AN729

LIN Protocol Implementation Using PICmicro® MCUs

Authors: Dan Butler
Thomas Schmidt
Thorsten Waclawczyk
Microchip Technology Inc.

INTRODUCTION

LIN Protocol was designed by a consortium of European auto manufacturers as a low cost, short distance, low speed network. Designed to communicate changes in switch settings and respond to switch changes, it is intended to communicate events that happen in "human" time (hundreds of milliseconds).

This Application Note is not intended to replace or recreate the LIN Protocol Specification. Rather, it is intended to provide a broad overview of the bus and provide a high level look at how it works, how to implement a Slave node on a PICmicro® device and what it's designed to do. The complete LIN Protocol Specification is expected to be available via the worldwide web at www.lin-subbus.com. However, until then, copies of the LIN Protocol Specification may only be distributed by Audi AG, BMW AG, DaimlerChrysler AG, Motorola, Inc., Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation.

BUS FEATURES

LIN Protocol supports bi-directional communication on a single wire, while using inexpensive microcontrollers driven by RC oscillators, to avoid the cost of crystals or ceramic resonators. Instead of paying the price for accurate hardware, it pays the price in time and software. The protocol includes an autobaud step on every message. Transfer rates of up to 20Kbaud are supported, along with a low power SLEEP mode, where the bus is shut down to prevent draining the battery, but the bus can be powered up by any node on the bus.

The bus itself is a cross between I^2C^{TM} and RS232. The bus is pulled high via a resistor and each node pulls it low, via an open collector driver like I^2C. However, instead of having a clock line, each byte is marked via start and stop bits and the individual bits are asynchronously timed like RS232.

ELECTRICAL CONNECTIONS

Figure 1 shows a typical LIN Protocol configuration.

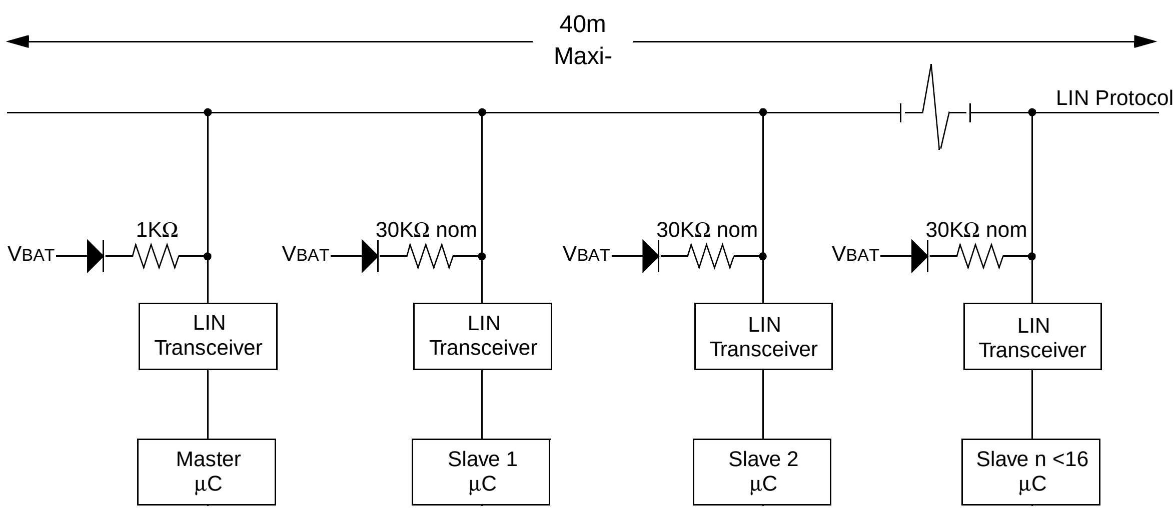
The bus uses a single wire pulled high through a resistor with open collector drivers. A Dominant state is signaled by a ground level on the bus and occurs when any node pulls the bus low. A Recessive state is when the bus is at VBAT (9 - 18V) and requires that all nodes let the bus float. In the idle state, the bus floats high, pulled up through the resistor.

The bus operates between 9V and 18V, but parts must survive 40V on the bus. Typically, the microcontroller is isolated from the bus levels by a line driver/receiver. This allows the microcontrollers to operate at 5V levels, while the bus operates at higher levels.

The bus is terminated to VBAT at each node. The Master is terminated through a 1KΩ resistor, while the Slaves are terminated through a 20-47KΩ resistor. Maximum bus length is designed to be 40 meters.

At press time (early 2000), K-Line drivers are used until true LIN drivers are available.

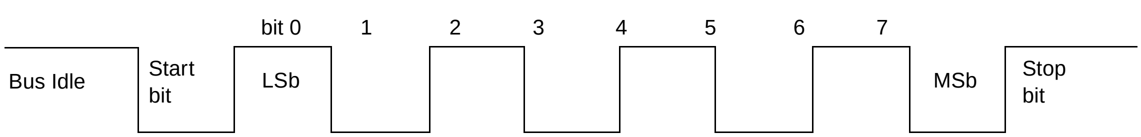
FIGURE 1: BUS CONFIGURATION



BYTE PROTOCOL

Each byte is framed by start and stop bits as shown in Figure 2. Within each byte, data is transmitted LSb first. The start bit is the opposite of the idle state or zero, and the stop bit equals the idle state (1).

FIGURE 2: BYTE PROTOCOL



Each byte is framed by a start bit and stop bit, and data is transmitted Least Significant bit first.

MESSAGE PROTOCOL

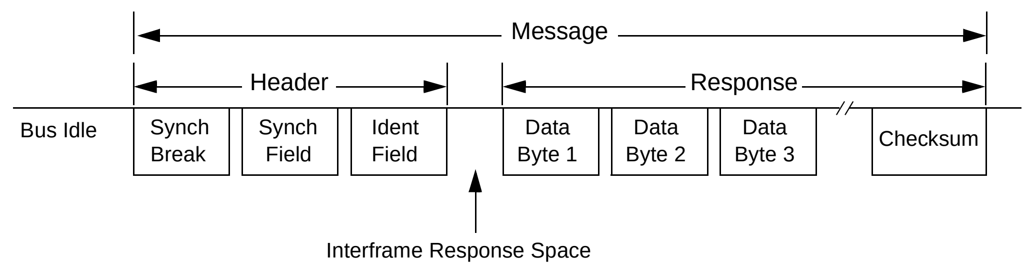
The Master controls the bus by polling Slaves to share their data with the rest of the bus. Slave nodes only transmit when commanded by the Master, which allows bi-directional communication without further arbitration. Message transfers start with the Master issuing a synch break, followed by a synch field and a message field. It also sets the clock for the entire bus by transmitting a synch field at the beginning of each message, which is used for clock synchronization. Each Slave must use this synch byte to adjust their baud rate.

The synch break is bus dominant, held for 13 bit times, followed by a stop bit (recessive). This lets the Slaves know that a message is coming. The Master and Slave clocks may have drifted as much as 15%. Therefore, the synch break may be received by a Slave as only 11 bit times, or as long as 15 bit times.

The second byte of each message is an ident byte, which tells the bus what data will follow and indicates which node should answer and how long the answer shall be. Only one Slave may respond to a given command.

Slaves only transmit data on the bus when directed by the Master. Once the data is on the bus, any node may receive that data. Therefore, communication from one Slave to another does not have to be directed through the Master.

FIGURE 3: MESSAGE PROTOCOL



CLOCK SYNCHRONIZATION

LIN Protocol is designed to use low cost RC oscillators on the controllers. To keep communication working as each node's clock drifts, Slaves must detect the Master's baud rate on every transfer and adjust to the current baud rate. For this reason, each transaction starts with a synch field. The synch field is a one byte 0x55 (alternating 0's and 1's). This allows every Slave node to detect 8 bit times. By counting these transitions, dividing by 8 and rounding, each Slave adjusts their timing to the Master.

IDENTIFIER FIELD

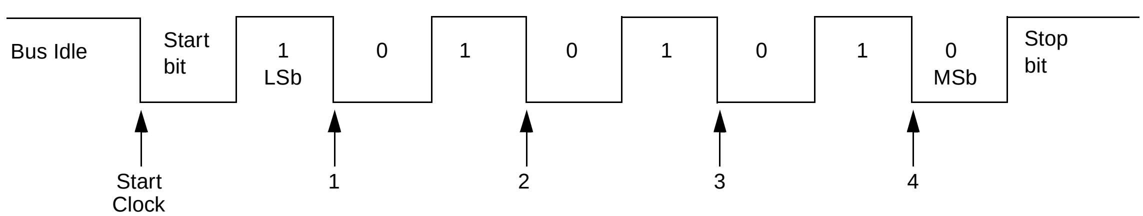
Following the synch field, is an identifier field, which tells the bus what's coming next. The ident field is broken up into 3 fields: 4 bits (0-3) address devices on the bus, 2 bits (4-5) indicate the length of the message to follow and the last 2 bits (6-7) are used for parity.

