Description of the LTpowerPlay™ -Calculated Configuration CRC Algorithm

Revision 0.1

Author: Mike Holloway, Nick Vergunst

Date: 12/22/2010

Revision History:

0.1	Initial Draft

Purpose

This document describes the algorithm that LTpowerPlay™ GUI uses to compute a special kind of configuration CRC that can be stored in the non-volatile memory (NVM) of the part. This CRC can optionally be used to identify a configuration. The full algorithm is provided in this document so that a third party can read all the configuration registers from the part and produce the same CRC for a more complete consistency check in applications where this is important.

The LTC2978 provides a great deal of power and configurability for custom applications. The part additionally provides onboard non-volatile memory (NVM) or EEPROM to store and recall configuration parameters. This architecture allows the IC to power up and load the desired customer configuration **autonomously with no I^2C/firmware interaction required**.

A common question customers ask is, “Now that I have settled on a particular configuration, how do I program this configuration into the IC’s onboard non-volatile memory (NVM)”. A continuum of options are presented in a separate document titled 'Overview of the LTC2978 Configuration EEPROM'.

When you order pre-programmed parts from a third party like LT or your CM, LTpowerPlay™ can generate the appropriate programming file. Each type of file has a CRC that is defined by the vendor to uniquely identify the contents of the file. It is important to note that these types of CRC's cannot be stored in the NVM of the part, because doing so would change the CRC. This is the reason that the User Configuration checksum was created.

The LTpowerPlay™ Configuration Checksum

The LTpowerPlay™ GUI can compute a special type of CRC and store it into the NVM of the part. LTpowerPlay™ calls this the 'Configuration Checksum' and it stores this 32-it CRC-32 into pages 6 and 7 of the MFR_SPARE3 scratchpad register.

Because this CRC is stored into the NVM of the part, it provides a convenient method for identifying a configuration by simply reading a register from the part. This method is both convenient and useful, but not bullet proof under all circumstances. For instance, a third party could easily modify a programmed part's configuration and store it to NVM with the STORE_USER_ALL command. Unless they have also computed and stored into MFR_SPARE3 a corresponding CRC for this configuration, the register now incorrectly indicates that it is still programmed with the original configuration.

Some customers prefer to compute their own CRC after reading all the configuration registers in the part. This method provides an extra level of completeness and confidence that no registers have been mis-programmed during device programming or re-programmed afterward. This becomes relevant if the customer has other software in their system that may modify the configuration.

Description of the Algorithm

LTpowerPlay™ uses the following pseudo-code algorithm to compute the Configuration Checksum:

```
1) Sort command registers from lowest command byte to highest
   (the same order shown in the "Register Command Set" table of the datasheet)
2) Initialize CRC to 0xFFFFFF
3) FOREACH command register in the above sorted list
    IF the register "NV Memory Type" is "EEPROM" AND is not in the CrcExclusionList**) THEN
      APPEND command address byte to CRC
      IF the register is paged (like OPERATION) THEN
        LOOP page from 0 to 7
          Append low byte to CRC
          IF the register is a WORD append high byte to CRC
        END LOOP
      ELSE (global register)
        Append low byte to CRC
        IF the register is a WORD append high byte to CRC
  END FOREACH
4) RETURN ~CRC as the final CRC-32
```

**See the next page for Excluded Registers

**Excluded Registers:

The CrcExclusionList contains the following registers which are not used in the CRC calculation:

CAPABILITY (command code 0x19)

VOUT_MODE (command code 0x20)

STATUS_BYTE (command code 0x78)

STATUS_WORD (command code 0x79)

STATUS_VOUT (command code 0x7A)

STATUS_INPUT (command code 0x7C)

STATUS_TEMP (command code 0x7D)

STATUS_CML (command code 0x7E)

STATUS_MFR_SPECIFIC (command code 0x80)

READ_VIN (command code 0x88)

READ_VOUT (command code 0x8B)

READ_TEMPERATURE_1 (command code 0x8D)

PMBUS_REVISION (command code 0x98)

MFR_VOUT_PEAK (command code 0xDD)

MFR_VIN_PEAK (command code 0xDE)

MVR_TEMPERATURE_PEAK (command code 0xDF)

MFR_DAC (command code 0xE0)

MFR_PADS (command code 0xE5)

MFR_SPECIAL_ID (command code 0xE7)

MFR_SPECIAL_LOT (command code 0xE8)

MFR_FAULT_LOG_STATUS (command code 0xED)

MFR_COMMON (command code 0xEF)

MFR_SPARE1 (command code 0xF8)

MFR_SPARE3 (command code 0xFA)

MFR_VOUT_MIN (command code 0xFB)

MFR_VIN_MIN (command code 0xFC)

MFR_TEMPERATURE_MIN (command code 0xFD)

Further Reading

The algorithm above utilizes the CRC-32 algorithm, which uses the standard CRC-32 polynomial x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1. If you are unfamiliar with CRC computations, a complete code example for implementing the CRC-32 is included in Appendix A.

Appendix B shows the steps required to generate an arbitrary example configuration that may be used to verify your software algorithms. This example configuration is referenced in Appendix C

Appendix C shows a step-by-step example trace of the algorithm as each register and its data byte(s) are added to the CRC in the appropriate order.

