AN730

CRC Generating and Checking

Authors: Thomas Schmidt
Microchip Technology Inc.

INTRODUCTION

This application note describes the Cyclic Redundancy Check (CRC) theory and implementation. The CRC check is used to detect errors in a message. Two implementations are shown:

- Table driven CRC calculation
- Loop driven CRC calculation

This application describes the implementation of the CRC-16 polynomial. However, there are several formats for the implementation of CRC such as CRC-CCITT, CRC-32 or other polynomials.

CRC is a common method for detecting errors in transmitted messages or stored data. The CRC is a very powerful, but easily implemented technique to obtain data reliability.

THEORY OF OPERATION

The theory of a CRC calculation is straight forward. The data is treated by the CRC algorithm as a binary number. This number is divided by another binary number called the polynomial. The rest of the division is the CRC checksum, which is appended to the transmitted message. The receiver divides the message (including the calculated CRC), by the same polynomial the transmitter used. If the result of this division is zero, then the transmission was successful. However, if the result is not equal to zero, an error occurred during the transmission.

The CRC-16 polynomial is shown in Equation 1. The polynomial can be translated into a binary value, because the divisor is viewed as a polynomial with binary coefficients. For example, the CRC-16 polynomial translates to 1000000000000101b. All coefficients, like x^2 or x^{15}, are represented by a logical 1 in the binary value.

The division uses the Modulo-2 arithmetic. Modulo-2 calculation is simply realized by XOR'ing two numbers.

EXAMPLE 1: MODULO-2 CALCULATION

```
        1 0 0 1 1 0 0 1 0 1
XOR     0 1 0 0 1 1 0 1 1 1
=       1 1 0 1 0 1 0 0 1 0
```

XOR-Function

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

EQUATION 1: THE CRC-16 POLYNOMIAL

$$P(x) = x^{16}+x^{15}+x^{2}+1$$

Example Calculation

In this example calculation, the message is two bytes long. In general, the message can have any length in bytes. Before we can start calculating the CRC value 1, the message has to be augmented by n-bits, where n is the length of the polynomial. The CRC-16 polynomial has a length of 16-bits, therefore, 16-bits have to be augmented to the original message. In this example calculation, the polynomial has a length of 3-bits, therefore, the message has to be extended by three zeros at the end. An example calculation for a CRC is shown in Example 1. The reverse calculation is shown in Example 2.

EXAMPLE 2: CALCULATION FOR GENERATING A CRC

```
Message    = 110101
Polynomial = 101

1 1 0 1 0 1 0 0 ÷ 1 0 1 = 1 1 1  0 1
1 0 1                          ^
-----                          |
  1 1 1                        |
  1 0 1                        |
  -----        Quotient (has no function in CRC calculation)
    1 0 0
    1 0 1
    -----
        1 1 0
        1 0 1
        -----
          1 1 0
          1 0 1
          -----
            1 1   <-- Remainder = CRC checksum
            ===

Message with CRC  = 1 1 0 1 0 1 1 1
```

EXAMPLE 3: CHECKING A MESSAGE FOR A CRC ERROR

```
Message with CRC = 11010111
Polynomial = 101

1 1 0 1 0 1 1 1 ÷ 1 0 1 = 1 1 1  0 1
1 0 1                          ^
-----                          |
  1 1 1                        |
  1 0 1                        |
  -----                    Quotient
    1 0 0
    1 0 1
    -----
        1 1 1
        1 0 1
        -----
          1 0 1
          1 0 1
          -----
            0 0   <-- Checksum is zero, therefore, no transmission error
            ===
```

 Preliminary

FIGURE 1: HARDWARE CRC-16 GENERATOR

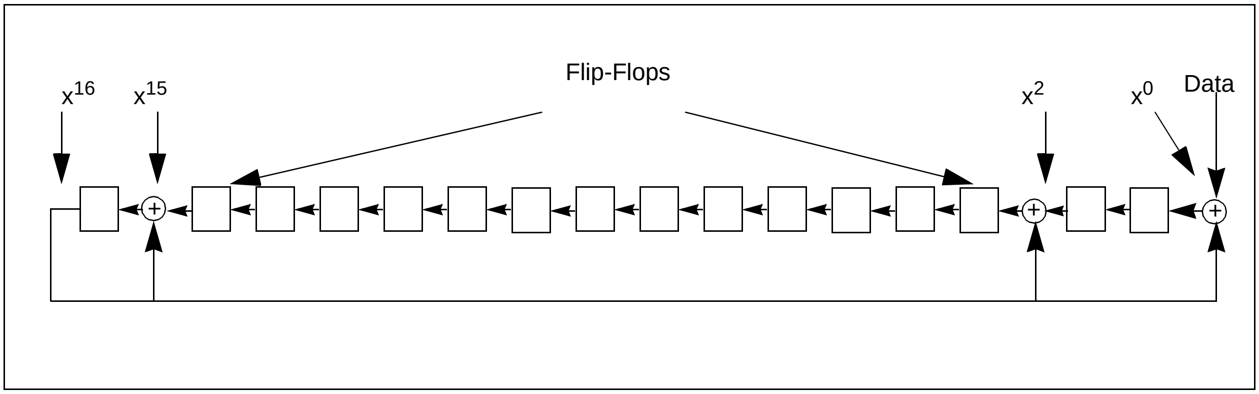
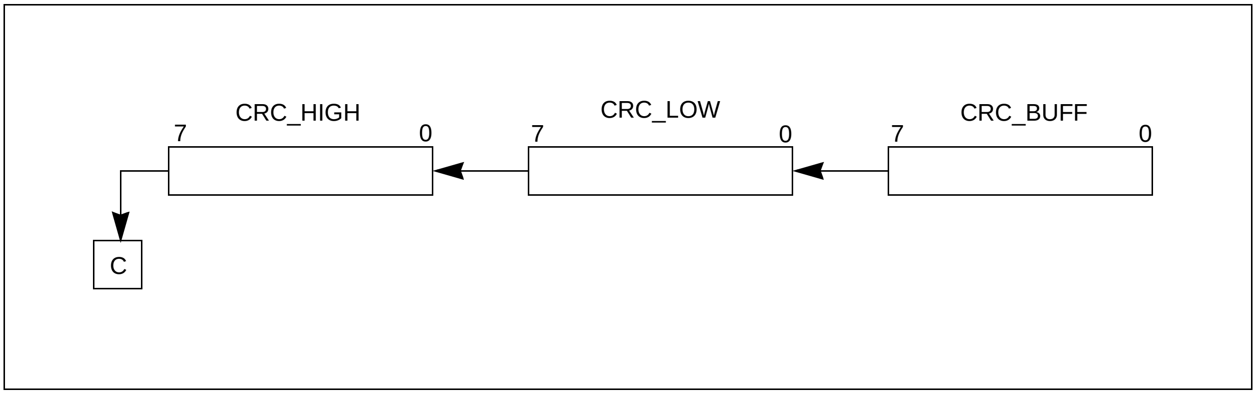


FIGURE 2: LOOP DRIVEN CRC IMPLEMENTATION



CRC Hardware Implementation

The CRC calculation is realized with a shift register and XOR gates. Figure 1 shows a CRC generator for the CRC-16 polynomial. Each bit of the data is shifted into the CRC shift register (Flip-Flops) after being XOR'ed with the CRC's most significant bit.

