TwoWire <--> RS232 BRIDGE

USER GUIDE

TW_RS232 is a device for testing and communicating with TwoWire chips via a standard RS232 interface.

***TW_RS232* does not claim to be compatible with other established Two-Wire protocols on the market.**

TW_RS232 can be powered by either 4 AA-sized batteries / NiXX accumulators or by an external 6 – 9 VDC Power Supply. When both power feeds are given at the same time, the stronger voltage supersedes the other one. The input voltage must exceed the desired operating voltage (3.3V or 5V) by at least 500mV. The internal voltage regulator can handle currents as high as 150mA and has Overtemperature / Overcurrent protection.

The **RS232** can be operated at one of 6 baud rates (from 1K2 to 115K2); the selection is made by the rotary switch. The appropriate baud rate must be set before switching on the device.

A 9 pole female D - SUB connector is for the connection to the terminal (mostly a personal computer with a terminal program, like HyperTerminal).The device automatically goes into idle mode, when no character was received via the RS232 interface the last 60 minutes.

The **TwoWire bus** can be operated with 3.3V or 5V; the selection is made by the toggle switch. The appropriate voltage must be set before switching on the device.

4 banana female connectors (4mm) provide the signals (SCL, SDA)
and the power connections (GND, VCC).

The TwoWire signals SCL and SDA can be programmed to be pulled – up by either 2KΩ, 3.3KΩ, 5.6KΩ, 100KΩ. The bus speed can be programmed to 100KHz, 400KHz, 1MHz.

Switch on the device by pressing the **ON/OFF push-button** for at least 2 seconds –
all LEDs will light – release the button – the ON – LED maintains on, the other LEDs are dark; the device reports is ON – state to the serial port and is now ready to receive commands from there.

Whenever a command string is received, the 232 – LED will flash; if the command string contains any error, the ERROR – LED will flash, too.

Switching off the device is like switching it on – simply press the ON/OFF button for 2 seconds and release it, when the ON – LED is off.

Furthermore **2 BNC connectors** put out programmable testing pulses with a duration of 5 µs.

The X – BNC is active high and a push-pull type.

The Y – BNC is active low and has an internal 100KΩ pull-up resistor.

The power consumption, is approximately 30mA @ 5V and 18mA @ 3.3V, respectively (only ON-LED on, no external chip connected).

COMMANDS (TwoWire):

A string of at most 95 characters is possible. All commands are sent by ASCII characters.
A command string must always be terminated by an ***E***, preceded by a <blank> (0x20).
The commands are case sensitive, unless noted otherwise.

Start Condition:

S<blank>
A Start condition is initiated.

ReStart Condition:

R<blank>
A ReStart condition is initiated.
The ***S*** command can also be used instead of this command; both do the same !

Stop Condition:

P<blank>
A Stop condition is initiated.

Send Data:

D<blank><8 bit data><blank><expect acknowledge / not acknowledge from slave>
Send 8 bits and check, if the slave chip responds as awaited.
The 8 bit data (in the range of 0 to 255) can be given as decimal, hexadecimal or binary.

Decimal:	decimal digits, preceding zeros can be omitted.

Hexadecimal:	case-insensitive hexadecimal digits,
	with an preceding ***x*** (lowercase);
	preceding zeros can be omitted.

Binary:	one's or zero's, with an preceding ***b*** (lowercase).

The anticipation of an Acknowledge (ACK) from slave is represented by an ***a;*** a Not Acknowledge (NACK) by a ***n***
Examples:

D x1a a	Send out 1A (hex) and expect an ACK from the slave chip.
D x1A a	the same as above
D 77 n	Send out 77 (decimal) and expect a NACK from the slave chip.
D b10001001 a	Send out 10001001 (binary) and expect an ACK from the slave chip

Receive Data:

d<blank><acknowledge / do not acknowledge>
Receives 8 bits and send an ACK / NACK to the slave chip.
The received byte is sent back via the RS232 interface as decimal (default), hexadecimal or binary. See special commands.
Master acknowledges is represented by an ***A,*** Master does not acknowledge is represented by a ***N***
Examples:

d A	Receive a byte from slave and acknowledge.
d N	Receive a byte from slave and do not acknowledge.

COMMANDS (Special):

The special commands are implemented to make settings, insert time delays or put out trigger test signals.
A command string must always be terminated by an ***E***, preceded by a <blank> (0x20).
The commands are case sensitive, unless noted otherwise.

Insert Time Delays:

T<blank><decimal number><blank><time unit>
An additional delay time will be generated.
The decimal number can be in the range of 0 to 65535.
The time units are ***u*** for microseconds and ***m*** for milliseconds.
Examples:

T 500 u	Generates a time delay of 500 µs.
T 55555 m	Generates a time delay of 55.555 s.

Since the built – in command interpreter takes also some time (up to 300 µs) to filter out one command, the time delay will often be longer than the given value.

Insert Trigger Test Signals:

X<blank>
Makes a HIGH pulse of approximately 5 µs on the X – BNC
Idle LOW, push-pull type !

Y<blank>
Makes a LOW pulse of approximately 5 µs on the Y – BNC
The HIGH state is pulled up with 100KΩ, while the LOW state is "hard".

Set the Clock Frequency:

C<blank><string>
Sets up the clock frequency.
String is one of the following:

100K	For 100KHz bus speed (default).
400K	For 400KHz bus speed.
1M	For 1MHz bus speed.

This command must always be sent separately and not as part of a command string ! TW_RS232 confirms the setting or gives an error message.
Example:

C 400K E	Set the clock speed to 400KHz.

Set the Output Format:

F<blank><string>
Sets up the output format for the values, which were received from a slave chip.
String is one of the following:

BIN	Binary Output (0b????????, where ? are bin – digits).
DEC	Decimal Output (default, 3 digits with preceding zeros).
HEX	Hexadecimal Output (0x??, where ? are a hex - digits).

