



# TDA 5051A DEMOBOARD

## *Application Note Update*



**TDA5051A ↔ uC 8051**

**Master / Slave Power Line  
Communication Example**

*Eric MICHAT - June 1998*

### ***IMPORTANT WARNING !***

- Application diagrams and softwares are intended for an evaluation purpose only.
- To avoid any personal injury and damages to the test equipment, always use an insulation 1/1 power transformer between mains and test bench.
- All instrument's grounds must be connected to the earth terminal of the mains.
- Performing such experiments is understood to be fully at the risk of the user.

### ***MISE EN GARDE IMPORTANTE !***

- Les schémas d'application et logiciels sont uniquement destinées à un usage de laboratoire ou d'évaluation.
- Pour éviter tout risque corporel et dommage matériel, toujours utiliser un transformateur d'isolement de rapport 1/1 entre le réseau électrique et les maquettes.
- La masse électrique de tout le matériel de mesure doit impérativement être connectée à la terre de l'installation électrique.
- Dans tous les cas, les expérimentations sont faites sous l'entière responsabilité de l'utilisateur.

**© MICHAT ELECTRONIQUE 1997-98 for PHILIPS SEMICONDUCTORS**

**No part of this application note may be reproduced or transmitted, in any form or by any mean, without the prior permission of PHILIPS SEMICONDUCTORS.**

**Electric diagrams and Program codes included in this note have been tested with care, but are not guaranteed for any particular purpose.**

## Summary

<b>1. POWER LINE DATA COMMUNICATION .....</b>	<b>4</b>
1.1. INTRODUCTION AND FEATURES.....	4
1.2. RULES OF OPERATION .....	4
1.3. MORE ABOUT THE PROTOCOL.....	5
<b>2. ELECTRIC DIAGRAMS AND CONNECTION TO THE TDA5051A DEMOBOARD .....</b>	<b>8</b>
2.1. ELECTRIC DIAGRAM OF THE MASTER .....	8
2.2. ELECTRIC DIAGRAM OF THE SLAVE .....	9
<b>3. UC 8051 SOFTWARE .....</b>	<b>10</b>
3.1. MASTER SOFTWARE.....	10
3.2. SLAVE SOFTWARE.....	11
<b>4. TEST BENCH .....</b>	<b>13</b>
4.1. TERMINAL CONFIGURATION.....	13
4.2. TEST BENCH SETUP.....	13
4.3. « USER SIDE » PROTOCOL AND SYNTAX .....	14

***Appendix 1 Master ASM51 Source Code***

***Appendix 2 Slave ASM51 Source Code***

***Appendix 3 Electric Diagram of the Master***

***Appendix 4 Electric Diagram of the Slave***

## 1. Power Line Data Communication

### 1.1. Introduction and features

This application note intends to demonstrate the use of a simple protocol to carry out a reliable data transmission on Power lines with a 8X51-based microcontroller.

The power line communication protocol is built from a simple Master-Slave architecture; the network consists of one Master and at least, one (or more) Slave.

An « user side » protocol, based on ASCII codes, is required to control the Master operation. In such a way, any « serial » data terminal (e.g. **Windows 3.1® Terminal** or **Windows 95® Hyperterminal**) can be used to transmit orders to the slaves.

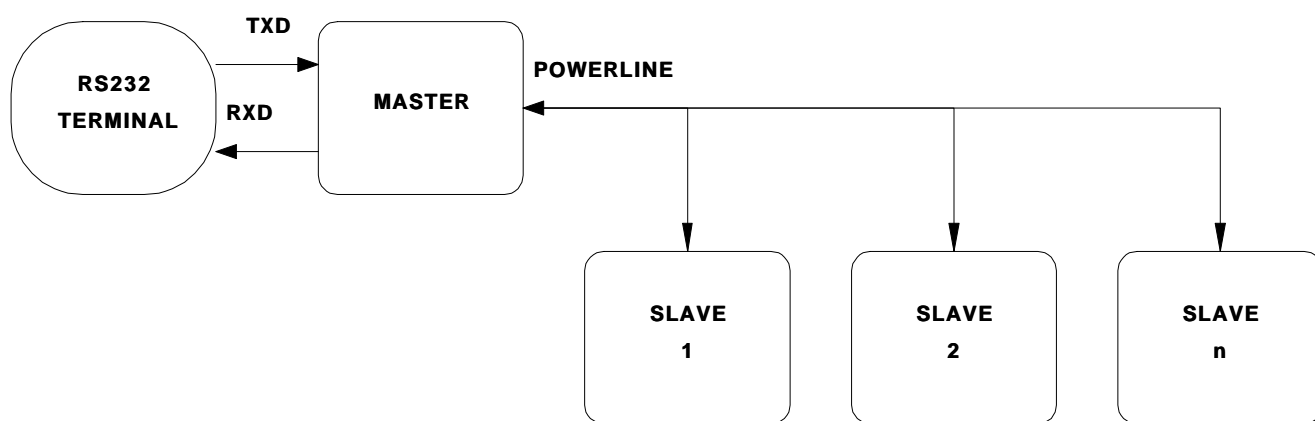


Fig. 1 - Network Architecture

### 1.2. Rules of operation:

a - Each Slave of the network receives the data frame coming from the Master, but only one of them is able to decode and acknowledge the transmission. This is done by using a unique address code for a given slave.

b - Each data frame contains a start byte, 12 data bytes and a checksum byte. Each byte has its own parity bit. A frame is acknowledged if the following conditions are satisfied:

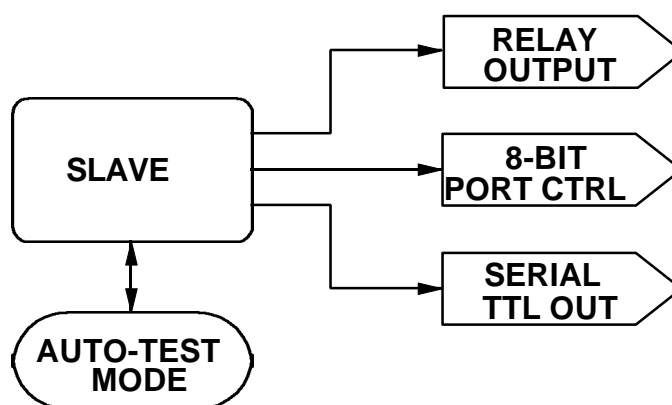
- the start byte is correct
- the 14 parity bits are correct
- the checksum byte is right
- the received address code matches with the Slave address (read on DIL switches)

c - When the Master sends a data frame to a slave, it waits for the acknowledge at the end of the transmission. If no acknowledge occurs, it sends the same message again.

d - After performing 10 unsuccessful transmissions, the Master declares a «time out » error, meaning that the addressed Slave is absent or is not able to receive the frame.

**Each Slave can achieve the following functions:**

- a - A « self » communication test with the Master: in this mode, the Master sends 100 data frames to a chosen Slave and calculates the error rate (displayed on the Terminal).
- b - Relay ON/OFF control: an order coming from the Master controls the relay.
- c - 8-Bit Port control: the Master can control each bit of a 8051 port.
- d - ASCII Mode: the Master sends to a Slave a 8-byte ASCII string: this string may be displayed on a dot matrix display or can be used to control anotheruC application.

**Fig. 2 - Slave functions****1.3. More about the protocol**

One have to keep in mind that in such an application, there are two **different** protocols:

I - The **ASCII 'User side'** protocol, for data exchange between Master and Terminal.

The 'User side' protocol is very simple and will be explained in section 4.

It's a standard RS232 serial communication, using ASCII codes for Master control.

**Example:** if one ask for a « self » communication test with the Slave #1: the terminal has to send to the Master the ASCII string {\$1}. The Master replies {Transmission test in progress...} and later {Error Rate: 05%}.

II - The **'Powerline side'** protocol, designed for a safe data transmission on the mains.

The 'Powerline protocol' is much more complex, and cannot be entirely explained in this application note. It would require a lot of time and paper !

(All the ASM51 source code with comments is given at the end of the note).

However, the basic principles are explained bellow.

