Sony S-Link gateway

Hardware layer, waveforms

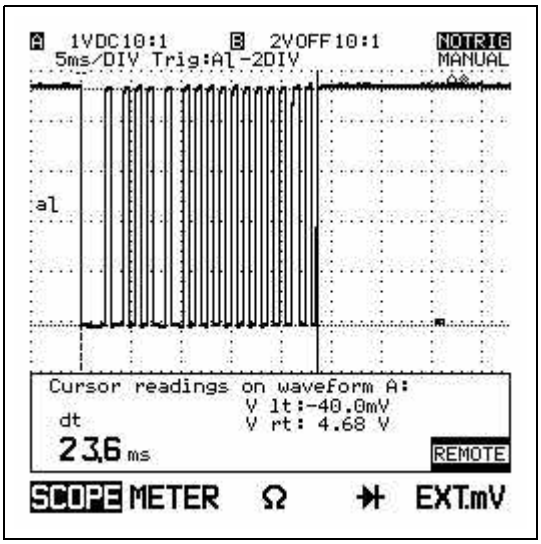
S-Link uses a two wire connection. 3.5mm mono jack connectors connect to Sony S-Link compatible components.

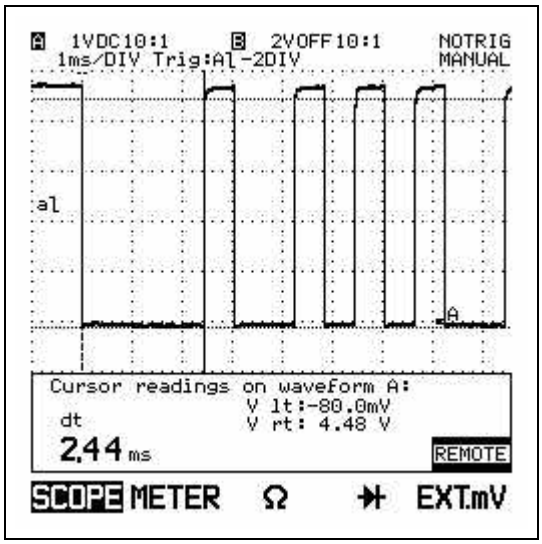
In idle state (99% of the time) the line is hold at 5V by internal pull up. In a short circuit condition current is limit to approx. 44mA. Any participant in communication must tie the line low for multiples of 600uS. This is the internal timebase for all S-Link communication. We differentiate four states:

- Synchronisation
  holding the line low for 2400uS indicates the beginning of new transmission
- Zero bit
  holding the line low for 600uS indicates a 0
- One bit
  holding the line low for 1200uS indicates a 1
- Delimiter
  between Sync, Ones and Zeros, the line must be released (return to high level) for 600uS

Synchronisation indicates the beginning of a new transmission. It does not precede every byte. Databits are organized as bytes (8 bit). The most significant bit (MSB) is transmitted first. A transmission consists of 2 ore more bytes, depending on the type of command.

The following waveforms show the message sent by a CDP-CX250 after pressing the play button.





Protocol layer

Most of this work has been done by others. A list of the commands discovered by BigDave you'll find at

