

# PSM Apps FAQs

PSM Applications

Exported on 07/23/2019

## Table of Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>PSM Apps Frequently Asked Questions (FAQs).....</b>   | <b>14</b> |
| 1.1      | How to use this FAQ .....  | 14        |
| 1.2      | Acknowledgement of Contributors.....   | 14        |
| 1.3      | User Last Update Edits Michael Holloway <a href="https://confluence.analog.com/display/~MHollowa">https://confluence.analog.com/display/~MHollowa</a> 54 days ago 7 Michael-A1 Jones <a href="https://confluence.analog.com/display/~MJones2">https://confluence.analog.com/display/~MJones2</a> 46 days ago 3 Michael Jones <a href="https://confluence.analog.com/display/~mjones">https://confluence.analog.com/display/~mjones</a> 1702 days ago 1 ..... | 15        |
| 1.4      | Recent FAQ Entries.....  | 0         |
| 1.5      | Most Recent FAQ Entries.....   | 15        |
| <b>2</b> | <b>Power System Managers .....</b>   | <b>17</b> |
| 2.1      | How Can I Quickly determine if a LTC297x device has a CRC memory Error? .....  | 17        |
| 2.1.1    | Symptoms: .....  | 17        |
| 2.1.2    | A Simple Screening Procedure using the Script Utility .....  | 17        |
| 2.2      | How Do I Configure LTC2977/2978 Odd Channels for Current Measurement?.....   | 18        |
| 2.3      | How Do I Configure Unused LTC297x Channels? .....  | 20        |
| 2.3.1    | How much current can the LTC297x VDD33 regulator supply? .....   | 22        |
| 2.4      | How Do VOUT_EN Pins Behave at Power-up? .....  | 22        |
| 2.5      | How much EEPROM does the LTC2977 have? .....   | 23        |
| 2.6      | How to Configure LTC297x to Only Monitor & Supervise Channels .....  | 23        |
| 2.7      | Is it OK to Float Unused Inputs? .....   | 24        |
| 2.8      | Is it possible that EEPROM can be written/read only with VDD33 applied (no Vpwr supply)? .....   | 25        |
| 2.9      | The Device reports a TON_MAX_FAULT, but the Rise Time of the Output is Fine. Why?..  | 25        |
| 2.10     | What replacement part is suggested for the PFET that powers LTC297x via USB dongle? .....  | 27        |
| <b>3</b> | <b>PSM Controllers.....</b>  | <b>28</b> |
| 3.1      | What causes the ADC Invalid bit in the MFR_PADS register to assert?.....   | 28        |
| 3.1.1    | What causes the ADC Invalid bit in the MFR_PADS register to assert?.....   | 28        |
| 3.1.1.1  | Retry Behavior.....  | 28        |
| 3.1.1.2  | LTC3880 Retry Behavior.....  | 28        |
| 3.1.1.3  | LTC3887 and other devices.....   | 28        |

|          |   |           |
|----------|---|-----------|
| 3.1.1.4  | LTM4676/A/B .....   | 28        |
| 3.1.1.5  | Practical Implications .....  | 28        |
| 3.2      | Can LTpowerPlay Calculate Total Rail Current for Polyphase Rails Utilizing Extra non-PMBus 'Slave' Phases? .....  | 29        |
| 3.3      | Controlled Off .....  | 30        |
| 3.4      | How can I Determine if a Device is a LTC3880 vs. LTC3880-1 .....  | 30        |
| 3.5      | How do I set up PGOOD? .....  | 30        |
| 3.6      | How do you read the ALERTB pin via PMBus? .....   | 30        |
| 3.7      | How long does it take for a PSM controller to reset? .....  | 31        |
| 3.8      | I have a need for a polyphase configuration that is not covered by the existing polyphase templates. What do I do? .....  | 31        |
| 3.9      | I Observe Bad Temp Readings. What are the Possible Causes? .....  | 33        |
| 3.10     | Misconnected Pins .....   | 33        |
| 3.10.1   | Symptoms .....  | 33        |
| 3.10.1.1 | Wrong Frequencies .....   | 33        |
| 3.11     | My PSM Controller is pulling ALERTB low. Is there a way to mask certain faults? .....   | 33        |
| 3.12     | Overlapping Rail Address .....  | 34        |
| 3.12.1   | Can you set the MFR_RAIL_ADDRESS to the effective device address? .....   | 34        |
| 3.13     | Storing user information in NVM .....   | 34        |
| 3.14     | The part is reporting a PLL_UNLOCK fault. What are the Possible Causes? .....   | 35        |
| 3.15     | What is the latency of a VOUT change? .....   | 35        |
| 3.16     | What is the real OC value? .....  | 35        |
| 3.17     | Why do I see a MFR (GPIOB_ASSERTED_LO) fault at power up? .....   | 36        |
| 3.18     | How do I set the temperature configuration when TSENSE pins are grounded on the LTC3882? .....  | 36        |
| <b>4</b> | <b>Supervisors .....</b>  | <b>37</b> |
| 4.1      | LTC2933 Apps FAQ .....  | 37        |
| 4.1.1    | For the HIGH RANGE setting on pin V1 the comparator threshold can be programmed as high at 15v, but the Absolute Maximum rating for pin V1 is 14v, can I drive 15v into pin V1 in HIGH RANGE? ..... | 37        |
| 4.1.1.1  | Answer: No Way! Always respect the absolute maximum ratings .....   | 37        |
| 4.1.2    | How much hysteresis do the LTC2933/36 input comparators have? .....   | 37        |
| 4.1.3    | If an application does not use GPI2 how should it be configured? .....  | 37        |
| 4.1.3.1  | Put the pin in a defined state and define its behavior. ....  | 37        |

|         |  |    |
|---------|--|----|
| 4.1.4   | What are the configuration register settings for the "typical application" on page1 of the data sheet? .....   | 38 |
| 4.1.4.1 | Register Settings: .....   | 38 |
| 4.1.5   | What is the difference between using a GPIO as RSTb and ALERTb?.....   | 39 |
| 4.1.5.1 | ALERTb is a pin behavior specified by the SMBUS spec. RSTb is a flexible reset function defined by the system.....   | 39 |
| 4.1.6   | What registers are stored in EEPROM with a STORE_USER command? .....   | 39 |
| 4.1.6.1 | Registers: .....   | 39 |
| 4.1.7   | Why can't LTpowerPlay write to the board after I load a project file? .....  | 39 |
| 4.1.7.1 | A: LTpowerPlay needs to "Go Online" before writing to the LTC2933 (or any other part).....   | 39 |
| 4.1.8   | Why does LTpowerPlay show data in reserved register bits?.....   | 40 |
| 4.1.8.1 | A:Reserved bits read-back unpredictable values. LTpowerPlay reports what it reads. ....  | 40 |
| 4.2     | LTC2937 Apps FAQ .....   | 40 |
| 4.2.1   | How do I monitor a negative supply? .....  | 40 |
| 4.2.2   | How do I use the fault log in the LTC2937? .....   | 41 |
| 4.2.2.1 | To READ the MONITOR_BACKUP register: .....   | 42 |
| 4.2.2.2 | To CLEAR the MONITOR_BACKUP register: .....  | 42 |
| 4.2.3   | If the I2C bus is not used, how do I tie the pins on the board? .....  | 42 |
| 4.2.4   | What do I do with unused channels on the LTC2937?.....   | 43 |
| 4.2.5   | What happens if I place an un-programmed (factory default) LTC2937 on my board? .....  | 44 |
| 4.2.6   | What is the "boot-up" time of the LTC2937? .....   | 45 |
| 4.2.7   | Why is Linduino having trouble communicating with the DC2313A on the I2C bus? .....  | 45 |
| 4.3     | LTC2936 Apps FAQ .....   | 45 |
| 4.3.1   | Why is Linduino having trouble communicating with my DC1605B on the I2C bus? .....   | 45 |
| 4.4     | "Add Default LTCXXXX Chip" button in LTpowerPlay does not add datasheet default register settings; why? .....  | 46 |
| 5       | LTpowerPlay FAQ.....   | 47 |
| 5.1     | Can I license LTpowerPlay if my computer is not connected to the internet? .....   | 47 |
| 5.2     | Can I have multiple versions of the GUI simultaneously installed on one computer? Can I 'snapshot' an old version of the GUI before updating so that I can run the old version OR the new one? ..... | 48 |
| 5.3     | How do I get detailed help for the selected register in LTpowerPlay?.....  | 49 |
| 5.3.1   | Example .....  | 49 |
| 5.3.1.1 | Selecting a Register and Launching Help: .....   | 49 |
| 5.3.1.2 | Results .....  | 49 |
| 5.3.2   | Register Information View .....  | 50 |

|         |  |    |
|---------|--|----|
| 5.4     | How do I translate settings shown in to LTpowerPlay into I2C command sequences?... 51            | 51 |
| 5.5     | Can LTpowerPlay Log Telemetry data to a file? .....  | 52 |
| 5.5.1   | A Note On .CSV Data Viewer Tools .....   | 53 |
| 5.5.1.1 | Microsoft DataSet Viewer .....   | 53 |
| 5.5.1.2 | LiveGraph .....  | 53 |
| 5.6     | Can I call LTpowerPlay from the command-line to program parts in system? .....                   | 54 |
| 5.7     | Can I use LTpowerPlay to run 'scripts' or sequences of I2C commands? .....                       | 55 |
| 5.7.1   | Recording a Script.....  | 55 |
| 5.7.1.1 | Enable Script Recording .....  | 55 |
| 5.7.1.2 | Recording Command Sequences.....   | 56 |
| 5.7.1.3 | Disable Script Recording .....   | 56 |
| 5.7.2   | Example Script File .....  | 56 |
| 5.7.3   | Playing Back a Script File.....  | 57 |
| 5.7.3.1 | Results .....  | 57 |
| 5.7.4   | Further Help .....   | 57 |
| 5.8     | Where do I download the Windows drivers for the DC1613A?.....                                    | 57 |
| 5.9     | Can I use LTpowerPlay to decode raw fault log bytes read by my host controller?.....             | 57 |
| 5.10    | How do I convert a project file containing multiple LTC2977 devices into LTM2987s? ...           | 59 |
| 5.11    | Can I use LTpowerPlay to Merge Two Project Files into one? .....                                 | 60 |
| 5.12    | What does this error message encountered immediately after installation mean? .....              | 60 |
| 5.12.1  | Problem: .....   | 60 |
| 5.12.2  | Solution: .....  | 61 |
| 5.13    | Can I use LTpowerPlay to compare two project (.proj) files? .....                                | 62 |
| 5.14    | How do I update LTpowerPlay on a machine without internet access?.....                           | 63 |
| 5.15    | Can I use my .license file on another computer?.....   | 63 |
| 5.16    | What does this error message, seen immediately after launching LTpowerPlay mean? .64             |    |
| 5.17    | How to I use LTpowerPlay to 'burn in' settings permanently into EEPROM.....                      | 65 |
| 5.18    | Is there a simple way to read, decode and save all the fault logs on my board at one time? ..... | 66 |
| 5.19    | My Views are Messed up. How can I recover to the default view configuration? .....               | 67 |
| 5.20    | I see LTpowerPlay in the taskbar, but I cannot see the window. How do I view the window? .....   | 68 |
| 5.21    | How do I Clear Individual Items In LTpowerplay?.....   | 69 |

|          |   |           |
|----------|---|-----------|
| 5.22     | Is There an Easy Way to Review Large Configurations.....  | 70        |
| 5.23     | I have a Register Command Code. Can LTpowerPlay tell me the Corresponding Register Name? .....              | 70        |
| 5.24     | How Can I Quickly Search for a Register by Name? .....  | 71        |
| 5.25     | How can a project file be used to program devices in a system.....  | 72        |
| 5.25.1   | How to Program .....  | 72        |
| 5.25.2   | Other Methods of Programming .....  | 73        |
| 5.25.3   | Programming Devices with CRC failures in the NVM/EEPROM.....  | 73        |
| 5.25.4   | Prerequisites .....   | 73        |
| 5.26     | How Can I Stop LTpowerPlay's Polling Loop, or make the Dongle's SCL/SDA Lines HiZ. 73                       |           |
| 5.27     | Can LTpowerPlay be run Without Administrative Privileges, or Under a Non-admin Account? .....               | 74        |
| 5.28     | Can I call LTpowerPlay from a batch file to program devices in-system? .....                                | 74        |
| 5.29     | When I click 'Add default LTCXXX Chip', why do the settings not match the datasheet/factory defaults? ..... | 75        |
| 5.30     | How do I get more information on the status/fault indicator XXXX? .....                                     | 76        |
| 5.31     | LTpowerPlay requires .NET Framework v2.0, and my computer does not have this version. How do I get it?..... | 79        |
| 5.31.1   | Method 1: Enable the .NET Framework 3.5 in Control Panel.....   | 79        |
| 5.31.2   | Method 2: Manually download and install the .NET Framework 3.5.....   | 79        |
| 5.32     | Can LTpowerPlay dump the contents of a device's EEPROM to a file?.....                                      | 79        |
| 5.33     | What is the syntax of scripts (command-files) in the I2C Utility? .....                                     | 80        |
| 5.34     | Why must I Manually Click 'Detect Chips' when Connected to a Customer Board?.....                           | 82        |
| <b>6</b> | <b>Device Programming FAQ.....</b>  | <b>83</b> |
| 6.1      | Can I program my PMBus Device over JTAG? .....  | 83        |
| 6.2      | Does BPM (or Arrow) Support my LTC device for programming? .....  | 84        |
| 6.3      | How do I export the correct programming files from LTpowerPlay?.....  | 84        |
| 6.4      | How do I hook-up a dongle to my customer board?.....  | 85        |
| 6.5      | How to Decode a Standard Intel Hex File .....   | 88        |
| 6.6      | How to power device In-Circuit without powering the entire system?.....                                     | 89        |
| 6.6.1    | How Do I Program Controllers without VIN .....  | 90        |
| 6.6.1.1  | How Do I Program Controllers without VIN? .....   | 90        |
| 6.6.1.2  | Manually Forcing the Address .....  | 90        |

|          |   |            |
|----------|---|------------|
| 6.6.1.3  | After the Address is Fixed.....   | 91         |
| 6.7      | What's the difference between In-Flight Update (IFU) & In-Circuit Programming (ICP)? .91  |            |
| 6.7.1    | In-Flight Update .....  | 91         |
| 6.7.2    | In-Circuit Programming.....   | 91         |
| 6.8      | What are the different CRC Options Available? .....                                       | 91         |
| 6.8.1    | Overview .....  | 91         |
| 6.8.2    | Internal CRC (1) .....  | 91         |
| 6.8.3    | Register CRC (2).....   | 92         |
| 6.8.4    | OEM CRC (3) .....   | 92         |
| 6.8.5    | SL CRC (4) .....  | 93         |
| 6.9      | What if I need more than 100mA from the USB PMBus Controller (i.e. DC1613A)? .....        | 93         |
| 6.10     | What is the format of the LT Programming File?.....                                       | 95         |
| 6.11     | What is the Programming Power Circuit? .....  | 95         |
| 6.12     | What options do I have for getting Programmed Devices? .....                              | 95         |
| 6.12.1   | Linear Technology Options .....   | 95         |
| 6.12.1.1 | LTExpress & Linear Technology SL#.....  | 95         |
| 6.12.2   | 3rd Party Programming of LTC Devices .....  | 96         |
| 6.12.2.1 | BPM.....  | 96         |
| 6.12.2.2 | Arrow .....   | 96         |
| 6.12.2.3 | Asset Intertech .....   | 96         |
| 6.12.2.4 | JTAG Technologies.....  | 96         |
| 6.12.2.5 | System General – ** NOTE NOT RECOMMENDED ** .....   | 96         |
| 6.13     | What Programming Option Should I choose? .....  | 96         |
| 6.14     | Where does LTpowerPlay store it's computed CRC in the part? .....                         | 96         |
| 6.14.1   | Example: .....  | 97         |
| 6.15     | Where do I find In Flight Update Information? .....                                       | 99         |
| 6.16     | Why Must all Devices Share a Common Base Address for In System Programming to Work? ..... | 99         |
| <b>7</b> | <b>Firmware FAQ.....</b>  | <b>100</b> |
| 7.1      | Can I print fault log data generated by my firmware? .....                                | 100        |
| 7.2      | Do I Need Polling? .....  | 100        |
| 7.2.1    | Background .....  | 100        |
| 7.2.2    | Solution .....  | 100        |

|         |   |     |
|---------|---|-----|
| 7.2.3   | Alternatives .....  | 101 |
| 7.3     | How are Voltages and Current Represented in PMBus Commands? ..... | 101 |
| 7.3.1   | How are Voltages and Current Represented in PMBus Commands? ..... | 101 |
| 7.3.1.1 | Basic Representation .....  | 101 |
| 7.3.1.2 | L11 Representation .....  | 102 |
| 7.3.1.3 | L16 Representation .....  | 102 |
| 7.3.1.4 | Conversions .....   | 103 |
| 7.3.1.5 | Conversions in Code .....   | 104 |
| 7.3.1.6 | Tricks .....  | 105 |
| 7.3.1.7 | Questions.....  | 105 |
| 7.4     | How Can I Extend the Retention Time of PSM Devices .....          | 105 |
| 7.4.1   | Background .....  | 105 |
| 7.4.2   | How EEPROM is Programmed .....                                    | 105 |
| 7.4.3   | Extending Retention .....   | 106 |
| 7.4.4   | Practical Implementation Concerns .....                           | 106 |
| 7.4.4.1 | STORE_USER_ALL.....   | 106 |
| 7.4.4.2 | MFR_EE_DATA.....  | 106 |
| 7.4.5   | In Flight Update (aka In System Programming) .....                | 107 |
| 7.4.6   | Design Considerations .....                                       | 107 |
| 7.4.7   | Closing .....   | 107 |
| 7.5     | How can I write maintenance code without polling? .....           | 107 |
| 7.5.1   | How can I write maintenance code without polling? .....           | 107 |
| 7.5.2   | Maintenance mode .....  | 108 |
| 7.5.3   | What is the generic time behavior of PSM devices? .....           | 108 |
| 7.5.3.1 | Managers .....  | 108 |
| 7.5.3.2 | Controllers.....  | 108 |
| 7.5.4   | Constraints that lead to deterministic behavior .....             | 108 |
| 7.5.4.1 | Managers .....  | 109 |
| 7.5.4.2 | Controllers.....  | 109 |
| 7.5.4.3 | Universal Constraints (Manager/Controller) .....                  | 109 |
| 7.5.5   | Memory operations under universal constraints .....               | 109 |
| 7.5.6   | Speeding things up .....  | 110 |
| 7.5.7   | Side effects .....  | 110 |
| 7.5.8   | Special Cases.....  | 110 |
| 7.5.8.1 | Storing with OPERATION on .....                                   | 110 |



|         |   |     |
|---------|---|-----|
| 7.5.9   | Other constraints (non maintenance mode).....                     | 110 |
| 7.5.10  | Alternatives .....  | 111 |
| 7.5.11  | Summary .....   | 111 |
| 7.6     | How do I know when LTC388X STORE_USER_ALL is complete? .....      | 111 |
| 7.6.1   | How do I know when LTC388X STORE_USER_ALL is complete?.....       | 111 |
| 7.6.1.1 | Wait For ACK.....   | 111 |
| 7.6.1.2 | Wait for Not BUSY.....  | 112 |
| 7.6.1.3 | Wait For EEPROM/NVM Complete .....                                | 113 |
| 7.6.1.4 | Putting It All Together.....                                      | 113 |
| 7.7     | Linduino FAQ.....   | 114 |
| 7.7.1   | Can I Read and Decode Fault Logs in Firmware? .....               | 114 |
| 7.7.1.1 | Background .....  | 114 |
| 7.7.1.2 | Methods of Extraction.....  | 114 |
| 7.7.1.3 | Decoding.....   | 115 |
| 7.7.1.4 | Storing Data .....  | 115 |
| 7.7.1.5 | Printing Data .....   | 115 |
| 7.7.1.6 | Cautions.....   | 115 |
| 7.7.1.7 | Summary .....   | 115 |
| 7.7.2   | How Do I Connect a Linduino to a PSM DC? .....                    | 115 |
| 7.7.3   | Is the Code Open Source? .....                                    | 115 |
| 7.7.4   | What about the Galileo Board? .....                               | 115 |
| 7.7.5   | What Arduino Boards Work?.....                                    | 116 |
| 7.7.6   | What If I Want to Use Other I2C DC At The Same Time? .....        | 116 |
| 7.7.7   | What Sketch's are Available?.....                                 | 116 |
| 7.8     | MFR_SPECIAL_ID Handling.....                                      | 116 |
| 7.8.1   | ID Format.....  | 116 |
| 7.8.2   | Coding Practice .....   | 116 |
| 7.8.3   | Coding Example .....  | 116 |
| 7.8.4   | How can firmware identify a LTC2978 without MFR_SPECIAL_ID? ..... | 117 |
| 7.8.4.1 | How can firmware identify a LTC2978 without MFR_SPECIAL_ID? ..... | 117 |
| 7.9     | What Example Firmware is Available from LTC? .....                | 118 |
| 7.9.1   | Linduino.....   | 118 |
| 7.9.2   | Linux Drivers.....  | 119 |
| 7.10    | What is In Flight Update and How do I use it? .....               | 119 |

|          |  |            |
|----------|--|------------|
| 7.10.1   | In Flight Update Code .....  | 119        |
| 7.10.2   | How In Flight Update Works .....   | 119        |
| 7.10.3   | How Many Devices Does it Program at a Time? .....                                  | 119        |
| 7.10.4   | How Do You Port In Flight Update? .....  | 119        |
| 7.10.5   | How Do You Debug In Flight Update? .....   | 120        |
| 7.10.6   | What is the .ISP File Format? .....  | 120        |
| 7.10.7   | LTC3882 Pre-Release Si Compatibility.....  | 120        |
| 7.11     | What options do I have to communicate with with PSM devices via PMBus? .....       | 120        |
| 7.11.1   | LTpowerPlay Scripting .....  | 120        |
| 7.11.2   | Linduino.....  | 121        |
| 7.11.3   | Linux .....  | 121        |
| <b>8</b> | <b>What's This Fault? .....</b>  | <b>122</b> |
| 8.1      | TBD:.....  | 0          |
| 8.1.1    | MS Only:.....  | 0          |
| 8.1.2    | DP Only: .....   | 0          |
| 8.1.3    | Potentially DP and MS: .....   | 0          |
| 8.2      | CML_INVALID_COMMAND .....  | 122        |
| 8.2.1    | Possible Causes:.....  | 122        |
| 8.2.2    | Scope Debug .....  | 122        |
| 8.3      | CML_INVALID_DATA .....   | 122        |
| 8.3.1    | The device received improper data for a supported command.....                     | 122        |
| 8.3.1.1  | Possible Causes:.....  | 122        |
| 8.3.1.2  | Remedies and Workarounds .....   | 123        |
| 8.4      | CML_OTHER_COM_FAULT .....  | 123        |
| 8.4.1    | The device detected traffic on the bus that does not match expected protocols..... | 123        |
| 8.4.1.1  | Possible Causes:.....  | 123        |
| 8.4.1.2  | Remedies and Workarounds .....   | 124        |
| 8.4.1.3  | Other Debugging Tips .....   | 124        |
| 8.5      | CML_PEC_FAILED .....   | 124        |
| 8.5.1    | Possible Causes:.....  | 124        |
| 8.5.2    | Remedies and Workarounds .....   | 124        |
| 8.5.3    | Scope Debug .....  | 125        |
| 8.6      | DAC_SATURATED .....  | 125        |
| 8.6.1    | Possible Causes:.....  | 125        |

|         |   |     |
|---------|---|-----|
| 8.6.2   | Remedies and Workarounds .....  | 125 |
| 8.6.3   | Other Debugging Tips .....  | 126 |
| 8.7     | DISCHARGE_FAULT .....   | 126 |
| 8.7.1   | Example Fault Scenario .....  | 126 |
| 8.7.2   | Possible Causes: .....  | 128 |
| 8.7.3   | Remedies and Workarounds .....  | 128 |
| 8.7.4   | Other Debugging Tips .....  | 128 |
| 8.7.5   | Check for glitches (quick toggles) on the CONTROL pin. If you have an oscilloscope, perhaps the fastest way to insight is to: .....   | 128 |
| 8.8     | FACTORY_TRIM_CRC_FAULT .....  | 129 |
| 8.8.1   | The device has an internal inconsistency in its EEPROM. This is often an indication the power was removed during a STORE operation or the device was incorrectly programmed. .... | 129 |
| 8.8.1.1 | Possible Causes: .....  | 129 |
| 8.8.1.2 | Remedies and Workarounds .....  | 129 |
| 8.9     | FAULT0_IN .....   | 129 |
| 8.9.1   | Device Behavior in a Shared Fault Scenario .....  | 130 |
| 8.9.2   | Possible Causes: .....  | 131 |
| 8.9.3   | Remedies and Workarounds .....  | 131 |
| 8.9.4   | Other Debugging Tips .....  | 131 |
| 8.9.5   | Latch Off Shared Fault Response .....   | 131 |
| 8.10    | FAULT1_IN .....   | 133 |
| 8.10.1  | Device Behavior in a Shared Fault Scenario .....  | 133 |
| 8.10.2  | Possible Causes: .....  | 134 |
| 8.10.3  | Remedies and Workarounds .....  | 134 |
| 8.10.4  | Other Debugging Tips .....  | 135 |
| 8.11    | FAULT_LOG_PRESENT .....   | 135 |
| 8.11.1  | Possible Causes: .....  | 135 |
| 8.11.2  | Tips .....  | 135 |
| 8.11.3  | Other Debugging Tips .....  | 136 |
| 8.12    | GPIOB_ASSERTED_LO .....   | 136 |
| 8.12.1  | Device Behavior in a Shared Fault Scenario .....  | 136 |
| 8.12.2  | Possible Causes: .....  | 137 |
| 8.12.3  | Remedies and Workarounds .....  | 137 |
| 8.12.4  | Other Debugging Tips .....  | 138 |
| 8.13    | INPUT_OFF .....   | 138 |

|          |  |     |
|----------|--|-----|
| 8.13.1   | Device Behavior for VIN and SHARE_CLK .....  | 138 |
| 8.13.2   | Possible Causes:.....  | 140 |
| 8.13.3   | Remedies and Workarounds .....   | 140 |
| 8.13.4   | Other Debugging Tips .....   | 140 |
| 8.14     | IOUT_OC_FAULT .....  | 140 |
| 8.14.1   | Possible Causes:.....  | 140 |
| 8.14.2   | Remedies and Workarounds .....   | 140 |
| 8.14.3   | Other Debugging Tips .....   | 141 |
| 8.15     | OFF .....  | 141 |
| 8.15.1   | Possible Causes:.....  | 141 |
| 8.16     | OT_FAULT .....   | 141 |
| 8.16.1   | Possible Causes:.....  | 141 |
| 8.16.2   | Remedies and Workarounds .....   | 141 |
| 8.16.3   | Other Debugging Tips .....   | 142 |
| 8.17     | PLL_UNLOCKED .....   | 142 |
| 8.17.1   | The PLL_UNLOCKED status bit is set because the PWM could not lock on the external clock supplied to the SYNC pin. ....             | 142 |
| 8.17.2   | Possible Causes:.....  | 142 |
| 8.17.2.1 | Remedies and Workarounds .....   | 142 |
| 8.17.2.2 | Other Debug Tips .....   | 142 |
| 8.18     | SHARE_CLK_LOW .....  | 142 |
| 8.18.1   | Device Behavior for VIN and SHARE_CLK .....  | 143 |
| 8.18.2   | Possible Causes:.....  | 144 |
| 8.18.3   | Remedies and Workarounds .....   | 144 |
| 8.18.4   | Other Debugging Tips .....   | 144 |
| 8.19     | TON_MAX_FAULT .....  | 145 |
| 8.19.1   | A TON_MAX fault was detected because the VSENSE voltage did not reach the VOUT_UV_FAULT_LIMIT before the TON_MAX_FAULT_LIMIT. .... | 145 |
| 8.19.1.1 | Possible Causes:.....  | 145 |
| 8.19.1.2 | Remedies and Workarounds .....   | 145 |
| 8.19.1.3 | Other Debugging Tips .....   | 145 |
| 8.20     | UT_FAULT .....   | 146 |
| 8.20.1   | Possible Causes:.....  | 146 |
| 8.20.2   | Remedies and Workarounds .....   | 146 |
| 8.20.3   | Other Debugging Tips .....   | 146 |

|          |   |            |
|----------|---|------------|
| 8.21     | VOUT_OV_FAULT .....                                     | 146        |
| 8.21.1   | Possible Causes:.....                                   | 147        |
| 8.21.2   | Remedies and Workarounds .....                          | 147        |
| 8.21.3   | Other Debugging Tips .....                              | 147        |
| 8.22     | VOUT_UV_FAULT .....                                     | 147        |
| 8.22.1   | Possible Causes:.....                                   | 148        |
| 8.22.2   | Remedies and Workarounds .....                          | 148        |
| 8.22.3   | Other Debugging Tips .....                              | 148        |
| <b>9</b> | <b>Design .....</b>                                     | <b>149</b> |
| 9.1      | How do you design the addressing of a PSM system? ..... | 149        |
| 9.1.1    | How do you design the addressing of a PSM system? ..... | 149        |

# 1 PSM Apps Frequently Asked Questions (FAQs)

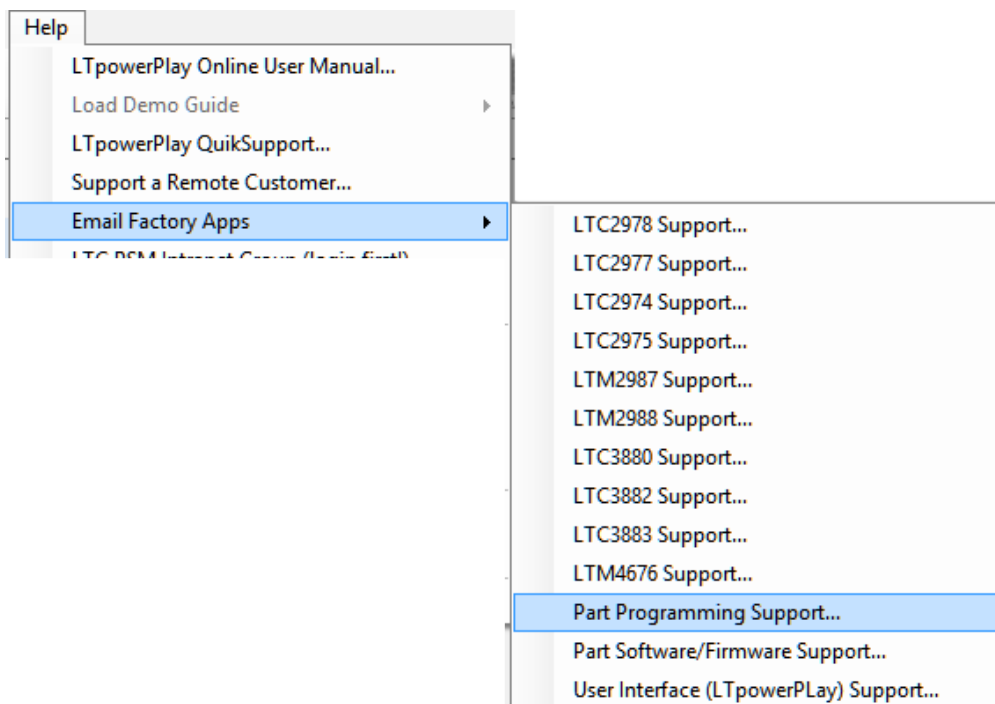
## 1.1 How to use this FAQ

This Document Contains PSM Applications Support Frequently Asked Questions Organized into Topic Areas. This document lists the common questions by topic area in a hierarchy or tree. There are a couple of ways to use this document:

- Just search for keywords (CTRL+F, and type search term)
- **New Folks:** If you are not yet familiar with the document, you can use the index on the first pages of this document to see a tree hierarchy complete listing of all content, with page numbers and hyperlinks to specific questions
  - You can browse the index by topic and
  - Click the question of interest to navigate to that specific question/answer
- **Frequent Flyers:** If you are already familiar with this document, consult the 'Most Recent FAQ Entries' list below to scan for recently added content.

*NOTE: Please forgive formatting errors, as this document is automatically generated from an internal website and as such may not be formatted perfectly. If you have questions about specific products or topics, please use the Help menu in LTpowerPlay to submit your question to the appropriate support person:*

If you have questions that this FAQ does not yet answer, please use the Help menu within LTpowerPlay to send email to the specific group that supports your topic area:



## 1.2 Acknowledgement of Contributors

This FAQ document is a work in progress. Multiple authors have contributed to the content. Below is a listing of the author contributions:


### 1.3 User Last Update Edits

|                               |               |   |
|-------------------------------|---------------|---|
| Michael Holloway <sup>1</sup> | 54 days ago   | 7 |
| Michael-A1 Jones <sup>2</sup> | 46 days ago   | 3 |
| Michael Jones <sup>3</sup>    | 1702 days ago | 1 |

### 1.4 Most Recent FAQ Entries


This is primarily intended for "Frequent Flyers" to provide a quick listing of the most recently added or edited content. Only the 20 most recent changes are shown:


#### Recently Updated




(see page 14)

Michael-A1 Jones<sup>4</sup>


 FAQ<sup>5</sup> created Jun 07, 2019


 What Example Firmware is Available from LTC?(see page 118) updated Jun 07, 2019 • view change<sup>6</sup>





(see page 14)

Michael Holloway<sup>7</sup>

 Firmware FAQ(see page 100) updated May 30, 2019 • view change<sup>8</sup>

 LTpowerPlay FAQ(see page 47) updated May 30, 2019 • view change<sup>9</sup>









 Supervisors(see page 37) updated May 30, 2019 • view change<sup>10</sup>



- 1 <https://confluence.analog.com/display/~MHolloway>
- 2 <https://confluence.analog.com/display/~MJones2>
- 3 <https://confluence.analog.com/display/~mjones>
- 4 <https://confluence.analog.com/display/~MJones2>
- 5 <https://confluence.analog.com/display/APCB/FAQ>
- 6 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543858&selectedPageVersions=2&selectedPageVersions=3>
- 7 <https://confluence.analog.com/display/~MHolloway>
- 8 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543844&selectedPageVersions=4&selectedPageVersions=5>
- 9 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543876&selectedPageVersions=5&selectedPageVersions=6>
- 10 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543840&selectedPageVersions=2&selectedPageVersions=3>



Anonymous

-  [PSM Controllers](#)(see page 28) updated May 30, 2019 • [view change](#)<sup>11</sup>
-  [Power System Managers](#)(see page 17) updated May 30, 2019 • [view change](#)<sup>12</sup>
-  [LTpowerPlay FAQ](#)(see page 47) updated Dec 17, 2014 • [view change](#)<sup>13</sup>
-  [Firmware FAQ](#)(see page 100) updated Dec 17, 2014 • [view change](#)<sup>14</sup>
-  [Supervisors](#)(see page 37) updated Dec 17, 2014 • [view change](#)<sup>15</sup>
-  [PSM Controllers](#)(see page 28) updated Dec 17, 2014 • [view change](#)<sup>16</sup>
-  [Power System Managers](#)(see page 17) updated Dec 17, 2014 • [view change](#)<sup>17</sup>
-  [What Example Firmware is Available from LTC?](#)(see page 118) updated Dec 17, 2014 • [view change](#)<sup>18</sup>

---

11 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543843&selectedPageVersions=4&selectedPageVersions=5>

12 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543841&selectedPageVersions=5&selectedPageVersions=6>

13 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543876&selectedPageVersions=4&selectedPageVersions=5>

14 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543844&selectedPageVersions=3&selectedPageVersions=4>

15 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543840&selectedPageVersions=1&selectedPageVersions=2>

16 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543843&selectedPageVersions=3&selectedPageVersions=4>

17 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543841&selectedPageVersions=2&selectedPageVersions=3>

18 <https://confluence.analog.com/pages/diffpagesbyversion.action?pagelId=94543858&selectedPageVersions=1&selectedPageVersions=2>



## 2 Power System Managers

This FAQ Section contains Frequently Asked Questions about Power System Managers

### 2.1 How Can I Quickly determine if a LTC297x device has a CRC memory Error?

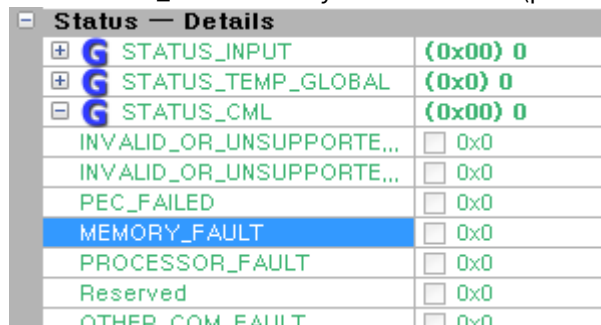
#### Procedure to Identify LTC297x with CRC Error

The following provides a method to quickly screen boards that may contain devices with EEPROM CRC errors. This procedure must be run before re-programming parts on the board to identify this problem. A 'pass' of the below procedure confidently rules out the possibility of a CRC failure. A 'fail' indicates that further analysis may be required – contact LT field apps before removing the devices or re-programming them.

#### 2.1.1 Symptoms:

An LTC297x device may exhibit a failure signature at power-up, resulting in the channels not powering up. It is also possible that one of the devices is missing from LTpowerPlay's system tree, in which case the device's address has changed.

1. Recover the base address in RAM. If a device has a CRC error, it may have lost its base address.
  - a. Perform a byte write to address 0x5B, command code 0xE6, data byte equal to the value of the common base address, for example 0x5C
2. For each device in the system:
  - a. Read STATUS\_CML and verify that bit 4 is zero (passing device)



| Status — Details         |   |
|--------------------------|---|
| STATUS_INPUT             | (0x00) 0                                |
| STATUS_TEMP_GLOBAL       | (0x0) 0                                 |
| STATUS_CML               | (0x00) 0                                |
| INVALID_OR_UNSUPPORTE... | <input type="checkbox"/> 0x0            |
| INVALID_OR_UNSUPPORTE... | <input type="checkbox"/> 0x0            |
| PEC_FAILED               | <input type="checkbox"/> 0x0            |
| MEMORY_FAULT             | <input checked="" type="checkbox"/> 0x0 |
| PROCESSOR_FAULT          | <input type="checkbox"/> 0x0            |
| Reserved                 | <input type="checkbox"/> 0x0            |
| OTHER_COM_FAULT          | <input type="checkbox"/> 0x0            |

#### 2.1.2 A Simple Screening Procedure using the Script Utility

The simplest way to screen a board is to build an I2C script and use LTpowerPlay to run the script on the board to check it for CRC errors. Once you've built a script file for the board, you can connect LTpowerPlay to a board and check it with a single button press. To check that your script works as expected, it is recommended to run the script on a known good board. The procedure to do this is outlined below.

There are two critical issues to be aware of:

- You must modify the base address to whatever is appropriate for the board, a base address that will ultimately resolve to the addresses in the project.
- You must have the correct resolved addresses (base + offset).

#### PROCEDURE

- Open Notepad or other text editor and paste the following into the editor:

```
# The first instruction recovers the base address. Modify the value 0x5C
# to the appropriate base address for your project/board.
0x5B,WB,0xE6,0x5C,# Write MFR_I2C_BASE_ADDRESS of all LTC PMbus devices on
# the bus.
#
# Instructions for building this script:
# Copy the below line starting with "0x5C," once for each LTC PMbus device
# in your system.
# Then change the address in the first column (0x5C) to the appropriate
# device address.
# There should be one line per device in your system to check each device
# for CRC errors.
0x5C,RB,0x7E,0x00,0x10 #Read STATUS_CML memory fault bit
0x5D,RB,0x7E,0x00,0x10 #Read STATUS_CML memory fault bit
0x5E,RB,0x7E,0x00,0x10 #Read STATUS_CML memory fault bit
```

- Follow the instructions in the file, creating one STATUS\_CML check line per device in your system. For each line, modify the address (first byte) to reflect the resolved device address. The script shown above is an example script. All three devices use the same base address with offsets of 0 (0x5C), 1 (0x5D), and 2 (0x5E).
- Save the file as board\_check.txt
- When you are done editing the file, you should have one of these lines per device in your system.
- Connect the DC1613A to your board
- Run the I2C script in LTpowerPlay to check the board:
  - Select “Utilities > I2C Utility...”
  - Select the “Command Files” tab
  - Click “...” and Browse to the board\_check.txt file
  - **UNCHECK the box titled “Substitute Selected Chip”**
  - Click ‘Run Cmd File’
- If the test succeeds, the windows will be green indicating there are no CRC failures
  - If the window text is red, the board may have CRC failures. Contact LT for further analysis.
- To test another board, connect the DC1613A to the next board and click “Run Cmd File” again.

## 2.2 How Do I Configure LTC2977/2978 Odd Channels for Current Measurement?

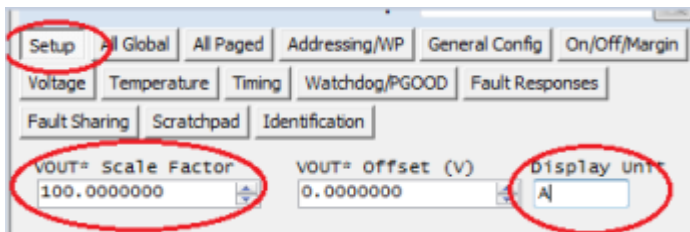
The LTC2977/78 devices have the ability to be configured to measure current on odd-numbered channels: CH1, CH3, CH5, CH7. Only odd channels may be configured for measuring current. To configure for current, the channel must be in hi-res mode (MFR\_CONFIG\_LTC2978, bit9). The VSENSEP and VSENSEM pins may be connected across an inductor (DCR) or resistive sense (Rsense) element. Even channels do not support this feature and the VSENSEM pin (CH0/2/4/6) must remain within 100mV of GND.

Unfortunately the LTC2977/78 devices do not have a current readout register or a register to store the DCR or Rsense value. Instead, use the READ\_VOUT command to get raw voltage readings. The system host can be used to calculate the current based on this reading divided by the sense resistor value. These values are given in L11 format and the units are mV.

| General Configuration Registers  |  |
|--|--|
| <input checked="" type="checkbox"/> MFR_CONFIG_LTC2978 (0x0280) Expand for Detail... |  |
| RESERVED15   | <input type="checkbox"/> 0x0   |
| RESERVED14   | <input type="checkbox"/> 0x0   |
| RESERVED13   | <input type="checkbox"/> 0x0   |
| RESERVED12   | <input type="checkbox"/> 0x0   |
| fast_servo_off   | <input type="checkbox"/> 0x0 (fast-servo enabled)                      |
| supervisor_resolution  | <input type="checkbox"/> 0x0 (Supervisor is HI-RES)                    |
| adc_hires  | <input checked="" type="checkbox"/> 0x1 (ADC is HI-RES)                |
| controln_sel   | <input type="checkbox"/> 0x0 (Control0 is Selected)                    |
| servo_continuous   | <input checked="" type="checkbox"/> 0x1 (Continuously servo VOUT to ta |

In this mode, the ONLY function that this channel serves is telemetry readback of the current. The *adc\_hires* bit disables the VOUT\_EN pin, and disables all fault responses. Essentially it forces the channel to be 'off' as far as the LTC2977 is concerned, and it only reads back the voltage in mV across the sense element.

LTpowerPlay has a feature that conveniently translates this mV reading to a current readback value in mA. There is a scale factor that can be used to generate an adjusted value in the READ\_VOUT register. This is accessed by clicking the Setup tab in the Config window.



The value entered into the VOUT Scale Factor box should be equal to  $1/R_{sense}$ . There is a Display Units field that can be changed from volts to amps by replacing 'V' with 'A'. These changes allow the readout to display a computed current that is consistent with the actual current based on the sense resistance in the circuit. For example, if the  $R_{sense}$  is 10 milliohm (0.01 ohm), the VOUT Scale Factor is 100. The READ\_VOUT register will now report a value in mA that reflects 100mA for every mV that is measured by the chip. In this example, a 0.5A load was applied to a supply rail with  $R_{sense}$  of 10mohm, and the chip was measuring 5mV.

| Telemetry — Output Voltage (V) |             |
|--------------------------------|-------------|
| MFR_VOUT_PEAK_LTC              | +0.000 A    |
| READ_VOUT                      | +501.875 mA |
| MFR_VOUT_MIN_LTC               | +0.000 A    |

Since the differential voltage ( $V_{SENSEPN} - V_{SENSEMN}$ ) is limited to  $\pm 170\text{mV}$ , the sense element must be chosen so that the IR drop does not exceed this limit. The common mode voltage of these pins is allowed up to 6V. For example, if the current is expected to be in the 2A range, a sense resistor of 50 milliohms provides a 150mV of voltage to the ADC and allows excursions to 3.4A. See datasheet p. 84 for further details.

*Note: No OV or UV faults or warnings are reported in this mode. The VOUT\_EN pin will assert low in this mode and cannot be used to control a DC/DC converter. The VDACP output pin is also unavailable.*

### Registers/Commands:

MFR\_CONFIG\_LTC2977 (0xD0)

READ\_VOUT (0x8B)

## 2.3 How Do I Configure Unused LTC297x Channels?

Some designs do not utilize all channels on LTC297x devices. This document covers both hardware and configuration recommendations for unused channels.

### HARDWARE

From a hardware point of view, unused ADC sense inputs (VSENSE $P_n$ , VSENSE $M_n$ , ISENSE $P_n$  or ISENSE $M_n$ ) must be tied to GND. In a system where the inputs are connected to removable cards and may be left floating in certain situations, connect the inputs to GND using 100k resistors. Place the 100k resistors before any filter components to prevent loading of the filter. The temperature sense inputs (TSENSE $n$ ) may be left floating, as they are not associated with channels. The temperature reported on floating TSENSE $n$  inputs will be the internal die temperature (READ\_TEMPERATURE\_2).

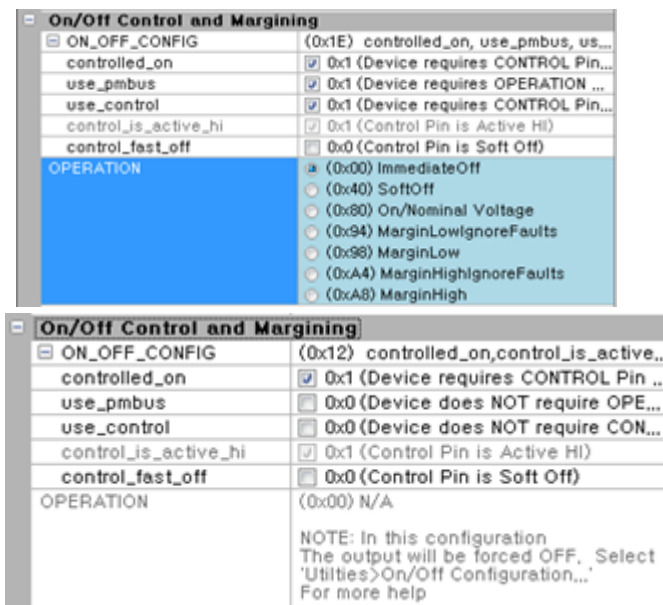
### CONFIGURATION

Summary checklist for the registers:

- Configure OPERATION register to 'Immediate Off' (0x00)
- Configure ON\_OFF\_CONFIG to a value of 0x12 (never power on)
- Program the DAC to 'disconnected'
- Disable Fault Sharing propagation
- Program Fault Responses to 'ignore'
- Exclude from POWER\_GOOD\_EN mask
- Exclude from PAGE\_FF mask

### OPERATION and ON\_OFF\_CONFIG

In LTpowerPlay, select the unused channel in the system tree then click the ON/OFF/MARGIN tab. Select 'ImmediateOff' for the OPERATION command. Uncheck the *use\_pmbus* and *use\_control* bits. The ON\_OFF\_CONFIG hex value should now read 0x12.



NOTE: Once the ON\_OFF\_CONFIG register is set to 0x12, the OPERATION register is a “don’t care”. However system software might inadvertently change the ON\_OFF\_CONFIG register to listen to *pm\_bus* or the *control* pin. In this case, it's best to set the OPERATION register to ‘ImmediateOff’ (0x00) as an extra layer of protection.

Select the All Paged tab and expand the MFR\_CONFIG\_LTC297x register. Select ‘DAC Disconnected’ mode on each unused channel.

|                  |   |
|------------------|---|
| servo_continuous | <input checked="" type="radio"/> 0x1 (Continuously servo VOUT to target)  |
| servo_on_warn    | <input type="radio"/> 0x0 (Do NOT allow the unit to re-servo w.   |
| dac_mode         | <input type="radio"/> 0x0 (DAC Soft Connect)<br><input checked="" type="radio"/> 0x1 (DAC Disconnected)<br><input type="radio"/> 0x2 (DAC Manual w/ Hard Connect)<br><input type="radio"/> 0x3 (DAC Manual w/ Soft Connect) |
| vofn when en     | <input type="checkbox"/> 0x0 (VOFN driver is tri-stated when ch is  |

### EXCLUDE PAGE\_FF and POWER\_GOOD

Select the Addressing/WP tab. If you want to exclude the channel from the MFR\_PAGE\_FF\_MASK (0xE4) command, unchecking the channel bit in this register allows the channel to ignore PAGE=0xFF commands to the device.

|   |  |
|---|--|
| <input checked="" type="checkbox"/> MFR_PAGE_FF_MASK... | (0x3F) chan5, chan4, chan3, chan2, che                       |
| chan7   | <input type="checkbox"/> 0x0 (Ignore PAGE=FF)                |
| chan6   | <input type="checkbox"/> 0x0 (Ignore PAGE=FF)                |
| chan5   | <input checked="" type="checkbox"/> 0x1 (Respond to PAGE=FF) |
| chan4   | <input checked="" type="checkbox"/> 0x1 (Respond to PAGE=FF) |
| chan3   | <input checked="" type="checkbox"/> 0x1 (Respond to PAGE=FF) |
| chan2   | <input checked="" type="checkbox"/> 0x1 (Respond to PAGE=FF) |
| chan1   | <input checked="" type="checkbox"/> 0x1 (Respond to PAGE=FF) |
| chan0   | <input checked="" type="checkbox"/> 0x1 (Respond to PAGE=FF) |

Select the Watchdog/PGOOD tab. To exclude the channel from the PWRGD logic, uncheck the channel bit in the MFR\_PWRGD\_EN register.

|  |   |
|--|---|
| <input checked="" type="checkbox"/> Power Good           |   |
| <input checked="" type="checkbox"/> MFR_PWRGD_EN_LTC2... | (0x03F) chan5, chan4, chan3, chan2, che                     |
| wdog   | <input type="checkbox"/> 0x0 (Don't Affect POWER_GOOD)      |
| chan7  | <input type="checkbox"/> 0x0 (Don't Affect POWER_GOOD)      |
| chan6  | <input type="checkbox"/> 0x0 (Don't Affect POWER_GOOD)      |
| chan5  | <input checked="" type="checkbox"/> 0x1 (Affect POWER_GOOD) |
| chan4  | <input checked="" type="checkbox"/> 0x1 (Affect POWER_GOOD) |
| chan3  | <input checked="" type="checkbox"/> 0x1 (Affect POWER_GOOD) |
| chan2  | <input checked="" type="checkbox"/> 0x1 (Affect POWER_GOOD) |
| chan1  | <input checked="" type="checkbox"/> 0x1 (Affect POWER_GOOD) |
| chan0  | <input checked="" type="checkbox"/> 0x1 (Affect POWER_GOOD) |

### IGNORE FAULT RESPONSES and FAULT SHARING

Select the unused channel in the system tree then click the Fault Responses tab. Set all fault responses for unused channels to ‘ignore’.

|  |               |
|--|---------------|
| <input checked="" type="checkbox"/> Fault Responses — Output Voltage |               |
| <input checked="" type="checkbox"/> TON_MAX_FAULT_RESPONSE           | (0x38) Ignore |
| <input checked="" type="checkbox"/> VOUT_UV_FAULT_RESPONSE           | (0x3F) Ignore |
| <input checked="" type="checkbox"/> VOUT_OV_FAULT_RESPONSE           | (0x38) Ignore |

Click the Fault Responses tab. Set all fault propagate bits to ‘Don’t Propagate’ (0x0).

|   |  |
|---|--|
| <input checked="" type="checkbox"/> Fault Sharing — Propagation |  |
| MFR_FAULTBz0_PROPAGATE_LTC2978                                  | <input type="checkbox"/> 0x0 (Don't Propagate) |
| MFR_FAULTBz1_PROPAGATE_LTC2978                                  | <input type="checkbox"/> 0x0 (Don't Propagate) |

### 2.3.1 How much current can the LTC297x VDD33 regulator supply?

The VDD33 regulator is guaranteed to provide at least 75mA of load current, typically 90mA. The VDD33 supply internally draws 13mA ( $I_{VDD33}$ ) worst case and is typically 10mA. The external current draw may come from pullup resistors, LEDs, or other peripheral circuitry. If the user places external loads that amount to, say, 10mA, then this should be added to the internal 13mA load. The total load current is 23mA (typ) in this case.

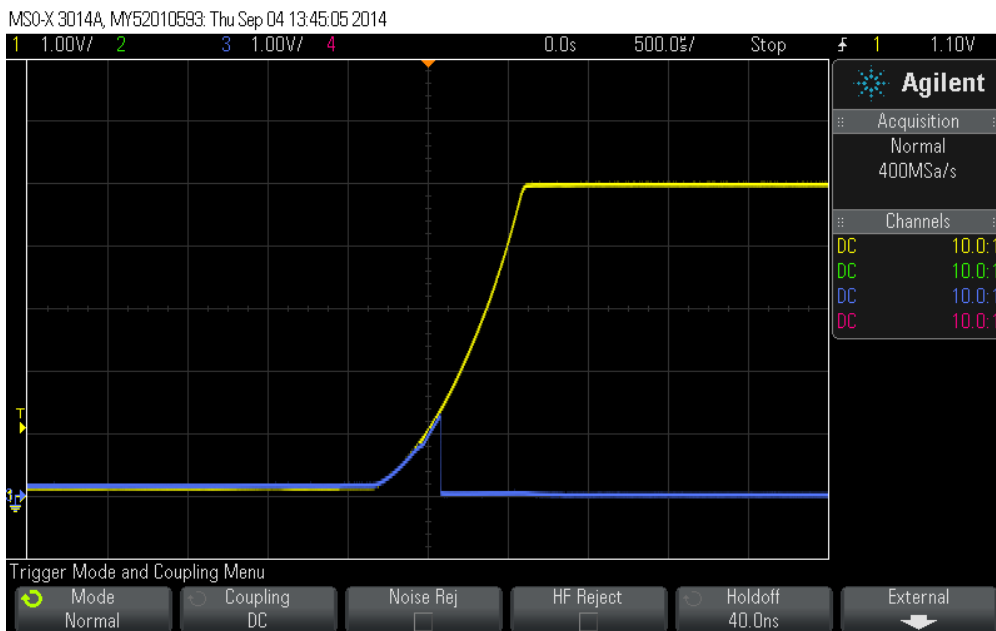
If the user powers other circuits from the VDD33 pin with external load currents in 50mA or 60mA range, then they should consider the power dissipation of the LTC297x. The user will be operating the chip under a higher die temperature due to self-heating which will affect the EEPROM's retention time and endurance. To calculate internal power dissipation, the delta voltage across the regulator multiplied by the total load current will provide the power dissipated by the device.  $(VPWR - 3.3V) * \text{load current} = P_d$ . Remember, the total load current includes both the chip's internal 13mA load and the external load current. For example, if the external load on VDD33 is 50mA, the total current is 63mA. If VPWR is 12V, then  $P_d = (12 - 3.3)0.063 = 0.548W$ . If a simple calculation based on the package's  $\Theta_{JA}$ ,  $0.55W/28C/W$ , you get a temp rise of 15degC above the ambient. That is, this delta T is the temperature rise and is added to the ambient temperature. So if  $T_a = 85C$ , the die temp is 100C. If this is the case, then some of the useful life of the EEPROM will be reduced due to the elevated temp. We show an example calculation on page 19 of the LTC2974/77 datasheet.

## 2.4 How Do VOUT\_EN Pins Behave at Power-up?

At system power-up, the LTC297x PSM devices are designed to keep the managed supply rails disabled until they have been commanded/biased on. This is accomplished by ensuring the VOUT\_EN pins (tied to RUN pin of DC/DC converters) are pulled low when sufficient input voltage is provided. The VOUT\_EN pin is an open drain N-ch output and will pull up to the voltage (through pullup resistor) you apply to the pin. When the VPWR pin reaches ~1.2V, the output enables pull low. This occurs independent of the configuration settings. The impedance of the N-ch pulldown is hundreds of ohms at ~1.2V. When VPWR reaches its minimum operating voltage of 4.5V, the N-ch pulldown is ~80 ohms to GND. The values provided are nominal values.

It is best practice to use the manager's VDD33 supply as the pullup supply. NOTE: If VPWR is 0.0V and the VOUT\_EN pins are pulled to a supply that is already powered up, the LTC2977 will **\*not\*** pull the enable low.

The scope shot below shows the behavior of a VOUT\_EN pin with respect to VPWR and VDD33. The VOUT\_EN pins are hi-Z until the VCC33 pin reaches approximately 1.2V. At VDD33 levels < 0.7V, the output enables are > 10Mohm.



**Figure 1. LTC2977 Vout\_EN pin Behavior**

VPWR (yellow), VOUT\_EN (blue)

## 2.5 How much EEPROM does the LTC2977 have?

For the LTC2974/75/77 devices, the EEPROM array is 8kbits in size (1024 bytes), some of which is used for trim/calibration or is unused. There are 680 bytes of user-accessible EEPROM, 255 of which is dedicated to the fault log. There are 425 bytes that are user configuration bytes.

## 2.6 How to Configure LTC297x to Only Monitor & Supervise Channels

The LTC297x devices have the ability to monitor, supervise, servo, and sequence channels, however you might want to only monitor a particular channel, that is, use the READ\_VOUT command to read the output voltage. The ADC updates every channel's READ\_VOUT register even if the channel has been disabled.

### HARDWARE

From a hardware point of view, connect the sense inputs ( $VSENSEPN$ ,  $VSENSEMN$ ) to the appropriate voltage rails. If the servo feature is not used, tie VDACMn pins to GND and leave VDACPn pins floating.

### CONFIGURATION

The following checklists cover two different cases.

On channels that are monitor/supervise only and **use** VOUT\_EN to enable the channel:

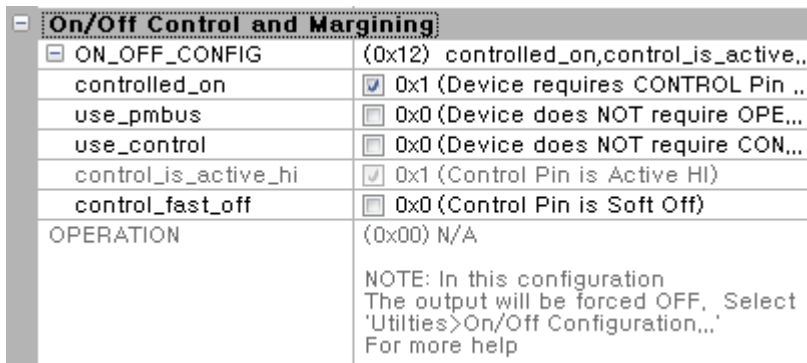
- Configure OPERATION register to 'On' (0x80)
- Configure ON\_OFF\_CONFIG to a value of 0x1E (respond to both PMBus and CONTROL pin)
- Configure the DAC mode to 'disconnected' (in MFR\_CONFIG)
- Configure VOUT\_OV\_FAULT\_LIMIT to 0xC000 (6V)

- Disable shared fault propagation for the channel

On channels that are monitor/supervise only and **do not use** VOUT\_EN to enable the channel:

- Configure OPERATION register to 'Immediate Off' (0x00)
- Configure ON\_OFF\_CONFIG to a value of 0x12 (never power on)
- Configure the DAC mode to 'disconnected' (in MFR\_CONFIG)
- Configure VOUT\_OV\_FAULT\_LIMIT to 0xC000 (6V)

In LTpowerPlay, select the ON/OFF/MARGIN tab. Uncheck the *use\_pmbus* and *use\_control* bits.



Once the ON\_OFF\_CONFIG is set to 0x12, the OPERATION register is a “don’t care”. However if the ON\_OFF\_CONFIG gets set to listen to *pm\_bus* or the *control* pin, OPERATION should be set to OFF (0x00) in that case.

For channels that use the VOUT\_EN pin to enable the channel, disable the Vout fault responses by setting the three responses to 'ignore'. This allows the channel to remain enabled and avoids the FAULTB and ALERTB pins from asserting low.

| Fault Responses — Output Voltage                             |               |
|--|---------------|
| <input checked="" type="checkbox"/> TON_MAX_FAULT_RESPON,... | (0x38) Ignore |
| <input checked="" type="checkbox"/> VOUT_UV_FAULT_RESPON,... | (0x3F) Ignore |
| <input checked="" type="checkbox"/> VOUT_OV_FAULT_RESPON,... | (0x3F) Ignore |

If you want to exclude the channel from the MFR\_PAGE\_FF\_MASK (0xE4) command, unchecking the channel bit in this register allows the channel to ignore PAGE=0xFF commands to the device.

## 2.7 Is it OK to Float Unused Inputs?

Unused sense lines should be shorted to ground and not floated. Floating lines may cause unexpected faults. These faults may interfere with sequencing if the FAULTB pins are connected. Chan A can be faulted because Chan B faulted. Even when faults are ignored, they can still assert ALERTB and create red nodes in LTpowerPlay.

When the lines are shorted to GND, you can set the registers of the manager in a way that consistently avoid faults.

Tie all unused ADC inputs (Vsensep, Vsensem, Isensep, Isensem) to GND.

In systems where the inputs are connected to removable cards and may be left floating in certain situations, connect the inputs to GND using 100k resistors. Place the 100k resistors before any filter components to prevent loading of the filter.



Unused Tsense pins on the LTC2974/2975 should be tied to GND. Floating Tsense pins is not recommended and may return unpredictable temperature values.

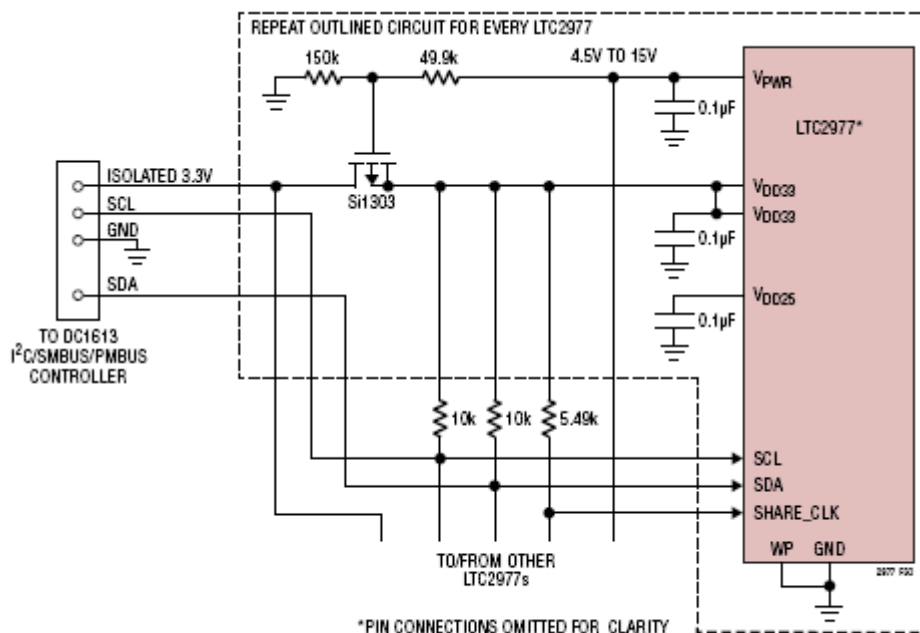
Do not leave CONTROLn pins floating. Pull up to 3.3V with a 10k resistor.

Do not leave WDI/RESET pin floating. Tie it to VDD33 with a 10k resistor. Do not connect a capacitor to the WDI/RESETB pin.

2.8 Is it possible that EEPROM can be written/read only with VDD33 applied (no Vpwr supply)?

*Additional concern: Customer wants to check EEPROM contents before main power is applied to be safe. My experience says that high voltage power supply must be applied for writing EEPROMs generally.*

Yes you may write the EEPROM by applying power only to VDD33. The internal memory write circuitry is based from VDD33 from which a higher supply is generated in order to perform EEPROM writes. Typically customers use our USB dongle and FET switch to power the 2977 without powering the entire system. They use this diagram from our datasheet (below). When main power is off (VPWR at 0V), the PFET is turned on and the isolated 3.3V supply from the DC1613 dongle powers the 2977. You can read and write the registers and write to NVM with dongle power. The dongle is also very useful for debug purposes.

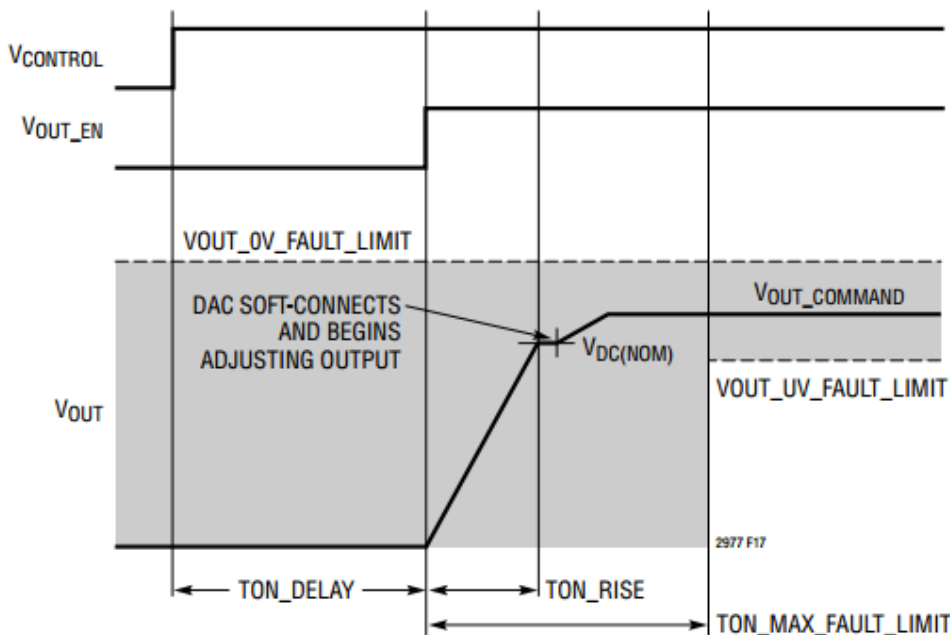


**Figure 30. DC1613 Controller Connections When  $V_{PWR}$  Is Used**

## 2.9 The Device reports a TON\_MAX\_FAULT, but the Rise Time of the Output is Fine. Why?

The TON\_MAX\_FAULT is a special kind of time-based under voltage fault. If the supply does not come up in time after being enabled, a TON\_MAX\_FAULT is declared. For instance, if a supply is shorted to ground, the output voltage will remain zero, and at  $t = \text{TON\_DELAY} + \text{TON\_MAX\_FAULT\_LIMIT}$  a TON\_MAX\_FAULT will be declared.

Figure 17 of the LTC2977 datasheet provides a nice reference waveform:



**Figure 17. Typical On Sequence Using Control Pin**

Note that the TON\_MAX\_FAULT\_LIMIT timer starts counting after TON\_DELAY expires (when the VOEN goes high to attempt to enable the output). If you put a scope probe on the output voltage, you may see a fast risetime on the output, but the part may still declare a TON\_MAX\_FAULT.

The supply may not be ramping immediately after VOEN goes high. Here are a few scenarios that could produce this behavior:

- VOEN may not be directly connected to the supply in question (it may be being enabled by something else, later)
- There could be additional internal delays in the supply after VOEN goes high before the voltage starts to ramp
- There may be other unfulfilled conditions that are preventing the supply from powering up. Perhaps the VIN for this supply is coming up later in the sequence, or the supply is tracking another supply that is not yet enabled, for example

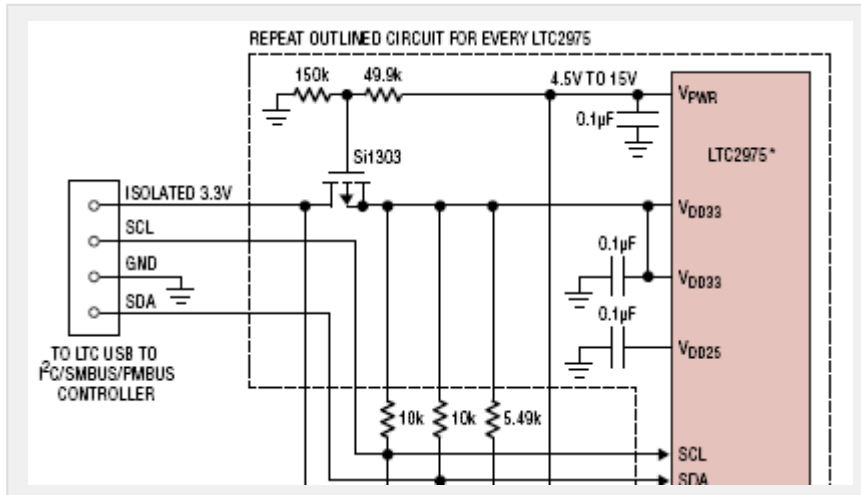
To troubleshoot the root cause, it is best to place a scope probe on VOEN, VOUT, and ALERTB (and any other relevant signals, like input supplies). Trigger the scope on the falling edge of ALERTB, and look at VOUT at this instant in time. Set the timebase to capture the rising edge of VOEN + at least TON\_MAX\_FAULT\_LIMIT ms. This may provide insight into the root cause.

The remedy depends on the specific scenario, but may involve:

- Adjusting the sequencing order to reflect power/tracking dependencies
- Increasing the TON\_MAX\_FAULT\_LIMIT to allow the supply more time to come up
- Resolving other issues in the application that may be preventing the supply from coming up

## 2.10 What replacement part is suggested for the PFET that powers LTC297x via USB dongle?

There is a circuit provided in the LTC297x power system manager datasheets that shows how a USB dongle (DC1613) can power an LTC297x chip. The circuit allows the chip to be powered from VPWR or from a USB dongle.



Power from the dongle is provided via a PFET that acts as a switch. The PFET part number is Si1303, a Vishay/Siliconix device. This part has been obsoleted. A few replacement parts follow:

NTS4173PT1G, DMP2160UW, Si2305CDS, Si2371EDS, DMP2100U

The important parameters that make a good replacement are  $V_t \text{ max} < 1.5\text{V}$ , and  $R_{ds-on} < 500 \text{ mohm}$ . The voltage divider on the PFET gate limits the gate voltage. This allows a wider selection of PFETs that may have a low  $V_{gs}$  rating (e.g.  $\pm 8\text{V}$ ). These alternative parts are available in SOT323 (SC70) and SOT23 packages.

## 3 PSM Controllers

This FAQ Section contains Frequently Asked Questions about PSM Controllers

### 3.1 What causes the ADC Invalid bit in the MFR\_PADS register to assert?

#### 3.1.1 What causes the ADC Invalid bit in the MFR\_PADS register to assert?

The LTC380X devices have a MFR\_PADS register with an "ADC Values Invalid" bit (bit 11). When VOUT is near 3.3V, an internal ADC read operation can fail to read, causing bit 11 to be a 1. The 1 value is not persistent, and the next valid ADC read operation clears it.

##### 3.1.1.1 Retry Behavior

When the ADC read operation fails to read, the device retries the measurement. However, the LTC3880 and other LTC388X devices have a different retry behavior.

##### 3.1.1.2 LTC3880 Retry Behavior

When the LTC3880 has an ADC Invalid bit, the telemetry loop is reset and telemetry readings begin from the beginning of the loop.

##### 3.1.1.3 LTC3887 and other devices

When the LTC3887 has an ADC Invalid bit, the current measurement is retried until it succeeds, and then telemetry continues.

##### 3.1.1.4 LTM4676/A/B

The LTM4676 contains an LTC3880 and therefore has its behavior. The LTM4676A and the new LTM4676B has an LTC3887, and therefore has its behavior.

##### 3.1.1.5 Practical Implications

An ADC Invalid bit of 1 is very infrequent in the non LTC3880 devices because the immediate retry of the measurement has a very high probability of success due to hysteresis of the ADC comparator. Therefore, it is rare to see this bit in non LTC3880 devices.

An ADC Invalid bit of 1 is occasionally seen in LTC3880 when the VOUT is near 3.3V.

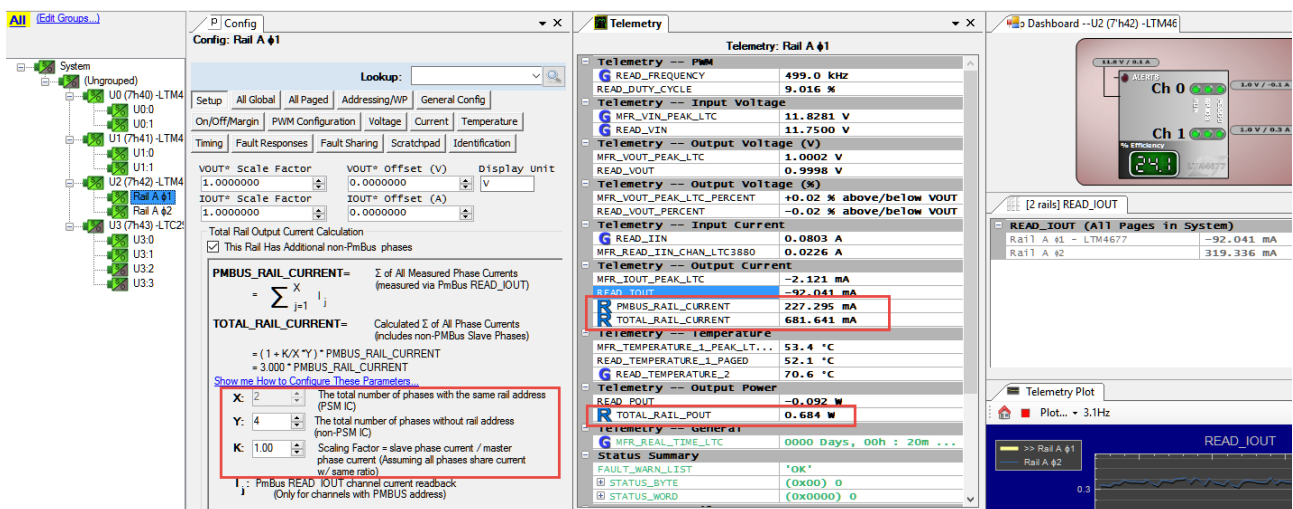
## 3.2 Can LTpowerPlay Calculate Total Rail Current for Polyphase Rails Utilizing Extra non-PMBus 'Slave' Phases?

Yes. Let's take an example, where you have a 2-phase LTM467X uModule design, with 4x additional external non-PMBus 'slave' phases for a total of a 6-phase output. For simplicity let's also assume that the slave-to-slave and slave-to-master current sharing is equal.

You would configure LTpowerPlay to calculate total RAIL current as follows:

- Select the Rail of interest in the system tree (add a polyphase template if you have not already)
- Select the 'Setup' tab in the configuration area
- Check the box that informs LTpowerPlay that this rail has additional non-PMBus phases
- X: is configured as 2 (as determined by the polyphase template, which in our case is a dual-phase single output LTC3882 template)
- Y: set to 4
- K: set to 1.0

Below is a screenshot that shows how to configure this, and the math LTpowerPlay will use to infer total rail current.



In the telemetry panel, you will notice some parameters marked with an 'R' icon. The 'R' indicates that this is a 'Rail' level pseudo-register (calculated by LTpowerPlay using the math above).

**PMBUS\_RAIL\_CURRENT:** Is the sum of the READ\_IOUT values for all PMBus channels participating in the rail group (same rail address – in our example there are 2x PMBus phases).

*NOTE: For PSM managers, X is fixed at 1, so the PMBUS\_RAIL\_CURRENT is always the same as READ\_IOUT for the channel.*

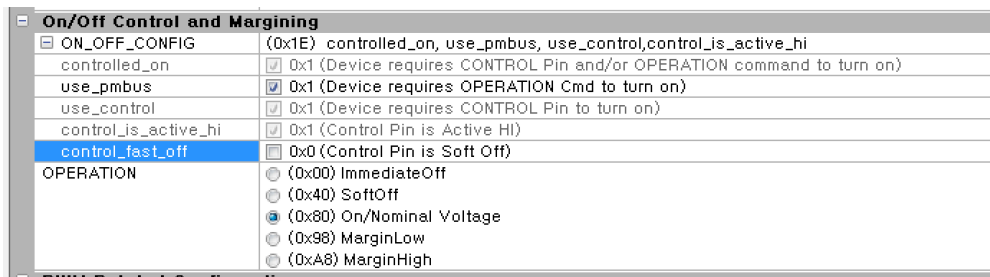
**TOTAL\_RAIL\_CURRENT:** Extrapolates further to infer total rail current when external slaves are used (In our case, this is calculated as PMBUS\_RAIL\_CURRENT \* 3.0 as indicated by LTpowerPlay).

**TOTAL\_RAIL\_POUT:** Is extrapolated/calculated as the sum of PMBUS READ\_POUT values \* the same PMBus-to-total extrapolation factor (3.0 in our case).

### 3.3 Controlled Off

The LTC388X family of devices have an immediate vs. controlled off when the RUN pin is pulled low.

The ON\_OFF\_CONFIG register has a bit called "control\_fast\_off," and when it is in the "soft off" state, pulling the RUN pin low will cause the device to sequence off in a controlled manner. When it is not set to "soft off", the rail immediately stops delivering power and relies on the load to decay.



### 3.4 How can I Determine if a Device is a LTC3880 vs. LTC3880-1

The MFR\_SEPCIAL\_ID for LTC3880 and LTC3880-1 are the same.

To determine which device is on the bus, you can use an unpublished command. The command is MFR\_REVISION (9B). This command is a block read, just like the command MFR\_MODEL. It will return a block of 5 bytes. The 5th byte returned will be a 0 for a LTC3880, and it will be a 1 for LTC3880-1.

The other bytes (1-4) from MFR\_REVISION contain information that LTC does not publish.

### 3.5 How do I set up PGOOD?

How do I set up PGOOD?

PSM Controllers like the LTC3880 support using the GPIOB pin as a FAULTB pin or PGOOD pin. The most common use is to use GPIOB as FAULTB, because when a PSM design shares a FAULTB pin, any fault in the system can inhibit other rails. LTpowerPlay is used to determine which rails respond to a FAULTB. In addition to controlling the inhibit behavior of a FAULTB input, LTpowerPlay can control which faults are propagated to FAULTB, meaning controlling it as an output. The reason the FAULTB can be both input and output is that the GPIOB pin is an open drain.

In applications that require using the GPIOB pin as a PGOOD, there are two methods to setup the GPIOB pin. The older, and no longer recommended approach, is to use bit 9/10 of the MFR\_GPIO\_PROPAGATE\_LTCNNNN register. This method used the internal ADC to detect when power is good. The second method, the one we recommend, is to enable the VOUT\_BELOW\_UV bit 12. This effectively uses a fast comparator for PGOOD. When using VOUT\_BELOW\_UV, other bits should be "off" so that a fault does not affect the behavior of the PGOOD signal. Therefore, only set the VOUT\_BELOW\_UV bit.

### 3.6 How do you read the ALERTB pin via PMBus?

How do you read the ALERTB pin via PMBus?

In order of preference, 1 being most preferred:

1. MFR\_COMMON bit #7. The advantage of looking at MFR\_COMMON over MFR\_PADS is the fact that MFR\_COMMON will always respond, even when the device is busy. Also, this is the same bit assignment common to all LTC PSM product families.

2. ARA (Alert Response Address). This is defined by the PMBus/SMBus standard. It is also a built in command so a busy device will not NACK or cause a BUSY fault.
3. Some products support MFR\_PADS bit #10. It is the least preferred and is an unstandardized method. For example, on the LTC2977, the ALERTB is reported on bit #14.

### 3.7 How long does it take for a PSM controller to reset?

How long does it take for a PSM controller to reset?

From the data sheets table the typical values are:

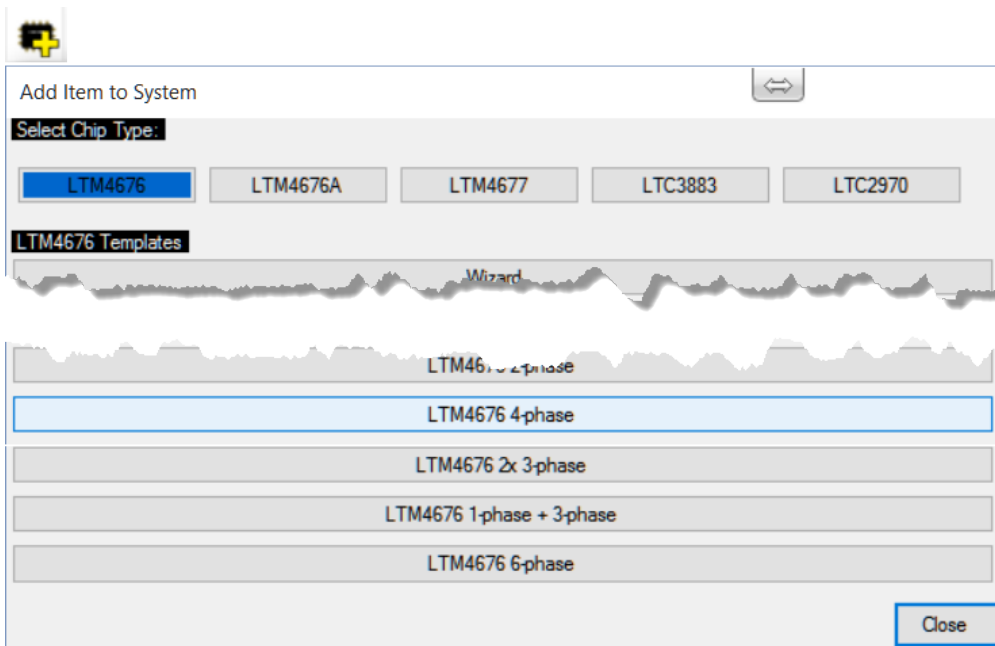
| Device  | typical | units |
|---------|---------|-------|
| LTC3880 | 145     | ms    |
| LTC3882 | 70      | ms    |
| LTC3883 | 145     | ms    |
| LTC3884 | 120     | ms    |
| LTC3886 | 70      | ms    |
| LTC3887 | 70      | ms    |

### 3.8 I have a need for a polyphase configuration that is not covered by the existing polyphase templates. What do I do?

For common polyphase scenarios, it is strongly recommended that you use the existing polyphase templates to configure polyphase rail groups, as they make quick work of configuring such rails. Follow these simple steps to use a polyphase template to add a polyphase rail:

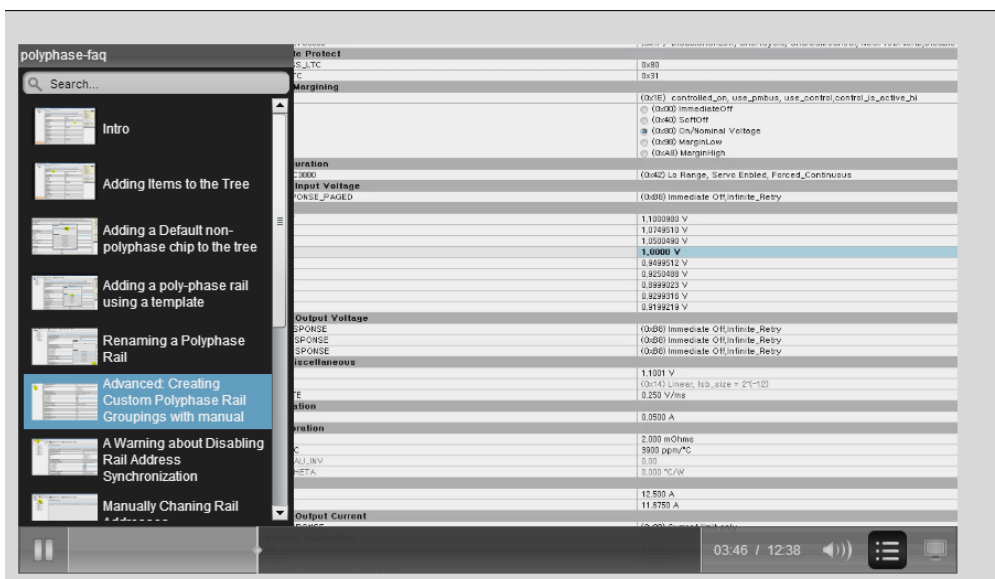
- Click the 'Add Item to System' button in the toolbar.
- Selecting the appropriate device type to add
- Then select the appropriate polyphase template from the choices offered. (i.e. 2-phase, 4-phase, 1+3, etc)
- The Polyphase wizard will guide you through entering the rest of the parameters to setup your rail and add it to the system.

The screenshot below shows an example of selecting the LTC4676 4-phase template:



Though polyphase templates are helpful for most common scenarios, they cannot cover every possible need. If your needs are not covered by an existing template, you can use an advanced technique in LTpowerPlay to group phases as you like. A polyphase faq video exists that covers basic polyphase templates as well as advanced use cases like this.

Specifically, to learn how to group the phases in custom ways, please [click this link](#)<sup>19</sup> to watch the polyphase faq tutorial video. Then use the table of contents links on the left side of the video to navigate to the advanced section as highlighted below:



19 <http://www.ltpowerplay.com/support/videos/polyphase-faq/>



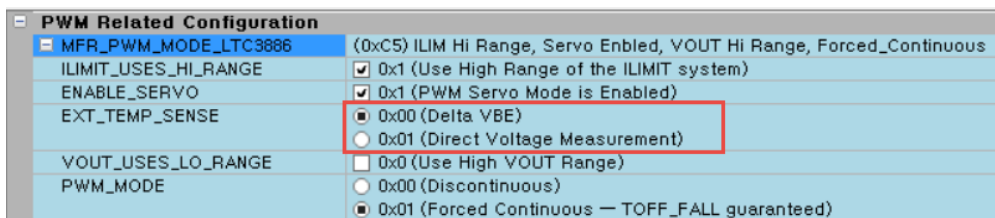
### 3.9 I Observe Bad Temp Readings. What are the Possible Causes?

A temperature reading of -273deg may indicate several problems:

1. Shorted lines sense lines
2. Open lines sense lines
3. Missing diode
4. Reversed diode
5. Incorrect capacitor value: very large C

If the sensing circuit is correct, and you still see errors (perhaps a high temperature or OT faults), double check that the sensing scheme is configured properly. Some newer chips (i.e. LTC3882 and newer) have a setting to select delta VBE (common) or direct VBE (less common) sensing schemes for temperatures. When this configuration bit does not match the sensing circuit, you can have large errors in temperature measurements.

The example screenshot below shows the external temp sense configuration field in the MFR\_PWM\_MODE register for the LTC3886:



### 3.10 Misconnected Pins

#### 3.10.1 Symptoms

##### 3.10.1.1 Wrong Frequencies

Connecting/Shorting SHARE\_CLK to SYNC can interfere with operation. Typically SHARE\_CLK runs slower (100Khz) than SYNC (350KHz+).

When they are shorted, the PWM will run the wrong frequency and will typically cause a fault.

### 3.11 My PSM Controller is pulling ALERTB low. Is there a way to mask certain faults?

Yes. The feature is called SMB\_ALERT\_MASK. Consult the datasheet to see if your device supports this feature. For devices that support it, you can Mask faults using these steps:

- Select the device of interest in the System Tree
- Navigate to the 'Fault Responses' section of the Configuration
- Navigate to the 'Fault Responses – ALERTB Masking' sub section

- Check the box next to the status items that you would like to mask (prevent from pulling ALERTB low). Here is an example:

| Fault Responses -- ALERTB Masking                  |   |                     |
|--|---|---------------------|
| <input type="checkbox"/> MASK_STATUS_CML           | (0xFB)                                  | INVALID_OR_UNSUP... |
| INVALID_OR_UNSUPPORTED...                          | <input checked="" type="checkbox"/> 0x1 |                     |
| INVALID_OR_UNSUPPORTED...                          | <input checked="" type="checkbox"/> 0x1 |                     |
| PEC_FAILED   | <input checked="" type="checkbox"/> 0x1 |                     |
| MEMORY_FAULT                                       | <input checked="" type="checkbox"/> 0x1 |                     |
| PROCESSOR_FAULT                                    | <input checked="" type="checkbox"/> 0x1 |                     |
| Reserved   | <input type="checkbox"/> 0x0            |                     |
| OTHER_COM_FAULT                                    | <input checked="" type="checkbox"/> 0x1 |                     |
| OTHER_UNKNOWN_FAULT                                | <input checked="" type="checkbox"/> 0x1 |                     |
| <input type="checkbox"/> MASK_STATUS_INPUT         | (0x00)                                  | 0                   |
| <input type="checkbox"/> MASK_STATUS_IOUT          | (0x00)                                  | 0                   |
| <input type="checkbox"/> MASK_STATUS_MFR_SPECIF... | (0x11)                                  | PLL_UNLOCK_FAULT... |
| <input type="checkbox"/> MASK_STATUS_TEMP          | (0x0)                                   | 0                   |
| <input type="checkbox"/> MASK_STATUS_VOUT          | (0xFE)                                  | VOUT_OV_FAULT, V... |
| VOUT_OV_FAULT                                      | <input checked="" type="checkbox"/> 0x1 |                     |
| VOUT_OV_WARN                                       | <input checked="" type="checkbox"/> 0x1 |                     |
| VOUT_UV_WARN                                       | <input checked="" type="checkbox"/> 0x1 |                     |
| VOUT_UV_FAULT                                      | <input checked="" type="checkbox"/> 0x1 |                     |
| VOUT_MAX_WARNING                                   | <input checked="" type="checkbox"/> 0x1 |                     |
| TON_MAX_FAULT                                      | <input checked="" type="checkbox"/> 0x1 |                     |
| TOFF_MAX_WARN                                      | <input checked="" type="checkbox"/> 0x1 |                     |
| VOUT_TRACKING_ERROR                                | <input type="checkbox"/> 0x0            |                     |

In the above example, all maskable STATUS\_CML and STATUS\_VOUT bits are set to be masked. This means that when the part encounters one of these conditions, it will NOT pull the ALERTB pin low, though it will continue to set the bits in the status register which can be read via PmBus.

**NOTE:** If a device does not allow masking of certain bits (i.e. VOUT\_TRACKING\_ERROR above), you will not be able to check these items in the GUI. A convenient way to learn which bits are maskable is to click text next to the register and type 0xFF for the value. For example, if you select the text next to MASK\_STATUS\_VOUT, and type '0xFE <Enter>', LTpowerPlay will set the value to 0xFE, as bit position 0 is not clearable for this device.

## 3.12 Overlapping Rail Address

### 3.12.1 Can you set the MFR\_RAIL\_ADDRESS to the effective device address?

The LTC38XX family allows setting the MFR\_RAIL\_ADDRESS to the effective address of the device. This will interfere with LTpowerPlay when it tries to read PAGEed commands. It may result in unexpected behavior when writing. The data sheet makes no guarantees of good behavior when there is overlap.

Therefore, it is advised to make the MFR\_RAIL\_ADDRESS unique within the PMBus Address Space.

## 3.13 Storing user information in NVM

16 bits of information may be stored in register USER\_DATA\_03 using LTpowerPlay, and LTpowerPlay, In Flight Update, and device programming at LTC, Arrow, or BPM programming at the factory will set/maintain this value, just like any other register value.

The USER\_DATA\_03 data may also be written by firmware, but programming devices with LTpowerPlay, In flight Update, programming at LTC, Arrow, or BPM at the factory will overwrite it. Firmware may also overwrite the value set by programming.

Nonetheless, none of the methods mentioned here write the USER\_DATA\_03 with CRC data or LTpowerPlay specific data.

### 3.14 The part is reporting a PLL\_UNLOCK fault. What are the Possible Causes?

There are a number of things to check when you get a PLL\_UNLOCK fault reported in the STATUS\_MFR\_SPECIFIC register:

- Check that VIN for the power stage is present.
- Check the READ\_FREQUENCY register in the telemetry view. If it is low or zero, there may be a SYNC clock issue
- Probe the SYNC Clock and insure that the frequency is somewhat near to the programmed clock rate. Otherwise you may have a SYNC clock issue
- In rare cases, PLL\_UNLOCK faults may be caused by extreme levels of noise in the system. Check the layout.

### 3.15 What is the latency of a VOUT change?

What is the latency of a VOUT change?

When the VOUT\_COMMAND is issued, a controller must process an interrupt followed by a calculation involving VOUT\_TRANSITION\_RATE. The total time is typically 500uS to 1ms. This holds as long as there are no other PMBus transactions to the controller that may interfere with processing time.

When VOUT\_COMMAND is issued, after the transition begins, bit 4 of MFR\_COMMON will become 0. When bit 4 goes high, it is certain that the transition is complete. However the update of bit 4 is part of a background loop inside the controller. The VOUT\_COMMAND will not respond until bit 4 goes low, which can add latency.

If the latency must be as consistent as possible, poll on bit 4 and only issue a VOUT\_COMMAND when the bit is high.

### 3.16 What is the real OC value?

What is the real OC value of a PMBus DC-DC converter, such as the LTC3880?

Each device uses a voltage comparator and reference value to OC. The data sheet will show the ranges and their resolution.

The actual trip value is the DAC value divided by Rsense. However, these devices also correct for temperature, therefore they may move a bit or two.

This fact can cause some confusion when validating a design. If the design uses a very small Rsense, say under 1mOhm, it is common that the lowest DAC value will be the actual setting. If you try to lower the value in LTpowerPlay to simulate a Fault, it will not fault because the device will be operating on the lowest possible DAC value.

Furthermore, LTpowerPlay accepts a number of any value and sends it to the device. The device will use the closest DAC value. This leads to some confusion. As a user you need to calculate the current table using the voltage table in the data sheet (LTC3880 is page 75), then note that temperature changes will move the DAC to the closes value to what was programmed with LTpowerPlay.

### 3.17 Why do I see a MFR (GPIOB\_ASSERTED\_LO) fault at power up?

In specific configurations, you may observe that the GPIOB\_ASSERTED\_LO bit is set in the STATUS\_MFR\_SPECIFIC status register after a power up.

PSM Controllers are designed to report this status fault when the GPIOB pin is asserted low externally. From an individual channel's perspective, if the GPIOB pin is left floating, the channel will not report a status fault when it is driving the pin low (for example when configured as PGOOD). However, if the pin is being driven externally, the chip will report this condition in the status register.

A specific configuration which can cause this bit to be set at power up is when you have a LTC388x channel where the GPIO pin is configured to propagate VOUT\_BELOW\_UV (power good like behavior), **and the GPIOB pins are shorted together** with other channels (for example to other phases of a polyphase rail). In this application chip logic sees that the GPIOB pin is driven externally (by the other phases) and it declares the status fault.

#### Solutions

Solutions that avoid the problem include a proper use and a workaround. It is recommended that you **do not short GPIOB pins together when configured with power good** like behavior (propagating VOUT\_BELOW\_UV for example). If this is not possible, you may work around the issue in a couple of ways. If you simply want to avoid an ALERT, you can configure the device not to drive ALERT when GPIO is asserted low. However, this will not prevent the status bit from being set in STATUS\_MFR\_SPECIFIC. You may issue a clear faults after each power up to clear this bit.

### 3.18 How do I set the temperature configuration when TSENSE pins are grounded on the LTC3882?

The MFR\_PWM\_MODE\_LTC3882 command sets important PWM controls for each channel. The addressed channel(s) must be turned off by its RUN pin, OPERATION command, or their combination, when this command is issued. Otherwise the LTC3882 will NACK the command byte, ignore the command and its data, and assert a BUSY fault.

When bit 5 is cleared, the LTC3882 computes temperature in °C from  $\Delta V_{BE}$  measured by the ADC at the TSNSn pin as

$$T = (G \cdot \Delta V_{BE} \cdot q / (K \cdot \ln(16))) - 273.15 + O$$

When bit 5 is set, the LTC3882 computes temperature in °C from TSNSn voltage measured by the ADC as

$$T = (G \cdot (1.35 - V_{TSNSn} + O) / 4.3e-3) + 25$$

For both equations,

$G = \text{MFR\_TEMP\_1\_GAIN} \cdot 2 - 14$ , and

$O = \text{MFR\_TEMP\_1\_OFFSET}$

Sometimes the user doesn't want to use the temperature sense function of the LTC3882 and ground TSENSE pins. In order to get normal temperature reading, the MFR\_TEMP\_1\_OFFSET needs to be set properly based on the chosen temperature sensing method:

If  $\Delta V_{BE}$  method is used, set MFR\_TEMP\_1\_OFFSET to be +300 (in °C) in order to read the temperature as 26.85;

If direct  $V_{BE}$  method is used, set MFR\_TEMP\_1\_OFFSET to be -1.35 (in Volt) in order to read the temperature as 25.

## 4 Supervisors

This FAQ Section contains Frequently Asked Questions about Simple Supervisor Chips

### 4.1 LTC2933 Apps FAQ

This Section contains Frequently Asked Questions about the LTC2933

#### 4.1.1 For the HIGH RANGE setting on pin V1 the comparator threshold can be programmed as high as 15v, but the Absolute Maximum rating for pin V1 is 14v, can I drive 15v into pin V1 in HIGH RANGE?

##### 4.1.1.1 Answer: No Way! Always respect the absolute maximum ratings.

The voltage threshold can be set as high as 15v, and the comparator will indicate that voltages below the threshold are low, but driving above abs max will damage the part.

#### 4.1.2 How much hysteresis do the LTC2933/36 input comparators have?

There are two types of comparators: Vn inputs, and GPI inputs.

There is no hysteresis on any of the comparators. There IS, however, a dependence of the comparator switching speed on the voltage overdrive at the inputs. This means that for voltages very close to the threshold, the comparator respond slowly. In a normal application where the voltage is sweeping past the threshold, this is sufficient to suppress glitches. Test Engineering has reported comparator chattering if the voltage sits right at the comparator threshold.

#### 4.1.3 If an application does not use GPI2 how should it be configured?

##### 4.1.3.1 Put the pin in a defined state and define its behavior.

Set GPI2 to have a weak pull-up, and to do something predictable, use GPI2 as a MARGb pin. Set it this way: GPI\_CONFIG[15:0] = 0x1041 (sets GPI2 as MARGb and not mapped, GPI1 enable CLEAR\_HISTORY response, GPI1 manual reset, GPI1 mapped to GPIO1). As an added bonus, pull the MARGb pin low to mask unwanted faults when changing register settings, or when powering-up.

It is a bad idea to use the "reserved" setting in GPI\_CONFIG (or anywhere else). "Reserved" means that the designer did not specify what would happen if you set the part that way. It could do anything, and it might even do different things on different parts. When you see "Reserved" you should stay away.

#### 4.1.4 What are the configuration register settings for the "typical application" on page1 of the data sheet?

##### 4.1.4.1 Register Settings:

| Address | Register       | Setting | Comment   |
|---------|----------------|---------|---|
| 0x00    | WRITE_PROTECT  | 0xAAA8  | unlocked, default key   |
| 0x01    | GPI_CONFIG     | 0x1001  | GPI1 is manual reset, mapped to GPIO1, does not clear history<br>GPI2 is MARGb, but is not mapped, and not used                         |
| 0x02    | GPIO1_CONFIG   | 0x002E  | GPIO1 is active low output with weak pull-up, 205ms release delay   |
| 0x03    | GPIO2_3_CONFIG | 0x0706  | GPIO2 is active low output with weak pull-up, no release delay<br>GPIO3 is active low ALERTB output with weak pull-up, no release delay |
| 0x04    | V1_THR         | 0xDBAB  | VTH_OV = 13.2v, VTH_UV = 10.8v  |
| 0x05    | V2_THR         | 0xE6B4  | VTH_OV = 5.5v, VTH_UV = 4.5v  |
| 0x06    | V3_THR         | 0x8868  | VTH_OV = 3.63v, VTH_UV = 2.97v  |
| 0x07    | V4_THR         | 0xE6B4  | VTH_OV = 2.75v, VTH_UV = 2.25v  |
| 0x08    | V5_THR         | 0x9975  | VTH_OV = 1.98v, VTH_UV = 1.62v  |
| 0x09    | V6_THR         | 0x785A  | VTH_OV = 1.65v, VTH_UV = 1.35v  |
| 0x0A    | V1_CONFIG      | 0x00BD  | High range, UV and OV map to GPIO1 & GPIO3, OV maps to GPIO2  |
| 0x0B    | V2_CONFIG      | 0x00BD  | Medium range, UV and OV map to GPIO1 & GPIO3, OV maps to GPIO2  |
| 0x0C    | V3_CONFIG      | 0x00BD  | Medium range, UV and OV map to GPIO1 & GPIO3, OV maps to GPIO2  |
| 0x0D    | V4_CONFIG      | 0x01BD  | Low range, UV and OV map to GPIO1 & GPIO3, OV maps to GPIO2   |
| 0x0E    | V5_CONFIG      | 0x01BD  | Low range, UV and OV map to GPIO1 & GPIO3, OV maps to GPIO2   |
| 0x0F    | V6_CONFIG      | 0x01BD  | Low range, UV and OV map to GPIO1 & GPIO3, OV maps to GPIO2   |
|         |                |         |   |

## 4.1.5 What is the difference between using a GPIO as RSTb and ALERTb?

### 4.1.5.1 ALERTb is a pin behavior specified by the SMBUS spec. RSTb is a flexible reset function defined by the system.

The datasheet shows on page 1 a typical application where GPIO1 is RSTb and GPIO3 is ALERTb.

ALERTb is a latched indicator that responds to any comparators or inputs mapped to it. It asserts when its inputs assert, but does not de-assert until the I2C bus gives an alert response, and the part responds. This behavior is intended to interrupt a host controller, which then polls the bus for the alerting device.

Note that an alert is only generated if something is mapped to the pin. You must map comparators and/or GPIs and/or GPIOs to the ALERTb pin.

## 4.1.6 What registers are stored in EEPROM with a STORE\_USER command?

### 4.1.6.1 Registers:

WRITE\_PROTECT (0x00)

GPI\_CONFIG (0x01)

GPIO1\_CONFIG (0x02)

GPIO2\_3\_CONFIG (0x03)

Vn\_THR (0x04 - 0x09)

Vn\_CONFIG (0x0A - 0x0F)

NOTE: BACKUP\_WORD is a special case. It is CLEARED with the following sequence:

- 1) CLEAR\_HISTORY command
- 2) STORE\_USER command (remember that any STORE\_USER command stores configuration registers too)
- 3) RESTORE\_USER command

## 4.1.7 Why can't LTpowerPlay write to the board after I load a project file?

### 4.1.7.1 A: LTpowerPlay needs to "Go Online" before writing to the LTC2933 (or any other part)

Press the "GO ONLINE" button on the LTpowerPlay button bar.



## 4.1.8 Why does LTpowerPlay show data in reserved register bits?

### 4.1.8.1 A: Reserved bits read-back unpredictable values. LTpowerPlay reports what it reads.

When you read a register with reserved bits, the result is unpredictable. The part may return either 1 or 0 for each reserved bit, and the value is not guaranteed to remain fixed.

When you program a register containing reserved bits, you can send any data to the reserved bits, but the part may not retain that data.

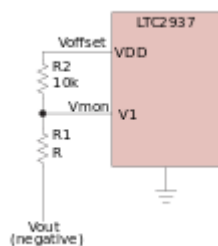
## 4.2 LTC2937 Apps FAQ

This Section contains Frequently Asked Questions about the LTC2937

### 4.2.1 How do I monitor a negative supply?

The datasheet description can be a little daunting. Here's the recipe:

(assume the negative supply is supervised by the LTC2937's channel 1)



First decide on OV and UV limits for your supply (Vout).

Vout(OV) is the most negative allowed voltage (furthest from ground)

Vout(UV) is the least negative allowed voltage (closest to ground)

Next decide on OV voltage at the LTC2937 V1 monitor pin (Vmon):

Vmon(OV) must be above +0.2V (the lowest selectable comparator threshold voltage)

Vmon(OV) must be below +1.2V (the highest selectable comparator threshold)

It is best to set it near +0.2V.

Then solve for the ratio of R1/R2:

$$\frac{R1}{R2} = \frac{Vout(OV) - Vmon(OV)}{Vmon(OV) - Voffset}$$

Vout(OV) is negative. Vmon(OV) is positive. Voffset is +3.3V

The resistor values should be 10s of kohms to keep their current low enough for the VDD pin.

Keep in mind that if you are using active discharge, the discharge FET will pull current through R1 and R2, and may give a skewed discharge measurement.



Try selecting  $R2 = 10k$ , then calculating  $R1$ .

Calculate the monitor voltage for the  $V_{out}(UV)$  voltage:

$$V_{mon}(UV) = V_{out}(UV) \cdot \left( \frac{R2}{R1 + R2} \right) + V_{offset} \cdot \left( 1 - \left( \frac{R2}{R1 + R2} \right) \right)$$

Next program the LTC2937 registers ( $V\_THRESHOLD\_n[15:0]$ ).

$$Code(OV) = 250 \cdot (V_{mon}(OV) - 0.18)$$

UV is a little more tricky because it is shifted lower by 5% to accommodate start-up hysteresis.

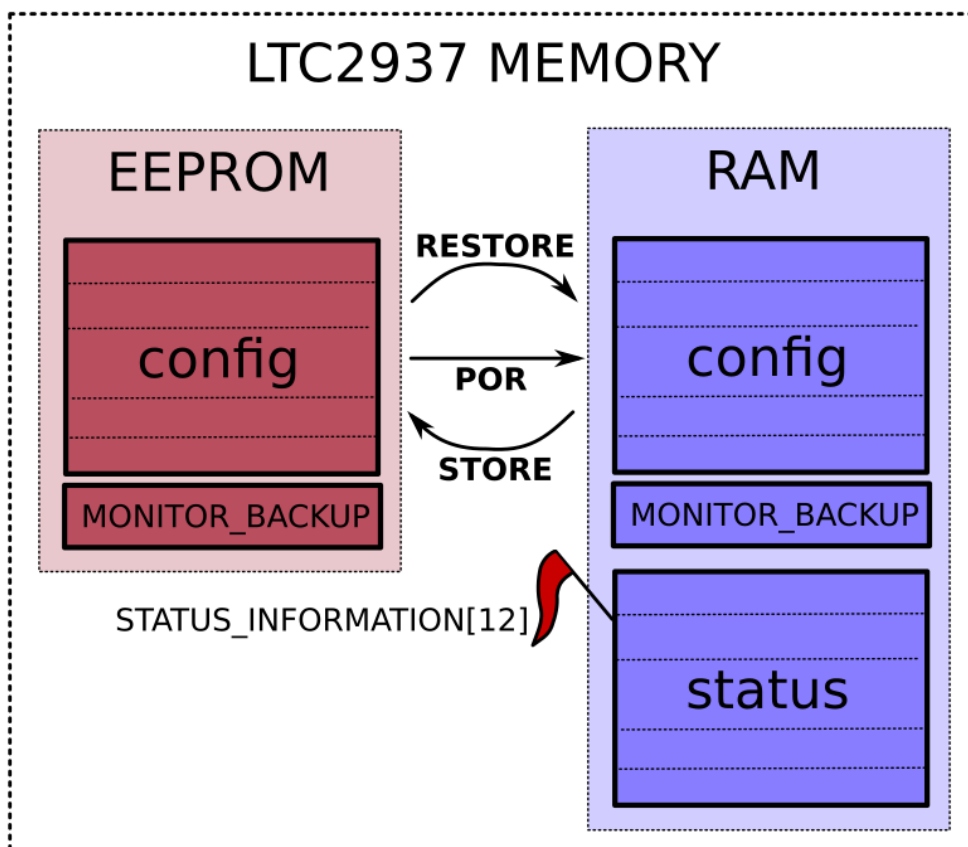
$$Code(OV) = 250 \cdot (V_{mon}(OV) - 0.18)$$

Remember to set the channel to negative adjustable range in the  $V\_RANGE$  register.

#### 4.2.2 How do I use the fault log in the LTC2937?

Q: How do I use the LTC2937 fault log?

A: The LTC2937 is different from a typical PSM part with its extensive EEPROM "black box" that stores telemetry related to what went wrong prior to a fault. The LTC2937 has several registers that store status and fault information, but only one of these is stored in EEPROM at the time of the fault for later retrieval. Here is a map of the memory in the LTC2937:



The LTC2937 has several status registers that indicate fault information current from the last time it was cleared, or power was cycled. These registers include: STATUS\_INFORMATION, MONITOR\_STATUS, and

MONITOR\_STATUS\_HISTORY. Only the MONITOR\_BACKUP register is stored in EEPROM at the time of the fault. Much of the fault status is cleared upon successful sequence-up, but the non-volatile MONITOR\_BACKUP register is maintained until it is deliberately cleared.

NOTE that the MONITOR\_BACKUP register is affected by the same commands that affect the configuration registers: STORE and RESTORE. Be cautious to avoid accidentally storing modified configuration settings into EEPROM while you are manipulating the fault information in MONITOR\_BACKUP.

The STATUS\_INFORMATION register contains a flag indicating when there is information stored in the EEPROM MONITOR\_BACKUP register. When STATUS\_INFORMATION[12] = 1 there is fault information to be retrieved from EEPROM.

#### 4.2.2.1 To READ the MONITOR\_BACKUP register:

You must retrieve the information from EEPROM.

Either cycle power, which will execute a POR operation, or:

- 1) sequence down to put the state machine in a safe state
- 2) issue a CLEAR command to further safe the state machine (may not always be necessary)
- 3) issue a RESTORE command to retrieve the MONITOR\_BACKUP and all configuration register contents

NOTE: this will restore configuration registers to their EEPROM state, which may overwrite modified RAM configuration

#### 4.2.2.2 To CLEAR the MONITOR\_BACKUP register:

- 1) sequence down to put the state machine in a safe state
- 2) issue a CLEAR command to clear both the state machine and the status and fault registers
- 3) issue a STORE command to store the cleared registers to EEPROM

NOTE: this will also store all config registers from RAM to EEPROM, potentially storing modified configuration settings

- 4) issue a RESTORE command to retrieve the cleared MONITOR\_BACKUP register and STATUS\_INFORMATION[12] bit into RAM to indicate their clear state

### 4.2.3 If the I2C bus is not used, how do I tie the pins on the board?

The LTC2937 has EEPROM to make it autonomous, so it does not require any bus communication to operate correctly in a system. Once it is programmed, it can power-up and perform all of its duties automatically. This implies that I don't need to hook-up the I2C bus, right?

NO!

It is true that the LTC2937 operates autonomously, and does not need the I2C bus under normal operating conditions, HOWEVER, there are still very important modes where the bus is used.

The LTC2937 contains a set of status registers that allow system debugging in the event of a problem. These registers are available on the I2C bus.

LTpowerPlay communicates with the LTC2937 (and all other PSM devices) through the I2C bus. There are many reasons to use LTpowerPlay, during prototype system testing and debug, during board bring-up, and during in-system fault debug. If something goes wrong, or if a question arises, LTpowerPlay can help.

If the system requirements ever change, the I2C bus is the simplest way to access the EEPROM and make changes.

THEREFORE: Always hook-up the I2C bus to the LTC2937.

The DC1613 dongle is the most convenient connection type. It uses a 12-pin header. If board space is a premium, use the 4-pin connector that only connects GND, SCL, SDA, and VDD.

#### 4.2.4 What do I do with unused channels on the LTC2937?

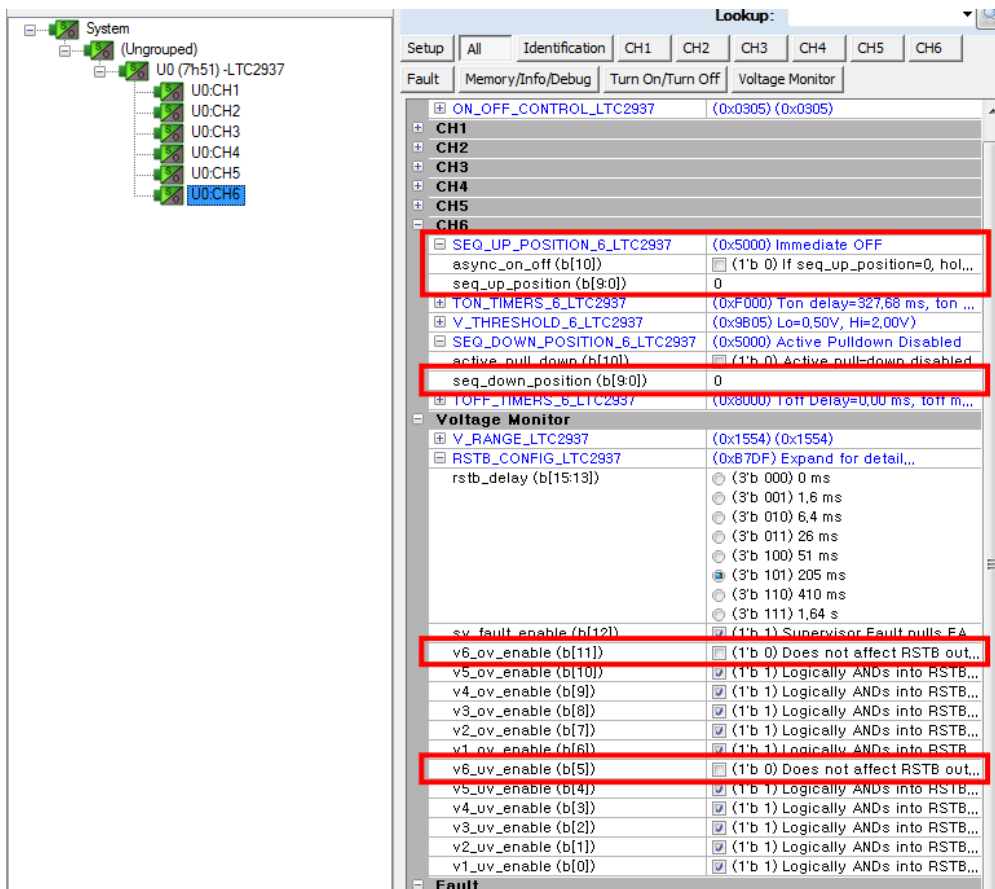
If a channel 'n' is unused, do the following:

##### TIE PINS

| PIN | STATE           |
|-----|-----------------|
| Vn  | ground          |
| ENn | float or ground |

##### SET REGISTERS

| REGISTER                  | VALUE | COMMENT                              |
|---------------------------|-------|--------------------------------------|
| SEQ_UP_POSITION_n[9:0]    | 0x000 | sequence-up position = 0             |
| SEQ_UP_POSITION_n[10]     | 1'b0  | channel ENn = 0; channel is disabled |
| SEQ_DOWN_POSITION_n[9:0]  | 0x000 | sequence-down position = 0           |
| RSTB_CONFIG(vn_ov_enable) | 1'b0  | Vn OV comparator doesn't matter      |
| RSTB_CONFIG(vn_uv_enable) | 1'b0  | Vn UV comparator doesn't matter      |



#### 4.2.5 What happens if I place an un-programmed (factory default) LTC2937 on my board?

Before you program the LTC2937 it will do nothing. The factory default settings in the LTC2937 take the philosophy of "do no harm." This means don't turn anything on, don't sequence, don't do anything until told otherwise.

The factory default setting for the ON\_OFF\_CONTROL register tells the part to ignore the ON pin input, and to remain in the sequence down state. In this state the ENn pins pull down as soon as the VDD supply is high enough to turn-on the NMOS pull-down FETs inside of the LTC2937. When VDD is low (not powered), these pins are Hi-Z.

Here are the default output pin states at power-up:

| PIN NAME  | FACTORY DEFAULT POWER-UP STATE |
|-----------|--------------------------------|
| ENn       | pulled low                     |
| RSTB      | pulled low** (probably*)       |
| FAULTB    | pulled low** (probably*)       |
| SPCLK     | pulled low (not sequencing)    |
| SHARE_CLK | begins pulsing **              |

| PIN NAME    | FACTORY DEFAULT POWER-UP STATE  |
|-------------|---------------------------------|
| VDD         | 3.3V LDO output                 |
| Vn (inputs) | Hi-Z; active pull-down disabled |

\*may depend upon the state of the Vn inputs

\*\*some pins may be shared, so may be affected by other devices

#### 4.2.6 What is the "boot-up" time of the LTC2937?

2ms.

When the LDO powers-up after VIN voltage exceeds 4.5V, the LTC2937 loads register settings from EEPROM into RAM. The EEPROM read process takes 2ms. Following this the device is ready, and begins operation based upon its register settings and inputs.

#### 4.2.7 Why is Linduino having trouble communicating with the DC2313A on the I2C bus?

One possibility is that you have selected address 0x50 or 0x51 for your LTC2937, using ASEL2,1,0 = HI,HI,HI, or HI,HI,Z.

Unfortunately, two things are conspiring against us here. First, the [Linduino One](#) platform shorts together the I2C main bus and the AUX I2C bus. Normally these buses are independent of each other, and address collisions are impossible, but as soon as you plug-in a Linduino, the two buses become one. Second, the DC2313A board contains a standard 24AA02 identity EEPROM, which talks on the AUX I2C bus, and which answers to addresses 0x50, 0x51, ..., 0x57 (the lowest three bits of the address are essentially ignored). Since the LTC2937 can be commanded to answer to addresses 0x50 and 0x51, we have the potential for address collisions when a Linduino is the bus master. Avoid address collisions by selecting any LTC2937 address other than 0x50 (ASEL=HI,HI,Z), or 0x51 (ASEL=HI,HI,Z).

NOTE: This is not a problem for the DC2313A demo board communicating with a DC1613 dongle.

NOTE: This is not a problem for the LTC2937 in a customer system (assuming they have done proper address planning).

### 4.3 LTC2936 Apps FAQ

This Section contains Frequently Asked Questions about the LTC2936

#### 4.3.1 Why is Linduino having trouble communicating with my DC1605B on the I2C bus?

One possibility is that you have selected address 0x50, 0x51, 0x52, 0x53, 0x54, or 0x55 for your LTC2936, using ASEL1 != H.

Unfortunately, two things are conspiring against us here. First, the [Linduino One](#) platform shorts together the I2C main bus and the AUX I2C bus. Normally these buses are independent of each other, and address collisions are impossible, but as soon as you plug-in a Linduino, the two buses become one. Second, the DC1605B board contains a standard 24AA02 identity EEPROM, which talks on the AUX I2C bus, and which answers to addresses 0x50, 0x51, ..., 0x57 (the lowest three bits of the address are essentially ignored). Since the LTC2936 can be commanded to answer

to addresses 0x50 - 0x55 (as well as 0x58, 0x59, and 0x5A), we have the potential for address collisions when a Linduino is the bus master. Avoid address collisions by selecting an LTC2936 address with ASEL1 = H. This will command an I2C address outside of the collision range.

NOTE: This is not a problem for the DC1605B demo board communicating with a DC1613 dongle (not using Linduino).

NOTE: This is not a problem for the LTC2936 in a customer system (assuming they have done proper address planning).

#### 4.4 "Add Default LTCXXXX Chip" button in LTpowerPlay does not add datasheet default register settings; why?

LTpowerPlay assumes the philosophy that it should be easy for the user to add a new chip and begin manipulating the registers in a meaningful way. It does not simply use datasheet default register values, since these may not provide the best user experience in setting-up a project file. Because the LTC2933 and LTC2936 are very generic in their function within a larger system, the datasheet defaults have very little value as a starting point for building a useful project file. It is better to start with obviously generic values that the user will need to change for a real system.

## 5 LTpowerPlay FAQ

This FAQ Section contains Frequently Asked Questions about the LTpowerPlay Graphical User Interface

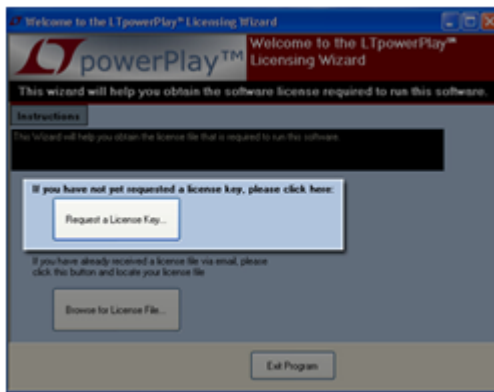
### 5.1 Can I license LTpowerPlay if my computer is not connected to the internet?

Yes. Just follow these steps:

Install LTpowerPlay (it will automatically launch)

If you have a DC1613A connected to your computer no license is required – the GUI will start automatically

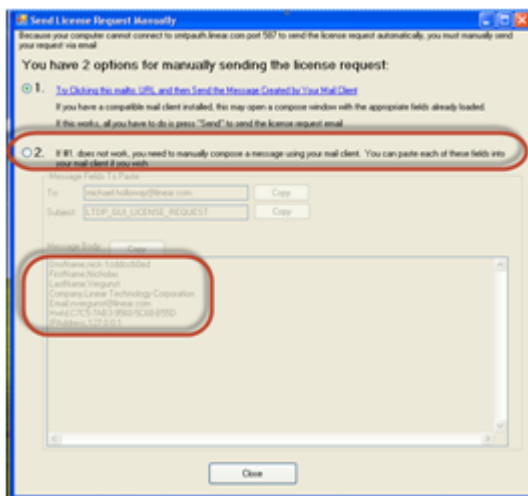
Otherwise, click ‘Request a License Key’ to start the process.



Click ‘Send License Request...’ after all requested information is filled in.



Because your computer is not connected to the network, the initial automated method will timeout and fail after some time. The following window will be displayed:



Select Option #2 (manual email request)

Copy the text into the body of an email with a subject titled “LTDP\_GUI\_LICENSE\_REQUEST” and send the mail to [licenserequest@ltpowerplay.com](mailto:licenserequest@ltpowerplay.com)<sup>20</sup>

NOTE: Licenses are handled automatically except for this type of license. These are handled manually. Please allow for approximately 1 business day for processing.

## 5.2 Can I have multiple versions of the GUI simultaneously installed on one computer? Can I 'snapshot' an old version of the GUI before updating so that I can run the old version OR the new one?

Of course! More details below

Whenever you launch any copy of the GUI it will automatically check the network for an update and prompt you to update. If you would like to, you can archive or snapshot your existing version of the GUI before updating as follows:

- When you launch the GUI, select 'no' when prompted to update to the newer version
- Open a new copy of Windows Explorer (file browser)
  - Navigate to the folder c:\<Program Files>\Linear Technology\
  - Right click on the LTpowerPlay folder, and select 'Copy'
  - Right click in an empty area next to the LTpowerPlay folder and select 'Paste' to make a copy of this folder and all files/subfolders within it. This is your archived version of the GUI
  - Now select this new folder (i.e. 'Copy of LTpowerplay'), right click and select 'Rename'
  - Type a name for this archived version. For example 'LTpowerPlay - 1.0.263.0'
- You have successfully created an archived version of the GUI. Here are a couple of notes to remember:
- If you would like to update your GUI you can re-launch the GUI from the start menu and accept the automatic update to update the version in c:\<Program Files>\Linear Technology\LTpowerPlay
  - You can always manually run the old version by navigating to the archive folder and running LTpowerPlay.Shell.Exe out of that folder. For example, in our scenario, navigate to c:\<Program Files>\Linear Technology\LTpowerPlay - 1.0.263.0\ using windows explorer, and double click 'LTpowerPlay.Shell.exe'

<sup>20</sup> <mailto:licenserequest@ltpowerplay.com>



- When you run the archived version, it will likely prompt you to update your GUI when it launches because it is comparing it's version to the latest version on the network and sees that there is an update. **Remember to deny the automatic update request when you launch the archived version.**
- The shortcuts on the desktop and start menu will always point to the latest version. If you like you can manually create a shortcut to the archived version so that you can conveniently launch the old version from the desktop. Just follow these instructions:
- Open windows explorer and navigate to the archive folder. For example: 'c:\<Program Files>\Linear Technology\LTpowerPlay - 1.0.263.0\'
  - Select the file LTpowerPlay.Shell.exe
  - Right click on this file and select 'Create Shortcut'. This creates a shortcut with the name 'LTpowerPlay.Shell.exe - Shortcut' or similar.
  - Right click on this shortcut file and select 'Rename'
  - Rename the file to 'Run LTpowerPlay 1.0.263.0' or similar

You may drag this shortcut onto the desktop or start menu to create a convenient shortcut to quickly launch the archived version

## 5.3 How do I get detailed help for the selected register in LTpowerPlay?

There are a couple ways to get help on registers within the LTpowerPlay environment. Both can be accessed by selecting a register in LTpowerPlay, and subsequently pressing 'F1' on your keyboard. Both External Datasheet help and the built-in Command Help windows are shown. The example below provides further detail.

### 5.3.1 Example

The example shown is for the MFR\_PWRGD\_EN register for a LTC2977:

#### 5.3.1.1 Selecting a Register and Launching Help:

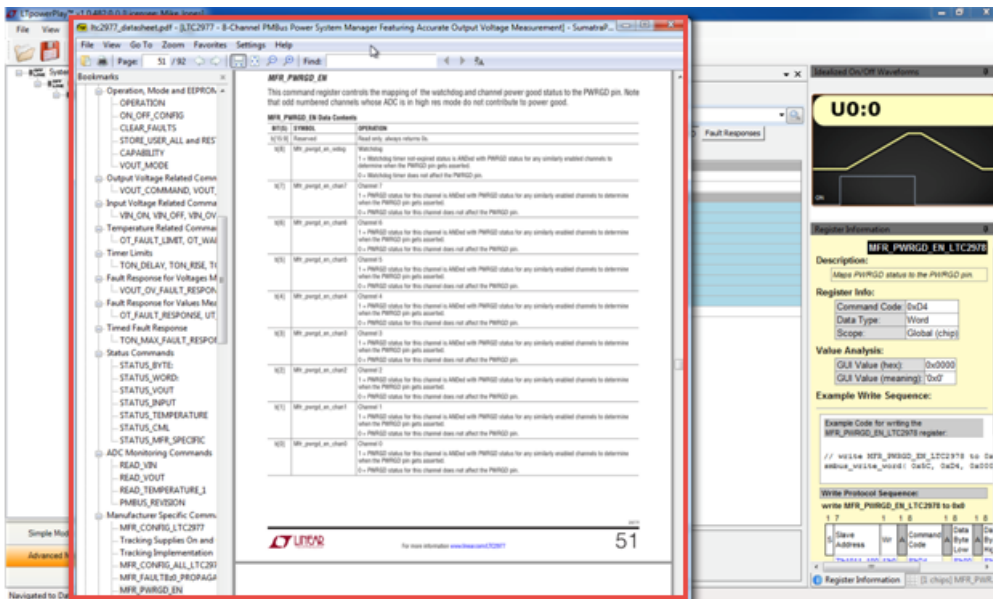
- Launch LTpowerPlay, work offline, and select the LTC2977 chip type
- Select the Watchdog/PGOOD Configuration Group
- Select the MFR\_PWRGD\_EN register
- Press 'F1' on your keyboard

#### 5.3.1.2 Results

Two forms of help are automatically launched when you press F1. They are discussed below.

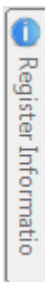
##### Detailed Datasheet Help


The datasheet for the relevant IC is automatically launched (as an acrobat PDF file). Then the viewer will automatically navigate to detailed section of the datasheet for the selected register. The example image shows the result of this highlighted in red below:



### 5.3.2 Register Information View

The Register Information View is also automatically launched when you press F1. This peripheral display window is normally hidden on the right side of your display. The view is automatically expanded and shown when you press F1. You can also expand it manually by moving your mouse over the icon shown below:



Once Expanded, you may also wish to 'pin' the window so that it does not hide away when you move focus elsewhere. You can do this by pressing the pushpin icon in the upper right side of the view: 

The Register Information View contains a number of useful items as shown in the below image:

1. Register Name
2. Description of Register
3. Register Information
  - a. I2C Command Code
  - b. Data Type
  - c. Scope (Global/Paged)
4. Value Analysis
  - a. GUI Value in Hex
  - b. Human Meaningful Interpretation (when available)
5. Example Write Sequence
  - a. C code to write register using SmBus transactions
  - b. Write Protocol Sequence: A detailed protocol diagram that shows bit-by-bit and byte-by-byte how the transaction should look on the bus
6. Example Read Sequence
  - a. C code to read register using SmBus transactions
  - b. Read Protocol Sequence: A detailed protocol diagram that shows bit-by-bit and byte-by-byte how the transaction should look on the bus

**Register Information**

**Description:**  
Maps PVRGD status to the PVRGD pin.

**Register Info:**  
Command Code: 0x04  
Data Type: Word  
Scope: Global (chip)

**Value Analysis:**  
GUI Value (hex): 0x0000  
GUI Value (meaning): '0x0'

**Example Write Sequence:**

Example Code for writing the MFR\_PVRGD\_EN\_LTC2978 register:

```
// write MFR_PVRGD_EN_LTC2978 to 0x0
ambus_write_word( 0x5C, 0x04, 0x0000 );
```

**Write Protocol Sequence:**  
write MFR\_PVRGD\_EN\_LTC2978 to 0x0

| 1 | 7             | 1   | 8            | 1 | 8             | 1 | 1              |
|---|---------------|-----|--------------|---|---------------|---|----------------|
| S | Slave Address | Wr  | Command Code | A | Data Byte Low | A | Data Byte High |
|   | 7b1011_100    | 1b0 | 8h04         |   | 8h00          |   | 8h00           |

**Example Read Sequence:**

Example Code for reading the MFR\_PVRGD\_EN\_LTC2978 register:

```
// read MFR_PVRGD_EN_LTC2978
ambus_read_word( 0x5C, 0x04 );
```

**Read Protocol Sequence:**  
read MFR\_PVRGD\_EN\_LTC2978

| 1 | 7             | 1   | 8            | 1 | 7  | 1             | 8   | 1               | 8                | 1 | 1 |
|---|---------------|-----|--------------|---|----|---------------|-----|-----------------|------------------|---|---|
| S | Slave Address | Wr  | Command Code | A | Sr | Slave Address | Rd  | Data A Byte Low | Data A Byte High | N | P |
|   | 7b1011_100    | 1b0 | 8h04         |   |    | 7b1011_100    | 1b1 |                 |                  |   |   |

**NOTE:** The information shown in the Register Information View is contextually specific. In this example, because device address 7'h5C is selected in the tree, and the present value of the register is 0x0000, the views adapt to represent how to write this specific value to the device at this address. If the register were paged, an appropriate page write sequence (for the page selected in the system tree) would be included in the write and read sequences.

## 5.4 How do I translate settings shown in to LTpowerPlay into I2C command sequences?

Customers with host controllers implementing I2C/PmBus capability often want to issue command sequences over the bus to achieve specific functionality. We have a number of resources for such customers writing firmware (see [Firmware FAQ](#)(see page 100)), including example Linduino code.

However, if you are running LTpowerPlay, you already have several tools at your fingertips that can help you interactively develop such command sequences, and translate these sequences to bus transactions. These tools are covered by other FAQ items, linked below.

Click a link below for more information:

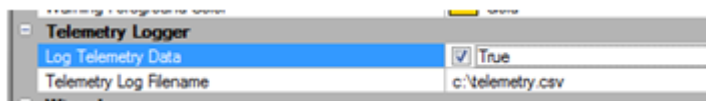
| Feature  | Overview   |
|--|--|
| <a href="#">F1 Datasheet/Command Help Lookup</a> (see page 49) | By selecting a register and pressing F1, you can quickly navigate to detailed register information in the datasheet and LTpowerPlay.   |
| <a href="#">Command Help Window</a> (see page 50)              | A detailed window on the periphery of LTpowerPlay that 'watches' the actively selected register and shows contextual help including summary information, example code, and detailed protocol diagrams for writing and reading the register over the bus. |

| Feature   | Overview   |
|---|--|
| <a href="#">Script Recording Tool</a> (see page 55) | A recording function that can record individual command writes/reads to a simple text file, so that when you write individual commands in LTpowerPlay you can collect them into a 'script file' that can be used to efficiently capture and communicate the command sequence to an I2C-savvy engineer. These script files can also be 'played back' to emulate sending the command sequence from your own host controller to aid firmware development. |

## 5.5 Can LTpowerPlay Log Telemetry data to a file?

Yes. LTpowerPlay has a built in “Telemetry Logger” function. You enable it as follows:

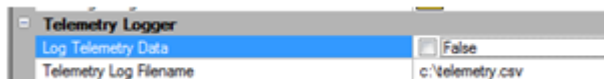
- Select “View > Preferences” on the menu
- Find the “Telemetry Logger” Category near the bottom of the preferences window
- Under ‘Telemetry Log File’, type in the .csv file name you would like to log to into the (no browse function at this point)
- Set the ‘Log Telemetry Data’ checkbox to ‘True’



- Click ‘OK’ to save the preferences

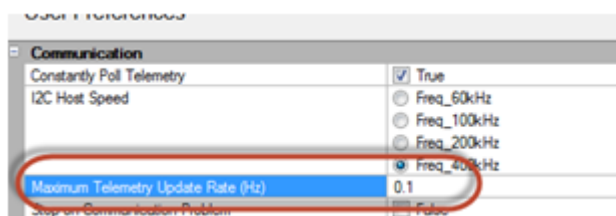
To later disable logging

- Select “View > Preferences” on the menu
- Under the ‘Telemetry Logger Category, set the ‘Log Telemetry Data’ checkbox to ‘False’
- Click ‘OK’ to save the preferences



Notes:

- The Telemetry Logger logs all telemetry data received to a large CSV file.
- Columns indicate the chip/channel and parameter name.
- Each Row contains all parameters sampled in a single polling loop for the system
- These files can get large if you leave the feature enabled for many days.
  - You can limit the polling rate by reducing the ‘Maximum Telemetry Update Rate (Hz)’ under the “Communication” category of the preferences
  - The below example restricts polling to once every 10 seconds:



- Recording remains enabled until you subsequently:
  - Set the ‘Log Telemetry Data’ checkbox to ‘False’

- Click OK to save the preferences
- There is presently no way to select or filter a subset of parameters to record (it just records all parameters for the entire system) – you can use Excel or other tools to filter data in post processing

### 5.5.1 A Note On .CSV Data Viewer Tools

There are many options available for viewing .csv files. You may already have your own favorite option. Many people prefer Excel because they have it and it's pretty easy to use for small datasets. Though Excel can be convenient for one time view of very small datasets, it has a number of limitations for both large datasets and for continuously monitoring a file that's being actively written (it basically can't). If you are using Excel, it is recommended that you make a copy of the .csv file (copy/paste in windows explorer) and then open the COPY of the .csv file. This is because Excel Locks the .csv file when you open it, so you cannot open the .csv file that LTpowerPlay is actively recording to. To avoid lots of unwanted errors, just make a copy and open it in Excel.

#### 5.5.1.1 Microsoft DataSet Viewer

If you need to handle large datasets, a better tool that's up for these challenges is the Microsoft DataSet Viewer (included in the Scientific Dataset Library), from the Microsoft Research Group. You can find this tool here:

<http://research.microsoft.com/en-us/um/cambridge/groups/science/tools/datasetviewer/datasetviewer.htm>

Advantages:

- It's easy to download/install (no Java Required)
- It automatically lists columns as parameters, so you can more conveniently find the parameter you want to plot
- It handles VERY large datasets well

Disadvantages:

- Doesn't handle gaps in data (must manually remove empty rows)

When opening a .csv file in Dataset Viewer:

- Make a copy of the .csv file (do not open a file that LTpowerPlay is actively recording to)
  - Edit the file and manually remove any empty rows
- Launch DataSetViewer
- Click "File > Open" on the menu
- Browse to the copy of the .csv file
- Delete the default plot added
- Click 'Add Plot...'
- Select Type = 'Polyline'
- Select the parameter you want to plot as the Y variable (for example "READ\_TEMPERATURE\_1\_PAGED-U1:0")
- Select "Time" as the X variable
- NOTE: You can repeat the above (from Add Plot.. down) to add more plots on the same Time axis, if you want to view multiple parameters)

#### 5.5.1.2 LiveGraph

If you want to continuously monitor the .csv file as LTpowerPlay records to it, another nice tool is called LiveGraph (<http://www.live-graph.org/download.html>). In addition to handling larger datasets, this tool can also actively monitor the .csv file as LTpowerPlay records to it.

**Advantages:**

- Handles larger data sets
- Can select individual columns/parameters to plot fairly easily
- Can continuously update plots while LTpowerPlay is recording the .csv file
- Quicker to add a plot than DataSetViewer

**Disadvantages:**

- Not as complete/capable as DataSetViewer
- Requires JAVA (separate sizable download) to be installed (<http://download.org/gus/download/java2.php>)

When using LiveGraph, here's how to plot the CSV file:

- Install Java Runtime first (see above link)
- In Windows Explorer Double Click the LiveGraph .jar file. This launches the program using Java
- Find the "Data file settings" window (top left)
  - Click Open... and browse to the .csv file that LTpowerPlay is recording to (this tool can handle reading from a 'live' file)
- Under the Data Series settings window (bottom right)
  - Click Hide All to keep from plotting all the parameters in one plot
  - Find the parameter(s) you wish to plot and check them under the "Show" Column (left)
- Find the "Data file settings" window (top left)
  - Click "Update Now" to update the plot
  - If you wish to automatically update the plot periodically:
    - Modify the update frequency as desired (Not recommended to update more than once every few seconds)
    - A countdown timer will show you how many seconds until the next update

## 5.6 Can I call LTpowerPlay from the command-line to program parts in system?

Yes. LTpowerPlay can be run with command-line arguments to perform in system programming or run script files. This could be appropriate in situations where you want to use LTpowerPlay and the DC1613A dongle to program devices in system or run I2C scripts like those in the I2C Utility dialog, but you do not want to use the GUI interactively. You could call the command-line from a batch or other automation environment for example, including Matlab, perl, pything, DOS BATCH, etc

- Launch and update LTpowerPlay if prompted
- Launch a command prompt DOS shell (start, run, 'cmd.exe')
- Type 'cd c:\Program Files (x86)\Linear Technology\LTpowerPlay\' or similar, depending on where you installed LTpowerPlay
- Type LTpowerPlay.shell.exe -help

This will display instructions. Note in particular Example #1 below which shows you how to program and Verify a set of devices in system from a windows command-prompt.

```
Administrator: C:\Windows\system32\cmd.exe

C:\Program Files (x86)\Linear Technology\LTpowerPlay>LTpowerPlay.Shell.exe -help

C:\Program Files (x86)\Linear Technology\LTpowerPlay>LTpowerPlay.Shell.exe version 2.0.0.0
Copyright (c) Linear Technology

Options:
-Comment, -c          Sets a user comment. Applies to: [-snap]
-DumpAll, -d          Dump All EEPROM contents to a dump
                      file (to send to Linear Technology MFR)
-Help, -h             Displays this help text
-Program, -p          Program the target system
-ProjectFile, -proj   Specifies the project file
-RunScript, -script   Specifies an I2C Script file to run
-SnapTelemetry, -snap Appends a snapshot of telemetry data to the specified
                      .csv file.  REQUIRES [-ProjFile] for column names
-TargetAddress, -t    Target's 7-bit slave address
                      (for dump operation), i.e. "0x5C"
-Verify, -v           Verify the target system

Example 1: Program and Verify the file 'c:\temp.proj' in-system using the dongle
:
>LTpowerPlay.Shell.exe -Program -Verify -ProjectFile="c:\temp.proj"

Example: To Run the I2C script file c:\myscript.txt:
>LTpowerPlay.Shell.exe -RunScript="c:\myscript.txt"
```

## 5.7 Can I use LTpowerPlay to run 'scripts' or sequences of I2C commands?

Yes. There are two aspects to doing this. The first is using the script recorder feature to 'record' a command sequence, and the second is using the I2C script file utility to playback the script file. You can also call LTpowerPlay from the command-line with parameters to run scripts from other environments.

### 5.7.1 Recording a Script

#### 5.7.1.1 Enable Script Recording

To record a script, do the following:

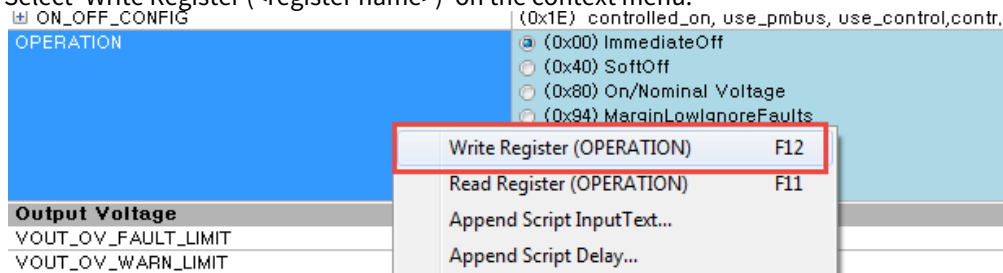
- Select View > Preferences on the LTpowerPlay menu
- Navigate to the 'Development' Section of preferences
- Optionally change the Script Filename (the script will be recorded to the specified location)
- Check the box next to 'Record Script File' to enable recording
- Click 'Close'

### 5.7.1.2 Recording Command Sequences

Once recording is enabled, LTpowerPlay will append script lines to the file for each individual register write or read you perform using the context menu. You can do this in one of two ways.

1. Using the Context Menu

- a. Select a register (OPERATION register shown in example below)
- b. Press the right mouse button
- c. Select 'Write Register (<register name>)' on the context menu:



- d. Alternately select Read on the context menu to read the register

2. Using the Keyboard

- a. Select a register
- b. Press F12 to write only the selected register to hardware and record a line to the script file
- c. Alternately press F11 to read only the selected register from hardware and record a line to the script file

*NOTE: Only individual read and writes are appended to the script file. For example, LTpowerPlay does not record the telemetry polling reads to the file, and does not record all registers written when you press 'Write All' on the toolbar. This allows you fine grained control when building command sequences.*

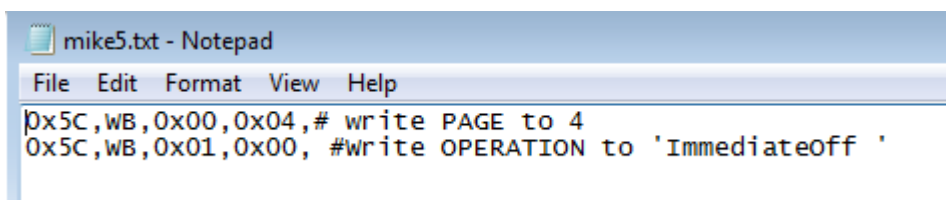
### 5.7.1.3 Disable Script Recording

When you are done building your script, to stop recording register writes and reads to the script file, follow these steps:

- Select View > Preferences on the LTpowerPlay menu
- Navigate to the 'Development' Section of preferences
- **Uncheck** the box next to 'Record Script File' to enable recording
- Click 'Close'

### 5.7.2 Example Script File

The image below shows the two lines that are appended to the script file when the OPERATION register is individually written as shown above





### 5.7.3 Playing Back a Script File

To playback a script file, follow these steps:

- On the LTpowerPlay menu, select 'Utilities > I2C Utility...'
- Select the 'Command Files' tab
- Click '...' and browse to your recorded script file
- Uncheck the box titled 'Substitute Selected Chip' (unless you want to substitute all addresses in the script file with the presently selected chip in the system tree)
- Click 'Run Cmd File'

#### 5.7.3.1 Results

The Log window shows the results of the script file. If the script executes without any errors (No NACKs, read values as specified, etc), the window will be green. Otherwise it will be red and you can consult the detailed log for the results.

### 5.7.4 Further Help

The script recording and playback functionality is an advanced feature. Not all of its capabilities are discussed here. For further information, [watch this video](#)<sup>21</sup>.

## 5.8 Where do I download the Windows drivers for the DC1613A?

The latest Windows drivers for the DC1613A can be downloaded from this location:

<http://www.ftdichip.com/Drivers/D2XX.htm>

The top right column in the table typically contains a link to a WHQL certified driver that you can download as a setup executable (.exe)

## 5.9 Can I use LTpowerPlay to decode raw fault log bytes read by my host controller?

Yes. Some context may be helpful. PSM Devices return a large number of bytes that represent a fault log. This is done via a PmBus 'block read' command. This large number of bytes must first be decoded in order to understand the meaning of the bytes. Decoding is different for each device (and in some cases dependent on the mode of the device). Depending on your situation you may want to decode these bytes in your host controller firmware, or just dump the raw bytes and use LTpowerPlay to decode.

This FAQ does not address situations where you wish to decode the fault log using your host controller code while 'in system'. If this is your scenario, you may wish to consult our Linduino reference code, which shows you how to do this yourself.

Some users prefer to simply capture the 'raw' bytes returned by a device, and later use LTpowerPlay to decode these bytes into meaningful fault log data. Follow these steps to accomplish this:

#### **Save the Fault Log from your Host Controller**

---

<sup>21</sup> <http://www.ltpowerplay.com/downloads/video/ltc2978/script-recorder-tutorial/script-recorder-tutorial.html>

- Read the 'raw' fault log bytes from the device
- Either save them as raw bytes or as an Intel Hex file (preferred)
- Transfer the file containing the raw bytes to a PC running LTpowerPlay
  - If you saved the file as raw bytes (.bin), use a bin2hex tool to convert the file to an Intel Hex file
  - It is recommended that you name the .hex file with the part number and/or device address so you can remember the source of the fault log

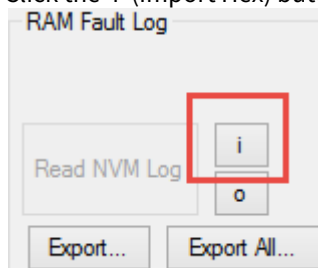
An example 'raw' fault log hex file for a LTC3880 fault log is shown below, for your reference:

**Example 'Raw' LTC3880 Fault Log Bytes (Intel Hex Format)**

```
:20000000FF421A160000000003700358027802DCA78DAF7DAE8DBC3DAFCDAF50036003487A6
:20002000EB87FBCA769B3300000841084100000035003480078011CA769B330000084108D3
:200040004100000036003580108017CA779B3300000841084100000035003480078007CAEB
:20006000769B33000008410841000000350034800487F6CA769B33000008410841000000A0
:1300800035003480038016CA769B3300000841084100004B
:00000001FF
```

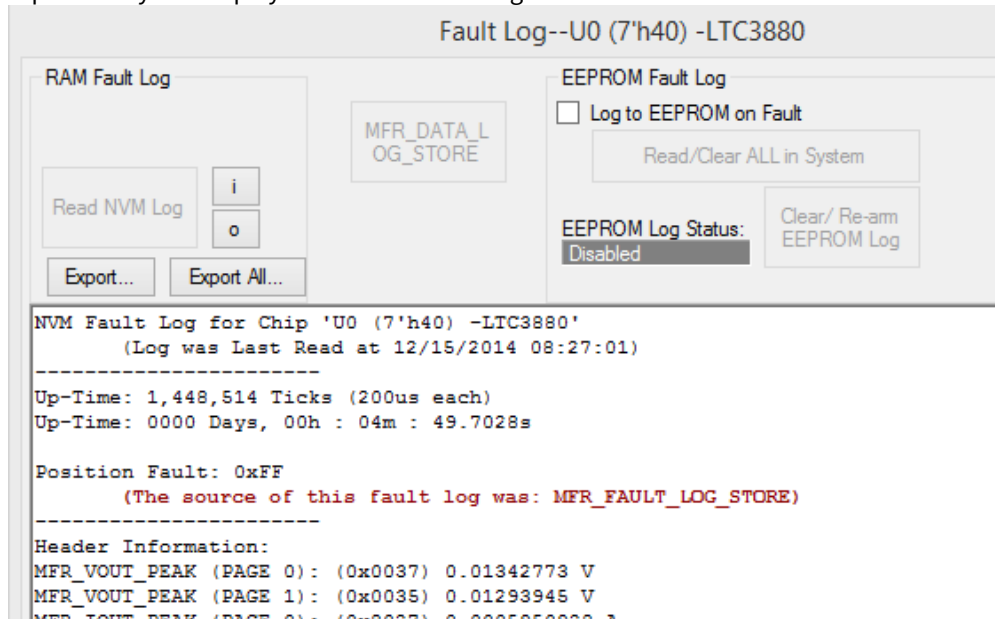
**Load the Fault Log Raw Bytes into the Fault Log tool in LTpowerPlay**

- In LTpowerPlay, load a project file that contains a configuration similar to the target device that generated the fault log (minimally, the project file should contain at least one device with the same model number as the one that generated the raw fault log bytes)
- Click "Go Online" in the toolbar
- Select the device in the system tree that represents the device from which the raw fault log bytes were originally read
  - This is important, because the file only contains raw fault log bytes, and the meaning of these bytes is different for each device type
  - Selecting a device in the tree informs LTpowerPlay which device type generated the original bytes, and thus how to decode them to get meaningful fault log information
- Display the Fault Log tool, by selecting "View > Windows > Fault Log" on the LTpowerPlay Menu
  - Click the 'i' (Import Hex) button near the top left side of this tool:



- Browse to the Intel Hex file that contains the raw Fault Log bytes, and Click "OK" to load the file into LTpowerPlay

- LTpowerPlay will display the decoded fault log in the GUI similar to the screenshot below:



**NOTE: At the time of writing, the fault log decode feature will not execute unless a DC1613A controller is connected to the USB port of your computer**

## 5.10 How do I convert a project file containing multiple LTC2977 devices into LTM2987s?

Though it's possible to start with LTC2977s and then convert the project file to LTM2987s (which contain 2x LTC2977s each), it is simpler if you know that you will use the LTM2987 devices to just add a LTM2987 device to your system tree from the beginning. If however you did not do that or you have an existing project file with LTC2977s and you wish to convert it to LTM2987s, you can do so by following this procedure:

- Launch LTpowerPlay and load your project file that contains LTC2977s
- Add a LTM2987 chip to the tree
  - Click Add Item to System Tree in the toolbar
  - Select the LTM2987 device type
  - Click 'Add Default LTM2987 Device' in this dialog
  - (Repeat the above for multiple LTM2987s)
- For each LTC2977 in your original project
  - Select a source chip from the top of your system tree (the LTC2977 from which to copy settings)
  - Right click the source device in the tree and select Copy
  - Select the appropriate LTM2987 A/B target that you just added (the LTM2987 sub chip that you want to copy settings to)
    - NOTE: Add LTM2987s as indicated above if you have not already
  - Right click the target device and select Paste
    - A warning will appear telling you that you are about to copy a large number of setting. Click OK to confirm
  - For the target LTM2987 A/B device, Right click the target device and change the address to match the source address you just copied from
  - For each channel of the target device
    - Click the label on the channel, and type the name to match the source channel name

- When you are done with this process, your project file will contain a set of LTC2977 devices and some new LTM2987 devices, and your LTC2977 device configurations should be copied to the appropriate LTM2987 sub device. Confirm that the configuration was copied properly:
  - In LTpowerPlay select "Utilities > Configuration / CRC..." in the menu
  - Inspect the 'Configuration Checksum' column for the first LTC2977 in your project
    - Find the corresponding LTM2987 sub device target to which you copied settings
    - Confirm that the target LTM2987 sub device has the same address as the source
    - Confirm that the target LTM2987 sub device has the same Configuration Checksum as the source
- Delete the unwanted LTC2977 devices in your system that you just migrated into LTM2987 devices
- Click "Save" in the toolbar and save the new project file with LTM2987s

## 5.11 Can I use LTpowerPlay to Merge Two Project Files into one?

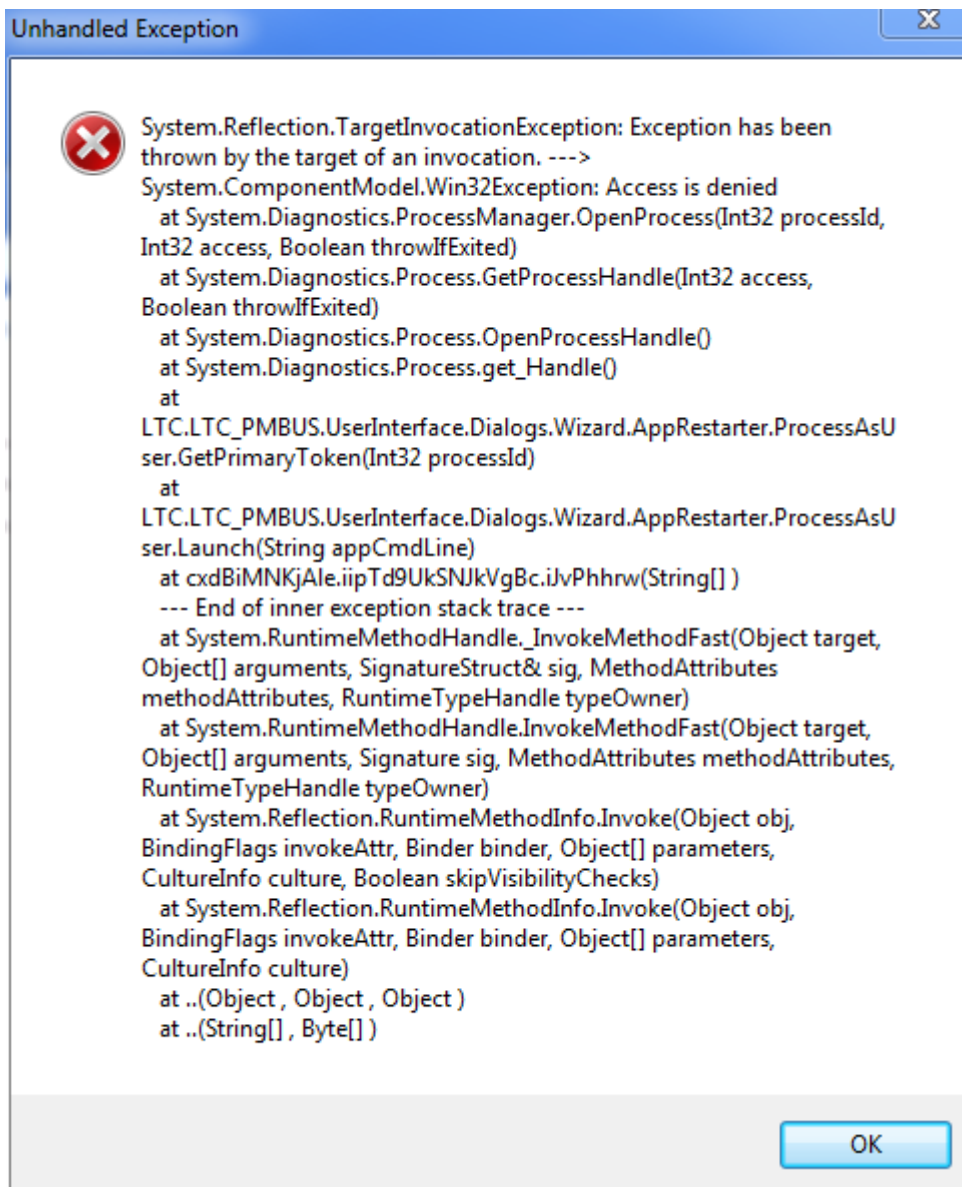
Project files can be merged using LTpowerPlay using the following procedure:

1. Start LTpowerPlay
2. Open the first project file
3. Select Append Project File... from the File menu
4. Select the second project file
5. Select Save... from the File menu and save to a new project file name

## 5.12 What does this error message encountered immediately after installation mean?

### 5.12.1 Problem:

Immediately after installing LTpowerPlay, I see an error message that looks similar to below:



What does this error mean and what can I do about it?

### 5.12.2 Solution:

This is a permissions issue that is isolated to the installer and should not persist after installation. The installer attempts to run the program after install. If it detects that it is running as a normal user instead of admin, it attempts to re-launch the executable as admin. This error occurs when permissions prevent this operation. It is likely not a concern

Please do the following.

- Close all error dialogs
- Relaunch LTpowerPlay from the normal desktop or start menu shortcut

If the above does not resolve the issue, please see <http://ltpowerplay.com/help/uac/><sup>22</sup> to adjust application compatibility settings.

### 5.13 Can I use LTpowerPlay to compare two project (.proj) files?

Yes. Let's take an example where you wish to compare an original project file a.proj (the reference) against another project file b.proj that may have changes relative to a.proj.

Follow this procedure:

- On the menu, select 'File > Open...'
- browse to a.proj
- On the menu, select 'File > Verify Configuration against Project File...'
- browse to b.proj

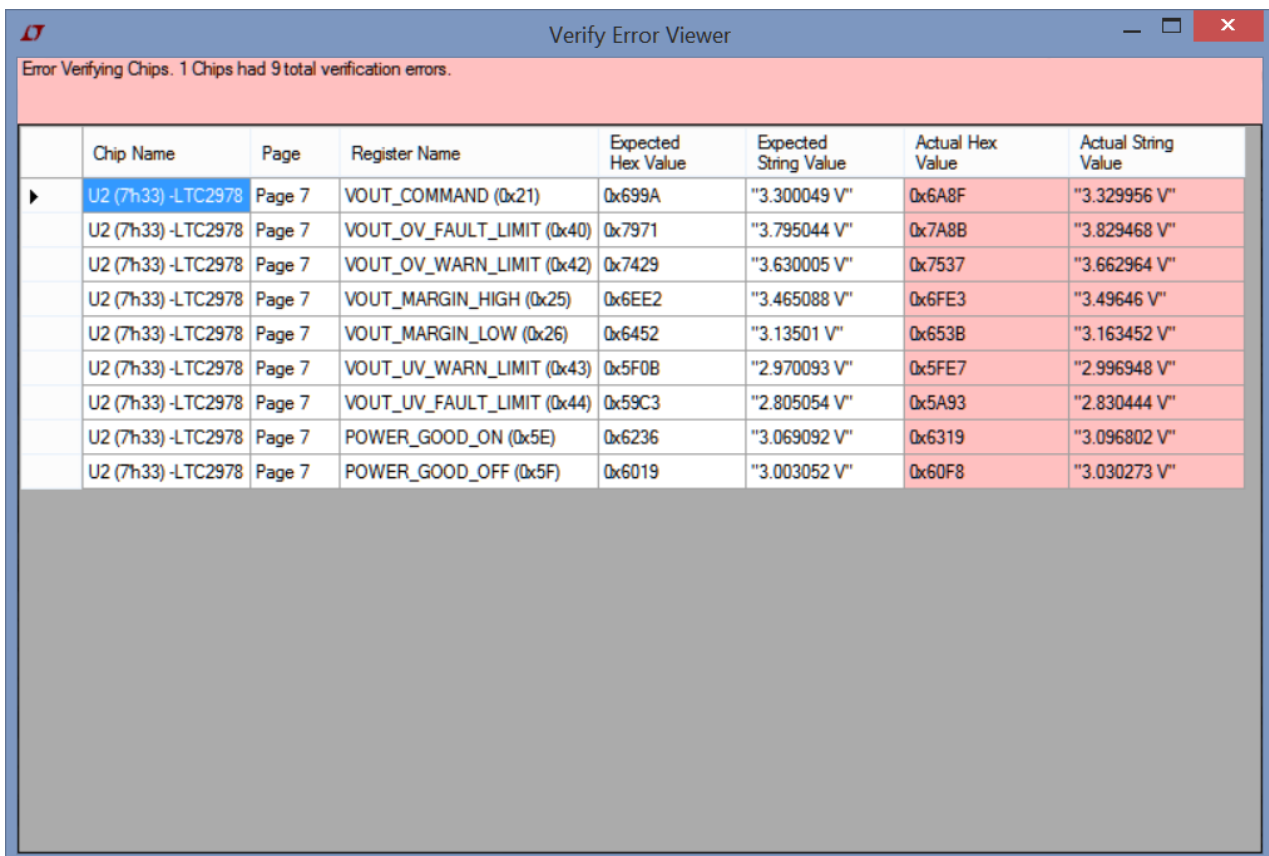
NOTE: The .proj files MUST have the same system tree (device addresses) for this to work.

LTpowerPlay will run a VERIFY procedure comparing a against b. Any differences will show up just as though you were comparing a.proj against hardware that contains the values in b.proj.

Let's say that you took project file a.proj and modified device U2:7's VOUT\_COMMAND (and related registers) from 3.3V (in a.proj) to 3.33V and saved the modified project as b.proj. If you ran the above procedure, you would see VERIFY errors as shown below:

---

<sup>22</sup>[http://cp.mcafee.com/d/k-Kr418g3zqb2r2bPzWr9KVJ5MsOCrhs7cLCQn1NEVhjpd79JdV5MsYeKrlBoDLdYjGuvkNFo570x-NVsSva7X7BPuCzCPXflfZvDbILnWZOWrdQ-hhovsjVqWdAkIkrFYG7DR8OJMddECQPt-jpoKMC--OCrKr01DPmGo-aN-BKnMDVWSdXr3t8QKsbicH78877\\_92JenelS5h7yvm9WSgQY54aJMJ\\_k-PsvFYjfNU6CPhOed7bzxI5zihEw2E9mGo-aN-Bzh1a59WuvYjh1qeQ9CSkjt7a5](http://cp.mcafee.com/d/k-Kr418g3zqb2r2bPzWr9KVJ5MsOCrhs7cLCQn1NEVhjpd79JdV5MsYeKrlBoDLdYjGuvkNFo570x-NVsSva7X7BPuCzCPXflfZvDbILnWZOWrdQ-hhovsjVqWdAkIkrFYG7DR8OJMddECQPt-jpoKMC--OCrKr01DPmGo-aN-BKnMDVWSdXr3t8QKsbicH78877_92JenelS5h7yvm9WSgQY54aJMJ_k-PsvFYjfNU6CPhOed7bzxI5zihEw2E9mGo-aN-Bzh1a59WuvYjh1qeQ9CSkjt7a5)



|   | Chip Name          | Page   | Register Name              | Expected Hex Value | Expected String Value | Actual Hex Value | Actual String Value |
|---|--------------------|--------|----------------------------|--------------------|-----------------------|------------------|---------------------|
| ▶ | U2 (7h33) -LTC2978 | Page 7 | VOUT_COMMAND (0x21)        | 0x699A             | "3.300049 V"          | 0x6A8F           | "3.329956 V"        |
|   | U2 (7h33) -LTC2978 | Page 7 | VOUT_OV_FAULT_LIMIT (0x40) | 0x7971             | "3.795044 V"          | 0x7A8B           | "3.829468 V"        |
|   | U2 (7h33) -LTC2978 | Page 7 | VOUT_OV_WARN_LIMIT (0x42)  | 0x7429             | "3.630005 V"          | 0x7537           | "3.662964 V"        |
|   | U2 (7h33) -LTC2978 | Page 7 | VOUT_MARGIN_HIGH (0x25)    | 0x6EE2             | "3.465088 V"          | 0x6FE3           | "3.49646 V"         |
|   | U2 (7h33) -LTC2978 | Page 7 | VOUT_MARGIN_LOW (0x26)     | 0x6452             | "3.13501 V"           | 0x653B           | "3.163452 V"        |
|   | U2 (7h33) -LTC2978 | Page 7 | VOUT_UV_WARN_LIMIT (0x43)  | 0x5F0B             | "2.970093 V"          | 0x5FE7           | "2.996948 V"        |
|   | U2 (7h33) -LTC2978 | Page 7 | VOUT_UV_FAULT_LIMIT (0x44) | 0x59C3             | "2.805054 V"          | 0x5A93           | "2.830444 V"        |
|   | U2 (7h33) -LTC2978 | Page 7 | POWER_GOOD_ON (0x5E)       | 0x6236             | "3.069092 V"          | 0x6319           | "3.096802 V"        |
|   | U2 (7h33) -LTC2978 | Page 7 | POWER_GOOD_OFF (0x5F)      | 0x6019             | "3.003052 V"          | 0x60F8           | "3.030273 V"        |

The values on the left (in white) show the original values from a.proj. The values on the right show the values from b.proj that differ. If the files are instead identical, a dialog will pop up that states there were no verification errors.

## 5.14 How do I update LTpowerPlay on a machine without internet access?

You can always find the latest public version here:

<http://ltpowerplay.com/updates/vin0/latest-version.php>

This page will display a link to the latest .msi installer. Follow these steps to update:

- Download the .msi installer provided in the above link
- Transfer the .msi to the machine without internet access (using a USB drive or similar)
- Run the .msi on a machine that has an earlier version of LTpowerPlay installed
- The .msi will automatically install over and upgrade an older version of LTpowerPlay

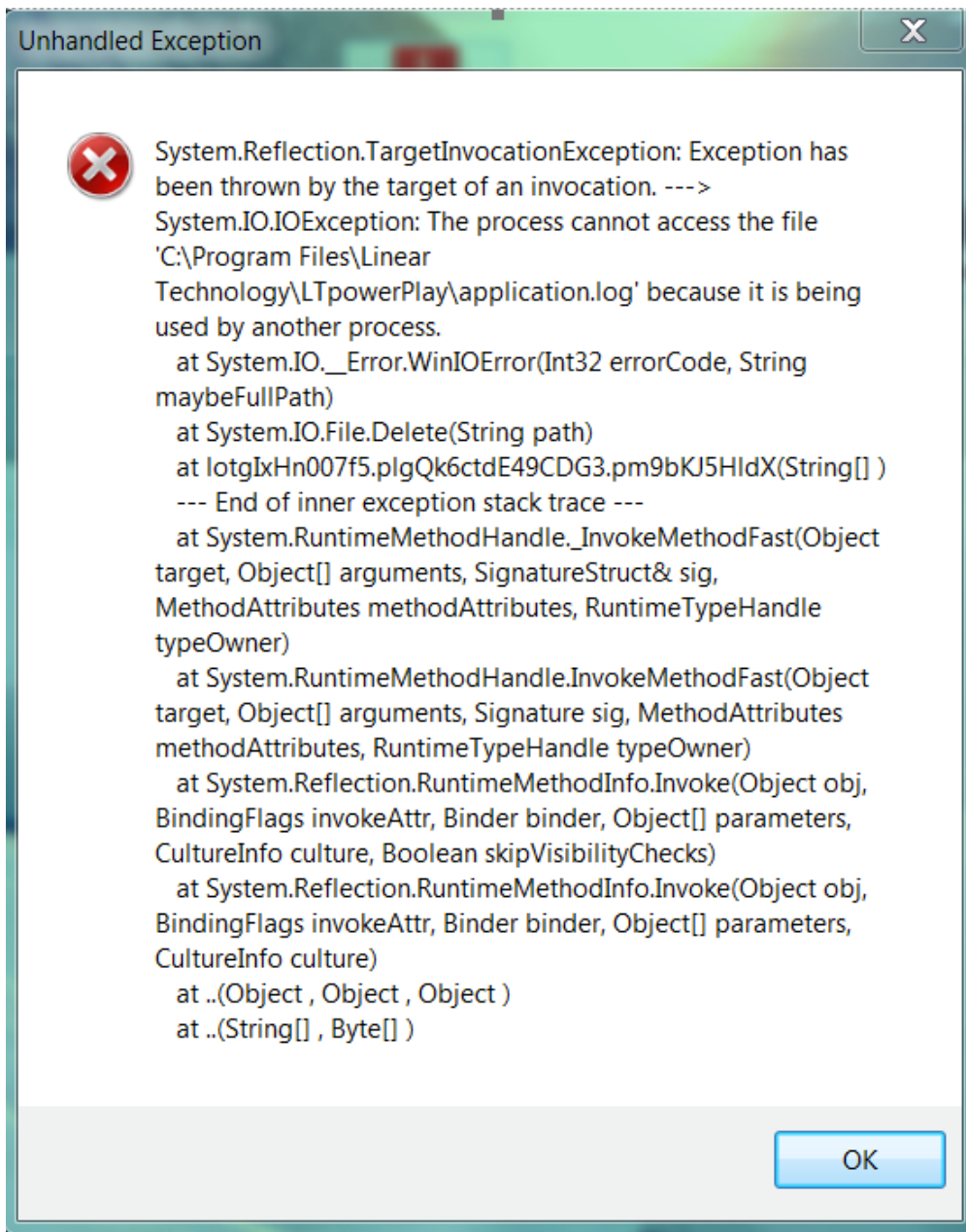
## 5.15 Can I use my .license file on another computer?

No. Licenses are "node locked" to a single physical machine. This means that if you attempt to use the .license file generated for one computer onto another computer, LTpowerPlay will reject the license file and ask you to apply for a new one. This is an easy and automated process. Just follow the instructions to request a new license for your new computer, and the license file will be sent to you automatically.

NOTE: If you perform major upgrades to the same physical machine (OS upgrade or perhaps major hardware upgrade), the hardware id may change, invalidating the old license file. If this occurs, LTpowerPlay will reject the old license and prompt you to fill in information to request a new one. Again, this process is automated so it should be pretty easy to get a new license after a machine upgrade.

## 5.16 What does this error message, seen immediately after launching LTpowerPlay mean?

Under certain circumstances, you may see an error similar to the one shown below immediately after attempting to launch LTpowerPlay:



This error may be associated with one of two issues:



1. The LTpowerPlay executable may not have been launched with sufficient permissions to generate the application.log file.
  - a. See <http://ltpowerplay.com/help/uac/> for instructions on how to resolve this issue
  - b. If this does not resolve the issue, check that you have permissions to modify the file (in this case application.log), and resolve the permissions issue.
2. You may have multiple instances of LTpowerPlay running at the same time, preventing one instance from accessing the log file
  - a. Launch the Task Manager
  - b. Select the "Processes" tab
  - c. Find all instances of LTpowerPlay.Shell.exe and select "Kill Process"

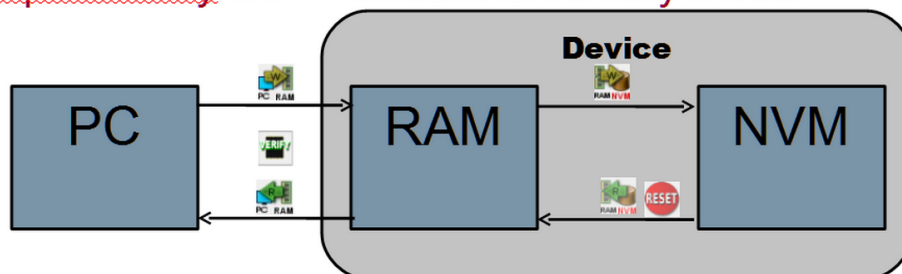
After resolving the issue with the appropriate method above, you should be able to launch LTpowerPlay successfully without error.

## 5.17 How to I use LTpowerPlay to 'burn in' settings permanently into EEPROM

The various devices that LTpowerPlay supports may contain hardware/RAM and EEPROM. Please note that all operations that transfer data are manual operations driven by clicking a toolbar button. One exception is the telemetry loop which runs constantly polling telemetry RAM and displaying values in LTpowerPlay when the user is online with hardware.

The below graphic depicts the LTpowerPlay Communication/Memory Model, and explains how settings are transferred between the GUI and Hardware/RAM, and also between Hardware/RAM and non-volatile-memory (NVM):

### LTpowerPlay Communication/Memory Model



- **Writes and Reads to Hardware are Manual Operations (toolbar buttons)**

- **Compare GUI to HW using VERIFY**

- **To Store all HW/RAM into NVM**

- **To Restore NVM setting into Hardware/RAM**


- **Or if the device contains a software reset**




## 5.18 Is there a simple way to read, decode and save all the fault logs on my board at one time?

Yes. It's called the System Snapshot tool. This tool provides a single-click method of storing all fault logs on the board into a single archive (.zip file). It's available on the menu (under View > Windows) and in the Fault Log Window. Follow the short procedure below to save all the fault logs:

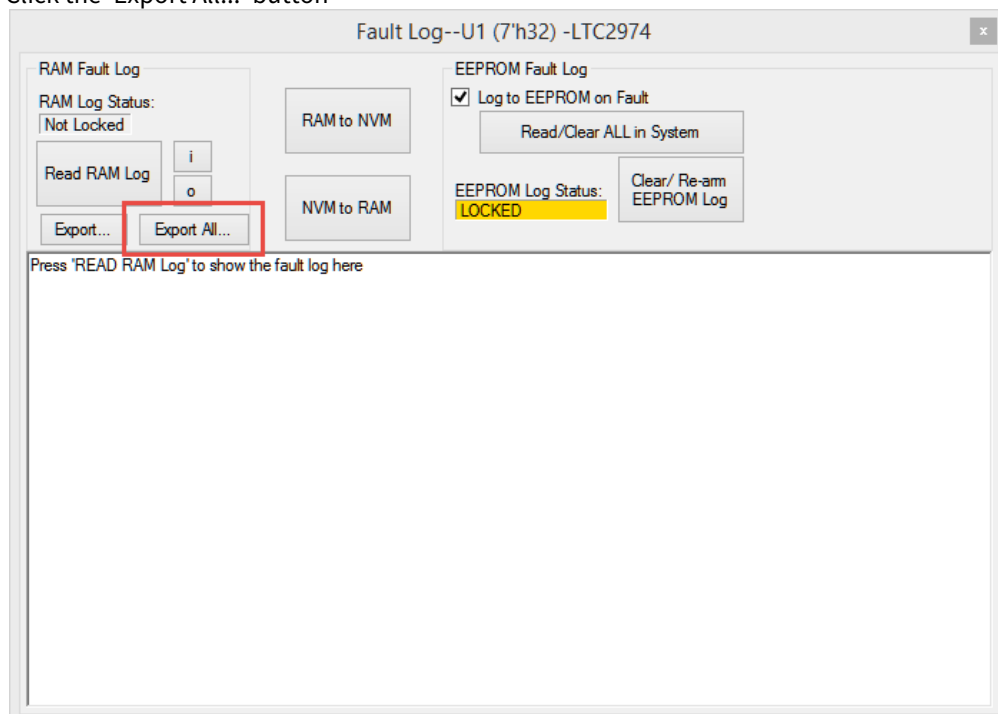
- Launch LTpowerPlay
- Load your project (.proj) file
- Go Online

- If the Fault Log button looks like this , there are no fault logs present in the system, and no further action is necessary

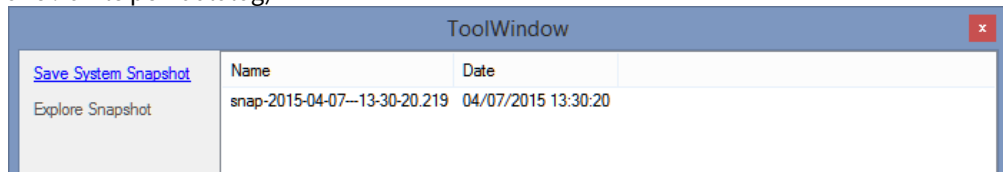
- If it looks like this , there are one or more chips in the system that contain non volatile fault logs. Continue.
- Click the Fault Log button
- If you have not already, select one of the chips in the system tree on the left-hand side of the GUI
- If the selected chip has a fault log, the EEPROM Log Status will indicate "LOCKED" as shown below:

EEPROM Log Status:  
**LOCKED**

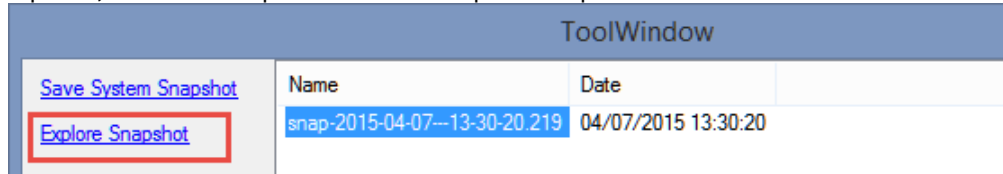
- Though you could read each individual fault log in the system one by one manually, there is a much faster way to save all fault logs to a .zip file:
  - Click the 'Export All...' button



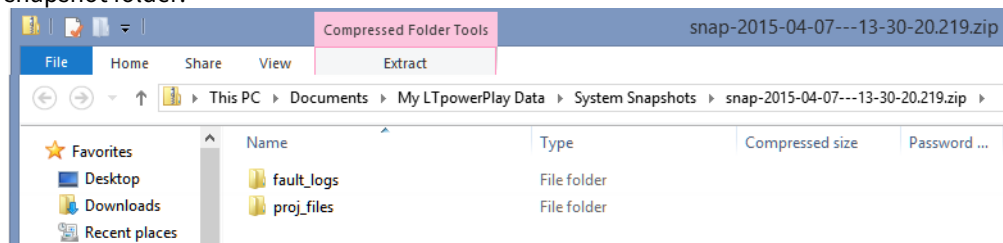
- This opens the System Snapshot tool and automatically saves a 'snapshot' (a .zip archive with one .rtf file per fault log)



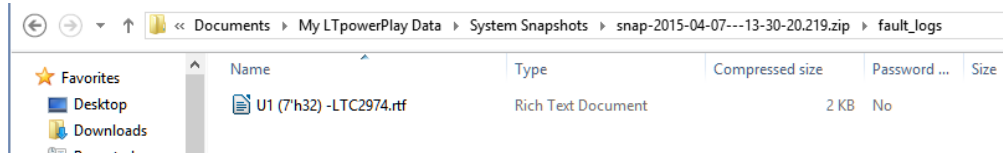
- The list of snapshots are shown on the right. If you want to explore the snapshot in windows explorer, select the snapshot and click "Explore Snapshot"



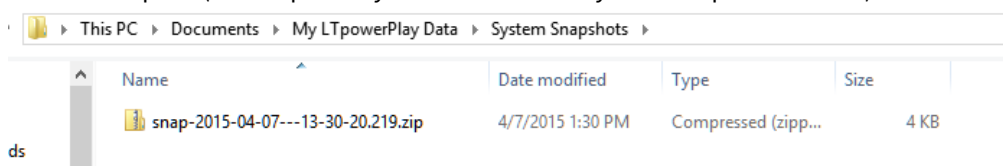
- This opens a windows explorer window that points to the selected .zip file within the snapshot folder:



- The fault\_logs subfolder of the zip file contains one .rtf (rich text file) per fault log on the board. In this specific case shown below, there was only one fault log for the device U1, a LTC2974 at 7-bit I2C address 7'h32:



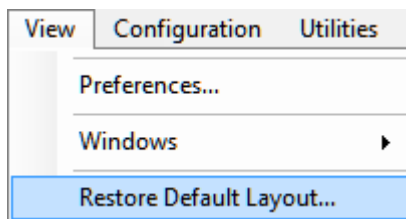
- If you want to see all the snapshot .zip files, click the 'Up' icon in windows explorer to navigate out of the .zip file (click 'Up' until you are at the 'System Snapshots' folder):



- At the System Snapshots folder level, you can see all the the .zip files, and copy/move them to other locations if you like for archival purposes

## 5.19 My Views are Messed up. How can I recover to the default view configuration?

Usually selecting 'View > Restore Default Layout...' on the LTpowerPlay menu will restore the views to their default layout:



However, in some rare circumstances, a recent change to the views may be causing a crash in LTpowerPlay. When you restart LTpowerPlay, it will attempt to restore the old view layout, and if the crash persists, you may be in an infinite loop of restart app/crash. These settings are persisted in a file in your file system under your user's 'AppData' folder. Follow the procedure below to delete these files and restore the views to their default state:

- Close any running copies of LTpowerPlay
- Open a Windows Explorer Window
- Navigate to `c:\Users\<your user name>\`
- Select the location bar at the top of the window that now contains "`C:\Users\<your username>`", and manually type in the 'AppData' subfolder (which is hidden by default)
  - For example, "`C:\Users\mholloway\AppData`"
- Navigate to the subfolder `Roaming\LTC\LTpowerPlay` underneath the AppData Folder. It will look like this:
- Delete the two .config files.
  - If you also wish to default all preferences, you can delete all files in this folder.

|      | Name                     | Date modified     | Type         | Size  |
|------|--------------------------|-------------------|--------------|-------|
|      | DockPanel.Offline.config | 4/27/2015 3:04 PM | CONFIG File  | 8 KB  |
| Is   | DockPanel.Online.config  | 4/23/2015 4:39 PM | CONFIG File  | 11 KB |
| ices | prefs.xml                | 4/28/2015 9:51 AM | XML Document | 3 KB  |

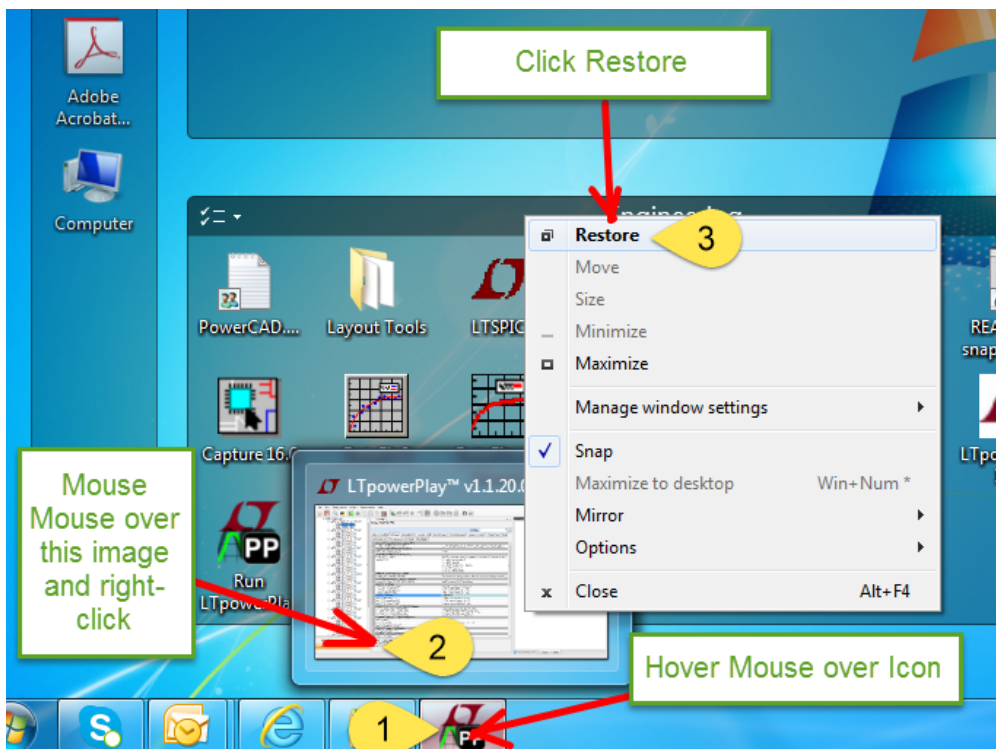
- Re-launch LTpowerPlay

LTpowerPlay should start up with the default view layouts

## 5.20 I see LTpowerPlay in the taskbar, but I cannot see the window. How do I view the window?

There are certain situations where the LTpowerPlay window may appear in the task bar, but you cannot see the window on your desktop. This is because LTpowerPlay attempts to restore the last position when it is launched. For instance, if the last time you launched LTpowerPlay you had a multiple monitor setup, and you have subsequently switched to a single monitor configuration, the saved position of LTpowerPlay may put the LTpowerPlay window out of the visible area of your desktop.

A simple workaround should restore LTpowerPlay to a visible area on your desktop. The following annotated images shows these steps:



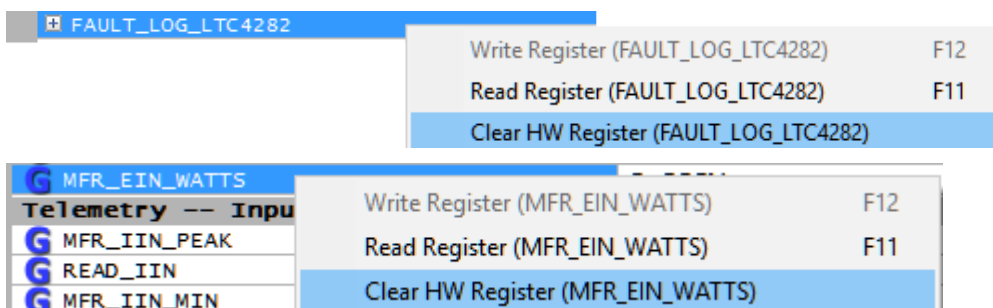
- Hover the mouse over the LTpowerPlay icon in the task bar
- Move the mouse over the LTpowerPlay 'thumbnail' image that pops up, and right-click the mouse on the image
- Select 'Restore' in the context menu

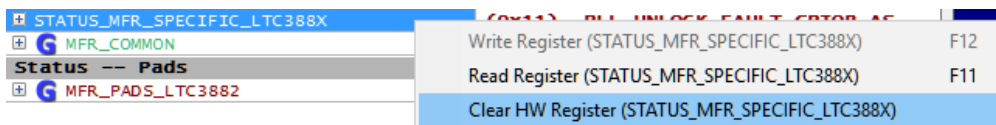
## 5.21 How do I Clear Individual Items In LTpowerplay?

It's easy to clear individual items like Fault Registers, Peaks, Energy meter accumulation, etc in LTpowerPlay. Consult the IC datasheet for which registers are individually clear-able. To clear a register that is clearable, please follow these steps:

- Select the register of interest in the Telemetry/Status Panel
- Click the Right button on your mouse
- Select 'Clear HW Register X' in the Menu, where X is the name of the Register you have selected
  - NOTE: If the register is not clearable, this item will not be available.

Example Screenshots:

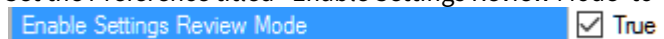




## 5.22 Is There an Easy Way to Review Large Configurations

There is a feature called 'settings review mode' which allows you to quickly visualize setting differences across larger systems using the 'All Pages in System' view. Here's how to use it:

- In LTpowerPlay, select 'View > Preferences' in the menu
- Set the Preference titled 'Enable Settings Review Mode' to 'True':



- Select a desired configuration register
- Look to the 'All Pages in System' View. It should look something like this:

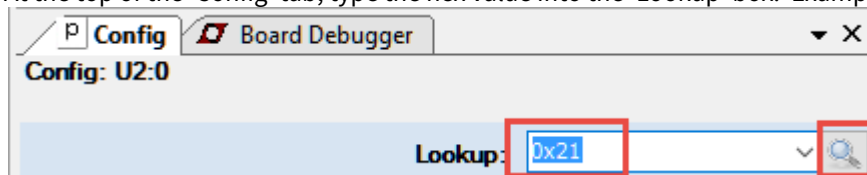
| TON_RISE (All Pages in System) |           |
|--------------------------------|-----------|
| U1:0 - LTC2975                 | 10.000 ms |
| U1:1                           | 10.000 ms |
| U1:2                           | 10.000 ms |
| U1:3                           | 10.000 ms |
| U2:0 - LTC3882                 | 8.000 ms  |
| U2:1                           | 8.000 ms  |

The contents of the display will depend on the selected register. Items configured with the same settings are colored identically. This allows you to quickly visualize settings differences. In the example above, there are two unique values for TON\_RISE (8ms and 10ms)

## 5.23 I have a Register Command Code. Can LTpowerPlay tell me the Corresponding Register Name?

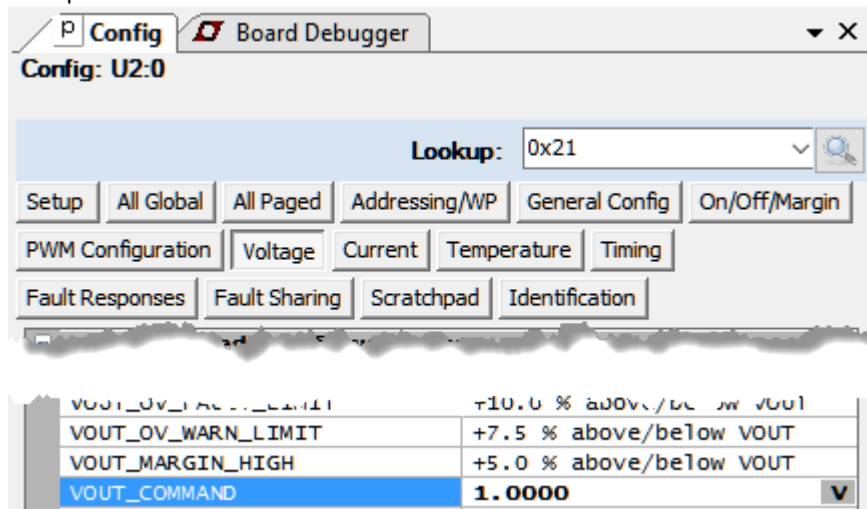
Yes, if the register is visible in one of the LTpowerPlay register views. Follow these steps:

- Select a device in the tree that is the same device type as yours
- At the top of the 'Config' tab, type the hex value into the 'Lookup' box. Example below:



- (i.e. type '0x21 <enter>'),
- LTpowerPlay will attempt to find the register that corresponds to this command code. If found, it will highlight the register in the Configuration or Telemetry/Status View, and if a Configuration register, will also select the register's primary configuration group ('Voltage' in this case). Here is the Result for the above

Example:

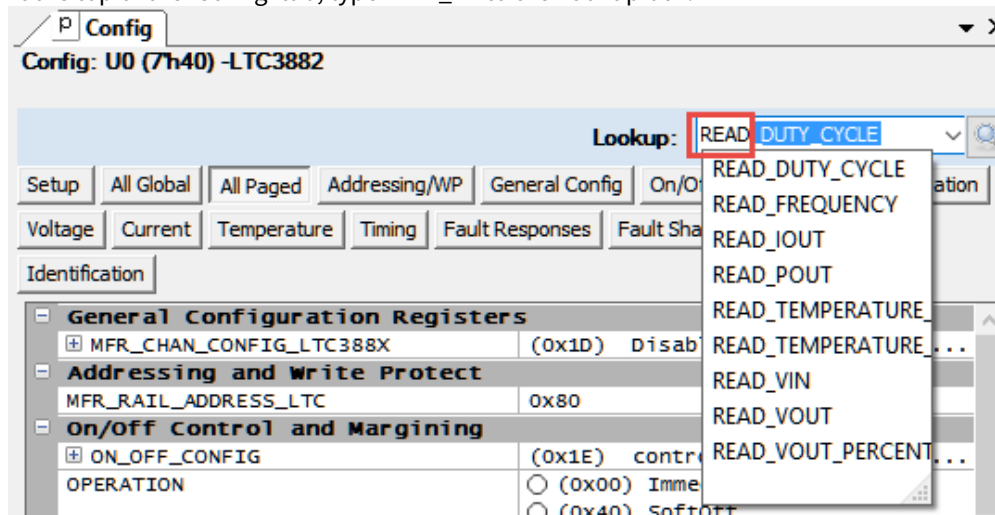


## 5.24 How Can I Quickly Search for a Register by Name?

If you know the beginning characters of the Register Name, you can type it into the 'Lookup' box at the top of the Config tab. LTpowerPlay will incrementally search for Registers that start with the characters you type, and will present a list of potential matches. You can select one of these to navigate to it.

**Example:**

- Select your device in the system tree
- At the top of the 'Config' tab, type 'MFR\_' into the Lookup box.



- LTpowerPlay will attempt to find the registers that begin with the letters 'READ', and will present a list of corresponding matches. Select the Individual Register you want.
- When an item is selected, LTpowerPlay will highlight the register in the Configuration or Telemetry/Status View, and if a Configuration register, will also select the register's primary configuration group/tab.

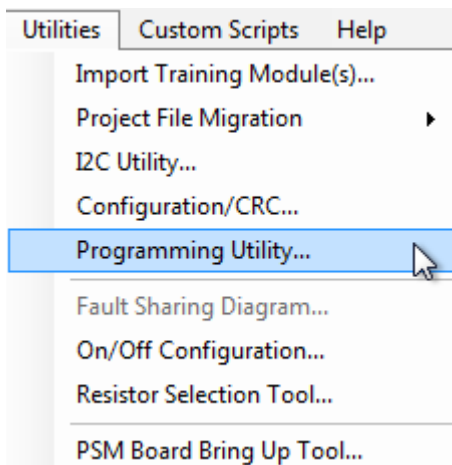
## 5.25 How can a project file be used to program devices in a system

How can a project file be used to program devices in a system

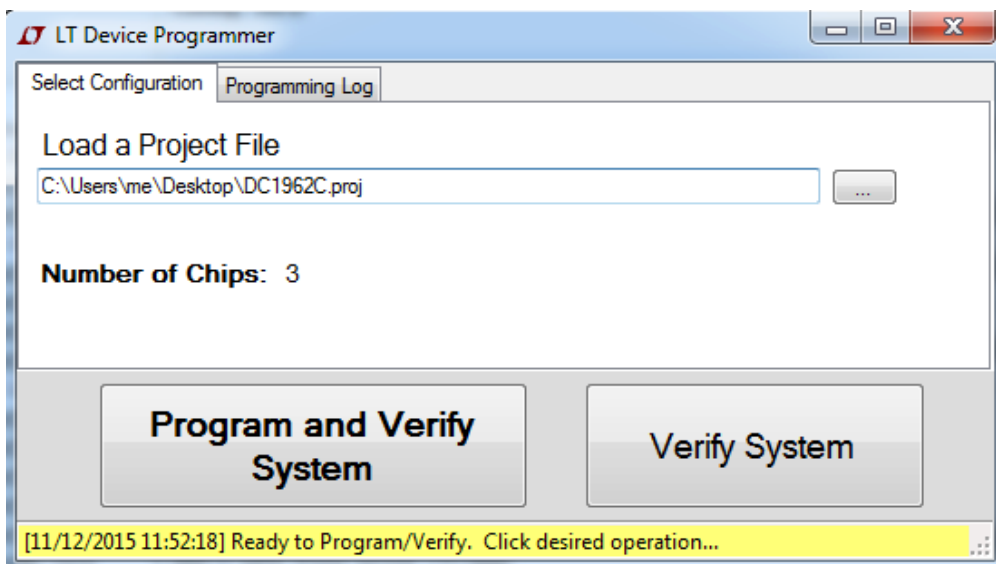
Programming devices in a system is called In System Programming. LTpowerPlay supports In System Programming through a Programming Utility.

### 5.25.1 How to Program

Invoke the Programming Utility menu:






After the Programming Utility Menu is invoked, there is an entry for a LTpowerPlay project file:



After entering a path or using the browser (... button), press "Program and Verify System" to program all devices in the project. Pressing "Verify System" will use the project file to verify the programming is correct.

The programming process will update the USER\_DATA register with a Checksum of the data.



| Fault Sharing   |              |        |
|---|--------------|--------|
| Scratchpad  |              |        |
| Identification  |              |        |
| <b>Scratchpad</b>   |              |        |
|  | USER_DATA_00 | 0x0000 |
|   | USER_DATA_01 | 0x78E3 |
|  | USER_DATA_02 | 0x0000 |
|   | USER_DATA_03 | 0x0000 |
|  | USER_DATA_04 | 0x0000 |

## 5.25.2 Other Methods of Programming

Devices can also be programmed before soldering to a board using OEM files. See [AN145](#)<sup>23</sup> for more information on programming methods.

## 5.25.3 Programming Devices with CRC failures in the NVM/EEPROM

If a device has CRC errors, the Programming Utility still able to program it. The utility will use the 0x5B global address to set the base address, cause the device/s to read their ASEL pins, and then program them.

## 5.25.4 Prerequisites

For any method of In System Programming to work, the following must be true:

1. All devices on the bus must share a common base address
2. All devices on the bus must have unique ASEL configurations
3. The LTpowerPlay project must contain the base address and the proper effective addresses (Base combined with ASEL)

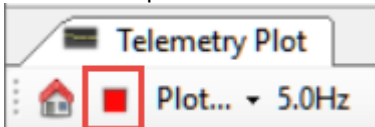
## 5.26 How Can I Stop LTpowerPlay's Polling Loop, or make the Dongle's SCL/SDA Lines Hi-Z

There are two methods you can use to instruct LTpowerPlay to stop using the I2C bus to poll telemetry/status of the devices in the system tree:

1. Click the 'Go Offline' Button in the toolbar at the top of the LTpowerPlay main window



2. Click the 'Stop' Icon in the telemetry plot



Both methods will insure that the dongle/host controller will leave the I2C SCL/SDA lines Hi-Z.

NOTE: With method 2, you can still 'manually' take action on the bus by clicking one of the toolbar icons (i.e. Write/Read All, etc), or by individually writing/reading registers (F12/F11), but LTpowerPlay will remain Hi-Z on the bus

<sup>23</sup> <http://cds.linear.com/docs/en/application-note/AN145f.pdf>

until explicit user action is taken that requires I2C bus access. Method 1 simply takes LTpowerPlay 'offline' so these I2C functions are no longer available until you go online again.

## 5.27 Can LTpowerPlay be run Without Administrative Privileges, or Under a Non-admin Account?

In some restricted environments (secure networks, etc), users may not be able to run programs with administrative privileges. Until very recently, admin privileges were required to install and run LTpowerPlay. This was done for simplicity, as running without admin privileges would render some functions void (like automatic updates). In the future, an update will be available that will allow LTpowerPlay to be run under a 'normal user' account (without admin privilege).

If you are running a version greater than 1.2.9.0 of LTpowerPlay, your version is already able to run without admin privileges. Otherwise, you can load the beta version of the software that does not require admin here:

BETA Version that does not require admin privilege:

<http://ltpowerplay.com/updates/vin0/1.2.10.0/SetupLTpowerPlay.ms<sup>24</sup>i>

## 5.28 Can I call LTpowerPlay from a batch file to program devices in-system?

Yes. As discussed earlier, this may be appropriate in situations where you want to use LTpowerPlay and the DC1613A dongle to program devices in system, but you do not want to use the GUI interactively. Other possibilities were discussed in other items, but this specific example illustrates how to call the LTpowerPlay executable from a DOS batch program, and check the return code for success or failure.

- Open NOTEPAD and create a batch file called program.bat file with the following contents:

### program.bat contents

```

1  @ECHO OFF
2  REM Attempt In System Device Programming
3  "c:\Program Files (x86)\Linear Technology\LTpowerPlay\LTpowerPlay.Shell.exe" -Program -Verify -
   ProjectFile="c:\my.proj" >detail.log
4  REM Check Return Status for Errors
5  IF %ERRORLEVEL% NEQ 0 (
6      ECHO "ERROR Programming. Detailed Log Follows:"
7      TYPE detail.log
8      GOTO END
9  )
10 ECHO "SUCESSFULLY PROGRAMMED!"
11 :END
12 PAUSE

```

- Substitute your .proj file for "c:\my.proj"
- Open a DOS command prompt (start, run, 'cmd.exe')
- Change directories to the location of the program.bat file you created
- type 'program.bat'

<sup>24</sup> <http://ltpowerplay.com/updates/vin0/1.2.10.0/SetupLTpowerPlay.msi>

- In the case of success, the following output will be printed:

#### Example Output for Success

```
C:\>program.bat
"SUCCESSFULLY PROGRAMMED!"
Press any key to continue . . .
```

- In the case of failure, the output stored in detail.log is echoed to the output as shown below:

#### Example Output for Failure

```
C:\Users\mholloway\Desktop>program.bat
"ERROR Programming.
Detailed Log Follows:"

LTpowerPlay.Shell.exe
version 2.0.0.0
Copyright (c) Linear Technology
Processing project file c:\my.proj
Exporting temporary ict in system programming hex file 'c:\my.ict'
Processing in system programming (.ict) hex file 'c:\my.ict'...

[2/23/2016 13:15:19.945] Checking for USB Host Controller...
[2/23/2016 13:15:20.035] Processing Programming and Verification Algorithm
[2/23/2016 13:15:20.042] Length=6,Type=24,EventValue=0 Event='BEFORE_BEGIN'
[2/23/2016 13:15:20.042] Length=6,Type=24,EventValue=0
[2/23/2016 13:15:20.044] Length=8,Type=27,MetaDataLength=2,0x0000 Meta Data
Type='SERIAL_NUMBER HOLDER'
[2/23/2016 13:15:20.044] Skipping Serialization
[2/23/2016 13:15:20.044] Length=6,Type=24,EventValue=16 Event='BEFORE_INSYSTEM_PROGRAMMING_BEGIN'
[2/23/2016 13:15:20.044] Length=6,Type=24,EventValue=16
[2/23/2016 13:15:20.044] Length=9,Type=16,PMBusWriteWord(0x05B,0x10,0xC000)
[2/23/2016 13:15:20.055] !!!!!!!ERROR: Device at address 0x5B NACKED WriteWord transaction CMD=0x10,
DATA=0xC000

Error Programming and Verifying Device! See Programming Log for details.
Press any key to continue . . .
```

This is one example of how to call LTpowerPlay from the command-line and inspect the return value for errors. Many other possibilities exist. This is meant as an illustrative example.

## 5.29 When I click 'Add default LTCXXXX Chip', why do the settings not match the datasheet/factory defaults?

In general, the factory/datasheet defaults for a chip, though aimed at ensuring safety when placed into target circuit, may not provide a simple/easy starting point for building configurations for board bring up. The GUI defaults for chips are aimed at making it easy for you to build a configuration for a final configuration.

**An example will illustrate the difference:**

The factory Default value for ON\_OFF\_CONFIG for LTC2978 is 0x12. This setting causes the chip to ignore both the OPERATION and the CONTROL pin, so that the output will never power up. While this may be helpful to you if you load a factory default part onto your board, to ensure safety, it creates unnecessary work for you to get your board rails up if you are building a configuration in LTpowerPlay. The GUI default for the same register is 0x1E, which tells the chip to require both CONTROL pin and OPERATION command to enable the output. The OPERATION command is defaulted to Off, so that the chip will still remain safely off until you decide to turn it on, but when you program the OPERATION command to On, the chip will turn on as expected. With the datasheet default setting of 0x12, turning the rail on would require extra steps, and create confusion.

Because of this, the GUI defaults may not match the datasheet defaults for a given part. The GUI defaults are chosen as a balance of safety and simplicity to help you build a configuration for your final application quickly.

*NOTE: You can find the datasheet defaults in the command table of the datasheet for the part. Press F1 in LTpowerPlay to bring up this document. You may also find the datasheet defaults .proj file under the project\_files folder of the installation folder of LTpowerPlay.*

### 5.30 How do I get more information on the status/fault indicator XXXX?

The last section of this FAQ document contains detailed information for various fault types. Starting from the table of contents of the FAQ document, navigate to the last section of the table of contents (titled "What's this Fault?"). Find the table of contents entry for the fault type of interest, and click its hyperlink to navigate to the detailed entry.

The "What's this Fault" entries contain a number of useful information including:

- A description of the fault
- Where relevant, an example reference waveform
- Discussion of how the part responds to the condition
- Common causes of the condition
- Remedies and Workarounds
- Troubleshooting tips, including scope debug tips where relevant

The detailed information is available 'in situ' while using LTpowerPlay, in a new tool called the "What's this Fault tool". This tool is similar to the "Why am I Off tool" in two ways:

- It sits at the periphery of the main LTpowerPlay window
- It continuously analyzes status for the selected channel and provides context specific help for that channel

Use the following procedure to get detailed help for status/fault information on the selected channel within LTpowerPlay:

- Select the channel of interest in the system tree on the left side of LTpowerPlay:



- Move your mouse to the right side of the main LTpowerPlay window and hover over the tab titled "WTF" (What's this Fault):



- The "What's this Fault tool" will fly out and you can inspect the detailed help as shown below (example shown is a TON\_MAX\_FAULT):

Group  
op

WTF?

### What's This Fault?

Status Items: (Select an Item for more Information)

TON\_MAX\_FAULT  
FAULT\_LOG\_PRESENT

A TON\_MAX fault was detected because the VSENSE voltage did not reach the VOUT\_UV\_FAULT\_LIMIT before the TON\_MAX\_FAULT\_LIMIT.

TON\_MAX fault detected because Vout did not reach UV limit before TON\_MAX\_FAULT time

VOUT\_EN

VOUT

TON\_MAX\_FAULT

### Possible Causes:

- Output is shorted, preventing Vout ramp.
- TON\_MAX\_FAULT\_LIMIT set shorter than Vout's ramp up time
  - Large soft start cap
- VOUT\_UV\_FAULT\_LIMIT set too high.
- Regulator feedback resistor value(s) incorrect or assembly issue.
- VOUT\_EN pin has no pullup resistor or power supply enable/RUN pin not pulled high.
- VOEN not physically connected to the power supply enable/RUN pin (failed or missing connection)
- Power supply's input is not enabled, or too weak for regulation
  - If its input is sequenced, enable it **before** this channel

### Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Increase TON\_MAX\_LIMIT to see if the problem goes away
- Set TON\_MAX\_FAULT\_RESPONSE to 'ignore' to see if the output voltage comes up
- Lower the VOUT\_UV\_FAULT\_LIMIT and see if the problem goes away

- If the channels contains more than one fault/status, the first one is selected in the list. You can select other statuses to show detailed information for the item of interest


## 5.31 LTpowerPlay requires .NET Framework v2.0, and my computer does not have this version. How do I get it?

Note that the .NET Framework 3.5 also contains version 2.0, so installing version 3.5 will also install version 2.0.

Until recently, the .NET Framework 2.0/3.5 was automatically installed with the operating system. Newer versions of Windows (>= Windows 8) have the option to install .NET Framework 3.5 from the control panel, but it is not installed by default. You can manually install .NET 3.5 as follows:

### 5.31.1 Method 1: Enable the .NET Framework 3.5 in Control Panel

You can enable the .NET Framework 3.5 through the Windows Control Panel. This option requires an Internet connection.

1. Press the Windows key  on your keyboard, type "Windows Features", and press Enter. The Turn Windows features on or off dialog box appears.
2. Select the .NET Framework 3.5 (includes .NET 2.0 and 3.0) check box, select OK, and reboot your computer if prompted.



### 5.31.2 Method 2: Manually download and install the .NET Framework 3.5

If the above method is not available or working, you can download and install the .NET Framework 3.5 using the following link:

<https://www.microsoft.com/en-us/download/details.aspx?id=251504><sup>25</sup>

## 5.32 Can LTpowerPlay dump the contents of a device's EEPROM to a file?

Yes. LTpowerPlay has a command-line option that takes both a target device address and the location of a text file to write. If you wish to do this, follow these steps:

- Close any running instances of LTpowerPlay
- Open a Command Prompt (preferably as administrator)
  - Click <start>
  - Type: cmd
    - Right click the command prompt shortcut and select 'Run as Administrator'
- change directories to the LTpowerPlay installation folder (C:\Program Files (x86)\Linear Technology\LTpowerPlay by default):
  - Type: cd "C:\Program Files (x86)\Linear Technology\LTpowerPlay"
- Run the program with the -t (your target adress) and -d (your dump file) options:
  - (Example) C:\Program Files (x86)\Linear Technology\LTpowerPlay>LTpowerPlay.Shell.exe -t=0x5c -d=c:\mydump1.txt
    - Replace "0x5C" above with the target address of the device from which you wish to read the EEPROM
    - Replace "c:\mydump1.txt" above with the location of the output file you wish to write

---

<sup>25</sup> <https://www.microsoft.com/en-us/download/confirmation.aspx?id=16614>

Running LTpowerPlay with this command-line option will read the EEPROM of the device at the specified address (0x5C in this example) directly and store the data into the specified text file (c:\mydump1.txt in this example).

*NOTE: Please insure that the account running the command prompt has sufficient write access to create the file. If not, substitute an appropriate file location, or run the command prompt as administrator as described above.*

An example file fragment is shown below for your information:

#### Example output file

```
Dump of data at chip address 0x5C
NVM Data...
Packing Code=0x7701
NVM DATA Length= 512 bytes (0x0200)
Word #0 (0x0000) = 0x7D96
Word #1 (0x0001) = 0x3F20
Word #2 (0x0002) = 0x0063
Word #3 (0x0003) = 0x00FF
```

## 5.33 What is the syntax of scripts (command-files) in the I2C Utility?

The format of command-files is .csv. There are 3 types of lines: Commands, comments, and special instructions. Command lines tell the script interpreter to issue some I2C/SMBus transaction (write or read) and for reads, optionally check the return value. A command instruction must be formatted as follows:

<addr>,<protocol>,<cmd>,<data>,<mask>

The first 2 fields are always required. A line should always have 4 commas. When fields are not used, keep the commas as shown in the example at the end.

Comments start with #. These lines are ignored functionally, except that the line is printed in the output log for convenience. Commas are also allowed after the last field if you place a '#' character there.

Special instructions start with #! and presently include the #!Sleep command. Sleep takes the value of the delay in ms.

A description of the command line fields follows:

- **<addr> (always required):** the 7-bit address of the device (same as what is shown in the LTpowerPlay system tree)
  - NOTE: All PSM devices respond to the special address 0x5B
- **<protocol> (always required):** Must be one of the following:
  - **WB:** SMBus write byte protocol
  - **RB:** SMBus read byte protocol. Print read value.
    - Optionally test read value against expected value and optional mask
  - **WW:** SMBus write word protocol (low byte sent first)
  - **RW:** SMBus read word protocol (low byte read first). Print read value.
    - Optionally test read value against expected value (if present in <data>) and optional mask (if present in <mask>).
  - **SB:** SMBus send byte protocol (command with no data, like STORE\_USER\_ALL for instance)
  - **MB:** Read/Modify/Write a byte command using read/write byte protocols
    - Read value at specified <cmd> using SMBus read-byte protocol
    - Modify bits that are ones in <mask> to equal the value provided in <data>
    - Write modified value to specified <cmd> using SMBus write-byte protocol
  - **MW:** Read/Modify/Write a word command using read/write word protocols and a mask



- Read value at specified <cmd> using SMBus read-word protocol
- Modify bits that are ones in <mask> to equal the value provided in <data>
- Write modified value to specified <cmd> using SMBus write-word protocol
- **RCVB:** SMBus receive byte protocol (address with no command, like SMBus ARA for instance)
  - Enter 0x00 for <cmd> even though the field is not used. The software requires <cmd> to be non-null
  - NOTE: <data> and <mask> fields can also be applied to the received byte (see example ARA check in below script)
- **<cmd> (always required):** the 8-bit sub-address or command code to write/read
  - (i.e. 0x01 for PMBus OPERATION command)
- **<data> (required for writes/modify, optional for reads)** hex value of the data
  - For write protocols (WW, WB) this is just the data value to write
  - For read protocols (RB, RW) , if this field is present the actual value read from hardware will be tested against this expected value, and fail the script if not equal
    - NOTE: If <mask> is provided, only the bits in the mask are tested against the <data> value provide
- **<mask> (required for modify, optional for reads)** A mask indicating the bits of concern for modification or testing of read values
  - For read protocols (RB, RW), if this field is present the actual value read from hardware will be masked with the <mask> value and then tested against a pre-masked <data> value. The script will fail if the <data> and masked read values are not equal.
  - For modify protocols (MB,MW), this field must be present. The mask indicates which bits should be modified after reading before writing back to hardware.

#### Example Script:

NOTE: The example uses device address 0x30

**Example Script**

```

1  #Write the PAGE register (command 0x00) to 0xFF (all pages)
2  0x30,WB,0x00,0xFF,
3  #Write the OPERATION register (command 0x01) to ON (0x40) - sequence all channels off
4  0x30,WB,0x01,0x40,
5
6  #Wait 1 second
7  #!Sleep 1000
8
9  #Write the OPERATION register (command 0x01) to ON (0x80) - turn all channels on
10 0x30,WB,0x01,0x80,
11
12 #Wait 1 second
13 #!Sleep 1000
14
15 #Check device channel 0 - set PAGE to 0
16 0x30,WB,0x00,0x00,
17 #Check that the device is ON. That is, that bit 6 of STATUS_BYTE (command 0x78) is a zero. Fail
   the script otherwise.
18 0x30,RB,0x78,0x00,0x80
19 #Check that STATUS_VOUT (command code 0x7A) is 0x0000. That is, there are no voltage related
   faults or warnings
20 0x30,RB,0x7A,0x00,0x80
21
22 #Check device channel 1- set PAGE to 1
23 0x30,WB,0x00,0x01,
24 #Check that the device is ON. That is, that bit 6 of STATUS_BYTE (command 0x78) is a zero. Fail
   the script otherwise.
25 0x30,RB,0x78,0x00,0x80
26 #Check that STATUS_VOUT (command code 0x7A) is 0x0000. That is, there are no voltage related
   faults or warnings
27 0x30,RB,0x7A,0x00,0x80
28
29 #Issue CLEAR_FAULTS (command code 0x03) send byte protocol
30 0x30,SB,0x03,,
31
32 #Wait 1 second
33 #!Sleep 1000
34
35 #Read ARA and assert that the value is 0xFF (no devices holding ALERTB low on the bus)
36 0x0C,RCVB,0x00,0xFF,0xFF

```

## 5.34 Why must I Manually Click 'Detect Chips' when Connected to a Customer Board?

LTpowerPlay knows when it's connected to a factory Demo Board (DCXXYY). When connected to such a board, LTpowerPlay will automatically connect to the main I2C bus and enumerate the system for you. However, when connected to another type of board (namely a customer board), LTpowerPlay will instead wait for the user to provide permission to become a host on the main I2C bus. This behavior is by design, and it prevents LTpowerPlay from potentially disrupting your onboard host controller without your knowledge or permission. Simply click 'Detect Chips' to tell LTpowerPlay it's OK to become a master on the I2C bus and enumerate your board.

## 6 Device Programming FAQ

This Section contains Frequently Asked Questions about Device Programming

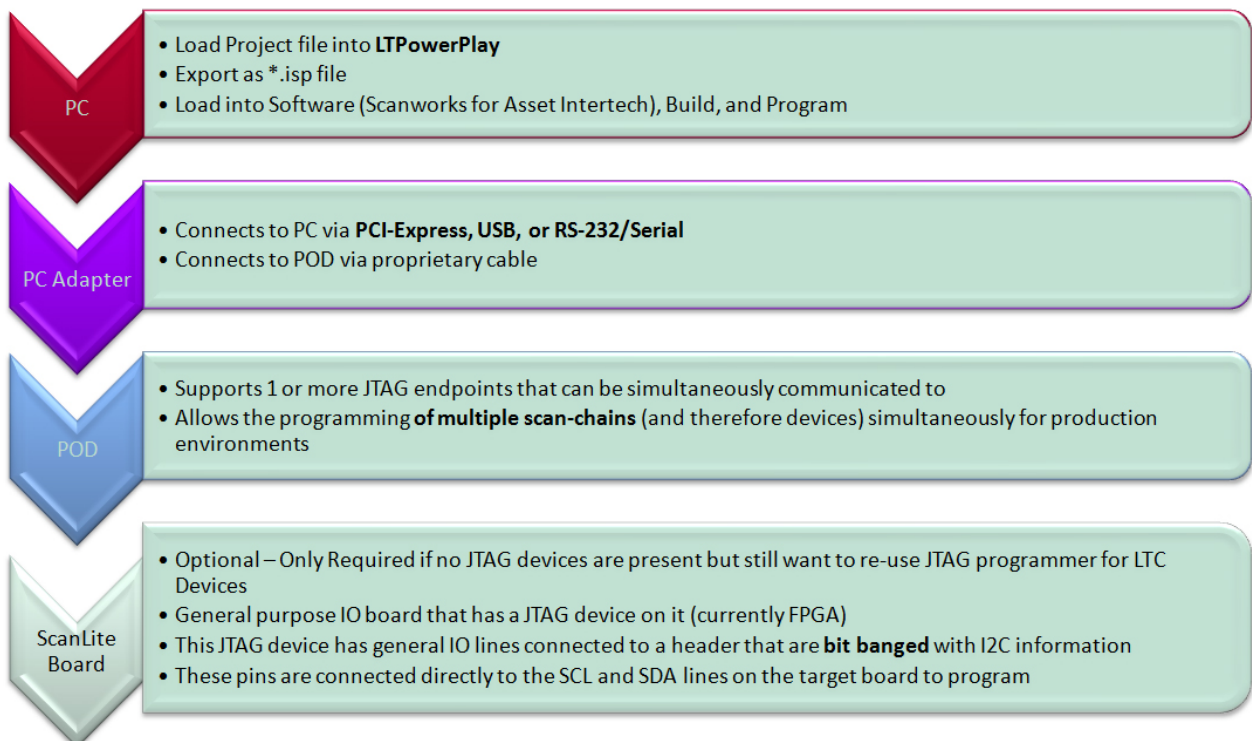
### 6.1 Can I program my PMBus Device over JTAG?

To be able to program your PMBus device via JTAG, what needs to happen is that you either use a JTAG device on the bus with boundary scan accessible pins connected to the PMBus, or you use an external device (Asset Intertech provides one, shown as #4 in the second picture) which connects into your normal JTAG chain and provides a device with some IO that can be connected to the PMBus.

In the second image, we are programming a powerstick via JTAG. Since there are no devices on the powerstick that accept JTAG, we need that ScanLite board that does the JTAG to bit-banged PMBus conversion. (#4)

On a customer board that already has JTAG devices, it is presumable that one of those JTAG devices is already connected to the PMBus. For instance if they are programming an FPGA or something which also is on the bus to communicate with the PSM devices, then all you need is to talk to the same FPGA. There would be part of the JTAG process that would actually program the FPGA, the other part would be manipulating the boundary scan nodes on the PMBus lines of the FPGA to wiggle high/low and bit-bang full PMBus.

The only real caveat is bus speed. As per PMBus, we need to keep the bus speed above 10KHz, and we like to keep it well above that. Since we are bit banging, every PMBus clock takes 2 full JTAG chain trips. This basically means that for large JTAG chains with thousands of nodes, you will need to keep the JTAG clock high enough to allow for a fast enough PMBus clock.



1. PC running Scanworks Software
2. PCI-X, USB, or RS232 Adapter card
3. Scanworks POD. 4 Port version shown
4. ScanLite Board
5. Target board to be programmed




## 6.2 Does BPM (or Arrow) Support my LTC device for programming?

As a general rule of thumb, we strive to enable programming support for every LTC device that could need it before it is released for sale. But to check the public status of any part, visit this external link to BPM's "Find-Your-Device" page and query for supported devices

<http://www.bpmmicro.com/find-your-device/>

For example, as of December 2014 the current list is as follows:

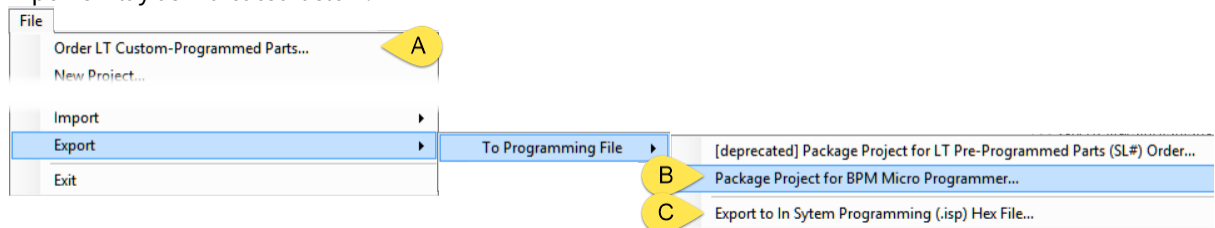
 Unknown macro: 'table'

## 6.3 How do I export the correct programming files from LTpowerPlay?

Which type of file you export depends on how you plan to program devices.

- Load your project file into LTpowerPlay
- Select the
  - A: Order **individually** pre-Programmed parts from LT
  - B: Order **individually** pre-programmed parts from, Arrow or a Contract Manufacturer (CM)
  - C: Program all the parts on your own board 'in system' via I2C\*\*

- Depending on the choice above, select the appropriate menu pick (marked A, B, or C) in the File menu of LTpowerPlay as indicated below:



Depending on your choice above, different file formats will be generated:

- Option A: LTpowerPlay generates a .zip file for your entire board (including programming files for each individual device in the .proj file) with multiple programming file formats within the .zip file. You need to upload this entire file to the Linear Express website as instructed by the programming wizard that launches when you select this option. The Linear Express server will parse the .zip file and allow you to choose one or more devices and easily order the programmed parts.
- Option B: LTpowerPlay generates a .zip file that contains one .OEM file for each device in the .proj file, as well as the original source .proj file. This is useful for archival purposes.
  - However, when you will send the individual .oem files to the programming house. One .oem file is stored per device in the project file. The files are stored in unique folders within the zip that are named and distinguished by device type, address, and U number for your convenience. Within the appropriate subfolder is the corresponding .oem file for that device. The .oem file is named to include the unique configuration CRC. Read the file ProgrammingInstructions.txt in the .zip file for more information on where the individual .oem files are stored within the .zip file.
  - Some customers prefer to have one file per device. This can be easily achieved by extracting the .zip file and then accessing the individual .oem files within the archive. The folder names will help you keep things straight. In the event you do this, it is still strongly recommended that you archive the generated .zip file which also contains the original .proj file that was used to generate all the programming files.
- Option C: LTpowerPlay generates a single .isp (in system programming file) that is capable of programming all the devices on your board 'in system' via I2C.\*\*

\*\*NOTE: Option C requires you to implement custom software to process the .isp (in system programming) file, validate and test your software, etc, which may require significant engineering investment on your part. Contact LT factory for more information.

## 6.4 How do I hook-up a dongle to my customer board?

The DC1613 "dongle" provides connectivity between a personal computer and an I2C bus connected to various Linear Technology solutions. The dongle connects to the PC through an isolated USB bus that provides a serial port and some limited power. The PC talks to the I2C bus through this serial port, and can use LTpowerPlay to control and visualize the operations of the LTC parts. The dongle connects to power and data through a 12-pin ribbon cable, with connections for the I2C bus, and for power and ground. Many LTC PSM demo boards have a 12-pin connector that accepts this 12-pin ribbon cable. Custom boards should include connections for the dongle, and can either include the 12-pin connector, or a minimal 4-pin connection.

The recommended part number for the 12-pin connector is: FCI, 98414-G06-12ULF

The 12-pin ribbon cable connector part number is: FCI, 90311-012LF

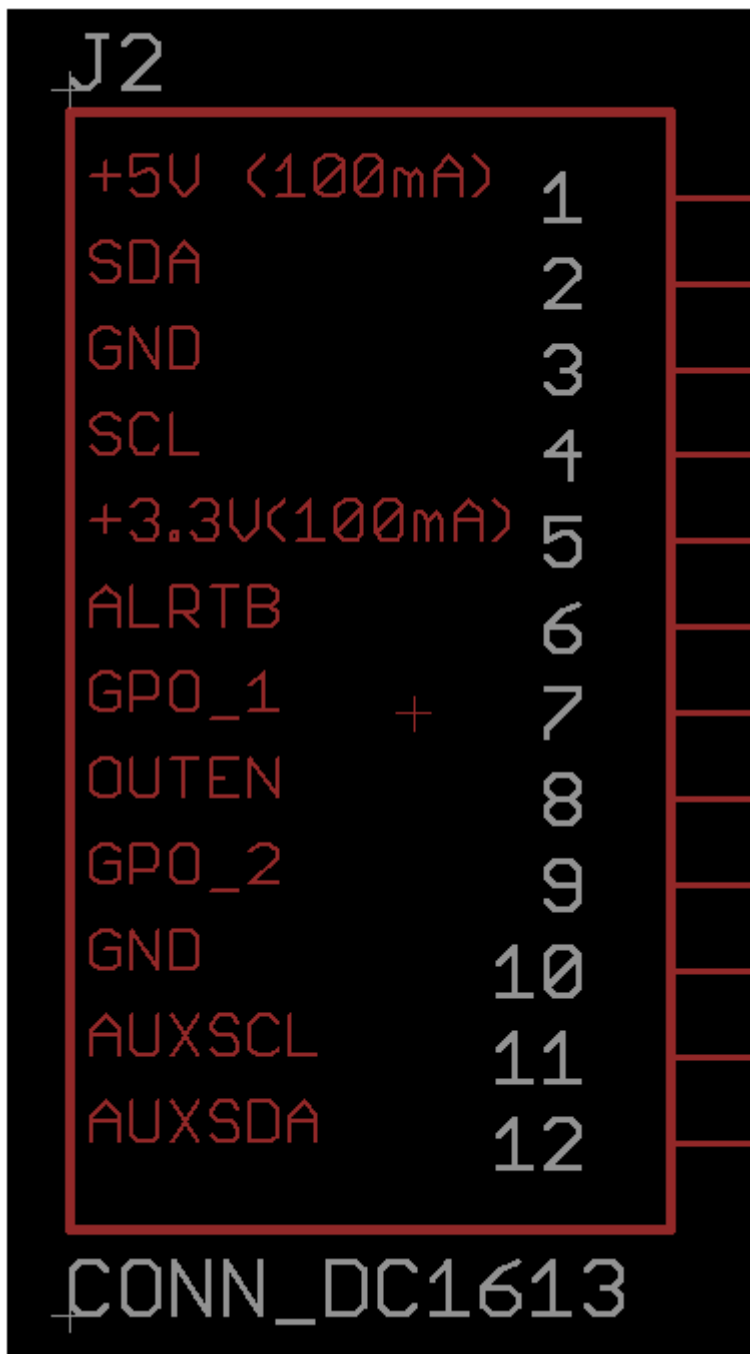
Alternatively, the 4-pin connector part number is: TE CONNECTIVITY 5-104071-7

The connectors are keyed to prevent inserting the connector incorrectly.

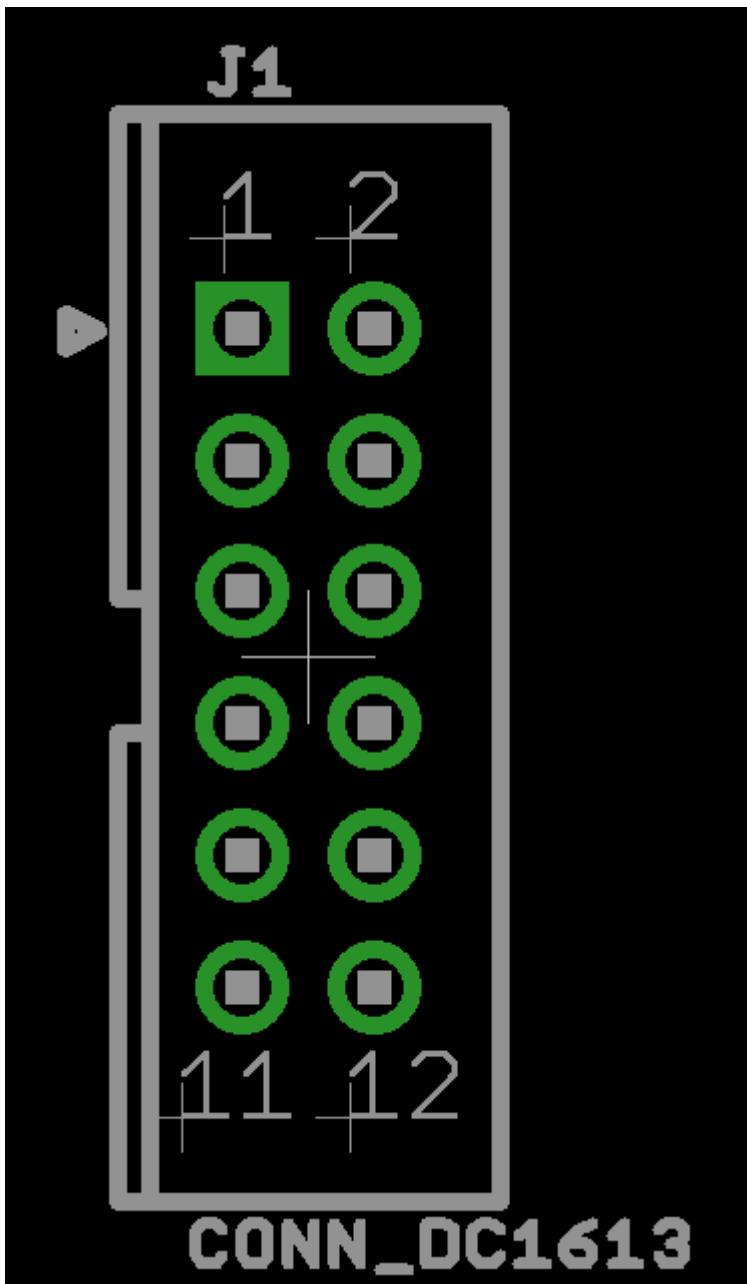
There are four required pins (the others are optional):

| NAME  | NUMBER | FUNCTION                      |
|-------|--------|-------------------------------|
| SDA   | 2      | I2C Data                      |
| SCL   | 4      | I2C Clock                     |
| GND   | 3      | Ground                        |
| +3.3V | 5      | +3.3V Logic power (100mA max) |

The 12-pin connector schematic symbol looks like this:



The 12-pin connector layout footprint (top view) looks like this:



## 6.5 How to Decode a Standard Intel Hex File

The listing below shows an example Intel Hex file. The colour coding below the listing defines the various fields of the Intel hex file:



```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

|  |             |
|--|-------------|
|  | Start code  |
|  | Byte count  |
|  | Address     |
|  | Record type |
|  | Data        |
|  | Checksum    |

As you can see, each line of the hex file starts with a “:” character, followed by a byte count, address, record type, some number of data bytes, and ends with a checksum for that line. Each line of the file in this case is an ‘Intel Hex File Data Record’ (record type 0x00), with the exception of the last line, which is a standard Intel Hex File ‘end of file’ (EOF) indicator.

The ‘address’ field for each line refers to the starting byte address of the first byte of the line. As you can see in the example, the first 16 data bytes are intended to be located starting at byte-address 0x0100. The next 16 data bytes (on the second row) are located starting at address 0x0110. So this file defines what we would call a linear sequence of bytes.

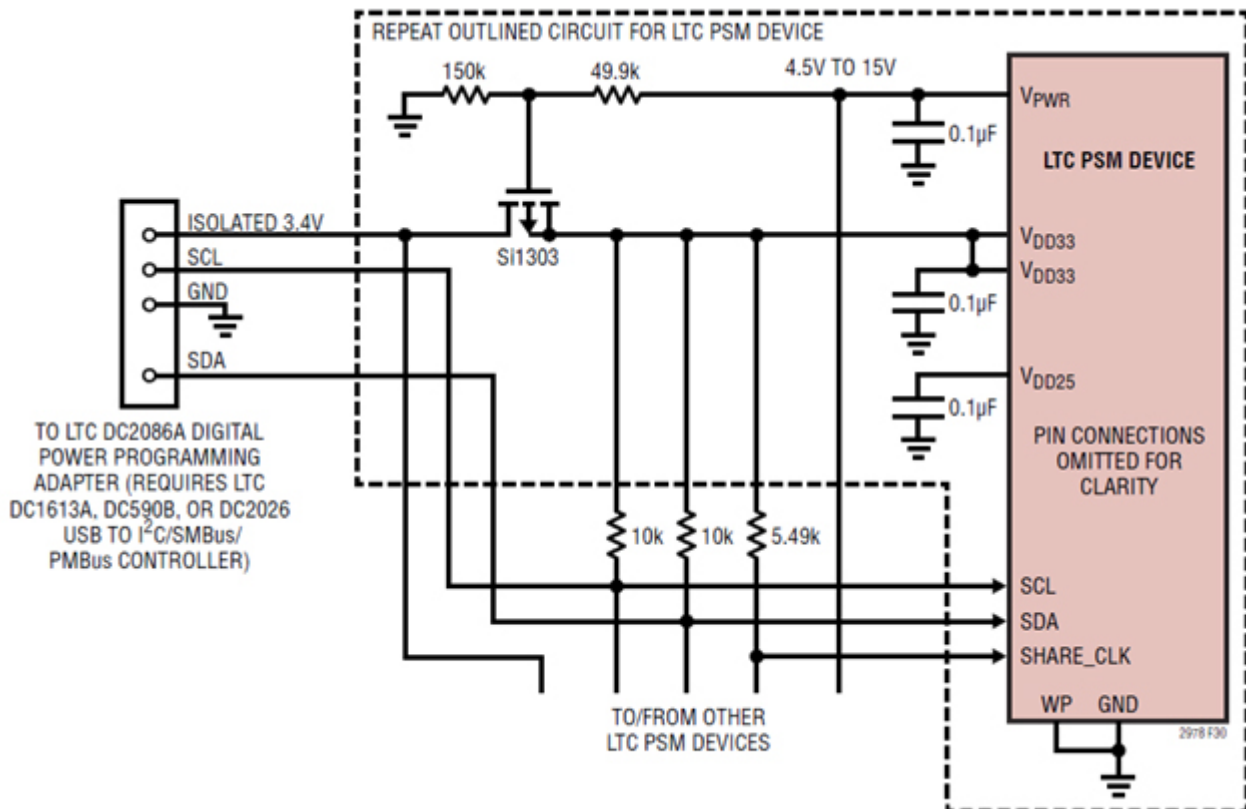
LT Programming Hex Files will always represent a linear/contiguous sequence of raw bytes starting at byte address 0x0000 and going upward.

The In-System Programming software has completed the first step of processing this file when it has built a raw byte buffer by reading the raw bytes of the hex file.

## 6.6 How to power device In-Circuit without powering the entire system?

By adding a PFET inline with the PSM devices' power inputs, you can keep the PSM products' brains alive even without powering the whole system. This is very useful for debugging where you might want the rails to turn off but to continue to monitor the large amount of useful telemetry data and fault logs available. The latched status registers won't get reset since the power is continuous making it easier to spot which actual faults are present.

The circuit below recommends a DC2086A Power Programming Adapter to use inbetween the USB PMBus Controller (i.e. DC1613A). While this is always recommended, you can sometimes connect your USB PMBus Controller directly to your board using a 12pin or 14pin cable (vs. 4pin, 12pin, or 14pin on the DC2086A) as long as your total current draw is less than the absolute maximum of 100mA. The DC2086A supplies up to 2.3A on the same power rails.



## 6.6.1 How Do I Program Controllers without VIN

### 6.6.1.1 How Do I Program Controllers without VIN?

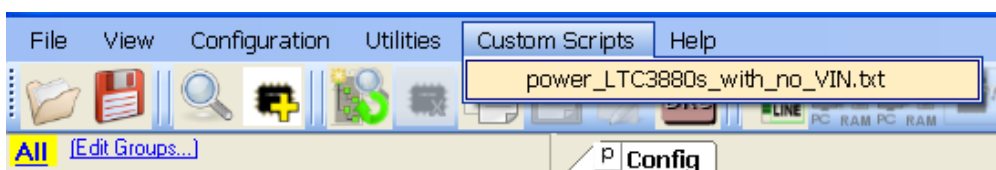
The FAQ Entry [How to power device In-Circuit without powering the entire system?](#)(see page 89) describes how to power the logic of Controllers and Managers without VIN. This allows you to communicate over PMBus to the device without board main power. This is a safe state for programming devices because power stages will not be powered.

In the Controller case, the Controller will not be fully reset. Even though the PMBus is active, LTpowerPlay and/or your system controller cannot communicate with the device using its expected address. This is because the device has not read the ASEL pins, nor the device read its MFR\_ADDRESS register from its EEPROM. A controller will only read and apply the address information if VIN is applied.

There is a way to force the Controller to read its MFR\_ADDRESS and apply the ASEL pins, so that LTpowerPlay and/or your system controller can communicate with the device.

### 6.6.1.2 Manually Forcing the Address

Under the Custom Scripts menu of LTpowerPlay, select the power\_LTC3880s\_with\_no\_VIN.txt item.



After this script runs, all Controllers on the bus will have their expected address.

#### 6.6.1.3 After the Address is Fixed

Once the address is fixed, you can open a LTpowerPlay project that has matching addresses, or use the LTpowerPlay programming utility found in the Utilities menu.

## 6.7 What's the difference between In-Flight Update (IFU) & In-Circuit Programming (ICP)?

### 6.7.1 In-Flight Update

- Tested and Supported Code
- This is best for customers to implement
- Contact Mike Jones for support

### 6.7.2 In-Circuit Programming

- Example C code for learning purposes only
- This is best for OEMs to implement (like BPM)
- Contact Nick Vergunst for support

## 6.8 What are the different CRC Options Available?

### 6.8.1 Overview

There are several CRCs used by PSM Controllers/Managers and tools to ensure reliability of operation and integrity of manufacturing processes. They are:

1. CRC within the device EEPROM to guarantee integrity of the device settings
2. CRC within a USER register to guarantee integrity of the programming process
3. CRC of OEM files to guarantee integrity of the programming process
4. CRC of SL configuration files to guarantee integrity of the programming process

### 6.8.2 Internal CRC (1)

The internal CRC are extra bytes stored in the device EEPROM that are calculated from EEPROM data. When the STORE\_USER\_ALL command is executed, the device calculates the CRC as part of the store process and places the CRC in the EEPROM. When a device is powered on or reset, the device fetches data from the EEPROM and stores it in RAM. During transfer, the CRC is recalculated from the EEPROM data and compared with the CRC also stored in EEPROM. If there is a mismatch, the device is not allowed to power on rails. The EEPROM has to be re-written to enable device operation.

Bulk programming the device sends data directly to EEPROM using PMBus commands, and the CRC is externally calculated and sent with the data. When the device is then power cycled or reset, the same comparison behavior described above takes place.

The purpose of this CRC is to ensure the device never operates with bad data.

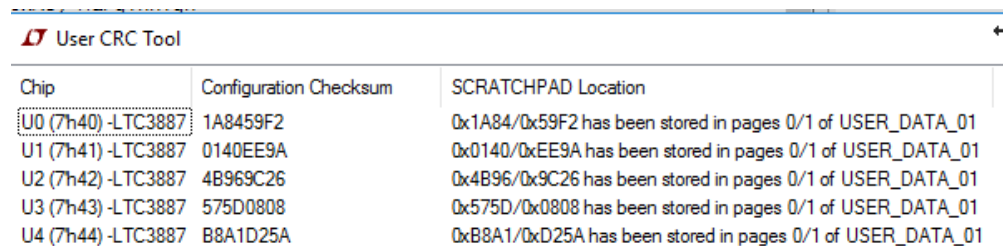
### 6.8.3 Register CRC (2)

The register CRC is a value placed into a USER register by external tools. The device does not use this value in any way. The device can't use this value because USER registers are just scratch pad registers for users. The fact that a CRC is placed in one of these is purely a practical convention.

When a device is programmed with one of the LTC tools/methods, a CRC is calculated using a subset command/register values. You can think of this as a signature of a particular device configuration. To work properly, any registers with dynamic contents, such as status or DAC values (servo), must not be part of the calculation. Essentially, this means the CRC covers the static settings of the device.

Because the device cannot use this value internally, if it is modified by firmware, the device will never know and will not change behavior. If any command/register values are changed, the device will act on them, but the inconsistency of the CRC will not affect the behavior of the device.

This means the CRC's value is limited to use by tools and engineers using LTpowerPlay (LTPP) or other means of reading data from the device.



| Chip               | Configuration Checksum | SCRATCHPAD Location  |
|--------------------|------------------------|--|
| U0 (7h40) -LTC3887 | 1A8459F2               | 0x1A84/0x59F2 has been stored in pages 0/1 of USER_DATA_01 |
| U1 (7h41) -LTC3887 | 0140EE9A               | 0x0140/0xEE9A has been stored in pages 0/1 of USER_DATA_01 |
| U2 (7h42) -LTC3887 | 4B969C26               | 0x4B96/0x9C26 has been stored in pages 0/1 of USER_DATA_01 |
| U3 (7h43) -LTC3887 | 575D0808               | 0x575D/0x0808 has been stored in pages 0/1 of USER_DATA_01 |
| U4 (7h44) -LTC3887 | B8A1D25A               | 0xB8A1/0xD25A has been stored in pages 0/1 of USER_DATA_01 |

From within LTPP, if you use the menu Utilities > Configuration/CRC, you will see a display like the above picture. This includes the CRC (checksum) value, and the location.

The following [document](#)<sup>26</sup> describes the algorithm for calculating the CRC (checksum)

### 6.8.4 OEM CRC (3)

The OEM CRC is a calculation on OEM files as part of the OEM export system found in the LTPP File menu. App Note 145 Options 2A, 2B, 3A, and 4B are programming methods that use OEM files. An OEM file is a programming file for a single device, typically in a socket. In contrast, an ISP file is a programming file for multiple devices, typically in a system.

You may want to know why these are different. In a socket, a device can be programmed using the global address 0x5B, and then the state of the address select pins does not matter. Furthermore, the device cannot power on, so the primary focus is on correct programming of a device. In a system, the address 0x5B cannot be used for the programming because each device has different settings. Furthermore, it has to be programmed in a way that there are no dangerous side effects on the load, so the primary focus is on correct programming of a system. This can be confusing if you have looked at the OEM or ISP files, because their content has the same formatting. The data language that contains the data and programming instructions is common to both file types, but the content is different for each file type.

<sup>26</sup> <http://www.ltpowerplay.com/help/content/Description%20of%20the%20LTpowerPlay-Calculated%20Configuration%20CRCAlgorithm%20-rev-0.1.pdf>

When OEM files are created for LTEXpress, the CRC values are displayed in the LTPP GUI during the ordering process. When the OEM files are exported for other methods, a zip file is created. The CRC values are contained in the documents of the zip file.

Refer to [AN145<sup>27</sup>](#) for details on these CRCs and how they are documented and used.

Now that you know how the value is created and used, you can leverage the CRC in your production environment in the following ways:

- Firmware can confirm the value
- Manufacturing tools can confirm the value
- Engineers using LTPP can confirm the value in the GUI
- Socket programming processes can pull QA samples and confirm the value

In every case, this means confirmation of the signature put into the device by tools.

Note that when using the LTPP Programming utility, programming a device creates the same algorithm as an ISP file, and executes it, and therefore the same CRC will be placed in the USER command.

Note that when using ISP files and firmware to program devices, it will place the same CRC in the USER command. This is because the CRC is calculated by LTPP and placed in the ISP data.

Basically, this means that all methods that program devices using OEM, ISP files, or the LTPP Programming utility all result in a CRC in a USER command that is stored in EEPROM.

### 6.8.5 SL CRC (4)

SL devices are ones that have an LTC-assigned part number that you can order. This order is backed by an OEM file, that is used during manufacturing. To keep track of the OEM file / part number relationship, a CRC is used. This CRC is the same CRC generated when exporting an OEM file. The process used by LTEXpress and SL are different, but both use the generated CRC to keep track of the files.

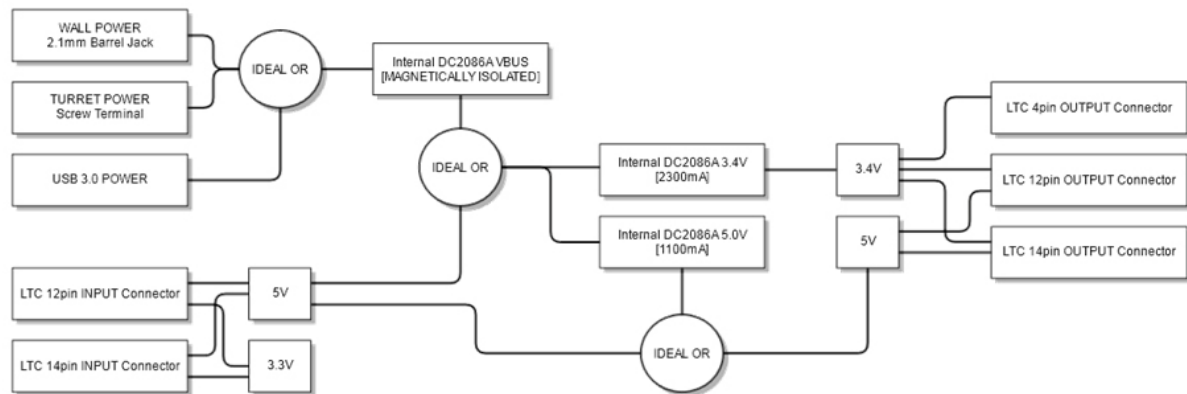
## 6.9 What if I need more than 100mA from the USB PMBus Controller (i.e. DC1613A)?

In many cases the DC1613A or compatible USB PMBus Controller can be used directly with demo boards and customer boards that have minimal components. For instance most of our demo boards have 1 device or a handful of devices that you want to power from the dongle. But what happens if you are lucky enough to have the problem that you have so many components that you are overloading the dongle?

The answer lies in the DC2086A, the Powered Programming Adapter, which can supply up to 2.3A on the 3.3v power rail. A block diagram of the DC2086A is shown below.

---

<sup>27</sup> <http://cds.linear.com/docs/en/application-note/AN145f.pdf>



#### Notes on the DC2086A:

- 4.5v to 18v DC Input (3 input power options)
- Beefed up 3.4v output up to 2.3A
- Beefed up 5.0v output up to 1.1A
- Full Isolation between input and output
- Provides cable adaptation between 12pin and 14pin inputs to 12pin, 14pin, and a smaller footprint 4-pin header
- USB is Power Only NOT FOR DATA it is for power

#### Images of the DC2086A



## 6.10 What is the format of the LT Programming File?

The LT Programming Hex File is a standard Intel 386 hex file, also known as I32HEX or INTEL 32 Hex file Format. For detailed information on the Intel hex format, please reference this Wikipedia article: [http://en.wikipedia.org/wiki/Intel\\_HEX](http://en.wikipedia.org/wiki/Intel_HEX).

At the lowest level of interpretation, the LT Programming Hex file represents a simple linear sequence (or ‘array’) of raw bytes. Higher level structures are imposed on this sequence of bytes to define a set of what we call “LT Programming Records”.

Each of these LT Programming Records instructs the In-System Programming software to perform a particular operation. As an example, one LT Programming Record type, PMBUS\_WRITE\_BYTE\_NOPEC instructs the In-System Programming software to execute a PMBus/SMBus write byte operation to a specified address with a specified command code and data byte.

Generally LT Programming Record ‘instructions’ are somewhat low level – that is, they do not burden the In-System Programming software with understanding the algorithm necessary to program a device – the In-System Programming software simply executes a series of simple instructions, and in the order they are defined in the LT Programming Hex File.

Once the In-System Programming software has successfully processed all the LT Programming Record ‘instructions’ it will have successfully programmed and verified the NVM for all chips for a particular system as defined in the file.



## 6.11 What is the Programming Power Circuit?

The Programming Power Circuit is referenced in many places and refers to a circuit capable of powering the LT devices independent of system or board power. This is extremely useful for debugging and essential for in-circuit or in-system programming (ICP or ISP).

Please refer to "[How to power device In-Circuit without powering the entire system?\(see page 89\)](#)" for one such example circuit.

## 6.12 What options do I have for getting Programmed Devices?

### 6.12.1 Linear Technology Options

#### 6.12.1.1 LTExpress & Linear Technology SL#

We offer programming options for all of our devices. By going with LTC, we can get you individual pre-programmed parts quickly as well as in mass. Because we program them ourselves, any fallout that may be experienced at other vendors is eliminated as we only ship you good programmed and tested devices. We can do marking on the top of the package, labels, and quick turn orders. Contact sales for more information.

## 6.12.2 3rd Party Programming of LTC Devices

### 6.12.2.1 BPM

We plan on supporting BPM fully with all devices being available at or near RFS. We also have many more non-RFS devices in the queue being prepared. To use this method, you must buy the hardware which includes your base programmer and a socket module specific to the device you want to program. You then program via BPWin software

- Supported: (13+) LTC2933, LTC2936, LTC2974, LTC2975, LTC2977, LTC2978, LTC2980, LTC3880, LTC3882, LTC3883, LTM2987, LTM2988, LTM4676

### 6.12.2.2 Arrow

Arrow uses BPM Hardware so all BPM supported devices are supported at Arrow at or near RFS. Arrow will program the devices without hardware investment or software maintenance on your end.

### 6.12.2.3 Asset Intertech

Supports all In-Circuit Programming devices when properly connected to JTAG device. If no JTAG devices are available or connected to the PMBus, they support programming through a "ScanLite" Board.


### 6.12.2.4 JTAG Technologies

Supports all In-Circuit Programming devices when properly connected to JTAG device.

### 6.12.2.5 System General – \*\* NOTE NOT RECOMMENDED \*\*

System General currently supports the LTC2974, LTC2978, and LTC3880. We do not plan on supporting them for any other devices. We have only verified and tested early programming methods with the 3 mentioned devices. As our algorithms change and improve periodically it is not a guaranteed statement that their programming will continue to work for all Linear Technology devices. We do not recommend using System General as a solution. If you do use System General, you will have to buy the hardware (Socket Module & Base Programmer) or have them program your devices.

## 6.13 What Programming Option Should I choose?

 Unknown macro: 'gliffy'

Note 1: Details about the Programming Power Circuit can be found here: [What is the Programming Power Circuit?](#) (see page 95)

## 6.14 Where does LTpowerPlay store it's computed CRC in the part?

LTpowerPlay computes a CRC that uniquely identifies the configuration and may store this into scratchpad EEPROM of your part. We call this the "LTpowerPlay Configuration Checksum", or unique configuration identifier. This CRC is used for a very specific purpose. Consult the FAQ on various CRC types used to understand other CRCs used.



This CRC, stored into the part, provides a convenient way to read the register (via firmware for instance) to identify the original configuration. This CRC will be stored into the project file and into generated programming files of all types, regardless of how you program your devices.

The location may differ depending on the part. To determine if/where LTpowerPlay stores the configuration identifier for your part. Do the following:

- Launch LTpowerPlay
- Open your project (.proj) file
- On the menu, select 'Utilities > Configuration CRC...'
- Select the chip of interest in the list, using the chip identifier in the first column
- Note the 32-bit configuration checksum (CRC) in the second column
- Note the location(s) used to store the 32-bit value in the notes in the 3rd column.

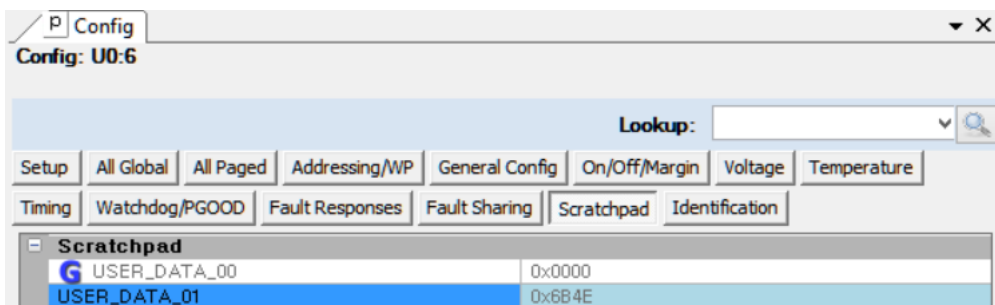
### 6.14.1 Example:

In the example shown below, we show a project file with 3 devices: LTC2977, LTC2978, and LTC3880. Here is the display of the Configuration/CRC Utility:

| User CRC Tool      |                        |  |
|--------------------|------------------------|--|
| Chip               | Configuration Checksum | SCRATCHPAD Location  |
| U0 (7h37) -LTC2977 | 6B4E84B8               | 0x6B4E/0x84B8 has been stored in pages 6/7 of USER_DATA_01 |
| U1 (7h38) -LTC2978 | EA298873               | 0xEA29/0x8873 has been stored in pages 6/7 of MFR_SPARE3   |
| U2 (7h39) -LTC3880 | 1B0AFAA7               | 0x1B0A/0xFAA7 has been stored in pages 0/1 of USER_DATA_01 |

So the 32-bit configuration checksum for the U0 device (LTC2977) is 0x6B4E884B8. The MS word (0x6B4E) of this 32-bit CRC is stored in page 6 of USER\_DATA\_01, and the LS word (0x84B8) is stored in page 7 of USER\_DATA\_01.

After closing this window, you may inspect the values by selecting the 'Scratchpad' tab of the Configuration window. The example below shows the value USER\_DATA\_01 for channel U0:6:



The register information tab provides further detail on this register, including the command code, protocol, and how to read it. The below example shows the register information display after selecting U0:6 in the tree. The contextual protocol help shows exactly how you would read the value of this register for page 6:

Register Information
✖

**Description:** **USER\_DATA\_01**

*A paged word of scratchpad reserved for use by LTpowerPlay. This is typically used for storage of GUI computed configuration CRC values.*

**Register Info:**

|               |       |
|---------------|-------|
| Command Code: | 0xB1  |
| Data Type:    | Word  |
| Scope:        | Paged |

**Value Analysis:**

|                      |          |
|----------------------|----------|
| GUI Value (hex):     | 0x6B4E   |
| GUI Value (meaning): | '0x6B4E' |

**Example Write Sequence:**

Example Code for writing the USER\_DATA\_01 register:

(This Register is Read Only)

**Example Read Sequence:**

Example Code for reading the USER\_DATA\_01 register:

```
// write PAGE to 6
smbus_write_byte( 0x37, 0x00, 0x06 );

// read USER_DATA_01
smbus_read_word( 0x37, 0xB1);
```

**Read Protocol sequence:**

write PAGE to 6

|   |               |      |   |              |   |           |   |   |
|---|---------------|------|---|--------------|---|-----------|---|---|
| 1 | 7             | 1    | 1 | 8            | 1 | 8         | 1 | 1 |
| S | Slave Address | Wr   | A | Command Code | A | Data Byte | A | P |
|   | 7'b0110_111   | 1'b0 |   | 8'h00        |   | 8'h06     |   |   |

read USER\_DATA\_01

|   |               |      |   |              |   |    |               |      |   |               |   |                |   |   |
|---|---------------|------|---|--------------|---|----|---------------|------|---|---------------|---|----------------|---|---|
| 1 | 7             | 1    | 1 | 8            | 1 | 1  | 7             | 1    | 1 | 8             | 1 | 8              | 1 | 1 |
| S | Slave Address | Wr   | A | Command Code | A | Sr | Slave Address | Rd   | A | Data Byte Low | A | Data Byte High | N | P |
|   | 7'b0110_111   | 1'b0 |   | 8'hB1        |   |    | 7'b0110_111   | 1'b1 |   |               |   |                |   |   |

*NOTE: Though reading the CRC from this EEPROM scratchpad provides a convenient mechanism to identify configurations, the part itself does NOT keep this CRC up to date. So if your host controller ever changes the configuration and issues a STORE\_USER\_ALL command without updating this CRC, reading the CRC will not be a reliable way to identify the configuration. The main purpose of this CRC is to validate that the correct configuration was programmed into a loose part before soldering onto the board, and the correct chip is located in the expected place on the board.*

## 6.15 Where do I find In Flight Update Information?

Where do I find In Flight Update Information?

Linduino has a full example sketch under User Contributed/DC1962/program. This code programs using the Record Syntax described on our website. Follow the links below. If anyone needs help implementing, Michael Jones maintains the code and helps customers.

[Linduino](#)<sup>28</sup>

[ISP Record Syntax](#)<sup>29</sup>

## 6.16 Why Must all Devices Share a Common Base Address for In System Programming to Work?

If all Devices on your board do not share a common base address, you may see errors when attempting to generate an in system programming (.isphex) file, or when attempting to program devices in-system using the programming-utility or command-line in system programming methods. These methods are designed to work in any number of scenarios where device addressing may be corrupted (due to memory corruption), by first recovering the system to a common base address using a special I2C address that all PSM devices will respond to (7-bit address 0x5B).

Near the beginning of the in-system programming process, the system issues a byte-write to the PSM Base Address command (0xE6) to all devices on the board (using the global 0x5B address) recover the device addresses.

Assuming that you have followed proper address planning for your board, communication will work after this step regardless of the prior state. This insures that in system programming is possible under all circumstances, but for this to work the system must share a common base address. If you have not setup your addressing following these guidelines then these methods will not work, and will produce errors.

A more detailed explanation and full documentation on address planning is provided in Application Note 152 here:

<http://cds.linear.com/docs/en/application-note/an152f.pdf>

---

<sup>28</sup> <http://www.linear.com/linduino>

<sup>29</sup> <http://www.linear.com/solutions/5710>

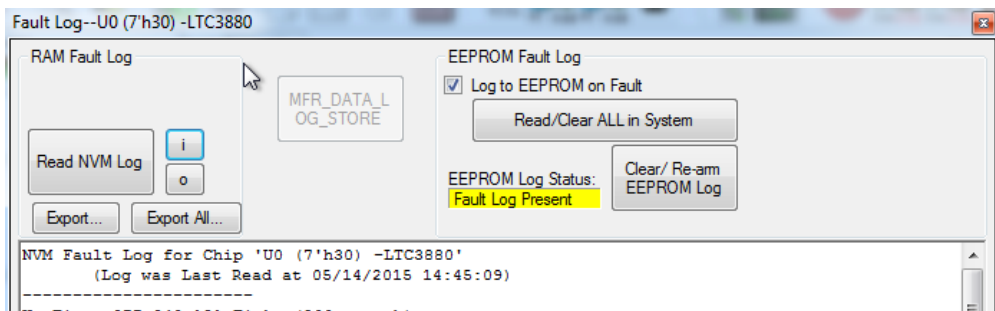
## 7 Firmware FAQ

This FAQ Section contains Frequently Asked Questions about Host Controller Firmware

### 7.1 Can I print fault log data generated by my firmware?

Can I print fault log data generated by my firmware?

LTpowerPlay has an import function to pretty printing fault log data:



By pressing the "i" button you can select a file and it will be formatted in the window at the bottom.

The format of the file must be Intel Hex, holding the hex values in data records. A general description is found at [Wikipedia](http://en.wikipedia.org/wiki/Intel_HEX)<sup>30</sup>.

The data sheet describes how to read the fault log from each device, and there is Linduino code demonstrating how to read the log and pretty print it. But, there is no Linduino code to generate an Intel Hex file.

### 7.2 Do I Need Polling?

#### 7.2.1 Background

All PSM devices can be too busy to accept a command. The most common times a device may be busy are:

- Resetting
- Turning on or off rails
- Memory operations
- Fault logging to EEPROM

To prevent NACK responses, polling should be used.

#### 7.2.2 Solution

There are different types of polling:

- Poll for ACK
- Poll for not BUSY
- Poll for not calculating

<sup>30</sup> [http://en.wikipedia.org/wiki/Intel\\_HEX](http://en.wikipedia.org/wiki/Intel_HEX)

Polling for ACK is using a safe command like read PAGE to see if the device can respond. Polling for BUSY is reading the MFR\_COMMON register BUSY bit to see if the device is ready to process a command. Polling for not calculating is reading the MFR\_COMMON register calculation PENDING bit to see if the device is using a slow calculation.

[AN135](#)<sup>31</sup> has details on the basics of polling. Linduino has examples of how to implement it.

### 7.2.3 Alternatives

Adding a delay before every command can solve the problem. However, it is difficult in complex firmware to characterize the delays and guarantee the firmware will never fail in a deployed system. Therefore, LTC recommends using polling for all firmware.

## 7.3 How are Voltages and Current Represented in PMBus Commands?

### 7.3.1 How are Voltages and Current Represented in PMBus Commands?

PMBus commands that represent decimal values are encoded by two formats: L11 and L16. The PMBus standard calls these LINEAR11 and LINEAR16 respectively. Current is always encoded as L11, but voltage may be encoded as either, depending on the command. Each data sheet gives the format in its command table. For example:

| COMMAND NAME           | CMD CODE | DESCRIPTION   | TYPE     | PAGED | DATA FORMAT | UNITS | NVM | DEFAULT VALUE   | PAGE               |
|------------------------|----------|---|----------|-------|-------------|-------|-----|-----------------|--------------------|
| VOUT_TRANSITION_RATE   | 0x27     | Rate the output changes when VOUT commanded to a new value.   | R/W Word | Y     | L11         | V/ms  | Y   | 0.25<br>AA00    | <a href="#">78</a> |
| FREQUENCY_SWITCH       | 0x33     | Switching frequency of the controller.  | R/W Word | N     | L11         | kHz   | Y   | 350<br>0xFABC   | <a href="#">70</a> |
| VIN_ON                 | 0x35     | Input voltage at which the unit should start power conversion.  | R/W Word | N     | L11         | V     | Y   | 6.5<br>0xCB40   | <a href="#">71</a> |
| VIN_OFF                | 0x36     | Input voltage at which the unit should stop power conversion.   | R/W Word | N     | L11         | V     | Y   | 6.0<br>0xCB00   | <a href="#">71</a> |
| IOUT_CAL_GAIN          | 0x38     | The ratio of the voltage at the current sense pins to the sensed current. For devices using a fixed current sense resistor, it is the resistance value in mΩ. | R/W Word | Y     | L11         | mΩ    | Y   | 1.8<br>0xBB9A   | <a href="#">74</a> |
| VOUT_OV_FAULT_LIMIT    | 0x40     | Output overvoltage fault limit.   | R/W Word | Y     | L16         | V     | Y   | 1.1<br>0x119A   | <a href="#">72</a> |
| VOUT_OV_FAULT_RESPONSE | 0x41     | Action to be taken by the device when an output overvoltage fault is detected.  | R/W Byte | Y     | Reg         |       | Y   | 0xB8            | <a href="#">81</a> |
| VOUT_OV_WARN_LIMIT     | 0x42     | Output overvoltage warning limit.   | R/W Word | Y     | L16         | V     | Y   | 1.075<br>0x1133 | <a href="#">72</a> |

This table, from the LTC3887, says that the command VOUT\_OV\_FAULT\_LIMIT is L16 (LINEAR16) and VIN\_OFF is L11 (LINEAR11).

LTpowerPlay performs conversions to and from decimal values for you, and Linduino code has conversion routines that do the same.

However, it is helpful to understand how the format works in case you have to calculate a few values by hand.

#### 7.3.1.1 Basic Representation

The basic idea of both formats is to represent the decimal value with the following equation:

$$Y = X * 2^{-N}$$

<sup>31</sup> <http://cds.linear.com/docs/en/application-note/an135f.pdf>

X and N are integer values, because they have to be sent over PMBus as binary data. This format also uses names for the values:

- X is the mantissa
- N is the exponent
- Y is the value

One way to think of this is the exponent (N) is a scale factor, or a resolution, that is applied to the mantissa. Changing N affects the range of value it can express when multiplied by the mantissa. There is an inherent trade off between resolution and range.

Let's use an example. Suppose:

- N = -10
- X = 870
- Units are microvolts

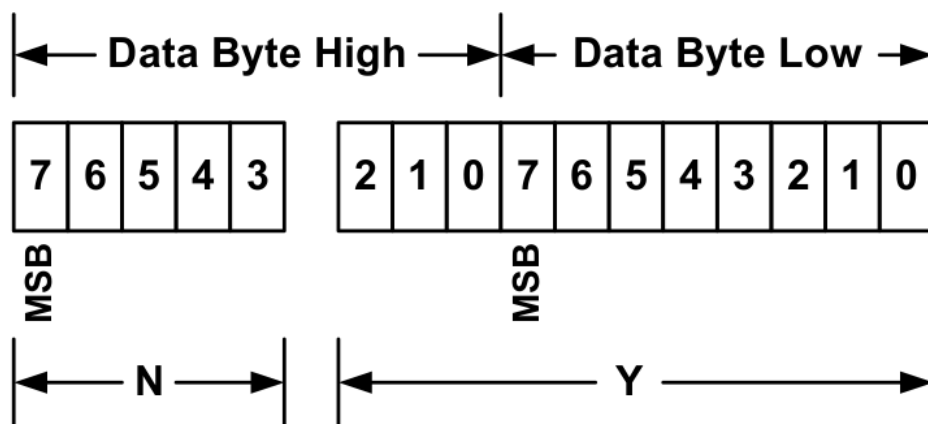
$$2^{-10} \Rightarrow 1/1024 \Rightarrow 977\mu\text{V}$$

$$977\text{E-}6 * 870 \Rightarrow 0.85\text{V}$$

If you think of this like a DAC (Digital To Analog Converter) the LSB is 977uV and the value is 870, which means the output voltage is 0.85V.

### 7.3.1.2 L11 Representation

The L11 format encodes N and Y in a two byte format. N is a 5 bit two's complement integer. Y is an 11 bit two's complement integer.

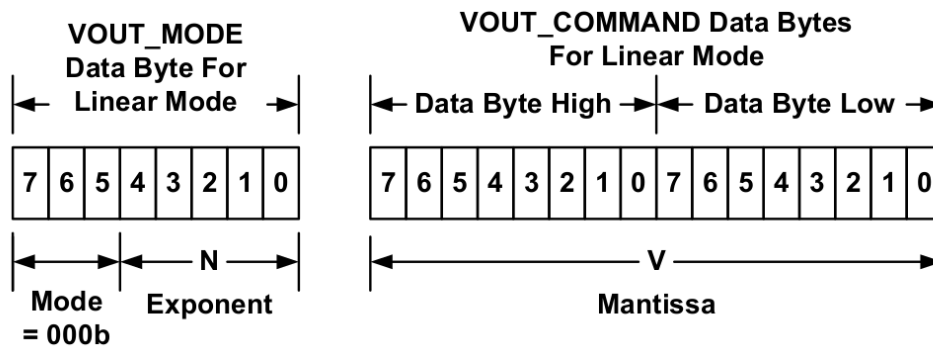


This shows that N is on the left side, in the high byte. X is split between the two bytes.

When the value is written by a master to a slave, the master controls the trade off between resolution and range. When the master reads the value, the slave controls this trade off.

### 7.3.1.3 L16 Representation

The L16 format does not encode both N and X in the data the master sends to the slave. Instead, each slave has a fixed N, and the master only sends X as a 16 bit value.



The master can read N from the VOUT\_MODE command, and it is a 5 bit two's complement value. Because it is read only, it means the resolution/range is fixed, and the master cannot change it.

The X value is a 16 bit data value, and the above example shows it for the VOUT\_COMMAND, but other commands that are L16 are the same.

The master does not have any control over the resolution/range trade off, so conversion between decimal and L16 is simpler.

#### 7.3.1.4 Conversions

The easiest way to encode a decimal value is to use LTpowerPlay. Let's turn a VOUT\_COMMAND value into L16.

First, find the value:

| Output Voltage      |                          |
|---------------------|--------------------------|
| VOUT_OV_FAULT_LIMIT | +10.0 % above/below VOUT |
| VOUT_OV_WARN_LIMIT  | +7.5 % above/below VOUT  |
| VOUT_MARGIN_HIGH    | +5.0 % above/below VOUT  |
| VOUT_COMMAND        | 1.0000 V                 |
| VOUT_MARGIN_LOW     | -5.0 % above/below VOUT  |
| VOUT_UV_WARN_LIMIT  | -7.5 % above/below VOUT  |
| VOUT_UV_FAULT_LIMIT | -10.0 % above/below VOUT |

Now, select the RegInfo Tab:

**Description:** **VOUT\_COMMAND**  
*Nominal DC/DC converter output voltage setpoint.*

**Register Info:**

|               |               |
|---------------|---------------|
| Command Code: | 0x21          |
| Data Type:    | LinearFloat16 |
| Scope:        | Paged         |

**Value Analysis:**

|                      |        |
|----------------------|--------|
| GUI Value (hex):     | 0x1000 |
| GUI Value (meaning): | '1V'   |

**Example Write Sequence:**

Example Code for writing the VOUT\_COMMAND register:

```
// write PAGE to 0
smbus->writeByte( 0x5B, 0x00, 0x00 );
// write VOUT_COMMAND to 1V
smbus->writeWord( 0x5B, 0x21, 0x1000 );
```

**Write Protocol Sequence:**

**write PAGE to 0**

|   |               |      |   |              |   |           |   |   |
|---|---------------|------|---|--------------|---|-----------|---|---|
| 1 | 7             | 1    | 1 | 8            | 1 | 8         | 1 | 1 |
| S | Slave Address | Wr   | A | Command Code | A | Data Byte | A | P |
|   | 7'b1011_011   | 1'b0 |   | 8'h00        |   | 8'h00     |   |   |

**write VOUT\_COMMAND to 1V**

|   |               |      |   |              |   |               |   |                |   |   |
|---|---------------|------|---|--------------|---|---------------|---|----------------|---|---|
| 1 | 7             | 1    | 1 | 8            | 1 | 8             | 1 | 8              | 1 | 1 |
| S | Slave Address | Wr   | A | Command Code | A | Data Byte Low | A | Data Byte High | A | P |
|   | 7'b1011_011   | 1'b0 |   | 8'h21        |   | 8'h00         |   | 8'h10          |   |   |

This tab shows the command Code, the Format, etc. Look at "Value Analysis." This shows the value in hex as 0x1000 and the decimal value of 1V. It even shows the code required to write or read the value. This value already accounted for the exponent (N) found in VOUT\_MODE.

There is no way to enter a hex value and then convert it to a decimal value. You could write it with the I2C utility, then have LTpowerPlay read it back, but this might program a value incorrectly, so I don't recommend it. Don't risk an over voltage from a mistake.

If you using LTpowerPlay for debug, telemetry polling automatically converts them to decimal, so you don't need to do any manual conversions anyway. Unless you are writing code...

### 7.3.1.5 Conversions in Code

Fortunately, Linduino code has all you need. Download the Linduino code from:

[www.linear.com/linduino](http://www.linear.com/linduino)<sup>32</sup>

<sup>32</sup> <http://www.linear.com/linduino>



unzip it and look for:

LTSketchbook/libraries/LT\_PMBUS/LT\_PMBus.c

Go all the way to the bottom and find these functions:

- L16\_to\_Float
- Float\_to\_L16
- L11\_to\_Float
- Float\_to\_L11

These are implementations that are easy to understand, and well commented. Just steal them and move on.

However, if you need super speed calculation or don't have "power" commands, look for:

LTSketchbook/libraries/LT\_PMBUS/LT\_PMBusMath.cpp

These are very highly optimized routines that are no easy to understand, but are fast and well tested.

If you use either of these sets of commands, you can blissfully write code and not have to worry about it.

#### 7.3.1.6 Tricks

There are tricks you can play not covered here, but noted in case you want to try them:

- Precode values and store them in code as binary or hex
- Pick an N, precompute the LSB size, and only calculate the mantissa
- Bump values up and down by one LSB with a add/sub 1 from a mantissa with N fixed

Just be careful with reading L11. Don't assume that N will not change. It is perfectly legal for a slave to choose N dynamically.

#### 7.3.1.7 Questions

If you have more questions or need help you can refer to:

- AN135
- [http://pmbus.org/Assets/Present/Using\\_The\\_PMBus\\_20051012.pdf](http://pmbus.org/Assets/Present/Using_The_PMBus_20051012.pdf)
- [mjones@linear.com](mailto:mjones@linear.com)

The pmbus.org PDF has examples of conversions done by hand.

## 7.4 How Can I Extend the Retention Time of PSM Devices

### 7.4.1 Background

Power System Management devices store settings in EEPROM. When a device is reset or powered on, EEPROM settings are copied from EEPROM to RAM. Operation of the device is based on the values in RAM. The data retention is specified as 10 years when EEPROM is programmed under the given data sheet conditions.

### 7.4.2 How EEPROM is Programmed

There are two ways to programming EEPROM: STORE\_USER\_ALL and MFR\_EE\_DATA (not available for LTC2978).

STORE\_USER\_ALL transfers the contents of RAM to EEPROM.

MFR\_EE\_DATA transfers data from PMBus to EEPROM.

### 7.4.3 Extending Retention

EEPROM retention is extended by re-writing the EEPROM data, while not violating the maximum number of write cycles in the data sheet. For a typical design, the suggested interval for re-writing is 6 months.

### 7.4.4 Practical Implementation Concerns

It is important that writing EEPROM is done under the following safe conditions:

1. Temperature less than 85 deg C
2. VDD33 supply in the data sheet operating range

By meeting this conditions, the retention time will be increased. By not meeting these conditions, the retention time may be decreased.

Firmware should measure the die temperature of the device before re-writing EEPROM. There is no way for the device to measure VDD33, so firmware must rely on product validation. If the VDD33 is generated by the device and not overloaded, it will always be in specification range. Designs that supply VDD33 externally will have to rely only on product validation.

#### 7.4.4.1 STORE\_USER\_ALL

The STORE\_USER\_ALL command is the simplest implementation, however, if power is lost during programming, it offers no recovery. When power is lost during this command, the EEPROM will have CRC errors at power cycle, which will prevent the device from finishing reset, and therefore no rails will power on. This will cause no harm, but it typically means a field service return for repair. The board can be repaired with LTpowerPlay or other means used during manufacturing.

Programming takes up to 4 seconds. One can calculate the probabilities of failure based on this time. The probability of failure can be minimized by scheduling the time of the store with some random data. This way if a data center suffers a power outage during programming, only one board will be affected.

#### 7.4.4.2 MFR\_EE\_DATA

This technique is more complicated, but more robust. It is more complicated in that it requires unlocking, erasing, locking, programming, etc. A typical sequence is:

- Unlock
- Erase
- Lock
- Unlock
- Program
- Lock
- Unlock
- Verify
- Lock

It is more robust because it can repair a board that has a CRC failure in a fix-on-fail mode.

This methodology can be used in two forms:

- Read data from working device and apply later

- Incorporate data into design

We do not recommend the first approach, because when there is a CRC failure, there are additional recovery steps. The main reason is that during a CRC failure:

1. The base address becomes default
2. Write protect is in an unknown state
3. PEC required is in an unknown state
4. Addresses may overlap

When the data is incorporated in the design, one ports the In Flight Update code found in the Linduino code base. This code handles all the difficult error cases and has been established as very robust.

### 7.4.5 In Flight Update (aka In System Programming)

This code is an engine that applies ISP (In System Programming) data exported from LTpowerPlay. The robust algorithm is part of LTpowerPlay and is maintained by LTC. The ported code is only a stable engine that follows the instructions of the ISP data.

### 7.4.6 Design Considerations

STORE\_USER\_ALL only requires live firmware and working firmware. In Flight Update requires the same, but for fix-on-fail, there is one more requirement. A BMC (Board Management Controller) cannot re-write EEPROM of devices on the PMBus if it does not have power itself. Typical designs that must fix-on-fail, power the BMC from an LDO derived off an IBC, not derived off a PSM controlled rail. This ensures that firmware always has access to all PMBus devices, even when there are CRC failures.

Note that if there are I2C MUXs in the design, these must also have power, as well as the VDD33 of each PSM device.

Generally the PSM devices will have power from an IBC in a CRC failure case, which ensures the VDD33 is available for In Flight Update. If there is a design case where the power supplies of a PSM device is lost because it is powered from another PSM device, provisions must be made in the design to supply the VDD33 externally. This can be done with a permanent LDO, or using the PFET techniques used for the DC1613 dongle connection described in the data sheets.

### 7.4.7 Closing

LTC actively assists customers with programming EEPROM with firmware. For more help contact your local FAE and copy "mjones AT linear DOT com" on the e-mail.

## 7.5 How can I write maintenance code without polling?

### 7.5.1 How can I write maintenance code without polling?

Before digging into code without polling, note that polling is always recommended for the following reasons:

- Performance becomes independent of voltage and temperature
- Code is more universal and portable
- It prevents maintenance side effects when code is maintained outside the original development context

## 7.5.2 Maintenance mode

The idea of a maintenance mode is firmware will probably reconfigure the device or perform an EEPROM operation, and it is allowable to constrain the operations of firmware to avoid polling. This includes powering off all rails.

This is an alternative to In System Programming (In Flight Update), which modifies EEPROM while the rails are operational, and relies on polling.

## 7.5.3 What is the generic time behavior of PSM devices?

Managers and Controllers have a different internal architecture, therefore their behaviors are not the same. This FAQ will try to give a universal guideline, but to do so implies an overall constraint tighter than for an individual family. The operating space will also be confined to a subset of commands, applicable to a maintenance mode.

### 7.5.3.1 Managers

Managers have a PMBus interface that accepts commands at any time except when the device is busy with an operation related to EEPROM. EEPROM activity fall into three categories:

1. EEPROM is being written
2. EEPROM is being read
3. A fault log is in progress

Basically, if the EEPROM is not being used, the devices can accept commands at the full PMBus clock rate, assuming there are no timing specification violations.

From a firmware perspective, category 3 is the most problematic, because a fault log can occur at any time. Category 1/2 are under user control, or in the case of 2, firmware knows the device's state.

Internal to the device, this behavior results from a processor without interrupts and the ability to multiply/divide faster than the PMBus can send commands.

### 7.5.3.2 Controllers

Controllers have a different behavior. Commands given to a controller are put in queues for processing. Some commands are slow. Some commands run in the background.

Slow operations fall into the following categories:

1. EEPROM is being written
2. EEPROM is being read
3. A fault log is in progress
4. An L11 or L16 register was written, triggering an internal multiply/divide
5. A range bit was written, triggering an internal multiply/divide
6. The temperature has changed, triggering an internal multiply/divide

Internal to the device, this behavior results from a processor that accepts interrupts, and the inability to multiply/divide faster than the PMBus can send commands.

## 7.5.4 Constraints that lead to deterministic behavior

It is possible to reduce the both the commands and the operating state of the devices and make the behavior of the devices more deterministic.

Here are some constraints for each device type:

#### 7.5.4.1 Managers

1. Force a fault log
2. Do not use any command related to EEPROM such as STORE/RESTORE\_USER\_ALL or MFR\_EE\_DATA/ERASE

#### 7.5.4.2 Controllers

1. Do not write to any L11 or L16 register
2. Do not change any range bit
3. Do not use any command related to EEPROM such as STORE/RESTORE\_USER\_ALL or MFR\_EE\_DATA/ERASE
4. Set operation to OFF

The controller items 1-3 almost work by themselves because temperature change is slow, thus probability of a temperature calculation is low. However, there is no way for firmware to know if the measured temperature is at a boundary condition, meaning a small change will trigger a calculation. Therefore, the low probability is low, not zero. Turning off the outputs prevents the temperature calculation (need to confirm with design), hence item 4.

A fault log operation is typically not an issue as long as firmware avoids EEPROM operations, as it just processes in the background.

#### 7.5.4.3 Universal Constraints (Manager/Controller)

1. Command off all rails
2. Force a fault log (wait for completion with long wait)
3. Do not write L11 or L16 registers
4. Do not change ranges
5. Do not use any EEPROM operations
6. Delay 1ms between all commands

Under these constraints firmware will have a very very low probability of a busy fault.

### 7.5.5 Memory operations under universal constraints

Given the above constraints, there are scenarios that use EEPROM operations during maintenance mode. Therefore, some memory delays are required for:

- STORE/RESTORE\_USER\_ALL
- MFR\_EE\_DATA/ERASE

A STORE\_USER\_ALL is the slowest operation of the device. The firmware can use a 4.1 second delay after the STORE\_USER\_ALL, which is the worst case value from the data sheet.

RESTORE\_USER\_ALL is basically a reset, and a 4.1 second wait will also be safe.

MFR\_EE\_DATA is an alternate way to write EEPROM, one word at a time. A delay of 4.1s/NumBytes can be used between each write.

MFR\_EE\_ERASE is faster than a STORE\_USER\_ALL, but a 4.1 seconds will cover all devices when the specification table is used to guarantee the delay required. The managers have additional data in the data sheet outside the specification table that can be used to reduce the delay.

## 7.5.6 Speeding things up

Delays can be reduced in two places, with some risk. The 1ms between commands can be reduced toward 200us. Our advice is resist this temptation, but if you ignore the advice, do more testing, including all operating conditions, especially voltage and temperature corners. Voltage means all voltages that power the controller and manager such as VIN or VDD3.3.

4.1 second delay on EEPROM operations can be reduced toward 500ms. The actual time required to program is related to the VDD3.3 voltage and temperature. The worst case is low VDD and high temperature. If the VDD3.3 is driven by the device and not from an external supply, you can assume it is within specification, and reducing the delay is less risky. Under this condition (not driving VDD3.3 externally) a 1 second delay is conservative. However, it would be advisable to test the system at full operating temperature to validate operation.

If you have to be more aggressive reducing delays, you can contact the factory to get more data. However, the better alternative is to use polling. A compromise is to use 1ms delays on normal transactions, and use polling for EEPROM operations, because the EEPROM delays are much more subject to operating conditions than non-EEPROM commands.

## 7.5.7 Side effects

The main side effects of these constraints are:

- Rails are turned off
- Fault logs are written

This means that when firmware takes the system into this constrained maintenance mode, the fault log will have to be cleared, and then the rails turned on. This ensures that any problems with power on can cause a fault log.

## 7.5.8 Special Cases

### 7.5.8.1 Storing with OPERATION on

There is no way to issue STORE\_USER\_ALL with OPERATION set to on without leaving maintenance mode. However, in this special case, the firmware should:

1. Set OPERATION to on
2. Wait for rails to power on so that none of the devices are busy
3. STORE\_USER\_ALL
4. Clear the fault log

There is some risk that the device will be busy at step 3 and the command will fail. Therefore, the firmware should watch for ALERTB or read status registers to ensure the store operation did not fail.

## 7.5.9 Other constraints (non maintenance mode)

There are other constraints that can be used in a system. For example, fault logs could be disabled, and commands cause calculations avoided, such that more common operations are safe. Or firmware can be constrained to telemetry, which as read only commands, are not affected by fault logs or calculations. These are not fully characterized in this FAQ entry, and you can ask an FAE for help.

### 7.5.10 Alternatives

The main alternative is to use In System Programming (In Flight Update). This process can safely update EEPROM while a system is online and operational. The polling required for this built into the definition of the In System Programming and the sample code.

### 7.5.11 Summary

Maintenance mode is typically used modify the EEPROM contents of a PSM device.

Best practice is to use polling. However, a system can be placed in a maintenance mode that is constrained enough to avoid polling, as long as the delays are acceptable, and the system is tested to ensure it does not fail over the operating range.

Placing the system in maintenance mode effectively takes the system offline, meaning supplies are off. This is different than In System Programming, which modifies EEPROM during normal operation.

## 7.6 How do I know when LTC388X STORE\_USER\_ALL is complete?

### 7.6.1 How do I know when LTC388X STORE\_USER\_ALL is complete?

When a STORE\_USER\_ALL command is issued to a LTC388X, the device will become busy for a period of time, but it will become un-busy before the command completes. This allows reading telemetry and status while the operation is in progress. To know when the operation completes, the device must be polled in three stages:

1. Waiting for the device to ACK
2. Waiting for the device to no be BUSY
3. Waiting for the operation to complete

The Linduino code has built in commands for these, but to help understand, we give here code written at the SMBus level to aid in understanding.

#### 7.6.1.1 Wait For ACK

##### Wait For ACK

```

1  uint8_t wait_for_ack(uint8_t address, uint8_t command)
2  {
3      uint8_t data;
4      uint16_t timeout = 8192;
5      while (timeout-- > 0)
6      {
7          if (0 == smbus->i2cbus()->readByteData(address, command, &data))
8              return 1;
9      }
10     return 0;
11 }
```

This routine performs a low level I2C command to read byte data. It is passed the address of the device, and the command to read from. It is typically called with the PAGE command for two reasons:

- The PAGE command is never BUSY
- It will not change the state of any device

This loops with a timeout and then returns 1 on success.

### 7.6.1.2 Wait for Not BUSY

#### Wait for Not Busy

```

1  uint8_t wait_for_not_busy(uint8_t address)
2  {
3      uint8_t mfr_common;
4      uint16_t timeout = 8192;
5
6      while (timeout-- > 0)
7      {
8          mfr_common = smbus->readByte(address, MFR_COMMON);
9          // If too busy to answer, poll again.
10         if (mfr_common == 0xFF)
11             continue;
12         if ((mfr_common & (NOT_BUSY | NOT_PENDING | NOT_TRANS)) == (NOT_BUSY | NOT_PENDING |
13             NOT_TRANS))
14             return 1;
15     }
16     return 0;
17 }
```

This routine loops on the MFR\_COMMON register, with a timeout, and also returns a 1 on success. To be certain the device is not doing anything internally, it checks the following:

- NOT\_BUSY - The device is not doing anything
- NOT\_PENDING - There are no commands in the device's queue
- NOT\_TRANS - The rails are not changing voltage

This is a very conservative form of not BUSY.

Notice the check for 0xFF. This is done because the device could be so busy it can't even answer the query.

#### Defines

```

1  #define USE_PEND      1
2
3  #define NOT_BUSY      1 << 6
4  #define NOT_TRANS     1 << 4
5  #ifdef USE_PEND
6  #define NOT_PENDING    1 << 5
7  #else
8  #define NOT_PENDING    0
9  #endif
```



This three bits are defined as above. Note the use of USE\_PEND. Not all devices support PEND, such as Managers like LTC297X, so this is a way to disable it.

### 7.6.1.3 Wait For EEPROM/NVM Complete

#### Wait For NVM Complete

```

1  #define MFR_EEPROM_STATUS      0xF1
2
3  uint8_t wait_for_nvm_done(uint8_t address)
4  {
5      uint8_t mfr_eeeprom_status;
6      // A real application should timeout at 4.1 seconds.
7      uint16_t timeout = 8192;
8
9      while (timeout-- > 0)
10     {
11         wait_for_ack(address, 0x00);
12         wait_for_not_busy(address);
13         mfr_eeeprom_status = smbus->readByte(address, MFR_EEPROM_STATUS);
14         if (mfr_eeeprom_status == 0xFF)
15             continue;
16         if ((mfr_eeeprom_status & 0xC0) == 0)
17             return 1;
18     }
19     return 0;
20 }
```

This routine contains wait for ACK and not BUSY. These are in a loop so that if a device becomes busy, it always follows the same pattern after it becomes busy. This could happen if there was another master, thread, etc. In simple cases the wait for ACKJ and not BUSY can be moved outside the loop.

The loop does a read from MFR\_EEPROM\_STATUS, and unpublished Read Byte command. It represents the internal state machine of the device. When the state is 0xC0, the STORE\_USER\_ALL is complete.

### 7.6.1.4 Putting It All Together

To know when STORE\_USER\_ALL completes, use the wait\_for\_nvm\_done() routine above. There is an equivalent function in the Linduino API like below. It can be called like: pmbus->waitForNvmDone(address).

```

1  uint8_t LT_PMBus::waitForNvmDone(uint8_t address)
2  {
3      uint16_t timeout = 8192;
4      uint8_t mfr_eeeprom_status;
5
6      while (timeout-- > 0)
7      {
8          smbus_->waitForAck(address, 0x00);
9          mfr_eeeprom_status = smbus_->readByte(address, MFR_EEPROM_STATUS);
10         if (mfr_eeeprom_status == 0xFF)
11             continue;
12         if ((mfr_eeeprom_status & 0xC0) == 0)
13             return SUCCESS;
14     }
15     return FAILURE;
16 }

```

## 7.7 Linduino FAQ

This Subsection contains Frequently asked questions about the Linduino platform

### 7.7.1 Can I Read and Decode Fault Logs in Firmware?

#### 7.7.1.1 Background

All PSM devices can store fault information in a Fault Log (EEPROM) for later analysis. Typically this data is extracted at the factory, but it can also be extracted by firmware, and even decoded in the firmware. When it is extracted by firmware, the Fault Log can then be reset and made ready for a new fault event to trigger a new log.

#### 7.7.1.2 Methods of Extraction

There are two methods of extraction:

- Read data with a block read command
- Read data with multiple read word commands

The simplest and primary method is to read the data with a command like MFR\_FAULT\_LOG (block read). Reading block data requires SMBus support for a command that reads up to 255 bytes. Many systems do not support this command in their firmware. In this case, the data can be read word by word using the MFR\_EE\_DATA command. Reading data with this command requires unlocking the EEPROM and reading the whole contents of EEPROM, locking the EEPROM, then extracting the data by knowing the offset of the data in the large block.

Rather than write the code yourself, it is better to download the Linduino Sketchbook from [www.linear.com/linduino](http://www.linear.com/linduino)<sup>33</sup> and look in the directory libraries/LTPSM\_PartFaultLogs. This directory has code for each device type that can easily be ported to your platform. Many of these files contain a #define RAW\_EEPROM. This define causes the compiler to use the MFR\_EE\_DATA mode of reading the data. Removing the define causes the compiler to use the block read command.

---

<sup>33</sup> <http://www.linear.com/linduino>

### 7.7.1.3 Decoding

Decoding the data is mostly mapping pointers from the block of data to C structs. In a few cases, such as managers, a few more tricks need to be played to put the events in the log in order.

The Linduino code mentioned above handles the mapping to C structs so that you don't have to figure it out. Once the pointers are in place, C code can navigate the data as a hierarchy of nested structs.

### 7.7.1.4 Storing Data

If the fault log will be stored off the device, it can either be stored in the original block form, or using the nest structs, stored in any desired format.

### 7.7.1.5 Printing Data

The Linduino code provides a simple pretty printer of the data. It depends on Linduino printing API, so it needs to be adapted to the your system.

### 7.7.1.6 Cautions

The EEPROM can wear out if written more than 10,000 times. The coder should put a failsafe mechanism in the code so that if there is a recurring fault, it does not reset the log 10,000 times in a fast loop. Resetting the fault log does an erase on the EEPROM log area, and the next fault log event writes data.

### 7.7.1.7 Summary

All PSM devices store fault logs, and Linduino has code to read and decode them to C structs. Because the block of data is unique per device, and because extracting the data word by word is not documented in the data sheet, the Linduino code should be ported, or used as a model.

## 7.7.2 How Do I Connect a Linduino to a PSM DC?

- Put wires in the Linduino socket and clip to the DC
- Use a DC2294A

Obviously using wires is not the best way. The DC2294A has a ribbon connector that will connect to all PSM DC boards, like the DC1962 (Power Stick). This board is not yet released, so you have to get one from Mike Jones for now.

## 7.7.3 Is the Code Open Source?

The PSM code has the same license as all Linduino code.

## 7.7.4 What about the Galileo Board?

There is an unreleased LT\_PMBus library available by request.

### 7.7.5 What Arduino Boards Work?

The LT\_I2C library is tied to the Atmel processors. The LT\_PMBus library is tied to a modified Wire library. This makes it easier to port to other processors. However, the modified Wire library supports block commands, so its API has 16 bit size variables rather than 8 size variables. To use another Arduino platform, you may have to slightly modify their Wire library.

### 7.7.6 What If I Want to Use Other I2C DC At The Same Time?

The ADC/DAC sketches and other products use library LT\_I2C. The PSM sketches use library LT\_PMBUS. The reason is to remove impact of library maintenance, and to address special needs of PMBus such as block commands and memory management.

To use devices that have sketches dependent on LT\_I2C, requires copying its code and adapting it to LT\_PMBUS. Typically this means dealing with byte order.

### 7.7.7 What Sketch's are Available?

LTC is committed to having a minimum of one sketch per DC. Please find these on the product landing page of each device.

For example, for LTC2977 code, go to <http://www.linear.com/product/LTC2977> and select the 'Code' link on the left side of the page.

## 7.8 MFR\_SPECIAL\_ID Handling

### 7.8.1 ID Format

Power System Management (PSM) devices in the LTC388X and LTC297X family have a MFR\_SPECIAL\_ID (0xE7) command that returns a unique word that identifies the device type. The nibbles are organized as a device/rev pair as follows:

DDD R

where DDD represents a 3 nibble device type, and R represents single nibble revision.

### 7.8.2 Coding Practice

All code that reads this register to identify the type of device should mask the R nibble, which means ignore it. This ensures that code will not break when the revision changes.

### 7.8.3 Coding Example

The Linduino example code at [www.linear.com/linduino](http://www.linear.com/linduino)<sup>34</sup> has examples of masking the R nibble.

Here is an example from LT\_PMBusDeviceLTC2977.h

---

<sup>34</sup> <http://www.linear.com/lindion>

**detect**

```

static LT_PMBusDevice *detect(LT_PMBus *pmbus, uint8_t address)
{
    uint16_t id;
    LT_PMBusDeviceLTC2977 *device;
    id = pmbus->readMfrSpecialId(address);
    if ((id & 0xFFF0) == 0x0130)
    {
        device = new LT_PMBusDeviceLTC2977(pmbus, address);
        ...
        return device;
    }
    else
        return NULL;
}

```

By anding id with 0xFFF0 and comparing to 0x0130, the rightmost nibble is ignored.

## 7.8.4 How can firmware identify a LTC2978 without MFR\_SPECIAL\_ID?

### 7.8.4.1 How can firmware identify a LTC2978 without MFR\_SPECIAL\_ID?

There is a quick way to distinguish a LTC2978 from all other LTC297X and LTC388X and modules with these devices in them. The trick is to use the MFR\_COMMON register. The LTC2978 has different reserved values that are unique from the other devices. The LTC2978 reserved bits are value 0. All other devices are value 1.

It is recommended to use Bit 2, because this bit is reserved for all devices.

To detect, read the MFR\_COMMON register, AND the value with 0x04, and check if the result is greater than 0.

LTC2978 MFR\_COMMON

#### MFR\_COMMON Data Contents

| BIT(S) | SYMBOL                   | OPERATION   |
|--------|--------------------------|---|
| b[7:2] | Reserved                 | Read only, always returns 0s  |
| b[1]   | Mfr_common_share_clk     | Returns status of share-clock pin<br>1: Share-clock pin is being held low<br>0: Share-clock pin is active |
| b[0]   | Mfr_common_write_protect | Returns status of write-protect pin<br>1: Write-protect pin is high<br>0: Write-protect pin is low        |

## LTC2977 MFR\_COMMON

**MFR\_COMMON Data Contents**

| <b>BIT(S)</b> | <b>SYMBOL</b>            | <b>OPERATION</b>  |
|---------------|--------------------------|---|
| b[7]          | Mfr_common_alertb        | Returns alert status.<br>1: ALERTB is de-asserted high.<br>0: ALERTB is asserted low.   |
| b[6]          | Mfr_common_busyb         | Returns device busy status.<br>1: The device is available to process PMBus commands.<br>0: The device is busy and will NACK PMBus commands. |
| b[5:2]        | Reserved                 | Read only, always returns 1s  |
| b[1]          | Mfr_common_share_clk     | Returns status of share-clock pin<br>1: Share-clock pin is being held low<br>0: Share-clock pin is active                                   |
| b[0]          | Mfr_common_write_protect | Returns status of write-protect pin<br>1: Write-protect pin is high<br>0: Write-protect pin is low  |

## LTC3887 MFR\_COMMON

| <b>BIT</b> | <b>MEANING</b>             |
|------------|----------------------------|
| 7          | CHIP NOT DRIVING ALERT LOW |
| 6          | CHIP NOT BUSY              |
| 5          | CALCULATIONS NOT PENDING   |
| 4          | OUTPUT NOT IN TRANSITION   |
| 3          | NVM Initialized            |
| 2          | Reserved                   |
| 1          | SHARE_CLK Timeout          |
| 0          | WP Pin Status              |

## 7.9 What Example Firmware is Available from LTC?

There are two sets of firmware:

1. Linduino
2. Linux Drivers

### 7.9.1 Linduino

The Linduino is comprised of a layered library and sketches. The LT\_PMBUS library has the following layers:

- PMBus
- SMBus
- I2C

- Wire

There is also support for Group Protocol and In Flight Update. Sketches are available for most PSM devices.

## 7.9.2 Linux Drivers

Linux drivers are implemented in user mode based on `/dev/i2c`. A Wandboard and Yocto Linux was used to implement the code. The example code does not include block commands or ALERTB. The code should work on any Linux platform with a supporting i2c driver.

However, support for i2c is sparse, and the level of support is not uniform. If you work with a different platform, you may have to do some driver work.

## 7.10 What is In Flight Update and How do I use it?

### 7.10.1 In Flight Update Code

In Flight Update code is available in two forms:

1. Linduino
2. Linux

Both versions are written in C/C++, but the underlying PMBus drivers are different.

The purpose of using C++ was to allow switching on/off PEC on the fly without a lot of conditional logic.

### 7.10.2 How In Flight Update Works

In Flight Update applies an .ISP file that is exported from LTpowerPlay to a final design. LTpowerPlay creates an Intel Hex file with data records. This file is processed to create C Structs that are executed by an engine that applies PMBus commands.

The In Flight Update software is similar to LTC/Arrow programming. They both use the same file and record formats. However, OEM and .ISP files are different recipes. OEM recipes are for single devices in a programming socket, and .ISP files are for working boards with multiple devices.

### 7.10.3 How Many Devices Does it Program at a Time?

You can generate an In System Programming (.ISP) file for a whole board, subsets of a board, or single devices. LTpowerPlay exports the .ISP file based on what is in the project tree.

### 7.10.4 How Do You Port In Flight Update?

Typically the following things have to be addressed:

1. Endianness
2. Structure packing
3. 32 bit pointer casting
4. PMBus API

The exist example code from eCos and Linduino determine the endianness of the example code. You may have to change it. Structures are packed because the Intel Hex records are just binary data that is mapped into packed structs with pointers. This means the memory layout is in the data, and packing structures matches this.

The pointer casting is only an issue if the code is ported to anything other than a 32 bit address processor.

The PMBus API can be managed in multiple ways:

1. Implement the PMBus.h for your platform
2. Port the PMBus layers to your platform
3. Rewrite the PMBus calls

### 7.10.5 How Do You Debug In Flight Update?

There are #defines to turn on print statements. You can also spy on PMBus traffic with a Beagle.

### 7.10.6 What is the .ISP File Format?

See [http://www.ltpowerplay.com/in\\_circuit\\_programming/](http://www.ltpowerplay.com/in_circuit_programming/)

### 7.10.7 LTC3882 Pre-Release Si Compatibility

The LTC3882 released two pre-release versions of silicon to the field. These versions had very different register maps from final Si. The packing codes for these versions were 0x0400 and 0x0500. Device programming (including In Flight Update) is not supported for these pre-release versions of the product. LTpowerPlay generates .ISP and OEM files for the final silicon which has code **0x420x**. If you are trying to perform In Flight Update on pre-release silicon on a customer board, it will fail on the packing code check. You must move to production silicon for In Flight Update to work properly.

## 7.11 What options do I have to communicate with with PSM devices via PMBus?

There are three methods to communication with PSM devices provided by Analog Devices:

1. LTpowerPlay Scripting
2. Linduino
3. Linux

### 7.11.1 LTpowerPlay Scripting

Scripting is a way to write files with commands, and have LTPP execute it, from the GUI or the command line, and produce an output. When calling from the command line, a script file is passed and the output is printed. The output can also be redirected to a file, or if the command line is called via an application, it can capture the output. The output is human readable, but it can also be parsed by application code.

Furthermore, scripts can be created from within LTPP by enables command capture, operating registers in LTPP, then closing the capture. This allows creating of scripts without having to learn the script language. This is very helpful when getting started.

Scripting is a good path when one needs to communicate with the DC1613 from a PC for evaluation or manufacturing tools.



In addition to scripting, LTPP supports general debugging via GUI, and programming via the command line.

Programming works like scripting in that it can be done from the GUI or command line. Because one can program from the command line, it can be used to program a board during manufacturing.

Scripting help can be found in this FAQ.

### 7.11.2 Linduino

The Linduino Sketchbook provided by Analog Devices includes a SMBus/PMBus API along with multiple sketches demonstrating how to communicate with PSM devices. This is a very good option when learning how to write microcontroller code, because it is very easy to get hardware and run examples. There is lots of Arduino help on the internet to help.

Linduino information can be found [here](https://www.analog.com/en/design-center/evaluation-hardware-and-software/linduino.html)<sup>35</sup>.

### 7.11.3 Linux

The linux code is similar to the Linduino code, but does not come with as many examples. It uses `/dev/i2c#` to communicate with PSM devices. It is more suitable for someone already working with linux and understands the basics of coding on the linux platform.

To obtain the linux code, contact your local FAE.

---

<sup>35</sup> <https://www.analog.com/en/design-center/evaluation-hardware-and-software/linduino.html>

## 8 What's This Fault?

This FAQ Section provides detailed help for faults/warnings/status that you may encounter using various parts with LTpowerPlay. This is the staging grounds for content that will ultimately be deployed with LTpowerPlay in the "What's this Fault? Tool". Some of this content may be in draft form, so please bear that in mind when reading it.

The names of the individual pages in this section are titled to match the items that are shown in the FAULT\_WARN\_LIST pseudo-register in LTpowerPlay. This pseudo-register is visible in the Telemetry/Status panel, and summarizes the faults/warnings/status for the selected channel. A new tool called the "What's this Fault? Tool" lists each of the statuses shown in the FAULT\_WARN\_LIST register and allows you to get detailed help on the selected status item of interest. The content from the subpages of this section are displayed when you select an item in the "What's this Fault? Tool".

Note that this content is under active development, so you may encounter scenarios where the selected status item does not yet have detailed help. In this case, LTpowerPlay will tell you that no content is available and will direct you to the part's datasheet for further information. If you have encountered this and the content is not located in this section, and you still need help, please contact your local LT Field Sales representative for further help.

What's This Fault? Topics

### 8.1 CML\_INVALID\_COMMAND

The device a command code that is unsupported via I2C

#### 8.1.1 Possible Causes:

- Attempted to write an unsupported register using I2C

#### 8.1.2 Scope Debug

- Trigger a scope on the falling edge of ALERTB while probing SCL/SDA
- Go Offline and then back Online in LTpowerPlay (forces a refresh of GUI cache values)
- Click Write All in LTpowerPlay
- Observe the command that was written to the device immediately preceding the falling ALERTB signal.

### 8.2 CML\_INVALID\_DATA

#### 8.2.1 The device received improper data for a supported command.

##### 8.2.1.1 Possible Causes:

- Attempted to write an unsupported value to a register using I2C
- A configuration stored in the EEPROM has an invalid combination of configuration values (i.e. VOUT higher than range allows)

### 8.2.1.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Read the configuration into LTpowerPlay and click the "DRC" button in the toolbar to check for problems
- Power Cycle the device and see if the problem returns,
  - If so, the unsupported data is programmed into NVM
  - If not, the unsupported data is programmed in to RAM only
- Further Isolate which register(s) are causing the problem
  - Issue a CLEAR\_FAULTS
  - Select registers one at a time in the 'All Global' and 'All Paged' (be sure to go through each page), and write the individual register (F12)
    - Look for the CML\_INVALID\_DATA fault to re-appear, indicating the offending command
- Debug with Scope
  - Trigger a scope on the falling edge of ALERTB while probing SCL/SDA
  - Go Offline and then back Online in LTpowerPlay (forces a refresh of GUI cache values)
  - Click Write All in LTpowerPlay
  - Observe the command that was written to the device immediately preceding the falling ALERTB signal.

## 8.3 CML\_OTHER\_COM\_FAULT

### 8.3.1 The device detected traffic on the bus that does not match expected protocols.

#### 8.3.1.1 Possible Causes:

- Improper SmBus read
  - A valid read from a PmBus device requires first sending a start, then address+write, then the command code, and then a repeated start (without a stop), before reading bytes from the slave
  - A common error for people familiar with I2C is to issue a stop, and then a new start with the address+read bit. This is invalid and PmBus devices are required to pull ALERTB low and set this fault to signal to the host that it is communicating improperly
- Poor signal integrity on the bus
- Other unexpected protocol.
- A temporary bus disruption (perhaps a device powering up that glitches or holds one of the lines low for some period of time)
- Master wrote or read too few bits. I2C is byte oriented. If the slave sees a stop condition in the middle of a byte (before 8-bits+ACK), it will set this error bit.
- Master wrote too few bytes. If the master sends a stop condition after writing too few bytes (i.e. after sending 1 of 2 required bytes), the slave will set this error bit.
- On some devices, if the device sees an empty start/stop sequence will trigger this fault.
  - Clock remains high while data goes low (start) then high (stop)
  - Some devices (like DC1613A, FPGAs or other logic) may issue such sequences from time to time
  - For devices that flag this fault, these devices typically also support SMB\_ALERT\_MASK. If you are unable to avoid creating the fault, you can mask this fault from pulling ALERTB low by using the SMB\_ALERT\_MASK feature.
- A device has just come out of reset or busy state (fault log, reset, restore, etc) and when it is first available, sees a partial transaction which looks like one of the above conditions or illegal protocol.

NOTE: Marginal timing on the bus (too slow rise/fall times, or violation of setup time specs) may produce scenarios that to the slave look like illegal protocol. It is recommended that you verify the signal integrity of the I2C lines in this scenario.

### 8.3.1.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Decrease the bus speed (View > Preferences) and see if the problem goes away
- Clear the fault and see if it returns / is recurrent
- Disable any other masters that may be conflicting with LTpowerPlay
  - Only one master should be communicating with the system at a time. If you wish, you can disable LTpowerPlay by 'going offline'
  - If you believe that there are no other masters, but want to double check, a quick way to do this is to configure the device in question to 'require PEC' (MFR\_CONFIG\_ALL). Then take LTpowerPlay offline. If the ALERTB pin is pulled low, connect LTpowerPlay again and see if there is a CML\_PEC error. If so, this probably indicates that there is another master on the bus

### 8.3.1.3 Other Debugging Tips

There are a number of scenarios which can create a CML\_OTHER\_COM\_FAULT. If you have an oscilloscope, perhaps the fastest way to insight is to:

- Trigger the scope on the falling edge of the ALERTB signal, and
- Look at SCL/SDA signals **at the pins of the IC**

The ALERTB signal will be pulled low by the IC at the time of the fault, indicating the time of the fault. Look at SCL/SDA immediately preceding this for clues. If possible, use a scope (preferred) with I2C Decode (or if not available alternately an I2C sniffer) to inspect the protocol leading up to the fault

## 8.4 CML\_PEC\_FAILED

The device received an additional byte beyond the required number of bytes, and the additional byte failed to match the correct PEC CRC value for the byte stream.

### 8.4.1 Possible Causes:

- Wrote too many bytes (i.e. 16 bit write to an 8-bit register)
- Wrote incorrectly computed PEC byte
- Signal Integrity issues with I2C
- Perturbation of the I2C bus

### 8.4.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

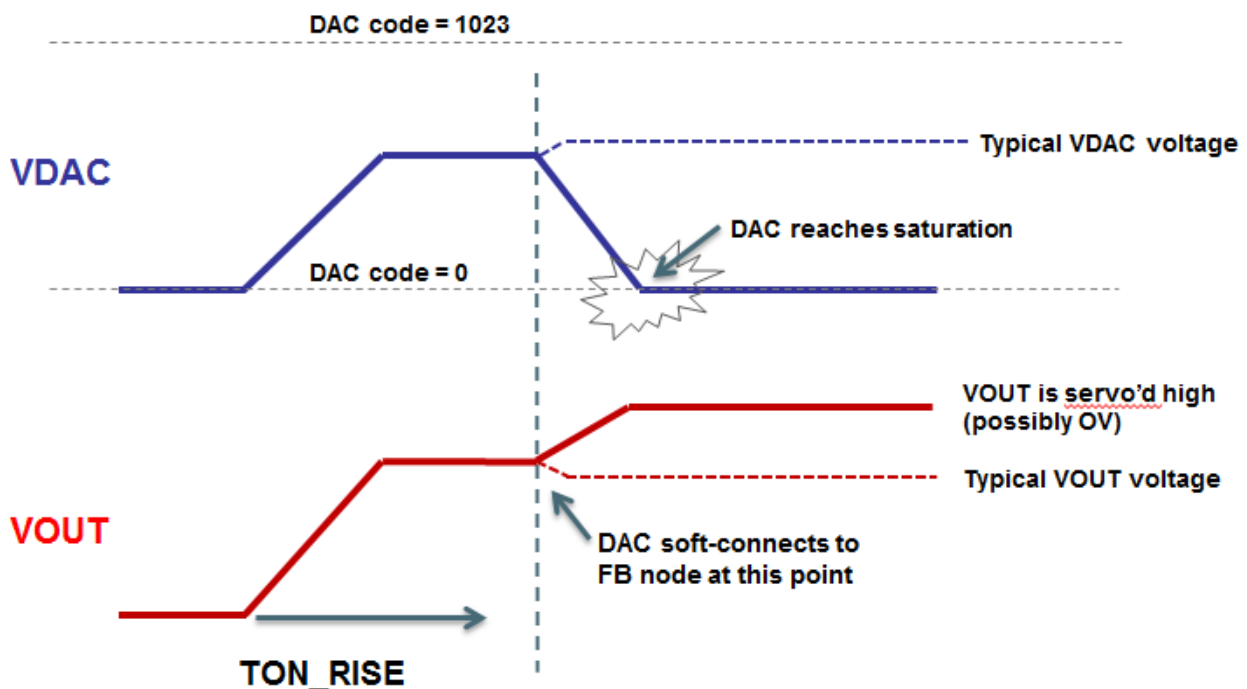
- Lower the bus speed (View > Preferences) and see if the problem goes away
- Clear the fault to see if it returns / is recurrent
- Disable other masters on the bus that may be conflicting with LTpowerPlay
  - Be sure only one master is talking at the same time. Optionally you can take LTpowerPlay 'offline'

### 8.4.3 Scope Debug

- Trigger the scope on the falling edge of the ALERTB signal, and
- Look at SCL/SDA signals preceding the ALERT
- This will tell you the offending command/data sequence and may illuminate other problems on the bus (contention, etc)

## 8.5 DAC\_SATURATED

A DAC\_SATURATED warning was detected because the DAC on this channel reached a maximum or minimum DAC value, and the VDAC pin has reached its limit. The channel may also have faulted off due to an OV or UV.



### 8.5.1 Possible Causes:

- Floating voltage too far away from VOUT\_COMMAND voltage for servo's compliance range
- DAC servo resistor value too high
- DAC servo resistor missing or cracked
- Regulator feedback resistor value(s) incorrect or assembly issue
- OPERATION register in Margin High or Margin Low state and DAC resistor value is too high

### 8.5.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Change DAC mode to disconnected and re-start the channel (if necessary)

- Observe READ\_VOUT floating voltage and compare this value with VOUT\_COMMAND setting. (Note: VOUT\_COMMAND is the target value used by the servo loop)
- Decrease DAC resistor value if the VDAC voltage does not have enough range to servo the output to the VOUT\_COMMAND voltage.
- Change the *dac\_gain* to higher setting if the VDAC voltage needs to be greater. If the regulator's feedback voltage is 0.6V or 0.8V, use the lower gain setting (1.38V full scale). For feedback voltages of 1.2V, use the high gain setting (2.65V full scale).
- Generally speaking, DAC resistor values of > 1M ohm will cause DAC\_SATURATED warnings.
- Change OPERATION register to 'On' (0x80), avoid Margin High and Margin Low states

### 8.5.3 Other Debugging Tips

There are a number of scenarios that can contribute to a DAC\_SATURATED fault. If you have an oscilloscope, perhaps the fastest way to insight is to:

- Probe the VDAC and VSENSE pins of the IC
- Under typical conditions, the VDAC voltage will be within 100mV of the feedback voltage on a properly working channel, unless channel in margining state

Note: The ALERTB pin might not be asserted if no warnings or faults are detected.

## 8.6 DISCHARGE\_FAULT

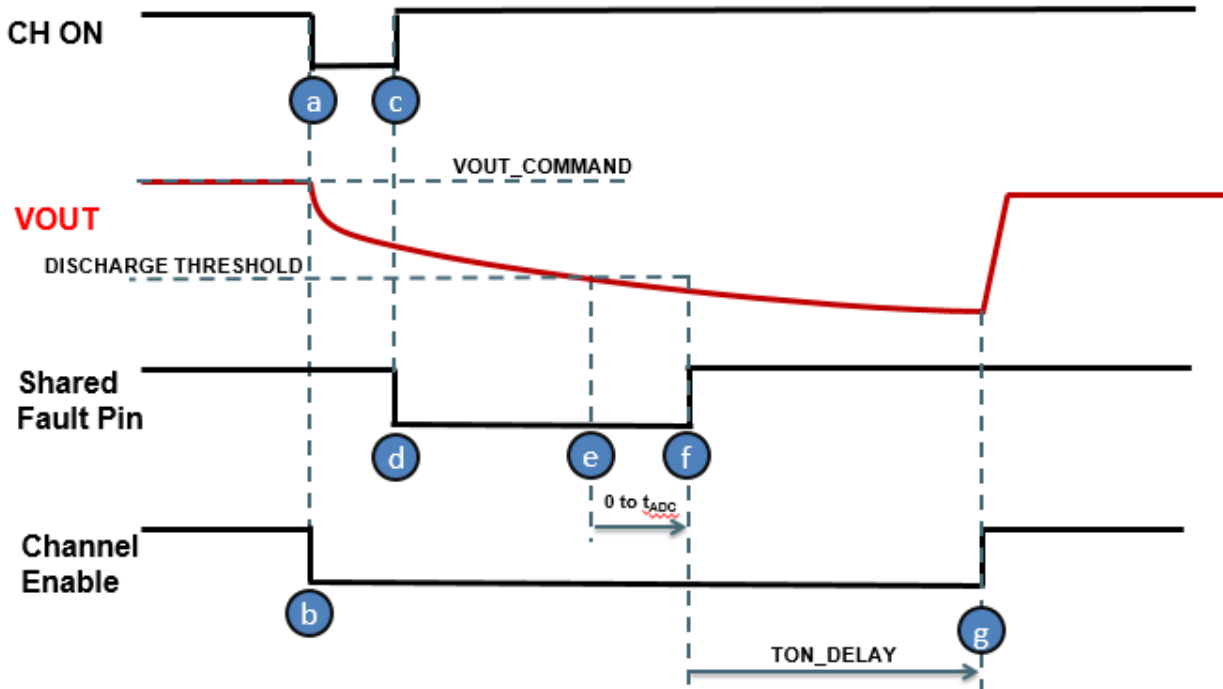
An DISCHARGE\_FAULT warning was detected because the channel was commanded to turn on before the output voltage the ADC measured the VSENSE voltage below its programmable discharge percentage ( $\text{MFR\_VOUT\_DISCHARGE\_THRESHOLD} * \text{VOUT\_COMMAND}$ ).

If Shared Fault propagation is enabled, this channel will also communicate this condition to other channels via holding the Shared Fault pin low until the ADC measures a sufficiently low voltage on the VSENSE pins.

### 8.6.1 Example Fault Scenario

The waveform below illustrates an example discharge fault and associated system behaviors:

## DISCHARGE\_FAULT



The example starts with the device in normal operating mode with no faults. The channel is commanded off and then on by the user. This can be via the CONTROL pin, OPERATION command, or VIN rising above the programmable VIN\_ON threshold, or combination thereof, depending on the configuration of the channel. For a clearer understanding of the selected channels on/off behavior, select 'Utilities > On / Off Configuration...' in the menu.

- a – The channel is commanded off. This may be because VIN dropped below VIN\_OFF, the CONTROL pin was de-asserted, or the PmBus OPERATION=Off command was issued
- b -- In response to the off condition, the channel's output enable is disabled, typically immediately, though options exist to honor the TOFF\_DELAY before disabling (soft off). When using this configuration, the output enable should be connected to a power supply so that de-asserting it disables the managed power supply.
- c – The channel is subsequently commanded back on via VIN/CONTROL/OPERATION or a combination. See the On / Off Configuration tool for a list of requirements for turn on.
- d -- At the time the channel is commanded on, the present ADC reading of the VSENSE pins is compared to the discharge threshold ( $VOUT\_COMMAND * MFR\_VOUT\_DISCHARGE\_THRESHOLD$ ). If the ADC reading (subject to ADC latency) is above this threshold, a DISCHARGE\_FAULT is declared. In the case (as in our example) that the channel is propagating faults to a shared fault pin, it will de-assert the shared fault pin signalling the DISCHARGE\_FAULT to other channels
- e -- In our example, at this point the VSENSE voltage (at the pins of the device) has decayed below the discharge threshold. However, the ADC is subject to t<sub>ADC</sub> latency (consult datasheet)
- f – When the ADC has measured the VSENSE voltage drop below the discharge threshold (0 to t<sub>ADC</sub> after the physical voltage has dropped below the threshold), the channel is ready to trigger a coordinate re-sequence of the system. It releases the shared fault pin to signal that it is ready, and initiate time=zero of the coordinated resequence.

- g – During the coordinated re-sequence, each channel, after waiting its programmed on delay (TON\_DELAY) will start ramping its voltage up
  - Controllers have the ability to ramp the voltage directly, and will ramp the output consistent with the programmed TON\_RISE setting.
  - Managers will assert the assert the VOEN pin to enable the managed supply at this time

This coordination feature insures that when power is toggled or when channels are toggled off/on, each channel is below its required discharge threshold before a coordinated re-sequence is attempted.

### 8.6.2 Possible Causes:

- Output Enable not connected to managed supply (board or assembly error)
- External device is over driving output enable (i.e. unprogrammed FPGA)
- Too short a MFR\_RESTART\_DELAY, or toggling off/on too fast
- Discharge threshold too low

### 8.6.3 Remedies and Workarounds

Here are some things you can try to narrow things down:

- Increase the MFR\_VOUT\_DISCHARGE\_THRESHOLD and see if the problem goes away ( 0.5 means 50% of VOUT\_COMMAND, 0.6 meant 60%, etc)
- Disable the MFR\_VOUT\_DISCHARGE\_THRESHOLD ( > 1 is disabled)
- Increase the MFR\_RESTART\_DELAY to stretch the length of disable pulses on the CONTROL pin

### 8.6.4 Other Debugging Tips

**8.6.5** Check for glitches (quick toggles) on the CONTROL pin. If you have an oscilloscope, perhaps the fastest way to insight is to:

- Probe the VSENSE pins of the IC
- Probe the CONTROL input
  - Alternately if CONTROL is not used, probe the output enable signal
- Trigger the scope on the falling edge of ALERTB

The ALERT pin will be pulled low at the time of the DISCHARGE\_FAULT. If the control pin is toggled too quickly, try the remedies and workarounds above to stretch the interval, or correct the source of the problem. If the output is not disabled when the output enable is de-asserted, check the connection between the output enable and the power supply. There may be a missing or failed connection (cracked resistor, etc).



## 8.7 FACTORY\_TRIM\_CRC\_FAULT

**8.7.1** The device has an internal inconsistency in its EEPROM. This is often an indication the power was removed during a STORE operation or the device was incorrectly programmed.

### 8.7.1.1 Possible Causes:

- Power was removed during a STORE\_USER\_ALL operation before the write to EEPROM completed
- The device was programmed with data incorrectly
- One or more bits in the EEPROM have changed from their initial value

### 8.7.1.2 Remedies and Workarounds

- Avoid removing power during a STORE\_USER\_ALL operation until the operation is complete
- For an improperly programmed device, re-program the device 'in system' using the programming utility

## 8.8 FAULT0\_IN

The channel was turned off or prevented from turning on at least once because a shared FAULTB pin was asserted low and the channel is configured to respond to the pin. The root cause item of interest is either another channel or external device in the system that asserted the pin low.

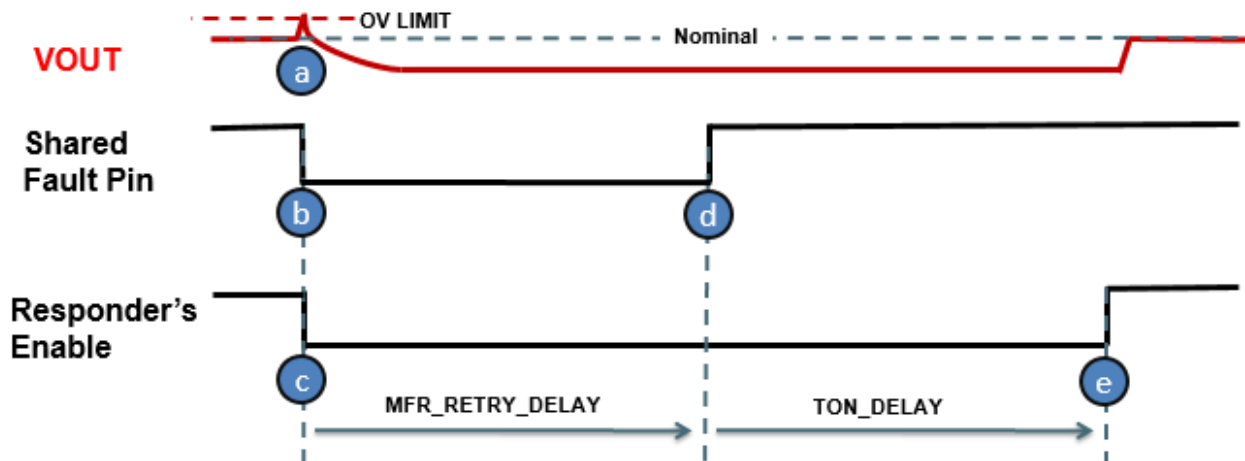
Shared Fault Pins are used as a bi-directional fault coordination bus for other PSM devices or external devices.

Some PSM devices name the shared fault pin "FAULTBX" while others name it "GPIOB". GPIOB pins can be used for other purposes as well, but are typically used for fault sharing. See the next section for a better understanding of how fault sharing behavior works.

FAULT0\_IN indicates a specific fault pin that the channel is responding to. For devices that have two FAULTB pins, this status indicates FAULTB0 pin as the source. For devices that support 4 FAULTB pins, FAULT0\_IN indicates the FAULTB00 pin for channels 0-3, and the FAULTB01 pin for channels 4-7 of the device.

### 8.8.1 Device Behavior in a Shared Fault Scenario

## SHARED FAULT RESPONSE



The example waveform above illustrates fault sharing behavior of the 'root offender' channel in conjunction with a 'responder channel'.

*NOTE: This is one specific example for the purpose of understanding. The following events occur in the example:*

- a -- A channel (called 'root offender') encounters a fault (over voltage in this example) with a non-ignore response setting, and disables its output (example VOUT\_OV\_FAULT\_RESPONSE= Deglitched Off, Infinite Retry)
- b -- When its output is disabled, the root offender, if configured to **propagate** to a shared fault pin (FAULTB OR GPIOB) will immediately pull low on the open drain shared fault pin.
- c -- A responder channel (configured to **respond** to the shared fault pin) will immediately stop power delivery (immediate off)
  - For PSM Controllers, power deliver is stopped by the controller itself, and the condition is reported as GPIOB\_ASSERTED\_LO
  - For PSM Managers, the output enable (VOEN) pin is de-asserted to disable the managed power supply, and the condition is reported as FAULTBX\_IN

*NOTE: The LTC2977, LTC2980, and LTM2987 may be optionally configured to honor off sequencing delays (TOFF\_DELAY) before de-asserting the VOEN pin upon fault.*

- d -- After the root offender has waited its programmable retry delay (MFR\_RETRY\_DELAY), if the fault is no longer present (as in this example), it releases the open drain shared fault pin.
  - This signals to all responder channels that a re-sequence should occur.
  - The rising edge of FAULTB sets time=zero for a coordinated re-sequence of the root offender and any responder channels.
- e -- During the coordinated re-sequence, a responder channel, after waiting its programmed on delay (TON\_DELAY) will start ramping its voltage up
  - Controllers have the ability to ramp the voltage directly, and will ramp the output consistent with the programmed TON\_RISE setting.

- Managers will assert the VOEN pin to enable the managed supply at this time

### 8.8.2 Possible Causes:

- Missing a pullup on the FAULTB pin
- Another chip or channel is pulling the FAULTB pin low. Check the rest of the system for faults:
  - If the responder channel remains off, the 'Why am I Off?' will provide a succinct list of faults in your system
- An External Device other than a LTC device may be holding FAULTB low.
  - Check the schematic for any other devices that may be connected to the GPIOB pin (an unprogrammed FPGA for instance)
- You may also scan the system for faults manually:
  - Select the FAULT\_WARN\_LIST pseudo-register in the Telemetry/Status window
  - Use the 'All Pages in System' view to scan the FAULT\_WARN\_LIST across the system to see all faults/warnings in the system in one place.
  - If there are any 'red' channels in the system tree, focus on these channels first.
  - Look for critical or important faults (OV, OC, OT, etc) on another chip that is propagating to the FAULTB pin

### 8.8.3 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Configure the channel to ignore the FAULTB pin in the "Fault Sharing" Section of the configuration. This will allow the channel to startup up regardless of the pin.
- To determine if the 'root cause' is a PSM device or external source (unprogrammed FPGA, etc) driving the pin low:
  - Disable propagation to FAULTB/GPIOB for all devices in the system
  - If the FAULTB pin remains low, this indicates an external device or circuit may be keeping the pin low.
- If the DRC check produces any warnings, fix these and see if the problem goes away.

### 8.8.4 Other Debugging Tips

If the channel is presently off, use the "Why am I Off? Tool". Otherwise, if you have an oscilloscope, do the following:

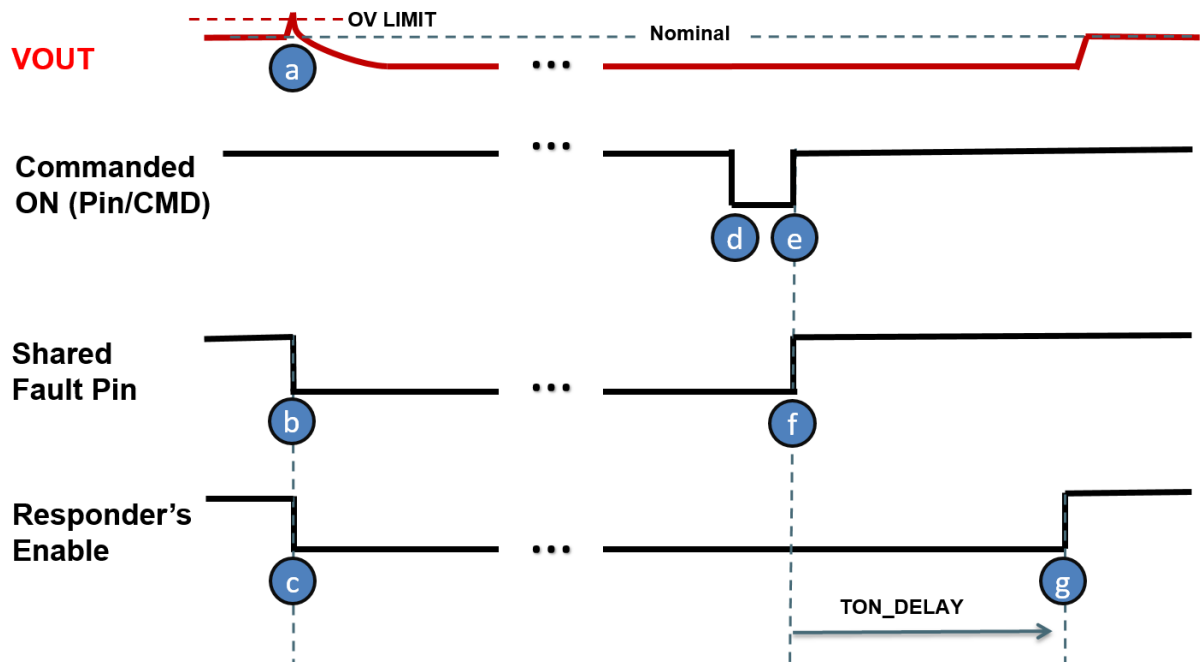
- Trigger the scope on the falling edge of the FAULTB pin and
- Look at other possible signals in the system that may be responsible driving FAULTB low
- Disable fault propagation for all devices. If the pin is still low, trace the schematic for any devices connected to the pin and insure they are not driving it low.

### 8.8.5 Latch Off Shared Fault Response

NOTE: The behavior of a shared fault pin is determined by the programmed fault response of the root offender. The example above shows a case where the root offender is configured to automatically retry after the fault occurs. In cases where the root offender's fault response is programmed to 'latch off' without retry, external user action (off then on) is required to re-sequence the system. Note that a CLEAR\_FAULTS command will only clear fault status bits, and will NOT have functional effects on the shared fault pins or enable states of supplies.

The example waveform below illustrates fault sharing behavior of the 'root offender' channel in conjunction with a 'responder channel' when the root offender is configured to 'latch off'.

## OV Shared Fault Response (Latch Off)



NOTE: This is one specific example for the purpose of understanding. The following events occur in the example:

- a -- A channel (called 'root offender') encounters a fault (over voltage in this example) with a 'latch off' response setting, and disables its output (example VOUT\_OV\_FAULT\_RESPONSE= Deglitched Off, NO Retry)
- b -- When its output is disabled, the root offender, if configured to **propagate** to a shared fault pin (FAULTB OR GPIOB) will immediately pull low on the open drain shared fault pin.
- c -- A responder channel (configured to **respond** to the shared fault pin) will immediately stop power delivery (immediate off)
  - For PSM Controllers, power deliver is stopped by the controller itself, and the condition is reported as GPIOB\_ASSERTED\_LO
  - For PSM Managers, the output enable (VOEN) pin is de-asserted to disable the managed power supply, and the condition is reported as FAULTBX\_IN

NOTE: The LTC2977, LTC2980, and LTM2987 may be optionally configured to honor off sequencing delays (TOFF\_DELAY) before de-asserting the VOEN pin upon fault.

NOTE that absent external action, the system will remain in this state. The root offender remains 'latched off', and it also holds its shared fault pin low to communicate to the rest of the system it should also remain off.

- d -- At some later time, an external action commands the root offender off. This could be due to one or more of the following conditions:
  - VIN drops below the programmed VIN\_OFF threshold
  - The CONTROL/RUN pin is toggled to the logical 'off' state
  - The PmBus OPERATION command (or ON\_OFF\_CONFIG) commands the channel off
- e -- An external action commands the root offender on again (off then on). The external On request could be due to one or more of the following conditions:
  - VIN rises above the programmed VIN\_ON threshold

- The CONTROL/RUN pin is toggled to the logical 'on' state
- The PmBus OPERATION command (or ON\_OFF\_CONFIG) commands the channel on
- f -- Upon seeing the external off then on request, the root offender releases the open drain shared fault pin.
  - This signals to all responder channels that a re-sequence should occur.
  - The rising edge of FAULTB sets time=zero for a coordinated re-sequence of the root offender and any responder channels.
- g -- During the coordinated re-sequence, a responder channel, after waiting its programmed on delay (TON\_DELAY) will start ramping its voltage up
  - Controllers have the ability to ramp the voltage directly, and will ramp the output consistent with the programmed TON\_RISE setting.
  - Managers will assert the assert the VOEN pin to enable the managed supply at this time

## 8.9 FAULT1\_IN

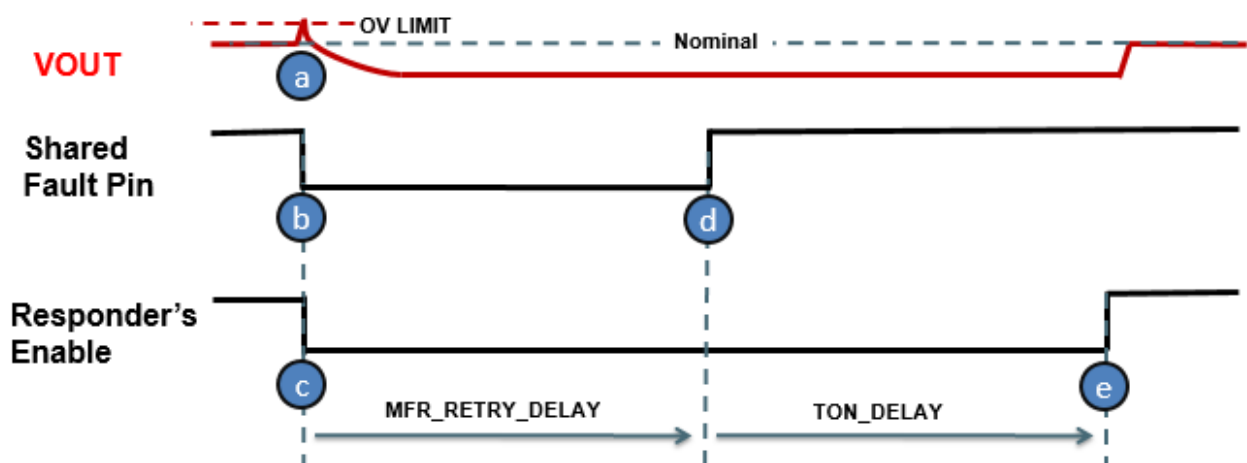
The channel was turned off or prevented from turning on at least once because a shared FAULTB pin was asserted low and the channel is configured to respond to the pin. The root cause item of interest either another channel or external device in the system that asserted the pin low.

Shared Fault Pins are used as a bi-directional fault coordination bus for other PSM devices or external devices. Some PSM devices name the shared fault pin "FAULTBX" while others name it "GPIOB". GPIOB pins can be used for other purposes as well, but are typically used for fault sharing. See the next section for a better understanding of how fault sharing behavior works.

FAULT1\_IN indicates a specific fault pin that the channel is responding to. For devices that have two FAULTB pins, this status indicates FAULTB1 pin as the source. For devices that support 4 FAULTB pins, FAULT1\_IN indicates the FAULTB10 pin for channels 0-3, and the FAULTB11 pin for channels 4-7 of the device.

### 8.9.1 Device Behavior in a Shared Fault Scenario

## SHARED FAULT RESPONSE



The example waveform above illustrates fault sharing behavior of the 'root offender' channel in conjunction with a 'responder channel'.

*NOTE: This is one specific example for the purpose of understanding. The following events occur in the example:*

- a -- A channel (called 'root offender') encounters a fault (over voltage in this example) with a non-ignore response setting, and disables its output (example VOUT\_OV\_FAULT\_RESPONSE= Deglitched Off, Infinite Retry)
- b -- When its output is disabled, the root offender, if configured to **propagate** to a shared fault pin (FAULTB OR GPIOB) will immediately pull low on the open drain shared fault pin.
- c -- A responder channel (configured to **respond** to the shared fault pin) will immediately stop power delivery (immediate off)
  - For PSM Controllers, power deliver is stopped by the controller itself, and the condition is reported as GPIOB\_ASSERTED\_LO
  - For PSM Managers, the output enable (VOEN) pin is de-asserted to disable the managed power supply, and the condition is reported as FAULTBX\_IN

*NOTE: The LTC2977, LTC2980, and LTM2987 may be optionally configured to honor off sequencing delays (TOFF\_DELAY) before de-asserting the VOEN pin upon fault.*

- d -- After the root offender has waited its programmable retry delay (MFR\_RETRY\_DELAY), if the fault is no longer present (as in this example), it releases the open drain shared fault pin.
  - This signals to all responder channels that a re-sequence should occur.
  - The rising edge of FAULTB sets time=zero for a coordinated re-sequence of the root offender and any responder channels.
- e -- During the coordinated re-sequence, a responder channel, after waiting its programmed on delay (TON\_DELAY) will start ramping its voltage up
  - Controllers have the ability to ramp the voltage directly, and will ramp the output consistent with the programmed TON\_RISE setting.
  - Managers will assert the assert the VOEN pin to enable the managed supply at this time

## 8.9.2 Possible Causes:

- Missing a pullup on the FAULTB pin
- Another chip or channel is pulling the FAULTB pin low. Check the rest of the system for faults:
  - If the responder channel remains off, the 'Why am I Off?' will provide a succinct list of faults in your system
- An External Device other than a LTC device may be holding FAULTB low.
  - Check the schematic for any other devices that may be connected to the GPIOB pin (an unprogrammed FPGA for instance)
- You may also scan the system for faults manually:
  - Select the FAULT\_WARN\_LIST pseudo-register in the Telemetry/Status window
  - Use the 'All Pages in System' view to scan the FAULT\_WARN\_LIST across the system to see all faults/warnings in the system in one place.
  - If there are any 'red' channels in the system tree, focus on these channels first.
  - Look for critical or important faults (OV, OC, OT, etc) on another chip that is propagating to the FAULTB pin

## 8.9.3 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Configure the channel to ignore the FAULTB pin in the "Fault Sharing" Section of the configuration. This will allow the channel to startup up regardless of the pin.
- To determine if the 'root cause' is a PSM device or external source (unprogrammed FPGA, etc) driving the pin low:
  - Disable propagation to FAULTB/GPIOB for all devices in the system
  - If the FAULTB pin remains low, this indicates an external device or circuit may be keeping the pin low.
- If the DRC check produces any warnings, fix these and see if the problem goes away.

## 8.9.4 Other Debugging Tips

If the channel is presently off, use the "Why am I Off? Tool". Otherwise, if you have an oscilloscope, do the following:

- Trigger the scope on the falling edge of the FAULTB pin and
- Look at other possible signals in the system that may be responsible driving FAULTB low
- Disable fault propagation for all devices. If the pin is still low, trace the schematic for any devices connected to the pin and insure they are not driving it low.

## 8.10 FAULT\_LOG\_PRESENT

The selected device has a non volatile fault log, either manually stored by the user or automatically stored at the time of a fault that disabled an output.

### 8.10.1 Possible Causes:

- A root level fault (with a non-ignore fault response) disabled a channel on this chip and recorded a fault log
- This device responded to a shared FAULTB pin by disabling one or more outputs and recorded a fault log
- The user manually stored a 'snapshot' of telemetry data into the fault log via I2C

### 8.10.2 Tips

Typically it's easier to debug a system if power has not yet been removed. For instance, during board development, you can power the ICs from the dongle (and optionally DC2086A for many devices). When a problem occurs, it's easy to use the latched status registers to debug faults very quickly. Otherwise, if power has been removed since the problem occurred, you can use the non volatile fault log for debugging.

- If power has not yet been removed from the devices, and one or more outputs are OFF, use the Why am I Off? tool to quickly understand why a selected channel is off
- If power has not yet been removed from the devices, scan the system for faults stored in latched registers:
  - Select the FAULT\_WARN\_LIST pseudo-register in the Telemetry/Status view
  - Use the 'All Pages in System' view to scan the entire system for potentially interesting faults.
- If power has been removed, click the Fault Log button in the toolbar and read the fault log into the GUI for further analysis:



### 8.10.3 Other Debugging Tips

If you have an oscilloscope, perhaps the fastest way to insight is to:

- Inspect the registers and the fault log for channels/signals of interest)
- Trigger the scope on the falling edge of the ALERTB signal, and
- Probe other signals of interest (preferably **at the pins of the IC**), and

The ALERTB signal will be pulled low by the IC at the time of the fault, indicating the time of the fault.

## 8.11 GPIOB\_ASSERTED\_LO

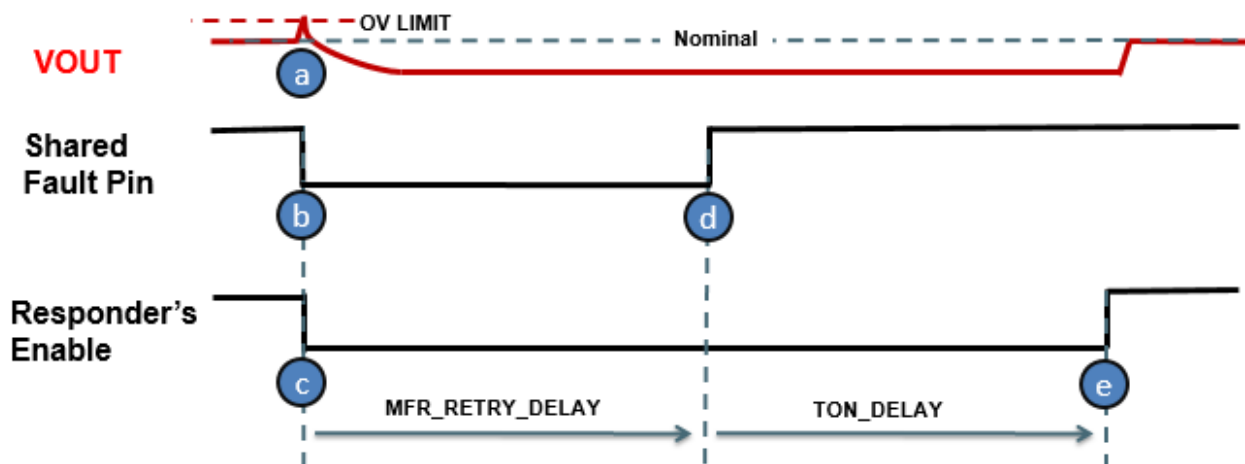
The channel was turned off or prevented from turning on at least once because the channel's GPIOB pin was asserted low and the channel is configured to respond to the pin. The root cause item of interest either another channel or external device in the system that asserted the pin low.

Shared Fault Pins are used as a bi-directional fault coordination bus for other PSM devices or external devices.

Some PSM devices name the shared fault pin "FAULTBX" while others name it "GPIOB". GPIOB pins can be used for other purposes as well, but are typically used for fault sharing. See the next section for a better understanding of how fault sharing behavior works.

### 8.11.1 Device Behavior in a Shared Fault Scenario

## SHARED FAULT RESPONSE



The example waveform above illustrates fault sharing behavior of the 'root offender' channel in conjunction with a 'responder channel'.

*NOTE: This is one specific example for the purpose of understanding. The following events occur in the example:*



- a -- A channel (called 'root offender') encounters a fault (over voltage in this example) with a non-ignore response setting, and disables its output (example VOUT\_OV\_FAULT\_RESPONSE= Deglitched Off, Infinite Retry)
- b -- When its output is disabled, the root offender, if configured to **propagate** to a shared fault pin (FAULTB OR GPIOB) will immediately pull low on the open drain shared fault pin.
- c -- A responder channel (configured to **respond** to the shared fault pin) will immediately stop power delivery (immediate off)
  - For PSM Controllers, power deliver is stopped by the controller itself, and the condition is reported as GPIOB\_ASSERTED\_LO
  - For PSM Managers, the output enable (VOEN) pin is de-asserted to disable the managed power supply, and the condition is reported as FAULTBX\_IN

*NOTE: The LTC2977, LTC2980, and LTM2987 may be optionally configured to honor off sequencing delays (TOFF\_DELAY) before de-asserting the VOEN pin upon fault.*

- d -- After the root offender has waited its programmable retry delay (MFR\_RETRY\_DELAY), if the fault is no longer present (as in this example), it releases the open drain shared fault pin.
  - This signals to all responder channels that a re-sequence should occur.
  - The rising edge of GPIOB sets time=zero for a coordinated re-sequence of the root offender and any responder channels.
- e -- During the coordinated re-sequence, a responder channel, after waiting its programmed on delay (TON\_DELAY) will start ramping its voltage up
  - Controllers have the ability to ramp the voltage directly, and will ramp the output consistent with the programmed TON\_RISE setting.
  - Managers will assert the assert the VOEN pin to enable the managed supply at this time

### 8.11.2 Possible Causes:

- Missing a pullup on the GPIOB pin
- Another chip or channel is pulling the GPIOB pin low. Check the rest of the system for faults:
  - If the responder channel remains off, the 'Why am I Off?' will provide a succinct list of faults in your system
- An External Device other than a LTC device may be holding GPIOB low.
  - Check the schematic for any other devices that may be connected to the GPIOB pin (an unprogrammed FPGA for instance)
- You may have a misconfiguration that may keep the GPIOB low. Read the configuration and click the "DRC" button in the toolbar to check it.
- You may also scan the system for faults manually:
  - Select the FAULT\_WARN\_LIST pseudo-register in the Telemetry/Status window
  - Use the 'All Pages in System' view to scan the FAULT\_WARN\_LIST across the system to see all faults/warnings in the system in one place.
  - If there are any 'red' channels in the system tree, focus on these channels first.
  - Look for critical or important faults (OV, OC, OT, etc) on another chip that is propagating to the GPIOB/FAULTB pin

### 8.11.3 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Configure the channel to ignore the GPIOB pin in the "Fault Sharing" Section of the configuration. This will allow the channel to startup up regardless of the pin.

- To determine if the 'root cause' is a PSM device or external source (unprogrammed FPGA, etc) driving the pin low:
  - Disable propagation to GPIOB/FAULTB for all devices in the system
  - If the GPIOB pin remains low, this indicates an external device or circuit may be keeping the pin low.
- If the DRC check produces any warnings, fix these and see if the problem goes away.

#### 8.11.4 Other Debugging Tips

If the channel is presently off, use the "Why am I Off? Tool". Otherwise, if you have an oscilloscope, do the following:

- Trigger the scope on the falling edge of this channel's GPIOB pin and
- Look at other possible signals in the system that may be responsible driving GPIOB low
- Disable fault propagation for all devices. If the pin is still low, trace the schematic for any devices connected to the pin and insure they are not driving it low.

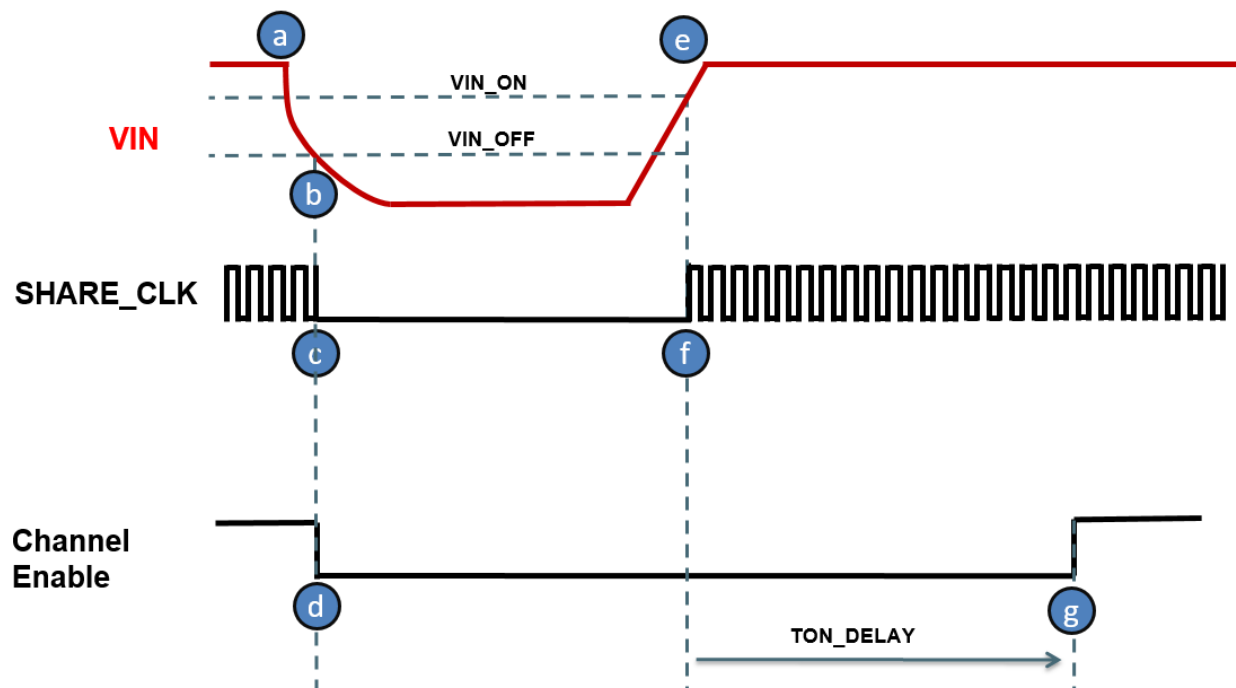
### 8.12 INPUT\_OFF

An INPUT\_OFF warning is detected because the VIN\_SNS pin voltage is below the VIN\_OFF setting. The device will also typically hold SHARE\_CLK low preventing other devices from turning on.

#### 8.12.1 Device Behavior for VIN and SHARE\_CLK

This section provides background information for a deeper understanding of how devices behave and coordinate when VIN is lost or applied.

## SHARE\_CLK and VIN



The example waveform above illustrates how a device behaves when its VIN is lost. It also describes how devices communicate this condition to each other via the SHARE\_CLK signal to insure coordinated behavior. The example starts with the device in normal operating mode with no faults. Typically the SHARE\_CLK signal is connected between PSM devices to insure coherent sequencing. Let's take the scenario where VIN for the system is removed and then re-applied. The behavior of the system is as describe in the following sequence points highlighted above:

- a – The system is operating normally, with VIN good and no faults. SHARE\_CLK is toggling normally to provide a common arbitrated timebase across devices for sequencing.
- b -- A device in the system detects that its VIN voltage drops below its programmed VIN\_OFF threshold.
- c – The first such device to see VIN drop below VIN\_OFF immediately holds SHARE\_CLK low to signal to the system that VIN was lost, and a normal power down should occur
- d -- When a channel (of any device connected to the common SHARE\_CLK) detects that the SHARE\_CLK has stopped, it immediately disables its output
  - For PSM Controllers, power deliver is stopped by the controller itself
  - For PSM Managers, the output enable (VOEN) pin is de-asserted to disable the managed power supply
- e -- VIN ramps back up. Devices detect when VIN rises above their programmed VIN\_ON threshold, and release the common SHARE\_CLK.
- f – When the last such device in the system detects VIN rising above its programmed VIN\_ON threshold, it releases the common SHARE\_CLK signal and the SHARE\_CLK starts toggling again. Devices connected to SHARE\_CLK detect that SHARE\_CLK has started, and initiate a coordinated re-sequence (the start of SHARE\_CLK marks time=zero).
- f – During the coordinated re-sequence, each channel, after waiting its programmed on delay (TON\_DELAY) will start ramping its voltage up
  - Controllers have the ability to ramp the voltage directly, and will ramp the output consistent with the programmed TON\_RISE setting.

- Managers will assert the VOEN pin to enable the managed supply at this time

This coordination feature insures that as power is applied (or lost) the system remains coordinated in its sequencing behavior, and all devices agree on time=zero for up sequencing, and turn off immediately when VIN is lost.

### 8.12.2 Possible Causes:

- VIN not applied
- VIN\_ON setting is too high
- Board or assembly error between the VIN\_SENSE pins of the device and the VIN sensed (missing or cracked resistor, etc)

### 8.12.3 Remedies and Workarounds

Here are some things you can try to narrow things down:

- Apply sufficient VIN voltage (as seen by VIN\_SNS pin)
- Change VIN\_ON and VIN\_OFF settings to lower values

### 8.12.4 Other Debugging Tips

- Even if VIN is applied and > VIN\_ON, all channels may be Off due to SHARE\_CLK pin being held low
- If this is the case, check that other PSM devices that use SHARE\_CLK are not detecting a VIN problem

## 8.13 IOUT\_OC\_FAULT

An IOUT\_OC fault was detected because the differential voltage seen on the ISENSE pins reached the IOUT\_OC\_FAULT\_LIMIT.

### 8.13.1 Possible Causes:

- A fast transient may have occurred. The current supervisor's update rate is 12us.
- IOUT\_CAL\_GAIN is set too low.
- IOUT\_OC\_FAULT\_LIMIT is set too low.
- If sensing current by inductor DCR, that is not captured by a scope
- Output has heavier load than expected or is shorted to GND.

### 8.13.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Changing the output current limits may resolve the problem, but possibly not. There is a deglitched fault response that might help. Set the IOUT\_OC\_FAULT\_RESPONSE to deglitched and set the delay\_count to a non-zero value. The delay\_count value represents how many 12us periods the supervisor checks for an OC condition.
- Change IOUT\_CAL\_GAIN to a larger value which will then translate to a lower output current value.

### 8.13.3 Other Debugging Tips

There are a number of scenarios which can create a IOUT\_OC fault. If you have an oscilloscope, perhaps the fastest way to insight is to:

- Trigger the scope on the falling edge of the ALERTB signal, and
- Look at the ISENSE and VSENSE **pins at the IC.**

The ALERTB signal will be pulled low by the IC at the time of the fault.

## 8.14 OFF

The selected channel is Off. This may be due to a number of possible reasons.

### 8.14.1 Possible Causes:

**TIP: Please use the 'Why am I Off?' tool in the upper right corner of LTpowerPlay for further insight.** If you do not see the tool, click 'View > Restore Default Layout' in the menu. Below is a list of common reasons that a channel may be off:

- Commanded off via I2C
- Commanded off via logic pin (RUN/CONTROL)
- SHARE\_CLK is not pulled up or pulled low, keeping the channel from turning on
- WDI\_RESET is pulled low, keeping the IC in reset (this also prevents communication with the device)
- One or more faults configured as non-ignore may have occurred
- The GPIOB/FAULTB pin may be asserted low
- Vin may be below the programmable VIN\_OFF threshold, which forces channels off

## 8.15 OT\_FAULT

An OT\_FAULT was detected because the temperature exceeded the OT\_FAULT\_LIMIT value. This fault applies to the die temperature on the 2977/2978 devices but is a paged fault on all other SPM devices. The OT/UT warning and fault limits and response are global registers on the 2977/78 and are paged on all other PSM devices.

### 8.15.1 Possible Causes:

- If the device is a LTC2977/78, this fault indicates that high temperature was detected on the internal 2977/78 die
- Otherwise, the fault indicates that high temperature was detected on an external temperature sensor on this PSM channel.
- OT\_FAULT\_LIMIT set too low.
- External TSENSE pin is floating.

### 8.15.2 Remedies and Workarounds

- Check for floating External TSENSE pins (if applicable) and ground them.
- If a VSENSE pin is floating, it is possible that the internal die temp measurement is grossly in error, causing an OT or UT fault or warning.

### 8.15.3 Other Debugging Tips

For 2977/78 devices, only an internal die temperature is available and therefore the OT/UT warn and fault limits are applied to the measured die temperature. For all other PSM devices, the OT/UT warn and fault limits are paged and applied to each channel's external TSENSE pin; there are no internal die temp limits for these devices.

## 8.16 PLL\_UNLOCKED

**8.16.1** The PLL\_UNLOCKED status bit is set because the PWM could not lock on the external clock supplied to the SYNC pin.

### 8.16.2 Possible Causes:

- The SYNC pin is shorted to ground or supply
- The external clock frequency is less than 1/2 the programmed PWM frequency
- The external clock has too much jitter on the falling edge
- The PWM frequency was changed via the PMBus

#### 8.16.2.1 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Check that VIN for the power stage is present
- Check the READ\_FREQUENCY register in the telemetry view; if it is low, zero, or or greater than 10% off the programmed value, there may be a SYNC clock issue
- Probe the SYNC Clock and insure that the frequency is somewhat near to the programmed clock rate; otherwise, you may have a SYNC clock issue
- Clear the fault and see if it returns

#### 8.16.2.2 Other Debug Tips

- The LTC3882 is more sensitive to PWM unlock because it has less filtering on the fault detector than current mode devices
- In general, SYNC should be set to output on one device, and input on all other devices
- The PLL detects the external clock on the falling edge
- A boot strap design that uses internal SYNC that is then over driven by an external clock may not start the external clock cleanly and cause a fault

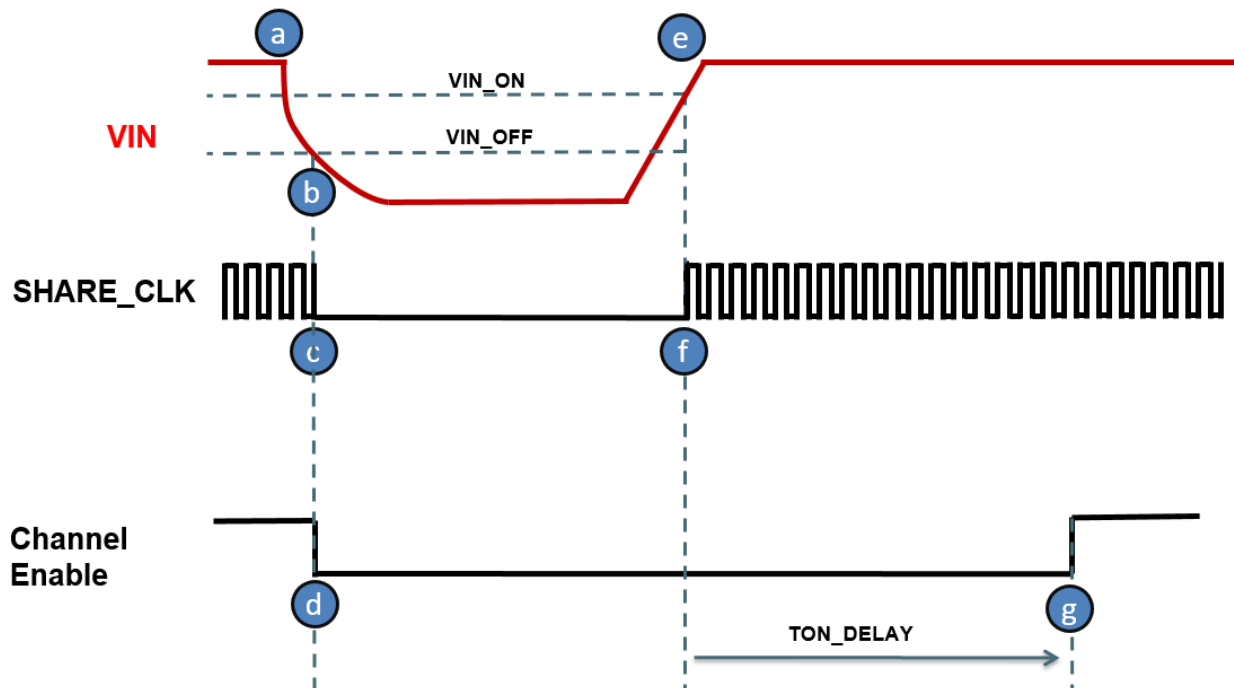
## 8.17 SHARE\_CLK\_LOW

The channel is being held off because the SHARE\_CLK signal is low. The open drain SHARE\_CLK signal is a bi-directional open drain coordination signal used between devices to enforce time=zero and time delay synchronization.

**NOTE: The 'Why am I Off?' tool may provide additional insights.**

## 8.17.1 Device Behavior for VIN and SHARE\_CLK

## SHARE\_CLK and VIN



The example waveform above illustrates the behavior of a the SHARE\_CLK signal in response to VIN, and the behavior of device channels in response to SHARE\_CLK stopping and restarting. The example starts with the device in normal operating mode with no faults. Typically the SHARE\_CLK signal is connected between PSM devices to insure coherent sequencing. Let's take the scenario where VIN for the system is removed and then re-applied. The behavior of the system is as describe in the following sequence points highlighted above:

- a – The system is operating normally, with VIN good and no faults. SHARE\_CLK is toggling normally to provide a common arbitrated timebase across devices for sequencing.
- b -- A device in the system detects that its VIN voltage drops below its programmed VIN\_OFF threshold.
- c – The first such device to see VIN drop below VIN\_OFF immediately holds SHARE\_CLK low to signal to the system that VIN was lost, and a normal power down should occur
- d -- When a channel (of any device connected to the common SHARE\_CLK) detects that the SHARE\_CLK has stopped, it immediately disables its output
  - For PSM Controllers, power deliver is stopped by the controller itself
  - For PSM Managers, the output enable (VOEN) pin is de-asserted to disable the managed power supply
- e -- VIN ramps back up. Devices detect when VIN rises above their programmed VIN\_ON threshold, and release the common SHARE\_CLK.
- f – When the last such device in the system detects VIN rising above its programmed VIN\_ON threshold, it releases the common SHARE\_CLK signal and the SHARE\_CLK starts toggling again. Devices connected to SHARE\_CLK detect that SHARE\_CLK has started, and initiate a coordinated re-sequence (the start of SHARE\_CLK marks time=zero).
- f – During the coordinated re-sequence, each channel, after waiting its programmed on delay (TON\_DELAY) will start ramping its voltage up

- Controllers have the ability to ramp the voltage directly, and will ramp the output consistent with the programmed TON\_RISE setting.
- Managers will assert the VOEN pin to enable the managed supply at this time

This coordination feature insures that as power is applied (or lost) the system remains coordinated in its sequencing behavior, and all devices agree on time=zero for up sequencing, and turn off immediately when VIN is lost.

### 8.17.2 Possible Causes:

- Missing or wrong pullup, preventing SHARE\_CLK from toggling properly. Consult the datasheet for proper pullup selection.
- Missing board connection (cracked or missing 0 ohm resistor to common SHARE\_CLK, board/assembly error, etc)
- One or more PSM devices in the system are holding SHARE\_CLK low
  - They may be in RESET, or
  - Their VIN may be below their programmed VIN\_ON threshold.
- An external device (for example an unprogrammed FPGA) is holding SHARE\_CLK low

### 8.17.3 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Configure the channel to ignore the FAULTB pin in the "Fault Sharing" Section of the configuration. This will allow the channel to startup up regardless of the pin.
- To determine if the 'root cause' is a PSM device or external source (unprogrammed FPGA, etc) driving the pin low:
  - Disable propagation to FAULTB/GPIOB for all devices in the system
  - If the FAULTB pin remains low, this indicates an external device or circuit may be keeping the pin low.
- If the DRC check produces any warnings, fix these and see if the problem goes away.

### 8.17.4 Other Debugging Tips

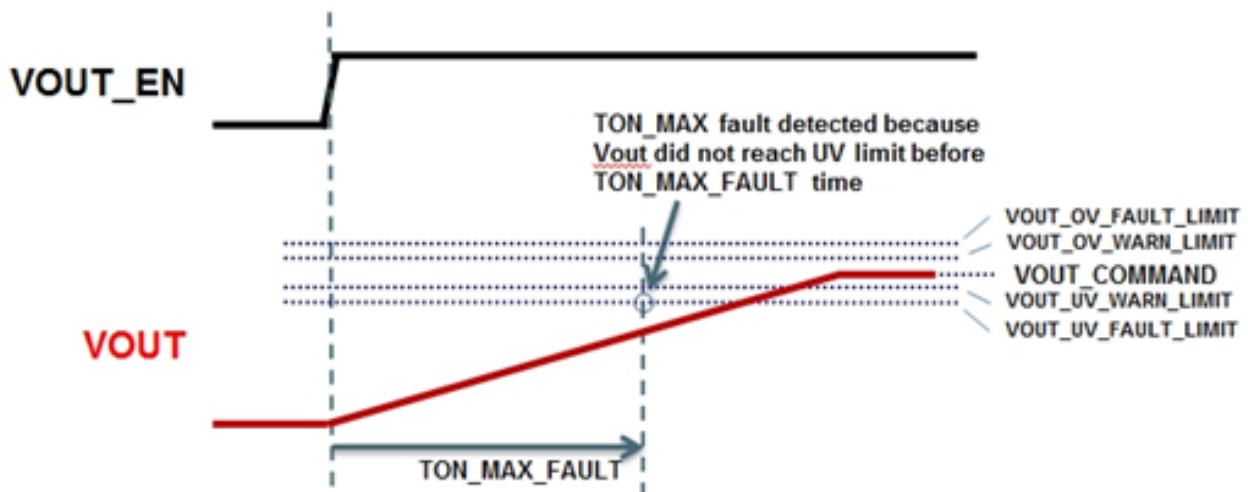
If the channel is presently off, use the "Why am I Off? Tool". Otherwise, if you have an oscilloscope, do the following:

- Trigger the scope on the falling edge of the FAULTB pin and
- Look at other possible signals in the system that may be responsible driving FAULTB low
- Disable fault propagation for all devices. If the pin is still low, trace the schematic for any devices connected to the pin and insure they are not driving it low.



## 8.18 TON\_MAX\_FAULT

**8.18.1** A TON\_MAX fault was detected because the VSENSE voltage did not reach the VOUT\_UV\_FAULT\_LIMIT before the TON\_MAX\_FAULT\_LIMIT.



### 8.18.1.1 Possible Causes:

- Output is shorted, preventing Vout ramp.
- TON\_MAX\_FAULT\_LIMIT set shorter than Vout's ramp up time
  - Large soft start cap
- VOUT\_UV\_FAULT\_LIMIT set too high.
- Regulator feedback resistor value(s) incorrect or assembly issue.
- VOUT\_EN pin has no pullup resistor or power supply enable/RUN pin not pulled high.
- VOEN not physically connected to the power supply enable/RUN pin (failed or missing connection)
- Power supply's input is not enabled, or too weak for regulation
  - If its input is sequenced, enable it **before** this channel

### 8.18.1.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Increase TON\_MAX\_LIMIT to see if the problem goes away
- Set TON\_MAX\_FAULT\_RESPONSE to 'ignore' to see if the output voltage comes up
- Lower the VOUT\_UV\_FAULT\_LIMIT and see if the problem goes away

### 8.18.1.3 Other Debugging Tips

There are a number of scenarios which can create a TON\_MAX\_FAULT. If you have an oscilloscope, perhaps the fastest way to insight is to:

- Trigger the scope on the falling edge of the ALERTB signal, and

- Look at the VOEN and VSENSE **at the pins of the IC**, and
- VIN for the offending channel.

The ALERTB signal will be pulled low by the IC at the time of the fault, indicating the time of the fault.

## 8.19 UT\_FAULT

A UT\_FAULT was detected because the measured temperature was less than the UT\_FAULT\_LIMIT value. This fault applies to the die temperature on the 2977/2978 devices but is a paged fault on all other PSM devices. The OT/UT warning and fault limits and responses are global registers on the 2977/78 and are paged on all other PSM devices.

### 8.19.1 Possible Causes:

- If the device is a LTC2977/78, low temperature was detected on either the internal die
- Otherwise (all other PSM devices) this fault indicates low temperature was detected on an external temperature sensor on this PSM channel.
- UT\_FAULT\_LIMIT set too high
- A VSENSEP pin is floating
- External TSENSE pin is floating

### 8.19.2 Remedies and Workarounds

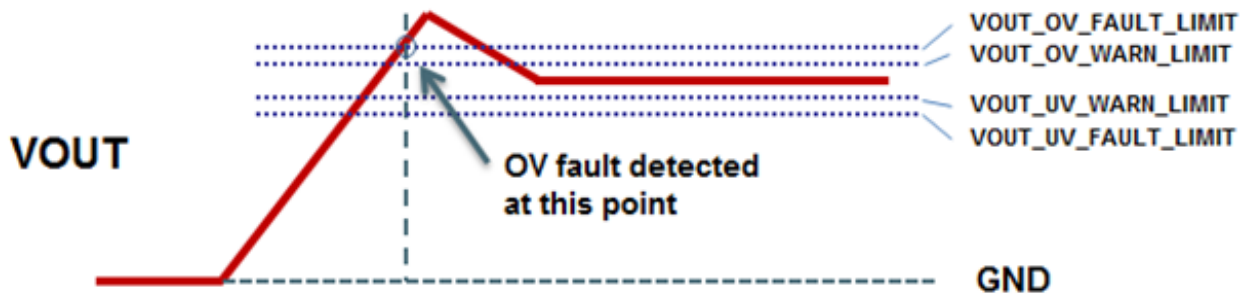
- If applicable, check for floating TSENSE pins and ground them
- If a VSENSE pin is floating, it is possible that the internal die temp measurement is grossly in error, causing an OT or UT fault or warning.

### 8.19.3 Other Debugging Tips

For 2977/78 devices, only an internal die temperature is available and therefore the OT/UT warn and fault limits are applied to the measured die temperature. For all other PSM devices, the OT/UT warn and fault limits are paged and applied to each channel's TSENSE pin; there are no internal die temp limits for these devices.

## 8.20 VOUT\_OV\_FAULT

A VOUT\_OV fault was detected because a voltage greater than the VOUT\_OV\_FAULT\_LIMIT was detected on this channel's VSENSE pin.



### 8.20.1 Possible Causes:

- VOUT\_OV\_FAULT\_LIMIT set too low.
- Regulator feedback resistor value(s) incorrect or assembly issue.
- VOUT\_COMMAND setting too high and DAC/servo moved output above VOUT\_OV\_FAULT\_LIMIT.
- Too short TON\_RISE, causing the PSM Manager to prematurely soft connect during the voltage ramp.

### 8.20.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Increase VOUT\_OV\_FAULT\_LIMIT (or decrease VOUT\_COMMAND) to see if the problem goes away
- Set VOUT\_OV\_FAULT\_RESPONSE to deglitched and increase deglitch interval to see if problem goes away
- Lower the VOUT\_UV\_FAULT\_LIMIT and see if the problem goes away

### 8.20.3 Other Debugging Tips

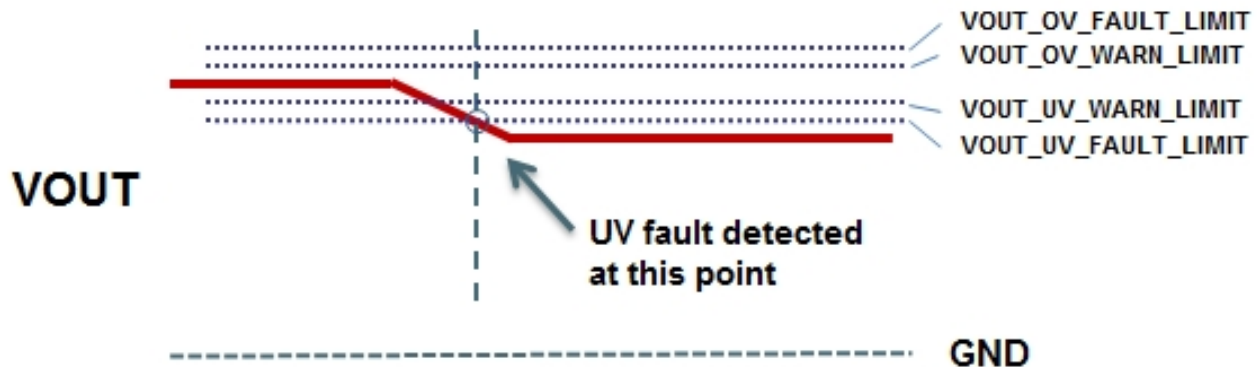
With an OV fault, it's best to be safe and look at the signals with an oscilloscope. Do the following:

- Trigger the scope on the falling edge of the ALERTB signal, and
- Look at VSENSE pins **at the IC**, and
- VIN for the offending channel (or other signals of interest)

The ALERTB signal will be pulled low by the IC at the time of the fault, indicating the time of the fault.

## 8.21 VOUT\_UV\_FAULT

A VOUT\_UV fault was detected because a voltage less than the VOUT\_UV\_FAULT\_LIMIT was detected on this channel's VSENSE pin.



### 8.21.1 Possible Causes:

- VOUT\_UV\_FAULT\_LIMIT set too high.
- VOUT\_COMMAND setting too low and DAC/servo moved output below VOUT\_UV\_FAULT\_LIMIT.
- Regulator feedback resistor value(s) incorrect or assembly issue.
- The enable/RUN pin of the device was disabled externally while the channel was ON
- The channel dipped under a load after sequencing up properly (after TON\_MAX\_FAULT\_LIMIT)

### 8.21.2 Remedies and Workarounds

Here are some experiments you can try to narrow things down:

- Lower VOUT\_UV\_FAULT\_LIMIT to see if the problem goes away
- Set the VOUT\_UV\_FAULT\_RESPONSE to 'deglitched off' and increase the deglitch interval until the problem goes away
- Set the VOUT\_UV\_FAULT\_RESPONSE to 'ignore' and see if the system sequences up. Then issue CLEAR\_FAULTS to determine whether the problem occurs during startup only or afterward.

### 8.21.3 Other Debugging Tips

One thing to keep in mind with UV faults are that they can only occur after the device has successfully risen above the VOUT\_UV\_FAULT\_LIMIT within the time budget of TON\_MAX\_FAULT\_LIMIT after the output is enabled. This means that the output did come up initially, and then fell below the VOUT\_UV\_FAULT limit while the channel was ON. If you have an oscilloscope, perhaps the fastest way to insight is to:

- Trigger the scope on the falling edge of the ALERTB signal, and
- Look at the VOEN and VSENSE **at the pins of the IC**, and
- VIN for the offending channel (or other relevant signals/channels)

The ALERTB signal will be pulled low by the IC at the time of the fault, indicating the time of the fault.

## 9 Design

This FAQ Section contains Frequently Asked Questions about Designs

### 9.1 How do you design the addressing of a PSM system?

#### 9.1.1 How do you design the addressing of a PSM system?

App Note AN152 contains all the information required to:

- Understand how PSM Controller and PSM Manager addresses works
- How to design the address map using address registers and external pins
- How to use bus muxes for large designs

See: <http://cds.linear.com/docs/en/application-note/an152f.pdf>