AVR054: Run-time calibration of the internal RC oscillator

8-bit AVR® Microcontrollers

Application Note

Features

- **Calibration of internal RC oscillator via UART**
- **LIN 2.0 compatible synchronization/calibration to within +/-2% of target frequency**
- **Alternate run-time synchronization/calibration to within +/-1% of target frequency**
- **Support for all AVRs with tunable RC oscillator**
- **Enables robust UART communication with low cost clock sources in varying operating conditions**

Introduction

This application note describes how to calibrate the internal RC oscillator via the UART. The method used is based on the calibration method used in the Local Interconnect Network (LIN) protocol, synchronizing a slave node to a master node at the beginning of every message frame. This allows a slave node to communicate with other nodes at baud rates within specified limits, even when running on a low-cost clock source, such as the internal RC oscillator.

The majority of the present AVR microcontrollers offer the possibility to run from an internal RC oscillator. The internal RC oscillator frequency can in most AVRs be calibrated at runtime to within +/-1% of the frequency specified in the datasheet for the device. This feature is ideal for synchronization purposes, and offers significant cost savings compared to using an external oscillator.

Note that this implementation uses the synchronization signal to alter the frequency of the internal RC oscillator, which again alters the baud rate of the UART module. The terms “synchronization” and “calibration” in this case essentially means the same, and will be used interchangeably. The choice of expression is merely related to the objective.

Rev. 2563A-AVR-07/04

Theory of operation – the internal RC oscillator

In production the internal RC is calibrated at either 5V or 3.3V. Refer to the datasheet of the individual devices for information about the operating voltage used during calibration. The accuracy of the factory calibration is within +/-3 or +/-10% (refer to the datasheet). If a design's need for accuracy is beyond what can be offered by the standard calibration in factory by Atmel, it is possible to perform a secondary calibration of the RC oscillator. By doing this it is possible to obtain a frequency accuracy within +/-1 (+/-2% for those with an 10% accuracy from factory calibration). A secondary calibration can thus be performed to improve or tailor the accuracy or frequency of the oscillator.

Clock selection

The AVR fuse settings control the system clock source being used. To use the internal RC oscillator, the corresponding fuse setting must be selected. An overview of the fuses is available in the datasheets.

Base-frequency

The following sections provide an overview of the internal RC oscillators available in the AVR microcontrollers.

Some AVRs have one RC oscillator, while others have up to 4 different RC oscillators to choose from. The frequency ranges from 1MHz to 9.6MHz. To make the internal RC oscillator sufficiently accurate an Oscillator Calibration register, OSCCAL, is present in the AVR IO file. The OSCCAL register is one byte wide. The purpose of this register is to be able to tune the oscillator frequency. This tuning is utilized when calibrating the RC oscillator.

When a device is calibrated by Atmel the calibration byte is stored in the Signature Row of the device. The calibration byte can vary from one device to the other, as the RC oscillator frequency is process dependent. If a device has more than one oscillator a calibration byte for each of the RC oscillators is stored in the Signature Row.

The default RC oscillator calibration byte is in most devices automatically loaded from the Signature Row and copied into the OSCCAL register at start-up. For example, the default ATmega8 clock setting is the internal 1MHz RC oscillator; for this device the calibration byte corresponding to the 1MHz RC oscillator is automatically loaded at start-up. If the fuses are altered so that the 4MHz oscillator is used instead of the default setting, the calibration byte must be loaded into the OSCCAL register manually. A programming tool can be used to read the 4MHz calibration byte from the Signature Row and store it in a Flash or EEPROM location. The main program reads this location and copies it into OSCCAL at run-time.

RC Oscillator overview

The base frequency of an oscillator is defined as the unscaled oscillator frequency. Different RC oscillators have been utilized in the AVR microcontrollers throughout the history. An overview of the devices and their RC oscillators is seen in Table 1. The device list is sorted by oscillator type, which is also more or less equivalent to sorting them by release date. Only devices with tunable oscillators are listed in the table.

Table 1. Oscillator frequencies and features of devices with internal RC oscillator(s). Grouped by oscillator version.

Oscillator version	Device	RC oscillator frequency [MHz]	CKDIV	PRSCK
1.1	ATtiny12	1.2	-	-
1.2	ATtiny15	1.6	-	-
2.0	ATmega163	1.0	-	-
2.0	ATmega323	1.0	-	-
3.0	ATmega8	1.0, 2.0, 4.0, and 8.0	-	-
3.0	ATmega16	1.0, 2.0, 4.0, and 8.0	-	-
3.0	ATmega32	1.0, 2.0, 4.0, and 8.0	-	-
3.1	ATmega64	1.0, 2.0, 4.0, and 8.0	-	XDIV (1)
3.1	ATmega128	1.0, 2.0, 4.0, and 8.0	-	XDIV (1)
3.0	ATmega8515	1.0, 2.0, 4.0, and 8.0	-	-
3.0	ATmega8535	1.0, 2.0, 4.0, and 8.0	-	-
4.0	ATmega162	8.0	Yes	Yes
4.0	ATmega169	8.0	Yes	Yes
4.1	ATtiny13	4.8 and 9.6	Yes	Yes
4.2	ATtiny2313	4.0 and 8.0	Yes	Yes
5.0	ATmega48	8.0	Yes	Yes
5.0	ATmega88	8.0	Yes	Yes
5.0	ATmega168	8.0	Yes	Yes

Note: 1. The prescaler register is in these devices are named XDIV.

Version 1.x oscillators

This version is the earliest internal RC for AVR that can be calibrated. It is offered with frequencies ranging from 1.2MHz to 1.6MHz. The calibration byte is stored in the Signature Row, but isn't automatically loaded at start-up. The loading of the OSCCAL register must be handled at run-time by the firmware. The oscillator frequency is highly dependent on operating voltage and temperature in this version.

Version 2.x oscillators

This oscillator is offered with a frequency of 1MHz. The dependency between the oscillator frequency and operating voltage and temperature is reduced significantly compared to version 1.x.

