

Application Note

Utilising the E2 Interface

Version 1.03

	Name	Date
Created:	Robert Mayr.	13.04.2011
Checked:		
Released:		
Reason for change:	Code batch according ACK/NACK	

CONTENTS

1	INTRODUCTION.....	3
2	EXAMPLE SYSTEM	3
3	DATA TRANSMISSION METHOD	3
3.1	Measured Value Interrogation	4
3.1.1	Temperature.....	4
3.1.2	Relative Humidity	6
3.2	Status Interrogation.....	6
4	SOFTWARE EXAMPLES FOR 8051 PROCESSORS	7
4.1	General	7
4.2	Main Module.....	7
4.2.1	Example Code.....	7
4.2.2	Header file	8
4.3	Software Functions of the E2 Interface Module.....	8
4.3.1	Temperature Interrogation	8
4.3.2	Humidity Interrogation.....	9
4.3.3	Status Interrogation.....	9
4.3.4	Functions.....	10
4.4	Return Values	12
4.5	Bus Speed.....	12
5	LITERATURE REFERENCES:.....	12
6	SUPPLEMENT	13
6.1	Definitions and Prototypes for the E2 Interface Module	13

1 Introduction

This Application Note explains the use of an E2 Interface (ref. [1]) based on a simple example. The E2 Interface is used for the digital, bidirectional data transmission between a master module and a slave module.

The data transmission takes place via synchronous and serial modes, and the Master is responsible for generating the clock pulse. The Slave cannot transmit any data independently.

As an example, the temperature, the relative humidity of air and the status of a transmitter are to be read periodically with an E2 Interface.

The section below provides a brief description of the system structure, explains the principle of the data transmission and gives a software example (in language C).

2 Example System

The E2 Interface is designed as Master/Slave setup. Independently of a special hardware interface (SM or I²C Bus Interface), an 8051 range processor is used as the communication master in this example. The pins P0.7 (SCL) and P0.6 (DAT) will be used as clock or data lines. These pins are configured as open drain I/O pins and connected to the supply voltage via two external pull-up resistors.

A transmitter of the type EE03 is used as the slave.

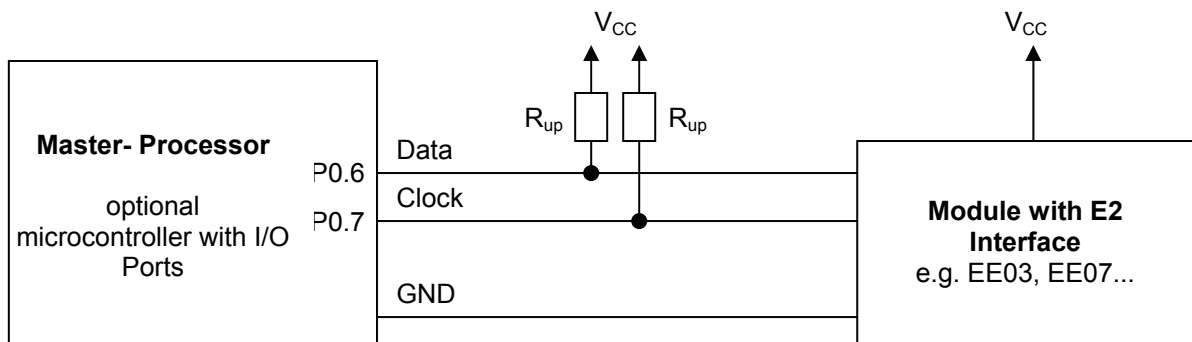


Figure 1: Setup of Master and Slave

Note: Note the compatibility of the voltage level between the E2 Interface levels [1] and the Master processor.

3 Data Transmission Method

Only the use (several times) of the „Read Byte from Slave“ command is necessary for data interrogation (ref. [1]). The „Read Byte from Slave“ command is a bidirectional command, which allows only 1 data byte to be transmitted from the Slave to the Master. With a control byte, the Master notifies the Slave which data byte it wishes to retrieve. The Slave answers in the same „Frame“ with the requested data byte and a checksum. If the value to be transmitted consists of several bytes, then a repeated execution of the „Read Byte from Slave“ command is necessary.

The detailed structure of the command is described in [1].

3.1 Measured Value Interrogation

3.1.1 Temperature

The data transmission method is explained using a multi-stage temperature interrogation. To transmit a 16 bit temperature value, a 2-fold execution of the „Read Byte from Slave“ command is necessary.

