LatticeMico Fault Logger

The LatticeMico Fault Logger IP allows the user's design to store Analog Sense and Control (ASC) RDAT (read data) to FLASH based upon user defined trigger criteria. It does this by continuously buffering RDAT information in FPGA memory, then sending that information to FLASH when the user defined trigger is asserted.

The Lattice IP offers two different ways to do fault logging:

- Smaller designs and fewer FPGA resources can use the fault logging function that is embedded on the ASC device.
- Larger designs or need features such as timestamp or user data logging can use the fault logger IP described here.

For more information refer to the Platform Designer documentation in Diamond online help.

Version

This document describes the 1.1 version of the LatticeMico Fault Logger.

Features

The LatticeMico Fault Logger IP must be used together with the Embedded Functional Block (EFB) of the MachXO2/Platform Manager 2 device, and has the following features to support the fault logging task

- RDAT frame buffering for any number of ASC's.

- Writes fault log snapshot to either MachXO2/Platform Manager 2 embedded User Flash Memory (UFM) or to an external FLASH memory.
- User defined trigger (to start the write-to-FLASH operation).
- Maskable interrupt, set by the user defined trigger.
- WISHBONE slave interface to LatticeMico8 microcontroller.
- Storage of FLASH related opcodes.
- Optional storage of up to four bytes of user defined data.
- Optional time stamp.
- All data is register-mapped and accessible by the LatticeMico8 microcontroller from the WISHBONE bus.
- Configurable to use either EBR or distributed RAM.

Functional Description

Refer to Figure 1 on page 3. The fault logger IP continuously buffers RDAT frames into FPGA memory during normal operation. When it receives a trigger from the user logic (typically Logibuilder logic), the fault logger IP will suspend RDAT frame buffering and asserts a LatticeMico8 microcontroller interrupt. The LatticeMico8 microcontroller will then read the fault log snapshot via WISHBONE byte reads and write the data to FLASH. When the entire fault log is stored to FLASH, the LatticeMico8 microcontroller will clear the fault log IP interrupt and the fault logger IP will resume normal operation.

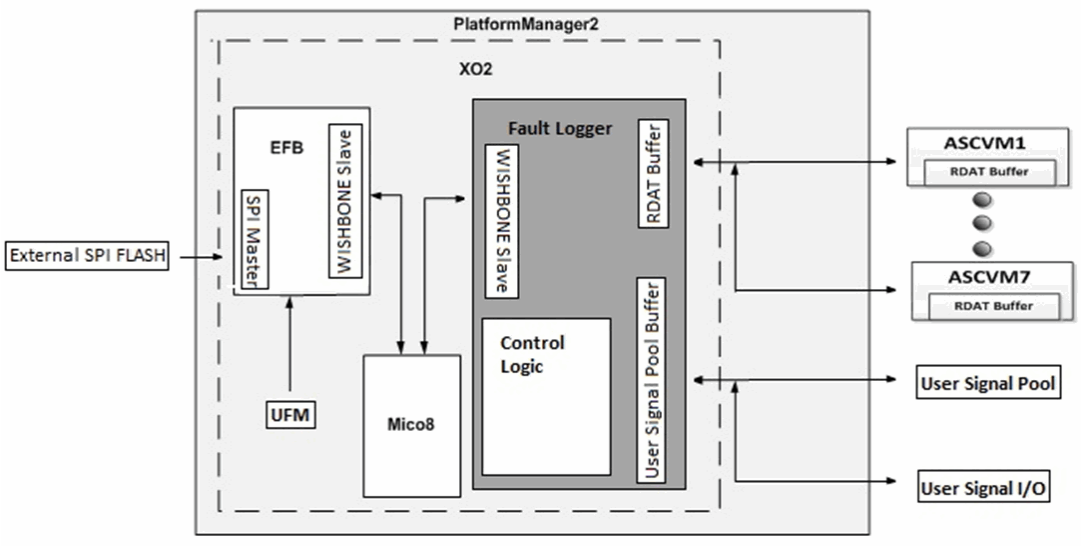
The fault logger can be configured to buffer between one and eight RDAT frames at a time, depending on user choice. A new RDAT frame is presented to the fault logger IP once every 16 µs by the ASCVM. The RDAT data for all logged ASC's is presented to the fault logger at the same time, but on separate interfaces. Malformed RDAT frames (CRC errors) are filtered out by the ASCVM and are not presented to the fault logger.

