SGS-THOMSON MICROELECTRONICS **APPLICATION NOTE**

Synchronous Power Line Modem Communication with ST9 Multifunction Timer

Required tools: ST9/ST7537 PLM Starter Kit, Author: O. GARREAU

INTRODUCTION

This application note provides an example of an ST9 MFT application handling a Home Automation synchronous protocol. It presents a way to easily communicate on a synchronous Network. Each node of this network may consist of the Power Line Modem (PLM) Starter Kit or of the Home Service (HS) macro-component, provided that it includes the ST9 plus ST7537 modem chipset.
This PLM Starter Kit helps also in developing the ST9 version of 'HOME SERVICE' European Home Automation Protocol.

The ST7537 modem may work in both synchronous and asynchronous modes at a Baud rate of 1200. There is no problem in building an asynchronous interface with the ST9 due to the capabilities of its SCI (Serial Communication Interface). The asynchronous protocol may be programmed directly and all work is done by the SCI. However the synchronous protocol has different requirements.

In order to program a synchronous protocol, it is not possible to use the SCI lines that are reserved for asynchronous communications. The solution resides in using the Multifunction Timer of the ST9.

The most simplified synchronous protocol may consist in a simple 3-wire link (The Receive Data line, RxD, the Transmit Data line, TxD and the signal Ground).

Similar to the asynchronous mode, no clock signal is available in the PLM synchronous mode and the time reference is included in the RxD signal. It is clear that the clock frequency should be known and determined in advance by both emitter and receiver. For our application working as the LAYER 0 of the 'HS' Protocol, this frequency is 1200Hz and generates thus a 1200 baud rate. Note that the signals concerned (RxD and TxD) are 'NRZ, Non Return to Zero' signals.

The first task that the following procedure performs is 'Transmission BitClock Recovery'. Here the Multifunction Timer T0 of the ST9 is used in the background. This basic clock signal is also output on a timer pin (T0OUTA).

The procedure that tests the link between distant modems, consists in sending from the master modem a standard frame, compatible with the 'HS' protocol or not, and in sending back from the slave modem the received frame in order to make a match test, for example, in the master system (which is, in our case, controlled by a Windows 3.1$_{(TM)}$ program).

This method can even validate asynchronous modes. The test frame is stored in the internal memory of the ST9 (in the first 128 bytes of the Register File).

In addition to the RxD and TxD lines, it is possible to improve protocol management with extra lines called RSTO, CDn, RXTXn and WDn. These lines mean respectively RESET modem controller, CARRIER DETECT, RECEIVE OR TRANSMIT, and WATCHDOG signal. 'n' means that the signal is active LOW.

These lines are specific to the ST7537 modem as it is working in its HALF-DUPLEX mode. The direction of data flow is chosen by RXTXn state. CDn indicates that the modem is about to receive data. The WATCHDOG signal is used by the modem to check the presence and activity of the CPU (ST9) and if activity (meaning a minimum of one negative transition per second) is absent, the modem will try to reset the CPU.

To prevent this shut down, WDn may be connected to the recovered BitClock (1200 transitions per second) or to an IO port (P76 in this case).

Note: To fully understand this Note, it is recommended that the reader refers to the relevant chapters of the ST9 databook for the Multifunction Timers.

1 HARDWARE LINK BETWEEN ST9 AND ST7537

Figure 1 shows the schematic of this link. As can be seen, no additional hardware or components are needed. This interface needs only 5 point-to-point wires plus the signal ground. It is the lowest cost way to connect a ST7537 to an ST9.

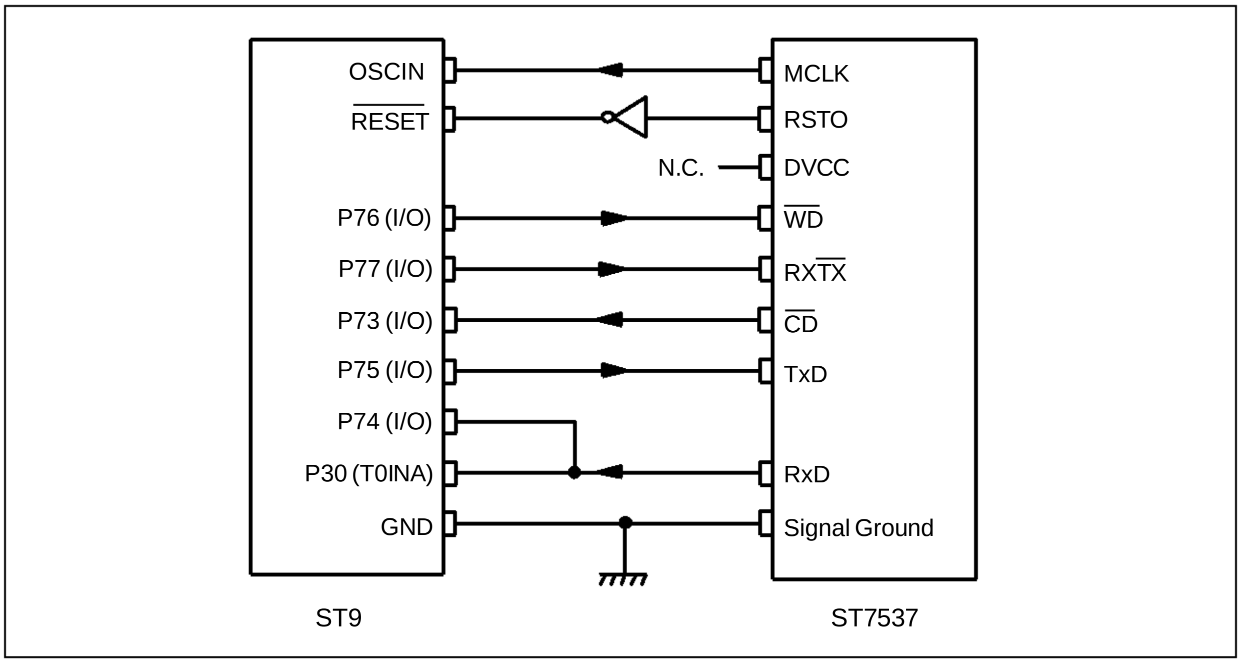
All wires are TTL compatible and mono-directional. Only one 8-bit port 7 of the ST9 is required, Port 3.0 (T0INA) is also reserved for this application, signal on T0OUTA (P3.1) is the recovered clock, event trigger and control the timer, P7.3 P7.4 P7.5 P7.6 P7.7 are used as simple TTL I/O bits.

The RxD line is managed by a technique which attaches a double function to this single signal. RxD triggers the timer and is acting as a synchroniser. Its level indicates also the input bit state. The kernel of this application is a sophisticated software PLL.

P7.3 is programmed as an input and reads Carrier Detect signal (CDn).
P7.4 is programmed as an input and reads the current input bit state (RxD).
P7.5 is programmed as an output and sends the current output bit (TxD).
P7.6 is programmed as an output and activates the WatchDog signal (WD).
P7.7 is programmed as an output and chooses the direction of data flow (RXTXn).

