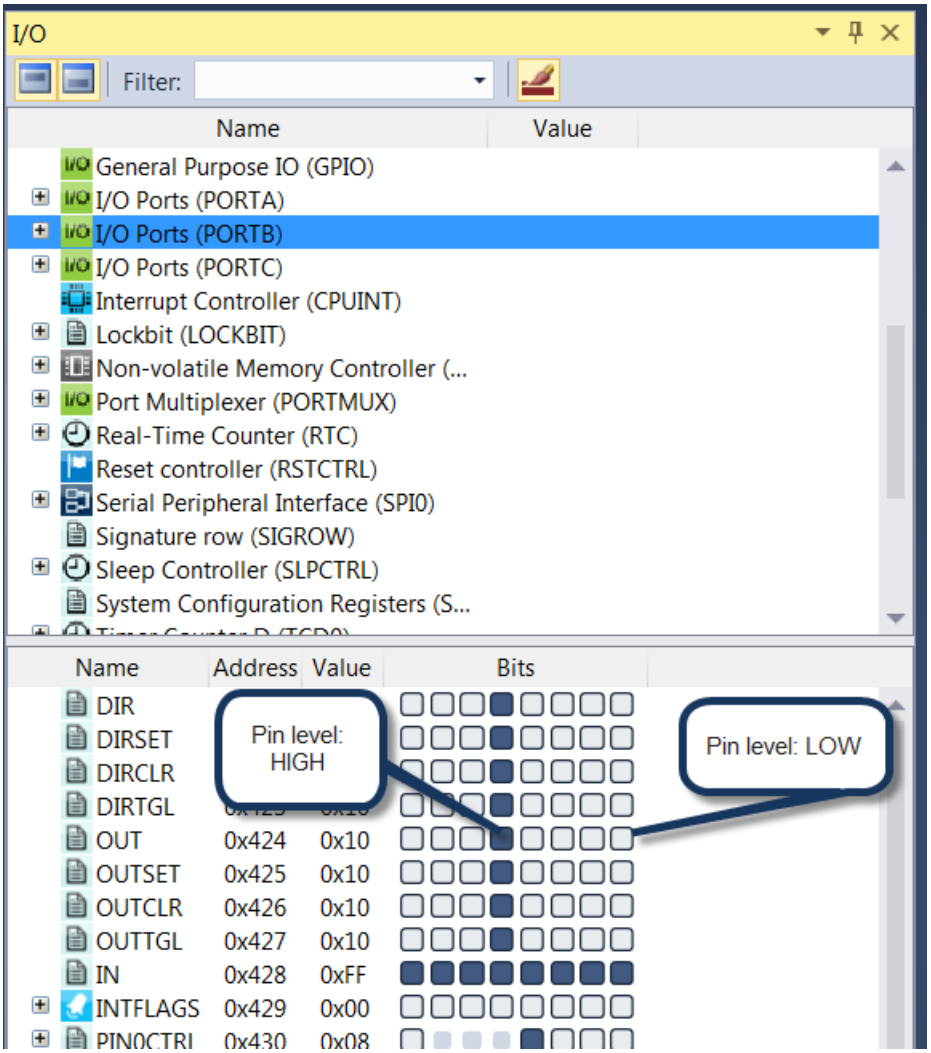
3. Click in the margin to the left of the `RTC.INTFLAGS = RTC_OVF_bm;` to place a breakpoint in the code in *driver_isr.c*, as shown above. This breakpoint is set after the `LED0_toggle_level();` function call is executed.
4. Run the application until the breakpoint is hit by clicking *Debug* → *Continue*, pressing F5, or by clicking the play button  .

Tip: The play button  is located in the top toolbar in Atmel Studio and in the Debug menu.

5. To check the status of all the PORTB pins, open the I/O View window by selecting *Debug* → *Windows* → *I/O* and click on the *I/O Ports (PORTB)* register group, as shown in Figure 2-13.

Info: In the I/O View, the status and register values of all peripherals can be inspected. Pin status is indicated for PB0 to PB7 from right to left under the *Bits* column in the *OUT* register. As shown in the image below, the level of pin PB4 (LED0) is HIGH, since a filled square corresponds to bit status 1. An empty square indicates that the bit status is 0/ LOW.

Figure 2-13. I/O View



6. Click on  to continue the execution and observe that the value of the OUT register in the PORTB register alternates between 0x00 and 0x10 each time the breakpoint is hit. This indicates that pin PB4 is being set and reset alternatively, and LED0 is toggling correspondingly.
7. Remove the breakpoint by clicking the  shown in Figure 2-11 and run the code again by pressing F5 or clicking  . Now LED0 should continuously toggle two times per second.
8. End the debugging session by clicking *Debug* → *Stop Debugging*, pressing *Ctrl+Shift+F5*, or by clicking  .

Result: An application that toggles LED0 on the RTC overflow interrupt has been successfully programmed on the ATtiny817 Xplained Pro evaluation kit.

3. Assignment 2: RTC Interrupt Triggers String Sent to USART Terminal

In this assignment it will be demonstrated how to configure the USART module and use it to send a string, as well as how to display the string "Hello World!" in the terminal embedded in the Atmel Studio Data Visualizer. An application will be developed so that the RTC overflow interrupt, when set, will trigger reading of the string from the USART and output it in the terminal.

First, *Atmel | START* will be used from within Studio to reconfigure the project from **Assignment 1**. In START, the USART component will be added to the project and then configured. The basic USART drivers will be automatically generated from START for the project in Studio. An extended USART function code is required to be added manually in Studio.

Peripherals used:

- USART
- RTC (from previous assignment)
- GPIO (PB4, from previous assignment)

Clock details:

- 3.333 MHz main clock
- 1 kHz RTC clock (from previous assignment)

3.1 USART Configuration in *Atmel | START*

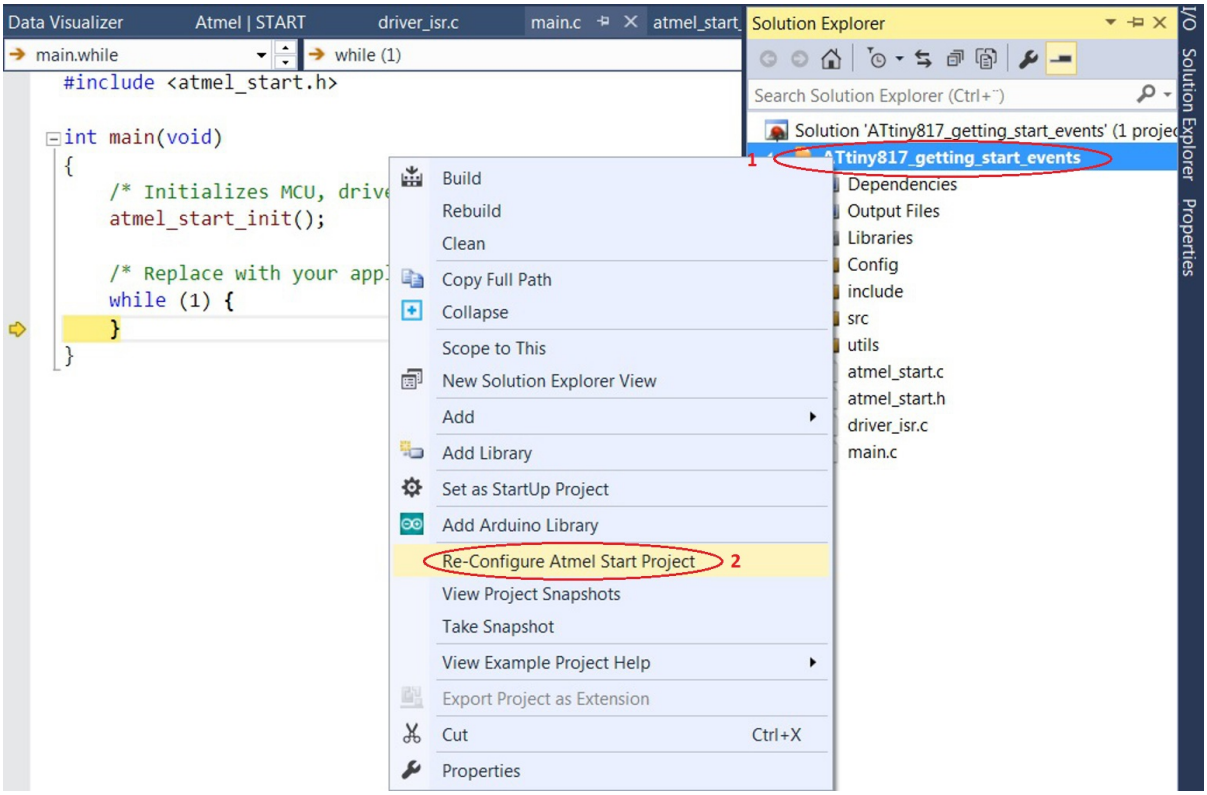
To do: Add the USART driver and configure the module in *Atmel | START* by reconfiguring the project from within Atmel Studio.

Info: The ATtiny817 Xplained Pro is equipped with the Atmel Embedded Debugger (EDBG), which identifies itself as a composite USB device with a Virtual COM Port interface when connected to a computer. The Virtual COM Port is connected to the UART on the ATtiny817 via the EDBG and provides an easy way to communicate with the target application through terminal software. It offers variable baud rate, parity, and stop bit settings. Note that the settings on the ATtiny817 must match the settings given in the terminal software.

Tip: The USART TX line on the device on ATtiny817 Xplained Pro connected to the EDBG Virtual COM Port is available on pin PB2.

1. Open the *ATtiny817_getting_started_events* project from **assignment 1**.
2. Right-click on *ATtiny817_getting_started_events*, in the *Solution Explorer* window the sub window pops up. In the sub-window, select *Re-Configure Atmel START Project*, as shown in the figure below.

Figure 3-1. Re-Configure *Atmel | START* Project

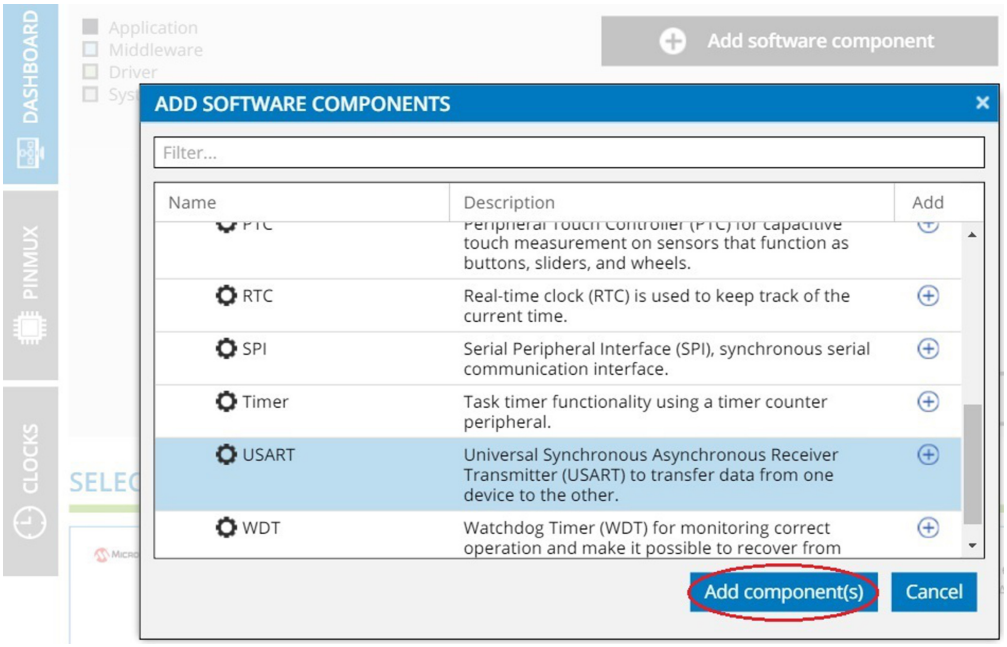


Info: *Atmel | START* will be opened in the default web browser in Studio.

To do: Add in the USART component.

3. Open the *ADD SOFTWARE COMPONENTS* window by clicking Add software component.
4. Expand the *Drivers* category and scroll down to locate the *USART* driver. Click it to select it and then click *Add Component(s)*, as shown below.