This command must always be sent separately and not as part of a command string ! TW_RS232 confirms the setting or gives an error message.
Example:

F HEX E Switches the Output Format to hexadecimal format.
A succeeding ***d*** will receive a byte from the slave chip and will
sent it back in hexadecimal format.

Set the Pull-Up Resistors:

U<blank><string>

Sets up the internal pull-up resistors for SCL and SDA.

String is one of the following:

2K	pull-up: 2KΩ (default)
3K3	pull-up: 3.3KΩ
5K6	pull-up: 5.6KΩ
100K	pull-up: 100KΩ

The 100K – mode affords the user to set his own appropriate pull-up resistors.

This command must always be sent separately and not as part of a command string ! TW_RS232 confirms the setting or gives an error message.

Example:

U 5K6 E	Pull – Up Resistors are now 5.6KΩ.

ERROR MESSAGES:

If the command string is all right, *TW_RS232* will answer with "OK" or confirms the settings for the ***C, F*** or ***U*** command; otherwise it will response to the RS232 with one of the following error messages:

COMMAND STRING TOO SHORT

A command string must have at least 3 characters, eg: ***S E***

COMMAND STRING TOO LONG

The limit of 95 characters was violated

COMMAND STRING STARTS WITH WRONG CHARACTER

The first character of the command string was wrong

COMMAND STRING GENERAL ERROR

Something is wrong in the command string (wrong command ?)

COMMAND STRING CONTAINS IMPROPER VALUES

Violation of the sent Data (wrong digits)

ACKNOWLEDGE ERROR FROM SLAVE

The slave chip did not respond as expected

START/RESTART ERROR (BUS BUSY, MISSING PULLUPS ?)

The Start / Restart command could not be executed.
Check if the bus is free and appropriate pull-up resistors are used.

PULL-UPs: NOT CHANGED !

A wrong resistor value was sent for the internal pull-ups.

MODE: NOT CHANGED !

A wrong clock speed was sent.

OUTPUT-FORMAT: NOT CHANGED !

A wrong output format was given.

<u>EXAMPLE 1 (DESCRIBED IN DETAIL):</u>

RAMTRON FRAM FM24C64 as Slave Chip:
We use the X – BNC for generating the trigger signal and are going to write 0x55 to the memory-cell at address 0x003C:

X S D xa0 a D 00 a D b00111100 a D x55 a P E

5µs HIGH Pulse on X – BNC
Start Condition
Send 0xA0 (device write adr) to the slave chip and check if ACK from slave
Send 0x00 (high adr) to the slave chip and check if ACK from slave
Send 0x3C (low adr) to the slave chip and check if ACK from slave
Send 0x55 (data) to the slave chip and check if ACK from slave
Stop Condition
Terminate Command String

The slave chip has now 0x55 on address 0x003C

Now we will read back from this memory-cell:

X S D xa0 a D 0 a D x3c a R D xa1 a d N P E

5µs HIGH Pulse on X – BNC
Start Condition
Send 0xA0 (device write adr) to the slave chip and check if ACK from slave
Send 0x00 (high adr) to the slave chip and check if ACK from slave
Send 0x3C (low adr) to the slave chip and check if ACK from slave
Restart Condition
Send 0xA1 (device read adr) to the slave chip and check if ACK from slave
Request Data from Slave and NACK
Stop Condition
Terminate Command String

The slave chip has answered with “085” (when in decimal format output), which is 0x55 !

EXAMPLE 2 (DESCRIBED IN DETAIL):

MAXIM/DALLAS DS1086 EconOscillator as Slave Chip:

With A2=A1=A0 (default), the address of the chip is 0xB0; we use the Y – BNC to generate a trigger signals and are going to generate a frequency of 11.0592 MHz, which is done by setting a prescaler value of 8 and a master frequency of 88.4736 MHz:

1.) We set the prescaler to 8 (with a dither of 4%):

Y S D xb0 a D x02 a D 3 a P E

5µs LOW Pulse on Y – BNC
Start Condition
Send 0xB0 (device write adr) to the slave chip and check if ACK from slave
Send 0x02 (prescaler adr) to the slave chip and check if ACK from slave
Send 0x03 (data) to the slave chip and check if ACK from slave
Stop Condition
Terminate Command String

The prescaler is now programmed to 8

2.) We switch to binary output Before we read the range – register:

F BIN E

Output format is now binary

Y S D xb0 a D x37 a R D xb1 a d N P E

5µs LOW Pulse on Y – BNC
Start Condition
Send 0xB0 (device write adr) to the slave chip and check if ACK from slave
Send 0x37 (range adr) to the slave chip and check if ACK from slave
Restart Condition
Send 0xB1 (device read adr) to the slave chip and check if ACK from slave
Request Data from Slave and NACK
Stop Condition
Terminate Command String

The slave chip has answered for example with "0b10010000"; only the last 5 bits are relevant – the range is therefore in this example 0x10 !

3.) Since the required master oscillator frequency is close to the center of OS – 2's frequency span, we will use this and the offset to be programmed is:
0x10 minus 2, 0xE thus, which we program now to the offset register.

