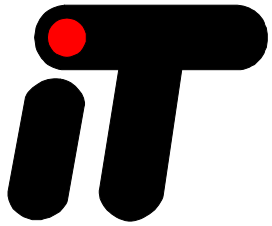


HP InfoTech

# Getting Started with the CodeVisionAVR Extension for Atmel Studio 6.1

Getting Started with the CodeVisionAVR Extension for Atmel Studio 6.1
Revision 1/03.2013

# Table of Contents

# 1. Introduction

The purpose of this document is to guide the user through the preparation, building and debugging of an example C program using the CodeVisionAVR V3.03 or later C compiler extension for Atmel Studio 6.1 or later.
The example is a simple program for the Atmel ATmega328 microcontroller on an Arduino UNO board.

# 2. Preparation

Download and install Atmel Studio from www.atmel.com
Install the CodeVisionAVR C Compiler by executing the *CodeVisionAVR.msi* installer.
When prompted, use the default installation directory suggested by the installer.
Please note that Administrator privileges are required under Windows for installing and using CodeVisionAVR.

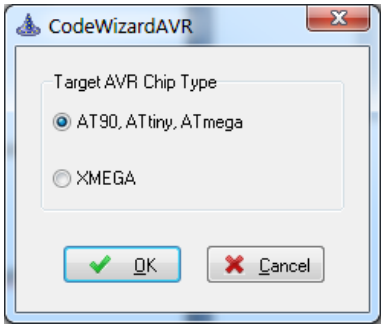
Make the following hardware preparations:

- If it is not already installed, solder a 6 pin header to the location marked ICSP on the Arduino UNO board
- Connect the cathodes of 8 LEDs to the outputs marked DIGITAL 0..7 on the board. These outputs correspond to PORTD pins PD0..PD7 of the microcontroller.
- Connect each LED's anode, using a 1k resistor, to the pin marked 5V of the board's POWER connector header.
- Connect the AVRISP MkII programmer to an USB port of your computer
- Connect the AVRISP MkII programmer to the ICSP header. Please check the correct orientation of the programmer's cable connector with the board's ICSP header pin 1.
- Connect the USB connector of the Arduino board to another USB port of your computer. This will provide power supply to the board.

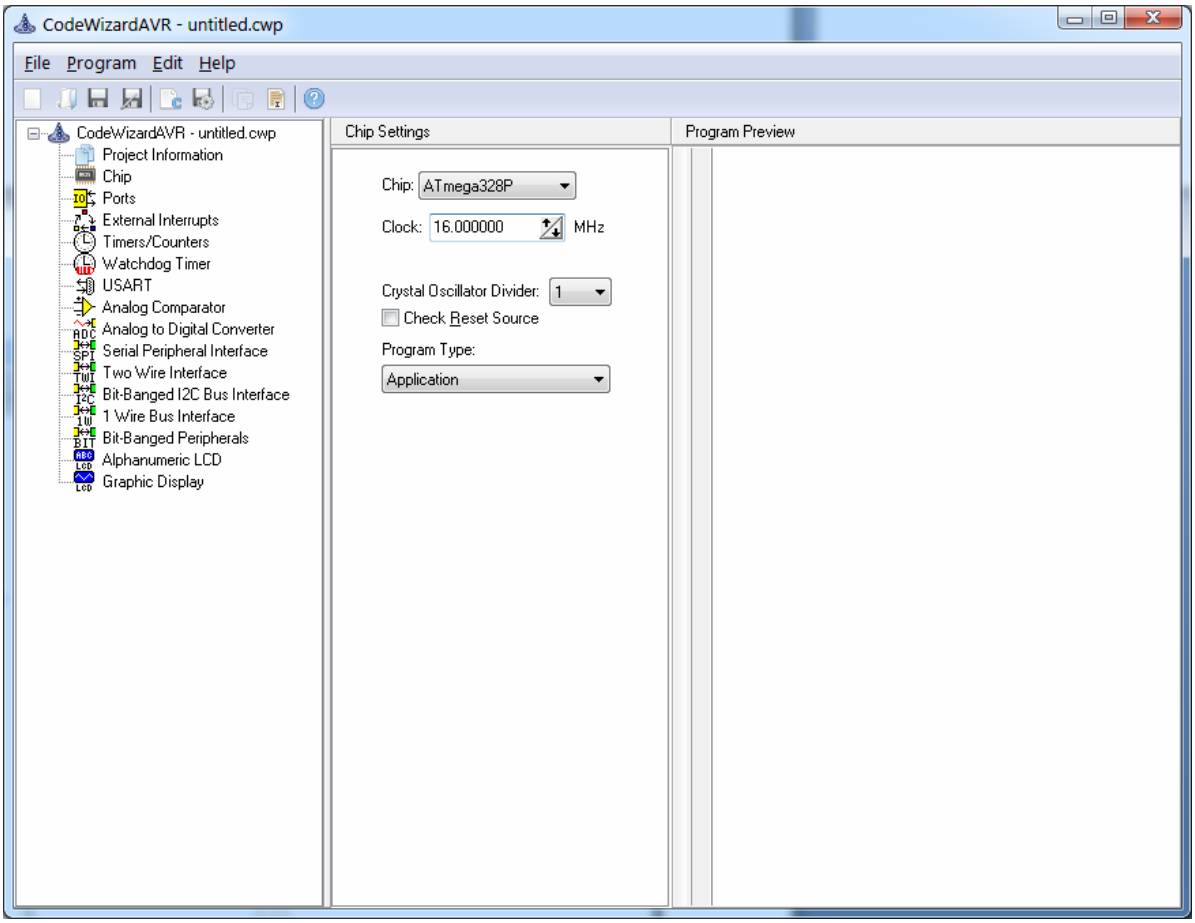
 

# 3. Creating a New Project

Launch the Atmel Studio IDE.
Execute the **File|New|Project Using the CodeWizardAVR...** menu command.
A dialog window will be displayed, allowing selecting the AVR chip family for which the CodeWizardAVR will create the program:

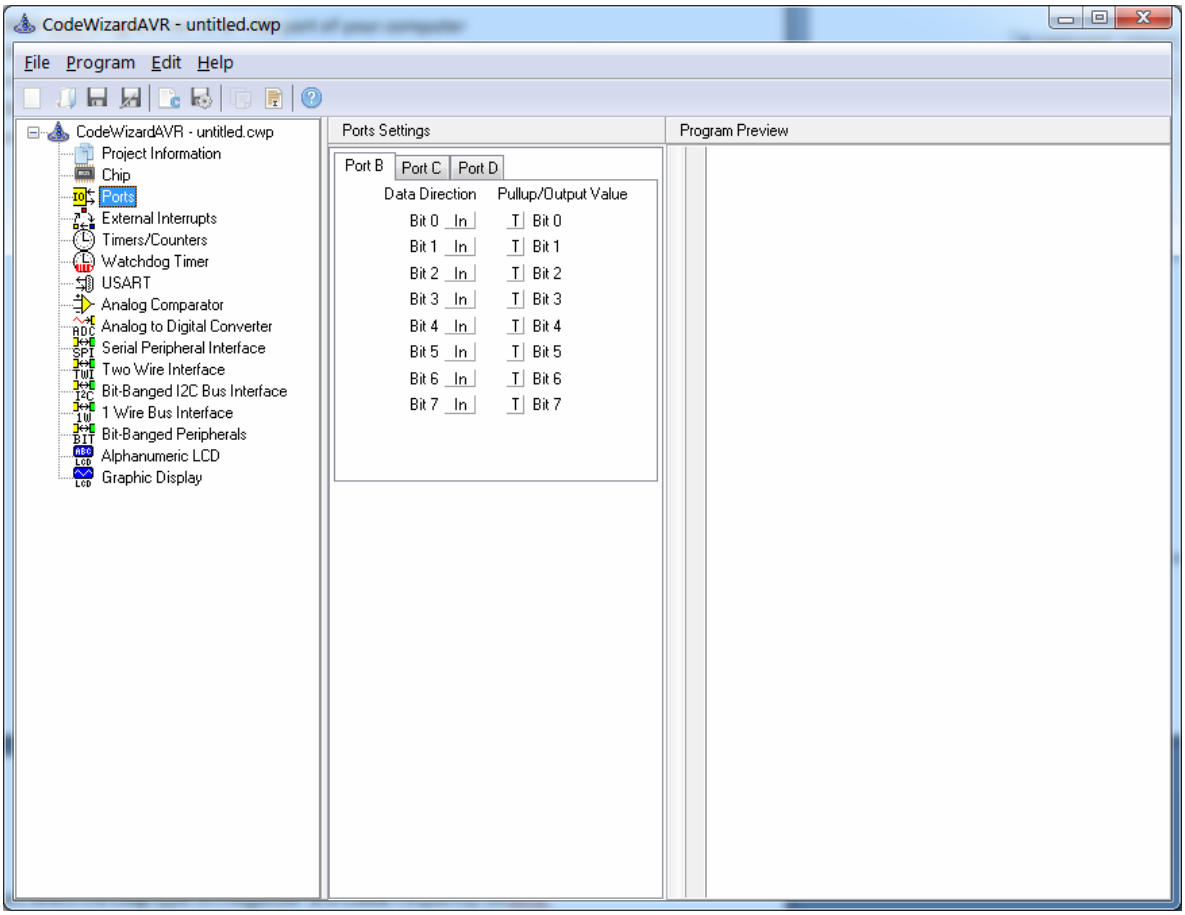



Select the **AT90, ATtiny, ATmega** option and click on the OK button.
The CodeWizardAVR will be launched and the following window will be displayed:

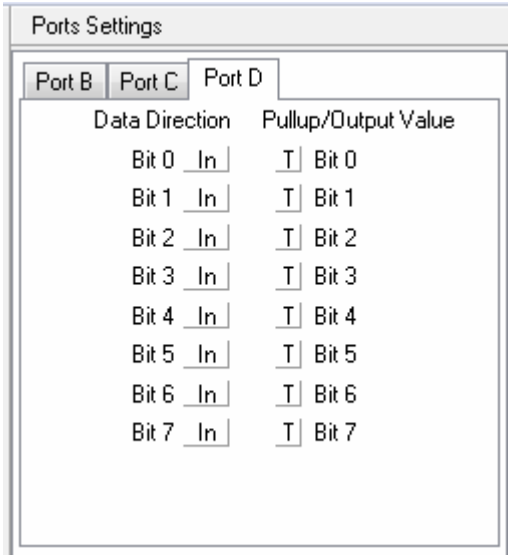



In the **Chip Settings** panel, select the **Chip** type: ATmega328P and **Clock** frequency: 16 MHz.