Fault Log Triggering

The fault log trigger is an input from the user signal pool and should be connected by the user. A rising edge on this signal (usp_trigger_i) constitutes a "trigger event". This will suspend all data buffering and will cause a fault logger interrupt to be sent to the LatticeMico8 microcontroller. The fault logger IP will also assert a busy signal output (usp_busy_o) to indicate that the fault log is being stored to FLASH and that it will ignore all trigger events until the LatticeMico8 microcontroller clears the fault logger IP interrupt.

The designer can select which user signal pool node will be connected to the trigger signal. The signal should be a registered node to keep it synchronous to the Fault Logger IP synchronous logic.

Figure 1: Fault Logger Block Diagram



User Log Storage

Users can optionally store up to 4 bytes of user signal pool data in the fault log. These fault logger inputs buffered into FPGA memory once every 4 µs and will be included as part of the fault log written to FLASH. Buffering of these signals is suspended while usp_busy_o is asserted.

The designer can enable user log bytes and to select which design signals will be connected to the user log inputs.

The user log bytes are WISHBONE register readable and are included as part of the fault log written to FLASH. Refer to “Fault Log Mapping Table” on page 4 to see where these signals reside in a fault log map. Refer to “ASC Fault Log Record Memory Map” on page 5 when ASC is configured for Fault Log Mode. More information about how to configure the user log bytes refer to Platform Designer documentation in Lattice Diamond online help.

Timestamp Function

The user may optionally include a timestamp function that logs the number of seconds since the last reset of the MachXO2/Platform Manager 2. The timestamp counts up to 18 years and will not roll over. It is WISHBONE register readable and is included as part of the fault log written to FLASH.

Please see the Fault Log Mapping Table (generated by Platform Designer) to see where the timestamp resides in a fault log map, or see Platform Designer for information about how to enable the timestamp option.

Please see “Configuration Parameters” for more information about the fault log timestamp.

Fault Log Mapping Table

Table 1 is the Fault Log Map. The length of the fault log will vary with differing user configurations. For example, a fault log that logs three ASC’s will be shorter than a fault log that includes eight ASC’s. The location of similar data within a fault log will also change with differing configurations. For example, the address of the timestamp will be at a lower address location in a fault log with two ASC’s than if the fault log includes seven ASC’s.

Platform Designer provides an ASCII based, Fault Log Mapping Table file to tell the user where individual RDAT bits and user log and timestamp bytes are for a given Fault Logger IP configuration. Please refer to the Platform Designer documentation in the Diamond online help for the usage and location of this automatically generated document.

Table 1: Fault Log Map

Element Number	Name	Number of Bytes	Required/ Optional	Description
0	HEADER	1	Required	The fault flag is the first byte of FLASH storage map. 0x3C in this location indicates that there is a fault log snapshot stored in this FLASH page. This flag is not stored in the fault logger IP, but is inserted by LatticeMico8 microcontroller.
1	LENGTH	1	Required	Total number of bytes in the storage element map. This value will be calculated by software at startup and then stored in parameter FAULT_LOG_LENGTH.
2	ASC0 RDAT	7	Required	This is the RDAT frame from ASC0. The contents of this element will be the same contents and ordering of the data within a 3WI RDAT frame.
3	ASC1 RDAT	7	Optional.	Same as ASC0 RDATA element, except that inclusion depends on parameter FAULTLOGGER_ASC_LOG_CNT >= 2.
4	ASC2 RDAT	7	Optional.	Same as ASC0 RDATA element, except that inclusion depends parameter FAULTLOGGER_ASC_LOG_CNT >= 3.
5	ASC3 RDAT	7	Optional.	Same as ASC0 RDATA element, except that inclusion depends parameter FAULTLOGGER_ASC_LOG_CNT >= 4.
6	ASC4 RDAT	7	Optional.	Same as ASC0 RDATA element, except that inclusion depends parameter FAULTLOGGER_ASC_LOG_CNT >= 5.
7	ASC5 RDAT	7	Optional.	Same as ASC0 RDATA element, except that inclusion depends parameter FAULTLOGGER_ASC_LOG_CNT >= 6.
8	ASC6 RDAT	7	Optional.	Same as ASC0 RDATA element, except that inclusion depends parameter FAULTLOGGER_ASC_LOG_CNT >= 7.
9	ASC7 RDAT	7	Optional.	Same as ASC0 RDATA element, except that inclusion depends parameter FAULTLOGGER_ASC_LOG_CNT >= 8.