Software Implementations

There are two different techniques for implementing a CRC in software. One is a loop driven implementation and the other is a table driven implementation.

The loop driven implementation works like the calculation shown in Figure 2. The data is fed through a shift register. If a one pops out at the MSb, the content is XORed with the polynomial. In the other, the registers are shifted by one position to the left.

Another method to calculate a CRC is to use precalculated values and XOR them to the shift register.

Note:	The mathematical details are not given within this application note. The interested reader may refer to the material shown in the Reference section.

LOOP DRIVEN CRC IMPLEMENTATION

This section describes the loop driven CRC implementation. This implementation is derived from the hardware implementation shown in Figure 1. The program for the loop driven CRC generation and CRC checking is shown in Appendix A.

CRC Generation

The implementation of a loop driven CRC generation is shown in Figure 2. In the first step the registers, CRC_HIGH and CRC_LOW, are initialized with the first two bytes of data. CRC_BUFF is loaded with the third byte of data. After that, the MSb of CRC_BUFF is shifted into the LSb of CRC_LOW and the MSb of CRC_LOW is shifted into the LSb of CRC_HIGH. The MSb of CRC_HIGH, which is now stored in the Carry flag, is tested to see if it is set. If the bit is set, the registers, CRC_HIGH and CRC_LOW, will be XORed with the CRC-16 polynomial. If the bit is not set, the next bit from CRC_BUFF will be shifted into the LSb of CRC_LOW.

This process is repeated until all data from CRC_BUFF is shifted into CRC_LOW. After this, CRC_BUFF is loaded with the next data byte. Then all data bytes are processed, and 16 zeros are appended to the message. The registers, CRC_HIGH and CRC_LOW, contain the calculated CRC value. The message can have any length. In this application note, the CRC value for 8 data bytes is calculated.

CRC Checking

The CRC check uses the same technique as the CRC generation, with the only difference being that zeros are not appended to the message.

TABLE DRIVEN CRC IMPLEMENTATION

A table driven CRC routine uses a different technique than a loop driven CRC routine. The idea behind a table driven CRC implementation is that instead of calculating the CRC bit by bit, precomputed bytes are XORed to the data. The source code for the table driven implementation is given in Appendix B.

The advantage of the table driven implementation is that it is faster than the loop driven solution. The drawback is that it consumes more program memory because of the size of the look-up table.

CRC Generation

The CRC at the table driven implementation is generated by reading a precomputed value out of a table and XOR, the result with the low and high byte of the CRC shift registers.

In the first step, the registers, CRC_BUFF, CRC_HIGH and CRC_LOW, are initialized with the first three bytes of data. After that, the value in CRC_BUFF is used as an offset to get the value for the precomputed CRC value out of the look-up table. Since the CRC-16 is 16 bits long, the look-up table is split up into two separate look-up tables. One for the high byte of the CRC register and one for the low byte of the CRC register (see Figure 3). The result from the look-up table of the high byte is XORed to the content of the CRC_HIGH register. The result for the low byte is XORed to the content of CRC_LOW. The results are stored back in CRC_LOW.

The next step is that the content from CRC_HIGH is shifted into CRC_BUFF and the content from CRC_LOW is shifted into the register, CRC_HIGH. Then the register, CRC_LOW, is loaded with the new data byte.

This process repeats for all data bytes. At the end of the CRC generation, the message has to be appended by 16 zeros. The 16 zeros are treated like the data bytes.

After the calculation is done, the content of the registers, CRC_HIGH and CRC_LOW, are appended to the message.

CRC Checking

The CRC check uses the same technique as the CRC generation, with the difference being that zeros are not appended to the message.

COMPARISON

Table 1 shows a comparison between the loop driven implementation and the table driven implementation. For the calculation, 8 data bytes were used.

TABLE 1: CRC-16 COMPARISON TABLE

Implementation	CRC Generation (in cycles)	CRC Check (in cycles)	Program Memory Usage (words)	Data Bytes
Loop Driven	865	870	85	6
Table Driven	348	359	612	5

ADVANTAGES OF CRC VS. SIMPLE SUM TECHNIQUES

The CRC generation has many advantages over simple sum techniques or parity check. CRC error correction allows detection of:

1. single bit errors
2. double bit errors
3. bundled bit errors (bits next to each other)

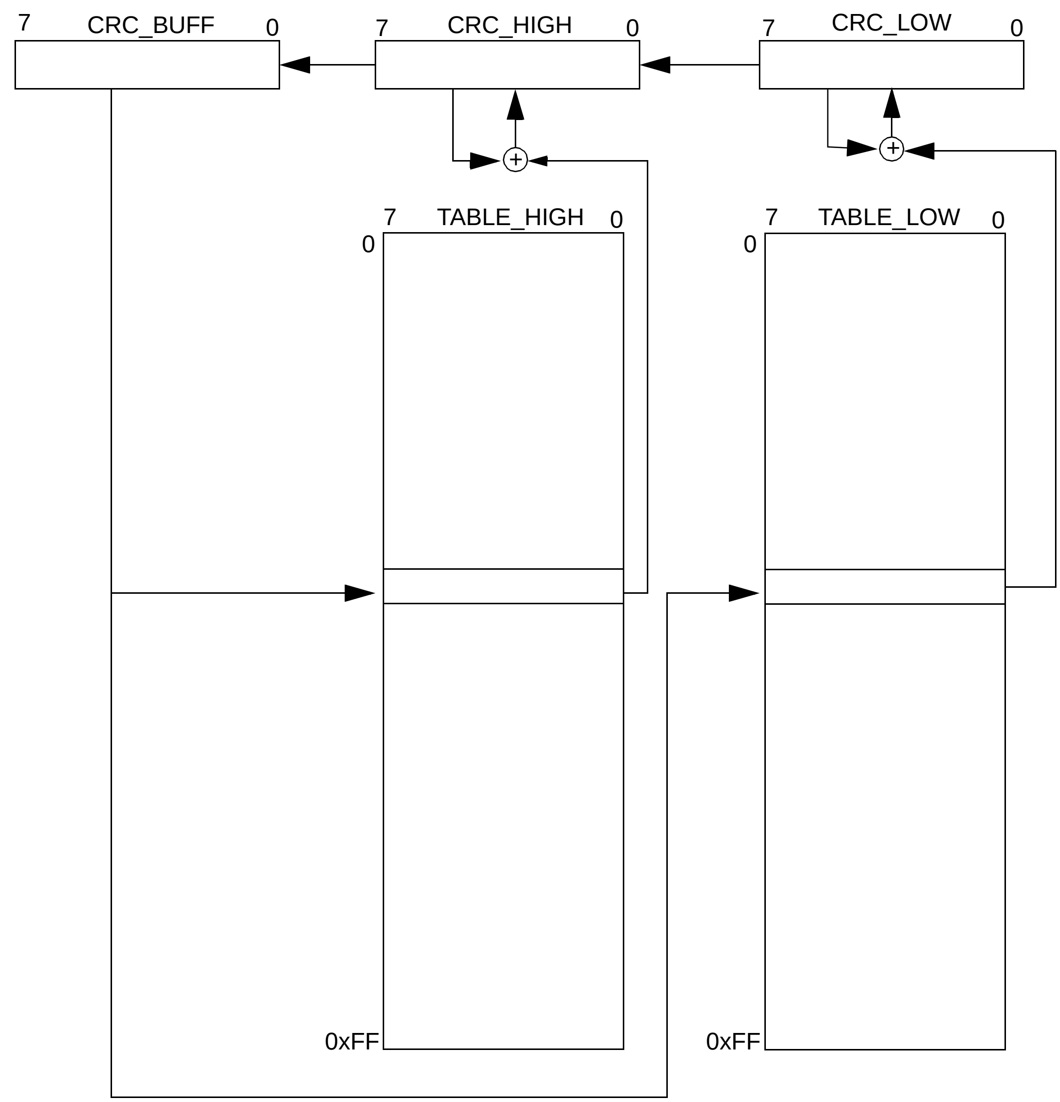
A parity bit check detects only single bit errors. The CRC error correction is mostly used where large data packages are transmitted, for example, in local area networks such as Ethernet.