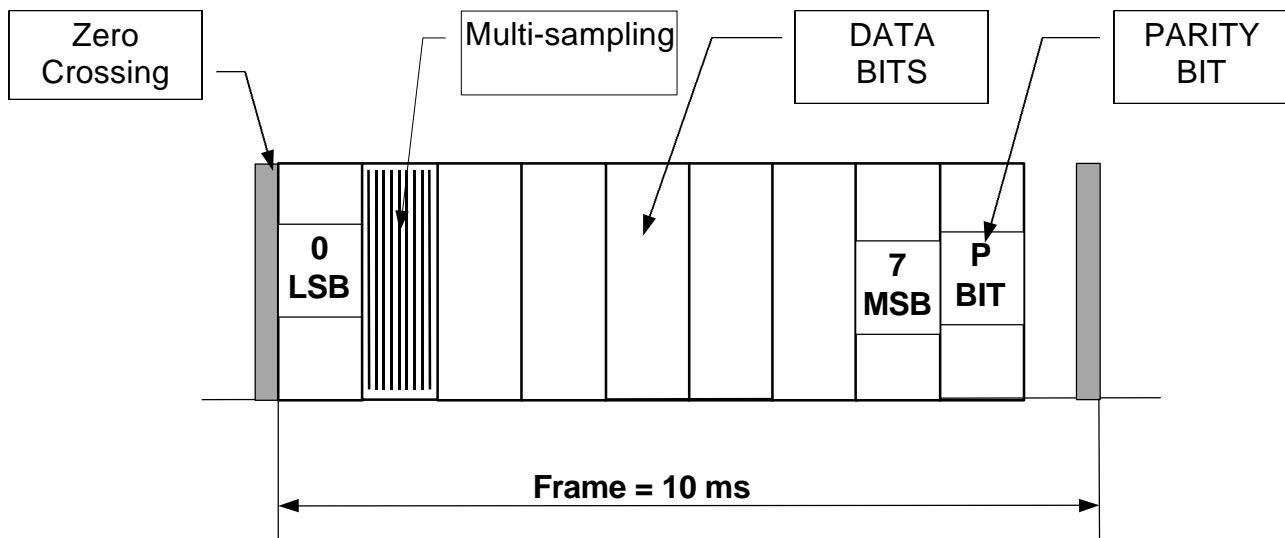
a- The protocol uses the zero-crossing of the 50Hz mains as a 'clock' in order to synchronize data transmission, like in a synchronous communication link. This is very efficient, because in such a case, the start of a frame is **absolutely sure**, even on noisy power line. **Of course, this kind of application operates ONLY on the SAME PHASE !**

b- A **Bit** is coded in **NRZ standard**. But, during reception, to improve the noise immunity, **each bit is sampled 9 times**. The bit level detection will depend on the number of samples at ONE or ZERO.

c- A '**frame**' consists of **8 bits (in NRZ code) plus a parity bit and a dead time**. The bit length is **1ms**, and a frame (9ms) is completely transmitted between 2 zero-crossing points of the mains. That means, for a 50Hz Powerline, a bit rate of  **$50 \times 2 \times 9 = 900$  Bits/Sec.**

d- A '**cluster**' consists of **14 frames: 1 start frame (1byte+Parity), 12 data frames (12\*(1byte+parity)) , 1 checksum frame (1byte+Parity)**. A cluster is error free if all 14 parity bits and the checksum byte are correct...

So, taking into account the error management, a **12 bytes** 'effective' data transfer is carried out in **140ms**.



**Fig. 3 - Typical BYTE FRAME**

The following figure shows a cluster transmission on the mains (successfully performed).

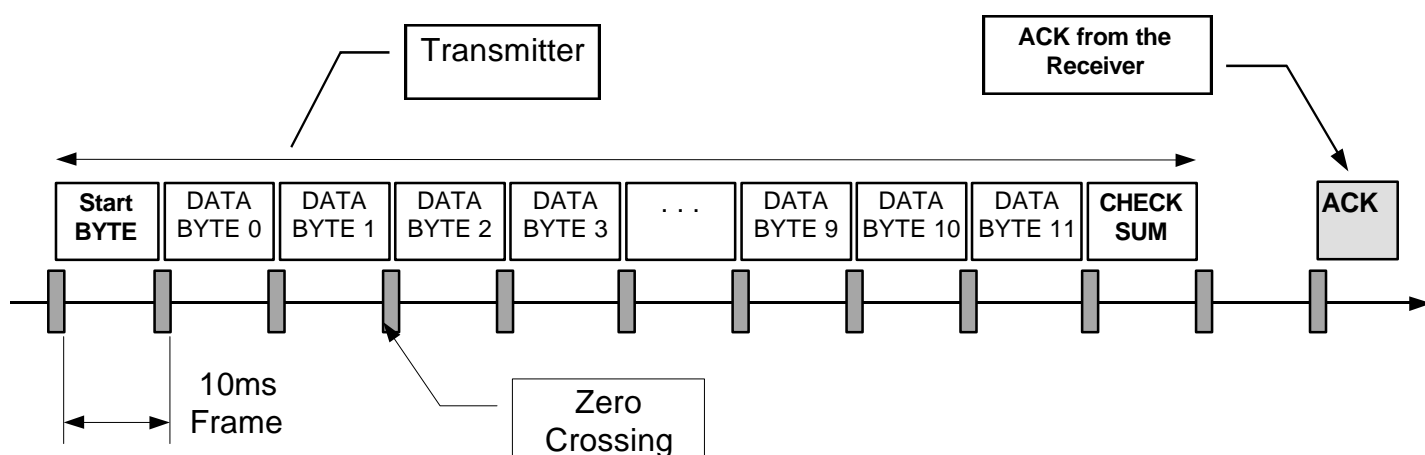


Fig. 4 - CLUSTER Transmission with acknowledgment

Start BYTE = 99h      DATA BYTE 0 = '&'      DATA BYTE 1 = '0' to '7' (Slave Address)

DATA BYTE 2 ... to ... DATA BYTE 11 = User ASCII codes (orders, strings, etc...)

In this example, the transmission has been successfully performed, meaning that the 14 parity bits and the checksum are correct.

In such a case, **if the received address matches with the « board » address code**, the Slave sends an acknowledge pulse after the second zero-crossing detection following the checksum byte, to inform the Master.

If the acknowledge pulse is absent, the Master repeats the same procedure, several times if required (10 max), to obtain an error free communication.

This may take a 'lot' of time, in noisy conditions, but this kind of protocols seems to be well suited for Powerline data communications, which usually don't require high transmission speed.

## 2. Electric diagrams and connection to the TDA5051A Demoboard

### 2.1. Electric Diagram of the Master

The complete electric diagram is given in Appendix 3.

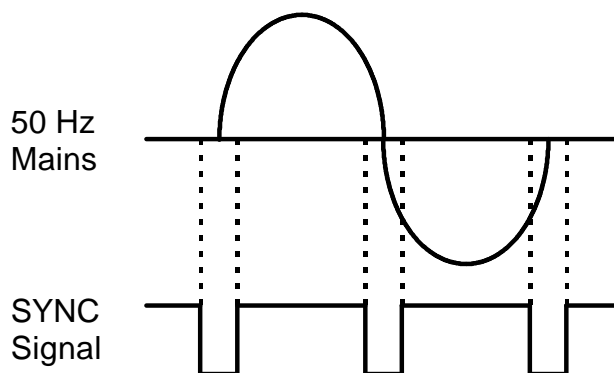
The TDA5051A demoboard is connected to the 80C51 microcontroller board by means of the 6-pin socket. The +5V DC Micro Switching Power Supply is used to feed the controller (IC2) and the others devices.

The CLK output of the demo-board is connected to the XT1 pin of the 80C51 (**Frequency**= $F_{osc}/2=7.37\text{MHz}/2=3.685\text{MHz}$ ).

**The software timing constants have been calculated for 50Hz operation ! If the application is intended to be used on a 60Hz mains, these parameters must be changed !**

The zero crossing detection circuit is fully insulated from the mains by using a TIL187 (OPT1) bi-directional optocoupler. A TTL signal is available at the SYNC pin of the detection stage (Collector of the BC557B).

This synchronization signal is used to trigger the INT0 pin of the 80C51 (IC2). In such a way, an internal interruption routine will define the start condition for **transmitting** a frame (or **sampling** a frame for a Slave).