The 4 address bits can address up to 16 Slaves, and each Slave can send a 2, 4 or 8 byte response, for a total of 64 different messages.

The LIN Protocol Specification does not define the content of each message, except for the SLEEP command detailed in the Lower Power Sleep section. Instead, that is left up to the application.

Messages from one node to another may be sent directly, as directed by the Master. The data does not have to be received by the Master and retransmitted to the receiving node. Instead, any message may be received and acted upon by any node.

FIGURE 4: SYNCH FIELD



Count time for 4 successive falling edges, then divide by 8 and round, to get a single bit time. The divide and round is easily implemented as 3 right shifts and add the carry back in.

 Preliminary

FIGURE 5: IDENT FIELD

$\overline{P1}$	P0	ID5	ID4	ID3	ID2	ID1	ID0

P0: Parity bit ID0 ⊕ ID1 ⊕ ID2 ⊕ ID4

$\overline{P1}$: Parity bit ID1 ⊕ ID3 ⊕ ID4 ⊕ ID5

ID0 - 3: Device Address

ID4 - 5: Message Length

ID5	ID4	Date Bytes
0	0	2
0	1	2
1	0	4
1	1	8

ERROR DETECTION

The following errors must be detected and counted within each node:

- Bit Errors: The transmitting node should compare what it thinks should be on the bus against what actually is on the bus. The controllers must wait long enough for the bus to respond before testing for the bit. Given the minimum edge slew rates are 1V/uS, and the maximum bus voltage (18V), the transmitter should wait 18µS before testing, to see if the bit on the bus is correct.
- Checksum Errors: The data content of each message is protected by a checksum byte, which is the inverted module-256 checksum of the data bytes.
- Parity Errors: The command byte uses 2 parity bits to protect the other 6. These need to be recalculated and compared.

If there is an error, the command should be ignored and the error logged.

ERROR REPORTING

There is no direct error reporting mechanism. However, each Slave node is expected to track it's own errors. The Master may then request error status as part of a normal message protocol.

CANBUS INTERFACE

LIN Protocol is not directly compatible to CANBUS, however, it is anticipated that the two will operate in conjunction with one another. CANBUS might be used for communication throughout the car, while LIN Protocol would only be used within a small section of the car, say within the door.

A CAN-LIN Protocol interface node would be necessary to connect the two busses. The interface node would collect information from the LIN Protocol nodes and pass that information along on CANBUS.

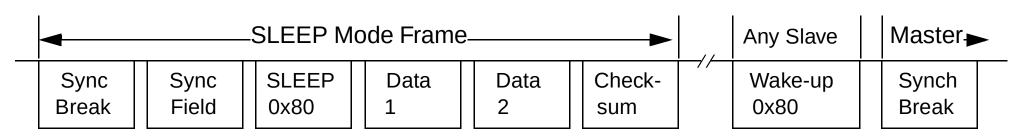
LOWER POWER SLEEP

The Master may direct all nodes to enter a SLEEP mode by sending a ident code of 0x80. This is the only message ID defined in the LIN Protocol Specification. The content of the data bytes following the SLEEP command is not defined. Slaves receiving the SLEEP command should set-up for a wake-up on change from the bus and power-down to minimum current drain. The bus will float high and not consume current.

Any node may wake-up the bus by sending a wake-up signal, which is a character 0x80 (low for 7 bit times followed by 1 bit time high). When this signal is received, all nodes should wake-up and wait for the Master to start polling the bus in the normal fashion.

If the Master fails to wake up after 128 bit times (6400uS @ 20Kbaud), the node that is attempting to wake the Master should try again. This may be attempted a total of 3 times before waiting 15000 bit times (750mS @ 20Kbaud).

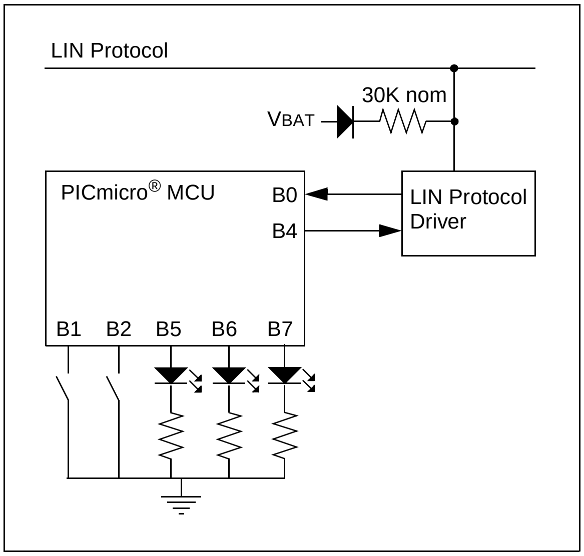
FIGURE 6: SLEEP MESSAGE



DEMONSTRATION SOFTWARE

The code in Appendix A demonstrates communication on the LIN Protocol. The hardware consists of 2 buttons and 3 LEDs, as shown in Figure 7. LED #1 changes state for every 10 button pushes of button #1. Likewise, LED #2 changes state for every 10 presses of button #2. In response to ID 1, the button counts are transmitted on the bus. In response to ID 4, the button counts are updated from the bus.

FIGURE 7: DEMONSTRATION HARDWARE



SOFTWARE OPERATION

The LIN Protocol code works on the interrupt as triggered from RB0. This is necessary to implement the SLEEP/Wake- up requirement. Once the interrupt is triggered, it counts the length of the low bit time. Then the synch byte is read and the local bit time is determined. This is then compared against the original bit time to determine if the original low time was more than 10 bit times and thus, signaled a synch break, or less than 10, signaling a wake up from SLEEP.

If it's a wake up from SLEEP, the code exits and continues waiting for a synch break.

If it's a synch break, it then reads in the command byte, checks the parity bits and checks the action table to determine it's actions from there. The action table defines the source or destination for the data on the bus.

SOFTWARE FUNCTION

In order to initialize the LIN Protocol Slave handler, the user has to call the routine `InitLinSlave`. This routine initializes the RB0 interrupt pin and the TMR0. TMR0 will be used to measure the bit length and to generate the baudrate. After initialization, the user can execute his code. The code will be interrupted once a falling edge is detected on RB0. If a falling edge is detected, the code will branch into the interrupt service routine. All interrupts, except for TMR0 and RB0, must be disabled to accurately time the synch field. After the baudrate is calculated, the interrupt service routine is exited. Upon the next interrupt on RB0, the LIN Protocol Slavehandler goes automatically into receive mode in order to receive the identifier field or data bytes. Once the start bit of the identifier field or data byte is detected, where the program branches into the interrupt service routine, the identifier field is received and decoded. Depending on the received identifier, code is executed, for example, storing data, turn on LED, etc. This code has to be included by the user into the subroutine `DecodeIdTable` after the routine is executed.

After the bus frame is completed, the flag `FCOMPLETE` is set. This flag indicates that all data is received correctly and ready for further processing. This flag has to be cleared by the user's firmware.

> **Note:** TMR0 is used for bit time measurement and baudrate generation. Therefore, TMR0 is not available to application software.

ERROR DETECTION

The Slave node process detects the following errors:

- Checksum error
- Bit errors
- Missing Stop bit
- Parity error
- Time-out errors

When the Slave code detects an error, it's recorded in the ERRORFLAGS register as shown in figure 8, and the message is ignored. These error flags are cleared once a valid ident field is received.

FIGURE 8: ERROR FLAGS

7	6 . . . 4	3	2	1	0
Time Out	Not Used	Bit Error	CRC Error	ID Parity Error	No Response

These flags are set when detected.

SOFTWARE PERFORMANCE

The LIN Protocol Slavehandler can operate up to a speed of 20Kbaud.

The LIN Protocol Slavehandler requires 420 words of program memory (not including program memory for macros for the In Out IDs) and 23 bytes of data memory.

This application is ideal for Microchip's internal RC oscillator operating at 4MHz.

INTEGRATION INTO CUSTOM CODE

The user has to edit the subroutine DecodeIDTable. In this section, the user defines what action has to be taken upon a certain identifier. Furthermore, the user can define what action has to be taken upon certain errors, (i.e., timing error or others).

