tinyAVR 1-series Training

Getting Started with the tinyAVR® 1-series

Prerequisites

- **Hardware Prerequisites**
  - Microchip ATtiny817 Xplained Pro board
  - Micro-USB cable (Type-A/Micro-B)
  - One female-to-female wire
  - Internet connection
- **Software Prerequisites**
  - Atmel Studio 7.0
  - Atmel Studio ATtiny_DFP version 1.2.112 or above
  - A list of supported browsers can be found here: http://start.atmel.com/static/help/ → Requirements and Compatibility → Supported Web Browsers.
- **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 you 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:

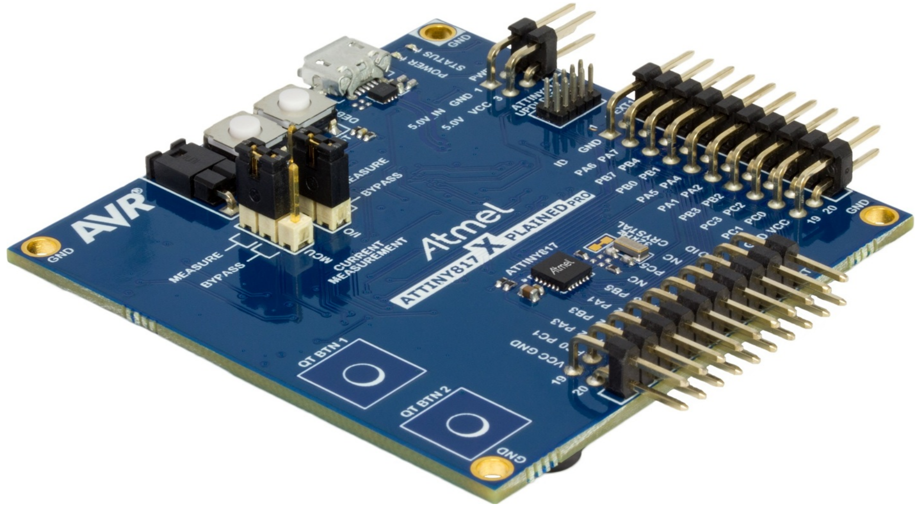
- Get help selecting the MCU based on both software and hardware requirements
- Find and develop examples
- Configure drivers, middleware, and example projects
- Get help with setting up a valid PINMUX layout
- Configure system clock settings

The ATtiny817 Xplained Pro evaluation kit is a hardware platform for evaluating the ATtiny817 microcontroller. A fully integrated embedded debugger is included in 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 module 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, timers TCA and TCB, Event System, USART, CCL (Configurable Custom Logic), and PIT (Periodic Interrupt Interval).

Figure 1. ATtiny817 Xplained Pro



The following topics are covered:

- Driver Configuration in Atmel START
- PINMUX driver configuration and check LED toggle on button press
- Generate a PWM by using timer counter A (TCA) and implement Variable-Pulse-Width by using the RTC interrupt
- Duty cycle and frequency measurement using input capture mode of TCB
- USART configuration
- Use the Data Visualizer tool to send data to the serial terminal
- CCL (Configurable Custom Logic): A programmable logic peripheral, which can be connected to the device pins, events, or peripherals, which allows the user to eliminate external logic gates for simple glue logic functions. Here, configure CCL to generate the specific signal using the event from GPIO and two PWM signals.

Note: Solution projects for this training can be found in Atmel START → BROWSE EXAMPLES: 'Getting Started with the tinyAVR® 1-series'

Table of Contents

1. Icon Key Identifiers

The following icons are used in this document 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.

Warning: Indicates important information.

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

2. Assignment 1: LED TOGGLE Application

An application will be developed that controls the LED using the push-button on the board. The LED will be OFF on pressing the button, default state is LED ON.

A project will be configured in Atmel START using PINMUX driver configuration and clock configuration, followed by generation of the corresponding Atmel Studio 7 project.

The code will be developed in Atmel Studio 7 using PINMUX driver functions generated by Atmel START configuration.

On the ATtiny817 Xplained Pro board, LED0 is connected to pin PB4, and the push-button (SW0) is connected to pin PB5.

For application:

- Peripherals used: GPIO (PB4, PB5).
- Clock: 3.33 MHz.

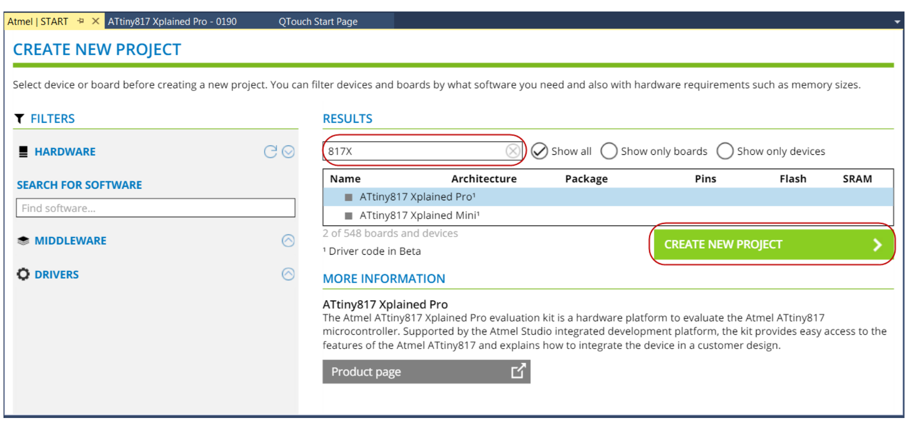
2.1 Atmel START Project Creation

Configure the PINMUX 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 **ATtiny817 Xplained Pro** is highlighted, then click on **CREATE NEW PROJECT**, as shown below.

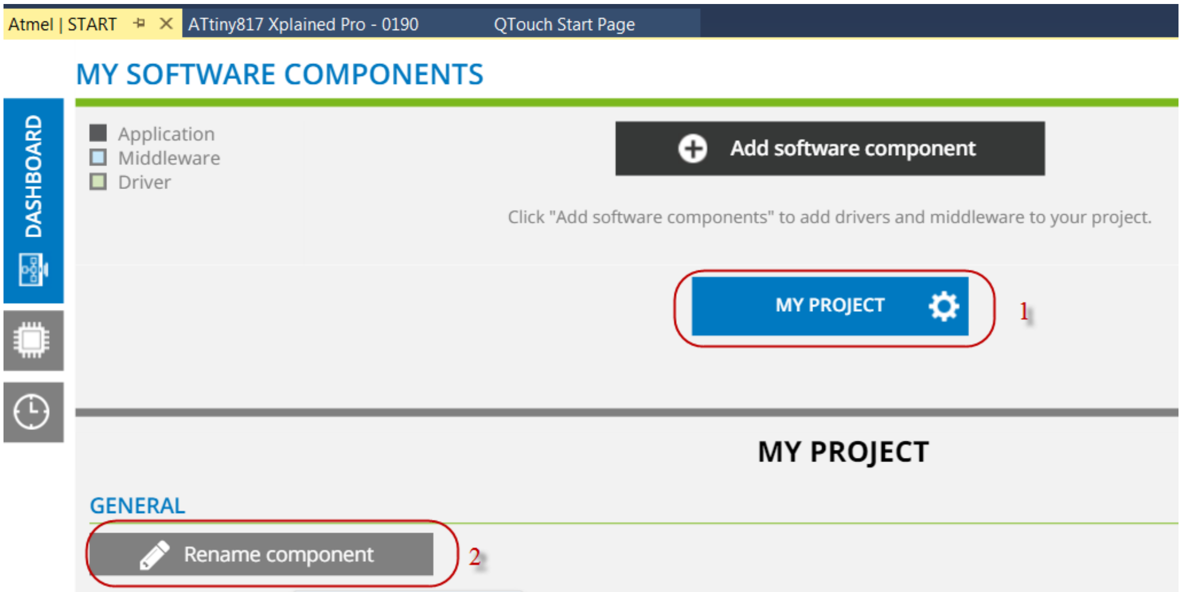
Figure 2-1. CREATE NEW PROJECT



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

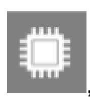
4. In **MY SOFTWARE COMPONENTS** window:
   - 1. Click on **MY PROJECT**.
   - 2. Select **Rename Component**.

Figure 2-2. Rename Component



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

5. In the **RENAME COMPONENT** window specify the new project name as "Assignment_ATtiny817" and select **Rename**.

6. Now, for PINMUX configuration, click on , the navigation tab on the left side of the window.

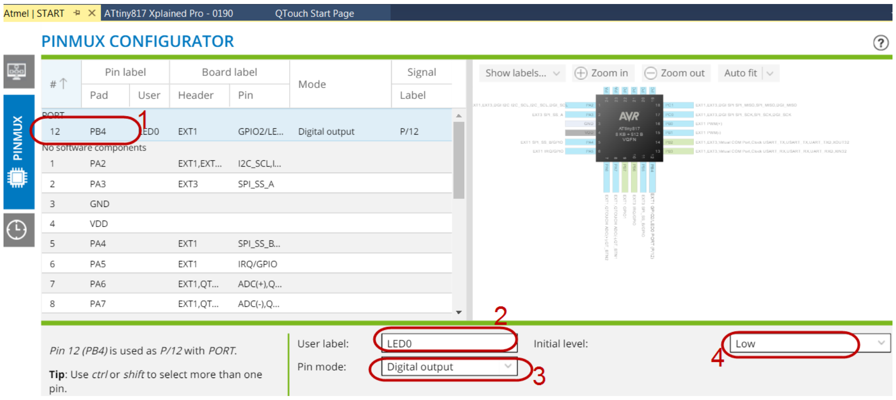
Info: The PINMUX configurator displays an illustration of the device package selected. It shows which pins are currently used by different peripherals. The GPIO pins can be configured here.

Info: Here PB4 is configured as LED0 and PB5 as SW0. Configuration is shown with four steps (the red markings numbered 1, 2, 3, and 4 in the figure below).

7. **Configuration of PB4**:
   - 1. Click on **PB4**.
   - 2. Enter "User label:" as **LED0**.
   - 3. Enter "Pin mode:" as **Digital output**.

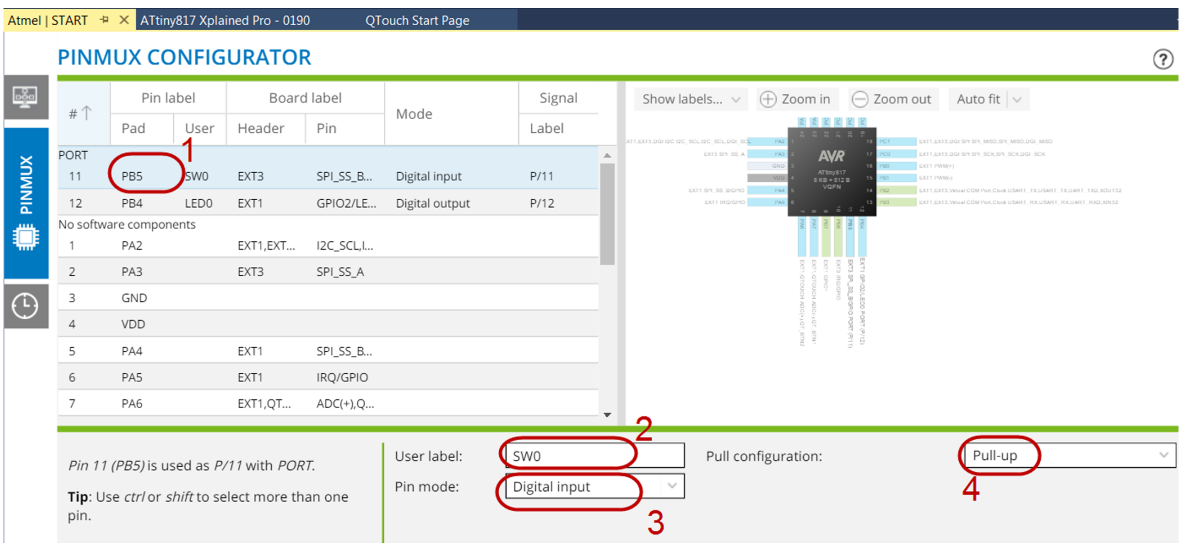
– 4. Select "Initial level:" as **Low**.

Figure 2-3. PINMUX Configuration LED0



8. **Configuration of PB5**:
   – 1. Click on **PB5**.
   – 2. Enter "User label:" as **SW0**.
   – 3. Enter "Pin mode:" as **Digital input**.
   – 4. Select "Initial level:" as **Pull-up**.

Figure 2-4. PINMUX Configuration SW0



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

9. **CLOCK CONFIGURATOR**: Now, for clock configuration, click on , the navigation tab on the left side of the window.