Appendix A: Example Code for generic CRC-32 calculation

The following example code is taken from the SharpZipLibrary which includes an open source implementation of the CRC-32 calculation in the c# language.

```
// CRC32.cs - Computes CRC32 data checksum of a data stream
// Copyright (C) 2001 Mike Krueger
//
// This file was translated from java, it was part of the GNU Classpath
// Copyright (C) 1999, 2000, 2001 Free Software Foundation, Inc.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License
// as published by the Free Software Foundation; either version 2
// of the License, or (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
//
// Linking this library statically or dynamically with other modules is
// making a combined work based on this library.  Thus, the terms and
// conditions of the GNU General Public License cover the whole
// combination.
//
// As a special exception, the copyright holders of this library give you
// permission to link this library with independent modules to produce an
// executable, regardless of the license terms of these independent
// modules, and to copy and distribute the resulting executable under
// terms of your choice, provided that you also meet, for each linked
// independent module, the terms and conditions of the license of that
// module.  An independent module is a module which is not derived from
// or based on this library.  If you modify this library, you may extend
// this exception to your version of the library, but you are not
// obligated to do so.  If you do not wish to do so, delete this
// exception statement from your version.
using System;
using ICSharpCode.SharpZipLib.Checksums;

namespace ICSharpCode.SharpZipLib.Checksums
{

    /// <summary>
    /// Generate a table for a byte-wise 32-bit CRC calculation on the polynomial:
    /// x^32+x^26+x^23+x^22+x^16+x^12+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x+1.
    ///
    /// Polynomials over GF(2) are represented in binary, one bit per coefficient,
    /// with the lowest powers in the most significant bit.  Then adding polynomials
    /// is just exclusive-or, and multiplying a polynomial by x is a right shift by
    /// one.  If we call the above polynomial p, and represent a byte as the
    /// polynomial q, also with the lowest power in the most significant bit (so the
    /// byte 0xb1 is the polynomial x^7+x^3+x+1), then the CRC is (q*x^32) mod p,
    /// where a mod b means the remainder after dividing a by b.
    ///
    /// This calculation is done using the shift-register method of multiplying and
    /// taking the remainder.  The register is initialized to zero, and for each
    /// incoming bit, x^32 is added mod p to the register if the bit is a one (where
    /// x^32 mod p is p+x^32 = x^26+...+1), and the register is multiplied mod p by
    /// x (which is shifting right by one and adding x^32 mod p if the bit shifted
    /// out is a one).  We start with the highest power (least significant bit) of
    /// q and repeat for all eight bits of q.
    ///
    /// The table is simply the CRC of all possible eight bit values.  This is all
```

```
    /// the information needed to generate CRC's on data a byte at a time for all
    /// combinations of CRC register values and incoming bytes.
    /// </summary>
    public sealed class Crc32 : IChecksum
    {
        readonly static uint CrcSeed = 0xFFFFFFFF;

        readonly static uint[] CrcTable = new uint[] {
            0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA, 0x076DC419,
            0x706AF48F, 0xE963A535, 0x9E6495A3, 0x0EDB8832, 0x79DCB8A4,
            0xE0D5E91E, 0x97D2D988, 0x09B64C2B, 0x7EB17CBD, 0xE7B82D07,
            0x90BF1D91, 0x1DB71064, 0x6AB020F2, 0xF3B97148, 0x84BE41DE,
            0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7, 0x136C9856,
            0x646BA8C0, 0xFD62F97A, 0x8A65C9EC, 0x14015C4F, 0x63066CD9,
            0xFA0F3D63, 0x8D080DF5, 0x3B6E20C8, 0x4C69105E, 0xD56041E4,
            0xA2677172, 0x3C03E4D1, 0x4B04D447, 0xD20D85FD, 0xA50AB56B,
            0x35B5A8FA, 0x42B2986C, 0xDBBBC9D6, 0xACBCF940, 0x32D86CE3,
            0x45DF5C75, 0xDCD60DCF, 0xABD13D59, 0x26D930AC, 0x51DE003A,
            0xC8D75180, 0xBFD06116, 0x21B4F4B5, 0x56B3C423, 0xCFBA9599,
            0xB8BDA50F, 0x2802B89E, 0x5F058808, 0xC60CD9B2, 0xB10BE924,
            0x2F6F7C87, 0x58684C11, 0xC1611DAB, 0xB6662D3D, 0x76DC4190,
            0x01DB7106, 0x98D220BC, 0xEFD5102A, 0x71B18589, 0x06B6B51F,
            0x9FBFE4A5, 0xE8B8D433, 0x7807C9A2, 0x0F00F934, 0x9609A88E,
            0xE10E9818, 0x7F6A0DBB, 0x086D3D2D, 0x91646C97, 0xE6635C01,
            0x6B6B51F4, 0x1C6C6162, 0x856530D8, 0xF262004E, 0x6C0695ED,
            0x1B01A57B, 0x8208F4C1, 0xF50FC457, 0x65B0D9C6, 0x12B7E950,
            0x8BBEB8EA, 0xFCB9887C, 0x62DD1DDF, 0x15DA2D49, 0x8CD37CF3,
            0xFBD44C65, 0x4DB26158, 0x3AB551CE, 0xA3BC0074, 0xD4BB30E2,
            0x4ADFA541, 0x3DD895D7, 0xA4D1C46D, 0xD3D6F4FB, 0x4369E96A,
            0x346ED9FC, 0xAD678846, 0xDA60B8D0, 0x44042D73, 0x33031DE5,
            0xAA0A4C5F, 0xDD0D7CC9, 0x5005713C, 0x270241AA, 0xBE0B1010,
            0xC90C2086, 0x5768B525, 0x206F85B3, 0xB966D409, 0xCE61E49F,
            0x5EDEF90E, 0x29D9C998, 0xB0D09822, 0xC7D7A8B4, 0x59B33D17,
            0x2EB40D81, 0xB7BD5C3B, 0xC0BA6CAD, 0xEDB88320, 0x9ABFB3B6,
            0x03B6E20C, 0x74B1D29A, 0xEAD54739, 0x9DD277AF, 0x04DB2615,
            0x73DC1683, 0xE3630B12, 0x94643B84, 0x0D6D6A3E, 0x7A6A5AA8,
            0xE40ECF0B, 0x9309FF9D, 0x0A00AE27, 0x7D079EB1, 0xF00F9344,
            0x8708A3D2, 0x1E01F268, 0x6906C2FE, 0xF762575D, 0x806567CB,
            0x196C3671, 0x6E6B06E7, 0xFED41B76, 0x89D32BE0, 0x10DA7A5A,
            0x67DD4ACC, 0xF9B9DF6F, 0x8EBEEFF9, 0x17B7BE43, 0x60B08ED5,
            0xD6D6A3E8, 0xA1D1937E, 0x38D8C2C4, 0x4FDFF252, 0xD1BB67F1,
            0xA6BC5767, 0x3FB506DD, 0x48B2364B, 0xD80D2BDA, 0xAF0A1B4C,
            0x36034AF6, 0x41047A60, 0xDF60EFC3, 0xA867DF55, 0x316E8EEF,
            0x4669BE79, 0xCB61B38C, 0xBC66831A, 0x256FD2A0, 0x5268E236,
            0xCC0C7795, 0xBB0B4703, 0x220216B9, 0x5505262F, 0xC5BA3BBE,
            0xB2BD0B28, 0x2BB45A92, 0x5CB36A04, 0xC2D7FFA7, 0xB5D0CF31,
            0x2CD99E8B, 0x5BDEAE1D, 0x9B64C2B0, 0xEC63F226, 0x756AA39C,
            0x026D930A, 0x9C0906A9, 0xEB0E363F, 0x72076785, 0x05005713,
            0x95BF4A82, 0xE2B87A14, 0x7BB12BAE, 0x0CB61B38, 0x92D28E9B,
            0xE5D5BE0D, 0x7CDCEFB7, 0x0BDBDF21, 0x86D3D2D4, 0xF1D4E242,
            0x68DDB3F8, 0x1FDA836E, 0x81BE16CD, 0xF6B9265B, 0x6FB077E1,
            0x18B74777, 0x88085AE6, 0xFF0F6A70, 0x66063BCA, 0x11010B5C,
            0x8F659EFF, 0xF862AE69, 0x616BFFD3, 0x166CCF45, 0xA00AE278,
            0xD70DD2EE, 0x4E048354, 0x3903B3C2, 0xA7672661, 0xD06016F7,
            0x4969474D, 0x3E6E77DB, 0xAED16A4A, 0xD9D65ADC, 0x40DF0B66,
            0x37D83BF0, 0xA9BCAE53, 0xDEBB9EC5, 0x47B2CF7F, 0x30B5FFE9,
            0xBDBDF21C, 0xCABAC28A, 0x53B39330, 0x24B4A3A6, 0xBAD03605,
            0xCDD70693, 0x54DE5729, 0x23D967BF, 0xB3667A2E, 0xC4614AB8,
            0x5D681B02, 0x2A6F2B94, 0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B,
            0x2D02EF8D
        };

        internal static uint ComputeCrc32(uint oldCrc, byte value)
        {
            return (uint)(Crc32.CrcTable[(oldCrc ^ value) & 0xFF] ^
(oldCrc >> 8));
        }

        /// <summary>
        /// The crc data checksum so far.
```