Figure 3-2. Add Software Components

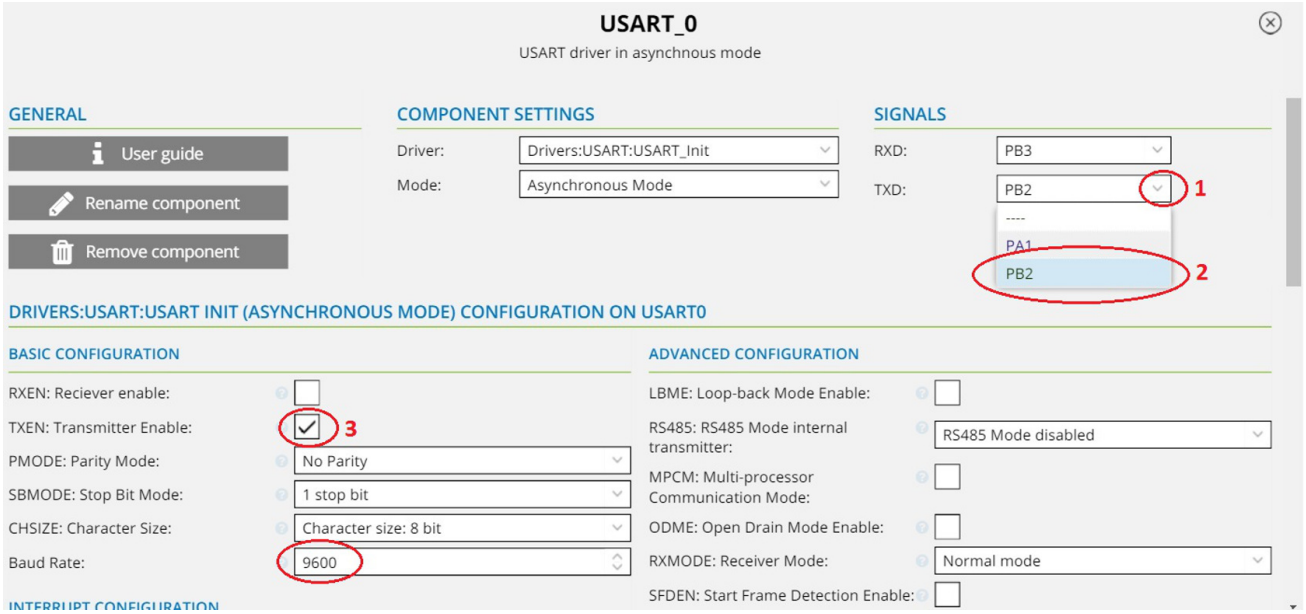


Result: The *USART* driver will now be added to the project.

To do: Configure the USART baud rate to 9600 and enable the transmitter.

5. Click the newly added *USART_0* box to open the driver configuration view.

Figure 3-3. USART Configuration

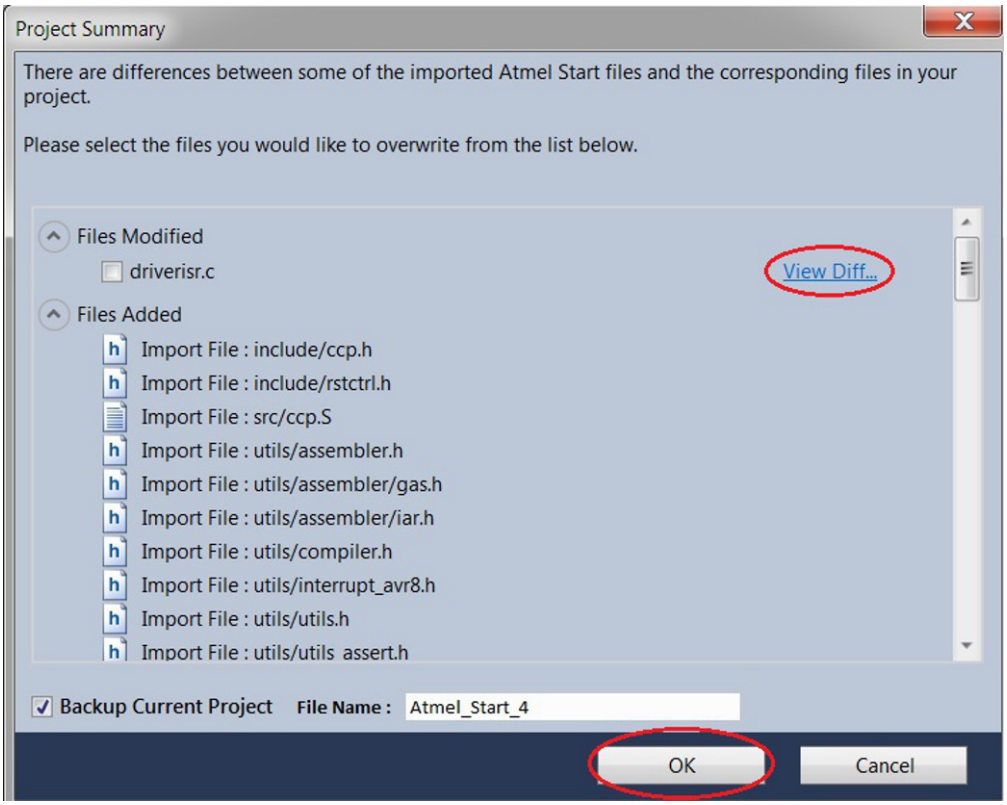


6. Perform the configuration steps in red markings as illustrated in the figure above:
   - Click the *TXD* dropdown menu under *SIGNALS*
   - Select *PB2* as TXD pin
   - Tick the *TXEN* checkbox to let the USART transmitter to be enabled in the USART initialization routine which will be generated by START
   - Observe that the default baud rate is 9600 and leave it as is

Result: The USART driver configuration is completed.

7. Click the *GENERATE PROJECT* button at the bottom of the window to regenerate the project in Atmel Studio.

Figure 3-4. Project Code Regenerated



Result: The *Project Summary* window will appear, as shown in the figure above.

Info: The *Project Summary* gives an overview of the differences between the original and the reconfigured Studio project. Clicking *View Diff...* as shown in figure above will open the external **WinMerge** tool, which shows the changes to that file. This external **WinMerge** tool is not installed within Studio by default. Here is the info on how to install it.

- Download the tool from http://downloads.sourceforge.net/winmerge/WinMerge-2.14.0-Setup.exe
- Install the tool at the default path (i.e. C:\Program Files (x86)\WinMerge) or a user-defined path
- In Atmel Studio, go to the *Tools Options → Atmel Start → File Compare* menu. In "*Path of the application used for comparing files*", enter **C:\Program Files (x86)\WinMerge \WinMergeU.exe**. In "*Command line arguments to be used for file comparison:*", enter **%original %mine /s /u**.
- Click *OK* and the *View Diff...*, as shown in Figure 3-4, should work now

8. Since we do not want to overwrite the *driverisr.c* file, there is no need to inspect this file. Leave the checkbox unticked and click *OK* as illustrated in the figure above to regenerate the project for Atmel Studio.

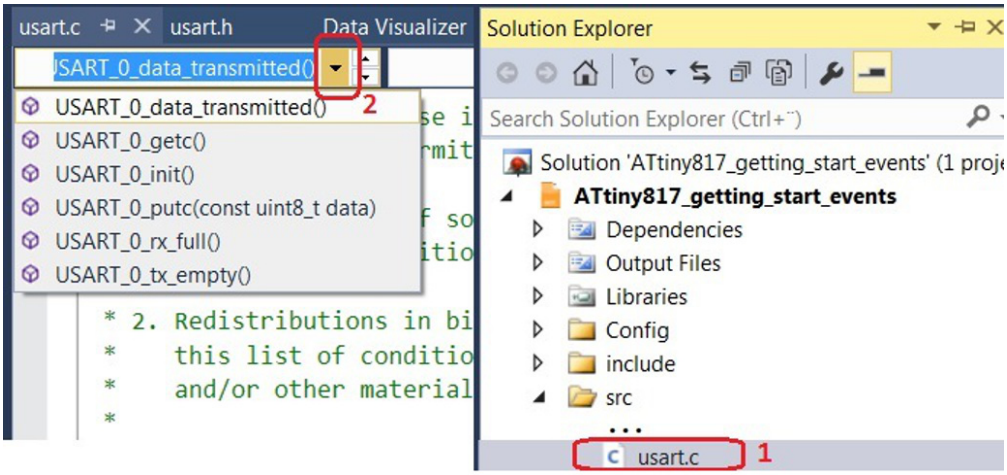
Result: The project will be regenerated in Studio including reconfiguration updates performed in *Atmel | START*.

3.2 Expand the USART Functionality

To do: In Atmel Studio, add code to the USART peripheral to send the string **"Hello World!"** and output it to terminal.

1. In Studio *Solution Explorer* window, locate the *usart.c* file under the *src* directory, as shown in Figure 3-5, and double-click to open it.
2. Expand the function list by clicking the arrow as shown in the red marking number 2. All USART driver functions are listed, which are automatically generated by *Atmel | START*. Inspect the functions and pay attention to the functionality of `USART_0_putc(const uint8_t data)` and `USART_0_tx_empty()`, which have been used in this project. Selecting a function will jump to the declaration of that function.

Figure 3-5. USART Driver Function List



3. In the *main.c* file, manually implement the function which will write a string to the USART TX buffer.
   - Declare a char array `hello` that holds `"Hello World!\n` and a `HELLO_LEN` definition that defines the string length:

     ```
     const char hello[]="Hello World!\n";
     #define HELLO_LEN 13
     ```

   - Implement a `usart_put_string()` function based on the `USART_0_putc()` and `USART_0_tx_empty()` functions, which are included in the *usart.c* file.

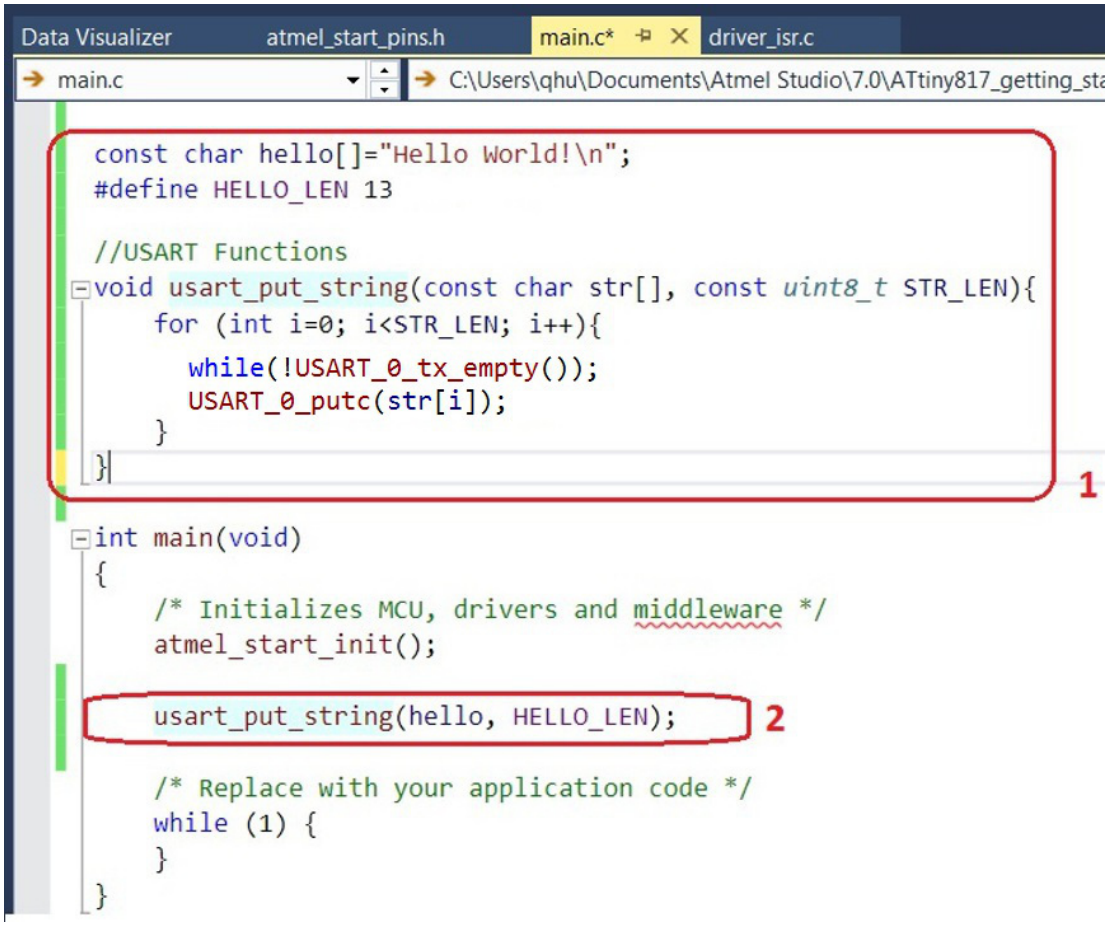
     ```
     //USART Functions
     void usart_put_string(const char str[], const uint8_t STR_LEN){
       for (int i=0; i<STR_LEN; i++){
         while(!USART_0_tx_empty());
         USART_0_putc(str[i]);
       }
     }
     ```

   - Add a call to `usart_put_string()` in the `main()` function before the `while(1)` loop:

     ```
     usart_put_string(hello, HELLO_LEN);
     ```

   - The complete code for *main.c* looks as Figure 3-6 (the code in red markings are supposed to be added in):

     Figure 3-6. Add USART Function in main.c

4. Build the solution by pressing *F7* or clicking *Build → Build Solution*.

Info: The solution should build without errors.