The corresponding timing chart of these signals is given in the next section. Note that it is not necessary to connect ST7537 signals like DVCC (Digital Output Supply Voltage), RSTO (Reset Output) and MCLK (Master Clock) when this application is running on the ST9 STARTER KIT. The signal ground is of course essential and common to all these lines. When the ST7537 Starter Kit is in stand-alone mode, the ST9 uses RSTO and MCLK.

Figure 1. Direct connection in the stand-alone mode



2 TIMING CHARTS OF CONTROL AND DATA SIGNALS

Figure 2 shows the timing and event charts considering an example of 3 bits input or output or both.

The first chart displays Carrier Detect signal, which enables the start of transfer.
The second timing shows the input signal used as trigger and data signal. Each ’0’ or ’1’ pulse lasts 833.3 microSeconds. The third line is a relative time axis for the internal counter T0. The next axis is the ’Event’ axis. The next chart displays the output waveform for the recovered BitClock; this preferably should have a 50% cyclic ratio.
The last chart is an example of the TxD output signal.

All the powerful features of a ST9 timer are used in this application. These include Hardware Trigger (HT), Software Trigger (ST), Compare (CMP0, CMP1) and Over/UnderFlow (OUF) Events. Both HT and ST events reload TIMER0 from the REG0R Load register. The HT event is automatically generated by every low to high or high to low transition on the RxD line. The CMP1 event is used to rebuild the initial clock frequency (of the emitter). The CMP0 event samples the input signal on RxD or outputs the transmitted signal on TxD or both. The OUF event (over-under/flow) allows the counter to be reloaded by an ST action, this acts as a WatchDog, controlled under software and corrects (synchronises) for frequency shift, frequency fluctuations and phase shift.
During the time gap (called DT), these parameters may vary and can be taken into account, if their variation is not too wide. A software parameter controls this correction : ’variation’ represents a percentage of the whole bit pulse period, called T.

