Fault Logging Using Platform Manager 2

January 2018 Technical Note TN1277

Introduction

Complex systems are required to monitor system power supplies, currents, temperatures, I/Os, or other critical signals. These systems benefit from the ability to report or log any faults when the system is operating out of its specifications. In addition, for maintenance and troubleshooting purposes it is very desirable to know the cause of a system failure, the type of failure condition (under or over specified limit), the time of failure, and the status of critical signals and I/Os at the time of failure.

Hardware Management Controllers based on Lattice Platform Manager 2 designs can realize an easy to implement fault logging component. The fault logging component includes the following features:

- Capture and record fault conditions at the moment of initial fault detection
- Capture and record fault timestamp
- Fault recording operates in parallel to system shutdown sequence or other housekeeping
- Store fault records in internal or external memory
- Record all voltage, current, and temperature monitor states as well as configurable user signals
- Support for fault log readback without power-down

The Lattice Diamond software provides the Platform Designer tool which is used to configure Hardware Management Controllers based on Platform Manager 2 designs. The Fault Log component of this tool allows easy configuration of the fault logging task with the desired features.

This document covers the different options and behaviors for fault logging with various Platform Manager 2 designs. Platform Manager 2 designs can be built using the following Lattice devices; LPTM21 alone or with L-ASC10(s), MachXO2 with L-ASC10(s), MachXO3LF with L-ASC10(s), and ECP5 with L-ASC10(s). The L-ASC10 is a hardware management expander and up to eight of them can be included in a single Platform Manager 2 design. The LPTM21 device has a single L-ASC10 device built in. The hardware and fault logging behavior is described in detail and Platform Designer tool interface is also covered.

Fault Logging Overview

Platform Manager 2 designs have two configurable fault logging mechanisms:

- V,I,T Fault Logging: L-ASC10 device/section built-in Voltage, Current, and Temperature Fault Log hardware and memory
- Full-Featured Fault Logging: FPGA IP based Fault Log using built-in User Flash Memory (UFM) or external SPI memory

The two mechanisms can be enabled independently in a hardware management controller. Table 1 compares the features of each of the fault logging approach.

Table 1. Fault Logging Feature Table

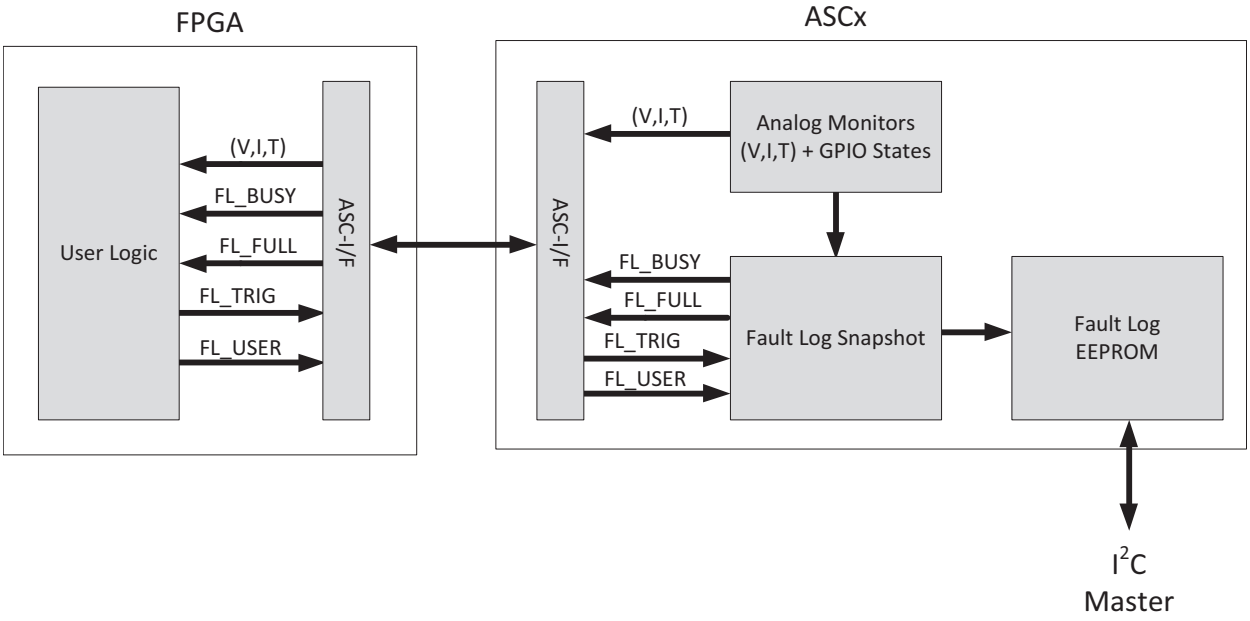
Feature	V,I,T Fault Logging	Full-Featured Fault Logging	
Memory Storage Location	L-ASC10 Fault Log EEPROM	UFM	External SPI Flash
Logic Resources for Implementation	N/A	Design Dependent[1]	Design Dependent[1]
Supported Devices (recommended minimum)	LPTM21, MachXO2-640, MachXO3LF-640, and ECP-12	MachXO2-2000, MachXO3LF-2100, and ECP-12	MachXO2-2000, MachXO3LF-2100, and ECP-12
Max Record Count	16	Design Dependent[2]	512
Fault Recording Time	< 5ms	Design Dependent[3]	Dependent on External SPI Flash write time
Timestamp Support	N/A	32-bit	32-bit
User Bytes	1 per ASC Device	Up to 4 bytes	Up to 4 bytes
L-ASC10 User Tag Memory Support	Disabled	Available	Available
Capture VMON, IMON, TMON + ASC GPIO status	Yes	Yes	Yes
Configurable Trigger Signal	Yes	Yes	Yes
User-Accessible Busy Signal	Yes	Yes	Yes
User-Accessible Full Signal	Yes	No	Yes
User-Accessible Record Count Signals	Yes	No	No
Readback Method	I2C, JTAG-I2C	I2C, JTAG	SPI

1. The resource count depends on the number of ASCs logged, as well as which other components are used in the design (VID, Dual Boot Golden Image, etc).
2. The number of records available to store in the UFM implementation depends on the record size (see Equation 2).
3. The fault recording time in the UFM implementation depends on the number of ASCs logged.

V,I,T Fault Logging relies on dedicated hardware in the L-ASC10 device. The block diagram is shown in Figure 1 below. The user logic design in the Hardware Management Controller FPGA is only responsible for triggering the fault recording process. The L-ASC10 fault log hardware keeps a current snapshot of the status of all Voltage Monitors (VMON), Current Monitors (IMON), and Temperature Monitors (TMON) as well as all ASC GPIO signal states. When the fault log trigger signal is asserted, the L-ASC10 fault log hardware will transfer this snapshot into a record in the dedicated fault log EEPROM.

The V,I,T fault logging method is covered in detail in the L-ASC10 device data sheet. The theory of operation area of the data sheet covers the basic mechanism and signals associated with the V,I,T fault log. The I^2C interface section of the data sheet covers the fault log readout and erase functions.

Figure 1. V,I,T Fault Logging



The Full-Featured Fault Log is implemented using a soft-IP component in the FPGA logic. A simplified block diagram of this implementation is shown in Figure 2 (using UFM) and Figure 3 (using external SPI). When the Full-Featured Fault Log component is enabled, IP is automatically instantiated in the background to handle the fault logging task. This IP snapshots the current status of the V,I,T monitors and records that status to either the on-chip UFM or external SPI Flash memory whenever the trigger signal is asserted.

Additional details on the fault record map, fault logger behavior, and software settings are provided later in this document.

Figure 2. Full-Featured Fault Log (UFM)

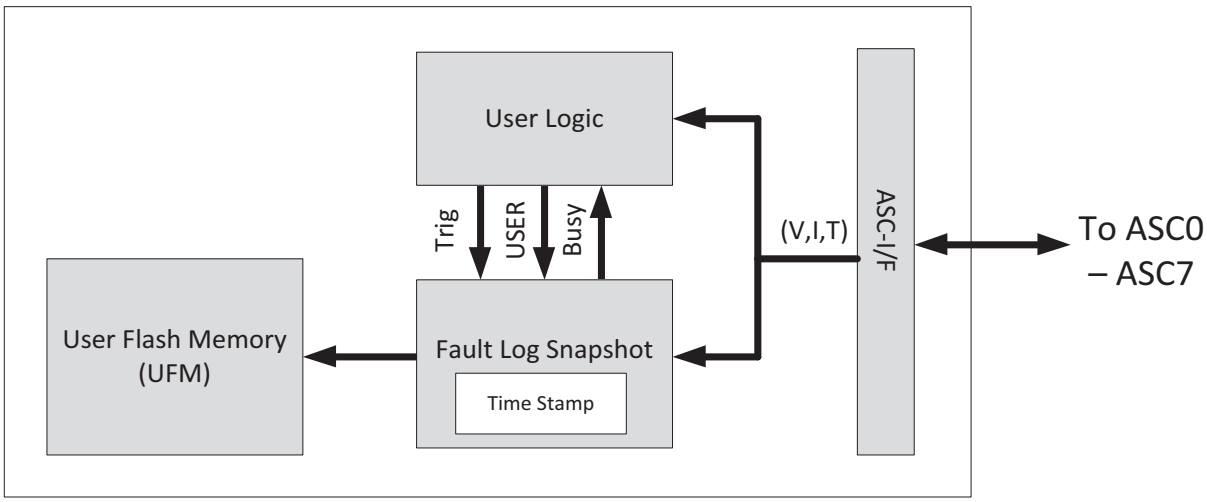
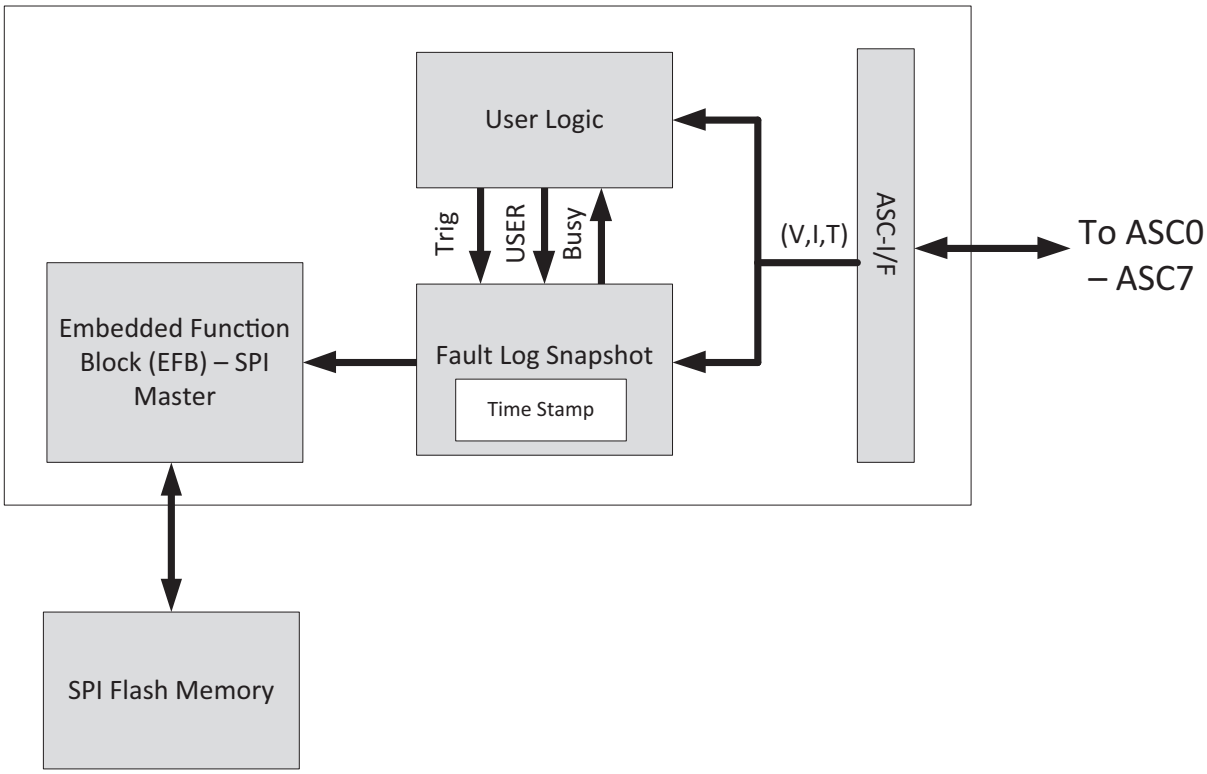


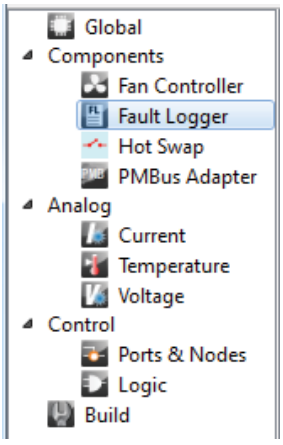
Figure 3. Full-Featured Fault Log (External SPI)



Platform Designer: Fault Logger Component

The Platform Designer tool (a part of Lattice Diamond software) is used to configure Hardware Management Controllers based on Platform Manager 2 designs. The Fault Logger Component View selection is shown in Figure 4.