Result: An application that transmits the string **"Hello World!"** to USART have been implemented.

3.3 USART Output to Data Visualizer Terminal

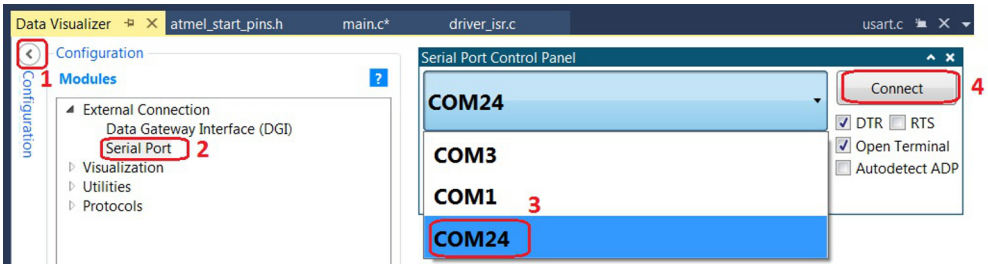
To do: Set up the Data Visualizer terminal in Studio to display the string transmitted from the evaluation kit via the EDBG Virtual COM Port.

1. In Studio 7, click *Tools → Data Visualizer* to open the Data Visualizer.
2. In the Data Visualizer window, click *Configuration* located all the way to the left, expand the *External Connection* group and double-click *Serial Port* as illustrated by the steps in the figure below.
3. Expand the drop-down list of COM ports in the *Serial Port Control Panel* and select the COM port associated with the Virtual COM Port of the EDGB on the evaluation kit, as shown in the figure below.

Tip: The EDBG Virtual COM Port number associated with the ATtiny Xplained Pro kit can be found in Windows Device Manager by clicking: *Start → Control Panel → Device Manager → Ports.*

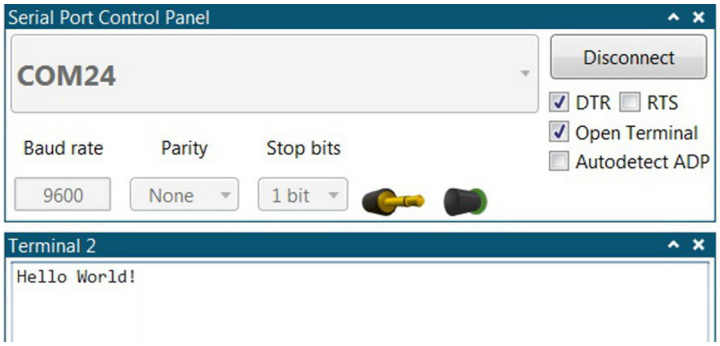
4. Leave the serial port settings at default values and click *Connect* as shown in the figure below to open the terminal.

Figure 3-7. Select USART Serial Port



5. Program the device and run the program by pressing *Ctrl+Alt+F5* or by clicking *Debug → Start Without Debugging*.
6. Inspect the terminal in the Data Visualizer tab and verify that **Hello World!** is printed as shown in the figure below.

Figure 3-8. String Printed to Terminal



Result: Transmission of **Hello World!** using USART has been verified with the Data Visualizer terminal.

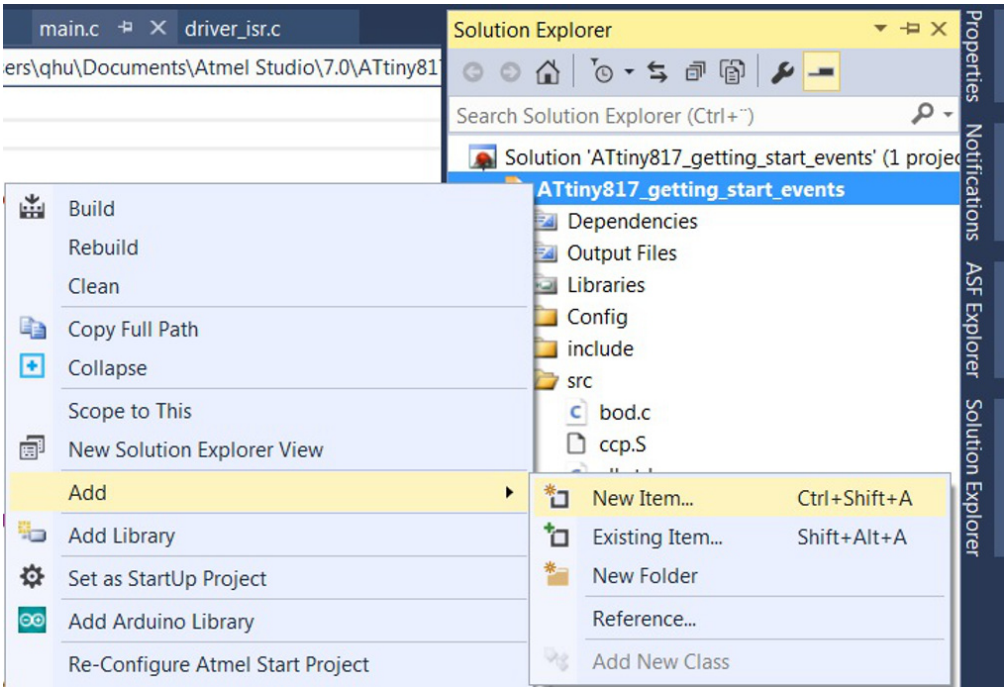
3.4 RTC Interrupt Triggers USART Transmission

To do: Update the application to enable the RTC overflow interrupt to trigger the USART transmission of **"Hello World!"** to the Data Visualizer Terminal.

To do: Add in the new *main.h* file.

1. Right-click on the project *ATtiny817_getting_start_events* in *Solution Explorer* and click *Add → New Item*, as shown below.

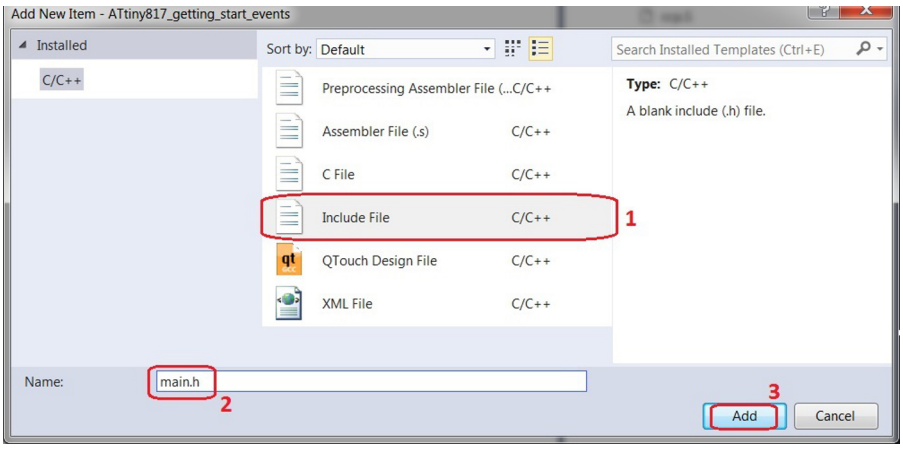
Figure 3-9. Add New Item



2. In the pop-up window, choose *Include File*, rename the file to *main.h* and click *Add*, as shown below.

Result: An empty file named *main.h* will be added to the project and listed just below the *main.c* file in *Solution Explorer*.

Figure 3-10. Add main.h File

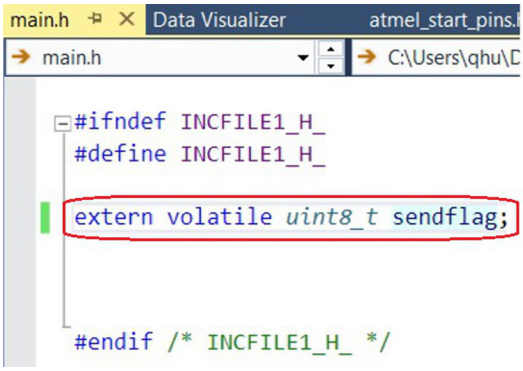


3. Locate the *main.h* file in *Solution Explorer* and open it.

To do: Update the newly created *main.h* file.

4. Add the line `extern volatile uint8_t sendflag;` in *main.h*, as shown below.

Figure 3-11. Updating main.h File



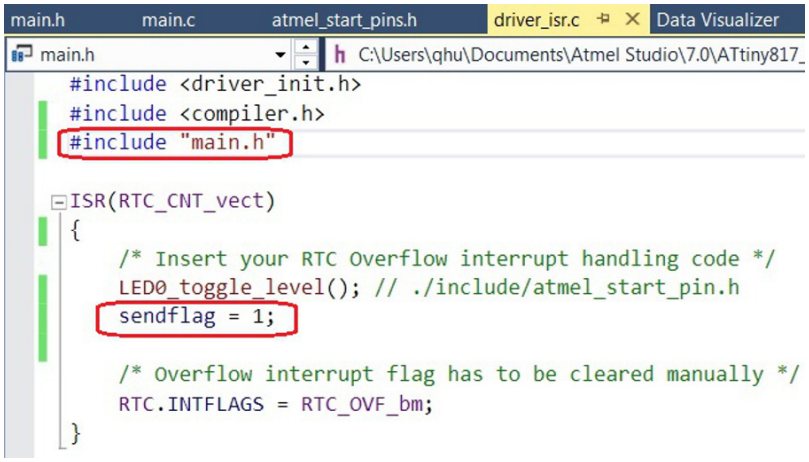
Info: The variable `sendflag` will be used by the application in *driver_isr.c* to initiate USART transmissions.

5. Open the *driver_isr.c* file from *Solution Explorer*.

To do: Update the *driver_isr.c* file.

6. Make the `sendflag` variable available in the *driver_isr.c* by including it as shown below.
7. Raise the flag by writing 1 to the `sendflag` variable in the RTC ISR, as shown below. This can serve as a notification to the CPU that the RTC overflow interrupt has been executed.

Figure 3-12. Updating driver_isr.c File



To do: Update the *main.c* file.

8. In *main.c*, include the *main.h* file and declare the `sendflag` variable as:

```
#include "main.h"
volatile uint8_t sendflag = 0;
```

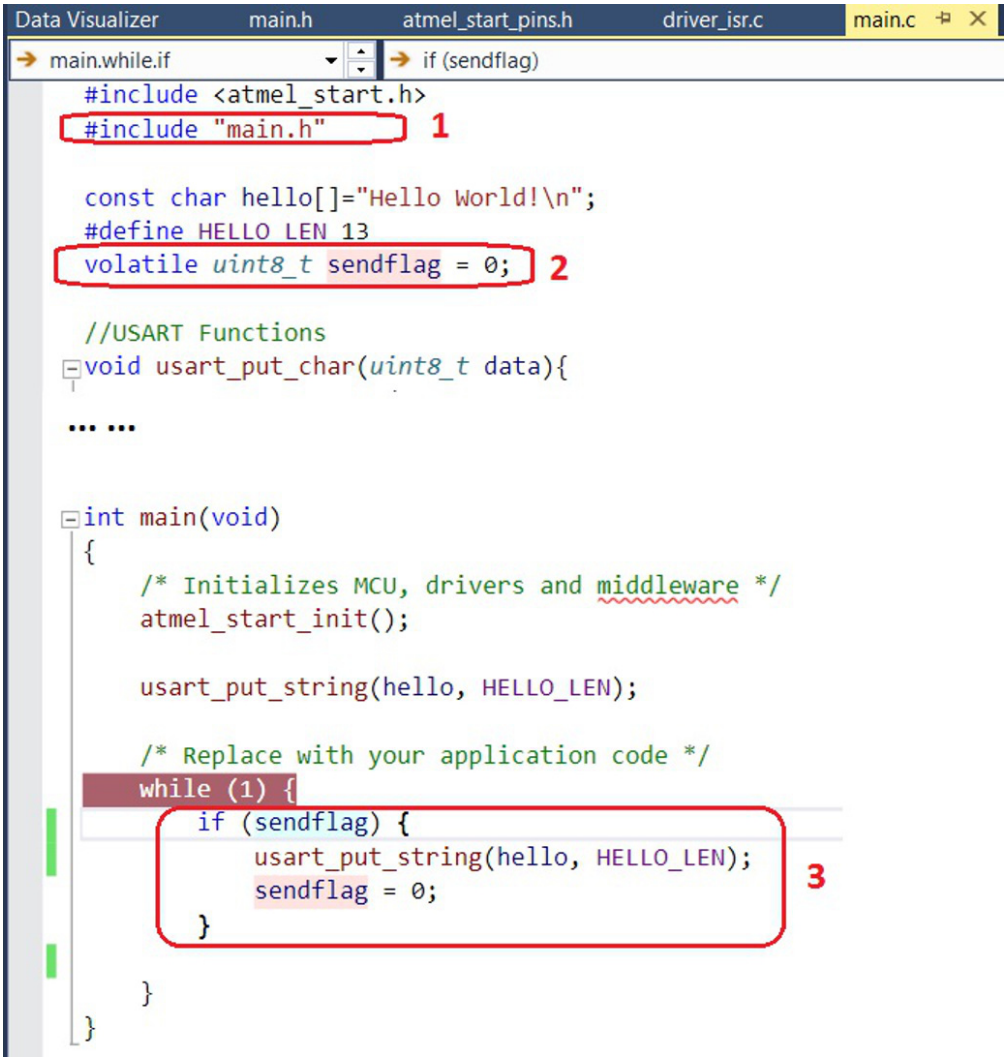
9. In *main.c*, add the following code in the `while(1)` loop of the *main()* function.

```
if (sendflag) {
  usart_put_string(hello, HELLO_LEN);
  sendflag = 0;
}
```