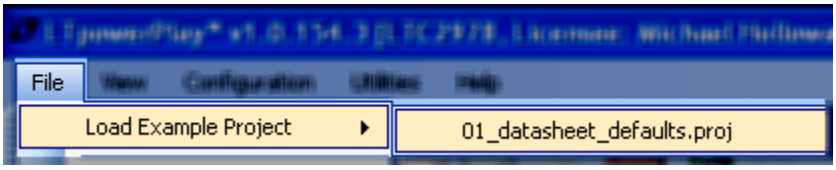
```
/// </summary>
uint crc;

/// <summary>
/// Returns the CRC32 data checksum computed so far.
/// </summary>
public long Value {
    get {
        return (long)crc;
    }
    set {
        crc = (uint)value;
    }
}

/// <summary>
/// Resets the CRC32 data checksum as if no update was ever called.
/// </summary>
public void Reset()
{
    crc = 0;
}

/// <summary>
/// Updates the checksum with the int bval.
/// </summary>
/// <param name = "value">
/// the byte is taken as the lower 8 bits of value
/// </param>
public void Update(int value)
{
    crc ^= CrcSeed;
    crc  = CrcTable[(crc ^ value) & 0xFF] ^ (crc >> 8);
    crc ^= CrcSeed;
}

/// <summary>
/// Updates the checksum with the bytes taken from the array.
/// </summary>
/// <param name="buffer">
/// buffer an array of bytes
/// </param>
public void Update(byte[] buffer)
{
    if (buffer == null) {
        throw new ArgumentNullException("buffer");
    }

    Update(buffer, 0, buffer.Length);
}

/// <summary>
/// Adds the byte array to the data checksum.
/// </summary>
/// <param name = "buffer">
/// The buffer which contains the data
/// </param>
/// <param name = "offset">
/// The offset in the buffer where the data starts
/// </param>
/// <param name = "count">
/// The number of data bytes to update the CRC with.
/// </param>
public void Update(byte[] buffer, int offset, int count)
{
    if (buffer == null) {
        throw new ArgumentNullException("buffer");
    }

    if ( count < 0 ) {
```

```
#if NETCF_1_0
                throw new ArgumentOutOfRangeException("count");
#else
                throw new ArgumentOutOfRangeException("count", "Count cannot be less than zero");
#endif
            }

            if (offset < 0 || offset + count > buffer.Length) {
                throw new ArgumentOutOfRangeException("offset");
            }

            crc ^= CrcSeed;

            while (--count >= 0) {
                crc = CrcTable[(crc ^ buffer[offset++]) & 0xFF] ^ (crc >> 8);
            }

            crc ^= CrcSeed;
        }
    }
}
```

