LatticeMico EFB

The LatticeMico EFB Is a hard architectural block that is known as the Embedded Function Block (EFB). The EFB includes a Serial Peripheral Interface (SPI), two I^2Cs, and a timer/counter peripheral. All of these hard IP peripherals are contained in the EFB block, will connect to the WISHBONE bus in slave mode, and share a common Serial Communications Interface (SCI) block.

Support for LatticeMico EFB is provided for MachXO2, Platform Manager 2, and MachXO3L devices.

Note

Some differences exist between EFB for MachXO2/Platform Manager 2 and EFB for MachXO3L. This document will indicate wherever there is a difference between MachXO2/Platform Manger 2 and MachXO3L.

MachXO2/Platform Manger 2 only: The User Flash Memory (UFM) and phase-locked loopse (PLLs) may also be addressed as WISHBONE elements via the EFB, but they are not physically located in the EFB.

Version

This document describes the 1.6 version of the LatticeMico EFB.

Features

The EFB includes a SPI, two I^{2}C's, and a timer/counter peripheral. MachXO2/ Platform Manager 2 also includes User Flash Memory (UFM).

SPI Features

SPI provides standard, fully configurable SPI ports including:

- Configurable master and slave mode
- Mode fault error flag with CPU interrupt capability
- Double-buffered data register
- Serial clock with programmable polarity and phase
- LSB First or MSB First data transfer

I^{2}C Features

MachXO2/Platform Manager and MachXO3L devices contain two hardened I^{2}C IP cores designated as the “Primary” and “Secondary” ^{2}C IP core. Either of the two cores can be configured as an I^{2}C master or as an I^{2}C slave. The difference between the two cores is that the primary core has pre-assigned I/ O pins, while the ports of the secondary core can be assigned by designers to any general purpose I/O. In addition, the primary core has access to the configuration logic of the MachXO2 device.

When an I^{2}C core is a master, it can control other devices on the I^{2}C bus through the physical interface. When a core is the slave, the device can provide I/O expansion to an I^{2}C master. The cores support the following I^{2}C functionality:

- Master/Slave mode support
- 7-bit and 10-bit addressing
- Clock stretching
- Supports 50KHz, 100KHz, and 400KHz data transfer speed
- General call support
- Interface to custom logic through 8-bit WISHBONE interface

Timer/Counter Features

The timer counter has four modes of operation:

- Clear Timer on Compare (CTC) match. (This mode includes the normal mode.)
- Watchdog

- Fast pulse-width modulation (PWM)
- Phase and frequency correct PWM

UFM Features (MachXO2/Platform Manager 2 Only)

The devices listed in Table 1 provide one sector of User Flash Memory (UFM). The UFM is a Flash sector that is organized in pages. The UFM is not byte addressable. Each page has 128 bits (16 bytes). Table 1 shows the UFM resources in each device, represented in bits, bytes and pages.

The UFM is a general purpose Flash memory. Common usages of UFM are for storing system level non-volatile data, initialization data for the on-chip Embedded Block RAM (EBR) blocks, and executable codes for microcontroller or embedded state machines.

Table 1: UFM Resources in MachXO2 and Platform Manager 2 Devices

	Mach XO2 - 256	Mach XO2 - 640	Mach XO2 - 640U	Mach XO2 - 1200	Mach XO2 - 1200U	Mach XO2 - 2000	Mach XO2 - 2000U	Mach XO2 - 4000	Mach XO2 - 7000	LPTM20	LPTM21
UFM bits	0	24448	65408	65408	81792	81792	98176	98176	261888	24448	65408
UFM Bytes	0	3056	8176	8176	10224	10224	12272	12272	32736	3056	8176
UFM Pages	0	191	511	511	639	639	767	767	2046	191	511

Functional Description

The EFB includes a SPI, two I^{2}C's (primary and secondary), and a timer/counter peripheral.

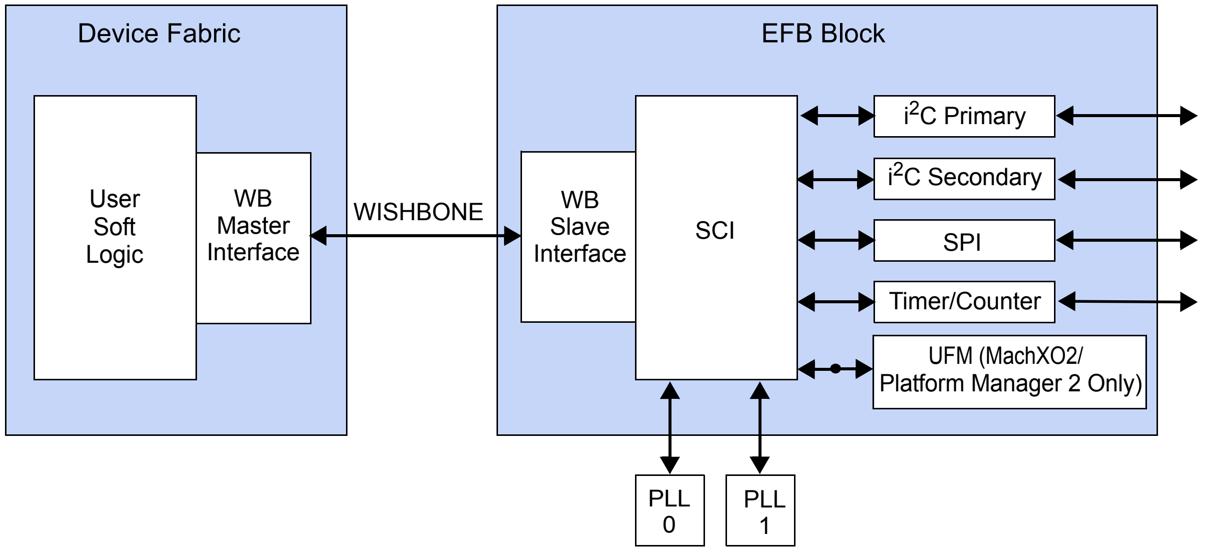
MachXO2/Platform Manger 2 only: The EFB includes User Flash Memory (UFM).

Figure 1 shows the blocks in the EFB.

Configuration

The following sections describe the graphical user interface (UI) parameters, the hardware description language (HDL) parameters, and the I/O ports that you can use to configure and operate the LatticeMico EFB.

Figure 1: WISHBONE/EFB Block Diagram



UI Parameters

Table 2 shows the UI parameters available for configuring the LatticeMico EFB through the Mico System Builder (MSB) interface.

Table 2: EFB UI Parameters

Dialog Box Option	Description	Allowable Values	Default Value
Instance Name	Specifies the name of the EFB instance.	Alphanumeric and underscores	machxo2_efb
Base Address	Specifies the base address for configuring the EFB. The minimum boundary alignment is 0x80.	0X80000000–0XFFFFFFFF	0X80000000
Diamond Project	Specifies the name of the Diamond Project		
Generated efb.v	<Instance Name>.v will be generated in <platform direcory> \components\efb\ipexpress		
User-Managed Timer Reset	Specifies the EFB Timer Reset as User Managed mode.	Selected | Not Selected	Not Selected

I/O Ports

Table 3 through Table 8 describe the input and output ports of the LatticeMico EFB.

