8-bit AVR® Microcontrollers

Application Note

Preliminary

AVR140: ATmega48/88/168 family run-time calibration of the Internal RC oscillator for LIN applications

Features

- **Calibration of internal RC oscillator via UART**
- **LIN 2.0 compatible synchronization/calibration to within +/-2% of target frequency**
- **Support for all ATmega48/88/168 AVR® family**
- **Enables robust LIN UART communication with low cost clock sources in varying operating conditions**

Introduction

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

The ATmega48/88/168 AVR microcontroller family offers the possibility to run from an internal RC oscillator. The internal RC oscillator frequency can be calibrated within ±1% of the frequency specified in the datasheet for the device. This feature is ideal for synchronization purposes, and offers significant cost savings compared to using an external oscillator.

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

Rev. 7653A–AVR–09/06

Theory of Operation - The internal RC oscillator

In production the internal RC is calibrated at 3.3V and ambient temperature. The accuracy of the factory calibration is within +/-1% for all automotive AVRs, including ATmega48/88/168 microcontroller familyfamily.

Clock Selection

The AVR fuse settings control the system clock source being used. The device is hipped with internal RC oscillator at 8.0MHz and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock. The startup time is set to maximum and time-out period enabled. (CKSEL = "0010", SUT = "10", CKDIV8 = "0"). The default setting ensures that all users can make their desired clock source setting using any available programming interface.To use the internal oscillator at 8MHz which is the recommended value for proper LIN application, the corresponding fuse setting must be set at CKSEL3..0= 0010 and CKDIV8 = “1”.

Base Frequency

The following section provides an overview of the internal RC oscillator available in the ATmega48/88/168 AVR microcontroller familyfamily.

The ATmega48/88/168 has one 8MHz internal RC oscillator. To make it sufficiently accurate, an Oscillator Calibration register, OSCCAL, is present in the AVR I/O file. The OSCCAL 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. The ATmega48/88/168 has a byte calibration value for the internal RC Oscillator. This byte resides in the high byte of address 0x000 in the signature address space. During reset, this byte is automatically written into the OSCCAL Register to ensure correct frequency of the calibrated RC Oscillator. A programming tool can be used to read the 8MHz calibration byte from the Signature Row and store it in a Flash or EEPROM location. The main program reads this location and copies it into OSCCAL at run-time.

RC Oscillator overview

An overview of the ATmega48/88/168 AVR microcontroller family and their oscillators is available in Table 1. The 8MHz oscillator is controlled by all the 8 bits of the OSCCAL register to tune the frequency. Auto loading of the default calibration value and system clock prescaler is present.

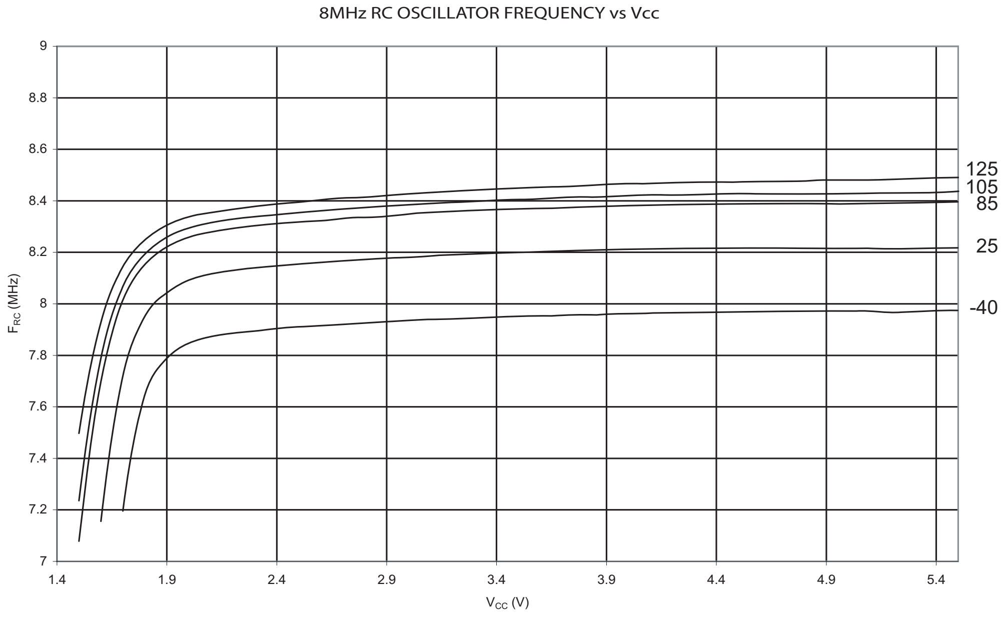
Table 1. RC oscillator main features for the ATmega48/88/168 microcontroller family