The measured value of the temperature corresponds to the measured variable 2 of the EE03 module (ref. [2]) .

To ensure that the data is consistent, it is necessary to interrogate the low-byte of a measured value first (ref. [1]).

The following steps are to be carried out for the temperature value interrogation:

First „Read Byte from Slave“ command for reading in the temperature_low value:

- Apply Start condition and control byte (0xA1) to the bus
- Read in and check ACK/NACK of the Slave
- Read in data byte temp_low
- Transmit acknowledgement to the Slave
- Read in checksum from the Slave
- Transmit NACK and Stop condition to the Slave
(the first „Read Byte from Slave“ command is thereby completed)
- Control of the checksum

If the checksum is correct, the second „Read Byte from Slave“ command can be started to read in the temperature_high value:

- Apply Start condition and control byte (0xB1) to the bus
- Read in and check ACK/NACK of the Slave
- Read in data byte temp_high
- Transmit acknowledgement to the Slave
- Read in checksum from the Slave
- Transmit NACK and Stop condition to the Slave
(The second „Read Byte from Slave“ command is thereby completed)
- Control of the checksum

With a positive control of the checksum, the Master can determine the actual temperature value.

After combining the High- and Low bytes to form a 16-bit value, this is divided by 100. To obtain the temperature in °C, an offset of 273.15 needs to be subtracted.

$$\text{Temperature}[^{\circ}\text{C}] = (\text{Temp_high} * 0x100 + \text{Temp_low}) / 100 - 273.15$$

Figure 2 provides the flow chart of this sequence.

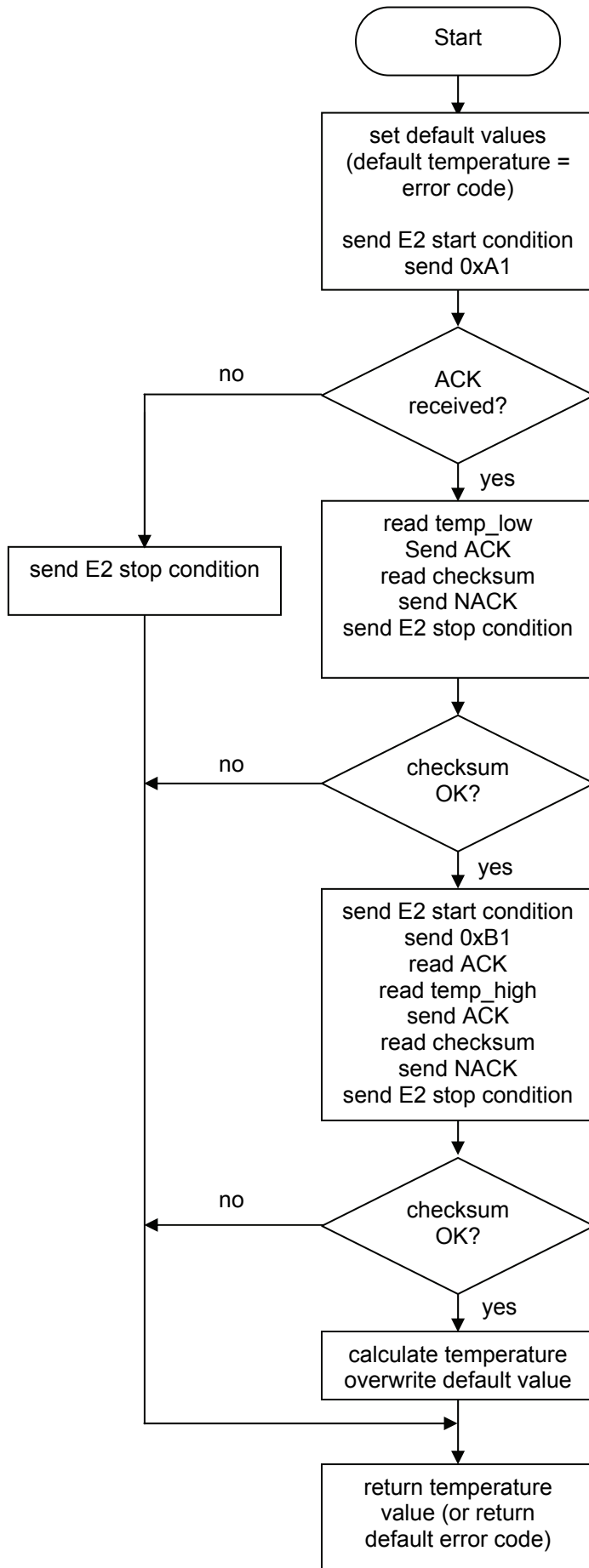


Figure 2: Flow chart of a Temperature Interrogation

3.1.2 Relative Humidity

The measured humidity value can be read similar to the method described above. The corresponding control bytes are 0x81 for the Low byte and 0x91 for the High byte of the measured humidity value. The actual humidity is calculated as follows:

$$\text{rel. humidity}[\%RH] = (\text{rh_high} * 0x100 + \text{rh_low}) / 100$$

3.2 Status Interrogation

To guarantee the validity of measured values, following the measured value interrogations, the status of the Slave module must also be read. For this, a further „Read Byte from Slave“ command is executed and the control byte 0x71 applied to the bus.

After receipt of the status byte, the checksum is read in and checked (as described above).

As shown in [1], the second bit (Bit 1) in the status byte provides information about the validity of the measured value of the temperature. The first bit (Bit 0) is assigned to the measured value of the humidity.

A “0” indicates a correct measured value has been transmitted. A “1” indicates a faulty measured value has been transmitted (e.g. with sensor failure).

Besides the information on the validity of the last measurement, the status interrogation has another purpose as well. After each status interrogation from the Master, a new measurement is started in the Slave. During the measuring time, the Slave cannot process any enquiries to the E2 Interface.

It is therefore advisable to read in the required measured values first, and then execute a status interrogation. By doing this the validity of the last measured values can be evaluated, and a new measurement is started at the same time. After waiting for the module-specific measuring time, the new values can be interrogated again.

4 Software Examples for 8051 Processors

4.1 General

In this Application Note, the functions the E2 Interface executes are grouped together in a separate software module. This achieves simple integration and reusability of the code.

By including the header file in the main module of the master code as specified below, the example functions can be used directly for reading the temperature value, the relative air humidity and the status byte.

The supplement specifies the required definitions of the variables and the function prototypes to be able to create a simple software module for the E2 Interface.

4.2 Main Module

After the initialisation, typically customer-specific actions are executed in a continuous loop in the main module. One option calling the interface routines in a favorable sequence is provided using symbols.

4.2.1 Example Code

```
      :
#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 Header file

To be able to implement the E2 Interface module routines, the following Header file (E2_Interface.h) needs to be imported into the master code main module:

```
/*
*****
/*   headerfile for E2_Interface.c module
*****
float RH_read(void);
float Temp_read(void);
unsigned char EE03_status(void);
```

4.3 Software Functions of the E2 Interface Module

The following functions permit the compilation of a complete E2-Interface software module using the definitions in the supplement. This code can be adapted to the required processor very easily. Only the DELAY_FACTOR, the HW-Pins and the designated functions are to be adapted.

4.3.1 Temperature Interrogation

```
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 Humidity Interrogation

```
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 Status Interrogation

```
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
{ clear_SCL();
  delay(15*DELAY_FAKTOR);
  clear_SDA();
  delay(15*DELAY_FAKTOR);
  set_SCL();
  delay(30*DELAY_FAKTOR);
  clear_SCL();
  set_SDA();
}
/*-----*/

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

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 Return Values

The following format for the return values is used by the routines specified above:

Humidity (float):

Return Value	Meaning
0.0...100.0	0.0 to 100.00% relative humidity
-1	Error code

Temperature (float):

Return Value	Meaning
> -273.15	temperature in Celsius (corresponding to measurement range)
-300	Error code

Status (unsigned char):

The meaning of the individual bits in the status bytes is defined for all modules with E2 Interface in [1].

4.5 Bus Speed

The bus speed is defined by the clock frequency of the master processor and the constant DELAY_FACTOR (see function: delay). The maximum bus speed that the Slave module can achieve is specified in its specification [2] or [3].

Attention:

The time delay is realised by means of a simple waiting loop. The optimisation level of the compiler should be selected so as to ensure that this loop is not „optimised out“!

5 Literature References:

[1]	Specification E2-interface; E + E Elektronik
[2]	E2_interface_EE03; E + E Elektronik
[3]	E2_interface_EE07; E + E Elektronik

6 Supplement

6.1 Definitions and Prototypes for the E2 Interface Module

```
/*
// 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);

*/
```