Figure 4. Fault Logger Component

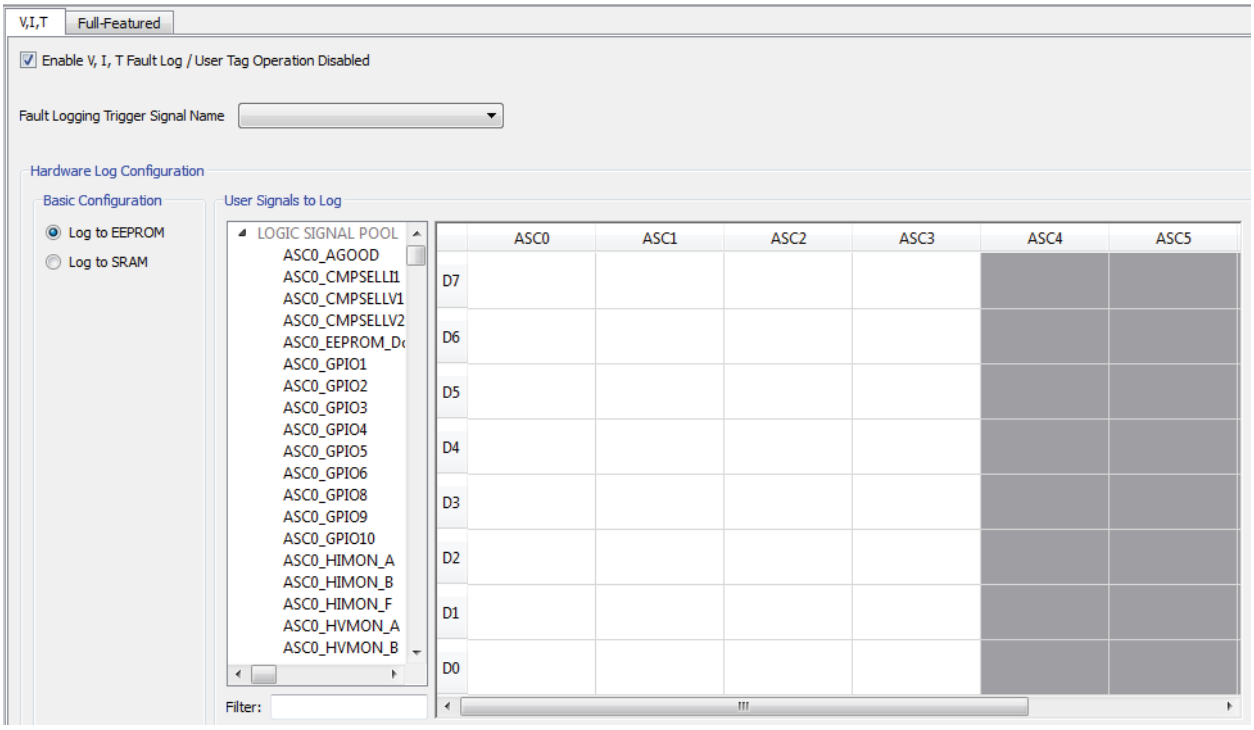


There are two tabs in the Fault Logger Component View: V,I,T and Full-Featured. The tabs are configured independently and are described below.

V,I,T Fault Logger Configuration

The V,I,T fault logger is configured using the settings below. These settings are shown in Figure 5.

Figure 5. V,I,T Fault Log - Configuration



Enable V,I,T Fault Log / User Tag Operation Disabled

This check box enables the V,I,T fault logging view for configuration. Additionally, this setting will configure the L-ASC10 device to reserve the EEPROM for Fault Logging instead of User Tag access. See the L-ASC10 device data sheet for more information.

Fault Logging Trigger Signal Name

This drop-down menu is used to assign a node (or port) from the design as the trigger signal for all ASC devices in the design. Whenever this signal is asserted, all ASC devices present in the design will record their current fault log snapshot to either EEPROM memory or SRAM memory (depending on Basic Configuration setting). The trigger signal is rising edge sensitive. For more details on the automatically generated signals associated with this component, see the L-ASC10 device data sheet, fault logging section.

Basic Configuration

These radio buttons select the record memory to be used in the design. The L-ASC10 provides an EEPROM memory for storing up to 16 records as well as an SRAM / volatile register that can store 1 fault record at a time. The SRAM will be overwritten every time a fault log is recorded, while the EEPROM memory location will increment automatically each time a record is stored.

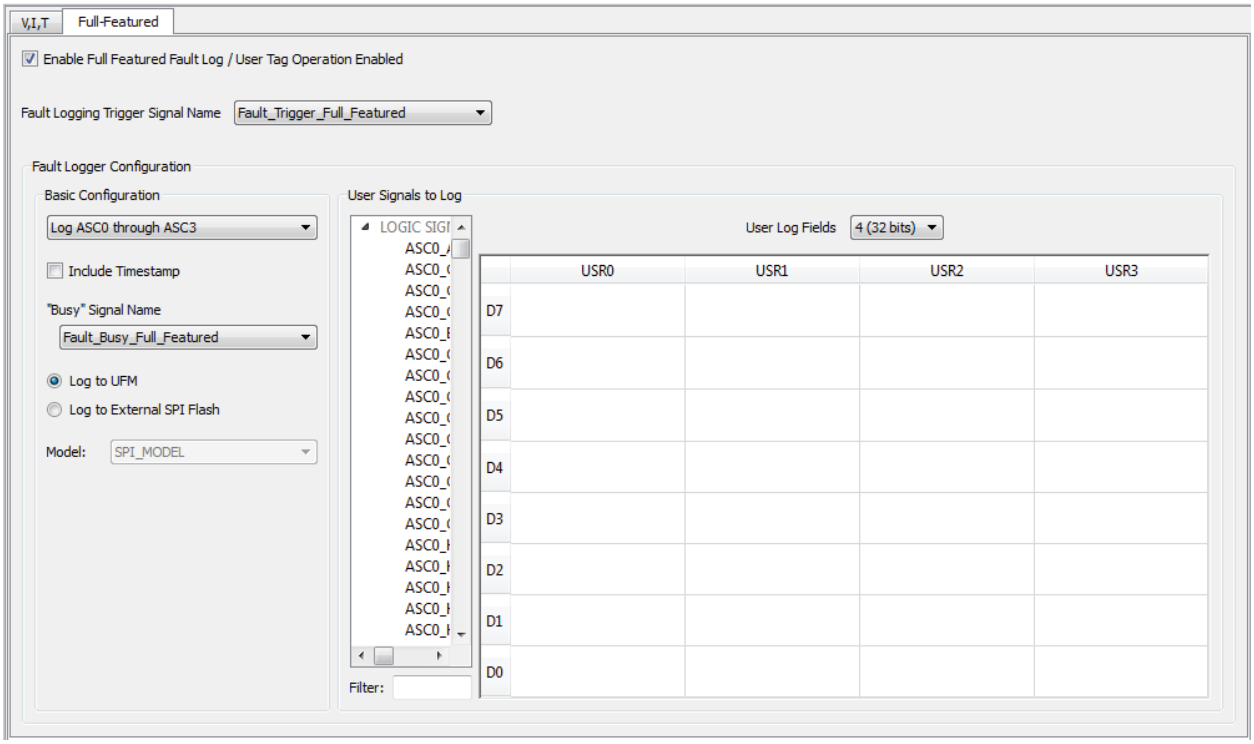
User Signals to Log

This area of the component view is a drag and drop interface for storing user-selected signals in the EEPROM memory of each ASC. These signals are stored in addition to the V,I,T status signals, which are automatically stored every time a fault is logged and do not have to be manually selected. Each ASC supports up to 8 user-selected signals. User Signal selections which are not defined will be automatically set to 0.

Full-Featured Fault Logger Configuration for LPTM21, MachXO2, and MachXO3LF

The Full-Featured Fault Logger component presents two different configuration views based on the FPGA family used for the Platform Manager 2 design. The more complex configuration view is used for ECP5 based designs and a simpler configuration view is used for LPTM21, MachXO2, and MachXO3LF based designs. The Full-Featured fault logger is configured using the settings below for Platform Manager 2 Designs based on LPTM21, MachXO2, or MachXO3LF. These settings are shown in Figure 6.

Figure 6. Full-Featured Fault Log Configuration for LPTM21, MachXO2, and MachXO3LF



Enable Full-Featured Fault Log

This check box enables the Full-Featured fault logging view for configuration. Additionally, this setting will instantiate the IP components associated with the Full-Featured fault log automatically in the background. This check box will be automatically disabled if the Platform Manager design does not have enough FPGA resources for the selected components and number of L-ASC10(s).

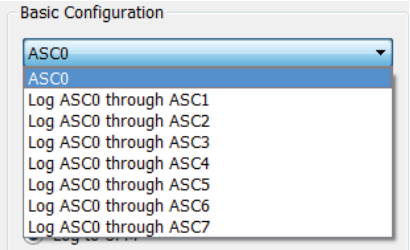
Fault Logging Trigger Signal Name

This drop-down menu is used to assign a node (or port) from the design as the trigger signal for the Full-Featured Fault Log. Whenever this signal is asserted, the IP component will snapshot the current V,I,T status, as well as the selected User Signals and Timestamp (if enabled). The IP will automatically write this snapshot to the chosen memory (either User Flash Memory or External SPI Memory). The trigger signal is rising edge sensitive. For more details on trigger handling and the automatically generated signals associated with this component, see the Fault Log Operation section of this document.

Basic Configuration

This drop-down menu is used to select the number of ASC devices which will be snapshot in the fault log memory. The configuration always includes ASC0 and can be updated to include ASC0 – ASCx (x can be 1 - 7 depending on how many ASCs are in the design) as shown in Figure 7. For each ASC included, the system will snapshot and record the V,I,T signals as well as the ASC GPIO and HVOUT status at the time of Fault Trigger. This configuration setting influences the Fault Record Length (as described in the Fault Logging Memory Maps section.)

Figure 7. Full-Featured Fault Log - Basic Configuration



Include Timestamp

This checkbox will prompt the Fault Logger Component to include a 32-bit timestamp with each fault record logged. This is a 10-year timestamp, with each bit equal to approximately 1.05 seconds. The timer used for the timestamp resets to 0x00000000 each time a Power-On-Reset or device configuration Refresh occurs.