**Fig. 5 - SYNC Signal**

The LTC1232 (IC3) is a WatchDog/Reset circuit which provides a safe RESET signal during power-up and prevents from any unexpected problems if the uC becomes uncontrolled.

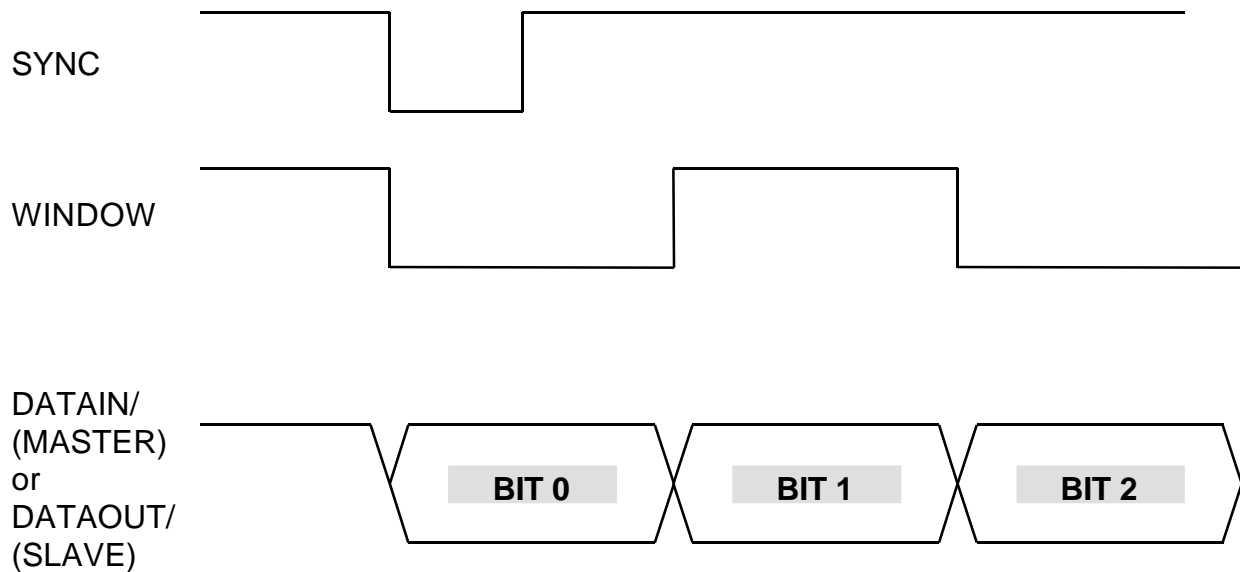
The Software must periodically apply a LOW pulse at the ST/ pin of the LTC1232, in order to reset the WatchDog time-out.

A standard MAX232 (IC1) (or equ.) RS232/TTL adapter is used to connect the TXD/RXD pins of the 80C51 to a SUB-D9 socket for PC terminal communication.

The WINDOW signal is a test point, that shows the «BIT-TIME» window, for both transmission (Master) and reception (Slave).

The edges of this signal represent the «bit» limits, as shown hereafter.





**Fig. 6 - WINDOW Signal**

A test push-button (PB1) is used to check the right operation of the TDA5051A and the zero-crossing detection circuit. When pressed, the software starts a carrier transmission (DATAIN/=LOW) and check the DATAOUT level of the TDA5051ADemoboard. Then, if a signal is received, the Green LED must flash slowly; in the other case, the RED LED indicates a problem.

## **2.2. Electric diagram of the Slave**

The diagram is given in the Appendix 4.

It is very similar to the Master electric diagram. The controller section and the zero-crossing circuit are the same.

The P1 port is connected to a 8-DIL Switch for Slave address coding. **P1.0, P1.1 and P1.2 bits are only in use** in this software version, providing 8 address codes (000 to 111).

The P3.4 pin is used to control a small (+5V DC coil) relay.

The TXD pin of the 80C51 is now connected to an ASCII dot matrix display (example: FUTABA M202SD01LA). In such a case, an external +5V DC/300mA power supply must be used for the display module.

**The UART setup for this output is: 1200 Bauds, 8 bits, No Parity, 1 Stop.**

The TXD TTL output may be used to control any other system having a serial port input, and being able to decode ASCII characters.

### 3. uC 8051 SOFTWARE

#### 3.1. MASTER Software

The complete 8X51 code is given in the **Appendix 1**

- The **INT0** is used to trigger the zero-crossing synchronization of the protocol, and the **Timer0** is used as a bit clock generator for the transmission on the Powerline.
- The **Timer1** is used as a Baud rate generator for the internal UART of the controller: **1200 Bauds, 7 bits, Parity EVEN, 1 STOP**

The software has three main parts:

#### I - Initializations and main loop:

The **main loop** selects the basic functions and manages the entire protocol, including:

- Serial Bytes reception from the PC terminal (ASCII)
- Selection of the MODE: Byte Transmission (start with a '&') , Auto-evaluation (start with a '\$') or Help (Start with 'H' or 'h').
- Wait for Acknowledgment
- Transmission counter (a message is sent 10 times max).

The Help file is small summary of available control codes.

#### II - « Powerline SIDE » main routines

**tx\_cluster**: transmission of the entire cluster (start byte + 12 data bytes + checksum) on the Powerline. The checksum is already calculated when **tx\_cluster** is called.

**tx\_one\_byte**: single byte transmission on the Powerline, with parity bit set (9 bits)

**tx\_bit**: Bit by Bit transmission on the Powerline, triggered by **TIMER0 interrupt** which defines the Baud rate (bit length=1ms).

**synchro**: Zero-crossing synchronization for transmitting a frame (9 bits), triggered by **INT0**.

**wait\_ack**: this routine detects the acknowledge pulse coming from the Slave board.

**fill\_cluster**: fills the RAM cluster with a predefined ASCII string (A to H) for the auto-test mode.

### III - « User SIDE » main routines

**rx\_cluster**: fills the RAM cluster with the serial data coming from the PC terminal.

**error\_to\_rs232, O\_to\_RS232, T\_to\_RS232, R\_to\_RS232** these messages indicate the state of the Master. See ASCII tables for explanations.

**in\_a , out\_a**: serial data in and serial data out for terminal communication.

**tx\_str**: transmission of an ASCII string to the terminal.

**test\_mode**: Auto-test when TEST push-button is pressed

**dcb\_to\_tx**: displays a DCB value on the terminal

**help**: displays the help file content on the terminal screen.

**dcb\_to\_ascii**: DCB to ASCII translation

#### 3.2. SLAVE Software

The complete 8X51 code is given in the **Appendix 2**

- The **INT0** is used to trigger the zero-crossing synchronization of the protocol, and the **Timer0** is used as a clock generator for the bit sampling.
- The **Timer1** is used as a Baud rate generator for the internal UART of the controller: **1200 Bauds, 8 Bits, No PARITY, 1 STOP.**

The software has three main parts:

#### I - Initializations and main loop:

The main loop manages the entire protocol, including:

- Powerline data sampling and decoding.
- Error management
- Acknowledge management
- Check Slave address and acknowledge if needed
- Decode and execute incoming orders from the MASTER.

## II - « Powerline SIDE » main routines

**rx\_cluster**: complete reception of (start byte + 12 data bytes + checksum), parity and checksum verifications, coming from the Powerline.

**wait\_byte, parity\_check**: single byte reception from the Powerline and parity test.

**rx\_bit**: Bit by Bit reception from the Powerline. The bit sampling is triggered by **TIMER0 interrupt**.

**synchro**: Zero-crossing synchronization for frame reception (9 bits), triggered by **INT0**.

**send\_ack**: this routine sends the acknowledge pulse to the transmitter.

## III - « User SIDE » main routines

**error\_to\_TTL**: indicates the error type (Parity or Checksum) if needed (send an error message to the dot matrix display)

**out\_a**: serial data out for serial communication with the dot matrix display.

**tx\_str**: ASCII string transmission to the display

**ctrl\_add, read\_address**: Compare the received address code with the Slave «local » address (DIL Switch).

**ctrl\_ident**: Decode the message coming from the Master: it can be a Relay control, an ASCII transfer to the display or a Port P2 control.