Table 3: EFB WISHBONE I/O Ports

I/O Port	Active	Direction	Description
wb_clk_i	High	I	Positive edge clock used by all WISHBONE Interface logic blocks.
wb_rst_i	High	I	Synchronous reset signal that will only reset the WISHBONE interface logic. This signal will not affect the contents of any registers. It will only affect ongoing bus transactions.
wb_cyc_i	High	I	Cycle input indicates the WB slave a valid bus cycle is present on the bus. In a multiple-master configuration, this signal serves as a bus request. Active high signal.
wb_stb_i	High	I	Strobe input indicating the WB slave is the target for the current transaction on the WB bus. The WB slave can only drive non-zero values on its outputs (wb_dat_o, wb_ack_o, etc.) while this signal is active. The WB slave asserts some form of an acknowledgment in response to the assertion of this signal.
wb_we_i	High	I	Write/Read indicator. 0 = Read transaction 1 = Write transaction
wb_adr_i [7:0]		I	Address input to the WB slave logic.
wb_dat_i [7:0]		I	Write data input.
wb_dat_o[7:0]		O	Read Data Output.
wb_ack_o	High	O	Transfer Acknowledge asserted by the WB slave to the master, indicating the requested transfer has been completed. This signal is qualified by wb_stb_i.

.

Table 4: EFB I^{2}C I/O Ports

I/O Port	Active	Direction	Description
i2c1_irqo	High	O	Primary I^{2}C interrupt request.
i2c2_irqo	High	O	Secondary I^{2}C interrupt request.
i2c1_scl		I/O	Primary I^{2}C bi-directional clock line. The signal is an output if the I^{2}C core is in master mode. The signal is an input if the I^{2}C core is in slave mode.
i2c1_sda		I/O	Primary I^{2}C bidirectional data line. The signal is an output when data is transmitted from the I^{2}C core. The signal is an input when the I^{2}C core receives data.
i2c2_scl		I/O	Secondary I^{2}C bi-directional clock line. The signal is an output if the I^{2}C core is in master mode. The signal is an input if the I^{2}C core is in slave mode.
i2c2_sda		I/O	Secondary I^{2}C bidirectional data line. TThe signal is an output when data is transmitted from the I^{2}C core. The signal is an input when the I^{2}C core receives data.

Table 5: EFB SPI I/O Ports

I/O Port	Active	Direction	Description
spi_csn[7:0]		O	SPI chip select.
spi_scsn	Low	I	Slave chip select. An external SPI master controller can assert this signal for selecting the slave SPI core of the device.
spi_clk		I/O	Bi-directional clock line of the SPI core. The signal is an output if the SPI core is in master mode. The signal is an input if the SPI core is in slave mode. t
spi_miso		I/O	Bi-directional data line of the SPI core. The signal is an input if the SPI core is in master mode. The signal is an output if the SPI core is in slave mode.
spi_mosi		I/O	Bi-directional data line of the SPI core. The signal is an output if the SPI core is in master mode. The signal is an input if the SPI core is in slave mode.

Table 6: EFB Timer/Counter I/O Ports

I/O Port	Active	Direction	Description
tc_clki	High	I	Timer/Counter clock input.
tc_rstn	Low	I	Timer/Counter reset input.
tc_int	High	O	Timer/Counter interrupt line.
tc_oc	High	O	Timer Counter output signal.
tc_ic	High	I	Timer/Counter input capture trigger event, applicable for non-PWM modes with WISHBONE interface.

Note: Timer/counter signal TC_IC will be brought to the top level if the TC is enabled in the “Dynamic register changes...” mode.

Table 7: PLL Ports

I/O Port	Direction	Description
pll0_bus_0[8:0]	I	Pll 0 bus input.
pll0_bus_o[16:0]	O	Pll 0 bus output.
pll1_bus_i[8:0]	I	Pll 1 bus input.
pll1_bus_o[16:0]	O	Pll 1 bus output.

Table 8: UFM I/O Port (MachXO2/Platform Manager 2 Only)

I/O Port	Direction	Description
wbc_ufm_irq	O	UFM irq signal.
ufm_sn	I	Select signal that must be asserted (driven low) when SPI is enabled and accesses to UFM are performed via SPI port.

For MachXO2/Platform Manager 2, the I/O ports will appear with the port enable selections listed in Table 9.

Table 9: I/O Port Enable Selections (MachXO2/Platform Manager 2)

I/O Port	WISHBONE	I^2C Primary	I^2C Secondary	SPI	Timer/ Counter	PLL0	PLL1	UFM
I2C Primary	X	X						
I2C Secondary	X		X					
SPI	X			X				
Timer/Counter (“User Static Settings...”)					X			
Timer/Counter (“Dynamic Register Changes...”)	X				X			
PLL (w/ 1 PLL)	X					X		
PLL (w/ 1 PLLS)						X	X	
UFM	X							X

Example 1: If the user selects the I^2C Primary, the UFM port, one PLL and the SPI port to be enabled, the WISHBONE, I^2C Primary, PLL0, UFM and SPI ports appear in the graphic. The “Primary” selections under the I^2C tab are enabled along with the selections under the UFM and SPI tabs.

Example 2: If the user selects only the Timer/Counter (“static settings”) to be enabled, then the only timer/counter port will appear in the graphic. Likewise, only the selections under the T/C tab are enabled. (Note that the WB bus does not appear in this case.)

Example 3: User selects I^2C Primary, I^2C Secondary, UFM, and two PLL’s. In this case, both I^2C’s, the UFM, and both PLL0 and PLL1 are enabled. Selections under the tabs under the I^2C and UFM are enabled.

For MachXO3L, the I/O ports will appear with the port enable selections listed in Table 10.

Table 10: I/O Port Enable Selections (MachXO3L)

I/O Port	WISHBONE	I^2C Primary	I^2C Secondary	SPI	Timer/ Counter	PLL0	PLL1	UFM
I2C Primary	X	X						
I2C Secondary	X		X					

Table 10: I/O Port Enable Selections (MachXO3L) (Continued)

I/O Port	WISHBONE	I^{2}C Primary	I^{2}C Secondary	SPI	Timer/ Counter	PLL0	PLL1	UFM
SPI	X			X				
Timer/Counter (“User Static Settings...”)					X			
Timer/Counter (“Dynamic Register Changes...”)	X				X			
PLL (w/ 1 PLL)	X					X		
PLL (w/ 1 PLLS)						X	X	

Example 1: If the user selects the I^{2}C Primary, one PLL and the SPI port to be enabled, the WISHBONE, I^{2}C Primary, PLL0 and SPI ports appear in the graphic. The “Primary” selections under the I^{2}C tab are enabled along with the selections under the SPI tabs.

Example 2: If the user selects only the Timer/Counter (“static settings”) to be enabled, then the only timer/counter port will appear in the graphic. Likewise, only the selections under the T/C tab are enabled. (Note that the WB bus does not appear in this case.)

Example 3: User selects I^{2}C Primary, I^{2}C Secondary, UFM, and two PLL’s. In this case, both I^{2}C’s, and both PLL0 and PLL1 are enabled. Selections under the tabs under the I^{2}C are enabled.

Register Descriptions

EFB Regsiter Map

The EFB module has a register map to allow the service of the hardened functions through the WISHBONE bus interface read/write operations. Refer to TN1205, *Using User Flash Memory and Hardened Control Functions in MachXO2 Devices*, for more information.

