AVR053: Calibration of the internal RC oscillator

8-bit AVR® Microcontrollers

Application Note

Features

- **Calibration using STK500, AVRISP, JTAGICE or JTAGICE mkII**
- **Calibration using 3rd party programmers**
- **Adjustable RC frequency with +/-1% accuracy**
- **Tune RC oscillator at any operating voltage and temperature**
- **Tune RC oscillator to any frequency within specification**
- **Support for all AVRs with tunable RC oscillator**
- **Selectable calibration clock frequency**

Introduction

This application note describes a fast and accurate method to calibrate the internal RC oscillator. It offers an easily adaptable calibration firmware source code, which can be used with any AVR with internal tunable RC oscillator. This firmware allows device calibration using the AVR tools STK500, AVRISP or JTAGICE, but can also be used for 3rd party calibration systems, e.g. based on production programmers.

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 +/-1% of the frequency specified in the datasheet for the device. This feature offers great flexibility and significant cost savings compared to using an external oscillator.

The calibration performed in the Atmel factory is made at a fixed operating voltage and temperature (25°C, typically 5V). As the frequency of the internal RC oscillator is affected by both operating voltage and temperature, it may be desired to perform a secondary calibration, which matches the specific application environment. This secondary calibration can be performed to gain higher accuracy than the standard calibration offers, to match a specific operating voltage or temperature, or even to tune the oscillator to a different frequency.

The calibration method described in this application note only takes a fraction of a second longer than reading the factory calibration byte from the signature row and writing it back to the to the device memory. Thus, the overall programming time is almost unaffected when performing calibration in the programming step in production.

Note that in some systems it may be more beneficial to perform run-time calibration of the oscillator. That may de desirable in applications that needs an accurate system clock over the entire temperature range and independent of operating voltage. In that case a watch crystal may offer a reliable and cost efficient solution. Runtime calibration is however not covered by the scope of this application note.

A Quick Start Guide is found last in this document.

Rev. 2555F-AVR-09/05

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 hence 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.

RC Oscillator overview

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	-	-
3.0	ATtiny26	1.0, 2.0, 4.0, and 8.0	-	-
4.0	ATmega162	8.0	Yes	Yes
4.0	ATmega169(2)	8.0	Yes	Yes
4.0	ATmega165	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, ATmega88, ATmega168	8.0	Yes	Yes
5.0	ATtiny25, ATtiny45, ATtiny85	8.0	Yes	Yes
5.0	ATmega325, ATmega3250, Atmega645, Atmega6450,	8.0	Yes	Yes
5.0	ATmega329, ATmega3290, Atmega649, Atmega6490,	8.0	Yes	Yes
5.0	AT90CAN128	8.0	Yes	Yes
5.0	AT90PWM2, AT90PWM3	8.0	Yes	Yes

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

2. ATmega169 revision F and onwards uses version 5.0 oscillator.

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

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.

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.