Figure 2. Timing chart of signal and data signals

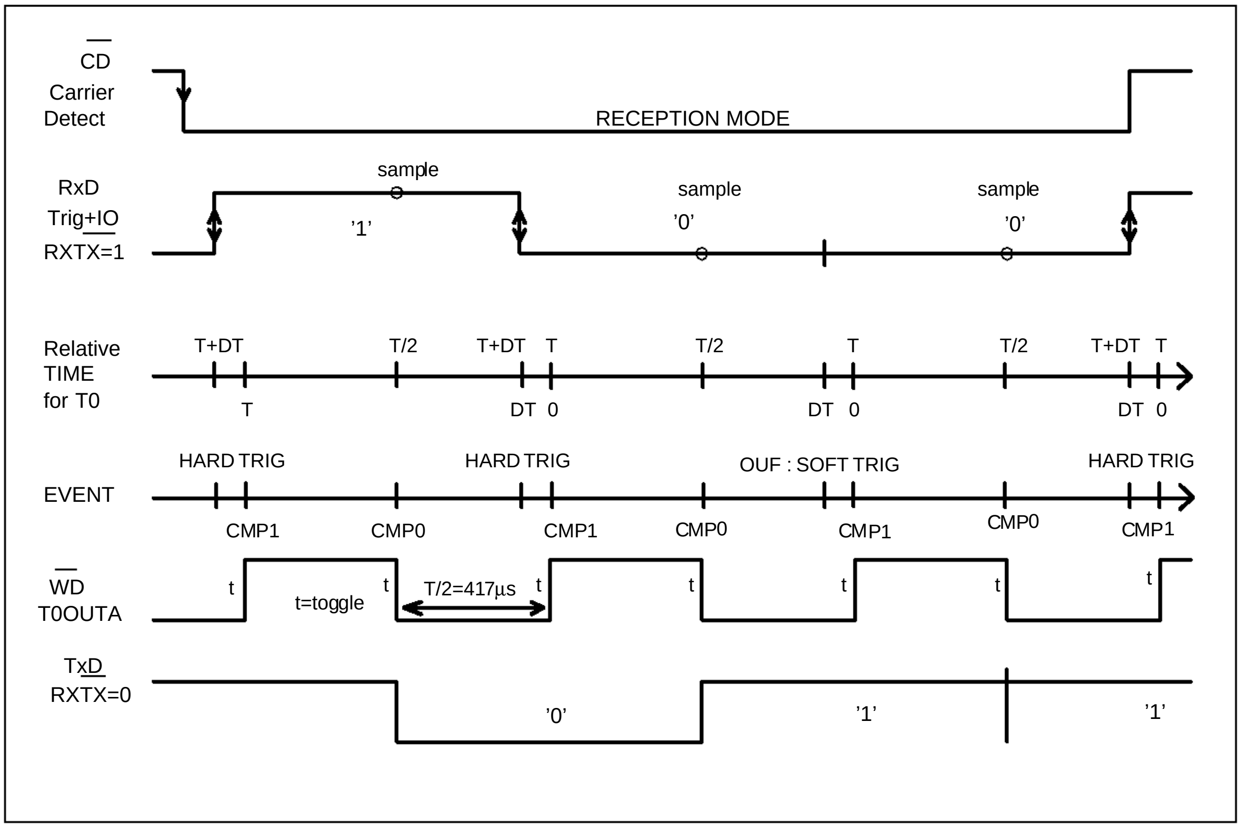


Figure 3. Software structure

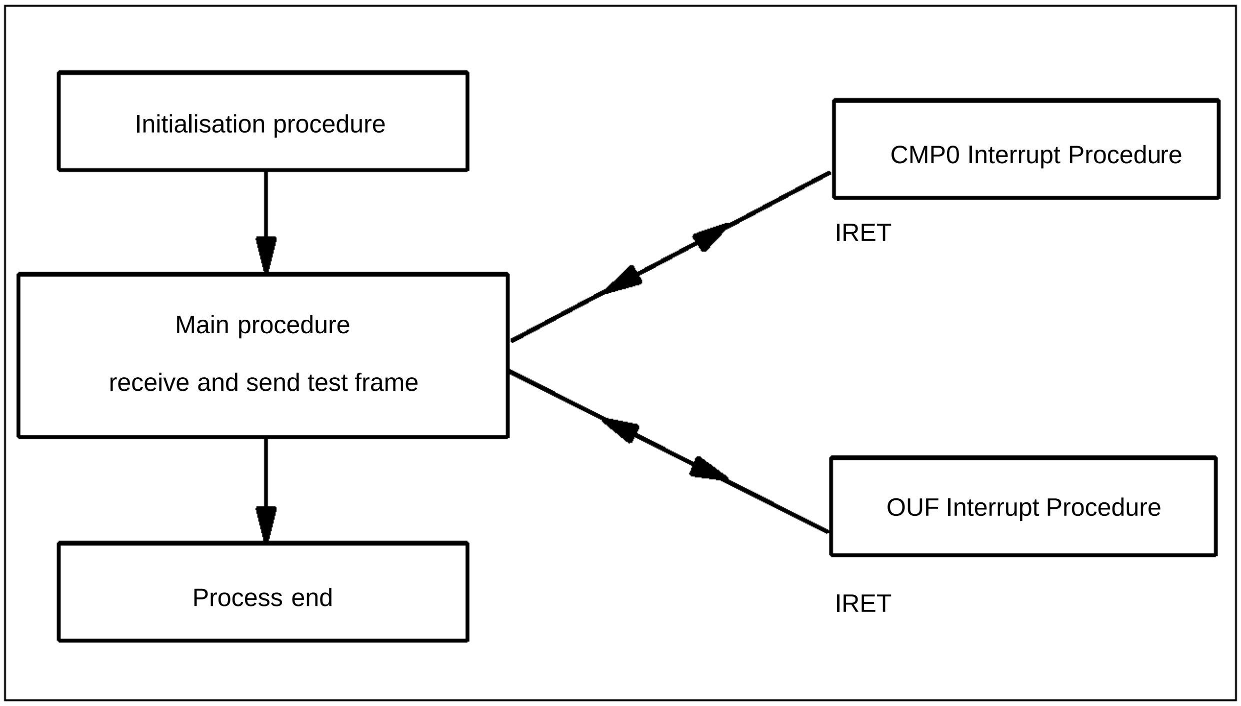
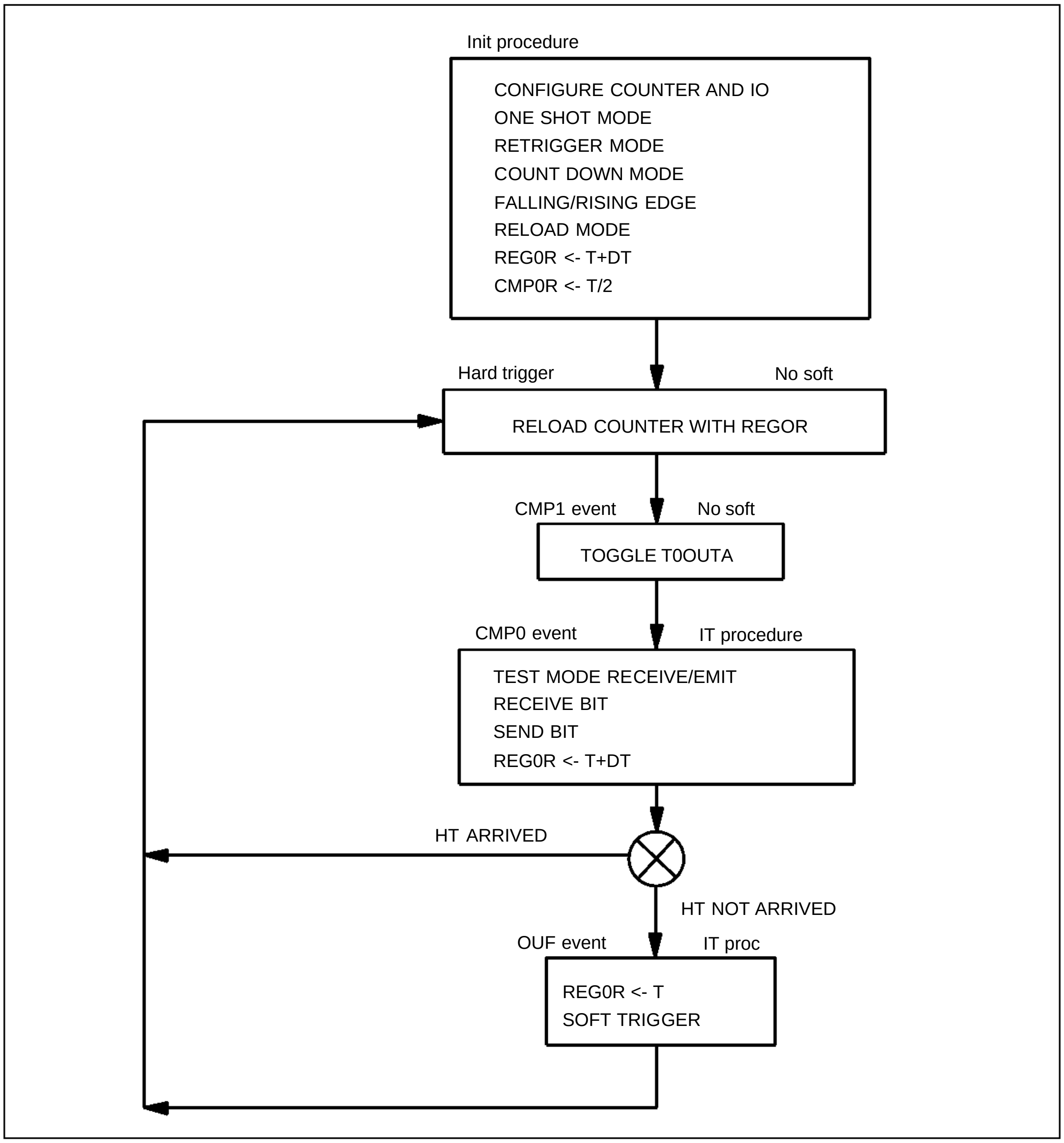


Figure 4. Procedure Algorithm



3 SUMMARY

The aim of this application note is to demonstrate to Hardware designers just how simple it is to interface an ST7537 with a member of the ST9 family. The schematic described in part one is an example of the electrical link, however it also depends on a good quality and conformity of input signals.

First, it is obvious that the RxD signal should be perfectly stable and defined during the acquisition phase. Then, its working frequency (1200 Hz) should be reasonably stable and the stability should not be worse than 90%. With practical testing, it is shown that the application works correctly between 1150 Hz and 1250 Hz. All these characteristics are software programmable.

This routine has been tested at various transmission rates, from 300 Hz up to 9600 Hz. The software defines constants called ’Rxxxxbds’ that correspond to these rates. That way, this application software will stay compatible with new generations of multi-speed ST7537 modem.

Warning : the user must RECALCULATE these constants if the ST9 has an external clock (crystal) DIFFERENT to the frequency used in this example.
The nominal working frequency should be 1200 Hz or 2400 Hz in the case of the ’HS’ protocol.

In this protocol, data packets are composed of a particular frame. Each frame starts with a specific header. Typically the beginning of such a header is 4 bytes: generally FFh, FFh, AAh and AAh. Many bits of the first FFh may be lost but the synchronisation is made actually on the AAh bytes. It is thus possible to improve the accuracy of the process. The byte AAh is a succession of bits at ’1’ and ’0’ and can allow frequency optimisation. Using first the timer 0 in its ’Capture’ mode will define the precise distant frequency, a pre-calibration.

This software improvement may be useful in case of a distant frequency very far from the expected working frequency (1200 Hz in this case).

The user is free to modify these routines. The bits of Port 7 (P73, P74, P75, P76 and P77) are used here as I/O bits. It is clear that another port may be defined as I/O port in order to free access to PORT 7.