Y S D xb0 a D xe a D xe a P E

5µs LOW Pulse on Y – BNC
Start Condition
Send 0xB0 (device write adr) to the slave chip and check if ACK from slave
Send 0x0E (offset adr) to the slave chip and check if ACK from slave
Send 0x0E (data) to the slave chip and check if ACK from slave
Stop Condition
Terminate Command String

The offset is now programmed to 0x0E

4.) Finally, the two-byte DAC value needs to be calculated and programmed:
(88.4736MHz – 81.92MHz) / 10KHz ≈ 655 (decimal)
Since the two – byte DAC register is left justified, 655 is converted to 0xA3C0, which we program to the DAC register

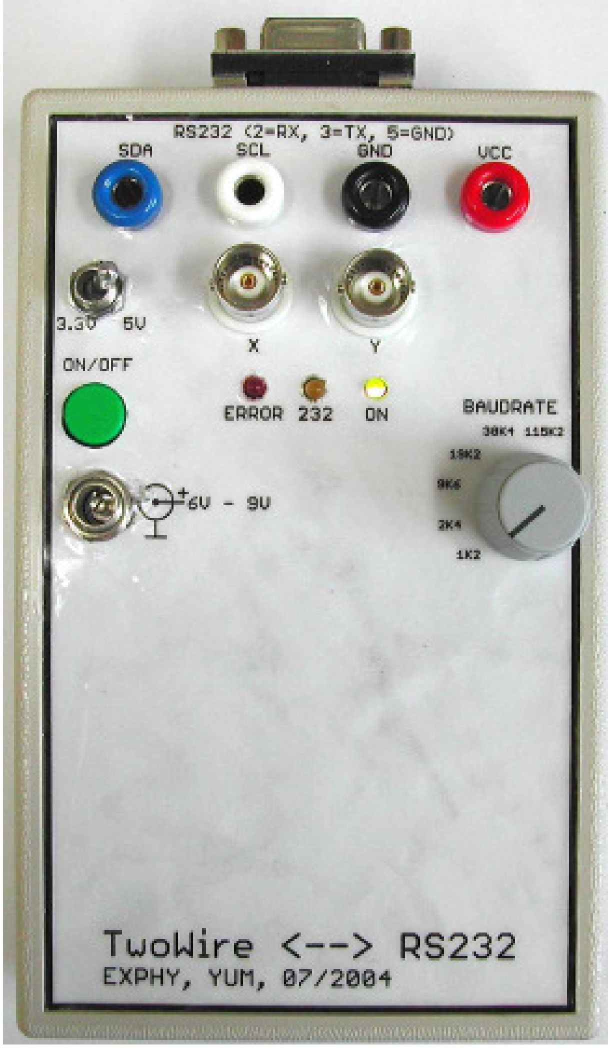
Y S D xb0 a D x8 a D xa3 a D xC0 a P E

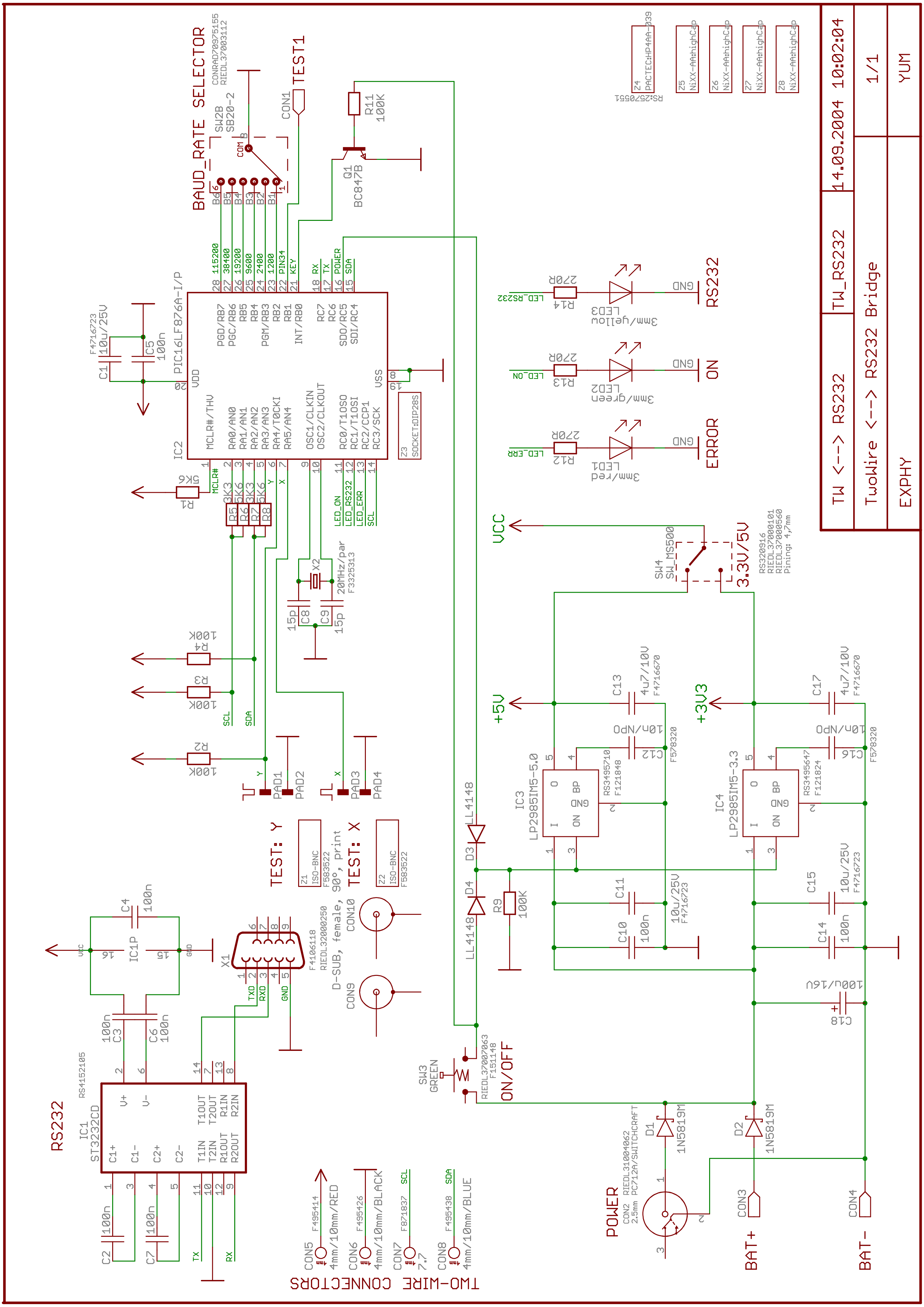
5µs LOW Pulse on Y – BNC
Start Condition
Send 0xB0 (device write adr) to the slave chip and check if ACK from slave

```
Send 0x08 (DAC high adr) to the slave chip and check if ACK from slave
Send 0xA3 (data) to the slave chip and check if ACK from slave
Send 0xC0 (data) to the slave chip and check if ACK from slave
Stop Condition
Terminate Command String
```

Since we have a dither span of 4%, we will measure a frequency of approximately 11.083 MHz, which is 2 % below 11.0592 MHz.