Busy Signal Name

This drop-down menu is used to assign a node (or port) from the design as the Busy signal for the Full-Featured Fault Log. The IP component will assert this signal whenever the Fault Trigger signal transitions from 0 to 1. The IP component will clear this signal after a Fault has been successfully logged to the UFM or external SPI memory. While the Busy signal is high, the Fault Log component will ignore any additional trigger assertions. (See Appendix A. Full-Featured Fault Log: SPI Flash – Execution Flow Charts for more details).

Log to UFM or Log to External SPI Flash

These radio buttons are used to select the memory which will store the Fault Log Records. The component has slightly different behavior depending on the memory selected – see Table 1 for the comparison overview. When External SPI Flash is chosen, the SPI Model selection drop down menu will be enabled. This menu contains the SPI Memory models which have been saved in the Platform Designer library. The option to Create New SPI Model will also be enabled - this option opens a dialog box where the SPI Flash features and command codes are defined. For most applications, the default settings will be acceptable.

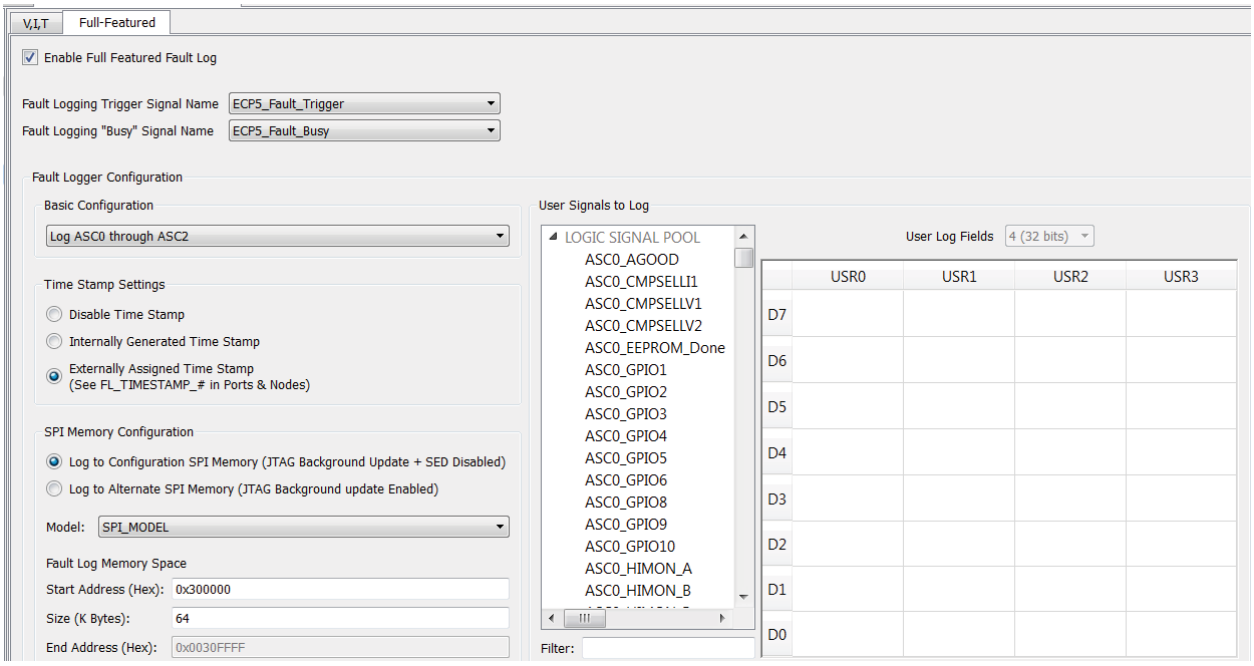
User Signals to Log

This area of the component view is a drag and drop interface for storing user-selected signals in the Fault Log Record. The drop-down User Log Fields allows the selection of none to four 8-bit fields to be configured. This configuration setting influences the Fault Record Length (as described in the Fault Logging Memory Maps section.)

Full-Featured Fault Logger Configuration for ECP5

The ECP5 based Platform Manger 2 designs do not offer the UFM option to log fault records because the ECP5 family does not include a UFM block. Therefore, these designs require the use of an external SPI memory. Platform Designer provides several additional features for Timestamp and SPI Memory Configuration as shown in Figure 8.

Figure 8. Full-Featured Fault Log Configuration for ECP5



Enable Full-Featured Fault Log

This check box enables the Full-Featured fault logging view for configuration. Additionally, this setting instantiates the IP components associated with the Full-Featured fault log automatically in the background.

Fault Logging Trigger Signal Name

This drop-down menu is used to assign a node (or port) from the design as the trigger signal for the Full-Featured Fault Log. Whenever this signal is asserted, the IP component snapshots the current V,I,T status, as well as the selected User Signals and Timestamp (if enabled). The IP automatically writes this snapshot to External SPI Memory (UFM is not available in the ECP5). The trigger signal is rising edge sensitive. For more details on trigger handling and the automatically generated signals associated with this component, see the Fault Log Operation section of this document.

Fault Logging “Busy” Signal Name

This drop-down menu is used to assign a node (or port) from the design as the Busy signal for the Full-Featured Fault Log. The IP component asserts this signal whenever the Fault Trigger signal transitions from 0 to 1. The IP component clears this signal after a Fault is successfully logged to the external SPI memory. While the Busy signal is high, the Fault Log component ignores any additional trigger assertions. (See Appendix A. Full-Featured Fault Log: SPI Flash – Execution Flow Charts for more details).

Basic Configuration

This drop-down menu is used to select the number of ASC devices which will be snapshot in the fault log memory. The configuration always includes ASC0 and can be updated to include ASC0 – ASCx (x can be 1 - 7 depending on how many ASCs are in the design) as shown in Figure 8. For each ASC included, the system snapshots and records the V,I,T signals as well as the ASC GPIO and HVOUT status at the time of Fault Trigger. This configuration setting influences the Fault Record Length (as described in the Fault Logging Memory Maps section.)

Time Stamp Settings

This section is used to configure the Fault Log Timestamp. The following three options are supported; none, internal, or external.

- **Disable Time Stamp** - When this radio button is selected, the Timestamp is not included in the Fault Log Record.
- **Internally Generated Time Stamp** - When this radio button is selected, the Fault Logger Component includes a 32-bit timestamp with each fault record logged. This is a 10-year timestamp, with each bit equal to approximately 1.05 seconds. The timer used for the timestamp resets to 0x00000000 each time a Power-On-Reset or device configuration Refresh occurs.
- **Externally Assigned Time Stamp** - When this radio button is selected, the Fault Logger Component includes a 32-bit signature from an external source with each fault record logged. The signature is obtained from a 32-bit group of Nodes named FL_TIMESTAMP, which is generated by Platform Designer when this option is selected. Using the Logic view, the FL_TIMESTAMP group can be connected to either internal logic (for a different time base) or external hardware such as a real time clock (for example: to capture Hour, Day, Month, Year).

SPI Memory Configuration

This section is used to select the SPI interface for the Fault Logger Component. Two options are provided; one uses the same SPI memory that is used to configure the ECP5, and the other supports a separate SPI memory that is dedicated to Fault Logging.

- **Log to Configuration SPI Memory...** - When this radio button is selected, the Fault Logger Component shares the same physical SPI memory that is used to configure the ECP5. When this option is selected, ECP5 features such as JTAG Background Update and SED are disabled. These features are disabled in order to support Fault Logging. In addition, Platform Designer automatically instantiates and names Ports and Nodes to support the interface to the SPI memory and SPI host, as shown in Figure 9 and Figure 10.

Figure 9. Full-Featured Fault Log Connections for ECP5 Design using Single SPI Memory

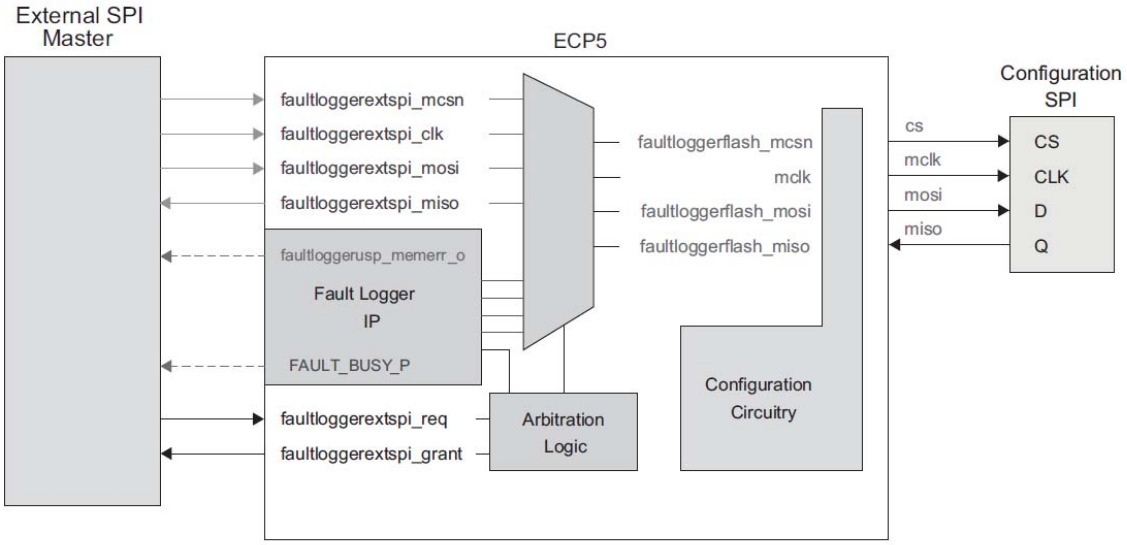
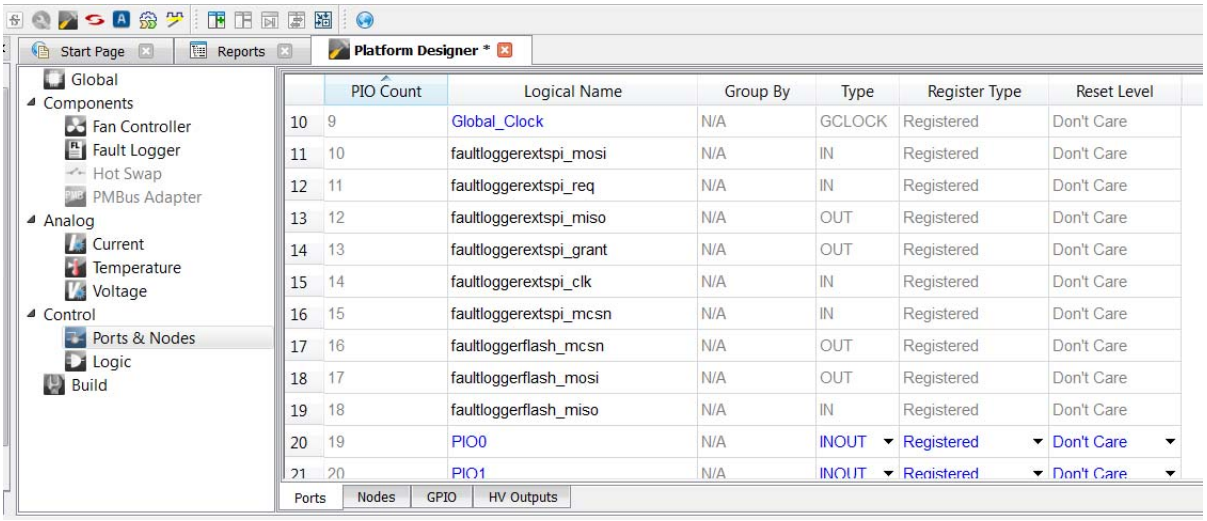


Figure 10. ECP5 Fault Log Single SPI Ports



- **Log to Alternate SPI Memory…** - When this radio button is selected, the Fault Logger Component stores all the Fault Records in an Alternate SPI Memory. In addition, Platform Designer automatically instantiates and name Ports and Nodes to support the interface to the SPI memory and SPI host, as shown in Figure 11 and Figure 12.

