Fault Log Decoding with Linduino PSM

Michael Jones

Introduction

LTC power system management devices are PMBus controlled point-of-load converters (LTC®388X) and power system managers (LTC297X). All LTC power system management (PSM) devices have a fault log that is written to EEPROM when there is a fault. Once the fault log is written, fault logs are no longer generated until they are re-enabled, typically after the data has been read back.

LTpowerPlay® helps debug a system by reading the fault log, decoding it into human-readable format, displaying it and saving it to a file. A system with a board management controller (BMC), can also read the fault log from EEPROM and re-enable it. Firmware can store, transfer over a network and decode fault data read from a PSM device.

The Linduino® Sketchbook contains an example sketch that reads fault logs, along with the supporting libraries for PMBus and fault log decoding.

This application note describes the basics of reading and decoding fault logs using Linduino.

Hardware

The hardware used for this application note is:

1. DC2026 (Linduino)
2. DC2294 (Shield)
3. DC1962 (Power Stick)

Software

The software used for this application note is:

1. Arduino 1.6.6
2. LTSketchbook

This application note is most useful if the hardware is obtained from LTC and the results duplicated using the LTSketchbook downloaded from www.linear.com/linduino. However, there is still enough material in the application note to explore and understand how fault logs work.

Fault Log Basics

LTC PSM devices constantly store telemetry data in RAM prior to a fault. When a fault occurs, the device transfers RAM data to EEPROM. The fault log data includes telemetry before and after the fault.

The PSM device uses a telemetry loop, which stores data in a circular RAM buffer. When the state machine reaches the end of the buffer, it wraps around and keeps writing. When a fault occurs, the device stores a little more data to finish the current telemetry loop and then writes the circular buffer to EEPROM. The device attempts to write the entire buffer into EEPROM until power is lost.

The fault log data contains several types of information:

1. Cause of the fault log
2. Position of telemetry MUX at time of fault
3. Clock tick value
4. Peak values of voltage, current and temperature
5. Multiple loops of telemetry

When the fault log data is read from the device by a board management controller (BMC), the device returns a block of data. Each PSM device data sheet contains a description of the data.

http://www.linear.com/products/Digital_Power_System_Management

Reading The Data

There are several PSM PMBus commands related to a PSM fault logs:

- MFR_FAULT_LOG_STORE (0xEA)
- MFR_FAULT_LOG_CLEAR (0xEC)
- MFR_FAULT_LOG (0xEE)

LTC297x devices also implement the following commands:

- MFR_FAULT_LOG_RESTORE (0xEB)
- MFR_FAULT_LOG_STATUS (0xED)

The PSM controllers (LTC388X and fully integrated µModules®) support reading directly from EEPROM and PSM managers (LTC297X) transfer the fault log to RAM via MFR_FAULT_LOG_RESTORE, which is then read using MFR_FAULT_LOG. The fault log memory in Figure 1, shows the topology of the LTC2977 memory.

The MFR_FAULT_LOG_STORE command forces a fault log for both families, which is useful for testing with Linduino. The MFR_FAULT_LOG command uses Block Read Protocol to return the contents of the fault log. The MFR_FAULT_LOG_STORE command uses a Send Byte Protocol, which is a command without a data byte.

LTpowerPlay and Linduino use the same commands and PMBus protocols for fault log management.

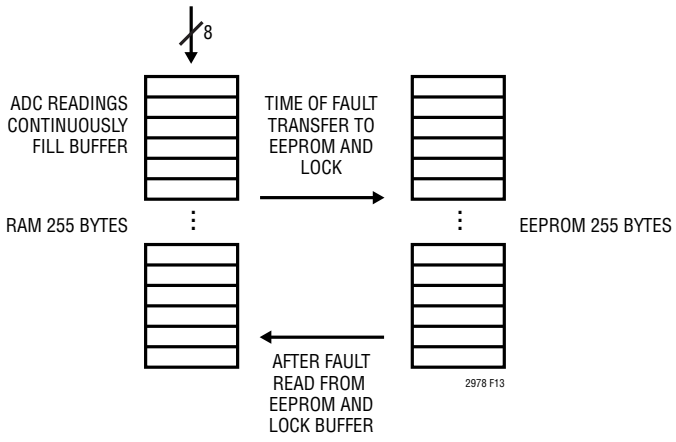

Figure 1. Fault Log Memory

Decoding The Data

The LTC3880 fault log data is structured into two entities:

1. Header information
2. Cyclical data

The header contains:

1. The cause of the fault
2. Internal time value at the time of fault
3. Peak telemetry values

The cyclical data contains:

1. VOUT
2. IOUT
3. VIN
4. IIN
5. STATUS VOUT
6. STATUS WORD
7. STATUS MFR SPECIFIC

There are five blocks of cyclical data, starting with the newest data and going backwards in time.