Exphy, YUM, r.daemon@tugraz.at, Graz, 06.08.2004


RS232 (2=RX, 3=TX, 5=GND)
SDA
SCL
GND
VCC
3.3V
5V
X
Y
ON/OFF
ERROR
232
ON
BAUDRATE
38K4
115K2
19K2
9K6
2K4
1K2
+6V - 9V
TwoWire <--> RS232
EXPHY, YUM, 07/2004



15.10.2010 12:58:05 f=0.91 C:\project\TW_RS232\hard\TW_RS232.sch (Sheet: 1/1)

```
TW_RS232.brd

11 different devices


          Qty    Value               Package           Parts

          2      15p                 0805              C8, C9
          8      100n                0805              C2, C3, C4, C5, C6, C7,
                                                       C10, C14
          2      4u7/10V             1206              C13, C17
          2      10n/NPO             1206              C12, C16
          3      10u/25V             1210              C1, C11, C15
          1      100u/16V            C2416             C18
          1      4mm/10mm/BLACK      BANANE2           CON6
          1      4mm/10mm/BLUE       BANANE2           CON8
          1      4mm/10mm/RED        BANANE2           CON5
          1      7.7                 BANANE2           CON7
          2      BNC2                BNC2              CON9, CON10
          3                          PAD-01            CON1, CON3, CON4
          1      2.5mm               SPG-ST21          CON2
          2      1N5819M             DO-213AB          D1, D2
          2      LL4148              SOD-80            D3, D4
          1      PIC16LF876A-I/P     DIL28-3           IC2
          1      ST3232CD            SO-16             IC1
          1      LP2985IM5-3.3       SOT23-5A          IC4
          1      LP2985IM5-5.0       SOT23-5A          IC3
          1      3mm/green           LED3MM1           LED2
          1      3mm/red             LED3MM1           LED1
          1      3mm/yellow          LED3MM1           LED3
          4                          PAD_T_2           PAD1, PAD2, PAD3, PAD4
          1      BC847B              SOT23_A           Q1
          2      3K3                 0805              R5, R7
          3      5K6                 0805              R1, R6, R8
          5      100K                0805              R2, R3, R4, R9, R11
          3      270R                0805              R12, R13, R14
          1      GREEN               KEY_D6R           SW3
          1      SW_MS500            SW-1UM2           SW4
          1      SB20-2              SW2X6             SW2
          1                          BOT               U$32
          2                          INDEX_1           U$33, U$34
          1                          TOP               U$31
          3      3.1                 Z_KREUZ3          U$39, U$40, U$41
          1      6.1                 Z_KREUZ3          U$42
          1      6.3                 Z_KREUZ3          U$36
          4      7.7                 Z_KREUZ3          U$2, U$14, U$24, U$35
          1      8.0                 Z_KREUZ3          U$38
          2      9.3                 Z_KREUZ3          U$46, U$47
          1      9.5                 Z_KREUZ3          U$37
          1      D-SUB, female, 90°, print F09HP             X1
          1      20MHz/par           HC49UP            X2
          2      ISO-BNC             Z_DUMMY1          Z1, Z2
          4      NiXX-AA:highCap     Z_DUMMY1          Z5, Z6, Z7, Z8
          1      PACTEC:HP4AA-039    Z_DUMMY1          Z4
          1      SOCKET:DIP28S       Z_DUMMY1          Z3

```

```
/* PIC16F876A, 20 MHz, TwoWire to RS232 bridge          */
/* Exphy                                                */
/* YUM - Ing. Reinhard Dämon                            */
/* TW_RS232.H, 07/04                                    */
/* compile with CCS, MPLAB 6.40                         */

#define bit int1

#byte INTCON = 0x0B
#bit INTF = INTCON.1
#byte PIR1 = 0x0C
#bit TXIF = PIR1.4
#bit RCIF = PIR1.5
#byte TXSTA = 0x98
#bit TRMT = TXSTA.1
#byte RCSTA = 0x18
#bit CREN = RCSTA.4
#bit FERR = RCSTA.2
#bit OERR = RCSTA.1
#byte PIE1 = 0x8C
#byte SSPCON = 0x14
#bit SSPOV = SSPCON.6
#bit  SSPEN = SSPCON.5
#bit SSPCKP = SSPCON.4
#byte SSPADD = 0x93
#byte SSPSTAT = 0x94
#bit SSPBF = SSPSTAT.0
#bit SSPCKE = SSPSTAT.6
#byte SSPBUF = 0x13

#define PORTA 0x05
#define SCL_3K3 PORTA*8+0
#define SCL_5K6 PORTA*8+1
#define SDA_3K3 PORTA*8+2
#define SDA_5K6 PORTA*8+3
#define YY PORTA*8+4
#define XX PORTA*8+5
#define PORTB 0x06
#define KEY   PORTB*8+0
#define TEST1 PORTB*8+1
#define BAUD1200 PORTB*8+2
#define BAUD2400 PORTB*8+3
#define BAUD9600 PORTB*8+4
#define BAUD19200 PORTB*8+5
#define BAUD38400 PORTB*8+6
#define BAUD115200 PORTB*8+7
#define PORTC 0x07
#define LED_ON PORTC*8+0
#define LED_RS232 PORTC*8+1
#define LED_ERR PORTC*8+2
#define SCL PORTC*8+3
#define SDA PORTC*8+4
#define POWER PORTC*8+5
#define TX  PORTC*8+6
#define RX  PORTC*8+7

// status:
```

```
bit ERROR;          // 1=error, 0=ok;
bit RS232;          // 0 = no char rec, 1 = char. from RS232 received

// serial buffer:
#define SER_BUFFER_LENGTH 96
char ser_buffer[SER_BUFFER_LENGTH];
#locate ser_buffer = 0x110
bit ser_buffer_cmd_ready;
unsigned char ser_buffer_pos;

//#define ERROR_TIMEOUT 12L     // in hours
//unsigned long error_timeout_counter;
#define RS232_TIMEOUT 1L    // in hours
unsigned long rs232_timeout_counter;

typedef enum {u,m} DELAY_BASE;
DELAY_BASE delaybase;
unsigned int delay_time;

typedef enum {SLOW, FAST, REALFAST} TW_MODE;
TW_MODE TWmode;
typedef enum {BIN, DEC, HEX} OUTPUT_FORMAT;
OUTPUT_FORMAT oformat;

typedef enum {R2K, R3K3, R5K6, R100K} PULL_UP;
PULL_UP pullup;