Version 3.x oscillators

The oscillator system is expanded to offer multiple oscillator frequencies. Four different RC oscillators with the frequencies 1, 2, 4, and 8MHz are present in the device. This version features automatic loading of the 1MHz calibration byte from the Signature Row. Due to the fact that 4 different RC oscillators are present, 4 different calibration bytes are stored in the Signature Row. If frequencies other than the default 1MHz are desired, the OSCCAL register should be loaded with the corresponding calibration byte at run-time.

Version 4.x oscillators

A single oscillator frequency of 8MHz is offered in version 4.0. For later 4.x versions, two frequencies are offered: 4 and 8MHz for ATtiny2313, and 4.8 and 9.6MHz for the ATtiny13. The OSCCAL register is changed so that only 7 bits are used to tune the frequency for the selected oscillator. The MSB is not used. Auto loading of the default calibration value is present and PRSCK is automatically set according to the CKDIV fuse.

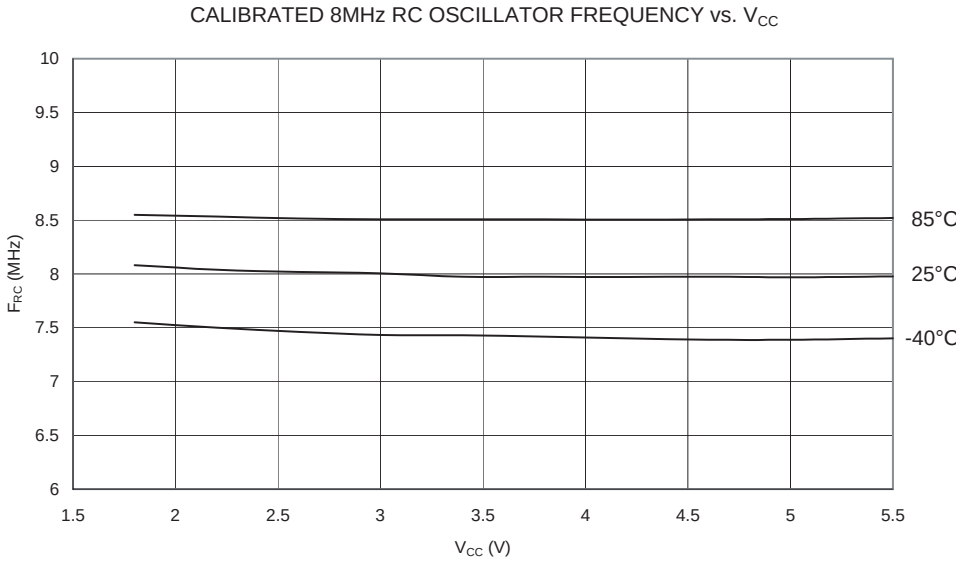
Version 5.x oscillators

A single oscillator frequency of 8MHz is offered in version 5.0 All 8 bits in the OSCCAL register are used to tune the oscillator frequency. Auto loading of the default calibration value and system clock prescaler is present. The OSCCAL register is split in two parts. The MSB of OSCCAL selects one of two overlapping frequency ranges, while the 7 least significant bits are used to tune the frequency within this range.

Oscillator characteristics

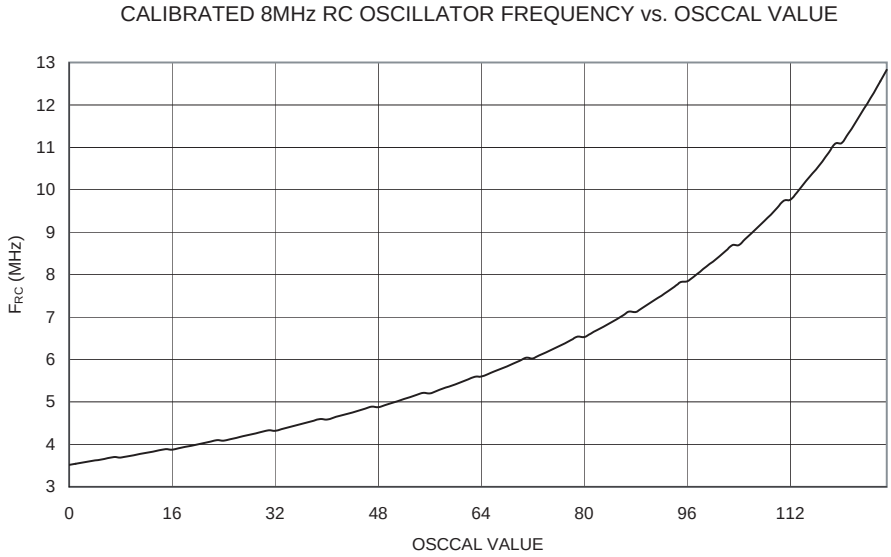
The frequency of the internal RC oscillator is depending on the temperature and operating voltage. An example of this dependency is seen in Figure 1, which shows the frequency of the 8MHz RC oscillator of the ATmega169. As seen from the figure, the frequency increases with increasing temperature, and decreases slightly with increasing operating voltage. These characteristics will vary from device to device. For details on a specific device refer to its datasheet.

Figure 1. Oscillator frequency and influence by temperature and operating voltage. ATmega169 calibrated 8MHz RC oscillator frequency vs. V_{cc}.



All devices with tunable oscillators have an OSCCAL register for tuning the oscillator frequency. An increasing value in OSCCAL will result in a “pseudo-monotone” increase in frequency. The reason for calling it pseudo-monotone is that for some unity increases of the OSCCAL value the frequency will not increase or will decrease slightly. However, the next unity increase will always increase the frequency again. In other words, incrementing the OSCCAL register by one may not increase the frequency, but increasing the OSCCAL value by two will always increase the frequency. This information is very relevant when searching for the best calibration value to fit a given frequency. An example of the pseudo-monotone relation between the OSCCAL value and the oscillator frequency can be seen in Figure 2, which is the 8MHz RC oscillator of ATmega169. Note that since the OSCCAL register only uses 7 bits for tuning the oscillator in ATmega169, the maximum frequency is corresponding to OSCCAL = 128. Version 5.x oscillators differ a little bit from this description, because the MSB of OSCCAL selects one of two frequency ranges. Within each of the frequency ranges (MSB constant), version 5.x oscillators exhibit the same pseudo-monotonic characteristics as other versions of the oscillator.

