AVR055: Using a 32kHz XTAL for run-time calibration of the internal RC

8-bit AVR® Microcontrollers

Application Note

Features

- **Calibration using a 32 kHz external crystal**
- **Adjustable RC frequency with maximum +/-2% accuracy**
- **Tune RC oscillator at any operating voltage and temperature**
- **Tune RC oscillator to any frequency within specification**
- **Full source code written in C**
- **Support for all AVRs with tunable RC oscillator and asynchronous timer**
- **Selectable calibration clock frequency**

1 Introduction

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 to within +/-2% of the frequency specified in the datasheet for the device, for some devices even +/-1% accuracy is achievable.

The calibration performed in the Atmel factory is made at a fixed operating voltage and temperature. As the frequency of the internal RC oscillator is affected by both operating voltage and temperature, it is sometimes desirable to perform a secondary calibration. AVR devices with internal RC Oscillator and an OSCCAL Calibration Register can be calibrated very accurately by using an external signal with stable frequency such as a 32 kHz watch crystal. This secondary calibration can be useful if running the device at a temperature or supply voltage level different from the default values (25°C, typically 5V) matching the specific application environment, or even to tune the oscillator to a different frequency.

Using an external watch crystal for internal oscillator run-time calibration results in a cost efficient solution for applications that needs an accurate system clock over the entire temperature range, and independent of operating voltage.

This application note describes a fast and accurate way to calibrate the internal RC oscillator using an external 32.768 kHz crystal as input to an asynchronous Timer/Counter.

Rev. 8002D-AVR-07/08

2 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 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.

3 Clock selection

The AVR fuse settings control the system clock source being used. To use the internal oscillator, the corresponding fuse setting must be selected. An overview of the fuses is available in the datasheets. When calibrating using an external crystal, the system clock must run from the internal oscillator.

4 Crystal Connection

In most AVR devices, external capacitors need to be connected to the TOSC1/2 pins, when connecting an external crystal. This applies to all parts with 1.8V capabilities. In other parts, the crystal can be connected directly between the TOSC1/2 pins. Refer do the device datasheet for details on crystal connections.

5 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 I/O 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 calibrating using an external crystal, the device timer must have the possibility to operate asynchronously of the system clock.

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, which is read by the main program and copied into OSCCAL at run-time.

In addition to the oscillator tuning using the OSCCAL register, some devices feature a system clock prescaler. The prescaler register (CLKPR) can be used to scale the system clock with predefined twos complement factors. Also, this prescaler can be preset through the AVR fuses; programming the CKDIV8 fuse will set the CLKPR to divide the system clock by 8. This can be done to ensure that the device is operated below a maximum frequency specification. The CLKPR can be modified at run-time to change the frequency of the system clock internally.

The base frequency of an oscillator is defined as the unscaled oscillator frequency.

6 RC Oscillator overview

Different RC oscillators have been utilized in the AVR microcontrollers throughout the history. An overview of the RC oscillators and some example devices is seen in Table 6-1. The device list is sorted by oscillator type, which is also more or less equivalent to sorting them by release date. Only example devices with tunable oscillators, and the possibility to operate asynchronously from an external crystal are listed in the table. For a complete list of supported devices, refer to the “device_specific.h” header file of the source code.

Table 6-1. Oscillator frequencies and features of devices with internal RC oscillator(s). Grouped by oscillator version.

Oscillator version	Device	RC oscillator frequency [MHz]	CKDIV	CLKPR
2.0	ATmega163	1.0	-	-
3.0	ATmega16	1.0, 2.0, 4.0, and 8.0	-	-
3.1	ATmega128(1)	1.0, 2.0, 4.0, and 8.0	-	XDIV (2)
4.0	ATmega169	8.0	Yes	Yes
5.0	ATmega169P	8.0	Yes	Yes

Notes: 1. The ATmega103 Compatible Mode fuse must be unprogrammed. Programming this fuse prevents using extended I/O and hence tuning the internal oscillator.