http://www.cc.gatech.edu/people/home/bigdave/cdplayer/control-a1.txt

```
Following a part of BigDaves list 'control-a1.txt'
-----------------------------------------------------------------------
Status messages sent by the player

Now the codes themselves (written in hex unless otherwise noted):

The first byte tells the controller which CD player is talking:
98 = CDP-1
99 = CDP-2
9A = CDP-3

Bytes 2 onward tell you what information or command is being given.
When they are commands:
00 = Play
01 = Stop
02 = Pause
06 = Moving the CD Carosel
08 = Ready
0C = 29 seconds left on current track (Displayed during play)

When they are "Status" or information bytes:
18 = Door Open
2E = Power On
2F = Power Off
50 <Disc#> <Track#> <Length min> <Length sec> = Playing Track
      - <Disc#>: 1 byte  Discs 1-99 are Binary Coded Decimal (BCD)
                         Discs 100-200 are HEX-54d
      - <Track#>: 1 byte  Tracks are BCD
      - <Length minutes>: 1 byte  Minutes are BCD
      - <Length seconds>: 1 byte  Seconds are BCD

52 <Disc#> = Displaying on front panel the track/time/memo info for <Disc#>
      - <Disc#>: 1 byte  Discs 1-99 are BCD
                         Discs 100-200 are HEX-54d

54 <Disc#> = Retrieving <Disc#>, or Loading <Disc#>
      - <Disc#>: 1 byte  Discs 1-99 are BCD
                         Discs 100-200 are HEX-54d

58 <Disc#> = Retrieved <Disc#>, or Loaded <Disc#>
      - <Disc#>: 1 byte  Discs 1-99 are BCD
                         Discs 100-200 are HEX-54d

61 <Disc Capacity> <something else> = Tells the controller something
                                      about the CDplayer itself.
                                      I call it the "CD Model
                                      Identifiers."
      - <Disc Capacity>: 1 byte  100 Disc Player is 00 hex
                                 200 Disc Player is FE hex
                                 50+1 Disc Player (I am speculating
                                    it will be 50 or 51 hex)
      - <Something else>: 1 byte  usually 0B hex
```

```
70 <00hex> <CD Playing status> <00hex> <Disc#> <00hex>
      - <00hex>: 1 byte  Could mean something...but for me it never
                         changes.
      - <CD Playing status>: 1 byte (divided into 2 half bytes)
            - Half byte 1: (4 Most Significant bits) b1b2b3b4
                  - bit 1: 0 = Scanning Discs (happens when you
                             open, then close the CD door)
                           1 = Discs known (knows which discs
                             are loaded and which are not)
                  - bit 2: 0 = Play mode: 1 Disc
                           1 = Play mode: All Discs
                  - bits 3,4: 00 = Play mode: Repeat Off
                              01 = Play mode: Repeat All
                              10 = Play mode: Repeat 1
            - Half byte 2: (4 Least Significant bits) b1b2b3b4
                  - bits 1-4: 0000 = Normal
                              0001 = Shuffle
                              0010 = Programm



-------------------------------------------------------------------------
Format for commands to send to the CD Player (in hex unless otherwise
noted).

The first byte tells which of the 3 possible CD players to execute
the command.  The format is as follows:

90 = Sends command to CD-1
91 = Sends command to CD-2
92 = Sends command to CD-3

Bytes 2 onward give the specific command to the CD player

00 = Play
01 = Stop
02 = Pause
03 = Toggle Pause
50 <Disc#> <Track#> = Play the song on Disc#, Track#
      - <Disc#>: 1 byte  Discs 1-99 are BCD
                         Discs 100-200 are HEX-54d
      - <Track#>: 1 byte  Tracks are BCD
-------------------------------------------------------------------------
NOTES:

<Disc#>:  1 byte can have 256 unique numbers.  Sony ignores disc 00
        on the 200 Disc Players.  Disc 100 on the 100 Disc Players
        is represented as 00 hex.  Also, their "hacked" format for
        200 discs (their format for the 100 disc players is simple,
        just BCD) allows a total of 201 possible discs.  This
        explains Discs 100-200 being equal to the
        hex value - 54(base 10) while Discs 1-99 are in BCD.  The
        algorithm to take the byte given for disc# and printing it
        on a computer screen in base 10 is as follows:

        //Comment: (hb1 = 4MSB and hb2 = 4LSB of the byte)
        if (hb1 > 9)
           print("Disc #" + (16*hb1 + hb2 - 54))
        else
           print("Disc #" + (10*hb1 + hb2))

<Track#>,<Length min>,<Length sec>: Simple BCD.  Algorithm to print
        in base 10 is as follows:

        print( 10*hb1 + hb2 )

Let's say that CD-1 is playing Disc 148, Track 3, which is 3m 48s
```

```
long.  The output from the player is (hex): 98 50 CA 03 03 48
Now for Disc 62, Track 14, which is 2m 14s long: 98 50 62 14 02 14

When the unit is first given power, the following information is
given:

-Power Off
-CD Model Identifiers: <Disc Capacity> 0B

When the On button is first pressed, the following information is
given:

-Power On
-Retrieved (or Loaded) Disc Number <Disc#>
-Retrieved (or Loaded) Disc Number <Disc#>
-CD Model Identifiers: <Disc Capacity> 0B
-Displaying on Front Panel the time/track/memo info for: <Disc#>
-Ready



---------------------------------------------------------------------
```

