Application Note of UTI

1. INTRODUCTION

This Application Note describes some typical applications of the UTI on the measurement of some physical quantities, such as temperature, position, displacement, humidity, pressure, acceleration, etc.. The UTI is a complete front end for many types of passive sensors, such as resistive, resistive-bridge and capacitive sensors. The UTI converts low-level signals from a sensor to a period-modulated microcontroller-compatible signal. By the application of continuous auto-calibration a high accuracy of up to 15 bits is obtained. Full data of the UTI is available in the UTI data sheet of SMARTEC B.V. and should be consulted in conjunction with this Application Note.

2. THREE-SIGNAL TECHNIQUE

In order to achieve a high accurate and stable measurement for a measurand, the UTI performs the measurement of a reference signal and a constant part (including offset) in exactly same way as the measurand signal during two additional phases. This is called the three-signal technique.

The three-signal technique is a technique to eliminate the effects of unknown offset and unknown gain in a linear system. In order to apply this technique, in addition to the measurement of the sensor signal, two reference signals are required to be measured in an identical way. Suppose a system has a linear transfer function of

$$M_i = kE_i + M_{off} . \tag{1}$$

The measured three signals are

$$\begin{aligned} M_{off} &= M_{off} \\ M_{ref} &= kE_{ref} + M_{off} \; . \\ M_x &= kE_x + M_{off} \end{aligned} \tag{2}$$

Then the measuring result is the ratio

$$M = \frac{M_x - M_{off}}{M_{ref} - M_{off}} = \frac{E_x}{E_{ref}} . \tag{3}$$

When the system is linear, then in this ratio the influence of the unknown offset M_{off} and the unknown gain k of the measurement system is eliminated.

According to the working principle of the three-signal technique, a memory is required to implement this technique. A better solution for this is to use a microcontroller which can perform the data storage and some calculations as well as the measurement of the period-modulated signals. Such a system combining a sensing element (sensor), a signal-processing circuit, such as UTI, and a microcontroller is called the microcontroller-based smart sensor system. Based on such a sensor system concept, some typical applications of the UTI are described in the following section.

3. THE APPLICATIONS

3.1. The measurement of temperature using the Pt100 (Pt1000)

The platinum resistor, Pt100 (or Pt1000), is one of often-used temperature sensor. In the UTI, two modes (mode 5 and 7) are designed to measure the platinum resistor signals. Here, the mode 5 is employed to demonstrate the application of the UTI in the measurement of the Pt100 signal.
Figure 1 shows a measurement setup of temperature by using the Pt100.

Both resistors R_{ref} and R_{Pt100} are connected to the UTI in a 4-wire connection, then the effect of lead resistances is completely eliminated. Resistor R_1 is used to set the current through the Pt100 and R_{ref}. The operating situation of the UTI is set by connecting the Sel1 ~ Sel4, SF, PD and TEST to the ground or the power supply of 5 V. The UTI is set in the slow mode (SF=0). The operating situation of the UTI can be programmingly set by a microcontroller also. The period-modulated output signal from the UTI is shown in Figure 2.

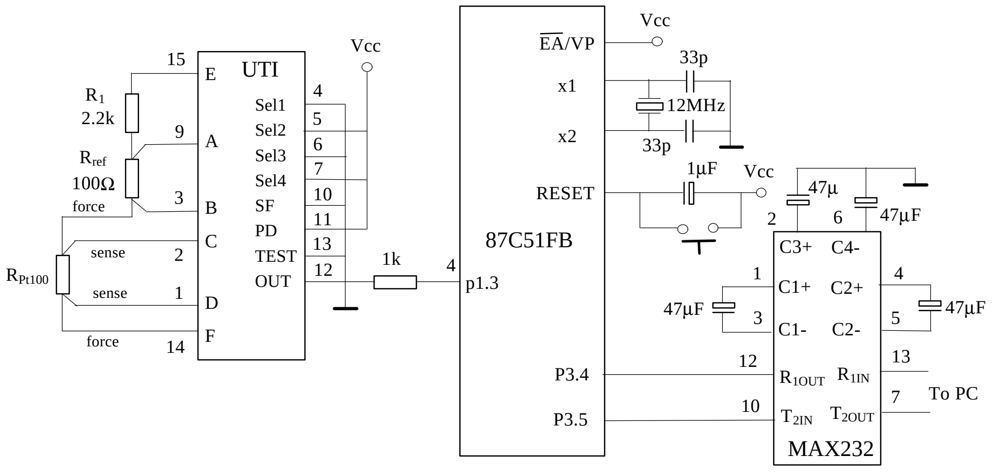


Figure 1 A measurement setup of temperature by using a Pt100.

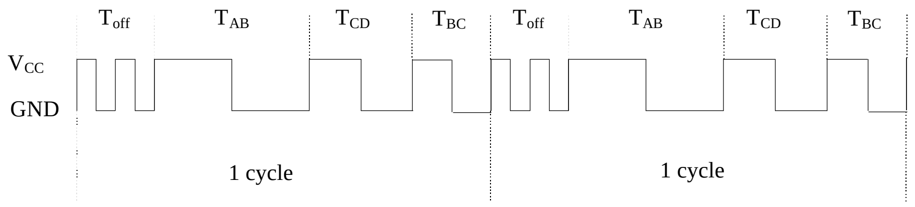


Figure 2 The output signal of the UTI.

A microcontroller (Intel 87C51FB) is used to measure the period-modulated signals from the UTI, to process the measured data and to output digital data to a personal computer via a serial communication interface (RS232). In section 4, the software in the microcontroller is described in detail. The RS232 interfacing chip offers serial communication between the microcontroller and, for instance a personal computer. A 12.0000 MHz crystal is used for the on-chip oscillator of the microcontroller. The power supply of 5 V (±10%) is used for the UTI, the µC and the RS232 interfacing.

The counting function the microcontroller (87C51FB) is used to measure the periods of the output signal from the UTI. Then the periods, T_{off}, T_{AB}, T_{CD} and T_{BC}, are quantized in numbers, N_{off}, N_{AB}, N_{CD} and N_{BC} respectively by the microcontroller. Using the three-signal technique, the measurand is represented by

$$R_{Pt100} = \frac{N_{CD} - N_{off}}{N_{AB} - N_{off}} \cdot R_{ref} . \tag{4}$$

This formula is performed by the microcontroller. It is shown that the measured accuracy of R_{Pt100} is directly dependent on the accuracy of the reference R_{ref}.