2. The prescaler register is in these devices named XDIV.

Note the when calibrating the ATCAN90 device (oscillator version 5.0) using the STK501 top module, an external crystal other than the 32kHz crystal of the STK501 is required for calibration within the accuracy limits of +/-2%. Refer to the device datasheet for details on external crystal connection.

6.1 Version 1.x oscillators

This version is the earliest internal RC for AVR that can be calibrated, though asynchronous operation is not possible. Due to this, parts having this version of the internal oscillator cannot be calibrated using an external crystal and does not appear in Table 6-1.

6.2 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.

6.3 Version 3.x oscillators

This version was introduced along with the first devices produced in the 35.5k process.

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.

6.4 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 and system clock prescaler is present.

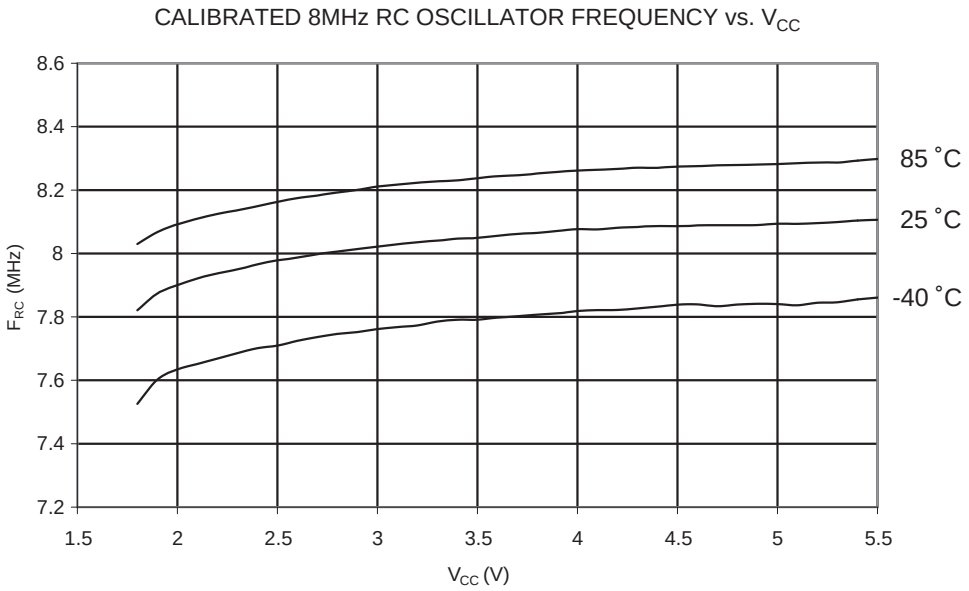
6.5 Version 5.x oscillators

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. As seen in Figure 7-2, 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.

7 Oscillator characteristics

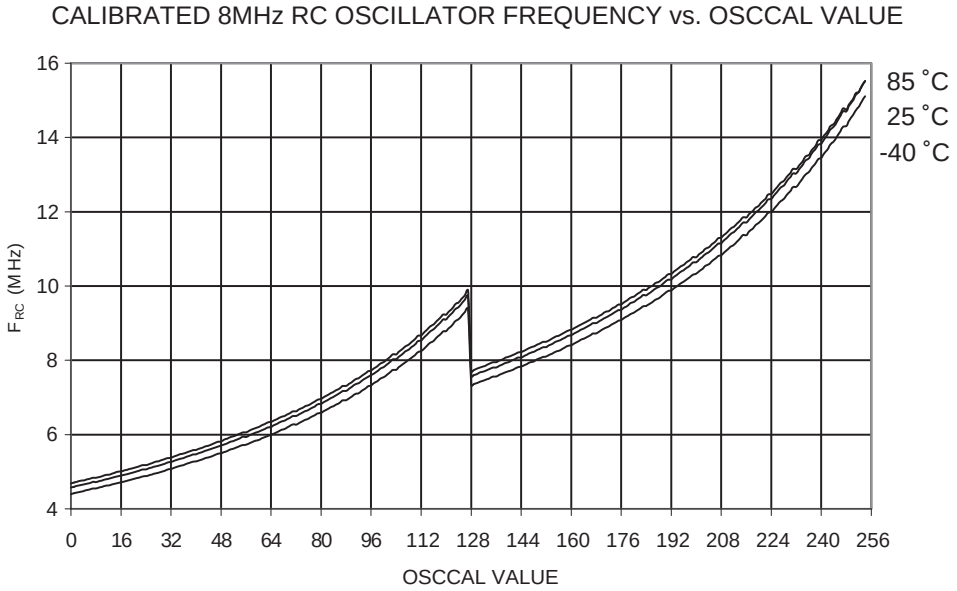
The frequency of the internal RC oscillator is depending on the temperature and operating voltage. An example of this dependency is seen in Figure 7-1, showing the frequency of the 8MHz RC oscillator of the ATmega3290. As seen from the figure, the frequency increases with increasing temperature and operating voltage. These characteristics will vary from device to device. For details on a specific device refer to its datasheet.