```

```
/* PIC16LF876A, 20 MHz, TwoWire to RS232 bridge                 */
/* Exphy                                                        */
/* YUM - Ing. Reinhard Dämon                                    */
/* TW_RS232.C, 07/04                                            */
/* compile with CCS, MPLAB 6.40                                 */

#include    "16f876A.h"
#device *=16

#define VERSION 0x0704
#define Fosc 20000000
#FUSES HS,WDT,PROTECT,NOBROWNOUT,PUT,NODEBUG,NOLVP,NOWRT
#TYPE  SHORT=8, INT=16, LONG=32
#USE delay(clock = Fosc)
#ZERO_RAM
#use standard_io(A)
#use standard_io(B)
#use standard_io(C)
#use rs232(baud=9600,parity=N,bits=8,xmit=PIN_C6,rcv=PIN_C7)
#use i2c(master, slow, sda=PIN_C4, scl=PIN_C3, force_hw)

// 100 KHz TW clock
#define TW_SLOW  ((Fosc / (4 * 100000)) -1)
// 400 KHz TW clock
#define TW_FAST  ((Fosc / (4 * 400000)) -1)
// 1 MHz TW clock
#define TW_REALFAST  ((Fosc / (4 * 1000000)) -1)

#include    "TW_RS232.H"
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <ctype.h>


//---------------------------------------------------------------------
// shutdown RS232 and MMC Power
void disable_peripheral()
 {
 output_float(SCL);
 output_float(SDA);
 output_low(LED_ON);
 output_low(LED_ERR);
 output_low(LED_RS232);
 output_float(TEST1);
 output_float(XX);
 output_float(YY);
 output_low(POWER);
 output_float(SCL_3K3);
 output_float(SCL_5K6);
 output_float(SDA_3K3);
 output_float(SDA_5K6);
 }
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// enable RS232 Power
```

```
void enable_peripheral()
 {
 output_high(SCL_3K3);
 output_high(SCL_5K6);
 output_high(SDA_3K3);
 output_high(SDA_5K6);
 output_high(POWER);
 delay_ms(100);
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// handles the errors
void error_handler(unsigned char err)
 {
 ERROR = 1;
 switch (err)
  {
  case 0x10: printf("\n\rCOMMAND STRING TOO SHORT\n\r"); break;
  case 0x11: printf("\n\rCOMMAND STRING TOO LONG\n\r"); break;
  case 0x20: printf("\n\rCOMMAND STRING STARTS WITH WRONG CHARACTER\n\r"); break;
  case 0x21: printf("\n\rCOMMAND STRING GENERAL ERROR\n\r"); break;
  case 0x22: printf("\n\rCOMMAND STRING CONTAINS IMPROPER VALUES\n\r"); break;
  case 0x30: printf("\n\rACKNOWLEDGE ERROR FROM SLAVE\n\r"); break;
  case 0x40: printf("\n\rSTART/RESTART ERROR (BUS BUSY, MISSING PULLUPS ?)\n\r");
break;
  default:   printf("\n\r");
  }
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// interupt comes every 104.85 ms
#int_timer1
void timer1_isr()
{
  static unsigned char key_debouncer = 0;
  static unsigned char rs232_timer = 0;
  static unsigned char error_timer = 0;

 output_high(TEST1);
 restart_wdt();
 if (input(KEY))
  key_debouncer = 0;
 else
  key_debouncer++;
 if (key_debouncer >= 20)             // key is pressed for 2 seconds --> reset CPU !
  {
  INTF = 0;
  output_bit(LED_ON,0);
  disable_peripheral();
  for(;;);
  }

 if (ERROR)
  {
  error_timer++;
  if (error_timer >= 20)
   { error_timer = 0; ERROR = 0; }
  }
 output_bit(LED_ERR,ERROR);

 rs232_timeout_counter++;
 if (rs232_timeout_counter >= RS232_TIMEOUT * 10L * 60L * 60L) //from hours to 100ms
  reset_cpu();

 if (RS232)
  {
  rs232_timer++;
  if (rs232_timer >= 5)
   { rs232_timer = 0; RS232 = 0; }
  }
 OUTPUT_BIT(LED_RS232,RS232);
 output_low(TEST1);
}
//----------------------------------------------------------------------


//----------------------------------------------------------------------
// initialize serial buffer:
void up_init_ser_buffer()
 {
 unsigned char counter;
 for (counter = 0; counter !=  SER_BUFFER_LENGTH; counter++)
  ser_buffer[counter] = 0x00;
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// fill serial buffer with characters from RS232:
void up_fill_ser_buffer_with_rs232_character(char ch)
 {
 ser_buffer[ser_buffer_pos++] = ch;
 if ( (ch == 'E') && ((ser_buffer[ser_buffer_pos - 2]) == ' ') )
  ser_buffer_cmd_ready = 1;
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// interupt: RS232 receive data available
#INT_RDA
void rs232_rx_int()
 {
  up_fill_ser_buffer_with_rs232_character(getc());
  rs232_timeout_counter = 0L;
  RS232 = 1;
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// init some global parameters
void up_init_global()
 {
 disable_interrupts(INT_RDA);
 setup_adc(ADC_OFF);
 setup_adc_ports(NO_ANALOGS);
 setup_ccp1 (CCP_OFF);
// disable_peripheral();
// port_b_pullups(TRUE);
 output_low(LED_ON);
 output_low(LED_ERR);