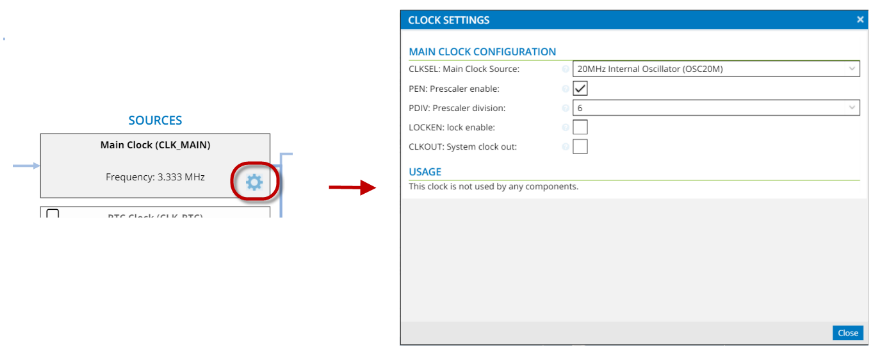
Info: Now the **CLOCK CONFIGURATOR** window will be displayed. It consists of oscillators and clock sources of different types. The required clock source can be selected and the calculated output frequency will be displayed.

- The **OSCILLATORS** section displays the oscillators available for the selected device. The oscillator parameters can be configured by selecting "Settings Dialog" (cog wheel icon).
- The **SOURCES** section is used to configure the clock frequency by selecting input signal and changing the multiplier.

10. Click on the "Settings Dialog" (cog wheel icon) to view the default **Main clock** settings from **SOURCES**, as shown in the figure below.

Info: The **CLOCK SETTING** window will be displayed.

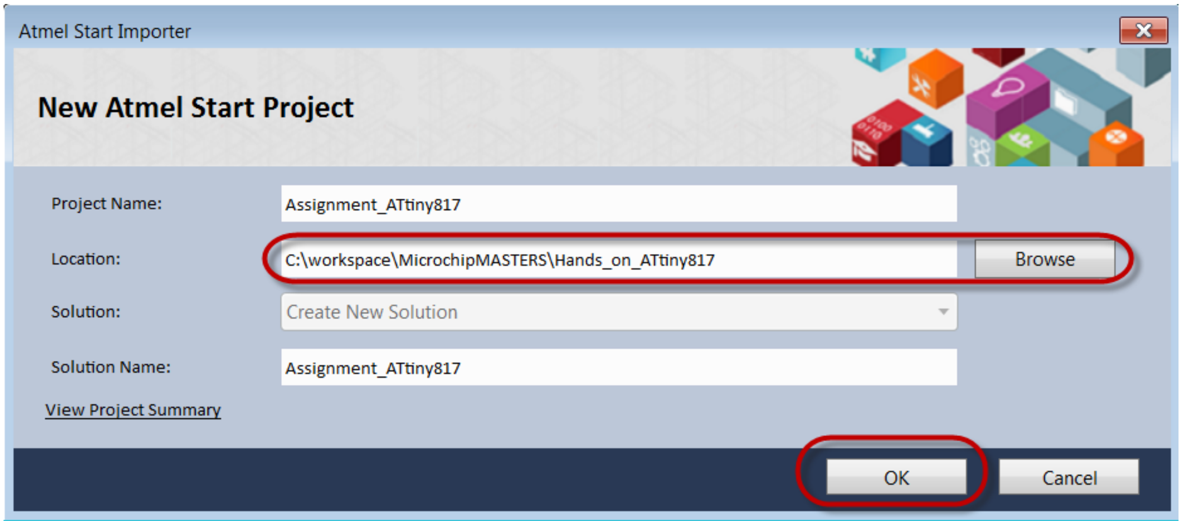
Figure 2-5. CLOCK SETTING



Info: For this application the default clock settings are kept as they are. Here, the Main Clock source is 20 MHz OSC, the prescaler is divided by 6. The resulting CPU clock frequency is 3.33 MHz. The click on the "question mark" next to each configuration: , will be directed to the data sheet description of individual bit settings.

11. Click **Close** in the **CLOCK SETTING** window.
12. Now, click on the **GENERATE PROJECT** button .
13. Select the desired path where the project should be stored, as shown in the figure below, and then click OK.

Figure 2-6. Project Importer Window



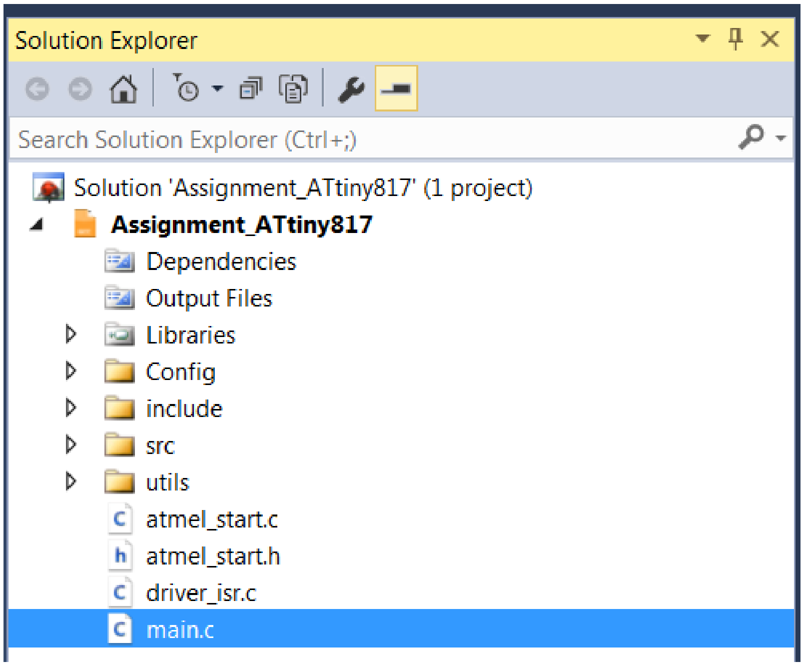
Result: An Atmel START project has been created.

2.2 Atmel START Project Overview

The project configured in Atmel START generates peripheral driver functions and files, as well as a main() function that initializes all the drivers.

About folders and files generated by Atmel START:

- The **Config** folder contains clock configuration. `F_CPU` is defined in clock_config.h.
- The driver header and source files are to be found in the **src** and **include** folders
- The **atmel_start_pins.h** file in the **include** folder contains the PINMUX driver functions
- 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 function `atmel_start_init()` initializes the MCU, the drivers, and the middleware in the project
- The **driver_isr.c** file contains ISR if the interrupts are enabled in the configuration of the project



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

1. In the **Assignment_ATtiny817** project, double-click the main.c file from the **Solution Explorer** window.
2. Select the `atmel_start_init` function, then **RIGHT CLICK** → **Goto Implementation**. Repeat the procedure for the `system_init()` function to direct to the function definition.

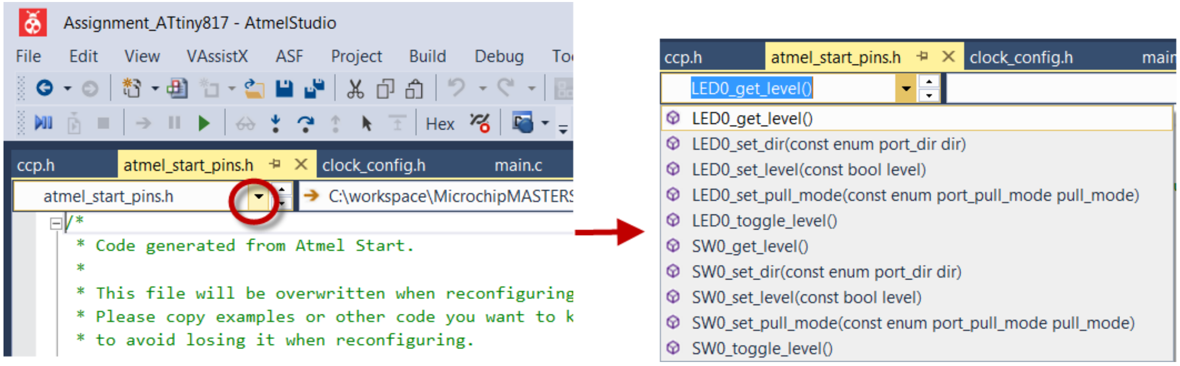
Info: The `mcu_init()` function enables the internal pull-up resistor on all pins to reduce the power consumption. All driver initialization functions are called from the `system_init()` function. Also, the LED0 and SW0 port pins are configured to output and input mode with initial pin status.

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

Tip: driver_init.c → `system_init()` → `CLKCTRL_init()`

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

4. Open **atmel_start_pins.h** and 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.



Result: The Atmel START project overview is completed.

2.3 Code Development

On the ATtiny817 Xplained Pro board, the behavior of the pins associated with LED0 and SW0 is as follows:

Table 2-1. Behavior of the Pins Associated with LED0 and SW0

SW0/LED0	Status	Pin level
SW0	Button depressed	PB5: LOW
SW0	Button released (default status)	PB5: HIGH
LED0	ON	PB4: LOW
LED0	OFF	PB4: HIGH

To do: Write a code that turns the LED OFF when the button is depressed and turns it back to ON when it is released.

1. Open the main.c file in the **Assignment_ATtiny817** project.
2. Insert the code in `while(1)` to read the SW0 status and to configure LED0 status as mentioned below.
   – When the button (SW0) is depressed, LED0 is OFF, (LED0 pin level = high)
   – When the button (SW0) is released, LED0 is ON, (LED0 pin level = low)

```
if(!SW0_get_level())                // (SW0)button pressed. PB5 is low
{
    LED0_set_level(true);       //LED0 is turned OFF : PB4 is high
    while(!SW0_get_level());      //wait till (SW0)button release.
    LED0_set_level(false);        //(SW0)button released. LED0 is turned ON: PB4 is low
}
```


Info: The `SW0_get_level()` function returns the SW0 pin status. The `LED0_set_level(true)` function sets the LED0 level high.

3. After the code completion, press F7 to build the solution. The build should finish successfully with no errors.

Result: The code should look like the image shown below.

Figure 2-7. Assignment 1 Code

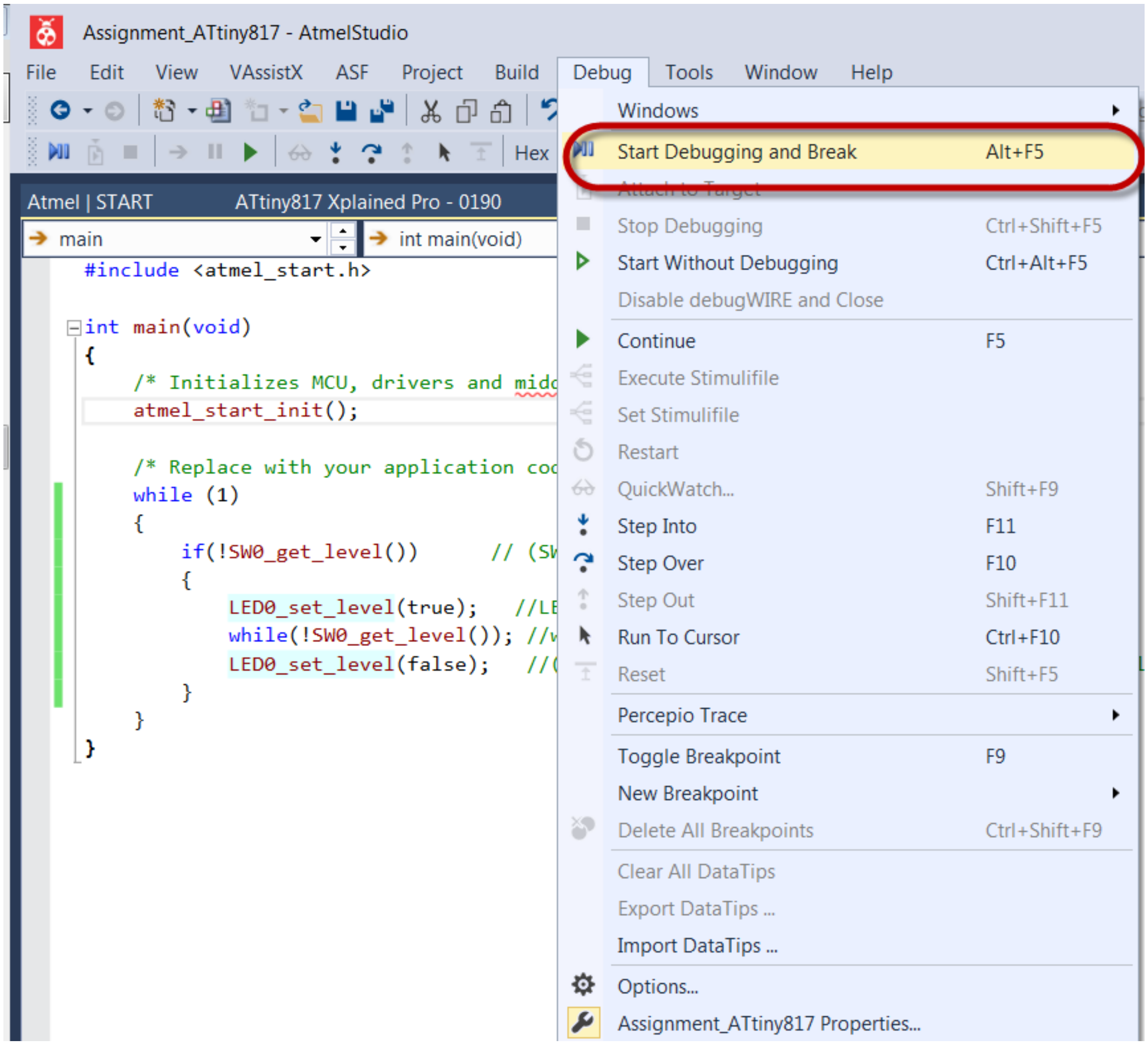
```
#include <atmel_start.h>

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

    /* Replace with your application code */
    while (1)
    {
        if(!SW0_get_level())      // (SW0)button pressed. PB5 is low
        {
            LED0_set_level(true);   //LED0 is turned OFF : PB4 is high
            while(!SW0_get_level()); //wait till (SW0)button release.
            LED0_set_level(false);   //(SW0)button released. LED0 is turned ON : PB4 is low
        }
    }
}
```

2.4 Debug Application

To do: Debug the application 'Assignment1: LED Toggle'.

