AVR 8-bit Microcontrollers

AVR053: Internal RC Oscillator Calibration for tinyAVR and megaAVR Devices

APPLICATION NOTE

Introduction

This application note describes a fast and accurate method to calibrate the internal RC oscillator of the Atmel® tinyAVR® and megaAVR® devices that have ISP or JTAG interfaces. It offers firmware source code that allows calibration using the AVRISP mkII, JTAGICE mkII, JTAGICE3, or Atmel-ICE programming tools. It could also be adapted to production programmers. These are covered in the "AT06015: Production Programming of Atmel Microcontrollers" application note.

Many tinyAVR and megaAVR devices offer the possibility to run from an internal RC oscillator. Typically, this oscillator can be user calibrated to within ±1% of the frequency specified in the datasheet. This feature offers significant cost savings compared to using an external oscillator.

Factory calibration is performed at a fixed operating voltage and temperature. The calibration technique in this application note can be performed by the user to achieve 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.

In some systems it may be necessary to perform run-time calibration of the oscillator using an external crystal. This is covered in the "AVR055: Using a 32kHz XTAL for run-time calibration of the internal RC" application note.

Features

- Calibration using the following programming tools: AVRISP mkII, JTAGICE mkII, JTAGICE3, and Atmel-ICE
- Support for tinyAVR and megaAVR devices with tunable RC oscillator and ISP, or JTAG interface
- Adjustable internal RC oscillator frequency with ±1% accuracy typical
- Tune RC oscillator to any frequency at any operating voltage and temperature within specification
- No external components required for calibration

Table of Contents

1. Internal RC Oscillator

Most tinyAVR and megaAVR devices include a factory-calibrated internal RC oscillator. This is usually the default clock source for the CPU and does not require any external components.

1.1. User Calibration of the Internal RC Oscillator

In production the internal RC oscillator is calibrated at a specific supply voltage, typically 5V or 3.3V. Refer to the datasheet of the individual device for information about the operating voltage used during calibration. The accuracy of the factory calibration is typically within ±3% or ±10%, depending on the device (refer to the datasheet). If a design needs accuracy beyond what can be offered by the standard factory calibration, it is possible to perform a secondary calibration of the RC oscillator. By doing this it is possible to obtain a frequency accuracy typically within ±1% (±2% for devices that have a ±10% accuracy from factory calibration). It is also possible to shift the frequency of the oscillator as part of the user calibration procedure.

1.2. Base Frequency

The base frequency of an oscillator is defined as the oscillator frequency prior to any prescaling. Some AVR® devices feature a system clock prescaler. The prescaler register (CLKPR) can be used to scale the system clock with predefined factors. Also, the prescaler can typically be preset through the AVR device fuses. For example, programming the CKDIV8 fuse on devices that offer this option will set up the CLKPR to divide the system clock by eight. This can be done to make sure 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.