The next step is to configure the PORTD pins PD0 to PD7 as outputs.
In order to achieve this, click on the **Ports** node of the CodeWizard's tree.
A new configuration panel for **Port Settings** will be displayed:



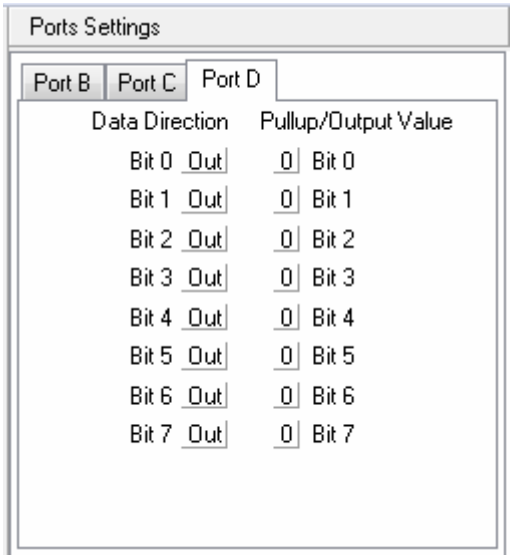

Click on the **Port D** tab in order to select the PORTD configuration:



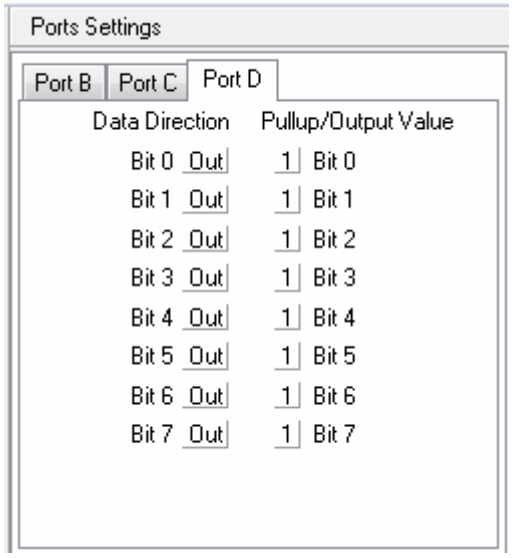

As can be seen, the Port D **Data Direction** for all I/O pins is set by default as inputs (**In**).

Click on each **Bit 0** to **Bit 7** button in order to set the I/O pins as outputs:

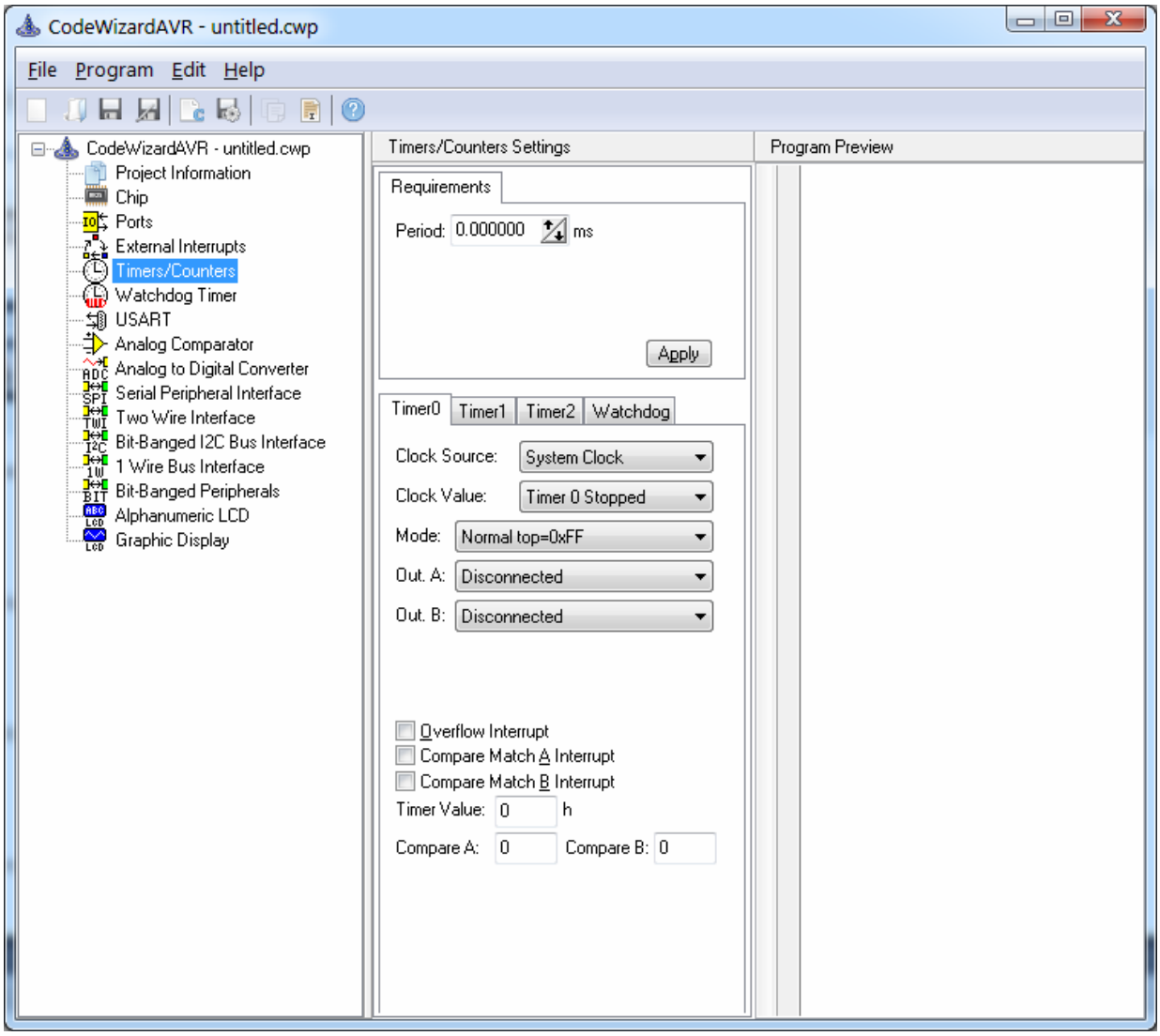



As the LEDs must be OFF after chip reset, the potential of their cathodes must be +5V, so the **Output Values** for Port D **Bit 0** to **Bit 7** must be set to 1 by clicking on the corresponding buttons:

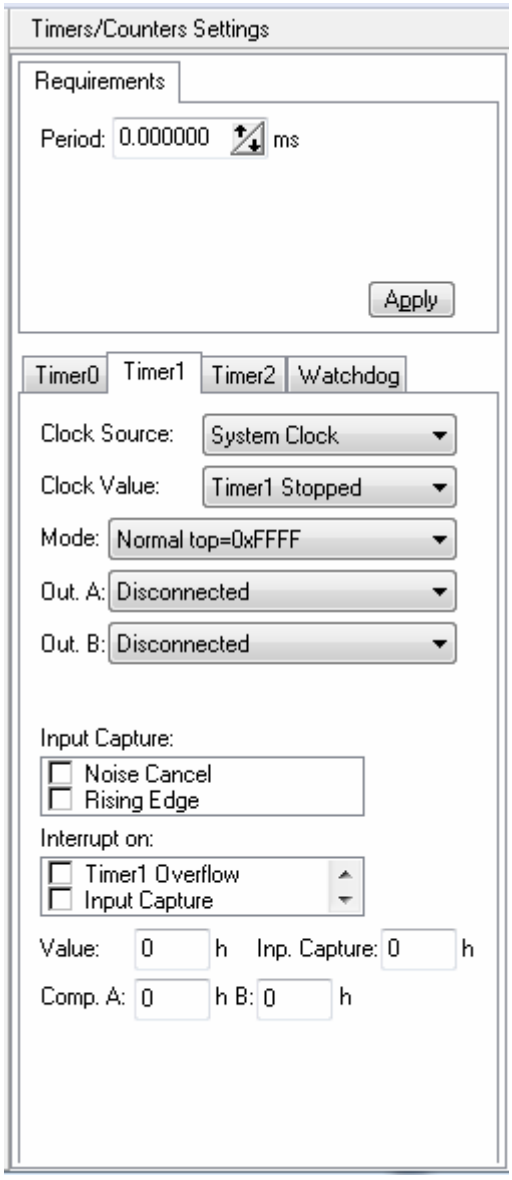


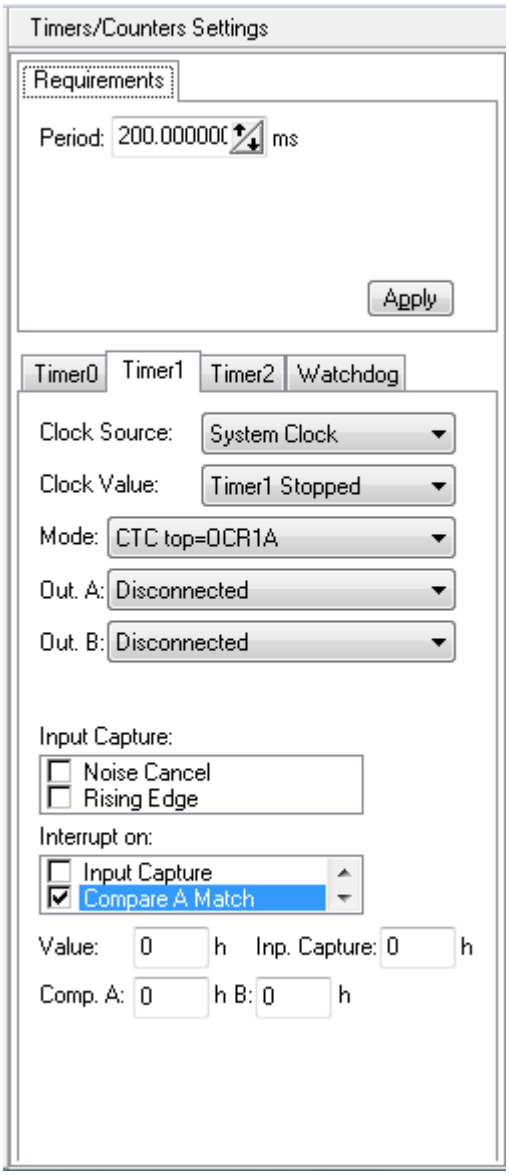
The next step is to configure a Timer/Counter to generate an interrupt after each 200 ms.
Click on the **Timers/Counters** node of the CodeWizard's tree.
A **Timers/Counters Settings** panel will be displayed:



**Timer1** will be used, so click on the corresponding tab:



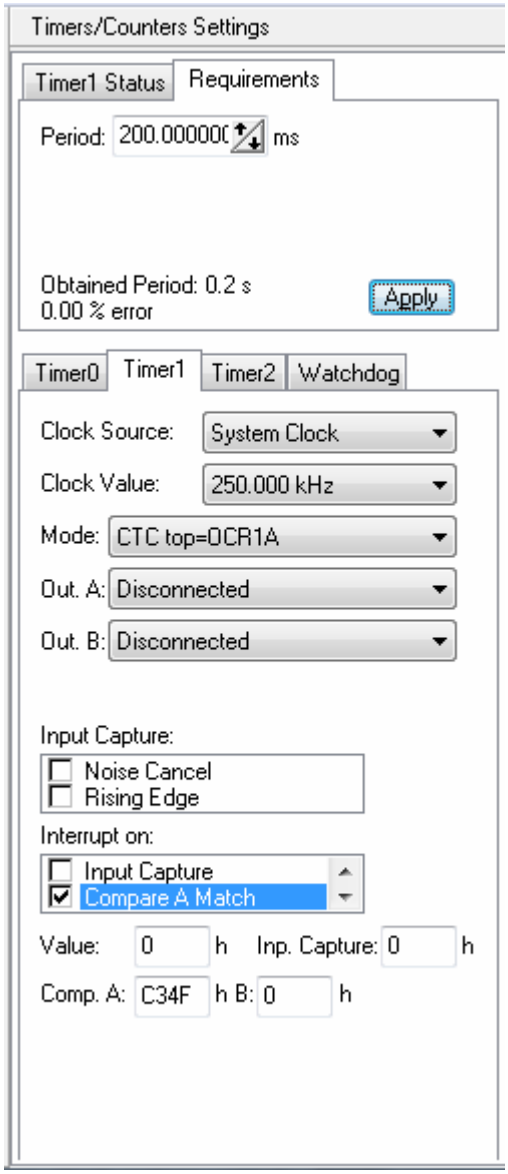
As we need a Timer 1 compare match interrupt after each 200 ms, we must select the operating **Mode** as: **CTC top=OCR1A,** specify the **Period** value: *200 ms* in the **Requirements** panel and check the **Interrupt on Compare A Match** check box:




In this operating mode the Timer 1 will count pre-scaled system clock pulses until the TCNT1 register will be equal with the value of the OCR1A register. When this will occur, the TCNT1 register will be automatically reset to 0 and a Timer 1 compare with OCR1A match interrupt will be generated.