3.2. The measurement of temperature using the thermistor

The thermistor is often-used to measure the temperature. In the UTI, two modes (mode 6 and 8) are designed to measure the thermistor signals. Here, the mode 6 is employed to demonstrate the application of the UTI in the measurement of the Thermistor signal. Figure 3 shows a measurement setup for the temperature by using the thermistor.

Both resistors R_{ref} and R_{Ther} are connected to the UTI in a 4-wire connection, then the effect of lead resistances is completely eliminated. The operating situation of the UTI is set by connecting the Sel1 ~ Sel4, SF, PD and TEST to the ground or the power supply of 5 V. The UTI is set in the slow mode (SF=0). The operating situation of the UTI can be programmingly set by a microcontroller also. The period-modulated output signal from the UTI for this mode is same as that shown in Figure 2.

For this application, the measurand is represented by

$$R_{Ther} = \frac{N_{CD} - N_{off}}{N_{AB} - N_{off}} \cdot R_{ref} . \tag{5}$$

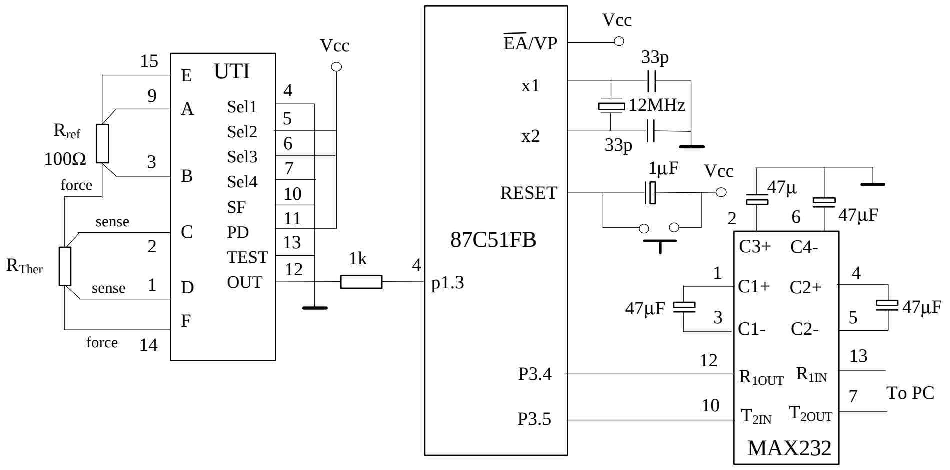


Figure 3 A measurement setup of temperature by using a Thermistor.

3.3. The measurement of the resistive bridge signal

The resistive bridges are often-used to measure the pressure, acceleration, force. In the UTI, six modes (modes 9 ~ 14) are designed to measure the resistive-bridge signals. Here, the mode 9 is employed to demonstrate the application of the UTI in the measurement of the resistive-bridge signal. In this mode, the maximum bridge imbalance of ±4% can be accurately measured.

Figure 4 shows a measurement setup for the resistive-bridge signals.

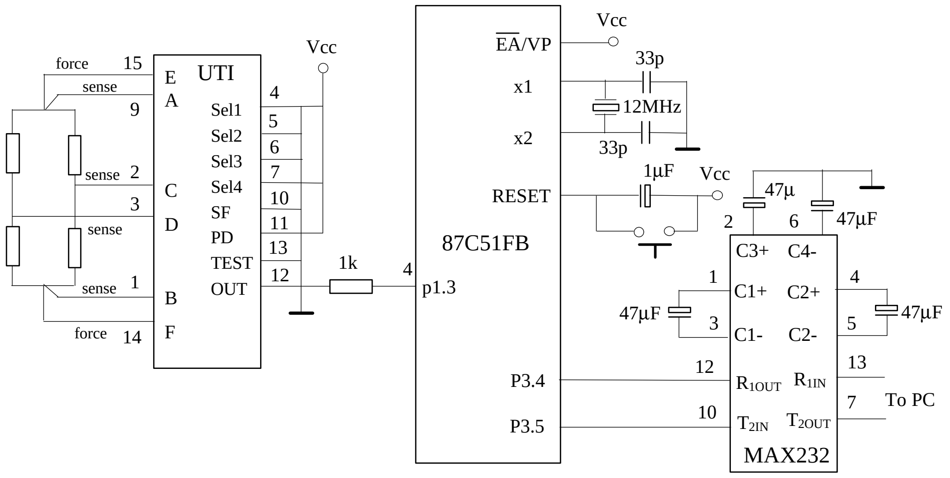


Figure 4 A measurement setup for the resistive-bridge signal.

The executing terminals of the resistive bridge are connected to the UTI in a 4-wire connection, then the effect of lead resistances is completely eliminated. The period-modulated output signal from the UTI is shown in Figure 5.

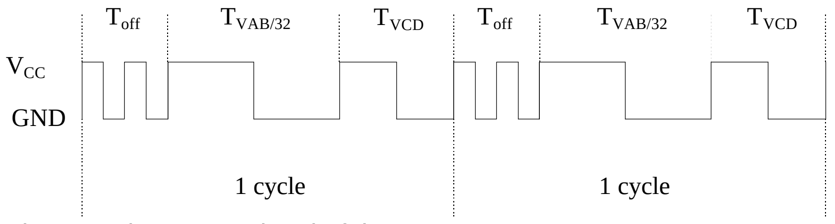


Figure 5 The output signal of the UTI.

The measured relative imbalance of the resistive bridge is represented by

$$\frac{V_{CD}}{V_{AB}} = \frac{1}{32} \cdot \frac{N_{VCD} - N_{off}}{N_{VAB/32} - N_{off}}. \quad (6)$$

3.4. The measurement of the potentiometric signal

Figure 6 shows a measurement setup for three resistive potentiometers by using mode 15 of the UTI. And Figure 7 shows the output signal of the UTI.