The following commands are discovered by myself, using a gateway described later in this document.

Commands (Byte 2)

08 = next track
09 = next track

20 = Display off
21 = Display on

25 = continous status output
26 = stop outputing status

2e = Power on
2f = Power off

Status (Byte 2)

0E = Power is Off

The gateway

The gateway should fit the following requirements:

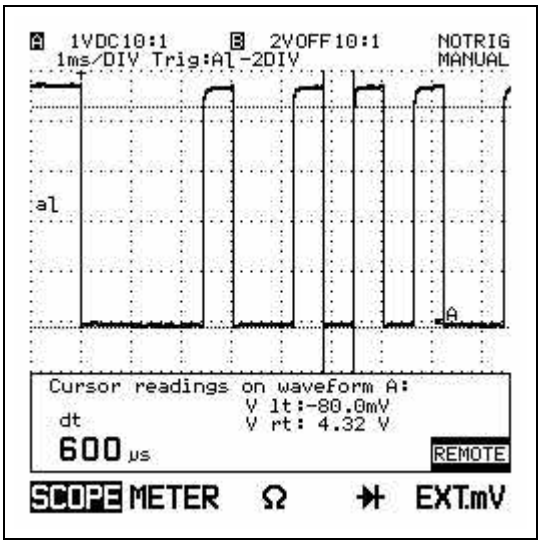
- Translation from S-Link coding to standard 8bit characters (with start and stop bit) and vice versa.
- Standalone operation without a PC. Status information from player are buffered, so the host can request them at any time.
- Flexible host interface, RS232 and/or RS485
- S-Link isolated by opto couplers. This prevents noise from grounding loops between audio equipment and PC.
- Supply from externel AC/DC adaptor.

My target system is a DS80520 uC from DALLAS Semiconductor. The DS87520 is a powerful 8bit controller based on Intels 8051 core.

Features:

- 2 Full duplex serial ports
- 3 Timers 16bit
- 5 external interrupts
- Watchdog / Power on reset
- 1kB SRAM on chip
- 16kB (E)PROM on chip

Concepts



Since the delimiter between any line low state is constant, it is not necessary to detect falling and rising edges. The delay between two rising edges can be interpreted as Sync, Ones and Zeros.

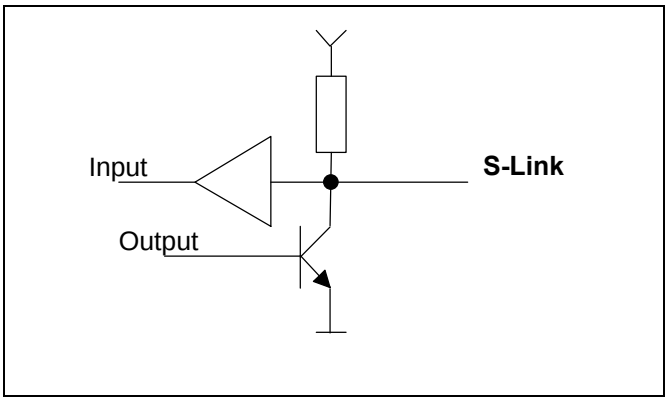
A delay of more than 2700uS (4.5 * 600uS) is always a synchronisation. Ones are longer than 1500uS and Zeros must have at least 900uS.

