Eine erste Möglichkeit die CRC-Bits zu berechnen – siehe 2. Bild:

Die CAN Specification 2.0, Part B enthält einen Vorschlag zur Berechnung der 15 CRC-Bits:

The remainder of the polynomial division is the CRC SEQUENCE transmitted over the bus. In order to implement this function, a 15 bit shift register CRC_RG(14:0) can be used. If NXTBIT denotes the next bit of the bit stream, given by the destuffed bit sequence from
START OF FRAME until the end of the DATA FIELD, the CRC SEQUENCE is calculated as follows:

```
CRC_RG = 0;   // initialize shift register
REPEAT
CRCNXT = NXTBIT EXOR CRC_RG(14);
CRC_RG(14:1) = CRC_RG(13:0);          // shift left by
CRC_RG(0) = 0;                // 1 position
IF CRCNXT THEN
CRC_RG(14:0) = CRC_RG(14:0) EXOR (4599hex);
ENDIF
UNTIL (CRC SEQUENCE starts or there is an ERROR condition)
```

Das CAN-Generatorpolynom

$M(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + x^0 = 1\,1\,0\,0\,0\,1\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1$

reduziert sich dabei um eine Stelle – die linke grüne 1 auf: 100010110011001.

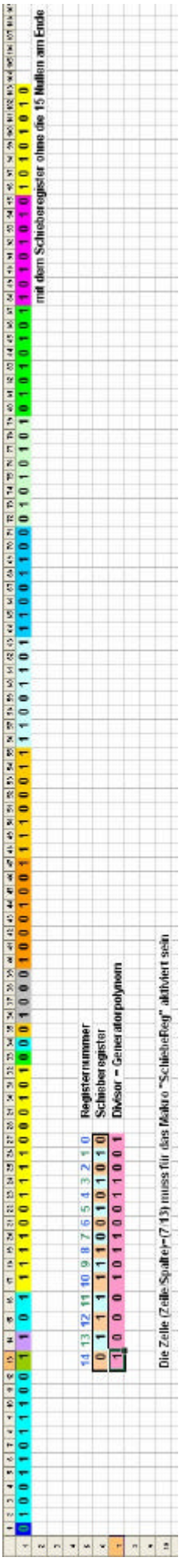
Ich habe einmal in Excel diese Methode nachgebildet. Um das Makro anzuwenden muss lediglich auf irgendeinem Tabellenblatt – das beim Aufruf des Makros aktiv sein muss – in der ersten Zeile die CAN-Bitfolge stehen – die Länge ist gleichgültig. Dann muss die Bitfolge 100010110011001 irgendwo auf dem Tabellenblatt – z.B. 6 Zeilen tiefer – eingetragen werden und – und das ist entscheidend – die Zelle mit der linken roten 1 des Tabellenblatts angeklickt, d.h. aktiviert werden. Startet man nun das Makro, erscheint über dem CAN-Generatorpolynom 100010110011001 eine Bitfolge 011111100101010, die – bis auf das Stuff-Bit – der Bitfolge 0111110100101010 des STR9-ComSticks entspricht. Die rote Null ist ein Stuff-Bit, welches man sich wegdenken muss.

Die Durchnummerierung des Registers und das Ausmahlen des Registers selbst ist Geschmacksache. Ich konnte dieser Spielerei nicht widerstehen.

Das 2. Bild zeigt das ausgemahlte Schieberegister. Daneben ist das Excel-Makro „SchiebeReg()“ abgedruckt.