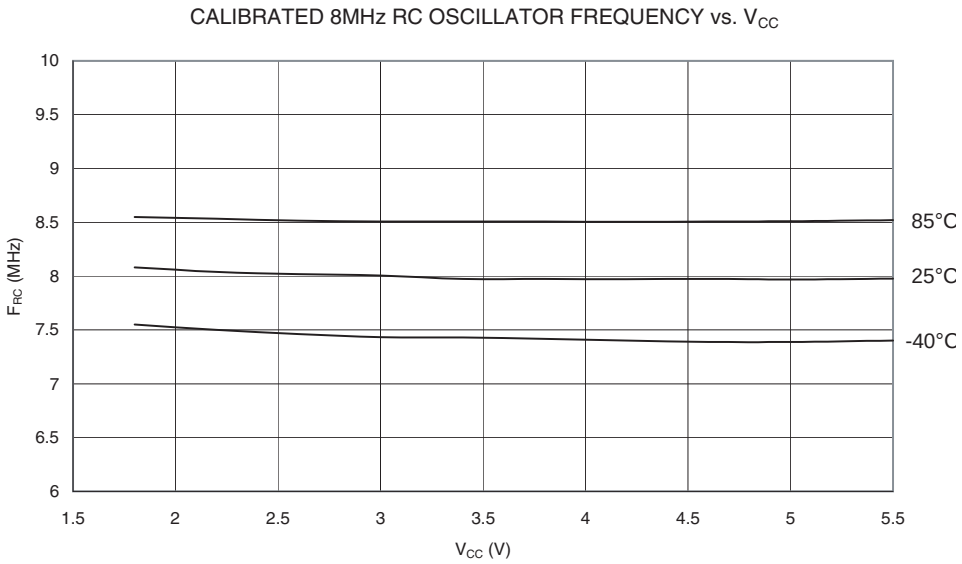
Version 5.x oscillators

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

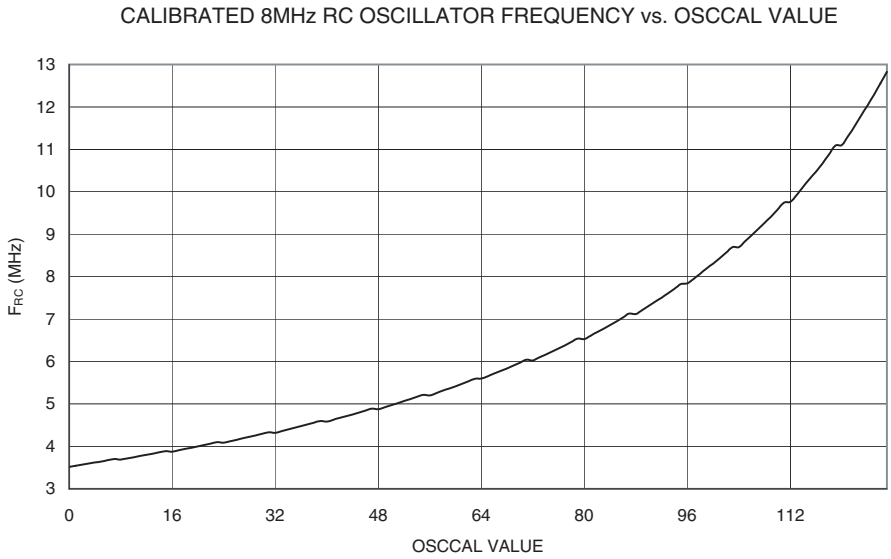
The frequency of the internal RC oscillator is depending on the temperature and operating voltage. An example of this dependency is seen in Figure 1, which shows the frequency of the 8MHz RC oscillator of the ATmega169 (revision A to E). As seen from the figure, the frequency increases with increasing 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 (8 bits in ATmega169 revision F and on) for tuning the oscillator in ATmega169, the maximum frequency is corresponding to OSCCAL = 128.

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[1]. 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%.

Implementation of the calibration

This section is divided into a description of the calibration protocol and a description of the firmware. The protocol can be adapted into any test or programming tool to support calibration. The AVR tools STK500, AVRISP, JTAGICE and JTAGICE mkII support the implemented calibration protocol. The usage of these tools to calibrate a device is described later.

The calibration support in the STK500, AVRISP, JTAGICE and JTAGICE mkII is at present only supported in the command-line version of the tools. The calibration is supported from AVR Studio version 4.11 SP1 (or later). The newest release of AVR Studio can be downloaded from http://www.atmel.com/avr/.

Calibration protocol

The protocol for calibration is kept simple and fast to ensure that it can be used in production environments. The pins used for programming the devices, that is the ISP interface or the JTAG interface (if present), are used for the calibration as they are most likely to be available in a final product (or on PCB).

Two pins are used for the calibration: MOSI and MISO on the ISP interface, or TDI and TDO on the JTAG interface. To simplify the description, only MOSI and MISO are referred to subsequently, though TDI and TDO can be used a well.