Finally, the user can test a link between two distant nodes, or on a bigger network, and use equally synchronous or asynchronous protocol .

As a conclusion, the ST9 microcontroller is a cost effective solution for home automation modem applications and a synchronous protocol like ’HS’ or other customer protocol can be performed in the background, requiring little CPU time.

Appendix 1. ST9 Software listing.

```
;____________________________________________________________________________
;                              MODEM+ST9 APPLICATION NOTE
;               FILE: HS.ST9  (compile with AST9)
;               First version: 26/05/93
;               Last revision: 13/10/93
;               Author  : Olivier Garreau
;               Department PPG, Section ST9 SUPPORT/APPLICATION (Grenoble)
;               Running on ST7537 STARTER KIT, here in stand-alone 11MHz
;               ST7537 starter kit version : MB076b
;               ST7537 version  : V 3.0
;____________________________________________________________________________

;****************************************************************************
;       goal    : SYNCHRONOUS BIT CLOCK RECOVERY, I/O BUFFERS MANAGER
;                 Validation software for a serial link between two
;                 based on ST 7537 modems, Full Duplex capability
;                 one modem driver is a PC, second modem driver is the ST9
;
;       example of application  : 'HOME SERVICE' AUTOMATION PROTOCOL
;       modem used      : ST7537 (always working in half duplex mode)
;       input signals   : RxD connected to T0INA(P30) AND P74
;                         CDn (carrier detect) connected to P73 (by jumper)
;       output signals   : Bit clock recovered on T0OUTA (P31).
;                         P76 output on modem WatchDog WDn.
;                         RXTXn (select direction) connected to P77
;                         TxD connected to P75
;       modified        : Input Bit buffer permanently filled
;                         Output Bit Buffer permanently sent
;
;       use underflow, compare0 and compare1 event interrupts
;       -OUF interrupt reload the timer
;       -COMPARE0 interrupt sample data on RxD line and update Bit Buffer
;                and manage TxD output signal
;       -COMPARE1 generate output recovered clock (no procedure)
;       Process running in background:
;               - read a synchronous frame (max 128*8 bits)
;               - write it back to the emitter
;               - then stand by to check watchdog circuit (wait for reset)
;****************************************************************************

;***************************
;* Include file definition *
;***************************
.include        "..\\..\\include\st904x.inc"


; REGISTERS DEFINITION
;********************

data            =       r0      ; data.b2 contains binary information
status          =       r1      ; hold flag of timer
ptr             =       r2      ; read buffer pointer
num_bit_r       =       r3      ; bit number in input byte
ptw             =       r4      ; output buffer pointer
num_bit_w       =       r5      ; bit number in output byte
mode            =       r6      ; mode selection(input,ouput or full duplex)
tempo1          =       r7      ; 8 bits register
tempo           =       rr8     ; 16 bits register
```

```
sys_stack       =       0CFh    ; system stack
end_buf         =       07Fh    ; common end of input and output buffer
start_buf       =       000h    ; start of both buffers
;*********************
; CONSTANTS DEFINITION
;*********************

pageF           =       (0Fh*2)
page8           =       (08h*2)

;time_step      =       250     ; minimum time step is 250 nanoseconds
;period         =       833333  ; 833333 nanoseconds corresponds to 1200 Hz
;T              =       period/time_step        ; period for timer
;DT             =       T/variation             ; Delta T for timer

                                ; different clock rate
R9600bds        =       416     ; 9600 baud
R4800bds        =       833     ; 4800 baud
R2400bds        =       1666    ; 2400 baud
R1200bds        =       3072    ; 1200 baud modified for 11.0592 Mhz
                                ; should be 3333 with 24 Mhz crystal (/2)
R600bds         =       6666    ; 600 baud
R300bds         =       13333   ; 300 baud

.defstr mode_transmit   ”r6.0”  ; flag for transmission
.defstr mode_receive    ”r6.1”  ; flag for reception

RxD             =       4         ; position of RxD bit
TxD             =       00100000b ; TxD bit
CD              =       3         ; position of Carrier Detect bit
RXTX            =       10000000b ; RXTX bit
WATCHDOG        =       01000000b ; WDn signal

T               =       R1200bds; ’Home Service’ clock frequency
variation       =       10      ; +-variation around correct signal edge
DT              =       T/variation     ; allow +-10% variation
                        ; the effective receive frequency is 1200.12 Hz

t0_vect         =       010h    ; Start of Timer 0 vector table.

;******************
; MACRO DEFINITIONS
;******************

.macro t0start                          ; start counter
        begin [PPR] {
        spp #T0D_PG
        or  T_IDMR,#gtien               ; T0 Global interrupt mask disabled.
        or  T_TCR,#cen                  ; Counter enabled.
        }
.endm

.macro t0stop                           ; stop counter
        begin [PPR] {
        spp #T0D_PG
        and T_TCR,#~cen                 ; Counter disabled.
        and T_IDMR,#~gtien              ; T0 Global interrupt mask enabled.
        }
.endm
```

```
.macro  set_mode_transmit               ; set transmit flag and line RXTX
        bset    mode_transmit           ; set flag
        begin [PPR] {                   ; save PPR
        spp     #P7D_PG                 ; set port7 data page
        and     P7DR,#~RXTX             ; select transmit mode of modem
        }
.endm

.macro  reset_mode_transmit             ; reset transmit flag and line RXTX
        bres    mode_transmit           ; reset flag
        begin [PPR] {                   ; save PPR
        spp     #P7D_PG                 ; set port7 data page
        or      P7DR,#RXTX              ; select receive mode of modem
        }
.endm

.macro  set_mode_receive                ; set receive flag and line RXTX
        bset    mode_receive            ; set flag
        begin [PPR] {                   ; save PPR
        spp     #P7D_PG                 ; set port7 data page
        or      P7DR,#RXTX              ; select receive mode of modem
        }
.endm

.macro  reset_mode_receive              ; reset receive flag and line RXTX
        bres    mode_receive            ; reset flag
        begin [PPR] {                   ; save PPR
        spp     #P7D_PG                 ; set port7 data page
        or      P7DR,#RXTX              ; select receive mode of modem
        }
.endm

.macro  wait_CD_high    ?loop_wait      ; wait for ”1 to 0” transition on CD
        begin [PPR] {
        spp     #P7D_PG
        loop_wait:
        ld      data,P7DR               ; read input CD bit (data.b4)
        btjt    data.CD,loop_wait       ; wait for low level
        }
.endm

.macro  wait_CD_low     ?loop_wait      ; wait for ”0 to 1” transition on CD
        begin [PPR] {
        spp     #P7D_PG
        loop_wait:
        ld      data,P7DR               ; read input CD bit (data.b4)
        btjf    data.CD,loop_wait       ; wait for high level
        }
.endm

.macro  do_tempo
        clr     tempo1                   ; first loop
        ldw     tempo,#3000              ; tempo value
        loopw   [tempo] {
                loop    [tempo1] {
                        }
        }
.endm

;******************
```

```
; INTERRUPT VECTORS
;******************

power_on::

.word   main                    ; RESET vector.
.word   main                    ; Divide by 0 vector not used
.word   main                    ; no top level interrupt

.org    t0_vect                 ; table of timer interrupt vectors
.word   ouf_proc                ; vector of ouf interrupt
.word   error_proc              ; not a vector
.word   error_proc              ; no capture procedure, but event used
.word   comp_proc               ; vector of compare0 interrupt

;**************
; STOP on error
;**************

error_proc::
        halt                    ; stops if capture event

;*****************
; Routine COMPARE0
;*****************

comp_proc::                     ; middle of bit pulse

        begin   [PPR,RP0R,RP1R] { ; save PPR and RPP

        spp     #P7D_PG         ; set page for port7
        xor     P7DR,#WATCHDOG  ; signal ST9 activity
        srp     #page8          ; select working register, bank 8
        btjf    mode_receive,no_receive ; mode receive not selected
        ld      data,P7DR       ; read input RxD bit (data.b4)
        cpl     num_bit_r       ; prepare a reset mask
        and     (ptr),num_bit_r ; reset read bit in buffer (default case)
        cpl     num_bit_r       ; restore initial value
        btjf    data.RxD,bit_zero ; jump if RxD = 0
        or      (ptr),num_bit_r ; put read bit in buffer
bit_zero::
        ror     num_bit_r       ; shift from bn to bn-1
        adc     ptr,#0          ; increment pointer if end of byte
        and     ptr,#end_buf    ; prevent from overflow of input buffer
no_receive::
        btjf    mode_transmit,no_transmit ;mode transmit not selected
        tm      (ptw),num_bit_w ; test bit to send
        jrne    bit_high        ; jump if bit = ”1”
        and     P7DR,#~TxD      ; clear TxD line
        jr      bit_ok          ; bit state is OK
bit_high::
        or      P7DR,#TxD       ; set TxD line
bit_ok::
        ror     num_bit_w       ; shift from bn to bn-1
        adc     ptw,#0          ; test end of byte
        and     ptw,#end_buf    ; force buffer end
no_transmit::
        spp     #T0D_PG         ; page data for timer 0
        ldw     T_REG0R,#T+DT   ; load T+DT into REG0R
```