This routine selects the following sub-routines:

**relay**: read the code and select Relay ON or Relay OFF

**ascii\_to\_TTL**: 8-byte ASCII transmission to the display

**port\_ctrl**: Port P2 (bit by bit) control

**test\_mode**: Auto-test when TEST push-button is pressed

## 4. Test Bench

### 4.1. Terminal Configuration

As mentioned before, one can use any PC serial data Terminal to control the Master. (Windows 3.1® Terminal or Windows 95® Hyperterminal)

The Master SUB-D9 header has to be connected to a free PC COM port by means of a RS232 cable. **CONFIGURE FIRST your Terminal BEFORE Master Power-up !**

The following configuration must be defined:

Parameter	Value
Terminal Type	<b>TTY (Generic)</b>
Baud Rate	<b>1200</b>
Data Bits	<b>7</b>
Parity	<b>EVEN</b>
Stop Bit	<b>1</b>
Data Flux Control	<b>NONE</b>
Local Echo	<b>YES</b>

### 4.2. Test bench setup

#### \* FOR THE MASTER

- 1 - **Be sure of using an insulated 230V AC 50Hz power line for evaluation !**
- 2 - Configure the PC Terminal as explained before.
- 3 - Connect the PC COM port to the SUB-D9 connector of the Master board by means of a RS232 standard cable.
- 4 - Then, **power-up the Master Board** . If the Terminal Configuration is OK, the following message is displayed on the screen: '> **Waiting for RS232...** '. Type **H** on the keyboard: the Help File content appears.
- 5 - Press the TEST push button of the Master. The GREEN LED must flash slowly

#### \* FOR A SLAVE

- 1 - On a Slave Board, switch ON the P1.0, P1.1 and P1.2 DIL Switches (LOW level=ON=000). That defines the address '0' for this Slave.
- 2 - Connect the Slave to the **insulated 230V AC 50Hz Power line** .
- 3 - Press the TEST push-button. The GREEN LED of the Slave must flash slowly.

The couple Mater/Slave is ready to use.

### 4.3. « User Side » Protocol and Syntax

#### a - Transmission test mode

In this mode, the Master sends 100 times a predefined ASCII string (ABCDEFGH) to a chosen Slave. During this trial, the Master counts the number of successful transmissions and then, displays the error rate on the terminal screen.

Syntax of this instruction: **\$x** where 'x' is the Slave Address '0' to '7' Ex: **\$0**

Function	Type on the terminal keyboard ...	The Terminal Screen displays ...
Request Transmission TEST with Slave 0	<b>\$0</b>	> Transmission test in progress...
End of Test xx=00 to 99		> Error rate: xx% > Waiting for RS232 ...

#### b - Relay control

Syntax: **&xR-1-----** for relay ON  
**&xR-0-----** for relay OFF

where 'x' is the Slave Address '0' to '7' and '-' is any ASCII code.  
The complete string must be 12 ASCII characters long.

Function	Type on the terminal keyboard ...	The Terminal Screen displays ...
Request Relay ON for the Slave #0 ...	<b>&amp;0R-1-----</b>	> Sending data ...
... The Relay is ON		> Data received ! > Waiting for RS232 ...
Request Relay OFF for the Slave #0	<b>&amp;0R-0-----</b>	> Sending data ...
... The Relay is OFF		> Data received ! > Waiting for RS232 ...



**d - ASCII transmission to a Slave**

Syntax:      **&xW-aaaaaaaa**      **'a' is any ASCII code**

Where 'x' is the Slave Address '0' to '7', ' - ' is any ASCII code.  
The complete string must be 12 ASCII characters long.

Function	Type on the terminal keyboard ...	The Terminal Screen displays ...
Request ASCII transmission of the string 'EXAMPLE1' to Slave # 0 ...	&0W-EXAMPLE1	> Sending data ...
The Dot Matrix Display of the Slave receives 'EXAMPLE1'		> Data received ! > Waiting for RS232 ...

**For purchasing the « TDA5051A DEMOBOARD » with the complete application note, ASM51 source code and hex files (DOS Floppy Disk)**

**[http:// www.michat.com](http://www.michat.com)**

**or send a mail to:**

**[tda5051info@michat.com](mailto:tda5051info@michat.com)**



# **Appendix 1**

## **MASTER**

## **SOFTWARE**

```

; -----
;                               (C) ERIC MICHAAT 1998
;                               8051 uC / TDA5051 POWERLINE COMMUNICATION
;                               EVALUATION SOFTWARE for the Master
; -----
$MOD51 ; Uses 8x51 standard predefined labels
; ++++++
; CLOCKIN of 87C51 = 3.68 MHz (TDA5051 XTAL=7.37MHz)
; Serial RS232 IN/OUT 1200 Bds, 7 bits, 1 Stop, EVEN
; ++++++
; -----
;                               Constant Parameters
; -----
; -----
;                               RAM Variables
; -----
; TX DATA BUFFER
data_buffer equ 20h ; D7 D6 D5 D4 D3 D2 D1 D0
; TX FLAGS
flag_tx equ 21h ; POB 0 0 0 CE PE TB DE
; POB= Parity Bit of DATA_BUFFER (ODD)
; CE= CHECKSUM ERROR
; PE= PARITY ERROR
; TB= TEST BIT
; DE= DATA_BUFFER EMPTY
cluster_buffer equ 22h ; 12-byte TX Cluster buffer (22h...2dh)
cluster_check equ 2eh ; Cluster Checksum
slave_add equ 2fh ; Slave Address (ASCII '0' to '3')

; -----
;                               I/O Configuration
; -----
;
; p3.2 (INT0) Zero Crossing Detection
; p0.4 Bit Time Window (for test only)
; p0.5 DATA_IN TDA5051
; p0.6 DATA_OUT TDA5051
; TX Serial DATA RS232 OUT
; RX Serial DATA RS232 IN
; p0.1 RED LED
; p0.2 GREEN LED
; p3.5 Test Push Button
; p0.0 Watchdog Reset

; -----
;                               Interrupt Vectors
; -----
; Reset
org 0000
ajmp init

; INT0 - Zero Synchro
org 0003h
ajmp synchro

; Timer 0 - Bit TX Rate 100us
org 000bh
ajmp tx_bit

```

ORG 40H

```

init:      ;+++++
;
;               Initialisations
;+++++
mov       sp,#60h      ; Stack pointer
mov       psw,#0       ; Register BANK 0 in use
mov       flag_tx,#0   ; Clear all flags
mov       slave_add,#30h ; Slave address = '0'

acall    wait          ; Wait
clr      p0.5          ; Reset TDA5051 (DATAIN=0)
mov      b,#1
acall    wait3         ; Wait
setb     p0.5          ; DATAIN=1
setb     p3.5          ; BP input
mov      p1,#255       ; P1 input
mov      p2,#0         ; P2 output

;-----
;   UART init for 1200 Bauds, 7 bits, 1 Stop
;   Timer 1 is used as a Baud Rate generator
;   for the RS232 communication timing
;-----

mov      scon,#01000000b ; UART mode 1 - F variable
setb     scon.4          ; Enable reception
clr      scon.0
mov      pcon,#0        ; SMOD=0
mov      th1,#0f8h      ; 1200 Bds @ 3.68MHz
mov      tmod,#00100001b ; Timer 1 Mode Auto-load
setb     tcon.6         ; Run Timer 1

;-----
;   Timer 0 init - Powerline communication timing
;-----

mov      th0,#0e1h      ; Time base T=100us
mov      tmod,#00100010b ; Timer 0 Mode 8 bits auto
setb     tcon.4         ; Run Timer 0

;-----
;               Interrupts configuration
;-----

setb     ie.7           ; Enables IT...
setb     ie.0           ; Enable INT0
setb     ip.1          ; MAX priority is Timer 0
setb     tcon.0        ; INT0 sensitive on fal. edges
clr      tcon.1