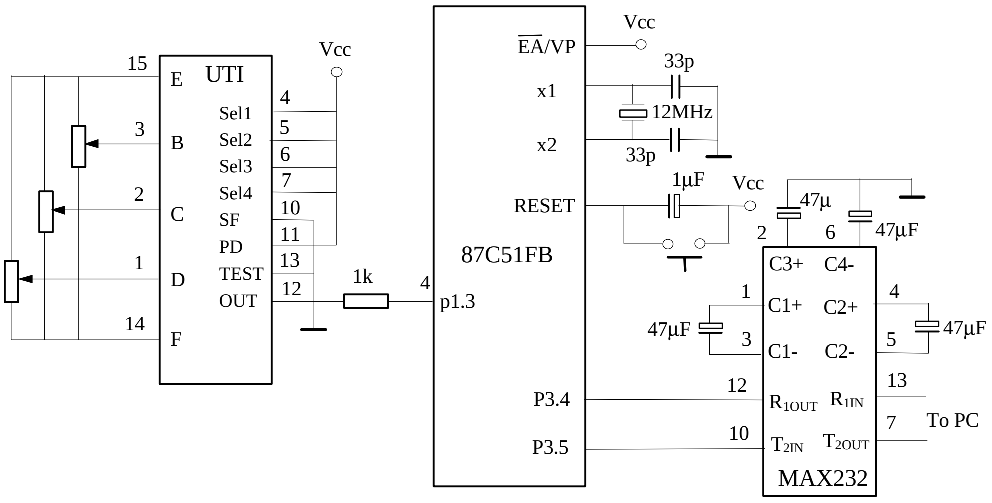


Figure 6 A measurement setup for the resistive potentiometers.

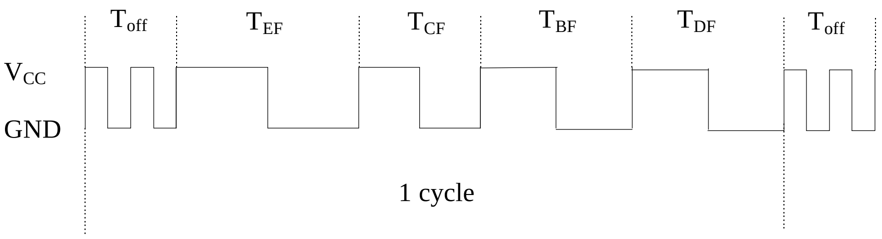


Figure 7 The output signal of the UTI

When only one potentiometer is used, for instance, its sliding terminal is connected to node B, the nodes C and D should be connected to node F.

3.5. The measurement of the capacitance signal

There are five modes (modes 0 ~ 4) for the capacitance measurement in the UTI. In this section, the mode 1 is employed to demonstrate the application of the UTI on the measurement of the small capacitance.

Figure 8 shows a measurement setup for the small capacitance by using mode 1 of the UTI. And Figure 9 shows the output signal of the UTI.

Using the three-signal technique, the measurand cab be represented by

$$\frac{C_x - C_0}{C_{ref} - C_0} = \frac{N_{AD} - N_{AB}}{N_{AC} - N_{AB}} . \tag{7}$$

When no capacitor is connected between pin A and pin B, the formula (7) is written as:

$$\frac{C_x}{C_{ref}} = \frac{N_{AD} - N_{AB}}{N_{AC} - N_{AB}} . \tag{8}$$

It is shown that when C_{ref} is accurately known, C_x can be accurately measured.

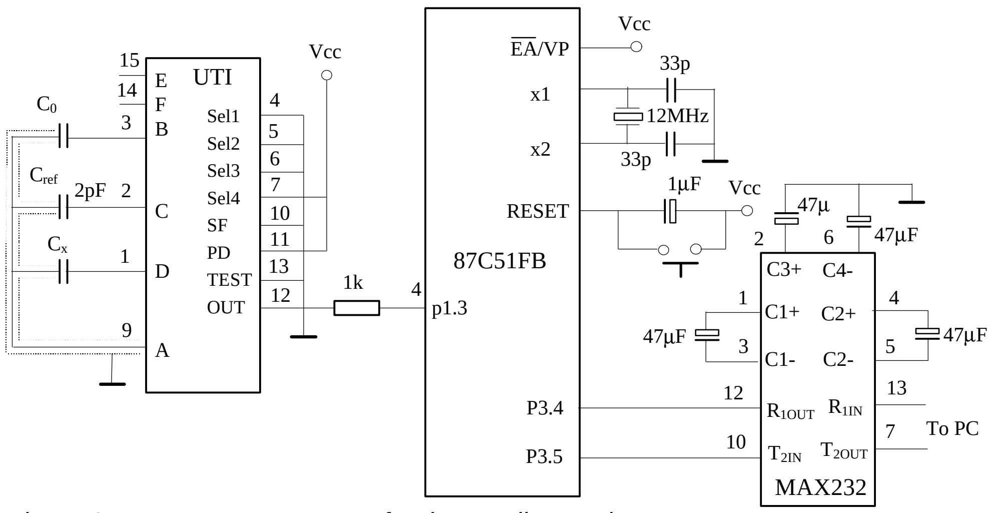


Figure 8 A measurement setup for the small capacitance.

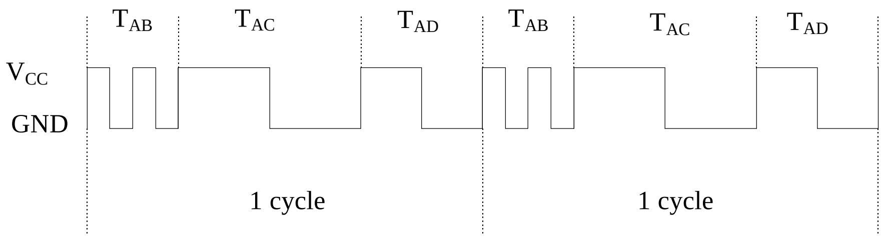


Figure 9 The output signal of the UTI.

Attention: In capacitive modes of the UTI, the node A is very sensitive for the interference. The coaxial cable wire has to be used to connect the node A with measure capacitors.

3.6. The measurement of the capacitance Humidity signal

The mode 4 of the UTI can be used to measure a relatively large value of capacitance up to 300 pF. The capacitor value of some capacitive humidity sensor is just in this range, for instance, Smartec Capacitive Humidity sensor has a maximum capacitor value of 270 pF. Figure 10 shows a measurement setup for the capacitive humidity sensor by using mode 4 of the UTI. The output signal of the UTI is same that shows Figure 9.

The measured capacitance is represented by formula (8).