This piece of code will check if *sendflag* is set and then call `usart_put_string()` to print the "**Hello World!**" string to the USART terminal. The *sendflag* is set each time the RTC overflow interrupt routine defined in *driver_isr.c* is executed. After the USART transmission, *sendflag* is reset and ready for a new check.

The complete modifications in the *main.c* file are highlighted in the red markings, as shown below.

Figure 3-13. Updating main.c File



10. Press *F7* to build the solution.

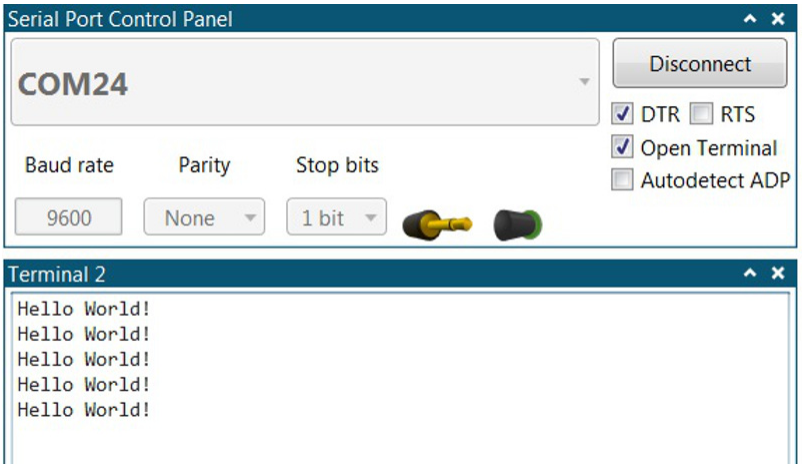
Info: The solution should build without errors.

11. Program the kit and start the application by pressing *Ctrl+Alt+F5* or by clicking *Debug → Start Without Debugging*.

Info: The application should start executing on the kit.

12. Open the Data Visualizer terminal and verify that **Hello World!** is printed approximately two times per second, as shown below.

Figure 3-14. RTC Interrupt Continuously Triggers String Print to Terminal



Result: The RTC overflow interrupt continuously triggers USART transmission of the **Hello World!** string to the Data Visualizer terminal two times per second.

4. Assignment 3: ADC ISRRDY Interrupt Triggers ADC Data Output to USART Terminal

The ATtiny817 has an ADC module, which will be configured. The application code will also be updated. A potentiometer will be connected to the ADC input pin to observe the added ADC functionality. ADC data will be sent to USART and observed in the Data Visualizer terminal window.

This is based on the project **Assignment 2: RTC Triggers Print to USART Terminal**. Here, an application will be developed demonstrating the ADC functionality with ADC data output to the USART terminal. First use *Atmel | START* to add in the ADC drivers and then get the module configured. Now two ADC functions must manually be added to Atmel Studio. The RTC overflow interrupt will trigger the start of an ADC conversion and the ADC data ready interrupt will trigger sending the ADC data to the USART terminal.

Peripherals used:

- ADC
- USART (from previous assignment)
- RTC (from previous assignment)
- GPIO (PB4, from previous assignment)

Clock details:

- 3.333 MHz ADC clock
- 3.333 MHz main clock (from previous assignment)
- 1 kHz RTC clock (from previous assignment)

4.1 Add ADC Driver In *Atmel | START*

To do: Add ADC driver to the project, using *Atmel | START*.

1. Open the *ATtiny817_getting_started_events* project from **Assignment 2**.
2. Select the project name, right-click in the *Solution Explorer* window, and select *Re-Configure Atmel START Project*.

   Info: *Atmel | START* will be opened in the default web browser.

3. Click Add software component in the *Atmel | START* window and expand *Drivers* from the *ADD SOFTWARE COMPONENT* window.
4. Add the ADC to the project by selecting *ADC* and clicking Add component(s).

Result: ADC driver is added to *ATtiny817_getting_started_events*.

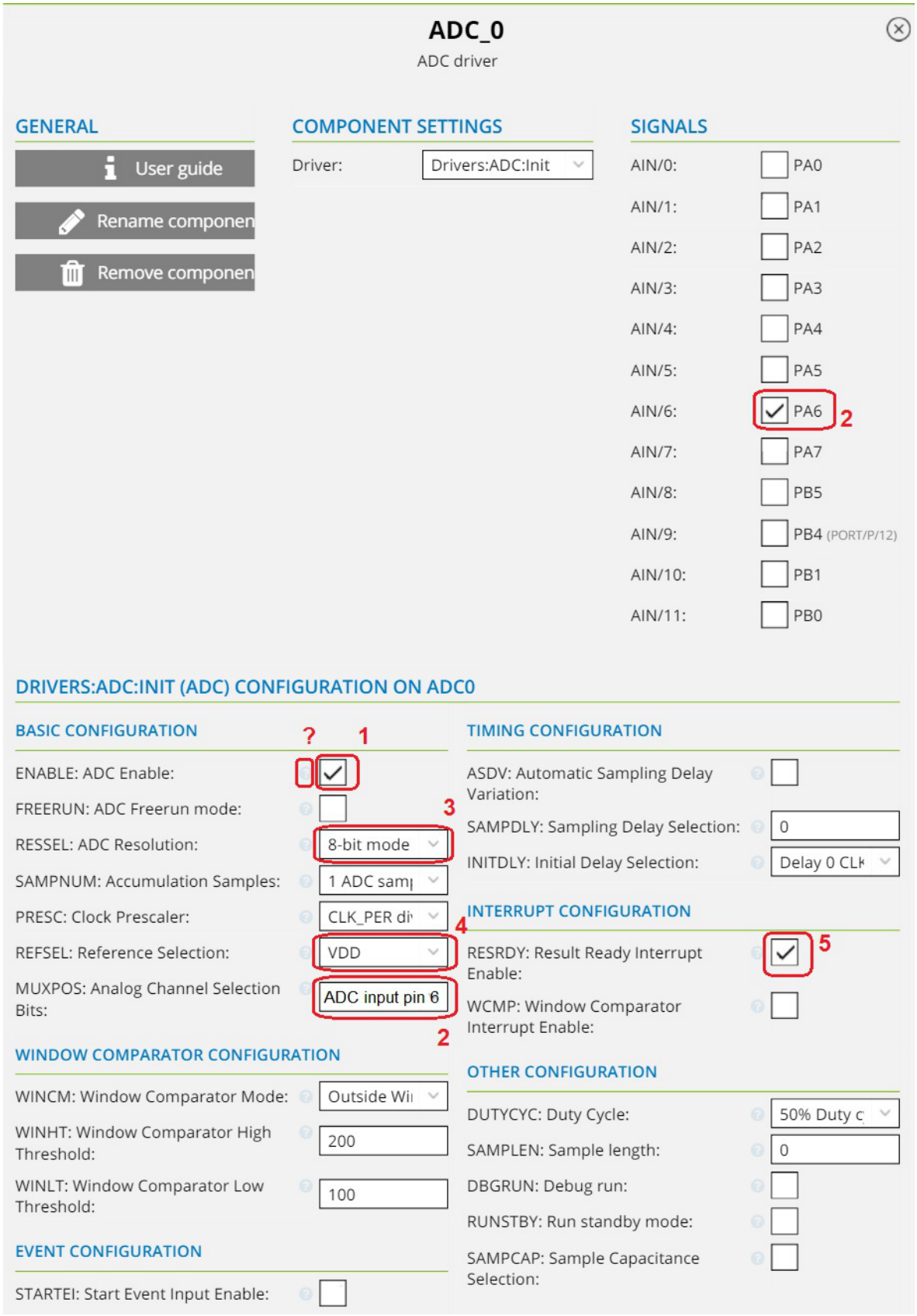
4.2 Configure ADC in *Atmel | START*

Once the ADC module has been added to *ATtiny817_getting_started_events* the ADC driver will be configured here.

To do: Configure the ADC Module.

1. Click *ADC_0* in the *Atmel | START* window.
2. Perform the configuration steps, marked with red, as depicted in Figure 4-1.
   - Enable the ADC by checking the *ENABLE: ADC Enable* checkbox
   - Enable PA6 as an analog input for the ADC by checking the *PA6* checkbox. Then select *ADC input pin 6* from the *MUXPOS:Analog Channel Selection Bits* dropdown menu to actually select the enabled PA6 as ADC input pin.
   - Configure the ADC resolution to be 8-bit by selecting *8-bit mode* from the *RESSEL:ADC Resolution* dropdown menu
   - Configure the voltage reference, *REFSEL*, to be *VDD* from the dropdown menu
   - Enable the Result Ready interrupt by checking the *RESRDY:Result Ready Interrupt Enable* checkbox

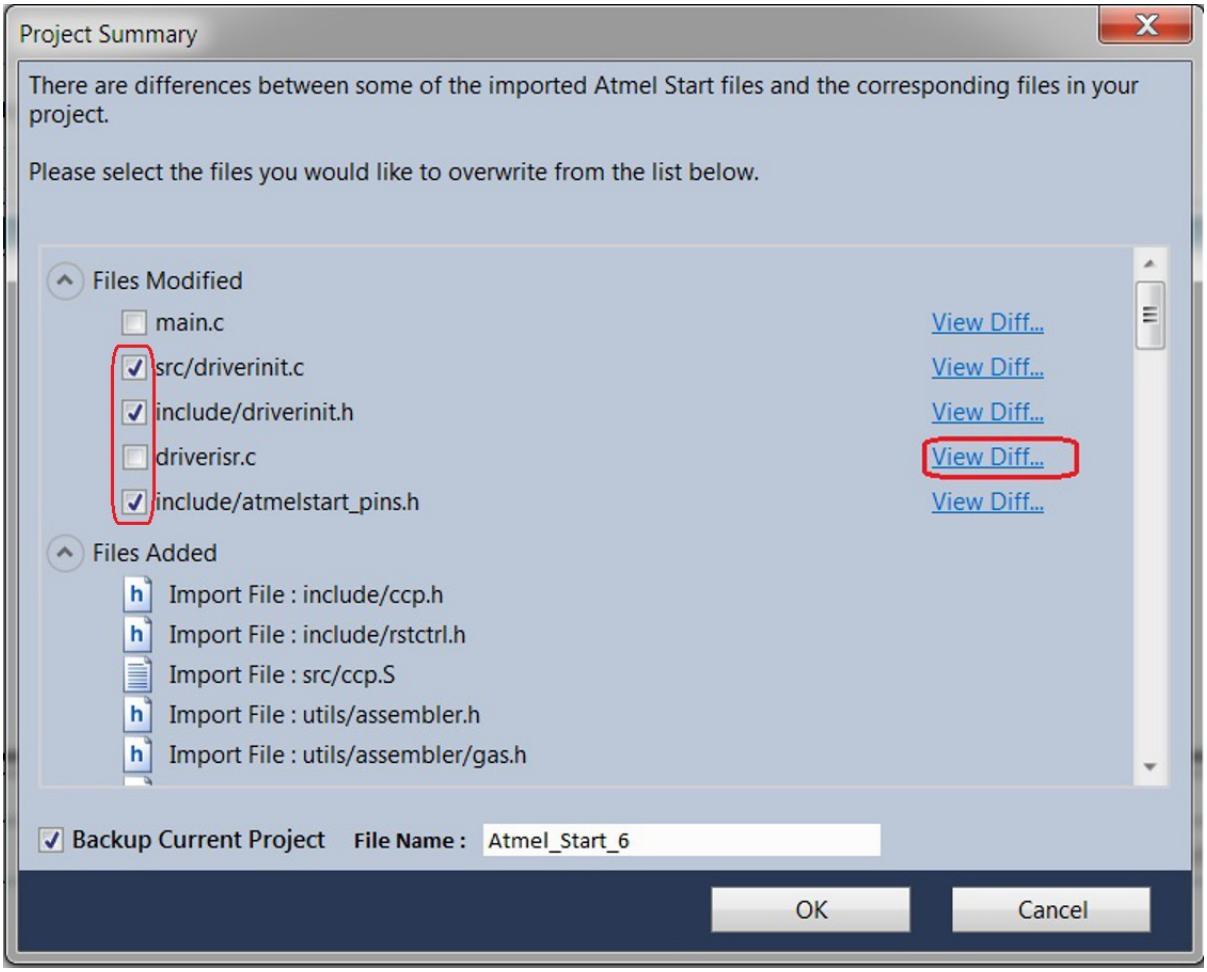
Figure 4-1. ADC Configuration in Atmel | START



Note: Clicking the question mark next to each configuration will provide the data sheet description of the individual bit setting.