Figure 11. Full-Featured Fault Log Connections for ECP5 Design using Two SPI Memories

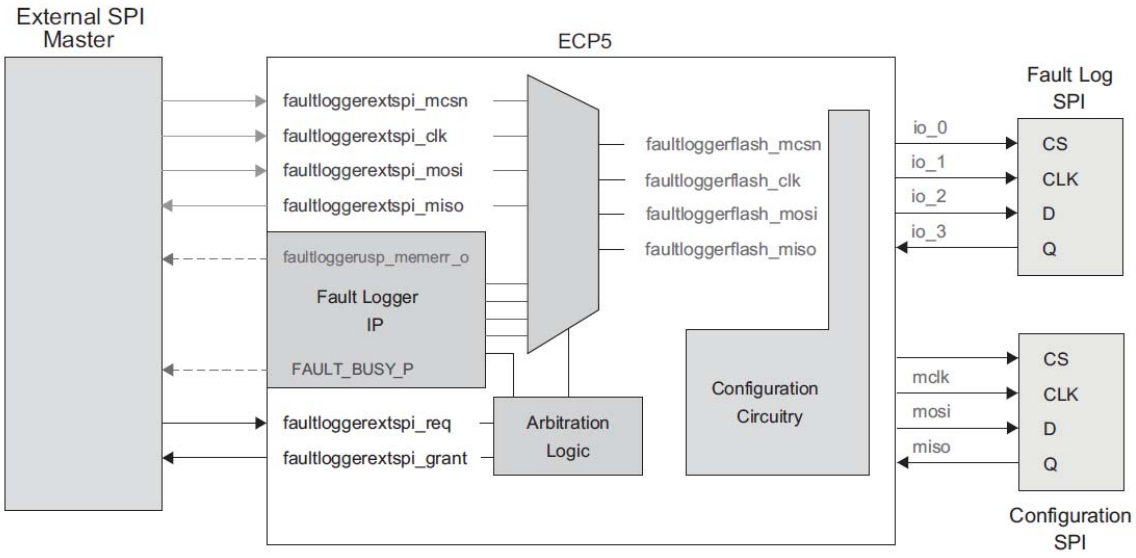
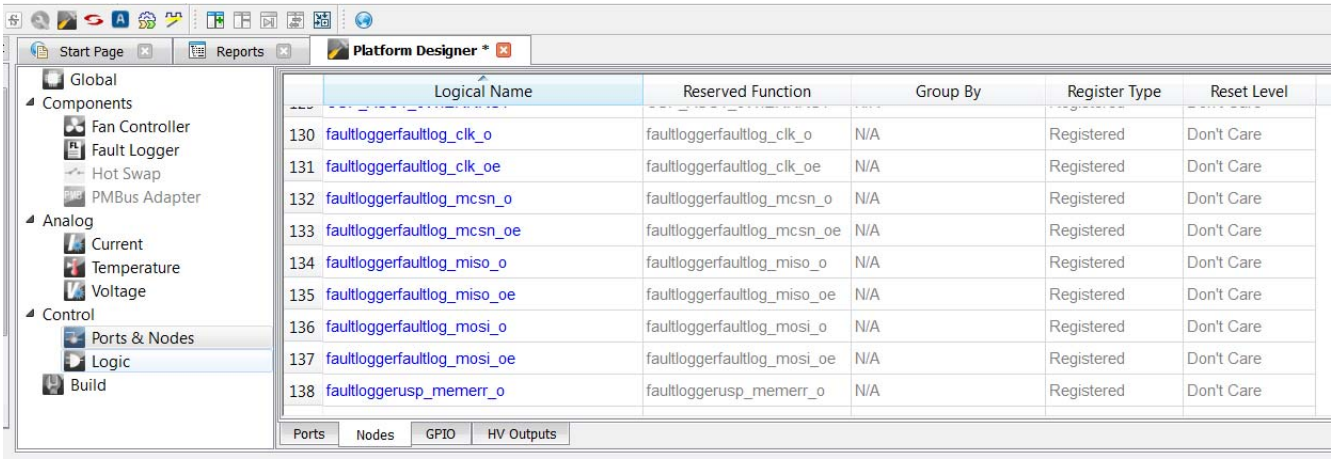


Figure 12. ECP5 Fault Log Dual SPI Nodes



SPI Memory Model

This drop-down list is used to select the type of external SPI used for fault logging. This sets the SPI commands to be used in fault logging. The list includes the SPI models which have been saved in the Platform Designer library and also the default SPI_MODEL_0 and Add / Edit SPI Model…. The Add / Edit SPI Model… can be used to define command codes for the SPI memory used. For most applications, the default settings will be acceptable.

Fault Log Memory Space

Two edit boxes are provided to enter the start address and the space allocated for fault logging.

- **Start Address (Hex)** - Use this edit box to set the Start Address. When logging faults to the Configuration SPI Memory, the Start Address should be set above the configuration data to prevent corrupting the boot image(s); the default value is 0x300000. The Start Address can be set to a higher value if multiple boot images are used or if some of the SPI memory needs to be reserved for other functions. If an Alternate SPI Memory is used to log faults, the Start Address can be set to zero or any value within the memory space.
- **Size (K Bytes)** - Use this edit box to provide the maximum space to be used by the Fault Logger. When you enter the size, the End Address is calculated so you can verify that the fault records will fit.

User Signals to Log

This area of the component view is a drag and drop interface for storing user-selected signals in the Fault Log Record. The drop-down User Log Fields is disabled and set to the maximum number of four for the ECP5.

Fault Logging Memory Maps

The V,I,T Fault Logging Record map and addressing is described in the L-ASC10 device data sheet - in both the fault logging section and the I^2C interface section. The Full-Featured Fault Logging Memory Map is described in detail in the following section.

Faults are recorded in a specific format (Fault Record Map) and have a specific length (Record Length). The Fault Record Map and Record Length are not fixed and depend upon user choices like the number of ASCs, use of Timestamp and number of User Log Fields.

Because the Fault Logging Component offers the designer many options, several examples are provided below to illustrate how the Map and Length may vary. Note that Platform Designer also automatically generates a text file called “Project_Name_fault_log_map.txt”. This file is generated upon Platform Designer compilation and can be found in the Project/Implementation/Project_ptm directory. This file contains the relative address for each Element in the Fault Record as described below. See an example in Appendix B.

Equation 1 can be used for calculating the Fault Log Record Length.

FAULT_LOG_LENGTH= 3 bytes of overhead (Fault Flag, Fault log length, End byte) (1)
+ (7 bytes for ASC0 DATA)
+ (0-49 bytes for ASC1-7 DATA)
+ (0-4 bytes for USER LOGs)
+ (0 or 4 bytes for TIMESTAMP)

The detailed record map for a single ASC (ASC0) and all 4 User Logs and a Timestamp is shown in Table 2 below.

Table 2. Detailed Fault Record for Single ASC with Time Stamp

BYTE	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
0	0	0	1	1	1	1	0	0
1	Size[7]	Size[6]	Size[5]	Size[4]	Size[3]	Size[2]	Size[1]	Size[0]
2	Reserved	Reserved	0	0	1	Reserved	1	Reserved
3	AGOOD	RGPIO10	RGPIO9	RGPIO8	RGPIO7	RGPIO6	RGPIO5	RGPIO4
4	RGPIO3	RGPIO2	RGPIO1	RHVOUT4	RHVOUT3	RHVOUT2	RHVOUT1	IMON_1B
5	IMON_1A	HIMONB	HIMONA	HVMONB	HVMONA	VMON_9B	VMON_9A	VMON_8B
6	VMON_8A	VMON_7B	VMON_7A	VMON_6B	VMON_6A	VMON_5B	VMON_5A	VMON_4B
7	VMON_4A	VMON_3B	VMON_3A	VMON_2B	VMON_2A	VMON_1B	VMON_1A	TMON_2B
8	TMON_2A	TMON_1B	TMON_1A	TMonInt_B	TMonInt_A	1	0	1
9	USR0[7]	USR0[6]	USR0[5]	USR0[4]	USR0[3]	USR0[2]	USR0[1]	USR0[0]
10	USR1[7]	USR1[6]	USR1[5]	USR1[4]	USR1[3]	USR1[2]	USR1[1]	USR1[0]
11	USR2[7]	USR2[6]	USR2[5]	USR2[4]	USR2[3]	USR2[2]	USR2[1]	USR2[0]
12	USR3[7]	USR3[6]	USR3[5]	USR3[4]	USR3[3]	USR3[2]	USR3[1]	USR3[0]
13	Timer[7]	Timer[6]	Timer[5]	Timer[4]	Timer[3]	Timer[2]	Timer[1]	Timer[0]
14	Timer[15]	Timer[14]	Timer[13]	Timer[12]	Timer[11]	Timer[10]	Timer[9]	Timer[8]
15	Timer[23]	Timer[22]	Timer[21]	Timer[20]	Timer[19]	Timer[18]	Timer[17]	Timer[16]
16	Timer[31]	Timer[30]	Timer[29]	Timer[2	Timer[27]	Timer[26]	Timer[25]	Timer[24]
17	0	0	1	0	1	0	1	0

Below are several example configuration along with their associated Record Map and Record Length

Example 1. Minimum Fault Record Length

Configuration to log: 1 ASC, no User Logs and no Timestamp

FAULT_LOG_LENGTH = 3 bytes overhead + 7 bytes for ASC0 = 10 bytes (0x0A)

Table 3. Minimum Fault Record Map

Element No	Name	No. of Bytes	Value
0	FAULTLOG FLAG	1	0x3C
1	FAULTLOG LENGTH	1	0x0A
2	ASC0 DATA	7	
3	FAULTLOG END	1	0x2A

Example 2. Midsize Fault Record Length #1

Configuration to log: 3 ASCs, 2 User Logs and no Timestamp

FAULT_LOG_LENGTH = 3 bytes overhead + 7 bytes for ASC0 + 14 bytes for ASC1 and ASC2 + 2 bytes for User Logs = 26 bytes (0x1A)

Table 4. Midsize Fault Record Map #1

Element No	Name	No. of Bytes	Value
0	FAULTLOG FLAG	1	0x3C
1	FAULTLOG LENGTH	1	0x1A
2	ASC0 DATA	7	
3	ASC1 DATA	7	
4	ASC2 DATA	7	
10	USER0	1	
11	USER1	1	
15	FAULTLOG END	1	0x2A

Example 3. Midsize Fault Record Length #2

Configuration to log: 3 ASCs, 2 User Logs and Timestamp

FAULT_LOG_LENGTH = 3 bytes overhead + 7 bytes for ASC0 + 14 bytes for ASC1 and ASC2 + 2 bytes for User Logs + 4 bytes for Timestamp = 30 bytes (0x1E)

Table 5. Midsize Fault Record Map #2

Element No	Name	No. of Bytes	Value
0	FAULTLOG FLAG	1	0x3C
1	FAULTLOG LENGTH	1	0x1E
2	ASC0 DATA	7	
3	ASC1 DATA	7	
4	ASC2 DATA	7	
10	USER0	1	
11	USER1	1	
14	TIMESTAMP	4	
15	FAULTLOG END	1	0x2A

Example 4. Maximum Fault Record Length

Configuration to log: 8 ASCs, 4 User Logs and Timestamp

FAULT_LOG_LENGTH = 3 bytes overhead + 7 bytes for ASC0 + 49 bytes for ASC1 - ASC7 + 4 bytes for User Logs + 4 bytes for Timestamp = 67 bytes (0x43)

Table 6. Maximum Fault Record Map

Element No	Name	No. of Bytes	Value
0	FAULTLOG FLAG	1	0x3C
1	FAULTLOG LENGTH	1	0x43
2	ASC0 DATA	7	
3	ASC1 DATA	7	
4	ASC2 DATA	7	
5	ASC3 DATA	7	
6	ASC4 DATA	7	
7	ASC5 DATA	7	
8	ASC6 DATA	7	
9	ASC7 DATA	7	
10	USER0	1	
11	USER1	1	
12	USER2	1	
13	USER3	1	
14	TIMESTAMP	4	
15	FAULTLOG END	1	0x2A

Record Addresses in User Flash Memory and SPI Flash

The base address of the Fault Log Records depends on the type of memory storage used. Both memory options start recording at address 0x000000 (the ECP5 based designs start at 0x300000 when using a single SPI memory).