Appendix B: Generating and Parsing the Example Configuration

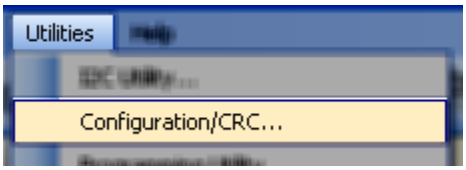
This section explains how to generate an arbitrary example configuration that you can use to verify your software. You can load the example configuration into the GUI as follows:

- Launch LTpowerPlay™, select the LTC2978 if necessary
- Select File, Load Example Project, 01_datasheet_defaults.proj from the menu as shown below to load the example configuration:

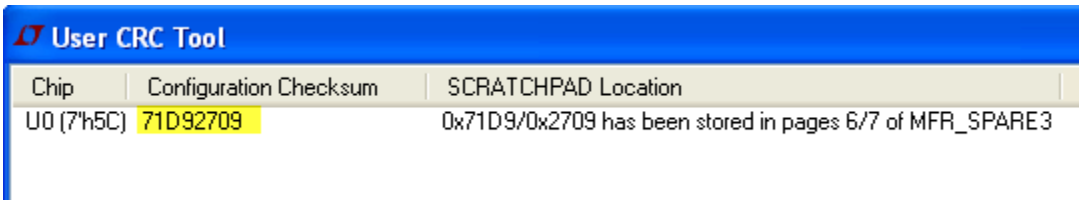


Note that LTpowerPlay™ has a built-in utility that will show you the Configuration checksum corresponding to a project file. You can launch it as follows:

- Select Utilities, Configuration/CRC from the menus as shown below:



- Note the 32-bit Configuration Checksum shown in this utility.



For the example configuration, our Configuration Checksum is **0x71D92709**. Depending on what you wish to do, you may use this configuration in one of two ways:

NOTE: LTpowerPlay™ reserves MFR_SPARE3 and for any configuration it will store the corresponding GUI-computed 32-bit CRC into pages 6 and 7 of MFR_SPARE3 as shown above.

Because LTpowerPlay™ keeps this register up to date, when you order programmed parts from any vendor (LT or your CM) this CRC value will be stored into the NVM of your programmed parts during programming.

Your software may subsequently read pages 6 and 7 of MFR_SPARE3 to quickly identify configurations if you do not wish or need to read all the configuration registers and generate your own CRC.

If you do wish to compute your own CRC, consult the details provided in **Scenario B** below.

Scenario A. You wish to read all the user configuration registers from hardware and generate a corresponding CRC

In this scenario, your software will read all the appropriate configuration registers and compute a CRC. To aid in verifying your software, you probably want to program a 'test part' with the example configuration. You can obtain programmed parts in a number of ways, as detailed in the document titled **'Overview of the LTC2978 Configuration EEPROM'**. Consult this document to program a part with the above configuration so that you can test your software.

Once you have a programmed part with the example configuration, you can use this example part to verify your software algorithms. Your software should read all the registers and compute a CRC in the order as shown in Appendix C. If your software is correct, it will produce the CRC-32 value of **0x71D92709** for the example configuration.

Your software needs to be informed of three key attributes for each register. The first is whether the register is included in the CRC computation. The register exclusion list is shown in the main body of this document.

The second is how many data bytes are needed for the register. For example, when your software attempts to read a WORD register, it must read 2 bytes from hardware, and then feed the command code, then the lo data byte, then the hi data byte to the CRC calculation. When your software encounters a byte register, it should read a single byte from hardware, and then feed the command code, then the data byte to the CRC calculation. Consult Appendix C if you have any confusion as to the order or number of bytes required for any given register.

The third attribute that your software needs to be informed of is whether the register is paged. If the register is not paged (global) you can simply read this register and append it to the CRC as indicated above. If the register is paged, you need to read the value for pages 0-7, insuring that you write the PAGE register to address a given page.