Figure 2. ATmega169 calibrated RC oscillator frequency as a function of the OSCCAL value.



For all tunable oscillators it is important to notice that it is not recommended to tune the oscillator more than 10% off the base frequency specified in the datasheet. The reason for this is that the internal timing in the device is dependent on the RC-oscillator frequency.

Knowing the fundamental characteristics of the RC oscillators, it is possible to make an efficient calibration routine that calibrates the RC oscillator to a given frequency, within 10% of the base frequency, at any operating voltage and at any temperature with an accuracy of +/-1%.

Frequency settling time

When a new OSCCAL value has been set, it can take some time for the internal RC oscillator to settle at the new frequency. This settling time will vary on the different versions of the RC oscillator. Generally, the oscillator settles faster for small changes in OSCCAL, than for large changes. This settling time is under no circumstances any longer than 5 microseconds. Allow the oscillator to settle at its new frequency before making any frequency measurements for calibration.

The LIN synchronization method

The Local Interconnect Network (LIN) standard is designed to make reliable communication possible even when using low-cost clock sources, such as the internal RC oscillator. Due to the inherent inaccuracy and environment-dependent characteristics of such clock sources, synchronization measures are included in the protocol. The LIN synchronization principles are used as a basis for the synchronization methods described in this application note.

A LIN network consists of one master node and several slave nodes. The master node is responsible for controlling all communication on the bus. In LIN terminology, communication occurs by sending message frames on the bus. Every message frame starts with a frame header, initiated by the master node. The header starts with a BREAK and SYNCH pattern, allowing slave nodes to synchronize to the master before any communication on the bus is initiated. The BREAK/SYNCH pattern consists of:

- BREAK signal: At least 13 bit times of dominant (low) value. See Figure 3.
- BREAK DELIMITER: At least 1 bit time of recessive (high) value. See Figure 3.
- SYNCH byte: A 0x55 is transmitted. Including the start and stop bits, this results in a transmitted bit pattern of 0101010101. See Figure 4. (Note that the bit-transmission order is lsb first).

Figure 3. Break signal

Figure 4. SYNCH byte

After the SYNCH byte, an identifier is transmitted. The identifier uniquely defines which slave node is supposed to transmit data on the bus, and what information is requested from that slave node.

The rest of the LIN protocol is outside the scope of this application note, since only the synchronization is of interest. Refer to “LIN specification package” for details on the complete LIN protocol.

In the following, the signal between two falling edges during SYNCH will be called a synchronization cycle. Using this terminology, the SYNCH byte consists of five synchronization cycles.

Binary and neighbor search

Three different calibration/synchronization methods are described in this application note. One is compatible with the LIN 2.0 specification. The other two are based on the LIN synchronization method, but are not LIN compatible. Common to all three is the way the OSCCAL value is calculated.

The search method

A binary search is used to calculate the OSCCAL value that will make the internal RC oscillator produce the desired frequency, within specified accuracy limits.

The binary search

The binary search works in the following way:

1. Start with an OSCCAL value in the middle of the search-range.
2. Set step-size to 1/4 of the search range.
3. Decide if the current frequency is too high or too low.
4. If the current frequency is too high, subtract step-size from OSCCAL, if it is too low, add step-size to OSCCAL. If frequency is within accuracy limits, do not change OSCCAL.
5. Divide step-size by 2.
6. If step-size is 0, the search is complete. Abort search, or jump to neighbor search. (Described later)
7. Jump to step 3

This search method is optimal (when it comes to worst-case run time) when searching data with a strictly monotone relationship. As we know, the relationship between OSCCAL and resulting oscillator frequency is not completely monotone. However, it is close enough for the binary search to be the most efficient way to find a

value in the neighborhood of the optimal OSCCAL value. From this point it is easy to find the optimal value.

The neighbor search

To do this, the neighbor values have to be examined. Since an increase in clock frequency is guaranteed when increasing OSCCAL by a value of 2 or more, problems are only expected at the last iteration of the binary search, when the step-size is 1. In this case an increased OSCCAL value might result in a decrease in clock frequency, and vice versa. One more step in the same direction guarantees a frequency change in the desired direction. This step might however just be large enough to counter the effect of the last step. Thus, to make sure that the optimal OSCCAL value is found, even one more step should be taken. The Timer/Counter0 value of each of these iterations is then saved, and compared to the desired number of clock cycles. The OSCCAL value that produces the clock frequency closest to the desired frequency is used. This will be referred to as a neighbor search.

Using the binary search with a synch signal

The SYNCH byte consists of only 5 synchronization cycles. If we were to measure complete synchronization cycles, then calculate a new OSCCAL value and then try again, only two iterations of the binary search would be possible. After a measurement, calculations must be performed, a new OSCCAL value must be set, and the oscillator must be allowed to settle at the new frequency.

Instead, the bit times can be measured when the RXD line is low to calculate the current oscillator frequency. The time left when the RXD line is high can then be used to calculate new OSCCAL values and let the oscillator settle at the new value. This, of course, requires the masters low and high bit times to be equal in length. This is the case for most UARTs.

Note that the maximum number of search iterations that can be performed equals the number of synchronization cycles transmitted on the bus.

Range

Equation 1 shows the maximum change of OSCCAL value during a binary search.

Equation 1.

$$c_{\max} = \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

Here c_{max} is the maximum change, and n is the number of iterations. The total range of one binary search then spans over $r = 2c_{\max} + 1 = 2^{n+1} - 1$ values.