 output_low(LED_RS232);
 ERROR = 0;
 setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8 );
 disable_interrupts(GLOBAL);     // disable global interrupts
 enable_interrupts(int_timer1);
// ext_int_edge( H_TO_L );   // Sets up EXT
// enable_interrupts(INT_EXT);
// output_float(POWER);
 RS232 = 0;
 output_low(TEST1);
 rs232_timeout_counter = 0L;
 ser_buffer_pos = 0;
 delay_time = 0;
 delaybase = u;
 twmode = SLOW;
 oformat = DEC;
 pullup = R2K;
 output_low(XX);
 output_high(YY);
 }
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// subroutine for slow(100K), fast(400K), realfast(1M) mode:
void up_cmd_c()
 {
 printf("\n\rMODE: ");
 if ( (ser_buffer[2] == '1') && (ser_buffer[3] == '0') &&
      (ser_buffer[4] == '0') && (ser_buffer[5] == 'K') )
  { SSPADD = TW_SLOW; twmode = SLOW; printf("100K\n\r"); }
 else
 if ( (ser_buffer[2] == '4') && (ser_buffer[3] == '0') &&
      (ser_buffer[4] == '0') && (ser_buffer[5] == 'K') )
  { SSPADD = TW_FAST; twmode = FAST; printf("400K\n\r"); }
 else
 if ( (ser_buffer[2] == '1') && (ser_buffer[3] == 'M') )
  { SSPADD = TW_REALFAST; twmode = REALFAST; printf("1M\n\r"); }
 else
  printf("NOT CHANGED !\n\r");
 }
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// subroutine for setting the output format(bin, dec, hex):
void up_cmd_f()
 {
 printf("\n\rOUTPUT-FORMAT: ");
 if ( (ser_buffer[2] == 'B') && (ser_buffer[3] == 'I') &&
      (ser_buffer[4] == 'N') )
  { oformat = BIN; printf("BINARY\n\r"); }
 else
 if ( (ser_buffer[2] == 'D') && (ser_buffer[3] == 'E') &&
      (ser_buffer[4] == 'C') )
  { oformat = DEC; printf("DECIMAL\n\r"); }
 else
 if ( (ser_buffer[2] == 'H') && (ser_buffer[3] == 'E') &&
      (ser_buffer[4] == 'X') )
  { oformat = HEX; printf("HEXADECIMAL\n\r"); }
 else
  printf("NOT CHANGED !\n\r");
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// subroutine for setting the pull-up resistors(2K, 3K3, 5K6):
void up_cmd_u()
 {
 printf("\n\rPULL-UPs: ");
 if ( (ser_buffer[2] == '2') && (ser_buffer[3] == 'K') )
  { pullup = R2K; printf("2K\n\r"); }
 else
 if ( (ser_buffer[2] == '3') && (ser_buffer[3] == 'K') &&
      (ser_buffer[4] == '3') )
  { pullup = R3K3; printf("3K3\n\r"); }
 else
 if ( (ser_buffer[2] == '5') && (ser_buffer[3] == 'K') &&
      (ser_buffer[4] == '6') )
  { pullup =  R5K6; printf("5K6\n\r"); }
 else
 if ( (ser_buffer[2] == '1') && (ser_buffer[3] == '0') &&
      (ser_buffer[4] == '0') && (ser_buffer[5] == 'K') )
  { pullup =  R100K; printf("100K\n\r"); }
 else
  printf("NOT CHANGED !\n\r");
 if (pullup == R2K)
  {
  output_high(SCL_3K3);
  output_high(SCL_5K6);
  output_high(SDA_3K3);
  output_high(SDA_5K6);
  }
 if (pullup == R3K3)
  {
  output_high(SCL_3K3);
  output_float(SCL_5K6);
  output_high(SDA_3K3);
  output_float(SDA_5K6);
  }
 if (pullup == R5K6)
  {
  output_float(SCL_3K3);
  output_high(SCL_5K6);
  output_float(SDA_3K3);
  output_high(SDA_5K6);
  }
 if (pullup == R100K)
  {
```

```
  output_float(SCL_3K3);
  output_float(SCL_5K6);
  output_float(SDA_3K3);
  output_float(SDA_5K6);
  }
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// the commander action:
#SEPARATE
unsigned char up_CMD_INTERPRETER_go_(char str[])
 {
 unsigned char data;
 unsigned char minicou;
 char *ptr;
 bit ack;
 bit ack_;

if (str[0] == 'X')
  {
  output_high(XX);
  delay_us(5);
  output_low(XX);
  }
else
if (str[0] == 'Y')
  {
  output_low(YY);
  delay_us(5);
  output_high(YY);
  }
else
if ( (str[0] == 'S') || (str[0] == 'R') )
  {
  i2c_start();
  }
else
if (str[0] == 'P')
  i2c_stop();
else
if (str[0] == 'T')
  {
  delay_time = 0;
  for (minicou = 2; minicou != 7; minicou++)
   {
   data = str[minicou];
   if ( (data < '0') || (data > '9') ) break;
   delay_time = (delay_time << 3) + (delay_time << 1) + (data - '0');
   }
  if (delaybase == u)
   {
   while (delay_time > 255)
    {
    delay_us(255);
    restart_wdt();
    delay_time -= 255;
    }
   delay_us((unsigned char)delay_time);
   }
  else
   {
   while (delay_time > 255)
    {
    delay_ms(255);
    restart_wdt();
    delay_time -= 255;
    }
   delay_ms((unsigned char)delay_time);
   }
  }
else
if (str[0] == 'D')
  {
  data = 0;
  if (str[2] == 'b')
   {
   ptr = &str[3];
   minicou = 7;
   do
    {
    data += (*ptr++ - 0x30) << minicou;
    }
   while(minicou--);
   if (str[12] == 'a') ack_ = 0;
   else ack_ = 1;
   }
  else
  if (str[2] == 'x')
   {
   if (isdigit(str[3]))