The count of databits is always a multiple of 8. A vioalation of this rule indicates an error.

S-Link has no bus arbitration logic. Messages can be sent at any time by any participant. Bus collision may occur. Because there is not much traffic, this happens extremly rare. Checking the receiver state immediately before starting a new transmission prevents from collisions in nearly all cases. Verification is the Job of the application programm or the operator.

Hardware

Data transmission uses 5V TTL levels. An open collector output with an optional pull up resistor is all you need to send data over the S-Link. The IO stage of the 87C520 fits this requirements directly. The same IO could be used for monitoring the line for receiving.



But why using 2 IO pins for input and output ? Opto couplers for isolating the S-Link from supply and other links are working unidirectional. If your PC is grounded and the CD is connected to an Amp which itself is wired to a tuner with connection to an antenna somewhere in town - then isolation might be a good idea.

Software

Sending and receiving from/to S-Link and RS232 is handled in interrupt service routines. The S-Link receivers saves all incoming characters in buffer. After a timeout (50mS) from the last data bit, a flag tells the main program that a message has arrived and is available in the input buffer. The main program copies the buffer to the RS232 output queue und activates the sender. Every data byte is sent as two nibbles in ASCII-HEX notation. Value 255 would be sent as string ‘FF’. Transmission string is terminated by a CR.