WISHBONE Addressable Registers for I^{2}Cs

Table 11: WISHBONE Addressable Registers for I^{2}C

Primary Register Name	I^{2}C Secondary Register Name	Register Function	Address I^{2}C Primary	Address I^{2}C Secondary	Access
I2C_1_CR	I2C_2_CR	Control	0x40	0x4A	Read/Write
I2C_1_CMDR	I2C_2_CMDR	Command	0x41	0x4B	Read/Write
I2C_1_BR0	I2C_2_BR0	Clock Pre-scale	0x42	0x4C	Read/Write
I2C_1_BR1	I2C_2_BR1	Clock Pre-scale	0x43	0x4D	Read/Write
I2C_1_TXDR	I2C_2_TXDR	Transmit Data	0x44	0x4E	Write

Table 11: WISHBONE Addressable Registers for I^{2}C (Continued)

Primary Register Name	I^{2}C Secondary Register Name	Register Function	Address I^{2}C Primary	Address I^{2}C Secondary	Access
I2C_1_SR	I2C_2_SR	Status	0x45	0x4F	Read
I2C_1_GCDR	I2C_2_GCDR	General Call	0x46	0x50	Read
I2C_1_RXDR	I2C_2_RXDR	Receive Data	0x47	0x51	Read
I2C_1_IRQ	I2C_2_IRQ	IRQ	0x48	0x52	Read/Write
I2C_1_IRQEN	I2C_2_IRQEN	IRQ Enable	0x49	0x53	Read/Write

I^{2}C Register Definition I2C_1_BR1/0 and I2C_2_BR1/0

The I^{2}C cores have a 10-bit pre-scale register, which is used to divide the WISHBONE clock to the clock frequencies supported by the I^{2}C bus (50KHz, 100KHz and 400KHz). I2C_1_BR0[7:0] and I2C_2_BR0[7:0] hold the lower eight pre-scale register bits (7:0). I2C_1_BR1[1:0] and I2C_2_BR1[1:0] hold the upper two pre-scale register bits (9:8).

Table 12: Command Register – I2C_1_CMDR and I2C_2_CMDR

Bit	Field	Description
7	STA	Generate (Repeated) start Condition
6	STO	Generate STOP Condition
5	RD	Read from Slave
4	WR	Write to Slave
3	ACK	Acknowledge Option — When receive, ACK transmission selection 0 = Send ACK 1 = Send NACK
2	CKSDIS	Clock Stretching Disable Option — Disable the clock stretching if desired by the user. Then overflow error flag must be monitored. 0 = Clock Stretching is Enabled 1 = Clock Stretching is Disabled
1	RSVD	Reserved bit.
0	RSVD	Reserved bit.

Table 13: Status Register – I2C_1_SR and I2C_2_SR

Bit	Field	Description
7	TIP	Transmitting In Progress — This bit indicates that one byte of data is being transferred. This bit will be set at rising edge of acknowledge cycle. 1 = Byte transfer completed. 0 = Byte transfer in progress.
6	BUSY	Bus busy --- This bit indicates the bus is involved in transaction. This will be set at start condition and cleared at stop.
5	RARC	Received Acknowledge — This flag represents acknowledge from the addressed slave 1 = No acknowledge received 0 = Acknowledge received
4	SRW	Slave RW: 1 = master receiving / Slave transmitting 0 = master transmitting / Slave receiving
3	ARBL	Arbitration Lost — This bit will go high if master has lost its arbitration in Master mode, It will cause an interrupt to WISHBONE Host if SCI set up allowed. 1 = Arbitration Lost 0 = Normal
2	TRRDY	Transmitter or Receiver Ready Bit --- This flag indicate that a Transmit Register ready to receive data or Receiver Register if ready for read depend on the mode (master or slave) and SRWbit. It will cause an interrupt to WISHBONE Host if SCI set up allowed. 1 = Transmitter or Receiver is ready 0 = Transmitter or Receiver is not ready
1	TROE	Transmitter or Receiver Overrun Bit --- This flag indicate that a Transmit or Receive Overrun Errors happened depend on the mode (master or slave) and SRW bit. It will cause an interrupt to WISHBONE Host if SCI set up allowed. 1 = Transmitter or Receiver Overrun 0 = Transmitter or Receiver Normal
0	HGC	Hardware General Call Received: --- This flag indicate that a hardware general call is received from the slave port. It will cause an interrupt to WISHBONE Host if SCI set up allowed. 1 = Hardware General Call Received in Slave Mode 0 = NO Hardware General Call Received in Slave Mode

Table 14: Transmitting Data Register – I2C_1_TXDR and I2C_2_TXDR

Bit	Field	Description
7:1	W	Next byte to transmit via I^2C
0	W	In case of a data transfer, this bit represents the data's LSB. In case of a slave address transfer this bit represents the RW bit 1 = Reading from slave 0 = Writing to Slave

Table 15: Register Definition I2C_1_IRQ and I2C_2_IRQ

Bit	Field	Description
7:4	RSVD	Reserved bits.
3	IRQARBL	Interrupt Request for Arbitration Lost status bit – This bit will go high if the master has lost its arbitration in Master mode. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. 1 = Arbitration Lost 0 = Normal
2	IRQTRRDY	Interrupt Request for Transmitter or Receiver Ready status bit – This flag indicates that the Transmit Register is ready to receive data or Receiver Register is ready for read, depending on the mode (master or slave). It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. 1 = Transmitter or Receiver is ready 0 = Transmitter or Receiver is not ready
1	IRQTROE	Interrupt Request for Transmitter or Receiver Overrun status bit – This bit indicates that a Transmit or Receive Overrun Error occurred, depending on the mode (master or slave). It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. 1 = Transmitter or Receiver Overrun 0 = Transmitter or Receiver Normal
0	IRQHGC	Interrupt Request for Hardware General Call Received status bit – This bit indicates that a hardware general call was received. It will cause an interrupt to the WISHBONE Host if the interrupt signal is utilized in the design. 1 = Hardware General Call Received in Slave Mode, 0 = No Hardware General Call Received in Slave Mode.

I^{2}C Register Definition I2C_1_IRQEN and I2C_2_IRQEN

Registers I2C_1_IRQEN and I2C_2_IRQEN are used to enable the interrupt features of the I^{2}C cores. The WISHBONE Host has Read/Write access to these registers.

Table 16: Register Definition I2C_1_IRQEN and I2C_2_IRQEN

Bit	Field	Description
7:4	RSVD	Reserved bits.
3	IRQARBLEN	Enable Interrupt Request for Arbitration Lost status bit. 1 = Enabled 0 = Disabled
2	IRQTRRDYEN	Enable Interrupt Request for Transmitter or Receiver Ready status bit. 1 = Enabled 0 = Disabled
1	IRQTROEEN	Enable Interrupt Request for Transmitter or Receiver Overrun status bit. 1 = Enabled 0 = Disabled
0	IRQHGCEN	Enable Interrupt Request for Hardware General Call Received status bit. 1 = Enabled 0 = Disabled

WISHBONE Addressable Registers for SPI Module

Table 17: WISHBONE Addressable Registers for SPI Module

SPI Register Name	Register Function	Address	Access
SPICR0	Control Register 0	0x54	Read/Write
SPICR1	Control Register 1	0x55	Read/Write
SPICR2	Control Register 2	0x56	Read/Write
SPIBR	Clock Pre-scale	0x57	Read/Write
SPICSR	Master Chip Select	0x58	Read/Write
SPITXDR	Transmit Data	0x59	Write
SPISR	Status	0x5A	Read
SPIRXDR	Receive Data	0x5B	Read
SPIIRQ	Interrupt Request	0x5C	Read/Write
SPIIRQEN	Interrupt Request Enable	0x5D	Read/Write

Table 18: SPI Control Register – SPICR0

Bit	Field	Description
7:6	TIdle XCNT	Tidle Extra Delay Count — These bits specify the extra system clock count for the interval time for mcsn goes active (low) in master mode. Default (00) for half mclk clock cycle.
5:3	TTrail XCNT	TTrail Extra Delay Count — These bits specify the extra system clock count for the timing between last mclk edge and mcsn goes high in master mode. Default (000) for half mclk clock cycle.
2:0	TLead XCNT	TLead Extra Delay Count — These bits specify the extra system clock count for the timing between mcsn goes low and first clock edge in master mode. Default (000) for half mclk clock cycle.