    data = (data << 4) + (str[3] - 0x30);
   if ( (str[3] >= 'A') && (str[3] <= 'F') )
    data = (data << 4) + (str[3] - 0x37);
   if ( (str[3] >= 'a') && (str[3] <= 'f') )
    data = (data << 4) + (str[3] - 0x57);
   if (isdigit(str[4]))
    data = (data << 4) + (str[4] - 0x30);
   if ( (str[4] >= 'A') && (str[4] <= 'F') )
    data = (data << 4) + (str[4] - 0x37);
   if ( (str[4] >= 'a') && (str[4] <= 'f') )
    data = (data << 4) + (str[4] - 0x57);
   if (str[4] == ' ')
    if (str[5] == 'a') ack_ = 0;
    else ack_ = 1;
   else
    if (str[6] == 'a') ack_ = 0;
    else ack_ = 1;
   }
  else
   {
   for (minicou = 2; minicou != 5; minicou++)
    {
    if (str[minicou] == ' ') break;
    data = (data << 3) + (data << 1) + (str[minicou] - '0');
    }
   if (str[++minicou] == 'a') ack_ = 0;
   else ack_ = 1;
   }
  ack = i2c_write(data);
  if (ack != ack_)    // check for acknowledge
   { error_handler(0x30); return (1); }
  }
else
if (str[0] == 'd')
  {
  if (str[2] == 'A')
   data = i2c_read(1);
  else
   data = i2c_read(0);
  if (oformat == BIN)
   {
   putc('\n');
   putc('\r');
   putc('0');
   putc('b');
   for (minicou = 0; minicou != 8; minicou++)
    {
    if (shift_left(&data, 1, 0))
     putc('1');
    else
     putc('0');
    }
   printf("\n\r");
   }
  if (oformat == DEC)
   printf("\n\r%03U\n\r",data);
  if (oformat == HEX)
   printf("\n\r0x%02X\n\r",data);
  }
 return (0);
 }
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// the command interpreter:
unsigned char up_cmd_interpreter()
 {
 #define STRING_LENGTH 20
 unsigned char pos;
 unsigned char cou;
 unsigned char minicou;
 char str[STRING_LENGTH];
 pos = 0;
 // start loop through ser_buffer:
 do
  {
  if (ser_buffer[pos] == ' ')
   {pos++; continue; }
   // filter one set of command/data:
   {
```

```
   for (cou = 0; cou != STRING_LENGTH; cou++)
    str[cou] = ' ';
   cou = 0;
   str[cou] = ser_buffer[pos];
if ( (str[cou] == 'S') || (str[cou] == 'R') ||
     (str[cou] == 'P') || (str[cou] == 'X') || (str[cou] == 'Y') )
    {
    str[++cou] = ser_buffer[++pos];
    if (str[cou] != ' ')
     { error_handler(0x21); return(1); }
    else
     { goto UP_CMD_INTERPRETER_GO; }
    }
else
if (str[cou] == 'T')
    {
    str[++cou] = ser_buffer[++pos];
    if (str[cou] != ' ')
     { error_handler(0x21); return (1);}
    else
     {
    // filter max 5 decimal values and one ' ':
     for (minicou = 0; minicou != 6; minicou++)
      {
      str[++cou] = ser_buffer[++pos];
      if (isdigit(str[cou]))
       continue;
      else
      if ( minicou && (str[cou] == ' ') )
       goto UP_CMD_INTERPRETER_UM;
      else
       {
       if (minicou == 5)
        { error_handler(0x21); return (1);}
       else
        { error_handler(0x22); return(1);}
       }
      }
UP_CMD_INTERPRETER_UM:
      str[++cou] = ser_buffer[++pos];       // expected 'u' for micro or 'm' for milli
      if (str[cou] == 'u')
       delaybase = u;
      else
      if (str[cou] == 'm')
       delaybase = m;
      else
       { error_handler(0x21); return(1); }
      str[++cou] = ser_buffer[++pos];
      if (str[cou] != ' ')
       { error_handler(0x21); return(1); }
      else
       { goto UP_CMD_INTERPRETER_GO; }
     }
    }
else
if (str[cou] == 'D')
    {
```

```c
      str[++cou] = ser_buffer[++pos];
      if (str[cou] != ' ')
       { error_handler(0x21); return (1);}
      else
       {
       str[++cou] = ser_buffer[++pos];
       if (isdigit(str[cou]))         // decimal value (1
        {
        str[++cou] = ser_buffer[++pos]; // (2
        if (str[cou] == ' ')
         goto UP_CMD_INTERPRETER_AN;
        else
         if (!isdigit(str[cou]))
          { error_handler(0x22); return(1);}
         else
          {
          str[++cou] = ser_buffer[++pos]; // (3
          if (str[cou] == ' ')
           goto UP_CMD_INTERPRETER_AN;
          else
           if (!isdigit(str[cou]))
            { error_handler(0x22); return(1);}
           else
            {
            str[++cou] = ser_buffer[++pos];
            if (str[cou] == ' ')
             goto UP_CMD_INTERPRETER_AN;
            else
             { error_handler(0x21); return(1);}
            }
          }
        }
       if (str[cou] == 'x')                   // hexadecimal value
        {
        str[++cou] = ser_buffer[++pos];       // (1
        if (!isxdigit(str[cou]))
         { error_handler(0x22); return (1); }
        else
         {