Fault Log Records are written to the UFM in 16-byte pages. If the Fault Log Record size is not an exact multiple of 16 bytes, extra 0x00 bytes are written so that the next Fault Record begins on a page boundary, as shown in Table 8. The Fault_Pages can be calculated from the FAULT_LOG_LENGTH using Equation 2.

$$\begin{aligned}\text{Fault\_Pages} = [\ &\text{FAULT\_LOG\_LENGTH} / 16\ , \\ &+0\ \ \text{If modulus (FAULT\_LOG\_LENGTH/16)} = 0.0\ , \\ &+1\ \ \text{If modulus (FAULT\_LOG\_LENGTH/16)} > 0.0\ ]\end{aligned} \tag{2}$$

The Max_Record_Num will depend on the size of the FPGA and can be calculated from the Max_UFM_Pages using Equation 3 and Table 8.

$$\text{Max\_Record\_Num} = \text{Max\_UFM\_Pages} / \text{Fault\_Pages} \tag{3}$$

Table 7. Max UFM Pages based on Device Size

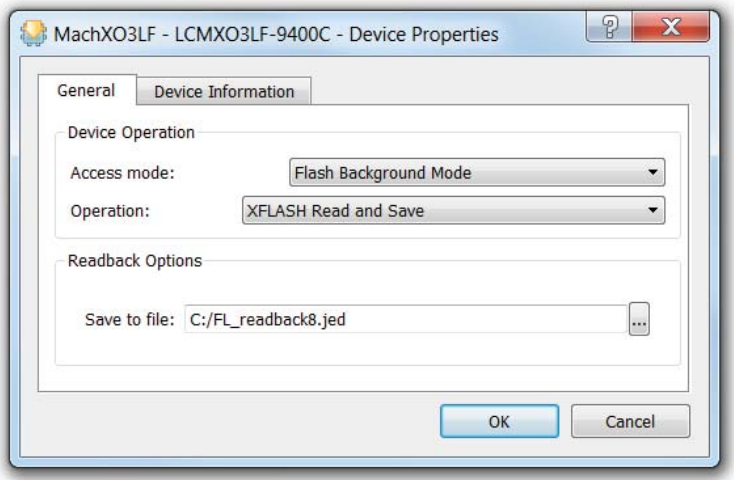
Max_UFM_Pages	UFM - kbits	MachXO2 Device	MachXO3 Device
625	80	–2000	–2100
750	96	–2000U, –4000	–4300
2000	256	–7000	–6900
3500	448	n/a	–9400

Table 8. Fault Record Addressing in UFM

Record Number	Base Record Address
1	0x000000
2	0x000000 + (Fault_Pages * 16)
3	0x000000 + 2 * (Fault_Pages * 16)
4	0x000000 + 3 * (Fault_Pages * 16)
…	…
Max_Record_Num	0x000000 + (Max_Record_Num – 1) * (Fault_Pages * 16)

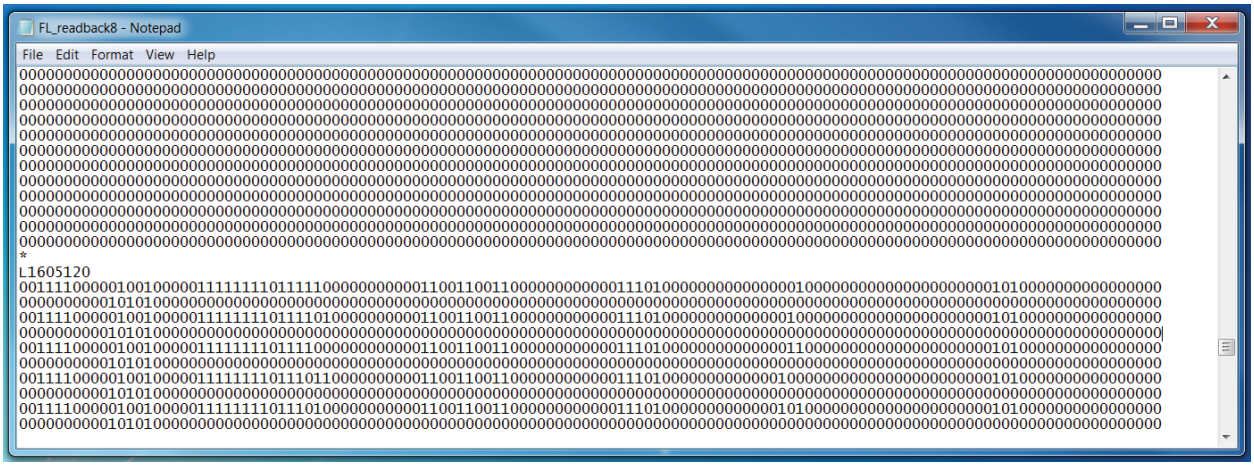
The Fault Records stored in the UFM can be read using an Operation within Diamond Programmer to read and save the Flash memory, as shown in Figure 13. This can be done in background mode to prevent resetting the Platform Manager.

Figure 13. Flash Background Mode Operation to Read and Save



The Read and Save operation saves the entire Flash memory bit by bit in a text based .jed file. A search for the character “L” can be used to find the beginning of the UFM. As shown in the Notepad view of Figure 14, each row is 128 bits long which corresponds to 16 bytes which corresponds to a single UFM page.

Figure 14. Notepad View of FL_readback8.jed File



The first few pages from the UFM shown in Figure 14 are expanded and labeled in Listing 1 to show how a Fault Log Record of 18 bytes is stored.

Listing 1. Four Pages of UFM Fault Records

```
L1605120

Record #1
0x3C Hdr18 bytes  ASC0-0  ASC0-1  ASC0-2  ASC0-3  ASC0-4  ASC0-5
001111000001001000001111111101111011000000000001100110011000000000

 ASC0-6   USR-0   USR-1   USR-2   USR-3 Timer-0 Timer-1 Timer-2
000111010000000000000001000000000000000000000101000000000000000000
End of page 1

 Timer-30x2A End---------- Six Unused bytes --------------------
000000000010101000000000000000000000000000000000000000000000000000

---------------------- Eight Unused bytes --------------------
000000000000000000000000000000000000000000000000000000000000000000
End of page 2


Record #2
001111000001001000001111111101111010000000000110011001100000000000
000111010000000000000001000000000000000000000001010000000000000000
End of page 3

000000000010101000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
End of page 4
```

The fault record storage arrangement in SPI memory is slightly different from the fault record storage arrangement just described for UFM. The fault records are stored in external SPI memory at even 128 byte intervals, up to address 0x10000 (0x310000 in the case of the ECP5), regardless of the Fault Log Length. The Fault Record Base Address for external SPI memory are at fixed locations to ensure that all log writes fall within Page Boundaries, as listed in Table 9. The Fault Logger uses the Byte Write command (see Flow Charts in Appendix A), which will auto increment the storage address of the data bytes shifted in the command. Data bytes must not be shifted in which would cause an increment past a page boundary, or the SPI Memory internal address write controller will wrap-around inside the current page which would corrupt the fault log record.

Table 9. Fault Record Addressing in External SPI Flash

	Record Base Address	
Record Number	**MachXO2, MachXO3LF**	**ECP5**
1	0x000000	0x300000
2	0x000080	0x300080
3	0x000100	0x300100
4	0x000180	0x300180
…	…	…
512	0x00FF80	0x30FF80

Fault Log Operation

Operation of the V,I,T Fault Log is described in the L-ASC10 device data sheet, Fault Logging section. The data sheet describes the behavior of several signals available in the Platform Designer Logic View which only apply to the V,I,T Fault Log - ASCx_Fault_Log_Full and ASCx_Fault_Log_In_Progress. Those signals do not indicate the behavior of the Full-Featured Fault Log component.

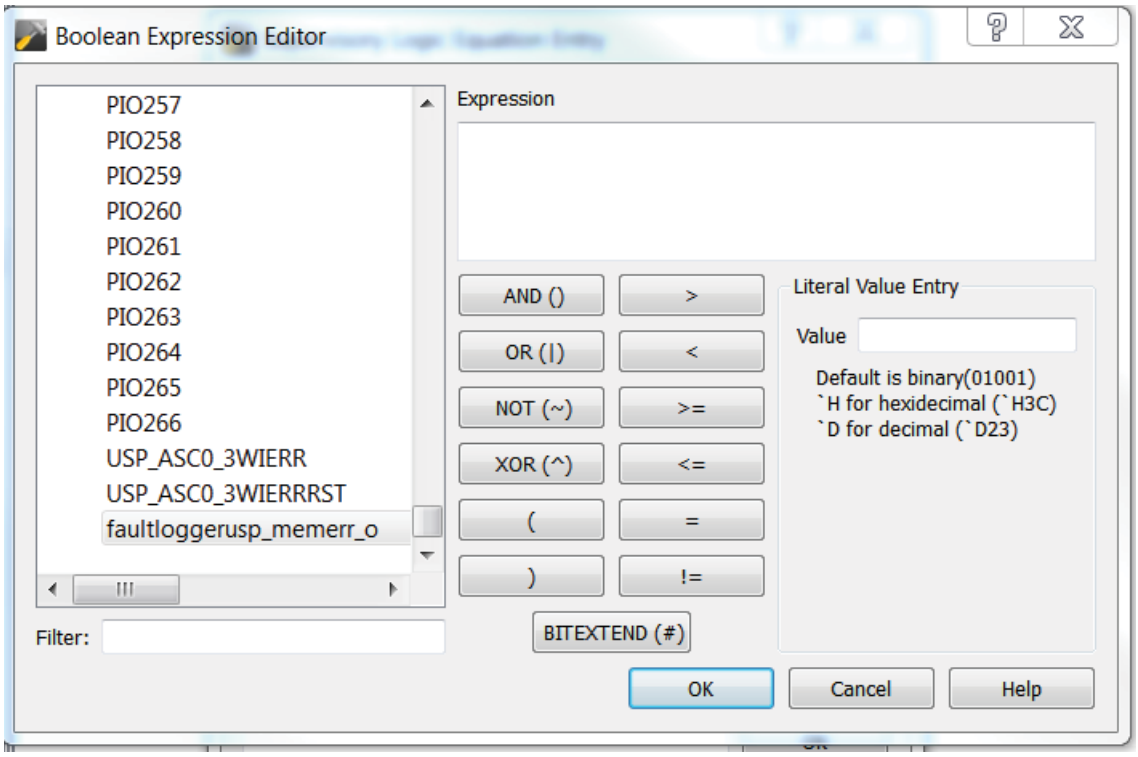
The following section describes the Fault Log operation of the Full-Featured Fault Log IP Component and the associated signals.

The Fault Log component performs two basic operations: Power On Scan and Fault Write. At Power On Scan, the Fault Logger searches the defined memory to initialize the record address pointer for the next write. During the scan, any trigger signal will be latched and written out on completion of the scan. The scan searches the memory, checking valid address locations for the Fault Record Header – 0x3C. 0x3C is written into the first byte of a new fault log to indicate start of a Fault Record. (The Flash memory will have a value of 0xFF in each byte when it is erased.) The memory scan can take up to 150 milliseconds, depending on the number of faults present in the memory at power-on.

After the Power On Scan, the Fault Logger Component will be ready to write new Fault Log Records to the next available empty page of the memory. The system will always write the Fault Log Records in sequential pages of memory. When erasing memory, it is important to not leave erased pages in between valid records otherwise the component will attempt to over-write previous records resulting in corruption of both the new and old data records.