The basic concept is that the programmer generates the calibration clock (C-clock), and that the device uses this as a reference to calibrate its internal RC oscillator.

When the device has completed the calibration it signals “OK” to the programmer on the MISO line.

The programmer is responsible for enabling a pull-up on the MISO line and the device for enabling pull-up on the MOSI line. This is done to ensure that noise is unlikely to corrupt the calibration.

The programmer can use 1024 C-cycles (cycles on the C-clock) as time-out period, as the calibration routine is guaranteed to be completed within this number of C-cycles.

The calibration procedure runs through the following steps:

1. The programmer writes the calibration firmware into the device, enables the MISO pull-up, and releases the reset line. The calibration clock is applied on the MOSI line. A frequency close to the frequency of a watch crystal (32.768kHz) is appropriate.
2. The device enables the internal pull-up on the MOSI line and starts listening for the calibration clock on MOSI.
3. When the device detects the calibration clock a binary search is used to find an OSCCAL value that meets the criteria of 1% accuracy. If the binary search does not reveal a value that meets this requirement, the neighboring values to the outcome of the binary search are tested to identify one that does.
4. The calibration value is stored in EEPROM (In the case of failing calibration, this step is skipped).
5. When calibration is completed successfully the MISO line is toggled 4 times by the device. The toggling of the MISO line is performed 5 to 10 CPU cycles after falling edge of the clock on the MOSI line (C-clock). In the case of failing calibration the MISO line is not toggled.
6. If the device does not have an EESAVE fuse, the programmer must read back the calibration byte from EEPROM, for later restoring when the calibration firmware has been erased from the Flash. If the device have an EESAVE fuse, this fuse can be set so that erasing the Flash does not also erase the EEPROM.

It is necessary to copy the calibration byte from EEPROM to the OSCCAL register at run-time. A routine for this must therefore be implemented in the final firmware.

The calibration firmware

The calibration code is written in assembly, for the AVR Studio 4.11 assembler with the calibration package installed.

The calibration firmware is structured in a way so that it can easily be changed to match any of the devices listed in Table 1. Also, the interface for calibration can be changed. All required changes are made in the root file “RC_Calibration.asm” when calibrating using the AVR Tools.

The root file refers to (includes) the following files:

1. A device specific file (select the one matching the target device), e.g. “m16.asm” for Atmega16. The device-specific file further includes the following:
    a. The register and bit definition distributed with AVR Studio.
    b. A memory map file that defines where the code is located and which EEPROM location to store the calibration byte in.
    c. An OSCCAL access macro file that controls how the OSCCAL register is accessed. The way of accessing the OSCCAL register depends on where in the IO file the OSCCAL register is located.

d. An oscillator version file. This file defines the initial step-size used in the binary search to account for the fact that some OSCCAL registers are 7 and some are 8 bits wide.

e. A Return Stack initialization macro file. Some devices have hardware stack, while others have a stack in SRAM that needs initialization.

f. A port access macro file, which defines how to access the registers related to the pins used in the calibration. This is needed since some registers are in the high part of the IO file and others are in the low part of the IO file.

g. Redefinitions of bit and register names may also be present in the device file.

Please notice that the device specific file has to be modified when using ATmega169 revision F. The oscillator version should be set to 5 in “m169.asm”.

2. A calibration interface specific file. This file assigns the ISP or JTAG port and pins with names (labels) used in the main code. The calibration clock frequency is specified in this file.

3. The file defining the macros used - “macros.inc”

4. The common calibration code “main.asm”

The structure of the calibration code is designed to make it easy to change, in order to match a desired target device and interface. Furthermore, the extensive use of macros ensures that the code gets the smallest possible footprint. Finally, the way devices and calibration interfaces are designed ensures that support for new devices or interfaces can be implemented with a minimum of effort.

