Training Manual

ADC and Power Optimization with tinyAVR® 0- and 1-series, and megaAVR® 0-series

Prerequisites

- **Hardware Prerequisites**
  - ATtiny817 Xplained Pro evaluation kit
  - Micro-USB cable (Type-A/Micro-B)
  - A potentiometer
  - Three male-to-female wires
  - Internet connection
- **Software Prerequisites**
  - Atmel Studio 7.0
  - Web browser. 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 training contains five hands-on applications doing ADC data conversion, with current consumption measured for each application. The training starts with a simple ADC conversion application. In the following applications, different techniques have been introduced in order to demonstrate how the current consumption can be reduced.

This training will also demonstrate how to use *Atmel | START* to get started with AVR® ADC applications development. The ADC applications have been developed step-by-step in *Atmel Studio*. This training has been developed on the *ATtiny817 Xplained Pro* evaluation kit, but should apply for all tinyAVR® 0- and 1-series, and megaAVR® 0-series devices.

Table of Contents

1. Relevant Devices

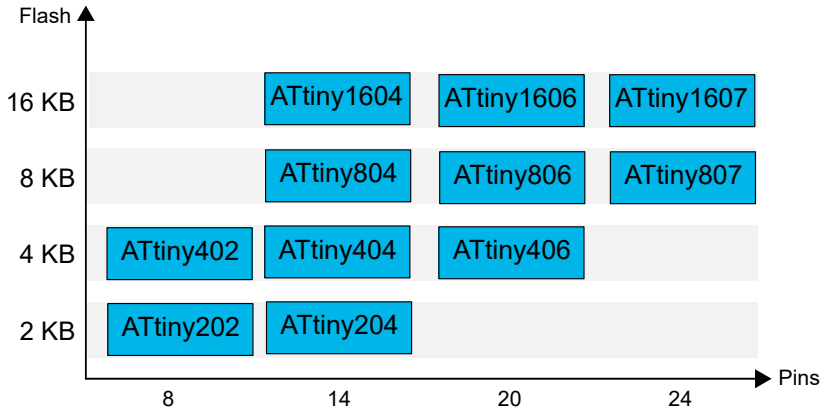
This chapter lists the relevant devices for this document.

1.1 tinyAVR® 0-series

The figure below shows the tinyAVR® 0-series, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin- and feature compatible.
- Horizontal migration to the left reduces the pin count and therefore, the available features.

Figure 1-1. Device Family Overview



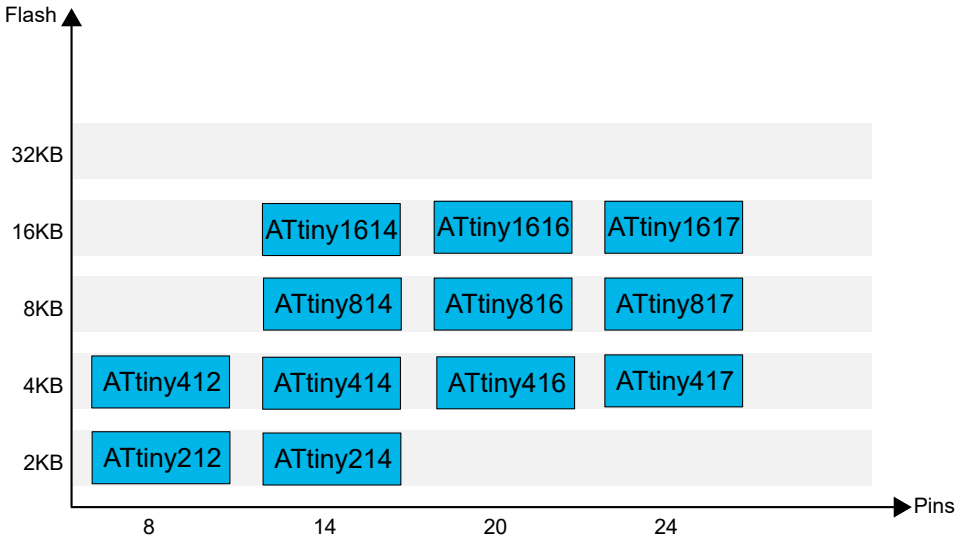
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-series

The figure below shows the tinyAVR® 1-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin compatible and provide the same or more features. Downward migration may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and therefore, the available features.

Figure 1-2. tinyAVR® 1-series Overview



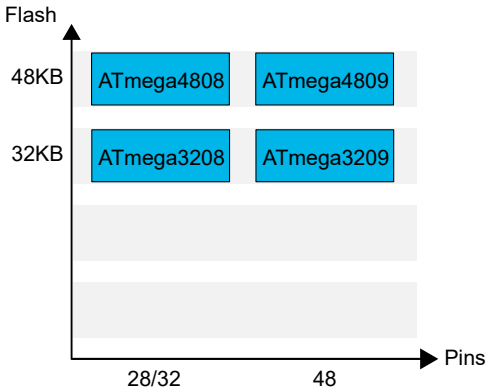
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

The figure below shows the megaAVR® 0-series devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible.
- Horizontal migration to the left reduces the pin count and therefore, the available features.

Figure 1-3. megaAVR® 0-series Overview



Devices with different Flash memory size typically also have different SRAM and EEPROM.

2. Icon Key Identifiers

The following icons are used to identify the different assignment sections and to reduce the complexity.

Info: Delivers contextual information about a specific topic.

Tip: Highlights useful tips and techniques.

To do: Highlights objectives to be completed.

Result: Highlights the expected result of an assignment step.

Indicates important information.

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

3. Assignment 1: ADC Conversion with USART Print Application

In this assignment, *Atmel Studio* will be used to develop an application using ADC and USART drivers from *Atmel | START*. The ADC will be configured to run in single conversion mode and a potentiometer will be connected to the ADC input pin to study the ADC functionality. The ADC data will be sent via USART to the embedded terminal in *Atmel Studio's Data Visualizer*. In *Data Visualizer* the current consumption of the application will be analyzed using the embedded *Power Analyzer*.

Peripherals used:

- ADC
- USART

Clock details:

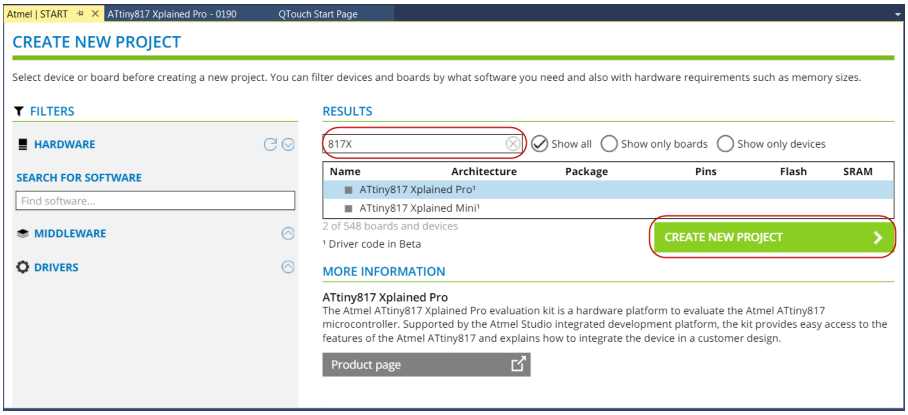
- Main Clock - 5 MHz
- ADC - 625 KHz (5 MHz main clock with prescaler 8)
- USART - 5 MHz

3.1 *Atmel | START* Project Creation

To do: Create a new *Atmel | START* Project, where CLOCK, ADC, and USART modules are configured.

1. Open *Atmel Studio 7*.
2. Create a new *Atmel | START* project in *Atmel Studio* by
   2.1. Open the *CREATE NEW PROJECT* window by selecting *File->New->Atmel START project*.
   2.2. Select the target hardware by entering *817X* in the *Filter on device...* text box and selecting *ATtiny817 Xplained Pro* from the list.
   2.3. Create the project by clicking *CREATE NEW PROJECT* as shown in Figure 3-1.

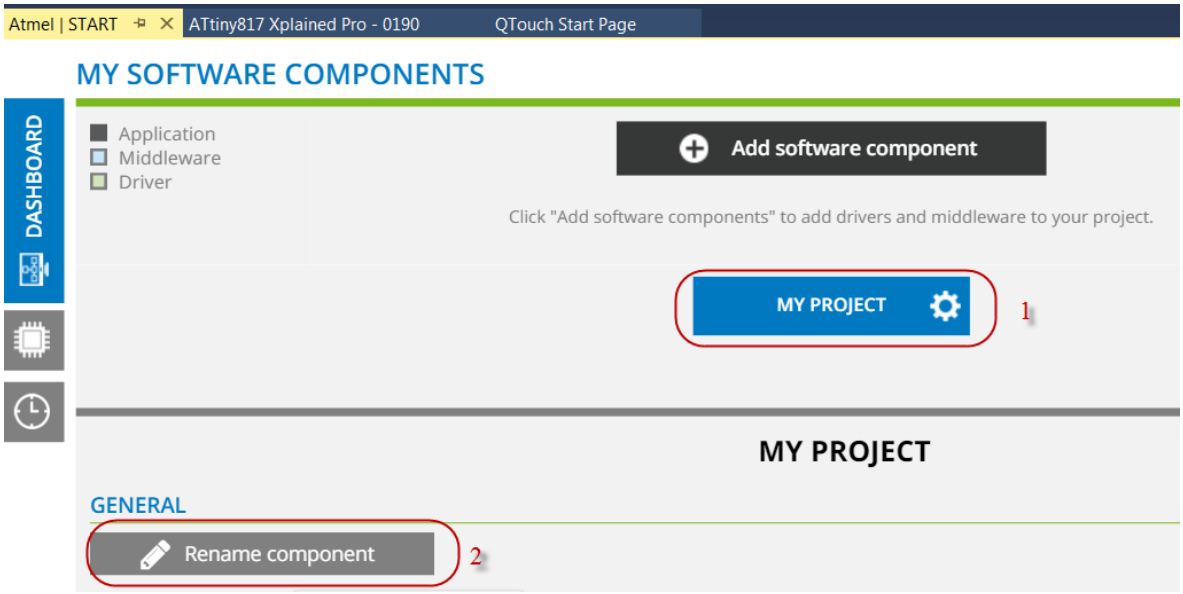
Figure 3-1. CREATE NEW PROJECT



Info: The *MY SOFTWARE COMPONENTS* window should appear.

3. Rename the project by clicking *MY PROJECT* in the *MY SOFTWARE COMPONENTS* window and select *Rename Component* as shown in Figure 3-2.

Figure 3-2. Rename Component

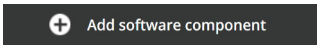


Info: The *RENAME COMPONENT* window should appear.

Enter the project name, for example, *ADC_Training*, in the popped up *RENAME COMPONENT* window and click *Rename* to close it.

Note: This step is optional to rename the project name.

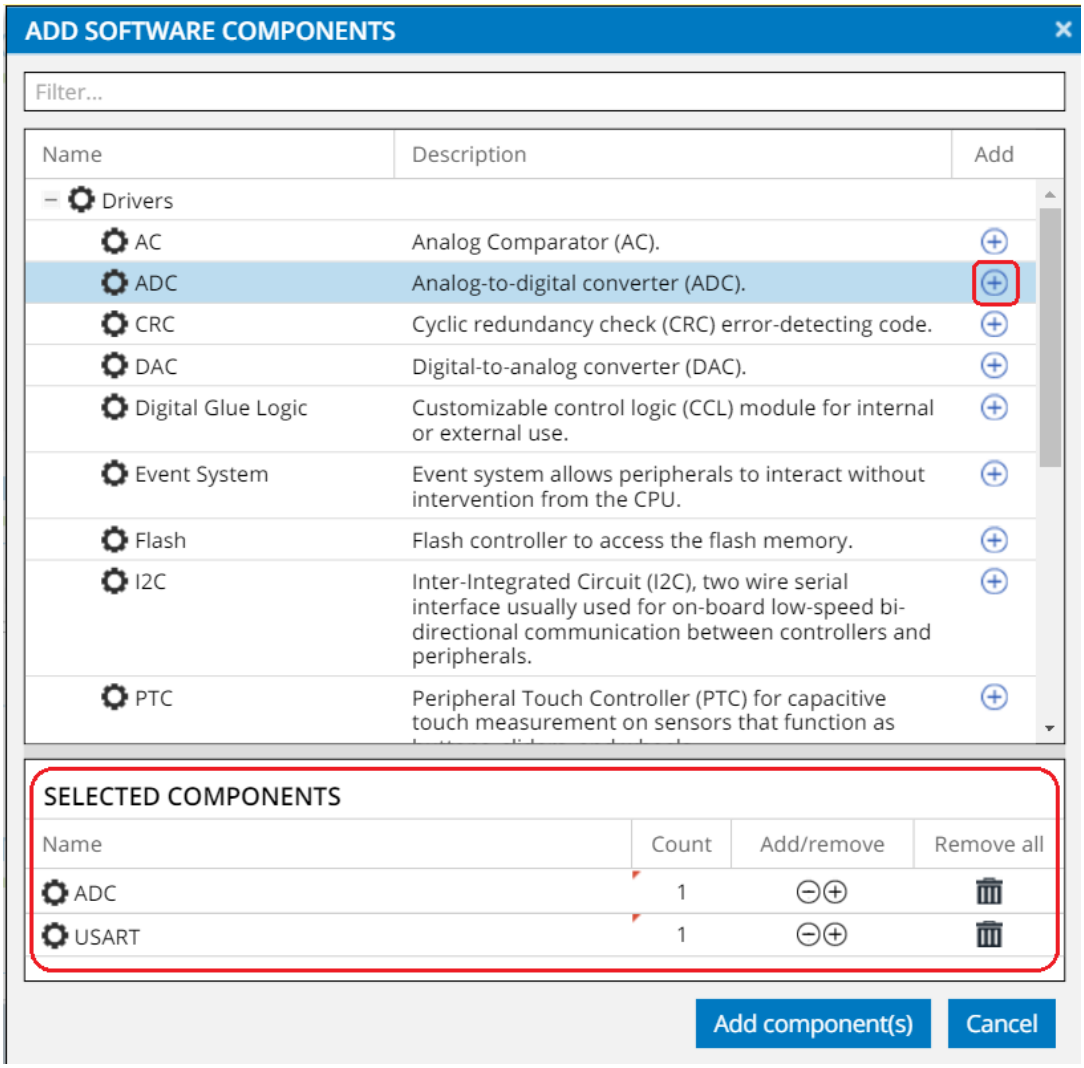
4. Add ADC and USART drivers to the project by
   - 4.1. Open the *ADD SOFTWARE COMPONENTS* window by clicking

   - 4.2. Expand *Drivers* by clicking +
   - 4.3. Select ADC and USART by clicking  , as shown in Figure 3-3

Info: ADC and USART will be highlighted as the selected modules as shown in Figure 3-3.

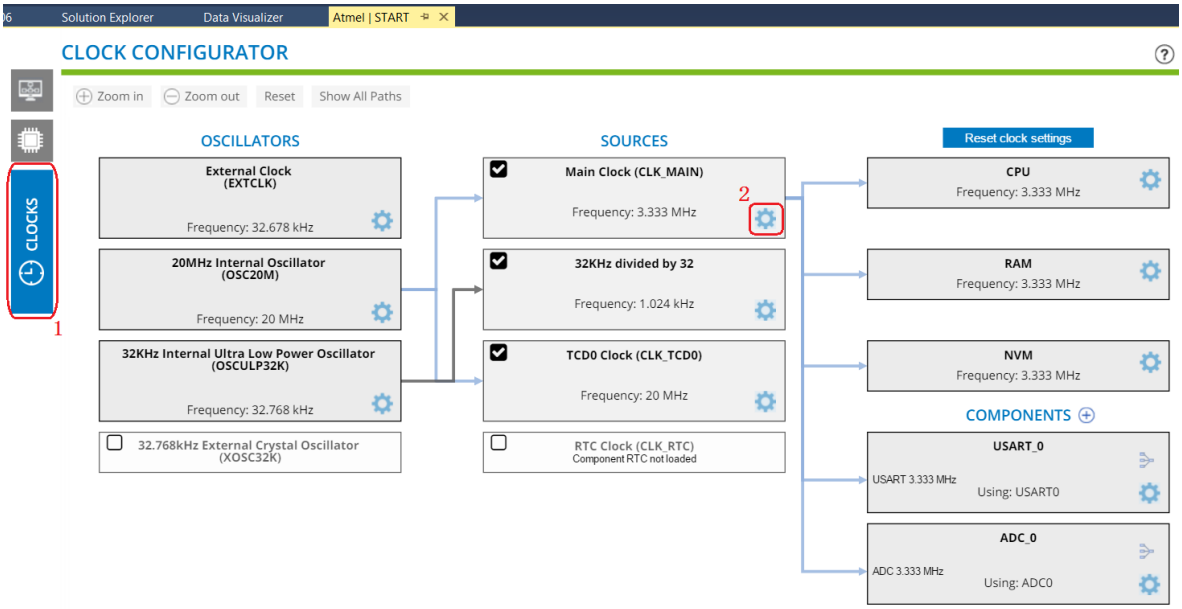
Figure 3-3. Add RTC and USART Modules



4.4. Add the drivers to the project by clicking the *Add component(s)* button.

5. Open the *CLOCK CONFIGURATOR* window by clicking  on the navigation tab at the left side of the window as shown in Figure 3-4.

Figure 3-4. Clock Configuration

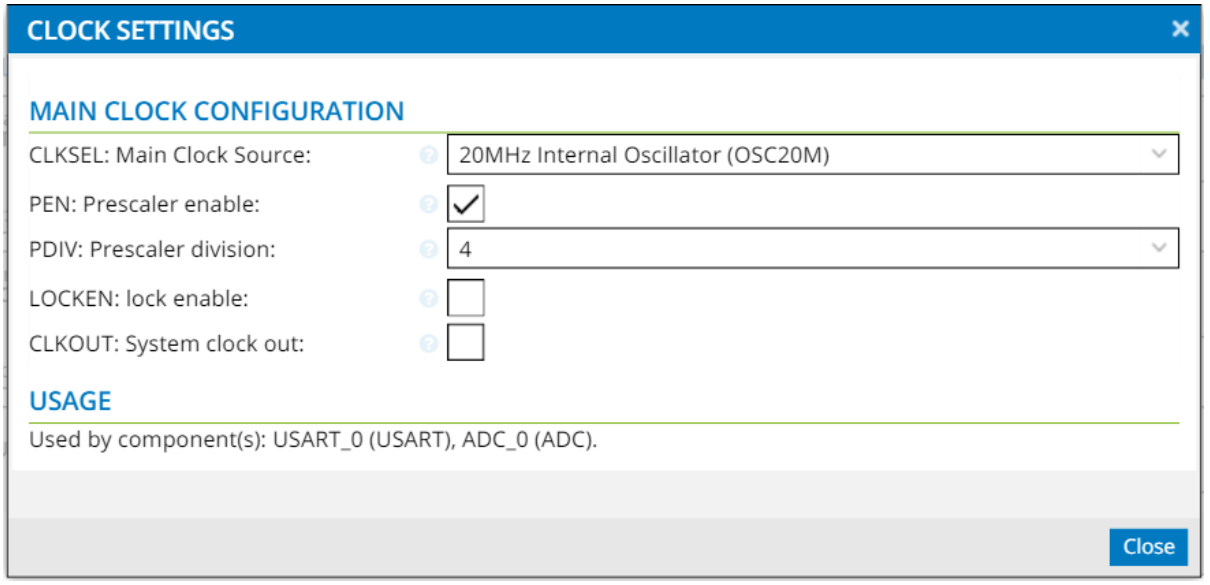


Info: The *CLOCK CONFIGURATOR* window consists of oscillators and clock sources of different types. Required clock source can be selected and the calculated output frequency will be displayed.