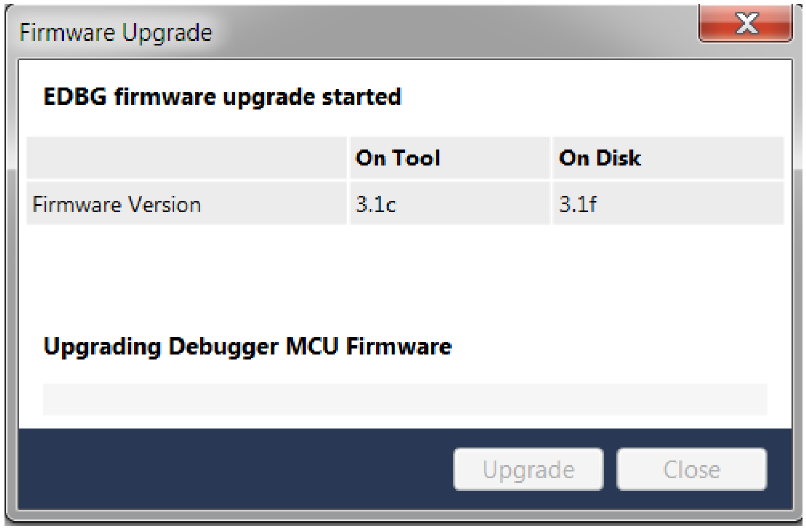
1. Power the ATtiny817 Xplained Pro board by connecting it with Micro-USB cable to the computer.
2. Select **Debug** → **Start Debugging and Break** (or Alt + F5). Follow the prompts to select the ATtiny817 Xplained Pro EDBG as programmer/debugger, and press Alt + F5 again.

Figure 2-8. Debug Menu



Info: If the firmware version of the on-board debugger is older than the one in the Atmel Studio installation, it will be asked to upgrade the firmware.

Figure 2-9. Firmware Upgrade

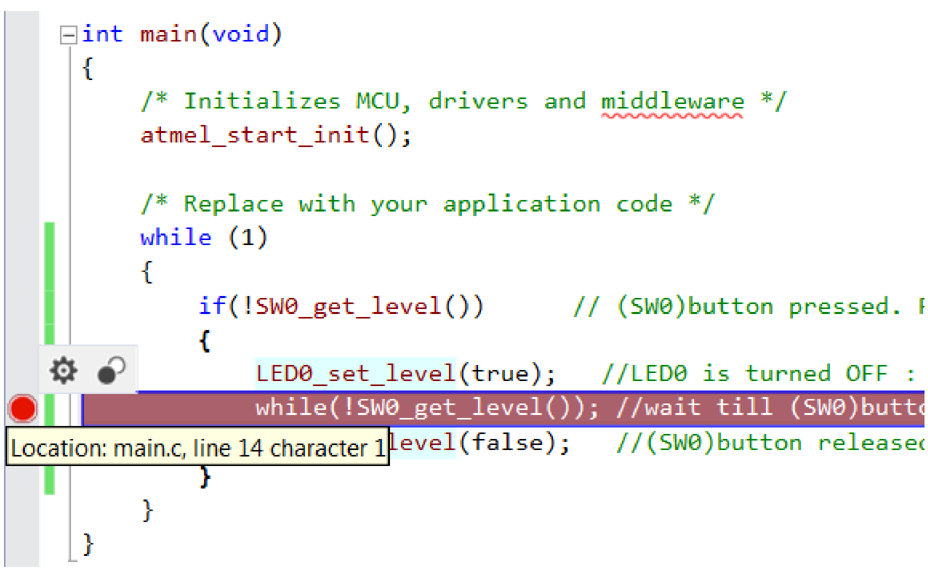


Select **Upgrade**. When the progress bar is complete, select **Close**. Now, start the debugging by selecting **Debug** → **Start Debugging and Break**. (Alternative: Alt+F5).

Result: The debugger is started and breaks in `main()`. Debugging can now be started.

3. Click on the margin to insert the breakpoint as shown in the figure below.

Figure 2-10. Breakpoint Inserted



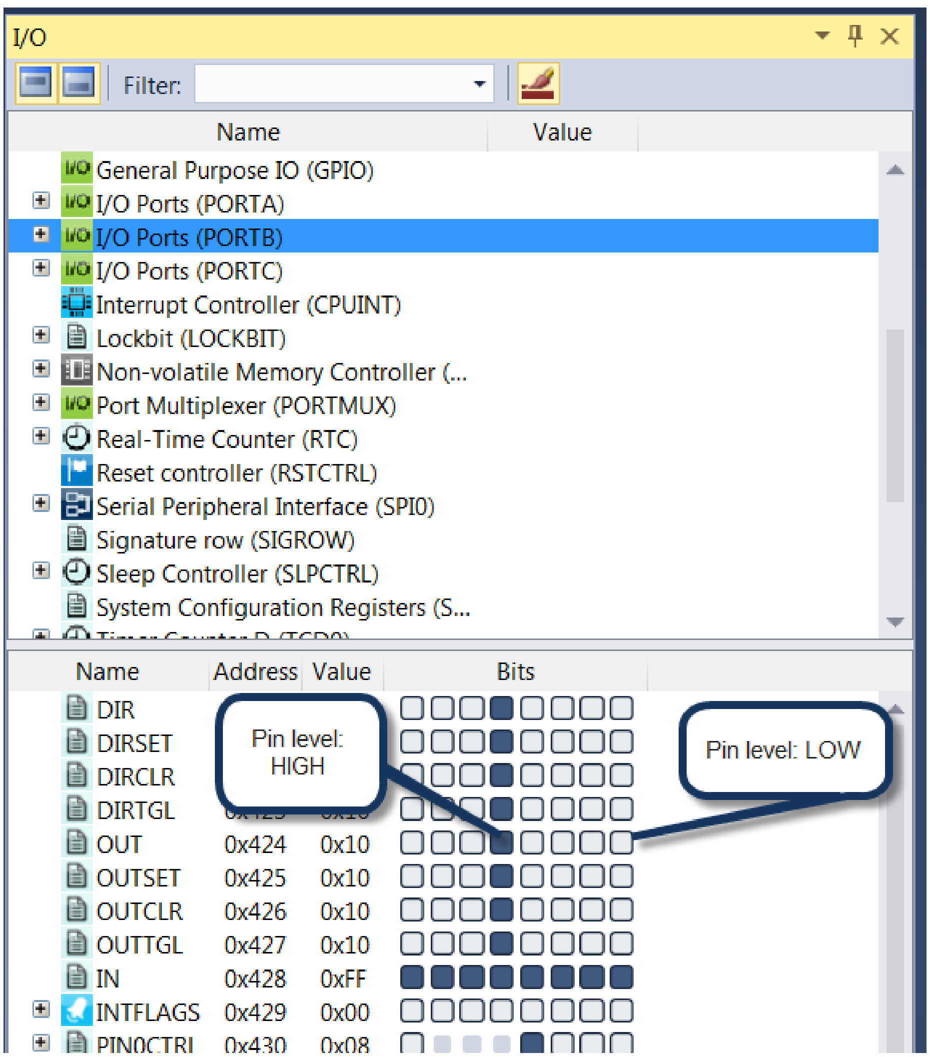
4. Run to breakpoint by clicking **Debug** → **Continue**.

Tip: The play button, ▸, located on the toolbar close to the top of the Atmel Studio 7 window can be used to start or continue debugging. Keyboard shortcut F5 can also be used.

5. Press the push button SW0 and observe that execution stops at the breakpoint.
6. 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 PORTB register group as shown in the image below.

Info: In the I/O view, the status of all the peripherals can be observed. 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, as a filled square corresponds to bit status 1. An empty square corresponds to bit status 0.

Figure 2-11. I/O View



7. Do single step debugging by pressing F10 and observe the status of PB4.

   Info: PB7 is the left most bit and PB0 is the right most bit.

8. Remove a breakpoint by clicking on it and run the code by clicking **Debug** → **Continue** or select .
9. Stop debugging by selecting .
10. Press the push-button SW0 and observe the LED0 toggle.

    Info: The application is programmed successfully to the ATtiny817 Xplained Pro board.

3. Assignment 2: Generate PWM, Measure Duty Cycle and Frequency

The ATtiny817 has two 16-bit Timer/Counter instances, TCA and TCB, and one 12-bit Timer/Counter, TCD.

Here, an application will be developed to generate the PWM using the TCA. The RTC interrupt will be used to vary the duty cycle of the PWM.

The TCA waveform output will be used as input to the TCB through the Event System, and the input capture mode of the TCB will be used to measure the duty cycle and the frequency of the waveform. The measured data will be sent to the terminal through the USART.

Single-Slope PWM Generation mode will be used for TCA. Here, the period is controlled by the PER register, while the values of the CMPn compare register controls the duty cycle of the waveform generated (WG) output, the WOn. The counter value is compared to the CMPx registers and the PER register to set the waveform period or pulse width.

The Atmel START project from **Assignment1: LED TOGGLE (Assignment_ATtiny817)** will be reconfigured to add drivers for TCA, TCB, Event System, RTC, and USART.

For the application the peripherals used are:
- TCA (waveform output WO0 on PB0)
- TCB
- Event System (event input from pin PA5)
- RTC
- USART

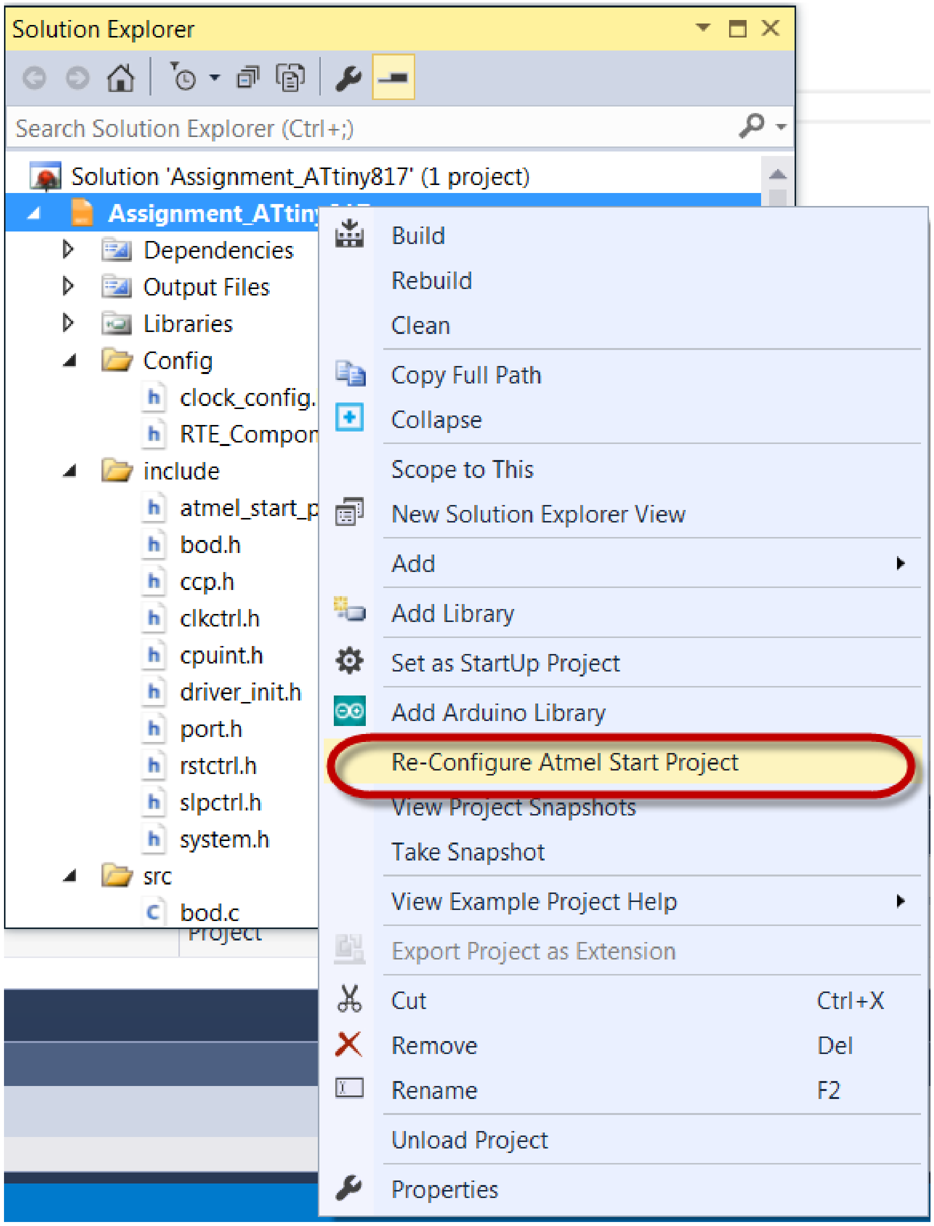
Clock:
- 3.33 MHz main clock
- 1 kHz RTC clock

3.1 TCA Driver

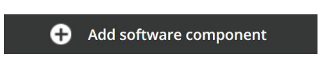
To do: Add TCA driver to generate PWM signal.

