
Getting Started with Universal Synchronous and Asynchronous Receiver and Transmitter (USART)

Introduction

Author: Alexandru Niculae, Microchip Technology Inc.

The purpose of this document is to describe step-by-step how to configure the USART peripheral on megaAVR® 0-series, tinyAVR® 0- and 1-series, and AVR® DA devices. While this is a complex peripheral and can work in various modes, this document will use it in Asynchronous mode for the following use cases:

- **Send 'Hello World' to a Terminal**
Demonstrates how to send a string to the PC and show it in the terminal.
- **Send Formatted Strings/Send String Templates Using 'printf'**
Enhances the first use case with the ability to use the 'printf' function to send strings over USART.
- **Receive Control Commands**
Many times, the USART is used to implement a command-line interface. This way, the microcontroller can receive control commands via the USART.

Additionally, this document provides information on how to configure the USART in Synchronous mode and One Wire mode.

Note: For each of the use cases described in this document, there are two code examples: One bare metal developed on ATmega4809 and one generated with MPLAB® Code Configurator (MCC) developed on AVR128DA48.



View the ATmega4809 Code Examples on GitHub

[Click to browse repository](#)



View the AVR128DA48 Code Examples on GitHub

[Click to browse repository](#)

Table of Contents

Introduction.....	1
1. Relevant Devices.....	3
1.1. tinyAVR® 0-series.....	4
1.2. tinyAVR® 1-series.....	5
1.3. megaAVR® 0-series.....	6
1.4. AVR® DA Family Overview.....	6
2. Overview.....	7
3. Send 'Hello World'.....	9
4. Send Formatted Strings/Send String Templates Using <code>printf</code>	13
5. Receive Control Commands.....	15
6. Other Implementation Modes.....	17
6.1. Synchronous Mode.....	17
6.2. One-Wire Mode.....	18
7. References.....	19
8. Revision History.....	20
The Microchip Website.....	21
Product Change Notification Service.....	21
Customer Support.....	21
Microchip Devices Code Protection Feature.....	21
Legal Notice.....	22
Trademarks.....	22
Quality Management System.....	23
Worldwide Sales and Service.....	24

1. Relevant Devices

This section lists the relevant devices for this document. The following figures show the different family devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features. Downward migration on tinyAVR® 1-series devices may require code modification due to fewer available instances of some peripherals
- Horizontal migration to the left reduces the pin count and, therefore, the available features
- Devices with different Flash memory sizes typically also have different SRAM and EEPROM