Oscillator version	Device	RC Oscillator Frequency (MHz)	CKDIV	PRSCK
5.0	ATmega48/88/168	8.0	Yes	Yes

Oscillator Characteristics

The frequency of the internal RC oscillator is depending on the temperature and operating voltage. An example of this dependency is illustrated in Figure 1, which shows the frequency of the 8MHz RC oscillator of the ATmega48/88/168. As seen from the figure, the frequency increases with increasing temperature, and increases slightly too with increasing operating voltage.

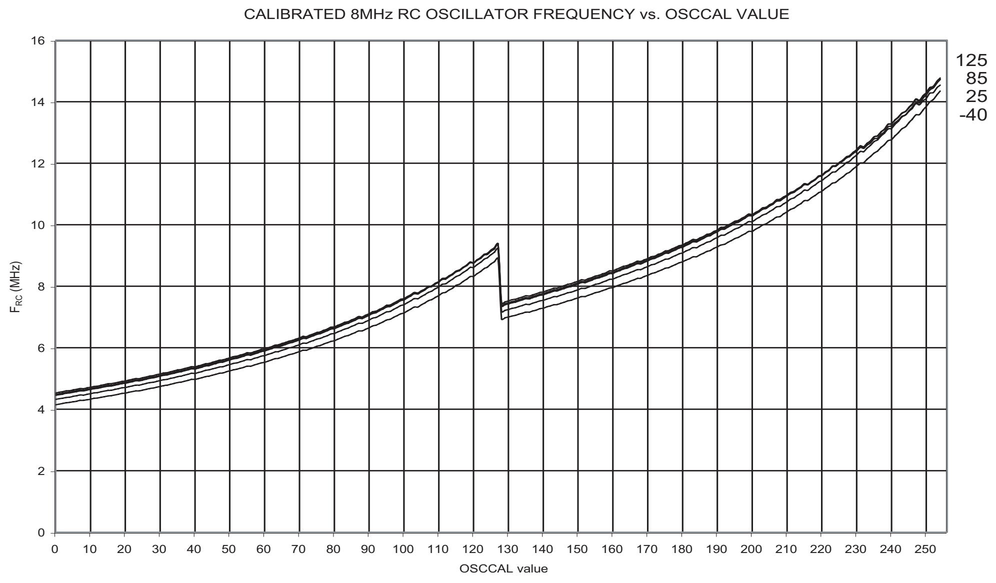
Figure 1. Non Calibrated Oscillator Frequency and Influence by Temperature and Operating Voltage. ATmega88 8MHz oscillator frequency vs V_{CC}



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. 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.

Important: The maximum deviation from the original calibration point is less than ±10% within the whole temperature and voltage range.

Figure 2. RC Oscillator Frequency vs OSCCAL value



An example of the pseudo-monotone relation between the OSCCAL value and the oscillator frequency of the ATmega48/88/168 can be seen in Figure 2. The figure shows the variation of the frequency for OSCCAL= 0 to 255. The OSCCAL register is split in two parts. The MSB of OSCCAL (OSCCAL[7]) selects one of two **overlapping** frequency ranges while the 7 least significant bits are used to tune the frequency within this range.

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

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 is under no circumstances any longer than 5 microseconds. Allow the oscillator to settle at its new frequency before making any frequency measurements for calibration.

The LIN Synchronization Method

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

A LIN network consists of one master node and several slave nodes. The master node is responsible for controlling all communication on the bus. In LIN terminology, communication occurs by sending message frames on the bus. Every message frame starts with a frame header, initiated by the master node (see Figure 3). The header starts with a BREAK and SYNCH pattern, allowing slave nodes to synchronize to the master before any communication on the bus is initiated. The BREAK/SYNCH pattern consists of:

– BREAK signal: At least 13 bit times of dominant (low) value.