A Fault Write operation is initiated whenever the trigger signal transitions from 0->1, provided the Power On Scan is complete and the Fault_Busy signal is clear. Any triggers which transition while Fault_Busy is asserted will be ignored. The moment a fault is triggered, the Fault Logger will latch the data and set the Fault_Busy signal high. It will then begin the fault log write process. At the conclusion of a fault write, the Fault_Busy signal is cleared, and the Fault Logger component is free to accept new triggers. The External SPI Flash Full-Featured Fault Log component also supports a memory full or memory scan error signal for user feedback. This signal is automatically added as a node into designs which enable the Full-Featured Fault Log External SPI Flash component. The signal is labelled faultloggerusp_memerr_o. Figure 20 below shows the faultloggerusp_memerr_o signal in the Boolean Expression Editor of the logic view. This signal will only be present when the Full-Featured Fault Logger External SPI Flash component is enabled.

Figure 15. Boolean Expression Editor of Logic View



Appendix A. Full-Featured Fault Log: SPI Flash – Execution Flow Charts includes a detailed description of the SPI Flash fault logger execution flow, including the Power On Scan and Fault Write operations.

Reading data from UFM or External Flash

The fault log data stored in the UFM can be read via the JTAG interface (as already shown in Figure 13 and Figure 14), external slave SPI port, or I^2C primary port. For more details on accessing UFM via I^2C refer to TN1246, Using User Flash Memory and Hardened Control Functions in MachXO2 Devices Reference Guide.

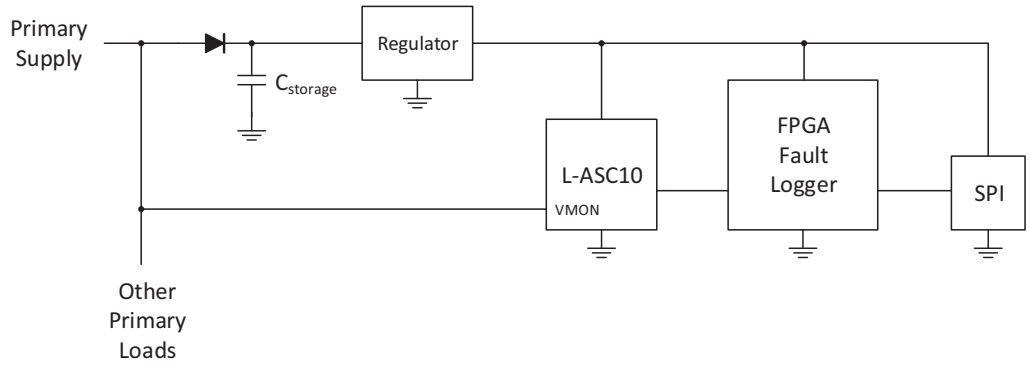
The fault log data stored in the external flash can be read via the SPI interface. In the case where the external memory is connected to the primary SPI port, Diamond Programmer can be used to read and save the Fault Log Records from the JTAG interface.

In both read cases, it is important that fault logging is disabled when attempting to read from the memory. This is left to the system designer to determine the appropriate way to handshake and disable triggering when performing a memory access.

Other Hardware Considerations

Oftentimes, system designers want to capture a fault when a primary supply fails. In such cases, it may become a race to record the fault and store the data before all the supplies shut down. To insure reliable Fault Record storage, it is recommended to add a few additional components to maintain power during a brown-out or power outage. As shown in Figure 16, the extra components consist of a diode, storage capacitor, and a regulator. A VMON can actually be used to sense the failure in a primary supply and the Platform Manager can initiate the shut-down sequence (not shown in Figure 16) at the same time the Fault Log is being stored. The storage capacitor should be sized to hold-up V_{CC} for all the devices associated with the fault detection and storage. The Regulator could be an LDO (low dropout) or a DC/DC converter.

Figure 16. Fault Log Hold-up Circuit



Another hardware consideration is the type of SPI memory used for storing Fault Log Records. Many SPI devices have multiple write protection features. For example, the Cypress S25FLxxxk series (used on Lattice's MachXO3LF-9400 Development Board) has both a global write protect register and a power-up timer. The power-up timer blocks any write operations and lasts about 10 ms after power-up. The global write protect register is usually set from the SPI factory.

The Fault Logger Component performs a write operation to remove the global write protection at power-up. Depending on the system configuration, this clearing of the global write protect register may occur before the SPI power-up timer is complete. In which case, Fault Log Records will not be stored in the SPI. However, the global write protect register can be cleared by the SPI master that is used to read out the Fault Log Records or it can be cleared at board test.

One last consideration is triggering the Fault Logger. It is recommended that a separate Logic Sequence is used to manage the Fault Logger Trigger Signal. To add a blank sequence click on the "+" tab in the Logic Sequence View. Right-click on any of the SMx tabs to bring up the **Multiple State Machine** button. Click on that button to open the Multiple State Machines window to name the state machines; for example SM2:Fault Log.

After naming the State Machine, add logic similar to the example shown in Figure 17. Step 1 waits for the user inserted node **Ready2LogFaults**, which is set in SM0 after the main power sequencing is complete. This prevents premature fault records as supplies are coming on line. Step 2 arms the trigger to logging faults; it waits for either of two supplies to be out of range (note: additional conditions could be added to this step). Step 3 sets the actual trigger signal to start the fault logging process by setting **Log_Faults_Node** true. It is highly recommended that the arming and trigger steps be adjacent in the design to minimize the time delay from event to logging the fault record; as can be seen in the following timing diagrams.

Figure 17. State Machine SM2 Fault Log

eports | Platform Designer *

Step	Sequence Instruction	Outputs	Interruptible	Comment
SM2 Step 0	Begin Startup Sequence		no	
SM2 Step 1	Wait for Ready2LogFaults or wait for 70ms using Timer4 If Timeout Then Goto 6	Log_Faults_Node = 0	no	If System is not powered up in 70ms log a fault and shut down.
SM2 Step 2	Wait for NOT ASC0_VM1_3v3_OK OR NOT ASC0_VM2_2v5_OK	Log_Faults_Node = 0	no	If either supply is out of range trigger a fault
SM2 Step 3	Log_Faults_Node = 1		no	
SM2 Step 4	Wait for NOT Fault_Log_Busy_Node		no	Wait for logging record to complete
SM2 Step 5	Go to step 2		no	
SM2 Step 6	Log_Faults_Node = 1		no	
SM2 Step 7	Begin Shutdown Sequence		no	
SM2 Step 8	Halt (end-of-program)		no	

Exception ID	Boolean Expression	Outputs	Exception Handler	Comment
	<end-of-exception-table>			

SM0 | SM1 | SM2:Fault Log | +

Figure 18. Fault Log Timing with 16 µs Glitch Filter, Logging to SPI

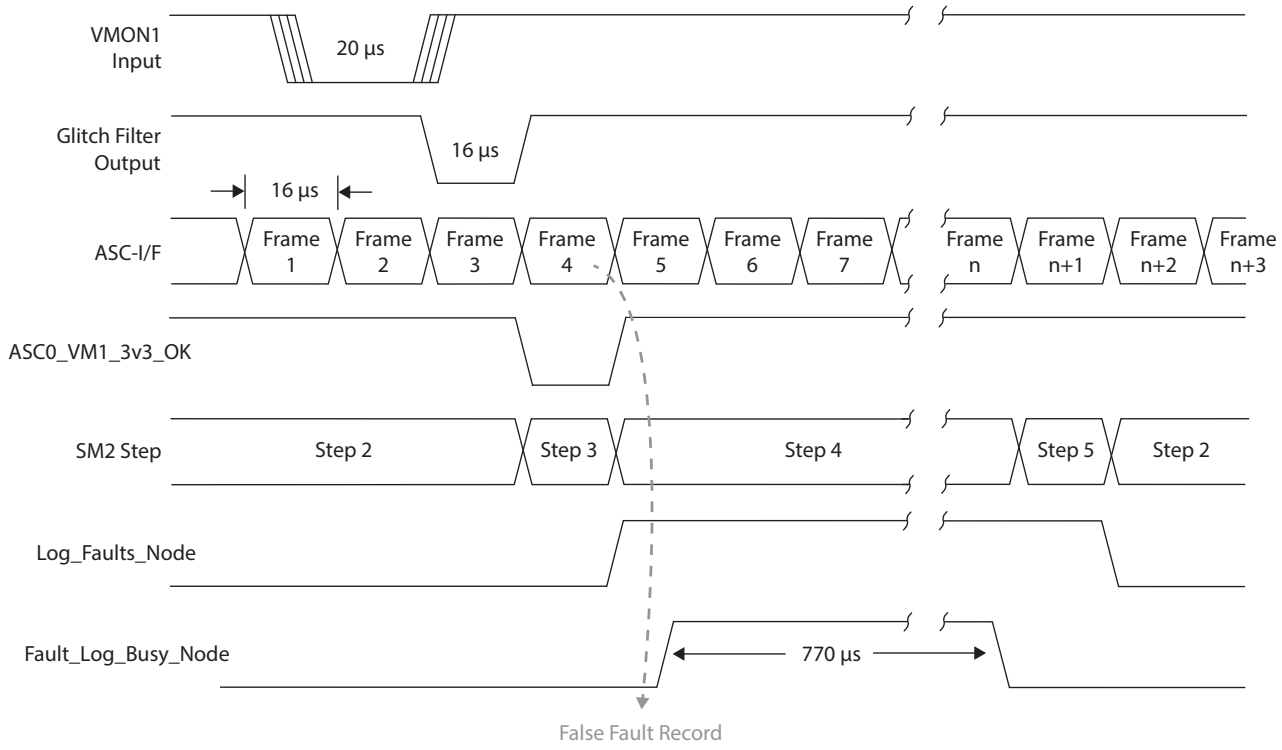
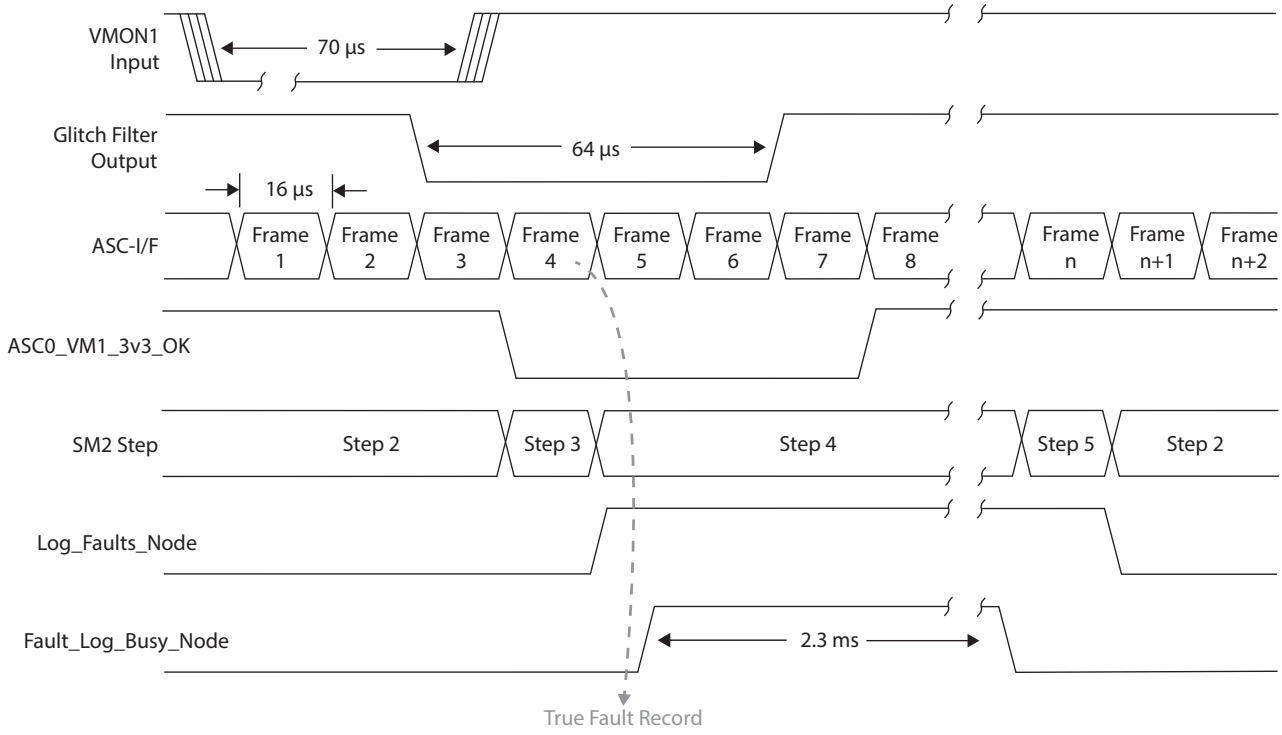


Figure 19. Fault Log Timing with 64 µs Glitch Filter, Logging to UFM



In both Figure 18 and Figure 19 the Fault Log Trigger signal is activated in Sequence Step 3 and the data for the Fault Log Record is captured from ASC-I/F Frame 4. Setting the Glitch Filter (in the Voltage view) to the minimum setting (16 µs) will provide the shortest delay from event to fault logging. However, it can also be the source of false fault records if the event is too short, as shown in Figure 18. Note that the output of the Glitch Filter is synchronized to the ASC-I/F.