If the neighbor search is used, 2 synchronization cycles must be reserved for it in the end of the synchronization. These must be subtracted from the cycles available to the binary search. To cover the whole range of an 8 bit OSCCAL register, 7 iterations of the binary search are needed, using 7 synchronization cycles. If neighbor search is included, 2 more cycles would be needed, giving a total of 9. Corresponding values for a 7 bit OSCCAL register is 6 without neighbor search, and 8 with neighbor search.

Accuracy

The accuracy of the search depends on the search method used:

If only the binary search is used, the optimal OSCCAL value is not always found. However, the frequency found should still be within +/-2% of the desired frequency for parts that can be calibrated to +/-1% of target frequency. When the binary search is used alone, the search should abort when a system clock frequency within the required accuracy limits is found. This should be done to avoid problems in the last step due to the pseudo-monotone relation between OSCCAL and internal RC oscillator frequency.

When the neighbor search is used in addition to the binary search, the optimal OSCCAL value, at the time of synchronization, will always be found if it lies inside the search range. The resulting clock frequency will then be within +/-1% of the desired

frequency, for parts that can be calibrated to within this accuracy. To accomplish this, the search is only aborted if a perfect match is found, i.e. the measured number of CPU cycles during synchronization signal low periods exactly matches the desired number of CPU clock cycles. Even if there is an OSCCAL value that produces a clock frequency closer to the desired one, we will not be able to measure the difference. Further search is thus not necessary.

The LIN synchronization method (LIN 2.0 compatible)

This method performs synchronization according to the LIN 2.0 specification.

According to the LIN bus specification, the synchronization should be finished after the synchronization byte is transmitted, and the slave should then be able to transmit at a baud rate with a deviation of no more than +/-2% of the master's baud rate.

The synchronization byte consists of 5 synchronization cycles, which equals 5 iterations of the binary search. The maximum OSCCAL range that can be searched during synchronization is then $r = 2^6-1 = 63$ values. It is not recommended to use a neighbor search when the LIN synchronization method is used, since the resulting search range will be very limited. The 63 OSCCAL values covered by the binary search, however, is a significant portion of the possible values. It is still possible to calibrate to within +/-2% of desired frequency, which is within LIN specification requirements.

OSCCAL should be set to the default value before every synchronization.

Double SYNCH byte synchronization

To search the full range of an 8 bit OSCCAL register, with neighbor search, 7 synchronization cycles are needed for the binary search, in additions to 2 synchronization cycles for the neighbor search. This equals 9 synchronization cycles, almost 2 synchronization bytes. (Including start and stop bits). This method is not LIN compatible, but it is capable of performing a search of the full OSCCAL register, allowing the part to operate in the full temperature and voltage range specified in the data sheet. To be able to search the full range of OSCCAL values, OSCCAL should be loaded with the maximum OSCCAL value divided by 2 before every synchronization.

To use this method, the single SYNCH byte must be replaced with two consecutive synch bytes All bit times during the dual SYNCH byte must be equal.

Repeated frame synchronization

This synchronization procedure is very similar to the LIN synchronization procedure. The difference is that the slave node does not guarantee that it can synchronize to the master in one attempt. If the first synchronization attempt is not sufficient, the master node will not be able to receive data from the slave correctly. In this case, the master will issue a new BREAK/SYNCH on the bus during the message frame. If the identifier following this repeated BREAK/SYNCH signal equals the identifier of the interrupted message frame, this is a signal to the slave that the last synchronization failed. In this case the slave will start a new synchronization without loading the default OSCCAL value, and then try to transmit again. The default OSCCAL value should only be loaded when the last transmission was successful.

The repeated frame synchronization method allows the full OSCCAL range to be searched, but it can require several attempts when searching for OSCCAL values far from the default.

LIN compatibility

Note that the last two methods described here, are not compatible with the LIN standard. LIN devices must be able to synchronize to within +/-2% of the master during one SYNCH byte. Only the first synchronization method guarantees this.

How to chose calibration / synchronization method?

The LIN synchronization method should be used if the device is supposed to operate in a LIN environment. It could also be used in other applications, if the accuracy provided is tight enough. The LIN synchronization method is well suited for real-time applications, since worst-case time taken to perform synchronization is well defined.

If the increased accuracy is needed, the double SYNCH byte synchronization method is recommended. This method also possesses the same real-time properties as the LIN synchronization method.

If the application must handle large variations in temperature and supply voltage, but operates in constant conditions most of the time, the Repeated frame synchronization can be a good alternative. It is not, however, recommended for real-time applications, since the worst-case synchronization time is a lot longer than average synchronization time.

Since additional code is required to implement the neighbor search, this should only be used when the added accuracy is really needed.

Implementation

This section describes how the run-time calibration can be implemented on an AVR.

Hardware

The UART module makes a good choice for a LIN implementation. This application note uses the UART to implement the LIN protocol; however any general I/O pin can be used.

External interrupt 0 (INT0) is used to detect edges to facilitate timing. The UART RXD pin must therefore be connected to the INT0 pin, to facilitate the synchronization timing.

Software

In the following sections the firmware needed to do run-time calibration is described.

Initialization

When the AVR is reset, it must be configured in the following way:

- Initialize the UART.
- Configure INT0 pin.
- Read default OSCCAL value from EEPROM or flash, if needed.

Detection of the break signal

The BREAK signal can be detected in two ways:

When the AVR is in sleep mode, INT0 must be set up to trigger on low level. The BREAK signal is of sufficient length to wake up the AVR before the SYNCH byte is issued on the bus. Since the UART RXD pin is connected to the INT0 pin, the UART receiver should be disabled before entering sleep to ensure that the BREAK is handled by the INT0 Interrupt Service Routine (ISR).