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

Figure 3. LIN Frame Format

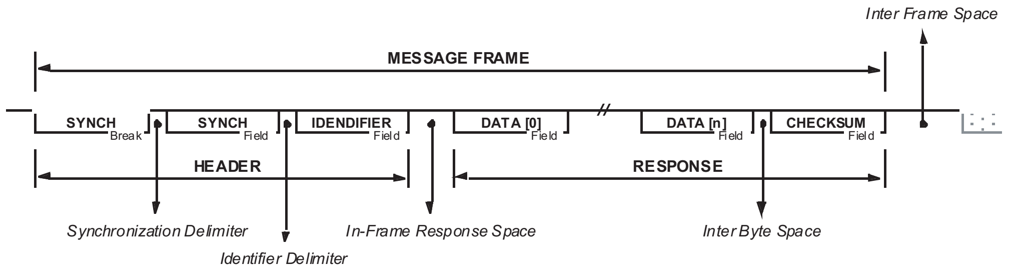
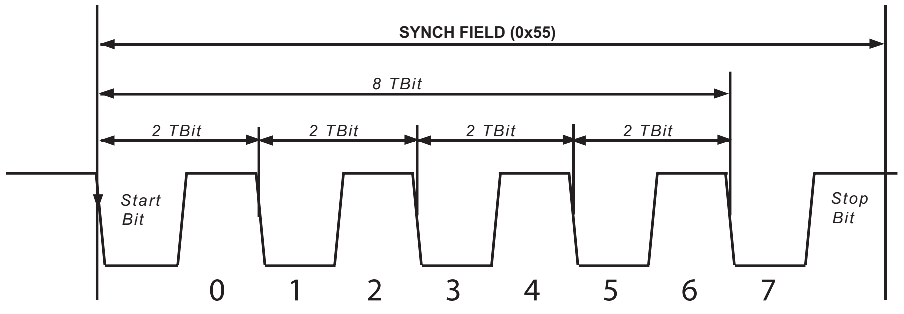


Figure 4. Synch Pattern (0x55) in the Frame Header



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

Consequence of the undefined Duty-Cycle

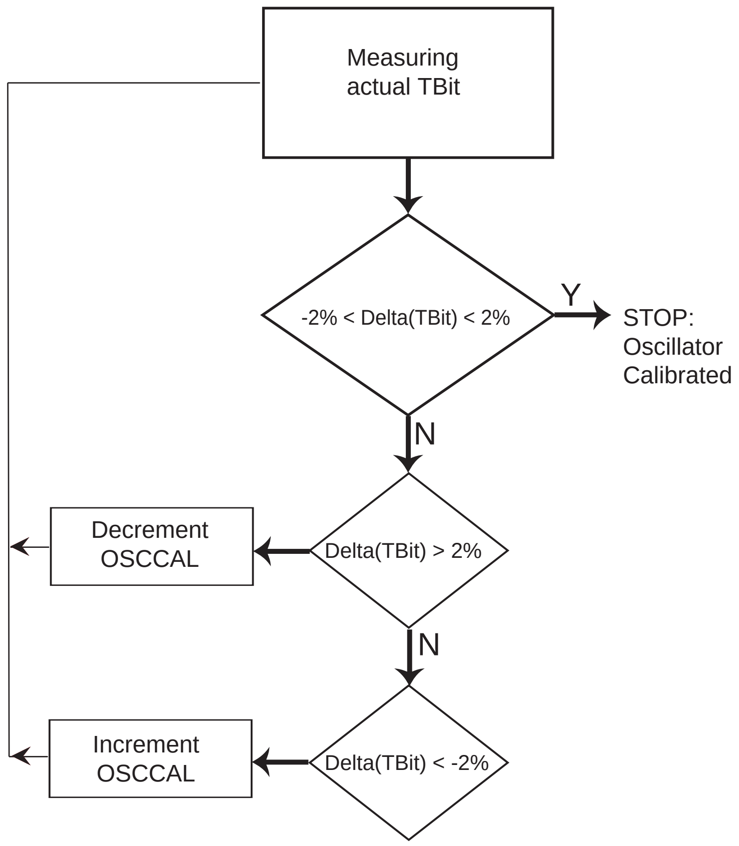
The duty-cycle of the LIN signals are depending from the hardware implementation, so may vary from application to one other. Therefore, the values for TBus_Dominant and TBus_Recessif are not equal. This, indeed, has a strong influence on the way to synchronize the oscillator when considering the 0x55 from the SYNCH byte. Instead of using all alternate of rising and falling edges of the SYNCH byte, only subsequent rising and subsequent falling edges must be considered.