Alternatively, setting the Glitch Filter to the maximum setting (64 µs) will add about 48us delay to logging the fault and only faults of 64 µs or more will trigger the fault logger. However, once triggered the output of the Glitch Filter maintains the event for an accurate fault record, as shown in Figure 19.

Summary

The Hardware Management Controllers based on Platform Manager 2 designs provide multiple options for logging faults during system operation. Platform Manager 2 designs can be built using the following Lattice devices: LPTM21 alone or with L-ASC10(s), MachXO2 with L-ASC10(s), MachXO3LF with L-ASC10(s), and ECP5 with L-ASC10(s). The Fault Logger GUI component of the Platform Designer tool provides the designer an easy and flexible tool to configure the Fault Logging operation. The Fault Record details and operation of the Full-Featured Fault Logging component have been described in detail in this document.

Related Literature

- DS1042, L-ASC10 Data Sheet
- DS1043, Platform Manager 2 Family Data Sheet
- TN1246, Using User Flash Memory and Hardened Control Functions in MachXO2 Devices Reference Guide
- TN1260, ECP5 and ECP5-5G sysCONFIG Usage Guide
- TN1293, Using Hardened Control Functions in MachXO3 Devices
- Platform Designer Software Tutorial

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Date	Version	Change Summary
January 2018	2.1	— Added reference to MachXO3LF and ECP5 — Added Full-Featured Fault Logger Configuration for ECP5 section. — Added information to the Reading data from UFM or External Flash section. — Added Other Hardware Considerations section. — Updated Appendix C. Full-Featured Fault Log: Component Updates vs Diamond Revisions. Created sub-sections and added Diamond 3.11 Improvements. — Other minor edits
February 2016	2.0	Comprehensive update to all sections.
March 2015	1.1	Removed LPTM20 references.
December 2013	01.0	Initial release.

Appendix A. Full-Featured Fault Log: SPI Flash – Execution Flow Charts

Figure 20. External SPI Flash - Power-Up Scan (Part 1)

