Keywords: DS1862, CRC-8, CRC8, PEC, XFP, laser, laser driver, digital diagnostics, packet error checking

APPLICATION NOTE 3749

A Quick-Start Guide to DS1862 Packet-Error Checking

Mar 20, 2006

Abstract: This application note explains packet-error checking (PEC) as it applies to the DS1862 laser-control and digital-diagnostics IC. The DS1862 incorporates a CRC-8 byte and several other special bytes to improve the reliability of the communication bus. Topics discussed include two methods for calculating the CRC-8, several calculation examples, and a how-to guide on implementation specific to the DS1862.

What Is PEC and How Does It Work?

The DS1862 features a packet-error checking (PEC) mode (PEC enable, 76h, bit 0, = 1) that is useful to improve the reliability of the communication bus by detecting bit errors. By enabling PEC, extra information is included in the data string during each read and/or write sequence. This extra information includes the number of bytes (to be read/written), CAB (CRC add-on-byte), and the CRC-8 byte (check-sum). The differences between normal I²C‡ and PEC communication are detailed below.

Reading with PEC

During read operations, the communication sequence includes bytes for chip address, memory address, number of bytes to be read, a repeated chip address, 1 to 128 bytes of data, and the CRC-8. It is important to note that not all of these bytes will be included in the calculation of CRC-8. The included bytes are memory address, the number of bytes to be read, and all the data (1 to 128 bytes). Once the final CRC-8 byte has been transmitted, it can be compared with the host's calculated CRC-8. If they match, the host will signal with an ACK; if not, a NACK and a reread must occur.

Writing with PEC

During write operations, the communication sequence includes bytes for chip address, memory address, number of bytes to be read, 1 to 4 bytes of data, a CRC-add-on (CAB), and the CRC-8. The CAB byte can be written to 00h, and is intended to delay the communication sequence to allow the DS1862 time to calculate the CRC-8. As with reads, it is also important to note that not all of these bytes will be included in the calculation of CRC-8. Included bytes are memory address, the number of bytes to be read, and the data (1 to 4 bytes). Once the final byte (CRC-8) has been transmitted, the DS1862 compares it with its own internally calculated CRC-8. If they match, the DS1862 will signal with an ACK, if not, a NACK and a rewrite must occur.

Two Methods for Calculating CRC: Speed vs. Expense

For the host to compare CRC-8 values, it must first calculate the CRC-8 value. Typically, an algorithm is incorporated into the host's software (or firmware) to solve this problem. Because the DS1862's I²C communication can operate at speeds of 400kHz, a CRC must be calculated quickly. Depending on the host processor's speed (instructions per second) and the availability of extra memory (EEPROM) that can be used for a lookup table, one of two CRC-8 calculation methods is possible: direct calculation of the CRC-8 or calculation of the CRC-8 by indexing a lookup table.

Direct calculation of the CRC-8 typically takes longer than the lookup table method because it must "grind" through every data bit within the data string until the whole string is processed. The advantage to this method, however, is that it is relatively straightforward to understand and does not require much overhead in memory resources (EEPROM for lookup tables). Depending on the throughput of the host processor, direct calculation of the CRC-8 might or might not be an option. If a CRC-8 value cannot be calculated and/or compared in a reasonable amount of time, then the lookup table method can be the best option.

Direct-Calculation Method

The DS1862 uses a CRC polynomial of type $C(x) = X^8 + X^2 + X + 1$. The width of the polynomial is 8 (hence CRC-8) and can be represented as 1 0000 0111b. To implement software code for calculating the CRC-8 directly, a simple procedure can be established and programmed accordingly:

1. Concatenate the end of the data string (LSB) with eight zeros.
2. Perform the XOR operation on the data string against the binary version of the polynomial.
    1. First, shift the data string until a "1" appears at the MSB of the register.
    2. Line up the first "1" of the polynomial ($1X^8$ part) so that it will logically operate against the first "1" of the data string when the XOR is performed.
    3. Perform the logical XOR of the 8 bits. There are actually 9 bits being operated on, but the first "1" bit will always result in a "0." This "0" is in the MSB place, and thus does not contribute to the magnitude of the final CRC value.
3. The result of the XOR operation should then be augmented by the "untouched" bits (those bits in the data string that are nine places to the right of the first "1" in the string). This augmented result is now saved in place of the data string.
4. Continue the process of shifting and XORing (from step 2) until the LSB of the polynomial does not line up with any of the added eight zeros (the end of the data string with eight zeros added has been shifted sufficiently so that all bits have been operated on). The result will be a completed CRC-8.

Example 1

This example displays the actual mechanics involved in the calculation, as well as a real-life, single-byte write sequence using PEC. In this example, the CAB (a dummy byte write) is set to 00h.