- The *OSCILLATORS* section displays the oscillators available for the selected device. The oscillator parameters can be configured by clicking to select *Settings Dialog*.
- The *SOURCES* section is used to configure clock frequency by selecting input signal and prescaler settings.

6. Configure the Main Clock (CLK_MAIN) clock source by
   - 6.1. Open the *CLOCK SETTINGS* window by clicking as shown in Figure 3-4.
   - 6.2. Select *20 MHz* as the *Main Clock Source* from the drop-down menu.
   - 6.3. Select *4* as the *Prescaler division* from the drop-down menu.
   - 6.4. Close *CLOCK SETTINGS* by clicking the *Close* button.

Figure 3-5. Clock Settings

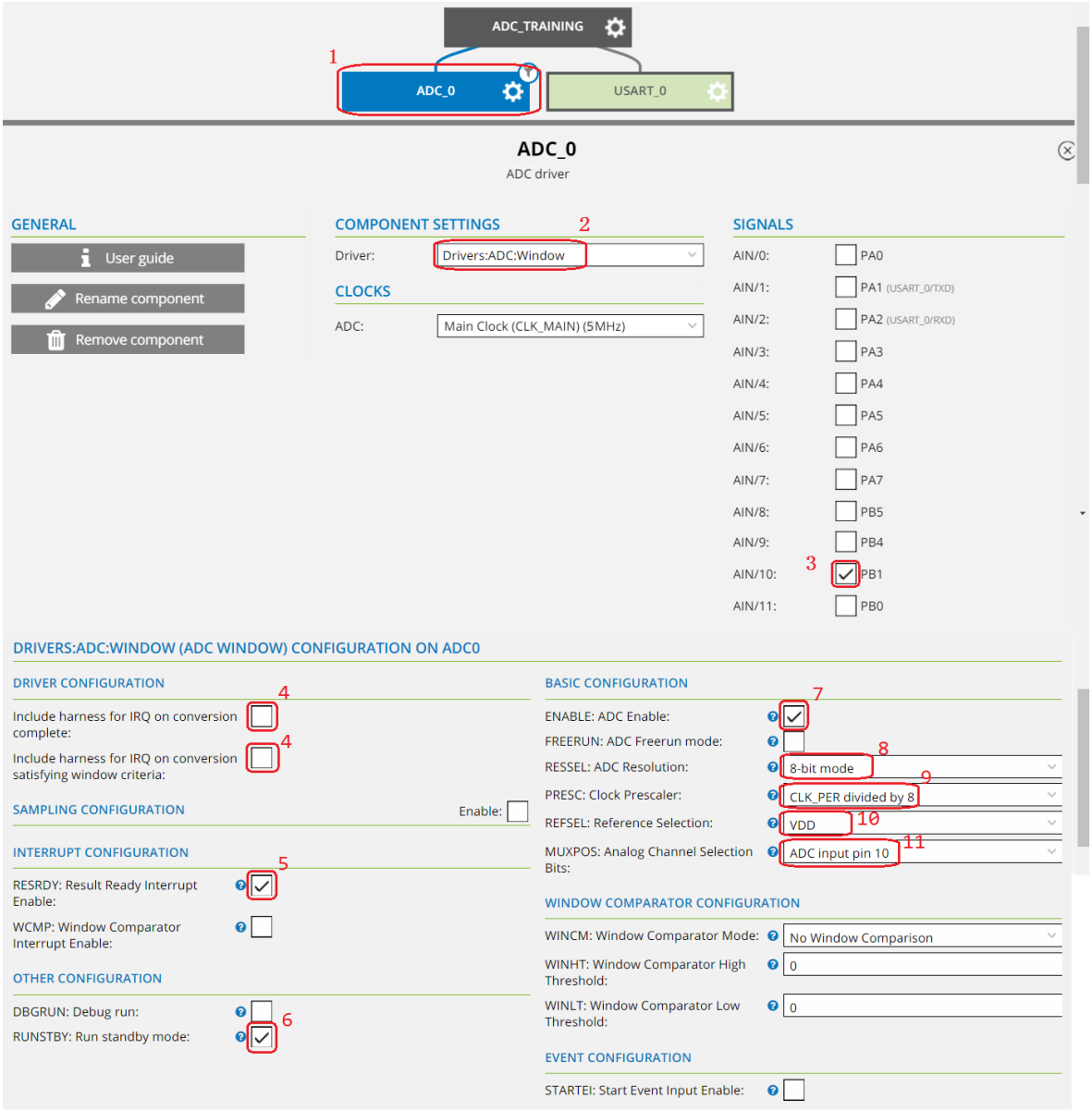


Info: For this application, 20 MHz OSC is chosen as the main clock source with prescaler division equal to 4. The resulting CPU clock frequency is 5 MHz.

For quick reference, clicking next to each configuration will provide the data sheet description of individual bit settings.

7. Return to *MY SOFTWARE COMPONENTS*, by clicking DASHBOARD in the navigation tab on the top of the left side of the window.
8. Configure the ADC module by following the steps in Figure 3-6:

Figure 3-6. ADC Configuration



8.1. Open the configuration menu for ADC, by clicking *ADC_0*.

8.2. Select *Drivers:ADC:Window* as *Driver* from the drop-down menu.

8.3. Select *PB1* as analog input *AIN/10* from the *SIGNALS* menu.

8.4. Ensure all interrupt handlers end up in *driver_isr.c* for this training, by un-checking *Include harness for IRQ on conversion*...

8.5. Enable the Result Ready Interrupt, by selecting the *RESRDY:Result Ready Interrupt Enable* checkbox.

8.6. Ensure the ADC runs when the device is in Standby Sleep mode, which will be used in the next assignment.

8.7. Ensure the ADC is enabled after initialization, by selecting *ENABLE:ADC Enable*.

8.8. Select *8-bit mode* as the *RESSEL:ADC Resolution*.

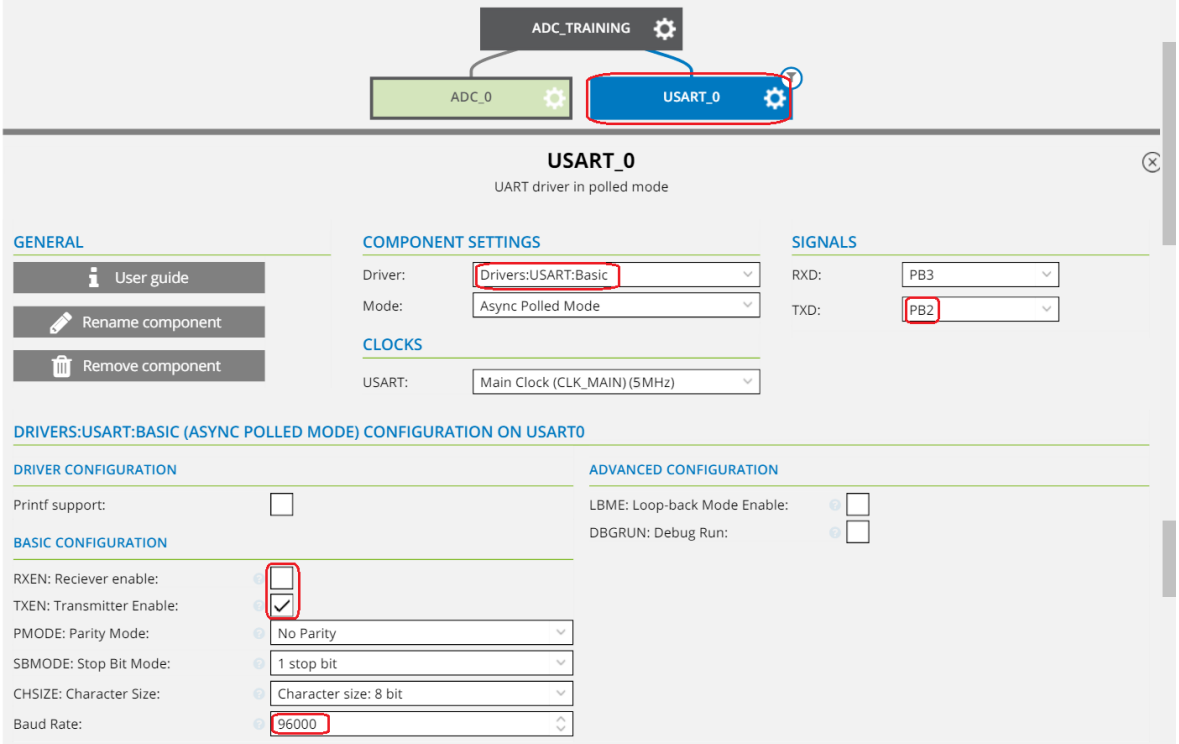
8.9. Configure the ADC clock to 625 KHz, by selecting *CLK_PER dividend by 8* from the *PRESC:Clock Prescaler* drop-down menu. The main clock runs at 5 MHz, which gives ADC clock at 625 KHz.

8.10. Select *VDD* as *Reference Selection* from the drop-down menu.

8.11. Select *ADC input pin 10* in the *MUXPOS: Analog Channel Selection Bits* field to select the analog input *AIN/10* connected to the ADC.

9. Configure the USART module by following the steps in Figure 3-7:

Figure 3-7. USART Configuration



9.1. Open the USART configuration menu, by clicking *USART_0*.

9.2. Select *Driver:USART:Basic* as *Driver* from the drop-down menu.

9.3. Select *PB2* as the USART output by selecting it from the *TXD* drop-down menu.

9.4. Ensure the output is enabled upon initialization, by selecting *TXEN:Transmitter Enable*.

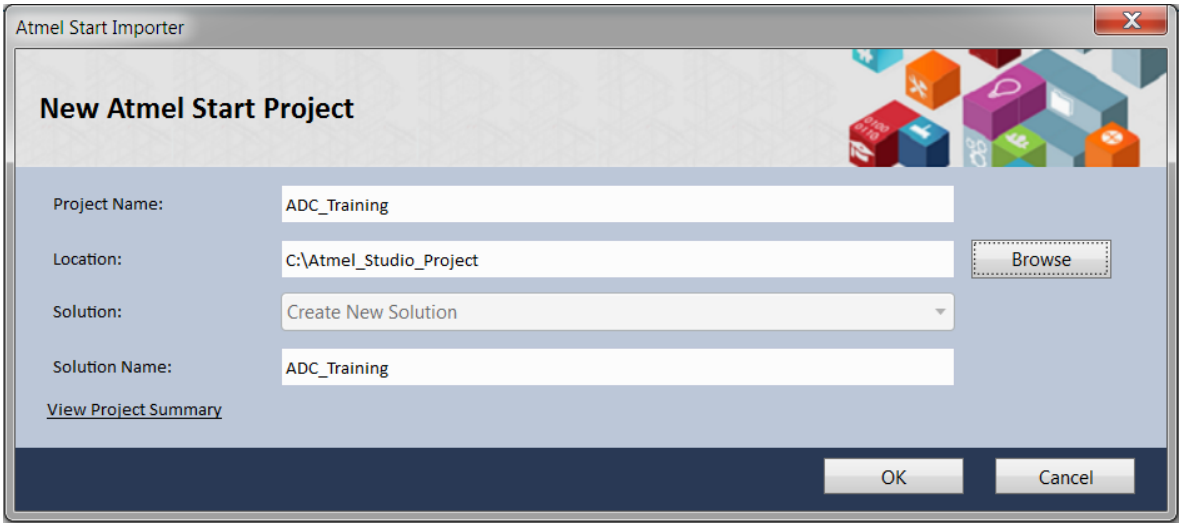
Info: The device will only send data over USART, so the *RXEN:Receiver Enable* can remain unchecked.

9.5. Configure a high baud rate in order to ensure fast USART data transfer, by setting the *Baud Rate* to *96000*.

Info: Transferring the USART data fast will enable the device to spend more time in Sleep mode, which will reduce the current consumption. The Sleep mode will be used in the following assignment.

10. Generate the project by clicking GENERATE PROJECT.
11. Select the desired path where the project should be stored as shown in Figure 3-8 as an example and click *OK*.

Figure 3-8. New Atmel Start Project

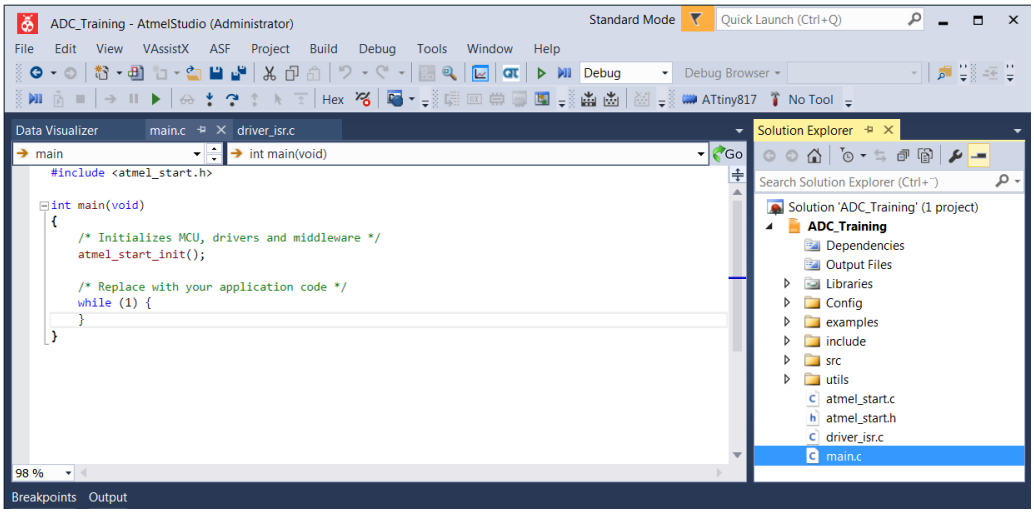


Result: An *Atmel | START* project, featuring an ADC and USART driver, has been created in *Atmel Studio*.

3.2 *Atmel | START* Project Overview in *Atmel Studio*

This section will provide a walk-through of the *Atmel | START* project generated in 3.1 Atmel | START Project Creation. The generated project contains peripheral driver functions and files, as well as `main()` function that initializes all drivers included in the project.

Figure 3-9. Project Overview



Info: The main window, as seen in Figure 3-9, shows the generated code for the *main.c* file. On the right side is the *Solution Explorer* window, which contains the folders and files generated by *Atmel | START*:

- The *Config* folder contains the clock configuration. The main clock is configured to 5 MHz and is defined by `#define F_CPU 5000000` in *clock_config.h*.
- Header and source files of the drivers are located in the *src* and *include* folders. For example, the *adc_basic.c* and *usart_basic.c* files contain the drivers which will be used for the two modules in the coming sections.
- The *utils* folder contains files that define some functions to be commonly used by the drivers and application.
- In the *atmel_start.c* file, the `atmel_start_init()` function initializes the MCU, drivers, and middleware in the project.
- When interrupts are enabled in the project's configuration, the *driver_isr.c* file contains the interrupt service routines (ISRs).

To do: Get to know the structure of the *Atmel | START* project.

1. Open the *main.c* file, by double-clicking it from the *Solution Explorer* window.
2. Go to the implementation of `atmel_start_init()` by
   2.1. Hover over `atmel_start_init()`.
   2.2. *Right-click → Goto Implementation*

   Info: A menu, showing the different locations, will appear.

   2.3. Jump to where the function is implemented by selecting the first option from the menu.

2.4. right-clicking → *Goto Implementation*

3. Go to the implementation of `system_init()` by

3.1. Hover over `system_init()`.

3.2. *Right-click → Goto Implementation*

Info: A menu, showing the different locations, will appear.

3.3. Jump to where the function is implemented by selecting the first option from the menu.

Info: The implementation of `system_init()`, as shown in Figure 3-10, should now be visible in *Atmel Studio's* editor window.

Figure 3-10. system_init() Function

```
void system_init()
{
    mcu_init();

    CLKCTRL_init();

    ADC_0_initialization();

    USART_0_initialization();

    CPUINT_init();

    SLPCTRL_init();

    BOD_init();
}
```

Info:

- The function `mcu_init()` enables the internal pull-up resistor on all pins to reduce power consumption.
- All driver initialization functions are called from the `system_init()` function.
- All the module initialization functions can be right-clicked in the same way as described above to see how each module is initialized. For example, `ADC_0_initialization()` and `USART_0_initialization()` configures the pins and initializes the registers for the ADC and USART module respectively.

4. Go to the implementation of `USART_0_initialization()` by

4.1. Hover over `USART_0_initialization()`.

4.2. *Right-click → Goto Implementation*

Info: A menu, showing the different locations, will appear.

4.3. Jump to where the function is implemented by selecting the first option from the menu.

Info: The implementation of `USART_0_initialization()`, as shown in Figure 3-11, should now be visible in the *Atmel Studio* editor window.

Figure 3-11. USART_0_initialization() Function

```
/* configure the pins and initialize the registers */
void USART_0_initialization(void)
{

    // Set pin direction to input
    PB3_set_dir(PORT_DIR_IN);

    PB3_set_pull_mode(
        // <y> Pull configuration
        // <id> pad_pull_config
        // <PORT_PULL_OFF"> Off
        // <PORT_PULL_UP"> Pull-up
        PORT_PULL_OFF);

    // Set pin direction to output
    PB2_set_dir(PORT_DIR_OUT);

    PB2_set_level(
        // <y> Initial level
        // <id> pad_initial_level
        // <false"> Low
        // <true"> High
        false);

    USART_0_init();
}
```

Figure 3-11 shows the initialization code for the USART module, where the *PB2* and *PB3* pins are configured for the TX and RX transmission.