Figure 1-1. tinyAVR® 0-series Overview

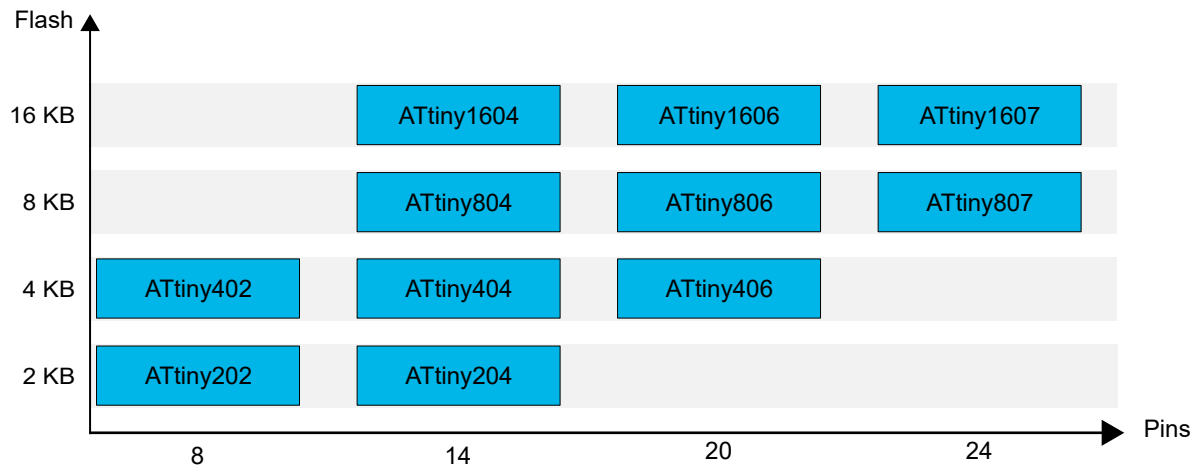


Figure 1-2. tinyAVR® 1-series Overview

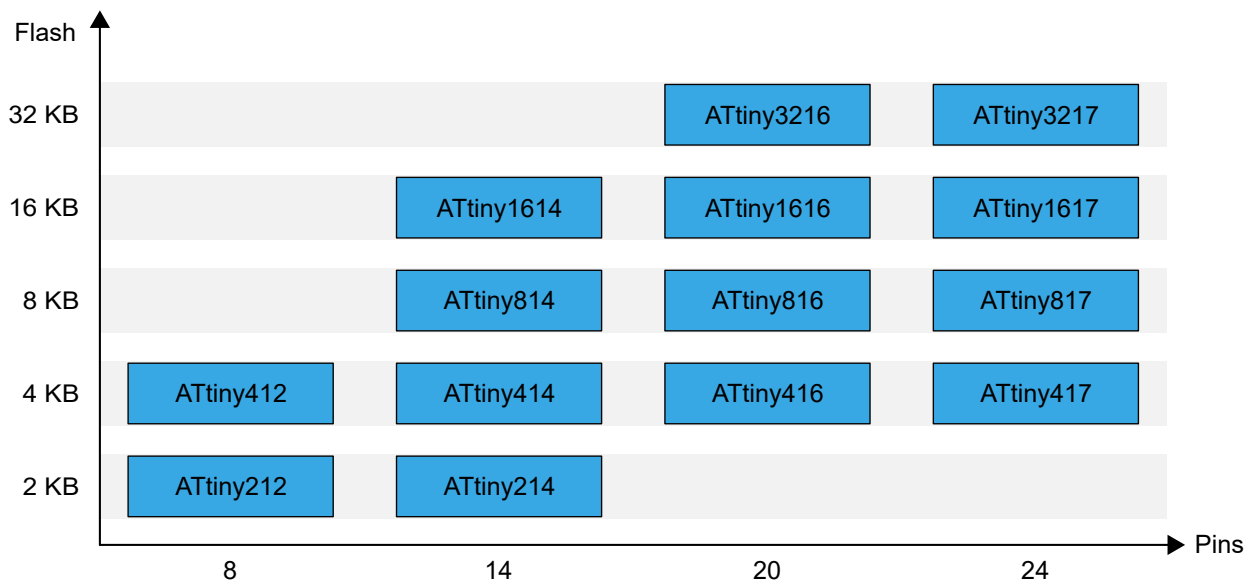


Figure 1-3. megaAVR® 0-series Overview

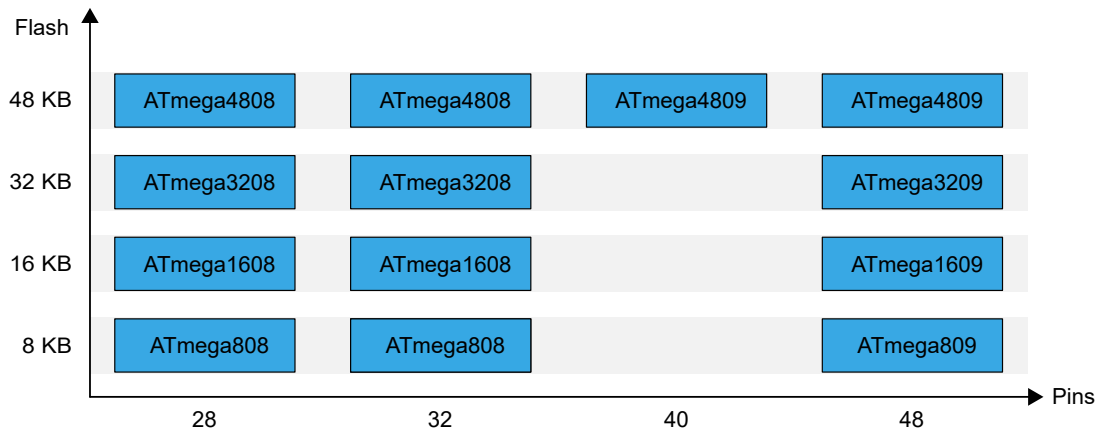
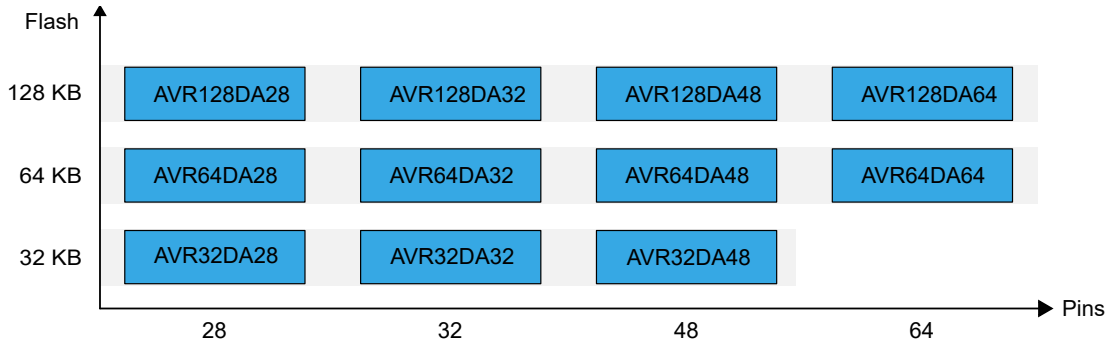