When the AVR is running, the “RX Complete Interrupt” must be enabled to detect the BREAK signal. The UART will try to interpret the BREAK signal as a byte being transmitted on the bus, but the length of the BREAK signal will cause a frame error because no stop bit is detected in time. This will cause the “Frame Error” (FE) and “UART Receive Complete” flags in the UART Control and Status registers to be set. Since “RX Complete Interrupt” is enabled, this will cause the UART Receive Complete (RXC) ISR to be run. The FE flag then indicates that a BREAK signal was issued.

This configuration ensures that either the INT0 ISR, or the UART RXC ISR is run as soon as a break signal is detected. See Figure 5. The “Process character” and “Perform synchronization” blocks refer to functionality that should be included in these ISRs, but are not relevant to the break detection. The “Perform synchronization” block is where the actual calibration/synchronization will take place. This block is covered in the next section. The “Prepare for synch” block is shown in Figure 6. It is common to both interrupt routines, and is implemented as a macro to avoid function calls within interrupt service routines.

Figure 5. BREAK detection in UART RXC and INT0 Interrupt Service Routines

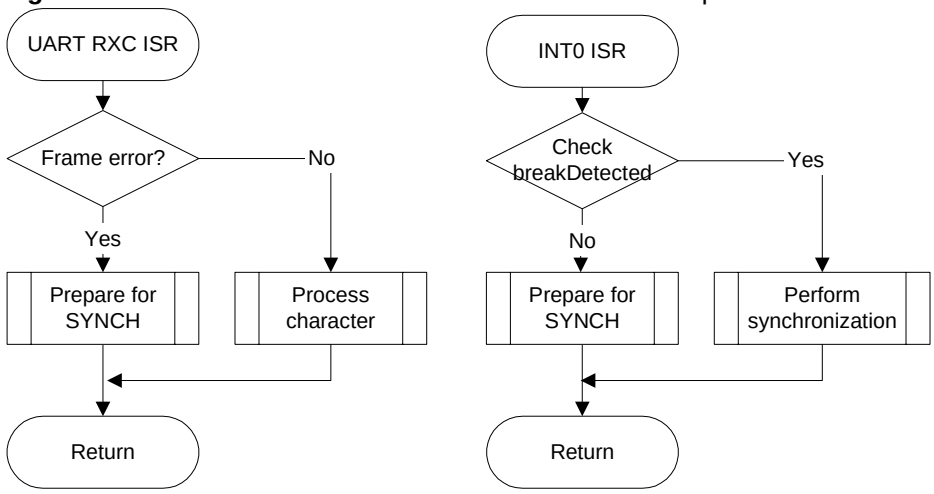
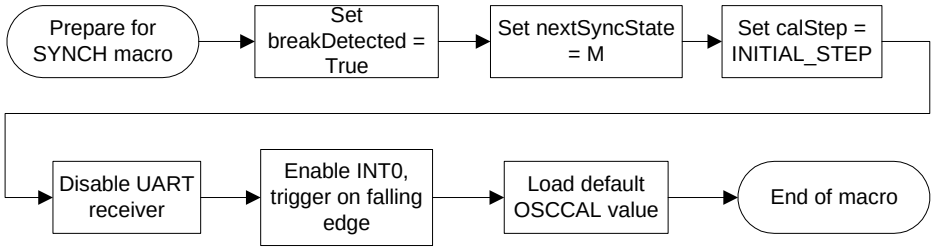


Figure 6. “Prepare for SYNCH” macro



Synchronization

Flowcharts for the LIN synchronization method and the Double SYNCH byte method are shown in Figure 7 and Figure 8. The Repeated frame synchronization method is not shown here, since it requires only a small modification to the LIN synchronization method.

The synchronization algorithm described in this document measures the baud rate on the RXD line during the SYNCH signal, and changes the frequency of the internal RC oscillator to obtain the desired clock frequency and/or baud rate. The UART Baud Rate Register is never changed during synchronization.

After the BREAK signal is detected, the device should be ready to process the SYNCH byte. The synchronization is performed in the INT0 ISR. It is vital to the synchronization algorithm that the timing is absolutely correct. All ISRs with higher priority should therefore be disabled during synchronization, and the “Global Interrupt Enable” flag must be set at all times during synchronization except during execution of ISRs, where interrupts are disabled by default by the AVR.

The 8 bit Timer/Counter0 is used to count the number of clock cycles per bit time on the bus. When running at high clock frequency to baud rate ratio, 8 bits might not be

enough to count the clock cycles during one bit time. The Timer/Counter0 overflow flag can then be used as a 9th bit.

Since the INT0 ISR performs both BREAK signal detection and synchronization timing, a global flag, “breakDetected” is set after a break is detected. This flag is then cleared when the synchronization procedure is finished.

A second global flag, nextSynchState, is used to control the actual synchronization. The value of this flag determines which synchronization state to enter when the next edge is detected on the bus. Three states are used: Measuring (M), Binary search (B) and Neighbor Search (N).

NextSynchState is set to M when waiting for a falling edge on the bus. When entering INT0 ISR in this state Timer/Counter0 is reset and INT0 is set up to trigger on rising edge. The nextSynchState flag is then changed to either B or N state.

NextSynchState is changed to B or N when waiting for a rising edge on the bus. When entering INT0 ISR in these states the value of Timer/Counter0 is read, and one iteration of the search corresponding to the nextSynchState flag is performed. This results in a new OSCCAL value to test. Unless this is the last iteration of the search, the nextSynchState flag is set to M and INT0 is set up to trigger on falling edge.

In this way, the algorithm alternates between the states until the synchronization is finished.

Timing accuracy

It can be seen from the flowcharts in Figure 7 and Figure 8 that Timer/Counter0 is both read and reset every time INT0 ISR is run. It would maybe be more intuitive to reset the timer in the beginning of every measurement iteration and read it in the beginning of every computational iteration. This would however add to the complexity of the code, making it larger, slower and giving it less predictable timing. It is extremely important that the number of cycles between the read statement and the reset statement can be found exactly, to make sure that the synchronization is correct.