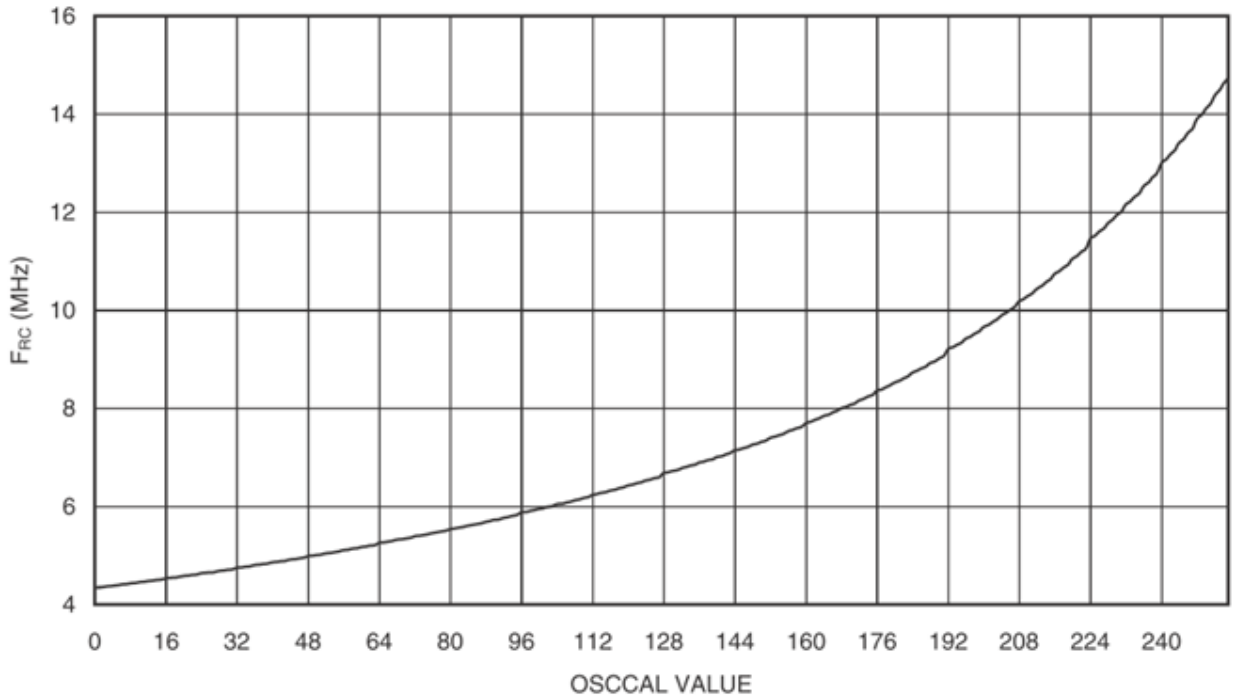
1.3. Clock Selection

The AVR device fuse settings control the system clock source being used. To calibrate the internal RC oscillator, it is required to use the internal RC oscillator as the CPU clock source by setting the appropriate fuse. An overview of the fuses is available in the datasheet of the specific device.

1.4. Tunable RC Oscillator

An RC oscillator whose frequency can be trimmed by software is called a tunable RC oscillator. In AVR devices, the Oscillator Calibration Register (OSCCAL) is used for this purpose. It can be used to trim the calibrated internal RC oscillator to remove process variations from the oscillator frequency, as shown in the figure below. The OSCCAL register is one byte wide.

Figure 1-1. Calibrated 8MHz RC Oscillator Frequency vs. OSCCAL Value (ATmega8)



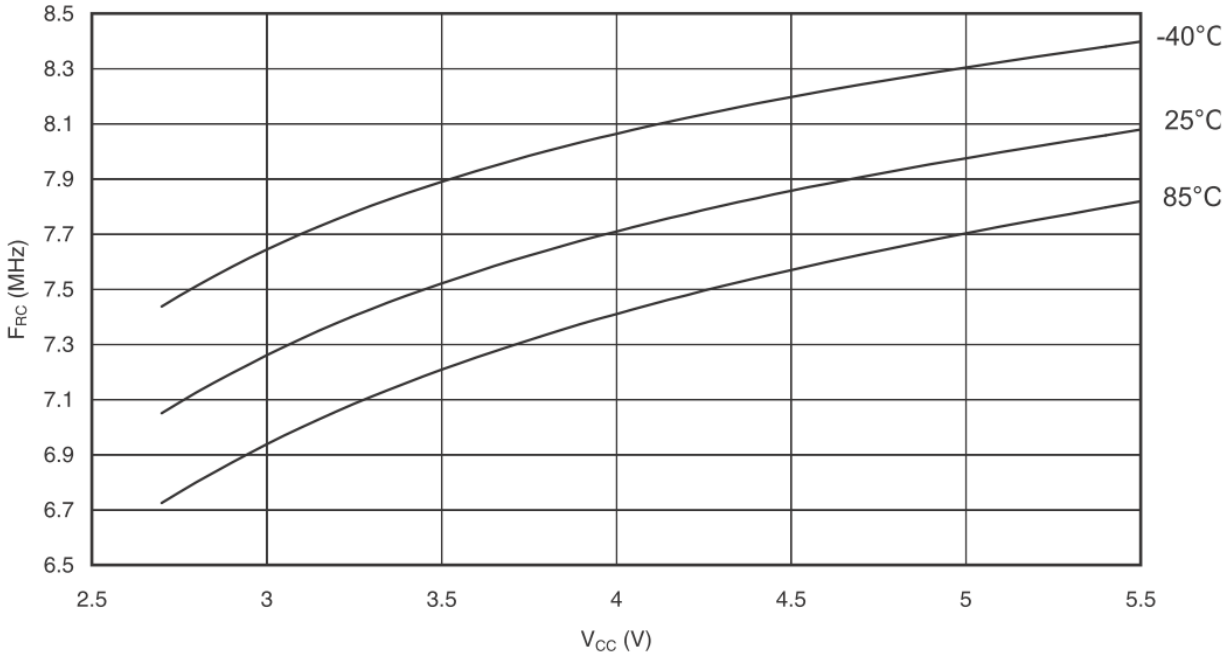
When an RC oscillator in a device is factory calibrated, the calibration byte is stored in the Signature Row of the device. The calibration byte can vary from one device to the other, as the RC oscillator frequency is process dependent. If a device has more than one oscillator, a calibration byte for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

In most devices, the default RC oscillator calibration byte is 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.

