CAN Baudrate Detection with Infineon CAN devices

This paper describes two different methods to detect the CAN baudrate with Infineon CAN devices. These methods allow non-initialized CAN nodes to connect to a running CAN system, although the current baudrate is not known. Hereby the running CAN system is not disturbed.

Author: Tobias Wenzel / Microcontroller Application Engineering
LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (www.infineon.com).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.
## Contents

1. **Overview** ................................................................. 3  
   1.1 Detection with CAN Analyzing modes ................................. 3  
   1.2 Measurements of dominant CAN bits via a Timer .................. 4  
   1.2.1 CAN Baudrate Detection with C16xC Microcontrollers ........... 5  
   1.2.2 Measurement of dominant CAN bits ............................... 6  
   1.2.3 CAN Bit Time Configuration .................................... 7  
   1.2.4 Program Code: baudrate detection ............................... 9  
   1.2.4.1 Baudrate Detection Routine “bauddet.c” ..................... 9  
   1.2.4.2 Dominant bit measurement “measure.c” ...................... 10
1 Overview

Usually in a CAN network each node participates in any CAN traffic when the Error management state of the connected nodes is Error Active (normal operation state). The node transmitting a message expects at the end of a CAN frame a dominant acknowledge bit from at least one other CAN node. On the one hand on each transaction on the bus, the connected nodes, when they are no transmitter and not Bus Off, react at least by transmitting the dominant acknowledge bit. On the other hand it does not influence the CAN communication if one of the CAN receive nodes does not acknowledge a transferred CAN frame as soon as another CAN receive node accepts the current message. This behavior can be used for detecting the CAN baudrate from non-initialized CAN nodes. These nodes monitor the bus traffic while detecting the CAN bit time, without any active access to the CAN bus. The following chapters describe two methods for a baudrate detection with Infineon CAN devices.

1.1 Detection with CAN Analyzing modes

In CAN Analyzing mode, which is implemented in the TwinCAN module, the CAN module is able to monitor the CAN bus without participating in the CAN bus protocol. It allows for monitoring bus transfers without any active transmission like acknowledge, active error frame or response to remote frames.

When CAN Analyzing mode is enabled, the CAN controller is able to switch into several bit time settings without transmitting error frames in case of faulty baudrate configuration. The CAN controller tries different baudrate settings, configured by CPU, during CAN frame reception until the suitable baudrate is found in the internal CAN bit time table. When one bit time setting matches, no receive error conditions occur any more and the CAN interface is now ready for participating in CAN
data transfers. The following flow chart describes CAN baudrate detection using the CAN Analyzing mode.

1.2 Measurement of CAN baudrate via Timer module

When the CAN module does not support the Analyzing mode as described in the previous chapter, an additional timer is needed for detecting the CAN baudrate. During the detection time, the non-initialized CAN module is disabled in order to avoid disturbances on the CAN bus.

The task of the timer module is to measure the length of a single CAN bit. The main problem is to filter a single CAN bit in the bit stream because CAN allows consecutive bits with the same polarity. Only the dominant acknowledge bit is located between two recessive CAN bits, which are the CRC delimiter and the acknowledge delimiter (see figure below). When the timer monitors all dominant
bitfields for more than one CAN frame time, at least the single dominant acknowledge bit indicates the baudrate of the running CAN system.

1.2.1 CAN Baudrate Detection with C166 Microcontrollers

The following example shows a procedure calculating the CAN baudrate with a timer. As an example a CAN microcontroller of the Infineon C166 family is used. It allows the connection of the non-initialized node into a running CAN system (hot plug in). Several CPU frequencies are supported and no additional pin is required for the timer connection. The calculation software should

![Diagram of CAN frame with labels for CRC Delimiter (CRCDel), Acknowledge (ACK), and Acknowledge Delimiter (ACKDel). The diagram shows a recessive bit, a dominant bit in the acknowledge slot, and two recessive bits at the end of the frame.]
CAN Baudrate Detection

be executed either on the external bus in 16-bit demultiplexed mode or from internal memory for reasons of execution time. The following flow chart describes the software structure.