References

Ross N. Williams - *A Painless Guide to CRC Error Detection Algorithms*

Donald E. Knuth - *The Art of Computer Programming,* Volume 2, Addisson Wesley

FIGURE 3: TABLE DRIVEN CRC CALCULATION IMPLEMENTATION



Preliminary

APPENDIX A: SOURCE CODE FOR LOOP DRIVEN CRC IMPLEMENATION

```
MPASM 02.30.11 Intermediate  CRC16_04.ASM   3-9-2000  13:00:00         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

; *******************************************************************************************
; * Title           : CRC16 calculation                                                     *
; * Author          : Thomas Schmidt                                                        *
; * Date            : 15.04.1999                                                            *
; * Revision        : 0.4                                                                   *
; * Last Modified   : 15.04.1999                                                            *
; * Core            : 12-bit, 14-bit (12-bit core tested)                                   *
; * Peripherals used: none                                                                  *
; * Registers used  :                                                                       *
; * Modifications   : crc16_01.asm Checksum check was added                                 *
; *                   crc16_03.asm Number of data bytes was increased from 2 to 8 bytes     *
; *                   crc16_04.asm added revers CRC
; * Description     :                                                                       *
; *                                                                                         *
; * This module calculates the checksum for the CRC16 polynom. The CRC16 polynome is defined    *
; * as x16+x15+x2+x0. The calculation is done by bitwise checking. The algorithm is designed    *
; * for a two byte wide message. The algorithm can easily be modified for longer messages. The  *
; * only thing what has to be done is to check after the low byte is shifted into the high byte *
; * that the low byte is loaded with a new data byte. The number of iteration has to be adjusted*
; * to the number of extra bytes in the data message. The number is calculated as follows:      *
; * n=16+x*messagebits. For example if the message is 4 bytes long the number of iterations is  *
; * n=16+16bits. The extra iterations have to be done because the message is extended with 16   *
```

```
                    00023 ; * zeros at the end of the message.                                                    *
                                                                                                              *
                    00024 ; *********************************************************************************************
                    00025
                    00026
                    00027                 LIST P=16C54B, r=hex
                    00028
                    00029                 #include “p16c5x.inc”
                    00001         LIST
                    00002 ; P16C5X.INC  Standard Header File, Version 4.00    Microchip Technology, Inc.
                    00313         LIST
                    00030
                    00031                 #define PolynomLow      b’00000101’     ; low byte of polynome
                    00032                 #define PolynomHigh     b’10000000’     ; high byte of polynome
                    00033                 #define PolynomLength   0x10            ; 16-bit polynome length
                    00034                 #define DataLength      0x09            ; Data length in Bits
                    00035                 #define Iterations      0x08            ; number of iterations for CRC calculation
                    00036
                    00037                 cblock  0x07
  00000007          00038                         CRC_HIGH                ; CRC shift registers
  00000008          00039                         CRC_LOW                 ; second CRC shift register
  00000009          00040                         CRC_BUFF                ; CRC buffer register
  0000000A          00041                         BITS                    ; number of data bits
  0000000B          00042                         DATABYTES               ; number of bytes of data
  0000000C          00043                         TEMP                    ; temporary register
                    00044                 endc
                    00045
01FF                00046                 ORG 0x1ff
01FF   0A00         00047                 goto    Begin
                    00048
                    00049
                    00050
0000                00051                 ORG 0x00
0000   0C10         00052 Begin           movlw   0x10                    ; Data for CRC generation
0001   0024         00053                 movwf   FSR                     ; Set point to begin of data block
0002   0CAA         00054                 movlw   0xAA                    ; data
                    00055
                    00056
                    00057                 ; initialization what has to be done before CRC16 routine can be
                    00058                 ; called. The FSR register contains the pointer to the first byte of
                    00059                 ; data and the register DATABYTES contains the number of data bytes
                    00060                 ; of the message.
0003   0C10         00061                 movlw   0x10                    ; set pointer to first data location
0004   0024         00062                 movwf   FSR                     ; initialize FSR register
                    00063
0005   0910         00064 Main            call    CRC16Generate           ; calculate CRC16 value
                    00065
```

```
                       ; append CRC to message
0006  02A4             incf    FSR, f             ; point to position behind last data byte
0007  0207             movf    CRC_HIGH, w        ; copy CRC_HIGH data into w-register
0008  0020             movwf   INDF               ; copy CRC_HIGH behing last data byte
0009  02A4             incf    FSR, f             ; point to next location
000A  0208             movf    CRC_LOW, w         ; copy CRC_LOW data into w-register
000B  0020             movwf   INDF               ; copy data into next location
000C  0C10             movlw   0x10               ; set pointer to first data location
000D  0024             movwf   FSR                ; initialize FSR register
000E  0924             call    CRC16Restore       ; restore CRC16 value

000F  0A0F  Stop       goto    Stop               ; do forever

                       ; ***********************************************************************
                       ; * Title:              CRC16 calculation                                *
                       ; * Input parameter:    Pointer to first data byte (pointer in FSR register)  *
                       ; *                     Number of data bytes stored in register DATABYTES     *
                       ; * Output:             CRC result stored in CRC_HIGH and CRC_LOW              *
                       ; ***********************************************************************
0010  0938  CRC16Generate  call    CRC16Init      ; initialize registers
0011  0C03             movlw   0x03               ; initialize TEMP register with 2
0012  002C             movwf   TEMP               ; move 0x02 into TEMP

0013  0947  NextCRC16  call    CRC16Engine        ; Calculate CRC16 for one byte
0014  02EB             decfsz  DATABYTES, f       ; Decrement the number of data bytes by one
0015  0A1E             goto    Reload             ; reload CRC_BUFF register with new data byte

0016  02EC             decfsz  TEMP, f            ; decrement TEMP register
0017  0A19             goto    AppendZeros        ; Append zeros to message
0018  0800             retlw   0x00               ; return to main
0019  0069  AppendZeros  clrf  CRC_BUFF           ; append data with zeros
001A  0C08             movlw   Iterations         ; initialize BITS register
001B  002A             movwf   BITS               ; with 0x08
001C  02AB             incf    DATABYTES, f       ; increment data bytes
001D  0A13             goto    NextCRC16          ; last iteration


                       ; Reload CRC buffer register with new data word.
001E  02A4  Reload     incf    FSR, f             ; point to next data byte
001F  0200             movf    INDF, w            ; copy data into w-register
0020  0029             movwf   CRC_BUFF           ; move data into CRC_BUFF register
0021  0C08             movlw   Iterations         ; initialize register BITS with 8
0022  002A             movwf   BITS               ; move 8 into register BITS
0023  0A13             goto    NextCRC16          ; calculate next CRC


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