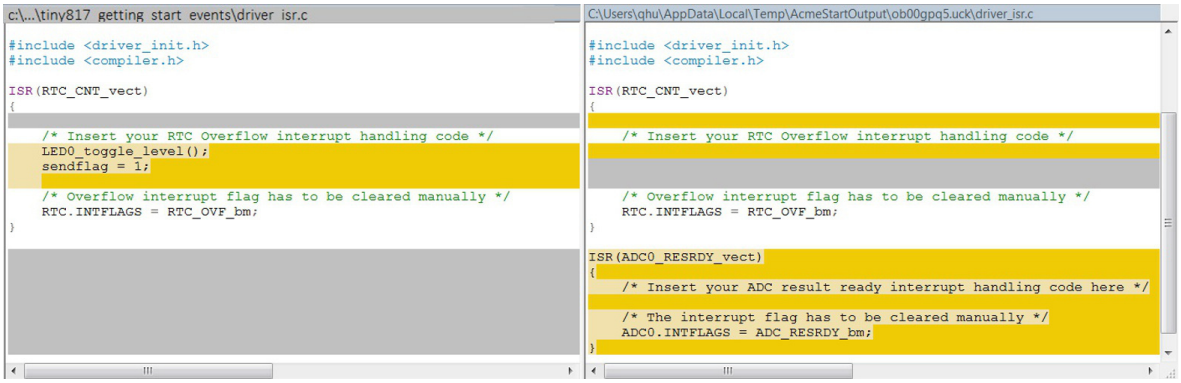
3. Generate the project by clicking GENERATE PROJECT. The following project summary window pops up:

Figure 4-2. Summary of Project Code Generated by *Atmel | START*



4. Click *View Diff* on the *driver_isr.c* file as in red marking in the Figure 4-2. A window pops up showing the code difference between **Assignment 2** version and the *Atmel | START* regenerated version as:

Figure 4-3. driver_isr.c View Difference After Atmel START Modification



As shown, the *ISR(ADC0_RESRDY_vect)* function has been added in while the two lines, *LED0_toggle_level();* and *sendflag = 1;*, are missing in the *Atmel | START* generated version of code. The regenerated version needs to be updated including the code that existed in the **Assignment 2** version.
By clicking on *View Diff* on other three files, one can see that these files can simply be overwritten without modification required.

5. Now, choose to regenerate the project in *Atmel | START* by selecting the three files as circled in red marking in Figure 4-2. These three selected files do not require modification and will be overwritten by the *Atmel | START* generated version of codes when clicking the *OK* button at the bottom in Figure 4-3. The *main.c* and *driver_isr.c* files needs to be updated manually as *driver_isr.c* is shown in Figure 4-3.

Result: The *Atmel | START* project has been regenerated in Atmel Studio, including the newly added ADC drivers.

4.3 Add ADC Functionality to Application Code

Once the ADC module has been added and reconfigured using *Atmel | START*, the application code needs to be updated using Atmel Studio.

To do: Update the code in *ATtiny817_getting_started_events* to make use of the ADC driver just added through *Atmel | START*. Specifically update the *main.c, main.h* and *driver_isr.c* files.

1. Update the code in *main.c* by performing the steps below. The complete code after updating is shown in Figure 4-4.
   - Add a variable declaration for the ADC result, `adc_result`:

     ```
     volatile uint8_t adc_result = 0;
     ```

   - Add the function for starting an ADC conversion, `ADC_start_conversion()` and the function for reading out the ADC result, `ADC_get_result()`.

     ```
     //ADC Functions
     void ADC_start_conversion(){
         ADC0.COMMAND = ADC_STCONV_bm; //Set start conversion enable mask bit
     }
     void ADC_get_result(){
         adc_result = ADC0.RESL;
     }
     ```

   - In the `while`-loop of the `main` function, comment out the code which sent the `hello` string from **assignment 2**, and add code for sending `adc_result` via the USART instead.

     ```
     //usart_put_string(hello, HELLO_LEN);
     USART_0_putc(adc_result);
     ```

Figure 4-4. Updates to *main.c* in Atmel Studio

```
#include <atmel_start.h>
#include "main.h"

const char hello[]="Hello World!\n";
#define HELLO_LEN 13
volatile uint8_t sendflag = 0;
volatile uint8_t adc_result = 0;                    1

//USART Functions
void usart_put_string(const char str[], const uint8_t STR_LEN){
    for (int i=0; i<STR_LEN; i++){
        while(!USART_0_tx_empty());
        USART_0_putc(str[i]);
    }
}
                                                    2
//ADC Functions
void ADC_start_conversion(){
    ADC0.COMMAND = ADC_STCONV_bm; //Set start conversion enable mask bit
}

void ADC_get_result(){
    adc_result = ADC0.RESL;
}

int main(void)
{
    /* Initializes MCU, drivers and middleware */
    atmel_start_init();

    usart_put_string(hello, HELLO_LEN);

    /* Replace with your application code */
    while (1) {
        if (sendflag) {                             3
            //usart_put_string(hello, HELLO_LEN);
            USART_0_putc(adc_result);
            sendflag = 0;
        }

    }
}
```

2. Add a global declaration of the `adc_result` variable, two functions `ADC_start_conversion()` and `ADC_get_result()` in *main.h*, as shown in Figure 4-5.

Figure 4-5. Updates to *main.h* in Atmel Studio

```
#ifndef INCFILE1_H_
#define INCFILE1_H_

extern volatile uint8_t sendflag;
volatile uint8_t sendflag;
volatile uint8_t adc_result;

void ADC_start_conversion();
void ADC_get_result();

#endif /* INCFILE1_H_ */
```

3. Update the code in *driver_isr.c* by following the steps shown in Figure 4-6.
   - Add in *#include "main.h"* in the header
   - Make the application code trigger an ADC conversion each time the RTC overflows by adding `ADC_start_conversion()` to the RTC interrupt handler, `ISR(RTC_CNT_vect)`.

```
ADC_start_conversion();
```

- Make the application code trigger a read-out of the ADC result for each ADC conversion by adding `ADC_get_result()` to the ADC interrupt handler, `ISR(ADC0_RESRDY_vect)`.

```
ADC_get_result();
```

- Ensure that the new ADC result is sent to the terminal window using the USART by moving `sendflag` from the RTC interrupt handler to the ADC interrupt handler.

```
sendflag = 1;
```

Figure 4-6. Updates to *driver_isr.c* in Atmel Studio

```
#include <driver_init.h>
#include <compiler.h>
#include "main.h"                          1

ISR(RTC_CNT_vect)
{
    /* Insert your RTC Overflow interrupt handling code */
    LED0_toggle_level();
    //sendflag = 1;                        4
    ADC_start_conversion();                2

    /* Overflow interrupt flag has to be cleared manually */
    RTC.INTFLAGS = RTC_OVF_bm;
}

ISR(ADC0_RESRDY_vect)
{
    /* Insert your ADC result ready interrupt handling code here */
    ADC_get_result();                      3
    sendflag = 1;                          4

    /* The interrupt flag has to be cleared manually */
    ADC0.INTFLAGS = ADC_RESRDY_bm;
}
```

4. Build the project by clicking  or *F7*. Make sure there are no compiler errors or warnings.

Result: The ADC functionality code is done.

4.4 Connect Potentiometer to ADC

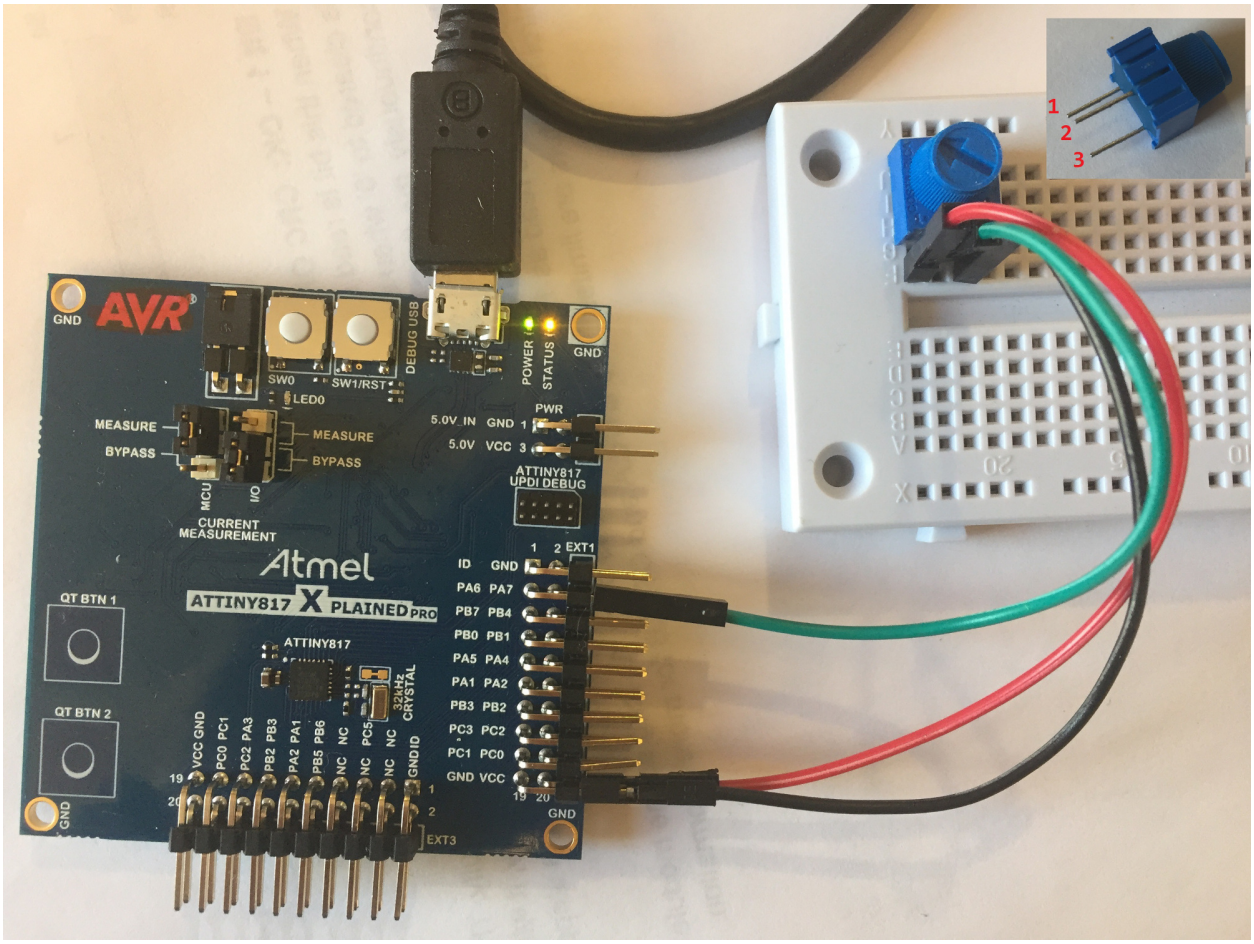
A potentiometer, also called a potmeter, is a three-terminal resistor with a sliding or rotating contact that provides an adjustable voltage divider. As shown in the upper left corner of the figure below, the potmeter has three pins marked in red as *1*, *2*, and *3* respectively.