1.5. Oscillator Characteristics

The frequency of the internal RC oscillator depends on the temperature and operating voltage. An example of this dependency is in the figure below, which shows the frequency of the 8MHz RC oscillator of the ATmega8. The frequency increases with increasing voltage, and decreases with increasing operating temperature. These characteristics will vary from device to device. For details on a specific device, refer to its datasheet.

Figure 1-2. Calibrated 8MHz RC Oscillator Frequency vs. VCC (ATmega8)



1.6. RC Oscillator Revision History

Different RC oscillators have been utilized in the AVR microcontroller throughout its history. An overview of the RC oscillators with device examples is provided in the table below. The list is sorted by oscillator type, which is also roughly equivalent to sorting by release date. Only tunable oscillators are listed in the table. For oscillator details on a specific device, refer to the device datasheet.

Table 1-1. Oscillator Versions

Oscillator version	Example device	RC oscillator frequency [MHz]	CKDIV	PRSCK
1.1	ATtiny12	1.2	-	-
1.2	ATtiny15	1.6	-	-
2.0	ATmega163	1.0	-	-
3.0	ATmega16	1.0, 2.0, 4.0, and 8.0	-	-
3.1	ATmega128	1.0, 2.0, 4.0, and 8.0	XDIV (1)	-
4.0	ATmega169 (2)	8.0	Yes	Yes
4.1	ATtiny13	4.8 and 9.6	Yes	Yes
4.2	ATtiny2313	4.8 and 9.6	Yes	Yes
5.0	ATmega169P (2)	8.0	Yes	Yes

1. The prescaler register is in these devices is named XDIV.
2. ATmega169 revision A-E uses oscillator version 4.0, while ATmega169P uses oscillator version 5.0.

1.6.1. Version 1.x Oscillators

This version is the earliest internal RC oscillator for AVR that can be calibrated. It is offered with frequencies ranging from 1.2 to 1.6MHz. The calibration byte is stored in the Signature Row, but is not 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.

1.6.2. Version 2.x Oscillators

This oscillator version has a nominal frequency of 1MHz. The dependency of oscillator frequency on operating voltage and temperature is reduced significantly compared to version 1.x.

1.6.3. Version 3.x Oscillators

This version of the oscillator system offers multiple oscillator frequencies. Four different RC oscillators with nominal 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. Because four different RC oscillators are present, four 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.

1.6.4. 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 seven 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.

1.6.5. Version 5.x Oscillators

A single oscillator frequency of 8MHz is offered in version 5.0. Auto loading of the default calibration value and system clock prescaler is present. All eight bits in the OSCCAL register are used to tune the oscillator frequency. The OSCCAL register is split in two parts. The MSB of OSCCAL selects one of two overlapping frequency ranges, while the seven least significant bits are used to tune the frequency within this range. The ATmega406 has a frequency of 4MHz but is otherwise the same.

2. Calibration

This chapter is divided into calibration protocol and calibration firmware. The protocol can be adapted into any test or programming tool to support calibration. The AVRISP mkII, JTAGICE mkII, JTAGICE3, and Atmel-ICE programming tools support the implemented calibration protocol. The usage of these tools to calibrate a device is described in later sections.

2.1. Calibration Protocol

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

Two pins are used for calibration: MOSI and MISO on the ISP interface, or TDI and TDO on the JTAG interface. To simplify the following explanations, only MOSI and MISO are mentioned, although these can be replaced with TDI and TDO for the case of the JTAG interface.

The overall concept is that the programming tool generates the calibration clock (C-clock), and 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 programming tool on the MISO line.

The device is responsible for enabling a pull-up on the MOSI line, and the programming tool is responsible for enabling a pull-up on the MISO line. Unfortunately the programming tool is in many cases behind level converters, so the device sets also the MOSI line high. This is done to make sure that noise is unlikely to corrupt the calibration.