APPENDIX A: SOURCE CODE

```
MPASM 02.30.09 Intermediate  LINSLAVE.ASM   1-27-2000  9:57:59         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                      ;                            Software License Agreement
                      ;
                      ; The software supplied herewith by Microchip Technology Incorporated (the "Company")
                      ; for its PICmicro® Microcontroller is intended and supplied to you, the Company’s
                      ; customer, for use solely and exclusively on Microchip PICmicro Microcontroller
                      ; products.
                      ;
                      ; The software is owned by the Company and/or its supplier, and is protected under
                      ; applicable copyright laws. All rights are reserved. Any use in violation of the
                      ; foregoing restrictions may subject the user to criminal sanctions under applicable
                      ; laws, as well as to civil liability for the breach of the terms and conditions of
                      ; this license.
                      ;
                      ; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
                      ; IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
                      ; MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE
                      ; COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
                      ; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
                      ;
                      ; ###############################################################################
                      ;       filename:               LINSLAVE.ASM
```

```
;                               LIN ( Local Interconnect Network ) SpecRev 1.0
;
; ###############################################################################
;
;       Author:                 Thorsten Waclawczyk
;       Company:                Arizona Microchip Technology GmbH
;
;       Revision:               1.7
;       Date:                   18-JAN-2000
;       Assembled using         MPASM 2.30.07
;
;################################################################################
;
;       include files:
;               p16C622.inc     Rev 1.01
;
;################################################################################
;
;       Implements the LIN slave handler conforming the LIN spec revision 1.0 with
;       using the external interupt INTE and TMR0 as an interrupt based system.
;
;       LIN HANDLER DESCRIPTION
;
;       The linslave handler must be initialized before starting the main task
;       by calling the routine "InitLinSlave".
;
;       After initialization the interrupt based LIN handler waits on first falling
;       and a second rising edge to measure the length of synchbreak lowtime.
;       Then the LIN handler waits for the next falling edge to count the processor
;       cycles between the next four falling edge detections.
;
;       This result is divided by 8 to get the bitlength. To make sure that
;       this is the header sequence, the bitlength is multiplied by 10 and
;       compared to the initial synchbreak count.  If the synchbreak count is
;       longer, we know it was a synchbreak.
;
;       After that the LIN handler is set in receive mode to read the identifier
;       by using the bitlength value set into the timer0.
;       When the byte has received it will be checked and decoded in the subroutine
;       "CheckIdentifierByte" so that the LIN handler can handle the incoming or
;       outgoing resonse frame.
;
;       The timing and error checking will be handled by the LIN handler so that
;       the only thing the user has to do is define the DecodeIdTable.
;       This gives an action to each of the 16 possible IDs that may come across
;       from the master.  Three actions are possible:   MacroLstMode, listen but
;       take no action on the data.  MacroRsMode, which provides a buffer for
```

```
;       recieving data.  MacroTxMode while sets up a buffer to transmit from.
;
;       When a complete message frame has been done a flag "COMFLAGS,FCOMPLETE"
;       is set.  So after that the user can process the data in the buffer
;       if there are no errors.  After data is processed the complete flag must
;       be cleared by the user to signal that the data has been processed.
;
;       DEMO PROGRAM DESCRIPTION
;
;       The slave node has two buttons and three LEDs. The buttons are read in
;       and counted.  The button counts are transmitted in response to ID1.
;
;       LEDs 1 and 2 change state when the value of the register corresponding
;       counter (COMPLED1 or COMPLED2) matches the delimiter value, currently
;       set to 10.
;
;       ID4 updates the counters via the LIN bus.
;
;       LED3 toggles each time it detects a complete message frame.
;
; ###############################################################################
;
;       what’s changed
;
;       changes                 date of changes
;
;
; ###############################################################################
;
;       program and data usage
;               program         :       0x0420
;               data            :       0x24
;               stacklevel      :       2
;
; ###############################################################################

        LIST p=16C622,F=INHX8M

#include <p16c622.inc>
        LIST
; P16C622.INC  Standard Header File, Version 1.01    Microchip Technology, Inc.
        LIST

2007   3FFB            __CONFIG   _CP_OFF & _WDT_OFF & _RC_OSC

; #### defines ##################################################################