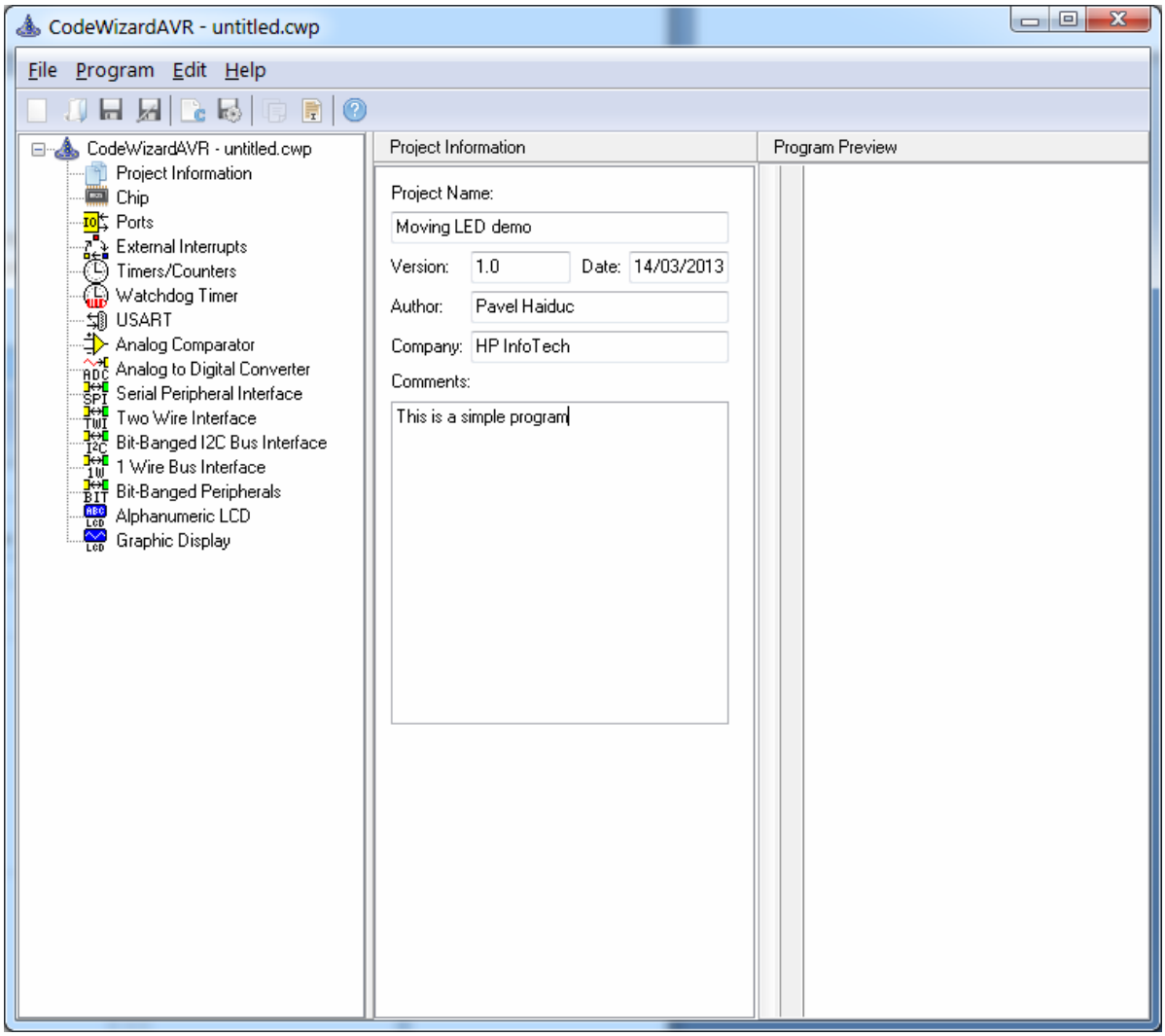
 

By clicking on the **Apply** button in the **Requirements** panel, the CodeWizardAVR will establish the required values for Timer 1 configuration registers:

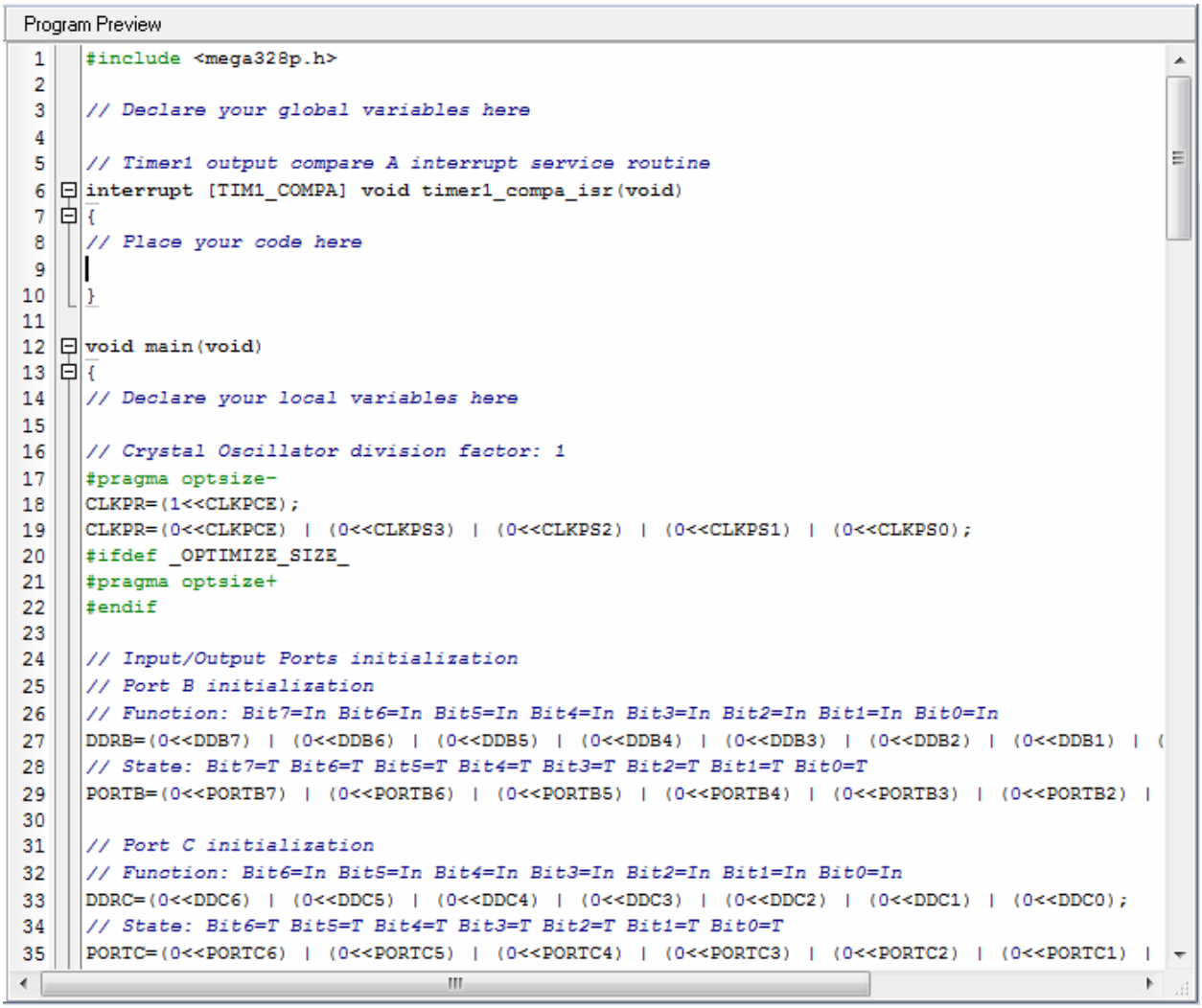



As can be seen in the above window, the 16 MHz system clock will be divided by 64 in order to obtain a Timer 1 **Clock Value** of 250 kHz and the OCR1A register will be initialized with the value *0xC34F*.
The obtained time period between two interrupts will be 0.2 seconds, matching our requirement with a 0% error.