Synchronization Algorithm

The possibility to change the value of OSCCAL during the Oscillator operation allows for in-situ calibration of the slave node to entering Master frames. The principle of operation is to measure the TBit during the SYNCH Byte and to change the calibration value of OSCCAL to recover from local frequency drifts due to local voltage or temperature deviation.

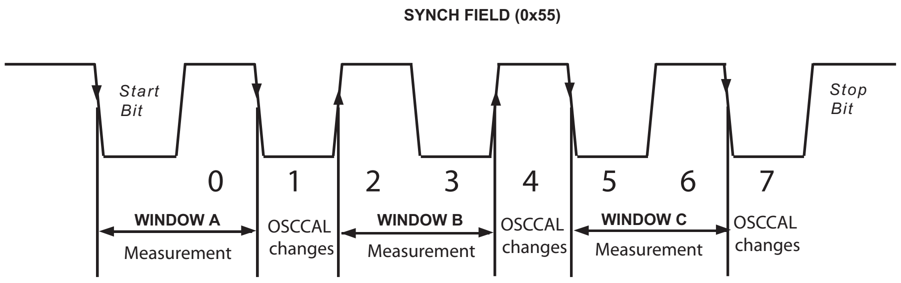
To do so, a dichotomy algorithm is proposed as described in Figure 5.

Figure 5. Automatic Dichotomize Calibration Algorithm for RC oscillator synchronization



The 0x55 of the SYNCH Byte offers three measurements and OSCCAL changes windows. See for more details in Figure 6.

Figure 6. SYNCH byte used for TBit measurements and OSCCAL changes



The entering signal (LINrx) is sent to the Input Capture of the 16-bit timer. During windowA, the first falling edge of the signal is time-stamped with the value of the 16-bit timer/counter. Then, the next falling edge of LINrx is once again time-stamped with the new value of the timer/counter. The difference (which represents 2xTBit), is compared with the reference and a decision is made how to change OSCCAL. The internal RC frequency must be settled before the next window (windowB) occurs (remember, it takes maximum 5µS to stabilize the frequency after OSCCAL has been changed). This procedure can be repeated up to three times total (WindowA, B and C), but can be ended as soon as the measured TBit gets into the required tolerance (±2%). See Figure 5 and Figure 6 for more details on both the algorithm and the SYNCH Byte).

Optimum Dichotomous Steps

The search of the optimum OSCCAL value for a correct synchronization to entering Master frame is dichotomous. Based on the three search windows allowed during the SYNCH Byte (see Figure 6) and the maximum deviation measured on worst case conditions, the following increment/decrement have been selected:

Table 2. Prefferred Dichotomous increment/decrement for proper LIN synchronization

Measuring Window	OSCCAL change (steps)
Window A	± 16
Window B	± 8
Window C	± 4

These three values are prefferred values which have been proven working all time whatever the original OSCCAL value and the operating conditions.

Calibration and Screening of Production Parts

As indicated in Table 1, production parts are shipped calibrated at 8MHz. The calibration routine is performed at ambiant temperature and 3V and the resulting OSCCAL[7] can be either ‘0’ or ‘1’.

To allow linear calibration and correct synchronization routine, the parts are screened for their frequency at OSCCAL = 127 and OSCCAL = 128. Since the maximum deviation of the Frequency over the full temperature and voltage ranges is ±10%, only parts with Frequency $\leq$ 8.8MHZ at OSCCAL = 127 or with Frequency $\geq$ 7.2MHZ at OSCCAL = 128 are accepted. These limits have been defined to satisfy the requirements of the LIN synchronization algorithm described here after.

Precaution Against OSCCAL Discontinuity

The Figure 2 illustrates the on-purpose discontinuity. For one correct re-synchronization, the frequency change must be kept on the same side of the discontinuity (no change of OSCCAL[7]). Since there will be no device having frequency changed by more than 10%, thus no reason to change the frequency value by more than 10%. Therefore, when calibration tries to cross the border because of subsequent increase (or decrease) in OSCCAL values, then the routine must be stopped.

example: For parts operating in the lower part of the curve, if New_OSCCAL >127 then New_OSCCAL = 127. Similar for parts operating on the high side of the discontinuity.