Note: In this training, only TX transmission is used.

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

3.3 Add ADC and USART Functionality to Application Code

Once the ADC and USART functional drivers have been added using *Atmel | START*, developing the application code can be started.

To do: Add code to the application which performs ADC conversion and sends the ADC result via USART to a terminal.

1. Include functions for adding delays to the application code by adding the following line of code at the beginning of *main.c*.

```
#include <util/delay.h>
```

2. Add ADC and USART functionality to the application, by adding the following piece of code in the `while(1)`-loop in the `main` function.

```
ADC_0_start_conversion(10); //Start ADC conversion on channel 10
while(!ADC_0_is_conversion_done()); //wait for ADC conversion is done
USART_0_write(ADC_0_get_conversion_result()); //USART write ADC result
while(!(USART0.STATUS & USART_TXCIF_bm)); //wait for USART TX complete
_delay_ms(500); // delay to easier observe changes to ADC input in terminal
```

Info: *main.c* should look similar to the code in Figure 3-12

Figure 3-12. main.c Code

```
#include <atmel_start.h>
#include <util/delay.h>

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

    /* Replace with your application code */
    while (1) {
        ADC_0_start_conversion(10); //Start ADC conversion on channel 10
        while(!ADC_0_is_conversion_done()); //wait for ADC conversion is done
        USART_0_write(ADC_0_get_conversion_result()); //USART write ADC result
        while(!(USART0.STATUS & USART_TXCIF_bm)); //wait for USART TX complete
        _delay_ms(500); // delay to easier observe changes to ADC input in terminal
    }
}
```

3. Go to the implementation of `ADC_0_start_conversion()` by
   3.1. Hover over `ADC_0_start_conversion()`.
   3.2. *Right-click → Goto Implementation*

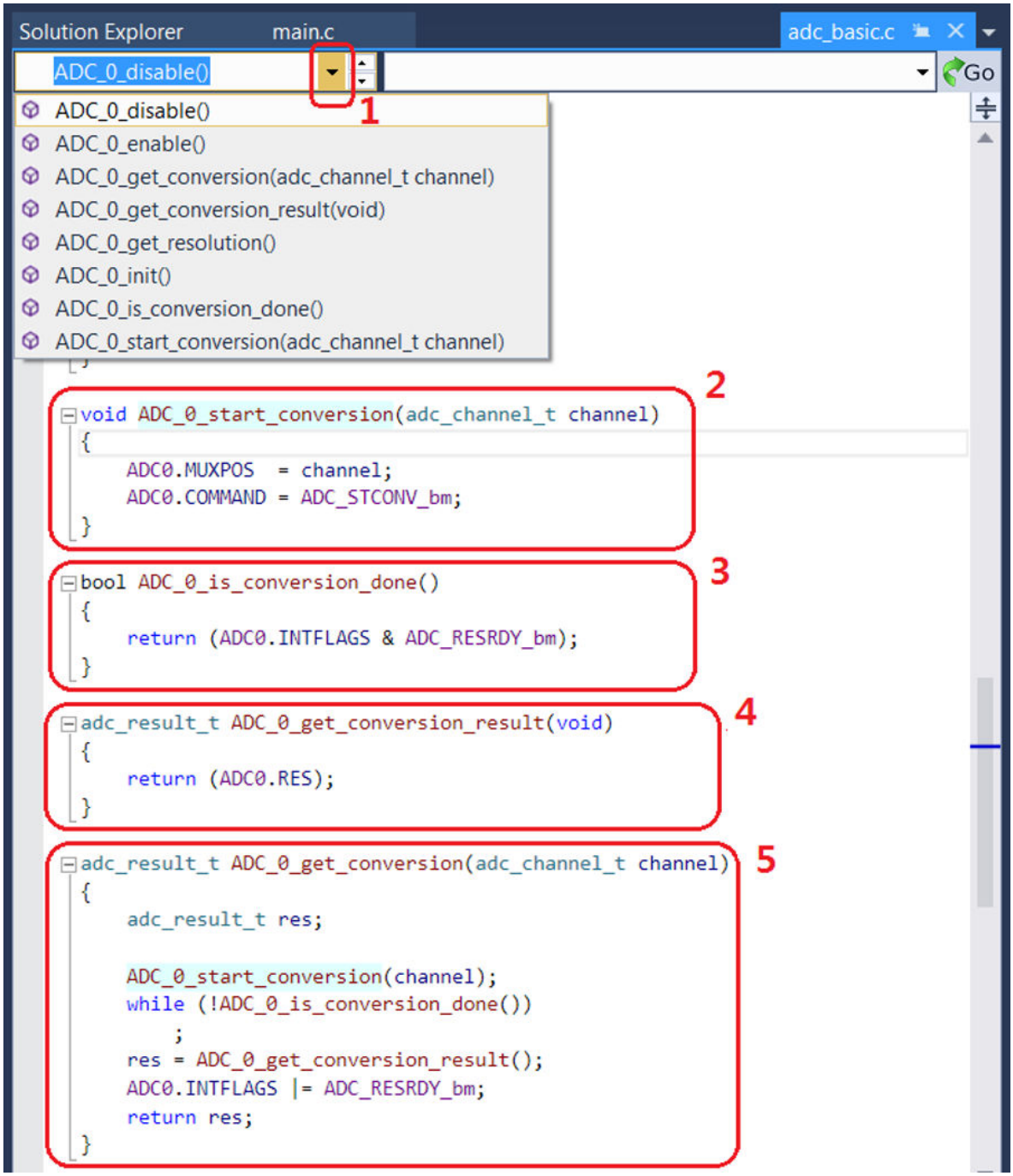
Info: A menu, showing the different locations, will appear.

   3.3. Jump to where the function is implemented by selecting the first option from the pop-up window.

Info: The implementation of `ADC_0_start_conversion()`, which is located in *adc_basic.c*, should now be visible in the *Atmel Studio* editor window.

4. View the complete list of ADC driver functions by clicking the arrow, shown in Figure 3-13.

Figure 3-13. ADC Driver Function List



Info: The ADC driver functions marked with 2-5 in Figure 3-13 are all used by this training.

5. Go to the implementation of `USART_0_write()` by
   5.1. Hover over `USART_0_write()` in the *main.c* file.

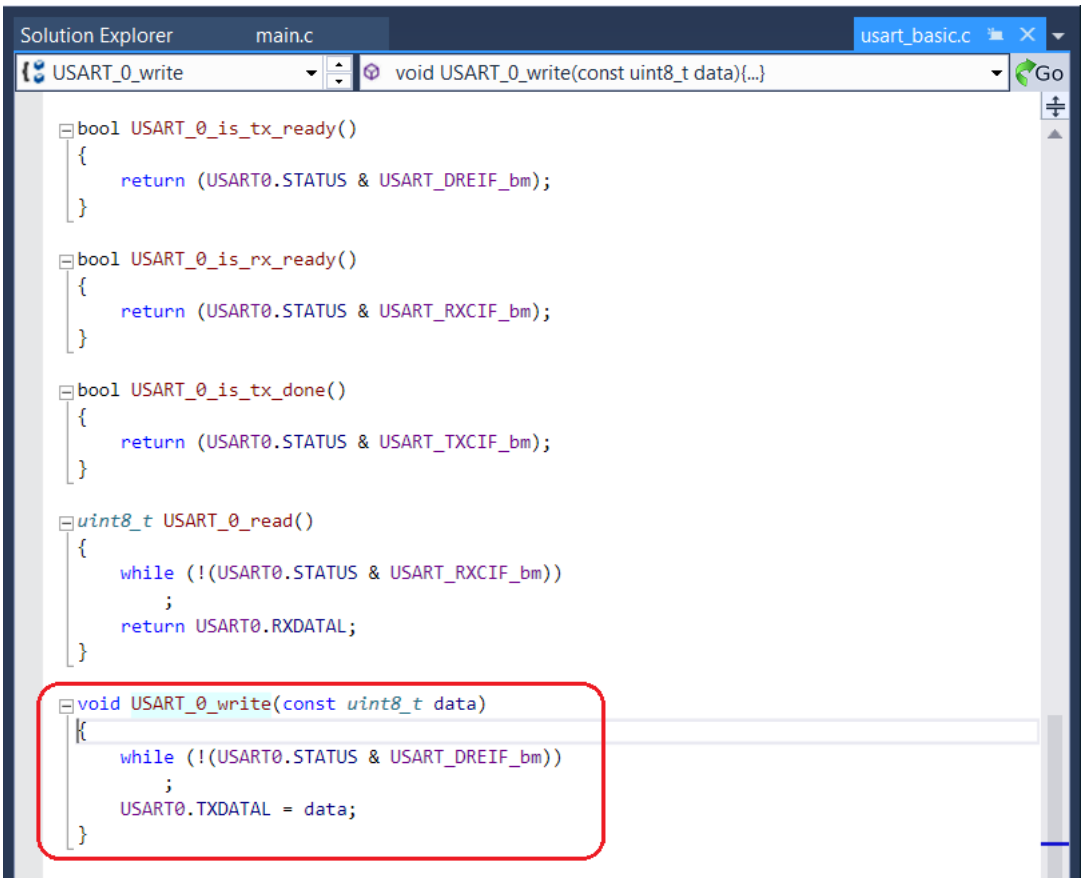
5.2. *Right-click → Goto Implementation*

Info: A menu, showing the different locations, will appear.

5.3. Jump to where the function is implemented by selecting the first option from the pop-up window.

Info: The implementation of `USART_0_write()`, which is located in *usart_basic.c* should now be visible in the *Atmel Studio* editor window, as shown in Figure 3-14.

Figure 3-14. USART Driver Function List



Info: `USART_0_write` is the only USART driver function used in this training.

Result: Adding ADC and USART functionality to the application code is completed.

3.4 Hardware Setup and Programming

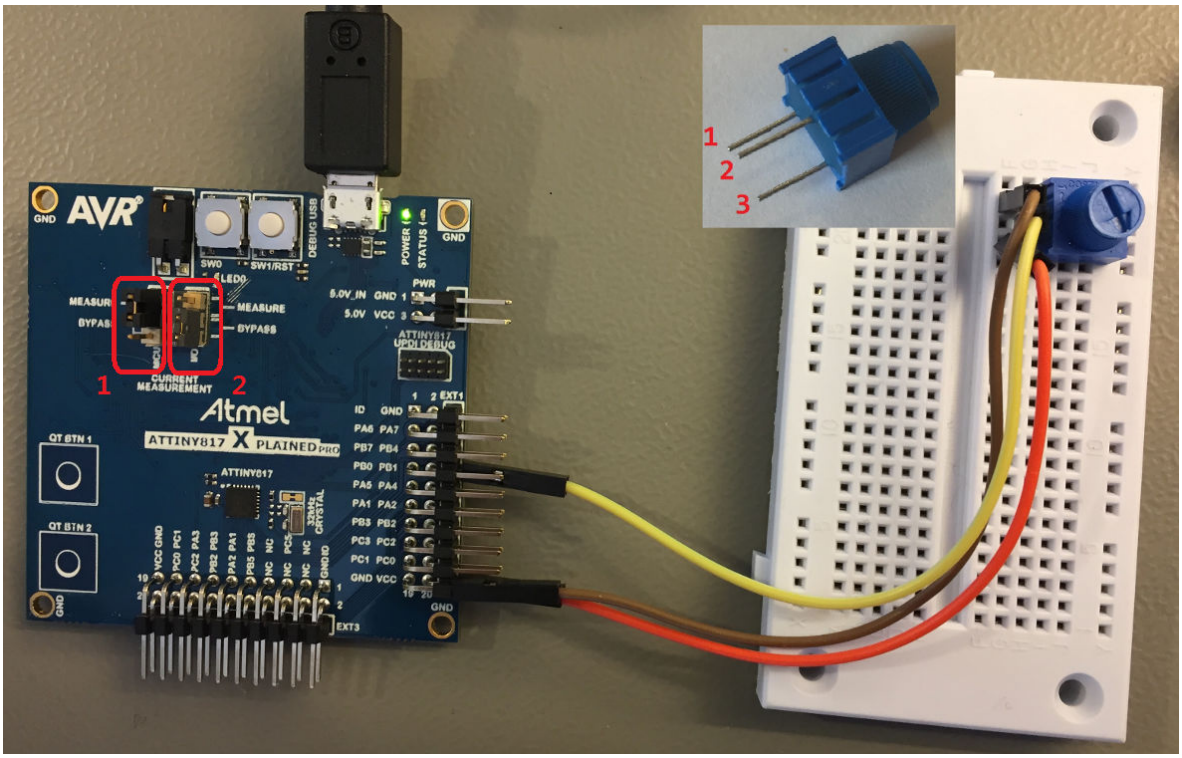
Once the application code development is completed it is time to connect the potentiometer to the ADC input pin available on ATtiny817 Xplained Pro and program the device on ATtiny817 Xplained Pro.

To do:

- Connect the potentiometer to the ADC input pin on ATtiny817 Xplained Pro
- Connect ATtiny817 Xplained Pro to *Atmel Studio*
- Program the device on ATtiny817 Xplained Pro

1. Connect the potentiometer to the breadboard. Notice how the pins correspond to the letters/ numbers on the breadboard.
2. Connect the pins on the potentiometer to the extension header of ATtiny817 Xplained Pro, as shown in Figure 3-15.
   2.1. Potmeter pin 1: Connect to the *GND* pin on the kit.
   2.2. Potmeter pin 2: Connect to the *PB1* pin on the kit.
   2.3. Potmeter pin 3: Connect to the *VCC* pin on the kit.

Figure 3-15. Connect Potentiometer to the *ATtiny817 Xplained Pro* Kit



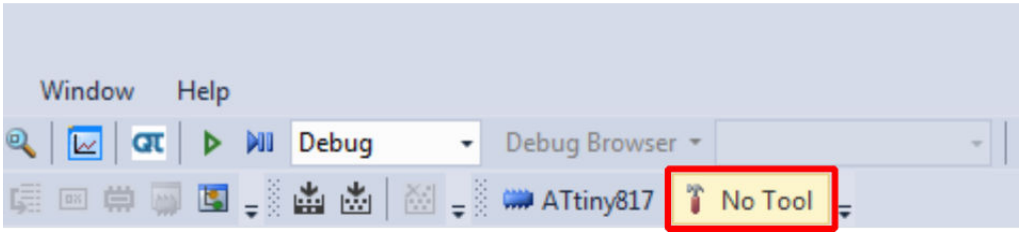
Info: A potentiometer, also called a potmeter, is a three-terminal resistor with a sliding or rotating contact that provides an adjustable voltage divider.

3. Enable current measurement of the device by placing the jumper on *MEASURE* for *MCU*, as shown in Figure 3-15.
4. Disable current measurement of the I/O pins by placing the jumper on *BYPASS* for *I/O*, as shown in Figure 3-15.
5. Connect ATtiny817 Xplained Pro to the computer by using a Micro-USB cable, as shown in Figure 3-15.

Info: The ATtiny817 screen shows up.

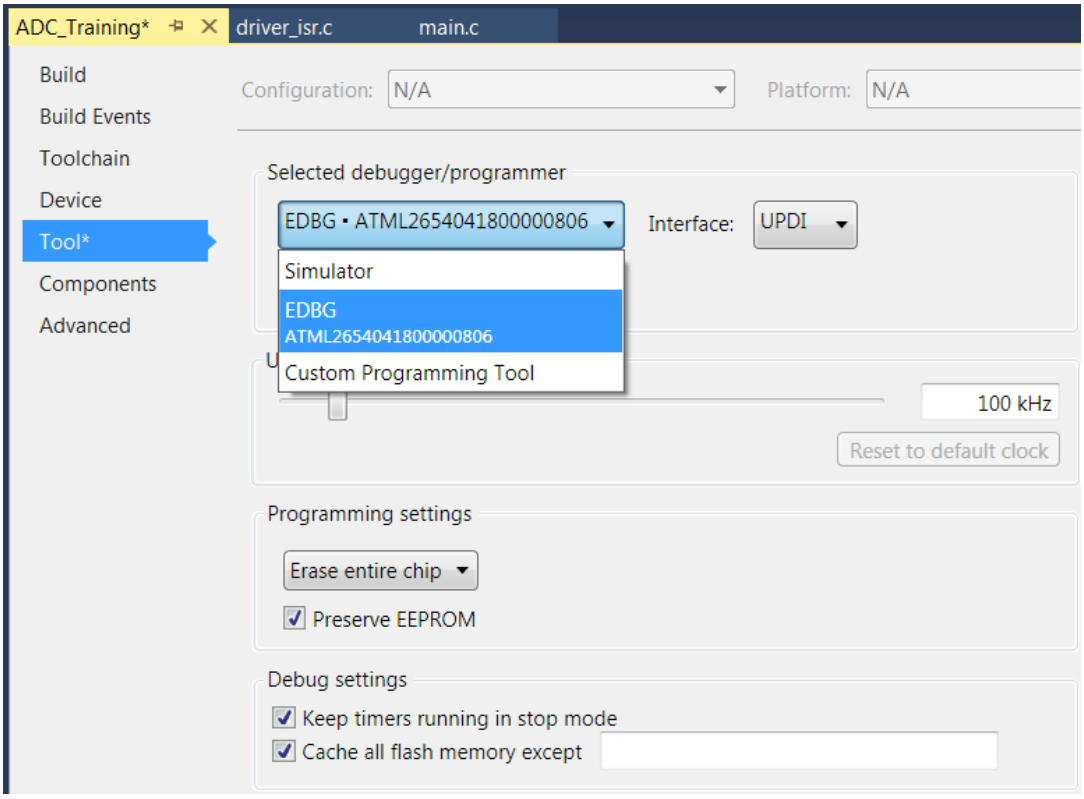
6. Program the device on ATtiny817 Xplained Pro by
   6.1. Open the *Tool* pane in project properties by clicking *No Tool*, as shown in Figure 3-16.

Figure 3-16. Tool Button



   6.2. Choose *EDBG* as the *Selected debugger/programmer* as shown in Figure 3-17.

Figure 3-17. Debugger Selection



6.3. Program the device by clicking *Debug->Start Without Debugging* or by using the shortcut *Ctrl+Alt+F5*.

Info: *Start Without Debugging* will build the project and program the device as long as there are no build errors.

Tip: The device can also be programmed by using the *Device Programming* menu, by clicking *Tools → Device Programming*.

Result: Hardware setup is completed and the application code is running on ATtiny817 Xplained Pro.

3.5 Observe ADC Functionality and Current Consumption in Data Visualizer

Once the application code has been developed and the hardware has been set up, it is time to run the application and to set up Data Visualizer in order to observe the ADC functionality and the power measurement.