For example, the first register included in the CRC is the OPERATION register, which is command code 0x01. This register is paged. So you must read the value of the OPERATION register for each page as follows:

- Write the PAGE register to **0** via SmBus Write Byte
- Read the OPERATION register as a single byte (let's say this returns 0x00)
- Append 0x01 (command code for OPERATION) to the CRC
- Append 0x00 (data value of OPERATION for PAGE 0) to the CRC
- Write the PAGE register to **1** via SmBus Write Byte
- Read the OPERATION register as a single byte (let's say this returns 0x80)
- Append 0x01 (command code for OPERATION) to the CRC
- Append 0x80 (data value of OPERATION for PAGE 1) to the CRC
- ...
- And so on through PAGE 7

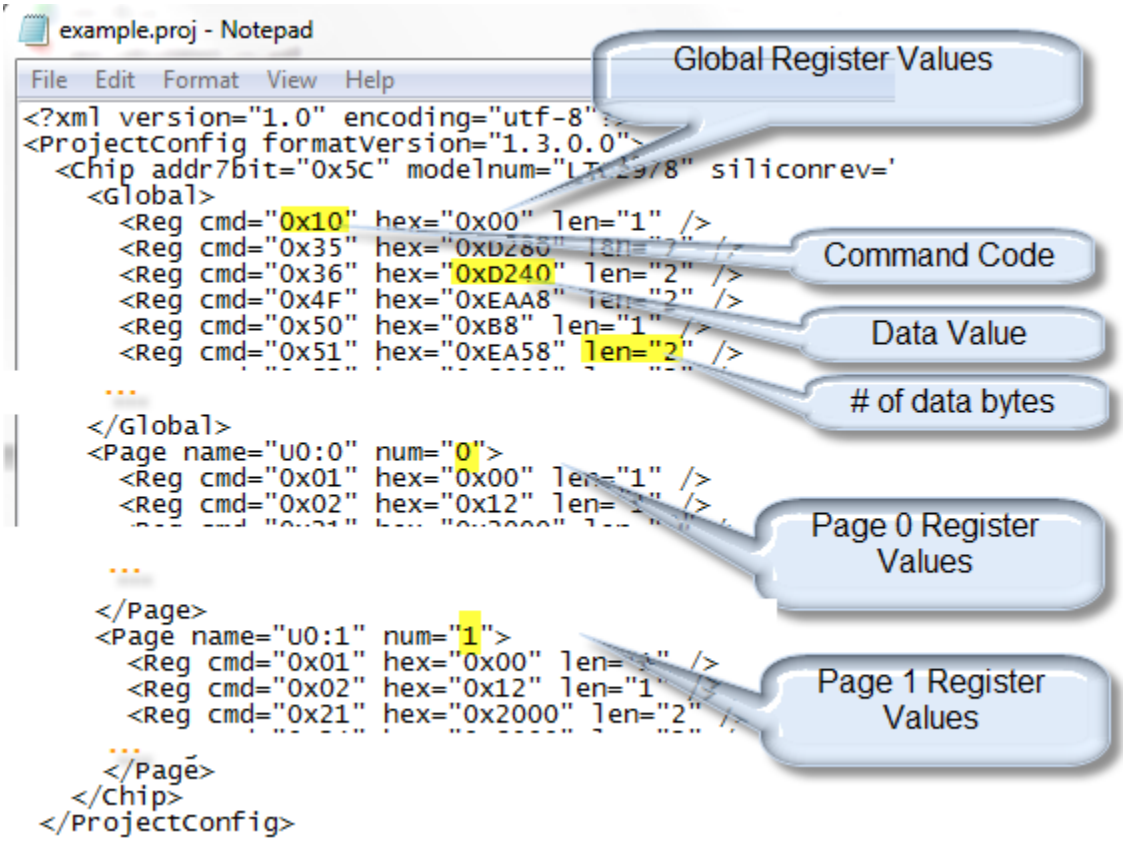
If you have any questions or confusion about the order of the bytes, consult Appendix C which shows a specific example with exact ordering along with a running CRC-32 computation for each register fed to the CRC calculation.

Scenario B. You wish to compute a CRC from an LTpowerPlay™ Project File (.proj file) without reading from hardware.

Though the GUI can compute a CRC from a project file as shown above, some customers prefer to write their own tools to process a .proj file and compute a CRC. This requires parsing the project file first. The project file is in a simple standard text format called XML (eXtensible Markup Language).

The structure of the file is a simple hierarchical text format the defines register settings in 'groups'. For example, a project file can contain one or more 'chips'. Our example only contains one chip, so it only has one '<Chip>' element. Register values for this chip are contained within a hierarchical XML 'element' named 'Chip'. For example, the beginninf of this chip is marked as <Chip> and an ending is marked as </Chip>. This flexible structure allows for easy representation of hierarchical structures like a multi-chip project configuration. Many standard tools/libraries exist to parse XML files. Once you understand the basic structure of the XML file, it is usually quite simple to parse it with standard libraries in your language of choice.

It may be helpful to look at a tangible example. Excerpts from the example configuration .proj file are shown below:



Let's take a look at what's inside. Our project contains only one chip so there is only one <Chip> element in the XML file. The XML 'attribute' named addr7bit defines the address of this chip. In our case, the address is 0x5C. Each chip contains one and only one <Global> element, which contains the register values for global registers.

Within the <Global> element are many <Reg> elements, each of which define the register value for one register. For example, the first <Reg> element along with it's cmd, hex, and len attributes tells us that the global register with command byte 0x10 (WRITE_PROTECT), has the data value 0x00 and it is a single byte register.

What follows is a set of eight <Page> elements, one for each page of the LTC2978. Each <Page> element is distinguished by its 'num' attribute, which defines the Page number. Within each <Page> multiple <Reg> elements define the register values for that page. For example, the value for command code 0x01 for PAGE 0 is the single byte 0x00.