Figure 1-4. AVR® DA Family Overview

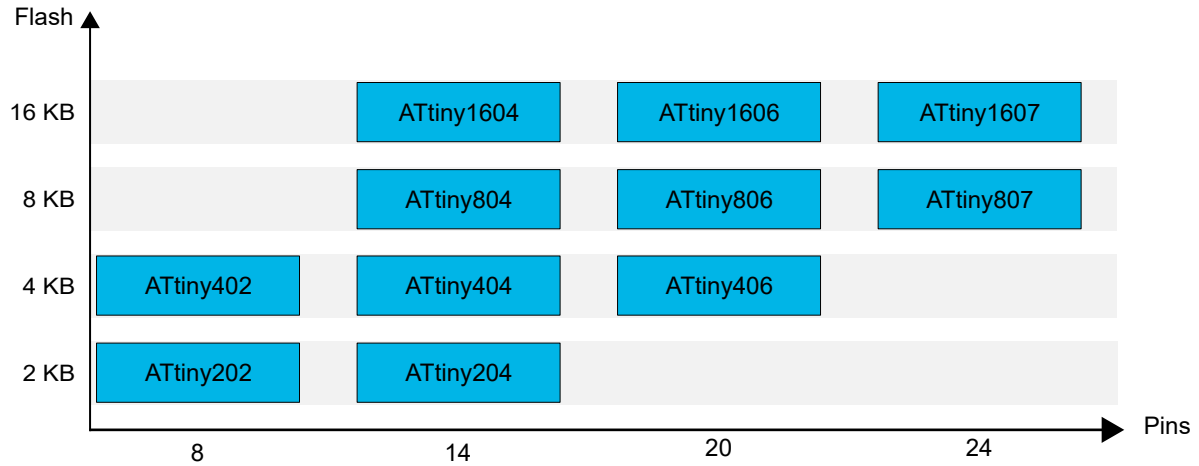


1.1 tinyAVR® 0-series

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

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features
- Horizontal migration to the left reduces the pin count and, therefore, the available features

Figure 1-5. tinyAVR® 0-series Overview



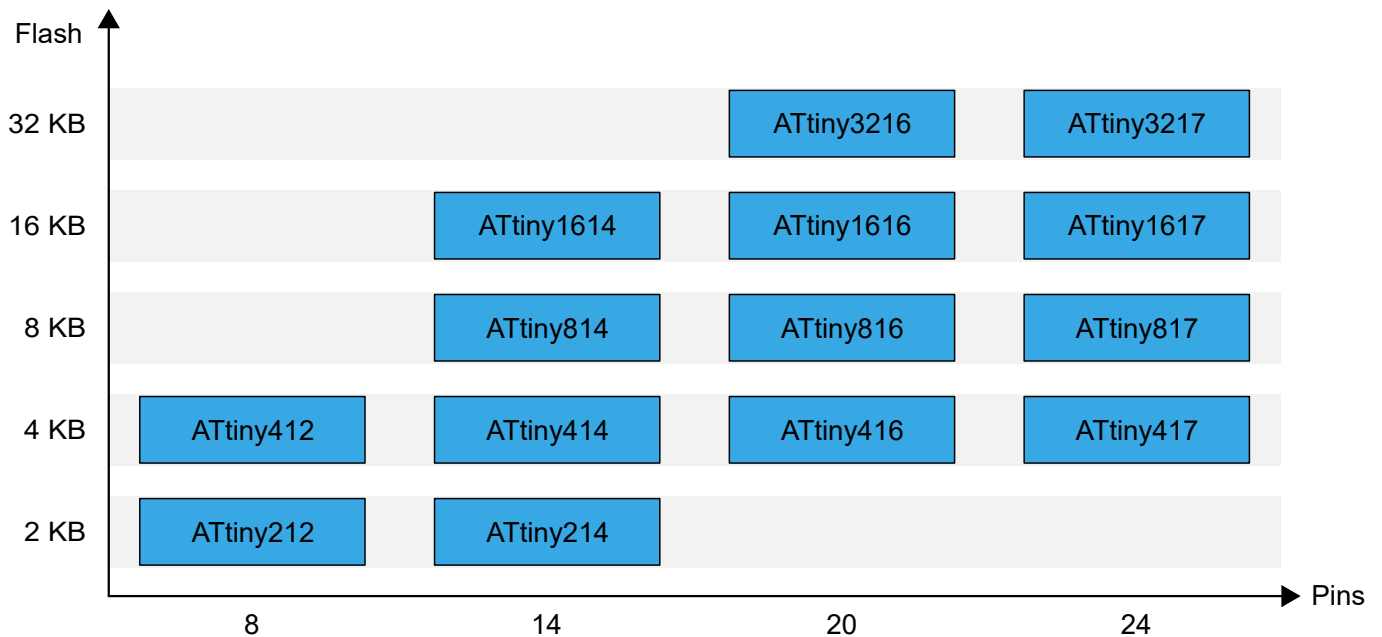
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-series

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

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

Figure 1-6. tinyAVR® 1-series Overview



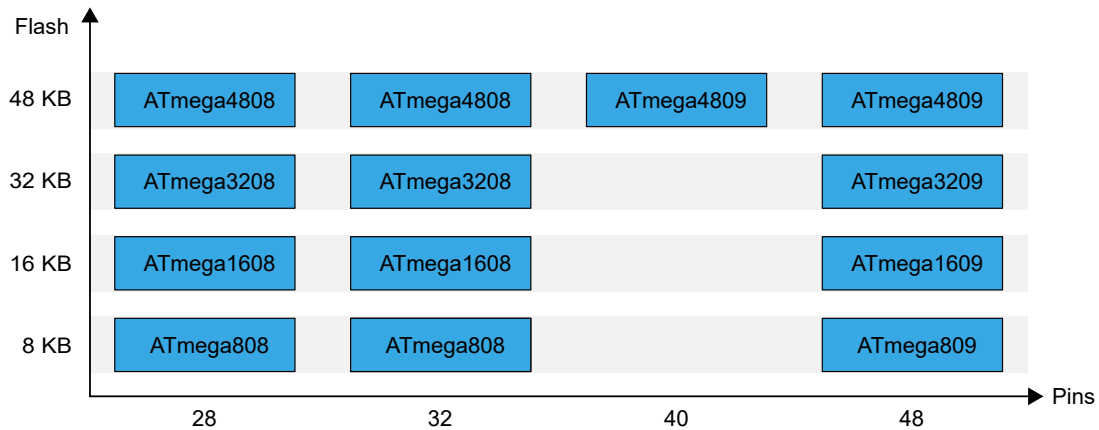
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

