Application Note

Verwendung der E2 Schnittstelle

Version 1.01

	Name	Datum
Erstellt:	T. Pflügl	26.11.2004
Geprüft:	P. Froschauer	15.12.2004
Freigegeben:	I. Enzenhofer	17.12.2004
Änderungsgrund:	Korrektur im Beispielcode	

INHALT

1 Einführung

In dieser Application Note soll die Verwendung einer E2 Schnittstelle (siehe [1]) anhand eines einfachen Beispiels erläutert werden. Die E2 Schnittstelle dient zur digitalen, bidirektionalen Datenübertragung zwischen einem Mastermodul und einem Slavemodul. Die Datenübertragung erfolgt synchron und seriell, wobei der sog. Master für die Generierung des Taktes zuständig ist. Der Slave kann selbstständig keine Daten senden.
Als Beispiel sollen periodisch die Temperatur, die relative Luftfeuchte und der Status eines Transmitters mit E2 Interface ausgelesen werden.
Im folgenden Abschnitt wird kurz der Aufbau des Systems dargestellt, das Prinzip der Datenübertragung erklärt und ein Softwarebeispiel (in der Sprache C) angeführt.

2 Systembeispiel

Die E2 Schnittstelle ist grundsätzlich als Master- Slave Verbindung konzipiert. Als Master der Kommunikation wird, ohne auf ein spezielles Hardwareinterface (SM- bzw. I²C Bus Interface) angewiesen zu sein, exemplarisch ein Prozessor der 8051 Familie verwendet. Als Clock- bzw. Data- Leitung werden in diesem Beispiel die Pins P0.7 (SCL) bzw. P0.6 (DAT) verwendet. Diese Pins sind als open Drain I/O Pins konfiguriert und über zwei externe Pull-up-Widerstände mit der Versorgungsspannung verbunden.
Als Slave wird ein Transmitter vom Typ EE03 verwendet.

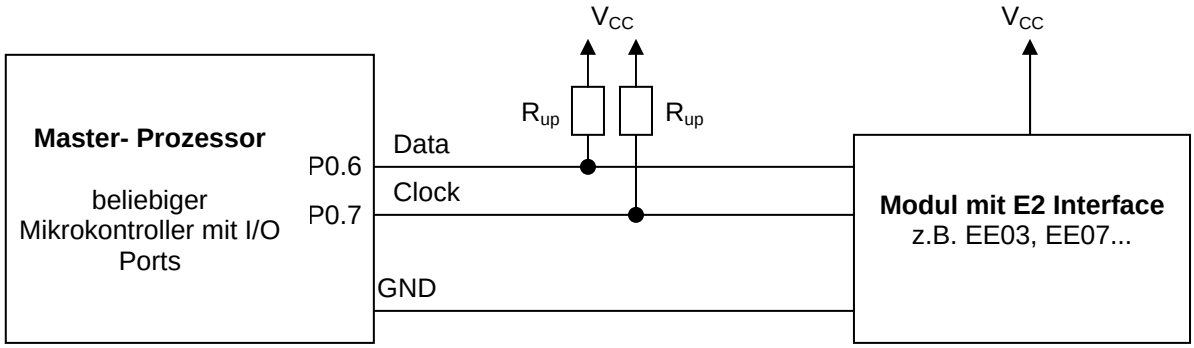

Abbildung 1: Zusammenschluss von Master und -Slave

Hinweis: Auf die Kompatibilität der Spannungspegel zwischen den E2 Schnittstellenpegeln [1] und dem Masterprozessor ist unbedingt zu achten.

3 Funktionsweise der Datenübertragung

Für eine Datenabfrage ist nur die (mehrmalige) Verwendung des „Read Byte from Slave“ Befehls notwendig (siehe [1]). Der „Read Byte from Slave“ Befehl ist ein bidirektionaler Befehl bei dem immer nur 1 Datenbyte vom Slave zum Master übertragen wird. Der Master gibt dem Slave über ein Kontrollbyte bekannt, welches Datenbyte er abrufen möchte. Der Slave antwortet im gleichen „Frame“ mit dem geforderten Datenbyte und einer Checksumme. Besteht der zu übertragene Wert aus mehreren Bytes, so ist eine wiederholte Ausführung des „Read Byte from Slave“ Befehls notwendig.