It is not really necessary to stop and start the counter in the INT0 ISR if Timer/Counter0 is used in 8 bit mode. In this case the counter must be started at initialization time. However, if the overflow flag is used as a ninth bit, the counter must be stopped while the counter register and the overflow flag is reset to ensure reading of the nine bit value as an atomic operation.

Code size

The source code included with this application note results in quite different code sizes when compiled, depending on synchronization method and timer resolution used. Table 2 lists approximate code sizes for different configurations. Only code generated from “lin_synch.c”, and “double_synch_byte.c” is included, since this best represents the “cost” of including this code in a project.

Note that these numbers are obtained by compiling the source code for ATtiny2313 with IAR C/EC++ Compiler for AVR v3.20A with no optimizations enabled. Code size will vary with different devices.

Table 2. Compiled code size for different configurations

Synchronization method	Timer resolution	Code size
LIN	8 bits	~ 370 B
LIN	9 bits	~ 410 B
Double SYNCH byte	8 bits	~ 550 B
Double SYNCH byte	9 bits	~ 570 B

Figure 7. The Lin synchronization method – Flowchart

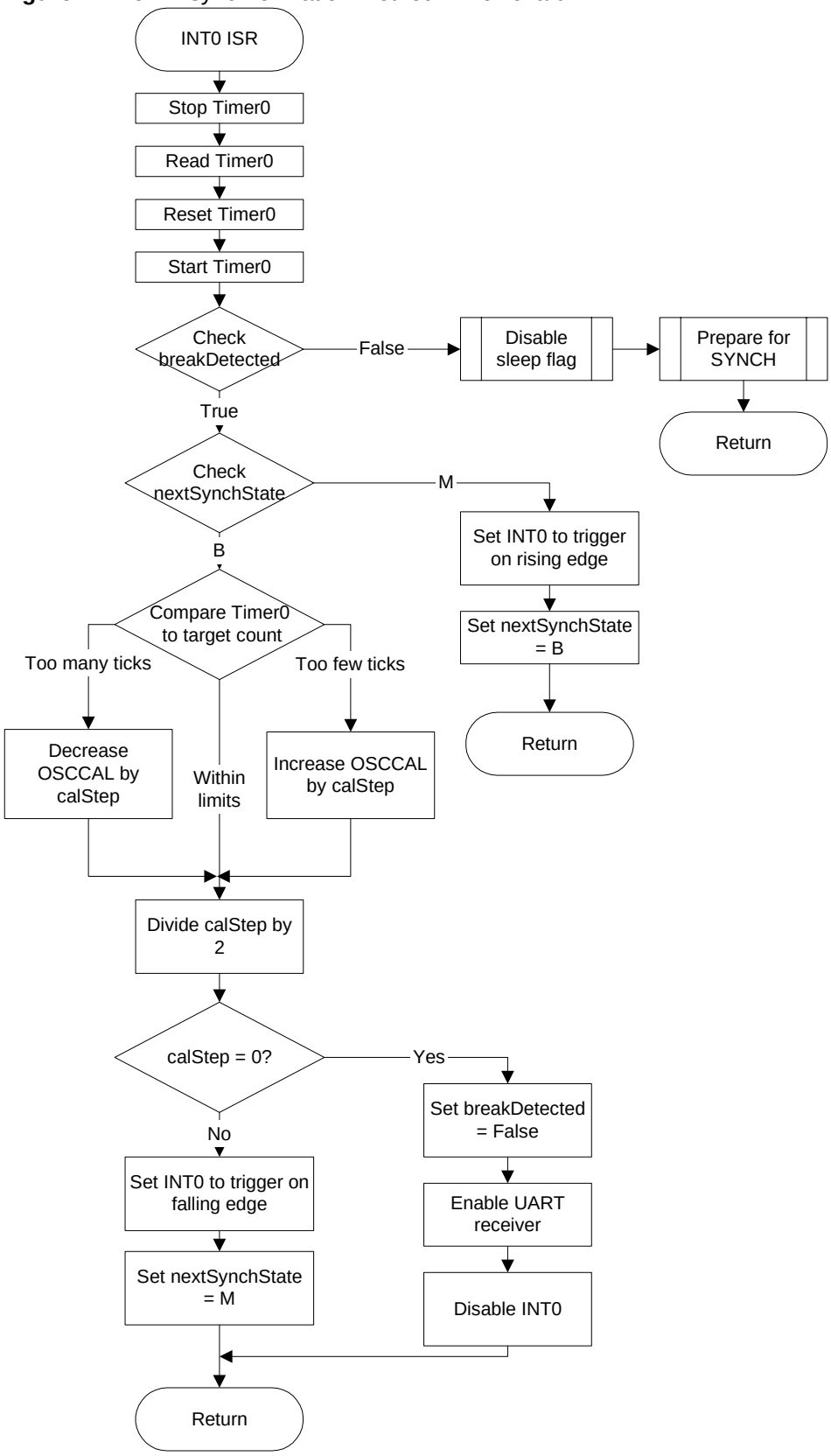
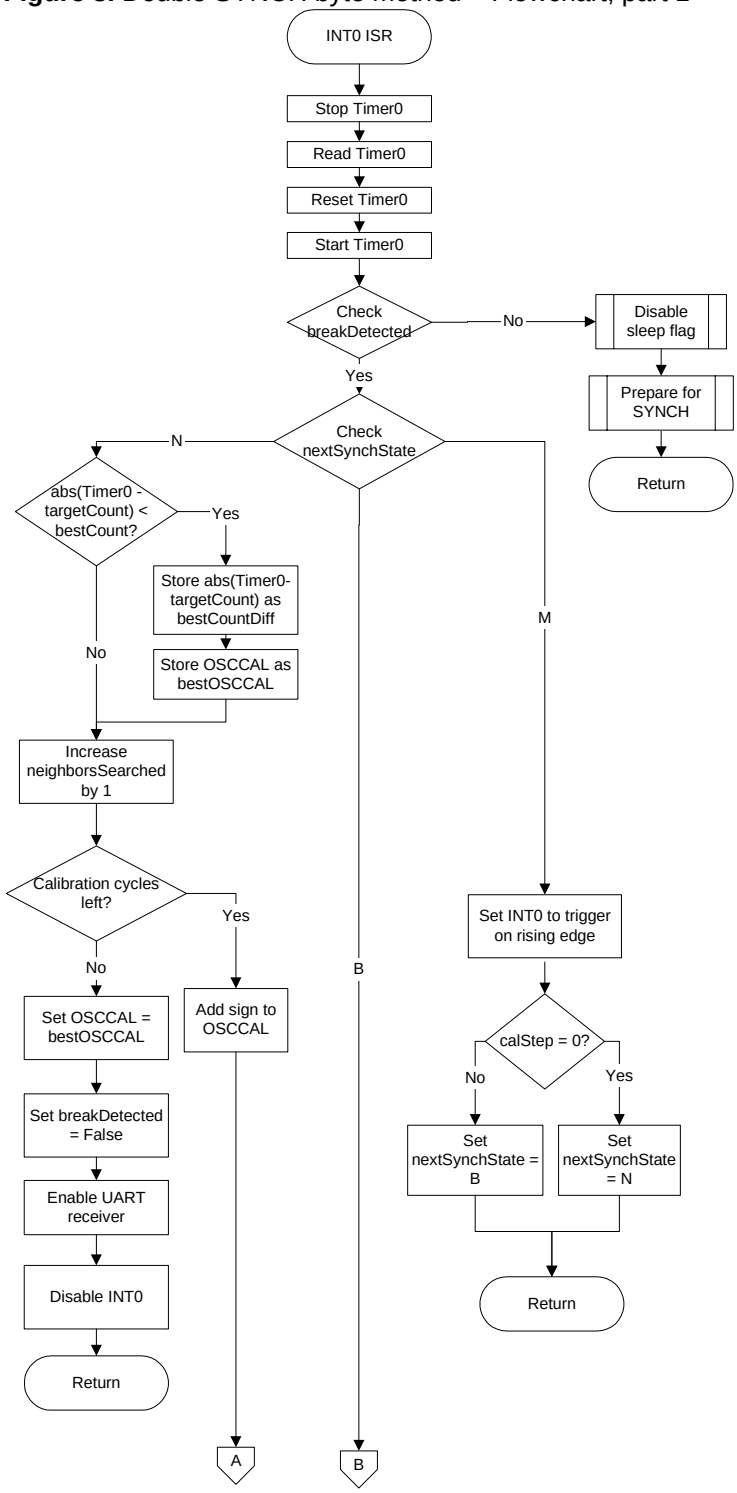
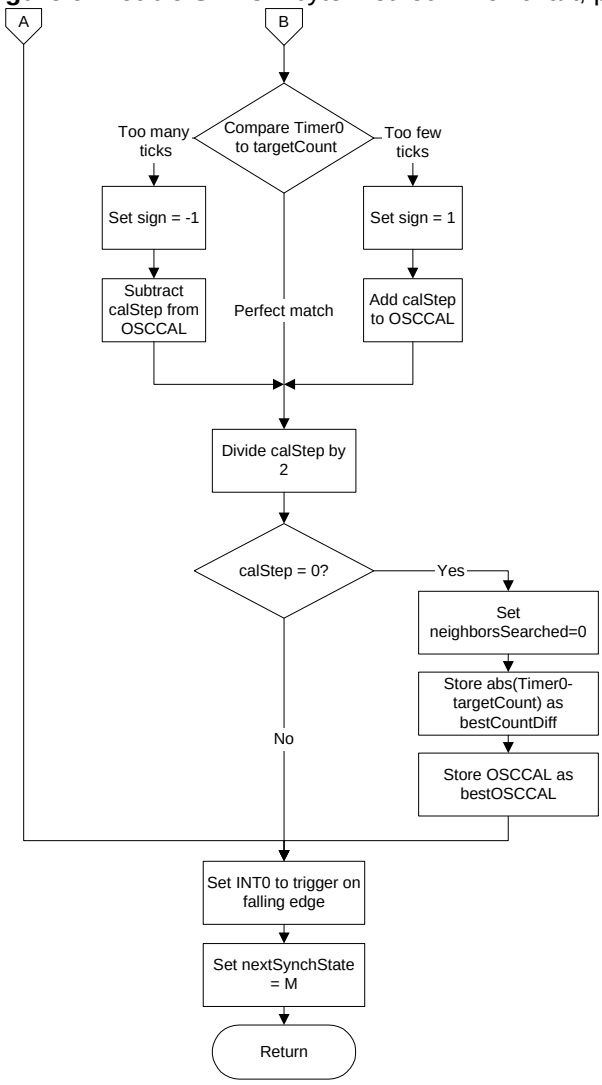