```
                        ; * Titel: Restore CRC function                                            *
                        ; * Input: Pointer to first data byte in FSR register                      *
                        ; * Output: w=0 CRC was restore sucessfull                                 *
                        ; *         w=1 CRC was not restored sucessfull                            *
                        ; ****************************************************************************
0024  0938  CRC16Restore    call    CRC16Init         ; initialize CRC registers
0025  0C02                  movlw   0x02              ; add offset to DATABYTES
0026  01EB                  addwf   DATABYTES, f      ; add offset to register DATABYTES

0027  0947  NextCRCRestore  call    CRC16Engine
0028  02EB                  decfsz  DATABYTES, f      ; Decrement the number of data bytes by one
0029  0A32                  goto    ReloadRestore     ; reload CRC_BUFF register with new data byte

                        ; check if CRC_HIGH and CRC_LOW equal to zero
002A  0227                  movf    CRC_HIGH, f       ; copy CRC_HIGH onto itself
002B  0743                  btfss   STATUS, Z         ; is content zero?
002C  0A31                  goto    CRCError          ; no, CRC error occured
002D  0228                  movf    CRC_LOW, f        ; copy CRC_LOW register onto itself
002E  0743                  btfss   STATUS, Z         ; is content zero?
002F  0A31                  goto    CRCError          ; no, CRC error occured
0030  0800                  retlw   0x00              ; return to main (0= no error)

0031  0801  CRCError        retlw   0x01              ; return to main with error code 1


                        ; Reload CRC buffer register with new data word.
0032  02A4  ReloadRestore   incf    FSR, f            ; point to next data byte
0033  0200                  movf    INDF, w           ; copy data into w-register
0034  0029                  movwf   CRC_BUFF          ; move data into CRC_BUFF register
0035  0C08                  movlw   Iterations        ; initialize register BITS with 8
0036  002A                  movwf   BITS              ; move 8 into register BITS
0037  0A27                  goto    NextCRCRestore    ; calculate next CRC


                        ; ****************************************************************************
                        ; * Titel: CRC16 Initialization                                            *
                        ; * Input: Pointer to first data byte in FSR register                      *
                        ; * Output: none                                                           *
                        ; ****************************************************************************
0038  0200  CRC16Init       movf    INDF, w           ; copy data into W-register
0039  0027                  movwf   CRC_HIGH          ; copy w-register into CRC_HIGH register
003A  02A4                  incf    FSR, f            ; set pointer to next location
003B  0200                  movf    INDF, w           ; copy data into w-register

003C  0028                  movwf   CRC_LOW           ; copy w-register into CRC_LOW
003D  02A4                  incf    FSR, f            ; set pointer to next location
003E  0200                  movf    INDF, w           ; copy next data into w-register
```

```
003F   0029                     movwf   CRC_BUFF              ; copy data into CRC buffer register
0040   0C08                     movlw   Iterations            ; initialize register BITs with
0041   002A                     movwf   BITS                  ; length of polynome for iteration
0042   0C09                     movlw   DataLength            ; copy number of data bytes
0043   002B                     movwf   DATABYTES             ; into register DataBytes
0044   0C03                     movlw   0x03                  ; decrement three from the number
0045   00AB                     subwf   DATABYTES, f          ; of data bytes, because three register
                                                              ; are now initialized
0046   0800                     retlw   0x00                  ; return from subroutine



                                ; ********************************************************************************
                                ; * Titel: CRC16 Engine                                                           *
                                ; * Input: Pointer to first data byte in FSR register                             *
                                ; * Output: none                                                                  *
                                ; ********************************************************************************
0047   0403       CRC16Engine   bcf     STATUS, C             ; clear carry flag
0048   0369                     rlf     CRC_BUFF, f           ; shift bit7 of CRC_BUFF into carry flag
0049   0368                     rlf     CRC_LOW, f            ; shift bit7 of CRC_LOW into carry flag
                                                              ; and shift 0 into bit7 of CRC_LOW
004A   0367                     rlf     CRC_HIGH, f           ; rotate carry flag into bit0 of CRC_HIGH
                                                              ; and rotate bit7 of CRC_HIGH into carry
                                                              ; flag
004B   0703                     btfss   STATUS, C             ; is carry flag set?
004C   0A51                     goto    NextBitEngine         ; carry flag is clear there next rotation
004D   0C80                     movlw   PolynomHigh           ; carry flag is set therefore XOR CRCSHIFT
                                                              ; registers
004E   01A7                     xorwf   CRC_HIGH, f           ; XOR CRC_HIGH register
004F   0C05                     movlw   PolynomLow            ; load w-register with low byte of polynom
0050   01A8                     xorwf   CRC_LOW, f            ; XOR CRC_LOW register
0051   02EA     NextBitEngine   decfsz  BITS, f               ; do for all bits
0052   0A47                     goto    CRC16Engine           ; calculate CRC for next bit
0053   0800                     retlw   0x00                  ; return from Subroutine