Die genaue Struktur des Befehls ist in [1] erläutert.

3.1 Messwertabfrage

3.1.1 Temperatur

Die Funktionsweise der Datenübertragung soll anhand einer mehrstufigen Temperaturabfrage erläutert werden. Um einen 16 Bit Temperaturwert übertragen zu können ist eine zweimalige Ausführung des „Read Byte from Slave“ Befehls erforderlich.
Der Temperaturmesswert entspricht beim verwendeten EE03 Modul (siehe [2]) der Messgröße 2.
Um die Datenkonsistenz gewährleisten zu können ist es notwendig immer zuerst das Low- Byte eines Messwertes abzufragen (siehe [1]).

Folgende Schritte sind zur Temperaturwertabfrage auszuführen:

Erster „Read Byte from Slave“ Befehl zum Einlesen des Temperatur- Low Wertes:

- Startbedingung und Kontrollbyte (0xA1) auf den Bus legen
- ACK/NACK des Slaves einlesen und prüfen
- Datenbyte temp_low einlesen
- Acknowledge an den Slave senden
- Checksumme vom Slave einlesen
- NACK und Stoppbedingung an den Slave senden
  (Der erste „Read Byte from Slave“ Befehl ist damit beendet)
- Kontrolle der Checksumme

Wenn die Checksumme in Ordnung ist, kann der zweite „Read Byte from Slave“ Befehl zum Einlesen des Temperatur- High Wertes gestartet werden:

- Startbedingung und Kontrollbyte (0xB1) auf den Bus legen
- ACK/NACK des Slaves einlesen und prüfen
- Datenbyte temp_high einlesen
- Acknowledge an den Slave senden
- Checksumme vom Slave einlesen
- NACK und Stoppbedingung an den Slave senden
  (Der zweite „Read Byte from Slave“ Befehl ist damit beendet)
- Kontrolle der Checksumme

Bei positiver Kontrolle der Checksumme kann der Master den tatsächlichen Temperaturwert ermitteln.
Nach dem Zusammenfügen von High- und Low- Byte zu einem 16 Bit Wert wird dieser durch 100 dividiert. Um die Temperatur in °C zu erhalten ist noch ein Offset von 273,15 zu subtrahieren.

$$Temperatur[°C] = (Temp\_high * 0x100 + Temp\_low) / 100 – 273{,}15$$

In Abbildung 2 ist das Flussdiagramm dieses Vorganges dargestellt.