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

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

Figure 1-7. megaAVR® 0-series Overview



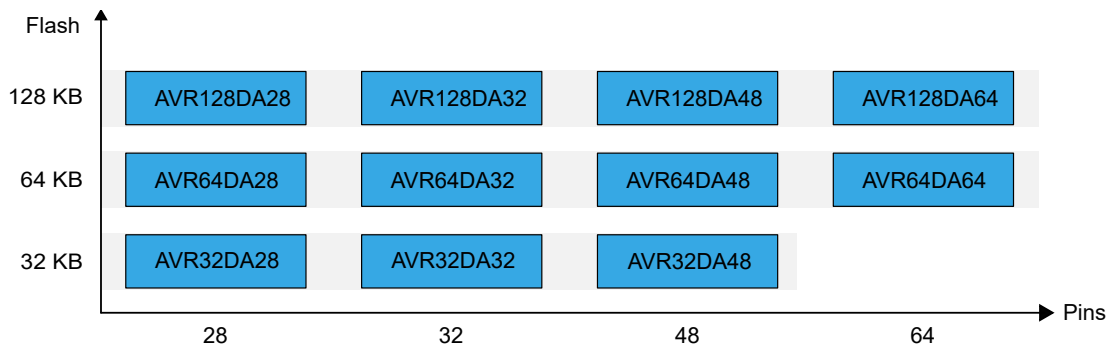
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.4 AVR® DA Family Overview

The figure below shows the AVR® DA devices, laying out pin count variants and memory sizes:

