Project Group ‘Racing Car IT’

AVR Timer Configuration
User guide

Peer Adelt

University of Paderborn

Summer Semester 2012

Advisers:
Prof. Dr. Marco Platzner, Tobias Beisel, Sebastian Meisner, Lars Schäfers

Contents

1 Introduction

This document describes the installation and usage of the AVR Timer Configuration Editor. The tool is delivered as an Eclipse plugin, allowing graphical configuration of the timer hardware of the Atmel AT90CAN32/64/128 microprocessor. An ANSI C code generator is included, enabling the AVR Timer Configuration Editor to output initialization code for the hardware as well as runtime API functions. The generated code is intended to be compiled with AVR-gcc.

After installation instructions are given for Windows, Linux and Mac OSX, this document provides detailed information about how to define an AVR Timer configuration and how to generate C code from it.

⚠ This document assumes, that the reader is already familiar with the hardware details of the AT90CAN microprocessor family. Furthermore, the reader must know the C programming language.

2 Installation

This section describes the necessary steps to install the AVR Timer Configuration plugin into an existing Eclipse environment.

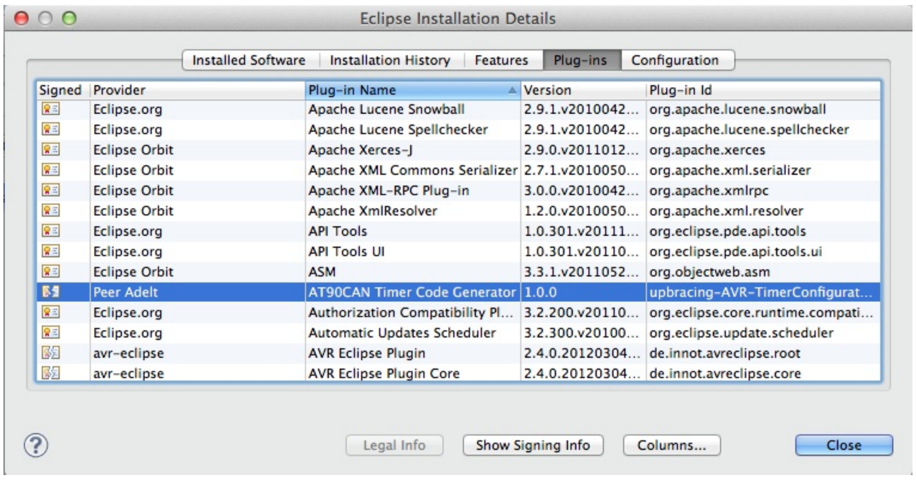
2.1 Prerequisites

This plugin was tested with *Eclipse Indigo Service Release 2*. Please ensure, that you work with this or a newer version of Eclipse. Additionally, it is recommended to install *The AVR Eclipse Plugin*[1]. This enables the Eclipse environment to cross-compile C code for AVR targets.

2.2 Deploying the plugin into the Eclipse installation

The graphical editor is installed from the *EclipsePlugins.zip* archive. This archive contains the UPBracingStatemachineEditor and AVR Timer-Configuration plugins. You need to install *AVR Timer-Configuration* to work with the editor. You can find a more detailed explanation in the appendix of the project group's main documentation.

To verify the installation, please navigate to *Help ->About Eclipse* and click on *Installation Details*. If the installation was successful, the plugin appears in the *Plug-Ins* pane within the *Eclipse Installation Details* dialog. The AVR Timer Configuration Editor plugin is highlighted in Figure 1.



Signed	Provider	Plug-in Name	Version	Plug-in Id
✓	Eclipse.org	Apache Lucene Snowball	2.9.1.v2010042...	org.apache.lucene.snowball
✓	Eclipse.org	Apache Lucene Spellchecker	2.9.1.v2010042...	org.apache.lucene.spellchecker
✓	Eclipse Orbit	Apache Xerces-J	2.9.0.v2011012...	org.apache.xerces
✓	Eclipse Orbit	Apache XML Commons Serializer	2.7.1.v2010050...	org.apache.xml.serializer
✓	Eclipse Orbit	Apache XML-RPC Plug-in	3.0.0.v2010042...	org.apache.xmlrpc
✓	Eclipse Orbit	Apache XmlResolver	1.2.0.v2010050...	org.apache.xml.resolver
✓	Eclipse.org	API Tools	1.0.301.v20111...	org.eclipse.pde.api.tools
✓	Eclipse.org	API Tools UI	1.0.301.v20110...	org.eclipse.pde.api.tools.ui
✓	Eclipse Orbit	ASM	3.3.1.v2011052...	org.objectweb.asm
▨	Peer Adelt	AT90CAN Timer Code Generator	1.0.0	upbracing-AVR-TimerConfigurat...
✓	Eclipse.org	Authorization Compatibility Pl...	3.2.200.v20110...	org.eclipse.core.runtime.compati...
✓	Eclipse.org	Automatic Updates Scheduler	3.2.300.v20100...	org.eclipse.update.scheduler
▨	avr-eclipse	AVR Eclipse Plugin	2.4.0.20120304...	de.innot.avreclipse.root
▨	avr-eclipse	AVR Eclipse Plugin Core	2.4.0.20120304...	de.innot.avreclipse.core

Figure 1: List of installed plugins

[1]`http://avr-eclipse.sourceforge.net/wiki/index.php/The_AVR_Eclipse_Plugin`

3 Timer Configuration

This section explains how to configure the AT90CAN timer hardware graphically with the AVR Timer Configuration Editor.