Binary search method

The search is based on a binary search method, a divide-and-conquer method:

1. The OSCCAL register is loaded with the initial value, which is half the maximum value of OSCCAL. The initial value of OSCCAL is defined as the initial Step-Size.

2. The frequency of the system clock is then compared to an external reference, the calibration clock.

   a. If the frequency is within 1% accuracy limit, goto 5.

   b. If the system clock is found to be too fast the OSCCAL value is reduced, and if the clock is too slow OSCCAL is increased. Goto 3.

3. Step-Size is assigned the value of half the previous Step-Size.

   a. If the Step-Size is zero, the binary search has not been successful, goto 4.

   b. If Step-Size is different from zero, the Step-Size is added to or subtracted from the current value in the OSCCAL register to increase or decrease the oscillator frequency. Repeat step 2.

4. Test the 4 nearest neighbor-values of OSCCAL. This is done to compensate for the lack of a strictly monotonous relationship between OSCCAL and oscillator frequency.

   a. If a tested OSCCAL value is within the accuracy limits, goto 5

   b. If none of the tested OSCCAL values are within the limits (not expected), signal on MISO that the calibration has failed by driving the line low.

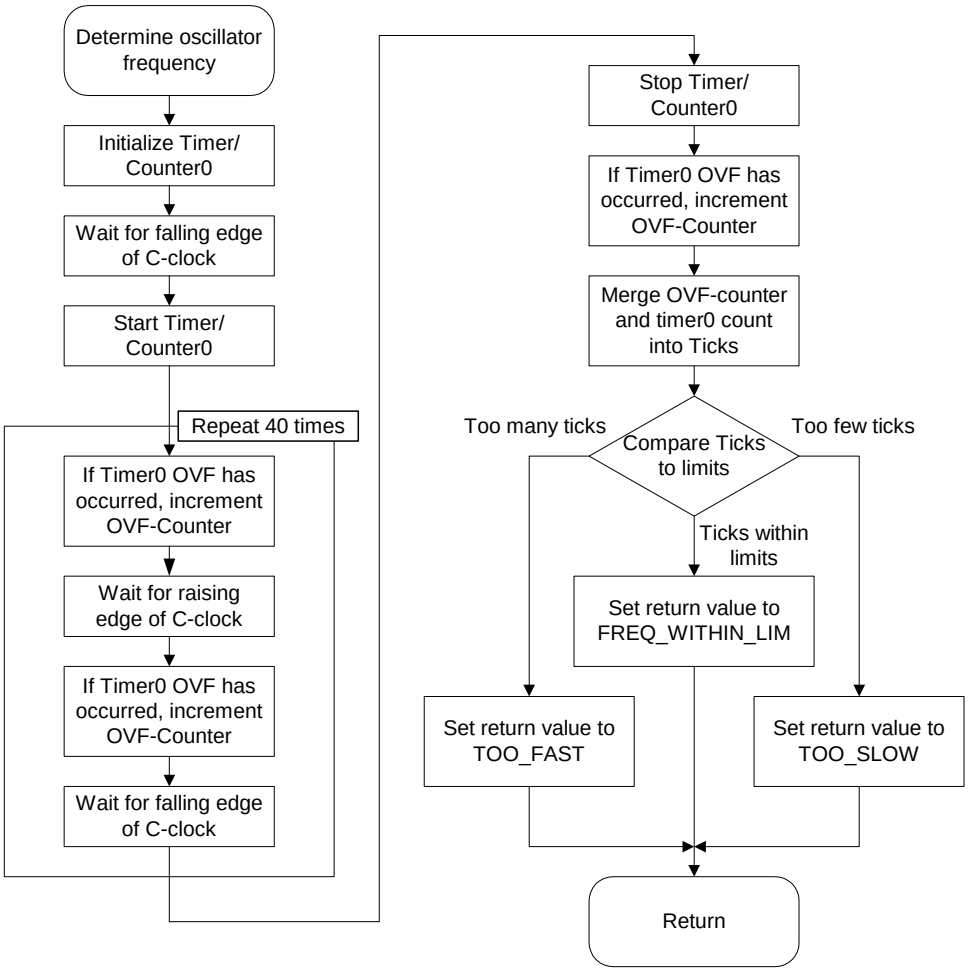
5. Store the calibration value in the EEPROM

6. Signal that calibration has been completed successfully by toggling the MISO line 4 times, synchronously to the calibration clock toggling.