Table 19: SPI Control Register – SPICR1

Bit	Field	Description
7	SPE	SPI System Enable Bit — This bit enables the SPI system functions. If SPE is cleared, SPI is disabled andforced into idle state, status bits in SPISR register are reset. 0 = SPI disabled 1 = SPI enabled, port pins are dedicated to SPI functions.
6	WKUPEN USR	Wakeup from Standby/Sleep (by SCSN Active) Enable Bit — This bit is enabled the SPI core to send a wakeup signal to the on chip power manager to wakeup the part from standby/sleep mode when the User SCSN goes low.
5	WKUPEN CFG	Wakeup from Standby/Sleep (by SCSN Active) Enable Bit — This bit is enabled the SPI core to send a wakeup signal to the on chip power manager to wakeup the part from standby/sleep mode when the CFG SCSN goes low.
4	TX EDGE	Data Transmitting selection bit --- This bit give user capability to select which clock edge to transmit data. 0 = Transmit data on the different clock edge of data receiving (receiving on rising / transmit on falling) 1 = Transmit data on the same clock edge of data receiving (receiving on rising / transmit on rising)
3:0	RSVD	Reserved bits.

Table 20: SPI Control Register – SPICR2

Bit	Field	Description
7	MSTR	SPI Master/Slave Mode Select Bit — This bit selects, if the SPI operates in master or slave mode. Changing this bit forces the SPI system into idle state. 0 = SPI is in slave mode 1 = SPI is in master mode
6	MCSH	SPI Master CSN Hold Bit --- This bit will hold the Master chip select active when the host is busy which will halt the data transmission without pulling the chip select high. Critical for configuration boot from external SPI boot PROM. 0 = Master running as normal 1 = Master hold chip select low even host Halt the data transmission
5	SRME	SPI Slave Slow Respond Mode Enable --- This bit enable the automatic insertion of the Lattice specific protocol to handle the issue caused by the slow respond time of the WISHBONE host at high SPI clock rate. 0 = Slave running as normal 1 = Slave automatically deploy the Lattice specific protocol.
4:3	SFSEL	SPI Special Feature Select --- This two bits select the special features for SPI port 00 = SPI port running as normal 01 = Send out 0h00 byte instead of 0hFF byte during slave write to indicate receiving register is full. 10 = Reserved 11 = Reserved
2	CPOL	SPI Clock Polarity Bit — This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state. 0 = Active-high clocks selected. In idle state SCK is low. 1 = Active-low clocks selected. In idle state SCK is high.
1	CPHA	SPI Clock Phase Bit — This bit is used to select the SPI clock format. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state. 0 = Sampling of data occurs at odd edges (1,3,5,...,15) of the SCK clock 1 = Sampling of data occurs at even edges (2,4,6,...,16) of the SCK clock
0	CPHA	LSB-First Enable — This bit does not affect the position of the MSB and LSB in the data register. Reads and writes of the data register always have the MSB in bit 7. In master mode, a change of this bit will abort a Transmission in progress and force the SPI system into idle state. 0 = Data is transferred most significant bit first. 1 = Data is transferred least significant bit first.

Table 21: SPI Baud Rate Register – SPIBR

Bit	Field	Description
7	RSVD	Reserved bit.
6	RSVD	Reserved bit.
5:0	DIVIDER	SPI Master SCK Frequency Divisor — Clock frequency divisor from the source clock for baud rate selection. Fmsck = Fsource / DIVIDER

Table 22: SPI Status Register – SPISR

Bit	Field	Description
7	TIP	SPI Transmitting In Progress — This bit indicate that the SPI port in the middle of transmitting/receiving data. 0 = SPI Transmitting is finished 1 = SPI Transmitting is in progress
6:5	RSVD	Reserved bits.
4	TRDY	SPI Transmit Ready Flag - Indicates the SPI transmit data register (SPITXDR) is empty. This bit is cleared by a write to SPITXDR. It will cause an interrupt to WISHBONE Host if SCI set up allowed. 0 = SPI Data register not empty 1 = SPI Data register empty
3	RRDY	SPI Receive Ready Flag - Indicates the receive data register (SPIRXDR) contains valid receive data. This bit is cleared by a read access to SPIRXDR. It will cause an interrupt to WISHBONE Host if SCI set up allowed. Host if SCI set up allowed. 0 = Transfer not yet complete 1 = New data copied to SPIRXDR
2	TOE	Transmit Overrun Error Flag — This bit indicates that the SPITXDR received new data before the previous data was moved to the shift register. The new data is discarded if occurs. It will cause an interrupt to WISHBONE. Host if SCI set up allowed.
1	ROE	Receive Overrun Error Flag — This bit indicates that the SPIRXDR received new data before the previous data was read. The previous data will be lost if occurs. It will cause an interrupt to WISHBONE Host if SCI set up allowed.
0	MDF	Mode Fault Flag — This bit is set if the SS input becomes low while the SPI is configured as a master and mode fault detection is enabled. The flag is cleared automatically by a write to the SPI Control Register. 0 = Mode fault has not occurred. 1 = Mode fault has occurred. It will cause an interrupt to WISHBONE Host if SCI set up allowed.

Table 23: Register Definition SPICSR

Bit	Field	Description
7	CSN_7	Active-Low, master chip select (MCSN[7])
6	CSN_6	Active-Low, master chip select (MCSN[6])
5	CSN_5	Active-Low, master chip select (MCSN[5])
4	CSN_4	Active-Low, master chip select (MCSN[4])
3	CSN_3	Active-Low, master chip select (MCSN[3])
2	CSN_2	Active-Low, master chip select (MCSN[2])
1	CSN_1	Active-Low, master chip select (MCSN[1])
0	CSN_0	Active-Low, master chip select (MCSN[0], has pre-assigned pin location)

SPI Register Definition SPIIRQ

Interrupt register SPIIRQ supports the status bits of the SPISR register. The WISHBONE Host can query these bits when an interrupt request is received.

Table 24: Register Definition SPIIRQ

Bit	Field	Description
7:5	RSVD	Reserved bits.
4	IRQTRDY	Interrupt request for SPI Transmit Empty Interrupt Flag – If set, this bit indicates that the transmit data register is empty. This bit is cleared by a write to SPITXDR register. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. 0 = SPI Data register not empty 1 = SPI Data register empty
3	IRQRRDY	Interrupt request for SPI Receive Interrupt Flag – This bit is set after a received data byte has been transferred into the SPIRXDR register. This bit is cleared after a read operation from the WISHBONE interface is performed. It will cause an interrupt to the WISHBONE Host if the interrupt signal is utilized in the design. 0 = Transfer not yet complete 1 = New data copied to SPIRXDR
2	IRQTOE	Interrupt request for Transmit Overrun Error Flag – This bit indicates that the SPITXDR received new data before the previous data was moved to the shift register for serial transfer over the SPI bus. The new data is discarded if the error occurs. It will cause an interrupt to the WISHBONE Host if the interrupt signal is utilized in the design. 0 = No Error 1 = Transmit Overrun Error has occurred

Table 24: Register Definition SPIIRQ (Continued)

Bit	Field	Description
1	IRQROE	Interrupt request for Receive Overrun Error Flag – This bit indicates that the SPIRXDR received new data before the previous data was read by the WISHBONE host. The previous data will be lost if the overrun occurs. It will cause an interrupt to the WISHBONE Host if the interrupt signal is utilized in the design. 0 = Error has not occurred 1 = Transmit Overrun Error has occurred
0	IRQMDF	Interrupt request for Mode Fault Flag – This bit is set if the SSCN input becomes low while the SPI is configured as a master controller. The flag is cleared automatically by a write to the SPICR2, which controls the mode of the core. It will cause an interrupt to the WISHBONE Host if the interrupt signal is utilized in the design. 0 = Mode Fault has not occurred 1 = Mode Fault has occurred

SPI Register Definition SPIIRQEN

Register SPIIRQEN is used to enable the interrupt features of the SPI core. The WISHBONE Host has Read/Write access to this register.