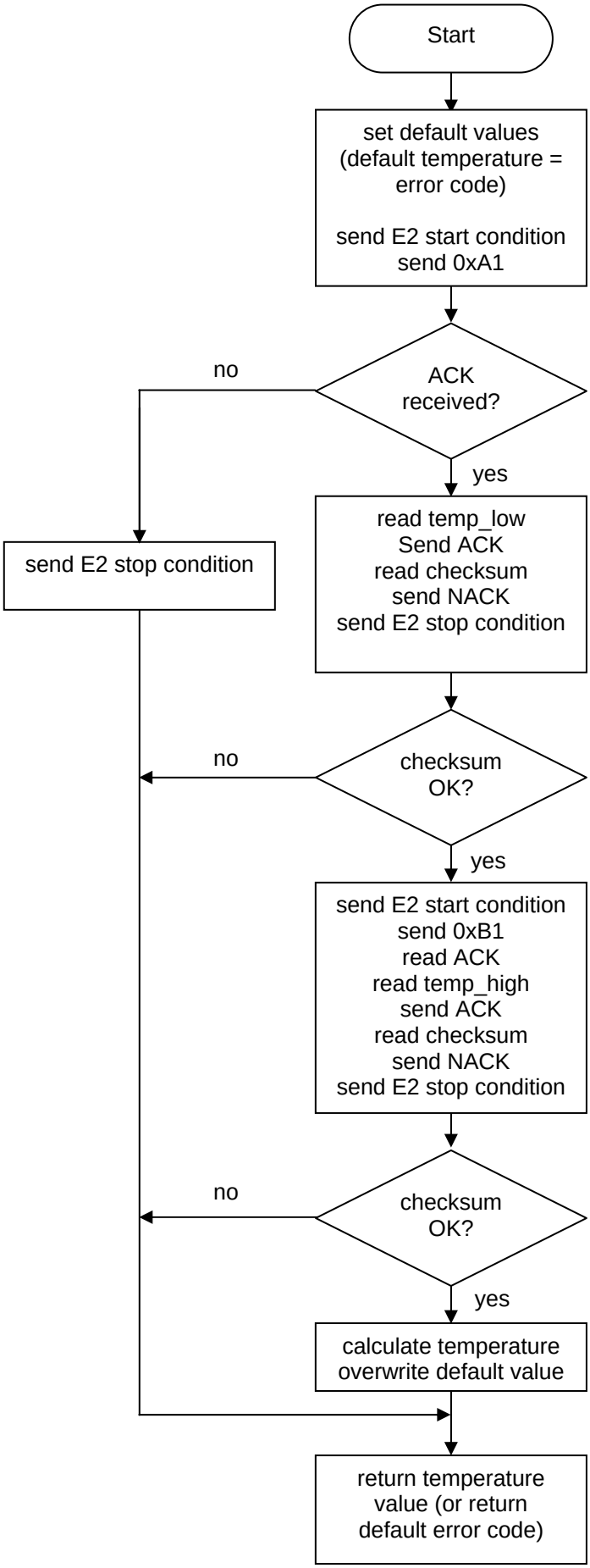

Abbildung 2: Flussdiagramm einer Temperaturabfrage

3.1.2 Relative Feuchte

Analog zur obigen Vorgehensweise kann der Feuchtemesswert ausgelesen werden. Die entsprechenden Kontrollbytes sind 0x81 für das Low- Byte und 0x91 für das High-Byte des Feuchtemesswertes.
Die tatsächliche Feuchte wird folgendermaßen errechnet:

rel. Feuchte[%RH] = (rh_high * 0x100 + rh_low) / 100

3.2 Statusabfrage

Um die Gültigkeit von Messwerten garantieren zu können muss anschließend an die Messwertabfragen noch der Status des Slavemoduls ausgelesen werden. Dazu wird ein weiterer „Read Byte from Slave“ Befehl ausgeführt und das Kontrollbyte 0x71 auf den Bus gelegt.
Nach dem Empfangen des Statusbytes wird (wie oben beschrieben) die Checksumme eingelesen und überprüft.
Wie in [1] dargestellt gibt das zweite Bit (Bit 1) im Statusbyte eine Auskunft über die Gültigkeit des Temperaturmesswertes. Das erste Bit (Bit 0) ist dem Feuchtemesswert zugeordnet.
Eine 0 bedeutet es wurde ein korrekter Messwert gesendet. Eine 1 bedeutet ein fehlerhafter Messwert wurde übermittelt (z.B. bei Sensorbruch).

Die Statusabfrage hat aber neben der Information über die Gültigkeit der letzten Messung noch einen weiteren Zweck. Nach jeder Statusabfrage vom Master wird im Slave eine neue Messung gestartet. Während der Messzeit kann der Slave keine Anfragen an der E2 Schnittstelle bearbeiten.
Es wird daher empfohlen, zuerst die gewünschten Messwerte einzulesen und danach eine Statusabfrage durchzuführen. Dadurch kann die Gültigkeit der letzten Messwerte beurteilt werden und gleichzeitig wird eine neue Messung gestartet. Nach dem Abwarten der modulspezifischen Messzeit können die neuen Werte wieder abgefragt werden.