1.2.2 Measurement of dominant CAN bits
For the measurement the PWM timer unit is used, because it supports the counting frequency with the highest resolution. In order to achieve the best measurement results, the timer is clocked with CPU frequency. The timer measures the duration of low levels on the CAN bus. When the bus is IDLE (no bus traffic), the program waits until the next CAN communication is monitored.

Figure 3 :Detection with timer

1.2.2 Measurement of dominant CAN bits
For the measurement the PWM timer unit is used, because it supports the counting frequency with the highest resolution. In order to achieve the best measurement results, the timer is clocked with CPU frequency. The timer measures the duration of low levels on the CAN bus. When the bus is IDLE (no bus traffic), the program waits until the next CAN communication is monitored.
The polling mechanism of pin CAN_RxD for one CAN low time is as follows:

- when a dominant level is detected on pin CAN_RxD, timer is started
- timer is stopped when a recessive level is detected again
- measured timer ticks are stored

This procedure is done 100 times in order to ensure measuring the dominant acknowledge bit on each starting point.

### 1.2.3 CAN Bit Time Configuration

The lowest counter value stored during the measurement is assumed to be the time for one CAN bit. This value is used for further calculation. The dependency between the timer ticks and the CAN bit time is calculated with the following equation.

\[
m = \frac{f_{cpu} \cdot t_{can}}{1/CAN\_baudrate} = \frac{2(BRP+1) \cdot [1 + (TSEG1+1) + (TSEG2+1)]}{f_{cpu}}
\]

PWM timer ticks \( m \):
CAN bit time \( t_{can} \):

\[
m = f_{cpu} \cdot t_{can}
\]

\[
t_{can} = \frac{1}{CAN\_baudrate}
\]

\[
t_{can} = \frac{2(BRP+1) \cdot [1 + (TSEG1+1) + (TSEG2+1)]}{f_{cpu}}
\]

The predefined values for the CAN bit timing for various CPU frequencies are located in a ROM table called CAN calculation table. On the one hand the CAN calculation table divides the timer ticks in several ranges around the nominal timer tick value expected as result of the equation described above. These ranges are needed because the handling of the timer by software leads to deviations of the measured tick result. On the other hand the table contains CAN bit time settings for \( Tseg1 \) (Time Segment1=\( TSEG1+1 \)), \( Tseg2 \) (Time Segment2=\( TSEG2+1 \)) and CAN prescaler (\( BRP+1 \)) related to the timer ticks. The CAN settings are derived from CAN application layer specification like sample point around 80% and SJW (synchronization jump width) of one time quanta.

As a result two different software versions are needed due to overlapping of timer tick ranges using several CPU frequencies.

Below the CAN calculation tables for \( f_{cpu} = 24/16/8 \) MHz and \( f_{cpu} = 20/10/5 \) MHz are shown.

**Table 1 : CAN calculation table - \( f_{cpu} = 24, 16, 8 \) MHz**

<table>
<thead>
<tr>
<th>( f_{cpu} / \text{MHz} )</th>
<th>CAN Baudrate [kBaud]</th>
<th>( (Tseg1+1) /\text{hex} )</th>
<th>( (Tseg2+1) /\text{hex} )</th>
<th>( (BRP+1) /\text{hex} )</th>
<th>Bit Timing Register / Hex</th>
<th>Sample Point in %</th>
<th>Expected measured timer ticks</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>1000</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>1800</td>
<td>80</td>
<td>20 ( \leq ) 24 ( &lt; ) 27</td>
</tr>
<tr>
<td>24</td>
<td>500</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>1801</td>
<td>80</td>
<td>40 ( &lt; ) 48 ( \leq ) 55</td>
</tr>
<tr>
<td>24</td>
<td>250</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>1803</td>
<td>80</td>
<td>86 ( &lt; ) 96 ( \leq ) 107</td>
</tr>
<tr>
<td>24</td>
<td>125</td>
<td>9</td>
<td>2</td>
<td>8</td>
<td>1807</td>
<td>80</td>
<td>180 ( &lt; ) 192 ( \leq ) 201</td>
</tr>
<tr>
<td>24</td>
<td>50</td>
<td>9</td>
<td>2</td>
<td>20</td>
<td>1814</td>
<td>80</td>
<td>470 ( &lt; ) 480 ( &lt; ) 489</td>
</tr>
</tbody>
</table>
Table 1: CAN calculation table - fcpu = 24, 16, 8 MHz