Method for determining the oscillator frequency

The comparison between the Calibration clock (C-clock) and the internal RC oscillator is performed using the 8-bit Timer/Counter0 (TC0). The 8-bit timer is used since it is present in all devices that have tunable RC oscillator. The idea is to time the duration of 40 C-clock cycles and compare the number of timer ticks to predefined limits. The C-frequency in the present implementation is given in the interface specific include file. The method for determining the oscillator frequency is described in the flowchart in Figure 3.

Figure 3. Flowchart of algorithm determining relationship between the C-clock and the internal oscillator frequency.



To be able to cover the full range of oscillator frequencies, from 1MHz to 9.6MHz, inspection of the TC0 overflow (OVF) flag is used to expand the timer by 8 bits, providing a 16-bit timer. The OVF flag is inspected once every half-cycle (of the C-clock), which is sufficiently often to ensure that all TC0 OVF are detected. In relation to the range of the 16-bit timer implemented, the worst-case for overflow is at 9.6MHz where the OSCCAL register is loaded with 0xFF. In this case, the oscillator can be 100% above the specified frequency. The timer will in this case count to 23,541, which is within the range of the 16-bit timer.

Going in the other direction, the lowest oscillator frequency must also be considered. The lowest obtainable frequency is when writing 0x00 to OSCCAL. In that case the frequency may be 50% lower than the specified one. Since the TC0 OVF flag is inspected every half-cycle, there is potentially no more than just above 7 CPU-cycles to handle the OVF flag and detect the next C-clock edge - at a specified frequency of 1MHz. This timing constraint can be met when the OVF flag is not set, but when the flag is set 8 cycles are required. This will cause a small error in the detection of the timing, but will not affect the overall outcome: the oscillator will correctly be determined as too slow.

These extremes are however very unlikely to be encountered due to the binary search method used. However, they may be relevant to consider if the calibration method is modified.

Correcting timing inaccuracies

Since it is not possible to use interrupt driven detection for the C-clock edges for all devices, a polling method is implemented. The consequence of this implementation is that the edge detection can be delayed by up to 2 CPU cycles. Potentially this can make the calibration fail to reach the desired accuracy of 1%. To compensate for this potential timing error, the limits are tightened by 2 timer-ticks (2 CPU-cycles).

All calculations of limits and constants are performed by the preprocessor, which uses 32 bit accuracy in AVRASM and 64-bit in AVRASM2. All values that cannot be represented (floats) are rounded towards a tighter accuracy and will therefore not endanger the goal of +/-1% accuracy for the oscillator.

The calibration firmware does not take into account inaccuracies in the calibration clock source. Refer to the “Calibration Clock Accuracy” section of this document for details on how to minimize the effect of this.

Using STK500, AVRISP, JTAGICE or JTAGICE mkII for calibration

The source code of the calibration firmware and the batch file provided is made as an example of how to use the STK500, AVRISP, JTAGICE or the JTAGICE mkII to perform calibration. The firmware needs few or no modifications to be used in other calibration systems.

Assembling the calibration firmware

The root file for the calibration firmware is the RC_Calibration.asm file. This file is added to an assembly project in AVR Studio 4.11 SP1 (or later). In this file it is possible to include the target device and specify the desired calibration interface: STK500, AVRISP, JTAGICE or JTAGICE mkII. Further, it is possible to specify the desired calibration accuracy, and not least the desired frequency of the target device.

Once these choices have been made, build the project to produce the binary file “rc_calib.hex”. This file is used to calibrate the device.