Communication in the other direction works in a similar manner. The RS232 receiver (interrupt handler) reads a CR terminated ASCII-HEX string from the serial port, converts to binary data, stores them in a buffer and informs the main program. The main program copies the buffer to the S-Link output queue und activates the sender. 50mS after the last incoming data bit from S-Link the sender starts transmission.

```
// Author   : Rolf Eigenheer                                    rolf_eigenheer@bluewin.ch
// Compiler : Keil C51
// Target   : DS87C520 @14,7456MHz

#pragma CODE DEBUG INTVECTOR OBJECTEXTEND SYMBOLS CODE SMALL PL(10000) PW(120) ROM(LARGE)
#pragma NOINTPROMOTE

#include <ABSACC.H>
#include <REG520.H>
#include <CTYPE.H>
#include <INTRINS.H>


code char *Logo1 = "+--------------+" \
                   "|    S-LINK    |" \
                   "|   Gateway    |" \
                   "| RS232-RS485  |" \
                   "+--------------+" \
                   "|        V0.10 |" \
                   "| Er /21.02.98 |" \
                   "+--------------" ;
code char *Logo2 = "+--------------+" \
                   "|       (c)1998|" \
                   "|Rolf Eigenheer|" \
                   "|CH8462 Rheinau|" \
                   "+--------------+";


/*************************************************************************
PIN Definitionen
*************************************************************************/

sbit _LED1      = P1^0;
sbit _LED2      = P1^1;
sbit RXD1       = P1^2;
sbit TXD1       = P1^3;
sbit INT2       = P1^4;
sbit _INT3      = P1^5;
sbit INT4       = P1^6;
sbit _INT5      = P1^7;

sbit RXD0       = P3^0;       // RS232 RxD  use MAX233 to convert levels to +/-12V
sbit TXD0       = P3^1;       // RS233 Txd  use MAX233 to convert levels to +/-12V
sbit _INT0      = P3^2;
sbit _INT1      = P3^3;
sbit SDA        = P3^6;
sbit SCL        = P3^7;

#define SWITCHES    ~P2

#define _TXEN_0      T0
#define SLINK0_OUT INT4       // Output to S-Link           (use opto couplers)
#define SLINK0_IN  INT0 // Inverted Input from S-Link       (use opto couplers)
                        // Inversion is necessary because Int0 triggers on
                        // the falling edge. We are interested on S-Link
                        // rising edges.


/*#########################################################################
#########################################################################*/


code  unsigned char Hex[0x10] = {0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
                                 0x38,0x39,0x41,0x42,0x43,0x44,0x45,0x46};



// RS232 communication variables
unsigned char       RxCount  = 0;
unsigned char       TxCount  = 0;
xdata unsigned char RxBuf[0x100];
xdata unsigned char TxBuf[0x100];

// S-Link communication variables
unsigned char       InCount  = 0;
unsigned char       OutCount = 0;
xdata unsigned char InBuf[0x100];
xdata unsigned char OutBuf[0x100];
bit                 StartSending = 0;
```

```
/*************************************************************************
Interrupthandler der Serieschnittstelle   LoPrio
*************************************************************************/
#pragma rb(1)

void HandleLink0 () interrupt 4 using 1 {    /* Handles Rx and Tx Interrupts */
  static unsigned char RC       = 0;
  static unsigned char TC       = 0;
  static bit           Rx2ndNib = 0;
  static bit           Tx2ndNib = 0;

  if(_testbit_(TI)) {
    if(TC < TxCount) {
      if(Tx2ndNib)
        SBUF = Hex[TxBuf[TC++] & 0x0f];             // Send lower Nibble
      else
        SBUF = Hex[TxBuf[TC] / 0x10];        // Send higher Nibble
      Tx2ndNib = !Tx2ndNib;
      }
    else {
      if(TxCount)
        SBUF =  0x0d;
      Tx2ndNib = 0;
      TC       = 0;
      TxCount  = 0;
      }
    }

  if(_testbit_(RI)) {
    if(SBUF == 0x0d) {
      RxCount  = RC;
      Rx2ndNib = 0;
      RC       = 0;
      }
    else {
      if(isxdigit(SBUF)) {
        if(Rx2ndNib)
          RxBuf[RC++] |= toint(SBUF);        // lower Nibble
        else
          RxBuf[RC] = toint(SBUF) * 0x10;    // higher Nibble  (first)
        Rx2ndNib = !Rx2ndNib;
        }
      }
    }
  }


void HandleLink1 () interrupt 7 using 1 {    /* Handles Rx and Tx Interrupts */