Table 1. LTC3880 Fault Log Header

HEADER INFORMATION			
Position_Fault		BYTE	0
MFR_REAL_TIME	[7:0}	BYTE	1
	[15:8}	BYTE	2
	[23:16]	BYTE	3
	[31:24]	BYTE	4
	[39:32]	BYTE	5
	[47:40]	BYTE	6
MFR_VOUT_PEAK (PAGE 0)	[15:8]	LIN 16	7
	[7:0]	LIN 16	8
MFR_VOUT_PEAK (PAGE 1)	[15:8]	LIN 16	9
	[7:0]	LIN 16	10

The LTC3880 data sheet gives a detailed description of the fault log header data, as shown in Table 1, and also specifies the data format. For example, MFR_VOUT_PEAK is LINEAR 16 (L16) format. The data values are the same values that are returned by reading the command of the same name and are formatted identically. The right most column gives the index within the block of bytes.

The LTC3880 position fault values are shown in Table 2. This is the first fault to occur and is the most important information. It is written to EEPROM before any other value.

Table 2. LTC3880 Position Fault

Explanation of Position_Fault Values	
POSITION_FAULT VALUE	SOURCE OF FAULT LOG
0xFF	MFR_FAULT_LOG_STORE
0x00	TON_MAX_FAULT Channel 0
0x01	VOUT_OV_FAULT Channel 0
0x02	VOUT_UV_FAULT Channel 0
0x03	IOUT_OC_FAULT Channel 0
0x05	OT_FAULT Channel 0
0x06	UT_FAULT Channel 0
0x07	VIN_OV_FAULT Channel 0
0x0A	MFR_OT_FAULT Channel 0
0x10	TON_MAX_FAULT Channel 1
0x11	VOUT_OV_FAULT Channel 1
0x12	VOUT_UV_FAULT Channel 1
0x13	IOUT_OC_FAULT Channel 1
0x15	OT_FAULT Channel 1
0x16	UT_FAULT Channel 1
0x17	VIN_OV_FAULT Channel 1
0x1A	MFR_OT_FAULT Channel 1

The LTC297X family has a similar fault log structure, with fault cause, header and looped data. Consult a LTC297X data sheet for more information[1].

Code Strategy

The Linduino code can read the raw data and decode it into a nested C structure. In most cases, the mapping is one-to-one, but in a few cases, some data must be copied. Once the data is structured, the code can process the elements of the log or generate human readable text.

In some applications, the BMC will not process the data at all. In these cases, the block of data may be sent over a network and processed on another computer. That computer can use the same strategy to process the data. Because the Linduino code is generic, it can run on the BMC or a server, laptop, etc.

Note 1. The Linduino code will take care of things, so there is no need to be an expert in the structure of the data.

Dumping Logs with a Sketch

Before reviewing the library code, let's review a sketch from the LTSketchbook that prints the fault logs of a DC1962 as shown in Figure 2. The DC1962 contains an LTC3880, LTC2974 and LTC2977.

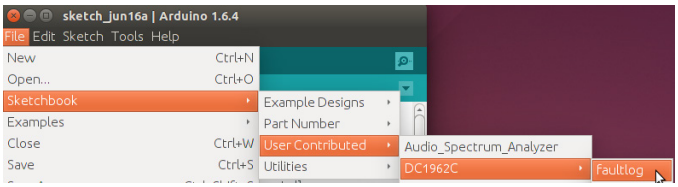

Figure 2. Loading Fault Log Sketch

The sketch is found in the User Contributed section under DC1962C. After loading it, compile, upload and open the console window (shown in Figure 3).

```
1-Dump Fault Logs
2-Clear Fault Logs
3-Clear Faults
4-Bus Probe
5-Reset
6-Store Fault Log

Enter a command:
```

Figure 3. Fault Log Sketch Menu

The menu has three main commands: Dump, Clear and Store. The Store menu item will generate a fault log, or you can cause a fault by pressing the CREATE FAULT button on the DC1962.

For example, short the CH0 output on the DC1962C to GND and then select Dump Fault Log from the menu. The resultant LTC3880 fault log is shown in Figure 4.

Notice the top of the data: the fault position is IOUT_OC_FAULT on channel 0. The Over Current Comparator faulted before the Under Voltage Comparator. Depending on the device's overcurrent and undervoltage settings, the undervoltage fault may occur first in this condition. A time is given, which tells how long the system has been running since reset. The time is useful for correlation between fault logs of other devices.