Table 1: Fault Log Map (Continued)

Element Number	Name	Number of Bytes	Required/ Optional	Description
10	USER0	1	Optional.	First byte of user log data and corresponds to signal USP_USRLOG0_I. Included if parameter FAULTLOGGER_USER_LOG_CNT>=1.
11	USER1	1	Optional.	Second byte of user log data and corresponds to signal USP_USRLOG1_I. Included if parameter FAULTLOGGER_USER_LOG_CNT>=2.
12	USER2	1	Optional.	Third byte of user log data and corresponds to signal USP_USRLOG2_I Included if parameter FAULTLOGGER_USER_LOG_CNT>=3.
13	USER3	1	Optional.	Fourth byte of user log data and corresponds to signal USP_USRLOG3_I Included if parameter FAULTLOGGER_USER_LOG_CNT>=4.
14	TIME	4	Optional	Timestamp. Indicates distance since last power on reset. Only included if parameter FAULTLOGGER_TIMESTAMP_ENABLED=1.
15	FOOTER	1	Required	End delimiter. Value is always 0x2A. Not stored in fault logger IP. This flag is not stored in the fault logger IP, but is inserted by LatticeMico8 microcontroller.

ASC Fault Log Record Memory Map

Table 2 is the ASC fault log record memory map. When the ASC is configured for Fault Log Mode, the memory block is used to record the status of the ASC GPIOs, VMON, IMON, TMON and other significant logic signals on the occurrence of the user defined fault trigger condition.

Each fault record has seven bytes: six bytes of ASC specific data and one byte of user specified FPGA signals.

Table 2: ASC Fault Log Record Memory Map

Byte	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	User bit7	User bit6	User bit5	User bit4	User bit3	User bit2	User bit1	User bit0
1	AGOOD	GPIO10	GPIO9	GPIO8	X	GPIO6	GPIO5	GPIO4
2	GPIO3	GPIO2	GPIO1	HVOUT4	HVOUT3	HVOUT2	HVOUT1	IMON1B
3	IMON1A	HIMONB	HIMONA	HVMONB	HVMONA	VMON9B	VMON9A	VMON8B
4	VMON8A	VMON7B	VMON7A	VMON6B	VMON6A	VMON5B	VMON5A	VMON4B
5	VMON4A	VMON3B	VMON3A	VMON2B	VMON2A	VMON1B	VMON1A	TMON2B
6	TMON2A	TMON1B	TMON1A	TMONINB	TMONINA	1	0	1

Configuration

The following sections describe the graphical user interface (UI) parameters, the hardware description language (HDL) parameters, and the I/O ports that user can use to configure and operate the LatticeMico Fault Logger. For more information refer to the Platform Designer documentation in Diamond online help.

UI Parameters

Table 3 shows the UI parameters available for configuring the LatticeMico Fault Logger through the Mico System Builder (MSB) interface.

Table 3: Fault Logger UI Parameters

Dialog Box Options	Description	Allowable Values	Default Value
Instance Name	Specifies the name of the Fault Logger instance.	Alphanumeric and underscores	faultlogger
Base Address	Specifies the base address for configuring the Fault Logger. The minimum boundary alignment is 0x80.	0X80000000–0XFFFFFFFF	0X80000000
Fault Logger Setting			
Time Stamp Enable	When selected, Time Stamp is enabled	selected | not selected	selected
Number of ASC Log	Specifies the number of ASC Log	1 – 8	1
Number of User Log	Specifies the number of User Log	0 – 4	0
RDAT Storage Setting			
Distributed RAM	When selected, RDAT data will be stored in Distributed RAM	selected | not selected	selected
EBR	When selected, RDAT data will be stored in EBR	selected | not selected	not selected
Flash Memory Setting			
Flash Mode			
Memory Storage	Specifies the number of bytes in the external Flash Storage	0- 268435456	100
User Flash Memory	When selected, fault log data will be stored in User Flash Memory	selected | not selected	selected
External SPI Flash	When selected, fault log data will be stored in external SPI Flash	selected | not selected	not selected

Table 3: Fault Logger UI Parameters (Continued)

Dialog Box Options	Description	Allowable Values	Default Value
SPI Chip Select Number	Specifies which chip select line is connected to external SPI Flash.	0 - 7	0
SPI Flash Command			
Read	Specifies the opcode for external SPI Flash Read (READ)	0 - 255	3
Write	Specifies the opcode for external SPI Flash Write (PP)	0 - 255	2
Write Enable	Specifies the opcode for external SPI Flash Write Enable (WREN)	0 - 255	6
Write Disable	Specifies the opcode for external SPI Flash Write Disable (WRDI)	0 - 255	4
Read Status	Specifies the opcode for external SPI Flash Read Status (RDSR)	0 - 255	5
Write Status	Specifies the opcode for external SPI Flash Write Status (WRSR)	0 - 255	1

HDL Parameters

Table 4 lists the parameters that appear in the HDL.