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

Figure 1-8. AVR® DA Family Overview

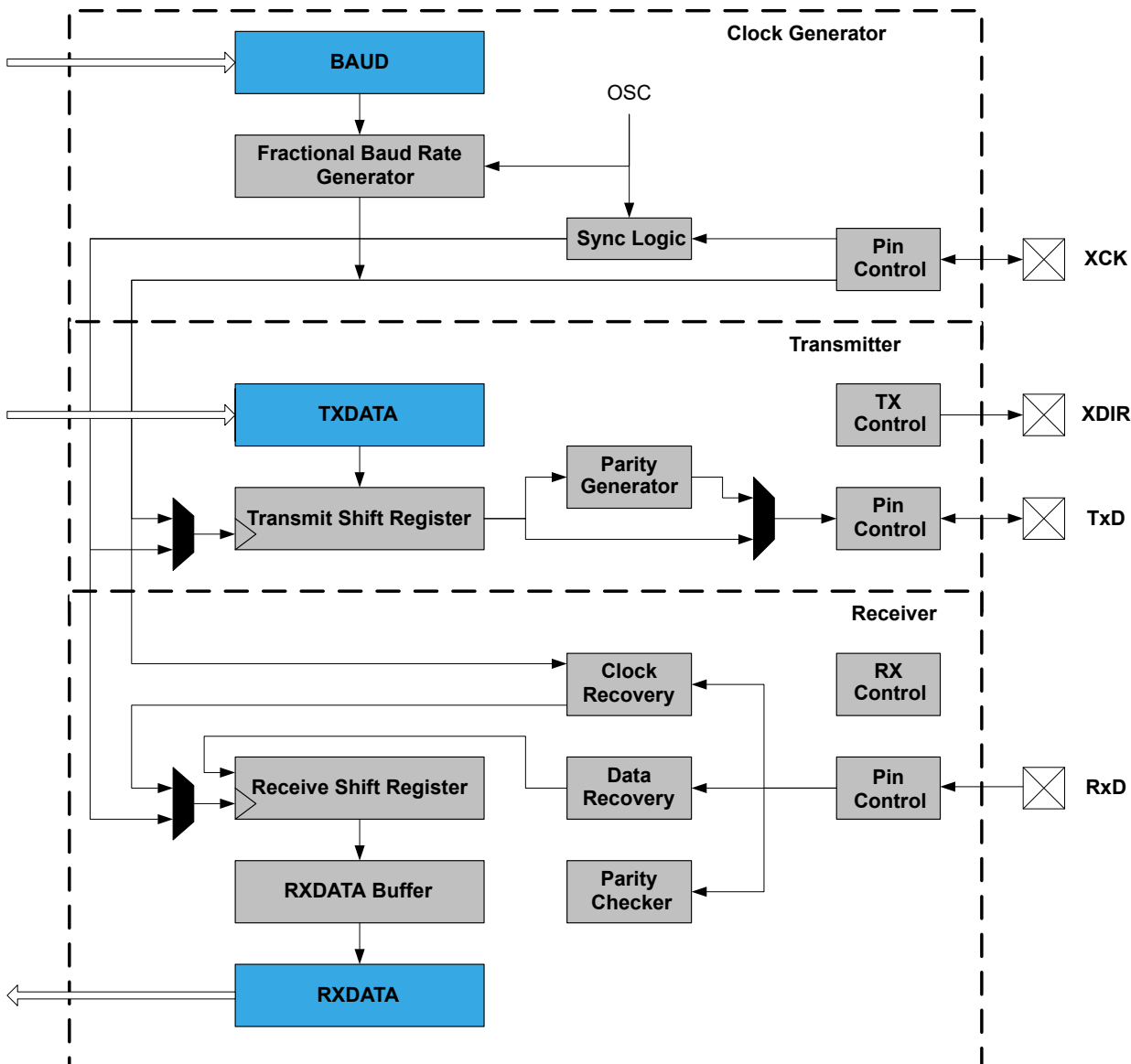


Devices with different Flash memory sizes typically also have different SRAM.

2. Overview

The USART module has four pins, named RX (receive), TX (transmit), XCK (clock) and XDIR (direction). In One-Wire mode only, the TX pin is used for both transmitting and receiving. The downside of this mode is that it only provides half-duplex communication. In Asynchronous mode, both RX and TX pins are used, thus achieving full-duplex communication. The XCK pin is used for clock signal in Synchronous mode, and the XDIR pin is used for RS485 mode.

Figure 2-1. USART Block Diagram



The most common USART configuration is referred to as “9600 8N1”, meaning 9600 baud rate, eight data bits, no parity, and one Stop bit. Therefore, a typical USART frame will have 10 bits (one Start bit, eight data bits, and one Stop bit) and will be able to represent one ASCII character, which means an “8N1” configuration will transmit $\text{BAUD_RATE}/10$ ASCII characters per second.

Note: All examples described in this document will use a 9600 baud rate and “8N1” frame format. The serial terminal must be set for this configuration.

Moreover, the USART is a complex peripheral and can be used to achieve a handful of other protocols such as:

- Host SPI
- Client LIN
- IR Communication
- Addressable USART (also called Multi Processor Communication)
- RS485

3. Send 'Hello World'

This use case demonstrates how to send a string to the PC and visualize it in the terminal. The USART will be configured for Asynchronous mode, and only the TX pin will be used.

The USART1 instance default pins, PC0 (TX) and PC1 (RX), are connected directly to the debugger interface. They can be used to send data bytes to a host PC by using the Virtual COM Port (CDC) interface of the Embedded Debugger (EDBG), bypassing the need for an additional UART to USB converter.

This use case follows the steps:

- Set the baud rate
- Enable the Transmitter (TX)
- Configure the pins

How to Configure the Baud Rate

The baud rate shows how many bits are sent per second. The higher the baud rate, the faster the communication. Common baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200, with 9600 being the most commonly used one.

On the megaAVR 0-series, the maximum baud rate is limited to $1/8 \times$ (maximum USART clock) in Asynchronous mode and $1/2 \times$ (maximum USART clock) in Synchronous mode. To set the baud rate, write to the USARTn.BAUD register:

```
USART1.BAUD = (uint16_t)USART1_BAUD_RATE(9600);
```

Notice the use of the USART1_BAUD_RATE macro to compute the register's value from the baud value. This macro must be defined based on the formula in the image below. This formula depends on the USART configurations, so it might not be the same in other modes.

Figure 3-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Conditions	Baud Rate (Bits Per Seconds)	USART.BAUD Register Value Calculation
Asynchronous	$f_{BAUD} \leq \frac{f_{CLK_PER}}{S}$	$f_{BAUD} = \frac{64 \times f_{CLK_PER}}{S \times BAUD}$	$BAUD = \frac{64 \times f_{CLK_PER}}{S \times f_{BAUD}}$
Synchronous Host	$f_{BAUD} \leq \frac{f_{CLK_PER}}{2}$	$f_{BAUD} = \frac{f_{CLK_PER}}{2 \times BAUD[15:6]}$	$BAUD[15:6] = \frac{f_{CLK_PER}}{2 \times f_{BAUD}}$

S is the number of samples per bit. In Asynchronous operating mode, it is 16 (NORMAL mode) or 8 (CLK2X mode). For Synchronous operating mode, S equals 2.

This is how the USART1_BAUD_RATE macro is defined. It uses F_CPU because the USART clock matches the CPU clock.

```
#define F_CPU 3333333
#define USART1_BAUD_RATE(BAUD_RATE) ((float)(F_CPU * 64 / (16 * (float)BAUD_RATE)) + 0.5)
```

How to Enable the Transmitter and Send Data

Depending on the application needs, the user may choose to enable only the receiver or the USART module transmitter. Since in this use case only the microcontroller sends messages, only the transmitter needs to be enabled.

```
USART1.CTRLB |= USART_TXEN_bm;
```

Before sending data, the user needs to check if the previous transmission is completed by checking the USARTn.STATUS register. The following code example waits until the transmit DATA register is empty and then writes a character to the USARTn.TXDATAL register:

```
void USART1_sendChar(char c)
{
    while (!(USART1.STATUS & USART_DREIF_bm))
    {
        ;
    }
    USART1.TXDATAL = c;
}
```

The Send register is nine bits long. Therefore, it was split into two parts: The lower part that holds the first eight bits, called TXDATAL, and the higher part that holds the remaining one bit, called TXDATAH. TXDATAH is used only when the USART is configured to use nine data bits. When used, this ninth bit must be written before writing to USARTn.TXDATAL, except if CHSIZE in USARTn.CTRLA is set to '9-bit - Low byte first', where USARTn.TXDATAL should be written first.