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

2.2. Steps Involved in the Calibration Procedure

1. The programming tool writes the calibration firmware into the device and possibly enables the MISO pull-up, and releases the reset line. If the JTAG interface is used, the JTAG disable bit in MCUCSR is set by the calibration firmware in the device. The calibration clock (approximately 32kHz) is applied on the MOSI line by the programming tool.
2. The device enables the internal pull-up on the MOSI line, sets the MISO line high, and starts listening for the calibration clock on MOSI.
3. When the device detects the calibration clock a binary search is performed 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 of the outcome of the binary search are tested to identify one that does. If calibration fails the MISO line is set low and program flow goes to step 6.
4. The calibration value is stored in EEPROM (skipped if calibration fails).
5. The MISO line is toggled 8 times/4 cycles by the device. The toggling of the MISO line is performed on the falling edge of the clock on the MOSI line (C-clock), but 5 to 10 CPU cycles delayed. In the case of failing calibration the MISO line is not toggled.
6. If necessary, the JTAG interface is re-enabled and the device goes into an infinite loop.
7. 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 memory. If the device has 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 or FLASH to

the OSCCAL register at run-time. A routine for this must therefore be implemented in the final firmware.

2.3. Calibration Firmware

The calibration code is written in AVRASM2 assembly, for Atmel Studio 7.0 (or later). The calibration firmware is structured in a way so that it can easily be updated to support new tinyAVR or megaAVR devices that have an ISP or JTAG interface. Also, the interface for calibration can be configured.

2.3.1. File Organization

File type	File path and name(s)	Description
Atmel Studio solution file	**rc_calib.atsln**	Solution file that should be opened in Atmel Studio to begin working with the project
Root code file	**rc_calib/RC_Calibration.asm**	Root assembly file that includes all the other files
Device specific file	**rc_calib/Device specific/all.asm**	Uses the device setting in Atmel Studio to determine additional parameters for the device, such as the oscillator version
Interface specific files	**rc_calib/Interface specific/ isp_atmelice_interface.inc, jtag_jtagice3_interface.inc, etc.**	Specify additional parameters that are specific to a tool. Only one should be included in **RC_Calibration.asm** and the others should be commented out.
Common files	**rc_calib/Common/macros.inc, main.asm, and memoryMap.inc**	

The root file, **RC_Calibration.asm**, refers to (includes) the following files:

1. A device specific file that defines additional parameters depending on which device is selected in Atmel Studio. This file further includes the following:
   1.1. A memory map file that defines where the code is located.
   1.2. An oscillator version number. This parameter determines 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.
   1.3. Redefinitions of bit and register names may also be present in the device specific file.
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 file, **main.asm**.

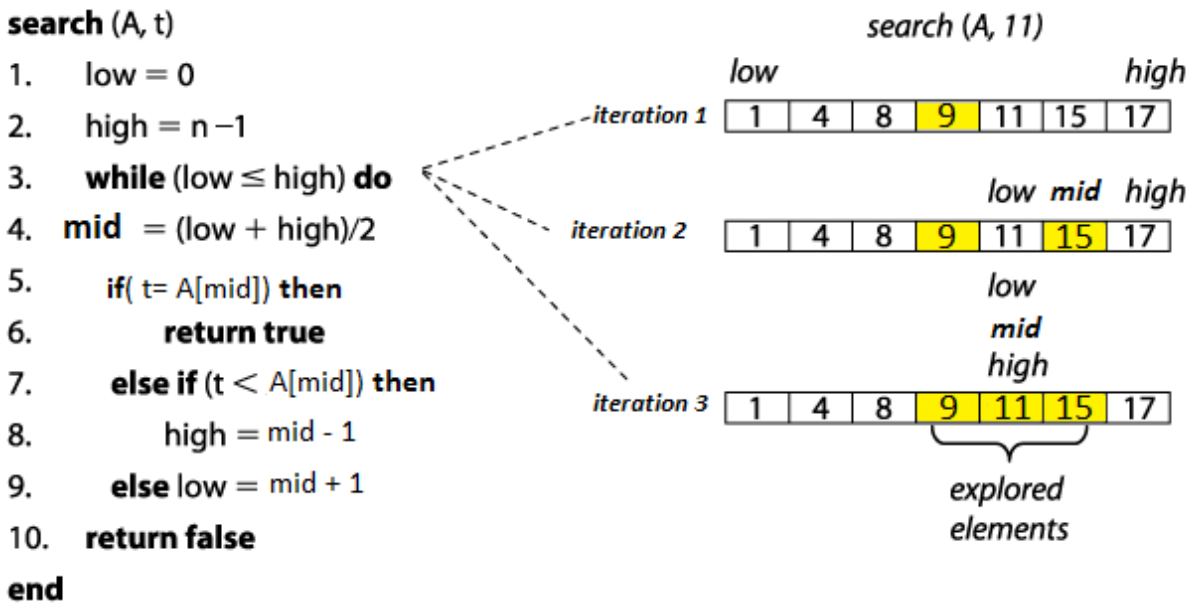
The calibration code is structured to make it easy to change, in order to match a desired target device and programming tool. Furthermore, the extensive use of macros ensures that the code has the smallest possible footprint. Finally, the way devices and calibration interfaces are designed ensures that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