Figure 7-1. Oscillator frequency and influence by temperature and operating voltage. ATmega3290 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 even 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 7-2, which is the 8MHz RC oscillator of ATmega3290. Note that some OSCCAL registers, that is OSCCAL registers in devices with version 5.0 oscillators, uses 7 bits for tuning the oscillator, as is the case in ATmega3290. The eight bit determines the range of operation. The two ranges are overlapping, as seen in Figure 7-2. Other devices use all 8 (or sometimes 7) bits in one continuous range

Figure 7-2. ATmega3290 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 related to writing EEPROM and flash 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%.

7.1 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.

8 Calibration using a 32.768 kHz crystal

The calibration is executed in a 6 or 7 cycles loop, depending on whether the asynchronous timer/counter register is located in extended I/O space or not. In every loop, a counter is incremented, and the value of the 32 kHz timer is read. After a (predefined) number of ticks on the 32 kHz watch crystal, a counter which speed depends on the internal RC oscillator, is compared with the timer value. Then the value of the oscillator calibration register OSCCAL is written according to the desired calibration frequency, using binary and neighbor search.

The relation between the repeat count value and the frequency can be obtained from the following equation:

Equation 8-1. Calculating the count value

$$Loop\ Cycles \times Count\ value \times RC\ period = XTAL\ period \times XTAL\ ticks$$

where :

$$RC\ period = \frac{1}{desired\ \ frequency} \qquad XTAL\ period = \frac{1}{XTAL\ \ frequency}$$

9 Binary and Neighbor search

Three different calibration methods are described in this application note. They are different in the way the search is performed and have different code sizes and durations.

In section 7 it was pointed out that some internal oscillators have a divided OSCCAL register which is overlapping for some of the frequencies in the frequency range. For these oscillators, a search in both ranges of the OSCCAL is needed to guarantee calibration to all frequencies. Figure 7-2 illustrates that it may also be convenient to compare calibrations in both ranges of the OSCCAL for overlapping frequencies in order to obtain the optimal OSCCAL value since, for instance, a frequency of 8 MHz is reached for both OSCCAL values ~104 and ~144 in ATmega3290.

9.1 The binary search

A binary search is used to find the OSCCAL value that will make the internal RC oscillator produce the desired frequency. 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 just right, do not change OSCCAL and abort since no further calibration is needed.
5. Divide step-size by 2.
6. If step-size is 0, the search is complete. 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 mentioned in section 7, 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.

9.2 The neighbor search

In this search method, the neighbor values have to be examined. A binary search has to be done prior to using this method. 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, one more step should be taken. The Timer/Counter0 value of each of these iterations is then saved, and compared with 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. Since additional code is required to implement the neighbor search, this could be omitted when the need for accuracy is less important than the code size.

9.3 Simple search

This method is very easy to implement, and gives reduced code size compared to binary search and binary with neighbor search. It has, however, a higher worst-case run-time. The search works as follows:

1. Start with an OSCCAL value in the middle of the search-range i.e. ½ of the OSCCAL resolution.
2. If the frequency is to high, subtract OSCCAL with one. Add one if frequency is to low. Abort if oscillator frequency has reached desired frequency.
3. Repeat step n times, where n is OSCCAL resolution divided by two.

