Modbus CRC uses the table lookup method for speed

Introduction:

This CRC (Cyclic Redundancy Check) is for the Modbus RTU protocol as used in RS232 and RS485 communications. The CRC is a 16 bit word that is calculated from all of the data in a message. It cannot correct any data corruption but it will indicate if a message has been corrupted in some way. This program uses the table lookup method for speed. It processes the message byte-by-byte rather than the bit-by-bit method which is more usual. This program generates the lookup table in software rather than entering the data into the table via a series of READ/DATA statements. It requires an integer table of 256 words.

PLEASE NOTE:

Although this is a Modbus CRC it is in no way restricted to be only used for the Modbus protocol. Any message structure consisting of a stream of 8-bit data bytes can utilise it. For example: Msg$ = “The quick brown fox jumps over the lazy dog”

Bill McKinley

An example a Modbus message (in Hex) could look like:

```
Slave Address             18
Function Code             03
Start Address (Hi)        0B
Start Address (Lo)        B9
Number of points (Hi)     00
Number of points (Lo)     01
CRC (Lo)                  55
CRC (Hi)                  C2
```

Note that the High/Low order of bytes in the CRC is the reverse of that used for the data. I don’t know why.

In MMBasic without the CRC, that message would look like:

```
TxMsg$ = CHR$(&H18)+CHR$(&H03)+CHR$(&H0B)+CHR$(&HB9)+CHR$(&H00)+CHR$(&H01)
```

Add the CRC to the message by:

```
TxMsg$ = TxMsg$ + CRC16$(TxMsg$)
```

Performing a CRC check on a message that already includes a valid CRC will produce a 'CRC' of &h000 so: Check the received message (eg RxMsg) by:

```
IF CRC16$(RxMsg$) = CHR$(0) + CHR$(0) THEN ' CRC is OK
' Process the response
ELSE
```

```
' Bad CRC - DO nothing - don't respond
END IF
```

To use the program, insert the code below into your program. The subroutine and function can be placed anywhere, preferably after the main body of the program. The table DIM statement should be near the top of the program and the 'genCRCtable' subroutine should be one of the first instructions as part of any initialisation.

Assumptions:

That the default: **OPTION BASE 0** is in force. If **OPTION BASE 1** is used then some changes will be needed.

The Program:

```
DIM INTEGER crc_table(255) ' lookup table for CRC generation

genCRCtable ' generate the lookup table


'-------------------------------------------------------------
'generates a lookup table for fast crc calculation
SUB genCRCtable
LOCAL i%, j%, k%, crc_const% = &HA001
FOR i% = 0 TO 255
  k% = i%
  FOR j% = 1 TO 8
    IF k% AND 1 THEN
      k% = k% >> 1
      k% = k% XOR crc_const%
    ELSE
      k% = k% >> 1
    END IF
  NEXT j%
 crc_table(i%)=k%
NEXT i%
END SUB
'
' returns the CRC as a string
FUNCTION CRC16$(a$)
  LOCAL CRC% = &HFFFF, n%
  FOR n% = 1 TO LEN(a$)
    CRC% = (CRC% >> 8) XOR crc_table(CRC% XOR ASC(MID$(a$, n%, 1)) AND &HFF)
  NEXT n%
  CRC16$ = CHR$(CRC% AND &HFF) + CHR$((CRC%>>8) AND &HFF)
END FUNCTION
```

```
'-------------------------------------------------------------
```

For the original Mono and Colour Maximites:

```
' Modbus CRC Generation using lookup table
' for the original Mono Maximite
' and the original Colour Maximite

DIM crc_table(255)

' Make the CRC table first
CRCTable
'
' Generates the lookup table
SUB CRCTable
local i, j, k
FOR i = 0 TO 255
  k = i
  FOR j = 1 TO 8
    IF k AND 1 THEN
      k = k \ 2
      k = k XOR &HA001
    ELSE
      k = k \ 2
    ENDIF
  NEXT j
 crc_table(i)=k
NEXT i
END SUB

' Calculates the Modbus CRC
FUNCTION CRC16(a$)
LOCAL CRC, n
  CRC = &HFFFF
  FOR n = 1 TO LEN(a$)
  CRC = (crc \ 256) XOR crc_table(crc XOR ASC(MID$(a$, n, 1)) AND &HFF)
NEXT n
CRC16 = (CRC MOD 256)*256 + CRC \ 256 MOD 256
END FUNCTION
```