3.1 Creating configuration files

The AVR Timer Configuration Editor works on configuration files *(*.tcxml)*. To create a new configuration file, please choose *File ->New ->Other...* from either the Eclipse menu or in the context menu of the folder in which you want the file to be created.

Select *AVR Timer Configuration ->New Timer Configuration* and click *Next >* (see Figure 2). Enter the destination folder and filename and click *Finish*. Eclipse automatically starts a new instance of the AVR Timer Configuration Editor after the file has been created.

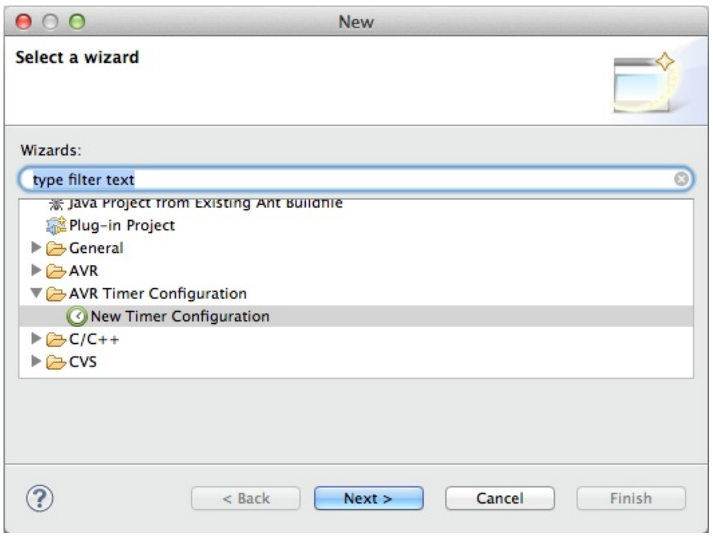
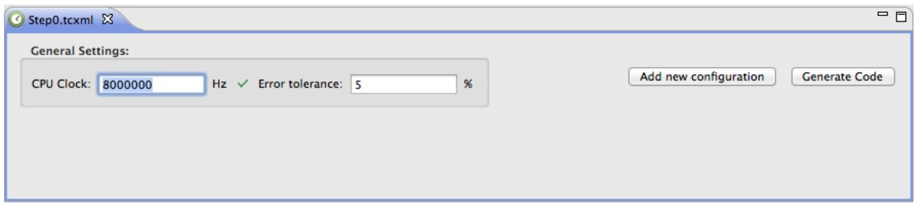


Figure 2: New file dialog

Figure 3 shows the editor view for the file *timer_config.tcxml*. By default, the processor frequency is set to 8MHz and the error tolerance to 5%. The processor frequency is required for register value calculation. The minimum and maximum frequencies are 1Hz and 16MHz. Other values will cause a validation error.

The error tolerance value decides, whether deviations of quantized periods from their desired periods entered in the editor will lead to warnings during validation or not (see subsection 3.3 for details).



General Settings:

CPU Clock: 8000000 Hz ✓ Error tolerance: 5 %

Figure 3: Initial configuration file

User input is automatically validated. Icons next to the input fields show their current *Validation Status*. There are three statuses defined:

Ok ✓

The user input is valid.

Warning ⚠

The tooltip text of the icon will show information about the warning.