                                END

Program Memory Words Used:    85
Program Memory Words Free:   427


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

APPENDIX B: SOURCE CODE TABLE DRIVEN CRC IMPLEMENTATION

```
MPASM 02.30.11 Intermediate  CRCTAB01.ASM   3-9-2000  13:02:59         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                      00001 ; *********************************************************************************************
                      00002 ; * Title           : CRC16 calculation table driven implementation                           *
                      00003 ; * Author          : Thomas Schmidt                                                          *
                      00004 ; * Date            : 22.03.1999                                                              *
                      00005 ; * Revision        : 0.1                                                                     *
                      00006 ; * Last Modified   : 15.04.1999                                                              *
                      00007 ; * Core            : 12-bit, 14-bit (12-bit core tested)                                     *
                      00008 ; * Peripherals used: none                                                                    *
                      00009 ; * Registers used  :                                                                         *
                      00010 ; * Modifications   : crctab01.asm: first program CRC generation                              *
                      00011 ; * Description     :                                                                         *
                      00012 ; *                                                                                           *
                      00013 ; * This module calculates the checksum for the CRC16 polynom. The CRC16 polynome is defined    *
                      00014 ; * as x16+x15+x2+x0. The calculation is done by bitwise checking. The algorithm is designed    *
                      00015 ; * for a two byte wide message. The algorithm can easily be modified for longer messages. The  *
                      00016 ; * only thing what has to be done is to check after the low byte is shifted into the high byte *
                      00017 ; * that the low byte is loaded with a new data byte. The number of iteration has to be adjusted*
                      00018 ; * to the number of extra bytes in the data message. The number is calculated as follows:      *
                      00019 ; * n=16+x*messagebits. For example if the message is 4 bytes long the number of iterations is  *
                      00020 ; * n=16+16bits. The extra iterations have to be done because the message is extended with 16   *
                      00021 ; * zeros at the end of the message.                                                          *
                                                                                                                      *
                      00022 ; *********************************************************************************************
                      00023
                      00024                 LIST P=16C58B, r=hex
                      00025
                      00026                 #include "p16c5x.inc"
                      00001         LIST
                      00002 ; P16C5X.INC  Standard Header File, Version 4.00    Microchip Technology, Inc.
                      00313         LIST
                      00027
                      00028                 #define DataLength              0x09            ; length of data field
                      00029                 #define LastTableElementHigh    0x2             ; last table element of high byte
                      00030                 #define LastTableElementLow     0x2             ; last table element of low byte
                      00031
                      00032                 cblock 0x07
  00000007            00033                         CRC_LOW         ; low byte of CRC register
  00000008            00034                         CRC_HIGH        ; high byte of CRC register
```

```
00000009                       CRC_BUFF        ; CRC buffer register
0000000A                       DATABYTES       ; contains number of data bytes
0000000B                       TEMP            ; temporary register
                       endc

07FF                   org 0x7ff
07FF   0A00            goto    Begin

0000                   org 0x00
0000        Begin
                       ; initialization what has to be done before CRC16 routine can be
                       ; called. The FSR register contains the pointer to the first byte of
                       ; data and the register DATABYTES contains the number of data bytes
                       ; of the message.
0000   0C10            movlw   0x10                    ; set pointer to first data location
0001   0024            movwf   FSR                     ; initialize FSR register

0002   090C Main       call    CRC16Generate           ; calculate CRC16 value

0003   0208            movf    CRC_HIGH, w             ; copy CRC_HIGH data into w-register
0004   0020            movwf   INDF                    ; copy CRC_HIGH behing last data byte
0005   02A4            incf    FSR, f                  ; point to next location
0006   0207            movf    CRC_LOW, w              ; copy CRC_LOW data into w-register
0007   0020            movwf   INDF                    ; copy data into next location
0008   0C10            movlw   0x10                    ; set pointer to first data location
0009   0024            movwf   FSR                     ; initialize FSR register
000A   093A TestPoint  call    CRC16Restore            ; Restore CRC
000B   0A0B Stop       goto    Stop                    ; do forever


                       ; ***********************************************************************
                       ; * Title:             CRC16 Table driven calculation                   *
                       ; * Input parameter:   Pointer to first data byte (pointer in FSR register)   *
                       ; *                    Number of data bytes stored in register DATABYTES      *
                       ; * Output:            CRC result stored in CRC_HIGH and CRC_LOW              *
                       ; ***********************************************************************
000C   095E CRC16Generate call CRC16Init               ; initialize CRC registers
000D   0C03            movlw   0x03                    ; initialize TEMP register
000E   002B            movwf   TEMP                    ; with 0x02

                       ; check if last CRC_BUFF points to last element in table. These elements
                       ; cannot be read from the look up table, because they are beyond the
                       ; program memory page.
000F   0209 NextValueGen movf  CRC_BUFF, w             ; load CRC_BUFF into w-register
0010   0FFF            xorlw   0xff                    ; check if content equals to 0xff
0011   0743            btfss   STATUS, Z               ; is result from XOR = 0?
0012   0A18            goto    CalculateCRC            ; no, calculate CRC
```

```
0013   0C02                  movlw   LastTableElementHigh   ; yes, get last table element for high byte
0014   01A8                  xorwf   CRC_HIGH, f            ; XOR with high byte
0015   0C02                  movlw   LastTableElementLow    ; get last table element for low byte
0016   01A7                  xorwf   CRC_LOW, f             ; XOR with low byte
0017   0A22                  goto    DecDATABYTES           ; goto end of loop

0018   0209   CalculateCRC   movf    CRC_BUFF, w            ; copy high byte of CRC into w-register
0019   04C3                  bcf     STATUS, PA1            ; select page 1
001A   05A3                  bsf     STATUS, PA0            ; select page 1
001B   0900                  call    CRC16TableHigh         ; get value for high byte
001C   01A8                  xorwf   CRC_HIGH,f             ; XOR table element with high byte