Table 25: Register Definition SPIIRQEN

Bit	Field	Description
7:5	RSVD	Reserved bits.
4	IRQTRDYEN	Enabled interrupt request for SPI Transmit Empty Interrupt Flag. 1 = Enabled 0 = Disabled
3	IRQRRDYEN	Enabled interrupt request for SPI Receive Interrupt Flag. 1 = Enabled 0 = Disabled
2	IRQTOEEN	Enabled interrupt request for Transmit Overrun Error Flag. 1 = Enabled 0 = Disabled
1	IRQROEEN	Enabled interrupt request for Receive Overrun Error Flag. 1 = Enabled 0 = Disabled
0	IRQMDFEN	Enabled interrupt request for Mode Fault Flag. 1 = Enabled 0 = Disabled

WISHBONE Addressable Registers for Timer/ Counter Module

This section lists the internal registers of the Timer/Counter hard IP. Software will be able to write some of registers based on attributes set by the users.

Table 26: WISHBONE Addressable Registers for Timer/Counter Module

Signal Name	I/O	Width	Description
tc_clki	Input	1	Timer/Counter input clock signal. Can be connected to the on-chip oscillator.
tc_rstn	Input	1	This is an active-low reset signal, which resets the 16-bit counter.
tc_ic	Input	1	This is an active-high input capture trigger event, applicable for non-PWM modes with WISHBONE interface. If enabled, a rising edge of this signal will be detected and synchronized to capture the counter value (TCCNT Register) and make the value accessible to the WISHBONE interface by loading it into TCICR register. The common usage is to perform a time-stamp operation with the counter.
tc_int	Output	1	This is an interrupt signal, indicating the occurrence of a specific event such as Overflow, Output Compare Match, or Input Capture.
tc_oc	Output	1	Timer/Counter output signal

Timer/Counter Registers

The Timer/Counter communicates with the PLD logic through the WISHBONE interface, by utilizing a set of control, status and data registers. Table shows the register names and their functions. These registers are a subset of the EFB register map. Refer to the EFB register map for specific addresses of each register.

Table 27: Timer/Counter Registers

Timer/Counter Register Name	Register Function	Address	Access
TCCR0	Control Register 0	0x5E	Read/Write
TCCR1	Control Register 1	0x5F	Read/Write
TCTOPSET0	Set Top Counter Value [7:0]	0x60	Write
TCTOPSET1	Set Top Counter Value [15:8]	0x61	Write
TCOCRSET0	Set Compare Counter Value [7:0]	0x62	Write
TCOCRSET1	Set Compare Counter Value [15:8]	0x63	Write
TCCR2	Control Register 2	0x64	Read/Write
TCCNT0	Counter Value [7:0]	0x65	Read
TCCNT1	Counter Value [15:8]	0x66	Read

Table 27: Timer/Counter Registers (Continued)

Timer/Counter Register Name	Register Function	Address	Access
TCTOP0	Current Top Counter Value [7:0]	0x67	Read
TCTOP1	Current Top Counter Value [15:8]	0x68	Read
TCOCR0	Current Compare Counter Value [7:0]	0x69	Read
TCOCR1	Current Compare Top Counter Value [15:8]	0x6A	Read
TCICR0	Current Capture Counter Value [7:0]	0x6B	Read
TCICR1	Current Capture Counter Value [15:8]	0x6C	Read
TCSR0	Status Register	0x6D	Read
TCIRQ	Interrupt Request	0x6E	Read/Write
TCIRQEN	Interrupt Request Enable	0x6F	Read/Write

Timer/Counter Register Definition TCCR0

Register TCCR0 is used to control the reset and the clock into the timer/counter. The WISHBONE host has full read/write access to the register. The register values can be updated dynamically during device operation.

Table 28: Register Definition TCCR0

Bit	Field	Description
7	RSTEN	The bit enables the reset signal (tc_rstn) to enter the Timer/Counter core from the PLD logic. 0 = Reset is disabled 1 = Reset is enabled
6	RSVD	Reserved bit.
5:3	PRESCALE	These three bits are used to divide the clock input to the Timer/Counter. 000 = Static 001 = Divide by 1 010 = Divide by 8 011 = Divide by 64 100 = Divide by 256 101 = Divide by 1024

Table 28: Register Definition TCCR0 (Continued)

Bit	Field	Description
2	CLKEDGE	This bit is used to select the edge of the input clock source. The Timer/Counter will update states on the edge of the input clock source. 0 = Rising Edge, 1 = Falling Edge.
1:0	CLKSEL	These two bits define the source of the input clock source. The clock can arrive from the clock tree or directly from the on-chip oscillator. 00 = Clock Tree 10 = On-Chip Oscillator States 01 and 11 are reserved

Timer/Counter Register Definition TCCR1

Register TCCR1 is used to control the reset and the clock into the timer/counter. The WISHBONE host has full Read/Write access to the register. The register values can be updated dynamically during device operation.

Table 29: Register Definition TCCR1

Bit	Field	Description
7	RSVD	Reserved bit.
6	SOVFEN	This bit enables the overflow flag to be used with the interrupt output signal. It is set when the Timer/Counter is standalone, with no WISHBONE interface. 0 = Disabled, 1 = Enabled. When this bit is set, other flags such as the OCRF and ICRF will not be routed to the interrupt output signal.
5	ICEN	This bit enables the ability to perform a capture operation of the counter value. Users can assert the “tc_ic” signal and load the counter value onto the TCICR0/1 registers. The captured value can serve as a timer stamp for a specific event. 0 = Disabled, 1 = Enabled.
4	TSEL	This bit enables the auto-load of the counter with a value that is presented through the WISHBONE bus. 0 = Disabled, 1 = Enabled. TCTOPSET0/1 registers are written into TCTOP0/1 registers (register update) automatically.

Table 29: Register Definition TCCR1 (Continued)

Bit	Field	Description
3:2	OCM	These bits select the function of the output signal of the Timer/Counter. The available functions are Static, Toggle, Set/Clear and Clear/Set. **All Timer/Counter modes:** 00 = The output is static low In non-PWM modes: 01 = Toggle on TOP match **In Fast PWM mode:** 10 = Clear on TOP match, Set on OCR match 11 = Set on TOP match, Clear on OCR match In Phase and Frequency Correct PWM mode: 10 = Clear on OCR match when the counter is incrementing, Set on OCR match when counter is decrementing 11 = Set on OCR match when the counter is incrementing, Clear on OCR match when the counter is decrementing
1:0	TCM	These bits define the mode of operation for the Timer/Counter. 00 = Watchdog Timer Mode 01 = Clear Timer on Compare Match Mode 10 = Fast PWM Mode 11 = Phase and Frequency Correct PWM Mode

Timer/Counter Register Definition TCCR2

Register TCCR2 is used to control for additional control functions such as resetting the counter with a Write command from the WISHBONE interface, pausing the counter and forcing the output of the Timer/Counter to update even if the update conditions have not been met. The WISHBONE host has full Read/Write access to the register. The register values can be updated dynamically during device operation.