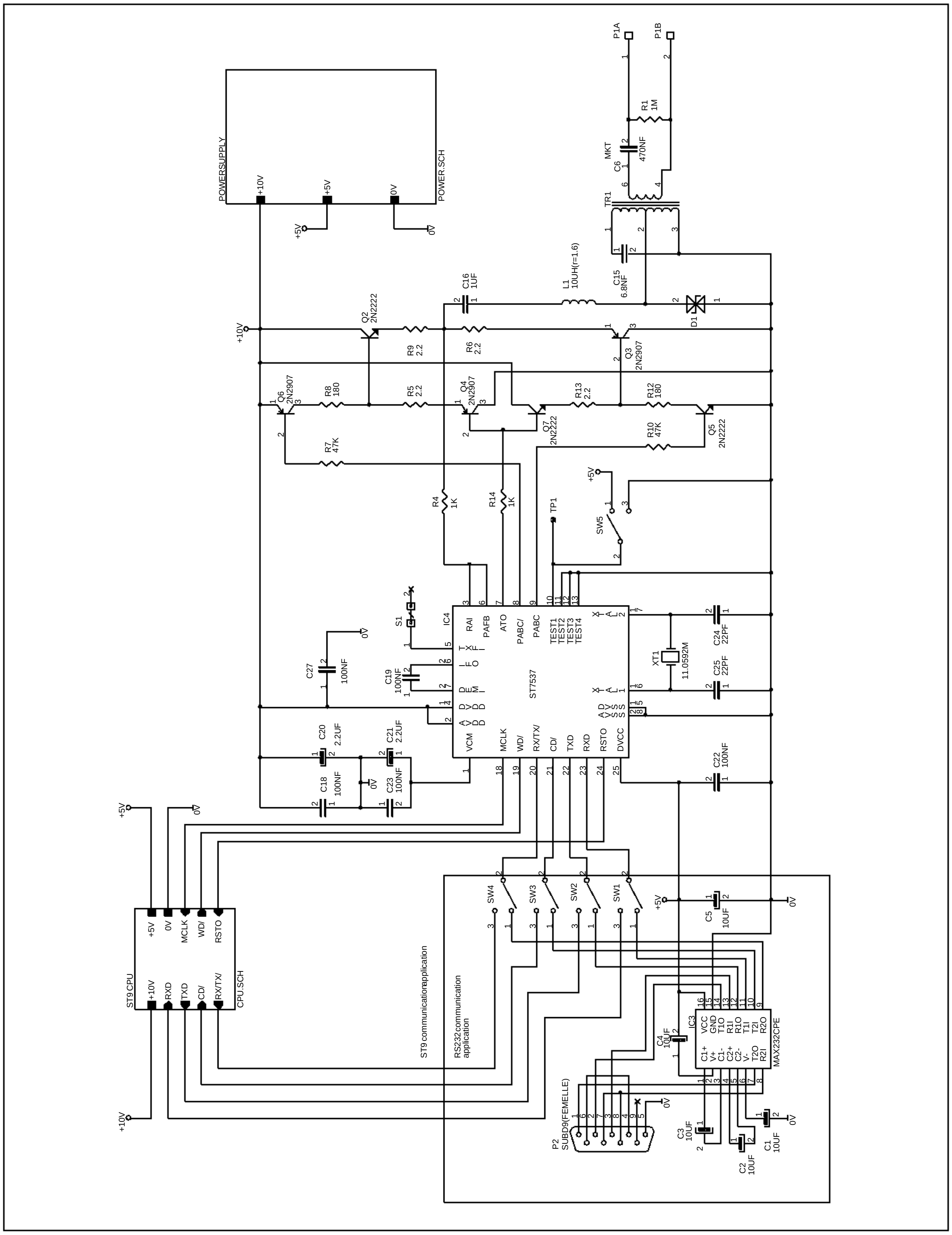
```
                                ; adjust timing : delay of DT
        ld      status,T_FLAGR  ; status flag, should read compare pending FL
        clr     T_FLAGR         ; reset pending bits
        }
        iret

;************
; Routine OUF
;************

ouf_proc::                      ; hardware watchdog -> no transition on RxD

        begin   [PPR,RP0R,RP1R] { ; save PPR and RPP

        spp     #T0D_PG         ; page data for timer 0
        ldw     T_REG0R,#T      ; load (T+DT)-DT into REG0R
                                ; adjust timing : advance of DT(synchronise)
        or      T_FLAGR,#cp0    ; launch counter with advanced value
        or      status,T_FLAGR  ; save pending bit (ouf pending bit)
        clr     T_FLAGR         ; reset pending bits
        }
        iret

;********************
; Routine periph_init
;********************

proc    periph_init     [PPR,RP0R,RP1R]       {

;************ init T0OUTA and T0INB ***************************************

        spp #P3C_PG             ; direction of signals      ST9 <==> ST7537
                                ; b0=T0INA=P3.0 : IN,TRI,CMOS   <--- RxD
        ld P3C2R,#00000000b     ; b1=T0OUTA=P3.1: AF,PP         ---> Clock
        ld P3C1R,#00000010b     ; b2-b7:unused
        ld P3C0R,#00000011b     ;


        spp #P7C_PG

        ld P7C2R,#00000000b     ; b0-b3         : free for SCI use
        ld P7C1R,#11100000b     ; b6=P7.6       : OUT,PP I/O    ---> WDn
        ld P7C0R,#00011000b     ; b7=P7.7       : OUT,PP I/O    ---> RXTX
                                ; b4=P7.4       : IN,TRI CMOS   <--- RxD
                                ; b5=P7.5       : OUT,PP I/O    ---> TxD
                                ; b3=P7.3       : IN,TRI CMOS   <--- CDn



;************ init timer modes ********************************************

        spp #T0D_PG             ; T0 data page
        srp #pageF              ; To access bank F with ”r” addressing mode.
        ldw t_reg0r,#T+DT       ; load REG0R with period + variation
        ldw t_cmp0r,#T/2        ; center to the middle of the pulse
        ldw t_cmp1r,#T          ; initiate (set) clock pulse
        ld  t_tcr,#0            ; Disable counter
                                ; Count down
        ld  t_tmr,#oe0|co       ; T0OUTB disable as timer output
                                ; T0OUTA enable
                                ; REG0R reload
                                ; No ECK clock
                                ; Retrigger mode
                                ; One shot mode
        ld t_icr,#ab_ti|exa_rf  ; T0INA trigger, T0INB I/O
                                ; T0INB nop, T0INA rising+falling edge
        ld t_prsr,#0            ; No prescaling
        ld t_oacr,#c0_res|c1_set|ou_nop ; CMP0R used to reset clock signal
                                ; CMP1R used to set clock signal
                                ; OUF not used
                                ; EOC not used
                                ; preload with '0'
        ld t_idmr,#gtien|cm0i|oui ; enable interrupt
                                ; enable compare0 interrupt
                                ; enable OUF interrupt
                                ; disable compare1 interrupt

        spp #T0C_PG

        ld t0_ivr,#t0_vect      ; pointer into vector table
        ld t0_idcr,#0C6h        ; priority level 6

        }

;*************
; MAIN PROGRAM
;*************

main::

        ld MODER,#11000000b     ; Ext clock NOT prescaled by 2.
                                ; Internal system and user stacks.
        ldw SSPR,#sys_stack     ; System stack pointer.

        spp #WDT_PG             ; select watchdog page
        ld  WCR,#wdgen          ; Watch dog mode disabled, no wait states.
        ld  EIMR,#0             ; Mask all channels interrupts.

;***********************  initialise I/O and timer  *********************

        call periph_init        ; Initialize timer and port

;***********************  init buffers and pointers  ********************

        srp     #page8
        ld      ptr,#start_buf  ; set read pointer to start
        ld      num_bit_r,#080h ; first bit to read : MSB
        ld      ptw,#start_buf  ; set write pointer to start
        ld      num_bit_w,#080h ; first bit to output : MSB

        clr     mode            ; reset both modes

;***********************  process itself  ********************************

        ei                      ; enable interrupts
        t0start                 ; start counter (down)

        ;***************    Read a frame   ********************************

        wait_CD_high            ; wait for the modem to receive
        set_mode_receive        ; activate RXTX line and fill input buffer
        wait_CD_low             ; wait for end of incoming data
        reset_mode_receive      ; stop receive mode
                                ; got the frame

        ;****************    Wait for a while   ****************************

        do_tempo                ; do a tempo

        ;****************    Emit the received frame   *********************

        set_mode_transmit       ; clear RXTX line and use output buffer

        while   [ ptw != ptr ] {
                                ; last input byte pointed by ptr
                                ; wait for the output buffer to be sent
        }                       ; stop at the last input byte

        while   [ num_bit_w != num_bit_r ] {
                                ; position of last bit pointed by num_bit_r
                                ; wait for the output buffer to be sent
        }                       ; stop at the last input bit (+1)

        reset_mode_transmit     ; stop emission mode

        di                      ; stop WatchDog to ST7537
        halt                    ; wait for reset

;***********************    end of file   ************************
```