1. Open the "Assignment1:LED TOGGLE" project **Assignment_ATtiny817**.
2. In the **Solution Explorer** window, click on the project name **Assignment_ATtiny817** and then **Right Click → Re-Configure Atmel Start Project** as shown below.

Figure 3-1. Re-Configure Menu

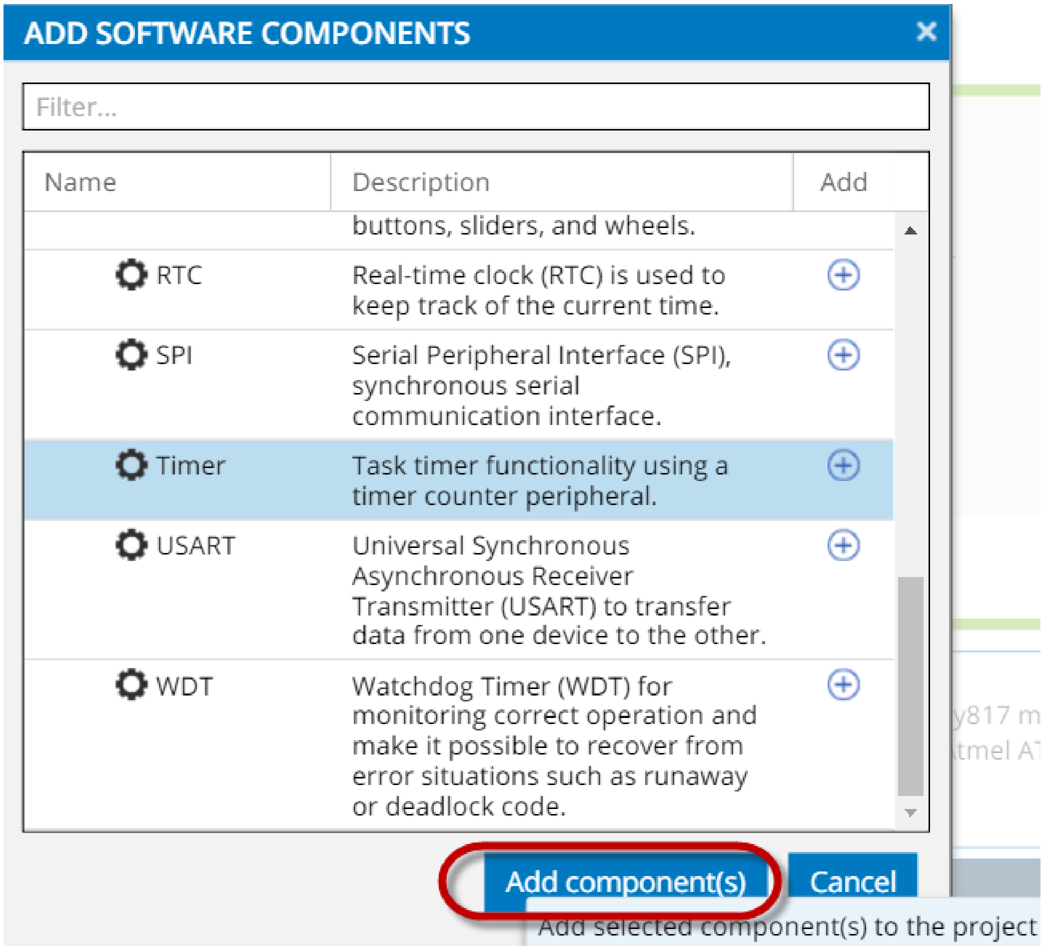


Info: "Atmel | START" window will appear.

3. Click the 

box and then expand **Drivers** from the **ADD SOFTWARE COMPONENTS** window.
4. Search for the **Timer** driver, select it and then click the **Add Component(s)**, as shown in the figure below.

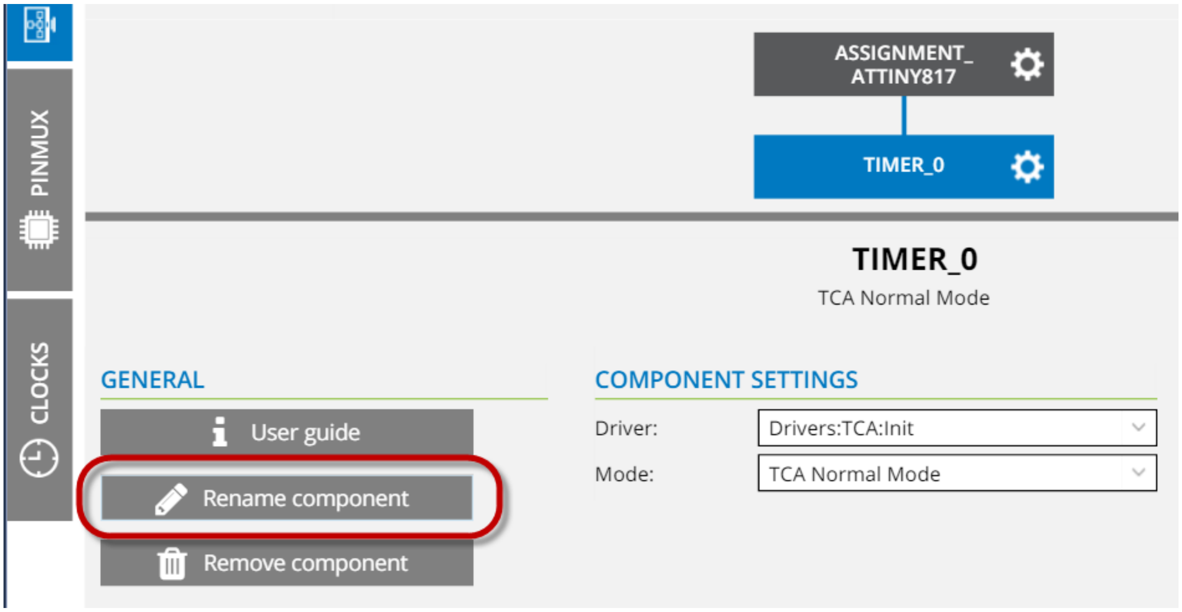
Figure 3-2. ADD SOFTWARE COMPONENT



Info: The **Timer_0** driver will be added to the project.

5. Click the **TIMER_0** and then click on **Rename component**, as shown in the figure below.

Figure 3-3. Rename TCA



6. Specify the new name as **TCA_PWM** and click **Rename**.

Info: Now the TCA peripheral configuration needs to be done to generate the PWM signal. This is shown below.

To do: Configure the TCA driver to generate PWM (W0) on pin PB0, frequency: 32 kHz, initial duty cycle: 10.

Info: The configuration is shown with red markings numbered 1 to 8 in Figure 3-4.

1. Select "Driver" as "TCA:Init" and mode as "TCA Normal Mode".

Info: Here, by default "Driver" is selected as "TCA:init" and mode is selected as "Normal mode". This timer has two modes; "Normal mode" (one 16-bit timer/counter) and "Split Mode". In the "Split Mode" the 16-bit timer/counter acts as two separate 8-bit timers, each with three compare channels for PWM generation.

2. Select waveform output WO/0 on pin PB0.

tip: To verify the WO/0 output pin of the TCA, refer the data sheet section "I/O Multiplexing and Considerations". Click on the "question mark" next to any configuration, ?, and go to section "I/O Multiplexing and Considerations" to check the TCA WO/0 pin.

3. Enable the peripheral by selecting the check-mark of "ENABLE: Module Enable".
4. Select clock frequency for the timer/counter as "System Clock".

Info: Here, by default it has been selected as "System Clock" (with no prescaler).

5. Enter the "PER: Period" register value as 100 (or 0x64).

Info: It contains 16-bit TOP value in the timer/counter. It decides the PWM frequency.

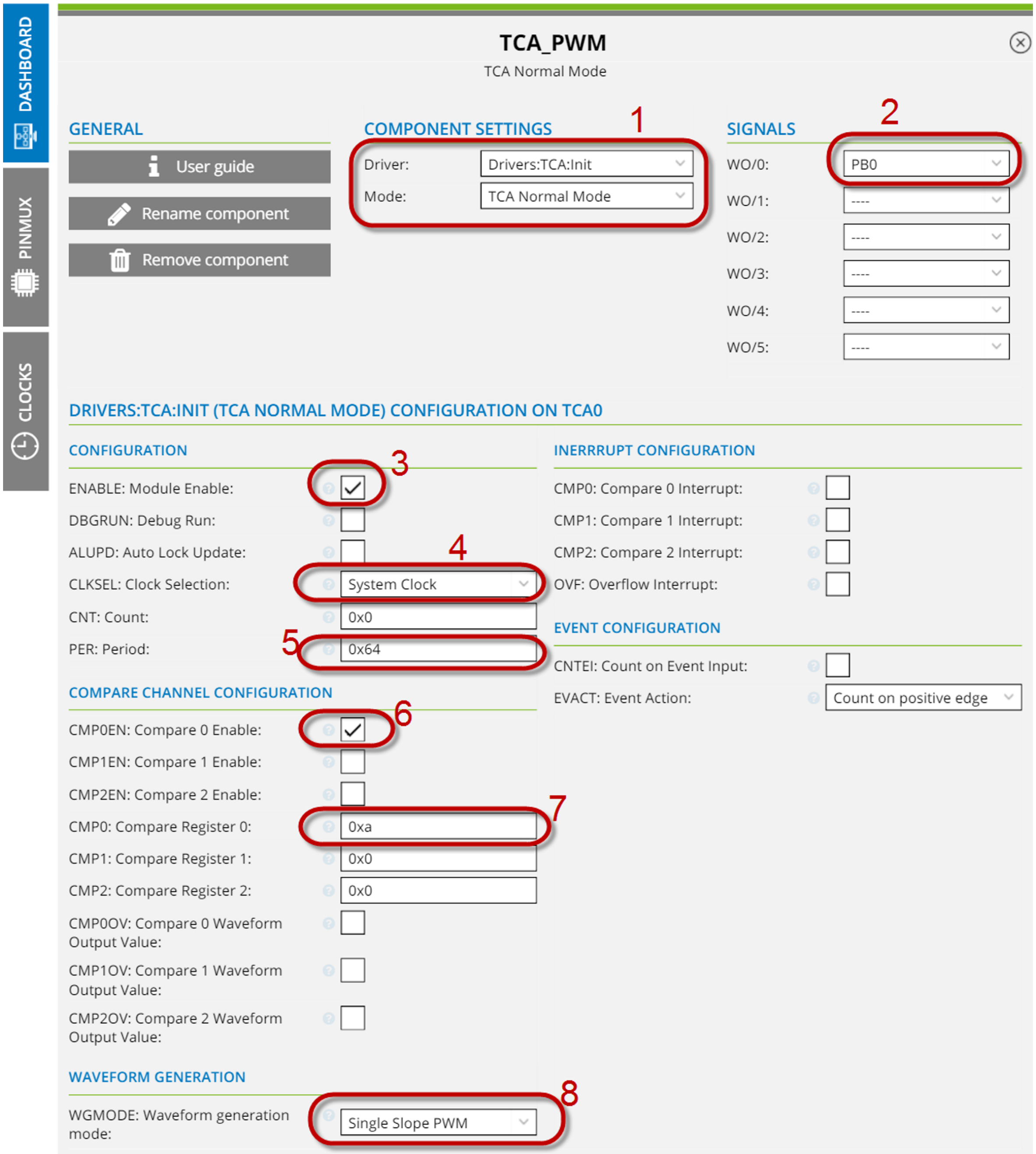
6. Select the check-mark CMP0EN to get the PORT output settings for WO0 output.
7. Enter "CMP0: Compare Register 0" value as 10 (or 0xa).

Info: This register is continuously compared to the counter value of the timer. Normally, the outputs from the comparators are then used for generating waveforms. This register decides the duty cycle of PWM. Initial duty cycle has been selected as 10 (0xa). Waveform output WO/0 is required so Compare Channel 0 is configured here.

8. Select "WGMMODE: waveform generation mode" as "Single Slope PWM".

Info: The waveform generation mode controls the counting sequence of the counter, TOP value, UPDATE condition, interrupt condition, and type of waveform that is generated. In this Single Slope PWM mode, the period is controlled by TCA.PER, while the values of TCA.CMPn control the duty cycle of the WG output. The single-slope PWM frequency (f_{PWM_SS}) depends on the period setting (TCA.PER), the system's peripheral clock frequency f_{CLK_PER} and the TCA clock prescaler (CLKSEL :clock selection). It is calculated by the following equation: $f_{PWM\_SS} = \frac{f_{CLK\_PER}}{N \cdot (PER + 1)}$. Here N = 1, PER =100, and (f_{PWM_SS}) = 3.3 MHz.