Note that it is important to ensure that the fuses are set up correctly before calibrating the device: it is not possible to calibrate a device to 8.0MHz if the 1MHz RC oscillator is selected by the fuse settings.

Using the command line tools

The calibration support in the STK500, AVRISP, JTAGICE and JTAGICE mkII is at present only supported in the command-line version of the tools (AVR Studio 4.11 SP1 or later). The software package that provides this support can be found at http://www.atmel.com/avr/. Please install this package for calibration support.

The package includes a new firmware for the AVR tools, which is required to enable calibration. The firmware upgrade is automatic when first connecting to the tool with AVR Studio 4.11 SP1 (or later) or manual as described in the AVR Studio help.

Three batch files are provided along with the source code. These batch files show how the command line tools can be used to program the calibration code into the

target device, perform the calibration and hence reprogram the device with the final firmware. The three batch files are performing calibration of the ATmega16 through the STK500 or ISP, JTAGICE and the JTAGICE mkII, respectively. Please study these batch files and the AVR Studio integrated help to understand the use of the STK500/ISP, JTAGICE and JTAGICE mkII command line tools. Table 2 includes a list of the new commands to the exe files that are related to the calibration operation.

Table 2. New oscillator calibration specific options in stk500.exe and jtagice.exe.

Command	Description
-Z [addr]	Read calibration byte from EEPROM memory. 'addr' is byte address. The read operation is performed before the "chip erase" is executed. Using '-S#' will re-write the value to flash or EEPROM after the chip erase.
-Y	Perform the oscillator calibration sequence. This command will override all other operations. The exe file will return an errorlevel 1 if it does not get the acknowledge signal from the target device.

Adding support for new devices

To add support for a new device, all that is needed is to copy the device file for a similar device (pin compatible if possible) and adapt it to the new device's characteristics. The checklist below can be used when adapting a file to a new device. The checklist uses the ATmega8535 as example.

1. Copy the device file for a pin and feature compatible device.
    a. The ATmega8535 is pin compatible with ATmega16, though the ATmega8535 has no JTAG interface. The file "m16.asm" is therefore copied and named "m8535.asm"
2. Change the register and bit definition file included to match the new device
    a. For the ATmega8535 the register and bit definition file is "m8535.inc"
3. Change the pin-out description file to match the pin-out of the device.
    a. Since the ATmega8535 does not have JTAG interface as the ATmega16, the pin-out file is changed to the "s8535_family_pinout.inc" file.
4. Change the oscillator version file to match the oscillator of the new device.
5. Add the new file to the device list in the RC_Calibration file.
6. Verify that it assembles correctly. If it does not, this is most likely due to changed register or bit names of ports, pins, or timers. ATtiny13 (t13.asm) is implemented as a reassignment of ATtiny12, and can be used as a reference to reassigning names.

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 183 to 240 bytes, depending on the target device and the interface used for calibration. The required time to program the firmware is therefore short.

The calibration routine is completed in less than 1024 calibration cycles. The shortest duration is however dependent on how fast the binary search algorithm can find a suitable OSCCAL value, and the write time of the EEPROM. In the present implementation, using STK500.exe or JTAGICE.exe, the calibration itself is completed in less than 32ms.

Calibration Clock Accuracy

The accuracy of the calibration is highly dependent on the accuracy of the external calibration clock. The calibration clock frequency generated by the AVR tools may vary. It is therefore important to measure the exact frequency of the tool used and enter it into the interface specific source file. Since resonators are dependent on both operating voltage and temperature, the calibration frequency should be measured when these parameters equals the conditions during calibration.

Quick Start Guide to Calibration of the internal RC

To get started using the calibration feature in one of the device already supported one can follow steps below.