```

```
                 #define RSLINEPIN       0                        ; PB0 connected to the receive line
                 #define TXLINEPIN       4                        ; PB4 connected to the send line

                 #define LINSLAVERAM     0x20                     ; startadress

                 ; ---- RAM location used by LINSLAVE modul -------------------------------------

                         CBLOCK  LINSLAVERAM
00000020                 COPYWREG                                 ; holds a copy of WREG
00000021                 COPYSTATUS                               ; holds a copy of STATUS

00000022                 HICOUNT:0,TIMEOUTLO,TIMEOUTHI            ; temporary highbyte for TMR0
                                                                  ; and timeout counter
00000024                 COUNTEDGES                               ;
00000025                 COUNTVALUE:0,COUNTVALUELO,COUNTVALUEHI   ; software counter for baudrate

00000027                 SYNCLENGTH:0,SYNCLENGTHLO,SYNCLENGTHHI
00000029                 BITREG:0,BITNBR,BITLENGTH                ; bitposition and bitlength
0000002B                 DATABLOCKLENGTH                          ; numbers of send/receive block
0000002C                 PREIDNUMBER                              ; carries the id field bits5..0
0000002D                 IDNUMBER                                 ; carries the ID_number
0000002E                 ERRORFLAGS                               ; protocols communication errors
0000002F                 COMFLAGS                                 ; linbus communication bits
00000030                 BUFFERPTR
00000031                 COMBUFFER
00000032                 DATACRC                                  ; checksum over xmit / rcsv block
00000033                 TXDATAFIELD:8                            ; data transmission buffer
0000003B                 RSDATAFIELD:8                            ; data receive block buffer
00000043                 DUMMY:2
                         ENDC

000000A0         MIRRCOPYWREG    EQU     (COPYWREG + 80h)         ; reserve RAM location on page1

                 #define LINSLAVEBLOCKLENGTH (DUMMY+1 - LINSLAVERAM)
                 #define LINBLOCKEND     (DUMMY+1)
                 ;############################################################
                 ;                                                ; calculate the needed RAM
                                                                  ; space by LINslave

                 ; #### bit defines #
                 ; defines for the errorflag variable

                 #define FTIMEOUT        7                        ; no communication on receive / transmit
                 #define FBITERROR       3                        ; an outgoing bit value is
                                                                  ; different than the line value
                 #define FCRCERROR       2                        ; CRC isn't correct
                 #define FIDPARITYERROR  1                        ; parity over ID not correct
```

```
         #define FNORESPONSE     0                       ; no echo on bus

         ; defines for the comflag variable

         #define FSYNCHBREAK     0                       ; synch_break detected
         #define FID             1                       ; ID byte receives as next byte
         #define FRSDATA         2                       ; data block read
         #define FTXDATA         3                       ; data block to be sent
         #define FCOMDATA        4                       ; communication is running
         #define FLISTENONLY     5                       ; ID byte is a command only for
         #define FCOMPLETE       6                       ; communication completed
         #define FSLEEPMODE      7                       ; sleep-mode frame


         ; ################################################################################
         ; #### main task declaration ######################################################
         ;
         ;       at this point used variables for the demo program will be defined
         ;

         #define LED1            5                       ; Toggles after 10 presses of button 1
         #define LED2            6                       ; toggles after 10 presses of button 2
         #define LED3            7                       ; indicates complete message frame

         #define BUTTON1         1                       ; input
         #define BUTTON2         2

                 CBLOCK  TXDATAFIELD
00000033         BUTTONPRESSED1                          ; register used by ID1
00000034         BUTTONPRESSED2                          ;
                 ENDC

                 CBLOCK  RSDATAFIELD
0000003B         COMPLED1                                ; register used by ID4
0000003C         COMPLED2                                ;
                 ENDC

         ; define global variables used by the main task

                 CBLOCK LINBLOCKEND+1
00000045         COMPERATOR                              ; contains the delimiter to
                                                         ; switch on/off the LEDs
00000046         B1FILTER                                ; debouncefilter button1

00000047         B2FILTER                                ; debouncefilter button2
00000048         DEBOUNCECOUNTER
00000049         COPYPORTB
```

```
0000004A                 EDGEDETECT
                         ENDC

                ; ################################################################################
                ;#################################################################################

                ; #### declare MODE macros ######################################################

                MacroRsMode MACRO       StartAddrOfPtr
                        bsf     COMFLAGS,FRSDATA                ; indicates the receive mode
                        movlw   StartAddrOfPtr                  ; get the wanted ram location
                                                                ; where the data will stored
                        movwf   BUFFERPTR                       ; and initialize the pointer
                        retlw   0
                        ENDM

                MacroTxMode MACRO       StartAddrOfPtr
                        bsf     COMFLAGS,FTXDATA                ; indicates the transmission mode
                        movlw   StartAddrOfPtr                  ; get the wanted ram location
                                                                ; where the data will read from
                        movwf   BUFFERPTR                       ; and initialize the pointer
                        retlw   0
                        ENDM

                MacroLstMode MACRO
                        bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                                                                ; without saving the received data
                        nop                                     ; this is necessary to have
                        nop                                     ; a correct jump table length
                        retlw   0
                        ENDM

                ; #### RESETvector ##############################################################

0000            RESET           org     0x0000                  ; reset-vector

0000   29A9             goto    Main

                ; #### INTvector ################################################################

0004            INTvector       org     0x0004
0004   00A0             movwf   COPYWREG                        ; save the used registers WREG
0005   0803             movf    STATUS,W                        ;
0006   1283             bcf     STATUS,RP0                      ; switch to bank0
0007   00A1             movwf   COPYSTATUS                      ; save STATUS in page0
0008            TMR0int
0008   1D0B             btfss   INTCON,T0IF
```

```
0009   2815           00253         goto    ExtInt                          ; if timer int enabled
                      00254
000A   0AA2           00255         incf    HICOUNT,F                       ; increment highbyte is also
000B   1D03           00256         btfss   STATUS,Z                        ; the lowbyte of timeout counter
000C   2810           00257         goto    TMR0Int1
                      00258
000D   0AA3           00259         incf    TIMEOUTHI,F                     ; timeout sequence after
000E   19A3           00260         btfsc   TIMEOUTHI,3                     ; 3000 * 256 cycles
000F   17AE           00261         bsf     ERRORFLAGS,FTIMEOUT             ; signal the timeout
                      00262
0010                  00263 TMR0Int1
0010   110B           00264         bcf     INTCON,T0IF                     ; clear overflow flag
0011   192F           00265         btfsc   COMFLAGS,FRSDATA                ; check if receiver mode
0012   2881           00266         goto    GetData
                      00267
0013   19AF           00268         btfsc   COMFLAGS,FTXDATA                ; check if transmit mode
0014   28BE           00269         goto    PutData
                      00270
0015                  00271 ExtInt
0015   1E0B           00272         btfss   INTCON,INTE                     ; check if external interupt
0016   28F8           00273         goto    IntrEnd                         ; is eanabled and
0017   1C8B           00274         btfss   INTCON,INTF                     ; possibly occured
0018   28F8           00275         goto    IntrEnd
                      00276
0019   3026           00277         movlw   0x26                            ; +12 cycles
001A   052F           00278         andwf   COMFLAGS,W                      ; if next byte is ID or falling
001B   1D03           00279         btfss   STATUS,Z                        ; edge of startbit initialize
001C   286D           00280         goto    GetDataInit                     ; reading a byte from bus
                      00281
001D   182F           00282         btfsc   COMFLAGS,FSYNCHBREAK            ; was it a synch_break?
001E   2832           00283         goto    CountSynchByte                  ; yes : then the sequence to measure
                      00284                                                 ; the synch_byte will be expected
001F   1683           00285         bsf     STATUS,RP0
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0020   1B01           00286         btfsc   OPTION_REG,INTEDG               ; is rising edge selected
0021   2827           00287         goto    CopySyncBreakLength             ; then save the count of synchbreak
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0022   1701           00288         bsf     OPTION_REG,INTEDG               ; else set rising edge sensitivity
0023   1283           00289         bcf     STATUS,RP0
                      00290
0024   0181           00291         clrf    TMR0                            ; initialize the used counter
0025   01A2           00292         clrf    HICOUNT                         ; to measure the synch break
0026   28F8           00293         goto    IntrEnd                         ; and now wait for rising edge int
                      00294
0027                  00295 CopySyncBreakLength
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0027   1301           00296         bcf     OPTION_REG,INTEDG               ; set the falling edge sensitivity
```

```
0028  1283          bcf     STATUS,RP0              ; to detect the startbit of
0029  0801          movf    TMR0,W                  ; the identifier byte
002A  00A7          movwf   SYNCLENGTH              ; save the counter value
002B  0822          movf    HICOUNT,W
002C  00A8          movwf   SYNCLENGTH+1            ; if the hibyte is cleared the
002D  1903          btfsc   STATUS,Z                ; received value was no synch break
002E  2868          goto    LowerSynchLength        ; sequence so start another
                                                    ; loop to detect a synch break
002F  01A4          clrf    COUNTEDGES              ; otherwise initialize the synch
0030  142F          bsf     COMFLAGS,FSYNCHBREAK    ; byte detection and measurement
0031  28F8          goto    IntrEnd

      ; ---- CountSynchByte -----------------------------------------------------------
      ;
      ;       Counts the cycles between five falling edges to calculate single bitlength
      ;       for communicating with the master.  The first falling edge clears the count
      ;       registers.  After the fifth edge, the bitlength is calculated by dividing
      ;       the count by 8 and rounding.