Figure 3-4. TCA Configuration



Result: The TCA configuration is completed.

3.2 RTC Driver

To do: Add RTC driver to generate overflow interrupt.

1. Click the **Add software component** box.
2. Expand **Drivers** from the **ADD SOFTWARE COMPONENT** window. Search for the **RTC** driver, select it and then click the **Add Component(s)**. (Refer steps 3 and 4 from the TCA driver.)
   Note: Add the **RTC** driver here.

   Info: The **RTC_0** driver will be added to the **Assignment_ATtiny817** project. Now the RTC peripheral configuration needs to be done as shown below.

To do: Click the **RTC_0** box and configure the RTC driver to generate overflow interrupt every 500 milliseconds.

Info: The configuration is showed with red markings numbered 1 to 5 in Figure 3-5.

1. Enable peripheral by selecting checkbox "RTCEN: Enable".
2. Select "RTC Clock Source Selection" as "32 kHz Internal Ultra-Low Power oscillator (OSCULP32K)".
3. Select "PRESCALER: Prescaling Factor" as "32".

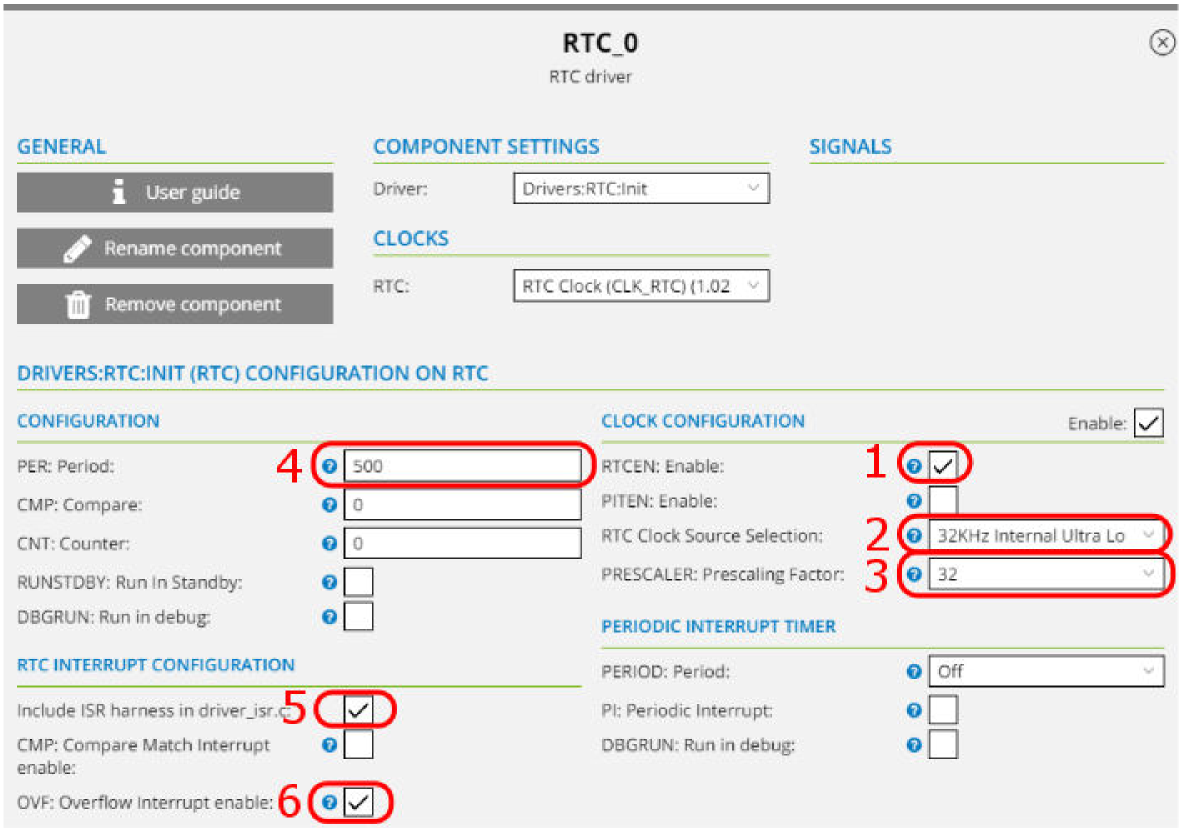
   Tip: Resulting RTC clock can be verified by selecting navigation button (on the right side of the window) **CLOCKS**. Select **DASHBOARD** to return to the driver configuration.

4. Select "PER: Period" register value as 500 (or 0x1f4).

   Info: This is the 16-bit Period Register: When the counter register reaches this value, the overflow interrupt is set and the counter register is reset.

5. Generate ISR code in driver_isr.c by selecting checkbox "Include ISR harness in driver_isr.c".
6. Enable overflow interrupt by selecting checkbox "OVF: Overflow Interrupt enable".

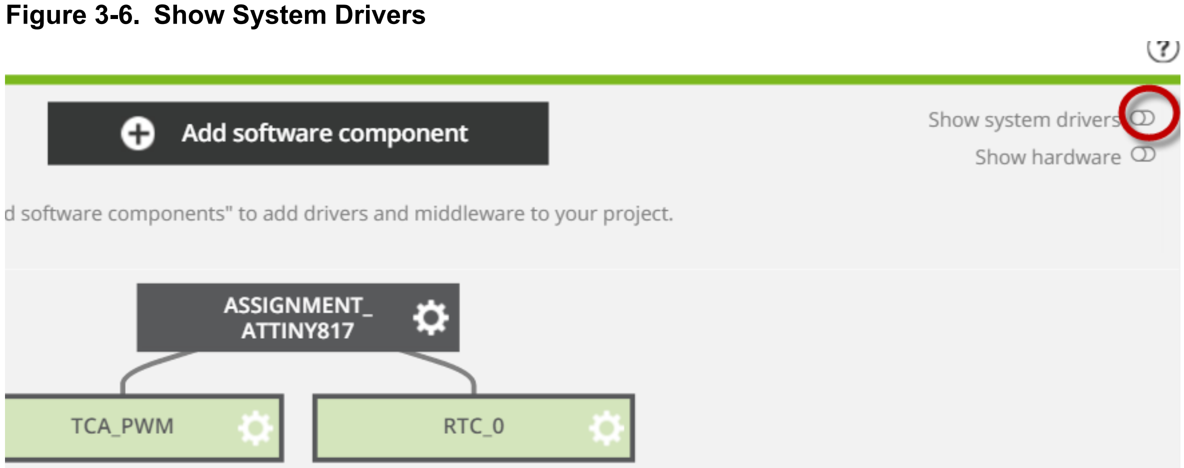
Figure 3-5. RTC Configuration



Info: To generate the interrupt it is required to enable the **Global Interrupt Enable bit** in the **Status Register**. This is described below.

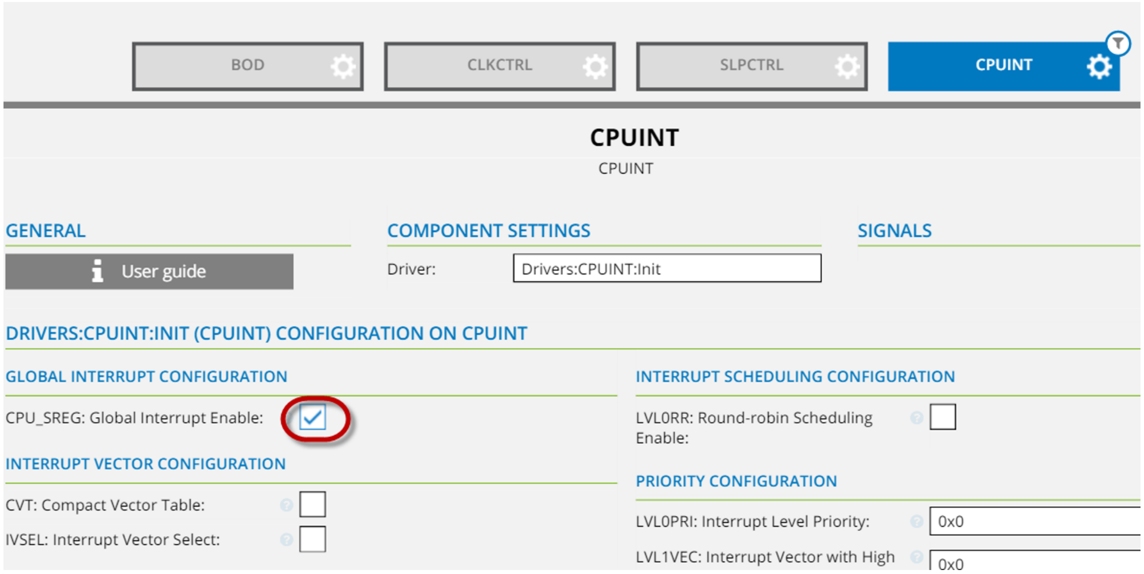
7. Click on the small circle from the **Show system drivers** to the right side, as shown in the figure below.

Figure 3-6. Show System Drivers



8. Click the **CPUINT** and then select the checkbox, **CPU_SREG: Global Interrupt Enable**, as shown in the figure below.

Figure 3-7. Enable CPU_SREG



Result: The RTC configuration is completed.

3.3 TCB Driver

The peripheral TCB will be configured to capture the PWM signal to test the capture functionality.

To do: Add TCB driver.

1. Click the **Add software component** box.
2. Expand **Drivers** from the **ADD SOFTWARE COMPONENT** window.
3. Search for the **Timer** driver, select it and then click on the **Add Component(s)**. (Refer steps 3 and 4 from the TCA driver.)

   Info: The **TIMER_0** driver will be added to the **Assignment_ATtiny817** project.

4. Click the **TIMER_0** and then click **Rename component**. Specify the new name as **TCB_Duty_Frequency_Measure** and click **Rename**. (Refer steps 5 and 6 from the TCA driver.)

   Info: Now the TCB peripheral configuration needs to be done for input capture mode.

To do: Configure the TCB driver with input capture mode to measure the duty cycle and frequency of the given input PWM.

Info: The configuration is shown with red markings numbered 1 to 5 in the Figure 3-8.

1. Select **Driver** as **TCB:Init**.

Info: Here, by default "Driver" is selected as "TCB:init".

2. Enable peripheral by selecting checkbox **ENABLE: Enable**.
3. Select the **CLKSEL: Clock select** as **CLK_PER (No Prescaling)**.

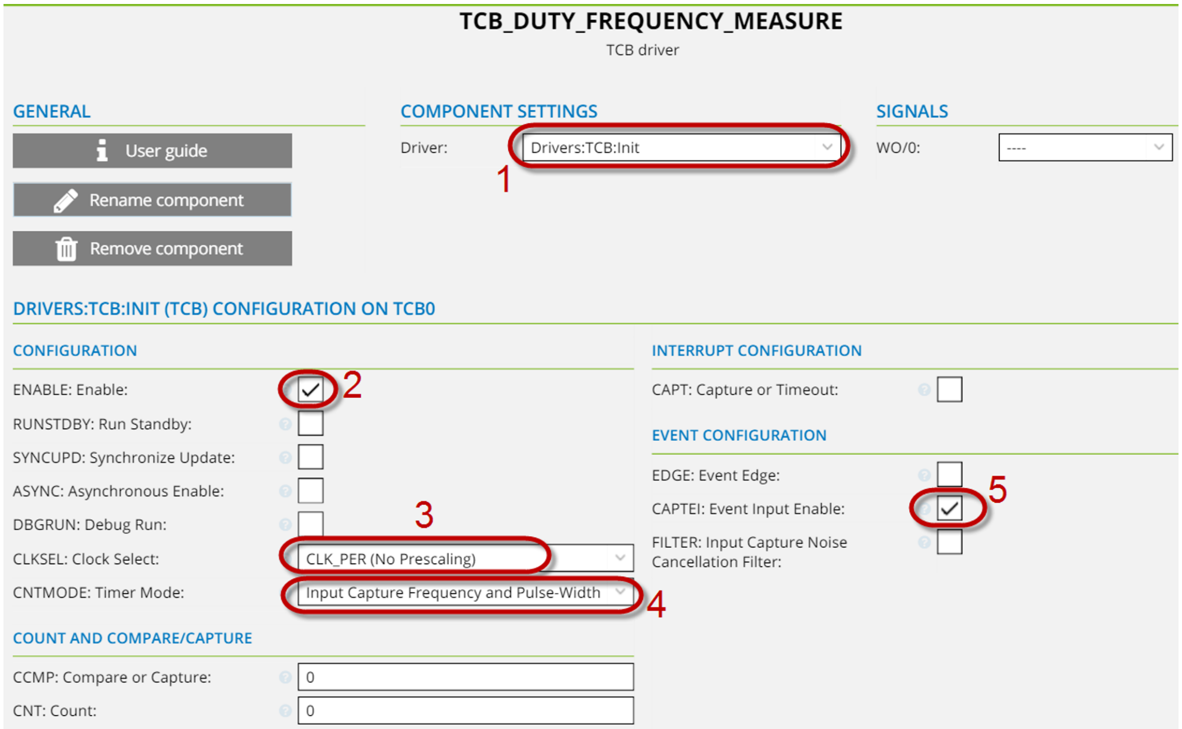
Info: Here, clock source for this peripheral is selected without prescaling. This peripheral uses the system's peripheral clock CLK_PER.