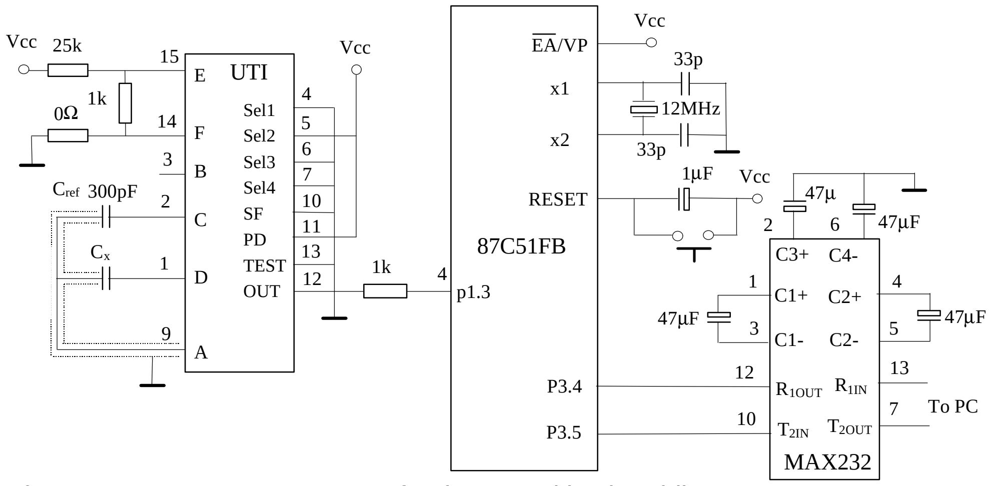


Figure 10 A measurement setup for the capacitive humidity sensor.

3.7. Multiple capacitance measurement

The multiple capacitances are required to be measured accurately in some applications, for instance, multiple capacitive sensors.

Combining with an external multiplexer, the mode 3 of the UTI can measure multiple capacitances. Figure 11 shows a setup for the measurement of multiple capacitances by using mode 3 of the UTI. The TEST of the UTI is controlled by the microcontroller. When TEST=1, the measurement range for the capacitance is 12 pF and when TEST=0 the measurement range for the capacitance is 2 pF. MUX is a multiplexer with nine outputs, and two control inputs.

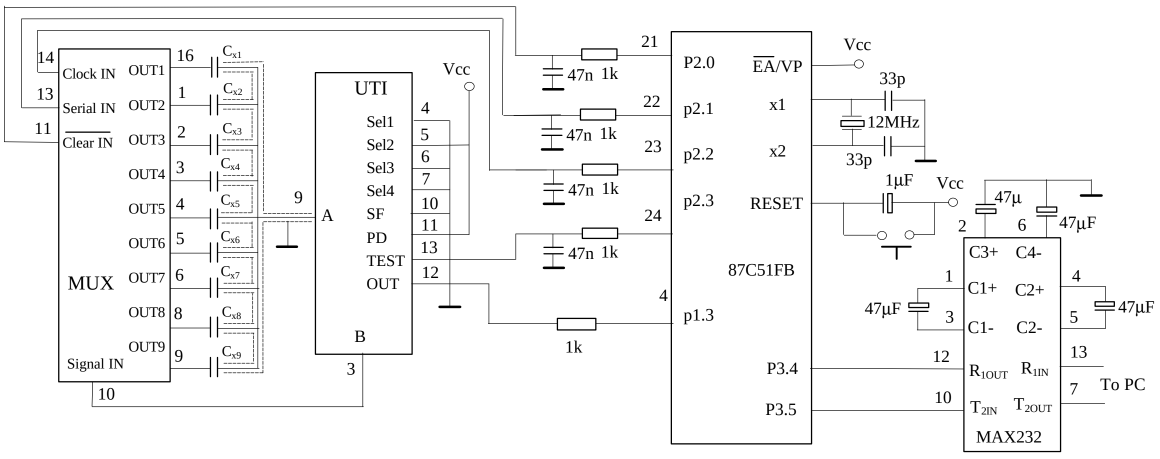


Figure 11 A measurement setup for the multiple capacitances.

3.8. Multiple-channel signal measurement system

Using Power Down function of the UTI, it is very easy to built up a multiple-channel signal measurement system, because the power down is effective (PD=0) the output impedance of the UTI is very high.

As an example, Figure 12 shows a setup for the measurement of multiple channel signal, Pt100, thermistor, capacitance and resistive bridge. The PD pins of UTIs are controlled by the microcontroller. When a channel signal is being measured, the PD of this UTI is set to “1” and others must be set to “0”.

3.9. DC voltage signal measurement system

The DC voltage signals can be measured by using capacitance measurement functions of the UTI. As an example, Figure 13(a) shows a schematic diagram circuit for the measurement of voltage signals using the mode C23 of the UTI. In this diagram, V_x denotes an unknown voltage (measurand), V_{ref} is a known reference voltage. The capacitor C_s and switches S_{x1}, S_{x2}, S_{ref1}, S_{ref2} sample the voltage signals V_x and V_{ref} alternatively and convert the voltage signals into charges alternatively. The UTI will convert the charges into the periods of the output of the UTI. Figure 13(b) shows the output signals of the UTI. Periods T_{off}, T_{ref} and T_x are corresponding to the measurements of original offset of UTI, V_{ref} and V_x, respectively.

Using the three-signal technique, the measured DC voltage is represented by

$$V_x = \frac{T_x - T_{off}}{T_{ref} - T_{off}} V_{ref} . \tag{9}$$

Formula (9) shows that the values and inaccuracies of the capacitance C_s and voltage power supply V_{CC}, and the error of the linear transfer coefficient of the UTI do not affect the measured result because of the applied three-signal technique.

The measurement principle described above is also suitable for the mode of C25, C12, CMUX and C300.

In order to guarantee that the UTI works in its optimum linear measurement range, the measurement range for the voltage signal $V_{x,range}$ and the value of the sampling capacitor C_s are determined by the following relationship:

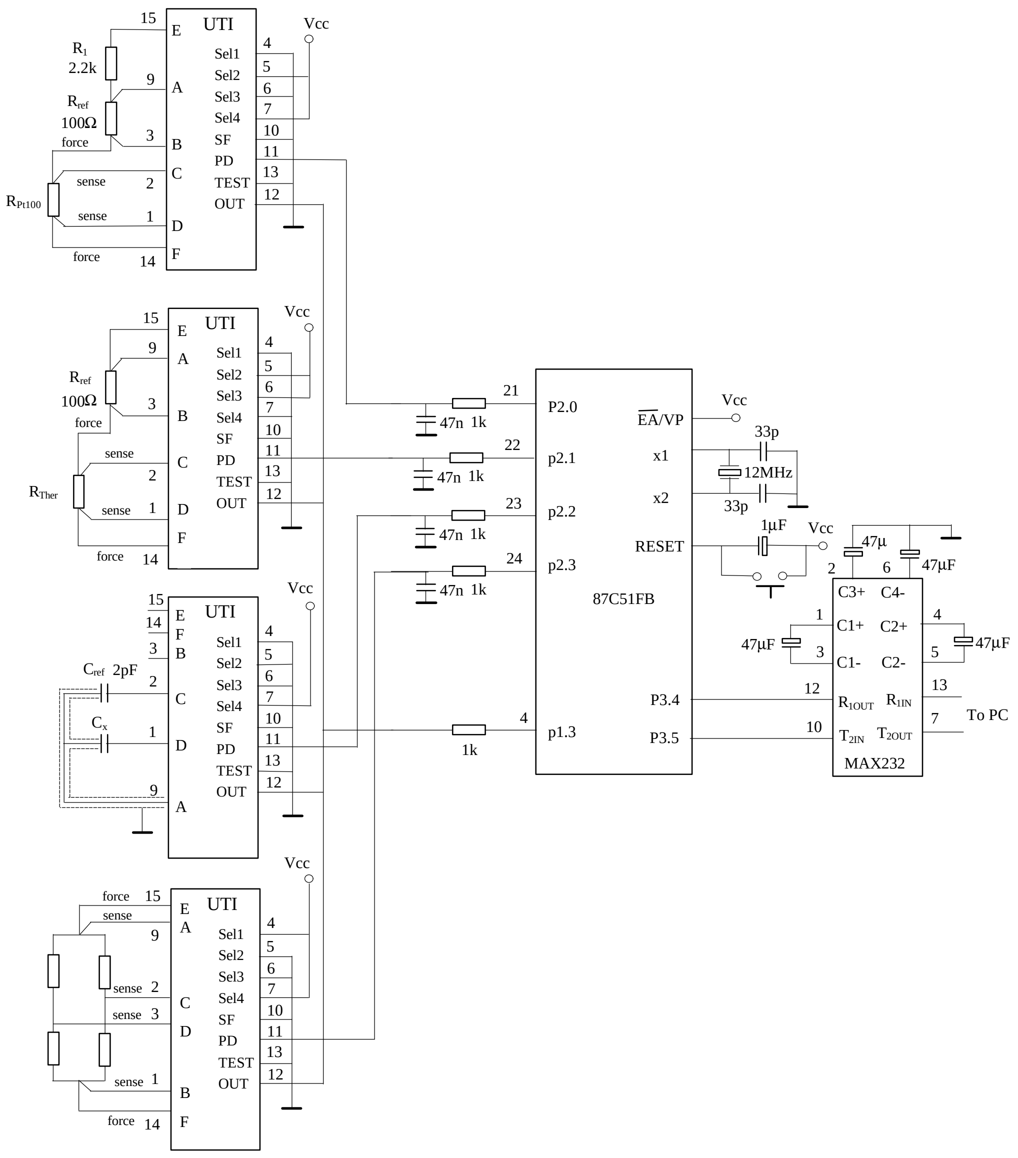
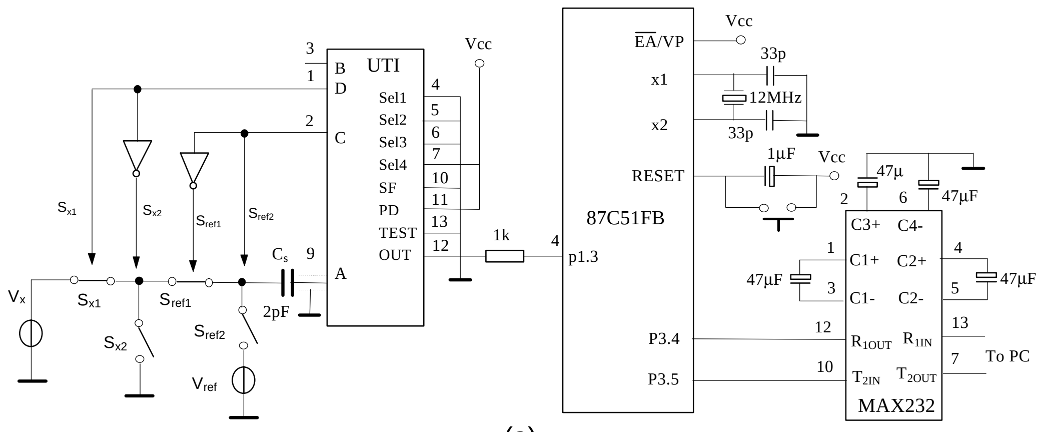


Figure 12 A setup for the measurement of multiple channel signal.

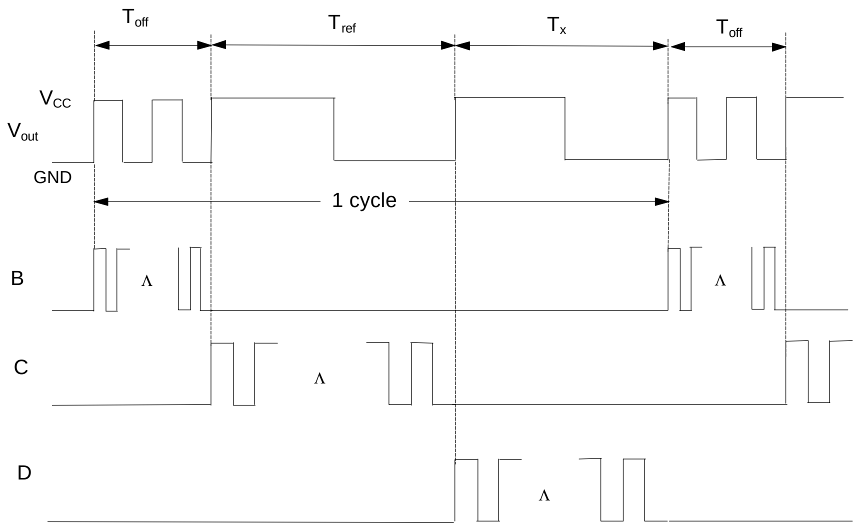
$$V_{x,range} C_s \leq V_{cc} C_{x,range} \, , \qquad (10)$$

where $C_{x,range}$ is the capacitance measurement range of the UTI. For instance, for the mode C23 of UTI, $C_{x,range}$ = 2 pF.

When the sampling switches are supplied by a power supply V_{DD} which is larger than the power supply V_{CC} of the UTI, the measurement system described by Figure 13(a) can measure a voltage signal larger than V_{CC}.



(a)



(b)

Figure 13 The schematic diagram of the measurement system. (b) Output signal V_{out} of the UTI.