Appendix 2. Procedure under Windows 3.1$_{\text{(TM)}}$

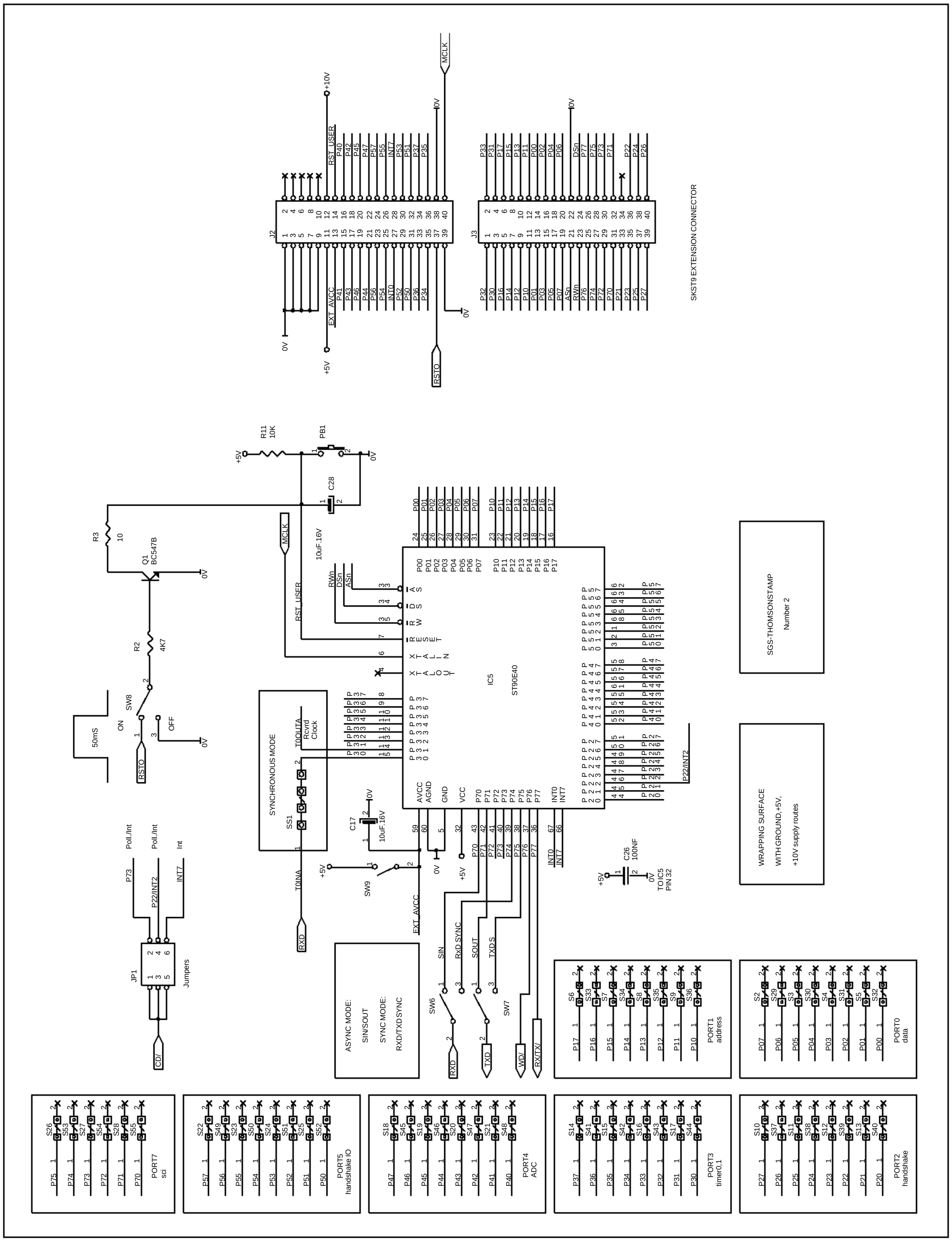
```
/*
;___________________________________________________________________________
;|
;|Routine Name  : PROCTEST
;|File Name     : PROCTEST.H(compile with MS QuickC)
;|Action        : is the Header of proctest.c file
;|Author(FN/LN) : Olivier Garreau / PPG / GRENOBLE
;|First rev date: 06/10/93
;|Last rev date : 13/10/93
;|Input paramet.: none
;|Output paramet: none
;|Modified var. : none
;|Global varia. : none
;|Comments      : define the constants of test procedure
;|                to be defined in a .MAK file
;|__________________________________________________________________________
*/

#define delay_RXTX_start      6     // wait 6 millisecond before first byte
#define delay_RXTX_end        6     // wait 6 millisecond after last byte
#define reset_delay           2000  // wait 2 seconds for Board2 ST9 reset
#define proper_buffer         800   // wait 0.8 second to flush input buffer
#define Watchdog              3000  // wait 3 seconds for ST9 answer

#define number_of_test        10    // perform 10 test loops MAX

#define size_buffer_in        1024
#define size_buffer_out       1024

#define test_string           ”This is a string to verify the links !”
#define length_string         sizeof (test_string)


void  init_com_port           (void);
void  send_frame              (void);
void  receive_frame           (void);
int   match_frames            (void);
void  display_result          (int);
void  wait                    (DWORD);
```

```
/*
_____________________________________________________________________________
;|
;|Routine Name  : PROCTEST
;|File Name     : PROCTEST.C(compile with MS QuickC)
;|Action        : Performs software validation of MB076b hardware
;|Author(FN/LN) : Olivier Garreau / PPG / GRENOBLE
;|First rev date: 06/10/93
;|Last rev date : 13/10/93
;|Input paramet.: none
;|Output paramet: none
;|Modified var. : none
;|Global varia. : none
;|Comments      : build a link this way :
;|
;|          Windows PC->modem1->modem2->ST9 ->!
;|                                           ! (TEMPO)
;|           Match test<-PC<-modem1<-modem2 <-!
;|
;|               So it checks both links of network.
;|               Compile with MS QuickC (Project: QuickWin EXE).
;|___________________________________________________________________________

*/

#include <windows.h>
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <conio.h>

#include ”proctest.h”

DCB         dcb_comm;
COMSTAT     com_stat;
char        buffer_in[size_buffer_in];
char        buffer_out[size_buffer_out] = test_string;// init string to send
int         com_device,test_key;


void  wait (tempo)

DWORD tempo;      // perform a tempo of ’tempo’ milliseconds
      {

DWORD now;
      now = GetCurrentTime();

        while (GetCurrentTime() < now + tempo) {
            // this loop wait for ’tempo’ delay
            }
      }

void  flush_in (void)

    {
int err;

            err=FlushComm(com_device,1);
            if    (err < 0) {
                  printf (”**** failed to flush input buffer ******\n”);
                  return ;
                  }
            strcpy (buffer_in ,”Time out buffer IN !!”);
      }

void  flush_out (void)

    {
int err;

            err=FlushComm(com_device,0);
            if    (err < 0) {
                  printf (”**** failed to flush output buffer ******\n”);
                  return ;
                  }
      }

void  init_com_port(void)// init COM1 for bidir link

      {
int   err;

            com_device=OpenComm(”COM1”,size_buffer_in,size_buffer_out);
            if    (com_device < 0) {
                  printf (”**** failed to open RS232 port ******\n”);
                  return ;
                  }

            err = BuildCommDCB(”COM1:1200,N,8,1”,&dcb_comm );
            if    (err < 0) {
                  printf (”**** can not setup COM1 ******\n”);
                  return ;
                  }

            else {

                  dcb_comm.fBinary  =     TRUE;
                  dcb_comm.fRtsDisable=   TRUE;
                  dcb_comm.fOutxCtsFlow=  FALSE;
                  dcb_comm.fOutxDsrFlow=  FALSE;
                  dcb_comm.fDtrDisable=   TRUE;
                  dcb_comm.fOutX    =     FALSE;
                  dcb_comm.fInX     =     FALSE;
                  dcb_comm.fDtrflow =     FALSE;
                  dcb_comm.fRtsflow =     FALSE;
                  }

            err = SetCommState(&dcb_comm);
            if    (err < 0) {
                  printf (”**** cannot setup COM1 ******\n”);
                  return ;
            }

        else {