<table>
<thead>
<tr>
<th>fcpu / MHz</th>
<th>CAN Baudrate [kBaud]</th>
<th>(Tseg1+1) /hex</th>
<th>(Tseg2+1) /hex</th>
<th>(BRP+1) /hex</th>
<th>Bit Timing Register / Hex</th>
<th>Sample Point in %</th>
<th>Expected measured timer ticks</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1000</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1400</td>
<td>75</td>
<td>10 &lt; 16 &lt; 19</td>
</tr>
<tr>
<td>16</td>
<td>500</td>
<td>C</td>
<td>3</td>
<td>1</td>
<td>2B00</td>
<td>81,2</td>
<td>28 &lt; 32 &lt; 39</td>
</tr>
<tr>
<td>16</td>
<td>250</td>
<td>C</td>
<td>3</td>
<td>2</td>
<td>2B01</td>
<td>81,2</td>
<td>56 &lt; 64 &lt; 69</td>
</tr>
<tr>
<td>16</td>
<td>125</td>
<td>C</td>
<td>3</td>
<td>4</td>
<td>2B03</td>
<td>81,2</td>
<td>120 &lt; 128 &lt; 139</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td>C</td>
<td>3</td>
<td>10</td>
<td>2B09</td>
<td>81,2</td>
<td>310 &lt; 320 &lt; 329</td>
</tr>
<tr>
<td>8</td>
<td>500</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1400</td>
<td>75</td>
<td>10 &lt; 16 &lt; 19</td>
</tr>
<tr>
<td>8</td>
<td>250</td>
<td>C</td>
<td>3</td>
<td>1</td>
<td>2B00</td>
<td>81,2</td>
<td>28 &lt; 32 &lt; 39</td>
</tr>
<tr>
<td>8</td>
<td>125</td>
<td>C</td>
<td>3</td>
<td>2</td>
<td>2B01</td>
<td>81,2</td>
<td>56 &lt; 64 &lt; 69</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>C</td>
<td>3</td>
<td>5</td>
<td>2B04</td>
<td>81,2</td>
<td>150 &lt; 160 &lt; 169</td>
</tr>
</tbody>
</table>

Table 2: CAN calculation table - fcpu = 20,10,5 MHz

<table>
<thead>
<tr>
<th>fcpu / MHz</th>
<th>CAN Baudrate [kBaud]</th>
<th>(Tseg1+1) /hex</th>
<th>(Tseg2+1) /hex</th>
<th>(BRP+1) /hex</th>
<th>Bit Timing Register / Hex</th>
<th>Sample Point in %</th>
<th>Expected measured timer ticks</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1000</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1600</td>
<td>80</td>
<td>14 &lt; 20 &lt; 27</td>
</tr>
<tr>
<td>20</td>
<td>500</td>
<td>F</td>
<td>4</td>
<td>1</td>
<td>3E00</td>
<td>80</td>
<td>34 &lt; 40 &lt; 47</td>
</tr>
<tr>
<td>20</td>
<td>250</td>
<td>F</td>
<td>4</td>
<td>2</td>
<td>3E01</td>
<td>80</td>
<td>74 &lt; 80 &lt; 87</td>
</tr>
<tr>
<td>20</td>
<td>125</td>
<td>F</td>
<td>4</td>
<td>4</td>
<td>3E03</td>
<td>80</td>
<td>154 &lt; 160 &lt; 167</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>F</td>
<td>4</td>
<td>10</td>
<td>3E09</td>
<td>80</td>
<td>390 &lt; 400 &lt; 409</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1600</td>
<td>80</td>
<td>14 &lt; 20 &lt; 27</td>
</tr>
<tr>
<td>10</td>
<td>250</td>
<td>F</td>
<td>4</td>
<td>1</td>
<td>3E00</td>
<td>80</td>
<td>34 &lt; 40 &lt; 47</td>
</tr>
<tr>
<td>10</td>
<td>125</td>
<td>F</td>
<td>4</td>
<td>2</td>
<td>3E01</td>
<td>80</td>
<td>74 &lt; 80 &lt; 87</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>F</td>
<td>4</td>
<td>5</td>
<td>3E04</td>
<td>80</td>
<td>190 &lt; 200 &lt; 209</td>
</tr>
<tr>
<td>5</td>
<td>250</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1600</td>
<td>80</td>
<td>14 &lt; 20 &lt; 27</td>
</tr>
<tr>
<td>5</td>
<td>125</td>
<td>F</td>
<td>4</td>
<td>1</td>
<td>3E00</td>
<td>80</td>
<td>34 &lt; 40 &lt; 47</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>1604</td>
<td>80</td>
<td>90 &lt; 100 &lt; 109</td>
</tr>
</tbody>
</table>
1.2.4 Program Code: baudrate detection