The next step, before generating the actual program code, is to specify some comments regarding our program by clicking on the **Project Information** node and completing the comments in the corresponding panel:



Using the **Program|Generate** menu or clicking on the toolbar button, will create the C program, which can be previewed in the **Program Preview** window:

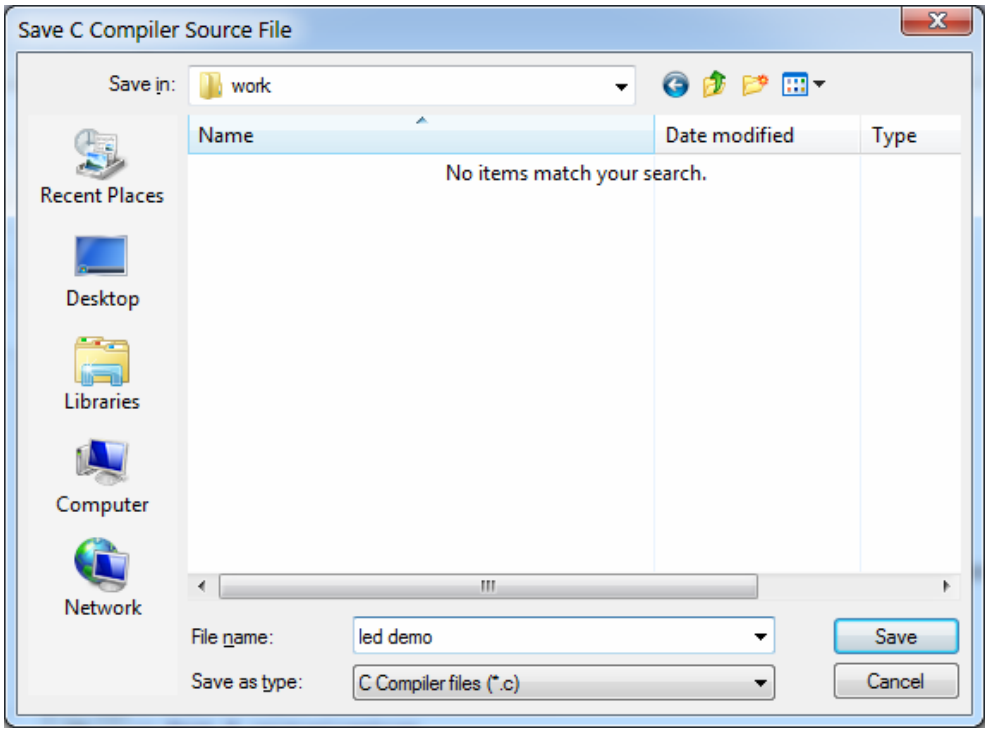



By clicking on a peripheral node in the CodeWizard's tree, the cursor in the **Program Preview** window will be positioned at the corresponding initialization code sequence for that peripheral.

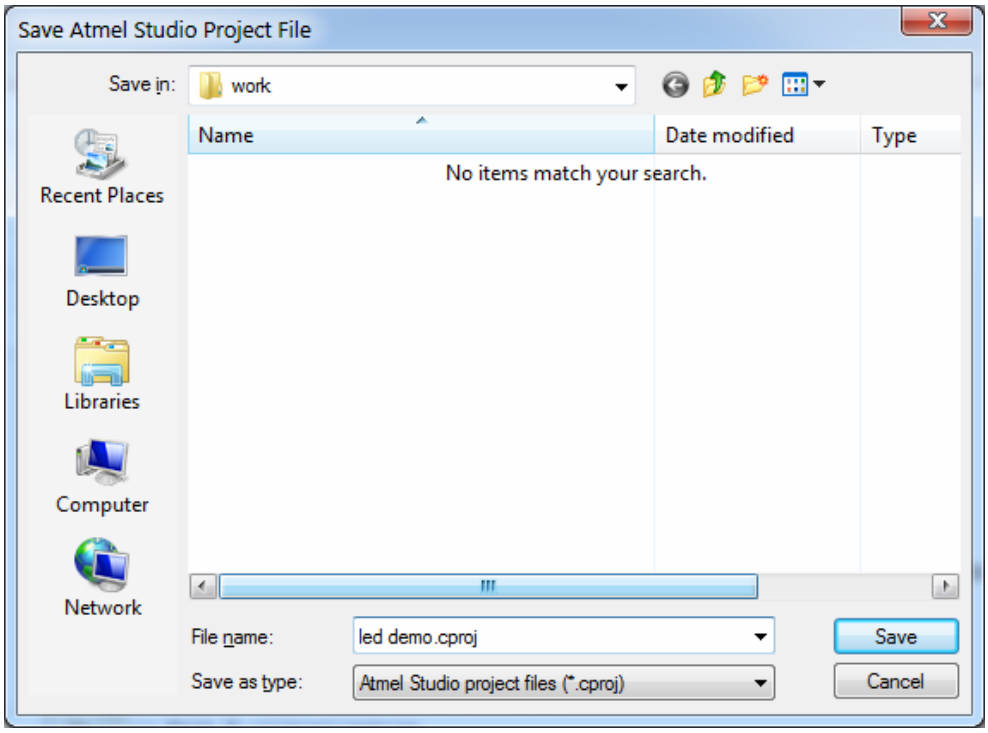
**Note:** By default the CodeWizardAVR generates initialization code even for peripherals that are not in use (disabled).
This is a safety measure to configure correctly the chip if a software reset occurred by jumping to address 0.
In order to reduce generated program size, this can be disabled by un-checking the **Program|Generate Code for Disabled Peripherals** menu option.

Once we are satisfied with the generated code, we must save the new program by using the **Program|Generate, Save and Exit** menu or clicking on the toolbar button.

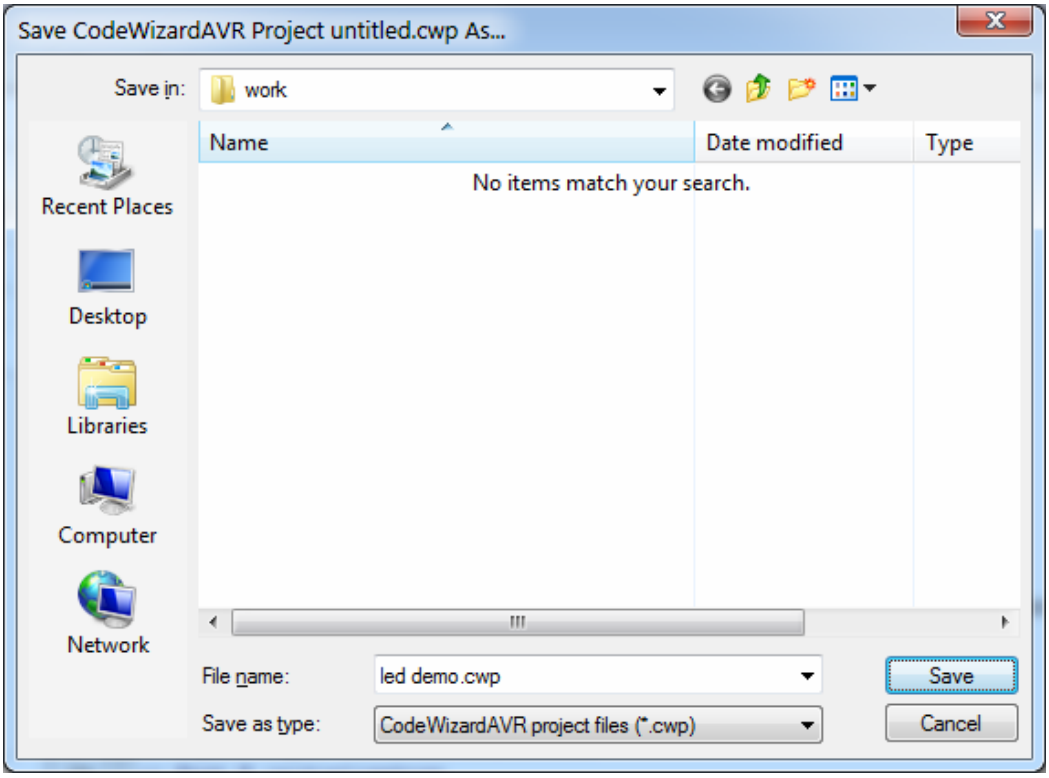
First we will be prompted for the name of the first **.c** source file of the project:




Next we will have to specify the name of the Atmel Studio project:

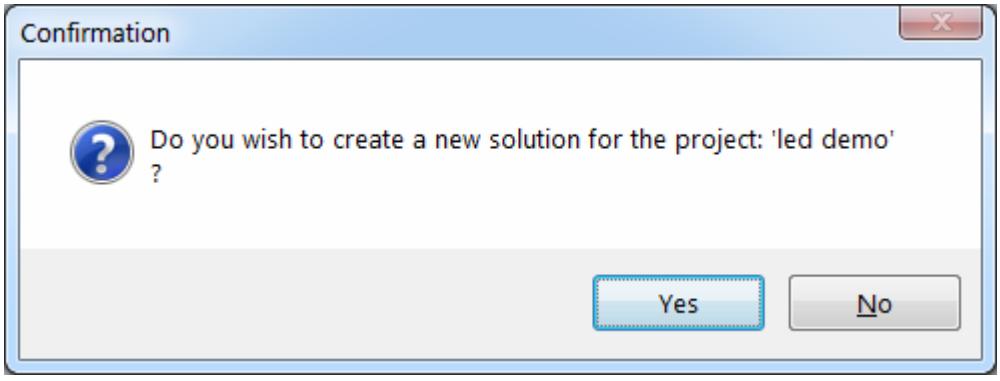


And finally we must save the peripheral configuration for our project in a CodeWizardAVR project **.cwp** file:

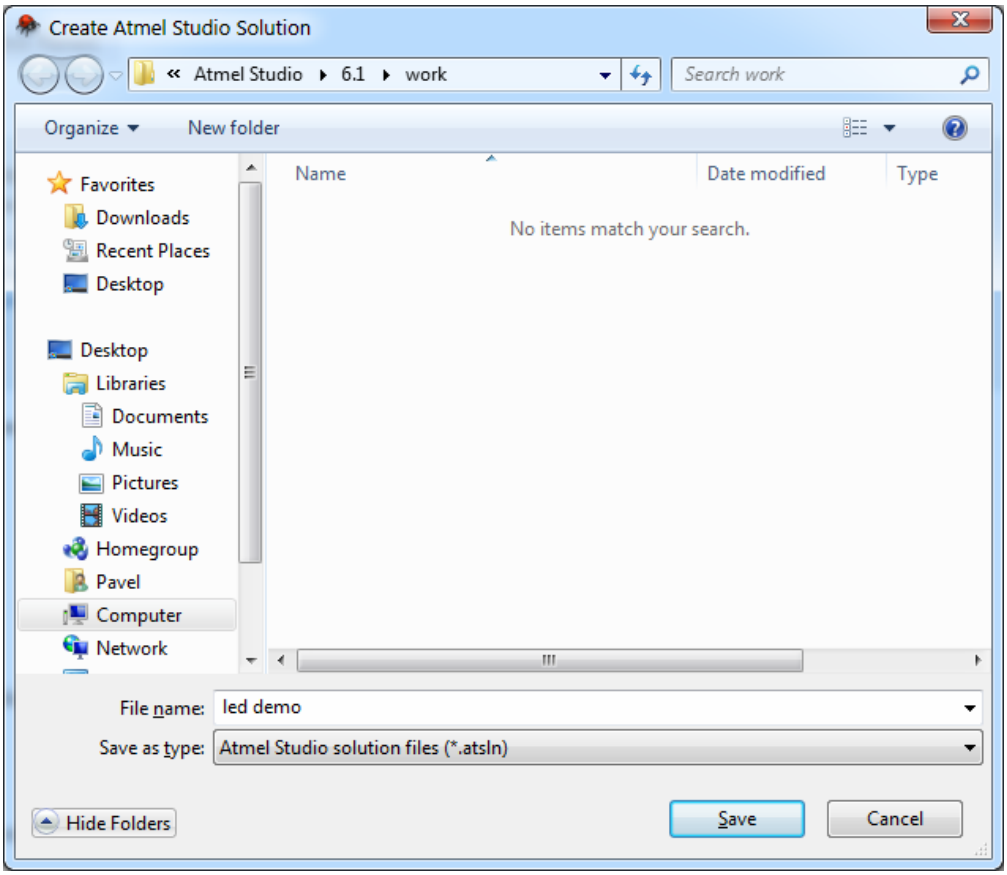



This will allow us to use the same peripheral configuration for other projects, just by reloading the **.cwp** file in the CodeWizardAVR.