001D   0209                  movf    CRC_BUFF, w            ; get value for low byte
001E   05C3                  bsf     STATUS, PA1            ; select page 2
001F   04A3                  bcf     STATUS, PA0            ; select page 2
0020   0900                  call    CRC16TableLow          ; get value out of table
0021   01A7                  xorwf   CRC_LOW,f              ; XOR with low byte
0022   04A3   DecDATABYTES   bcf     STATUS, PA0            ; select page 0
0023   04C3                  bcf     STATUS, PA1            ; select page 0
0024   02EA                  decfsz  DATABYTES, f           ; decrement data bytes
0025   0A30                  goto    ReloadGen              ; reload values

0026   02EB                  decfsz  TEMP, f                ; append two bytes with zeros
0027   0A29                  goto    AppendZeros            ; append zeros to message (do twice)
0028   0800                  retlw   0x00                   ; return to main
0029   0208   AppendZeros    movf    CRC_HIGH, w            ; copy high byte into w-register
002A   0029                  movwf   CRC_BUFF               ; and from there to CRC_BUFF
002B   0207                  movf    CRC_LOW,w              ; Copy low byte into w-register
002C   0028                  movwf   CRC_HIGH               ; and from there into CRC_HIGH
002D   0067                  clrf    CRC_LOW                ; and from there into CRC_LOW
002E   02AA                  incf    DATABYTES, f           ; increment for additonal iteration
002F   0A0F                  goto    NextValueGen           ; calculate CRC for next byte

0030   0932   ReloadGen      call    Reload                 ; reload registers
0031   0A0F                  goto    NextValueGen           ; calculate next CRC value

                             ; *******************************************************************************
                             ; * Titel: Reload CRC_HIGH, CRC_LOW and CRC_BUFF register                       *
                             ; * Input: Pointer to next data byte in FSR register                            *
                             ; * Output:                                                                     *
                             ; *******************************************************************************
0032   0208   Reload         movf    CRC_HIGH, w            ; copy high byte into w-register
0033   0029                  movwf   CRC_BUFF               ; and from there to CRC_BUFF
0034   0207                  movf    CRC_LOW,w              ; Copy low byte into w-register
0035   0028                  movwf   CRC_HIGH               ; and from there into CRC_HIGH
0036   0200                  movf    INDF, w                ; copy next data into w-register
0037   0027                  movwf   CRC_LOW                ; and from there into CRC_LOW
0038   02A4                  incf    FSR, f                 ; point to next data byte
```

```
0039   0800                       retlw   0x00                   ; calculate CRC for next byte


                                  ; ************************************************************************
                                  ; * Titel: Restore CRC function                                          *
                                  ; * Input: Pointer to first data byte in FSR register                   *
                                  ; * Output: w=0 CRC was restore sucessfull                              *
                                  ; *         w=1 CRC was not restored sucessfull                         *
                                  ; ************************************************************************
003A   095E      CRC16Restore     call    CRC16Init              ; initialize CRC registers
003B   0C02                       movlw   0x02                   ; add two onto
003C   01EA                       addwf   DATABYTES, f           ; register DATABYTES

                                  ; check if last CRC_BUFF points to last element in table. These elements
                                  ; cannot be read from the look up table, because they are beyond the
                                  ; program memory page.
003D   0209      NextValueRes     movf    CRC_BUFF, w            ; load CRC_BUFF into w-register
003E   0FFF                       xorlw   0xff                   ; check if content equals to 0xff
003F   0743                       btfss   STATUS, Z              ; is result from XOR = 0?
0040   0A46                       goto    CalculateCRCRes        ; no, calculate CRC
0041   0C02                       movlw   LastTableElementHigh   ; yes, get last table element for high byte
0042   01A8                       xorwf   CRC_HIGH, f            ; XOR with high byte
0043   0C02                       movlw   LastTableElementLow    ; get last table element for low byte
0044   01A7                       xorwf   CRC_LOW, f             ; XOR with low byte
0045   0A50                       goto    DecDATABYTESRes        ; goto end of loop

0046   0209      CalculateCRCRes  movf    CRC_BUFF, w            ; copy high byte of CRC into w-register
0047   04C3                       bcf     STATUS, PA1            ; select page 1
0048   05A3                       bsf     STATUS, PA0            ; select page 1
0049   0900                       call    CRC16TableHigh         ; get value for high byte
004A   01A8                       xorwf   CRC_HIGH,f             ; XOR table element with high byte
004B   0209                       movf    CRC_BUFF, w            ; get value for low byte
004C   05C3                       bsf     STATUS, PA1            ; select page 2
004D   04A3                       bcf     STATUS, PA0            ; select page 2
004E   0900                       call    CRC16TableLow          ; get value out of table
004F   01A7                       xorwf   CRC_LOW,f              ; XOR with low byte
0050   04A3      DecDATABYTESRes  bcf     STATUS, PA0            ; select page 0
0051   04C3                       bcf     STATUS, PA1            ; select page 0
0052   02EA                       decfsz  DATABYTES, f           ; decrement data bytes
0053   0A5C                       goto    ReloadRes              ; calculate next CRC16 value

                                  ; check if CRC_HIGH and CRC_LOW equal to zero
0054   0228                       movf    CRC_HIGH, f            ; copy CRC_HIGH onto itself
0055   0743                       btfss   STATUS, Z              ; is content zero?
0056   0A5B                       goto    CRCError               ; no, CRC error occured
0057   0227                       movf    CRC_LOW, f             ; copy CRC_LOW register onto itself
0058   0743                       btfss   STATUS, Z              ; is content zero?
```

```
0059   0A5B                          goto    CRCError              ; no, CRC error occured
005A   0800                          retlw   0x00                  ; return to main (0= no error)

005B   0801        CRCError          retlw   0x01                  ; return to main with error code 1

005C   0932        ReloadRes         call    Reload                ; reload register
005D   0A3D                          goto    NextValueRes          ; calculate next value


                                     ; ***********************************************************************
                                     ; * Titel: CRC16 Initialization                                         *
                                     ; * Input: Pointer to first data byte in FSR register                   *
                                     ; * Output: none                                                        *
                                     ; ***********************************************************************
005E   0200        CRC16Init         movf    INDF, w               ; copy data into W-register
005F   0029                          movwf   CRC_BUFF              ; copy w-register into CRC_BUFF register
0060   02A4                          incf    FSR, f                ; set pointer to next location