This last method is obviously slower than the first, but occupies less memory space due to its small code size. This method can also be used in fine-tuning, where step 1 is omitted and calibration is run from current OSCCAL. The repeat factor n can in this case be reduced considerably, sometimes just a few steps might be needed. A big drawback is that the simple search does not evaluate the goodness of the calibration, and hence will continue until it has done calibration cycles equal to ½ of the OSCCAL resolution unless perfect calibration is obtained.

9.4 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 will always be found if it lies inside the search range. The resulting clock frequency will be within +/-1% of the desired frequency for parts supporting this accuracy. The search is only aborted if a perfect match is found, i.e. the measured number of CPU cycles during a given amount of ticks on the external crystal exactly matches the desired number of CPU clock cycles. When the search is finished

without finding a perfect match, the calibrated OSCCAL value is the value that best matches the desired frequency. Even if there is an OSCCAL value that produces a clock frequency closer to the desired value, we will not be able to measure the difference. Further search is thus not necessary.

Accuracy can also be adjusted by increasing the amount of ticks on the external crystal, improving measurement accuracy. Few ticks will give fast calibration, but lower accuracy. Increasing the number up to a certain limit will increase accuracy. Notice that a higher amount of ticks does not necessarily increase the accuracy, since the resolution on the OSCCAL register is fixed. Improving the measurement accuracy does not improve the OSCCAL accuracy.

10 Implementation

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

A working implementation written in C is included with this application note. Full documentation of the source code and compilation information is found by opening the ‘readme.html’ file included with the source code.

10.1 Hardware

This application uses a 32.768 kHz watch crystal for oscillator calibration, as it is the optimal choice, both economically, and when it comes to frequency stability. The low-frequency crystal oscillator is designed for use with crystal running at this frequency.

It is important to notice that the external watch crystal needs time to stabilize. This applies to situations when the crystal is not already running prior to calibration.

10.2 Software

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

The main idea is to run a counter asynchronously on a separate 32KHz watch crystal for a certain number of cycles. Meanwhile a simple CPU loop is done, incrementing a word. Knowing the number of cycles of the inner loop, and assuming the asynchronous timer is correct; the CPU will count to a certain level. Then comparing to a number corresponding to the desired frequency, one can adjust the CPU frequency up/down through OSCCAL.

10.2.1 Defining the search range

Some devices have a divided OSCCAL register. As shown in Figure 7-2, the ATmega3290 OSCCAL register has two ranges, one going from values 0 to 127, the other from 128 to 255. When calibrating to a frequency between ~7 MHz and ~10 MHz, there are two possible OSCCAL values that gives an appropriate and satisfying frequency.

Before calibration can start, knowledge about the device OSCCAL register size is needed. A macro checks if the OSCCAL register is divided in two or not for the specific device. If the OSCCAL is divided, the search is first done in the lower part of OSCCAL followed by a search in the upper part as long as no perfect match is obtained during the first search. After a second search, the two calibrations are compared, and the optimal value found.

For devices with one 8 bit (or 7 bit in some devices) range containing all possible frequencies, only one search range is defined and a double search and comparison is unnecessary.

10.2.2 Initiating calibration

Before starting the calibration, OSCCAL should be written according to the value equal to ½ of the search range. For parts with a two-range OSCCAL, calibration starts in the lower OSCCAL, and OSCCAL is written to ½ of the lower search range. This is done to make the binary search work properly.

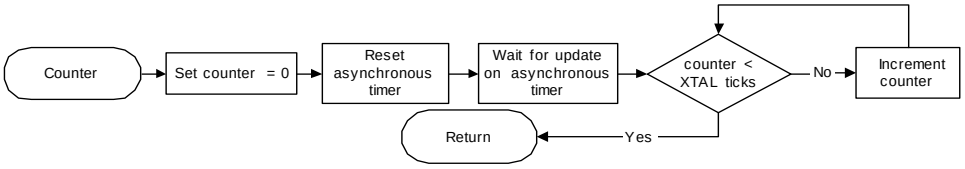
The asynchronous timer must be set up properly in order to use an external watch crystal as clock source. The Asynchronous (AS) bit in the ASSR register must be set to ensure that the timer is asynchronous from the CPU clock with at external crystal driving it. If interrupts are used, these must be disabled prior to using the counter function in order to ensure correct calibration.