Error ⊗

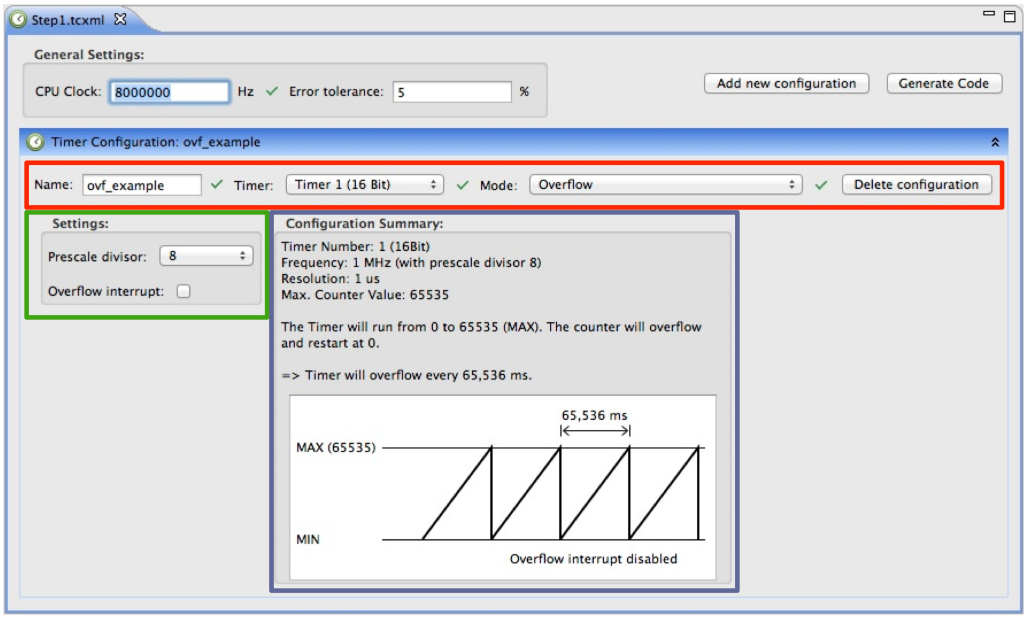
The tooltip text of the icon will show information about the error. Code generation is not possible, if the configuration contains errors.

3.2 Creating timer configurations

The button *Add new configuration* in the top area of the editor window creates a new use case oriented timer configuration and displays it in the editor immediately. The configurations are shown in a vertical expand bar. To expand the newly created configuration, click on its title bar.

Figure 4 shows an expanded timer configuration. The red area is equal for each timer configuration, regardless of its settings. It contains an input field for a name, a dropdown menu for the timer to use and a dropdown menu to select the mode of operation. The button *Delete configuration* deletes this configuration after the user has been asked for confirmation.

The name of each configuration must be a valid C identifier, since it is used for naming the C functions being created later in the process. Furthermore, no two configurations in one *tcxml* file shall share the same name.


Step1.tcxml

General Settings:

CPU Clock: 8000000 Hz ✓ Error tolerance: 5 %

Add new configuration | Generate Code

Timer Configuration: ovf_example

Name: ovf_example ✓ Timer: Timer 1 (16 Bit) ✓ Mode: Overflow ✓ Delete configuration

Settings:

Prescale divisor: 8

Overflow interrupt: ☐

Configuration Summary:

Timer Number: 1 (16Bit)
Frequency: 1 MHz (with prescale divisor 8)
Resolution: 1 us
Max. Counter Value: 65535

The Timer will run from 0 to 65535 (MAX). The counter will overflow and restart at 0.

=> Timer will overflow every 65,536 ms.

Figure 4: Overview of the editor elements

The green area in Figure 4 is dynamically rendered according to the timer being used and its mode of operation. It contains all user input fields necessary to fully configure the selected operation mode.

For each configuration mode, a descriptive text and image is dynamically created (dark blue area). It explains the behavior of the timer in the current operation mode. The next section describes how to configure the timer for different operation modes.

3.3 Modes of operation

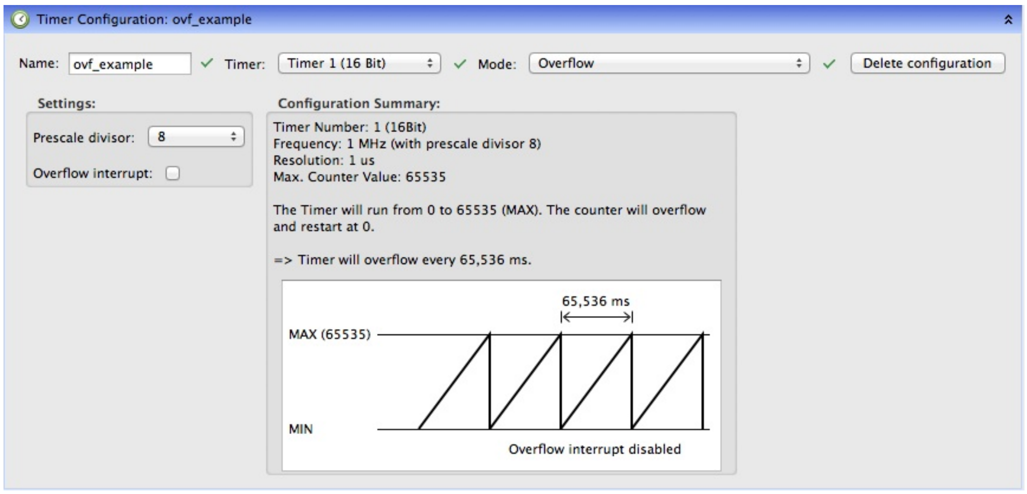
The AVR Timer configuration Editor supports the configuration of *Overflow* mode, *Clear on Compare Match* mode, *Fast PWM* mode, *Phase correct PWM* mode and *Phase and frequency correct PWM* mode.