0061   0200                          movf    INDF, w               ; copy data into W-register
0062   0028                          movwf   CRC_HIGH              ; copy w-register into CRC_HIGH register
0063   02A4                          incf    FSR, f                ; set pointer to next location
0064   0200                          movf    INDF, w               ; copy data into w-register
0065   0027                          movwf   CRC_LOW               ; copy w-register into CRC_LOW
0066   02A4                          incf    FSR, f                ; point to next location
0067   0C09                          movlw   DataLength            ; copy number of data bytes
0068   002A                          movwf   DATABYTES             ; into register DataBytes
0069   0C03                          movlw   0x03                  ; decrement three from the number
006A   00AA                          subwf   DATABYTES, f          ; of data bytes, because three register
                                                                   ; are now initialized
006B   0800                          retlw   0x00                  ; return from subroutine

                                     ; ***********************************************************************
                                     ; * Titel: CRC16 Table for High byte                                    *
                                     ; * Input: Pointer to table element in w-register                       *
                                     ; * Output: look-up value in w-register                                 *
                                     ; ***********************************************************************
0200                                 org 0x200
0200   01E2        CRC16TableHigh    addwf   PCL, f       ; add to low byte of PC
0201   0800 0880 0880                 dt     0,      0x80,   0x80,   0
       0800
0205   0880 0800 0800                 dt     0x80,   0,      0,      0x80
       0880
0209   0880 0800 0800                 dt     0x80,   0,      0,      0x80
       0880
020D   0800 0880 0880                 dt     0,      0x80,   0x80,   0
       0800
0211   0880 0800 0800                 dt     0x80,   0,      0,      0x80
       0880
```

```
0215   0800 0880 0880 00218                 dt     0,      0x80,   0x80,   0
       0800
0219   0800 0880 0880 00219                 dt     0,      0x80,   0x80,   0
       0800
021D   0880 0800 0800 00220                 dt     0x80,   0,      0,      0x80
       0880
0221   0880 0800 0800 00221                 dt     0x80,   0,      0,      0x80
       0880
0225   0800 0880 0880 00222                 dt     0,      0x80,   0x80,   0
       0800
0229   0800 0880 0880 00223                 dt     0,      0x80,   0x80,   0
       0800
022D   0880 0800 0800 00224                 dt     0x80,   0,      0,      0x80
       0880
0231   0800 0880 0880 00225                 dt     0,      0x80,   0x80,   0
       0800
0235   0880 0800 0800 00226                 dt     0x80,   0,      0,      0x80
       0880
0239   0880 0800 0800 00227                 dt     0x80,   0,      0,      0x80
       0880
023D   0800 0880 0880 00228                 dt     0,      0x80,   0x80,   0
       0800
0241   0881 0801 0801 00229                 dt     0x81,   0x1,    0x1,    0x81
       0881
0245   0801 0881 0881 00230                 dt     0x1,    0x81,   0x81,   0x1
       0801
0249   0801 0881 0881 00231                 dt     0x1,    0x81,   0x81,   0x1
       0801
024D   0881 0801 0801 00232                 dt     0x81,   0x1,    0x1,    0x81
       0881
0251   0801 0881 0881 00233                 dt     0x1,    0x81,   0x81,   0x1
       0801
0255   0881 0801 0801 00234                 dt     0x81,   0x1,    0x1,    0x81
       0881
0259   0881 0801 0801 00235                 dt     0x81,   0x1,    0x1,    0x81
       0881
025D   0801 0881 0881 00236                 dt     0x1,    0x81,   0x81,   0x1
       0801
0261   0801 0881 0881 00237                 dt     0x1,    0x81,   0x81,   0x1
       0801
0265   0881 0801 0801 00238                 dt     0x81,   0x1,    0x1,    0x81
       0881
0269   0881 0801 0801 00239                 dt     0x81,   0x1,    0x1,    0x81
       0881
026D   0801 0881 0881 00240                 dt     0x1,    0x81,   0x81,   0x1
       0801
0271   0881 0801 0801 00241                 dt     0x81,   0x1,    0x1,    0x81
```

```
        0881
0275    0801 0881 0881 00242                    dt     0x1,    0x81,   0x81,   0x1
        0801
0279    0801 0881 0881 00243                    dt     0x1,    0x81,   0x81,   0x1
        0801
027D    0881 0801 0801 00244                    dt     0x81,   0x1,    0x1,    0x81
        0881
0281    0883 0803 0803 00245                    dt     0x83,   0x3,    0x3,    0x83
        0883
0285    0803 0883 0883 00246                    dt     0x3,    0x83,   0x83,   0x3
        0803
0289    0803 0883 0883 00247                    dt     0x3,    0x83,   0x83,   0x3
        0803
028D    0883 0803 0803 00248                    dt     0x83,   0x3,    0x3,    0x83
        0883
0291    0803 0883 0883 00249                    dt     0x3,    0x83,   0x83,   0x3
        0803
0295    0883 0803 0803 00250                    dt     0x83,   0x3,    0x3,    0x83
        0883
0299    0883 0803 0803 00251                    dt     0x83,   0x3,    0x3,    0x83
        0883
029D    0803 0883 0883 00252                    dt     0x3,    0x83,   0x83,   0x3
        0803
02A1    0803 0883 0883 00253                    dt     0x3,    0x83,   0x83,   0x3
        0803
02A5    0883 0803 0803 00254                    dt     0x83,   0x3,    0x3,    0x83
        0883
02A9    0883 0803 0803 00255                    dt     0x83,   0x3,    0x3,    0x83
        0883
02AD    0803 0883 0883 00256                    dt     0x3,    0x83,   0x83,   0x3
        0803
02B1    0883 0803 0803 00257                    dt     0x83,   0x3,    0x3,    0x83
        0883
02B5    0803 0883 0883 00258                    dt     0x3,    0x83,   0x83,   0x3
        0803
02B9    0803 0883 0883 00259                    dt     0x3,    0x83,   0x83,   0x3
        0803
02BD    0883 0803 0803 00260                    dt     0x83,   0x3,    0x3,    0x83
        0883
02C1    0802 0882 0882 00261                    dt     0x2,    0x82,   0x82,   0x2
        0802
02C5    0882 0802 0802 00262                    dt     0x82,   0x2,    0x2,    0x82
        0882
02C9    0882 0802 0802 00263                    dt     0x82,   0x2,    0x2,    0x82
        0882
02CD    0802 0882 0882 00264                    dt     0x2,    0x82,   0x82,   0x2
        0802
```

```
02D1   0882 0802 0802 00265                      dt     0x82,   0x2,    0x2,    0x82
       0882
02D5   0802 0882 0882 00266                      dt     0x2,    0x82,   0x82,   0x2
       0802
02D9   0802 0882 0882 00267                      dt     0x2,    0x82,   0x82,   0x2
       0802
02DD   0882 0802 0802 00268                      dt     0x82,   0x2,    0x2,    0x82
       0882
02E1   0882 0802 0802 00269                      dt     0x82,   0x2,    0x2,    0x82
       0882
02E5   0802 0882 0882 00270                      dt     0x2,    0x82,   0x82,   0x2
       0802
02E9   0802 0882 0882 00271                      dt     0x2,    0x82,   0x82,   0x2
       0802
02ED   0882 0802 0802 00272                      dt     0x82,   0x2,    0x2,    0x82
       0882
02F1   0802 0882 0882 00273                      dt     0x2,    0x82,   0x82,   0x2
       0802
02F5   0882 0802 0802 00274                      dt     0x82,   0x2,    0x2,    0x82
       0882
02F9   0882 0802 0802 00275                      dt     0x82,   0x2,    0x2,    0x82
       0882
02FD   0802 0882 0882 00276                      dt     0x2,    0x82,   0x82
                      00277
                      00278                     ; *******************************************************************************
                      00279                     ; * Titel: CRC16 Table for low byte                                             *