Figure 8. Double SYNCH byte method – Flowchart, part 1



(The flowchart continues on the next page.)

Figure 9. Double SYNCH byte method - Flowchart, part 2



Conditions for successful operation

The following section lists some important guidelines that should be followed to obtain the desired accuracy of the calibration.

Timing accuracy

Since Timer/counter0 only counts integer numbers, a truncation error is introduced. The inaccuracy introduced by this truncation is limited by the baud rate to clock frequency ratio. Decreasing this ratio will result in more accurate timing. To obtain a frequency accuracy of 1%, this ratio must be less than 0.01. This means that a baud rate of 19200kbps will require a clock frequency of minimum 2MHz.

OSCCAL overflow

No measures have been made to ensure that an overflow does not occur when the OSCCAL register is modified during calibration. This is done on purpose, since it is not necessary to do overflow checking at run-time. Instead it is recommended to test the default OSCCAL value at programming time, when it is stored to EEPROM or flash. If the search range is large enough to risk OSCCAL overflow, change the default OSCCAL value stored in EEPROM accordingly, so an overflow cannot occur. This applies only to the LIN synchronization method.

UART Baud rate generation

Using the UART module for communications, it is not always possible to match the desired baud rate at all clock frequencies. To eliminate this inaccuracy, make sure the desired baud rate can be exactly generated at the calibration target frequency.

For instance, when running at a clock frequency of exactly 2MHz, it is impossible to obtain a UART baud rate of 19.2kbps with less than 7% error. At a clock frequency of exactly 2.150.400Hz, a baud rate of 19.2kbps can be exactly generated. In this case, calibrating towards 2MHz would be useless if the intention is to communicate at 19.2kbps.

Timer/counter resolution

It is extremely important that the selected timer/counter resolution is sufficient to count all the clock cycles during the low cycle of the synchronization signal. The needed resolution is dictated by the max clock frequency/baud rate ratio. Make sure that the counter resolution is sufficient even when the clock frequency reaches the highest value during synchronization.

Version 5.0 oscillators

Since the OSCCAL register on version 5.0 oscillators is split in two parts, the binary search method is not suited to search the whole range. The included source code supports searching of only one half of the OSCCAL register. The active half must be decided at compile time. If the whole range needs to be searched, one search for each half of the register must be performed, and the results must be compared to choose the optimal OSCCAL value. This method is not implemented in the included source code, but could be done with small changes to the included source code.

Getting started

The following steps will get you started with synchronization on the AVR:

The included source code is written for IAR Embedded Workbench V3. Modifications are needed to compile the source code with other compilers.

Download the source code for AVR054 from www.atmel.com and unzip it. Two directories will be created. The “synchronization” directory contains the synchronization source code. The “The “test” directory contains source code that generates a master synchronization signal for testing.

Synchronization source code

Open the file “online_synch.eww” from the “synchronization” directory in IAR Embedded Workbench. This project is set up with the necessary files and settings to get started with synchronization.

Select device from “Processor configuration” in the “Project/Options” menu of IAR embedded workbench.

“online_synch.h” contains several flags that must be set before compiling. Complete the following steps to tailor the source code for your needs.

1. Select calibration method by uncommenting one of the SYNCH_METHOD_XXXXXX lines.
2. Change TARGET_FREQUENCY to the frequency you want to calibrate the AVR towards.
3. Change SYNCH_FREQUENCY to the frequency of the master SYNCH signal.
4. Change SYNCH_UBRR to the UART Baud Rate Register (UBRR) value needed for the UART to communicate at SYNCH_FREQUENCY.
5. If LIN synchronization is selected in step 1, change the accuracy in frequency needed after synchronization. 10 equals +/-1% of TARGET_FREQUENCY.

6. If LIN synchronization is selected, change DEFAULT_OSCCAL_ADDRESS to reflect the location in EEPROM of the default OSCCAL value to be loaded on every synchronization attempt. Also remember to write the default OSCCAL value to EEPROM when programming the chip.
7. Decide if a 9 bit timer is needed. If only 8 bits are needed, comment out the line defining NINE_BIT_TIMER.
8. If a device with two frequency ranges is used, it must be decided which one is used. (Refer to the data sheet of the device for more information.) Uncomment one of the lines defining DEFAULT_OSCCAL_MASK to select range.

The following files do not need to be changed to test the synchronization, but a brief description of each file is given below to help understand how an application can be integrated with the synchronization source code.

“lin_synch.c” and “double_synch_byte.c” contain the implementation of the LIN synchronization method, and the double synch byte method respectively. The UART RXC interrupt service routine (ISR) is a part of this implementation. To implement a communication protocol, code must be added in one of these files to handle reception of UART data.

“main.c” contains the main loop of the application and a function “sleep()”. The sleep function is never run in the included code, but could be used in a protocol implementation to save power while there is no activity on the bus. It takes care of all the necessary steps to make the device enter one of the sleep modes and still be prepared to receive a BREAK signal. By default it will enter “Idle” mode. To enter a different sleep mode change the SMn bits that are set in MCUCR.

Device specific differences are handled in device_specific.h. It should not be necessary to change the contents of this file, unless a different Timer/counter or UART module is desired or support for a new device is needed. For this file to function properly, a symbol describing the targeted device must be defined at compilation time. In IAR Embedded Workbench, this is automatically done by selecting “Processor configuration” in the “Project/Options” menu.

Test source code

“test.c” contains code that will generate a BREAK/SYNCH signal as described by this document. It can be run on a second AVR to act as a master device to test synchronization. Follow these steps to compile the test project:

1. Open the “test-node.eww” file from the “test” directory in IAR Embedded Workbench.
2. Select device from “Processor configuration” in the “project/options” menu of IAR embedded workbench.
3. Open the file “test.c” from the project browser in IAR embedded workbench.
4. In “test.c”, change UART_BAUD_RATE_REG to change the baud rate of the synchronization signal. See the data sheet for the selected device to find the correct UBRR setting at the current frequency.
5. In “test.c”, change the value of NUM_SYNCH_BYTES to generate different synchronization sequences. NUM_SYNCH_BYTES should be 1 for the LIN synchronization method and 2 for the double synchronization byte method.

The signal transmitted on the bus will be in the following form:

- BREAK signal
- The specified number of SYNCH bytes

- One byte of data, starting at 0, increasing by one for each transmission.

The last data byte can be used to confirm that the slave device is able to receive data correctly after synchronization. The signal is available on the TXD pin of the “master” device.

Putting it all together

To test the synchronization, program one AVR with the master test software. Program a different AVR with the synchronization software. Make sure that the fuses of the slave device are configured for internal RC operation at the target speed. Connect the TXD pin on the master to RXD and INT0 pins on the slave. The main function of the synchronization code sets up Timer1 to output a waveform on the OC1A pin equal to half the speed of the internal RC oscillator. The frequency of this signal can be measured to verify that the synchronization works.

References

- “The LIN Specification package, Revision 2.0”, © LIN Consortium, 2003, http://www.lin-subbus.org/

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

2563A-AVR-07/04