To do: Connect a potentiometer to ATtiny817 Xplained Pro by using a breadboard and three male-to-female cables.

1. Place the potmeter in the breadboard and notice how the pins corresponds with the letters/numbers on the breadboard.
2. Connect the pins of the potmeter to ATtiny817 Xplained Pro by using male-to-female cables:
   - Pin 1 to *VCC*
   - Pin 2 to *PA6*

– Pin 3 to *GND*

Figure 4-7. Potentiometer Connected to ATtiny817 Xplained Pro



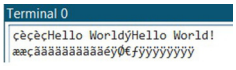
4.5 Observe ADC Functionality

Now, the ADC driver has been added to the project, the application code has been updated to make use of the ADC, and the potentiometer has been connected to the ADC input pin. It is time to observe the ADC functionality, using the Data Visualizer in Atmel Studio.

Observe ADC functionality using Data Visualizer.

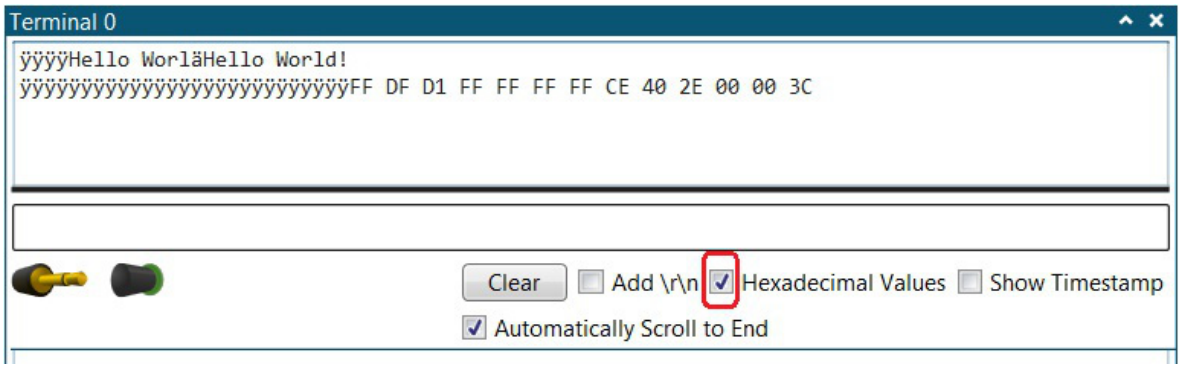
1. Program the code by selecting *Debug* → *Start Without Debugging* or clicking  on the top of menu bar.
2. Open the Data Visualizer by clicking the *Data Visualizer* tab. The output to the USART terminal should look as shown in Figure 4-8.

   Figure 4-8. Output from USART Terminal Window

   Note: The output in the USART terminal window is currently in ASCII format, which does not give the best readability when it comes to observing how turning the potmeter knob affects the output.
3. Select the output format to be in hexadecimal values by checking the *Hexadecimal Values* check-box, as depicted in red marking in Figure 4-9.

Figure 4-9. Output in Hex Format from USART Terminal Window



Result: By rotating the potentiometer knob, the voltage input to ADC input pin, PA6, is changing. This can be observed as the ADC result being sent to USART terminal window is changing as one rotates the knob.

Note: When rotating the potentiometer knob, the ADC result printed in the USART terminal window is in the range of 0x00-0xFF. This is expected as the ADC resolution has been configured to be 8 bits.

5. Assignment 4: ADC WCMP Interrupt Triggers ADC Data Print to Data Visualizer

In this assignment an application will be developed that will configure ADC Window Comparison (WCMP) interrupt triggering ADC data print to USART terminal and graph in Data Visualizer.

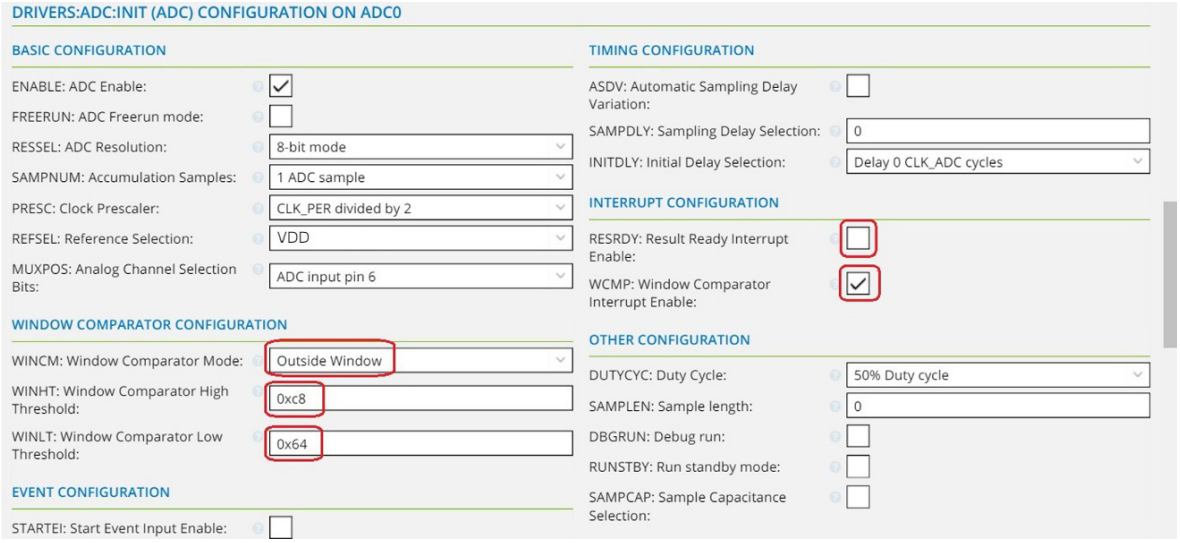
The ADC window comparison is a feature which allows the user to further filter the ADC output in their predefined way. It is enabled when the window comparator interrupt enable (INTCTRL.WCOMP) bit is set. There are four window mode options: output ADC results only when they are below window, above window, within window, or outside window. The mode is configured by writing the CTRLE.WINCM field. WINLT and/or WINHT registers also need to be configured to set the window comparator to low and/or high threshold. The ADC interrupt will then be triggered when the ADC output is available and it also satisfies the configured mode.

First, *Atmel | START* will be used within Atmel Studio to reconfigure the project from **Assignment 3**. In *Atmel | START*, the ADC driver will be updated using the WCMP interrupt option instead of a result ready (RESRDY) interrupt.

5.1 ADC Re-Configuration in Atmel START

1. Open the *ATtiny817_getting_started_events* project from **Assignment 3**.
2. Right-click on the project name *ATtiny817_getting_started_events* in *Solution Explorer* and then right-click *Re-Configure Atmel Start Project*.
3. In the **"Atmel | START"** window, click the *ADC* component.
4. Scroll down and reconfigure ADC as shown in Figure 5-1. Reconfiguration is shown with red markings.

Figure 5-1. ADC Reconfiguration in START

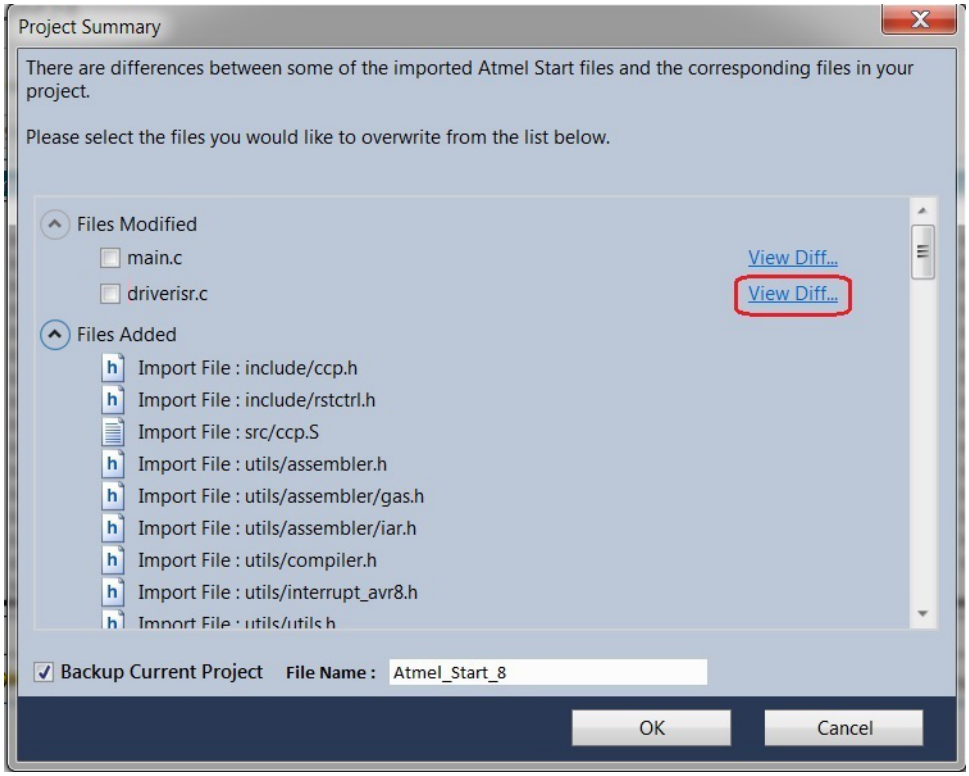


5. Take a look at the configuration steps described below:
   - Unselect the checkbox for the *RESRDY: Result Ready Interrupt Enable* field
   - Select the checkbox for the *WCMP: Window Comparator Interrupt Enable* field as it now replaces the just unselected RESRDY interrupt
   - Select *Outside Window* from the *WINCM: Window Comparator Mode* dropdown menu
   - Enter 200 (0xc8 hexadecimal value) in the *WINHT: Window Comparator High Threshold* field.

– Enter 100 (0x64 hexadecimal value) in the *WINLT: Window Comparator Low Threshold* field.

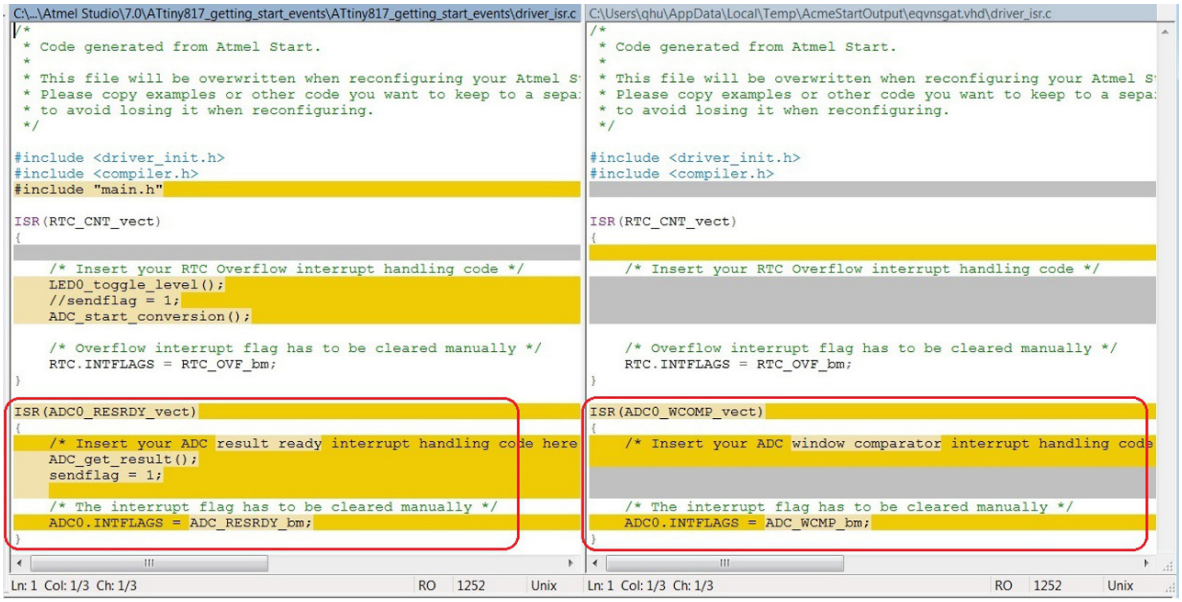
6. Click the *GENERATE PROJECT* button at the bottom of the window in order to regenerate the project in Atmel Studio. The project summary window pops up as shown in Figure 5-2.

Figure 5-2. Project Code Regenerated



7. For the *driver_isr.c* file, click on *View Diff* in red marking to see the difference before and after the *Atmel | START* regeneration, as shown below.