In each mode, prescaling can be applied. A dropdown menu allows to select the prescale divisors 1, 8, 64, 256 and 1024. For all of the following examples, a processor frequency of 8 MHz is assumed. The error tolerance is left at its default value 5%.

3.3.1 Overflow

In Overflow mode, the timer repeatedly runs from 0 to MAX (8 Bit: 255, 16 Bit: 65535) and restarts. The overrun frequency can only be influenced by prescaling. Overflow interrupts can be enabled in the editor.

Figure 5 shows an example overflow mode configuration for timer 1 (16 Bit). Prescaling is set to 8. With this prescale divisor, timer ticks will occur every µs. This results in a timer overflow every 65536 µs (MAX+1 timer ticks). In this example, overflow interrupts are disabled. If the developer requires these notifications, he needs to enable the *Overflow interrupt* checkbox.


Timer Configuration: ovf_example

Name: ovf_example ✓ Timer: Timer 1 (16 Bit) ✓ Mode: Overflow ✓ Delete configuration

Settings:

Prescale divisor: 8

Overflow interrupt: ☐

Configuration Summary:

Timer Number: 1 (16Bit)
Frequency: 1 MHz (with prescale divisor 8)
Resolution: 1 us
Max. Counter Value: 65535

The Timer will run from 0 to 65535 (MAX). The counter will overflow and restart at 0.

=> Timer will overflow every 65,536 ms.

Figure 5: Detail view (*Overflow* mode)

The *Configuration Summary* on the right displays detailed information about the overflow configuration. The image at the bottom shows the behavior of the timer in this mode. The reset period is shown on top of the waveform. If overflow interrupts are enabled, this is displayed below the waveform. Additionally, blue bullets are shown on top of the waveform to indicate their times of occurrence.

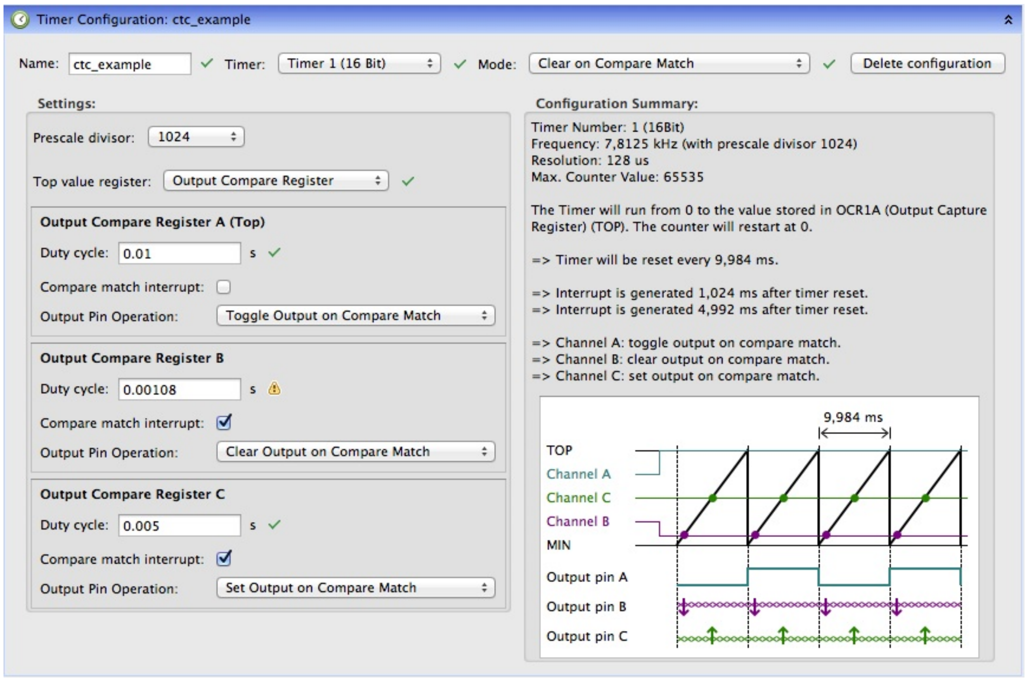
3.3.2 Clear on Compare Match

In Clear on Compare Match (CTC) mode, the timer repeatedly runs from 0 to the user defined TOP value. The reset frequency can thus be influenced by the TOP value as well as by the prescale divisor.

Figure 6 shows an example CTC mode configuration for timer 1 (16 Bit). Pescaling is set to 1024. With this prescale divisor, timer ticks will occur every 128 µs. *Output Compare Register A* was chosen as the TOP value register. The desired TOP period was set to 10 ms. Although this value cannot be reached exactly (quantized to 9.984 ms) it is validated with *Ok* status, because the deviation from the desired value is less than 5%. The output pin associated with this channel is toggled on every compare match.