4. Select **CNTMODE: Timer mode** as **Input Capture Frequency and Pulse-Width measurement**.
5. Select the checkbox **CAPTEI: Event Input Enable**.

Info: Event Input is enabled as the PWM signal will be given as event input to TCB. In the "EVENT CONFIGURATION", EDGE is NOT selected (unchecked) so the timer will start counting when a positive edge is detected on the event input signal. On the following falling edge, the count value is captured. The counter stops when the second rising edge of the event input signal is detected. If EDGE is selected (checked) the timer will start counting when a negative edge is detected.

Note: The Event System configuration will be shown in the next section.

Figure 3-8. TCB Configuration



Result: The TCB configuration is completed.

3.4 Event System Driver

The Event System (EVSYS) 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. The mode must be selected based on the requirements of the application.

Here the PWM signal from TCA will be connected to pin PA5 and be used as event input.

To do: Add Event System driver.

1. Click on the **Add software component** box.
2. Expand **Drivers** from the **ADD SOFTWARE COMPONENT** window.
3. Search for the **Event System** driver, select it and then click the **Add Component(s)**. (Refer steps 3 and 4 from the TCA driver.)

Info: The **EVENT_SYSTEM_0** driver will be added to the **Assignment_ATtiny817** project. Now the EVENT_SYSTEM peripheral configuration needs to be done.

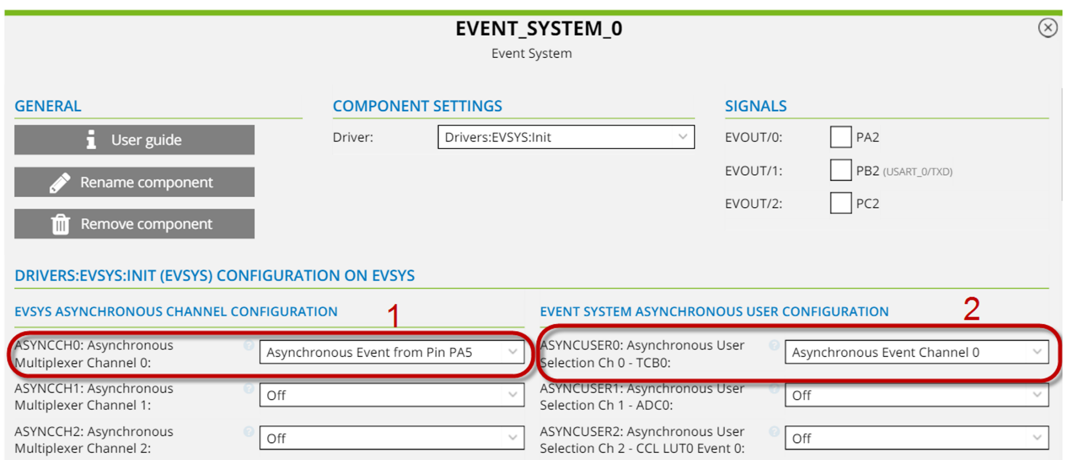
To do: Configure the Event System driver with event input at pin PA5 and event user as TCB.

Info: The EVENT_SYSTEM configuration is shown with red markings numbered 1 and 2 in the Figure 3-9.

1. Click the **EVENT_SYSTEM_0** box.
2. Select "ASYNCCH0: Asynchronous Multiplexer Channel 0" as "Asynchronous Event from pin PA5".
3. Select "ASYNCUSER0: Asynchronous User Selection Ch0 -TCB0" as "Asynchronous Event Channel 0".

Info: The Event User is TCB0 and the event input is routed through ASYNCCH0 so EVENT SYSTEM USER CONFIGURATION is selected as **Asynchronous Event Channel 0.**

Figure 3-9. EVENT SYSTEM Configuration



Result: The EVENT SYSTEM configuration is completed.

3.5 USART Driver

The peripheral USART will be configured to send the calculated duty cycle and frequency of the given input PWM signal.

The ATtiny817 Xplained Pro contains the Embedded Debugger (EDBG), composite USB device with Virtual COM Port interface. The Virtual COM Port is connected to a UART on the ATtiny817 and provides an easy way to communicate with the target application through the 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.

Virtual COM port connection: PB2 UART TXD (ATtiny817 TX line), PB3 UART RXD (ATtiny817 RX line).

To do: Add the USART driver.

1. Click the **Add software component** box.
2. Expand **Drivers** from the **ADD SOFTWARE COMPONENT** window.
3. Search for the **USART** driver, select it and then click the **Add Component(s)**. (Refer steps 3 and 4 from the TCA driver.)
   Note: Add the **USART** driver here.

Info: The **USART_0** driver will be added to the **Assignment_ATtiny817** project. Now the USART peripheral configuration needs to be done.

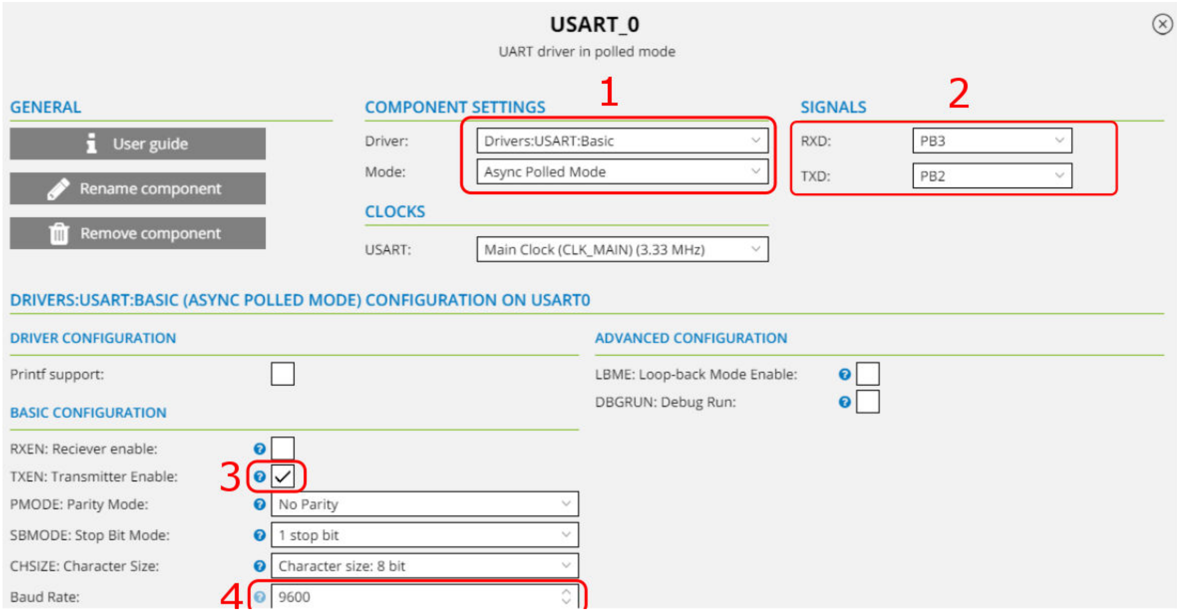
To do: Configure the USART driver for baud rate 9600, PB2:TXD, PB3:RXD.

Info: The configuration is shown with red markings numbered 1 to 3 in the Figure 3-10.

1. Under **COMPONENT SETTINGS**, set **Driver:** as **Drivers:USART_Basic** and **Mode:** as **Async Polled Mode**.
2. Under **SIGNALS**, select **RXD = PB3** and **TXD = PB2**
3. Enable Transmitter by selecting checkbox **TXEN: Transmitter Enable**. If **RXEN: Receiver enable** is checked, uncheck it
4. Select **baud rate** to **9600**.

Info: Here, by default the baud rate is 9600.

Figure 3-10. USART Configuration



Result: The USART configuration is completed.

3.6 Generate Project, Code Development

All the drivers needed have been added. The project will now be generated and code will be added to:

- Vary duty cycle of PWM in RTC ISR
- Calculate duty cycle and frequency of given PWM signal
- Send calculated duty cycle and frequency through USART

To do: Generate the project.

1. Click the 

box (at the bottom of the window).

Info: The "Project Summary" window lists files modified and files added to this reconfigured project. All the new and configured driver files will be listed here.

2. Select the main.c file and then click **OK**.

Info: Selecting the main.c file will create a new and empty main.c file.

3. Verify that all the configured drivers are initialized in the generated project, in the `system_init()` function.

Tip: src → driver_init.c → system_init()

Result: The project with configured drivers is generated.

To do: Add code to the vary duty cycle of the PWM, generated by TCA in the RTC ISR.

1. Open the driver_isr.c file.
2. Define a variable *above* the ISR, which will be changed in the ISR and initialized to 10 as initial duty cycle is configured as 10%.

```
volatile uint8_t change_duty_cycle=10;
```

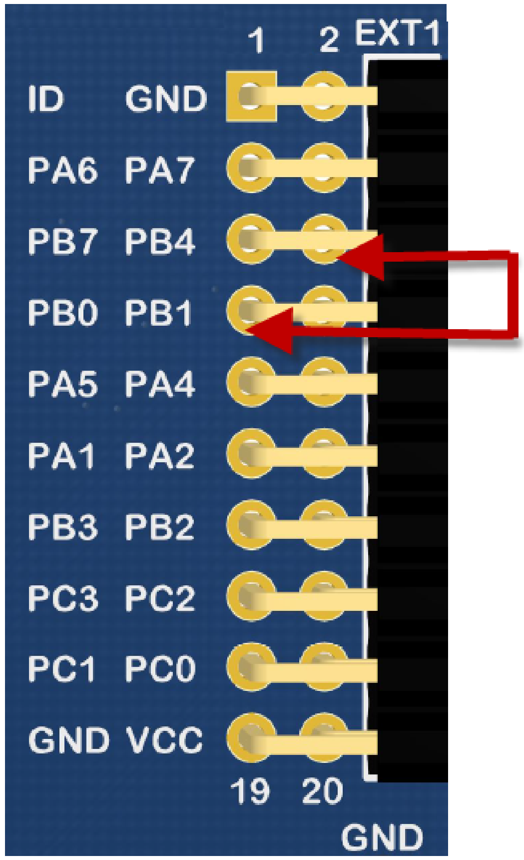
3. Add the code below *inside* the ISR to vary the duty cycle by 10% and roll over to 10% after reaching 100% in ISR.

```
change_duty_cycle+=10;
if (change_duty_cycle >100)
{
  change_duty_cycle =10;
}
 TCA0.SINGLE.CMP0 = change_duty_cycle;
```

Info: The CMP0 register decides the duty cycle of TCA PWM.

4. Press 'F7' to build the project. (Build should finish successfully with no errors.)
5. Program the updated code to the device by pressing the Ctrl + Alt + F5 keys.
6. Connect the wire from PB0 to PB4 on the ATtiny Xplained Pro board as shown in the figure below.

Figure 3-11. HW Connection



Note: On the EXT1 connecter, column '1' (top) corresponds to the left side PINs column and column '2' (bottom) corresponds to the right side PINs column so PB0 is at column 1 (top) and PB4 is at column 2 (bottom).

Info: The TCA PWM output is generated on pin PB0 and the LED is connected to pin PB4 on the ATtiny Xplained Pro board. So, connecting PB0 and PB4 will vary the LED intensity with the generated PWM.

Result: The LED is observed with varied intensity as the PWM duty cycle varies.

To do: Add code to calculate the duty cycle and frequency of the given PWM.

The mode in TCB is configured as 'Input Capture Frequency and Pulse Width Measurement Mode'.

In this mode, the timer will start counting when a positive edge is detected on the event input signal. On the following falling edge, the count value is captured. The counter stops when the second rising edge of the event input signal is detected. This will also set the interrupt flag. Reading the capture will clear the interrupt flag. When the capture register is read or the interrupt flag is cleared the TC is ready for a new capture sequence.

1. Open the main.c file.
2. Define variables to read the timer registers and store the calculated duty cycle and frequency above `main()`, as shown below:

```
volatile uint16_t period_after_capture = 0;
volatile uint16_t pulse_width_after_capture = 0;

volatile uint8_t capture_duty = 0;
volatile uint16_t capture_frequency = 0;
```

3. Add the code below in `while(1)` to calculate the duty cycle and frequency.

```
if(TCB0.INTFLAGS)
{
    TCB0.INTFLAGS=1;

    period_after_capture = TCB0.CNT;
    pulse_width_after_capture = TCB0.CCMP;

    capture_duty = ((pulse_width_after_capture * 100 )/period_after_capture);
    if (capture_duty>100)
    {
        capture_duty=0;
    }
    capture_frequency = F_CPU /period_after_capture;
}
```

Info: `TCB0.INTFLAGS` is set when captured values are available. This flag is cleared by writing 1 to it. The period is read through `TCB0.CNT`, the pulse width is read through the `TCB0.CCMP` register, and the duty cycle is calculated in %. The capture frequency is in Hz.