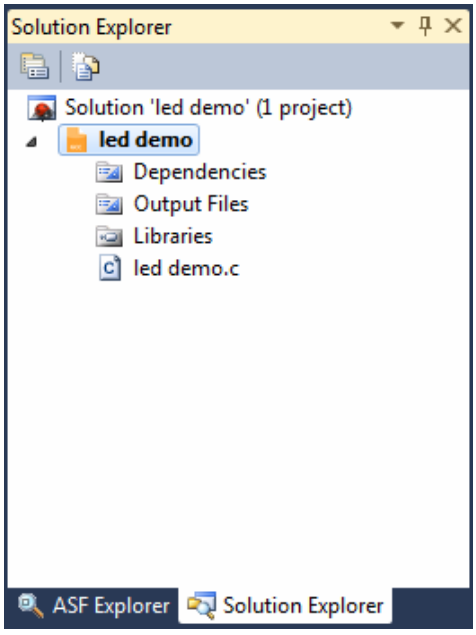
Once all these files were saved, Atmel Studio will prompt us if we wish to create a new solution for the “led demo” project:



We will click on the **Yes** button and will be prompted for the solution name:

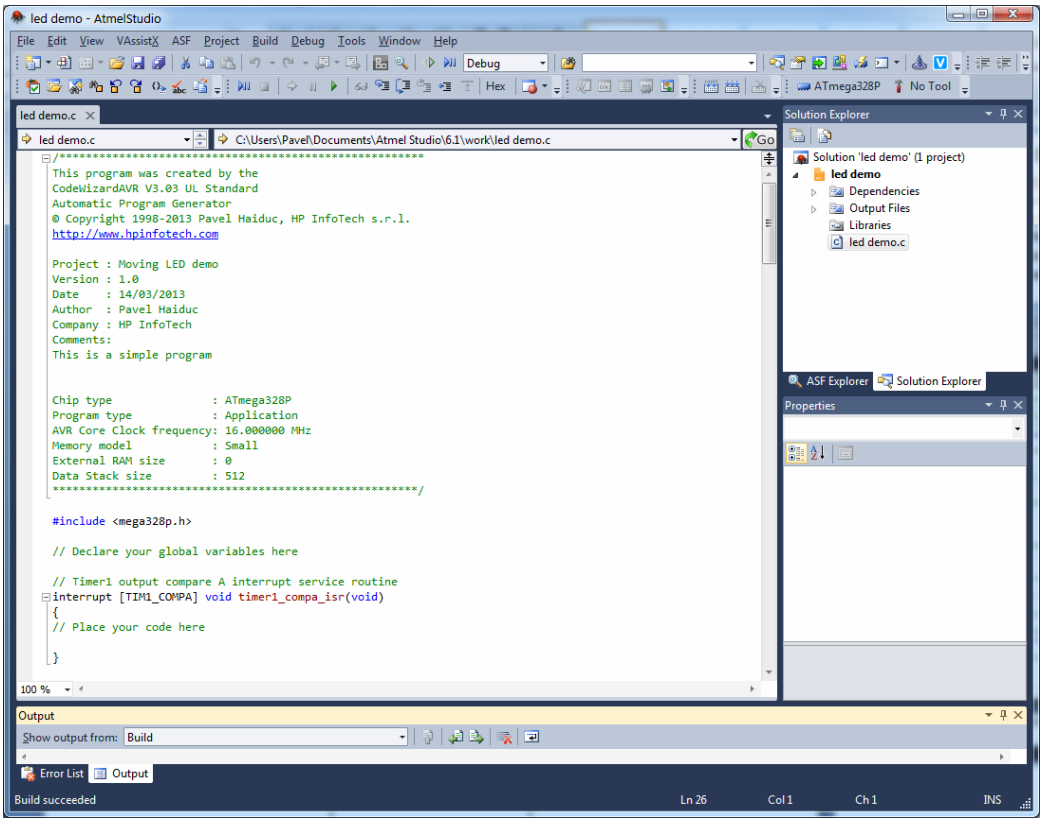



Once this is done, the new solution and project are loaded in Atmel Studio and the corresponding information is displayed in the Solution Explorer:



# 4. Editing the Source Code

The CodeWizardAVR has created all the code needed to initialize the peripherals for our application, now we must add the part of code required to execute our task: light sequentially with a 200 ms delay, each of the 8 LEDs connected to PORTD.
In order to open the project's C source file in the editor, we must double click on the **led demo.c** node in the Solution Explorer.
Once the file is loaded in the Editor window we can apply changes to it:




All the action of the program is performed by Timer 1 compare with OCR1A match interrupt service routine **timer1_compa_isr** which is called every 200 ms.
The required code will be added there, in bold text:

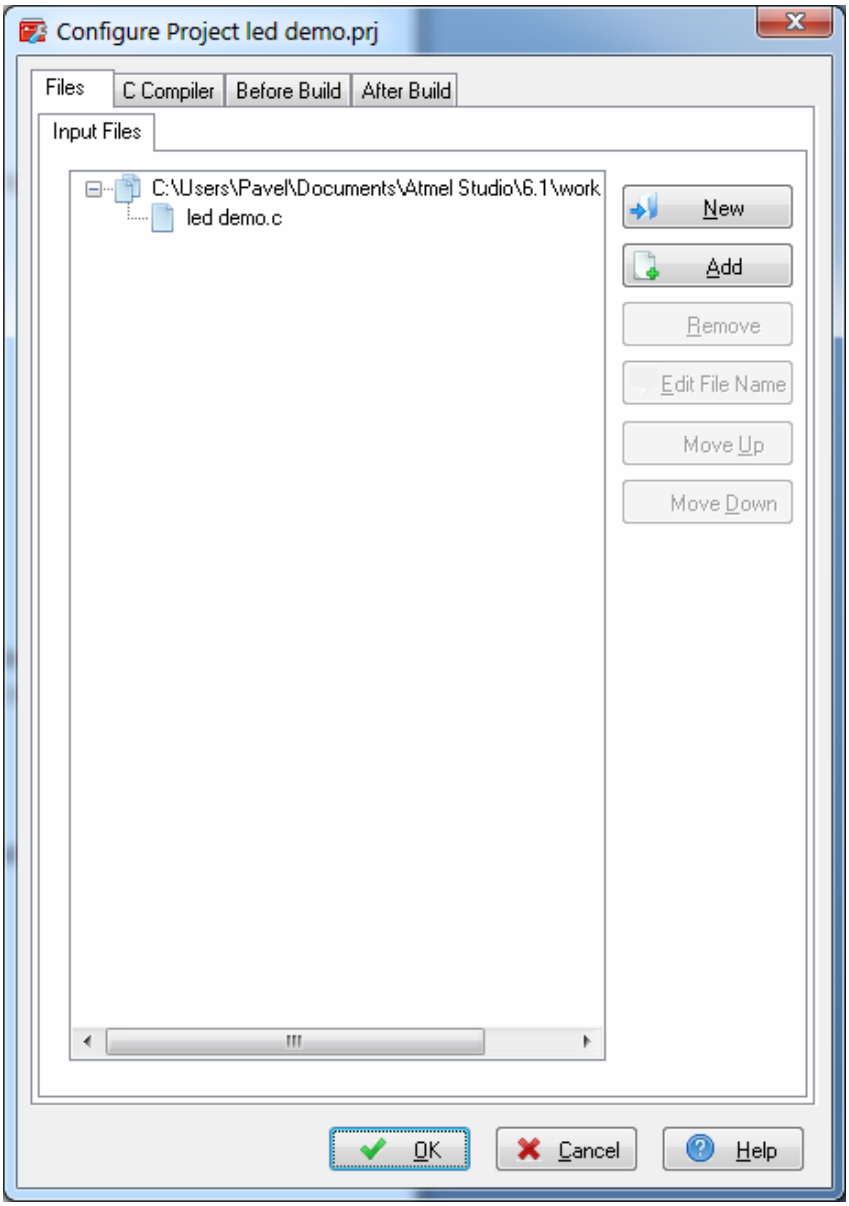
```
// Timer1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
// Place your code here

// If all LEDs are off, light the first one
if (PORTD == 0xFF) PORTD = 0xFE;
// One of the LEDs is already lighted, turn it off and light the next one
else PORTD = (PORTD << 1) | 1;
}
```

**Note:** As the LEDs' anodes are connected to +5V by a resistor and the cathodes are connected to PORTD outputs, in order for them to be lighted the corresponding output must be set to logic level 0.

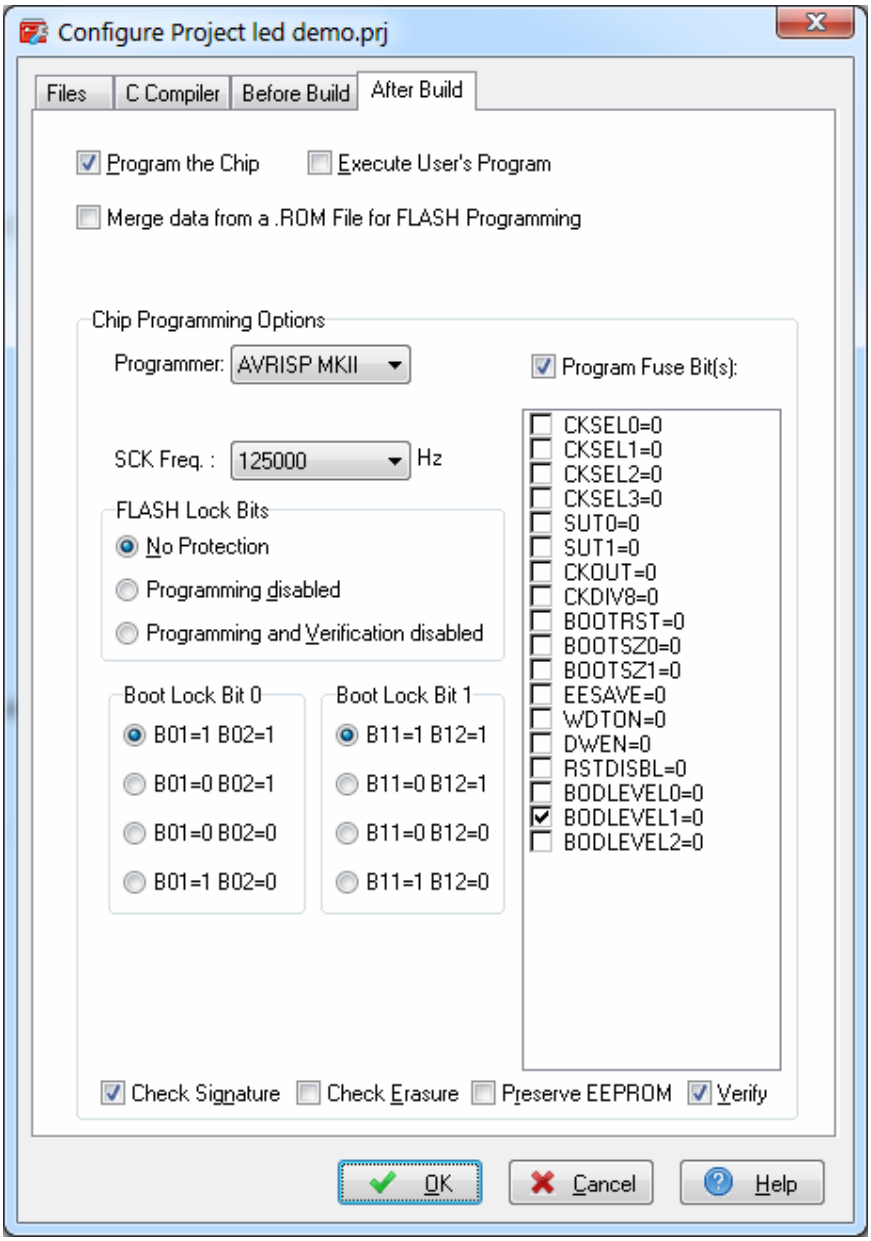
# 5. Configuring the Project

Once the required changes were made to the program's source code, the next step is to configure the project's build options using the **Project|Configure** menu.
The dialog window will be displayed:




We need to automatically program the chip after a successful **Build**.