Note that when computing a CRC from this .proj file the first step would be to parse the XML file to obtain all the register values. Then you must follow the pseudo-code algorithm sketched in the main body of this document to order the register values appropriately, because CRC computation is sensitive to the order in which the bytes are fed to the CRC algorithm. If you have any questions or confusion about the order of the bytes, consult Appendix C which shows a specific example with exact ordering along with a running CRC-32 computation for each register fed to the CRC calculation.

Appendix C: Example Trace of Configuration Checksum Algorithm

The table below shows a step-by-step trace of the algorithm using the arbitrary example configuration generated in Appendix B. Each row of this table represents the CRC calculation for one register, in the order that the registers are added. For paged registers, page 0 is added first, then page 1, all the way up to page 7.

The bytes for an individual register command/value are added to the CRC in the order shown in the table from left to right. That is, the command code is always added to the CRC first. Then the low data byte is added to the CRC. If the register is a WORD (16-bit) register, the high data byte is added to the CRC.

Each row in the table also shows the accumulated (running CRC-32) calculation for all the bytes up to that point.

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
'OPERATION'	'01'	'00'		'58C223BE'	Because OPERATION is paged, there are 8 entries, one for each page. The first value is for PAGE=0.
'OPERATION'	'01'	'00'		'80E38938'	Because this is a byte register, only one data byte is sent to the CRC (0x0)
'OPERATION'	'01'	'00'		'DB392422'	
'OPERATION'	'01'	'00'		'C3735A76'	
'OPERATION'	'01'	'00'		'AA198657'	
'OPERATION'	'01'	'00'		'A9E36421'	
'OPERATION'	'01'	'00'		'9E823AFF'	
'OPERATION'	'01'	'00'		'0DEA99FC'	This is the value of the OPERATION register for PAGE=7
'ON_OFF_CONFIG'	'02'	'12'		'B18C898E'	This is the value of the ON_OFF_CONFIG register for PAGE=0
'ON_OFF_CONFIG'	'02'	'12'		'B1323EF8'	
'ON_OFF_CONFIG'	'02'	'12'		'9D52C619'	
'ON_OFF_CONFIG'	'02'	'12'		'695152FA'	
'ON_OFF_CONFIG'	'02'	'12'		'EB60518B'	
'ON_OFF_CONFIG'	'02'	'12'		'444DCAB7'	
'ON_OFF_CONFIG'	'02'	'12'		'506356A9'	
'ON_OFF_CONFIG'	'02'	'12'		'7D9BA237'	This is the value of the ON_OFF_CONFIG register for PAGE=7
'WRITE_PROTECT'	'10'	'00'		'A3C41FC5'	Because WRITE_PROTECT is a global register, there is only one entry
					Skipping Register CAPABILITY (0x19) -not included in CRC

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
					Skipping Register VOUT_MODE (0x20) -not included in CRC
'VOUT_COMMAND'	'21'	'00'	'20'	'3B978932'	Because this is a WORD register, 2 data bytes are sent to the CRC (0x0 and then 0x20)
'VOUT_COMMAND'	'21'	'00'	'20'	'5E65C12D'	
'VOUT_COMMAND'	'21'	'00'	'20'	'220C7338'	
'VOUT_COMMAND'	'21'	'00'	'20'	'DA7C0719'	
'VOUT_COMMAND'	'21'	'00'	'20'	'F8A5E6F8'	
'VOUT_COMMAND'	'21'	'00'	'20'	'E8EFF490'	
'VOUT_COMMAND'	'21'	'00'	'20'	'40D6B645'	
'VOUT_COMMAND'	'21'	'00'	'20'	'56DEE4B9'	
'VOUT_MAX'	'24'	'00'	'80'	'BC4BB8FC'	
'VOUT_MAX'	'24'	'00'	'80'	'5C7182AE'	
'VOUT_MAX'	'24'	'00'	'80'	'D0EC7F5B'	
'VOUT_MAX'	'24'	'00'	'80'	'4C4FBEE5'	
'VOUT_MAX'	'24'	'00'	'80'	'1EB1793E'	
'VOUT_MAX'	'24'	'00'	'80'	'41D33D2F'	
'VOUT_MAX'	'24'	'00'	'80'	'6B9E7E42'	
'VOUT_MAX'	'24'	'00'	'80'	'F8A807F0'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'2488E1E2'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'0860E282'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'C54B1007'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'A883B3DE'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'3A4E18A4'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'CB0E2E9F'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'1CF0F4FA'	
'VOUT_MARGIN_HIGH'	'25'	'9A'	'21'	'733E0F68'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'0A14898D'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'135F6DA4'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'FFDFBE22'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'5DCACC00'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'16D7A1B3'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'629A9882'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'41DAA7C6'	
'VOUT_MARGIN_LOW'	'26'	'66'	'1E'	'184BE955'	
'VIN_ON'	'35'	'80'	'D2'	'8A7AE924'	
'VIN_OFF'	'36'	'40'	'D2'	'472B82FC'	
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'C000C9EA'	
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'679A7921'	
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'0E7FD4C5'	
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'6B5D7883'	
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'C8983B95'	
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'E069CE0D'	

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'B018F01F'	
'VOUT_OV_FAULT_LIMIT'	'40'	'33'	'23'	'C97DDD86'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'D0B8B742'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'D249348B'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'BCC088E7'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'B75F3D62'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'4AA01DF0'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'32B9285F'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'4D953443'	
'VOUT_OV_FAULT_RESPONSE'	'41'	'80'		'BFE34A7D'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'0524E288'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'D5F532FB'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'F3AEC37F'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'E48C2885'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'3435B557'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'CC6AABA6'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'57F9C103'	
'VOUT_OV_WARN_LIMIT'	'42'	'66'	'22'	'7E5E1FC0'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'B5130091'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'1D5A776E'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'CCBC552E'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'52FDD952'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'99FBEEB9'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'4528D432'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'923B7D98'	
'VOUT_UV_WARN_LIMIT'	'43'	'9A'	'1D'	'BDAFE387'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'35C71AAC'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'87C800BB'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'BE3352B6'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'1409D59E'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'90991552'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'33D93EF8'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'E79A5A14'	
'VOUT_UV_FAULT_LIMIT'	'44'	'CD'	'1C'	'A926CD8D'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'6795343F'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'75077C0E'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'CB5DA08C'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'4D8D7AE0'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'E2C3E351'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'96CEF806'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'E9518E74'	
'VOUT_UV_FAULT_RESPONSE'	'45'	'7F'		'4D3E3E96'	
'OT_FAULT_LIMIT'	'4F'	'A8'	'EA'	'DBBCECB1'	
'OT_FAULT_RESPONSE'	'50'	'B8'		'6ABA9E03'	
'OT_WARN_LIMIT'	'51'	'58'	'EA'	'66B2029A'	
'UT_WARN_LIMIT'	'52'	'00'	'80'	'9A1C2D8E'	

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
'UT_FAULT_LIMIT'	'53'	'80'	'CD'	'906D5F0D'	
'UT_FAULT_RESPONSE'	'54'	'B8'		'14CE1354'	
'VIN_OV_FAULT_LIMIT'	'55'	'C0'	'D3'	'37904934'	
'VIN_OV_FAULT_RESPONSE'	'56'	'80'		'F4AA565E'	
'VIN_OV_WARN_LIMIT'	'57'	'80'	'D3'	'0E2B4968'	
'VIN_UV_WARN_LIMIT'	'58'	'00'	'80'	'BB95B2A0'	
'VIN_UV_FAULT_LIMIT'	'59'	'00'	'80'	'FE7AB927'	
'VIN_UV_FAULT_RESPONSE'	'5A'	'00'		'69F5E016'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'C472E9AF'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'ACE38157'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'3D18E12A'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'2EA35A8B'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'AD9B70B8'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'CCDF5F44'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'13AC6875'	
'POWER_GOOD_ON'	'5E'	'B8'	'1E'	'6F0E67BF'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'4F06338C'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'B9277807'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'0BA30A00'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'F930407A'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'C3C6C87F'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'62894FC4'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'39230C19'	
'POWER_GOOD_OFF'	'5F'	'14'	'1E'	'5125FC11'	
'TON_DELAY'	'60'	'00'	'BA'	'72CBFB40'	
'TON_DELAY'	'60'	'00'	'BA'	'1746E65C'	
'TON_DELAY'	'60'	'00'	'BA'	'6E1F2FAC'	
'TON_DELAY'	'60'	'00'	'BA'	'D354D952'	
'TON_DELAY'	'60'	'00'	'BA'	'CFE65916'	
'TON_DELAY'	'60'	'00'	'BA'	'A69936FD'	
'TON_DELAY'	'60'	'00'	'BA'	'21F11210'	
'TON_DELAY'	'60'	'00'	'BA'	'324A59A8'	
'TON_RISE'	'61'	'80'	'D2'	'CF1111CF'	
'TON_RISE'	'61'	'80'	'D2'	'469081E6'	
'TON_RISE'	'61'	'80'	'D2'	'9AF2C5AC'	
'TON_RISE'	'61'	'80'	'D2'	'D0564713'	
'TON_RISE'	'61'	'80'	'D2'	'C65A634C'	
'TON_RISE'	'61'	'80'	'D2'	'59E4A75A'	
'TON_RISE'	'61'	'80'	'D2'	'C200F4C6'	
'TON_RISE'	'61'	'80'	'D2'	'00F89D3F'	
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'388CBEA8'	
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'2911D2A1'	
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'6136226D'	
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'483519D7'	
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'2547DCBA'	
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'6DCEB7BF'	

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'7968FCF4'	
'TON_MAX_FAULT_LIMIT'	'62'	'C0'	'D3'	'299F0ED2'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'9E05B220'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'7A587922'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'44D8E4C5'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'DB595097'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'9FD88755'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'7EA3316F'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'790818D4'	
'TON_MAX_FAULT_RESPONSE'	'63'	'B8'		'3C8BF060'	
'TOFF_DELAY'	'64'	'00'	'BA'	'BD2A9075'	
'TOFF_DELAY'	'64'	'00'	'BA'	'63EB09B6'	
'TOFF_DELAY'	'64'	'00'	'BA'	'BCE2E204'	
'TOFF_DELAY'	'64'	'00'	'BA'	'BE6E5A26'	
'TOFF_DELAY'	'64'	'00'	'BA'	'4DB52311'	
'TOFF_DELAY'	'64'	'00'	'BA'	'836D1914'	
'TOFF_DELAY'	'64'	'00'	'BA'	'28075FAE'	
'TOFF_DELAY'	'64'	'00'	'BA'	'EB8A4406'	
					Skipping Register STATUS_BYTE (0x78) -not included in CRC
					Skipping Register STATUS_WORD (0x79) -not included in CRC
					Skipping Register STATUS_VOUT (0x7A) -not included in CRC
					Skipping Register STATUS_INPUT (0x7C) -not included in CRC
					Skipping Register STATUS_TEMP (0x7D) -not included in CRC
					Skipping Register STATUS_CML (0x7E) -not included in CRC
					Skipping Register STATUS_MFR_SPECIFIC_LTC 2978 (0x80) -not included in CRC
					Skipping Register READ_VIN (0x88) -not included in CRC
					Skipping Register READ_VOUT (0x8B) -not included in CRC
					Skipping Register READ_TEMPERATURE_1 (0x8D) -not included in CRC

