APPLICATION NOTE

EIE/AN91009
Driver for 8xC851 E2PROM

January 1992

Philips
Semiconductors

PHILIPS

1. INTRODUCTION

A set of software functions is given to access the E^{2}PROM on the 8xC851 microcontrollers. These functions can be called from application programs written in assembly, PL/M-51 or C. The functions are found in the E2PROM.OBJ file that can be linked to the application program.

The driver is written and tested with the following software tools from BSO-Tasking:

Assembler:	ASM51 V3.2	(OM4142)
PL/M51:	PL/M51 V3.0A	(OM4144)
C Comp.:	C51 V2.0	(OM4136)
Debugger:	XRAY51 V1.4c	(OM4129)

Resources used by driver:

Exclusive use of 1 register bank (default RB1)
Accumulator
PSW
1 static bit addressable RAM byte

2. FUNCTION DESCRIPTIONS

The functions that use write and/or erase actions are interrupt driven except for E2PROM_wr_byte_pol. The application can check the status of these actions by testing the flag E2PROM_BUSY. This flag is available via the function E2PROM_status.

In the 8xC851, the E^{2}PROM interrupt is combined with the UART interrupt. To enable the E^{2}PROM interrupt, EA (in the IE-register) must be set (should be done in application program), the combined UART/E^{2}PROM enable bit must be set (ES in the IE-register, done with function E2PROM_int_en) and the E^{2}PROM interrupt enable bit (EEINT in ECNTRL register) must be set. The E^{2}PROM interrupt flag is automatically set by functions that use erase/write actions. This means that the UART interrupt enable cannot be disabled while the E^{2}PROM interrupt is completely enabled. The E^{2}PROM can be disabled separately with the E2PROM_int_dis function.

The priority level for UART and E^{2}PROM interrupt are the same and are defined with the E2PROM_int_en function.

The E^{2}PROM driver has a link to a UART interrupt handler. When a UART interrupt occurs, the status of the controller is pushed on the stack and then interrupt flags are tested to determine the source of the interrupt. When the source of the interrupt is the UART, then subroutine _UART_HDL is called. the implementation of the UART interrupt handler is done by the user. On the disk a file UART.SRC is available that contains this subroutine. This routine will only clear the trx-interrupt flag (TI) and rcv-interrupt flag (RI).

2.1 E2PROM_init

Function description:

This function must be called before any of the other functions is called. The timing register for writing/erasing the E^2PROM is initialized and the register bank that the E^2PROM functions can use is defined.

The default registerbank is RB1; the ETIM register which determines the write/erase timing is default initialized with 0x7B (XTAL = 12MHz). If other values are required, the parameters REGISTERBANK and XTAL must be changed in the equate list of the source file (E2PROM.ASM).

The E^2PROM/UART interrupt is enabled and set to priority level ‘0’.

Calling Sequence:

E2PROM_init();

Function prototype:

void E2PROM_init (void)

Parameters:

None

2.2 E2PROM_int_en

Function description:

This function will enable the E^2PROM/UART interrupt. The global enable bit EA is not effected and must be controlled by the application program.

The priority level of the E^2PROM/UART is controlled by the parameter ‘Pr_Level’.

Calling Sequence:

E2PROM_int_en (Pr_Level);

Function prototype:

void E2PROM_int_en (data char Pr_Level)

Parameters:

Pr_Level: This parameter determines the priority level on which the E^2PROM/UART interrupts are handled. Values greater than 0x01 will be interpreted as 0x01.

2.3 E2PROM_int_dis

Function description:

This function will disable the E^2PROM/UART interrupt.

Calling Sequence:

E2PROM_Int_Dis;

Function prototype:

void E2PROM_Int_Dis (void)

Parameters:

None

2.4 E2PROM_status

Function description:

This function will return the E2PROM_BUSY bit, which indicates whether a read/write transfer from/to the E^2PROM, or an erase action is finished.

‘1’ indicates that a read/write transfer is still in progress.
‘0’ indicates that no read/write transfer is in progress.

If the application program calls an E^2PROM function while another E^2PROM function is still in progress, parameters may be overwritten and an erroneous result will be obtained. There are 2 exceptions on this rule. When the functions “E2PROM_rd_byte_pol” or “E2PROM_wr_byte_pol” are called, parameters are passed to different registers in the registerbank, the E2PROM status is stored and the transfer is done by polling.

Note that the E2PROM_BUSY bit is not the same as EWP-flag in the ECNTRL register. For write operations the EWP-flag indicates when the writing/erasing of a byte to the E^2PROM is finished. The E2PROM_BUSY flag indicates when a complete block of data (e.g., from the E2PROM_wr_block function) has been written to the E^2PROM.

Calling Sequence:

```
bit Status;
Status = E2PROM_Status;
```

Function prototype:

```
bit E2PROM_Status (void)
```

Parameters:

None

2.5 E2PROM_wr_byte

Function description:

This function will write a byte to E^2PROM.
If the source byte has the same value as the byte in the E^2PROM, no write action will take place.

Byte transfer is done on interrupt basis. The status of the transfer can be checked with the “E2PROM_Status” function.

This function will automatically enable the E^2PROM interrupt. The application program should take care of the E^2PROM/UART interrupt enable (with E2PROM_int_en) and the EA bit.

Calling Sequence:

```
E2PROM_wr_byte (Src_Byte,Dest_Ptr);
```

Function prototype:

```
void E2PROM_wr_byte (data char Src_Byte,data char Dest_Ptr)
```

Parameters:

Src_Byte:	Byte to be written to E^2PROM
Dest_Ptr:	Address of E^2PROM

2.6 E2PROM_rd_byte

Function description:

This function will read a byte from E^{2}PROM. The status of the transfer can be checked with the “E2PROM_Status” function.

Calling Sequence:

```
data char Result;
Result = E2PROM_rd_byte (Src_Ptr);
```

Function prototype:

```
char E2PROM_rd_byte (data char Src_Ptr)
```

Parameters:

Src_Ptr:	Address of E^{2}PROM

2.7 E2PROM_wr_block

Function description:

This function will write a block of data from internal RAM to E^{2}PROM.

Byte transfer is done on interrupt basis. The status of the transfer can be checked with the “E2PROM_Status” function.

This function will automatically enable the E^{2}PROM interrupt. The application program should take care of the E^{2}PROM/UART interrupt enable (with E2PROM_int_en) and the EA bit.

If the source bytes are the same as the bytes in the E^{2}PROM, no write action will take place. This function will automatically generate ROW-erases, whenever this will reduce programming time. If during execution of this function, the destination address to the E^{2}PROM is equal to the beginning of an E^{2}PROM row (3 least significant address bits are ‘0’) and at least 8 more bytes have to be programmed, a check will be done whether a ROW-erase will reduce programming time. If no RWO-erase is done, the time to program the ROW will be:

$T_{prog} = A.t_W+B.(t_E+t_W)$	A: Byte in E^{2}PROM is 0x00; source byte in RAM is not 0x00
	B: Byte in E^{2}PROM is not 0x00; source byte in RAM <> E^{2}PROM byte

If a ROW-erase is done, programming the ROW will take:

$T_{prog} = t_E + C.t_W$	C: Source byte in RAM <> ‘0’

Because the erase time (t_E) and the write time (t_W) are equal, the function will do a ROW-erase if A+2.B–C–1 >= 0

Calling Sequence:

```
E2PROM_wr_block (Src_Ptr,Dest_Ptr,Nr_Bytes);
```

Function prototype:

```
void E2PROM_wr_block (data char *data Src_Ptr, data char Dest_Ptr, data char Nr_Bytes)
```

Parameters:

Src_Ptr:	Address pointer to internal RAM
Dest_Ptr:	Address of first E^{2}PROM byte
Nr_Bytes:	Number of bytes to write to E^{2}PROM

2.8 E2PROM_rd_block

Function description:

This function will read a block of data from E^2PROM and store it in internal RAM. The status of the transfer can be checked with the “E2PROM_Status” function.

Calling Sequence:

E2PROM_rd_block (Src_Ptr,Dest_Ptr,Nr_Bytes);

Function prototype:

void E2PROM_rd_block (data char Src_Ptr, data char *data Dest_Ptr, data char Nr_Bytes)

Parameters:

Src_Ptr:	Address of first E^2PROM byte
Dest_Ptr:	Address pointer to internal RAM
Nr_Bytes:	Number of bytes to read from E^2PROM

2.9 E2PROM_wr_byte_pol

Function description:

This function will write a byte from internal RAM to E^2PROM.

If the source byte has the same value as the byte in the E^2PROM, no write action will take place.

If an E^2PROM function is in progress (except E2PROM_rd_byte_pol), this function will be interrupted but its status will be saved so that the interrupted function will be resumed when the E2PROM_wr_byte_pol function is finished.

Byte transfer is done by polling.

This function may be used, e.g., in interrupt service routines, where the possibility exists that the interrupted main program has already started an E^2PROM transfer.

Calling Sequence:

E2PROM_wr_byte_pol (Src_Byte,Dest_Ptr);

Function prototype:

void E2PROM_wr_byte_pol (data char Src_Byte,data char Dest_Ptr)

Parameters:

Src_Byte:	Byte to be written to E^2PROM
Dest_Ptr:	Address of E^2PROM

2.10 E2PROM_rd_byte_pol

Function description:

This function will read a byte from E^2PROM.

If an E^2PROM function is in progress (except E2PROM_wr_byte_pol), this function will be interrupted but its status will be saved so that the interrupted function will be resumed when the E2PROM_rd_byte_pol function is finished.

This function may be used, e.g., in interrupt service routines, where the possibility exists that the interrupted main program has already started an E^2PROM transfer.

Calling Sequence:

```
data char Result;
Result = E2PROM_rd_byte_pol (Src_Ptr);
```

Function prototype:

```
char E2PROM_Rd_Byte_Pol (data char Src_Ptr)
```

Parameters:

Src_Ptr:	Address of E^2PROM

2.11 E2PROM_block_erase

Function description:

This function will erase all 256 E^2PROM bytes.

The erase function is done on interrupt basis. The status of the transfer can be checked with the “E2PROM_Status” function.

This function will automatically enable the E^2PROM interrupt. The application program should take care of the E^2PROM/UART interrupt enable (with E2PROM_int_en) and the EA bit.

Calling Sequence:

```
E2PROM_block_erase();
```

Function prototype:

```
void E2PROM_block_erase (void)
```

Parameters:

None

2.12 E2PROM_security_on

Function description:

This function inhibits access to E^{2}PROM from external program memory.
The following scheme gives the access possibilities when this function is executed.

$\overline{EA}$ pin	Address of E2PROM access instruction	Access to E2PROM
1	< 4096	YES
1	>= 4096	NO
0	< 4096	NO
0	>= 4096	NO

The write function is done on interrupt basis. The status of the transfer can be checked with the “E2PROM Status” function.

This function will automatically enable the E^{2}PROM interrupt. The application program should take care of the E^{2}PROM/UART interrupt enable (with E2PROM_int_en) and the EA bit.

Calling Sequence:

E2PROM_security_on();

Function prototype:

void E2PROM_security_on (void)

Parameters:

None

2.13 E2PROM_security_off

Function description:

This function will remove E^{2}PROM protection. Access to E^{2}PROM from external program memory is now possible if this function is executed from the right program memory.

The following scheme gives the possibilities when 'E^{2}PROM_security_off' is executed after completion of the 'E^{2}PROM_security_on' function.

The following table assumes that the address at which 'E^{2}PROM_security_off' resides is smaller than 4096.

$\overline{EA}$ pin	Address of E2PROM access instruction	Mode	Access to E2PROM	E2PROM erased
1	< 4096	0	YES	NO
1	>= 4096	0	YES	NO
1	< 4096	1	YES	YES
1	>= 4096	1	YES	YES
0	< 4096	0	NO	NO
0	>= 4096	0	NO	NO
0	< 4096	1	YES	YES
0	>= 4096	1	YES	YES

The following table assumes that the address at which 'E^{2}PROM_security_off' resides is greater than 4096.

$\overline{EA}$ pin	Address of E2PROM access instruction	Mode	Access to E2PROM	E2PROM erased
1	< 4096	0	YES	NO
1	>= 4096	0	NO	NO
1	< 4096	1	YES	YES
1	>= 4096	1	NO	YES
0	< 4096	0	NO	NO
0	>= 4096	0	NO	NO
0	< 4096	1	YES	YES
0	>= 4096	1	YES	YES

Calling Sequence:

E2PROM_Security_Off;

Function prototype:

void E2PROM_Security_Off (data char Mode)

Parameters:

Mode: If '0', then the protection will only be removed when this function is executed from internal program memory. When executed from external memory, the protection remains.
If '1', then the protection can also be removed when this function is executed from external memory, however, all E^{2}PROM bytes will be erased.

3. PROGRAM EXAMPLES

Three examples are given that show how to use these functions with C, PL/M51 and assembly programs. In the examples, a string of characters is written to and read from E^{2}PROM. When reading back the string, spaces are replaced by underscores.

3.1 C example

The disk contains the file E2PROM.H that should be included in the C application program. E2PROM.H contains the function prototypes of the E^{2}PROM functions. The example program can be found on the disk in file TEST_C.C.

When the application module is compiled and assembled, it should be linked to the E^{2}PROM function module E2PROM.OBJ and the UART interrupt handler UART.OBJ.

3.2 PL/M51 example

The disk contains the file E2PROM.DCL that should be included in the PL/M51 application program. E2PROM.DCL contains the external function declarations for the E^{2}PROM functions. The example program can be found on the disk in file TEST_PLM.PLM.

When the application module is compiled and assembled, it should be linked to the E^{2}PROM function module E2PROM.OBJ and the UART interrupt handler UART.OBJ. When linking, the linking control ‘NOCASE’ must be used!

3.3 Assembly example

The disk contains the file E2PROM.MAC which contains the macro definitions of the functions. Including these macro’s in the source file eases programming. For instance, the sequence:

```
MOV _E2PROM_rd_block_BYTE  ,Src_Ptr        ;Pointer to E2PROM
MOV _E2PROM_rd_block_BYTE+1,Dest_Ptr       ;Pointer to RAM
MOV _E2PROM_rd_block_BYTE+2,#Nr_Bytes      ;Number of bytes to transfer
LCALL _E2PROM_rd_block                     ;Call function
```

can be replaced by

```
%E2PROM_rd_block(Src_Ptr,Dest_Ptr,#Nr_Bytes)
```

The file E2PROM.GLO contains the EXTRN-definitions of the functions and constants that are used by the driver. Only the definitions used by the source program should be included.

When the application module is compiled and assembled, it should be linked to the E^{2}PROM function module E2PROM.OBJ and the UART interrupt handler UART.OBJ.

3.4 Listing of examples

LISTING C EXAMPLE:

```
#include “E2PROM.h”
#include <string.h>
#define E2PROM_Base_Address  0x58

rom char txt_tab[]= “This is an E2PROM test for 8xC851”’

void main(void)
{
 data char      Data_Buffer[35];
 data char      Count;

 E2PROM_init();         /* Initialize E2PROM */
 E2PROM_int_en(0x01);   /* E2PROM interrupt level 1 */
 EA=1;                  /* Global interrupt enable */

 romidmove(&Data_Buffer,&Txt_tab,sizeof(Txt_tab)–1); /* Copy string from ROM
                                                      to RAM */
 E2PROM_wr_block(&Data_Buffer,E2PROM_Base_Address,sizeof(Txt_tab)–1);
                                                    /* Copy setring to E2PROM */

 /*  Time to do other useful things while data is being written to
    E2PROM on interrupt basis                                     */

 while (E2PROM_status()); /* Wait till transfer to E2PROM is finished */

 /* Read string from E2PROM and replace spaces “ ” by underscores “_” */
 for (Count=0;Count != sizeof(Txt_tab)–1;Count++)
     {
      /* Read E2PROM byte */
      Data_Buffer[Count] = E2PROM_rd_byte(E2PROM_Base_Address+Count);
      if (Data_Buffer[Count] == ‘ ’)
          Data_Buffer[Count] = ‘_’;
     }

 E2PROM_block_erase();  /* Erase E2PROM */
 /* Time to do other things while erasing */

 while (E2PROM_status()); /* Wait till erasing is finished */
 EA=0;
}
```

LISTING PL/M51 EXAMPLE:

```
$DEBUG
$CODE

E2PROM: Do;
$INCLUDE (E2PROM.DCL)
$INCLUDE (UTIL51.DCL)

        Test: Do;

        Declare E2PROM_Base_Address literally ’58H’;
        Declare Txt_tab(*) Byte Constant
                (’This is an E2PROM test for 8xC851’);
        Declare Data_Buffer(35) Byte Main;
        Declare Count Byte Main;

        Call E2PROM_init;         /* Initialize E2PROM */
        Call E2PROM_int_en(01);   /* E2PROM interrupt level 1 */
        Enable;                   /* Global interrupt enable */

        /* Copy string from ROM to RAM */
        Call MOVCD1(.Txt_tab,.Data_Buffer,length(Txt_tab));

        /* Copy string to E2PROM */
        Call E2PROM_wr_block(.Data_Buffer,E2PROM_Base_Address,length(Txt_tab));

        /*  Time to do other useful things while data is being written to
            E2PROM on interrupt basis                                     */

        Do While E2PROM_status = 1;  /* Wait till transfer to E2PROM is finished */
        End;

        /* Read string from E2PROM and replace spaces “ ” by underscores “_” */
        Do Count=0 To length(Txt_tab);
           /*Read E2PROM byte */
           Data_Buffer(Count) = E2PROM_rd_byte(E2PROM_Base_Address+Count);
           If (Data_Buffer(Count) = ‘ ’) then Data_Buffer(Count) = ‘_’;
        End;

        Call E2PROM_block_erase;  /* Erase E2PROM */
        /* Time to do other things while erasing */

        Do While E2PROM_status = 1; /* Wait till erasing is finished */
        End;
        Disable;

        End Test;
End E2PROM;
```

LISTING ASSEMBLY EXAMPLE:

```
$DEBUG
$CASE

;*=======================================================================*/
;*                                                                       */
;*                 INCLUDE FILE : E2PROM.GLO                             */
;*                 PACKAGE      : E2PROM                                 */
;*                                                                       */
;*=======================================================================*/

EXTRN CODE   (_E2PROM_init)

EXTRN CODE   (_E2PROM_int_en)
EXTRN CODE   (_E2PROM_int_en_BYTE)

EXTRN CODE   (_E2PROM_status)

EXTRN CODE   (_E2PROM_rd_byte)
EXTRN NUMBER (_E2PROM_rd_byte_BYTE)

EXTRN CODE   (_E2PROM_wr_block)
EXTRN NUMBER (_E2PROM_wr_block_BYTE)

EXTRN CODE (_E2PROM_block_erase)

;*=======================================================================*/
;*                     Include macro definitions                         */
;*=======================================================================*/
$INCLUDE(E2PROM.MAC)

        BUFFER SEGMENT DATA
        RSEG BUFFER
Data_Buffer:    ds 35
Count:          ds 1
Stack:          ds 15

        TABLE SEGMENT CODE
        RSEG TABLE
Txt_tab:        db   “This is an E2PROM test for 8xC851”

E2PROM_Base_Address     EQU 58H
Length_Txt              EQU 33

        CSEG AT 00              ;Reset vector
        LJMP MAIN

        TEST_ASM SEGMENT CODE
        RSEG TEST_ASM

MAIN:   MOV SP,#Stack–1         ;Initialize stack pointer
        %E2PROM_init            ;Initialize E2PROM
        %E2PROM_int_en(#10)     ;E2PROM interrupt level 1
        SETB EA                 ;Enable global interrupt

        MOV DPTR,#Txt_tab       ;Initialize pointers to copy Txt_tab to RAM
        MOV R0,#Data_Buffer
        MOV R2,#Length_Txt
```

```
COPY_LOOP:                      ;Copy Txt_tab to RAM
        CLR A
        MOVC A,@A+DPTR          ;Get byte from ROM
        MOV @R0,A               ;Store in RAM
        INC DPTR                ;Update pointers
        INC R0
        DJNZ R2,COPY_LOOP       ;Check if all copied

                                ;Write data to E2PROM
        %E2PROM_wr_block(#Data_Buffer,#E2PROM_Base_Address,#Length_Txt)
        ;
        ; Time to do other useful things while data is being written to
        ;  E2PROM on interrupt basis
        ;
NEW_CHECK:
        %E2PROM_status          ;Wait till transfer to E2PROM is finished
        JC NEW_CHECK

        ;Read string from E2PROM and replace spaces “ ” by underscores “_”
        MOV R0,#Data_Buffer     ;Initialize pointers
        MOV R1,#E2PROM_Base_Address
        MOV R2,#Length_Txt
READ_LOOP:
        %E2PROM_rd_byte(R1)
        MOV @R0,A               ;Store byte in RAM
        CJNE A,#“ ”,NEXT_READ   ;Check if byte is “ ”
        MOV @R0,#“_”            ;If yes, replace with “_”
NEXT_READ:
        INC R0                  ;Update pointers
        INC R1
        DJNZ R2,READ_LOOP

        %E2PROM_block_erase     ;Erase E2PROM

        ;Time to do other things while erasing */

NXT_CHECK:
        %E2PROM_status          ;Wait till transfer to E2PROM is finished
        JC NXT_CHECK
        CLR EA                  ;Disable interrupts
        JMP $                   ;End of program
```

4. DEBUG MACROS

The disk contains some debug macros that ease the debugging of programs that use the 8xC851 E^{2}PROM. These macros can be executed by the XRAY51 High Level Language debugger. The user can read from and write to E^{2}PROM bytes without programming the individual SFRs.