      ;
0032        CountSynchByte                          ; uses 17 cycles to arrive
0032  0824          movf    COUNTEDGES,W
0033  1D03          btfss   STATUS,Z                ; if first falling edge then
0034  2837          goto    CountSynchEdges
0035  0181          clrf    TMR0                    ;initialize bytelength couter
0036  01A2          clrf    HICOUNT

0037        CountSynchEdges
0037  1924          btfsc   COUNTEDGES,2            ; else check count of passed edges
0038  283B          goto    GetSynchLength
0039  0AA4          incf    COUNTEDGES,F            ; wait for next falling edge
003A  28F8          goto    IntrEnd
003B        GetSynchLength
003B  190B          btfsc   INTCON,T0IF             ; if all edges counted in
003C  0AA2          incf    HICOUNT,F               ; calculate actual bitlength in
003D  0801          movf    TMR0,W                  ; cycle counts by
003E  00AA          movwf   BITLENGTH
003F  1003          bcf     STATUS,C                ; dividing the 16Bit result by 8
0040  0CA2          rrf     HICOUNT,F
0041  0CAA          rrf     BITLENGTH,F
0042  1003          bcf     STATUS,C
0043  0CA2          rrf     HICOUNT,F
0044  0CAA          rrf     BITLENGTH,F
0045  1003          bcf     STATUS,C
0046  0CA2          rrf     HICOUNT,F
0047  0CAA          rrf     BITLENGTH,F             ; result is in bitlength
0048  082A          movf    BITLENGTH,W             ;
0049  3C31          sublw   .49                     ; BITLENGTH lower than fastest
```



```
004A  1803           btfsc   STATUS,C                  ; bitrate
004B  2868           goto    LowerSynchLength

            ; ---- CheckSynchBreakLength ---------------------------------------------------
            ;
            ;        Multiplies the counted bitlength by 10 to see if the synch break
            ;        is longer than a normal data byte.  A normal data byte contains a dominant
            ;        startbit, 8 databits and a recessive stopbit, so if the measured lowtime
            ;        of the synchbreak is longer than 10 bitlength times it's a synchbreak.
            ;

004C        CheckSynchBreakLength
004C  01C4           clrf    DUMMY+1                   ; no -> do synch break length
004D  082A           movf    BITLENGTH,W               ; store bitlength
004E  00C3           movwf   DUMMY
004F  1003           bcf     STATUS,C
0050  0DC3           rlf     DUMMY,F                   ; multiply bitlengh with 8
0051  0DC4           rlf     DUMMY+1,F                 ;
0052  0DC3           rlf     DUMMY,F
0053  0DC4           rlf     DUMMY+1,F
0054  0DC3           rlf     DUMMY,F
0055  0DC4           rlf     DUMMY+1,F
0056  07C3           addwf   DUMMY,F                   ; and add bitlength value
0057  1803           btfsc   STATUS,C                  ; twice to multiply by 10
0058  0AC4           incf    DUMMY+1,F
0059  07C3           addwf   DUMMY,F
005A  1803           btfsc   STATUS,C
005B  0AC4           incf    DUMMY+1,F
005C  0844           movf    DUMMY+1,W
005D  0228           subwf   SYNCLENGTH+1,W            ; first check the highbytes
005E  1C03           btfss   STATUS,C                  ; bitlength *10 < synch length
005F  2868           goto    LowerSynchLength          ; yes -> no header detected
0060  1D03           btfss   STATUS,Z                  ; bitlength *10 = synch length
0061  286B           goto    HigherSyncLength          ; no -> header detected
0062  0843           movf    DUMMY,W                   ; highbytes are equal
0063  0227           subwf   SYNCLENGTH,W              ; check the lowbytes
0064  1C03           btfss   STATUS,C
0065  2868           goto    LowerSynchLength
0066  1D03           btfss   STATUS,Z
0067  286B           goto    HigherSyncLength

0068        LowerSynchLength
0068  01AF           clrf    COMFLAGS                  ; reset the linslave
0069  160B           bsf     INTCON,INTE               ; allow external interrupt
006A  28F8           goto    IntrEnd                   ; to dedicate first faling edge

006B        HigherSyncLength
```

```
006B  14AF          bsf    COMFLAGS,FID              ; next databyte comming in is the
006C  28F8          goto   IntrEnd                   ; identifier byte

              ; ---- GetDataInit --------------------------------------------------------------
              ;
              ;       This function initializes the bitlength timer (TMR0) when the falling
              ;       edge of an start bit is detected so reading the bit takes place in the
              ;       center of the bit time.
              ;
              ;       For typical communication speeds 9600 to 19.2kbaud, we can set timer 0
              ;       to 1.5 bit times so we skip the remainder of the start bit and wake up
              ;       midway through the 1st data bit.  However for slow baud rates, slower
              ;       than about 6 kbaud, 1.5 bit times may overflow the 8 bit timer.  In
              ;       this case we set timer 0 to a half bit time so we'll wake up mid way
              ;       through the start bit and adjust the bit counter to account for the
              ;       extra bit.
              ;

006D          GetDataInit
006D  01B1          clrf   COMBUFFER                 ; clear input buffer
006E  01A9          clrf   BITNBR                    ; and the bit counter
006F  132F          bcf    COMFLAGS,FCOMPLETE        ; clear communication complete
                                                     ; indicator
0070  082A          movf   BITLENGTH,W               ; check if baudrate is less than
0071  3CA6          sublw  .166                      ; ca. 6 kBd
0072  1C03          btfss  STATUS,C                  ; if higher then
0073  2879          goto   SetHalfStartbitLength     ; set center position into startbit

0074  1003          bcf    STATUS,C                  ; else center position into
0075  0AA9          incf   BITNBR,F                  ; first databit by calculating
0076  0C2A          rrf    BITLENGTH,W               ; one and a half bitlength
0077  072A          addwf  BITLENGTH,W
0078  287A          goto   GetDataInitEnd

0079          SetHalfStartbitLength
0079  0C2A          rrf    BITLENGTH,W               ; center stopbit

007A          GetDataInitEnd
007A  3AFF          xorlw  0xff                      ; build complement for timer
007B  3E3E          addlw  (.62)                     ; correct timer to center stopbit
007C  0081          movwf  TMR0
007D  3020          movlw  0x20                      ; INTE off,T0IE on, clear flags
007E  008B          movwf  INTCON                    ; set interrupts
007F  152F          bsf    COMFLAGS,FRSDATA          ; indicates receive mode
0080  28F8          goto   IntrEnd

              ; ---- GetData ------------------------------------------------------------------
```

```
                ;
                ;       Timer 0 has been set up to return here in time to test the incoming bit
                ;       midway through the bit time.  This function samples the bit and rotates
                ;       the bit value into the data buffer.
                ;
                ;       If it was an identifier byte, we call DecodeIDTable to figure out what
                ;       to do next (Recieve message, transmit message or ignore message).
                ;       If it was a data byte, it is stored in the buffer previously set up by
                ;       the call to DecodeIDTable.  When the all data has been received, the
                ;       checksum is calculated and compared against the transmitted value.  If
                ;       there are no errors, the FCOMPLETE flag is set to signal that the
                ;       buffer should be processed.
                ;
                ;       We sample after TMR0 overflows, the routine copies the pin value into the
                ;       communication buffer and at checks the stopbit level to see if the data
                ;       is correct.  After receiving all bits of a byte there are two possibilities.
                ;
                ;       1.      The received byte is the identifier byte.  At this point the
                ;               routine analyzes the byte to either set up the receive buffer,
                ;               transmit buffer or neither as designated by the function
                ;               "DecodeIDTable".
                ;
                ;       2.      The received byte is a data byte of the response frame.
                ;               The data is stored in the receive buffer and the value is added
                ;               with the previous carry to build a modulo 256 checksum.  The
                ;               last byte is the message checksum, which is compared with the
                ;               calculated checksum to check for message integrity.
                ;

0081            GetData                                         ; delay of 13 cycles
0081   01A2             clrf    TIMEOUTLO                       ; reset the timeout counter
0082   01A3             clrf    TIMEOUTHI
0083            CheckBitPosition
0083   3009             movlw   .9                              ; how many bits left
0084   0629             xorwf   BITNBR,W                        ; stopbit received ?
0085   1903             btfsc   STATUS,Z                        ; no : copy receive pin value
0086   2891             goto    GetStopbit

0087   1C06             btfss   PORTB,RSLINEPIN                 ; copy portvalue in data buffer
0088   1003             bcf     STATUS,C                        ; using the Carryflag
0089   1806             btfsc   PORTB,RSLINEPIN
008A   1403             bsf     STATUS,C
008B   0CB1             rrf     COMBUFFER,F                     ; shift carry into the buffer

008C            GetDataSetTMR
008C   0AA9             incf    BITNBR,F                        ; count received bit
008D   092A             comf    BITLENGTH,W                     ; and center TMR0 to next bit
```

```
008E   3E1F             addlw   (.31)                       ; with the correct value
008F   0081             movwf   TMR0
0090   28BD             goto    GetDataEnd

0091             GetStopbit
0091   1C06             btfss   PORTB,RSLINEPIN             ; check polarity of stopbit
0092   15AE             bsf     ERRORFLAGS,FBITERROR        ; if low level set error flag

0093   18AF             btfsc   COMFLAGS,FID                ; if identifier byte is received
0094   28AF             goto    GetAction                   ; check this byte to initialize
                                                            ; the commanded slave mode
0095   032B             decf    DATABLOCKLENGTH,W           ; else if all bytes came in
0096   1903             btfsc   STATUS,Z
0097   28A7             goto    GetCheckCRC                 ; the last byte is the Checksum
0098   03AB             decf    DATABLOCKLENGTH,F           ; else wait for the next byte

0099   082E             movf    ERRORFLAGS,W                ; check if there are no errors
009A   1D03             btfss   STATUS,Z                    ; if an error has detected
009B   28A5             goto    SetNextLoc                  ; don't store the value

009C   1AAF             btfsc   COMFLAGS,FLISTENONLY        ; or if slave monitors line
009D   28A5             goto    SetNextLoc                  ; ignore the next steps

009E   0830             movf    BUFFERPTR,W                 ; get pointer
009F   0084             movwf   FSR                         ; and point to location
00A0   0831             movf    COMBUFFER,W                 ; catch the actual data
00A1   0080             movwf   INDF                        ; and ship it into the block

                 ;
                 ;      caluclate the checksum over received data bytes with MOD256
                 ;      DATACRC = DATACRC + COMBUFFER + Carry
                 ;
00A2   07B2             addwf   DATACRC,F                   ; add new data into CRC
00A3   1803             btfsc   STATUS,C                    ; add carry if mod 256
00A4   0AB2             incf    DATACRC,F                   ; produces an overflow

00A5             SetNextLoc
00A5   0AB0             incf    BUFFERPTR,F                 ; point to next location
00A6   28B0             goto    InitGetData

00A7             GetCheckCRC
00A7   1AAF             btfsc   COMFLAGS,FLISTENONLY        ; ignore checksum calculation
00A8   28B7             goto    GetDataFinish               ; if slave reads only messages

                 ;
                 ; generate sum of inverted mod 256 over data block plus CRC is 0xFF
                 ; calculate 0xFF - SUM[RSDATAFIELD] = received CRC
```