To do: Add code to send the calculated duty cycle and frequency of the given PWM through the USART.

As the RTC interrupt is generated every 500 ms and the duty cycle is varied every 500 ms, the code will be added to send the calculated duty cycle and frequency after every 500 ms.

1. Open the main.c file.
2. Define a variable to set the flag at every 500 ms and define the transmit buffer string, as shown below, above `main()`.

```
volatile uint8_t rtc_500ms_flg=0;
const char tx_buf[]={"\ncaptured data:"};
```

3. Open driver_isr.c. Define the `rtc_500ms_flg` as extern in the driver_isr.c file above ISR.

```
extern uint8_t rtc_500ms_flg;
```

4. Set the `rtc_500ms_flg` in RTC ISR in driver_isr.c.

```
rtc_500ms_flg=1;
```

5. In the main.c file, define function `send_data()` above `main()`, to convert the calculated duty cycle and frequency to string and send it through the USART, as shown below.

```
void send_data()
{
    uint8_t i=0;
    char duty_str[4]={0},freq_str[10]={0},tx_data[30]={0};

    itoa(capture_duty, duty_str, 10);      //duty cycle to ASCII
    ltoa(capture_frequency, freq_str, 10); //frequency to ASCII

    strcat(tx_data,tx_buf);         //tx_data=tx_buf+duty cycle string+frequency string
    strcat(tx_data,duty_str);
    strcat(tx_data,"% ");
    strcat(tx_data,freq_str);
    strcat(tx_data,"Hz");

    while(tx_data[i] !=0)             //check for null character
    {
        USART_0_write(tx_data[i]);      // send data
        i++;
    }
}
```

6. Include the file string.h at the top of the file.

```
#include <string.h>
```

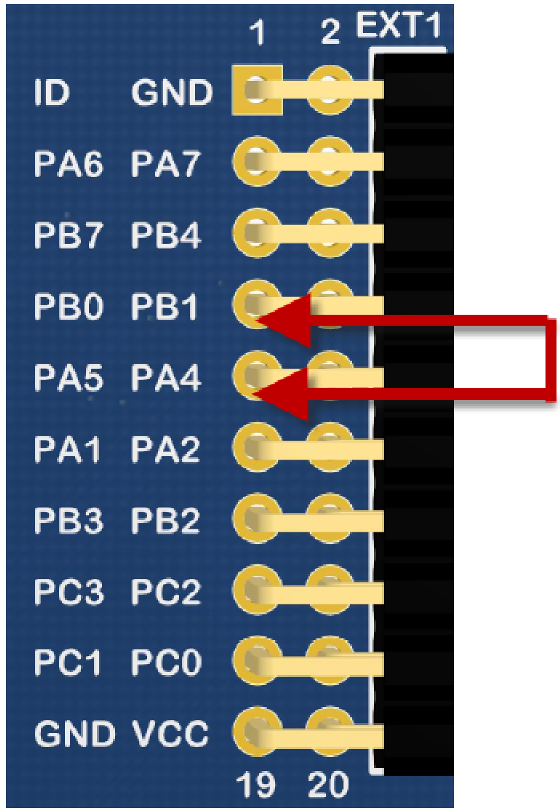
Info: The `itoa` and `ltoa` functions defined in string.h converts the duty cycle and frequency to ASCII. The `strcat` function concatenates the `tx_buf`, the duty cycle string, the frequency string, and the resulting string with the `tx_data` string will be sent to the terminal. The `USART_0_putc` function writes data to the TXDATA USART register. TXDATA can only be written when the Data Register Empty Flag (DREIF in USART.STATUS) is set, indicating that the register is empty and ready for new data.

7. Add code in `while(1)` to call the `send_data()` function after every 500 ms.

```
if (rtc_500ms_flg==1)
{
     rtc_500ms_flg=0;
     send_data();
}
```

8. Press 'F7' to build the project. (Build should finish successfully with zero errors.)
9. Program the device with the updated code by selecting **Debug** → **Start without Debugging**. (Alternative Ctrl + Alt + F5.)
10. Connect the wire from PB0 to PA5 on the ATtiny Xplained Pro board as shown in the figure below.

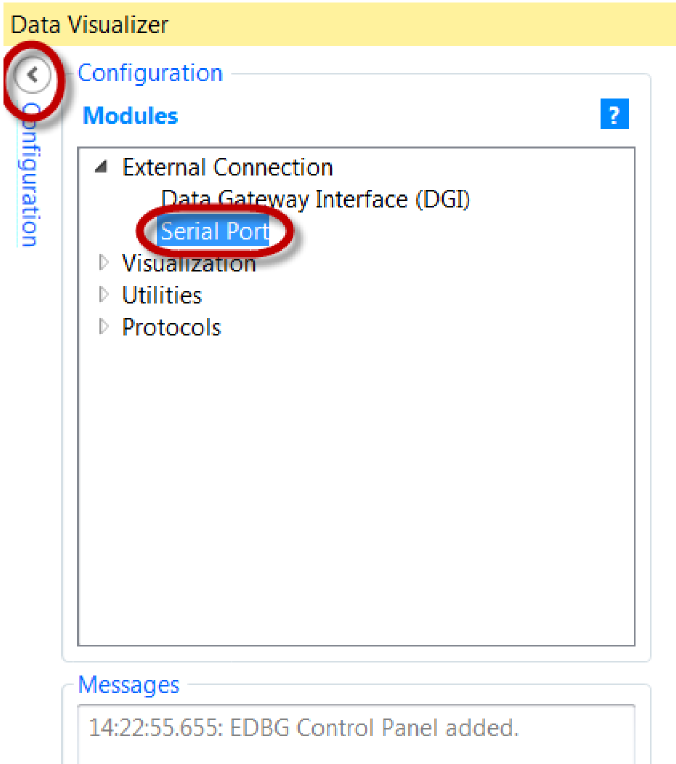
Figure 3-12. HW Connection PB0 to PA5



Info: TCB has been configured to capture the PWM signal at PA5, and TCA PWM is generated at PB0.

11. In Atmel Studio 7, from the menu, select **Tools** → **Data Visualizer**.

12. In the Data Visualizer window, click the **Configuration** tab and then double click the **Serial Port**.



13. In the "Serial Port Control Panel" select the **EDBG Virtual COM Port** number listed and set **Baud rate** to **9600**. Click the **Open Terminal** button then **Connect.**

Tip: The EDBG Virtual COM Port number of the connected ATtiny Xplained Pro board is also listed in **Start** → **Control Panel** → **Device Manager** → **Ports.**

Result: The captured duty cycle and frequency of the given PWM signal is displayed in the terminal window.

```
captured data:49% 33333Hz
captured data:59% 33333Hz
captured data:69% 33333Hz
captured data:79% 33333Hz
captured data:89% 33333Hz
captured data:99% 33333Hz
captured data:9% 33333Hz
captured data:19% 33333Hz
captured data:29% 33333Hz
captured data:39% 33333Hz
captured data:49% 33333Hz
captured data:59% 33333Hz
captured data:69% 33333Hz
captured data:79% 33333Hz
captured data:89% 33333Hz
captured data:99% 33333Hz
captured data:9% 33333Hz
```

The final code should look as shown below.

```
#include <string.h>
#include <atmel_start.h>

volatile uint16_t period_after_capture = 0;
volatile uint16_t pulse_width_after_capture = 0;
volatile uint8_t capture_duty = 0;
volatile uint16_t capture_frequency = 0;

volatile uint8_t rtc_500ms_flg=0;
const char tx_buf[]={"\ncaptured data:"};


void send_data()
{
    uint8_t i=0;
    char duty_str[4]={0},freq_str[10]={0},tx_data[30]={0};
    itoa(capture_duty, duty_str, 10); //duty cycle to ASCII
    ltoa(capture_frequency, freq_str, 10); //frequency to ASCII
    strcat(tx_data,tx_buf); //tx_data=tx_buf+duty cycle string+frequency string
    strcat(tx_data,duty_str);
    strcat(tx_data,"% ");
    strcat(tx_data,freq_str);
    strcat(tx_data,"Hz");
    while(tx_data[i] !=0) //check for null character
    {
        USART_0_write(tx_data[i]); // send data
        i++;
    }
}

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

    /* Replace with your application code */
    while (1) {
        if(TCB0.INTFLAGS)
        {
            TCB0.INTFLAGS=1;
            period_after_capture = TCB0.CNT;
            pulse_width_after_capture = TCB0.CCMP;
            capture_duty = ((pulse_width_after_capture * 100 )/period_after_capture);
            if (capture_duty>100)
            {
                capture_duty=0;
            }
            capture_frequency = F_CPU /period_after_capture;
        }
        if (rtc_500ms_flg==1)
        {
            rtc_500ms_flg=0;
```

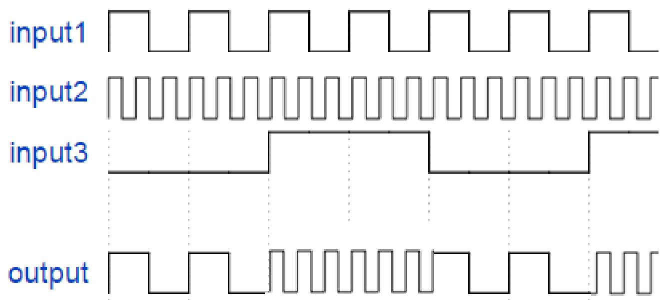
```
            send_data();
        }
    }
}
```

4. Assignment 3: Basis of a Binary Frequency-Shift Keying Scheme

The CCL (Configurable Custom Logic) module is a programmable logic peripheral, which can be connected to the pins, events, or peripherals on the device. It allows the user to eliminate external logic gates for simple glue logic functions.

In this assignment, a small CCL-based application will be developed. It will blink an LED at two different frequencies depending on whether a button is depressed or not. Two pulse trains of different frequencies will be generated and routed to the CCL. The CCL will be configured to select which of the pulse trains to pass on to the output based on the state of a third input signal as shown in the figure below.

Figure 4-1. CCL Input/Output Waveform



Here, output = input1 when input3 is LOW, and output = input2 when input3 is HIGH.

The application can be the basis of a binary frequency-shift keying scheme that encodes binary data as discrete shifts between two frequencies.

So, in this application:

- "input1" is the PWM signal generated using TCA in Assignment 2
- "input2" event output from PIT
- "input3" is button state (SW0)
- "output" is CCL output on pin PA7

The LED will be connected to PA7 and blink with different frequencies on button press and button release.

The RTC peripheral offers two timing functions; the Real-Time Counter (RTC) and a Periodic Interrupt Timer (PIT). The PIT functionality can be enabled independent of the RTC functionality.

By using the same clock source as the RTC function, the PIT can request an interrupt or trigger an output event on every n^{th} clock period. n can be selected from {4, 8, 16,.. 32768}. Here the Event System will be configured to output events from the PIT. The event signals from the PIT has the form of clock signals with periods corresponding to the respective number of the RTC clock periods. This application uses the event signal corresponding to 8192 RTC clock periods. The event signal from the PIT has a frequency of 32 kHz/8192 = 3.9 Hz (period approximately 250 ms).

The **Assignment_ATtiny817** project will be reconfigured to add the CCL driver and edit the Event System and RTC driver.

For this application, the peripherals used are:

- TCA (waveform output WO0 on PB0)
- PIT (event output)
- CCL (output PA7)
- Event System

- PB5 (SW0)

Clock details:

- 3.33 MHz main clock
- PIT 32 kHz

4.1 CCL Driver

The Configurable Custom Logic (CCL) is a programmable logic peripheral, which can be connected to the device pins, to events, or to other internal peripherals. The CCL can serve as "glue logic" between the device peripherals and external devices.

The CCL peripheral has one pair of Lookup Tables (LUT). Each LUT consists of three inputs, a truth table, and a filter/edge detector. Each LUT can generate an output as a user programmable logic expression with three inputs. The output can be generated from the inputs with different combinations.

In this application LUT1 will be used.

- "input1" is PWM generated using TCA: (WO0 of TCA)
- "input2" is event output from PIT: Event Source 1
- "input3" is button press (SW0): Event Source 0
- "output" is CCL output on pin PA7
  The Truth Table for the needed combination is shown in the table below. IN[2] is button (SW0):

  when IN[2] = 0, OUTPUT= IN[1] and when IN[2] =1, OUTPUT = IN[0].

  So, OUT= 0xAC.

Table 4-1. Truth Table of LUT

IN[2]	IN[1]	IN[0]	OUT
0	0	0	0 (LSB)
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

To do: Add the CCL driver.

1. Open the **Assignment_ATtiny817** project.
2. In the **Solution Explorer** window, click on the project name **Assignment_ATtiny817** and then **Right Click** → **Re-Configure Atmel Start Project**.
3. In the "Atmel | START" window, click on the Add software component box and then expand **Drivers** from the **ADD SOFTWARE COMPONENT** window.

4. Search for the **Digital Glue Logic (CCL)** driver, select it and then click on the **Add Component(s)**. (Refer steps 3 and 4 from the TCA driver.)