Compare channel B is expected to match 1.08 ms after timer reset. Because quantization leads to a value with an error percentage of more than 5, a *Warning* status is displayed. The hover text shows, that this value is quantized to 1.024ms. The output pin associated with this channel is cleared on every compare match.

Channel C is expected to match 5 ms after timer reset. Like the period of channel A, the desired value lies within the tolerated range of 5% around the optimal value. The output pin associated with this channel is set on every compare match.



Timer Configuration: ctc_example

Name: ctc_example ✓ Timer: Timer 1 (16 Bit) ✓ Mode: Clear on Compare Match ✓ Delete configuration

Settings:

Prescale divisor: 1024

Top value register: Output Compare Register ✓

Output Compare Register A (Top)

Duty cycle: 0.01 s ✓

Compare match interrupt: ☐

Output Pin Operation: Toggle Output on Compare Match

Output Compare Register B

Duty cycle: 0.00108 s ⚠

Compare match interrupt: ☑

Output Pin Operation: Clear Output on Compare Match

Output Compare Register C

Duty cycle: 0.005 s ✓

Compare match interrupt: ☑

Output Pin Operation: Set Output on Compare Match

Configuration Summary:

Timer Number: 1 (16Bit)
Frequency: 7,8125 kHz (with prescale divisor 1024)
Resolution: 128 us
Max. Counter Value: 65535

The Timer will run from 0 to the value stored in OCR1A (Output Capture Register) (TOP). The counter will restart at 0.

=> Timer will be reset every 9,984 ms.

=> Interrupt is generated 1,024 ms after timer reset.
=> Interrupt is generated 4,992 ms after timer reset.

=> Channel A: toggle output on compare match.
=> Channel B: clear output on compare match.
=> Channel C: set output on compare match.

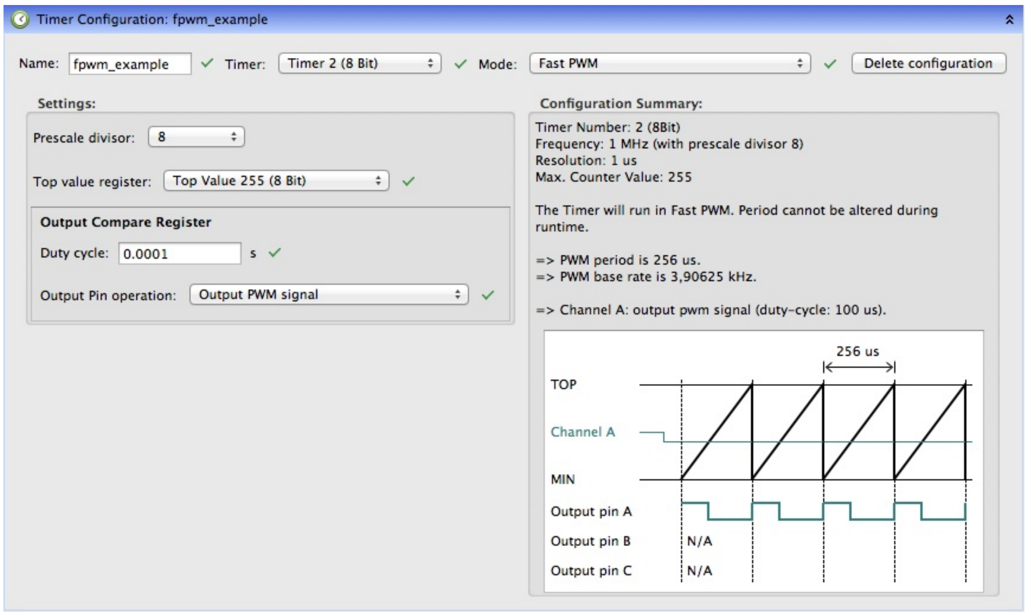
Figure 6: Detail view (*Clear on Compare Match* mode)

The *Configuration Summary* on the right displays detailed information about the overflow configuration. The image at the bottom shows the behavior of the timer in this mode. Channels A, B and C are displayed as horizontal lines on top of the waveform. Channels B and C generate compare match interrupts, indicated by green and purple bullets. The bottom area of the image shows previews for all the output pins.

3.3.3 Fast PWM

In Fast PWM mode, the timer repeatedly runs from 0 to TOP. It depends on the timer, which values can be chosen as TOP values. For 8 Bit timers, only the fixed value 255 can be used. For 16 Bit timers, possible fixed values are 255, 511 and 1023. Additionally, *Input Capture Register* or *Output Compare Register A* can be used for defining a custom TOP value.

Figure 7 shows an example Fast PWM configuration for timer 2 (8 Bit). Prescaling was set to 8, resulting in a PWM base period of 256 µs (3.90625 kHz). Duty cycle for channel A is set to 100 µs.