Table 30: Register Definition TCCR2

Bit	Field	Description
7:3	RSVD	Reserved bits.
2	WBFORCE	In non-PWM modes, this bit forces the output of the counter, as if the counter value matched the compare (TCOCR) value or it matched the top value (TCTOP). 0 = Disabled 1 = Enabled

Table 30: Register Definition TCCR2 (Continued)

Bit	Field	Description
1	WBRESET	Reset the counter from the WISHBONE interface by writing a ‘1’ to this register location. It is a one cycle assertion in WISHBONE clock domain once a ‘1’ is written to this register location. 0 = Disabled 1 = Enabled This bit has higher priority then WBPAUSE.
0	WBPAUSE	Writing a ‘1’ to this register bit will pause counting of the 16-bit counter.

Timer/Counter Register Definition

TCTOPSET0/1 Registers TCTOPSET0 and TCTOPSET1 are 8-bit registers, which combined, receive a 16-bit value from the WISHBONE host. They serve for double-registering the loading of the top value for the counter. The value is loaded from TCTOPSET0/1 to the TCTOP0/1 registers once the counter has completed the current counting cycle. Refer to the Timer/Counter Modes of Operation for usage details.

TCTOPSET0 register holds the lower 8-bit value [7:0] of the top value. TCTOPSET1 register holds the upper 8-bit value [15:8] of the top value.

Timer/Counter Register Definition TCOCRSET0/1

Registers TCOCRSET0 and TCOCRSET1 are 8-bit registers, which combined, receive a 16-bit value from the WISHBONE host. They serve for double-registering the loading of the compare value for the counter. The value is loaded from TCOCRSET0/1 to the TCOCR0/1 registers once the counter has completed the current counting cycle. Refer to the Timer/Counter Modes of Operation for usage details.

TCOCRSET0 register holds the lower 8-bit value [7:0] of the compare value. TCOCRSET1 register holds the upper 8-bit value [15:8] of the compare value.

Timer/Counter Register Definition TCCNT0/1

Registers TCCNT0 and TCCNT1 are 8-bit registers, which combined, hold the counter value. The WISHBONE host has Read-Only access to these registers.

TCCNT0 register holds the lower 8-bit value [7:0] of the counter value. TCCNT1 register holds the upper 8-bit value [15:8] of the counter value.

Timer/Counter Register Definition TCTOP0/1

Registers TCTOP0 and TCTOP1 are 8-bit registers, which combined, receive a 16-bit value from the TCTOPSET0/1. The data stored in these registers represents the top value of the counter. The registers update once the counter has completed the current counting cycle. Refer to the Timer/Counter Modes of Operation for usage details.

TCTOP0 register holds the lower 8-bit value [7:0] of the top value. TCTOP1 register holds the upper 8-bit value [15:8] of the top value.

Timer/Counter Register Definition TCOCR0/1

Registers TCOCR0 and TCOCR1 are 8-bit registers, which combined, receive a 16-bit value from the TCOCRSET0/1. The data stored in these registers represents the compare value of the counter. The registers update once the counter has completed the current counting cycle. Refer to the Timer/Counter Modes of Operation for usage details.

TCOCR0 register holds the lower 8-bit value [7:0] of the compare value. TCOCR1 register holds the upper 8-bit value [15:8] of the compare value.

Timer/Counter Register Definition TCICR0/1

Registers TCICR0 and TCICR1 are 8-bit registers, which combined, can hold the counter value. The counter value is loaded onto these registers once a trigger event, tc_ic IP signal, is asserted. The capture value is commonly used as a time-stamp for a specific system event.

The WISHBONE host has Read-Only access to these registers.

TCICR0 register holds the lower 8-bit value [7:0] of the counter value. TCICR1 register holds the upper 8-bit value [15:8] of the counter value.

Timer/Counter Register Definition TCSR0

TCSR0 is a status register, used for setting or clearing operation flags. The four flags that are used with the Timer/Counter represent statuses such as overflow, compare match, bottom state and capture counter value.

The WISHBONE host has Read/Write access to register TCSR0.

Table 31: Register Definition TCSR0

Bit	Field	Description
7:4	RSVD	Reserved bits.
3	BTF	Bottom flag when the counter reaches value zero. The bit is cleared after a Write operation from the WISHBONE interface.
2	ICRF	Capture Counter flag when the user asserts the trigger event tc_ic IP signal. The counter value is captured into the TCICR0/1 registers. The bit is cleared after a Write operation from the WISHBONE interface. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.
1	OCRF	Compare match flag when counter matches the value loaded in the TCOCR0/1 registers. The bit is cleared after a Write operation from the WISHBONE interface. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. The interrupt line is asserted for one clock cycle.
0	OVF	Overflow flag when the counter matches the top value of the counter loaded onto the TCTOP0/1 registers. The bit is cleared after a Write operation from the WISHBONE interface. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. The interrupt line is asserted for one clock cycle.

Timer/Counter Register Definition TCIRQ

Register TCIRQ holds the interrupt bits caused by the status flags in the status register. Timer/Counter supports interrupt requests for events such as counter overflow, compare match, and capture counter value.

The WISHBONE host has Read/Write access to register TCSR0.

Table 32: Register Definition TCIRQ

Bit	Field	Description
7:3	RSVD	Reserved bits.
2	IRQICRF	Interrupt caused by the Capture Counter flag when the user asserts the trigger event tc_ic IP signal. The counter value is captured into the TCICR0/1 registers. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.
1	IRQOCRF	Interrupt caused Compare match flag when counter matches the value loaded in the TCOCR0/1 registers. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. The interrupt line is asserted for one clock cycle.
0	IRQOVF	Interrupt caused Overflow flag when the counter matches the top value of the counter loaded onto the TCTOP0/1 registers. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design. The interrupt line is asserted for one clock cycle.

Timer/Counter Register Definition TCIRQEN

Register TCIRQEN holds the enable/disable bits for the available interrupt support of the Timer/Counter IP.

Table 33: Register Definition TCIRQEN

Register	Field	Description
7:3	RSVD	Reserved bits.
2	IRQICRFEN	Enabled interrupt request for Capture counter flag. 1 = Enabled 0 = Disabled
1	IRQOCRFEN	Enabled interrupt request for Compare match flag. 1 = Enabled 0 = Disabled
0	IRQOVFEN	Enabled interrupt request for overflow flag. 1 = Enabled 0 = Disabled

WISHBONE Addressable Registers for UFM Module (MachXO2/Platform Manager 2 Only)

Table 34: WISHBONE Addressable Registers for UFM Module

Register Name	Register Function	Address	Access
CFGCR	Control	0x70	Read/Write
CFGTXDR	Transmit Data	0x71	Write
CFGSR	Status	0x72	Read
CFGRXDR	Receive Data	0x73	Read
CFGIRQ	Interrupt Request	0x74	Read/Write
CFGIRQEN	Interrupt Request Enable	0x75	Read/Write

Table 35: UFM Control Register - CFGCR

Bit	Field	Description
7	WBCE	WISHBONE Connection Enable. Enables the WISHBONE to establish the read/write connection to the UFM/Configuration logic. This bit must be set prior to executing any command through the WISHBONE port. Likewise, this bit must be cleared to terminate the command. 1 = Enabled 0 = Disabled
6	RSTE	WISHBONE Connection Reset. Resets the input/output FIFO logic. The reset logic is level sensitive. After setting this bit to '1' it must be cleared to '0' for normal operation. 1 = Reset 0 = Normal operation
5:0	RSVD	Reserved bits.

Table 36: UFM Transmit Data Register - CFGTXDR

Bit	Field	Description
7:0	W	This register holds the byte that will be written to the UFM logic.