How to Configure Pins

The TX pin must be configured as an output. By default, each peripheral has some associated pin positions. The pins are described in the *Multiplexed Signals* section in the device-specific data sheet. Each USART has two sets of pin positions. The default and alternate pin positions for USART1 are shown below.

Table 3-1. Multiplexed Signals

Pin name ^(1,2)	USARTn
PC0	1,TxD
PC1	1,RxD
PC2	1,XCK
PC3	1,XDIR
VDD	
GND	
PC4	1,TxD ⁽³⁾
PC5	1,RxD ⁽³⁾
PC6	1,XCK ⁽³⁾
PC7	1,XDIR ⁽³⁾

Notes:

- Pin names are of type Px n , with x being the PORT instance (A,B,C, ...) and n the pin number. Notation for signals is PORTx_PIN n . All pins can be used as event input.
- All pins can be used for external interrupt, where pins Px2 and Px6 of each port have full asynchronous detection.
- Alternate pin positions. For selecting the alternate positions, refer to the PORTMUX documentation.

For this use case, the default USART1 pin position is used; this is PC0 to PC1. The following code sets the TX pin direction to output.

```
PORTC.DIR |= PIN0_bm;
```

To use the alternate pin positions, write to the PORTMUX.USARTROUTEA register.

```
PORTMUX.USARTROUTEA |= PORTMUX_USART10_bm;
```

Note: In this example, the default pin position is used, not the alternate one.

Demo Code

This code example is used to continuously send the 'Hello World!' string through USART. A string is sent character by character. The 'USART1_sendString' function calls the 'USART1_sendChar' function for each character in the 'Hello World!' string. Before sending each character, the 'USART1_sendChar' function waits for the previous character transmission to be completed. This is done by polling the Data Register Empty Interrupt Flag, DREIF, from the STATUS register until it is set.

```
#define F_CPU 3333333
#define USART1_BAUD_RATE(BAUD_RATE) ((float)(F_CPU * 64 / (16 * (float)BAUD_RATE)) + 0.5)

#include <avr/io.h>
#include <util/delay.h>
#include <string.h>

void USART1_init(void);
void USART1_sendChar(char c);
void USART1_sendString(char *str);

void USART1_init(void)
{
    PORTC.DIR &= ~PIN1_bm;
    PORTC.DIR |= PIN0_bm;

    USART1.BAUD = (uint16_t)USART1_BAUD_RATE(9600);

    USART1.CTRLB |= USART_TXEN_bm;
}

void USART1_sendChar(char c)
{
    while (!(USART1.STATUS & USART_DREIF_bm))
    {
        ;
    }
    USART1.TXDATAL = c;
}

void USART1_sendString(char *str)
{
    for(size_t i = 0; i < strlen(str); i++)
    {
        USART1_sendChar(str[i]);
    }
}

int main(void)
{
    USART1_init();

    while (1)
    {
        USART1_sendString("Hello World!\r\n");
        _delay_ms(500);
    }
}
```

Note: For the delay function to work properly, define the CPU frequency before including the `<avr/delay.h>` header.

Note: The default configurations are used for the CPU clock frequency and the USART frame structure. The default CPU and peripheral clock frequency is 3.33 MHz. The USART default frame structure is comprised of eight data bits, no parity bit, and one stop bit (8N1).



View the ATmega4809 Code Example on GitHub

[Click to browse repository](#)

An MPLAB® Code Configurator (MCC) generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:



View the AVR128DA48 Code Example on GitHub

[Click to browse repository](#)

4. Send Formatted Strings/Send String Templates Using `printf`

It is a common use case for an application to send a string with variable fields, for example, when the application reports its state or a counter value. Using formatted strings is a very flexible approach and reduces the number of code lines. This can be accomplished by changing the output stream of the `printf` function.

This use case follows these steps:

- Configure the USART peripheral the same as for the first use case
- Create a user-defined stream
- Replace the standard output stream with the user-defined stream

Usually, when using `printf`, the characters are sent to a stream of data, called standard output stream. On a PC, the standard output stream is handled by the function to display characters on the screen. But streams can be created so that another function handles their data.