Timer Configuration: fpwm_example

Name: fpwm_example ✓ Timer: Timer 2 (8 Bit) ✓ Mode: Fast PWM ✓ Delete configuration

Settings:

Prescale divisor: 8

Top value register: Top Value 255 (8 Bit) ✓

Output Compare Register

Duty cycle: 0.0001 s ✓

Output Pin operation: Output PWM signal ✓

Configuration Summary:

```
Timer Number: 2 (8Bit)
Frequency: 1 MHz (with prescale divisor 8)
Resolution: 1 us
Max. Counter Value: 255

The Timer will run in Fast PWM. Period cannot be altered during
runtime.

=> PWM period is 256 us.
=> PWM base rate is 3,90625 kHz.

=> Channel A: output pwm signal (duty-cycle: 100 us).
```

Figure 7: Detail view (*Fast PWM* mode)

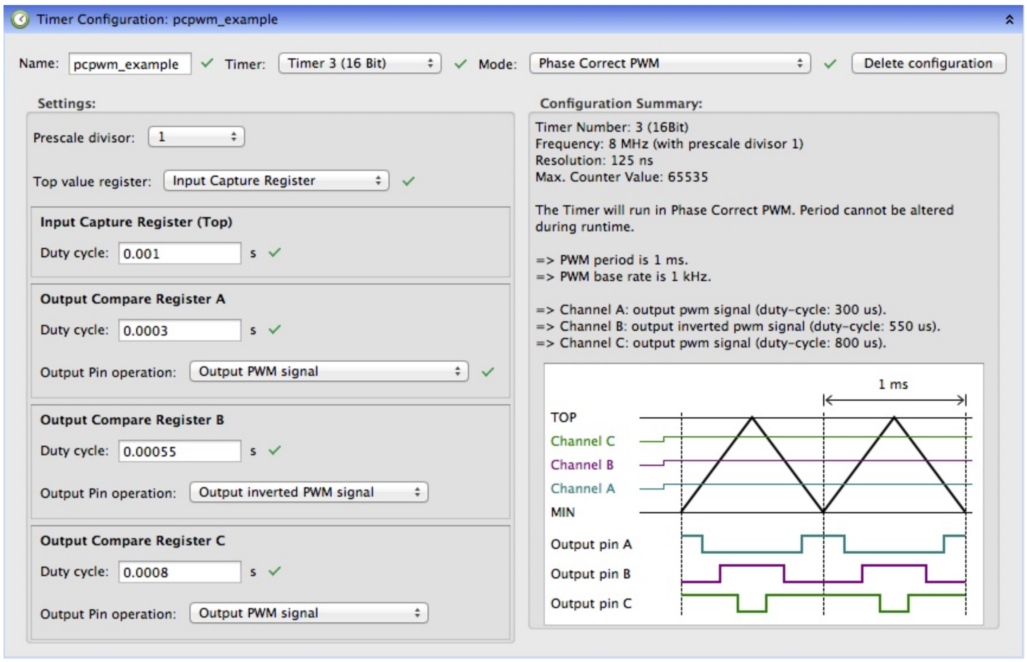
The *Configuration Summary* on the right displays detailed information about the Fast PWM configuration. The image at the bottom shows the behavior of the timer in this mode. Channel A is displayed as a horizontal line on top of the waveform. The bottom area of the image shows previews for all the output pins. Since channels B and C are not available for this timer, "N/A" is displayed in the image.

3.3.4 Phase Correct PWM

In Phase Correct PWM mode, the timer runs in dual-slope operation. It counts up from 0 to TOP and then decrements down to 0. This happens in an endless loop. Like in Fast PWM mode, 255, 511 and 1023 are predefined TOP values. Additionally, *Input Capture Register* or *Output Compare Register A* can be used for defining a custom TOP value.

Figure 8 shows an example Phase Correct PWM configuration for timer 3 (16 Bit). Prescaling was disabled. *Input Capture Register* was chosen for storing the TOP value. This frees all three compare match channels to output different signals simultaneously. In this example, the PWM base period is set to 1 ms.

The duty-cycle for channel A is set to 300 µs, Channel B ist set 550 µs and C to 800 µs. Channels A and C output non-inverted PWM signals, while the signal of Channel B is inverted.


Timer Configuration: pcpwm_example

Name: pcpwm_example ✓ Timer: Timer 3 (16 Bit) ✓ Mode: Phase Correct PWM ✓ Delete configuration

Settings:

Prescale divisor: 1

Top value register: Input Capture Register ✓

Input Capture Register (Top)

Duty cycle: 0.001 s ✓

Output Compare Register A

Duty cycle: 0.0003 s ✓

Output Pin operation: Output PWM signal ✓

Output Compare Register B

Duty cycle: 0.00055 s ✓

Output Pin operation: Output inverted PWM signal

Output Compare Register C

Duty cycle: 0.0008 s ✓

Output Pin operation: Output PWM signal

Configuration Summary:

Timer Number: 3 (16Bit)
Frequency: 8 MHz (with prescale divisor 1)
Resolution: 125 ns
Max. Counter Value: 65535

The Timer will run in Phase Correct PWM. Period cannot be altered during runtime.