To do:

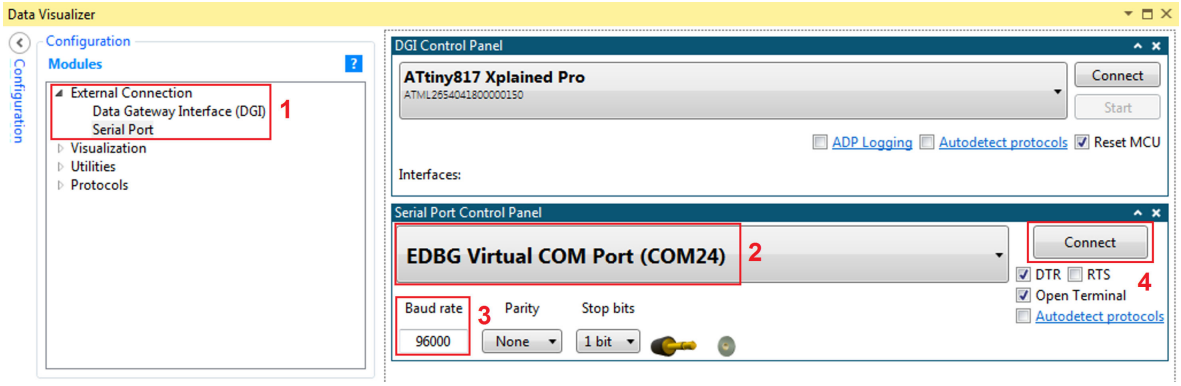
- Set up the *Data Visualizer* terminal in *Atmel Studio* to display the ADC data transmitted from the evaluation kit via the EDBG Virtual COM Port, and observe the ADC functionality.
- Set up the power analysis in Data Visualizer and observe the microcontroller's current consumption in *Data Visualizer*.

1. Open the *Data Visualizer* in *Atmel Studio* by clicking *Tools* → *Data Visualizer*.
2. Connect *ATtiny817 Xplained Pro* to *Data Visualizer*, refer to Figure 3-18, by
   2.1. Open *Serial Port Control Panel* by clicking *Configuration* → *External Connection* → double-click *Serial Port*.
   2.2. Select *ATtiny817 Xplained Pro*'s Virtual COM port by selecting *EDBG Virtual COM Port* from the drop-down list.

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.*

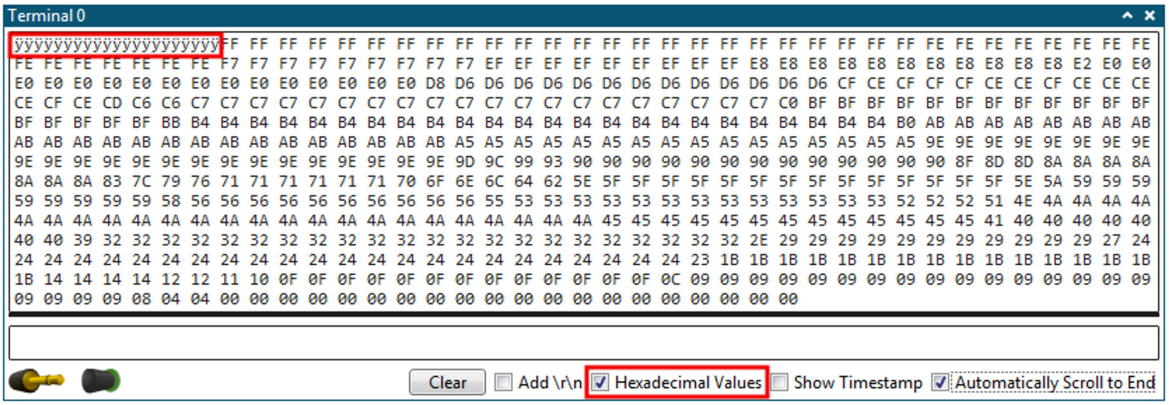
   2.3. Set the Baud rate to 96000 in the *Baud rate* field.
   2.4. Open the terminal window by clicking *Connect* in the *Serial Port Control Panel*.

Figure 3-18. Connect to Data Visualizer



3. Ensure human-readable print-out from the terminal window, by selecting *Hexadecimal Values*, see Figure 3-19
4. Observe the ADC functionality by seeing how the values in the terminal window changes as the potmeter knob is rotated, as shown in Figure 3-19.

Figure 3-19. ADC Results in Terminal Window



Info: As the ADC resolution has been configured to 8 bits, the ADC result printed to the terminal window will range from 0x00 to 0xFF, when the potentiometer knob is rotated.

5. Open the *Power Analysis* window in *Data Visualizer*, as shown in Figure 3-20, by

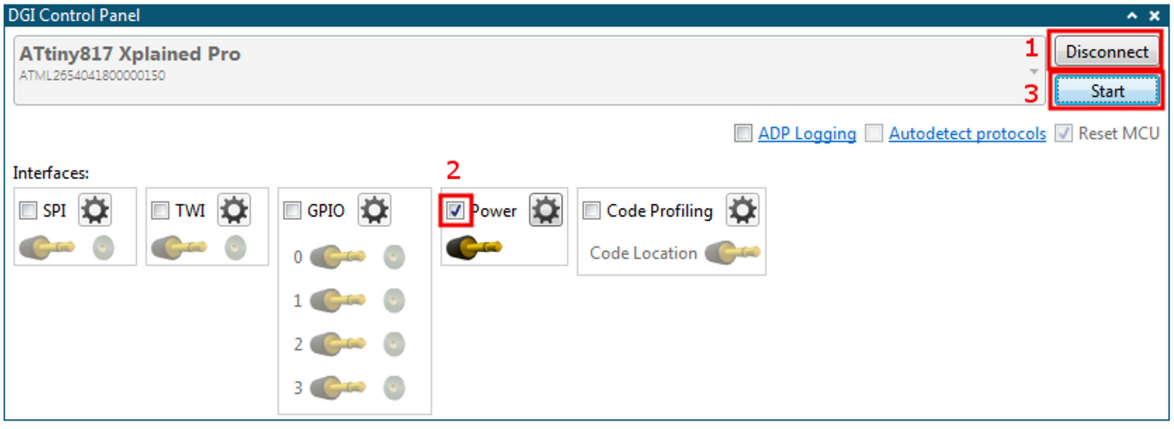
5.1. Connect *ATtiny817 Xplained Pro* to the data gateway interface by clicking *Connect* in *DGI Control Panel* by pressing the *Connect* button.

Info: This button will show as *Disconnect* once *ATtiny817 Xplained Pro* is connected.

5.2. Enable for power data being sent from *ATtiny817 Xplained Pro* to the data gateway interface by selecting *Power* in *DGI Control Panel*.

5.3. Start receiving data through the data gateway interface by clicking *Start*.

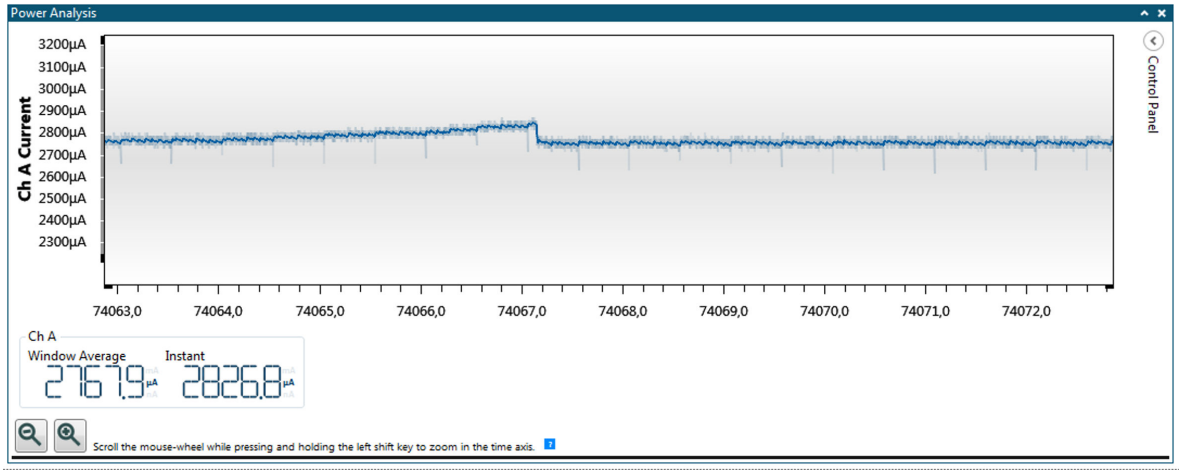
Figure 3-20. DGI Control Panel



Info: A *Power Analysis* window should appear in *Data Visualizer*.

6. Observe the current consumption for *ATtiny817* in *Power Analysis*, as shown in Figure 3-21.

Figure 3-21. Power Analysis Window



Info: As shown in Figure 3-21, the average current consumption of *ATtiny817* is around 2.8 mA, with small variations when rotating the potentiometer knob.

Note: The power consumption may vary depending on the value of the potentiometer connected (e.g. the current consumption can be 2.1 mA with another potentiometer). The power consumption measured in the following assignments could also vary slightly.

Info: The power consumption is high compared to the performance of *ATtiny817*. The following assignments will use different techniques which will reduce the power consumption.

Result: By connecting *ATtiny817 Xplained Pro* to *Data Visualizer* one can easily observe the ADC functionality through a terminal window, and by using the current measurement features of *ATtiny817 Xplained Pro*, the device's consumption can be observed.

4. Assignment 2: RTC Interrupts Triggers ADC and USART Print

In this assignment, the *Real Time Counter* (RTC) module will be used. The RTC overflow interrupt will be used to trigger an ADC conversion every half second. ADC Result Ready (RESRDY) interrupt will trigger a print of the ADC result to the USART terminal. When RTC overflow interrupt is not triggered, the device is kept in Sleep Standby mode in order to reduce the power consumption.

Atmel | START will be used to add the RTC module and to configure the RTC, ADC, CPUINIT, and SLEEPCTRL drivers. An *Atmel Studio* project will be regenerated afterward.

Peripherals used:

- RTC
- ADC (reconfigured from previous assignment)
- USART (from previous assignment)
- CPUINIT
- SLPCTRL

Clock details:

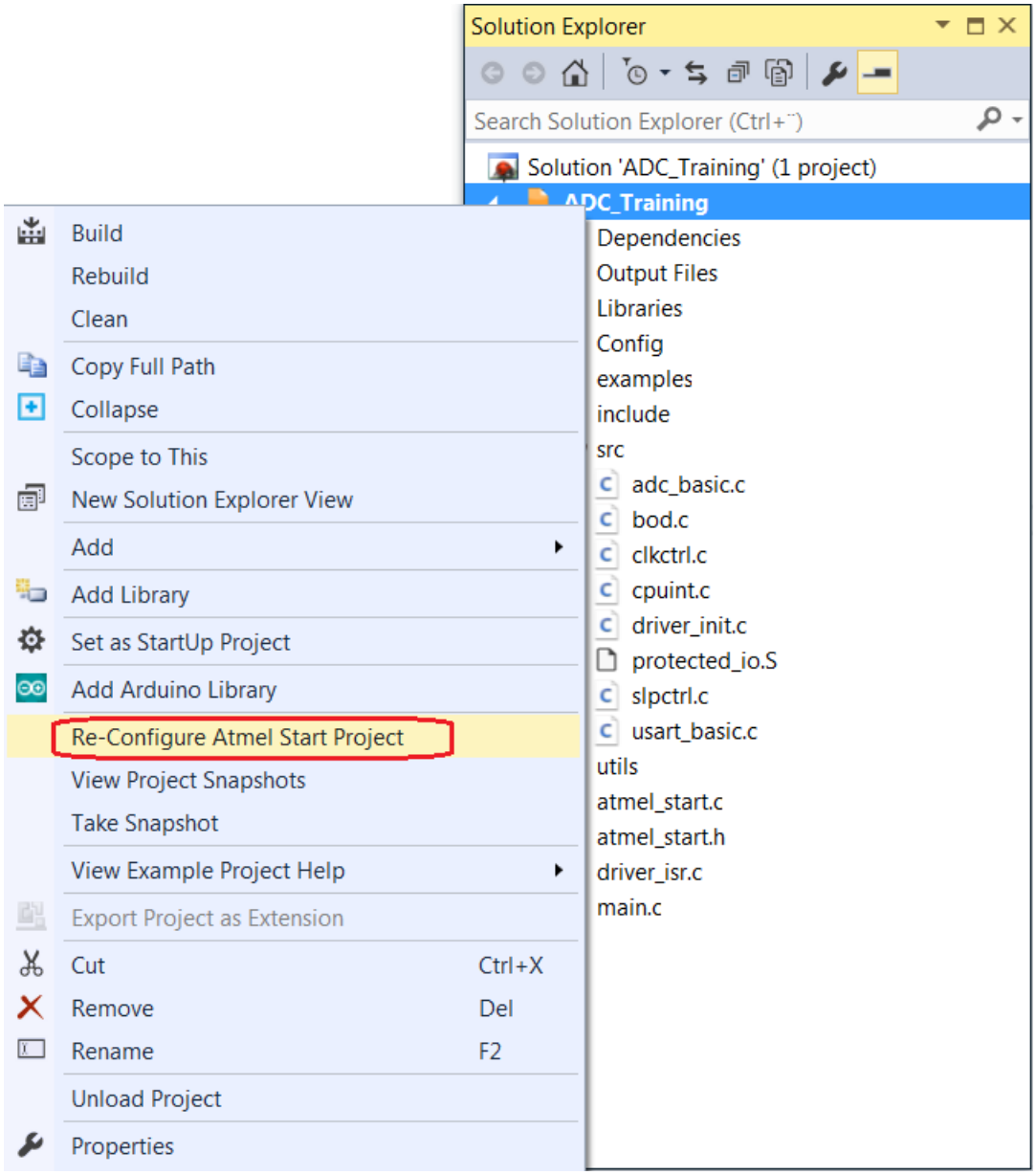
- CPU and USART - 5 MHz
- ADC - 625 KHz (5 MHz/8)
- RTC - 1 KHz

4.1 Add RTC Driver in *Atmel | START*

Add RTC drivers to the project in *Atmel | START* with the following steps:

1. Select the project by right-clicking the *ADC_Training* project in the *Solution Explorer* window from the previous assignment.
2. Reconfigure the project by clicking the *Re-Configure Atmel | START Project* option in the menu as shown in Figure 4-1.

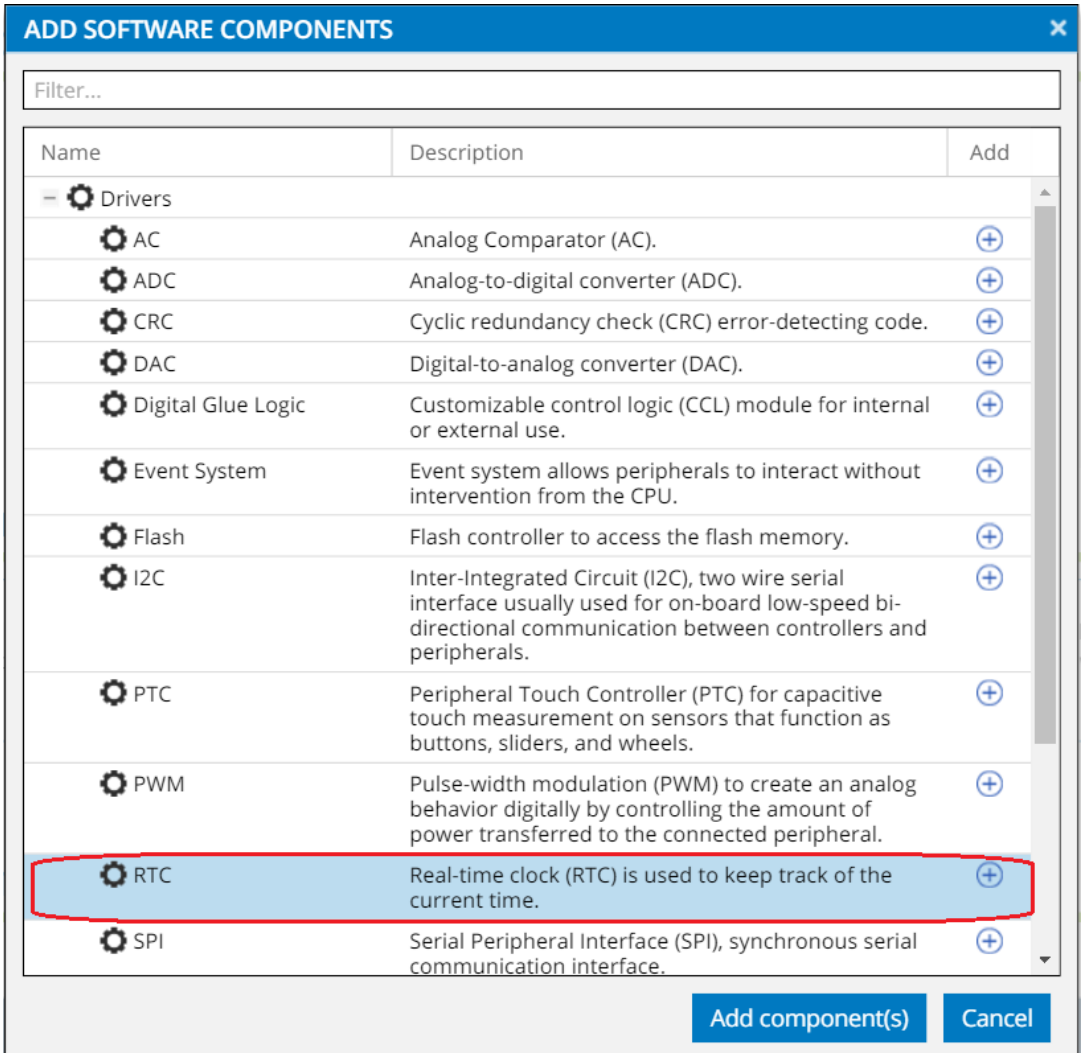
Figure 4-1. Reconfigure Atmel | START



Info: The *Atmel | START* window should appear within *Atmel Studio*.

3. Add the RTC component in *Atmel | START*:
   3.1. Expand *Drivers* from the *ADD SOFTWARE COMPONENT* window by clicking Add software component in the *Atmel | START* window.
   3.2. Select *RTC* by clicking ⊕ as shown in Figure 4-2.

Figure 4-2. Add RTC Component in Atmel | START



3.3. Add the selected RTC component by clicking Add component(s).

Info: The RTC module will be added to the ADC and Power Optimization project.