1. Download and unzip the source code for AVR053 (any location can be used, here called \AVR053\).
2. Download and install AVR Studio 4.11 SP1 from http://www.atmel.com/avr/
3. Open AVR Studio, make a new project called “rc_calib”, and add the root source code file, RC_Calibration.asm, to the project.
4. Select a target device from the list in RC_Calibration.asm, by removing and adding the semi-colon (";") in front of the device lines.
5. Select the interface, which is going to be used for the calibration in the same way as for the device selection.
6. Measure the frequency of the calibration clock with a frequency counter or an oscilloscope. This signal can be found on the MOSI pin on STK500/AVRISP and the TDI pin on JTAG ICE. Change the line in the interface specific file “.EQU CALIB_CLOCK_FREQ = XXXX” to reflect the measured frequency.
7. Specify the desired target frequency and the desired accuracy. Note that if the accuracy is too tight it may not be possible to calibrate the device and the calibration will fail. Refer to the data sheet for obtainable accuracy.
8. Assemble the project to generate the hex binary file that should be programmed into the device.
9. If the STK500/AVRISP is going to be used for the calibration:
    a. Open the file “\AVR053\AVR Asm\Batch file\ISP_rc_calib.bat” in an editor. (STK500.exe -h for info on arguments).
    b. Edit the file to match the desired device, by changing the -datmega16 argument to -d[target device].
    c. Change the fuse setting to the desired setting. Make sure that the settings correspond with the desired calibration: select 8MHz internal RC if calibrating the device to 8MHz. The fuse setting is specified through the arguments -E (extended fuses) and -f (high/low fuses). Make sure that the Watchdog Timer always on fuse is not set.
    d. If the install path for AVR Studio differs from the one used in the batch file (the standard in English Windows versions), please changes the path to the stk500.exe file.
    e. Save the file.
10. If the JTAGICE is going to be used for the calibration: Please note that the reset line must be available for the JTAGICE.

a. Open the file \AVR053\AVR Asm\Batch file\JTAGICE_rc_calib.bat in an editor. (jtagice.exe -h for info on arguments).

b. Edit the file to match the desired device, by changing the -datmega16 argument to -d[target device].

c. Change the fuse setting to the desired setting. Make sure that the setting corresponds with the desired calibration: select 8MHz internal RC if calibrating the device to 8MHz. The fuse setting is specified through the arguments -E (extended fuses) and -f (high/low fuses). Make sure that the Watchdog Timer always on fuse is not set.

d. If the install path for AVR Studio differs from the one used in the batch file (the standard in English Windows versions), please changes the path to the jtagice.exe file.

e. Save the file.

11. If the JTAGICE mkII is going to be used for the calibration: Please note that the reset line must be available for the JTAGICE mkII.

    a. Open the file \AVR053\AVR Asm\Batch file\JTAGICE_mkII_rc_calib.bat in an editor. (jtagiceii.exe -h for info on arguments).

    b. Edit the file to match the desired device, by changing the -d ATmega16 argument to –d [target device].

    c. Change the fuse setting to the desired setting. Make sure that the setting corresponds with the desired calibration: select 8MHz internal RC if calibrating the device to 8MHz. The fuse setting is specified through the arguments -E (extended fuses) and -f (high/low fuses). Make sure that the Watchdog Timer always on fuse is not set.

    d. If the install path for AVR Studio differs from the one used in the batch file (the standard in English Windows versions), please changes the path to the jtagiceii.exe file.

    e. Save the file.

12. Connect the STK500, AVRISP, JTAGICE or the JTAGICE mkII to the target board. Power the tool and application. Make sure that the serial cable is attached between the tool and the PC.

13. Open a command shell window (a DOS prompt). Navigate to the directory “\AVR053\AVR Asm\Batch file\”. Execute the batch file (ISP_rc_calib.bat, JTAGICE_rc_calib.bat or JTAGICE_mkII_rc_calib.bat).

14. Wait a short while for the calibration to complete.

The batch file can also be modified to program a custom firmware rather than the test.hex firmware after the calibration. Be aware that the new calibration value should be loaded into the OSCCAL register at runtime by the firmware.

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

2555F-AVR-09/05