Calculation Method

MSB LSB
10000000 00000001 10100011 = Data string (80 01 A3h)

$C(x) = X^8 + X^2 + X + 1 \rightarrow 100000111$ = Polynomial

```
100000000000000110100011000000000 = Original data string with eight zeros added
100000111                         = XOR polynomial
000000111000000110100011000000000 = XOR polynomial with augmented data string 0th shift
      100000111                   = XOR polynomial
      011000111110100011000000000 = XOR polynomial with augmented data string 1st shift
       100000111                  = XOR polynomial
       010001000101000011000000000 = XOR polynomial with augmented data string 2nd shift
        100000111                 = XOR polynomial
        000010110010001100000000 = XOR polynomial with augmented data string 3rd shift
            100000111             = XOR polynomial
            001100011011000000000 = XOR polynomial with augmented data string 4th shift
              100000111           = XOR polynomial
              010001010100000000 = XOR polynomial with augmented data string 5th shift
               100000111          = XOR polynomial
               000010010000000000 = XOR polynomial with augmented data string 6th shift
                   100000111      = XOR polynomial
                   000100111000000 = XOR polynomial with augmented data string 7th shift
                       100000111  = XOR polynomial
                       000111111 0 = Completed CRC-8 value (7Eh)
```

Therefore, a complete write sequence to the DS1862 would look like the following (**Figure 1**):

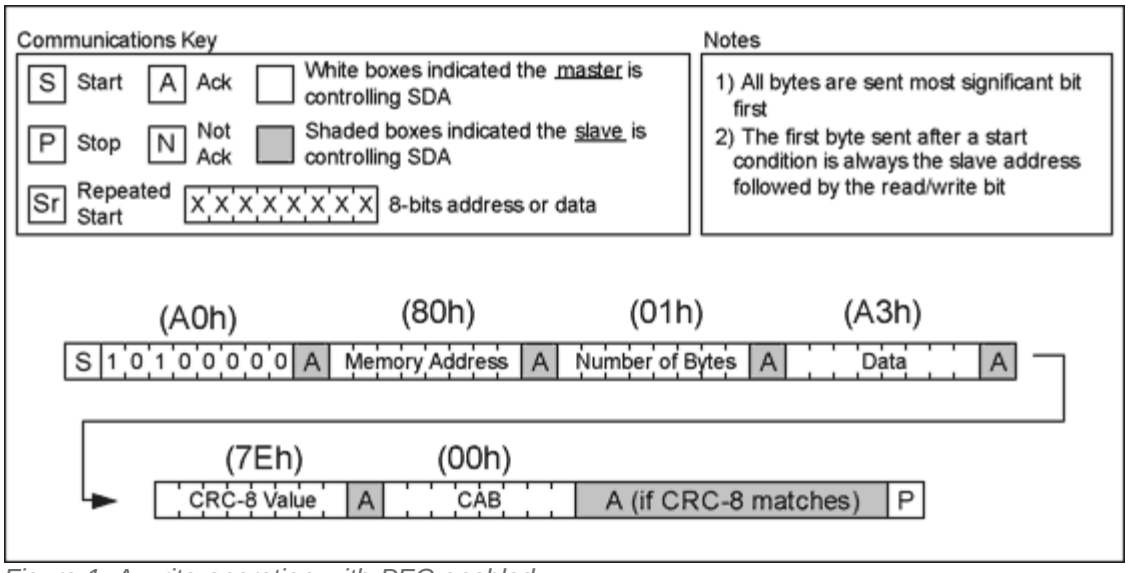

Figure 1. A write operation with PEC enabled.

Table-Driven Method

Some host systems might not be able to directly calculate the CRC-8 in a reasonable amount of time.

This is especially likely if the worst-case communication string is read: a long string with 128 data bytes. In this case, a table-driven method can be used to speed up the process. The table method is more efficient because it effectively processes a number of bits at a time. Consequently, the larger the table, the faster the CRC is calculated. Because the DS1862 communicates eight bits at a time, the most reasonable choice is to use a 256-location table capable of processing one byte at a time.

To use this method, a static 256-byte lookup table must first be available in memory and preloaded with the CRC values. Then during the calculation, the table has to be indexed and the result returned. A text file containing the 256-byte lookup table is available.

The procedure follows:

1. Initialize Register (variable) with zeros.
2. Perform the XOR operation on the data string, eight bits at a time.
    1. XOR 8 bits of the data string with the contents of Register (variable)
    2. The 8-bit result of the XOR operation is now used as a pointer into the 256-byte lookup table.
    3. Return the correct table entry.
    4. Continue the XOR operation on the 8-bit result of the lookup table query against the remaining bytes in the data string, one byte at a time, until the end of data string is reached.
3. The last value in the (variable) register is the CRC-8.

Example 2

This example displays the actual mechanics involved in the calculation, as well as a real-life read sequence of one byte using PEC.