4.2 Configure RTC, CPUINIT, and SLEEPCTRL in *Atmel | START*

Once the RTC drivers have been added to ADC and Power Optimization, the RTC initialization function will be automatically called when the `atmel_start_init();` function in the *main.c* file is executed. The CPUINIT and SLEEPCTRL modules are also required to be configured for this assignment.

To do: Configure the RTC, CPUINIT, and SLEEPCTRL modules.

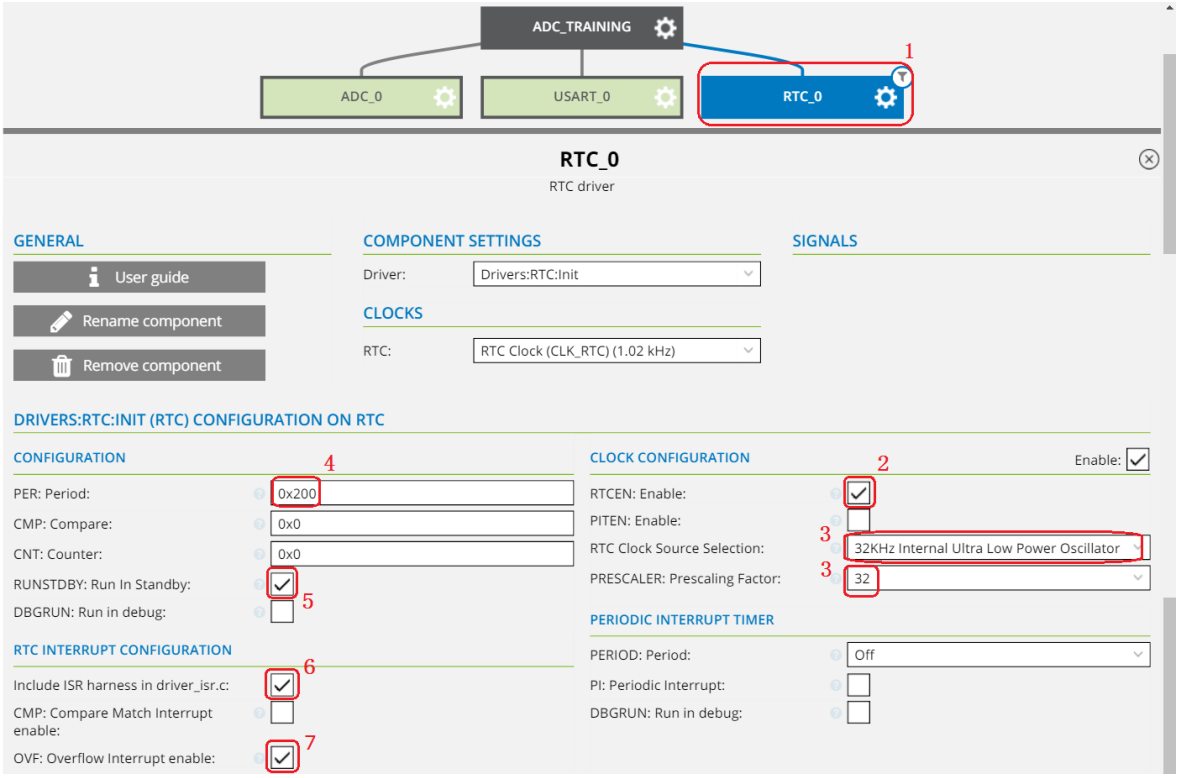
1. Configure the RTC driver by following the configuration steps, as marked with numbers in Figure 4-3.
   1.1. Open the RTC driver configuration page by clicking *RTC_0* in the *Atmel | START* window.
   1.2. Ensure the RTC is enabled after initialization by selecting RTC:Enable.
   1.3. Configure the RTC clock prescaler value by clicking the *PRESCALER* drop-down menu and select *32*.

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

   1.4. Entering 512, equivalent to heximal number 0x200, in the *PER* text box to define the RTC period.
   Note: It may not allow changing this field due to an *ATMEL | START* bug. It works by typing the value anywhere, and copying/pasting it in this field.
   1.5. Tick the *Run in Standby* checkbox to enable RTC to run in the Sleep Standby mode.
   1.6. Tick the *Include ISR harness in driver_isr.c* checkbox to include interrupt harness in *driver_isr.c*.
   1.7. Tick the *OVF* checkbox to enable the RTC overflow interrupt.

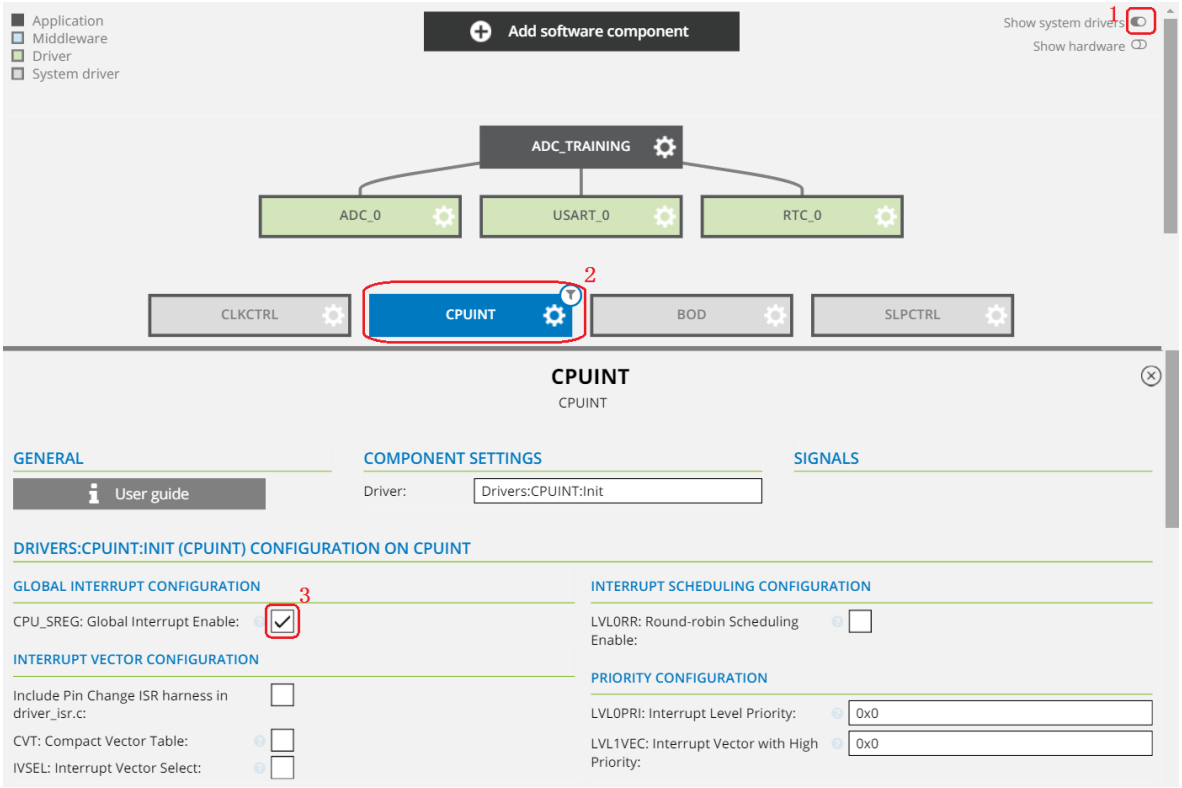
Info: The RTC overflow interrupt is configured to trigger twice per second since the period is set to 512 with an RTC clock frequency of 1 KHz.

Figure 4-3. RTC Configuration in Atmel | START



2. Enable CPUINT Global Interrupt in three steps as illustrated in Figure 4-4.
   2.1. Enable system driver visibility by toggling the *Show system drivers* slider in the top right corner of the "*DASHBOARD*"-view.
   2.2. Click *CPUINIT* to open the CPUINT configuration view.
   2.3. Tick the *CPU_SREG: Global Interrupt Enable* checkbox so that the initialization routine generated by *Atmel | START* will enable global interrupts.

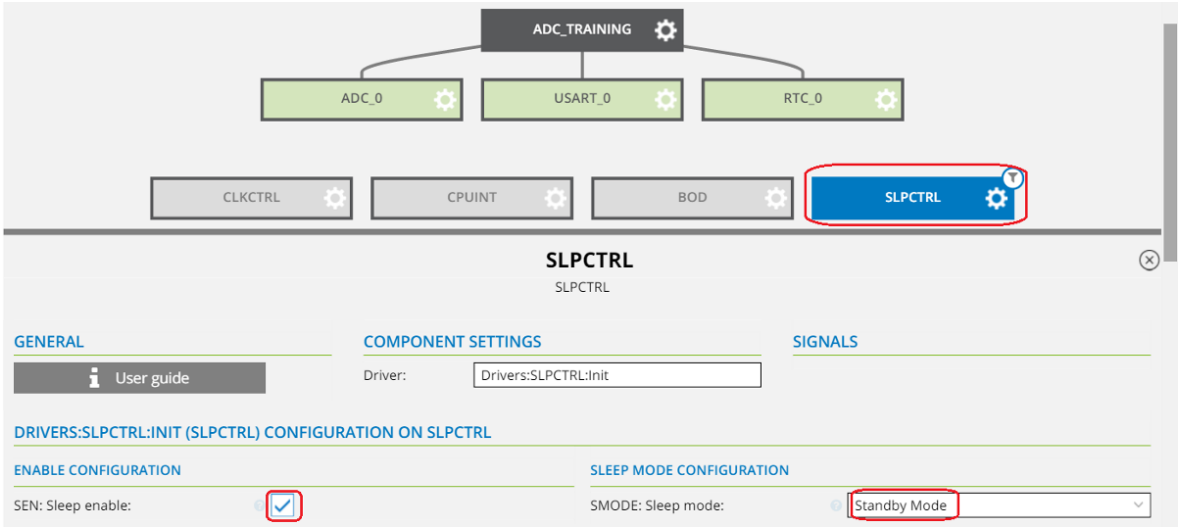
Figure 4-4. CPUINT Configuration



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

3. Configure *SLPCTRL* by clicking it in the *Atmel | START* window and following the configuration steps as marked in Figure 4-5.
   3.1. Tick the *SEN* checkbox to set sleep enabled.
   3.2. Select "Standby Mode" in "SMODE:Sleep mode" column.

Figure 4-5. SLPCTRL Configuration in Atmel | START



Info: Figure 4-6 is a snapshot from the data sheet and gives the Sleep mode overview. It shows that the RTC can be used as a wake-up source while the RTC and ADC clocks run in Standby Sleep mode. Standby mode is chosen for this assignment since it will result in much lower power consumption than in idle mode.

Figure 4-6. Sleep Mode Overview

Table 11-2. Sleep Mode Overview

Sleep Mode	Active clock domain						Oscillators			Wake-up Sources						
	CPU clock	Peripheral clock	WDT clock	RTC clock	TCB clock	ADC/PTC clock										
	CLK_CPU	CLK_PER	CLK_WDT	CLK_RTC	CLK_PER	CLK_PER	Main Clock source	WDT oscillator	RTC clock source	INTn and pin change	TWI address match	USART start of frame	PTC window	RTC interrupt	TCB	All other interrupts
Idle		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Standby			X	X*	X*	X*	X*	X	X*	X	X	X*	X*	X*	X	
Power Down			X					X		X	X					

Note: X means active. X* indicates that the RUNSTBY bit of the corresponding peripheral must be set to enter active state.

4. Generate the project by clicking GENERATE PROJECT.

Info: The project summary window should appear.

Figure 4-7. Summary of Project Code Generated by *Atmel | START*

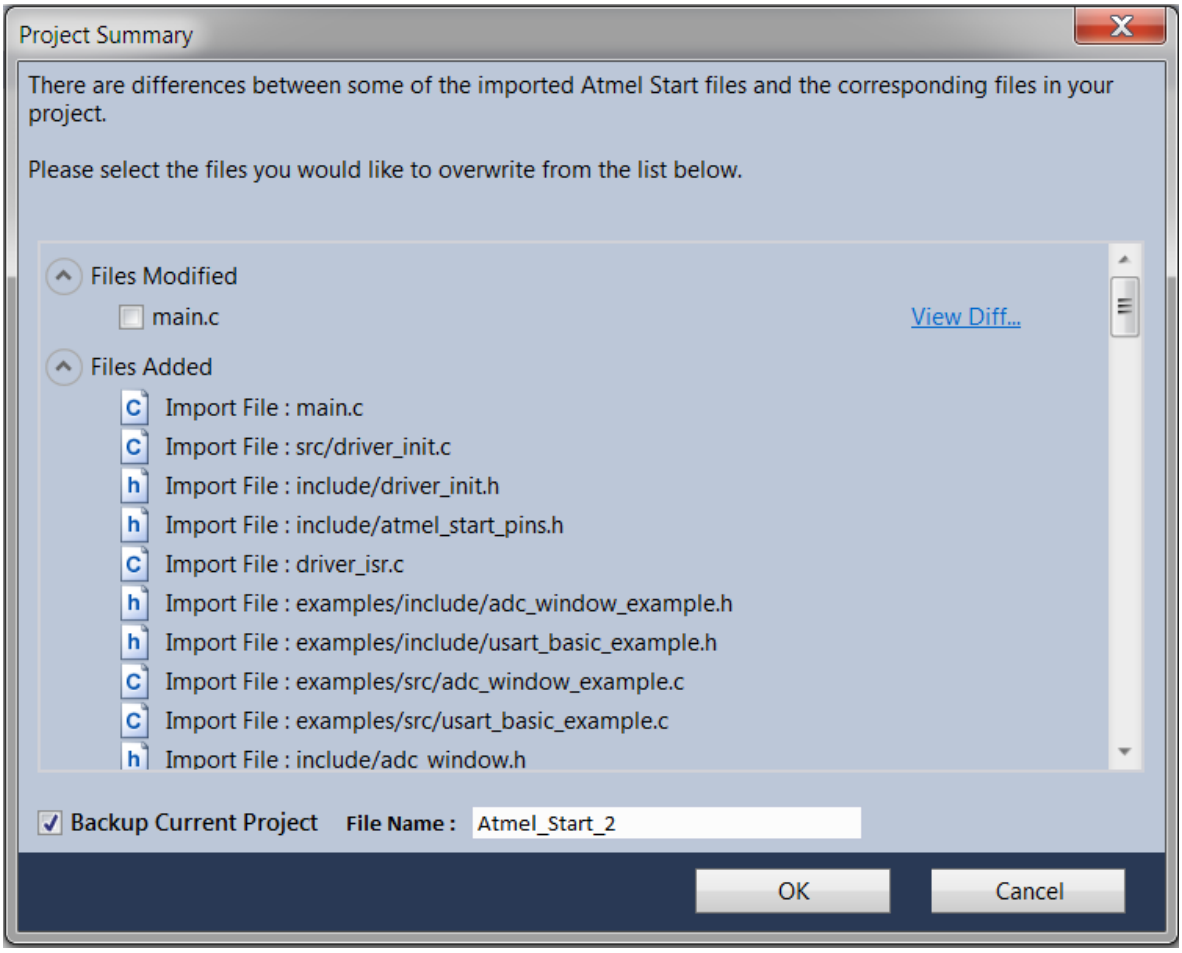


Figure 4-7 gives an overview of the files that have been changed between the original and the reconfigured Studio project. As shown, the only modified file is *main.c*.

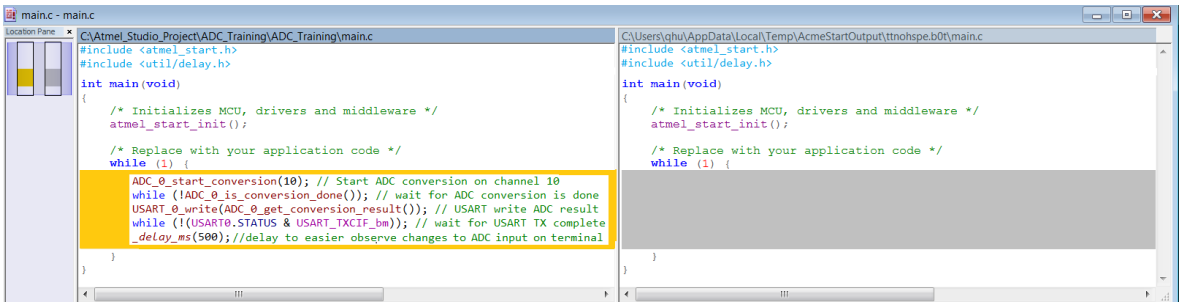
5. Compare the changes for the selected file by clicking *View Diff* for the only modified *main.c* file and an external installed tool, *WinMerge*, will be opened.

tip: 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 it at the default path (i.e. C:\Program Files (x86)\WinMerge) or a user-defined path
- In Atmel Studio, go to menu *Tools Options -> Atmel Start -> File Compare*. In "*Path of the application used for comparing files*", fill in `C:\Program Files (x86)\WinMerge\WinMergeU.exe`. In "*Command line arguments to be used for file comparison:*", fill in `%original %mine /s /u` and click *OK*.
- Click *View Diff...* as shown in Figure 4-7 and now it should work.

The difference for *main.c* between the version already present in *Atmel Studio* and the version re-generated by *Atmel | START* is compared as shown in Figure 4-8:

Figure 4-8. main.c Difference Between Atmel Studio Version and Regenerated by *Atmel | START*



6. Regenerate the project by clicking the OK button in *Atmel | START* without ticking the checkbox for the *main.c* file. The *main.c* file will be kept as it is without being overwritten by the *Atmel | START* regenerated version.

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

4.3 Add RTC, ADC, and SLPCTRL Functionality in Application Code

Once the RTC, CPUINIT, and SLPCTRL modules have been added and reconfigured using *Atmel | START*, the application code needs to be updated in *Atmel Studio*.

To do: Update the code in *Atmel Studio* to make use of these module drivers just added through *Atmel | START*. Specifically, update the *main.c* and the *driver_isr.c* file.