Table 37: UFM Status Register - CFGSR

Bit	Field	Description
7	WBCACT	Indicates that the WISHBONE to UFM interface is active and the connection is established.
6	RSVD	Reserved bit.
5	TXFE	Indicates that the Transmit FIFO register is empty 1 = FIFO empty 0 = FIFO not empty
4	TXFF	Indicates that the Transmit FIFO register is full 1 = FIFO full 0 = FIFO not full
3	RXFE	Indicates that the Receive FIFO register is empty 1 = FIFO empty 0 = FIFO not empty
2	RXFF	Indicates that the Receive FIFO register is full 1 = FIFO full 0 = FIFO not full
1	SSPIACT	Indicates the Slave SPI port has started actively communicating with the UFM Logic while WBCE was enabled. 1 = Slave SPI port active 0 = Slave SPI port not active
0	I2CACT	Indicates the I2C port has started actively communicating with the UFM Logic while WBCE was enabled. 1 = I2C port active 0 = 2C port not active

Table 38: UFM Receive Data Register - CFGRXDR

Bit	Field	Description
7:0	R	This register holds the byte that will be read to the UFM logic.

UFM Register Definition CFGIRQ

Interrupt register CFGIRQ supports the status bits of the CFGSR register. The WISHBONE Host can query these bits when an interrupt request is received.

Table 39: UFM Interrupt Status Register – CFGIRQ

Bit	Field	Description
7:6	RSVD	Reserved bits.
5	IRQTXFE	Interrupt request for UFM Transmit FIFO Empty Interrupt Flag – If set, this bit indicates that the transmit data register is empty. This bit is cleared by write a "1" to this register. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.
4	IRQTXFF	Interrupt request for UFM Transmit FIFO Full Interrupt Flag – If set, this bit indicates that the transmit data register is full. This bit is cleared by write a "1" to this register. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.
3	IRQRXFE	Interrupt request for UFM Receive FIFO Empty Interrupt Flag – If set, this bit indicates that the receive data register is empty. This bit is cleared by write a "1" to this register. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.
2	IRQRXFF	Interrupt request for UFM Receive FIFO Full Interrupt Flag – If set, this bit indicates that the receive data register is full. This bit is cleared by write a "1" to this register. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.
1	IRQSSPIACT	Interrupt request for UFM Slave SPI Active Interrupt Flag – If set, this bit indicates that the Slave SPI is asserted. This bit is cleared by write a "1" to this register. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.
0	IRQI2CACT	Interrupt request for UFM I2C Active Interrupt Flag – If set, this bit indicates that the I2C is asserted. This bit is cleared by write a "1" to this register. It will cause an interrupt to WISHBONE Host if the interrupt signal is utilized in the design.

UFM Register Definition CFGIRQEN

Register CFGIRQEN is used to enable the interrupt features of the UFM core. The WISHBONE Host has Read/Write access to this register.

Table 40: UFM Interrupt Enable Register – CFGIRQEN

Bit	Field	Description
7:6	RSVD	Reserved bits.
5	IRQTXFEEN	Interrupt Enable for Transmit FIFO Empty 1 = Enabled 0 = Disabled

Table 40: UFM Interrupt Enable Register – CFGIRQEN (Continued)

Bit	Field	Description
4	IRQTXFFEN	Interrupt Enable for Transmit FIFO Full 1 = Enabled 0 = Disabled
3	IRQRXFEEN	Interrupt Enable for Receive FIFO Empty 1 = Enabled 0 = Disabled
2	IRQRXFFEN	Interrupt Enable for Receive FIFO Full 1 = Enabled 0 = Disabled
1	IRQSSPIACTEN	Interrupt Enable for Slave SPI Active 1 = Enabled 0 = Disabled
0	IRQI2CACTEN	Interrupt Enable for I^{2}C Active 1 = Enabled 0 = Disabled

Usage Model

For more information about EFB usage in Lattice MachXO2 devices, refer to TN1205, *Using User Flash Memory and Hardened Control Functions in MachXO2 Devices*.

LatticeMico32 Microprocessor Software Support

This section describes the LatticeMico32 software support provided for the LatticeMico EFB component.

Device Driver

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

Device Context Structure

This section describes the type definitions for the LatticeMico EFB device context structure. This structure, shown in Figure 2, contains the EFB 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 EFB for component-specific information from the platform specification file. You should not manipulate the members directly, because this structure is for exclusive use by the device driver.

Table 41 describes the parameters of the LatticeMico EFB device context structure for MachXO2/Platform Manager 2 shown in Figure 2.

Figure 2: LatticeMico EFB Device Context Structure (MachXO2/Platform Manager 2)

```
struct st_MicoEFBCtx_t {
  const char *name ;
  unsigned int base ;
  unsigned int intrLevel ;
  void *desc_i2c1 ;
  unsigned int user_i2c1 ;
  void *desc_i2c2 ;
  unsigned int user_i2c2 ;
  void *desc_spi ;
  unsigned int user_spi ;
  void *desc_tc ;
  void *desc_wbcfg ;
  void *desc_pcs0 ;
  void *desc_pcs1 ;
  void *desc_pcs2 ;
  void *desc_pcs3 ;
  void *desc_pcs4 ;
  DeviceReg_t loopupReg ;
  void *prev ;
  void *next ;
} MicoEFBCtx_t;
```

Table 41 describes the parameters of the LatticeMico EFB device context structure for MachXO3L, shown in Figure 3.

Figure 3: LatticeMico EFB Device Context Structure (MachXO3L

```
struct st_MicoEFBCtx_t {
  const char *name ;
  unsigned int base ;
  unsigned int intrLevel ;
  void *desc_i2c1 ;
  unsigned int user_i2c1 ;
  void *desc_i2c2 ;
  unsigned int user_i2c2 ;
  void *desc_spi ;
  unsigned int user_spi ;
  void *desc_tc ;
  void *desc_pcs0 ;
  void *desc_pcs1 ;
  void *desc_pcs2 ;
  void *desc_pcs3 ;
  void *desc_pcs4 ;
  DeviceReg_t loopupReg ;
  void *prev ;
  void *next ;
} MicoEFBCtx_t;
```

Table 41: LatticeMico EFB Device Context Structure Parameters

Parameter	Data Type	Description
name	const char *	Instance name, as specified by the customer in MSB when instantiating it.
base	unsigned int	MSB-assigned base address for this instance.
intrLevel	unsigned int	LatticeMico32 interrupt line to which this instance is connected.
desc_i2c1	void *	I^2C 1 interrupt descriptor structure.
user_i2c1	unsigned int	I^2C 1 interrupt routine implemented by the customer.
desc_i2c2	void *	I^2C 2 interrupt descriptor structure.
user_i2c2	unsigned int	I^2C 2 interrupt routine implemented by the customer.
desc_spi	void *	SPI interrupt descriptor structure.
user_spi	unsigned int	SPI interrupt routine is implemented by the customer.
desc_tc	void *	Timer/counter interrupt descriptor structure.
desc_wbcfg	void *	Configuration interrupt descriptor structure. **(MachXO2/Platform Manager 2 only)**
desc_pcs0	void *	PCS 0 interrupt descriptor structure.
desc_pcs1	void *	PCS 1 interrupt descriptor structure.
desc_pcs2	void *	PCS 2 interrupt descriptor structure.
desc_pcs3	void *	PCS 3 interrupt descriptor structure.
desc_pcs4	void *	PCS 4 interrupt descriptor structure.
lookupReg	DeviceReg_t	Used by the device driver to register the LatticeMico EFB instance with the LatticeMico32 lookup service. Refer to LatticeMico32 Software Developer User Guide for a description of the DeviceReg_t data type.

Table 41: LatticeMico EFB Device Context Structure Parameters (Continued)

prev	void *	Used internally by the lookup service for tracking multiple registered instances of EFB.
next		Used internally by the lookup service for tracking multiple registered instances of EFB.