```

```

;+++++
;
;                               MAIN LOOP
;+++++

main_loop: mov     dptr,#str_wait
          acall    tx_str           ; Message

          acall    in_a             ; Read RS232
          cjne     a,#'&',main_2    ; Cluster start ?
          sjmp     mode_1           ; YES

main_2:   cjne     a,#'$',main_3    ; Auto- eval. mode ?
          acall    in_a             ; Read Slave address
          mov      slave_add,a      ; Save
          sjmp     auto_eval

main_3:   cjne     a,'#h',main_4    ; Help ?
          acall    help
          sjmp     main_loop

main_4:   cjne     a,'#H',main_loop ; Help ?
          acall    help
          sjmp     main_loop

; -----
;                               RS232 to Powerline mode
; -----

mode_1:   acall    rx_cluster       ; 12-byte cluster reception
          ; from RS232
          acall    R_to_rs232      ; Cluster received !
          mov      r3,#10          ; R3 is a transmission counter
tx_again: acall    tx_cluster       ; TX Cluster on the Powerline

          acall    wait_ack        ; Wait for ACKNOWLEDGE from
          jnc      ack_received    ; the receiver
          acall    error_to_rs232  ; No acknowledge ! Error !
          acall    wait            ; Wait
          djnz    r3,tx_again      ; TX again - 10 times max.
          acall    O_to_rs232      ; Time-Out error - The slave does
          sjmp     main_loop       ; not respond !

ack_received:acall T_to_rs232      ; Transmission performed !
          sjmp     main_loop       ; Goto Main loop

; -----
;                               Transmission test (auto) mode
; -----

auto_eval: mov     dptr,#str_tx_test
          acall    tx_str           ; Message

          acall    fill_cluster    ; fill cluster with 41h...48h
          ; fill checksum with 24h
          mov      r1,#0           ; Reset clusters counter
          mov      r2,#0           ; Reset errors counter

tx_again_2:mov     b,#50           ; Wait before two transmissions
          acall    wait3
          cpl      p0.0           ; WatchDog Reset
          acall    tx_cluster      ; TX Cluster on the Powerline

```

```

inc      r1          ; Clusters counter
acall    wait_ack    ; Wait for ACKNOWLEDGE from
                    ; the Receiver
jnc      ack_rec_2    ; Transmission successfull !

mov      a,r2        ; One error !
add      a,#1
da       a           ; DCB adjust
mov      r2,a

ack_rec_2: cjne      r1,#99,tx_again_2 ; 100 clusters transmitted
mov      dptr,#str_error_rate
acall    tx_str      ; Message
mov      a,r2        ;
acall    dcb_to_tx   ; Readout Errors %
mov      a,#'%'
acall    out_a
mov      a,#10       ; LF
acall    out_a
mov      a,#13       ; CR
acall    out_a
sjmp    main_loop    ; Goto Main loop

; ++++++
; "POWERLINE SIDE" / TX DATA ROUTINES
; ++++++
;-----
; Cluster Transmission on Powerline
; -> Start byte (99h) + 12 bytes +checksum
;-----
tx_cluster:mov      a,#99h          ; Start byte
acall    tx_one_byte              ;
mov      r0,#cluster_buffer      ; Data pointer
next_byte: mov      a,@r0          ; Current byte
acall    tx_one_byte
inc      r0                       ; Increment pointer
cjne    r0,#cluster_check+1,next_byte
cpl     p0.0                       ; Reset WatchDog
ret

;-----
; Transmission of A+Parity (9bits) on the Powerline
;-----
tx_one_byte:mov     data_buffer,a    ; Load data_buffer
clr      flag_tx.7                 ; Clear POB
mov      a,psw                    ; Parity of Acc
anl     a,#1                       ; -> in Acc.0
rr      a                          ; -> in Acc.7
orl     a,flag_tx                  ;
mov      flag_tx,a                 ; -> in POB of Flag_tx
clr     flag_tx.0                  ; Reset flag Buffer Empty
setb    flag_tx.1                  ; Enable byte TX on INT0
wait_tx: jnb     flag_tx.0,wait_tx  ; Wait for byte TX
clr     ie.1                       ; Disables TIMER 0 INT
setb    ie.0                       ; Enable INT0
clr     flag_tx.1                  ; A byte has been
                                        ; transmitted
ret

```

```

; -----
;       Bit Transmission on Timer 0 Interrupt
;       A frame is (8 bits + 1 Parity bit)= 9 bits
; R7 ----> Transmitted bits counter      (in a FRAME)
; R6 ----> Time counter                  (in a BIT)
; -----
tx_bit:  push    acc
        push    psw

        cjne    r7,#9,tx_bit_2           ; Last bit of the frame ?
        setb    flag_tx.0                ; -> YES, Set Flag "BUFFER
EMPTY"

        setb    p0.5                      ; DATAIN=HIGH !
        pop     psw
        pop     acc
        reti

tx_bit_2:  cjne    r6,#0,tx_bit_3         ; Start of a bit ?
                                                ; -> YES
        cjne    r7,#8,tx_byte           ; Last bit of the byte ?
                                                ; -> YES, Tran.Parity bit
        jb      flag_tx.7,tx_one         ; Parity=1
        sjmp    tx_zero                  ; Parity=0

tx_byte:  mov     a,data_buffer           ; Read current bit
        rrc     a                          ;
        mov     data_buffer,a            ; Store byte
        jc      tx_one                    ; C=1 Transmit a ONE
tx_zero:  setb    p0.5                      ; C=0 Transmit a ZERO
                                                ; DIN=HIGH

        sjmp    tx_bit_3

tx_one:   clr     p0.5                      ; DIN=LOW
tx_bit_3:  inc     r6
        cjne    r6,#10,tx_bit_4         ; End of the bit ?
                                                ; YES
        mov     r6,#0                      ; Reset time counter
        CPL     p0.4                      ; Show bit time window
        inc     r7                          ; Next bit
tx_bit_4:  pop     psw                      ; Not the end of a bit
        pop     acc
        reti

;-----
;       Enable frame transmission on INT0 interrupt
;-----
synchro:  mov     r7,#0                      ; Reset bits counter
        mov     r6,#0                      ; Reset time counter
        jnb    flag_tx.1,sync_2; Do not TX a byte

        clr     ie.0                      ; Disable INT0 interrupt
        setb    ie.1                      ; Enables TIMER 0 interrupt
sync_2:   jb      p3.5,synchro_end; Normal operation
        acall   test_mode                  ; Go to Test Mode if p3.5=LOW
synchro_end:reti

```

```

;-----
; Wait for ACK pulse reception from the receiver
;-----
wait_ack:  mov     b,#2                ;
          acall   wait3                ; Wait for 3.2ms
          cpl    p0.4                ; Test Marker
wait_ack_2:jb    p3.2,wait_ack_2      ; Wait for fal.edge
          mov     a,#7                ; Wait for 50us
wait_ack_4:djnz  acc,wait_ack_4       ; (de-bounce)

wait_ack_3:jnb   p3.2,wait_ack_3      ; Wait for ris. edge
          mov     a,#200              ; Delay before sampling
wait_pulse:djnz  acc,wait_pulse       ; DATAOUT
          cpl    p0.4                ; Indicate the sampling point
          jb     p0.6,ack_error       ; Test DATAOUT
          clr    c                    ; DATAOUT=LOW -> OK
          ret

ack_error: cpl   c                    ; DATAOUT=1 -> Problem !
          ret                          ; C=1 No ACK

;-----
; Fill cluster with &(Add)W_ABCDEFGH +checksum
; for auto-evaluation mode
;-----
fill_cluster:mov  r0,#cluster_buffer  ; Data pointer
          mov     a,#'&'              ; Cluster Head
          mov     @r0,a                ;
          inc     r0                    ;
          mov     a,slave_add          ; ASCII Slave Address '0'
          ; to '3'
          mov     @r0,a                ;
          inc     r0                    ;
          mov     a,#'W'              ; Select ASCII transfer
          ; mode
          mov     @r0,a                ;
          inc     r0                    ;
          mov     a,#' '              ;
          mov     @r0,a                ;
          inc     r0                    ;
          ;
          mov     a,#41h               ; A=41h
fill_2:  mov     @r0,a                ; Fill current byte
          inc     a
          inc     r0
          cjne   r0,#cluster_check,fill_2; Checksum byte ?
          mov     a,slave_add          ; Calculate checksum
          add    a,#0c1h              ; slave_add-030h+0F1h
          mov     @r0,a
          ret