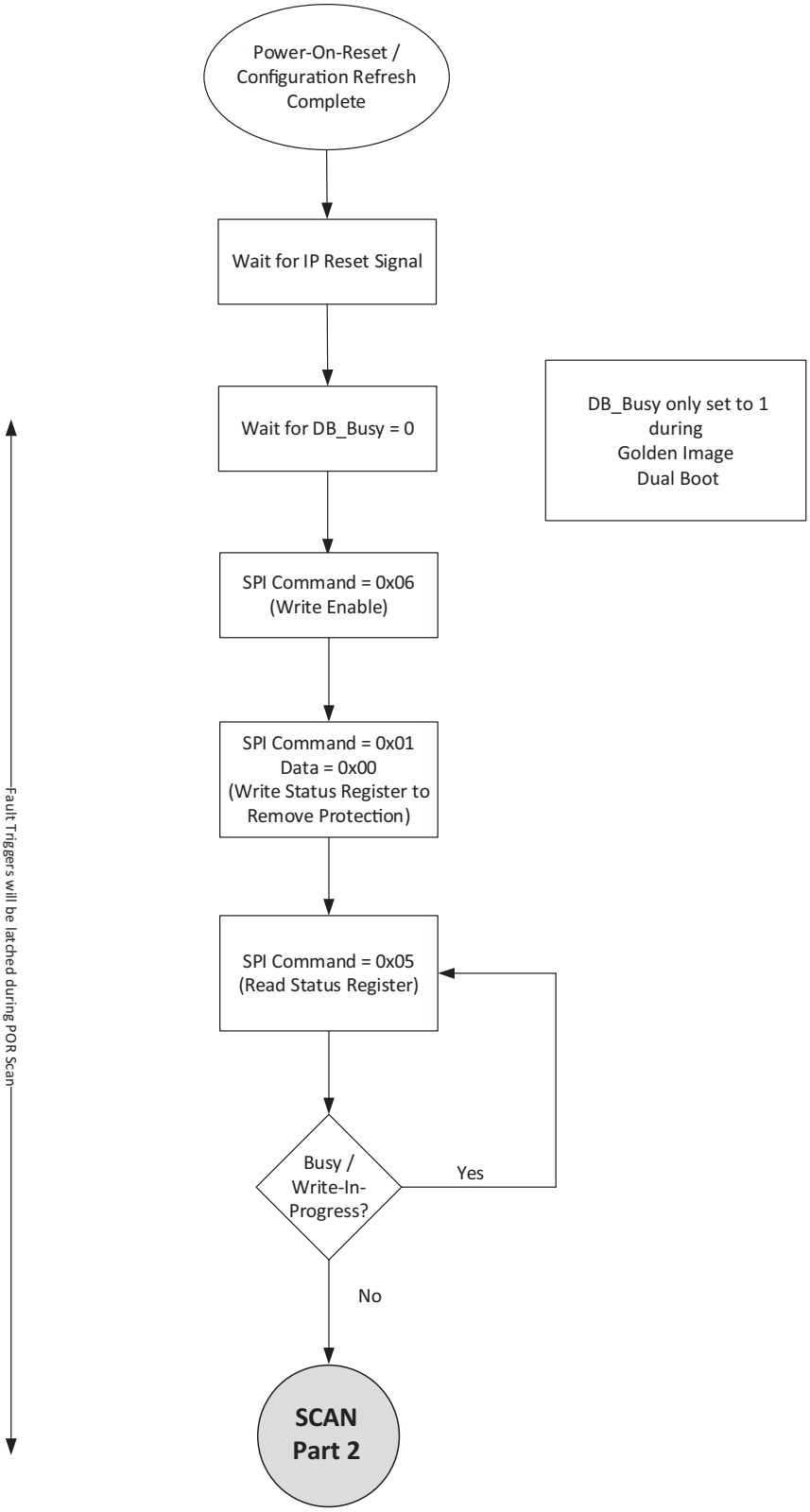


Figure 21. External SPI Flash - Scan Operation - Part 2

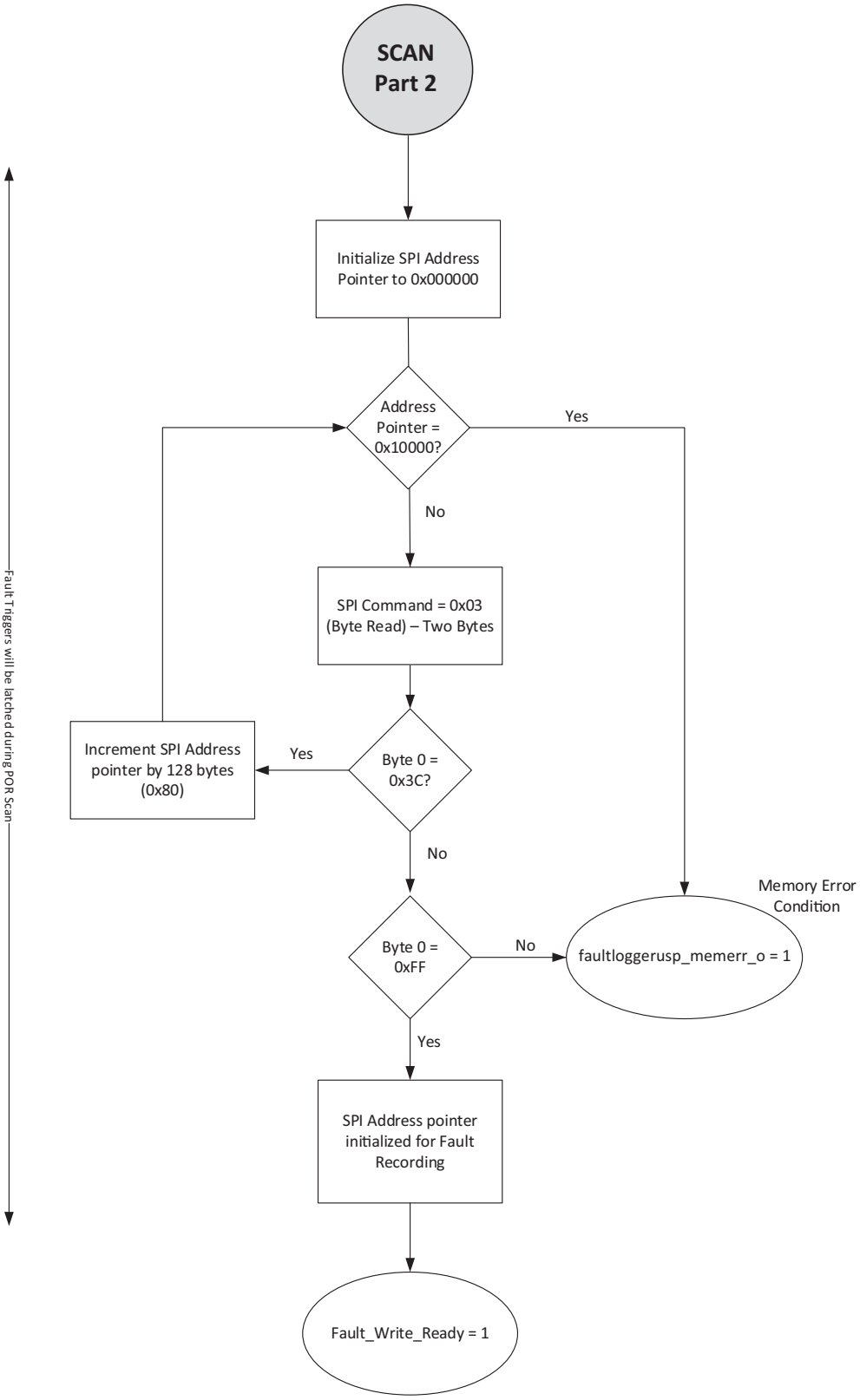


Figure 22. External SPI Flash - Write Operation - Part 1

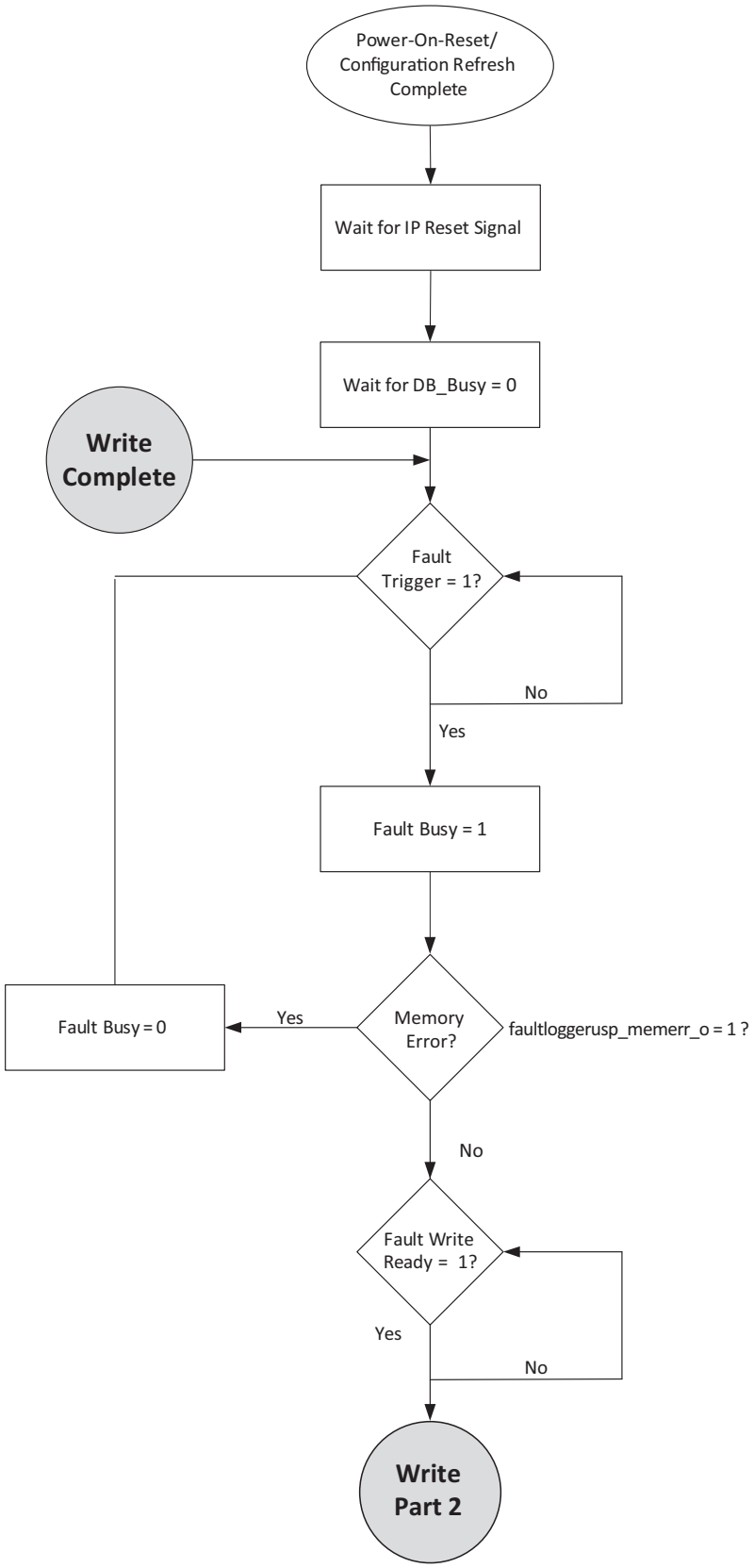


Figure 23. External SPI Flash - Write Operation - Part 2

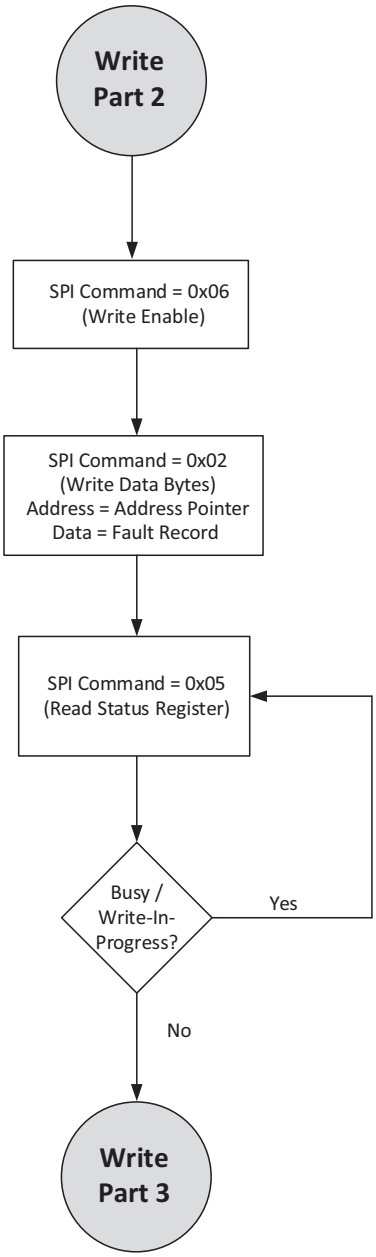
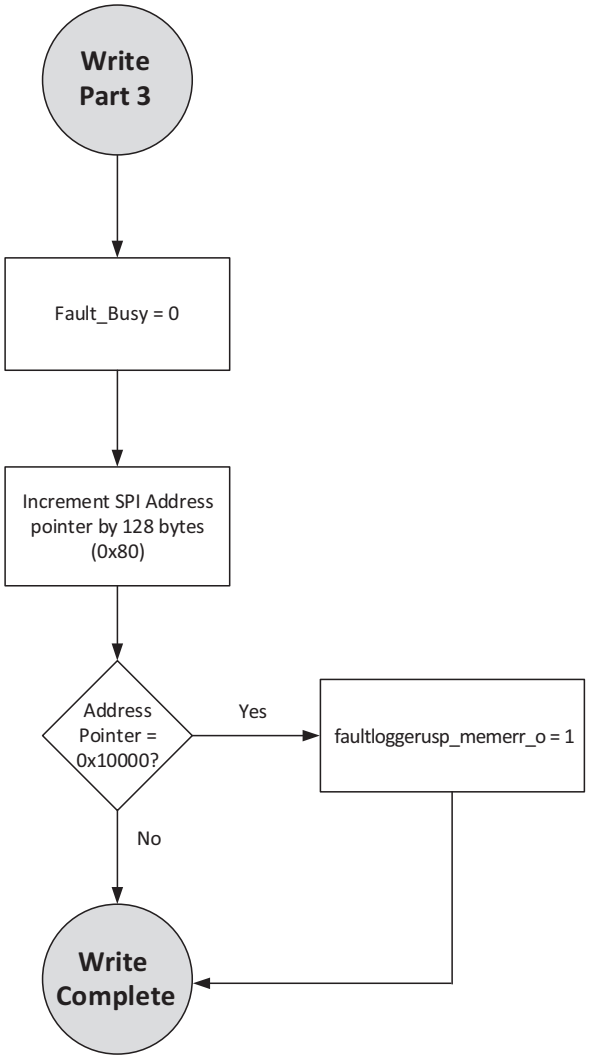


Figure 24. External SPI Flash Fault Log - Write Operation - Part 3



Appendix B. Full-Featured Fault Log: Fault Log Map Text File

Below is an example of the Platform Designer generated *my_design_fault_log_map.txt* file, which is written into the \my_project\implementation\my_design_ptm folder. This corresponds to Example 3 in Table 5 above.

Relative Address	Element	Length (byte)
0x00	FAULTLOG FLAG(0x3C)	1
0x01	FAULTLOG LENGTH(0x17)	1
0x02	ASC0	7
0x09	ASC1	7
0x10	ASC2	7
0x17	USR0	1
0x18	USR1	1
0x19	TIMESTAMP	4
0x1D	FAULTLOG END(0x2A)	1

Appendix C. Full-Featured Fault Log: Component Updates vs Diamond Revisions

The Full-Featured Fault Log Component behavior has been improved across several versions of Lattice Diamond Software. The updates are listed below:

Diamond 3.6 Improvements

- Memory Space Overrun Protection: Prior to Diamond 3.6 release version, the Fault Logger component (SPI Memory only) in general release versions of Diamond did not stop Fault Logging at the memory boundary reserved for Fault Log (Addresses 0x000000 – 0x010000). This may result in corruption of the Golden Configuration Image stored in external SPI Flash memory.
- Page Boundary Protection: Prior to Diamond 3.6 release version, the Fault Logger component (SPI Memory only) in general release versions of Diamond did not protect against “Page Wraparound” behavior of external SPI Flash. The fault logger uses a sequential byte program command (0x02) to write data to the external SPI Flash. Standard SPI memories will wrap-around and begin to write byte 0 of a page if the page boundary is reached during byte program. For example: Fault Logger writes 11 bytes, starting at address 0x7D. The first two bytes will be stored at addresses 0x7E, 0x7F. The next 9 bytes will be stored in addresses 0x00-0x08.

  To correct this behavior, starting in Diamond 3.6, the Fault Logger (SPI version) only writes records starting at 128 byte intervals as described above.
- Global Unprotect at Power-On: Prior to Diamond 3.6 release version, the Fault Logger component in general release versions of Diamond did not include a Global Unprotect command during the power-on scan. Diamond 3.6 includes this command as part of the Power-On. See execution flow above.
- Global Unprotect before Every Write Log Removed: In the Diamond 3.6 release version, the Fault Logger (SPI component) executed a Write Status Register command prior to every log write. In Diamond 3.7 release version and later, this behavior was removed.

Diamond 3.7 Improvements

- Add Write-In-Progress / Busy Check after Status Register Write & Log Write: Prior to the Diamond 3.7 release version, the Fault Logger (SPI component) did not check the Status Register Write-In-Progress bit after Status Register Writes or Memory Writes. Diamond 3.7 and later versions include this check as shown in the execution flow above.
- Update Fault Busy Behavior: Prior to the Diamond 3.7 release version, the Fault Logger (SPI component) asserted the Fault Busy signal upon receiving a trigger, and cleared the signal upon start of SPI transaction. In Diamond 3.7, the Fault Busy signal is held high until the completion of a fault log write as indicated by the Write-In-Progress or Busy bit in the SPI Flash Status Register being cleared.
- Update Fault Log Full Behavior: Prior to the Diamond 3.7 release version, the Fault Logger (SPI component) provided no special indication that the available memory space was full with logs. In Diamond 3.6 and earlier, the Fault Busy signal would be asserted high indefinitely when a fault was triggered to a full memory. Diamond 3.7 and later versions include the Fault Log Full indicator (node signal labeled faultloggerusp_memerr_o) and the Fault Busy signal is cleared shortly after a trigger is asserted when the memory is full.

Software Patch Improvements

Note: At the time of this document revision, Diamond 3.10 was just released. The following improvements are available in software patches for Diamond 3.9 and Diamond 3.10.

- Update Fault Log Full Behavior: Prior to the software patch, the Fault Logger (UFM component) would write Fault Records past the end of the UFM space and eventually wrap around. The software patch is limited to the size of the UFM as listed in Table 7. Also, the Fault Log Full indicator (node signal labeled faultloggerusp_memerr_o) is active when the last record that will fit in the UFM is written.
- Update Fault Log Full set at Power-On: Prior to the software patch, the Fault Logger (SPI component) would scan the SPI memory at Power-On to set the address pointer for the next record. If the pointer was on the last record, a fault would have to be triggered for the Fault Log Full indicator to be set. The software patch sets the Fault Log Full indicator at Power-On if the last record is already written into the SPI memory.
- Update Address Pointer Reset: Prior to the software patch, the Fault Logger (UFM component) would need to be reset after the UFM is erased in order to reset the Fault Log Record address pointer. The Power-On scan of the UFM would set the address pointer to zero if the UFM is erased or just past the last Fault Log Record. The software patch now performs a quick check before writing a Fault Log Record; if the UFM is erased then the address pointer is reset. Otherwise, the Fault Log Record is written and the pointer is adjusted. This supports reading and erasing UFM Fault Log Records without having to reset the system.