10.2.3 Defining the count value

When calibrating to a desired frequency, the count value in Equation 8-1 is needed. This equation is implemented as a macro, using the device specific loop cycle value. The number of loop cycles is 6 for devices with the TCNT register located in standard I/O space, and 7 in devices with the TCNT register located in extended I/O.

Now having the count value, and using the counter function to increment the counter, the counter and the desired count value can be compared to proceed with the calibration.

Figure 10-1. The counter function

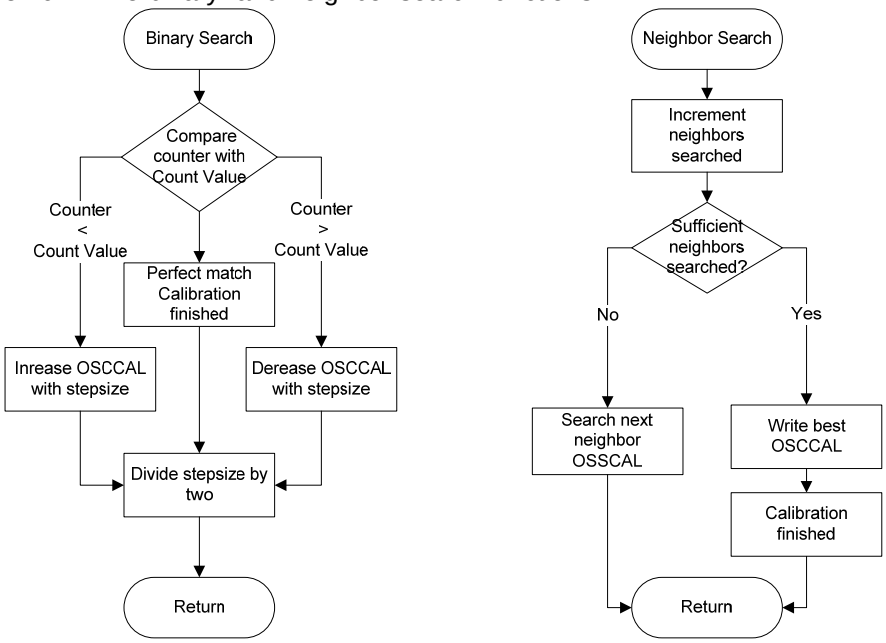


It is important that interrupt is disabled in order to ensure proper operation of the counter function and the asynchronous timer.

10.3 Searching

When a fast calibration procedure is needed, a binary search performs better than any other search method. When combining it with neighbor search, greater accuracy is achieved, since the relation between the OSCCAL value and the oscillator frequency is pseudo-monotone. There is no guarantee that the binary search will reach the optimal value when the relationship is not strictly monotone. The neighbor search can be omitted without a considerable loss of accuracy as mentioned in section 9.4, and also improve run-time and reduce code size. The neighbor search will be omitted whenever perfect calibration is reached.