```

```

; ++++++
;                               'USER SIDE' ROUTINES
; ++++++

;-----
; 12-byte cluster Reception - Checksum calculation
;-----
rx_cluster:mov     r0,#cluster_buffer           ; Data pointer
              mov     @r0,a                     ; First byte is
already                               ; in Acc
              inc     r0
rx_cluster_2:acall in_a
              mov     @r0,a                     ; Store current
                                              ; byte
              inc     r0
              cjne   r0,#cluster_check,rx_cluster_2 ; Last byte ?
;
              mov     r0,#cluster_buffer       ; Data pointer
              mov     b,#0                     ; Checksum=0
rx_cluster_3:mov   a,@r0                       ; Read data
              add     a,b
              mov     b,a                       ; b=Checksum
              inc     r0
              cjne   r0,#cluster_check,rx_cluster_3 ; Last byte ?
              mov     a,b
              mov     @r0,a                     ; (2eh)=Checksum
              ret

;-----
;          Powerline Transmission error ->
;          Send message to RS232
;-----
error_to_rs232:mov  dptr,#str_error
                 acall  tx_str           ; Error Message
                 acall  led_off
                 acall  red_on
                 ret

;-----
;          Powerline Transmission Time-Out error ->
;          Send error message to RS232
;-----
O_to_rs232:mov    dptr,#str_timeout
             acall  tx_str           ; Error Message
             acall  led_off
             acall  red_on
             ret

;-----
;          Powerline Transmission successfull ->
;          Send message to RS232
;-----
T_to_rs232:mov    dptr,#str_clust_rx
             acall  tx_str           ; OK Message
             acall  led_off
             acall  green_on        ; ACK received - Transmission OK
             ret

```



```

;-----
;           RS232 Reception successfull ->
;Transmission in progress -> Send message to RS232
;-----
R_to_rs232:mov     dptr,#str_bytes_tx
            acall   tx_str           ; OK Message
            ret

;-----
;           LEDs Control
;-----
green_on:  clr     p0.2              ; No comment !
            ret
red_on:    clr     p0.1
            ret
led_off:   setb    p0.1
            setb    p0.2
            ret

;-----
; RS232 Serial Output of A with parity bit set
;-----
out_a:     cpl     p0.0              ; WatchDog reset
            clr     acc.7            ; Parity bit clear
            jnb    psw.0,out_a_1
            setb    acc.7           ; Parity bit set
out_a_1:   mov     sbuf,acc
            clr     scon.1
out_a_2:   jnb    scon.1,out_a_2     ; Wait for TX buffer
            ret                    ; empty

;-----
;           RS232 Serial String Output
;           DPTR Contains the string address; end=255
;-----
tx_str:    mov     a,#0
next_2:    push    acc
            movc   a,@a+dptr
            cjne  a,#255,next_ch
            pop    acc
            ret
next_ch:   acall   out_a
            pop    acc
            inc    a
            sjmp  next_2

;-----
; RS232 Serial Input in A with parity bit check
;-----
in_a:      cpl     p0.0              ; Reset WatchDog
            jnb    scon.0,in_a       ; Wait for serial DATA
            clr     scon.0
            mov    a,sbuf
            push   acc
            anl   a,#01111111b      ; Parity check 7-bit Data
            jnb    psw.0,in_a_1
            pop    acc              ; psw.0=1

```

```

jnb    acc.7,in_error      ; Parity error
anl    a,#01111111b       ; RX DATA Ok
ret
in_a_1: pop    acc          ; psw.0=0
        jb     acc.7,in_error ; Parity error
        anl    a,#01111111b   ; RX DATA Ok
        ret
in_error: sjmp   in_a        ; Ignore RX if parity
                                   ; error

;-----
;                               Test MODE
;-----
test_mode: clr    p0.5          ; TDA5051 DATAIN=0
           acall  wait
           jnb   p0.6,do_low    ; DATAOUT must be LOW
           acall  red_on        ; Error: DATAOUT is HIGH
           sjmp  end_test
do_low:   acall  green_on       ; Test OK
end_test: acall  wait
           acall  led_off
           setb  p0.5           ; DATAIN=1
           ret

wait:     mov    b,#255         ; Software Delay
wait3:    mov    a,#255
wait2:    cpl    p0.0          ; WatchDog Reset
           djnz  acc,wait2
           djnz  b,wait3
           ret

; -----
;                               DCB value to Serial OUT
; -----
dcb_to_tx: acall  dcb_to_ascii
           acall  out_a
           mov    a,b
           acall  out_a
           ret

; -----
;                               HELP FILE display
; -----
help:     mov    dptr,#help_1
           acall  tx_str
           mov    dptr,#help_2
           acall  tx_str
           mov    dptr,#help_3
           acall  tx_str
           ret

```

```

; -----
;           DCB to ASCII translation
;           DCB in A, ASCII MSB in A, LSB in B
; -----
dcb_to_ascii:push  acc
              anl   a,#1111b
              add   a,#30h
              mov   b,a
              pop   acc
              swap  a
              anl   a,#1111b
              add   a,#30h
              ret

;-----
;           ASCII TABLES
;-----
str_error:   db      '> Error / No acknowledge !',13,10,255
str_timeout: db      '> Time-out error !',13,10,255
str_bytes_tx: db     '> Sending data...',13,10,255
str_clust_rx: db     '> Data received !',13,10,255
str_error_rate: db   '> Error rate: ',255
str_tx_test:  db     '> Transmission Test in
                    progress...',13,10,255
str_wait:    db      '> Waiting for RS232...',13,10,255

help_1: db ' ----- HELP FILE ----- '
        db 13,10,13,10
        db ' The symbol ? represents the SLAVE_ADDRESS from 0 to 3'
        db 13,10
        db ' the symbol - represents any ASCII code.'
        db 13,10,13,10
        db ' 1/ TRANSMISSION TEST WITH A SLAVE:',13,10
        db ' Send: $?                               > Ex:$3'
        db 13,10,13,10,255
help_2: db ' 2/ RELAY CONTROL:',13,10
        db ' Send: &?R-1----- for RELAY ON         > Ex:&0R-1-----'
        db 13,10
        db ' Send: &?R-0----- for RELAY OFF         > Ex:&0R-0-----'
        db 13,10,13,10
        db ' 3/ ASCII TRANSMISSION TO A SLAVE:'
        db 13,10,255
help_3: db ' Send: &?W- and 8 ASCII codes             > Ex:&2W-EXAMPLES'
        db 13,10,13,10
        db ' 4/ SLAVE PORT P2.7 to P2.1 BIT CONTROL:'
        db 13,10
        db ' Send: &?P-0000000- to &?P-1111111-       > Ex:&1P-1010101-'
        db 13,10,13,10,255

end

```

## **Appendix 2**

### **SLAVE**

### **SOFTWARE**

```

; -----
;                               (C) ERIC MICCHAT 1998
;                               8051 uC / TDA5051 POWERLINE COMMUNICATION
;                               EVALUATION SOFTWARE for the Slave
; -----

$MOD51    ; Uses 8x51 standard predefined labels

; ++++++
; CLOCKIN of 87C51 = 3.68 MHz (TDA5051 XTAL=7.37MHz)
;   Serial TTL OUT 1200 Bds, 8 bits, 1 Stop, No Par
; ++++++

; -----
;                               Constant Parameters
; -----

wait_init    equ    4        ; Initial delay before sampling
(*100us)

n_samples    equ    6        ; Num. of bit samples HIGH for a ONE
ack_width    equ    255     ; ACK pulse is 1.6ms long

; -----
;                               RAM Variables
; -----

; RX DATA BUFFER
data_buffer  equ    20h     ; D7 D6 D5 D4 D3 D2 D1 D0

; RX FLAGS
flag_rx      equ    21h     ; POB 0 0 0 CE PE TB DV
; POB= Parity Bit of DATA_BUFFER (ODD)
; CE= CHECKSUM ERROR
; PE= PARITY ERROR
; TB= TEST BIT
; DV= DATA_BUFFER VALID

cluster_buffer equ    22h     ; 12-byte RX Cluster buffer (22h...2dh)
cluster_check equ    2eh     ; Cluster Checksum
var_add      equ    2fh     ; 2-bit Slave Address [0000 00XX]

; -----
;                               I/O Configuration
; -----

;
; p3.2 (INT0)          Zero Crossing Detectio n
; p0.4                Bit Time Window (for test only)
; p0.5                DATA_IN TDA5051
; p0.6                DATA_OUT TDA5051
; TX                  Serial TTL DATA OUT (For display)
; p0.1                RED LED
; p0.2                GREEN LED
; p3.5                Test Push Button
; p0.0                WatchDog Reset
; p2.0 to p2.7        Extension Port (output)
; P1.0 to p1.7        Address DIL Switches