2.4. Binary Search

A binary search algorithm finds the position of a specified input value within an array sorted by values. The sorted array space is repeatedly cut in half according to how the required value compares with the

middle element. The figure below explains this algorithm with an example to find value t = 11 from a sorted array A.

Figure 2-1. Binary Search



2.4.1. Binary Search of OSCCAL

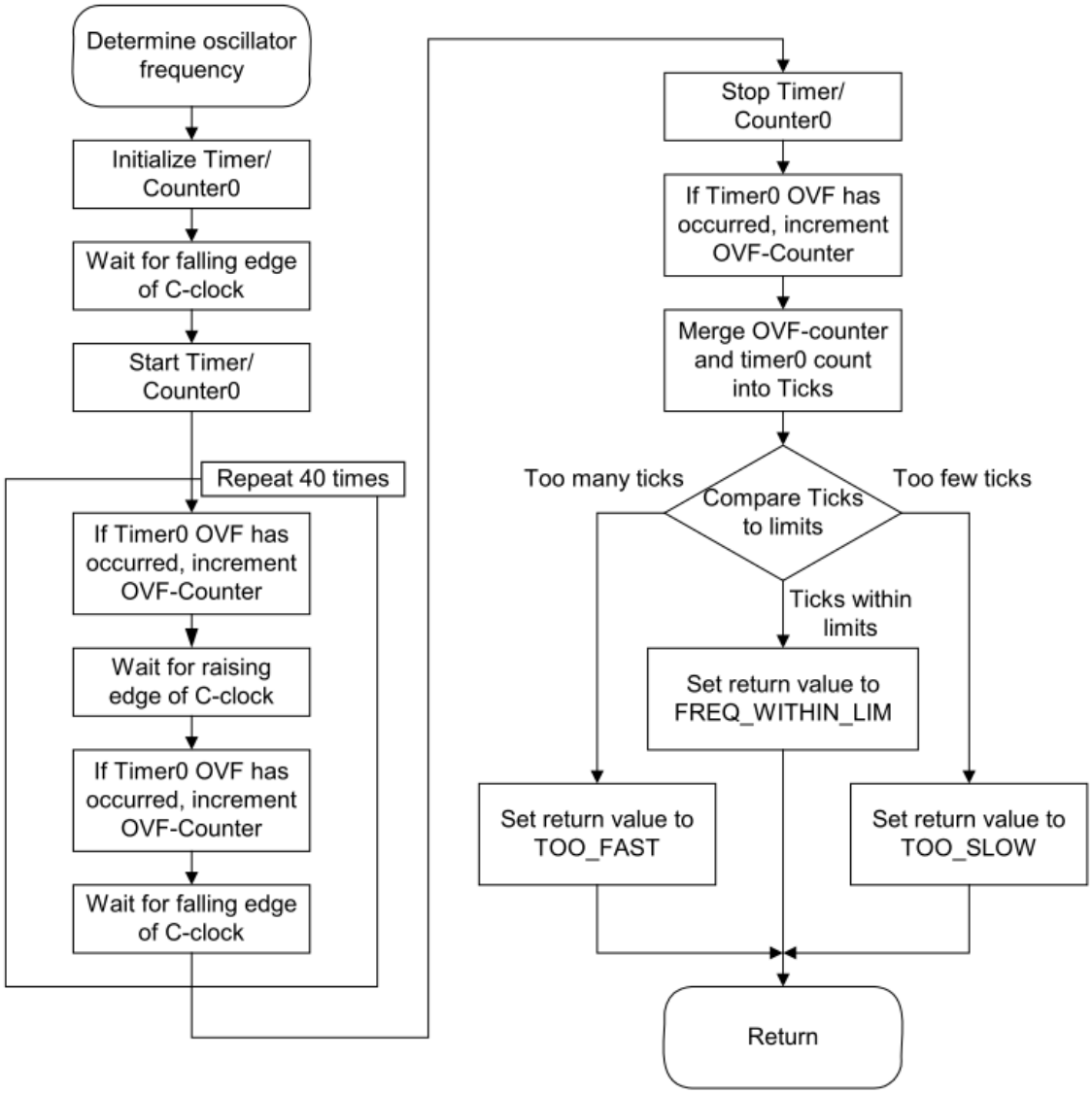
The search is based on a binary search method, a divide-and-conquer method:

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.
   2.1. If the frequency is within 1% accuracy limit, go to step 5.
   2.2. 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, go to step 3.
3. The step size is assigned the value of half the previous step size.
   3.1. If the step size is zero, the binary search has been unsuccessful. Go to step 4.
   3.2. If the step size is not zero, the step size is added to or subtracted from the current value in the OSCCAL register to increase or decrease the oscillator frequency, then go to step 2.
4. Test the four nearest neighbor-values of OSCCAL. This is done to compensate for the lack of a strictly monotonic relationship between OSCCAL and oscillator frequency.
5. If a tested OSCCAL value is within the accuracy limits, continue to store value in EEPROM, signal success and stop calibration.
6. 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 and go to stop calibration.

2.4.2. Determining 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 a 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 is specified in the interface specific include file. The method for determining the oscillator frequency is described in the flowchart below.

Figure 2-2. Flowchart of Algorithm Determining Relationship Between the C-clock and Internal RC Frequency



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 range by eight bits, effectively 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 events 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 still 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 occurs 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 seven 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 eight 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. However, these extremes are very unlikely to be encountered due to the binary search method used.

2.4.3. Correcting Timing Inaccuracies

Since it is not possible to use interrupt driven detection for the C-clock edges in all devices, a polling method is implemented. The consequence of this implementation is that edge detection can be delayed by up to two 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 two timer-ticks (two CPU-cycles). All calculations of limits and constants are performed by the preprocessor, which uses 64-bit accuracy 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.

3. Steps to Calibrate Device Using AVR Tools

The following sections explain how to calibrate the device using Atmel AVR programming tools.

3.1. Assembling the Calibration Firmware

The root assembly file for the calibration firmware is **RC_Calibration.asm**. This file is pointed to by the Atmel Studio solution file **rc_calib.atsln**. In the **RC_Calibration.asm** file it is possible to specify the desired calibration interface: Atmel-ICE, AVRISP mkII, JTAGICE3, or JTAGICE mkII. In addition, it is possible to specify the desired calibration accuracy. Once these choices have been made, the project should be built to produce the binary file **rc_calib.hex** in the **default** sub-directory. This file is downloaded to the device through the programming tool.

Important: 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.

3.2. Command Line Tool

Calibration support in the Atmel-ICE, AVRISP mkII, JTAGICE3, and JTAGICE mkII tools is only supported in the command-line tool **atprogram**. In the **Batch file** sub-directory, batch files are provided that use **atprogram** to program the calibration code into the target device, perform the calibration and then reprogram the device with the final firmware. The batch files perform calibration of the device through the Atmel-ICE, AVRISP mkII, JTAGICE3, and JTAGICE mkII.

3.3. Batch Files

Since multiple commands are executed to perform various operations pertaining to calibration, typing individual commands manually is a tedious process. Batch files are therefore used to automate the process. Based on the programming tool and interface used, the batch file naming convention is:

atprogram_<toolname>_<interface>_rc_calib.bat

Example: **atprogram_ATMELICE_ISP_rc_calib.bat**

If the tool used is JTAGICE3 and interface is JTAG, for example, the file name would be:

atprogram_JTAGICE3_JTAG_rc_calib.bat

3.4. Performance of the Calibration Firmware

The code has been written with a focus on efficiency, since the entire calibration should be performed 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 typically a few hundred bytes, depending on the target device and the interface used for calibration. The required time to program the firmware is thus 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.

3.5. 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 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 equal the conditions desired during calibration.