The following code creates a user-defined stream that will be handled by the `USART1_printChar` function. This function is a wrapper of the `USART1_sendChar` function but has a slightly different signature to match what `FDEV_SETUP_STREAM` expects as a parameter.

```
static void USART1_sendChar(char c)
{
    while (!(USART1.STATUS & USART_DREIF_bm))
    {
        ;
    }
    USART1.TXDATA = c;
}

static int USART1_printChar(char c, FILE *stream)
{
    USART1_sendChar(c);
    return 0;
}

static FILE USART_stream = FDEV_SETUP_STREAM(USART1_printChar, NULL, _FDEV_SETUP_WRITE);
```

Then replace the standard output stream with the user-defined stream, handled by the USART send function.

```
stdout = &USART_stream;
```

The application can now use `printf` instead of writing to USART registers directly.

```
uint8_t count = 0;
while (1)
{
    printf("Counter value is: %d\r\n", count++);
    _delay_ms(500);
}
```

Note: The `printf` function uses specifiers to mark where to insert variables in the string template. Some of the available specifiers are in the table below:

Table 4-1. printf Specifiers

Specifier	Description
%d	Insert a signed integer
%s	Insert a sequence of characters
%c	Insert a character
%x	Insert integer unsigned in hex format

Other settings do not change and are, therefore, skipped in the code snippets above. See the full code example on GitHub.



View the ATmega4809 Code Example on GitHub

[Click to browse repository](#)

An MPLAB MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:



View the AVR128DA48 Code Example on GitHub

[Click to browse repository](#)

5. Receive Control Commands

One important usage of the USART represents the implementation of a command-line interface. This way, the microcontroller can receive control commands via USART. It is convenient to use the line terminator as a command delimiter, so, for this use case, the USART will read full lines.

This use case follows the steps:

- Configure the USART peripheral same as for the first use case
- Enable the receiver
- Read and store the incoming data until the end of line
- Check if the received data are a valid command; if so, execute it

How to Enable the Receiver and Receive Data

For USART1, the default pin position for RX is Port C pin 1 (PC1). The following line sets the PC1 direction to input.

```
PORTC.DIR &= ~PIN1_bm;
```

Same as the transmitter, the receiver is enabled by writing to the USARTn.CTRLB register.

```
USART1.CTRLB |= USART_RXEN_bm;
```

Before reading the data, the user must wait for the data to be available by polling the Receive Complete Interrupt Flag, RXCIF.

```
uint8_t USART1_read()
{
    while (!(USART1.STATUS & USART_RXCIF_bm))
    {
        ;
    }
    return USART1.RXDATAL;
}
```

How to Read a Line

The following code snippet reads one line of data and stores it in an array. It assumes that a valid line is shorter than the array length.

The array index is reset to zero when reaching the array end to avoid a buffer overflow error in case of longer lines received. The characters '\n' (line feed) and '\r' (carriage return) are ignored because they are part of the line terminator. When '\n' is found, the string end (NULL) is added to the command, and the function 'executeCommand' will call a function based on the value of the command string.

```
char command[MAX_COMMAND_LEN];
uint8_t index = 0;
char c;

/* This delay invalidates the initial noise on the TX line, after device reset. */
_delay_ms(10);

while (1)
{
    c = USART1_readChar();
    if(c != '\n' && c != '\r')
    {
        command[index++] = c;
        if(index > MAX_COMMAND_LEN)
        {
            index = 0;
        }
    }

    if(c == '\n')
    {
        command[index] = '\0';
        executeCommand(command);
        index = 0;
    }
}
```

```
command[index] = '\0';  
index = 0;  
executeCommand(command);  
}  
}
```

In the following code example on GitHub, the USART receives 'ON' and 'OFF' commands, and the microcontroller controls a GPIO output, which can, for example, toggle an LED.



View the ATmega4809 Code Example on GitHub

[Click to browse repository](#)

An MPLAB MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:



View the AVR128DA48 Code Example on GitHub

[Click to browse repository](#)

6. Other Implementation Modes

The applications described above demonstrate the basic USART functionalities. This section describes the USART configured in Synchronous mode and One-Wire mode.

6.1 Synchronous Mode

Figure 6-1. USART Communication Mode (CMODE) Bit Field in Control C Register

Bit	7	6	5	4	3	2	1	0
	CMODE[1:0]		PMODE[1:0]		SBMODE	CHSIZE[2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	1	1

The CMODE bit field in the CTRLC register controls the communication modes.

The disadvantage of the Asynchronous mode is that the receiver chip and the transmitter chip need to use the same baud rate, and exact timing is required. The asynchronous protocols use a separate line for the clock signal, so the chip that generates the clock dictates the communication speed, which is much more flexible in terms of exact timings and creates two roles in the communication: The server that generates the clock and the client that receives the clock.

In the Synchronous USART mode, an additional clock pin, XCK, is used. Same as the RX and TX pins, XCK has a default pin, and changing the PORTMUX register will also change XCK. Configuring the XCK direction decides if the device is a server (generates clock) or a client (receives clock).

To activate the Synchronous mode:

- Configure the XCK pin (PC2) direction as output;

```
PORTC.DIR &= ~PIN2_bm;
```

- Write 0x01 to the CMODE bit field in the USARTn.CTRLC register.

Figure 6-2. USART Communication Mode

Value	Name	Description
0x0	ASYNCHRONOUS	Asynchronous USART
0x1	SYNCHRONOUS	Synchronous USART
0x2	IRCOM	Infrared Communication
0x3	MSPI	Host SPI

```
USART1.CTRLC = USART_CMODE_SYNCHRONOUS_gc;
```



View the ATmega4809 Code Example on GitHub

[Click to browse repository](#)

An MPLAB MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:



View the AVR128DA48 Code Example on GitHub

[Click to browse repository](#)

6.2 One-Wire Mode

Using only one wire effectively reduces the number of pins used for USART communication to one. RX and TX are internally connected, and only TX is used, which means that both incoming and outgoing data will share the same wire, so transmission and reception cannot happen simultaneously. This is called half-duplex communication.