Next, the data shows all peak values since reset or clear command. This is often helpful if the temperature or output current is higher than normal, giving clues to the cause.

Finally, the data displays loops of telemetry. Notice that Chan0 is 0.0V in Loop 0. Loop 0 is the most recent telemetry, where the fault occurred, and Loop 1 occurred just before the fault, etc. At Loop 1, the voltage was 0.849V and, at Loop 0, it is 0.0V. The telemetry loop takes approximately 100ms and, when there is a fault, the last loop completes. Therefore, it is possible to have output voltage data before the fault occurred, even in Loop 0.

Note: The output currents are small because there are very small loads on the DC1962.

```
LTC3880 Log Data
Fault Position IOUT_OC_FAULT Channel 0
Fault Time 0x00000002dcc5
187589 Ticks (200us each)

Header Information:
--------
VOUT Peak 0 0.852295
VOUT Peak 1 1.100830
IOUT Peak 0 0.002304
IOUT Peak 1 0.002975
VIN Peak 4.984375
Temp External Last Event Page 0 30.156250
Temp External Last Event Page 1 30.218750
Temp External Last Event 36.875000
Temp External Peak Page 0 30.468750
Temp External Peak Page 1 30.625000

Fault Log Loops Follow:
(most recent data first)
-------
Loop: 0
-------
Input: 4.929687 V, 0.099976 A
Chan0: 0.000000 V, -0.000168 A
  STATUS_VOUT: 0x00
  STATUS_MFR_SPECIFIC: 0x00
  STATUS_WORD: 0x4851
Chan1: 0.094971 V, 0.000092 A
  STATUS_VOUT: 0x00
  STATUS_MFR_SPECIFIC: 0x01
  STATUS_WORD: 0x1841
-------
Loop: 1
-------
```

Figure 4. LTC3880 Fault Log

```
LTC2974 Log Data
Fault Time 0x00000002a218
172568 Ticks (200us each)

Peak Values and Fast Status
-----
Vout0: Min: 1.499512, Peak: 1.500732
Temp0: Min: 32.437500, Peak: 32.687500
Iout0: Min: 0.000122, Peak: 0.002777
Fast Status0
  STATUS_VOUT0: 0x00
  STATUS_IOUT0: 0x00
  STATUS_MFR0: 0x60

Vin: Min 4.937500, Peak: 4.992187

Vout1: Min: 1.799194, Peak: 1.802978
Temp1: Min: 32.312500, Peak: 32.437500
Iout1: Min: 0.000229, Peak: 0.003357
Fast Status1
  STATUS_VOUT1: 0x00
  STATUS_IOUT1: 0x00
  STATUS_MFR1: 0x60

Vout2: Min: 1.999512, Peak: 2.000854
Temp2: Min: 32.00000, Peak: 32.312500
Iout2: Min: 0.000565, Peak: 0.004166
Fast Status2
  STATUS_VOUT2: 0x00
  STATUS_IOUT2: 0x00
  STATUS_MFR2: 0x68

Vout3: Min: 2.199219, Peak: 2.200928
Temp3: Min: 31.312500, Peak: 31.468750
Iout3: Min: 0.000076, Peak: 0.004303
Fast Status3
  STATUS_VOUT3: 0x00
  STATUS_IOUT3: 0x00
  STATUS_MFR3: 0x68

Fault Log Loops Follow:
(most recent data first)
-------
Loop: 0
-------
CHAN3
  READ_POUT: 0.009109 W
  READ_IOUT: 0.004135 A
  STATUS_IOUT: 0x00
  STATUS_TEMP: 0x00
  READ_TEMP: 31.312500 C
  STATUS_MFR: 0x18
  STATUS_VOUT: 0x00
  READ_VOUT: 2.199463 V
CHAN2:
```

Figure 5. LTC2974 Fault Log

Following the LTC3880 fault log is the fault data for the LTC2974 as shown in Figure 5. Why is there a fault log for a device that was not shorted?[2] Look at the STATUS_MFRn in the Peak Values and Fast Status and in Loop 0. There are the following values: 0x60, 0x60, 0x68 and 0x68. In Loop 0 it is 0x18.

The LTC2974 STATUS_MFR_SPECIFIC, as shown in Table 3, shows the meaning of the STATUS_MFRn values. From Fast Status the following bits are high:

- Status_mfr_fault1_in
- Status_mfr_fault0_in
- Status_mfr_dac_connected

On the DC1962, the LTC3880 GPIOB lines are connected to LTC2974 FAULTB lines and the LTC2974 is configured to fault off when they are pulled to ground, which caused the fault log. All channels report a fault0/1_status event, indicating that these channels have responded to a shared fault pin. Channels 0 and 1 also report a disconnected DAC after disabling their outputs. The other channels lost their DAC connection shortly thereafter.