3.6. Quick Start guide

To quickly get started performing calibration on a supported device, follow these steps:

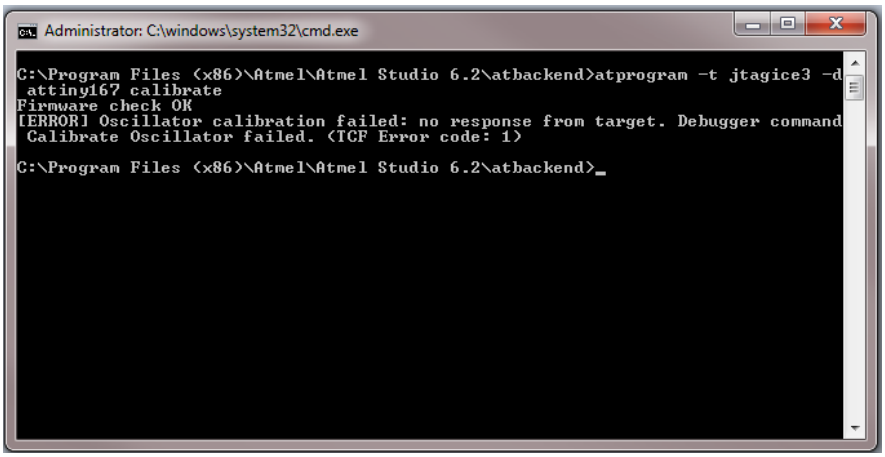
1. Download and install Atmel Studio 7.0 (or later) from http://www.atmel.com.
2. Download and unzip the code for this application note, AVR053 (any location can be used, here called **\AVR053**).
3. Open the **rc_calib.atsln** project/solution in Atmel Studio.
4. Select the target device in Atmel Studio (the default device is ATmega168PB). Many, but not all, tinyAVR and megaAVR devices are supported.
5. Click on the **RC_Calibration.asm** icon to open the file.
6. Select the desired interface by removing the semicolon from one of the include lines in **RC_Calibration.asm** as shown in the figure below.

Figure 3-1. Selecting the Interface in RC_Calibration.asm

```
;*****************************************************************
;* Include the file for the interface the the calibration should
;* be performed on
;*****************************************************************
;.include "Interface specific\isp_AVRISP_mkII_interface.inc"
;.include "Interface specific\jtag_atmelice_interface.inc"
.include "Interface specific\isp_atmelice_interface.inc"
;.include "Interface specific\isp_jtagice3_interface.inc"
;.include "Interface specific\jtag_jtagice3_interface.inc"
```

7. Measure the frequency of the calibration clock with a frequency counter or an oscilloscope.
   - 7.1. The calibration clock is generated on the MOSI pin on an ISP tool or the TDI pin on a JTAG tool.
   - 7.2. To obtain this signal on the tool, connect the tool to a PC and open a command shell window (a DOS prompt) and navigate to the directory **C:\Program Files (x86)\Atmel \Studio\7.0\atbackend**. Note that the exact path to the **atbackend** directory may vary slightly from this depending on the versions of Atmel Studio and Windows® and how Atmel Studio was installed. Type in “atprogram –t <toolname> -d <devicename> calibrate”, and press enter. An example command line is shown in the figure below. This will cause the calibration clock to be generated for a few seconds.

Figure 3-2. Command to Obtain Calibration Clock on the Tool



7.3. The error “no response from target” may occur, but it is OK in this case because the purpose of the command is to force the tool to generate the calibration clock so it can be measured accurately.

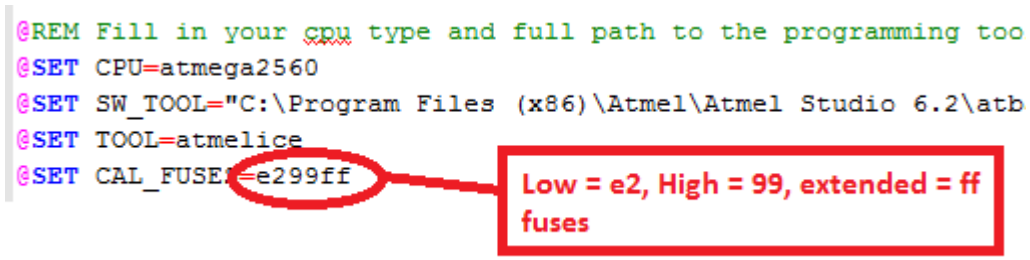
8. Change the line in the interface specific file “.EQU CALIB_CLOCK_FREQ = XXXX” to reflect the measured frequency in Hz. An example is shown in the figure below for the measured frequency of 32092Hz.