Figure 14 shows an example of some C software for the microcontroller. Its main functions are:

- Setting up serial communication with PC. For serial communication, no interrupt connected with the SCI is enabled and the transmission rate is set 19200 Baud.
- Setting the situations of the UTI. The port p2.0 ~ p2.6 will output signals to control the situation of the UTI.
- Period measurement. The PCA function of the microcontroller is employed to measure the periods of the output signal from the UTI. The sampling frequency is fos/4 = 3 MHz.
- Data processing. The counted numbers of the periods are arranged in a certain order using the synchronous signal, T_{off}.
- Pint out the data to the serial port of the PC. According to the certain order, the measured data (counted number of the periods) are printed out. For instance, for the measurement Pt100 signal, the order of the print out data is: N_{off}, N_{AB}, N_{CD} and N_{BC}.

_4. SOFTWARE IN THE MICROCONTROLLER

The software in the microcontroller is indispensable to enable that the measurement setups which are presented in section 3 can work well. In these applications, a microcontroller, INTEL 87C51FB, is employed to measure the periods of the output signal from the UTI, to control the situations of the UTI, to process the data and to communicate with PC via the interface chip RS232.

A universal software is designed to satisfy all basic modes of the UTI which are described in the UTI data sheet.

```
#include <reg51f.h>
#include <stdio.h>
#include <pca.h>

/* Menu text */
#define MNUTXTN  15
code char *menukop[ MNUTXTN ] =
 {
  "                ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ» ",  /*  1 */
  "                º                                              º",     /*  2 */
  "                º  PHASES MEASUREMENT PROGRAM   º",   /*  3 */
  "                º                                              º",    /*  4 */
  "                ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼",  /*  5 */
  "                                                        ",   /*  6 */
  "                    s: set slow/fast mode                  ",   /*  7 */
  "                    t: test mode                            ",   /*  9 */
  "                    p: power down                          ",   /* 11 */
  "                    m: start measurement                 ",    /* 13 */
  "                                                          ",   /* 14 */
  "                   Please make a choice                 " ,  /* 15 */
 };

code char *modemenutext[ 22 ] =
 {
  "         Set measurement mode 0--f:                          ",  /*  2 */
  "                                                            ",  /*  3 */
  "      0: C25    --5 capacitors(0-2pf)                    ",  /*  4 */
  "      1: C23    --3 capacitors(0-2pf)                    ",  /*  5 */
  "      2: C12    --5 capacitors(0-12pf)                   ",  /*  6 */
  "      3: CMUX   --capacitors(0-2pf/0-12pf)                ",  /*  7 */
  "      4: C300   --3 capacitors(0-300pf)                         ",  /*  8 */
  "      5: Pt     --Platinum resistor(Pt100-Pt1000) 4-wire     ",  /*  9 */
  "      6: Ther   --Thermistor(1-25kohm) 4-wire                ",  /* 10 */
  "      7: Pt2    --2 or 3 Platinum resistor(Pt100-Pt1000)     ",  /* 11 */
  "      8: Ther2  --2 or 3 Thermistor(1-25kohm)                ",   /* 12 */
  "      9: Ub2    --Resistive bridge, ref. is Vbr, +/-200mv    ",   /* 13 */
  "      a: Ub1    --Resistive bridge, ref. is Vbr, +/-12.5mv   ",   /* 14 */
  "      b: Ib2    --Resistive bridge, ref. is Ibr, +/-200mv           ",  /* 15 */
  "      c: Ib1    --Resistive bridge, ref. is Ibr, +/-12.5mv           ",  /* 16 */
  "      d: Brg2   --Res. bridge and two resistors, +/-200mv    ",  /* 17 */
  "      e: Brg1   --Res. bridge and two resistors, +/-12.5mv   ",  /* 18 */
  "      f: Potm   --3 Potentiometers (1-50kohm)                    ",  /* 19 */
  "                                                             ",  /* 20 */
  "         Please make a choice                                ",  /* 21 */
};

code char *SetSFtext[ 9 ] =
 {
  "                                                          ",  /*  1 */
  "           Set Slow/Fast measurement mode              ",  /*  2 */
  "           s :  Slow measurement mode                          ",  /*  4 */
  "           f :  Fast measurement mode                  ",  /*  6 */
  "                                                         ",  /*  7 */
 "           Please make a choice                              ",  /*  8 */
};
code char *powertext[ 9 ] =
 {
 "                                                               ",  /*  1 */
 "           Set power down mode                               ",  /*  2 */
 "           d :  power is down                                ",  /*  4 */
 "           o :  power is on                                  ",  /*  6 */
 "                                                               ",  /*  7 */
 "           Please make a choice                              ",  /*  8 */
 };

code char *testtext[ 9 ] =
 {
 "                                                     ",  /*  1 */
 "           Set test mode                              ",  /*  2 */
 "           m :  for the normal measurement           ",  /*  3 */
 "           l :  for linearity measurement              ",  /*  4 */
 “                                                        “, /* 5 */
 "           Please make a choice                        ",  /*  6 */
  };

/* Serial communication  by rs232, "rs232.h"
 *
 * For osc.freq = 12 MHz  LJG
 *  RCPAP2H RCPAP2L
 * -  FF      63   2400 baud
 * -  FF      B1   4800 baud
 * -  FF      D8   9600 baud
 * -  FF      EC  19200 baud
  */
#define SCON_TI         0x02
#define SCON_EN_RECEP   0x10 /* Enable serial reception */
#define SCON_UART8      0x40     /* 8-bit UART */
#define SetupCom2_LJG( Mode ) ( RCAP2H = 0xFF, RCAP2L = 0xEC,\
                                        T2CON = 0x34, SCON = (( Mode ) | SCON_TI ) )

/* MACRO's and CONSTANTEN */
#define Clear_PCA()     CH = CL = 0;      /* Clear PCA counter */
#define Start_Cnt()           EC = CR = 1;       /* PCA: enable interrupt /counter run */
#define Stop_Cnt()            EC = CR = 0;       /* PCA: disable interrupt /counter stop */
#define M0_Cap_Posedg() CCAPM0 = 0x21;             /* Module 0 capture on pos. edge */

#define FALSE  0
#define TRUE   1

/*Data structure for the period measurement */
struct PCAcntr
 {
  unsigned char OvHi;   /* high order part of overflow counter */
  unsigned char OvLo;   /* low order part of overflow counter  */
  unsigned char Hi;        /* high order part of  PCA counter       */
  unsigned char Lo;       /* low order part of PCA counter       */
 };

union FourByte
 {
  unsigned long Word;
  struct PCAcntr Byte;
 };

union FourByte Capture;

/* Variables used in the whole program  */
unsigned long Cyclus[ 7 ];         /* Data array for the measured periods */
unsigned long Capture1, Capture2;  /* The “last” and the “resent” counted values */
unsigned char EdgeCnt;             /* Index of variables for the periods */
unsigned char phasenaantal;         /* The period's number in one cycle of measurement */
bit first, flag, Cyclus_rdy , meet;

sbit PIN1 = 0x80;  /* SEL1 of SEL1-SEL4 */
sbit PIN2 = 0x81;  /* SEL2 of SEL1-SEL4 */
sbit PIN3 = 0x82;  /* SEL3 of SEL1-SEL4 */
sbit PIN4 = 0x83;  /* SEL4 of SEL1-SEL4 */
sbit PIN5 = 0x84;  /* S/F */
sbit TEST = 0x85; /* TEST */
sbit PIN7 = 0x86;  /* PD */
sbit pin6 = P1^0;

/* Function prototypes */
void HoofdMenu( void );
void PhaseInstelMenu( void );
void Initialisatie( void );
void Meting( void );
void PcaIntFunc( void );
void SetSFmodle( void );
void Initialization( void );
void Setfunction( void );
void powerdown( void );
void testmode( void );

/*  PCA interrupt function */

void PcaIntFunc(void) interrupt 6 using 1 /* This function is to do the counting of one cycle and to put the counted numbers in Cyclus[i] */
 {
  bit ovl = CF;
  bit cap = CCF0;

  if (ovl && cap)
    if (CCAP0H == 0)
    {
      CF = ovl = 0;
      Capture.Byte.OvLo++;
    }
  if (cap)
   {
    CCF0 = cap = 0;
    if (first)                              /* first period */
     {
      first          = 0;
      Capture.Byte.Hi = CCAP0H;
      Capture.Byte.Lo = CCAP0L;
      Capture1        = Capture.Word;
     }
    else                                    /* for the rest periods    */
    {
      Capture.Byte.Hi  = CCAP0H;
      Capture.Byte.Lo  = CCAP0L;
      Capture2         = Capture.Word;
      Cyclus[EdgeCnt]  = (Capture2 - Capture1);
      Capture1         = Capture2;
      if (++EdgeCnt == phasenaantal)
       {
              EdgeCnt = 0;
              StopPCA();                            /* stop PCA-counter          */
              SetupPCA (CMOD_F12);
              Cyclus_rdy = 1;
              SetCaptureOff();
      };
    };
  };
if (ovl)
  {
    CF = ovl = 0;
    Capture.Byte.OvLo++;
  }
}

void HoofdMenu( void )
{
  char i;

  for( i = 0; i < MNUTXTN; i++ )
  printf( "%s\n", menukop[ i ] );
}
```