            printf (”Please reset both board 1 and board 2\n”);

            printf(”type ’ENTER’ when done...\n”);
             test_key=getchar();
            }
      }

void  send_frame (void)

      {
int   err;

          err = EscapeCommFunction(com_device, SETRTS);// SET RXTX LOW
          if      (err < 0) {
                  printf (”**** can not clear RXTX signal ******\n”);
                  return ;
                  }

          wait (delay_RXTX_start); // wait for RXTX to be set up LOW

          err = WriteComm(com_device,buffer_out,length_string);
                                   // SEND BUFFER
          if      (err < 0) {
                  printf (”**** can not send test frame ******\n”);
                  return ;
            }

          do {

            err = GetCommError(com_device, &com_stat);
                  // WAIT TIL BUFFER OUT EMPTY
            if    (err < 0) {
                        printf (”**** can not wait for buffer
                                transmitted ******\n”);
                        return ;
                  }
            }
          while (com_stat.cbOutQue != 0);

          wait (delay_RXTX_end);    // wait for RXTX to be set up HIGH

          err = EscapeCommFunction(com_device, CLRRTS);// SET RXTX HIGH
          if      (err < 0) {
                  printf (”**** can not set RXTX signal ******\n”);
                  return ;
            }

      }

void  receive_frame(void)

      {

int   err;
DWORD read_time;

        wait(proper_buffer);// wait to have a proper input buffer
        flush_in();                       // flush input buffer

        read_time = GetCurrentTime();
        do {

            err = GetCommError(com_device, &com_stat);
                  // WAIT TIL BUFFER IN FULL
            if    (err < 0) {
                  printf (”**** can not wait for buffer input ***** \n”);
                        return ;
                  }
            }
            while ((com_stat.cbInQue
<length_string)&((GetCurrentTime()-read_time)<Watchdog));

            // Wait until size_buffer_in

            err = ReadComm(com_device,buffer_in,length_string);
                              //Read frame
            if    (err < 0) {
                  printf (”**** can not receive back frame ******\n”);
                  return ;
            }
      }

int   match_frames(void)

      {
            printf(”String sent    : %s\n”,buffer_out);
            printf(”String received: %s\n”,buffer_in);

            if(strcmp(buffer_in,buffer_out) == 0)

                  { return 1 ;}

            else

                  { return 0 ;}

      }

void  display_result(int value)

      {
     if (value)

      {printf(”----- GOOD MATCH !!!!\n”);}

     else

      {printf(”***** BAD MATCH !!!!\n”);}
      }

// ********************** MAIN PROCEDURE ****************************

void  main(void)

      {
int         counter,result,fail,err;

      printf(”Validation Software for ST7537 Starter Kit \n”);
      printf(”performs a bidirectional test from : \n”);
      printf(”MB076b board 1 to MB076b board 2\n”);
      printf(”boundary of board 1 is a PC\n”);
      printf(”boundary of board 2 is the ST9\n\n”);

      while(TRUE)
      {
      init_com_port();              // init RS232 com

      fail = 1;                     // number of communication attempts

      for (counter=0;counter<number_of_test;counter++)

            {

            send_frame();           // send frame through network

            receive_frame();        // wait until back frame

            result = match_frames(); // test if identical

            display_result(result);// display result of attempt n

            if    (result)
                  {
                  break;             // get out of the loop
                  }
            else
                  {
                  fail ++;           // increment attempt number
                  wait(reset_delay);
                  // wait during 2 seconds to reset Board 2 ST9
                  }

            }

      err = CloseComm(com_device);
      if    (err < 0) {
            printf(”**** cannot close COM ******\n”);
            return ;
            }

      printf(”End of test !!.\n\n”);

      if (result)
            {
            printf(”The current boards are GOOD after %dattempt(s).\n”
            ,fail);
            }
      else
            {
            printf(”The current boards are NOT GOOD !!\n”);
            printf(”Please send them back to designer.\n”);
            }

      printf(”End of procedure\n\n”);

      printf(”type ’ENTER’ to test new boards ...\n”);
      test_key=getchar();

    }

}
```