         str[++cou] = ser_buffer[++pos];      // (2
         if (str[cou] == ' ')
          goto UP_CMD_INTERPRETER_AN;
         else
          {
          if (!isxdigit(str[cou]))
           { error_handler(0x22); return (1); }
          str[++cou] = ser_buffer[++pos];
          if (str[cou] == ' ')
           goto UP_CMD_INTERPRETER_AN;
          else
           { error_handler(0x21); return (1); }
          }
         }
        }
       if (str[cou] == 'b')                   // binary value
        {
        for (minicou = 0; minicou != 8; minicou++)
         {
         str[++cou] = ser_buffer[++pos];
         if ( (str[cou] != '0') && (str[cou] != '1') )
          { error_handler(0x22); return (1); }
         }
         str[++cou] = ser_buffer[++pos];
         if (str[cou] == ' ')
           goto UP_CMD_INTERPRETER_AN;
         else
          { error_handler(0x21); return (1); }
        }
  UP_CMD_INTERPRETER_AN:
        str[++cou] = ser_buffer[++pos];       // expected ack/nack from slave
        if ( (str[cou] != 'a') && (str[cou] != 'n') )
         { error_handler(0x21); return(1); }
        else
         {
         str[++cou] = ser_buffer[++pos];
         if (str[cou] != ' ')
          { error_handler(0x21); return(1); }
         else
          { goto UP_CMD_INTERPRETER_GO; }
         }
       }
      }
  else
  if (str[cou] == 'd')
      {
      str[++cou] = ser_buffer[++pos];
      if (str[cou] != ' ')
       { error_handler(0x21); return (1);}
      else
       {
       str[++cou] = ser_buffer[++pos];
       if ( (str[cou] != 'A') && (str[cou] != 'N') )
        { error_handler(0x21); return(1);}
       else
        {
        str[++cou] = ser_buffer[++pos];
        if (str[cou] != ' ')
         { error_handler(0x21); return(1); }
        else
         { goto UP_CMD_INTERPRETER_GO; }
        }
       }
      }
  else
      { error_handler(0x21); return (1);}
     }
  // command set is filtered now !
  UP_CMD_INTERPRETER_GO:
    disable_interrupts(GLOBAL);
    delay_us(delay_time);
    enable_interrupts(GLOBAL);
    if (up_cmd_interpreter_go_(str))
     return (1);
    } // end of loop
```

```c
 while (pos < (ser_buffer_pos - 2) );
 return (0);
 }
//----------------------------------------------------------------------

//----------------------------------------------------------------------
void main()
 {
 unsigned char retry;

 restart_wdt();
 setup_wdt(WDT_2304MS);
 disable_peripheral();
 port_b_pullups(TRUE);

 delay_ms(50);
 retry = 0x00;
 do
  {
  retry++;
  delay_ms(8);
  restart_wdt();
  }
 while ( (retry) && (!input(KEY)) );
 if (retry)
  for(;;);

 enable_peripheral();
 output_high(LED_ON);
 output_high(LED_RS232);
 output_high(LED_ERR);

 // wait until key is released for 1 second:
 retry = 0x00;
 do
  {
  delay_ms(4);
  if (!input(KEY))
   retry = 0x00;
  retry++;
  restart_wdt();
  }
 while (retry);

 up_init_global();
 output_bit(LED_ON,1);

 // determine and set RS232 baudrate:
   {
   if (!input(BAUD1200)) set_uart_speed(1200);
   if (!input(BAUD2400)) set_uart_speed(2400);
   if (!input(BAUD9600)) set_uart_speed(9600);
   if (!input(BAUD19200)) set_uart_speed(19200);
   if (!input(BAUD38400)) set_uart_speed(38400);
   if (!input(BAUD115200)) set_uart_speed(115200);
   }
  printf("\n\r---- TWO WIRE <---> RS232 BRIDGE ----\n\r");
  printf("MODE: 100K\n\r");
  printf("OUTPUT-FORMAT: DECIMAL\n\r");
  printf("PULL-UPs: 2K\n\r");

  delay_ms(100);
 for(;;)
  {
  disable_interrupts(GLOBAL);
  if (kbhit()) getc();
  if (kbhit()) getc();
  if (kbhit()) getc();
  FERR = 0; OERR = 0; CREN = 0; CREN = 1; // re-initialize USART
  RCIF = 0; TXIF = 0; INTF = 0;
  enable_interrupts(GLOBAL);
  enable_interrupts(INT_RDA); // RS232 int allowed
  up_init_ser_buffer();
  ser_buffer_pos = 0;
  ser_buffer_cmd_ready = 0;
  while ( (!ser_buffer_cmd_ready) && (ser_buffer_pos <= SER_BUFFER_LENGTH) );
  // sequence is now in ser_buffer - start working:
  disable_interrupts(INT_RDA);
  if (ser_buffer_pos < 3)
   error_handler(0x10);
  else
  if (ser_buffer_pos >= SER_BUFFER_LENGTH)
   error_handler(0x11);
  else
  if (ser_buffer[0] == 'C')
   up_cmd_c();
  else
  if (ser_buffer[0] == 'F')
   up_cmd_f();
  else
  if (ser_buffer[0] == 'U')
   up_cmd_u();
  else
   if (!up_cmd_interpreter())
    printf("\n\rOK\n\r");
  }   // end of endless loop
 }  // end of main()
//----------------------------------------------------------------------

```