1. Update the *main.c* file:
   1.1. Include the sleep header file at the top of the file:

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

   1.2. Enable the Sleep Standby mode by adding the following piece of code in the main function after initialization:

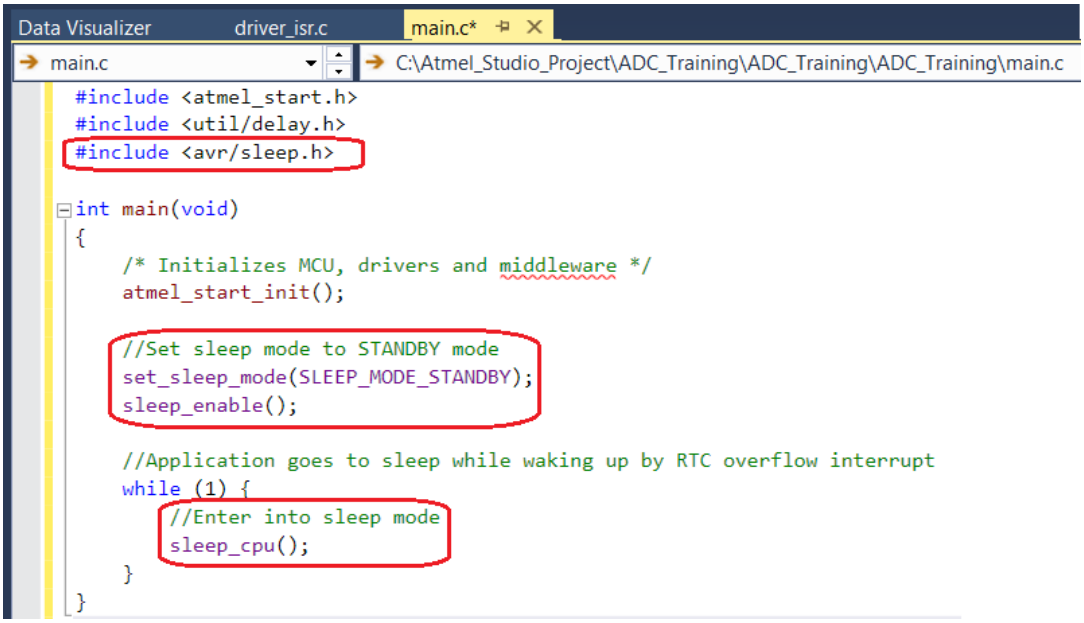
```
//Set sleep mode to STANDBY mode
set_sleep_mode(SLEEP_MODE_STANDBY);
sleep_enable();
```

   1.3. Set the device to Sleep mode by replacing the following piece of code in the *while-loop* of the *main* function:

```
//Enter into sleep mode
sleep_cpu();
```

   The complete code of the *main.c* file should look like:

Figure 4-9. main.c Code



2. Edit the *driver_isrc.c* file by double-clicking and open it in the *Solution Explorer* in *Atmel Studio*:

   2.1. Start ADC conversion when RTC interrupt is triggered by adding the following line of code in the RTC interrupt routine:

```
    /* RTC Overflow Interrupt handling */
    ADC_0_start_conversion(10); //start ADC conversion on channel 10
```

   2.2. Manually add the ADC result ready interrupt routine as:

```
ISR(ADC0_RESRDY_vect)
{
    /* ADC result ready Interrupt handling: start USART transmission */
    USART_0_write(ADC_0_get_conversion_result()); //USART write ADC result
    while(!(USART0.STATUS & USART_TXCIF_bm)); //wait for USART TX complete
    USART0.STATUS = USART_TXCIF_bm; //Clear TXCIF flag

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

   In the ADC result ready interrupt routine, the ADC result ready interrupt will trigger for the ADC result to be sent over USART.

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

Figure 4-10. driver_isr.c Code

Data Visualizer | driver_isr.c* | main.c*

driver_isr.c | C:\Atmel_Studio_Project\ADC_Training\ADC_Training\ADC_Training\driver_isr.c

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

ISR(RTC_CNT_vect)
{
    /* RTC Overflow Interrupt handling */
    ADC_0_start_conversion(10); //start ADC conversion on channel 10

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

ISR(ADC0_RESRDY_vect)
{
    /* ADC result ready interrupt handling: start USART transmission */
    USART_0_write(ADC_0_get_conversion_result()); //USART write ADC result
    while(!(USART0.STATUS & USART_TXCIF_bm)); //wait for USART TX complete
    USART0.STATUS = USART_TXCIF_bm; //Clear TXCIF flag

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

Program the device by clicking *Debug → Start without Debugging* on the top menu window or by using the shortcut *Ctrl+Alt+F5*.

Info: *Start Without Debugging* will build the project and program the device as long as there are no build errors.

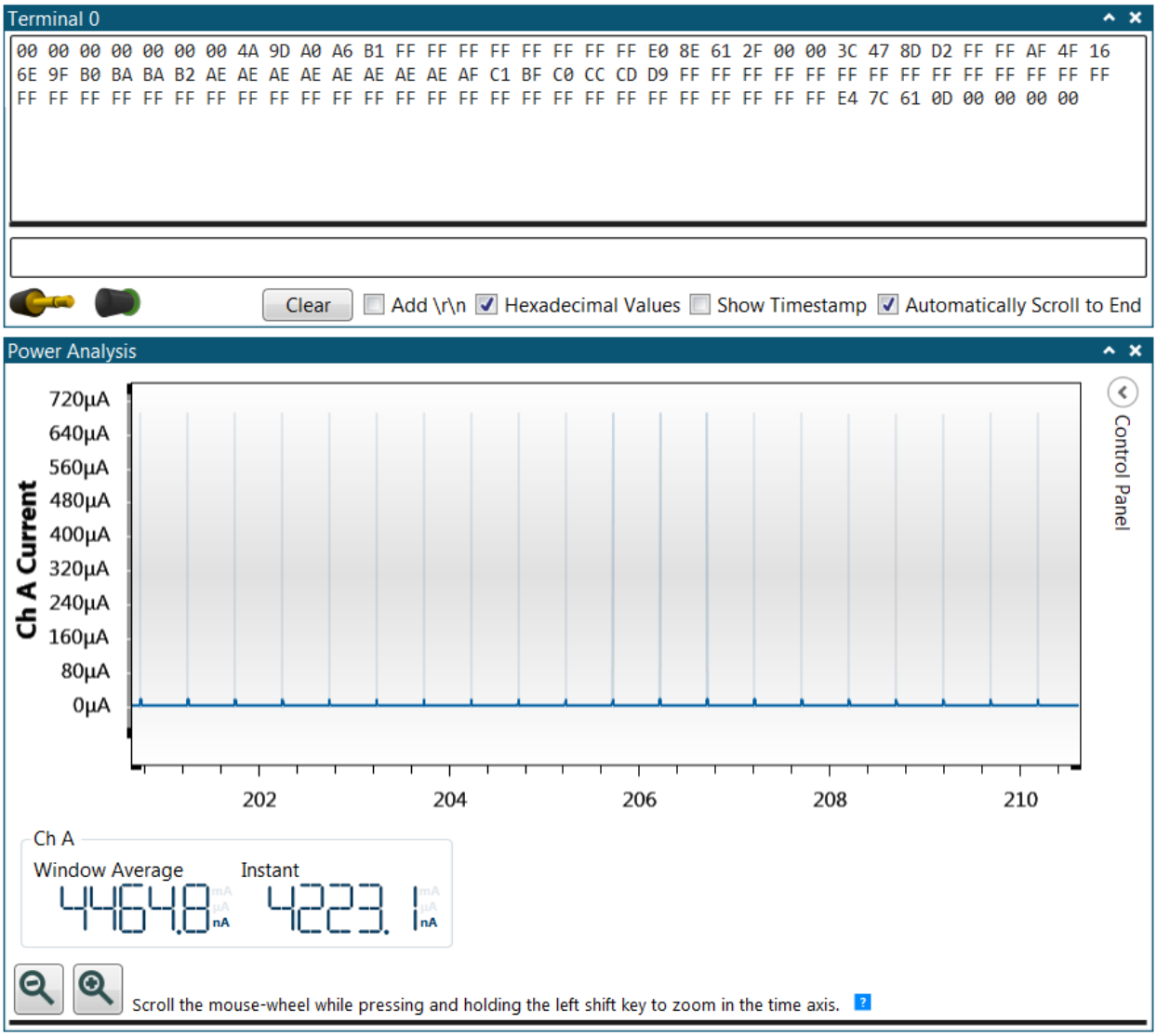
4.4 Observe ADC Functionality and Power Consumption in Data Visualizer

The device has been programmed to start ADC conversion when RTC interrupt is triggered. When one ADC conversion is completed, the ADC result will be printed to the USART terminal. The power consumption on the device can be observed in embedded power debugger in Data Visualizer.

To do: Observe ADC functionality and power consumption in *Data Visualizer*.

Click the *Data Visualizer* tab to observe the ADC functionality and power consumption, as shown in Figure 4-11.

Figure 4-11. ADC Data Print to USART Terminal and Power Analysis



The ADC result is printed as shown in the terminal, in the same manner as in the previous assignment. When rotating the potmeter knob, the print ADC result changes accordingly.

Result: By rotating the potentiometer knob, the voltage input to ADC input pin, PB1, is changing. This can be observed as the ADC result being sent to the terminal window is changing as one rotates the knob.

When looking at the current consumption in the power analysis window, the average current is about 4.5 µA. This is a significant reduction from the 2.8 mA current consumption in the previous assignment. The high peak glitches are about 690 µA, which refers to the case when doing the ADC conversion and USART transmission to the terminal. This scenario consumes most power but is still a significant drop from 2.8 mA.

Result: When using the RTC overflow interrupt and Standby Sleep mode, the average current consumption can be reduced from 2.8 mA from previous assignment down to the 4.5 µA. The peak glitches when doing the ADC conversion and USART transmission has also been reduced from 2.8 mA to 690 µA.

5. Assignment 3: Power Optimization on I/O Pins

In this assignment, the digital input buffer on I/O pins will be disabled in order to reduce the current consumption. The current consumption will further be reduced when the USART TX pin is configured as a high impedance pin during no data transmission period.

The same drivers and configurations from the previous assignment will be used in this assignment. *Atmel Studio* will be used to further develop the code.

5.1 Application Code Update in *Atmel Studio*

To do:

- Update *main.c* to disable the digital input buffer on all I/O pins.
- Update *driver_isr.c* to configure the USART TX pin as output pin during data transmission and as input pin during no data transmission period.

1. Edit the *main.c* file:
   1.1. Disable the digital input buffer on all I/O pins by adding in the following function before the `main()` function.

```
//Disable digital input buffer on all IO pins
void io_init(void)
{
    for (uint8_t pin=0; pin < 8; pin++)
    {
        (&PORTA.PIN0CTRL)[pin] = PORT_ISC_INPUT_DISABLE_gc; //Disable on PAx pin
        (&PORTB.PIN0CTRL)[pin] = PORT_ISC_INPUT_DISABLE_gc; //Disable on PBx pin
        (&PORTC.PIN0CTRL)[pin] = PORT_ISC_INPUT_DISABLE_gc; //Disable on PCx pin
    }
}
```

Info: *PORT_ISC_INPUT_DISABLE_gc* will set the digital input buffer disabled. Hover over it and *right-click ->Goto Implementation* will open the *iotn817.h* file and point to where it is defined.

   1.2. Call the function by adding the following code after the *atmel_start_init();* line in *main()* function.

```
//Disable digital input buffer on all IO pins
io_init();
```

2. Edit the *driver_isr.c* file:
   2.1. Add the following line of code at the start of the `ISR(ADC0_RESRDY_vect)` function:

```
VPORTB.DIR |= PIN2_bm; //Configure USART TX pin PB2 as output pin
```

   2.2. Add the following line of code after the USART transmission is completed in the `ISR(ADC0_RESRDY_vect)` function:

```
VPORTB.DIR &= ~PIN2_bm; //Configure USART TX pin PB2 as input
```

Info: This will configure the USART TX pin as an input. The reason behind is that, when the USART TX pin is configured as an output, there will be a current flow between the on-chip TX pin and the onboard EDBG pin since there are supply voltage variations between them. By configuring it as input while there is no data transmission, unnecessary current flow is avoided.

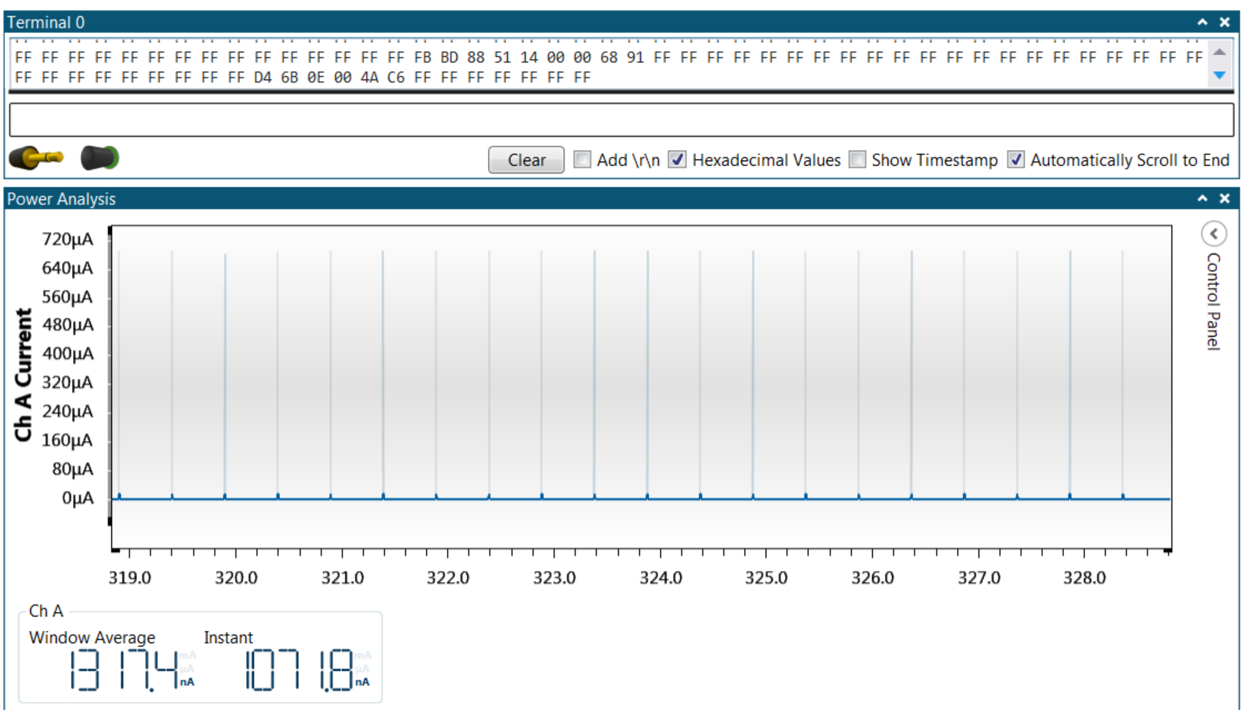
3. Program the device by clicking *Debug → Start without Debugging* on the top menu window or by using the shortcut *Ctrl+Alt+F5*

Info: *Start Without Debugging* will build the project and program the device as long as there are no build errors.

5.2 Observe ADC Functionality and Current Consumption in Data Visualizer

Open the *Data Visualizer* by clicking the *Data Visualizer* tab to observe the ADC functionality and power consumption, as shown in the figure below.

Figure 5-1. ADC Data Print to USART Terminal and Power Analysis



As shown in terminal 0 in Figure 6-5, the ADC result is printed in the same manner as in previous assignment. When rotating the potmeter knob, the print ADC result changes in the same manner. The functionality of this assignment remains the same as the previous one.

When looking at the current consumption in the power analysis window, the average current is constant 1.32 µA when rotating potmeter knob. In comparison, the average current consumption is about 4.5 µA in the previous assignment. There are two factors which contribute to this current consumption reduction:

- The average current consumption is around 4.0 µA when the digital input buffer on all I/O pins is disabled.
- The average current is further reduced from 4.0 µA to 1.32 µA when the USART TX pin is configured as input during the no ADC data transmission period and only configured as output during the ADC data transmission.

This is a significant reduction of the average current from 3.7 µA to 109 µA in the previous assignment to 1.3 µA in this assignment. The high peak glitches are about 690 µA, the same level as in the previous assignment. It refers to the current consumed for the ADC conversion and USART transmission.

Result: Average current consumption has been reduced significantly from 4.5 µA in the previous assignment to 1.32 µA by disabling the digital input buffer on all I/O pins and by configuring the USART TX pin as input during the no data transmission period.

6. Assignment 4: ADC Conversion Using Window Compare Mode

In this assignment, the ADC result ready interrupt will be replaced by the ADC WCMP interrupt, to trigger a USART transmission. In this case, the ADC result, which is below ADC window threshold value, will trigger USART transmission. The ADC results, which are above the window threshold value, will be ignored and not trigger any USART transmission.

Atmel | START will be used to reconfigure the ADC module, and the *Atmel Studio* project will be updated with the new configuration.

6.1 ADC Reconfiguration in *Atmel | START*

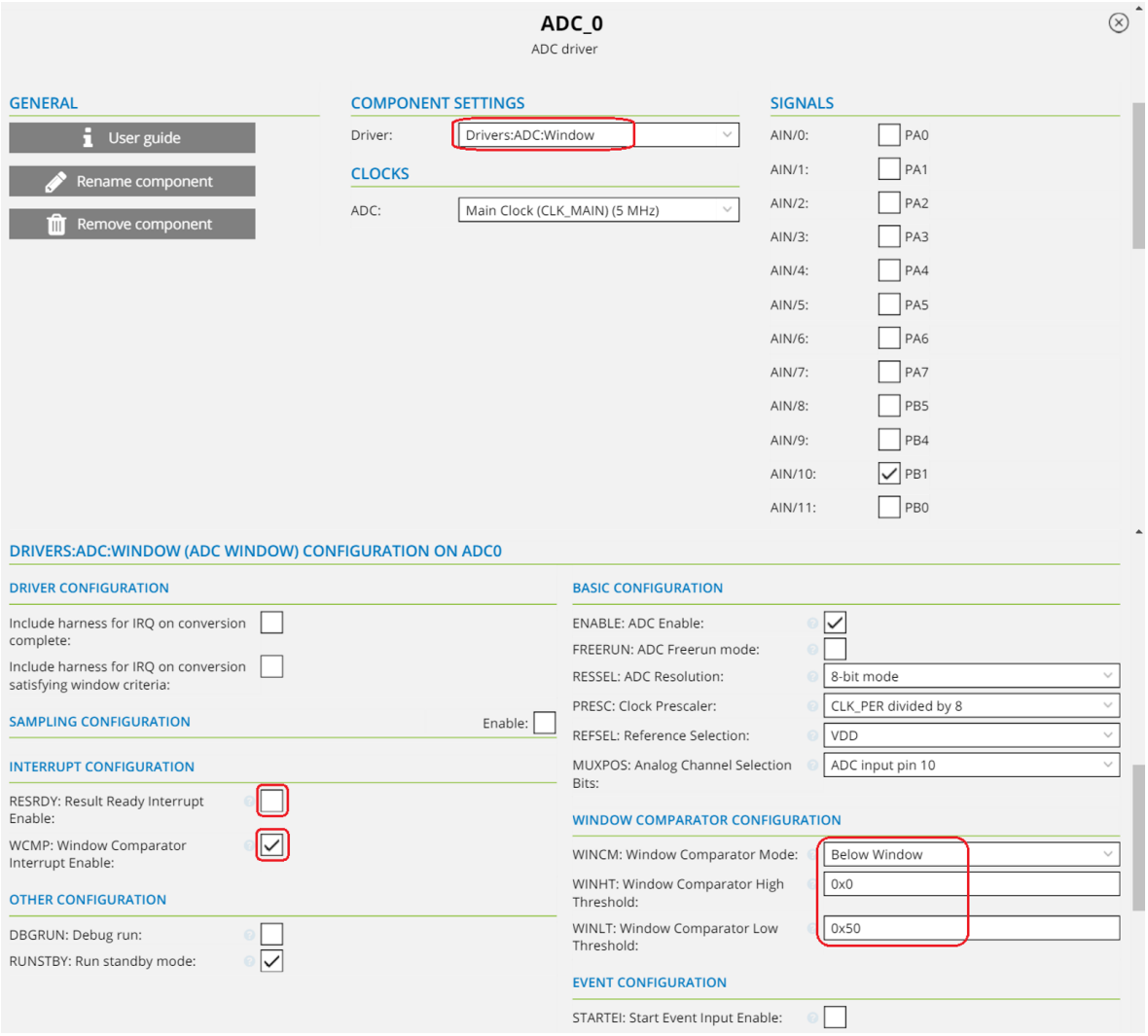
To do:

- Reconfigure the ADC module with WCMP mode configuration in *Atmel | START*.

Reconfigure the project in *Atmel | START*:

1. Select the project by right-clicking the *ADC_Training* project in the *Solution Explorer* window from the previous assignment.
2. Reconfigure the project by clicking the *Re-Configure Atmel Start Project* option in the menu, the same way as described in the previous assignment.
3. Reconfigure the ADC module by clicking the *ADC* component in the *Atmel | START* window and then configuring it as shown with red markings in Figure 6-1.

Figure 6-1. ADC Reconfiguration in START



3.1. Select *Drivers:ADC:Window* for window compare mode drivers in the *driver* column.

3.2. Unselect the checkbox for the *RESRDY: Result Ready Interrupt Enable* field.

3.3. Select the checkbox for the *WCMP: Window Comparator Interrupt Enable* field as it now replaces the just unselected RESRDY interrupt.

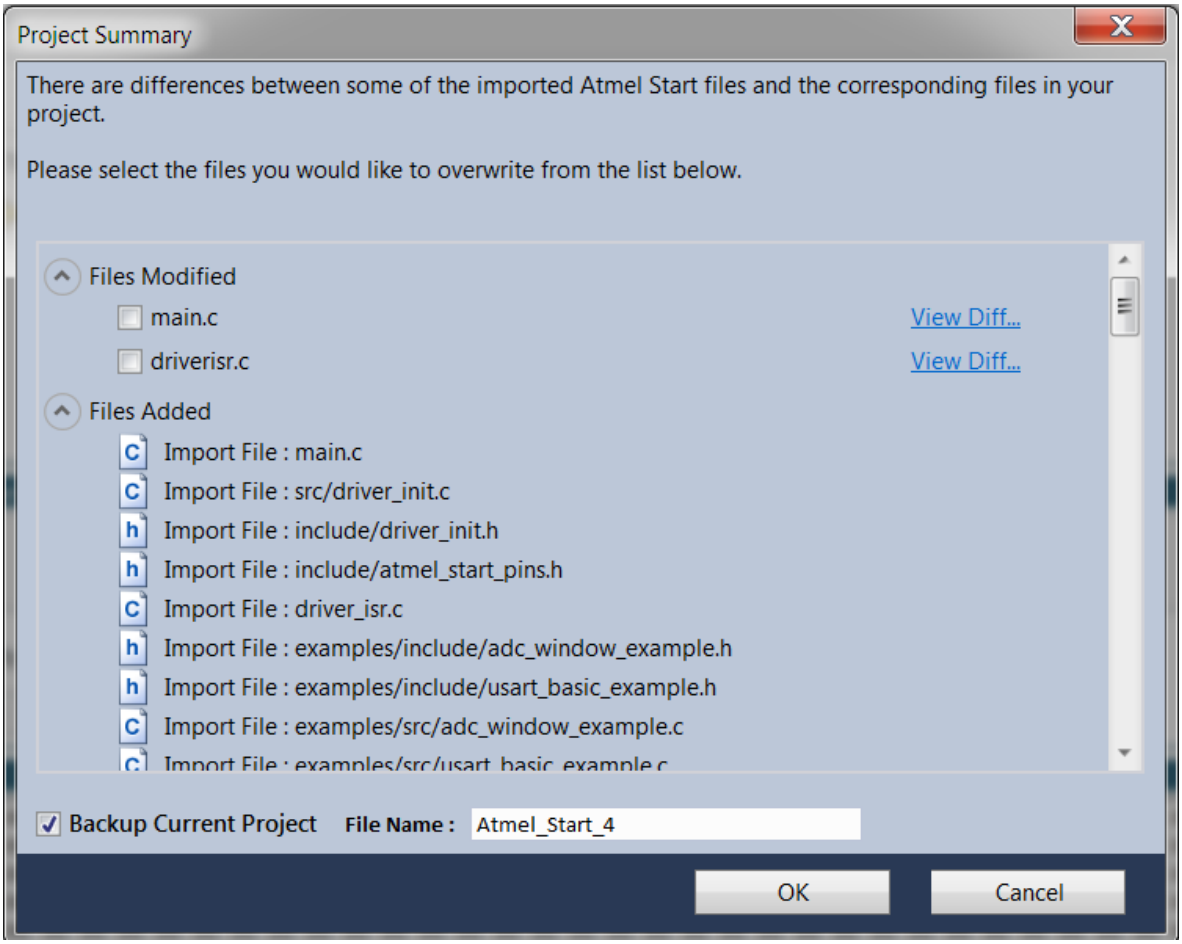
3.4. Select *Below Window* from the *WINCM: Window Comparator Mode* drop-down menu.

3.5. Enter 80 (0x50 hexadecimal value) in the *WINHT: Window Comparator Low Threshold* field.

4. Regenerate the project by clicking the *GENERATE PROJECT* button at the bottom of the window.

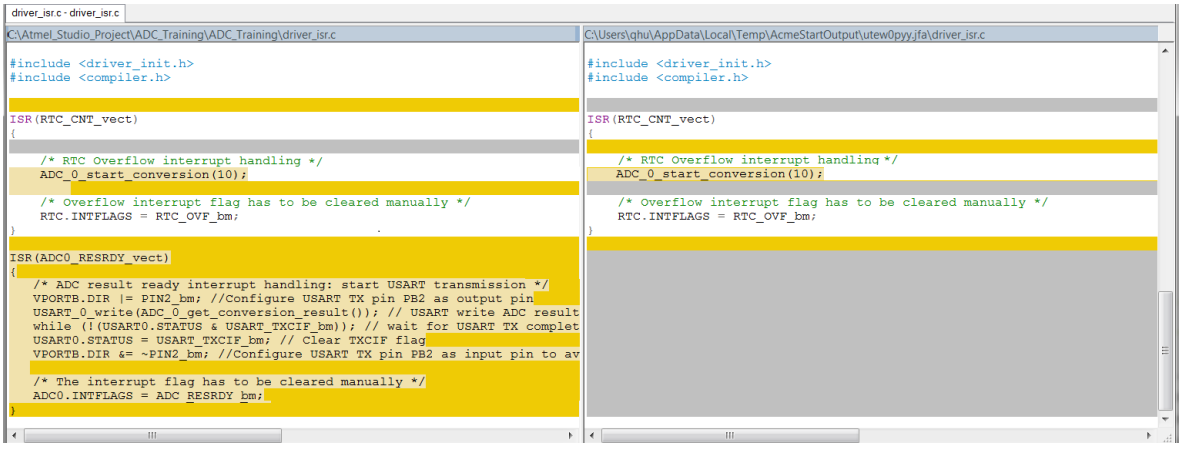
Info: The project summary window should appear as shown in Figure 6-2.

Figure 6-2. Project Code Regenerated



5. For *driver_isr.c*, click *View Diff* to see the difference between the previous and the newly generated project as shown in Figure 6-3. The ADC interrupt routine needs to be updated.

Figure 6-3. Driver_isr.c View Diff



6. Update the project by clicking the *OK* button.

Info: The *main.c* and *driver_isr.c* are not selected and need to be manually updated afterward.

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

6.2 Update Application in Atmel Studio

To do: Update *driver_isr.c* to replace the *ISR(ADC0_RESRDY_vect)* routine with the *ISR(ADC0_WCOMP_vect)* routine.

Following the red markings in Figure 6-4:

1. Replace the *ISR(ADC0_RESRDY_vect)* function by the *ISR(ADC0_WCOMP_vect)* function.
2. Replace the interrupt flag source *ADC_RESRDY_bm* by *ADC_WCMP_bm*.

Figure 6-4. Driver_isr.c Updates

driver_isr.c | main.c | Data Visualizer

driver_isr.c — C:\Atmel_Studio_Project\ADC_Training\ADC_Training\driver_isr.c

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


ISR(RTC_CNT_vect)
{
    /* RTC Overflow Interrupt handling: */
    ADC_0_start_conversion(10); //start ADC conversion on channel 10

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

ISR(ADC0_WCOMP_vect)
{
    /* ADC result ready interrupt handling: start USART transmission */
    VPORTB.DIR |= PIN2_bm;  //configure USART TX pin PB2 as output pin
    USART_0_write(ADC_0_get_conversion_result()); //USART write ADC result
    while(!(USART0.STATUS & USART_TXCIF_bm)); //wait for USART TX complete
    USART0.STATUS = USART_TXCIF_bm; //Clear TXCIF flag
    VPORTB.DIR &= ~PIN2_bm; //configure USART TX pin PB2 as input pin to avoid current flow to onboard EDBG pin

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

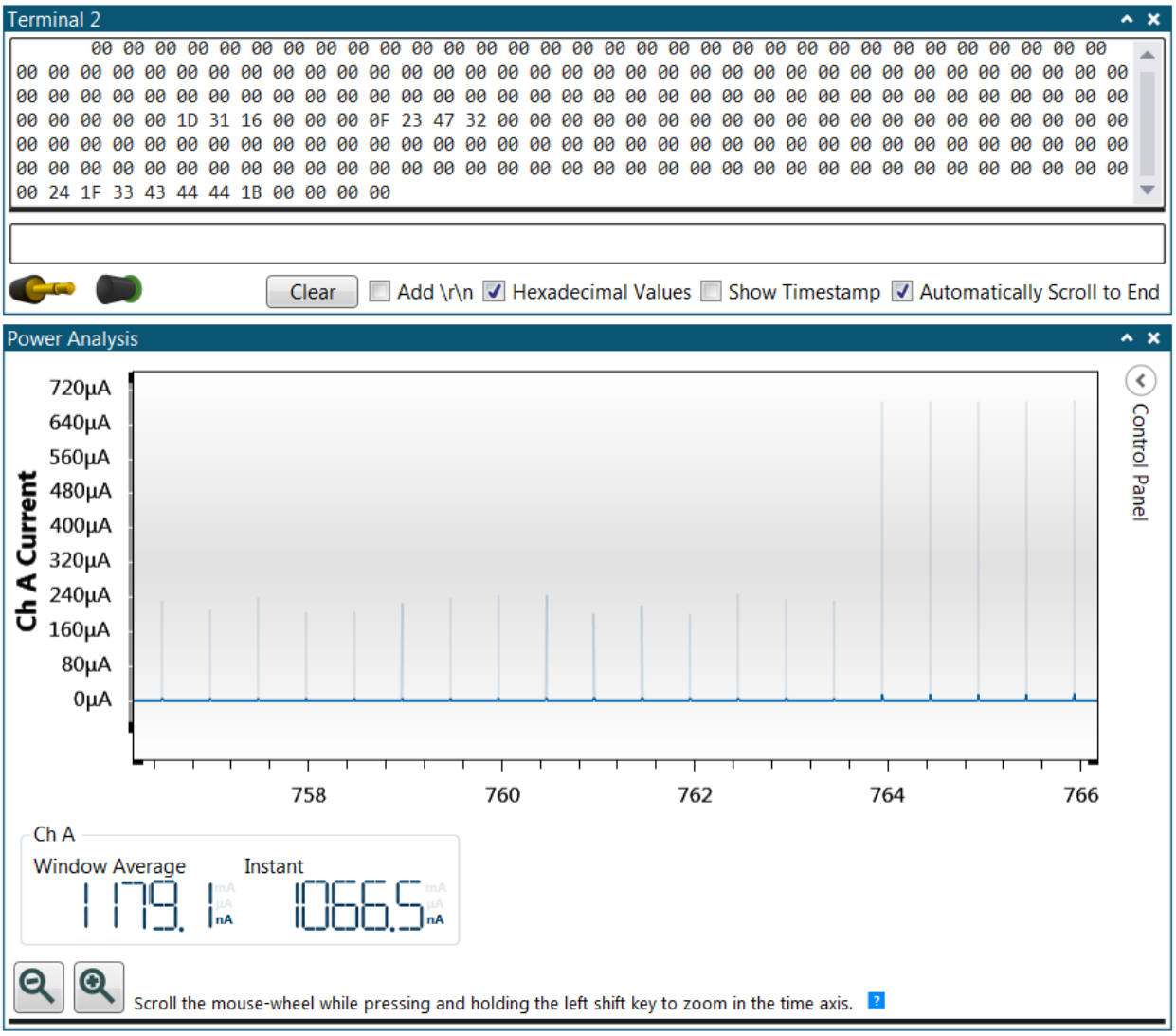
Program the device by clicking *Debug → Start without Debugging* on the top menu window or by using the *Ctrl+Alt+F5* shortcut.

Info: *Start Without Debugging* will build the project and program the device as long as there are no build errors.

6.3 Observe ADC Functionality and Current Consumption in Data Visualizer

Open *Data Visualizer* by clicking the *Data Visualizer* tab to observe the ADC functionality and look at the power consumption, as shown in Figure 6-5.

Figure 6-5. ADC Data Print to USART Terminal and Power Analysis for ADC WCMP Mode



By rotating the potentiometer knob, the voltage input to ADC input pin, PB1, is changing. The ADC result is printed to the terminal only when it is below 80, which is the ADC window low threshold value. When the value is above that, ADC WCMP interrupt will not be triggered and hence no data will be printed to the terminal.

In the power analysis window, when looking at the current consumption, the average current consumption is about 1.18 µA which is slightly reduced from 1.32 µA in the previous assignment. This is because the ADC WCMP mode is now used. There are now two categories of the high peak glitches:

- When the RTC overflow interrupt triggers to start the ADC conversion while the ADC result is above the ADC window low threshold value and the USART transmission is NOT performed. This is illustrated in the first half of the power analysis graph, with a current consumption around 240 µA. This has been reduced from constant 690 µA in the previous assignment at where no window filter feature is available.
- When the RTC overflow interrupt triggers to start the ADC conversion while the ADC result is below the ADC window low threshold value and the USART transmission is performed. This is illustrated in the second half of the power analysis graph, with a current consumption around 690 µA. This is equivalent to the peak glitches in the previous assignment.

As shown, this application with implementing the ADC WCMP mode can reduce the peak current consumption, when the ADC result is out of interest and no USART transmission, is triggered.

Result: The ADC WCMP mode slightly reduces the average current consumption. It mainly reduces the current consumption for the scenario when the ADC result is above the window threshold value and no USART transmission triggers. The benefit varies depending on how often this scenario happens. Note that there are four different window compare modes available to fit the user's requirement.

7. Assignment 5: Event System (EVSYS) Used to Replace the RTC Interrupt Handler

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

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

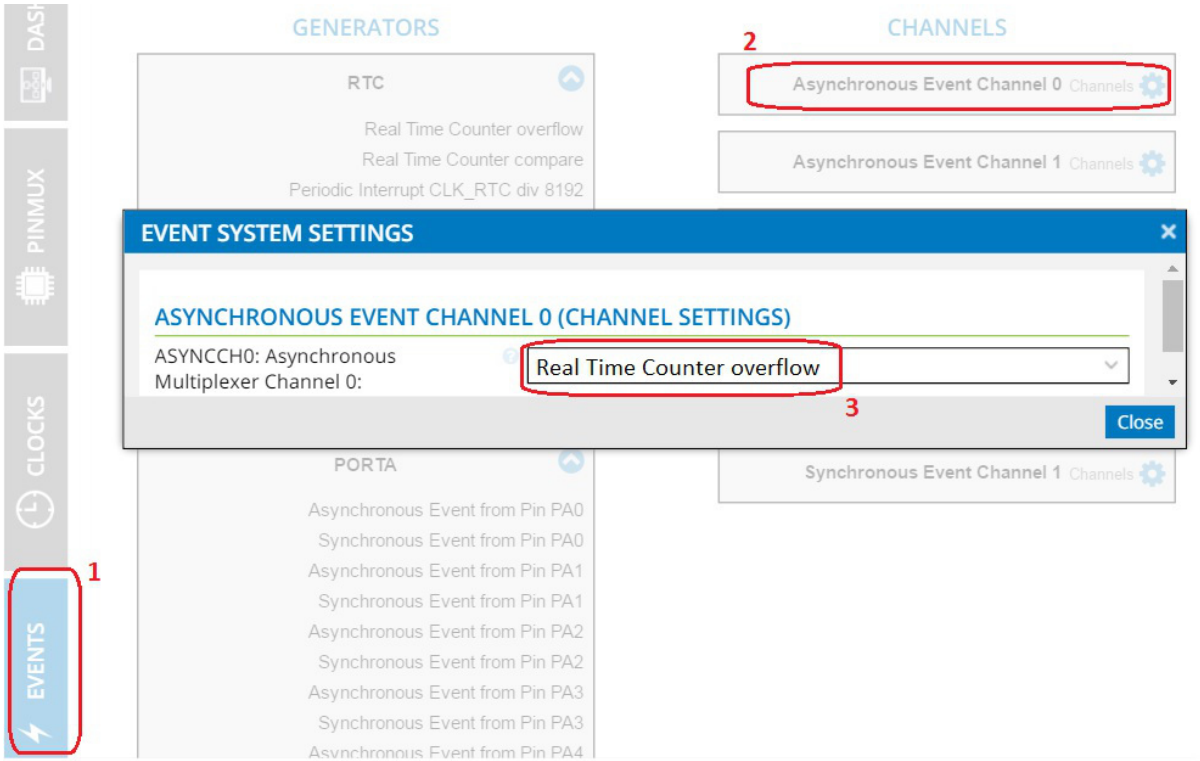
7.1 Event System Configuration in *Atmel | START*

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

1. Select *Re-Configure Atmel Start Project* by right-clicking the *ADC_Training* project in the *Solution Explore* window from the previous assignment.
2. Add in the *Events System* component in the opened *Atmel | START* window:
   2.1. Expand the driver from the *ADD SOFTWARE COMPONENT* window by clicking Add software component in the *Atmel | START* window.
   2.2. Select the *Events System* driver by clicking .
   2.3. Add in the selected *Events System* component by clicking Add component(s).

   Info: The *Events System* driver will be added to the project.
3. Configure the Asynchronous Event Channel 0 as shown in Figure 7-1.

Figure 7-1. Event Source Selection



3.1. Select the *EVENTS* icon on the left side of the window.

3.2. Select *Asynchronous Event Channel 0*.

Info: The *EVENT SYSYEM SETTINGS* window should appear.

3.3. Scroll down and select the *Real Time Counter overflow* option in the popped up *EVENT SYSTEM SETTINGS* window and close the window.

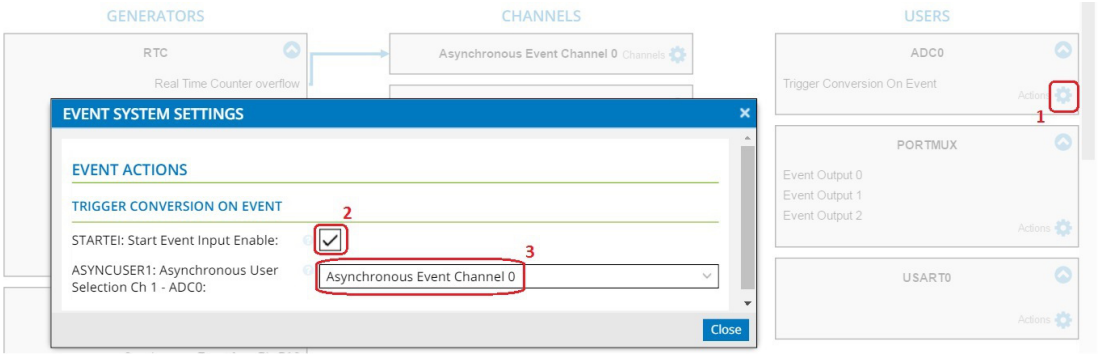
4. Configure the Event user as shown in Figure 7-2:

4.1. Click the setting dialog under *USERS -> ADC0*.

4.2. Select the checkbox of *Start Event Input Enable* in the popped up *EVENT SYSTEM SETTINGS* window.

4.3. Select *Asynchronous Event Channel 0* as ASYNCUSER1 in the popped up *EVENT SYSTEM SETTINGS* window and close the window.