Preliminary

```
00A9   30FF                  movlw   0xFF                        ; do the complement of CRC
00AA   0632                  xorwf   DATACRC,W

00AB   0631                  xorwf   COMBUFFER,W                 ; check received CRC with
00AC   1D03                  btfss   STATUS,Z                    ; calculated CRC
00AD   152E                  bsf     ERRORFLAGS,FCRCERROR        ; set flag if not identical
00AE   28B7                  goto    GetDataFinish

00AF                  GetAction
00AF   212C                  call    CheckIdentifierByte         ; decode and check the ID byte

00B0                  InitGetData
00B0   01A9                  clrf    BITNBR                      ; clear register for next action
00B1   1DAF                  btfss   COMFLAGS,FTXDATA            ; if next action transmission
00B2   28BA                  goto    GetDataFinish+3             ; initialize only the interrupts

00B3   120B                  bcf     INTCON,INTE                 ; disable external interrupt
00B4   092A                  comf    BITLENGTH,W                 ; do a delay before sending data
00B5   0081                  movwf   TMR0
00B6   28BD                  goto    GetDataEnd

00B7                  GetDataFinish
00B7   30C0                  movlw   0xC0
00B8   05AF                  andwf   COMFLAGS,F                  ; clear all flags used bits
00B9   172F                  bsf     COMFLAGS,FCOMPLETE          ; indicates receive mode complete
00BA   3010                  movlw   0x10                        ; no T0IE, set INTE, clear flags
00BB   008B                  movwf   INTCON                      ; wait for next falling synch break
00BC   0181                  clrf    TMR0

00BD                  GetDataEnd
                      ;      bcf     INTCON,T0IF
00BD   28F8                  goto    IntrEnd

                      ; ---- PutData -----------------------------------------------------------------
                      ;
                      ;      This routine shifts out all wanted databits including a start and stopbit
                      ;      By setting the stopbit the routine will calulate the mod256 checksum
                      ;      and after all outgoing databit the checksum byte will be sent out.
                      ;

00BE                  PutData
00BE   01A2                  clrf    TIMEOUTLO
00BF   01A3                  clrf    TIMEOUTHI

00C0   0829                  movf    BITNBR,W                    ; if first call set startbit
00C1   1003                  bcf     STATUS,C
```

```
00C2  1903        00579         btfsc   STATUS,Z
00C3  28D7        00580         goto    SetStartbit
                  00581
00C4  0829        00582         movf    BITNBR,W                ; if all bits are out
00C5  3A09        00583         xorlw   .9                      ; do the stopbit
00C6  1903        00584         btfsc   STATUS,Z
00C7  28E3        00585         goto    SetStopbit
                  00586
00C8              00587 ShiftDatabitOut
00C8  0CB1        00588         rrf     COMBUFFER,F             ; else copy bits into C
00C9  0806        00589         movf    PORTB,W                 ; get Portvalue
00CA  39EF        00590         andlw   ((1<<TXLINEPIN)^0xFF)   ; clear used bit
00CB  1803        00591         btfsc   STATUS,C                ; and then set txline value
00CC  3810        00592         iorlw   (1<<TXLINEPIN)          ; if nessasary
00CD  0086        00593         movwf   PORTB
                  00594
00CE              00595 CheckBitPending
00CE  0DC3        00596         rlf     DUMMY,F                 ; shift carry into variable
00CF  0606        00597         xorwf   PORTB,W                 ; test polarity of incomming bit
00D0  3901        00598         andlw   (1<<RSLINEPIN)          ; maskout of the indicated bit
00D1  1D03        00599         btfss   STATUS,Z                ; if same polarity it’s okay
00D2  15AE        00600         bsf     ERRORFLAGS,FBITERROR
                  00601
00D3              00602 PutDataSetTMR0
00D3  092A        00603         comf    BITLENGTH,W             ; preload timer with bitlength
00D4  3E27        00604         addlw   (.39)                   ; correct the timer
00D5  0081        00605         movwf   TMR0
00D6  28F7        00606         goto    PutDataEnd
                  00607
00D7              00608 SetStartbit
00D7  1206        00609         bcf     PORTB,TXLINEPIN
00D8  0830        00610         movf    BUFFERPTR,W
00D9  0084        00611         movwf   FSR                     ; and point to it
00DA  0800        00612         movf    INDF,W                  ; fetch the actual data
00DB  00B1        00613         movwf   COMBUFFER
                  00614
                  00615 ;
                  00616 ;  generate CRC with MOD 256 -> DATACRC = COMBUFFER + DATACRC + Carry
                  00617 ;
00DC              00618 CalculateTxCRC
00DC  07B2        00619         addwf   DATACRC,F               ; create checksum
00DD  1803        00620         btfsc   STATUS,C                ; add Carry onto data_CRC
00DE  0AB2        00621         incf    DATACRC,F               ; if overflow occurs
00DF  092A        00622         comf    BITLENGTH,W             ; preload timer with bitlength
00E0  3E2E        00623         addlw   (.46)                   ; correct the timer
00E1  0081        00624         movwf   TMR0
00E2  28F7        00625         goto    PutDataEnd
```

```
00E3                      SetStopbit
00E3   30FF                       movlw   -1
00E4   00A9                       movwf   BITNBR                  ; prepare for clearing reg
00E5   1606                       bsf     PORTB,TXLINEPIN         ; set line for stopbit
00E6   0181                       clrf    TMR0                    ; wait 1*Tbit + x us
00E7   0AB0                       incf    BUFFERPTR,F             ; array ptr to next cell
00E8   03AB                       decf    DATABLOCKLENGTH,F       ; all data bytes xmit ?
00E9   082B                       movf    DATABLOCKLENGTH,W
00EA   1903                       btfsc   STATUS,Z                ; all data bytes out include
00EB   28F4                       goto    PutDataFinish           ; the CRC byte ?
00EC   3A01                       xorlw   .1                      ; no : is there only the CRC byte ?
00ED   1D03                       btfss   STATUS,Z
00EE   28F7                       goto    PutDataEnd              ; no : okay wait for the next bit

                          ;
                          ; generate inverted MOD256 Checksum
                          ; COMBUFFER = DATACRC XOR 0xFF, so SUM(TxData_field)+DATACRC = 0xFF
                          ;

00EF                      SetPtr2CRC
00EF   30FF                       movlw   0xFF
00F0   06B2                       xorwf   DATACRC,F               ; generate correct checksum

00F1   3032                       movlw   DATACRC                 ; get adress of checksum reg
00F2   00B0                       movwf   BUFFERPTR               ; and save value
00F3   28F7                       goto    PutDataEnd

00F4                      PutDataFinish
00F4   01AF                       clrf    COMFLAGS                ; reset communication flags
00F5   172F                       bsf     COMFLAGS,FCOMPLETE      ; indicate the complete flag
00F6   160B                       bsf     INTCON,INTE             ; and new start with
                                                                  ; synchbreak detection
00F7                      PutDataEnd
00F7   0AA9                       incf    BITNBR,F                ; set bitcounter to zero

                          ; ---- IntrEnd -----------------------------------------------------------------
                          ;
                          ;       Restore the saved registers and clear the interrupt flags, return from
                          ;       interrupt.

00F8                      IntrEnd
00F8   30F0                       movlw   0xF0                    ; clear all pending int flags
00F9   058B                       andwf   INTCON,F
00FA   168B                       bsf     INTCON,T0IE             ; and set timer0 overflow int

00FB   0821                       movf    COPYSTATUS,W            ; restore status
```

```
00FC   0083          00673         movwf   STATUS
00FD   0EA0          00674         swapf   COPYWREG,F                      ; and fetch old W from further
00FE   0E20          00675         swapf   COPYWREG,W                      ; ram bank
00FF   0009          00676         retfie
                     00677
                     00678
                     00679 ; #### subroutines ##############################################################
                     00680 ;
                     00681 ;
                     00682 ; #### init_LinSlave ############################################################
                     00683 ;
                     00684 ;       clears all ram locations which are used from the linslave and initializes
                     00685 ;       the portpins and interruptflags
                     00686 ;
                     00687
0100                 00688 InitLinSlave
0100   3020          00689         movlw   LINSLAVERAM
0101   0084          00690         movwf   FSR
0102   3024          00691         movlw   LINSLAVEBLOCKLENGTH
0103                 00692 ClearLINusedRAM
0103   0180          00693         clrf    INDF                            ; clear register
0104   0A84          00694         incf    FSR,F                           ; point to next cell
0105   0804          00695         movf    FSR,W
0106   3C44          00696         sublw   (LINSLAVEBLOCKLENGTH+LINSLAVERAM)
0107   1D03          00697         btfss   STATUS,Z                        ; all done well?
0108   2903          00698         goto    ClearLINusedRAM                 ; no : next loop
                     00699
0109   1606          00700         bsf     PORTB,TXLINEPIN                 ; Portpin is high on programstart
010A   1683          00701         bsf     STATUS,RP0
010B   1406          00702         bsf     PORTB,RSLINEPIN                 ; receive line as input
010C   1206          00703         bcf     PORTB,TXLINEPIN                 ; and transmit line as output
010D   3008          00704         movlw   0x08                            ; set TMR0 as counter driven
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
010E   0081          00705         movwf   OPTION_REG                      ; with internal clock
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
010F   1301          00706         bcf     OPTION_REG,INTEDG
0110   1283          00707         bcf     STATUS,RP0
0111   108B          00708         bcf     INTCON,INTF
0112   160B          00709         bsf     INTCON,INTE
0113   178B          00710         bsf     INTCON,GIE
0114   0008          00711         return
                     00712
                     00713 ; #### InitWakeupLIN ############################################################
                     00714 ;
                     00715 ;       This wakes up the bus by pulling the bus low for 7 bit times followed by
                     00716 ;       a high stop bit.  This would be used after the bus had been put to sleep,