Comparison with LIN proposed method

The LIN specification proposes a synchronization methods which is based on the measurement of local TBit at once during the SYNCH field. This allows the correction of local oscillator before the remaining part of the frame is interpreted. The method described in this application note offers to the user the possibility to have up to three local RC oscillator corrections, thus improving the precision (e.g. robustness) of the system. With this procedure, no frame, even at start-up can be mis-interpreted for synchronization reasons.

C Code Example

The following C code gives an example of the implementation of the above described algorithm for a ATmega88 target. It is called by a LIN controller whose description is not part of this application note.

```
/*
**
*****************************************************************************
**
**
**          Copyright (c) 2004/2005 - Atmel Corporation
**          Proprietary Information
**
** Project    : AVR LIN SLAVE CONTROLLER
** Module     :
** Description: input capture interrupt routine for tbit calcualtion
** Target       : AVR ATMEGA48/88/168
** Compiler     : IAR Embedded Workbench
**
**
** Version :     Date:        Author:      Comment:
**    1.0        19.08.2004    E.G.         Creation
**    1.1        19.11.2004    E.G.         3 cycle RC calibration (OSCCAL +/- computed offset)
**    1.2        25.11.2004    E.G.         divide and conquer 3 steps for RC Oscillators v8.0
**    1.3        08.06.2005    E.G.         No cross over frequency discontinuity version
**
**
**
**
** LICENSE -
**
** ATMEL - 2004/2005
** All software programs are provided 'as is' without warranty of any kind:
** Atmel does not state the suitability of the provided materials for any
** purpose. Atmel hereby disclaim all warranties and conditions with regard
** to the provided software, including all implied warranties, fitness for
** a particular purpose, title and non-infringement.In no event will Atmel
** be liable for any indirect or consequential damages or any damages
** whatsoever resulting from the usage of the software program.
*****************************************************************************
**
*/
//
/*_____ I N C L U D E S ____________________________________________________*/
#include "config.h"
#include "lib_mcu/lin_uart/slave_lin.h"
#include "lib_mcu/lin_uart/runtime_calibration_lib.h"


//divide and conquer 3 cycles for LIN run-time Internal Oscillator Calibration
```

```
//initial step = 16 , 2nd step =8, 3rd and last step = 4

//Ressource usage: Timer 1 (16 bit) + Input Capture module
//System Frequency: Internal RC oscillator @ 8MHz


/*_____ D E F I N I T I O N S ______________________________________________*/
volatile U8  Timer_IC_cnt;
volatile U8 _lin_synchronized;    //lin is synchronized with master if set (within 2%)
volatile U16 TimeStamp_IC1;
volatile U16 TimeStamp_IC2;

//***************************************************************************
// Timer Input Capture interrupt service routine
// use to calculate LIN master Tbit value (for minimum jitter)
//***************************************************************************
#pragma vector= TIMER1_CAPT_vect
__interrupt void TIMER1_CAPT_ISR (void){
#ifdef _RUN_TIME_RC_CALIBRATION_ENABLE
  U8 new_osccal;
  U8 osccal_prior_synchr;
  U8 osccal_step;
  U16 measured_master_tbit;
  signed int tbit_diff;

  //******************* 1st Cycle (Window A) *******************//
  if (Timer_IC_cnt == 0) {
    TimeStamp_IC1 = ICR1; //timestamp for 1st SynchField falling edge (First Measure Cycle)
  } else if (Timer_IC_cnt == 1) {
    TimeStamp_IC2 = ICR1; //timestamp for 2nd SynchField falling edge (First Measure Cycle), 2 Master
Tbit measured
    TCCR1B  = (1<<ICES1) | (1<<CS10);  //input capture on rising edge for next cycle, no prescaler
    osccal_step = 16;
    measured_master_tbit = TimeStamp_IC2 - TimeStamp_IC1  ;
    tbit_diff = EXPECTED_TBIT - measured_master_tbit;
    osccal_prior_synchr = OSCCAL ;

    //test if we are in the right range, if true stop capturing
    if ((tbit_diff <= TBIT_DIFF_THRESHOLD_MAX) && (tbit_diff >= TBIT_DIFF_THRESHOLD_MIN)){
      Timer_stop_capture(); // stop autocalibration sequence
      _lin_synchronized = 1;  //lin slave is now synchronized correctly