Figure 6-3. Loop-back Mode Enable (LBME) Bit in Control A Register

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	DREIE	RXSIE	LBME	ABEIE	RS485[1:0]	
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Use the LBME bit in the CTRLA register to enable an internal loopback connection between RX and TX. An internal connection between RX and TX can be created by writing to USARTn.CTRLA.

```
USART1.CTRLA |= USART_LBME_bm;
```

This will internally connect the RX and TX pins, but only the TX pin is used. As the TX pin is used for both transmit and receive, the pin direction needs to be configured as an output before each transmission and switched back to input when the transmission ends.

Since RX is connected internally to TX during transmission, it will receive the data sent, which can be used as a collision detection mechanism. If there is another transmission occurring, the received data will not match the transmitted data. An advanced one-wire driver could take advantage of this strategy.



View the ATmega4809 Code Example on GitHub

[Click to browse repository](#)

An MPLAB MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:



View the AVR128DA48 Code Example on GitHub

[Click to browse repository](#)

7. References

1. ATmega4809 web page: <https://www.microchip.com/wwwproducts/en/ATMEGA4809>
2. [megaAVR® 0-series Manual](#) (DS40002015)
3. [ATmega3209/4809 – 48-pin Data Sheet megaAVR® 0-series](#) (DS40002016)
4. ATmega4809 Xplained Pro web page: <https://www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro>
5. AVR128DA48 Product Page: www.microchip.com/wwwproducts/en/AVR128DA28
6. AVR128DA48 Curiosity Nano Evaluation Kit web page: www.microchip.com/Developmenttools/ProductDetails/DM164151
7. [AVR128DA28/32/48/64](#) (DS4000218)
8. [Getting Started with the AVR® DA Family](#) (DS00003429)

8. Revision History

Doc Rev.	Date	Comments
C	01/2021	Updated the GitHub repository links. Added the AVR® DA Family Overview, References, and Revision History sections. Added MCC versions for each use case, running on AVR128DA48. Minor editorial corrections.
B	6/2019	Updated code examples in section 3. 'Hello World' and section 4. 'Send Formatted Strings/ Send String Templates Using <code>printf</code> '. Revision History added. Minor editorial corrections.
A	12/2018	Initial document release.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **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 (FAQs), technical support requests, online discussion groups, Microchip design partner 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

Product Change Notification Service

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: 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 specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "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 is provided for the sole purpose of designing with and using Microchip products. Information 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.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". 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 ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. 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, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Klear, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, WinPath, and ZL 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, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICKit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQL, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, 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.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks 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.

© 2021, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-7425-8

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

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: www.microchip.com/support Web Address: www.microchip.com	Australia - Sydney Tel: 61-2-9868-6733 China - Beijing Tel: 86-10-8569-7000 China - Chengdu Tel: 86-28-8665-5511 China - Chongqing Tel: 86-23-8980-9588 China - Dongguan Tel: 86-769-8702-9880 China - Guangzhou Tel: 86-20-8755-8029 China - Hangzhou Tel: 86-571-8792-8115 China - Hong Kong SAR Tel: 852-2943-5100 China - Nanjing Tel: 86-25-8473-2460 China - Qingdao Tel: 86-532-8502-7355 China - Shanghai Tel: 86-21-3326-8000 China - Shenyang Tel: 86-24-2334-2829 China - Shenzhen Tel: 86-755-8864-2200 China - Suzhou Tel: 86-186-6233-1526 China - Wuhan Tel: 86-27-5980-5300 China - Xian Tel: 86-29-8833-7252 China - Xiamen Tel: 86-592-2388138 China - Zhuhai Tel: 86-756-3210040	India - Bangalore Tel: 91-80-3090-4444 India - New Delhi Tel: 91-11-4160-8631 India - Pune Tel: 91-20-4121-0141 Japan - Osaka Tel: 81-6-6152-7160 Japan - Tokyo Tel: 81-3-6880-3770 Korea - Daegu Tel: 82-53-744-4301 Korea - Seoul Tel: 82-2-554-7200 Malaysia - Kuala Lumpur Tel: 60-3-7651-7906 Malaysia - Penang Tel: 60-4-227-8870 Philippines - Manila Tel: 63-2-634-9065 Singapore Tel: 65-6334-8870 Taiwan - Hsin Chu Tel: 886-3-577-8366 Taiwan - Kaohsiung Tel: 886-7-213-7830 Taiwan - Taipei Tel: 886-2-2508-8600 Thailand - Bangkok Tel: 66-2-694-1351 Vietnam - Ho Chi Minh Tel: 84-28-5448-2100	Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 Denmark - Copenhagen Tel: 45-4485-5910 Fax: 45-4485-2829 Finland - Espoo Tel: 358-9-4520-820 France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 Germany - Garching Tel: 49-8931-9700 Germany - Haan Tel: 49-2129-3766400 Germany - Heilbronn Tel: 49-7131-72400 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: 47-72884388 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