                     00717 ;       and the node needs to wake up the bus.  One the bus has reawakened, the
```

```
                       00718 ;       master starts polling the slaves to find out why.
                       00719 ;
                       00720
0115                   00721 InitWakeupLIN
0115   082A            00722         movf    BITLENGTH,W
0116   1903            00723         btfsc   STATUS,Z                 ; if there was no further communication
0117   2926            00724         goto    InitWakeupEnd            ; do not do this function
0118   3001            00725         movlw   .1
0119   00AB            00726         movwf   DATABLOCKLENGTH          ; realize a one byte data block using
011A   3080            00727         movlw   0x80                     ; the CRC reg as data container
011B   00B2            00728         movwf   DATACRC
011C   3032            00729         movlw   DATACRC                  ; initialize data pointer onto CRC
011D   00B0            00730         movwf   BUFFERPTR
011E   01A9            00731         clrf    BITNBR                   ; and start transmission
011F   15AF            00732         bsf     COMFLAGS,FTXDATA
                       00733
0120   0181            00734         clrf    TMR0                     ; do a delay before sending out the
                       00735                                          ; wakeup sequence
0121   30A0            00736         movlw   0xA0                     ; set TMR0 int
0122   008B            00737         movwf   INTCON
0123   19AF            00738         btfsc   COMFLAGS,FTXDATA
0124   2923            00739         goto    $-1
0125   13AF            00740         bcf     COMFLAGS,FSLEEPMODE      ; wait to end wakeup sequence
0126                   00741 InitWakeupEnd
0126   0008            00742         return
                       00743
                       00744 ; #### CheckIdentifierByte ######################################################
                       00745 ;
                       00746 ;       this function is called by interrupt after the synch break and synch byte
                       00747 ;       is detected and the bitlength is calculated.
                       00748 ;       The identifier byte is read from bus and at this point the byte will be
                       00749 ;       checked if odd/even parity bis are correct, then extract the blocklength
                       00750 ;       and ID number and in dependence of the ID the handler mode will be set.
                       00751 ;
                       00752 ;
0127                   00753 DataBlockLengthTable                     ; decoder table for receive
0127   0782            00754         addwf   PCL,F                    ; byte length
0128   3403 3403 3405  00755         DT      3,3,5,9                  ; there is one byte more for the checksum
       3409
                       00756                                          ; byte is at the end of the data array
                       00757
012C                   00758 CheckIdentifierByte
012C   01AF            00759         clrf    COMFLAGS                 ; clear all flags
012D   0831            00760         movf    COMBUFFER,W              ; get identifier field
012E   3A80            00761         xorlw   0x80                     ; and test if there is a SLEEP command
012F   1903            00762         btfsc   STATUS,Z
0130   29A0            00763         goto    ActionBusSleep           ; yes : break the function
```

```
0131                  CheckEvenParityBit6                                ; count ones for even parity
0131   01A9                   clrf    BITNBR
0132   1831                   btfsc   COMBUFFER,0
0133   0AA9                   incf    BITNBR,F
0134   18B1                   btfsc   COMBUFFER,1
0135   0AA9                   incf    BITNBR,F
0136   1931                   btfsc   COMBUFFER,2
0137   0AA9                   incf    BITNBR,F
0138   1A31                   btfsc   COMBUFFER,4
0139   0AA9                   incf    BITNBR,F

013A   1B31                   btfsc   COMBUFFER,6                        ; check bit6 contains the even
013B   0AA9                   incf    BITNBR,F                           ; value
013C   1829                   btfsc   BITNBR,0                           ; if even then check odd parity bit
013D   294A                   goto    SetParityError
013E                  CheckOddParityBit7                                 ; count ones for odd parity
013E   01A9                   clrf    BITNBR                             ; and invert
013F   18B1                   btfsc   COMBUFFER,1
0140   0AA9                   incf    BITNBR,F
0141   19B1                   btfsc   COMBUFFER,3
0142   0AA9                   incf    BITNBR,F
0143   1A31                   btfsc   COMBUFFER,4
0144   0AA9                   incf    BITNBR,F
0145   1AB1                   btfsc   COMBUFFER,5
0146   0AA9                   incf    BITNBR,F

0147   1FB1                   btfss   COMBUFFER,7
0148   0AA9                   incf    BITNBR,F                           ; if odd then split off
0149   1829                   btfsc   BITNBR,0                           ; the blocklength
014A                  SetParityError
014A   14AE                   bsf     ERRORFLAGS,FIDPARITYERROR          ; set parity error bit
                                                                         ; if there is an error
014B                  GetPreID
014B   0831                   movf    COMBUFFER,W                        ; get the id number plus
014C   393F                   andlw   0x3F                               ; the blocklength subsets
014D   00AC                   movwf   PREIDNUMBER                        ; and save the value

014E                  DecodeBlockLength
014E   3001                   movlw   high DataBlockLengthTable          ; get address of decoder table
014F   008A                   movwf   PCLATH

0150   0E31                   swapf   COMBUFFER,W                        ; switches bit3,4 into bit0,1
0151   3903                   andlw   .3                                 ; kill the rest
0152   2127                   call    DataBlockLengthTable               ; and decode the result
0153   00AB                   movwf   DATABLOCKLENGTH                    ; store the used blocklength
                                                                         ; including the checksum byte
```



```
0154   01B2           clrf    DATACRC                       ; initialize the checksum
0155   01AE           clrf    ERRORFLAGS                    ; and error flag register

0156          DecodeIdNumber
0156   162F           bsf     COMFLAGS,FCOMDATA             ; set communication flag

0157   3001           movlw   HIGH DecodeIDTable            ; initial the high program
0158   008A           movwf   PCLATH                        ; counter with table adress

0159   0831           movf    COMBUFFER,W                   ; get the buffer value
015A   390F           andlw   .15
015B   00AD           movwf   IDNUMBER                      ; save ID number
015C   00B1           movwf   COMBUFFER                     ; generate the jump width
015D   0DB1           rlf     COMBUFFER,F                   ; by multiplying id with
015E   0D31           rlf     COMBUFFER,W                   ; four and do the jump
015F   0782           addwf   PCL,F                         ; by manipulating the PCL

              ; ---- DecodeIDTable ------------------------------------------------------------
              ;
              ;       Decode table using the ID numbers
              ;
              ;       This is the point where the user can decide how the slave will react on
              ;       the decoded identifier number.  Also he can decide where the received data
              ;       will be stored or where the send data come from.
              ;
              ;       Three modes are possible:
              ;               MacroRsMode (ptr)       ->  read data from bus and save
              ;                                           it in the provided buffer (ptr)
              ;               MacroTxMode (ptr)       ->  send data out on bus from the
              ;                                           provided buffer.
              ;               MacroLstMode            ->  only monitor the bus
              ;
              ;       In the monitoring mode the user has the possibility to use the value of
              ;       the PREIDNUMBER to handle 64 commands without any data messages.

0160          DecodeIDTable
                      MacroLstMode
0160   16AF       M   bsf     COMFLAGS,FLISTENONLY          ; indicates the receive mode
                  M                                         ; without saving the received data
0161   0000       M   nop                                   ; this is necessary to have
0162   0000       M   nop                                   ; a correct jump table length
0163   3400       M   retlw   0
                      MacroTxMode  ( BUTTONPRESSED1 ) ; ID1: Transmit the button press counts
0164   15AF       M   bsf     COMFLAGS,FTXDATA              ; indicates the transmission mode
0165   3033       M   movlw   ( BUTTONPRESSED1 )            ; get the wanted ram location
                  M                                         ; where the data will read from
0166   00B0       M   movwf   BUFFERPTR                     ; and initialize the pointer
```

```
0167   3400              M         retlw   0
                     00849         MacroLstMode
0168   16AF              M         bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                         M                                                 ; without saving the received data
0169   0000              M         nop                                     ; this is necessary to have
016A   0000              M         nop                                     ; a correct jump table length
016B   3400              M         retlw   0
                     00850         MacroLstMode
016C   16AF              M         bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                         M                                                 ; without saving the received data
016D   0000              M         nop                                     ; this is necessary to have
016E   0000              M         nop                                     ; a correct jump table length
016F   3400              M         retlw   0
                     00851         MacroRsMode  ( COMPLED1 )       ; ID4: Receive LED counters
0170   152F              M         bsf     COMFLAGS,FRSDATA                ; indicates the receive mode
0171   303B              M         movlw   ( COMPLED1 )                    ; get the wanted ram location
                         M                                                 ; where the data will stored
0172   00B0              M         movwf   BUFFERPTR                       ; and initialize the pointer
0173   3400              M         retlw   0
                     00852         MacroLstMode