Figure 10-2. The binary- and neighbor search functions

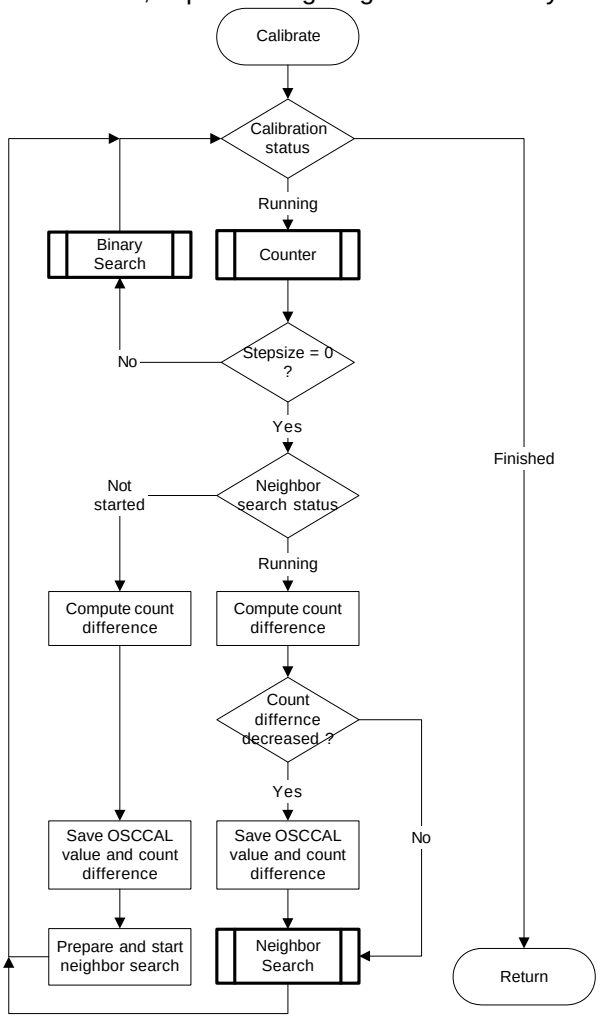


It can be noticed that when the OSCCAL value of the desired calibration frequency is close to the OSCCAL value of the running frequency, the simple search method may be a better choice. Leaving the OSCCAL register unchanged prior to calibrating, only a few steps might be necessary. This may be interesting in cases such as fine-tuning or re-calibrating as it reduces code size and can outperform binary search when it comes to time consumption.

10.3.1 Calibration

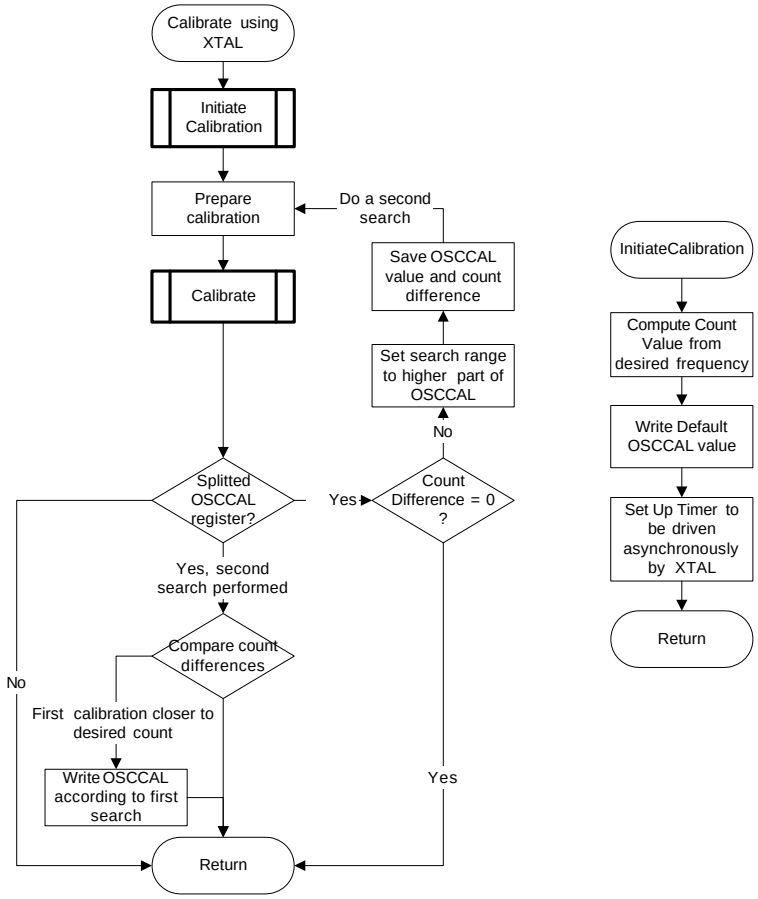
The calibration routine will do a binary search until the step size reaches zero, or finish if a prefect match should occur during binary searching. When the step-size has reached zero, neighbor search begins. The optimal OSCCAL is obtained by evaluating the difference between the count value corresponding to the desired frequency, and the value returned by the counter function. The lowest deviation and the corresponding OSCCAL value is saved and the OSCCAL register is written according to the lowest deviation when the search is finished, in order to obtain the optimal value.