This can be achieved by selecting the **After Build** tab and enabling the **Program the Chip** option:




The following options must be set:

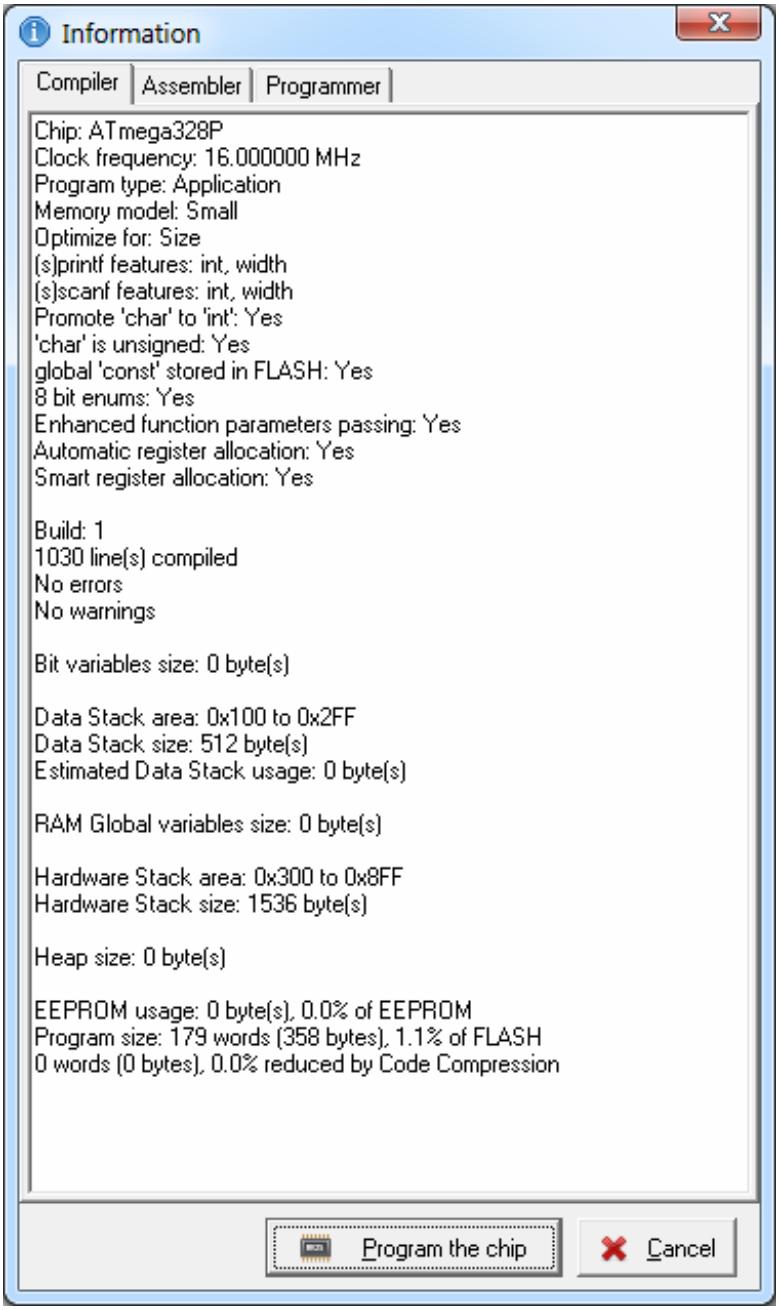
- **Programmer**: AVRISP MKII
- **SCK Freq**: 125000 Hz
- **Program Fuse Bit(s)**: enabled
- All fuse bits **=0** not programmed state (not checked), except **BODLEVEL1=0** which must be in programmed state (checked).

The project configuration changes must be confirmed by clicking on the **OK** button.

# 6. Building the Project and Programming the Chip

The final step is compiling and linking our program using the **Build|Build** *led demo* menu command.
After a successful build the following **Information** window will be displayed:



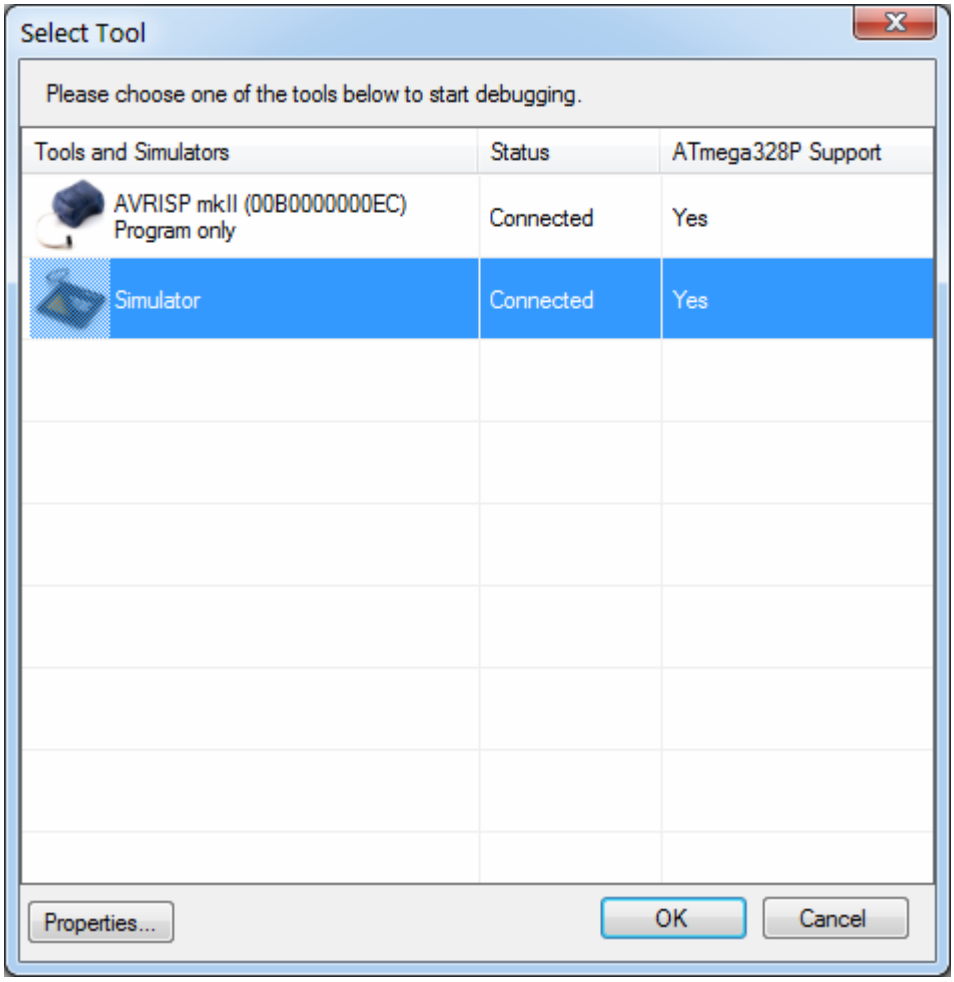

Clicking on the **Program the chip** button will transfer the compiled program to the chip's FLASH and program the fuse bits.
Once these operations are performed, the program will start executing.

# 7. Debugging the Program

Once the program was successfully built, it can be debugged in source level form using the Atmel Studio's **Simulator**.

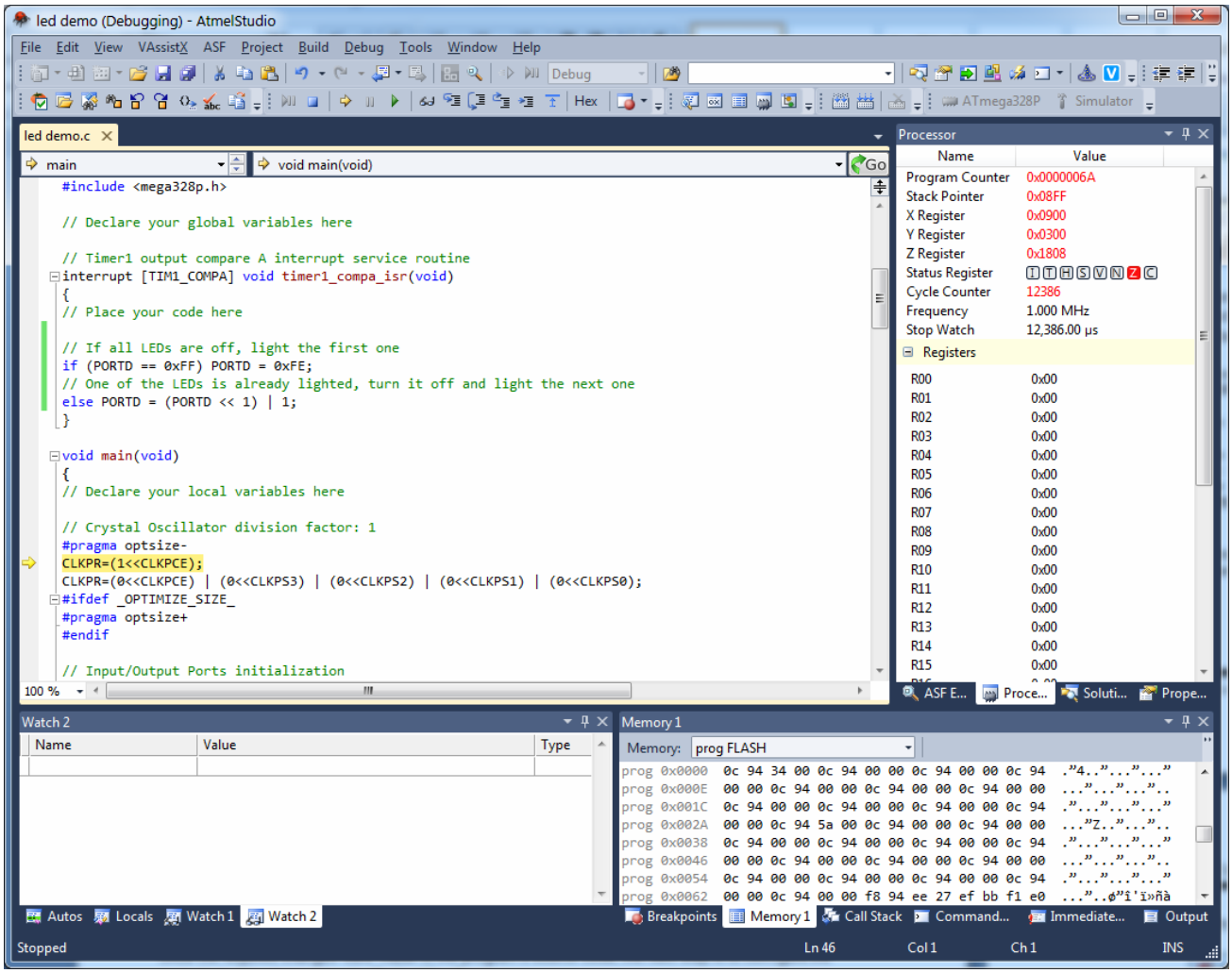
A debugging session can be started using the **Debug|Start Debugging and Break** menu command, the  toolbar button or by pressing the **Alt+F5** keys.
If the source files were modified since the last **Build**, a **Rebuild** will be automatically performed, before the debugging will be started.
Atmel Studio uses the **.cof** object file produced by CodeVisionAVR for debugging.