4 Softwarebeispiele für 8051 Prozessoren

4.1 Allgemein

In dieser Application Note sind die Funktionen, welche die E2 Interface Operationen ausführen, in einem eigenen Softwaremodul zusammengefasst. Dadurch wird eine einfache Einbindung und Wiederverwendbarkeit des Codes erreicht.
Durch Einbinden des unten angeführten Headerfiles im Mainmodul des Mastercodes können die Beispielfunktionen zum Auslesen eines Temperaturwertes, der relativen Luftfeuchte, und des Statusbytes direkt verwendet werden.
Im Anhang sind die nötigen Definitionen der Variablen und die Funktionsprototypen angegeben um ein einfaches Softwaremodul für das E2 Interface erstellen zu können.

4.2 Mainmodul

Im Mainmodul werden nach der Initialisierung typischerweise kundenspezifische Aktionen in einer Endlosschleife ausgeführt. Es wird symbolisch eine Möglichkeit gezeigt, die Interfaceroutinen in günstiger Reihenfolge aufzurufen.

4.2.1 Beispielcode

```
      :
#include "E2_Interface.h"
      :
      :

void init_main (void)

{     /* initialise uP */
      :
      :

dummy = EE03_status();              // to start a measurement
      :

while (1)                           // main loop
  {   :
      /* user Code */               // minimum delay of measurement period…
      :                             // …see [2]
      humidity = RH_read();         // read humidity
      temperature = Temp_read();    // read temperature
      status = EE03_status();       // read status; start a new measurement

      /* analyse status and measured values */
      :
  }
}
```

4.2.2

Headerfile

Um die Routinen des E2 Interfacemoduls verwenden zu können ist das folgende Headerfile (E2_Interface.h) im Mainmodul des Mastercodes zu importieren:

```
/*****************************************************************************/
/*    headerfile for E2_Interface.c module                                   */
/*****************************************************************************/

float RH_read(void);
float Temp_read(void);
unsigned char EE03_status(void);
```

4.3 Softwarefunktionen des E2 Interfacemoduls

Die folgenden Funktionen ergeben, mit den Definitionen im Anhang, ein vollständig kompilierbares E2-Interface Softwaremodul. Dieser Code kann sehr einfach an den gewünschten Prozessor angepasst werden. Es sind dazu nur der DELAY_FAKTOR, die HW-Pins und die gekennzeichneten Funktionen zu adaptieren.

4.3.1 Temperaturabfrage

```
float Temp_read(void)
{
 temperature = -300;               // default value (error code)
 E2Bus_start();
 E2Bus_send(0xA1);                 // MW2-low request

 if (check_ack()==ACK)
 {     temp_low = E2Bus_read();
       send_ack();
       checksum_03 = E2Bus_read();
       send_nak();                 // terminate communication
       E2Bus_stop();

      if (((0xA1 + temp_low) % 256) == checksum_03)  // checksum OK?
      {     E2Bus_start();
            E2Bus_send(0xB1);       // MW2-high request
            check_ack();
            temp_high = E2Bus_read();
            send_ack();             // terminate communication
            checksum_03 = E2Bus_read();
            send_nak();
            E2Bus_stop();

            if (((0xB1 + temp_high) % 256) == checksum_03) // checksum OK?
            {  temp_ee03=temp_low+256*temp_high;  //yes->calculate temperature
               temperature=((float)temp_ee03/100) - 273.15;
                                    // overwrite default (error) value
            }
      }
      E2Bus_stop();
 }
 return temperature;
}
```