                      00280                     ; * Input: Pointer to table element in w-register                               *
                      00281                     ; * Output: look-up value in w-register                                         *
                      00282                     ; *******************************************************************************
0400                  00283                     org 0x400
0400   01E2           00284 CRC16TableLow       addwf   PCL, f          ; add to low byte of PC
0401   0800 0805 080F 00285                      dt     0,      0x5,    0xf,    0xa
       080A
0405   081B 081E 0814 00286                      dt     0x1b,   0x1e,   0x14,   0x11
       0811
0409   0833 0836 083C 00287                      dt     0x33,   0x36,   0x3c,   0x39
       0839
040D   0828 082D 0827 00288                      dt     0x28,   0x2d,   0x27,   0x22
       0822
0411   0863 0866 086C 00289                      dt     0x63,   0x66,   0x6c,   0x69
       0869
0415   0878 087D 0877 00290                      dt     0x78,   0x7d,   0x77,   0x72
       0872
0419   0850 0855 085F 00291                      dt     0x50,   0x55,   0x5f,   0x5a
       085A
041D   084B 084E 0844 00292                      dt     0x4b,   0x4e,   0x44,   0x41
```

```
        0841
0421    08C3 08C6 08CC 00293              dt      0xc3,    0xc6,    0xcc,    0xc9
        08C9
0425    08D8 08DD 08D7 00294              dt      0xd8,    0xdd,    0xd7,    0xd2
        08D2
0429    08F0 08F5 08FF 00295              dt      0xf0,    0xf5,    0xff,    0xfa
        08FA
042D    08EB 08EE 08E4 00296              dt      0xeb,    0xee,    0xe4,    0xe1
        08E1
0431    08A0 08A5 08AF 00297              dt      0xa0,    0xa5,    0xaf,    0xaa
        08AA
0435    08BB 08BE 08B4 00298              dt      0xbb,    0xbe,    0xb4,    0xb1
        08B1
0439    0893 0896 089C 00299              dt      0x93,    0x96,    0x9c,    0x99
        0899
043D    0888 088D 0887 00300              dt      0x88,    0x8d,    0x87,    0x82
        0882
0441    0883 0886 088C 00301              dt      0x83,    0x86,    0x8c,    0x89
        0889
0445    0898 089D 0897 00302              dt      0x98,    0x9d,    0x97,    0x92
        0892
0449    08B0 08B5 08BF 00303              dt      0xb0,    0xb5,    0xbf,    0xba
        08BA
044D    08AB 08AE 08A4 00304              dt      0xab,    0xae,    0xa4,    0xa1
        08A1
0451    08E0 08E5 08EF 00305              dt      0xe0,    0xe5,    0xef,    0xea
        08EA
0455    08FB 08FE 08F4 00306              dt      0xfb,    0xfe,    0xf4,    0xf1
        08F1
0459    08D3 08D6 08DC 00307              dt      0xd3,    0xd6,    0xdc,    0xd9
        08D9
045D    08C8 08CD 08C7 00308              dt      0xc8,    0xcd,    0xc7,    0xc2
        08C2
0461    0840 0845 084F 00309              dt      0x40,    0x45,    0x4f,    0x4a
        084A
0465    085B 085E 0854 00310              dt      0x5b,    0x5e,    0x54,    0x51
        0851
0469    0873 0876 087C 00311              dt      0x73,    0x76,    0x7c,    0x79
        0879
046D    0868 086D 0867 00312              dt      0x68,    0x6d,    0x67,    0x62
        0862
0471    0823 0826 082C 00313              dt      0x23,    0x26,    0x2c,    0x29
        0829
0475    0838 083D 0837 00314              dt      0x38,    0x3d,    0x37,    0x32
        0832
0479    0810 0815 081F 00315              dt      0x10,    0x15,    0x1f,    0x1a
        081A
```

```
047D   080B 080E 0804 00316          dt     0xb,    0xe,    0x4,    0x1
       0801
0481   0803 0806 080C 00317          dt     0x3,    0x6,    0xc,    0x9
       0809
0485   0818 081D 0817 00318          dt     0x18,   0x1d,   0x17,   0x12
       0812
0489   0830 0835 083F 00319          dt     0x30,   0x35,   0x3f,   0x3a
       083A
048D   082B 082E 0824 00320          dt     0x2b,   0x2e,   0x24,   0x21
       0821
0491   0860 0865 086F 00321          dt     0x60,   0x65,   0x6f,   0x6a
       086A
0495   087B 087E 0874 00322          dt     0x7b,   0x7e,   0x74,   0x71
       0871
0499   0853 0856 085C 00323          dt     0x53,   0x56,   0x5c,   0x59
       0859
049D   0848 084D 0847 00324          dt     0x48,   0x4d,   0x47,   0x42
       0842
04A1   08C0 08C5 08CF 00325          dt     0xc0,   0xc5,   0xcf,   0xca
       08CA
04A5   08DB 08DE 08D4 00326          dt     0xdb,   0xde,   0xd4,   0xd1
       08D1
04A9   08F3 08F6 08FC 00327          dt     0xf3,   0xf6,   0xfc,   0xf9
       08F9
04AD   08E8 08ED 08E7 00328          dt     0xe8,   0xed,   0xe7,   0xe2
       08E2
04B1   08A3 08A6 08AC 00329          dt     0xa3,   0xa6,   0xac,   0xa9
       08A9
04B5   08B8 08BD 08B7 00330          dt     0xb8,   0xbd,   0xb7,   0xb2
       08B2
04B9   0890 0895 089F 00331          dt     0x90,   0x95,   0x9f,   0x9a
       089A
04BD   088B 088E 0884 00332          dt     0x8b,   0x8e,   0x84,   0x81
       0881
04C1   0880 0885 088F 00333          dt     0x80,   0x85,   0x8f,   0x8a
       088A
04C5   089B 089E 0894 00334          dt     0x9b,   0x9e,   0x94,   0x91
       0891
04C9   08B3 08B6 08BC 00335          dt     0xb3,   0xb6,   0xbc,   0xb9
       08B9
04CD   08A8 08AD 08A7 00336          dt     0xa8,   0xad,   0xa7,   0xa2
       08A2
04D1   08E3 08E6 08EC 00337          dt     0xe3,   0xe6,   0xec,   0xe9
       08E9
04D5   08F8 08FD 08F7 00338          dt     0xf8,   0xfd,   0xf7,   0xf2
       08F2
04D9   08D0 08D5 08DF 00339          dt     0xd0,   0xd5,   0xdf,   0xda
```

```
        08DA
04DD    08CB 08CE 08C4 00340                  dt      0xcb,    0xce,    0xc4,    0xc1
        08C1
04E1    0843 0846 084C 00341                  dt      0x43,    0x46,    0x4c,    0x49
        0849
04E5    0858 085D 0857 00342                  dt      0x58,    0x5d,    0x57,    0x52
        0852
04E9    0870 0875 087F 00343                  dt      0x70,    0x75,    0x7f,    0x7a
        087A
04ED    086B 086E 0864 00344                  dt      0x6b,    0x6e,    0x64,    0x61
        0861
04F1    0820 0825 082F 00345                  dt      0x20,    0x25,    0x2f,    0x2a
        082A
04F5    083B 083E 0834 00346                  dt      0x3b,    0x3e,    0x34,    0x31
        0831
04F9    0813 0816 081C 00347                  dt      0x13,    0x16,    0x1c,    0x19
        0819
04FD    0808 080D 0807 00348                  dt      0x8,     0xd,     0x7
                       00349
                       00350                 END

Program Memory Words Used:   621
Program Memory Words Free:  1427


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     5 reported,     0 suppressed
```

NOTES:

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

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

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

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

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

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

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

New York
Microchip Technology Inc.
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

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

China - Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing, 100027, P.R.C.
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Shanghai
Microchip Technology
Unit B701, Far East International Plaza,
No. 317, Xianxia Road
Shanghai, 200051, P.R.C.
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore, 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
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
Tel: 82-2-554-7200 Fax: 82-2-558-5934

ASIA/PACIFIC (continued)

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
10F-1C 207
Tung Hua North Road
Taipei, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

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

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

Italy
Arizona 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 5858 Fax: 44-118 921-5835

03/23/00

DNV Certification, Inc.
USA

DNV MSC
The Netherlands
Accredited by the RvA

ANSI·RAB ACCREDITED QMS

DNV

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.