The fault log of the LTC2977 has faults similar to the LTC2974.

Table 3. LTC2974 STATUS_MFR_SPECIFIC Data Contents

BIT(S)	SYMBOL	OPERATION	CHANNEL	FAULT
b[7]	Status_mfr_discharge	1 = A V_{OUT} discharge fault occurred while attempting to enter the ON state. 0 = No V_{OUT} discharge fault has occurred.	Current Page	Yes
b[6]	Status_mfr_fault1_in	This channel attempted to turn on while the FAULTB1 pin was asserted low, or this channel has shut down at least once in response to a FAULTB1 pin asserting low since the last CONTROL pin toggle, OPERATION command ON/OFF cycle or CLEAR_FAULTS command.	Current Page	Yes
b[5]	Status_mfr_fault0_in	This channel attempted to turn on while the FAULTB0 pin was asserted low, or this channel has shut down at least once in response to a FAULTB0 pin asserting low since the last CONTROL pin toggle, OPERATION command ON/OFF cycle or CLEAR_FAULTS command.	Current Page	Yes
b[4]	Status_mfr_servo_target_reached	Servo target has been reached.	Current Page	No
b[3]	Status_mfr_dac_connected	DAC is connected and driving V_{DAC} pin.	Current Page	No
b[2]	Status_mfr_dac_saturated	A previous servo operation terminated with maximum or minimum DAC value.	Current Page	Yes
b[1]	Status_mfr_auxfaultb_faulted_off	AUXFAULTB has been de-asserted due to a V_{OUT} or I_{OUT} fault.	All	No
b[0]	Status_mfr_watchdog_fault	1 = A watchdog fault has occurred. 0 = No watchdog fault has occurred.	All	Yes

Note 2. All three devices faulted at the same time and the sketch read fault logs from all three devices.

Sketch Code

The sketch code uses an API found in the library file LT_PMBus/LT_FaultLog.h, shown in Figure 6.

```
bool hasFaultLog (uint8_t address);
void enableFaultLog(uint8_t address);
void disableFaultLog(uint8_t address);
void clearFaultLog(uint8_t address);
virtual void print(Print* printer = 0) = 0;
```

Figure 6. Fault Log API

The code behind the API detects the type of device and does "the right thing." Using this API, the code to print the fault log is simple, as shown in the Figure 7 fault log sketch code. The FaultLog class is a general or base class that is specialized by subclasses for each device type.

```
static LT_3880FaultLog *faultLog3880 =
  new LT_3880FaultLog(smbus);

if (
  faultLog3880->hasFaultLog(ltc3880_i2c_address))
{
  faultLog3880->read(ltc3880_i2c_address);
  faultLog3880->print(&Serial);
  faultLog3880->release();
}
else
  Serial.println(F("No LTC3880 Fault Log"));
```

Figure 7. Fault Log Sketch Code

The code only needs to check if there is a log using a device-specific object and, if so, read it, print it and release it. The "read" function will allocate internal memory for the data, "print" will print it and "release" will release the memory. If you need direct access to the structure, there is a "get" function on each subclass, which returns a structure for that device type. With the structure, you can write code that directly processes the raw data.

One way to think of this design is the FaultLog class defines a common API and the subclasses implement the device-specific behavior. Because the devices have slightly different PMBus commands and the data formats are different, the subclasses are implementing the specialized behavior but using the same common API.

Porting

The library code is generic other than the print function. To port the code to a non-Arduino platform, remove or replace the print function. Also, make sure the structures are packed and that the pointer cast does not have any memory alignment problems.

If you wish to use your own PMBus library, then you also must change the SMBus/PMBus calls for your library.

Summary

Fault logs are a very powerful feature of power system management devices and offer a non-volatile historical account of a fault. Reading and interpreting the raw data of the LTC388X and LTC297X families is easy with the provided library.

The Linduino PSM fault log library and example sketch make the job straightforward, with a common API. Porting the code is simple because the only Arduino-specific code is for printing, which is easy to replace compared to the decoding of the bytes and learning the PMBus commands for each device type.

If you have any questions about the Linduino PSM Fault Log Library or Linduino PSM, contact your local LTC field application engineer or sales office (www.linear.com/contact).

Linear Technology Corporation
1630 McCarthy Blvd., Milpitas, CA 95035-7417
(408) 432-1900 • FAX: (408) 434-0507 • www.linear.com

LT 0117 • PRINTED IN USA