Figure 5-3. Driver_isr.c View Diff



8. *main.c* and *driver_isr.c* are not checked as in Figure 5-2 and need to be manually modified afterward.

Click *OK* to regenerate the project.

Result: The *Atmel | START* project has been regenerated in Atmel Studio, including the newly updated ADC drivers.

5.2 Update Application in Atmel Studio and Output ADC Data to Data Visualizer Graph

To do: Update the code in the *driver_isr.c* file.

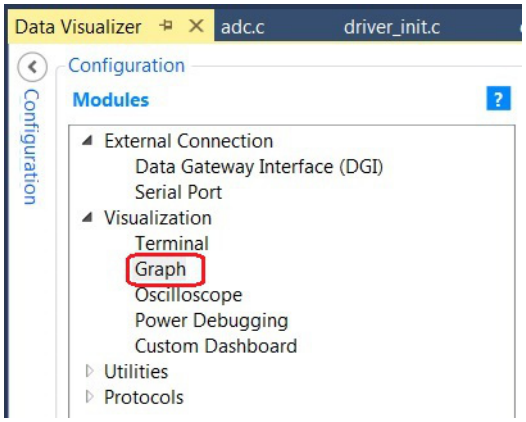
1. In Atmel Studio open the *driver_isr.c* file and replace the *ISR(ADC0_RESRDY_vect)* function with the *ISR(ADC0_WCOMP_vect)* function as shown below in red marking.

Figure 5-4. Driver_isr.c Updates

```
#include <driver_init.h>
#include <compiler.h>
#include "main.h"

ISR(RTC_CNT_vect)
{
    /* Insert your RTC Overflow interrupt handling code */
    LED0_toggle_level();
    ADC_start_conversion();

    /* Overflow interrupt flag has to be cleared manually */
    RTC.INTFLAGS = RTC_OVF_bm;
}

ISR(ADC0_WCOMP_vect)
{
    /* Insert your ADC window comparator interrupt handling code here */
    ADC_get_result();
    sendflag = 1;

    /* The interrupt flag has to be cleared manually */
    ADC0.INTFLAGS = ADC_WCMP_bm;
}
```

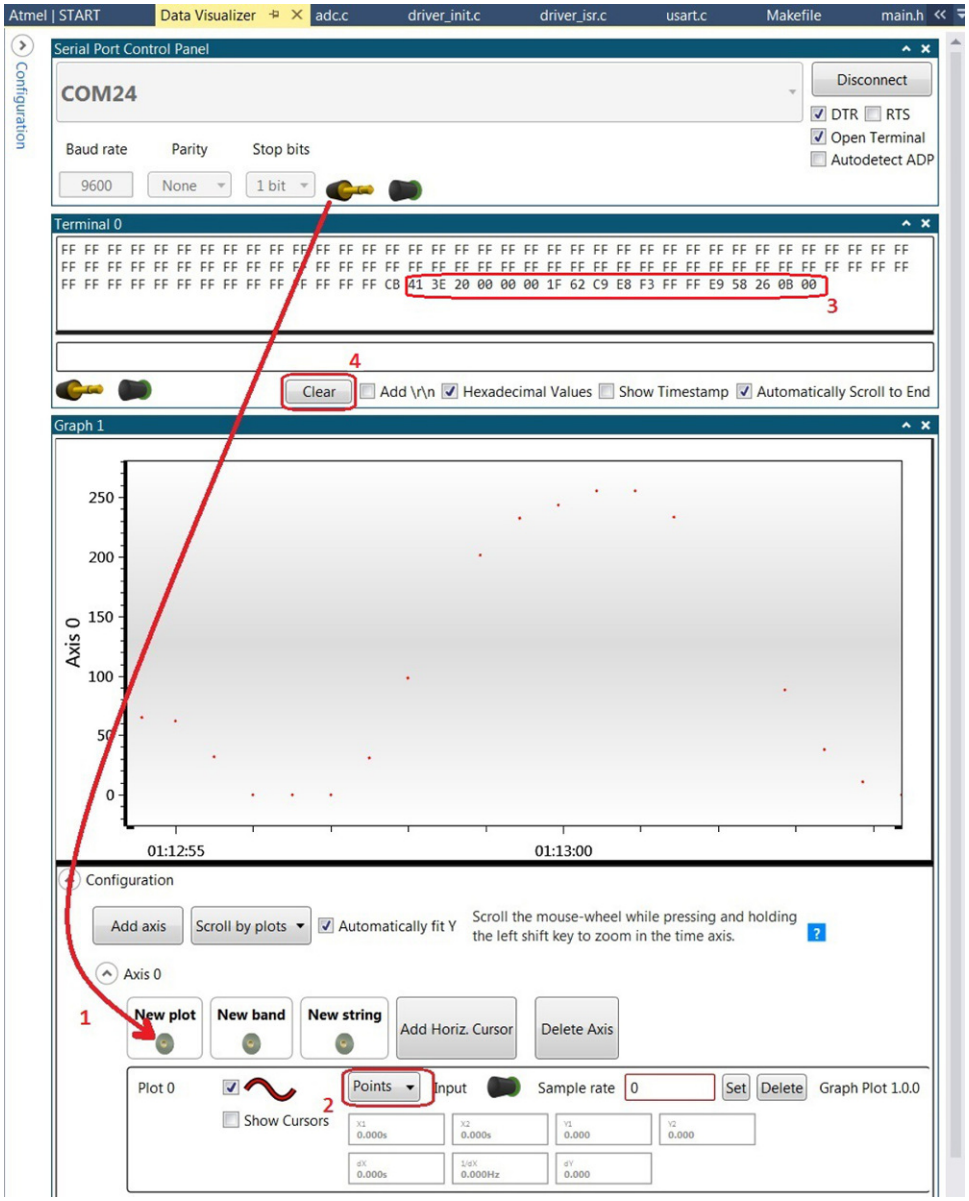
2. Click the *Data Visualizer* tab and expand the *Configuration* tab on the left side of the window. Click *Visualization* and double-click *Graph*, as shown below.

Figure 5-5. Data Visualizer Graph Selection



3. As shown in Figure 5-6 (red marking number 1), drag the terminal input from *Serial Port Control Panel* to the *New plot* in the Graph window to connect the terminal input to the graph. Then select input as *Points* (red marking number 2).

Figure 5-6. Data Visualizer Graph Data Output



4. Program the code by selecting *Debug → Start Without Debugging*.
5. Rotate the potmeter and observe the plotted ADC data in the *Graph*.
6. Observe that the plotted points for the ADC value matches the ADC value printed in the *Terminal* (red marking number 3 in Figure 5-6). When the ADC value is within the 100-200 range, points are not plotted in the *Graph*. Observe that the plotted values in the graph in the Figure 5-6 matches the ADC values printed in the *terminal* outlined by red marking number 3.

Result: The Data Visualizer Graph shows the plotted ADC data.

6. Assignment 5: RTC Interrupt Replaced by Event System

In this assignment the RTC overflow event signal, instead of the RTC overflow interrupt, will be used to trigger an ADC conversion.

The Event System (EVSYS) enables direct peripheral-to-peripheral signaling. It allows a change in one peripheral (the Event Generator) to trigger actions in the other peripherals (the Event Users) through Event channels without using the CPU. A channel path can either be asynchronous or synchronous to the main clock.

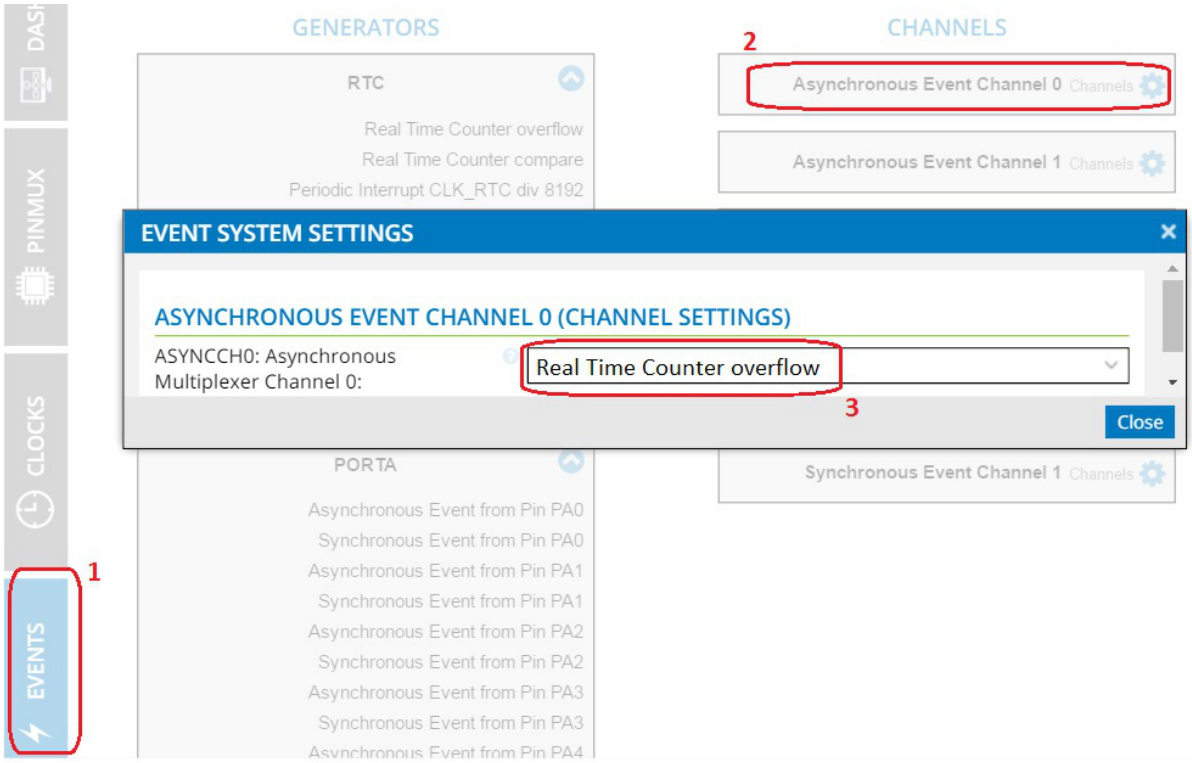
Here the RTC overflow event signal will be used to trigger an ADC conversion.

6.1 Event System Configuration in Atmel START

To do: Configure Event system in *Atmel* | *START*.

1. Open project *ATtiny817_getting_started_events* **assignment 4**.
2. Right-click on *ATtiny817_getting_started_events* in the *Solution Explore* window, and select *Re-Configure Atmel Start Project*.
3. In the *Atmel* | *START* window, click the *ADD SOFTWARE COMPONENT* box and then expand *Drivers* from the *ADD SOFTWARE COMPONENT* window**.**
4. Search for the *Events* driver, select it and then click the *Add Component(s)*. (Refer to Figure 2-4 for adding component of RTC as an example.)
   Note: The *Events* driver will be added to the project. Now the EVENT_SYSTEM peripheral configuration needs to be done.
5. Configure Asynchronous Event Channel 0 as shown below (see the red markings numbered 1 to 3).

Figure 6-1. Event Source Selection

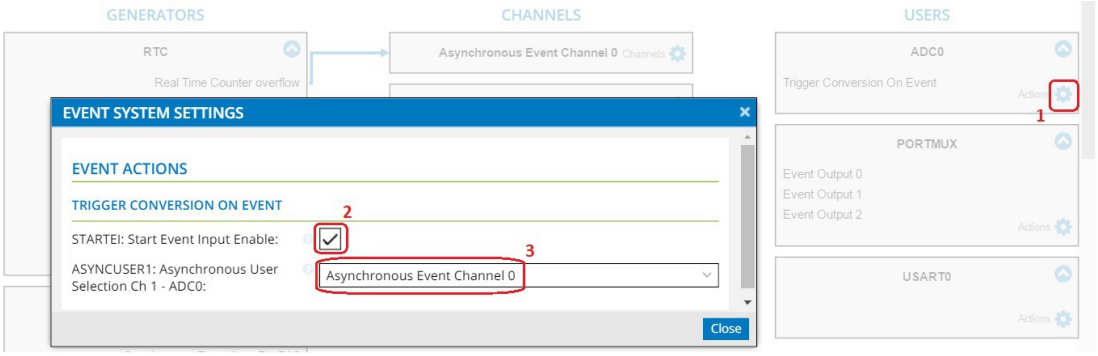


Execute the configuration steps described below:

- Select the "EVENTS" icon on the left side of the window (red marking number 1).
- Select *Asynchronous Event Channel 0* (red marking number 2) and the *EVENT SYSYEM SETTINGS* window pops up.
- In the popped up *EVENT SYSTEM SETTINGS* window, scroll down and select the *Real Time Counter overflow* option, and close the window.