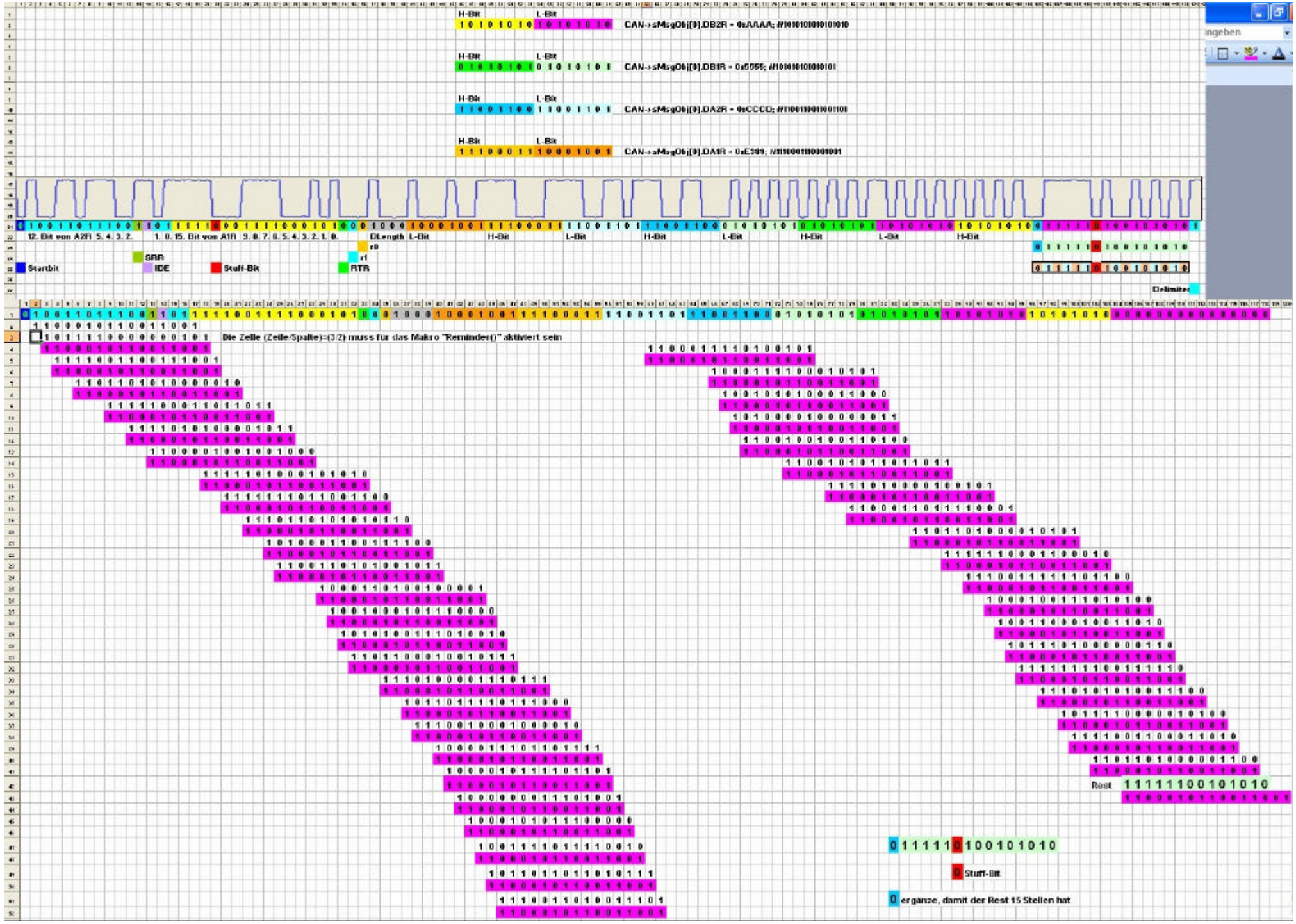
Setzt man jetzt noch an das Ende dieser CRC-Bits das Delimiter-Bit, dann ist man am Ende der im 3. Bild abgebildetet CAN-Data-Frame-Bitfolge angekommen. Das Ack-Field mit seinen 2 Bits und das End-of-Frame-Field mit seinen 7 Bits standen beim STR9-ComStick alle auf 1. In diesem Bereich werden laut CAN Specification keine Stuff-Bits mehr gesetzt.

```
Sub SchiebeReg()
   Name = ActiveSheet.Name
   Set ZL = Worksheets(Name).Cells
   Set TB = Worksheets(Name)
   TB.Select
   n_z = ActiveCell.Row
   n_s = ActiveCell.Column
   '-----wieviele Daten----------------------------------------
   txt = ZL(1, 1).Value
   k = 0
   While txt <> ""
      k = k + 1
      txt = ZL(1, k).Value
   Wend
   n_data = k - 1
   '******Divisorlaenge*********************************
   k = 0
   txt = ZL(n_z, n_s + k).Value
   While txt <> ""
      k = k + 1
      txt = ZL(n_z, n_s + k).Value
   Wend
   n_reg = k - 1
   reg_z = n_z - 1 'Registerzeile
   div_z = n_z 'Divisorzeile
   'Register initialisieren
   For i = 0 To n_reg
      ZL(reg_z, n_s + i).Value = 0
   Next i
   'Bit fuer Bit einlesen
   For i = 1 To n_data
      bit = ZL(1, i).Value 'lese i. bit
      crc = bit Xor ZL(reg_z, n_s).Value  'neues Bit xor Reg(14)
      For j = 0 To n_reg - 1 'Reg(14:1)<--Reg(13:0)
         ZL(reg_z, n_s + j).Value = ZL(reg_z, n_s + j + 1).Value
      Next j
      ZL(reg_z, n_s + n_reg).Value = 0 'Reg(0)=0
      If crc = 1 Then
         For ii = 0 To n_reg
            ZL(reg_z, n_s + ii).Value = ZL(reg_z, n_s + ii).Value Xor ZL(div_z, n_s + ii).Value
         Next ii
      End If
   Next i
End Sub
```



2. Bild

3. Bild


H-Bit
L-Bit
CAN->sMsgObj[0].DB2R = 0xAAAA; //1010101010101010
CAN->sMsgObj[0].DB1R = 0x5555; //0101010101010101
CAN->sMsgObj[0].DA2R = 0xCCCD; //1100110011001101
CAN->sMsgObj[0].DA1R = 0xE399; //1110001110011001
12. Bit von A2R 5. 4. 3. 2.
1. 0. 15. Bit von A1R 9. 8. 7. 6. 5. 4. 3. 2. 1. 0.
DLength
L-Bit
H-Bit
r0
r1
SRR
IDE
Stuff-Bit
RTR
Startbit
Die Zelle (Zelle/Spalte)=(3/2) muss für das Makro "Reminder()" aktiviert sein
Rest
0 1 1 1 1 1 0 1 0 0 1 0 1 0 1 0
0 Stuff-Bit
0 ergänze, damit der Rest 15 Stellen hat

Eine zweite Möglichkeit die CRC-Bits zu berechnen – siehe 3. Bild:

Aus den Tiefen des Internets: … Bei dem Verfahren der zyklischen Redundanzprüfung (Cyclic Redundancy Check CRC) ist die Prüfoperation im Prinzip eine Division. Betrachten wir zunächst ein Beispiel mit Dezimalzahlen. Übertragen werden soll eine große Dezimalzahl wie etwa 35628828297292. Weiterhin wählt man einen Divisor D aus. Nimmt man dafür beispielsweise den Wert 17 und führt die Division aus, so erhält man bei ganzzahliger Division (Modulo-Operator % in C) einen Rest von 8. Der Sender kann damit zu der großen Zahl als Prüfwert die 8 senden. Der Empfänger rechnet dann zur Kontrolle (35628828297292-8)%17. Bleibt bei der Division ein Rest übrig, so liegt ein Übertragungsfehler vor.