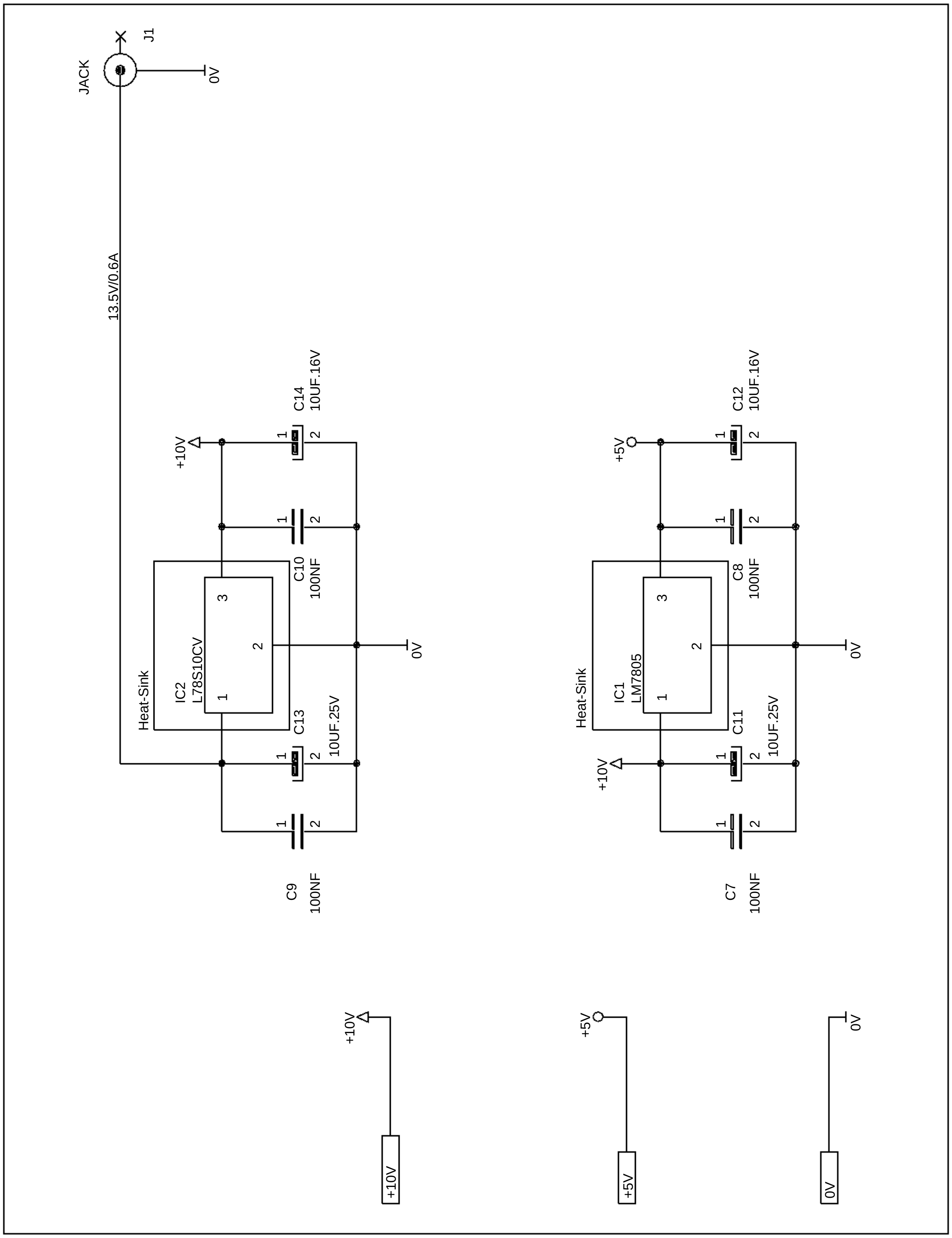
Appendix 3. Schematics of ST7537 Starter Kit: PLM interface



Schematics of ST7537 Starter Kit: ST9 connections



Schematics of ST7537 Starter Kit: Power supply


J1
JACK
0V
13.5V/0.6A
+10V
C14
10UF.16V
C10
100NF
Heat-Sink
IC2
L78S10CV
0V
C13
10UF.25V
C9
100NF
+5V
C12
10UF.16V
C8
100NF
Heat-Sink
IC1
LM7805
0V
C11
10UF.25V
+10V
C7
100NF
+10V
+5V
0V