1.2.4.1 CAN Baudrate detection routine “bauddet”

/********************************************
Infineon Technologies (copyright 1999)
Application Engineering Munich
Program “bauddet.c”
Compiler used: Keil µVision 1.24 compiler V3.x (DLL 1.24)
Hardware requirements: C166 with integrated CAN interface and PWM timer unit
Last modifications: 4/99
Revision: 1.0
Authors: Tobias Wenzel
Christian Bauer
DESCRIPTION: Main program of bauddet.prj:
Measure baudrate on bus
Init CAN / MOs
Send MO2 id 2 for acknowledgement
*********************************************/

#include <REG167.H> /* 2260 register definitions C167 */
#include <canr_16x.h>
#include <intrins.h>
#define Measures 100
extern unsigned int MeasCnt;
extern unsigned int *res_add;
extern void measurement();

void main ()
{
/* be careful with initialisation of global variables */
/* for correct work use original start167.a66(Keil)) */
{
    unsigned int *msg2cp=0xef20;
    unsigned int *mptr;
    unsigned int bdrt;
    unsigned int i,j,min_dt,dt;
    unsigned int ticks[Measures]; // measurement result table

    /*PWM-init : */
    PWMCON0=0;
    PWMCON1=0;
    PWMIC=0;
    PP0=0xf000;
    DP44=0xdf; // set P4.5 to input
    do {
        // until successful initialisation of CAN
        CR=0x01; // switch off CAN
        SR=0; // clear status reg
        bdrt=0; // clear BTR result
        while(bdrt==0) { // until measurement successful
            /*Measurement */
            MeasCnt=Measures; // 100 measurements
            res_add=ticks;
            measurement();
            /*Search for lowest count*/
            min_dt=ticks[0];
            for(i=1;i<Measures;i++)
            {
                dt=ticks[i]-ticks[i-1];
                if(dt<min_dt) min_dt=dt;
            }
            /*Comparison*/
            /*For 8,16,24 MHz : */
            if((min_dt>=10)&&(min_dt<20)) bdrt=0x1400; //16MHz, 1MbAud / 8MHz, 500kBaud
            else if((min_dt>=28)&&(min_dt<40)) bdrt=0x3a00; //16MHz, 500kBaud / 8MHz, 250kBaud
            else if((min_dt>=56)&&(min_dt<70)) bdrt=0x3a01; //16MHz, 250kBaud / 8MHz, 125kBaud
        }
    }
}
else if((min_dt>=120)&&(min_dt<140)) bdrt=0x3a03;  //16MHz,125kBaud
else if((min_dt>=310)&&(min_dt<330)) bdrt=0x3a09;  //16MHz, 50kBaud
else if((min_dt>=150)&&(min_dt<170)) bdrt=0x3a04;  // 8MHz, 50kBaud
else if((min_dt>= 20)&&(min_dt< 28)) bdrt=0x2700;  //24MHz,  1MBaud
else if((min_dt>= 40)&&(min_dt< 56)) bdrt=0x6f00;  //24MHz,500kBaud
else if((min_dt>= 86)&&(min_dt<108)) bdrt=0x6f01;  //24MHz,250kBaud
else if((min_dt>=180)&&(min_dt<202)) bdrt=0x6f03;  //24MHz,125kBaud
else if((min_dt>=470)&&(min_dt<490)) bdrt=0x6f09;  //24MHz, 50kBaud
*/
/*For 5,10,20 MHz (don’t enable both parts !):*/
if((min_dt>= 14)&&(min_dt< 28)) bdrt=0x1600;  //20MHz, 1MBaud / 10MHz,500kBaud /
//5Mhz,250kBaud
else if((min_dt>= 34)&&(min_dt< 48)) bdrt=0x3e00;  //20MHz,500kBaud / 10MHz,250kBaud /
//5MHz,125kBaud
else if((min_dt>=74)&&(min_dt< 88)) bdrt=0x3e01;  //20MHz,250kBaud / 10MHz,125kBaud
else if((min_dt>=190)&&(min_dt<210)) bdrt=0x3e03;  //20MHz,125kBaud
else if((min_dt>=390)&&(min_dt<410)) bdrt=0x3e09;  //20MHz, 50kBaud
else if((min_dt>=470)&&(min_dt<490)) bdrt=0x3e04;  //10MHz, 50kBaud
else if((min_dt>=90)&&(min_dt<110)) bdrt=0x1604;  // 5MHz, 50kBaud
}