0174   16AF              M         bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                         M                                                 ; without saving the received data
0175   0000              M         nop                                     ; this is necessary to have
0176   0000              M         nop                                     ; a correct jump table length
0177   3400              M         retlw   0
                     00853         MacroLstMode
0178   16AF              M         bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                         M                                                 ; without saving the received data
0179   0000              M         nop                                     ; this is necessary to have
017A   0000              M         nop                                     ; a correct jump table length
017B   3400              M         retlw   0
                     00854         MacroLstMode
017C   16AF              M         bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                         M                                                 ; without saving the received data
017D   0000              M         nop                                     ; this is necessary to have
017E   0000              M         nop                                     ; a correct jump table length
017F   3400              M         retlw   0
                     00855         MacroLstMode
0180   16AF              M         bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                         M                                                 ; without saving the received data
0181   0000              M         nop                                     ; this is necessary to have
0182   0000              M         nop                                     ; a correct jump table length
0183   3400              M         retlw   0
                     00856         MacroLstMode
0184   16AF              M         bsf     COMFLAGS,FLISTENONLY            ; indicates the receive mode
                         M                                                 ; without saving the received data
0185   0000              M         nop                                     ; this is necessary to have
```



```
0186   0000           M        nop                                  ; a correct jump table length
0187   3400           M        retlw   0
                  00857        MacroLstMode
0188   16AF           M        bsf     COMFLAGS,FLISTENONLY         ; indicates the receive mode
                      M                                             ; without saving the received data
0189   0000           M        nop                                  ; this is necessary to have
018A   0000           M        nop                                  ; a correct jump table length
018B   3400           M        retlw   0
                  00858        MacroLstMode
018C   16AF           M        bsf     COMFLAGS,FLISTENONLY         ; indicates the receive mode
                      M                                             ; without saving the received data
018D   0000           M        nop                                  ; this is necessary to have
018E   0000           M        nop                                  ; a correct jump table length
018F   3400           M        retlw   0
                  00859        MacroLstMode
0190   16AF           M        bsf     COMFLAGS,FLISTENONLY         ; indicates the receive mode
                      M                                             ; without saving the received data
0191   0000           M        nop                                  ; this is necessary to have
0192   0000           M        nop                                  ; a correct jump table length
0193   3400           M        retlw   0
                  00860        MacroLstMode
0194   16AF           M        bsf     COMFLAGS,FLISTENONLY         ; indicates the receive mode
                      M                                             ; without saving the received data
0195   0000           M        nop                                  ; this is necessary to have
0196   0000           M        nop                                  ; a correct jump table length
0197   3400           M        retlw   0
                  00861        MacroLstMode
0198   16AF           M        bsf     COMFLAGS,FLISTENONLY         ; indicates the receive mode
                      M                                             ; without saving the received data
0199   0000           M        nop                                  ; this is necessary to have
019A   0000           M        nop                                  ; a correct jump table length
019B   3400           M        retlw   0
                  00862        MacroLstMode
019C   16AF           M        bsf     COMFLAGS,FLISTENONLY         ; indicates the receive mode
                      M                                             ; without saving the received data
019D   0000           M        nop                                  ; this is necessary to have
019E   0000           M        nop                                  ; a correct jump table length
019F   3400           M        retlw   0
                  00863
                  00864
                  00865 ; ---- ActionBusSleep -----------------------------------------------------------
                  00866 ;
                  00867 ;      Arriving on this sequence the master will not longer talk to the
                  00868 ;      slaves so he gives the command to hangup the line.
                  00869 ;      After finishing the bus read the main task is able to activate
                  00870 ;      the wakeup sequence to give the master the directive to reactivate
                  00871 ;      the communication.
```

```
01A0                ActionBusSleep
01A0   17AF                 bsf     COMFLAGS,FSLEEPMODE
01A1   3003                 movlw   .3
01A2   00AB                 movwf   DATABLOCKLENGTH                 ; decomp length control bits
01A3   3080                 movlw   0x80
01A4   00AD                 movwf   IDNUMBER                        ; set identifier on sleep id
                            MacroRsMode ( RSDATAFIELD )             ; save the two received bytes
01A5   152F             M   bsf     COMFLAGS,FRSDATA                ; indicates the receive mode
01A6   303B             M   movlw   ( RSDATAFIELD )                 ; get the wanted ram location
                        M                                           ; where the data will stored
01A7   00B0             M   movwf   BUFFERPTR                       ; and initialize the pointer
01A8   3400             M   retlw   0

                    ; #### MAIN #####################################################################

01A9                Main

01A9   300A                 movlw   .10
01AA   00C5                 movwf   COMPERATOR                      ; after x counts set LED1&2 on
01AB   00C8                 movwf   DEBOUNCECOUNTER

01AC   01CA                 clrf    EDGEDETECT

01AD   1683                 bsf     STATUS,RP0
01AE   301F                 movlw   0x1F                            ; set PORTB7..5 as output
01AF   0086                 movwf   PORTB                           ; to manage LEDs
01B0   1283                 bcf     STATUS,RP0

01B1   2100                 call    InitLinSlave                    ; initialize linbus functions

01B2                MainLoop
01B2   1F2F                 btfss   COMFLAGS,FCOMPLETE              ; message frame complete ?
01B3   29B7                 goto    CheckLevel                      ; no -> do rest of main task

01B4   132F                 bcf     COMFLAGS,FCOMPLETE              ; clear flag
01B5   3080                 movlw   (1<<LED3)                       ; and toggle LED
01B6   0686                 xorwf   PORTB,F

01B7                CheckLevel
01B7   0845                 movf    COMPERATOR,W                    ; compare delimiter
01B8   023B                 subwf   COMPLED1,W                      ; with first value
01B9   1C03                 btfss   STATUS,C                        ; if not equal
01BA   29BE                 goto    CheckNextLevel                  ; check next value

01BB   01BB                 clrf    COMPLED1                        ; else manipulate value
01BC   3020                 movlw   (1<<LED1)                       ; and toggle LED1
```

```
01BD   0686          xorwf   PORTB,F

01BE               CheckNextLevel
01BE   0845          movf    COMPERATOR,W                  ; compare second value
01BF   023C          subwf   COMPLED2,W
01C0   1C03          btfss   STATUS,C
01C1   29C5          goto    TestButtons

01C2   01BC          clrf    COMPLED2                      ; and if equal
01C3   3040          movlw   (1<<LED2)                     ; toggle LED2
01C4   0686          xorwf   PORTB,F

01C5               TestButtons
01C5   0BC8          decfsz  DEBOUNCECOUNTER,F
01C6   29B2          goto    MainLoop                      ; do the loop


01C7   3006          movlw   (1<<BUTTON1) | (1<<BUTTON2)   ; maskout of the used
01C8   0506          andwf   PORTB,W                       ; portpins and save
01C9   00C9          movwf   COPYPORTB                     ; the result

01CA               CheckEdgeChange
01CA   064A          xorwf   EDGEDETECT,W                  ; check actual and last
01CB   0549          andwf   COPYPORTB,W                   ; value to detect any
01CC   1903          btfsc   STATUS,Z                      ; rising edge on input
01CD   29D7          goto    ButtomTestEnd                 ; nothing -> do the loop

01CE   00C8          movwf   DEBOUNCECOUNTER               ; save the pin who changed

01CF               RisingEdgeButtom1
01CF   3002          movlw   (1<<BUTTON1)                  ; mask of button1
01D0   0548          andwf   DEBOUNCECOUNTER,W             ; buttom1 pressed ?
01D1   1D03          btfss   STATUS,Z
01D2   0AB3          incf    BUTTONPRESSED1,F              ; yes : count the rising edge

01D3               RisingEdgeButtom2
01D3   3004          movlw   (1<<BUTTON2)                  ; mask of button2
01D4   0548          andwf   DEBOUNCECOUNTER,W             ; button 2 pressed ?
01D5   1D03          btfss   STATUS,Z
01D6   0AB4          incf    BUTTONPRESSED2,F              ; yes : count the rising edge

01D7               ButtomTestEnd
01D7   0849          movf    COPYPORTB,W                   ; save actual values to detect
01D8   00CA          movwf   EDGEDETECT                    ; the next rising edge

01D9   300A          movlw   .10                           ; reload the debounce
01DA   00C8          movwf   DEBOUNCECOUNTER
```

```
01DB   29B2           00961         goto    MainLoop                        ; and do the loop again
                      00962
                      00963         end
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

MPASM 02.30.09 Intermediate  LINSLAVE.ASM   1-27-2000  9:57:59         PAGE 25


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ---------------- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   473
Program Memory Words Free:  1575
```

NOTES:

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02