/* The Meting function is to get data from function PcaIntFunc(void), arrange them so that the offset is always first one, print the data out. */

```
void Meting( void )
{
 unsigned long bewaar;
 unsigned int i, j, k;
 unsigned int idex;
 bit Ready;

 Initialisatie();
 RI   =    0;
 EA   =    1;          /* enable interrupts */
 CMOD = 0x03;   /* ENABLE setup PCA timer overflow en CLK/4 */
 Start_Cnt();       /* enable PCA-interrupt en PCA-counter run */
 k = 0;
 meet = 1;
 while( meet )
  {
    k = k + 1;
    if( k > 20000 ) { meet = 0;}
    if( Cyclus_rdy )
     {
       Stop_Cnt();        /* disable PCA-interrupt PCA-counter stop */
       k = 0;
       CCAPM0 = 0x00;
       CMOD   = 0x00;
       Ready =  0;

       /* Rearrange the order of the data */
       while( Ready != 1 )
         {
           bewaar = Cyclus[ 0 ];
           idex = 0;

           for( i = 1; i < phasenaantal; i++ )
           {
             if( Cyclus[ i ] <= bewaar)
              {
               bewaar = Cyclus[ i ] ;
                idex = i;
              }
           }

          for( i = 0; i < idex; i++ )
           {
             bewaar = Cyclus[ 0 ];
              for( j = 0; j < (phasenaantal - 1); j++ )
              {
               Cyclus[ j ] = Cyclus[ j + 1 ];
              }
```

```
            Cyclus[ phasenaantal - 1 ] = bewaar;
           }

          if( Cyclus[ 0 ] < 9*Cyclus[ 1 ]/10 )
           {
            bewaar = Cyclus[ phasenaantal - 1 ];
            for( j = 0; j < phasenaantal - 1; j++ )
             {
               i = phasenaantal - 1 - j;
               Cyclus[ i ] = Cyclus[ i - 1 ];
             }
            Cyclus[ 0 ] = bewaar;
           }
          Ready = 1;
        }

       printf( "%lu\t", Cyclus[ 0 ] + Cyclus[ 1 ] );
       for( i = 2; i < phasenaantal; i++ )
          printf( "%lu\t", Cyclus[ i ] );
       putchar( '\n' );                          /* New line */

       /* An additional time delay   */
       for( i = 0; i < 5000; i++);
         pin6 = 0;

       /* Key interrupt  */
       if( RI == 1 )
        meet = 0;
       else
        {
          Initialisatie();
          EA   =    1;                  /* enable interrupts */
          CMOD = 0x03;            /* ENABLE setup PCA timer overflow */
          Start_Cnt();                /* enable PCA-interrupt en PCA-counter run */
        }
      }
   }
}

void Initialisatie( void )
{
   RI         = 0;
   EdgeCnt    = 0;   /* The measured data will be put from Cyclus[0] to Cyclus[phasenaantal -1] */
   Cyclus_rdy = 0;
   CF         = 0;
   CCF0       = 0;
   Capture.Byte.OvHi = 0;
   Capture.Byte.OvLo = 0;
   first      = 1;                 /* For first period */
   Clear_PCA();            /* Clear the counter register */
   M0_Cap_Posedg();        /* Initialize module 0 trigger on positive edge */
}

void main( void )
{
  char toets;
  SetupCom2_LJG( SCON_UART8 | SCON_EN_RECEP );
  Initialization();           /* Initialize the PCA counter and I/O system   */
  HoofdMenu();
  Initialisatie();           /* Set default conditions for system */

  while( 1 )
   {
     RI = 0;
     while( RI == 0 );
     switch( SBUF )
```

```
      {
        case 'm': Setfunction();
                       Meting();
                       HoofdMenu();
                       break;
        case 's': SetSFmodle();
                     HoofdMenu();
                     break;
        case 'p': powerdown();
                     HoofdMenu();
                     break;
        case 't': testmode();
                    HoofdMenu();
                    break;
        default: printf("Input Error");
                    HoofdMenu();
                    break;
      }
    }
}

void SetSfmodle( void )
 {
  char i; char choise;
  for( i = 0; i < 9; i++ ) {
    printf( "%s\n", SetSFtext[ i ] );
    }
  i = FALSE;
  while(i == FALSE)
   {
     RI = 0;
     while( RI != 0 );
     while( RI == 0 );
     choise = SBUF;
     switch( choise )
      {
        case 's': PIN5 = 0; i = TRUE; break;  /* set slow mode */
        case 'f': PIN5 = 1; i = TRUE; break;  /* set fast mode */
        default :
                      printf ("Input Error\n");
                      for( i = 0; i < 9; i++ ) {
                         printf( "%s\n", SetSFtext[ i ] );
                       }
                       i = FALSE;
                       break;
      }
   }
 }

void powerdown( void )
 {
  char i; char choise;