4.3.2

Feuchteabfrage

```
float RH_read(void)

{rh = -1;                          // default value (error code)

 E2Bus_start();
 E2Bus_send(0x81);                 // MW1-low request

 if (check_ack()==ACK)
 {     rh_low = E2Bus_read();
       send_ack();
       checksum_03 = E2Bus_read();
       send_nak();                 // terminate communication
       E2Bus_stop();

       if (((0x81 + rh_low) % 256) == checksum_03)   // checksum OK?
       {    E2Bus_start();
            E2Bus_send(0x91);      // MW1-high request
            check_ack();
            rh_high = E2Bus_read();
            send_ack();
            checksum_03 = E2Bus_read();
            send_nak();            // terminate communication
            E2Bus_stop();

            if (((0x91 + rh_high) % 256) == checksum_03)   // checksum OK?
            {  rh_ee03=rh_low+256*(unsigned int)rh_high;
                                   // yes-> calculate humidity value
               rh=(float)rh_ee03/100;
                                   // overwrite default (error) value
            }
       }
       E2Bus_stop();
 }
 return rh;
}
```

4.3.3 Statusabfrage

```
unsigned char EE03_status(void)

{unsigned char stat_ee03;

 E2Bus_start();                    // start condition for E2-Bus
 E2Bus_send(0x71);                 // main command for STATUS request
 if (check_ack()==ACK)
 {     stat_ee03 = E2Bus_read();   // read status byte
       send_ack();
       checksum_03 = E2Bus_read(); // read checksum
       send_nak();                 // send NAK ...
       E2Bus_stop();               // ... and stop condition to terminate
       if (((stat_ee03 + 0x71) % 256) == checksum_03)      // checksum OK?
          return stat_ee03;
 }
 return 0xFF;                      // in error case return 0xFF
}
```

4.3.4

Functions

```
void E2Bus_start(void)       // send Start condition to E2 Interface
{set_SDA();
 set_SCL();
 delay(30*DELAY_FAKTOR);
 clear_SDA();
 delay(30*DELAY_FAKTOR);
}
/*---------------------------------------------------------------------------*/

void E2Bus_stop(void)        // send Stop condition to E2 Interface
{clear_SCL();
 delay(20*DELAY_FAKTOR);
 clear_SDA();
 delay(20*DELAY_FAKTOR);
 set_SCL();
 delay(20*DELAY_FAKTOR);
 set_SDA();
 delay(20*DELAY_FAKTOR);
}
/*---------------------------------------------------------------------------*/

void E2Bus_send(unsigned char value)     // send Byte to E2 Interface
{unsigned char i;
 unsigned char maske = 0x80;

 for (i=8;i>0;i--)
 {    clear_SCL();
      delay(10*DELAY_FAKTOR);
      if ((value & maske) != 0)
            {set_SDA();}
      else
            {clear_SDA();}
      delay(20*DELAY_FAKTOR);
      set_SCL();
      maske >>= 1;
      delay(30*DELAY_FAKTOR);
      clear_SCL();
 }
 set_SDA();
}
/*---------------------------------------------------------------------------*/

unsigned char E2Bus_read(void)           // read Byte from E2 Interface
{unsigned char data_in = 0x00;
 unsigned char maske = 0x80;

 for (maske=0x80;maske>0;maske >>=1)
 {    clear_SCL();
      delay(30*DELAY_FAKTOR);
      set_SCL();
      delay(15*DELAY_FAKTOR);
      if (read_SDA())
            {data_in |= maske;}
      delay(15*DELAY_FAKTOR);
      clear_SCL();
 }
 return data_in;
}
/*---------------------------------------------------------------------------*/
```

```
char check_ack(void)                // check for acknowledge
{bit input;

 delay(30*DELAY_FAKTOR);
 set_SCL();
 delay(15*DELAY_FAKTOR);
 input = read_SDA();
 delay(15*DELAY_FAKTOR);
 if(input == 1)
    return NAK;
 else
    return ACK;
}
/*---------------------------------------------------------------------------*/

void send_ack(void)                 // send acknowledge
{
 delay(15*DELAY_FAKTOR);
 clear_SDA();
 delay(15*DELAY_FAKTOR);
 set_SCL();
 delay(30*DELAY_FAKTOR);
 set_SDA();
}
/*---------------------------------------------------------------------------*/

void send_nak(void)                 // send NOT-acknowledge
{
 delay(15*DELAY_FAKTOR);
 clear_SDA();
 delay(15*DELAY_FAKTOR);
 set_SCL();
 delay(30*DELAY_FAKTOR);
 set_SCL();
}
/*---------------------------------------------------------------------------*/

void delay(unsigned int value)      // delay- routine
{ while (--value != 0); }
/*---------------------------------------------------------------------------*/

// adapt this code for your target processor !!!

void set_SDA(void)
{ SDA = 1; }                        // set port-pin (SDA)

void clear_SDA(void)
{ SDA = 0; }                        // clear port-pin (SDA)

bit read_SDA(void)
{ return SDA; }                     // read SDA-pin status

void set_SCL(void)
{ SCL = 1; }                        // set port-pin (SCL)

void clear_SCL(void)
{ SCL = 0; }                        // clear port-pin (SCL)
```