Calculation Method

```
MSB                      LSB
10000000 00000001 10100011  = Data string  (80 01 A3h)

C(x) = X8 + X2 + X + 1  →  100000111 = Polynomial

          10000000000000110100011 = Original data string
               100000111          = XOR polynomial

               0000000            = Initialize variable with 00h
          10000000000000110100011 = Shift in 1st 8 bits
                    10000000      = XOR and use result as an pointer to CRC table
                    10001001      = Returned value from the table (pointer = 80h)
                00000000110100011 = Shift in 2nd 8 bits
                         10001000 = XOR and use result as a pointer
                         10110001 = Returned value from the table (pointer = 88h)
                         10100011 = Shift in 3rd 8 bits
                         00010010 = XOR and use result as a pointer
                         01111110 = Returned value from the table (pointer = 12h)

                         01111110 = Complete CRC-8 value (CRC-8 = 7Eh)
```

Therefore, a complete read sequence from the DS1862 would look like **Figure 2**:

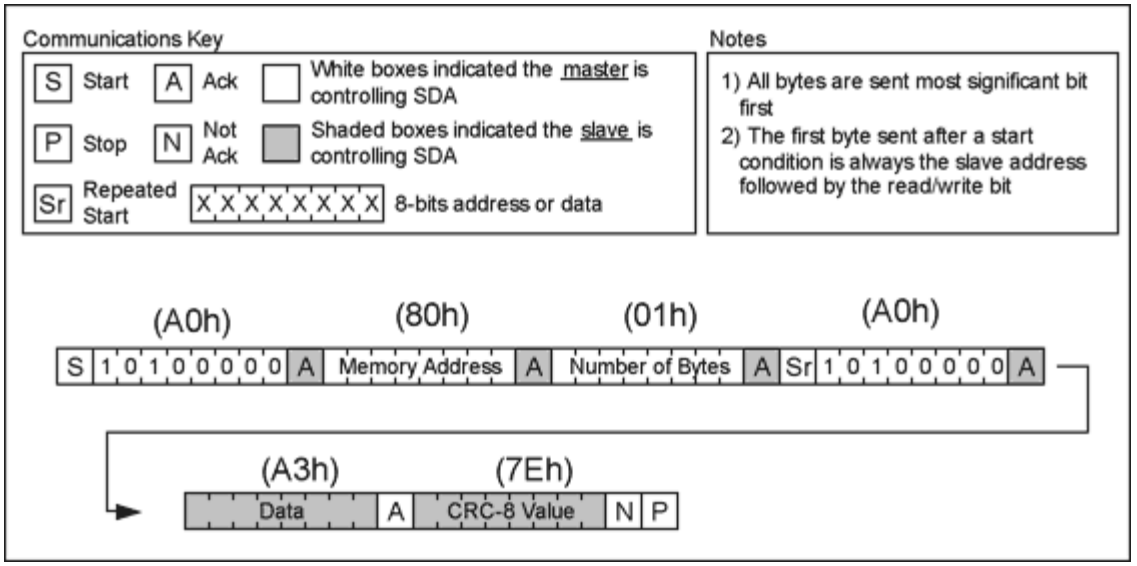

Figure 2. A read operation with PEC enabled.

To further illustrate the programming for the table-driven method, actual C code is provided below:

```
        unsigned char addressbyte, databyte;
        unsigned char CRCbyte, CRCresult;               // Required for PEC
transactions

        // Read 2 bytes from the specified memory address from the device at
slave address
        Start();
        WriteSlave(SlaveAddress & 0xFE);                // R/W bit = 0
        WriteSlave(addressbyte);                        // Starting memory
addr to read
        WriteSlave(0x01);                               // Read 1 byte
        Start();                                        // Repeated start
        WriteSlave(SlaveAddress | 0x01);                // R/W bit = 1
        ReadSlave(databyte,ACK);                        // Read first data
byte and ACK
        ReadSlave(CRCbyte,NACK);                        // Read CRC from
DS1862 and NACK
        Stop();

        // Calculate CRC - addressbyte, no. of bytes, databyte
        CRCresult = 0x00;
        CRCresult = CRC8LookUpTable(CRCresult ^ addressbyte);
        CRCresult = CRC8LookUpTable(CRCresult ^ 0x01);
        CRCresult = CRC8LookUpTable(CRCresult ^ databyte);


        // Verify CRC is correct
        if (CRCresult == CRCbyte)
        {  // CRCs match }
        else
        {  // Error - CRCs do not match! }
```

Conclusion

This application note provides a brief, yet functional understanding of packet-error checking (PEC) as it applies to the DS1862 laser-control and digital-diagnostics IC. The calculation and table-driven methods for calculating the CRC-8 are presented with examples, and are a helpful, insightful guide for calculating the CRC-8 and performing reads and writes.

If you have questions, comments, and suggestions about this application note, please contact us at: MixedSignalApps@maximintegrated.com.

Related Parts		
DS1862	XFP Laser Control and Digital Diagnostic IC	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 3749: http://www.maximintegrated.com/an3749
APPLICATION NOTE 3749, AN3749, AN 3749, APP3749, Appnote3749, Appnote 3749