    }else{
      //otherwise increment or decrement only by +/- step
      _lin_synchronized = 0;
      if (tbit_diff > 0) {
        new_osccal =  osccal_prior_synchr + osccal_step ;
      } else {
        new_osccal =  osccal_prior_synchr - osccal_step ;
      }
      if (new_osccal<128 & osccal_prior_synchr>=128) {
        new_osccal = 128;   //high to low side saturation
      }
      if (new_osccal>=128 & osccal_prior_synchr<128){
        new_osccal = 127;  //low to high side saturation
      }
      OSCCAL = new_osccal;  //takes about 5us to get the new frequency
    }
    //******************* 1st Cycle (Window A( *******************//

    //******************* 2nd Cycle ("Window B")*******************//
  } else if (Timer_IC_cnt == 2) {
    TimeStamp_IC1 = ICR1;   //timestamp for 2nd SynchField rising edge (Second Measurement Cycle)
  } else if (Timer_IC_cnt ==3) {
    TimeStamp_IC2 = ICR1;    //timestamp for 3rd SynchField rising edge (Second Measure Cycle), 2
Master Tbit measured
    TCCR1B  &= ~(1<<ICES1);  //input capture on falling edge for next cycle
    measured_master_tbit = TimeStamp_IC2 - TimeStamp_IC1;
    tbit_diff = EXPECTED_TBIT - measured_master_tbit;
    osccal_prior_synchr = OSCCAL ;
    osccal_step = 8;

    //test if we are in the right range, if true stop capturing
    if ((tbit_diff <= TBIT_DIFF_THRESHOLD_MAX) && (tbit_diff >= TBIT_DIFF_THRESHOLD_MIN)){
      Timer_stop_capture();//stop autocalibration sequence
      _lin_synchronized = 1; //lin slave is now synchronized correctly
    }else{
      //otherwise increment or decrement by +/- step
      _lin_synchronized = 0;
      if (tbit_diff > 0) {
        new_osccal =  osccal_prior_synchr + osccal_step ;
      } else {
        new_osccal =  osccal_prior_synchr - osccal_step ;
      }
      if (new_osccal<128 & osccal_prior_synchr>=128) {
        new_osccal = 128;  //high to low side saturation
      }
      if (new_osccal>=128 & osccal_prior_synchr<128){
        new_osccal = 127;  //low to high side saturation
      }
      OSCCAL = new_osccal;  //takes about 5us to get the new frequency
    }
    //******************* 2nd Cycle ("Window B")*******************//

    //******************* 3rd Cycle ("Window C")*******************//
  }else if (Timer_IC_cnt == 4) {
    TimeStamp_IC1 = ICR1;  //timestamp for 4th SynchField falling edge (Second Measurement Cycle)
  } else if (Timer_IC_cnt ==5) {
    TimeStamp_IC2 = ICR1; //timestamp for 5th and last SynchField falling edge (Third and last Measure
Cycle), 2 Master Tbit measured
    osccal_step = 4;
    measured_master_tbit = TimeStamp_IC2 - TimeStamp_IC1  ;
    tbit_diff = EXPECTED_TBIT - measured_master_tbit;
    osccal_prior_synchr = OSCCAL ;

    //test if we are in the right range, if true stop capturing
    if ((tbit_diff <= TBIT_DIFF_THRESHOLD_MAX) && (tbit_diff >= TBIT_DIFF_THRESHOLD_MIN)){
      Timer_stop_capture();//stop autocalibration sequence
      _lin_synchronized = 1; //lin slave is now synchronized correctly
    }else{
      //otherwise increment or decrement  by +/- step
      _lin_synchronized = 0;
      if (tbit_diff > 0) {
        new_osccal =  osccal_prior_synchr + osccal_step ;
      } else {
        new_osccal =  osccal_prior_synchr - osccal_step ;
      }
      if (new_osccal<128 & osccal_prior_synchr>=128) {
        new_osccal = 128;  //high to low side saturation
      }
      if (new_osccal>=128 & osccal_prior_synchr<128){
        new_osccal = 127;  //low to high side saturation
      }
      OSCCAL = new_osccal;  //takes about 5us to get the new frequency
      Timer_stop_capture();   //stop autocalibration sequence
      _lin_synchronized = 1; //lin slave is now synchronized correctly
    }
  }
  //******************* 3rd Cycle ("Window C")*******************//
  Timer_IC_cnt ++ ;
#endif
}
```

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

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 Tsimhatsui
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
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131
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
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
TEL 1(719) 576-3300
FAX 1(719) 540-1759

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

e-mail
literature@atmel.com

Web Site
http://www.atmel.com