When the debugging session will be started for the first time, the user will be prompted to select the tool that will be using for tracing the program execution:

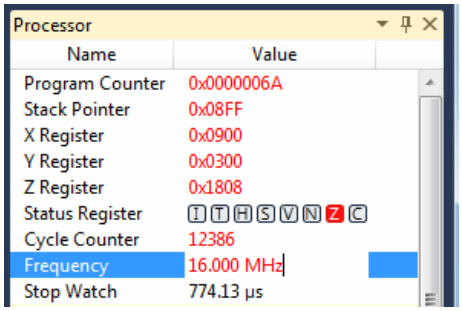



For our example, we will select the **Simulator** and click on the **OK** button.

After the debugging tool was selected, the debugging session actually starts:

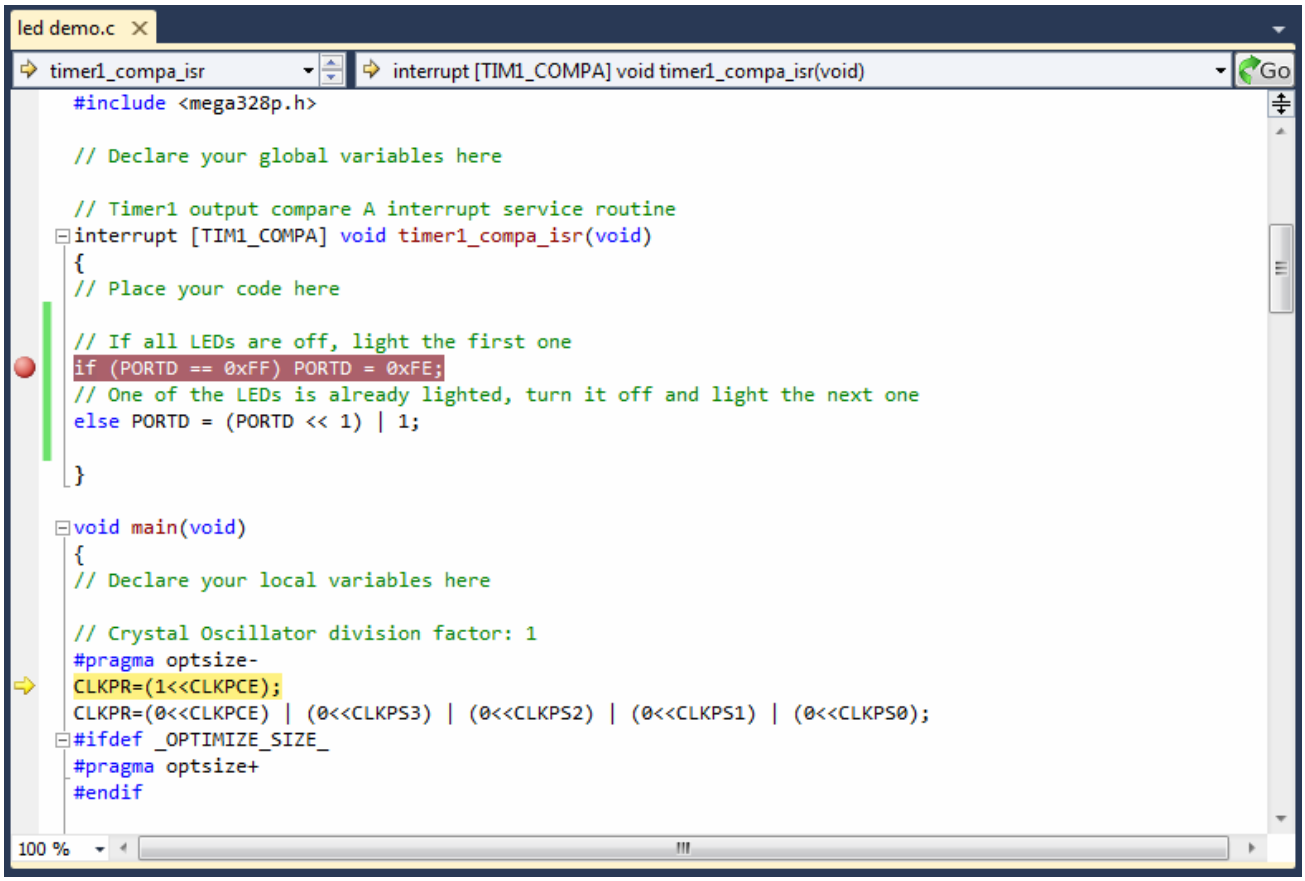



The simulator first executes the low level startup initialization code, which consists of filling all RAM locations with 0s, initializing the global variables, data and hardware stack pointers.
Once these operations are finished, the program execution is passed to the **main** function of the program.
The debugger stops the program execution at the first C source line of **main**, allowing the user to single-step the program from there.
The AVR chip registers are displayed in the **Processor** window.
By default the simulator uses a clock frequency of 1.000MHz, but in our example the chip is clocked at 16.000MHz, so this must be modified by clicking on the **Frequency** field and entering the correct value there:



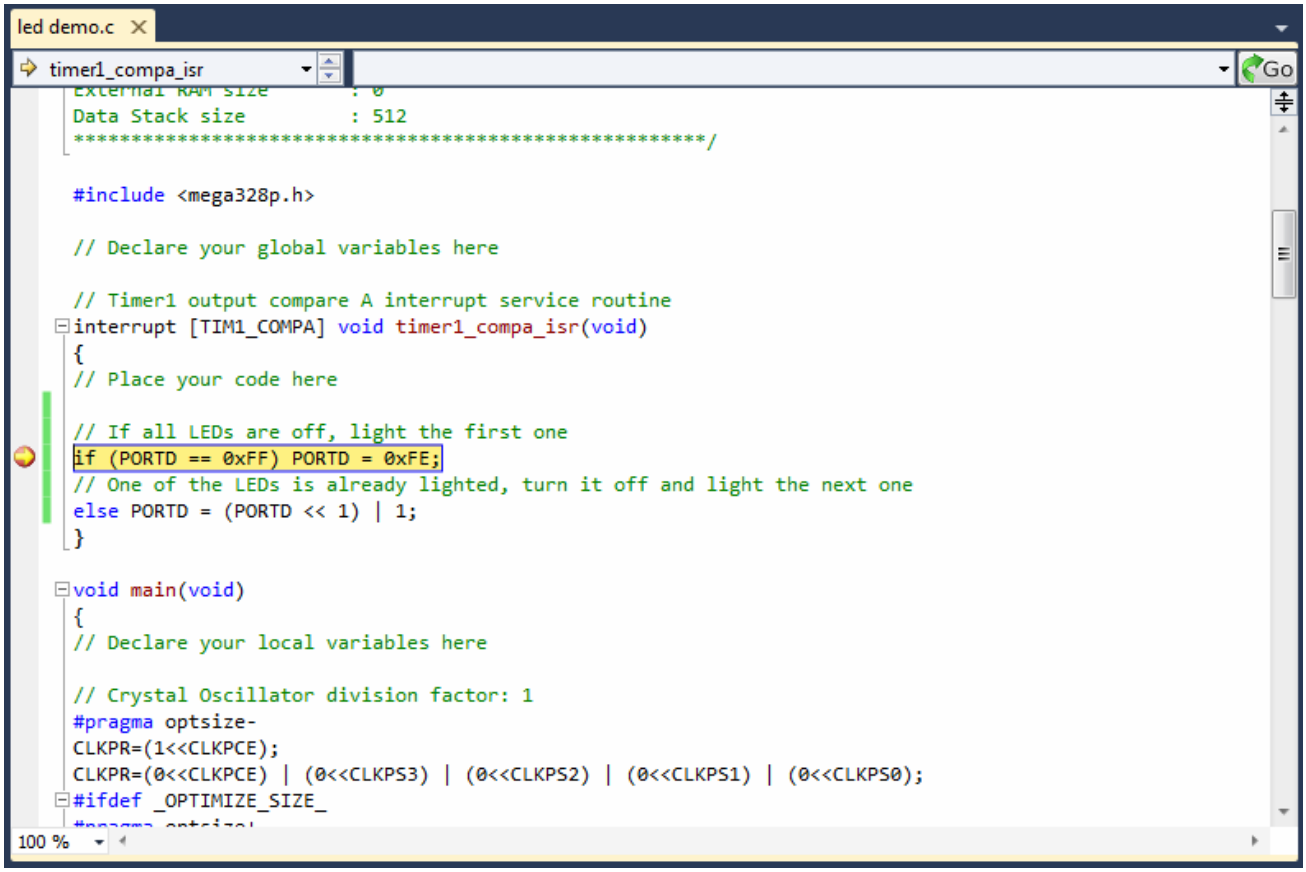

and pressing **Enter**.