Figure 3-3. Updating Calibration Clock Frequency

```
;*********************************************************************
;*  Specify Calibration clock frequency
;*********************************************************************
.EQU    CALIB_CLOCK_FREQ    = 32092 ;Calibration Clock frequency in Hz
```

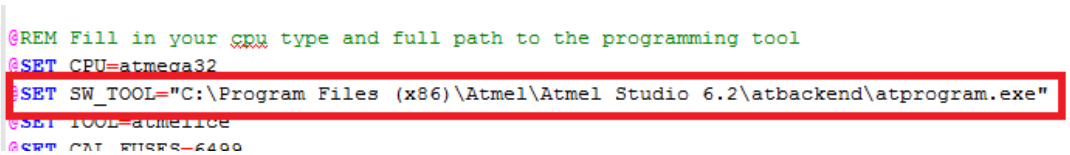
9. In the **RC_Calibration.asm** file, specify the desired target frequency (.EQU TARGET_FREQ = XXXX) and the desired accuracy (.EQU ACCURACY = XX).
   Note: If the accuracy is too tight it may not be possible to calibrate the device and the calibration will fail. Refer to the device data sheet for achievable accuracy.
10. Save all modified files in Atmel Studio by clicking on "File" then "Save All".
11. Click on "Build" then "Build rc_calib" to assemble the project and generate the hex file that should be programmed into the device.
12. Exit Atmel Studio.
13. Using any text editor, open the batch file corresponding to the chosen programming tool and interface.
14. Edit the file to match the desired device, by modifying the "@SET CPU=xxxx" line, for example "@SET CPU=atmega168pb".
15. If the JTAG interface is going to be used for calibration, note that the reset line must be available for the Atmel-ICE, JTAGICE3, or JTAGICE mkII.
16. Edit the fuse settings to the desired settings for the device as shown in the figure below by modifying the argument to CAL_FUSES. Make sure that the settings correspond with the desired calibration frequency: select 8MHz internal RC if calibrating the device to 8MHz. Verify that the "Watchdog Timer always on" fuse is not set.

Figure 3-4. Fuse Settings



17. If the install path for Atmel Studio differs from the one used in the batch file, change the path to the relevant **atprogram.exe** file as shown in the figure below.

Figure 3-5. Tool Path



18. For production calibration the @PAUSE command at successful calibration can be removed.
19. Save the batch file.
20. Connect the tool to the target board. Power the target board and tool. Verify that the serial or USB cable is attached between the tool and the PC.
21. Open a command shell window (a DOS prompt), navigate to the directory **“rc_calib\Batch file\”**, and execute the appropriate batch file (**atprogram_ATMELICE_ISP_rc_calib.bat, atprogram_JTAGICE3_ISP_rc_calib.bat,** etc.).
22. Wait several seconds for the calibration to complete. The batch file can also be modified to program custom firmware rather than the calib_test.hex firmware after the calibration. Be aware that the new calibration value should be loaded into the OSCCAL register at run-time by the firmware.

3.7. Adding Support for New Devices

To add support for a new device, it is necessary to add appropriate definitions for the device to the **all.asm** file. The "#ifdef #endif" block for an old device can be copied, pasted, and edited in the **all.asm** file to 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 an example:

Copy the "#ifdef #endif" block for a pin and feature compatible device.

The ATmega8535 is pin compatible with the ATmega16, though the ATmega8535 has no JTAG interface. The "#ifdef _M16DEF_INC #endif" block in **all.asm** is copied and renamed "#ifdef _M8535DEF_INC #endif”.

Change the oscillator version to match the oscillator version of the new device.

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 is implemented as a reassignment of ATtiny12, and can be used as a reference for reassigning names.

4. References

AVR054: Run-time Calibration of the Internal RC Oscillator

http://www.atmel.com/Images/doc2563.pdf

AVR055: Using a 32kHz XTAL for Run-time Calibration of the Internal RC

http://www.atmel.com/Images/doc8002.pdf

5. Revision History

Doc Rev.	Date	Comments
H	09/2016	Complete modernization and refurbishing of document.
G	05/2006	
F	03/2006	
E	01/2006	Updated details on handshaking and removed duplicate info in 2.7.1

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

 / Rev.: Atmel-2555H-Internal-RC-Oscillator-Calibration-for-tinyAVR-and-megaAVR-Devices_AVR053_Application Note-09/2016

Calibration of internal RC oscillator