Table 4: Fault Logger HDL Parameters

Parameter Name	Description	Allowable Values
LATTICE_FAMILY	Define the device family for the IP	MACHXO2 | LPTM2
FAULTLOGGER_ASC0_ASC1012	A value of 1 defines the ASC0 is configured as ASC1220, otherwise, ASC0 is configured as ASC1012	0 | 1
FAULTLOGGER_ASC_LOG_CNT	Define the total number of ASC Log	1 to 8
FAULTLOGGER_USER_LOG_CNT	Define the total number of User Log	0 to 4
FAULTLOGGER_TIMESTAMP_ENABLED	A value of 1 defines the Time Stamp is enabled	0 | 1
FAULTLOGGER_STORAGE_DISTRAM	A value of 1 defines the Distributed Ram is used for storage, otherwise, EBR is used for storage	0 | 1
Control Register Parameters		
FAULTLOGGER_SPIFLASH_READ_CMD	Define the opcode for SPI Flash Read (READ)	0 to 255
FAULTLOGGER_SPIFLASH_WRITE_CMD	Define the opcode for SPI Flash Write (PP)	0 to 255
FAULTLOGGER_SPIFLASH_WREN_CMD	Define the opcode for SPI Flash Write Enable (WREN)	0 to 255

Table 4: Fault Logger HDL Parameters (Continued)

Parameter Name	Description	Allowable Values
FAULTLOGGER_SPIFLASH_WRDI_CMD	Define the opcode for SPI Flash Write Disable (WRDI)	0 to 255
FAULTLOGGER_SPIFLASH_RDSR_CMD	Define the opcode for SPI Flash Read Status (RDSR)	0 to 255
FAULTLOGGER_SPIFLASH_WRSR_CMD	Define the opcode for SPI Flash Write Status (WRSR)	0 to 255
FAULTLOGGER_SPI_CHIP_SELECT	Define the SPI Flash chip select line	0 to 7
FAULTLOGGER_LOG_CNT	Define the number of bytes in the UFM or SPI Flash	0 to 268435456

I/O Ports

Table 5 describes the input and output ports of the LatticeMico Fault Logger.

Table 5: Fault Logger I/O Ports

I/O Port	Direction	Active	Description
System Clock and Reset			
clk_wishbone	I	—	WISHBONE System Clock
clk_3wi	I	—	Logibuilder Clock used to enable loading of the user log signals from the user signal pool buffers.
resetn	I	Low	System Reset
WISHBONE Slave Signal			
WBS_CYC_I	I	High	Indicates a valid bus cycle is present on the bus.
WBS_STB_I	I	High	Asserts an acknowledgment in response to the assertion of the WISHBONE Master strobe.
WBS_WE_I	I	—	Level sensitive Write/Read control signal. Low - Read operation, High - Write operation
WBS_ADR_I	I	—	32-bit wide address used to select a specific register
WBS_DAT_I	I	—	8-bit data used to read a byte of data from a specific register

Table 5: Fault Logger I/O Ports (Continued)

I/O Port	Direction	Active	Description
WBS_CTI_I	I	—	Not used, always tied to 0
WBS_BTE_I	I	—	Not used, always tied to 0
WBS_LOCK_I	I	—	Not used, always tied to 0
WBS_SEL_I	I	—	Not used, always tied to 0
WBS_DAT_O	O	—	8-bit data used to read a byte of data from a specific register
WBS_ACK_O	O	High	Indicates the requested transfer is acknowledged.
WBS_ERR_O	O	—	Indicates the address is incorrect
WBS_RTY_O	O	—	Not used, always tied to 0
User Signal Pool Ports			
usp_trigger_i	I	Low to High	This input triggers storage of all signals to FLASH memory. A transition from 0 to 1 constitutes a trigger
usp_db_busy_i	I	High	Dual-boot busy signal
usp_busy_o	O	High	This output indicates that the fault logger is currently having its fault log contents being stored to FLASH. While this busy signal is asserted, all fault log triggers will be ignored
usp_usrlog0_i	I	—	User log signal byte 0. This port will only be used if USR_LOG_CNT>=1.
usp_usrlog1_i	I	—	User log signal byte 1. This port will only be used if USR_LOG_CNT>=2.
usp_usrlog2_i	I	—	User log signal byte 2. This port will only be used if USR_LOG_CNT>=3.
usp_usrlog3_i	I	—	User log signal byte 3. This port will only be used if USR_LOG_CNT=4.
ASCVM Interface Ports			
rdat0_data_i	I	—	This is the RDAT data from ASC0. This port will be connected to ASCVM output fl_asc0_data.