Schon an diesem einfachen Beispiel werden einige Grundeigenschaften deutlich. Zunächst einmal sind nicht alle Divisoren gleich gut geeignet für diesen Zweck. Beispielsweise wäre 10 ein schlechter Divisor, da sich dann nur Fehler an der letzten Stelle auswirken würden.

Weiterhin wird ein größerer Divisor einen besseren Schutz bieten. Allerdings muss dann entsprechend mehr Platz für den Rest vorgehalten werden. Der Rest kann maximal den Wert D-1 annehmen. Der Divisor selbst hat sinnvollerweise einen festen Wert und braucht dann nicht mehr übertragen zu werden.

Die Übertragung dieses Ansatzes auf Binärdaten beruht auf der Interpretation von Bitfolgen als Polynom. Für eine Bitfolge abcdef schreibt man:

$$M(x) = a \cdot x^5 + b \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x^1 + f \cdot x^0$$

Dabei genügt es, die Terme mit Einsen anzugeben. Für die Bitfolge 100101 erhält man dann zum Beispiel:

$$M(x) = 1 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 \quad = \quad 1 \cdot x^5 + 1 \cdot x^2 + 1 \cdot x^0 = x^5 + x^2 + x^0$$

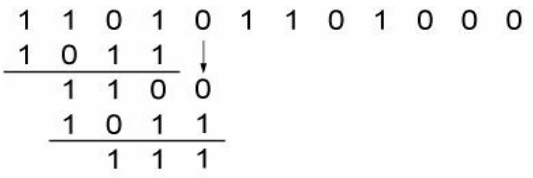
Die Prüfbits werden durch eine Division von solchen Polynomen bestimmt. CRC benutzt eine polynomiale Modulo-2-Arithmetik – denn der Rest kann ja nur 0 oder 1 je Stelle sein. In dieser Arithmetik reduziert sich die Subtraktion auf eine XOR-Operation. Die Division lässt sich – wie bei der üblichen Division – durch fortgesetzte Subtraktion ausführen: 3=12:4, denn 12-4-4-4=0

Beispiel für die zyklische Redundanzprüfung

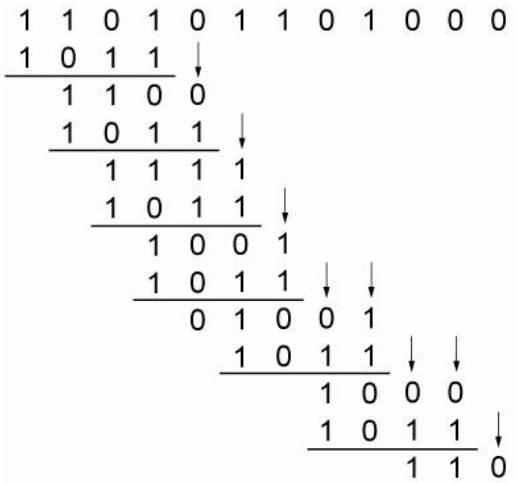
Es sei als Beispiel $M(x) = x^3 + x^1 + x^0 = x^3 + x + 1$ das Divisorpolynom und damit 1011 das zugehörige Bitmuster. Die Nachricht 110101101 ist zu übertragen. Zunächst wird sie entsprechend der höchsten Potenz im Divisor um drei Bits erweitert: 110101101000. Genau in diese zusätzlichen Bits kommen später die Prüfbits. Der erste Schritt in der Division ist dann:

```
1 1 0 1 0 1 1 0 1 0 0 0
1 0 1 1
-------
  1 1 0
```

Das Ergebnis der Division wird nicht benötigt und daher nicht notiert. Dann wird die nächste Stelle von oben übernommen und der Divisor erneut subtrahiert:



insgesamt ergibt sich:



Bei der Division bleibt ein Rest von 110. Zieht man diesen von der erweiterten Zahl ab, so ist die neue Zahl ohne Rest durch den Divisor teilbar. Da Subtraktion einer XOR-Operation entspricht, erhält man 110101101110. Diese um die drei Prüfbits erweiterte Bitfolge wird gesendet. Der Empfänger führt ebenfalls die Division durch. Falls dabei ein Rest übrig bleibt, liegt ein Übertragungsfehler vor.

Wie bei dem Beispiel mit den Dezimalzahlen gibt es auch für die Polynomdivision mehr oder weniger gut geeignete Divisorpolynome. CAN_CRC arbeitet mit einem 15 Bit Generatorpolynom:

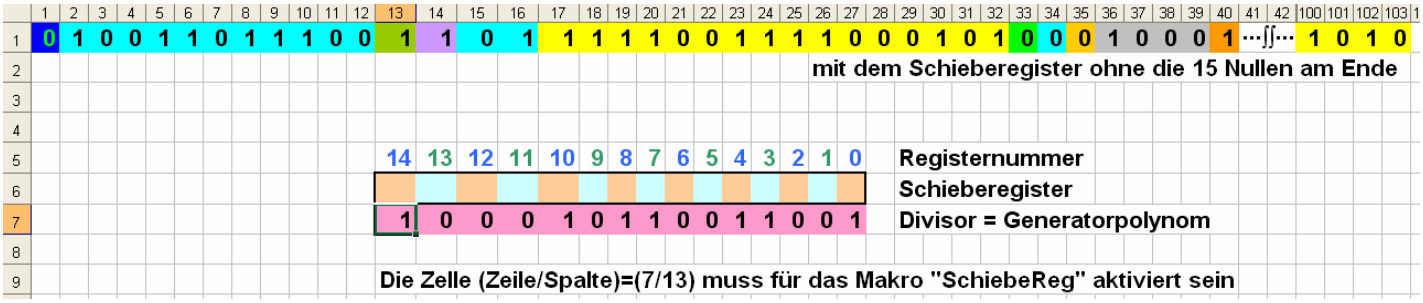
$$M(x) = x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + x^{0} = 1100010110011001$$

Da die CAN-Data-Frame-Bitfolge lang ist, habe ich mir ein Excel-Makro geschrieben, das die Restwertbildung durchführt. Eine Anwendung des Excel-Makros „Reminder()“ ist im 3. Bild illustriert.

```
Sub Reminder()
   Name = ActiveSheet.Name
   Set ZL = Worksheets(Name).Cells
   Set TB = Worksheets(Name)
   TB.Select
   n_z = ActiveCell.Row
   n_s = ActiveCell.Column
   'Datenlaenge
   txt = ZL(1, 1).Value
   k = 0
   While txt <> ""
      k = k + 1
      txt = ZL(1, k).Value
   Wend
   n_data = k - 1
```

```
    'Generatorlaenge
    n1_z = n_z
    n1_s = n_s
    txt = ZL(n1_z - 1, n1_s).Value
    k = 0
    While txt <> ""
       k = k + 1
       n1_s = n1_s + 1
       txt = ZL(n1_z - 1, n1_s).Value
    Wend
    n_potenz = k - 1
    no_1 = True
    t_s = 0
    While t_s < n_data
       t_s = 0
       no_1 = True
       For i = n_s To n_s + n_potenz
          wert = ZL(n_z - 1, i).Value Xor ZL(n_z - 2, i).Value
          If no_1 And (wert = 0) Then
             k_s = i + 1
          Else
             no_1 = False
             If t_s = 0 Then
                t_s = i 'hier beginnt der reminder
             End If
             ZL(n_z, i).Value = wert
          End If
       Next i
       r_s = n_s + n_potenz + 1
       j_s = n_s
       For i = k_s To k_s + n_potenz 'Divisor erneut schreiben
          ZL(n_z + 1, i).Value = ZL(n_z - 1, j_s).Value
          ZL(n_z + 1, i).Interior.ColorIndex = 7
          j_s = j_s + 1
       Next i
       For i = r_s To k_s + n_potenz 'weitere Ziffern aus dem Dividenden
          ZL(n_z, i).Value = ZL(1, i).Value
       Next i
       If k_s + n_potenz >= n_data Then
          t_s = n_data
       End If
       n_z = n_z + 2 'Beginn der Division: Zeile
       If t_s = 0 Then
          n_s = r_s 'Beginn der Division: Spalte
       Else
          n_s = t_s 'Beginn der Division: Spalte
       End If
    Wend
End Sub
```

Voraussetzungen für die Excel-Makros „Reminder()“ und „SchiebeReg()“ im Überblick:



CANsend.c war wie folgt geändert

```
#define _MAIN_H_
void SCU_Configuration (void);
void CAN_Configuration (void);
// Include Files
#include "STR91x.h"
#include "main.h"
#define CAN_CMR_WRRD          0x0080
#define CAN_CMR_MASK          0x0040
#define CAN_CMR_ARB           0x0020
#define CAN_CMR_CONTROL       0x0010
#define CAN_CMR_CLRINTPND     0x0008
#define CAN_CMR_TXRQSTNEWDAT  0x0004
#define CAN_CMR_DATAA         0x0002
#define CAN_CMR_DATAB         0x0001
#define CAN_M2R_MXTD          0x8000
#define CAN_M2R_MDIR          0x4000
/* IFn / Arbitration 2 register*/
#define CAN_A2R_MSGVAL        0x8000
#define CAN_A2R_XTD           0x4000
#define CAN_A2R_DIR           0x2000
/* IFn / Message Control register*/
#define CAN_MCR_NEWDAT        0x8000
#define CAN_MCR_MSGLST        0x4000
#define CAN_MCR_INTPND        0x2000
#define CAN_MCR_UMASK         0x1000
#define CAN_MCR_TXIE          0x0800
#define CAN_MCR_RXIE          0x0400
```

```
#define CAN_MCR_RMTEN         0x0200
#define CAN_MCR_TXRQST        0x0100
#define CAN_MCR_EOB           0x0080

/* Main Program */
int main(void)
{
u32 i;
SCU_Configuration();
CAN_Configuration();
while (1)
  {
    CAN->sMsgObj[0].DA1R = 0xE389;
    CAN->sMsgObj[0].DA2R = 0xCCCD;
    CAN->sMsgObj[0].DB1R = 0x5555;
    CAN->sMsgObj[0].DB2R = 0xAAAA;
    CAN->sMsgObj[0].CRR = 1;
  }
}

void SCU_Configuration (void)
{
  // Initialise STR912 CPU
  // All SCU-PCGR1 peripherals clocks are disabled by default in START912.S
  // Do not clear any bits in SCU_PCGR0 as this is the SRAM, VIC etc!
  SCU->PCGR1 |= (SCU->PCGR1 & ~0x00080000) | 0x00080000 ;    /* Clock for GPIO5 is b19 */
  SCU->PRR1 &= ~0x00080000 ;                                /* Force reset */
  SCU->PRR1 |= 0x00080000 ;                                 /* Release reset */
  SCU->GPIOOUT[5] = 0x0008 ;                                /* P5.0 = alt input 1, P5.1 = alt output 2 */
  SCU->GPIOIN[5] = 0x03 ;                                  /* P5.0 = alt input 1 */
  SCU->GPIOTYPE[5] = 0x0000 ;                               /* All output pins are push-pull */
  GPIO5->DDR = 0x02 ;                                      /* No simple IO used */
  /* Initialise CAN Module */
  /* Enable CAN peripheral clocks */
  SCU->PCGR1 |= (SCU->PCGR1 & ~0x00000400) | 0x00000400 ;    /* Clock for CAN is b10 */
  SCU->PRR1 &= ~0x00000400 ;                                /* Force reset */
  SCU->PRR1 |= 0x00000400 ;                                 /* Release reset */
  //SCU->CLKCNTR = (SCU->CLKCNTR & ~0x00000180) | 0x00000100 ; //PCLK = RCLK/4
  //SCU->CLKCNTR = (SCU->CLKCNTR & ~0x00000180) | 0x00000180 ; //PCLK = RCLK/8
  //SCU->CLKCNTR = (SCU->CLKCNTR & ~0x0000019C) | 0x00000184 ; //PCLK = RCLK/8 u. RCLK=FMASTR/2
  //SCU->CLKCNTR = (SCU->CLKCNTR & ~0x0000019C) | 0x00000188 ; //PCLK = RCLK/8 u. RCLK=FMASTR/4
  SCU->CLKCNTR = (SCU->CLKCNTR & ~0x0000019C) | 0x0000018C ; //PCLK = RCLK/8 u. RCLK=FMASTR/8
  //SCU->CLKCNTR = (SCU->CLKCNTR & ~0x0000019C) | 0x00000190 ; //PCLK = RCLK/8 u. RCLK=FMASTR/16
  //SCU->CLKCNTR = (SCU->CLKCNTR & ~0x0000019C) | 0x00000194 ; //PCLK = RCLK/8 u. RCLK=FMASTR/1024
}
void CAN_Configuration (void)
{
  /* Begin1: Initialise CAN0 for full (part2.0B) CAN operation */
  CAN->CR = 0x00000001; //Init=1; init and config change enable
  CAN->CR = 0x000000C1; //11000001 Test=1, CCE=1 (Configuration Change Enable), Init=1
```

```
  CAN->BRPR = 0x00000000;/*Baud Rate Prescaler Register; Clear the extended BRP */
  CAN->BTR = 0x0000755F;   //111010101011111 Set bit rate*/
  CAN->CR = 0x00000000; /* End CAN initialisation mode. End1*/
  CAN->sMsgObj[0].CMR = 0x00F3; //11110011 IF1 (Interface Register 1): Command Mask Register = CAN_IF1_CMR
  /* Reset the MSGVAL bit to enable ID, DIR and DLC=Data Length Code change */
  CAN->sMsgObj[0].A2R = 0;
  CAN->sMsgObj[0].A1R = 0xF3C5;//(15.Bit)1111001111000101(0.Bit)
  CAN->sMsgObj[0].A2R = 0x1371;   //(12.Bit)1001101110001(0.Bit)
  /* Setup type information */
  CAN->sMsgObj[0].A2R |= CAN_A2R_DIR|CAN_A2R_XTD;
  /* Setup data bytes */
  CAN->sMsgObj[0].MCR = 0x8980 | 8;  //1000100110000000 or (1000 for 8 Byte) ; DataLengthCode
  CAN->sMsgObj[0].A2R |= CAN_A2R_MSGVAL;//ohne das geht nichts
}
```