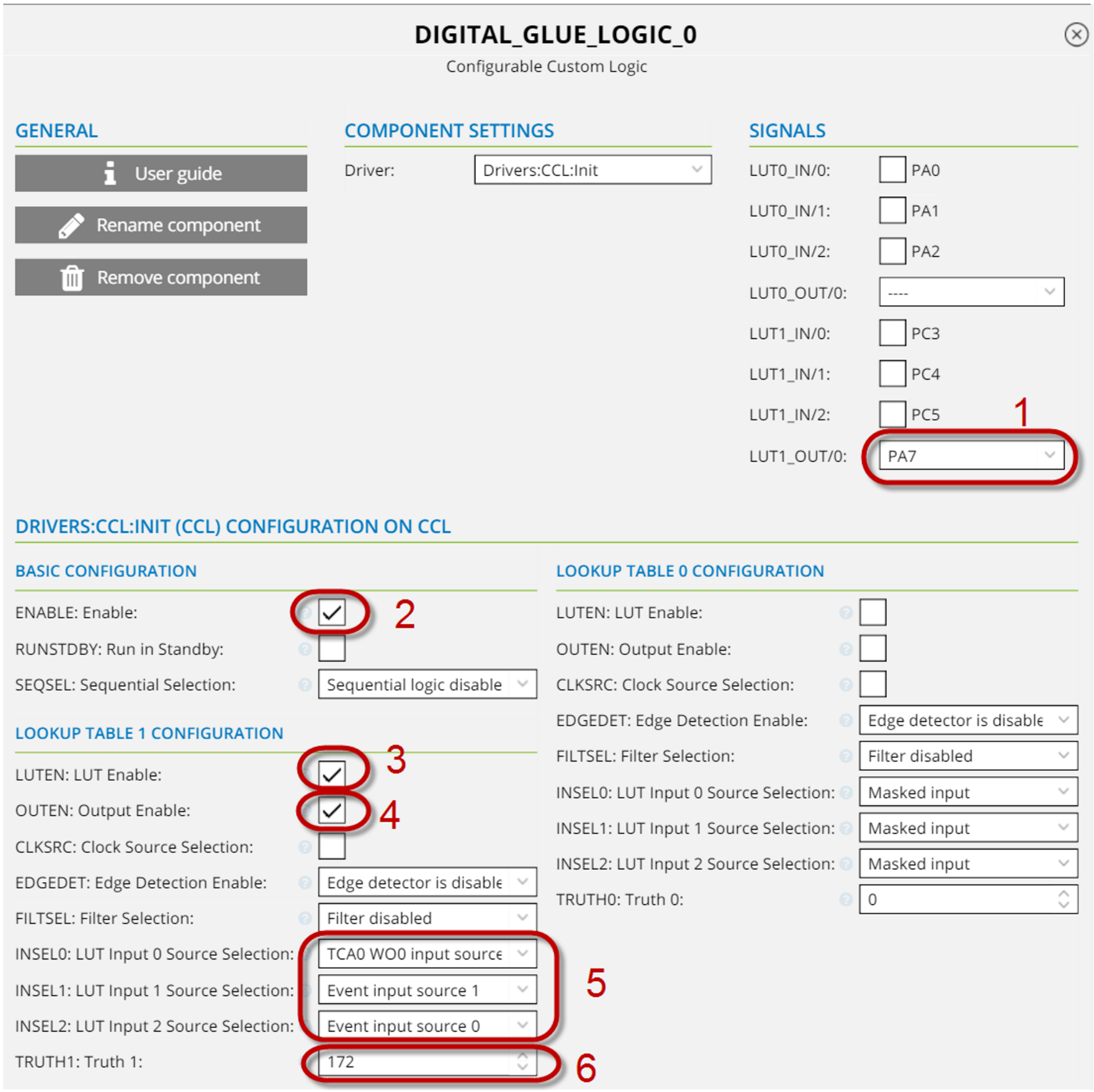
Info: The **DIGITAL_GLUE_LOGIC_0** driver will be added to the **Assignment_ATtiny817** project. Now the CCL configuration needs to be done as below.

To do: Configure the CCL driver to use with IN[2] = Event Source 0, IN[1]= Event Source 1, IN[0] = WO0 TCA and output on PA7.

Info: The configuration is shown with red markings numbered 1 to 6 in the Figure 4-2.

1. Select **LUT1_OUT/0** on pin **PA7**.
2. Enable peripheral by selecting check-mark **ENABLE: Enable**.
3. Select checkbox **LUTEN: LUT Enable** under **LOOKUP TABLE 1 CONFIGURATION**.
4. Select checkbox **OUTEN: Output Enable** under **LOOKUP TABLE 1 CONFIGURATION**.
5. Select Input as:
   – **INSEL0:LUT input 0 source selection: = TCA WO0 input source**
   – **INSEL1:LUT input 1 source selection: = Event input source 1**
   – **INSEL2:LUT input 2 source selection: = Event input source 0**
6. Enter truth table value **TRUTH1: Truth 1** as **172** (or 0xAC). (Check OUT column from Table 4-1.)

Figure 4-2. CCL Configuration



Result: The CCL configuration is completed.

4.2 Event System Driver

The Event System (EVSYS) 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.

Here the event signal will be generated from the SW0 (PB5) push button and the PIT (Periodic Interrupt Timer).

Table 4-2. Event User and Generator

Event User	Event Generator
CCL LUT1 Event 0	Event from PB5
CCL LUT1 Event 1	Event from PIT : 8192 RTC clock periods interval.

To do: Click the **EVENT_SYSTEM_0** box and configure the EVENT_SYSTEM as per the Table 4-2.

Info: The configuration is shown with red markings numbered 1 to 4 in the Figure 4-3.

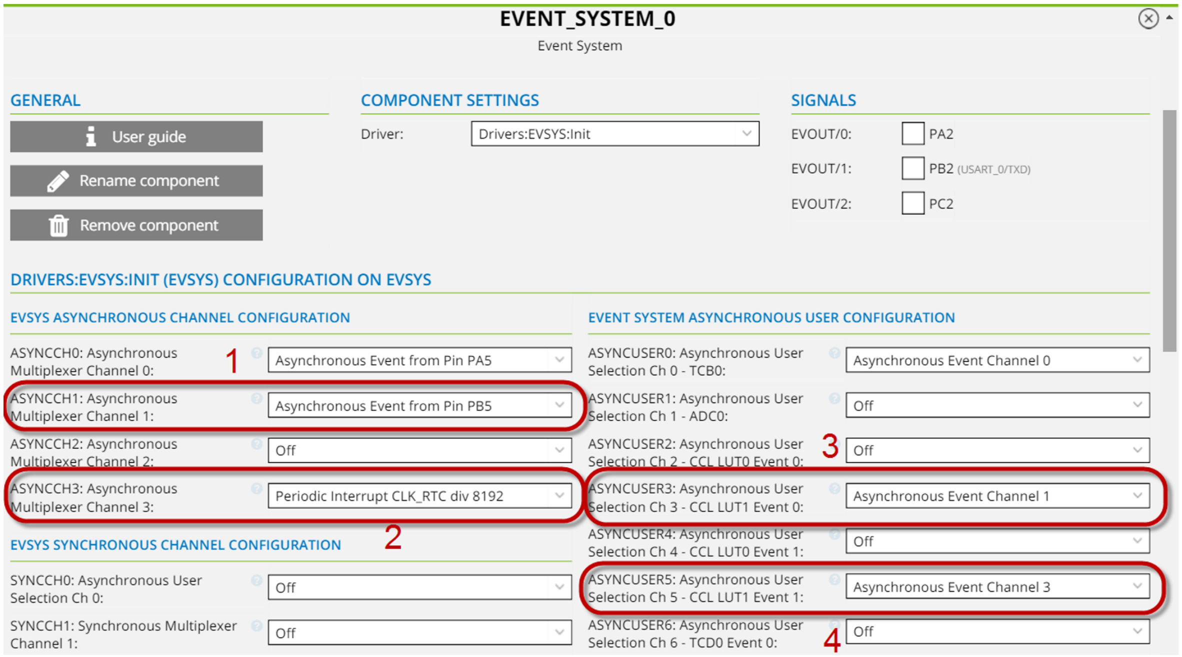
1. Select "ASYNCCH1: Asynchronous Multiplexer Channel 1" as "Asynchronous Event from pin PB5".
2. Select "ASYNCCH3: Asynchronous Multiplexer Channel 3" as "Periodic Interrupt CLK_RTC div 8192".
3. ASYNCUSER3: Asynchronous User Selection Ch3 -CCL LUT1 Event 0" as "Asynchronous Event Channel 1".

   Info: Event User is **CCL LUT1 Event 0** so the ASYNCUSER3 user configuration is through ASYNCCH1.

4. "ASYNCUSER5: Asynchronous User Selection Ch5 - CCL LUT1 Event 1" as "Asynchronous Event Channel 3".

   Info: Event User is **CCL LUT1 Event 1** so the ASYNCUSER5 user configuration is through ASYNCCH3.

Figure 4-3. EVENT SYSTEM Configuration



Result: The EVENT SYSTEM configuration is completed.

4.3 PIT Driver

The RTC peripheral offers two timing functions; the Real-Time Counter (RTC) and a Periodic Interrupt Timer (PIT). The PIT functionality can be enabled independent of the RTC functionality.

To do: Edit the RTC driver to enable PIT.

1. Click **RTC_0**.
2. Scroll down and configure RTC to enable PIT. Check the box next to PITEN.

Result: PIT is enabled.

4.4 Generate Project, Run Code

All the needed drivers have been added and the project can now be generated.

To do: Generate the project.

1. Click the 

box (at the bottom of the window).

   Info: A "Project Summary" window lists files modified and files added on this reconfigured project. All the new and configured driver files will be listed here.

2. In the "Project Summary" window, click **OK**.

   Info: If the main.c and driver_isr.c files are selected, any earlier code will be overwritten and these two files will be created as empty files.

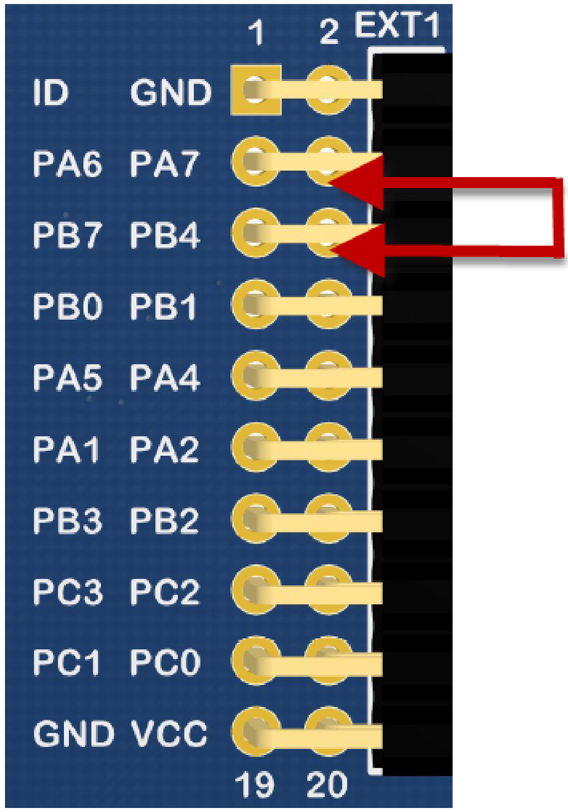
3. Verify that all the configured drivers are initialized in the generated project in folder 'src', file 'driver_init.c', in function `system_init()`.

Result: The project with the configured drivers is generated.

To do: Run the code.

1. Press 'F7' to build the project. (Build should finish successfully with no errors.)
2. Program the updated code to the device by selecting **Debug** → **Start without Debugging**.
3. Connect the wire from PA7 to PB4 on the ATtiny Xplained Pro board, as shown in the figure below.

Figure 4-4. HW Connection



Note: On the EXT1, column '1' (top) corresponds to the left side PINs column and column '2' (bottom) corresponds to the right side PINs column so PA7 and PB4 both are in column 2 (bottom).

Info: The CCL output is generated on pin PA7 and the LED is connected to pin PB4 on the ATtiny Xplained Pro board. The LED blinking with different frequencies can be observed by connecting PB4 and PA7.

4. Press the push button SW0 and observe the LED, release the push button and observe the LED.

Result: When the push button is released the LED blinks with TCA PWM that is the duty cycle is changing after every 500 ms and LED intensity varies. When the push button is pressed the LED blinks with a PIT RTC_clock /8192 period interval.

Info: This application can be made core independent if the RTC OVF interrupt is disabled. To view the LED blinking, edit the TCA driver to generate the PWM signal such that the LED blinking is visible. (E.g. 12 Hz signal with 50% duty cycle.)

5. Conclusion

This training demonstrated:

- The driver configuration through Atmel START
- The use of automatically generated driver functions in the code development
- Re-configuration of the project to edit and add drivers

You also learned:

- The different peripherals of tinyAVR 1-series
- How to use Event System to generate Event
- How to use CCL to generate output

With Atmel Studio it is easy to run real-time debugging of an application and use the 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
- Single Stepping
- I/O view

6. 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

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

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

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

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

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

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

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

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

ISBN: 978-1-5224-1997-6

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	**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