Figure 10-3. The calibrate routine, implementing neighbor and binary search



In devices where the OSCCAL register contains two ranges, both of the ranges may hold a value that matches the desired frequency, but the value in one range may give a more accurate calibration than the other. The two calibrations are compared to guarantee that the best possible solution is found. When using devices with one OSCCAL range, double search is unnecessary.

Figure 10-4. The main routine



10.4 Code size

The source code included with this application note results in quite different code sizes when compiled, depending on calibration method, timer resolution, and type of OSCCAL register used. Table 10-1 shows some examples of code size for two devices, ATmega3290 with a divided OSCCAL register, and ATmega32 with a continuous OSCCAL register.

Calibration cycles are also given to illustrate the duration. A calibration cycle means running the counter with or without a write to OSCCAL, using binary, neighbor or simple search.

Table 10-1.Compiled code size and calibration cycles

Device	Calibration method	Calibration Cycles	Code size
ATmega3290	Simple search	max 128	~270 B
	Binary search	max 14	~370 B
	Binary with neighbor search	max 22	~380 B
ATmega32	Simple search	max 128	~240 B
	Binary search	max 8	~280 B
	Binary with neighbor search	max 12	~290 B

10.5 Performance of the Calibration firmware

The code has been written with focus on efficiency: The entire calibration should be performed fairly quickly. The performance therefore depends on the size of the calibration firmware and the time it takes to complete the calibration.

The calibration firmware is 240 to 380 bytes, depending on the target device and calibration method used, giving short programming time.

The calibration routine is completed in less than 22 (worst-case) calibration cycles for ATmega3290 and 12 for ATmega32, using binary with neighbor search. The duration is dependent on whether perfect calibration is reached during binary search or not.

10.6 Calibration Clock Accuracy

The accuracy of the calibration is highly dependent on the accuracy of the external calibration clock. It is therefore important to know the exact frequency of the crystal used and enter it into the interface specific source file. Using a 32.768 kHz watch crystal, optimal accuracy is achieved.

11 Getting started

The source code is compatible with all current AVR devices with a tunable internal oscillator and a timer with asynchronous operation mode. These devices are listed in the “device_specific.h” header file coming with the source code.

Download the source code for AVR055 from www.atmel.com and unzip it.

11.1 Calibration source code

All functions and the main routine are all collected in one file, “calib_32kHz.c”. Macros, calibration specific values and flags, and device specific definitions are in separate header files.

11.1.1 Source code files

The root file “calib_32kHz.c” refers to the following files:

1. A device specific file, “device_specific.h”. This file will ensure that the root file uses the proper bit and register definitions corresponding to the selected device.
2. A calibration interface specific file, “calib_values.h”. This file holds the desired calibration value along with other values relevant to the calibration. Some macros are also defined, as well as flags choosing different calibration methods.

11.1.2 Performing a calibration

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

1. If a search method other than binary with neighbor search is desired, uncomment one of the two CALIBRATION_METHOD_XXXXX lines.
2. Change the frequency to desired calibration frequency by modifying the CALIBRATION_FREQUENCY value, which is set to 1MHZ by default.

3. If crystals other than a 32.768 kHz watch crystal is used, change XTAL_FREQUENCY according to the crystal used for calibration.
4. EXTERNAL_TICKS can be modified to decrease/increase accuracy and calibration running time. 100 ticks are recommended to guarantee +/-1% accuracy, but this number can be reduced to speed up the calibration. The maximum value is 255 due to the size of the asynchronous timer register. For most devices 40 ticks is feasible, but a higher value gives increased accuracy.

Headquarters

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

International

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

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel 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

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature