PLM51 I^{2}C software interface IIC51 (version 0.5) ETV/AN89004

Author: R.C.J. Brink, Eindhoven

1. INTRODUCTION

1.1. Purpose

This document is a user manual for the I^{2}C software module IIC51. It is intended for Intel PLM51 users who need to control an I^{2}C bus. This document assumes some basic knowledge about I^{2}C and Intel PLM51.

1.2. Scope

IIC51 is a software module to provide an Intel PLM51 user with a set of procedures to control a bi-directional I^{2}C bus. These procedures have been coded in Intel ASM51 and have been optimized for speed. IIC51 supports all common used I^{2}C master transmitter and master receiver protocols. Each different protocol corresponds to one of the procedures in IIC51. IIC51 is available in two different versions:

IIC51S:
IIC51S is a module for singlemaster I^{2}C to be used on microcontrollers of the 8XC51 family. It directly controls the microcontroller I/O pins by software without the need of any specific hardware. No other I^{2}C masters are allowed on the bus. Note that the electrical characteristics of this microcontroller family are not conform the I^{2}C specifications.

IIC51M:
IIC51M is a module for multimaster I^{2}C to be used on microcontrollers of the PCB8XC552/C652 family. It makes use of the built-in I^{2}C interface hardware (SIO1) of these microcontrollers. Since this hardware is a multimaster interface, other I^{2}C masters are allowed on the bus.

All I^{2}C transfer procedures in IIC51S are fully software interface compatible with IIC51M. This allows a single PLM51 program using I^{2}C to be written for both mentioned microcontroller families.

1.3. Definitions, Acronyms and Abbreviations

I^{2}C	Inter-IC bus
PLM51	High level Program Language for 8051 family Microcontrollers
ASM51	Assembly Language for 8051 family Microcontrollers
RL51	Relocating Linker for 8051 family Microcontrollers
S	I^{2}C Message Start Condition
P	I^{2}C Message Stop Condition
A	I^{2}C Message Acknowledge
N	I^{2}C Message Negative Acknowledge
SlvW	I^{2}C Message Slave Address + Write
SlvR	I^{2}C Message Slave Address + Read
Sub	I^{2}C Slave Subaddress
NV-Memory	I^{2}C Controlled Non Volatile Memory (E^{2}PROM)

1.4. References

1. I^{2}C Specification
   I^{2}C-bus compatible ICs
   Philips Components Data Handbook IC12a 1989

2. PL/M-51 User’s Guide for DOS Systems
   Intel Corporation 1986

3. MSC-51 Macro Assembler User’s Guide for DOS Systems
   Intel Corporation 1986

4. MSC-51 Utilities User’s Guide for DOS Systems
   Intel Corporation 1986

5. Single-chip 8-bit microcontrollers PCB83C552/PCB80C552, PCB83C652/PCB80C652 etc.
   I^{2}C-bus compatible ICs
   Philips Components Data Handbook IC12a 1989

6. Single-chip 8-bit microcontroller PCB80C51
   Integrated circuits Book IC14 1987

2. GENERAL DESCRIPTION

2.1. Perspective

IIC51 is designed for use in stand-alone microcontroller I^{2}C systems. It is mainly written to provide a standard set of procedures for computer controlled television/teletext concepts based on 8051 family microcontrollers.

2.2. Functions

IIC51 contains the following functions:

– Initialisation of the I^{2}C interface (software and hardware)
– Transfer of I^{2}C messages to and from an I^{2}C slave device
– Error detection
– Automatic retrying if an error occurs during a transfer (up to 5 attempts)
– Error recovery if the bus is held by a slave device that is out of bit-sync
– Optional slave receiver/transmitter function (IIC51M only)

2.3. User Characteristics

IIC51 is designed to be an easy to use package. All needed code and data is defined in a single object module (IIC51M.OBJ or IIC51S.OBJ). The PLM51 user needs only to link this module to his own application object modules, using Intel's RL51. Procedures and data of concern to the user can be declared EXTERNAL by including the file IIC51.DCL.

2.4. General Constraints

IIC51 is coded for and translated by the Intel MSC-51 Macro Assembler. It is tested together with Intel PLM51 modules. Intel utilities used for testing:

– MSC-51 Macro Assembler, ASM51.EXE, Version V2.3
– PL/M-51 Compiler, PLM51.EXE, Version V1.2 and V1.3
– MSC-51 Relocator and Linker, RL51.EXE, Version V3.1

Development is done on an IBM-PC/AT running DOS.

IIC51S needs:

– 350 Bytes CODE (approx.)
– 6 Bytes DATA
– 1 Byte Bit-Addressable DATA
– 1 Bit

IIC5MS needs:

– 400 Bytes CODE (approx.)
– 6 Bytes DATA
– 1 Byte Bit-Addressable DATA
– 1 Bit
– Exclusive use of Register Bank 1

3. FUNCTIONAL DESCRIPTION

3.1. Master Mode Functions

This section describes the available functions in IIC51 on a procedure by procedure basis.

Each procedure must be declared EXTERNAL by the PLM51 user. In this declaration the user can specify the type returned by each procedure. All procedures (except Init_IIC) can return a BIT or a BYTE (depending on the chosen EXTERNAL declaration). The BIT or BYTE returned is 0 if the I^{2}C transmission was successful. If the user decides to declare a procedure untyped, the result of the previous I^{2}C transmission can always be checked by examining the static BIT variable IIC_Error. Note that typed procedures must be called using an expression. If the result of an I^{2}C procedure is to be ignored, a dummy assignment must be done for a typed procedure. An untyped procedure can be called by the PLM51 CALL statement, without any additional overhead. The examples in the following section assume the procedures to be declared untyped.

Note that the least significant bit of all slaveaddresses passed to the I^{2}C procedures must be 0.

3.1.1. Init_IIC

Declaration:

```
Init_IIC:
  PROCEDURE ( Own_Slave_Address ) EXTERNAL ;
  DECLARE   ( Own_Slave_Address ) BYTE ;
  END ;
```

Description:

Init_IIC must be called once after reset, before any other procedure is used. It initialises all I^{2}C internal static data and hardware. The Own_Slave_Address is passed to Init_IIC for the optional slave function in a multimaster I^{2}C system (IIC51M). In a singlemaster I^{2}C system (IIC51S), the Own_Slave_Address is ignored. Note that Init_IIC does not effect the global interrupt enable flag (EA). IIC51M requires the user to enable interrupts afterwards (see example).

Example:

```
CALL Init_IIC ( 54h ) ;
ENABLE ;               /*Enable Interrupts; EA = 1 */
```

3.1.2. IIC_Test_Device

Declaration:

```
IIC_Test_Device:
  PROCEDURE ( Slave_Address ) [ BIT | BYTE ] EXTERNAL ;
```

Description:

IIC_Test_Device just sends the slaveaddress on the I^{2}C bus. It can be used to check the presence of a device on the I^{2}C bus.

I^{2}C Protocol:

S	SlvW	A	P

(Device is Present, IIC_Error = 0)

OR

S	SlvW	N	P

(Device is Not Present, IIC_Error = 1)

Example:

```
DECLARE  IIC_Error BIT EXTERNAL ;
.......
CALL IIC_Test_Device ( 8Ch ) ;
IF (IIC_Error ) THEN
  “Device is Not Present Handling”
ELSE
  “Device is Present Handling”
```

3.1.3. IIC_Write

Declaration:

```
IIC_Write:
  PROCEDURE ( Slave_Address, Count, Source_Ptr ) [ BIT | BYTE ] EXTERNAL ;
  DECLARE   ( Slave_Address, Count, Source_Ptr ) BYTE ;
  END ;
```

Description:

IIC_Write is the most basic procedure to write a message to a slave device.

I^{2}C Protocol:

```
L          = Count
D1[0..L–1] BASED by Source_Ptr
```

S	SlvW	A	D1[0]	A	D1[1]	A	// ...... //	A	D1[L–1]	A	P

Example:

```
DECLARE Data_Buffer ( 4 ) BYTE ;
.....
CALL IIC_Write ( 0C2h, LENGTH ( Data_Buffer ), .Date_Buffer ) ;
```

3.1.4. IIC_Write_Sub

Declaration:

```
IIC_Write_Sub:
  PROCEDURE ( Slave_Address, Count, Source_Ptr, Sub_Address ) [ BIT | BYTE ] EXTERNAL ;
  DECLARE   ( Slave_Address, Count, Source_Ptr, Sub_Address ) BYTE ;
  END ;
```

Description:

IIC_Write_Sub writes a message preceded by a subaddress to a slave device.

I^{2}C Protocol:

```
L          = Count
Sub        = Sub_Address
D1[0..L–1] BASED by Source_Ptr
```

S	SlvW	A	Sub	A	D1[0]	A	D1[1]	A	// ...... //	A	D1[L–1]	A	P

Example:

```
DECLARE Data_Buffer ( 8 ) BYTE ;
.....
CALL IIC_Write_Sub ( 48h, LENGTH ( Data_Buffer ), .Date_Buffer, 2 ) ;
```

3.1.5. IIC_Write_Sub_SWInc

Declaration:

```
IIC_Write_Sub_SWInc:
   PROCEDURE ( Slave_Address, Count, Source_Ptr, Sub_Address) [ BIT| BYTE ] EXTERNAL ;
   DECLARE   ( Slave_Address, Count, Source_Ptr, Sub_Address ) BYTE ;
   END ;
```

Description:

Some I^{2}C devices addressed with a subaddress do not automatically increment the subaddress after reception of each byte. IIC_Write_Sub_SWInc can be used for such devices the same way IIC_Write_Sub is used. IIC_Write_Sub_SWInc splits up the message in smaller messages and increments the subaddress itself.

I^{2}C Protocol:

```
L           = Count
Sub         = Sub_Address
D1[0..L–1] BASED by Source_Ptr
```

S	SlvW	A	Sub	A	D1[0]	A	P

S	SlvW	A	Sub+1	A	D1[1]	A	P

....

S	SlvW	A	Sub+L–1	A	D1[L–1]	A	P

Example:

```
DECLARE Data_Buffer ( 6 ) BYTE ;
.....
CALL IIC_Write_Sub_SWInc ( 80h, LENGTH ( Data_Buffer ), .Date_Buffer, 0 ) ;
```

3.1.6. IIC_Write_Memory

Declaration:

```
IIC_Write_Memory:
   PROCEDURE ( Slave_Address, Count, Source_Ptr, Sub_Address) [ BIT| BYTE ] EXTERNAL ;
   DECLARE   ( Slave_Address, Count, Source_Ptr, Sub_Address ) BYTE ;
   END ;
```

Description:

I^{2}C Non-Volatile Memory devices (such as PCF8582) need an additional delay after writing a byte to it. IIC_Write_Memory can be used to write to such devices the same way IIC_Write_Sub is used. IIC_Write_Memory splits up the message in smaller messages and increments the subaddress itself. After transmission of each small message a delay of 40 milliseconds is inserted.

I^{2}C Protocol:

```
L           = Count
Sub         = Sub_Address
D1[0..L–1] BASED by Source_Ptr
```

S	SlvW	A	Sub	A	D1[0]	A	P	< Delay 40 ms >

S	SlvW	A	Sub+1	A	D1[1]	A	P	< Delay 40 ms >

....

S	SlvW	A	Sub+L–1	A	D1[L–1]	A	P	< Delay 40 ms >

Example:

```
DECLARE Data_Buffer ( 10 ) BYTE ;
.....
CALL IIC_Memory ( 0A0h, LENGTH ( Data_Buffer ), .Date_Buffer, 0F0h ) ;
```

3.1.7. IIC_Write_Sub_Write

Declaration:

```
IIC_Write_Sub_Write:
   PROCEDURE ( Slave_Address, Count1, Source_Ptr1, Sub_Address, Count2, Source_Ptr2 )
                                                         [ BIT| BYTE ] EXTERNAL ;
   DECLARE   ( Slave_Address, Count1, Source_Ptr1, Sub_Address, Count2, Source_Ptr2 ) BYTE ;
   END ;
```

Description:

IIC_Write_Sub_Write write 2 data blocks preceded by a subaddress in one message to a slave device. This procedure can be used for devices that need an extended addressing method, without the need to put all data into one large buffer. Such a device is the ECCT (I^{2}C controlled teletext device; see example).

I^{2}C Protocol:

```
L          = Count1
M          = Count2
Sub        = Sub_Address
D1[0..L–1] BASED by Source_Ptr1
D2[0..M–1] BASED by Source_Ptr2
```

S	SlvW	A	Sub	A	D1[0]	A	D1[1]	A	// ...... //	A	D1[L–1]	A

D2[0]	A	D2[1]	A	// ...... //	A	D2[M–1]	A	P

Example:

```
PROCEDURE Write_CCT_Memory ( Chapter, Row, Column, Data_Buf, Data_Count ) ;
DECLARE ( Chapter, Row, Column, Data_Buf, Data_Count ) BYTE;

 /*
     The extended address (CCT-Cursor) is formed by Chapter, Row and Column. These
     three bytes are written after the subaddress (8) followed by the actual data which
     will be stored relative to the extended address.
 */
 CALL IIC_Write_Sub_Write ( 22h, 3, .Chapter, 8, Data_Buf, Data_Count ) ;

END Write_CCT_Memory ;
```

3.1.8. IIC_Read

Declaration:

```
IIC_Read:
   PROCEDURE ( Slave_Address, Count, Dest_Ptr ) [ BIT | BYTE ] EXTERNAL ;
   DECLARE   ( Slave_Address, Count, Dest_Ptr ) BYTE ;
   END ;
```

Description:

ICC_Read is the most basic procedure to read a message from a slave device.

I^{2}C Protocol:

```
M          = Count
D2[0..M–1] BASED by Dest_Ptr
```

S	SlvR	A	D2[0]	A	D2[1]	A	// ...... //	A	D2[M–1]	N	P

Example:

```
DECLARE Data_Buffer ( 4 ) BYTE ;
.....
CALL IIC_Read ( 0B4h, LENGTH ( Data_Buffer ), .Data_Buffer ) ;
```

3.1.9. IIC_Read_Status

Declaration:

```
IIC_Read_Status:
   PROCEDURE ( Slave_Address, Dest_Ptr ) [ BIT | BYTE ] EXTERNAL ;
   DECLARE   ( Slave_Address, Dest_Ptr ) BYTE ;
   END ;
```

Description:

A lot of I^{2}C devices have only a one status byte that can be read via I^{2}C. IIC_Read_Status can be used for this purpose. IIC_Read_Status works the same as IIC_Read but the user does not have to pass a count parameter.

I^{2}C Protocol:

```
M          = Count
Status     BASED by Dest_Ptr
```

S	SlvR	A	Status	N	P

Example:

```
DECLARE Status_Byte BYTE ;
.....
CALL IIC_Read_Status ( 84h, .Status_Byte ) ;
```

3.1.10. IIC_Read_Sub

Declaration:

```
IIC_Read_Sub:
   PROCEDURE ( Slave_Address, Count, Dest_Ptr, Sub_Address ) [ BIT | BYTE ] EXTERNAL ;
   DECLARE   ( Slave_Address, Count, Dest_Ptr, Sub_Address ) BYTE ;
   END ;
```

Description:

IIC_Read_Sub reads a message from a slave device preceded by a write of the subaddress. Between writing the subaddress and reading the message, an I^{2}C restart condition is generated without surrendering the bus. This prevents other masters from accessing the slave device in between and overwriting the subaddress.

I^{2}C Protocol:

```
M          = Count
Sub        = Sub_Address
D2[0..M–1] BASED by Dest_Ptr
```

S	SlvW	A	Sub	A	S	SlvR	A	D2[0]	A	D2[1]	A	// ...... //	A	D2[M–1]	N	P

Example:

```
DECLARE Data_Buffer ( 5 ) BYTE ;
.....
CALL IIC_Write_Sub ( 0A2h, LENGTH ( Data_Buffer ), .Data_Buffer, 2 ) ;
```

3.1.11. IIC_Write_Sub_Read

Declaration:

```
IIC_Write_Sub_Read:
   PROCEDURE ( Slave_Address, Count1, Source_Ptr1, Sub_Address, Count2, Dest_Ptr2 )
                                                       [ BIT | BYTE ] EXTERNAL ;
   DECLARE   ( Slave_Address, Count1, Source_Ptr1, Sub_Address, Count2, Dest_Ptr2 ) BYTE ;
   END ;
```

Description:

IIC_Write_Sub_Read writes a data block preceded by a subaddress, generates an I^{2}C restart condition, and reads a data block. This procedure can be used for devices that need an extended addressing method. Such a device is the ECCT (I^{2}C controlled teletext device; see example).

I^{2}C Protocol:

```
L          = Count1
M          = Count2
Sub        = Sub_Address
D1[0..L–1] BASED by Source_Ptr1
D2[0..M–1] BASED by Dest_Ptr2
```

S	SlvW	A	Sub	A	D1[0]	A	D1[1]	A	......	A	D1[L–1]	A	

	S	SlvR	A	D2[0]	A	D2[1]	A	......	A	D2[M–1]	N	P

Example:

```
PROCEDURE Read_CCT_Memory ( Chapter,Row, Column, Data_Buf, Data_Count );
DECLARE ( Chapter, Row, Column, Data_Buf, Data_Count ) BYTE ;

 /*
   The extended address (CCT–Cursor) is formed by Chapter, Row and Column. These
   three bytes are written after the subaddress (8). After that the actual data will be
   read relative to the extended address.
 */
 CALL IIC_Write_Sub_Read ( 22h, 3, .Chapter, 8, Data_Buf, Data_Count ) ;

END Read_CCT_Memory ;
```

3.2. Slave Mode Functions

I^{2}C slave mode is provided by IIC51M only. All slave mode actions (except initialisation) take place in the SIO1 interrupt procedure. Slave mode I^{2}C protocol is very application dependent. If a specific slave mode is required, the user have to modify three procedures in IIC51M at source level. The following sections describe these procedures. The program examples of the procedures implement an I^{2}C slave protocol to read and write the microcontroller's on chip RAM via I^{2}C. This can be a useful feature during program development and debugging.

3.2.1. Init_Slave

This procedure is called from IIC_Init. In this procedure the user can initialise all static data concerning slave mode functions (if any).

Example:

```
Slave_Sub_Address:  db  1
....
Init_Slave:         mov Slave_Sub_Address,#0  ; Initialise Sub Address
                    ret
```

3.2.2. Receive_Slave

Receive_Slave is a procedure called from the SIO1 interrupt procedure each time a byte is received from another I^{2}C master. The procedure can make use of the bit "IICntrl.BYTE1EXPECTED", as defined in IIC51M. This bit is set to logic 1, every time the first data byte of an I^{2}C message is about to be received. Receive_Slave can use this bit to detect the start of a new message.

Normally all bytes received from the other master will be acknowledged (i.e., SIO1 control bit Assert Acknowledge is set, AA = 1). If AA is cleared by Receive_Slave subsequent bytes in the message will be ignored and a negative acknowledge will be transmitted after reception of each byte. Note that the example does not make use of this feature.

Constraints:

– Receive_Slave must read the S1DAT register.

– Receive_Slave may clear the SIO1 control bit AA, to stop acknowledging data.

– Receive_Slave may not effect any other SIO1 hardware registers/bits.

– Receive_Slave is only allowed to use the accumulator and register R0 in the current registerbank.

Example:

```
Receive_Slave:  mov a,S1DAT                         ; Pick up data
                mov r0,#Slave_Sub_Address           ; Prepare for 1st byte
                jbc IICCntrl.BYTE1EXPECTED,Save_Byte  ; Jump if 1st byte
                mov r0,Slave_Sub_Address            ; Else data byte
                inc Slave_Sub_Address               ; Postincrement Sub.
Save_Byte:      mov @r0,a                           ; Save Data
                ret                                 ; Exit
```

3.2.3. Send_Slave

Send_Slave is a procedure called during the SIO1 interrupt procedure each time a byte has to be transmitted to another I^{2}C master. This occurs after reception of I^{2}C startcondition followed by the microcontroller's own slaveaddress (as passed to Init_IIC) with read-bit. Send_Slave will be called again after transmission of each subsequent byte, until a negative acknowledge is received from the reading I^{2}C master.

Constraints:

– Send_Slave must write to the S1DAT register.

– Send_Slave may not effect any other SIO1 hardware registers/bits.

– Send_Slave is only allowed to use the accumulator and register R0 in the current registerbank.

Example:

```
Send_Slave:     mov r0,Slave_Sub_Address             ; Pick up Sub Address
                mov S1DAT,@r0                        ; Send Data
                inc Slave_Sub_Address                ; Postincrement Sub.
                ret
```