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
					Skipping Register PMBUS_REVISION (0x98) - not included in CRC
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'D4BEE80F'	
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'F869D4EF'	
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'3ACF3AEB'	
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'C28E4A4A'	
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'34D07515'	
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'873D4AA1'	
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'C30FA827'	
'MFR_CONFIG_LTC2978'	'D0'	'80'	'00'	'82280FBF'	
'MFR_CONFIG_ALL_LTC2978'	'D1'	'7B'		'EDCF63DB'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'44A0749B'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'370A1736'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'F81896DA'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'906C569E'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'9F1D765B'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'12FE7AE9'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'CDBB771B'	
'MFR_FAULTBz0_PROPAGATE_LTC2978'	'D2'	'00'		'958057DC'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'334E5FE7'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'01279E23'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'42D24C26'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'5778486E'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'68B55CD0'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'08293456'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'2699826D'	
'MFR_FAULTBz1_PROPAGATE_LTC2978'	'D3'	'00'		'3829555C'	
'MFR_PWRGD_EN_LTC2978'	'D4'	'00'	'00'	'9519A26F'	
'MFR_FAULTB00_RESPONSE_LTC2978'	'D5'	'00'		'672B0310'	
'MFR_FAULTB01_RESPONSE_LTC2978'	'D6'	'00'		'457054A6'	

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
'MFR_FAULTB10_RESPONSE_LTC2978'	'D7'	'00'		'1BFB8AD5'	
'MFR_FAULTB11_RESPONSE_LTC2978'	'D8'	'00'		'F91A1D77'	
'MFR_VINEN_OV_FAULT_RESPONSE_LTC2978'	'D9'	'00'		'125B165B'	
'MFR_VINEN_UV_FAULT_RESPONSE_LTC2978'	'DA'	'00'		'97951CFF'	
'MFR_RETRY_DELAY_LTC2978'	'DB'	'20'	'F3'	'17278A2B'	
'MFR_RESTART_DELAY_LTC2978'	'DC'	'20'	'FB'	'944776CC'	
					Skipping Register MFR_VOUT_PEAK_LTC2978 (0xDD) -not included in CRC
					Skipping Register MFR_VIN_PEAK_LTC2978 (0xDE) -not included in CRC
					Skipping Register MFR_TEMPERATURE_PEAK_LTC2978 (0xDF) -not included in CRC
					Skipping Register MFR_DAC_LTC2978 (0xE0) -not included in CRC
'MFR_POWERGOOD_ASSERTION_DELAY_LTC2978'	'E1'	'20'	'EB'	'FC551924'	
'MFR_WATCHDOG_T_FIRST_LTC2978'	'E2'	'00'	'80'	'07DABC5F'	
'MFR_WATCHDOG_T_LTC2978'	'E3'	'00'	'80'	'F0CB873F'	
'MFR_PAGE_FF_MASK_LTC2978'	'E4'	'FF'		'7DCB07EE'	
					Skipping Register MFR_PADS_LTC2978 (0xE5) -not included in CRC
'MFR_I2C_BASE_ADDRESS_LTC2978'	'E6'	'5C'		'75B96D40'	
					Skipping Register MFR_SPECIAL_ID_LTC2978 (0xE7) -not included in CRC
					Skipping Register MFR_SPECIAL_LOT_LTC2978 (0xE8) -not included in CRC
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'3E6C02D6'	
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'D32E97EC'	
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'6E36A9A4'	
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'4216F64A'	

Register Name	command code	lo data byte	hi data byte (if word)	running CRC-32	Notes
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'E7276A9F'	
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'E04F58CD'	
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'216E1F0F'	
'MFR_VOUT_DISCHARGE_THRESHOLD_LTC2978'	'E9'	'00'	'C2'	'4014947C'	
					Skipping Register MFR_FAULT_LOG_STATUS_LTC2978 (0xED) -not included in CRC
					Skipping Register MFR_COMMON (0xEF) -not included in CRC
'MFR_SPARE0_LTC2978'	'F7'	'00'	'00'	'1DD6B650'	
					Skipping Register MFR_SPARE1_LTC2978 (0xF8) -not included in CRC
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'F4C7C13F'	
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'BD828208'	
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'4144D88D'	
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'94E9D341'	
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'47CFBA2E'	
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'627663F3'	
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'0560B0CB'	
'MFR_SPARE2_LTC2978'	'F9'	'00'	'00'	'71D92709'	
					Skipping Register MFR_SPARE3_LTC2978 (0xFA) -not included in CRC
					Skipping Register MFR_VOUT_MIN_LTC2978 (0xFB) -not included in CRC
					Skipping Register MFR_VIN_MIN_LTC2978 (0xFC) -not included in CRC
					Skipping Register MFR_TEMPERATURE_MIN_LTC2978 (0xFD) -not included in CRC