=> PWM period is 1 ms.
=> PWM base rate is 1 kHz.

=> Channel A: output pwm signal (duty-cycle: 300 us).
=> Channel B: output inverted pwm signal (duty-cycle: 550 us).
=> Channel C: output pwm signal (duty-cycle: 800 us).

Figure 8: Detail view (*Phase Correct PWM* mode)

The *Configuration Summary* on the right displays detailed information about the Phase Correct PWM configuration. The image at the bottom shows the behavior of the timer in this mode. Channels A, B and C are displayed as horizontal lines on top of the waveform. The bottom area of the image shows previews for all the output pins.

3.3.5 Phase and Frequency Correct PWM

The configuration process of Phase and Frequency Correct PWM mode is the same as for Phase Correct PWM mode. The only difference between this mode and Phase Correct PWM mode is the number of API functions generated. While Phase Correct PWM mode does not support altering the base rate at runtime, this is possible with Phase and Frequency Correct PWM mode. API functions for altering the base rate are provided (see section 5 for details).

This mode is only available for 16 Bit timers. Only *Input Capture Register* and *Output Compare Register A* can be used for storing the top value. No fixed values are usable.

4 Generating code

This section will explain how to generate code from timer configuration files. Each configuration file is translated into two C programming language files. The *Header (.h)* file contains function declarations for all functions generated. The corresponding *C (.c)* file provides the implementation for these functions.

The functions generated are specified in section 5.

4.1 Inside Eclipse

Code generation can be invoked by either the *Generate Code* button in the top right corner of the AVR Timer Configuration Editor window or by a command from the context menu of the Configuration file (*.tcxml) in Eclipse's *Project Explorer* view. See Figure 9 for details.

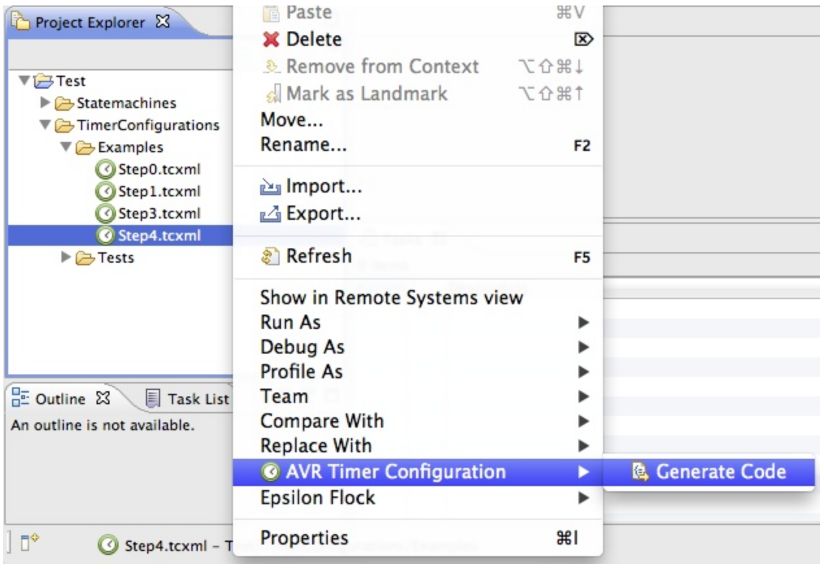


Figure 9: Generating code via context menu

After code has been successfully generated, it is automatically added to the project at the same location as the configuration file. Eclipse automatically refreshes its Project Explorer view. The file names are equal to the name of the configuration file with file endings replaced with either *.c* or *.h*. Figure 10 shows generated code files for configuration file "Step4.tcxml".

If errors occur during code generation, this will be reported. No code files will be added to the project.

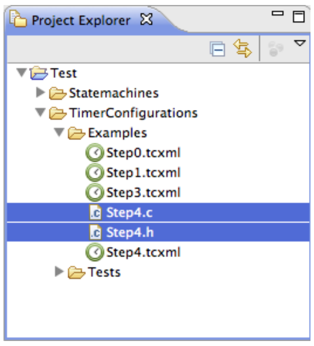


Figure 10: Project explorer with generated code files

There is no functional difference between the context menu command and the *Generate Code* button in the editor window. The generated code will look exactly the same. The context menu provides the option to generate code for models, which are not currently open in an editor instance, though.

4.2 Using the upbracing-CodeGenerator project

The developer might want the code to be generated automatically right before a system build. For this purpose, the *Code Generator* project was developed. Please see the project group's main documentation for guidelines on how to load a **.tcxml* file into the code generator program.

5 API specification

This section describes all API functions being generated for the supported use cases.

5.1 Common functions

The following functions are applicable to all use cases.