  for( i = 0; i < 9; i++ ) {
     printf( "%s\n", powertext[ i ] );
    }
  i = FALSE;
  while(i == FALSE)
   {
     RI = 0;
     while( RI != 0 );
     while( RI == 0 );
     choise = SBUF;
     switch( choise )
     {
      case 'd': PIN7 = 0; i = TRUE; break;  /* power is down */
      case 'o': PIN7 = 1; i = TRUE; break;  /* power is on */
      default :
                   printf ("Input Error\n");
                   for( i = 0; i < 9; i++ ) {
                      printf( "%s\n", powertext[ i ] );
                    }
                   i = FALSE;
                  break;
    }
  }
}

void testmode( void )
 {
  char i; char choise;
  for( i = 0; i < 9; i++ ) {
    printf( "%s\n", testtext[ i ] );
   }
  i = FALSE;
  while(i == FALSE)
   {
     RI = 0;
     while( RI != 0 );
     while( RI == 0 );
     choise = SBUF;
     switch( choise )
      {
case 'm': TEST = 0; i = TRUE; break;  /* normal measurement */
        case 'l':   TEST = 1; i = TRUE; break;  /* linearity measurement */
        default :
                     printf ("Input Error\n");
                     for( i = 0; i < 9; i++ ) {
                       printf( "%s\n", testtext[ i ] );
                      }
                     i = FALSE;
                    break;
      }
   }
}

void SetFunction( void )
 {
  char choise; char i;
  for( i = 0; i < 22; i++ ) {
    printf( "%s\n", modemenutext[ i ] );
   }
  i = FALSE;
  while(i == FALSE)
   {
     RI = 0;
     while( RI != 0 );
     while( RI == 0 );
     choise = SBUF;
     switch( choise )
      {
        case '0':  PIN1 = PIN2 = PIN3 = PIN4 = 0;  phasenaantal = 5 + 1;
                      if(TEST) {phasenaantal = 5;}
                       i = TRUE;
                       break;
        case '1':  PIN1 = PIN2 = PIN3 = 0; PIN4 = 1; phasenaantal = 3 + 1;
                      if(TEST) {phasenaantal = 5;}
                      i = TRUE;
                      break;
        case '2':  PIN1 = PIN2 = PIN4 = 0; PIN3 = 1; phasenaantal = 5 + 1;
                      if(TEST) {phasenaantal = 5;}
                      i = TRUE;
                      break;
        case '3':  PIN1 = PIN2 = 0; PIN4 = 1; PIN3 = 1; phasenaantal = 5 + 1;
                   i = TRUE;
                   break;
        case '4':  PIN1 = PIN3 = PIN4 = 0; PIN2 = 1; phasenaantal = 3 + 1;
                   if(TEST) {phasenaantal = 5;}
                    i = TRUE;
                   break;
        case '5':  PIN1 = 0; PIN3 = 0; PIN2 = PIN4 = 1; phasenaantal = 4 + 1;
                   i = TRUE;
                   break;
        case '6':  PIN1 = PIN4 = 0; PIN2 = PIN3 = 1; phasenaantal = 4 + 1;
                      i = TRUE;
                      break;
        case '7':  PIN1 = 0; PIN2 = PIN3 = PIN4 = 1; phasenaantal = 5 + 1;
                      if(TEST) {phasenaantal = 5;}
                      i = TRUE;
                      break;
        case '8':  PIN1 = 1; PIN2 = PIN3 = PIN4 = 0; phasenaantal = 5 + 1;
                      if(TEST) {phasenaantal = 5;}
                      i = TRUE;
                      break;
        case '9':  PIN1 = PIN4 = 1; PIN2 = PIN3 = 0; phasenaantal = 3 + 1;
                      i = TRUE;
                      break;
        case 'a':  PIN1 = PIN3 = 1; PIN2 = PIN4 = 0; phasenaantal = 3 + 1;
                      i = TRUE;
                      break;
        case 'b':  PIN1 = PIN3 = PIN4 = 1; PIN2 = 0; phasenaantal = 3 + 1;
                      i = TRUE;
                      break;
        case 'c':  PIN1 = PIN2 = 1; PIN3 = PIN4 = 0; phasenaantal = 3 + 1;
                      i = TRUE;
                      break;
        case 'd':  PIN1 = PIN2 = PIN4 = 1; PIN3 = 0; phasenaantal = 5 + 1;
                      if(TEST) {phasenaantal = 5;}
                      i = TRUE;
                      break;
        case 'e':  PIN1 = PIN2 = PIN3 = 1; PIN4 = 0; phasenaantal = 5 + 1;
                      if(TEST) {phasenaantal = 5;}
                      i = TRUE;
                      break;
        case 'f':  PIN1 = PIN2 = PIN3 = PIN4 = 1; phasenaantal = 5 + 1;
                      if(TEST) {phasenaantal = 5;}
                      i = TRUE;
                      break;
        default:
                  printf("Input Error\n");
                  i = FALSE;
                  break;
      };
   }
}

void Initialization( void )
 {
  PIN1 = PIN2 = PIN3 = PIN4 = 0;
  PIN5 = 0;TEST = 0;PIN7 = 1;
 }
```

Figure 14 Software listing for the microcontroller.