main.c aus ADC sah folgendermaßen aus:

```
#include "defines.h"
#define global extern   /* to declare external variables and functions      */
#include "91x_lib.h"
#include "main.h"
#define GPIO_Alt1 0x01
#include <stdio.h>
#include <string.h>
/* Private variables ---------------------------------------------------------*/
  UART_InitTypeDef UART_InitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;
  TIM_InitTypeDef   TIM_InitStructure;
  ADC_InitTypeDef   ADC_InitStructure;
/* Private function prototypes -----------------------------------------------*/
  void SCU_Configuration(void);
  void GPIO_Configuration(void);
  void ADC_Configuration(void);
  void UART1_Configuration(void);
  static void Delay(u32 nCount);

int main (void)
{
  u16 Conversion_Value = 0;

  /* Configure the system clocks */
  SCU_Configuration();
  /* Configure the GPIOs */
  GPIO_Configuration();
  /* Configure and start the ADC */
  ADC_Configuration();
  UART1_Configuration();
  /* endless loop */
  while (1)
```

```
  {
    /* Wait until conversion completion */
    while(ADC_GetFlagStatus(ADC_FLAG_ECV) == RESET);
    /* Get the conversion value */
    Conversion_Value = ADC_GetConversionValue(ADC_Channel_6);
     /* Clear the end of conversion flag */
    ADC_ClearFlag(ADC_FLAG_ECV);

    //send u16-data
    UART_SendData(UART1,(u8)(Conversion_Value >> 8));
    while(UART_GetFlagStatus(UART1, UART_FLAG_TxFIFOFull) != RESET);
    UART_SendData(UART1,(u8)Conversion_Value);
    while(UART_GetFlagStatus(UART1, UART_FLAG_TxFIFOFull) != RESET);

    /* Insert delay to help SerialPort*/
    Delay(0xF);
  }
}

void SCU_Configuration(void)
{
  SCU_MCLKSourceConfig(SCU_MCLK_OSC);
  SCU_PLLFactorsConfig(192,25,2);           /* PLL = 96 MHz */
  SCU_PLLCmd(ENABLE);                       /* PLL Enabled  */
  SCU_MCLKSourceConfig(SCU_MCLK_PLL);       /* MCLK = PLL   */
  FMI_BankRemapConfig(4, 2, 0, 0x80000);    /* Set Flash banks size & address */
  FMI_Config(FMI_READ_WAIT_STATE_2, FMI_WRITE_WAIT_STATE_0, FMI_PWD_ENABLE,\
  FMI_LVD_ENABLE, FMI_FREQ_HIGH); /* FMI Waite States */
  /* Set the PCLK Clock to MCLK/2 */
  SCU_PCLKDivisorConfig(SCU_PCLK_Div2);
   /* Enable the UART1 Clock */
  SCU_APBPeriphClockConfig(__UART1, ENABLE);
  SCU_APBPeriphClockConfig(__ADC, ENABLE);    /* Enable the clock for the ADC */
  /* Enable the clock for TIM0 and TIM1 */
  SCU_APBPeriphClockConfig(__TIM01, ENABLE);
  SCU_APBPeriphReset(__TIM01, DISABLE);
  SCU_APBPeriphClockConfig(__TIM23, ENABLE);
  SCU_APBPeriphReset(__TIM23, DISABLE);
  /* Enable the GPIO3 Clock */
  SCU_APBPeriphClockConfig(__GPIO3, ENABLE);
  SCU_APBPeriphClockConfig(__GPIO4, ENABLE);  /* Enable the clock for the GPIO4 */
  /* Enable the __GPIO8 */
  SCU_APBPeriphClockConfig(__GPIO8 ,ENABLE);
  /* Enable the __GPIO9 */
  SCU_APBPeriphClockConfig(__GPIO9 ,ENABLE);
}

/* GPIO Configuration ----------------------------------------------------*/
void GPIO_Configuration(void)
{
```

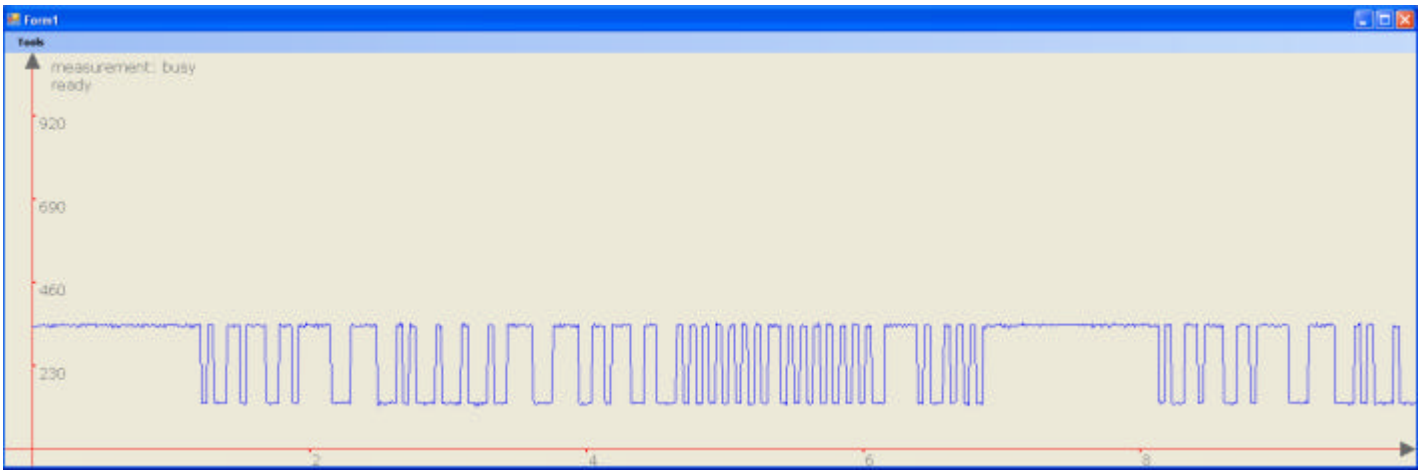
```
  GPIO_DeInit(GPIO3);
  GPIO_DeInit(GPIO4);                    /* GPIO4 Deinitialization */
  GPIO_DeInit(GPIO9);
  /* IOs */
  GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
  GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull ;
  GPIO_Init (GPIO4, &GPIO_InitStructure);
  /* onboard LED */
  GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull ;
  GPIO_Init (GPIO9, &GPIO_InitStructure);
  GPIO_WriteBit(GPIO9, GPIO_Pin_0, Bit_RESET);
  /* ADC */
  /* Configure the GPIO4 pin 6 as analog input */
  GPIO_ANAPinConfig(GPIO_ANAChannel6, ENABLE);
  /* configure UART1_Rx pin GPIO3.2*/
  GPIO_InitStructure.GPIO_Direction = GPIO_PinInput;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Type = GPIO_Type_OpenCollector ;
  GPIO_InitStructure.GPIO_IPConnected = GPIO_IPConnected_Enable;
  GPIO_InitStructure.GPIO_Alternate = GPIO_Alt1  ;
  GPIO_Init (GPIO3, &GPIO_InitStructure);
  /*Gonfigure UART1_Tx pin GPIO3.3*/
  GPIO_InitStructure.GPIO_Direction = GPIO_PinInput;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull ;
  GPIO_InitStructure.GPIO_Alternate = GPIO_OutputAlt2  ;
  GPIO_Init (GPIO3, &GPIO_InitStructure);
}

void ADC_Configuration(void)
{
   ADC_DeInit();                         /* ADC Deinitialization */
    /* ADC Structure Initialization */
  ADC_StructInit(&ADC_InitStructure);
  /* Configure the ADC in continuous mode conversion */
  ADC_InitStructure.ADC_Channel_6_Mode = ADC_NoThreshold_Conversion;
  ADC_InitStructure.ADC_Select_Channel = ADC_Channel_6;
  ADC_InitStructure.ADC_Scan_Mode = DISABLE;
  ADC_InitStructure.ADC_Conversion_Mode = ADC_Continuous_Mode;
  /* Enable the ADC */
  ADC_Cmd(ENABLE);
  /* Prescaler config */
  ADC_PrescalerConfig(0x0);
  /* Configure the ADC */
  ADC_Init(&ADC_InitStructure);
  /* Start the conversion */
  ADC_ConversionCmd(ADC_Conversion_Start);
```

```
}

/* UART1 configuration ------------------------------------------------------*/
void UART1_Configuration(void)
 {
  UART_InitStructure.UART_WordLength = UART_WordLength_8D;
  UART_InitStructure.UART_StopBits = UART_StopBits_1;
  UART_InitStructure.UART_Parity = UART_Parity_No ;
  UART_InitStructure.UART_BaudRate = 460800; //230400; 115200;
  UART_InitStructure.UART_HardwareFlowControl = UART_HardwareFlowControl_None;
  UART_InitStructure.UART_Mode = UART_Mode_Tx_Rx;
  UART_InitStructure.UART_FIFO = UART_FIFO_Enable;
  UART_InitStructure.UART_TxFIFOLevel = UART_FIFOLevel_1_2; /* FIFO size 16 bytes, FIFO level 8 bytes */
  UART_InitStructure.UART_RxFIFOLevel = UART_FIFOLevel_1_2; /* FIFO size 16 bytes, FIFO level 8 bytes */
  UART_DeInit(UART1);
  UART_Init(UART1, &UART_InitStructure);
  /* Enable the UART0 */
  UART_Cmd(UART1, ENABLE);
}

static void Delay(u32 nCount)
{
  u32 j = 0;
  for(j = nCount; j != 0; j--);
}
/****************************************** EOF ******************************************/
```