4.4 Rückgabewerte

Von den oben angeführten Routinen wird folgendes Format für die Rückgabewerte verwendet:

Feuchte (float):

Rückgabewert	Bedeutung
0,0...100,0	0,0 bis 100,00% relativer Feuchte
-1	Fehlercode

Temperatur (float):

Rückgabewert	Bedeutung
> -273,15	absolute Temperatur in Celsius (entsprechend Messbereich)
-300	Fehlercode

Status (unsigned char):

Die Bedeutung der einzelnen Bits im Statusbytes ist für alle Module mit E2 Schnittstelle in [1] definiert.

4.5 Busgeschwindigkeit

Die Busgeschwindigkeit wird definiert durch die Taktfrequenz des Masterprozessors und der Konstante DELAY_FAKTOR (siehe Function: delay). Die maximale erreichbare Busgeschwindigkeit der Slavemodule ist in deren Spezifikation [2] bzw. [3] angegeben.

Achtung:
Die Zeitverzögerung wir durch eine einfache Warteschleife realisiert. Der Optimierungslevel des Compilers ist so zu wählen, dass diese Schleife nicht „wegoptimiert“ wird!

5 Literaturhinweise:

[1]	Specification E2-interface; E + E Elektronik
[2]	E2_interface_EE03_Vxx; E + E Elektronik
[3]	E2_interface_EE07_Vxx; E + E Elektronik

6 Anhang

6.1 Definitionen und Prototypen für das E2 Interfacemodul

```
/*****************************************************************************/
// SW-modul for E2 Interface
// Target: SiLabs C8051F005
// Compiler: Keil C51 Version 5.1
//
// created:  04/2004
//
/*****************************************************************************/

#include "F000.h"            // register definition for SiLabs C8051F00x

/*****************************************************************************/
// definitions

#define DELAY_FAKTOR 10      // setup clock-frequency
#define ACK       1          // define acknowledge
#define NAK       0          // define not-acknowledge

sbit SDA = P0^6;             // define port-pin for data line
sbit SCL = P0^7;             // define port-pin for clock line

/*****************************************************************************/
// variables

unsigned char rh_low;
unsigned char rh_high;
unsigned char temp_low;
unsigned char temp_high;
unsigned char checksum_03;
unsigned int  rh_ee03= 0;
unsigned int  temp_ee03= 0;
float rh = 0;
float temperature = 0;

/*****************************************************************************/
// functions

char check_ack(void);
void send_ack(void);
void send_nak(void);
void E2Bus_start(void);            // send start condition
void E2Bus_stop(void);             // send stop condition
void E2Bus_send(unsigned char);
void set_SDA(void);
void clear_SDA(void);
bit read_SDA(void);
void set_SCL(void);
void clear_SCL(void);
unsigned char E2Bus_read(void);    // read one byte from E2-Bus
void delay(unsigned int value);

/*****************************************************************************/
```