Before the macros can be executed, they must be loaded by XRAY51. This will be done automatically if the file ‘E2PROM.INC’ is included when invocating XRAY51 or during a debug session. E2PROM.INC will load the macros and define some symbols used by the macros. If not all macros are used, the file E2PROM.INC can be edited to prevent the loading of these macros. This may be necessary when there is insufficient memory to load the macros, because, for instance, other macros have been loaded. Another advantage of only loading the relevant macros is reduction of loading time.

When a macro is called from the debugger, the following SFRs will remain unchanged: ECNTRL, EADRH, EADRL and ETIM. During macro execution, all interrupts will be disabled. Access to the E^{2}PROM with the macros is independent of the state of the security bit. The execution and results of the macro are visible on the I/O screen of XRAY51 (VSCREEN 3).

Read(Start address, Stop address):

The value of E^{2}PROM bytes from ‘START ADDRESS’ to ‘STOP ADDRESS’ will be shown.
If ‘START ADDRESS’ <= ‘STOP ADDRESS’ only the value of ‘START ADDRESS’ will be shown.

Write(Start address, Stop address, Value):

The E^{2}PROM bytes from ‘START ADDRESS’ to ‘STOP ADDRESS’ will be programmed with ‘VALUE’.
If ‘START ADDRESS’ > ‘STOP ADDRESS’, no E2PROM bytes will be programmed.
If the ETIM register contains the value 0x08, it is considered that ETIM is not initialized. The macro will give a warning, and return to the debug screen.

Copyto(Ram address, E2PROM address, Count):

Macro will copy ‘COUNT’ bytes, starting from internal RAM address ‘RAM ADDRESS’ to the E^{2}PROM, starting at address ‘E2PROM ADDRESS’.
If during copying, the RAM address becomes > 0x7F or the E^{2}PROM address becomes > 0xFF, copying will be stopped and a warning is given that an address limit is reached.
If the ETIM register contains the value 0x08, it is considered that ETIM is not initialized. The macro will give a warning, and return to the debug screen.

Copyfrom(E2PROM address, Ram address, Count):

Macro will copy ‘COUNT’ bytes from E^{2}PROM address ‘E2PROM ADDRESS’ to the internal RAM, starting at address ‘RAM ADDRESS’.
If during copying, the RAM address becomes > 0x7F or the E^{2}PROM address becomes > 0xFF, copying will be stopped and a warning is given that an address limit is reached.

Erase():

All E^{2}PROM bytes will be erased.
If the ETIM register contains the value 0x08, it is considered that ETIM is not initialized. The macro will give a warning, and return to the debug screen.

5. CONTENTS OF DISK

The disk contains the following 3 directories:

1. \USER

This directory contains the files that may be included or linked to the source program.

E2PROM.ASM	:Source file of E^{2}PROM driver
E2PROM.OBJ	:Object file of E^{2}PROM driver
E2PROM.H	:Header file for C applications
E2PROM.DCL	:Declaration file for PL/M51
E2PROM.MAC	:Macro definitions for assembly applications
E2PROM.GLO	:Global definitions for assembly applications
UART.SRC	:UART interrupt handler (will only clear flags; user should customize it)
UART.OBJ	:Object file of UART interrupt handler

2. \DEBUG

This directory contains the macros and include file used with XRAY51 debugger.

E2PROM.INC	:Include file that reads macro files in XRAY51
*.MAC	:XRAY51 macros

3. \EXAMPLE

This directory contains the source files of the example programs described in the note

TEST_C.C	:C example
TEST_PLM.PLM	:PL/M51 example
TEST_ASM.ASM	:ASM51 example

Philips Semiconductors and Philips Electronics North America Corporation reserve the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified. Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

LIFE SUPPORT APPLICATIONS
Philips Semiconductors and Philips Electronics North America Corporation Products are not designed for use in life support appliances, devices, or systems where malfunction of a Philips Semiconductors and Philips Electronics North America Corporation Product can reasonably be expected to result in a personal injury. Philips Semiconductors and Philips Electronics North America Corporation customers using or selling Philips Semiconductors and Philips Electronics North America Corporation Products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors and Philips Electronics North America Corporation for any damages resulting from such improper use or sale.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381