Table 5: Fault Logger I/O Ports (Continued)

I/O Port	Direction	Active	Description
rdat0_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc0_valid.
rdat1_data_i	I	—	This is the RDAT data from ASC1. This port will be connected to ASCVM output fl_asc1_data. This port will only be used if ASC1 is logged
rdat1_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc1_valid. This port will only be used if ASC1 is logged
rdat2_data_i	I	—	This is the RDAT data from ASC2. This port will be connected to ASCVM output fl_asc2_data. This port will only be used if ASC2 is logged
rdat2_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc2_valid. This port will only be used if ASC2 is logged
rdat3_data_i	I	—	This is the RDAT data from ASC3. This port will be connected to ASCVM output fl_asc3_data. This port will only be used if ASC3 is logged
rdat3_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc3_valid. This port will only be used if ASC3 is logged
rdat4_data_i	I	—	This is the RDAT data from ASC4. This port will be connected to ASCVM output fl_asc4_data. This port will only be used if ASC4 is logged
rdat4_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc4_valid. This port will only be used if ASC4 is logged
rdat5_data_i	I	—	This is the RDAT data from ASC5. This port will be connected to ASCVM output fl_asc5_data. This port will only be used if ASC5 is logged
rdat5_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc5_valid. This port will only be used if ASC5 is logged
rdat6_data_i	I	—	This is the RDAT data from ASC6. This port will be connected to ASCVM output fl_asc6_data. This port will only be used if ASC6 is logged
rdat6_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc6_valid. This port will only be used if ASC6 is logged
rdat7_data_i	I	—	This is the RDAT data from ASC7. This port will be connected to ASCVM output fl_asc7_data. This port will only be used if ASC7 is logged
rdat7_valid_i	I	High	This is the RDAT valid bit for the data. This port will be connected to ASCVM output fl_asc7_valid. This port will only be used if ASC7 is logged

Table 5: Fault Logger I/O Ports (Continued)

I/O Port	Direction	Active	Description
Other signals			
fl_irq_o	O	High	Interrupt from fault logger IP to LatticeMico8 microcontroller. This signal will go high when the fault logger has received a fault log trigger. This interrupt is sticky (and will stay high until the interrupt is cleared by the LatticeMico8 microcontroller)

Register Descriptions

The LatticeMico Fault Logger WISHBONE module has a register map to allow the service of the hardened functions through the WISHBONE bus interface read/write operations. Table 6 through Table 8 describe the register map of the Fault Logger module.

Table 6: WISHBONE Addressable Registers for Fault Logger Module

Register Name	Register Function	Address	Access
CONTENT	Holds the information of the Fault Log	0x00 - 0x59	Read
IRQ	Interrupt Request Register	0x60	Read/Write
SPI_RD_CMD	Holds the opcode for SPI Flash Read (READ)	0x61	Read
SPI_WR_CMD	Holds the opcode for SPI Flash Write (PP)	0x62	Read
SPI_WREN_CMD	Holds the opcode for SPI Flash Write Enable (WREN)	0x63	Read
SPI_WRDI_CMD	Holds the opcode for SPI Flash Write Disable (WRDI)	0x64	Read
SPI_RDSR_CMD	Holds the opcode for SPI Flash Read Status (RDSR)	0x65	Read
SPI_WRSR_CMD	Holds the opcode for SPI Flash Write Status (WRSR)	0x66	Read
SPI_CHP_SEL	Holds the chip select line of the SPI Flash	0x67	Read
LOG_CNT	Indicates data capacity (in bytes) of the fault log flash	0x70 - 0x73	Read/Write

Table 7 and Table 8 provide details about each register in the LatticeMico Fault Logger.

Fault Log Content Register Definition – CONTENT

The WISHBONE host has Read-Only access to these registers. The fault log actual length of the fault log will vary, but the MachXO2/Platform Manager 2

can use up to 0x45 (dec 69). Addresses 0x46-0x59 are reserved for future expansion of the fault log. For details on the byte per byte contents of the fault log contents, please see the “Fault Log Map” on page 4.

Interrupt Request Register Definition – IRQ

The WISHBONE host has Read and Write access to these registers.

Table 7: CHx_INFO Register Bit Definition

Bit	Field	Description	Access
0	IRQ	Interrupt Bit	Read
1	IRQCLR	Interrupt Clear Bit	Read/Write
2	IRQEN	Fault Logger I Interrupt Enable	Read/Write
3	MEMFULL	Indicate the flash is full	Read/Write
7:4	RSVD	Reserved Bit	Read/Write

SPI Flash Command Register Definition

SPI Flash Command Registers are 8-bit registers, each register holds a specific SPI Flash command and correlates to a corresponding Verilog parameter. The WISHBONE host has Read-Only access to these registers.