/*INIT CAN : */
_bfld_ (P4, 0x0060, 0x0060);  // Port
_bfld_ (DP4, 0x0060, 0x0040); //  init
CR=0x41;// set CCE and INIT in Control Register (EF00h)
SR=0x00;// Clear Status Partition (EF01h)
BTR=bdrt;    // set baudrate
GMS=0xe0ff;// Global Mask Short (EF06h) each bit of standard ID must match to store msg
UGML=0xffff;// Upper Global Mask Long (EF08h)
LGML=0xf8ff;// Lower Global Mask Long (EF0Ah) each bit of extended ID must match to store msg
LMLM=0x0000;// Upper Mask of Last Message (EF0Ch)
LMLM=0x0000;// Lower Mask of Last Message (EF0Fh) every message into MO 15 (Basic CAN Feature)
// reset msg obj regs
mptr=0xef10;
for(i=1;i<16;i++) {
  *mptr++=0x5555;                    // msg object ctrl reg
  for(j=1;j<8;j++) *mptr+=0x0000;   // msg object regs
}
mptr=0xef00;*mptr=0x5595;  // Define MO15 for acception of undefined messages (not necessary
// if detection takes place during other nodes communicat)
CR = 0x00;   // end initialization (CCE=0, INIT=0); Interrupts EIE=0, SIE=0, IE=0 :
while(!(SR&0x17)) {}      // wait for RXOK or error{LEC}
}
while((SR&0x07));         // if LEC>0 measure again

/*DEFINE MOs 2,15 :
 nr|xtd|   id|dir|dlc|txie|rxie
  +---+-----+---+---+----+----
  2|  0|0x002|  1|  8|   0|   0*/

mptr=0xef22;*mptr=0x40000; // Arbitration reg
mptr=0xef20;*mptr=0x5595;  // MsgCtrl reg
mptr=0xef26;*mptr=0x0088;  // MsgConf reg
XPOIC=0x44;  // load CAN-Module Interrupt Register
IEN = 1;    // global interrupt enable
*msg2cp=0xfaff;  // set NEWDAT,CPUUPD
*msg2cp=0x?ff;   // clear CPUUPD
*msg2cp=0xefff;  // send MO 2
while(1) {}
#pragma asm
    P4DEFR DEFR OFFC8H
    P7 DEFR OFFD0H
    PT0 DEFR 0F030H
    PWMCON0 DEFR OFF30H
    BUSCON0 DEFR OFF0CH
    MOV R5, res_add
    MOV R6, MeasCnt
    SHL R6, #1
    ADD R6, R5 ; calculate last result address(R6)
    MOV R7, BUSCON0; save BUSCON0
    MOV BUSCON0, #004AFH; set fast bus mode
    EXTR #1
    MOV PT0, #0

    Beg: XOR P7, #1
    Lp1: JNB P4.5, Lp1 ; wait for end of current low level
    Lp2: JB P4.5, Lp2 ; wait for new low level
    BSET PWMCON0.0 ; start timer
    Lp3: JNB P4.5, Lp3 ; wait for end of low level
    BCLR PWMCON0.0 ; stop timer
    MOV [R5], PT0 ; save measurement
    ADD R5, #2
    CMP R5, R6
    JMPR cc_C, Beg
    MOV BUSCON0, R7; restore BUSCON0
#pragma endasm
}

CAN Baudrate Detection