Name:	void <**config**>_**init**(void)
Description:	This function initializes the timer module for the use case <config>, which was specified in the configuration. No parameters are passed to this function, since all configuration details are hard-coded. There is no return value, because there are no conditional statements in the initialization function, which could result in different execution paths.
Parameters:	none
Return value:	none
Remarks:	none

Name:	void <**config**>_**start**(void)
Description:	This function starts the timer module for configuration <config>. The timer counts upwards from its current value. Note, that the timer has to be initialized before. If the timer was already started, no action shall take place.
Parameters:	none
Return value:	none
Remarks:	none

Name:	void <**config**>_**stop**(void)
Description:	This function stops the timer module with assigned configuration <config>. The timer value is not reset. If the timer was already stopped, no action shall take place.
Parameters:	none
Return value:	none
Remarks:	none

Name:	[INT] <**config**>**_getCounterValue**(void)
Description:	This function returns the current counter value of the timer with assigned configuration <config>. The return type [INT] depends on the type of timer. For timers 0 and 2, an *uint8_t* variable is returned. Functions for sixteen bit timers 1 and 3 return a *uint16_t* variable.
Parameters:	none
Return value:	[INT]: current counter value
Remarks:	none

Name:	void <**config**>**_setCounterValue**([INT] t)
Description:	This function sets the counter value of the timer with assigned configuration <config> to t. The datatype [INT] of t depends on the type of timer. For timers 0 and 2, an *uint8_t* is required. The function for the sixteen bit timers 1 and 3 requires a *uint16_t* variable.
Parameters:	[INT] t: new counter value
Return value:	none
Remarks:	none

5.2 Overflow

There are no API functions that are specific for Overflow mode.

5.3 Clear on Compare Match

The following API functions are specific to Clear on Compare Match mode.

Name:	[INT] <**config**>**_getPeriod_ChannelM**(void)
Description:	This function returns the current period of the M-th channel of the timer with configuration <config>. The return type [INT] depends on the type of timer. For timers 0 and 2, an *uint8_t* variable is returned. Functions for sixteen bit timers 1 and 3 return a *uint16_t* variable.
Parameters:	none
Return value:	[INT]: current time period value in ticks
Remarks:	none

Name:	void <**config**>**_setPeriod_ChannelM**([INT] p)
Description:	This function sets the period of the M-th channel of the timer with configuration <config> to p. For timers 0 and 2, the datatype of p is *uint8_t*. For timers 1 and 3, the datatype is *uint16_t*. Timers 0 and 2 only have one channel (A), while timers 1 and 3 have three channels (A, B, C). (See datasheet of AT90CAN128 for details.)
Parameters:	[INT] p: new time period value in ticks
Return value:	none
Remarks:	none

5.4 PWM

The following functions are applicable to all PWM modes.

Name:	[INT] <**config**>**_getPWM_ChannelM**(void)
Description:	This function returns the current PWM duty-cycle value of the M-th output compare channel of the timer with configuration <config>. The return type [INT] depends on the type of timer. For timers 0 and 2, an *uint8_t* variable is returned. Functions for sixteen bit timers 1 and 3 return a *uint16_t* variable.
Parameters:	none
Return value:	[INT]: current PWM duty-cycle in ticks
Remarks:	none

Name:	void <**config**>**_setPWM_ChannelM**([INT] p)
Description:	This function sets the PWM duty-cycle value of the M-th output compare channel of the timer with configuration <config> to p. For timers 0 and 2, the datatype of p is *uint8_t*. For timers 1 and 3, the datatype is *uint16_t*. Timers 0 and 2 only have one channel (A), while timers 1 and 3 have three channels (A, B, C). (See datasheet of AT90CAN128 for details.)
Parameters:	[INT] p: new PWM duty-cycle in ticks
Return value:	none
Remarks:	To avoid phase shifts, you should use Phase Correct or Phase and Frequency Correct PWM mode if you are planning to alter the duty-cycle(s) of a timer at runtime.

5.4.1 Fast PWM

There are no API functions that are specific for Fast PWM mode.

5.4.2 Phase Correct PWM

There are no API functions that are specific for Phase Correct PWM mode.

5.4.3 Phase and Frequency Correct PWM

The following API functions are specific to Phase and Frequency Correct PWM mode.

Name:	[INT] <**config**>**_getPWM_BaseRate**(void)
Description:	This function returns the current PWM base rate value of the timer with configuration <config>. The return type [INT] depends on the type of timer. For timers 0 and 2, an *uint8_t* variable is returned. Functions for sixteen bit timers 1 and 3 return a *uint16_t* variable.
Parameters:	none
Return value:	[INT]: current PWM base rate in ticks
Remarks:	none

Name:	void <**config**>**_setPWM_BaseRate**([INT] p)
Description:	This function sets the PWM base rate of the timer with configuration <config> to p. For timers 0 and 2, the datatype of p is *uint8_t*. For timers 1 and 3, the datatype is *uint16_t*.
Parameters:	[INT] p: new PWM base rate in ticks
Return value:	none
Remarks:	none

References