Table 8: SPI Flash Command Register Definition

Register Name	Corresponding Verilog Parameter	Address	Access
SPI_RD_CMD	FAULTLOGGER_SPIFLASH_READ_CMD	0x61	Read
SPI_WR_CMD	FAULTLOGGER_SPIFLASH_WRITE_CMD	0x62	Read
SPI_WREN_CMD	FAULTLOGGER_SPIFLASH_WREN_CMD	0x63	Read
SPI_WRDI_CMD	FAULTLOGGER_SPIFLASH_WRDI_CMD	0x64	Read
SPI_RDSR_CMD	FAULTLOGGER_SPIFLASH_RDSR_CMD	0x65	Read
SPI_WRSR_CMD	FAULTLOGGER_SPIFLASH_WRSR_CMD	0x66	Read

SPI Flash Chip Select Register Definition - SPI_CHP_SEL

SPI_CHP_CMD is a 8-bit register that specifics which chip select line (LatticeMico EFB SPI Master) is connected to fault logging SPI flash. This register correlates to Verilog parameter FAULTLOGGER_SPI_CHIP_SELECT.The WISHBONE host has Read-Only access to these registers.

Fault Log Data Capacity Register Definition – LOG_CNT

LOG_CNT is a 32-bit registers that indicates data capacity (in bytes) of the fault log FLASH. This field is context dependent: it can show the max storable data in either the UFM or external SPI FLASH. This register correlates to Verilog parameter FAULTLOGGER_LOG_CNT. The WISHBONE host has Read and Write access to this register.

LatticeMico8 Microcontroller Software Support

This section describes the LatticeMico8 microcontroller software support provided for the LatticeMico Fault Logger component.

Device Driver

The Fault Logger device driver interacts directly with the Fault Logger instance. This section describes the limitations, type definitions, structure, and functions of the Fault Logger device driver.

Type Definitions

This section describes the type definitions for the Fault Logger device context structure. This structure, shown in Figure 2, contains the Fault Logger component instance-specific information and is dynamically generated in the DDStructs.h header file. This information is largely filled in by the managed build process by extracting the Fault Logger component-specific information from the platform specification file. As part of the managed build process, designers can choose to control the size of the generated structure, and hence the software executable, by selectively enabling some of the elements in this structure via C preprocessor macro definitions. These C preprocessor macro definitions are explained later in this document. You should not

manipulate the members directly, because this structure is for exclusive use by the device driver. Table 9 describes the parameters of the Fault Logger device context structure shown in Figure 2.

Device Context Structure

Figure 2 shows the Fault Logger device context structure.

Figure 2: Fault Logger Device Context Structure

```
struct st_MicoFLCtx_t {
    const char *   name;
    size_t   base;
    unsigned char   intrLevel;
    unsigned char   mem_mode;
    unsigned long   curr_addr;
    void *   p_efb;
} MicoFLCtx_t;
```

Table 9 describes the Fault Logger device context parameters.

Table 9: Fault Logger Device Context Parameters

Parameter	Data Type	Description
name	const char*	Fault Logger instance name (entered in MSB)
base	size_t	MSB-assigned base address for this instance
intrLevel	unsigned char	Processor interrupt line to which this instance is connected
mem_mode	unsigned char	This value specific the current Flash Memory Mode (UFM/SPI Flash)
curr_addr	unsigned long	This value specific the current Flash Memory Address
max_addr	unsigned long	This value specific the maximum flash storage
p_efb	void*	This value points to the EFB instance used by Fault Logger

C Preprocessor Macro Definitions

This section describes the C preprocessor macro definitions that are available to the software developer. There are two types of macro definitions: 'object-like' and 'function-like'.

The 'object-like' macro definitions do not take any arguments and are used to control the size of the generated application executable. There are three ways an 'object-like' macro definition can be used by the software developer.

1. Manually adding the -D<*macro name*> option to the compiler's command line in the application's 'Build Properties'. Refer to the LatticeMico8 Developer User Guide for more information on how to manually add the macro definition in the application's 'Build Properties' GUI.

2. Automatically adding the -D<*macro name*> option to the compiler's command-line in the application's 'Build Properties' by enabling the 'check-box' associated with the macro definition. Refer to the LatticeMico8 Developer User Guide for more information on how to set up the check/ uncheck the macro definitions in the application's 'Build Properties' GUI.
3. Manually adding the macro definition to the C code using the following syntax:

```
#define <macro name>
```

It is recommended that the developer use option 1 or 2.