Figure 7-2. Event Channel Selection

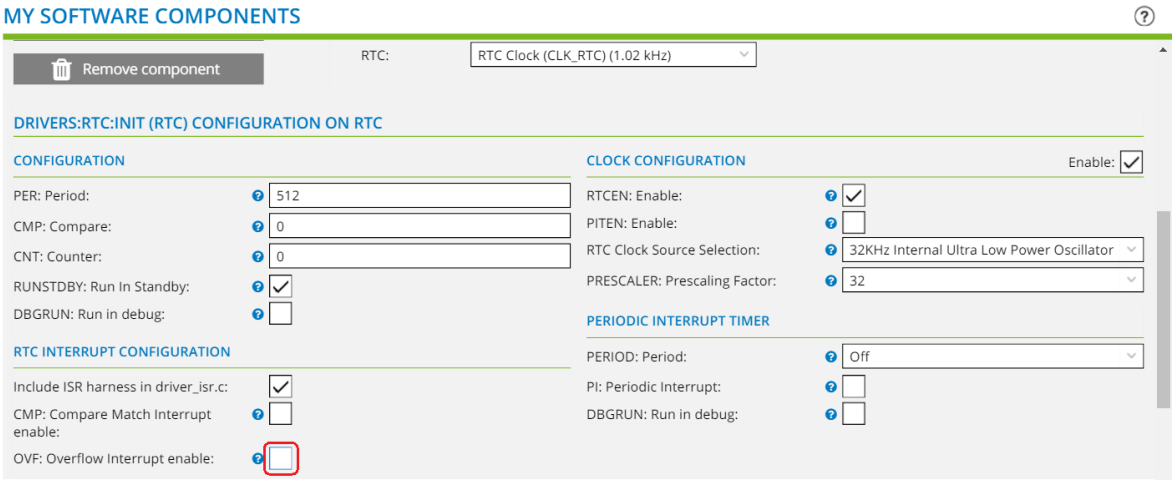


Info: The Event system configuration is now completed with the event generator, event channel, and event user defined as shown in Figure 7-3.

Figure 7-3. Event User Selection

5. Reconfigure RTC in *Atmel | START*:
   5.1. Reopen the RTC configuration window by clicking *DASHBOARD* on the left side of the window.
   5.2. Click the existing *RTC_0* module.
   5.3. Unselect the checkbox for *OVF: Overflow Interrupt enable* as marked with red in Figure 7-4.

Figure 7-4. RTC Reconfiguration



6. Regenerate the project by clicking the *GENERATE PROJECT* button. The project summary window pops up. Click *OK*.

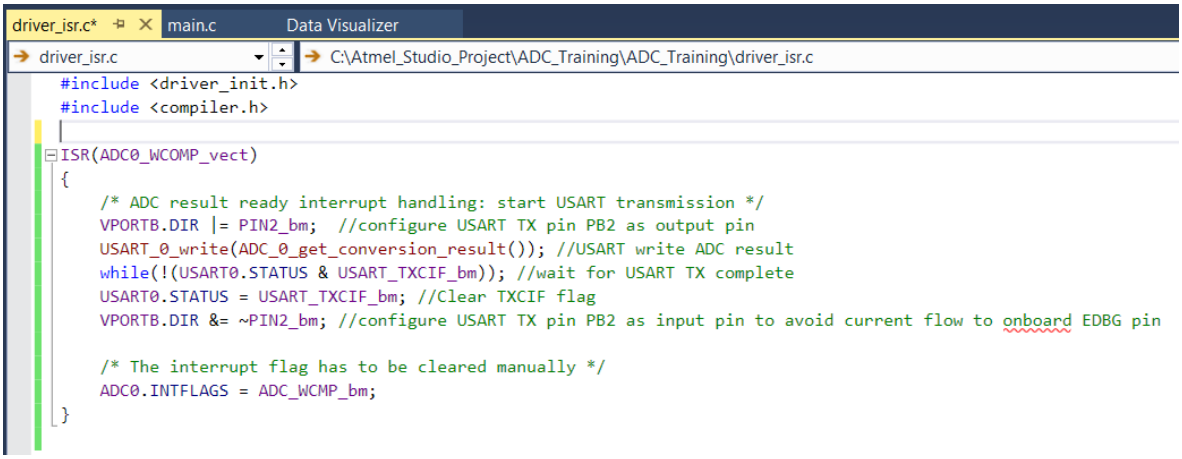
7.2 Event System Driver Code Development

To do: Clean up *driver_isr.c* with removal of the RTC interrupt routine.

Once the event system configuration has been updated in *Atmel | START*, the application code needs to be updated in *Atmel Studio*.

1. Remove the RTC interrupt routine in *driver_isr.c* as its function has been realized by the event system. The *driver_isr.c* file should look like:

Figure 7-5. Assign5_driver_isr.c_view



Info: Since the RTC interrupt routine has been removed, the device will be kept in Standby Sleep mode while the event system with the RTC overflow event source triggers to start the ADC conversion.

2. Program the device by clicking *Debug → Start without Debugging* on the top menu window or by using the *Ctrl+Alt+F5* shortcut.

Info: *Start Without Debugging* will build the project and program the device as long as there are no build errors.

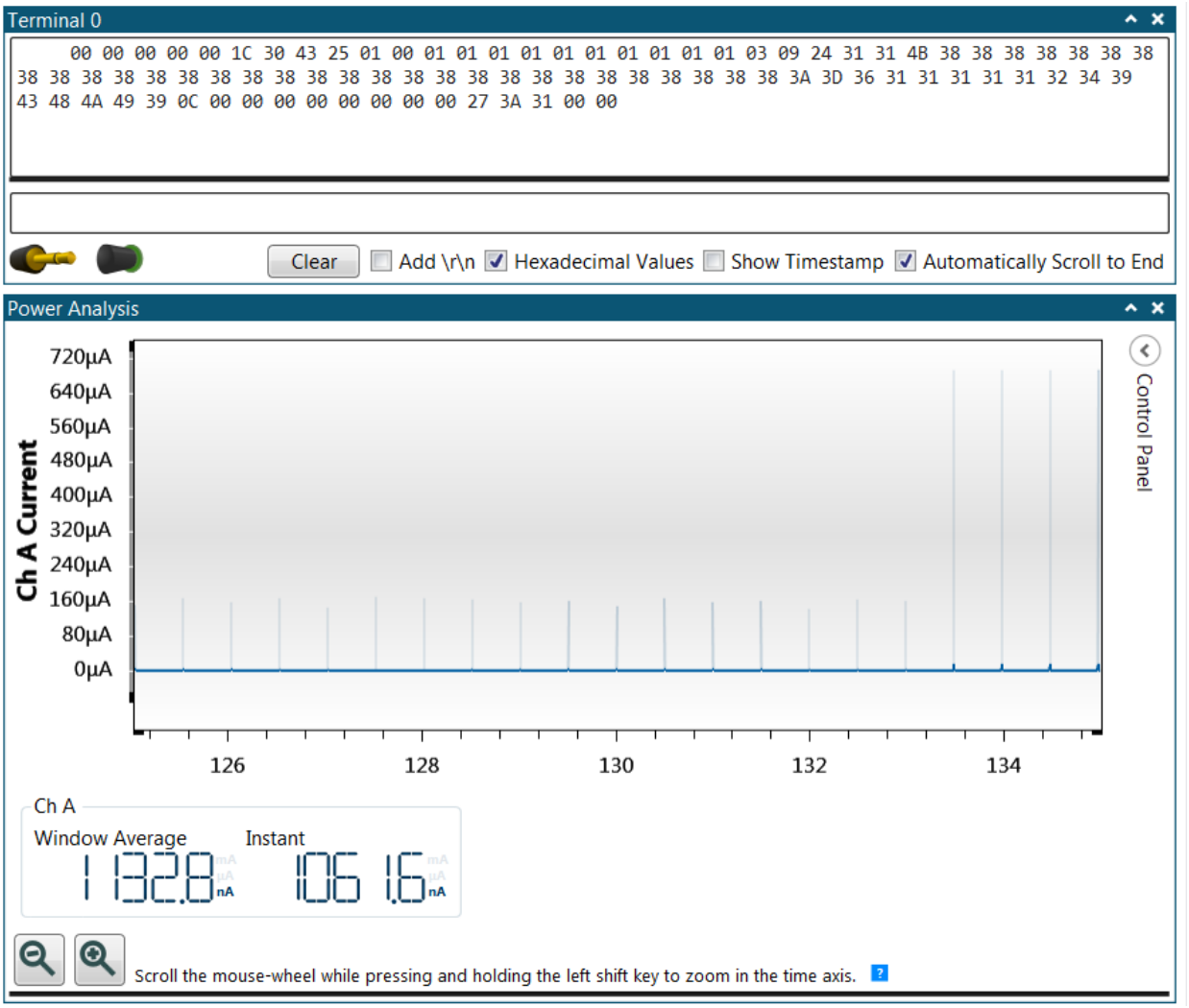
7.3 Observe ADC Functionality and Current Consumption in Data Visualizer

The device has been programmed using the event system instead of the RTC interrupt routine to trigger an ADC conversion.

Observe the ADC functionality and power analysis using Data Visualizer.

Click the *Data Visualizer* tab to observe the ADC functionality and power consumption, as shown in Figure 7-6.

Figure 7-6. ADC Data Print to USART Terminal and Power Analysis



As shown in the terminal, the ADC result is printed only when the ADC result is below 80, which is the ADC window low threshold value. When the value is above that, ADC WCMP interrupt will not be triggered and hence no data is printed to terminal *0*. This is the same behavior as in the previous assignment.

Result: By rotating the potentiometer knob, the voltage input to the ADC input pin, PB1, is changing. The ADC result is printed to the terminal only when it is below the ADC window low threshold value. This is the same behavior as in the previous assignment.

Now, when looking at the current consumption in the power analysis window, the average current is about 1.13 µA which is slightly reduced from 1.18 µA in the previous assignment. This is because the RTC interrupt has been replaced by the event system. It will not wake up the device from the Standby Sleep mode to trigger the ADC conversion. The power reduction is not significant because the device still requires to be waked up from Standby Sleep mode.

Info: The ADC WCMP interrupt is the same interrupt source of the PTC window interrupt as circled in Figure 4-6. It can wake up the device from Standby Sleep mode.

The event system solution also reduces the current consumption of the high peak glitches. There are still two categories of the high peak glitches:

- First, when the RTC overflow event source triggers the ADC conversion through the event system, while the ADC result is above the ADC window low threshold value. There is no USART transmission performed and the current consumption is 170 µA, as illustrated in the first half of the power analysis graph. This is reduced by 33% from 240 µA from the previous assignment.
- Second, when the RTC overflow event source triggers the ADC conversion through the event system, while the ADC result is below the ADC window low threshold value. Here, the USART transmission is performed and the current consumption as shown in second half of the power analysis graph is at the same level as in previous assignment, about 690 µA.

As shown, the ADC EVSYS approach slightly reduces the average current consumption from the previous assignment. It mainly reduces the peak current consumption by 33% from 240 µA down to 170 µA.

Result: The ADC EVSYS approach slightly reduces the average current consumption and mainly reduces the peak current consumption, while the same functional behavior is achieved.

8. Conclusion

This training demonstrates how to perform ADC conversion with the purpose of power optimization. The ADC result is expected to be printed through USART to a terminal. With the goal of reducing the current consumption, various techniques have been implemented to reduce the current consumption through the last four assignments such as:

- Configure the RTC overflow interrupt to start the ADC conversion and keep the CPU in Standby mode between conversions
- Disable digital input buffer on all I/O pins to avoid power supply current and configure the USART TX pin as input during no ADC data transmission period to avoid unnecessary current flow
- The ADC WCMP mode is used, instead of the ADC RESRDY interrupt, to make sure that only interested ADC data is sent to the terminal through USART
- Replace the RTC overflow interrupt routine by Event System to trigger the ADC conversion

In this training, the average current consumption improvement is significant by being reduced from 2.8 mA in *assignment 1* and down to 1.13 µA in *assignment 5*. The peak current consumption has also been reduced from 2.8 mA to 170 µA or 690 µA depending on what scenario it belongs to. As shown, the current consumption gets reduced significantly when Standby Sleep mode is implemented and when optimization on I/O pins is performed. The ADC window compare mode further reduces the current consumption when the scenario with uninterested ADC result occurs. The last assignment with event system implementation reduces the current consumption, but not that significant. This is due to that the device still needs to wake up from the Sleep mode. It has been shown that the current consumption could be reduced significantly for an application with event system implementation when the device does not need to wake up from sleep.

With *Atmel | START*, it is easy to configure or reconfigure a project by adding/removing module(s) and configuring/reconfiguring them with automatically generated drivers.

With *Atmel Studio*, it is easy to edit, build, and run an application. With the embedded Data Visualizer in *Atmel Studio*, it is very easy to check serial data printed to the USART terminal. The power analysis feature in Data Visualizer is so convenient that it can be used to check power consumption during the application optimization proceed.

9. Get Source Code from Atmel | START

The example code is available through Atmel | START, which is a web-based tool that enables configuration of application code through a Graphical User Interface (GUI). The code can be downloaded for both Atmel Studio and IAR Embedded Workbench® via the direct example code-link(s) below or the *BROWSE EXAMPLES* button on the Atmel | START front page.

Atmel | START web page: start.atmel.com

Example Code

- ADC and Power Optimization Solution 1 ADC conversion:
  - http://start.atmel.com/#example/Atmel:ADC_and_Power_Optimization:1.0.0::Application:ADC_and_Power_Optimization_Solution_1_ADC_conversion:
- ADC and Power Optimization Solution 2 RTC SLPCTRL:
  - http://start.atmel.com/#example/Atmel:ADC_and_Power_Optimization:1.0.0::Application:ADC_and_Power_Optimization_Solution_2_RTC_SLPCTRL:
- ADC and Power Optimization Solution 3 IO pin optimize:
  - http://start.atmel.com/#example/Atmel:ADC_and_Power_Optimization:1.0.0::Application:ADC_and_Power_Optimization_Solution_3_IO_pin_optimize:
- ADC and Power Optimization Solution 4 ADC WCMP:
  - http://start.atmel.com/#example/Atmel:ADC_and_Power_Optimization:1.0.0::Application:ADC_and_Power_Optimization_Solution_4_ADC_WCMP:
- ADC and Power Optimization Solution 5 EVSYS:
  - http://start.atmel.com/#example/Atmel:ADC_and_Power_Optimization:1.0.0::Application:ADC_and_Power_Optimization_Solution_5_EVSYS:

Press *User guide* in Atmel | START for details and information about example projects. The *User guide* button can be found in the example browser, and by clicking the project name in the dashboard view within the Atmel | START project configurator.

Atmel Studio

Download the code as an .atzip file for Atmel Studio from the example browser in Atmel | START, by clicking *DOWNLOAD SELECTED EXAMPLE*. To download the file from within Atmel | START, click *EXPORT PROJECT* followed by *DOWNLOAD PACK*.

Double-click the downloaded .atzip file and the project will be imported to Atmel Studio 7.0.

IAR Embedded Workbench

For information on how to import the project in IAR Embedded Workbench, open the Atmel | START user guide, select *Using Atmel Start Output in External Tools*, and *IAR Embedded Workbench*. A link to the Atmel | START user guide can be found by clicking *About* from the Atmel | START front page or *Help And Support* within the project configurator, both located in the upper right corner of the page.

10. Revision History

Doc. Rev.	Date	Comments
B	04/2018	Added key words to open document in context from Developer Help
A	02/2018	Initial document release.

The Microchip Web Site

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

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

Customer Change Notification Service

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

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

Customer Support

Users of Microchip products can receive assistance through several channels:

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

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

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

Microchip Devices Code Protection Feature

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

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

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

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

Legal Notice

Trademarks

ISBN: 978-1-5224-2945-6

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	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820