  if(_testbit_(TI_1)) {
    }
  if(_testbit_(RI_1)) {
    }
  }

#pragma rb(0)




/*************************************************************************
Link0 Intr0 Timer0 Interupt   HiPrio
*************************************************************************/
#define TSync   2950
#define TOne    1845
#define TZero   1110

#pragma rb(2)

unsigned char BitMask   = 0x80;
unsigned char ByteCount = 0x00;
bit           TimedOut     = 0;
bit           Receiving    = 0;
bit           Sending      = 0;


ProcessMsg() {
  if(ByteCount && (BitMask == 0x80)) {
    InCount = ByteCount;
    }
  ByteCount = 0x00;
  BitMask   = 0x80;
  }
```

```
void HandleSLinkTimeOut() interrupt 1 using 2 {             // timer0
  if(!Sending) {
    TimedOut  = 1;
    ProcessMsg();
    Receiving = 0;
    EX0       = 1;            // enable Receiver
    }
  if(!SLINK0_OUT) {           // if was a data bit
    SLINK0_OUT = 1;           // GAP
    TH0 = 0xFD;               // 600uS until overflow
    TL0 = 0x1D;
    }
  else if(StartSending) {
    if(SLINK0_IN) return;       // if link is busy return
    if(SLINK0_IN) return;
    if(SLINK0_IN) return;
    if(SLINK0_IN) return;
    if(SLINK0_IN) return;
    SLINK0_OUT   = 0;         // Syncbit
    TH0       = 0xF4;         // 2400uS until overflow
    TL0       = 0x79;
    ByteCount = 0x00;
    BitMask   = 0x80;
    Sending      = 1;
    StartSending = 0;
    EX0          = 0;         // disable Receiver
    }
  else if(Sending) {
    if(ByteCount < OutCount) {       // if was Gap
      SLINK0_OUT = 0;                // mark
      if(OutBuf[ByteCount] & BitMask) {
        TH0 = 0xFA;           // 1200uS until overflow
        TL0 = 0x3C;
        }
      else {
        TH0 = 0xFD;                  // 600uS until overflow
        TL0 = 0x1D;
        }
      BitMask >>= 1;
      if(BitMask == 0x00) {
        BitMask = 0x80;
        ByteCount++;
        }
      }
    else {
      ByteCount = 0x00;
      BitMask   = 0x80;
      OutCount  = 0;
      Sending   = 0;
      EX0       = 1;          // enable Receiver
      }
    }
  }


void HandleSLinkIntr () interrupt 0 using 2 {       // int0
  unsigned int TimerVal;
  if(!Sending) {
    TR0 = 0; TimerVal = 0x100*TH0 + TL0; TR0 = 1;
    if(INT0) return;                        // return if only a spike
    if(INT0) return;
    if(INT0) return;
    if(INT0) return;
    if(INT0) return;
    if(TimedOut || (TimerVal > TSync)) {     // is Synchronisation
      TH0 = TL0 = 0x00; TF0 = 0;
      TimedOut  = 0;
      ProcessMsg();
      Receiving = 1;
      }
    else if(TimerVal > TOne) {                      // is bit One
      TH0 = TL0 = 0x00; TF0 = 0;
      InBuf[ByteCount] |= BitMask;
      BitMask >>= 1;
      if(BitMask == 0x00) {
        BitMask = 0x80;
        ByteCount++;
        }
      }
```

```
    else if(TimerVal > TZero) {                    // is bit Zero
      TH0 = TL0 = 0x00; TF0 = 0;
      InBuf[ByteCount] &= ~BitMask;
      BitMask >>= 1;
      if(BitMask == 0x00) {
        BitMask = 0x80;
        ByteCount++;
        }
      }
    }
  }


#pragma rb(0)



/*************************************************************************
Timer2 Interupt   LoPrio    0.25mS
*************************************************************************/
#pragma rb(1)

void Timer2() interrupt 5 using 1 {
  static unsigned int TCount = 0x00;
  TF2 = 0;
  if(++TCount >= 400) {  /*100mS */
    TCount = 0;
    _LED1 = !_LED1;
    }
  }

#pragma rb(0)



/*************************************************************************
Initialisierung der Schnittstelle und der Timer
*************************************************************************/

void InstallRS485(unsigned char Reload, bit Bit9) {
  TCON &= 0x0f;                                /* Stop Timer0 Timer1 */

  ES   = 0;                                    /* Disable Serial Interupt */
  PCON &= 0x7f;                                /* Reset SMOD */
  SCON =  0x40;                                /* Serialport Operating Mode */
  SM0  =  Bit9;                                /* 9bit */
  SM2  =  0;
  TB8  =  0;
  REN = 1;                                     /* Enable RS232 Receiver */
  PS  = 0;                                     /* RS232 low Priority */