Im Graphikprogramm der Datei Messen.pdf (siehe oben) war der SerialPort wie folgt eingestellt:

```
private void OpenPort()
      {
         //_serialPort = new SerialPort("COM3", 115200, Parity.None, 8, StopBits.One);
         //_serialPort = new SerialPort("COM3", 230400, Parity.None, 8, StopBits.One);
         _serialPort = new SerialPort("COM3", 460800, Parity.None, 8, StopBits.One);
         if (_serialPort != null)
         {
            _serialPort.ReadTimeout = 1;
            _serialPort.Open();
            _serialPort.DiscardInBuffer();
            com_is_open = true;
         }
      }
```

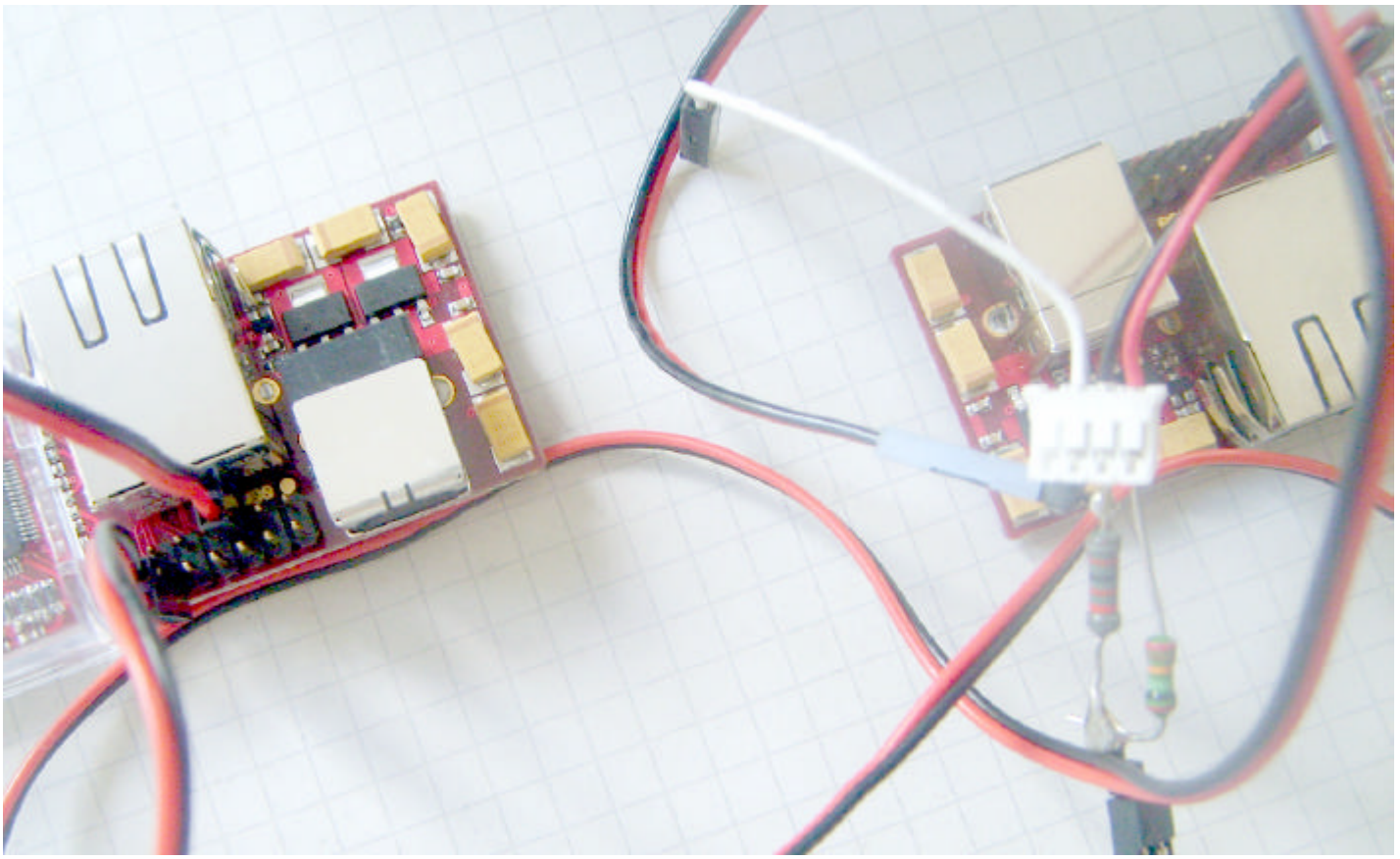
Damit erhält man mit dem Messprogramm folgende CAN-Data-Frame-Bitfolge:



Mit dem Photo-Editor habe ich mir die CAN-Data-Frame-Bitfolge ausgeschnitten, in Excel eingefügt, passend gestreckt und die einzelnen Datenbits als 0 oder 1 darunter geschrieben.

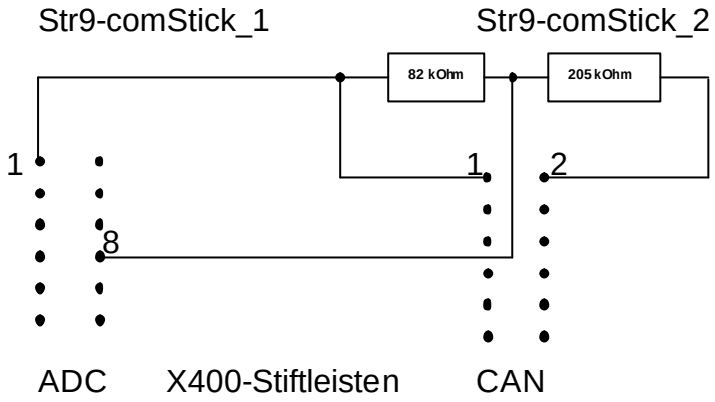
Mit CAN->BTR = 0x397F;//TSeg2:011 TSeg1:1001 SJW:01 BRP:111111 kann man die Bitrate weiter verlangsamen; dann allerdings passt die CAN-Data-Frame-Bitfolge nicht mehr auf eine Bildschirmseite.

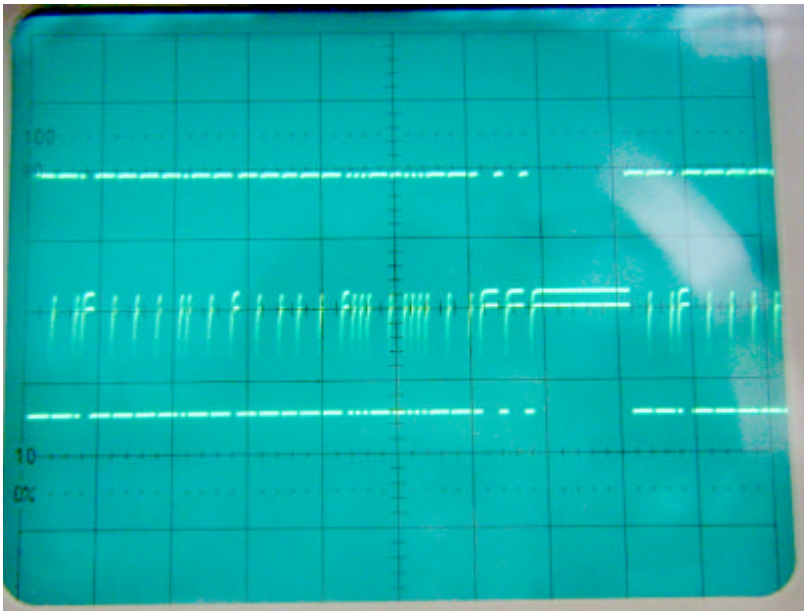
Zum Schluss noch die Beschaltung der beiden STR9-ComSticks:



Sie sehen links den AD-Wandler und rechts den CAN-Sender.

Es kommt nur auf das Verhältnis der Widerstände an, damit der AD-Wandler nicht übersteuert wird; andere Werte tun´s auch.





Die Dateien Messen_02_Teil_A.pdf und Messen_02_Teil_B.pdf bilden eine Einheit.
Für www.mikrocontroller.net waren sie in Form einer Datei leider zu groß.

Viel Spaß: Edgar Marx

edgarmarx@t-online.de