```

```

; -----
;                               Interrupt Vectors
; -----
;       Reset
org     0000
ajmp   init
;       INT0 - Zero Synchro
org     0003h
ajmp   synchro
;       Timer 0 - Bit Sampling Rate 100us
org     000bh
ajmp   rx_bit

ORG     40H
init:   ;+++++
;                               Initialisations
;+++++
mov     sp,#60h           ; Stack pointer
mov     psw,#0           ; Register BANK 0 in use
mov     flag_rx,#0       ; Clear all flags
clr     p0.7             ; Powerdown=0
clr     p0.5             ; Reset TDA5051 (DATAIN=0)
mov     b,#1
acall  wait3             ; Wait
setb    p0.5             ; DATAIN=1
clr     p3.4             ; Relay off
mov     p2,#0           ; Init P2 output
mov     p1,#255         ; Init P1 input
setb    p3.5             ; BP input
setb    p3.7             ; Opto input
setb    p0.3             ; DS1820 input
;-----
;       UART init for 1200 Bauds, 8 bits, 1 Stop
;       Timer 1 is used as a Baud Rate generator
;       for the TTL communication timing
;-----

mov     scon,#01000000b ; UART mode 1 - F variable
mov     pcon,#0         ; SMOD=0
mov     th1,#0f8h      ; 1200 Bds @ 3.68MHz
mov     tmod,#00100001b ; Timer 1 Mode Auto-load
setb    tcon.6         ; Run Timer 1

;-----
;       Timer 0 init - Powerline communication timing
;-----

mov     th0,#0e1h      ; Time base T=100us
mov     tmod,#00100010b ; Timer 0 Mode 8 bits auto
setb    tcon.4         ; Run Timer 0

;-----
;                               Interrupts configuration
;-----

setb    ie.0           ; Enable INT0
clr     ie.1           ; Disable TIMER0 interrupt
setb    ip.1           ; MAX priority is Timer 0
setb    tcon.0         ; INT0 sensitive on fal. edges
clr     tcon.1

```

```

;-----
;           Serial Init for display
;-----
mov  dptr,#init_dsp
acall tx_str

;+++++
;           MAIN LOOP
;+++++

main_loop: cpl    p0.0           ; Reset WatchDog
           setb   ie.7          ; Enable INT
           acall  rx_cluster    ; 12-bytes cluster reception
           clr    ie.7          ; Disable INT
           jnc    cluster_ok    ; Cluster is OK ?
           acall  error_to_TTL  ; No, Error
           sjmp   main_loop

cluster_ok:acall  ctrl_add      ; Read Slave Address
           jnc   main_loop      ; Not for me !
           acall  send_ack      ; Send ACK pulse if OK
           acall  ctrl_ident    ; Read data and execute
           sjmp   main_loop

; ++++++
;           "POWERLINE SIDE" / RX DATA RO UTINES
; ++++++

;-----
;           Cluster reception (12 bytes+checksum)
;           R0 is the data pointer of the cluster_buffer
;           Returns C=0 if the cluster is OK else C=1
;-----

rx_cluster:acall  wait_byte      ; Wait for the start byte
           jnc   start_ok        ; Parity is OK
byte_nok:  sjmp   abort_cluster_p ; Parity is wrong

start_ok:  mov    a,data_buffer  ; Read DATA buffer RX
           cjne  a,#99h,rx_cluster ; If DATA <> 99h
           ; else

           mov    r0,#cluster_buffer
next_byte: acall  wait_byte
           jnc   next_ok
           sjmp  abort_cluster_p ; Parity is wrong
next_ok:   mov    @r0,data_buffer
           inc    r0
           cjne  r0,#cluster_check+1,next_byte
           mov    r0,#cluster_buffer ; Verify Cluster checksum
           mov    b,#0

chk_cl:   mov    a,@r0           ; Read byte
           add    a,b
           mov    b,a           ; B=checksum
           inc    r0           ; Next byte
           cjne  r0,#cluster_check,chk_cl ; Cluster END ?
           mov    a,@r0         ; Read Checksum
           cjne  a,b,abort_cluster_c ; Checksum is wrong !
           ; Checksum is OK !

           clr    c           ; carry=0
           ret                ; Parity & Checksum=OK

```

```

abort_cluster_c:
    setb    flag_rx.3                ; Set CE flag
    sjmp    end_abort                ; Checksum error
abort_cluster_p:
    setb    flag_rx.2                ; Set PE flag
                                        ; Parity error

end_abort:  clr     c
            cpl     c                ; carry=1
            ret

;-----
;                               Wait for a byte
;-----

wait_byte:  jnb     flag_rx.1,wait_byte_2 ; Is a frame received ?
wait_byte_3:cpl     p0.0                ; Reset WatchDog
            jb      flag_rx.0,wait_byte_3 ; YES
wait_byte_2:cpl     p0.0                ; Reset WatchDog
            jnb     flag_rx.0,wait_byte_2 ; Wait for a frame
            setb    flag_rx.1          ; A frame is received
            acall   parity_check        ; Check the parity of the
byte
            ret

;-----
; DATA_BUFFER Parity check - Return Carry=0 if OK
;                               Carry=1 else
;-----

parity_check:mov    a,data_buffer
            mov     a,psw
            anl     a,#1
            mov     b,a                ; Parity of received data
            mov     a,flag_rx
            rl      a
            anl     a,#1                ; Received Parity bit
            xrl     a,b
            clr     c
            jz      parity_ok
            cpl     c
parity_ok:  ret

; -----
;                               Bit Sampling on Timer 0 Interrupt
;                               A frame is (8 bits + 1 Parity bit)= 9 bits
; R7 ----> Received bits counter      (in a FRAME)
; R6 ----> Samples counter            (in a BIT)
; R5 ----> Initial delay counter      (* 100 us)
; R4 ----> "LOW level" samples counter (in a BIT)
; -----

rx_bit:    push    acc
            push    psw
            cjne   r5,#wait_init,rx_bit_2 ; Wait for initial delay

            cjne   r7,#9,sample_bit      ; Last bit of the frame ?
            setb   flag_rx.0            ; Set Flag "DATA VALID"
            pop     psw
            pop     acc
            reti

```



```

sample_bit:jb      p0.6,rx_0          ; START SAMPLING - Read
DATAOUT
                inc      r4          ; DATAOUT=0
rx_0:           cjne     r6,#9,rx_bit_3 ; Last sample of the bit ?
                ; YES
                mov     a,r4
                clr     c
                subb    a,#n_samples ; Bit analysis, if
                ; C=0 r4 >= n_samples then
ONE
                cpl     c          ; C=1 r4 < n_samples then
ZERO
                push    psw
                cjne    r7,#8,rx_bit_4 ; Last bit of the byte ?
                ; YES
                pop     psw
                mov     a,#0
                rrc     a
                mov     flag_rx,a    ; Store RX Parity Bit
                inc     r7          ; Next BIT
                cpl     p0.4        ; Shows the "bit time" window
                pop     psw
                pop     acc
                reti

rx_bit_4:       pop     psw
                mov     a,data_buffer ;
                rrc     a          ; Shift current bit
                mov     data_buffer,a ; Store current bit in buffer
                mov     r6,#0       ; Reset Samples counter
                mov     r4,#0       ; Reset LOW samples counter
                inc     r7          ; Next bit
                cpl     p0.4        ; Shows the "bit time" window
                pop     psw
                pop     acc
                reti

rx_bit_3:       inc     r6          ; Next sample in a BIT
                pop     psw
                pop     acc
                reti

rx_bit_2:       inc     r5          ; Initial delay counter
                pop     psw
                pop     acc
                reti

;-----
;      Enables Frame Sampling on INT0 interrupt
;-----
synchro:       clr     ie.0        ; Disable INT0
                clr     ie.1        ; Disable TIMER 0 interrupt
                mov     r7,#0       ; Reset bits counter
                mov     r6,#0       ; Reset samples counter
                mov     r5,#0       ; Reset delay
                mov     r4,#0       ; Reset LOW samples counter
                clr     flag_rx.0    ; Reset flag DataValid
                clr     flag_rx.1    ; Reset flag Bit Test