  ES  = 1;                                     /* Enable RS232 Interupt */

  ES_1 = 0;                                    /* Disable Serial Interupt */
  WDCON &= 0x7f;                               /* Reset SMOD_1 */
  SCON1 =  0x40;                               /* Serialport Operating Mode */
  SM0_1 =  Bit9;                               /* 9bit */
  SM2_1 =  0;
  TB8_1 =  0;
  REN_1 = 1;                                   /* Enable RS232 Receiver */
  PS_1  = 0;                                   /* RS232 low Priority */
  ES_1  = 1;                                   /* Enable RS232 Interupt */

  ET0  = 0;                                    /* Disable Timer0 Interupt */
  ET1  = 0;                                    /* Disable Timer1 Interupt */
  TMOD =  0x21;                                /* Timer0 16Bit - Timer1 Auto-Reload */
  TH1 = TL1 = Reload;                          /* Reload-Value Timer1 */
  TH0 = TL0 = 0x00;                            /* Reset Timer0 */
  PT0 = 0;                                     /* Timer0 low Priority */
  PT1 = 0;                                     /* Timer1 low Priority */
  TR0 = 1;                                     /* Start Timer0 (free running 16bit) */
  TR1 = 1;                                     /* Start Timer1 (Baud Rate Generator) */
  }



void InstallSLink() {
  PX0 = 1;                                     /* Ext0 High Priority */
  IT0 = 1;                                     /* Ext0 Edge Triggered */
  EX0 = 1;                                     /* aktiviere ExtInt0 */
  PT0 = 0;                                     /* Timer0 low Priority */
  ET0 = 1;                                     /* Enable Timer0 Interupt */
  TR0 = 1;                                     /* Start Timer0 (free running 16bit) */
  }
```

```
unsigned int InstallTimer(float fosz, float interval) {     /* Installs a periodic Interrupt
*/
  unsigned int reload;
  reload = (0xffff - (interval * (fosz /12)));
  RCAP2L = reload;
  RCAP2H = reload >> 0x08;
  T2CON = 0x04;
  PT2   = 0;
  ET2   = 1;
  return(reload);
}

#define ENABLERTCLK   ET2 = 1;
#define DISABLERTCLK  ET2 = 0;



/***************************************************************************
Allgemeine Funktionen
***************************************************************************/


void SwitchOnChipRAM(bit On) {
  PMR &= ~DME1;    /* 0400H-FFFF EXTERN */
  if(On)
    PMR |= DME0;   /* 0000H-03FF INTERN */
  else
    PMR &= ~DME0;  /* 0000H-03FF EXTERN */
  }

void SwitchALE(bit On) {
  if(On)
    PMR &= ~ALEOFF;
  else
    PMR |= ALEOFF;
  }

#pragma DISABLE
void InitWatchdog() {   /* ca. 500mS Timeout */
  CKCON |= WD1;
  CKCON &= ~WD0;
  TA  = 0xaa;
  TA  = 0x55;
  RWT = 1;
  TA  = 0xaa;
  TA  = 0x55;
  EWT = 1;
  }

#pragma DISABLE
void WDTrigger() {
  TA  = 0xaa;
  TA  = 0x55;
  RWT = 1;
  }



/*-------------------------------------------------------------------------
Timing Functions  DS87C520 Optimized   14.7456MHz
-------------------------------------------------------------------------*/

void Delay10uS (unsigned int DTime) {   /* 10uS Delay  bei 14.xxMHz */
  while(DTime--) {
    _nop_(); _nop_(); _nop_();
    _nop_(); _nop_(); _nop_();
    _nop_(); _nop_(); _nop_();
    _nop_(); _nop_(); _nop_();
    _nop_(); _nop_(); _nop_();
    _nop_(); _nop_(); _nop_();
    _nop_(); _nop_(); _nop_();
    _nop_(); _nop_(); _nop_();
    }
  }

void Delay1mS (unsigned int DTime) {
  while(DTime--) {
    WDTrigger();
    Delay10uS(100);
    }
  }
```

```
/*############################################################################
############################################################################*/


Reset() {
  SwitchALE(0);
  SwitchOnChipRAM(1);
  InitWatchdog();
  InstallRS485(0xff,0);
  InstallTimer(14.7456e6,0.25e-3);            /* 0.25mS */
  InstallSLink();
  EA = 1;
  }


main() {
  Reset();

  while(1) {
    WDTrigger();
    if(RxCount) {
      OutCount = 0;
      while(RxCount--) {
        OutBuf[OutCount] = RxBuf[OutCount];
        OutCount++;
        }
      RxCount      = 0;
      StartSending = 1;
      }

    if(InCount) {
      TxCount = 0;
      while(InCount--) {
        TxBuf[TxCount] = InBuf[TxCount];
        TxCount++;
        }
      InCount      = 0;
      TI           = 1;
      }
    }
  }
```