The next step in debugging our program is to set a breakpoint at the beginning of the Timer 1 output compare A interrupt service routine **timer1_compa_isr**.
This is achieved by placing the cursor on the first line of code in the above mentioned function and selecting the **Debug|Toggle Breakpoint** menu command or pressing the **F9** key:

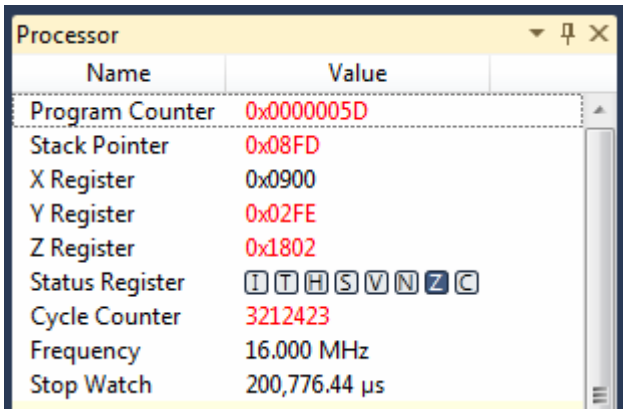



The line with the breakpoint will be highlighted.
After the breakpoint was set, we can continue single stepping through our program using **Debug|Step Into** menu command, the **F11** key or the  toolbar button.
We can start executing our program, until the breakpoint is reached, by selecting the **Debug|Continue** menu command, pressing the **F5** key or the  toolbar button.

**Important Note:** The Atmel Studio **Simulator** executes the program at a much lower speed than the real AVR chip, so reaching the breakpoint set in the **timer1_compa_isr** function doesn't occur in real time after 200 ms.
Depending on the host PC speed the simulator will appear to hang for 1...2 minutes, but finally the program execution stops at the breakpoint:



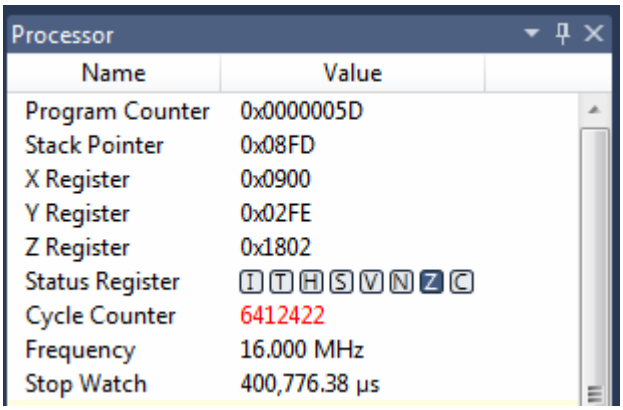

As the program execution stopped inside the **timer1_compa_isr** function, we will note the **Stop Watch** value of **200776.44 µs** in the **Processor** window:




This value is slightly larger than the 200 ms which we have specified as the Timer 1 output compare interrupt period.
The additional 776.44 µs are required by the startup initialization code, after chip reset and the moment the Timer 1 registers are first initialized and counting is started.

We will press the **F5** key again to continue program execution.
The simulator will stop again at the breakpoint in **timer1_compa_isr** after 1...2 minutes, depending on the host PC speed.
This time the **Stop Watch** value in the **Processor** window will be **400776.38 µs**:




So the time interval between two Timer 1 output compare A interrupts will be:

**400776.38 µs - 200776.44 µs = 199999.94 µs = 199.99994 ms**

which is equal to the required 200 ms with a negligible error.

The following additional commands can be used when debugging:

- **Debug|Step Over**, F10 key or toolbar button to execute one instruction.
  If the instruction contains a function call, the function is executed as well.
- **Debug|Step Out**, Shift+F11 keys or toolbar button to continue execution until the current function has completed
- **Debug|Run To Cursor**, Ctrl+F10 keys or toolbar button to continue execution until the current cursor position in the source file or disassembly view is reached
- **Debug|Reset**, Shift+F5 keys or toolbar button to restart program execution from the beginning
- **Debug|Restart** or toolbar button to restart the debugger and reload the debugged program
- **Debug|Toggle Breakpoint** or F9 key to set a breakpoint at the current cursor position in the C source file or disassembly view
- **Debug|New Breakpoint|Break at Function** to set a breakpoint at the beginning of a particular function
- **Debug|Delete All Breakpoints** or Ctrl+Shift+F9 keys to delete all the breakpoints that were set
- **Debug|Disable All Breakpoints** to temporarily disable all the breakpoints that were set
- **Debug|Enable All Breakpoints** to re-enable all the breakpoints that were set
- **Debug|Break All**, Ctrl+F5 keys or toolbar button to stop program execution
- **Debug|Windows** allow displaying specific windows for watching variables, processor registers, I/O and peripheral registers, memory contents, code disassembly, etc.
- **Debug|Stop Debugging,** Ctrl+Shift+F5 keys or toolbar button stops the debugging session.

To obtain more information about using the debugger, please consult the Atmel Studio Help.

**Note:** The compiler applies some optimization techniques that may prevent correct debugging of the executable program.
Therefore it is recommended to select the **Project|Configure|C Compiler|Code generation|Optimize for: Speed** option for code debugging.
If the program fits in the chip's FLASH, this option must be left enabled for **Release** too, as the program will execute faster this way.

# 8. Conclusion

This document provides the necessary starting guidelines for rapid development and debugging of an application using the CodeVisionAVR extension for Atmel Studio.
CodeVisionAVR is supplied with a comprehensive Help and User Manual, which must be extensively studied in order to take full advantage of all the features of the C compiler, associated peripheral libraries, CodeWizardAVR and LCD Vision font/image editor/converter for graphic displays.
The compiler is supplied with a large number of example programs covering:

- Alphanumeric LCD
- Graphic LCD, TFT and OLED displays
- SD Memory Cards
- USART
- TWI, $I^2C$
- SPI
- ADC
- Boot loaders
- USB
- Web Server
- XMEGA EBI
- XMEGA Quadrature Encoder
- XMEGA DAC
- DS1820, DS18B20, LM75, DS1621 temperature sensors

These programs are located in the **\Examples** and **\Examples ATxmega** subdirectories of the CodeVisionAVR installation directory.

# Appendix A – The Source Code

```
/*****************************************************
This program was created by the
CodeWizardAVR V3.03 Standard
Automatic Program Generator
® Copyright 1998-2013 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : Moving LED demo
Version : 1.0
Date    : 14/03/2013
Author  : Pavel Haiduc
Company : HP InfoTech
Comments:
This is a simple program


Chip type               : ATmega328P
Program type            : Application
AVR Core Clock frequency: 16.000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 512
*****************************************************/

#include <mega328p.h>

// Declare your global variables here

// Timer1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
// Place your code here

// If all LEDs are off, light the first one
if (PORTD == 0xFF) PORTD = 0xFE;
// One of the LEDs is already lighted, turn it off and light the next one
else PORTD = (PORTD << 1) | 1;
}

void main(void)
{
// Declare your local variables here

// Crystal Oscillator division factor: 1
#pragma optsize-
CLKPR=(1<<CLKPCE);
CLKPR=(0<<CLKPCE) | (0<<CLKPS3) | (0<<CLKPS2) | (0<<CLKPS1) | (0<<CLKPS0);
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif
```

```
// Input/Output Ports initialization
// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) | (0<<DDB1) |
     (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) |
      (0<<PORTB1) | (0<<PORTB0);

// Port C initialization
// Function: Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRC=(0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) | (0<<DDC1) | (0<<DDC0);
// State: Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) |
      (0<<PORTC0);

// Port D initialization
// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
DDRD=(1<<DDD7) | (1<<DDD6) | (1<<DDD5) | (1<<DDD4) | (1<<DDD3) | (1<<DDD2) | (1<<DDD1) |
     (1<<DDD0);
// State: Bit7=1 Bit6=1 Bit5=1 Bit4=1 Bit3=1 Bit2=1 Bit1=1 Bit0=1
PORTD=(1<<PORTD7) | (1<<PORTD6) | (1<<PORTD5) | (1<<PORTD4) | (1<<PORTD3) | (1<<PORTD2) |
      (1<<PORTD1) | (1<<PORTD0);

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=0xFF
// OC0A output: Disconnected
// OC0B output: Disconnected
TCCR0A=(0<<COM0A1) | (0<<COM0A0) | (0<<COM0B1) | (0<<COM0B0) | (0<<WGM01) | (0<<WGM00);
TCCR0B=(0<<WGM02) | (0<<CS02) | (0<<CS01) | (0<<CS00);
TCNT0=0x00;
OCR0A=0x00;
OCR0B=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 250.000 kHz
// Mode: CTC top=OCR1A
// OC1A output: Disconnected
// OC1B output: Disconnected
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer Period: 0.2 s
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: Off
TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11) | (0<<WGM10);
TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (1<<WGM12) | (0<<CS12) | (1<<CS11) |
       (1<<CS10);
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0xC3;
OCR1AL=0x4F;
```

```
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2A output: Disconnected
// OC2B output: Disconnected
ASSR=(0<<EXCLK) | (0<<AS2);
TCCR2A=(0<<COM2A1) | (0<<COM2A0) | (0<<COM2B1) | (0<<COM2B0) | (0<<WGM21) | (0<<WGM20);
TCCR2B=(0<<WGM22) | (0<<CS22) | (0<<CS21) | (0<<CS20);
TCNT2=0x00;
OCR2A=0x01;
OCR2B=0x00;

// Timer/Counter 0 Interrupt(s) initialization
TIMSK0=(0<<OCIE0B) | (0<<OCIE0A) | (0<<TOIE0);

// Timer/Counter 1 Interrupt(s) initialization
TIMSK1=(0<<ICIE1) | (0<<OCIE1B) | (1<<OCIE1A) | (0<<TOIE1);

// Timer/Counter 2 Interrupt(s) initialization
TIMSK2=(0<<OCIE2B) | (0<<OCIE2A) | (0<<TOIE2);

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// Interrupt on any change on pins PCINT0-7: Off
// Interrupt on any change on pins PCINT8-14: Off
// Interrupt on any change on pins PCINT16-23: Off
EICRA=(0<<ISC11) | (0<<ISC10) | (0<<ISC01) | (0<<ISC00);
EIMSK=(0<<INT1) | (0<<INT0);
PCICR=(0<<PCIE2) | (0<<PCIE1) | (0<<PCIE0);

// USART initialization
// USART disabled
UCSR0B=(0<<RXCIE0) | (0<<TXCIE0) | (0<<UDRIE0) | (0<<RXEN0) | (0<<TXEN0) | (0<<UCSZ02) |
       (0<<RXB80) | (0<<TXB80);

// Analog Comparator initialization
// Analog Comparator: Off
// The Analog Comparator's positive input is
// connected to the AIN0 pin
// The Analog Comparator's negative input is
// connected to the AIN1 pin
ACSR=(1<<ACD) | (0<<ACBG) | (0<<ACO) | (0<<ACI) | (0<<ACIE) | (0<<ACIC) | (0<<ACIS1) |
     (0<<ACIS0);
ADCSRB=(0<<ACME);
// Digital input buffer on AIN0: On
// Digital input buffer on AIN1: On
DIDR1=(0<<AIN0D) | (0<<AIN1D);

// ADC initialization
// ADC disabled
ADCSRA=(0<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (0<<ADPS2) |
       (0<<ADPS1) | (0<<ADPS0);
```

```
// SPI initialization
// SPI disabled
SPCR=(0<<SPIE) | (0<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) | (0<<CPHA) | (0<<SPR1) |
     (0<<SPR0);

// TWI initialization
// TWI disabled
TWCR=(0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWEN) | (0<<TWIE);

// Global enable interrupts
#asm("sei")

while (1)
      {
      // Place your code here

      }
}
```