- __MICOFL_NO_SPI_INIT_VALIDATION__

  This preprocessor macro definition disables code and data structures within the device driver that disable the LatticeMico8 EFB SPI module in the software driver and application. In order words, LatticeMico8 assumes the connected SPI Flash is NOT shared with other SPI Master, and does not check whether the SPI is occupied by other SPI Master or not during the power cycle. It is not defined by default.

- __MICOFL_USER_IRQ_HANDLER__

This preprocessor macro definition disables code and data structures within the device driver that allow the user to define the custom interrupt routine, the default routine will be disabled. It is not defined by default.

Table 10: C Preprocessor Function-like Macros For Fault Logger

Macro Name	Second Argument to Macro / Third Argument to Macro (if exist.	Description
MICO_FL_READ_CONTENT	The 8-bit value reads from the Fault Log content / address offset	This macro reads a character from the Fault Log content register with a specific address offset
MICO_FL_READ_IRQ	The 8-bit value reads from the Interrupt register.	This macro reads a character from the Interrupt register.
MICO_FL_WRITE_IRQ	The 8-bit value reads from the Interrupt register.	This macro reads a character to the Interrupt register.
MICO_FL_READ_SPI_RD_CMD	The 8-bit value reads from the SPI Flash Read Command register.	This macro reads a character to the SPI Flash Read Command register
MICO_FL_READ_SPI_WR_CMD	The 8-bit value reads from the SPI Flash Write Command register.	This macro reads a character to the SPI Flash Write Command register
MICO_FL_READ_SPI_WREN_CMD	The 8-bit value reads from the SPI Flash Write Enable Command register.	This macro reads a character to the SPI Flash Write Enable Command register
MICO_FL_READ_SPI_WRDI_CMD	The 8-bit value reads from the SPI Flash Write Disable Command register.	This macro reads a character to the SPI Flash Write Disable Command register

Table 10: C Preprocessor Function-like Macros For Fault Logger (Continued)

Macro Name	Second Argument to Macro / Third Argument to Macro (if exist.	Description
MICO_FL_READ_SPI_RDSR_CMD	The 8-bit value reads from the SPI Flash Read Status Command register.	This macro reads a character to the SPI Flash Read Status Command register
MICO_FL_READ_SPI_WRSR_CMD	The 8-bit value reads from the SPI Flash Write Status Command register.	This macro reads a character to the SPI Flash Write Status Command register
MICO_FL_READ_SPI_CHP_SEL	The 8-bit value reads from the SPI Flash Chip Select register.	This macro reads a character to the SPI Flash Chip Select register
MICO_FL_READ_LOG_CNT_OFFSET	The 8-bit value reads from Fault Log Data Capacity register / address offset	This macro reads a character from the Fault Log Data Capacity register with a specific address offset
MICO_FL_WRITE_LOG_CNT_OFFSET	The 8-bit value writes to Fault Log Data Capacity register / address offset	This macro writes a character to the Fault Log Data Capacity register with a specific address offset

Note: The first argument to the macro is the Fault Logger address.

Functions

This section describes the implemented device-driver-specific functions.

MicoFLInit Function

```
void MicoFLInit (MicoFLCtx_t *ctx);
```

This is the Fault Logger initialization function.

Table 11 describes the parameter in the MicoFLInit function syntax

Table 11: MicoFLInit Function Parameter

Parameter	Description
MicoFLCtx_t	Pointer to a valid MicoFLCtx _t structure representing a valid Fault Logger instance.

MicoFLRegisterEFB Function

```
void MicoFLRegisterEFB (MicoFLCtx_t *ctx,
                        MicoEFBCtx_t *p_efb);
```

This function registers an EFB instance into the Fault Logger instance. This EFB will be used for the communication between Fault Logger control and the UFM or external SPI Flash.

Table 12 describes the parameters in the MicoFLRegisterEFB function syntax.

Table 12: MicoFLRegisterEFB Function Parameters

Parameter	Description
MicoFLCtx_t	Pointer to a valid MicoFLCtx_t structure representing a valid Fault Logger instance.
MicoEFBCtx_t	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.

MicoFL_UFMValidation Function

```
void MicoFL_UFMValidation(MicoFLCtx_t *ctx);
```

This function validate whether the User Flash memory (UFM) contains a valid Fault Log and check for the next available empty slot for the next entry. This function should be called only once during the system power-up. Error code will return when the validation process is failed.

Table 13 describes the parameter in the MicoFL_UFMValidation function syntax.