6. Now, configure the Event user as shown in Figure 6-2 below (see the red markings numbered 1 to 3):
   - Click the settings dialog under *USERS → ADC0* (red marking number 1)
   - In the popped up *EVENT SYSTEM SETTINGS* window, select the checkbox of *Start Event Input Enable* (red marking number 2)
   - Select *Asynchronous Event Channel 0* (red marking number 3) as ASYNCUSER1 and close the window

Figure 6-2. Event Channel Selection

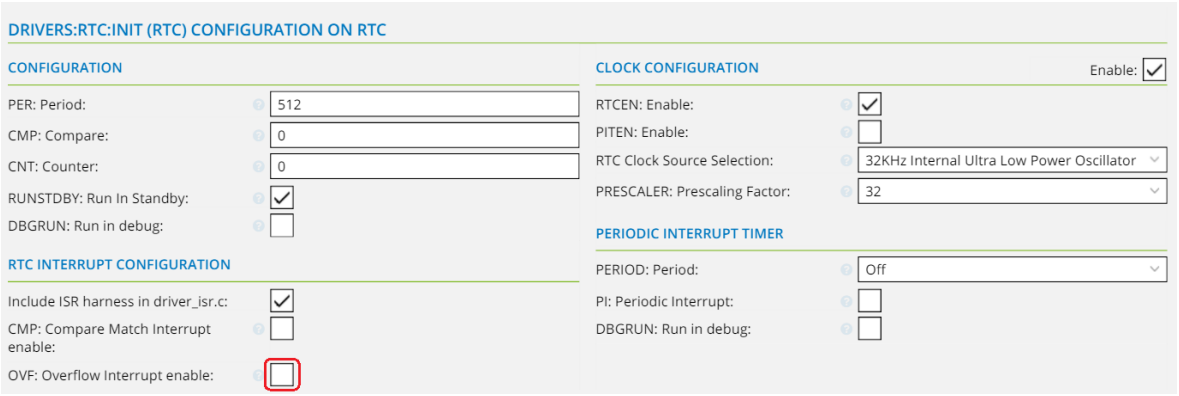


Info: The Event system configuration is now completed with the event generator, event channel, and event user defined, as shown below.

Figure 6-3. Event User Selection

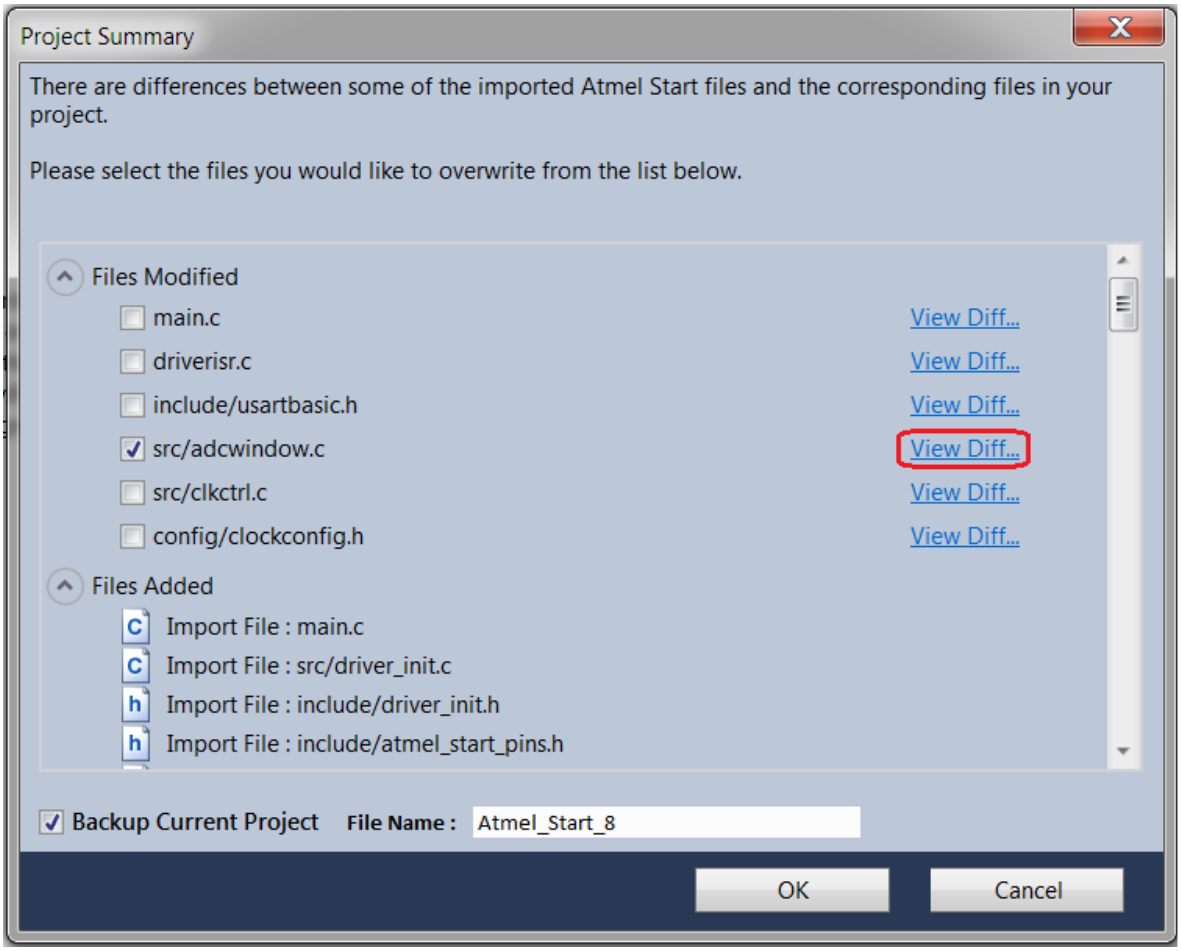
7. RTC will be reconfigured in *Atmel | START*:
   Click *DASHBOARD* on the left side of the window and click on the existing *RTC_0* module to reopen the RTC configuration window. Unselect the checkbox for *OVF: Overflow Interrupt enable* in the red marking, as shown below.

Figure 6-4. RTC Reconfiguration



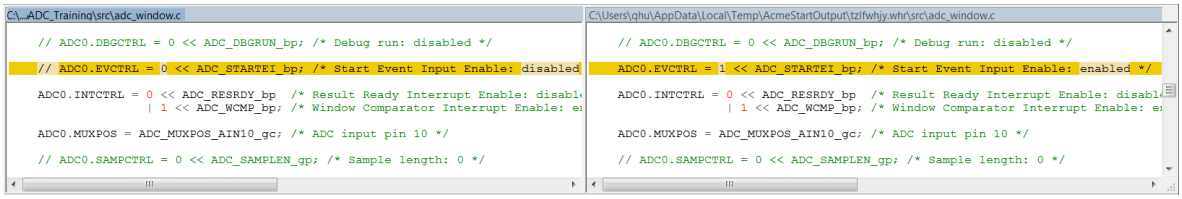
8. Click the *GENERATE PROJECT* button. The project summary window pops up as shown in the figure blow:

Figure 6-5. Project Summary After Regenerated



9. Click *View Diff* for the *src/adc_window.c* file (see the red marking) to inspect the difference, as shown below:

Figure 6-6. driver_isr.c View Diff



10. Tick the checkbox in order to overwrite the *src/adc_window.c* file.
11. Click *OK* to regenerate the project, with overwriting *adc_window*.

6.2 Event System Driver Code Development

To do: Add code in driver_isr.c to send ADC data to the terminal when the result is outside of the configured window range.

1. Update the *driver_isr.c* file:

- Add `#include "main.h"` as the difference is shown in Figure 6-6
- Add in the two lines of `ADC_get_result();` and `sendflag=1;` in the `ISR(ADC0_WCOMP_vect)` function, as shown in the comparison figure above
- The `ISR(RTC_CNT_vect)()` function should now have been removed as it has been replaced by event system

2. The complete code of the *driver_isr.c* looks like:

```
#include <driver_init.h>
#include <compiler.h>
#include "main.h"

ISR(ADC0_WCOMP_vect)
{
    /* Insert your ADC window comparator interrupt handling code here */
    ADC_get_result();
    sendflag = 1;

    /* The interrupt flag has to be cleared manually */
    ADC0.INTFLAGS = ADC_WCMP_bm;
}
```

3. Press *F7* to build the solution.

Info: The solution should build without errors.

4. Run the project by clicking *Starting Without Debugging* or use the hot-key *CTRL+ALT+F5*.

Info: The application should start executing on the kit.

5. Click the *Data Visualizer* tab. Now the potentiometer can be adjusted. The ADC data printed in the terminal and the plotted ADC data shown in the graph should have the same behavior, as shown in Figure 5-6.

Result: The *Event system* functions as expected and has successfully replaced the RTC overflow interrupt.

7. Conclusion

This training demonstrated configuring RTC triggering ADC conversation using interrupt or event system. With the feature of Atmel Studio 7: Data Visualizer, it is easy to check the data output on the USART terminal or even in the graph view.

With *Atmel | START*, it is easy to configure/reconfigure a project by adding/removing peripheral drivers and use automatically generated driver functions in the code.

With *Atmel Studio*, it is easy to run real-time debugging of an application and use I/O view, which provides register view capability and allows modifying the microcontroller registers in real-time. It is possible to debug the application using various debugging methods such as:

- Breakpoints
- I/O view

8. Revision History

Doc Rev.	Date	Comments
A	08/2017	Initial document release.

The Microchip Web Site

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

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

Customer Change Notification Service

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

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

Customer Support

Users of Microchip products can receive assistance through several channels:

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

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

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

Microchip Devices Code Protection Feature

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

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

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

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

Legal Notice

Trademarks

ISBN: 978-1-5224-2066-8

Quality Management System Certified by DNV

ISO/TS 16949

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

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Asia Pacific Office** Suites 3707-14, 37th Floor Tower 6, The Gateway Harbour City, Kowloon **Hong Kong** Tel: 852-2943-5100 Fax: 852-2401-3431 **Australia - Sydney** Tel: 61-2-9868-6733 Fax: 61-2-9868-6755 **China - Beijing** Tel: 86-10-8569-7000 Fax: 86-10-8528-2104 **China - Chengdu** Tel: 86-28-8665-5511 Fax: 86-28-8665-7889 **China - Chongqing** Tel: 86-23-8980-9588 Fax: 86-23-8980-9500 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 Fax: 86-571-8792-8116 **China - Hong Kong SAR** Tel: 852-2943-5100 Fax: 852-2401-3431 **China - Nanjing** Tel: 86-25-8473-2460 Fax: 86-25-8473-2470 **China - Qingdao** Tel: 86-532-8502-7355 Fax: 86-532-8502-7205 **China - Shanghai** Tel: 86-21-3326-8000 Fax: 86-21-3326-8021 **China - Shenyang** Tel: 86-24-2334-2829 Fax: 86-24-2334-2393 **China - Shenzhen** Tel: 86-755-8864-2200 Fax: 86-755-8203-1760 **China - Wuhan** Tel: 86-27-5980-5300 Fax: 86-27-5980-5118 **China - Xian** Tel: 86-29-8833-7252 Fax: 86-29-8833-7256	**China - Xiamen** Tel: 86-592-2388138 Fax: 86-592-2388130 **China - Zhuhai** Tel: 86-756-3210040 Fax: 86-756-3210049 **India - Bangalore** Tel: 91-80-3090-4444 Fax: 91-80-3090-4123 **India - New Delhi** Tel: 91-11-4160-8631 Fax: 91-11-4160-8632 **India - Pune** Tel: 91-20-3019-1500 **Japan - Osaka** Tel: 81-6-6152-7160 Fax: 81-6-6152-9310 **Japan - Tokyo** Tel: 81-3-6880- 3770 Fax: 81-3-6880-3771 **Korea - Daegu** Tel: 82-53-744-4301 Fax: 82-53-744-4302 **Korea - Seoul** Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934 **Malaysia - Kuala Lumpur** Tel: 60-3-6201-9857 Fax: 60-3-6201-9859 **Malaysia - Penang** Tel: 60-4-227-8870 Fax: 60-4-227-4068 **Philippines - Manila** Tel: 63-2-634-9065 Fax: 63-2-634-9069 **Singapore** Tel: 65-6334-8870 Fax: 65-6334-8850 **Taiwan - Hsin Chu** Tel: 886-3-5778-366 Fax: 886-3-5770-955 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 Fax: 886-2-2508-0102 **Thailand - Bangkok** Tel: 66-2-694-1351 Fax: 66-2-694-1350	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **France - Saint Cloud** Tel: 33-1-30-60-70-00 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820