```

```

mov     data_buffer,#0    ; DATA=0
setb    ie.0              ; Enable INTO interrupt
setb    ie.1              ; Enable TIMER 0 interrupt

jb      p3.5,synchro_end; Normal operation
acall   test_mode         ; Go to Test Mode if p3.5=LOW
synchro_end:reti

;-----
;   Send Acknowledge Pulse if cluster is OK
;-----
send_ack: mov     b,#2          ; Wait for 3.2ms
         acall    wait3

send_ack_1:jb     p3.2,send_ack_1 ; Wait for fal.edge
         mov     a,#6          ; Wait for 50us
send_ack_4:djnz   acc,send_ack_4 ; de-bounce

send_ack_2:jnb    p3.2,send_ack_2 ; Wait for ris. edge
         cpl    p0.4          ; Test marker
         clr    p0.5          ; DATAIN=0
         mov    a,#ack_width

send_ack_3:djnz   acc,send_ack_3 ; Wait for 1.6ms
         setb   p0.5          ; DATAIN=1
         cpl   p0.4          ; Test marker
         ret

; ++++++
;                               'USER SIDE' ROUTINES
; ++++++
;-----
;   Cluster error -> send message to TTL
;-----
error_to_TTL:mov  a,#'/'
         acall   out_a          ; Error message follows
         acall   led_off
         acall   red_on
         jb     flag_rx.2,p_e_m
         jb     flag_rx.3,c_e_m
         ret
p_e_m:   mov     a,#'P'          ; It'a a Parity error
         acall   out_a
         clr    flag_rx.2
         ret
c_e_m:   mov     a,#'C'          ; It's a Checksum Error
         acall   out_a
         clr    flag_rx.3
         ret

;-----
;   TTL Serial Output of Acc with no parity bit
;-----
out_a:   clr     acc.7           ; Parity bit clear
         mov    sbuf,acc
         clr    scon.1
out_a_2: jnb    scon.1,out_a_2   ; Wait for TX buffer
         ret                    ; empty

```

```

;-----
;                               TTL Serial String Output
;       DPTR Contains the string address; end=255
;-----

tx_str:   mov     a,#0
next_2:   push    acc
          movc   a,@a+dptr
          cjne  a,#255,next_ch
          pop    acc
          ret

next_ch:  acall   out_a
          pop    acc
          inc   a
          sjmp  next_2

;-----
;Read received Slave address and compare with local
;  address code (switch) - C=0 if not the same
;-----

ctrl_add: mov     r0,#cluster_buffer      ; Read First byte
          mov     a,@r0
          cjne  a,'#&',no_ack            ; Is it '&' ?
          inc    r0                       ; YES, read next byte
          mov     a,#30h
          mov     b,@r0
          acall  ascii_to_dcb            ; Translate Add. to DCB
          push   acc
          acall  read_address             ; Compare with local add.
          pop    acc
          cjne  a,var_add,no_ack
          clr    c
          cpl   c                         ; It is the same
          ret

no_ack:   clr    c                       ; Not the same
          ret

;-----
;       READ local address code and save
;       0 0 0 0 0 0 (p1.1) (p1.0)
;-----

read_address:mov  a,p1                   ; Read p1
              anl  a,#00000111b         ; Mask
              mov  var_add,a            ; Save into variable
              ret

```

```

;-----
; Identify the received code coming from the master
;           r or R: RELAY CONTROL
;           w or W: ASCII transfer
;           p or P: Port p2.0/P2.7 Control
;           if others, flash the green LED
;-----

ctrl_ident:inc    r0
               mov    a,@r0                ; Read CTRL byte

               cjne   a,#'r',R
               sjmp   R_1                  ; it's 'r' -> RELAY CONTROL
R:             cjne   a,#'R',write
R_1:          acall  relay                  ; it's 'R' -> RELAY CONTROL
               sjmp   end_ctrl

write:        cjne   a,#'w',W              ; it's 'w' -> ASCII transfer
               sjmp   W_1
W:           cjne   a,#'W',port           ; it's 'W' -> ASCII transfer
W_1:        acall  ascii_to_TTL
               sjmp   end_ctrl

port:        cjne   a,#'p',port1          ; it's 'p' -> PORT control
               sjmp   P_1
port1:       cjne   a,#'P',bad_code
P_1:        acall  port_ctrl              ; it's 'P' -> PORT control
               sjmp   end_ctrl

bad_code:    acall  flash                  ; Others
end_ctrl:    ret

;-----
;                               RELAY Control
;-----

relay:       inc    r0
               inc    r0
               mov    a,@r0                ; Read control byte
               cjne   a,#'0',next_relay
               clr    p3.4                 ; it's '0' -> Relay OFF
               ret
next_relay:  cjne   a,#'1',bad_relay
               setb   p3.4                 ; it's '1' -> Relay ON
               ret
bad_relay:   acall  flash                  ; Others ->
               ret

;-----
; ASCII transfer requested -> Send 8 bytes to TTL
;-----
ascii_to_TTL:mov a,#13                    ;
               acall  out_a                ; CR
               mov    a,#10
               acall  out_a                ; LF

c_t_rs:     mov    r0,#cluster_buffer+4    ; Data pointer
               mov    a,@r0                ; Load data

```

```

acall  out_a                ; Send Data to TTL
inc    r0
cjne   r0,#cluster_check,c_t_rs

acall  led_off
acall  green_on            ; LED OK
ret

;-----
;                p2.0 to P2.7 PORT CONTROL
;-----

port_ctrl: inc    r0
          mov     b,#0          ; Tempor. port byte
          mov     r1,#8        ; Bit counter
next_port_bit:inc  r0
          mov     a,@r0        ; Read ASCII code
          cjne   a,'#0',next_port
          clr     c            ; it's a ZERO
          sjmp   next_port_1
next_port: cjne   a,'#1',bad_port
          clr     c
          cpl    c            ; it's a ONE
next_port_1:mov   a,b          ; Read tempor. port
          rlc    a            ; Shift current bit
          mov    b,a          ; Save tempor. port
          djnz   r1,next_port_bit; Next bit
          mov    p2,a        ; Write PORT
          ret
bad_port: acall   flash        ; Others
          ret

;-----
;                Conversion ASCII to DCB
;                ASCII MSB in A, LSB in B. DCB in A
;-----

ascii_to_dcb:clr   c
          subb   a,#30h
          anl   a,#00001111b
          rl    a
          rl    a
          rl    a
          rl    a
          push  acc
          mov   a,b
          subb  a,#30h
          anl   a,#00001111b
          mov   b,a
          pop   acc
          orl   a,b
          ret

```

```

;-----
;                               LEDs Control
;-----
green_on:  clr     p0.2           ; No comment !
           ret
red_on:    clr     p0.1
           ret
led_off:   setb    p0.1
           setb    p0.2
           ret

;-----
;                               Flash Green LED
;-----

flash:     mov     r0,#3         ; No comment !
back:      clr     p0.2
           acall   wait
           setb    p0.2
           acall   wait
           djnz   r0,back
           ret

;-----
;                               Test MODE - PB pressed
;-----

test_mode:  clr     p0.5         ; TDA5051 DATAIN=0
           acall   wait
           jnb    p0.6,do_low    ; DATAOUT must be LOW
           acall   red_on        ; Error: DATAOUT is HIGH
           sjmp   end_test
do_low:     acall   green_on      ; Test OK
end_test:   acall   wait
           acall   led_off
           setb    p0.5         ; DATAIN=1
           ret

wait:       mov     b,#255       ; Software Delay
wait3:      mov     a,#255
wait2:      cpl     p0.0
           djnz   acc,wait2
           djnz   b,wait3
           ret

init_dsp:   db     10,13,'      DISPLAY INIT      ',255

end

```

## **Appendix 3**

# **ELECTRIC DIAGRAM OF THE MASTER**






# **Appendix 4**

## **ELECTRIC DIAGRAM**

### **OF THE SLAVE**

	TITLE:
	DB8051R
Document:	8051+DEMOBOARD - SLAVE
Date:	05/06/1998 10:23:00 Sheet: 1/1