Table 13: MicoFL_UFMValidation Function Parameter

Parameter	Description
MicoFLCtx_t	Pointer to a valid MicoFLCtx _t structure representing a valid Fault Logger instance.

Table 14 describes the values returned by the MicoFL_UFMValidation Function.

Table 14: Values Returned by the MicoFL_UFMValidation Function

Parameter	Description
0	Valid log file
-1	Invalid log file
-2	Unexpected return

MicoFL_SPIValidation Function

```
void MicoFL_SPIValidation(MicoFLCtx_t *ctx);
```

This function validate whether the external SPI memory contains a valid Fault Log and check for the next available empty slot for the next entry. This function should be called only once during the system power-up. Error code will return when the validation process is failed.

Table 15 describes the parameter in the MicoFL_SPIValidation function syntax.

Table 15: MicoFL_SPIValidation Function Parameter

Parameter	Description
MicoFLCtx_t	Pointer to a valid MicoFLCtx _t structure representing a valid Fault Logger instance.

Table 16 describes the values returned by the MicoFL_SPIValidation Function.

Table 16: Values Returned by the MicoFL_SPIValidation Function

Parameter	Description
0	Valid log file
-1	Invalid log file
-2	Unexpected return

MicoFL_WriteUFM Function

```
void MicoFL_WriteUFM (MicoFLCtx_t *ctx);
```

This function record the fault logger snapshot to the User Flash Memory (UFM). Error code will return when the write process is failed.

Table 17 describes the parameter in the MicoFL_WriteUFM function syntax.

Table 17: MicoFL_WriteUFM Function Parameter

Parameter	Description
MicoFLCtx_t	Pointer to a valid MicoFLCtx _t structure representing a valid Fault Logger instance.

Table 18 describes the values returned by the MicoFL_WriteUFM Function.

Table 18: Values Returned by the MicoFL_WriteUFM Function

Parameter	Description
0	Valid log file
-1	Write to an invalid memory (Not enough space)

MicoFL_WriteSPI Function

```
void MicoFL_WriteSPI (MicoFLCtx_t *ctx);
```

This function record the fault logger snapshot to the external SPI Flash memory. Error code will return when the write process is failed.

Table 19 describes the parameter in the MicoFL_WriteSPI function syntax.

Table 19: MicoFL_WriteUFM Function Parameter

Parameter	Description
MicoFLCtx_t	Pointer to a valid MicoFLCtx _t structure representing a valid Fault Logger instance.

Table 20 describes the values returned by the MicoFL_WriteSPI Function.

Table 20: Values Returned by the MicoFL_WriteSPI Function

Parameter	Description
0	Write successfully
-1	Write to an invalid memory (not enough space),

MicoFLISR Function

```
void MicoFLISR (MicoFLCtx_t *ctx);
```

This function is the Fault Logger Interrupt handler. Each Fault Logger instance has it's own default interrupt handler implementation. If the developer wishes to use his own interrupt handler, he must define the __MICOFL_USER_IRQ_HANDLER__ preprocessor define.

Table 21 describes the values returned by the MicoFLISR Function.

Table 21: MicoFL_WriteUFM Function Parameter

Parameter	Description
MicoFLCtx_t	Pointer to a valid MicoFLCtx _t structure representing a valid Fault Logger instance.

Software Usage Example

This section provides an example of using the Fault Logger. The example is shown in Figure 3 and assumes the presence of a Fault Logger component named “faultlogger”, and a EFB component named “efb”.

Figure 3: Fault Logger Software Example

```
#include "MicoUtils.h"
#include "DDStructs.h"
#include "MicoEFB.h"
#include "MicoFL.h"

int main(void){
MicoFLCtx_t *fl = &faultlogger_faultlogger;
  MicoEFBCtx_t *efb = &efb_efb;

  size_t fl_address = (size_t)(fl->base);

  // Register the EFB instance into Fault Logger
  MicoFLRegisterEFB(fl, efb);

  // Validate the Fault Logger Flash Memory
  MicoFL_SPIValidation(fl);

  #ifdef __MICO_NO_INTERRUPTS__
  do{
      MICO_FL_READ_IRQ(fl_address, status);
    if (status & MICO_FL_IRQ_TRI){
      MicoFL_WriteSPI(fl);
    }
  }while(1);
  #else
  MICO_FL_WRITE_IRQ(fl_address ,MICO_FL_IRQ_EN);
  #endif

  return(0);
}
```

Revision History

Component Version	Description
1.0	Initial release.
1.1	Fixed Fault Logger masked-out issue that occurred when Dual-Boot is busy. Added section “ASC Fault Log Record Memory Map” on page 5.

Trademarks