Interrupt Management

The EFB silicon has 10 individual interrupt sources. The LatticeMico EFB component ORs all these interrupts in to a single interrupt source to be connected to one of the interrupt lines of LatticeMico32. When an interrupt event occurs from any of the 10 sources, it is seen as an interrupt event from the LatticeMico EFB as far as LatticeMico32 is concerned. At this point of time, LatticeMico32 interrupt management software will transfer program control to the global interrupt servicing routine within LatticeMico EFB. This routine is responsible for identifying which of the 10 sources caused the interrupt event, and then transfers control to the servicing routine associated with this source. The LatticeMico32 software device driver provides sample implementations of the interrupt servicing routines for I^2C and SPI interfaces. All the remaining interrupt servicing routines must be implemented by the customer in the user application. The LatticeMico32 software device driver also provides a mechanism to override the Lattice-implemented I^2C and SPI interrupt servicing routines with user-optimized versions. To register a user-implemented servicing routine, the customer must set up the corresponding interrupt source's interrupt descriptor structure via Lattice-provided API. Table 42 through Table 50 describe the interrupt descriptors for each interrupt source, as shown in Figure 4 through Figure 12.

Figure 4: I^2C 1 and 2 Interrupt Descriptor

```
typedef void (*I2CCallback_t)(MicoEFBCtx_t *ctx) ;
struct st_I2CDesc_t {
  void *data ;
  I2CCallback_t onCompletion ;
}
```

Table 42: I^2C 1 and 2 Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to I^2C descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*I2CCallback_t)	Pointer to the interrupt servicing routine.

Figure 5: SPI Interrupt Descriptor

```
typedef void (*SPICallback_t)(MicoEFBCtx_t *ctx) ;
```

```
struct st_SPIDesc_t {
  void *data ;
  SPICallback_t onCompletion ;
}
```

Table 43: SPI Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to SPI descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*SPICallback_t)	Pointer to the interrupt servicing routine.

Figure 6: Timer/Counter Interrupt Descriptor

```
typedef void (*TCCallback_t)(MicoEFBCtx_t *ctx) ;
struct st_TCDesc_t {
  void *data ;
  TCCallback_t onCompletion ;
}
```

Table 44: Timer/Counter Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to Timer/Counter descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*TCCallback_t)	Pointer to the interrupt servicing routine.

Figure 7: Configuration Interrupt Descriptor (MachXO2/Platform Manager 2 only)

```
typedef void (*WBCFGCallback_t)(MicoEFBCtx_t *ctx) ;
struct st_WBCFGDesc_t {
  void *data ;
  WBCFGCallback_t onCompletion ;
}
```

Table 45: Configuration Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to Configuration descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*WBCFGCallback_t)	Pointer to the interrupt servicing routine.

Figure 8: PCS 0 Interrupt Descriptor

```
typedef void (*PCS0Callback_t)(MicoEFBCtx_t *ctx) ;
struct st_PCS0Desc_t {
  void *data ;
  PCS0Callback_t onCompletion ;
}
```

Table 46: PCS 0 Interrupt Descriptor Parameters

Parameter	Data Type	Description
data	void *	Pointer to PCS 0 descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*PCS0Callback_t)	Pointer to the interrupt servicing routine.

Figure 9: PCS 1 Interrupt Descriptor

```
typedef void (*PCS1Callback_t)(MicoEFBCtx_t *ctx) ;
struct st_PCS1Desc_t {
  void *data ;
  PCS1Callback_t onCompletion ;
}
```

Table 47: PCS 1 Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to PCS 1 descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*PCS1Callback_t)	Pointer to the interrupt servicing routine.

Figure 10: PCS 2 Interrupt Descriptor

```
typedef void (*PCS2Callback_t)(MicoEFBCtx_t *ctx) ;
struct st_PCS2Desc_t {
  void *data ;
  PCS2Callback_t onCompletion ;
}
```

Table 48: PCS 2 Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to PCS 2 descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*PCS2Callback_t)	Pointer to the interrupt servicing routine.

Figure 11: PCS 3 Interrupt Descriptor

```
typedef void (*PCS3Callback_t)(MicoEFBCtx_t *ctx) ;
struct st_PCS3Desc_t {
  void *data ;
  PCS3Callback_t onCompletion ;
}
```

Table 49: PCS 3 Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to PCS 3 descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*PCS3Callback_t)	Pointer to the interrupt servicing routine.

Figure 12: PCS 4 Interrupt Descriptor

```
typedef void (*PCS4Callback_t)(MicoEFBCtx_t *ctx) ;
struct st_PCS4Desc_t {
  void *data ;
  PCS4Callback_t onCompletion ;
}
```

Table 50: PCS 4 Interrupt Descriptor Parameters

Parameter	Data Type	Description
Data	void *	Pointer to PCS 4 descriptor used to hold data that is passed between applications and interrupt servicing routine and vice-versa.
onCompletion	void (*PCS4Callback_t)	Pointer to the interrupt servicing routine.

Functions

This section describes the implemented device driver-specific functions.

MicoEFBInit Function

```
void MicoEFBInit (MicoEFBCtx_t *ctx) ;
```

This function initializes a LatticeMico EFB instance according to the passed EFB for context structure. This initialization function is responsible for re-initializing the EFB for, clearing all pending interrupts, and initializing members of the passed EFB context. As part of the managed build process, the LatticeDDInit function calls this initialization routine for each EFB instance in the platform.

Table 51 describes the parameter in the MicoEFBInit function syntax.

Table 51: Parameter in the MicoEFBInit Function Syntax

Parameter	Data Type	Description
ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.

MicoEFB_ISR Function

```
void MicoEFB_ISR (unsigned int intrLevel, void *ctx) ;
```

This function implements the global interrupt servicing routine of LatticeMico EFB. It is responsible for identifying the interrupt source and transferring control to the corresponding source's interrupt handler. It clears the source's interrupt request flag in the global interrupt request registers (IRQ0 and IRQ1) of the EFB. If two or more sources request an interrupt simultaneously, then this routine will call the interrupt handlers based on the priority shown in Table 52 (MachXO2/Platform Manager 2) and Table 53 (MachXO3L).

Table 52: Priority of Interrupt Sources in LatticeMico EFB (MachXO2/Platform Manager 2)

Interrupt Source	Priority
I^2C 1 (Primary)	0 (Highest)
I^2C 2 (Secondary)	1
SPI	2
Timer/Counter	3
Configuration	4
PCS 0	5
PCS 1	6
PCS 2	7
PCS 3	8
PCS 4	9 (Lowest)

Table 53: Priority of Interrupt Sources in LatticeMico EFB (MachXO3L)

Interrupt Source	Priority
I^2C 1 (Primary)	0 (Highest)
I^2C 2 (Secondary)	1
SPI	2
Timer/Counter	3
PCS 0	4
PCS 1	5

Table 53: Priority of Interrupt Sources in LatticeMico EFB (MachXO3L) (Continued)

PCS 2	6
PCS 3	7
PCS 4	8 (Lowest)

Table 54 describes the parameter in the MicoEFB_ISR function syntax.

Table 54: Description of Parameters in MicoEFB_ISR Function Syntax

Parameter	Data Type	Description
intrLevel	unsigned int	The interrupt line of LatticeMico32 in to which the EFB interrupt sinks.
Ctx	void *	Pointer to the EFB context representing a valid EFB context.

MicoEFB_SPITransfer Function

```
char MicoEFB_SPITransfer (MicoEFBCtx_t *ctx,
        unsigned char isMaster,
        unsigned char slvIndex,
        unsigned char insertStart,
        unsigned char insertStop,
        unsigned char *txBuffer,
        unsigned char *rxBuffer,
        unsigned int bufferSize,
        unsigned int irqmode) ;
```

This function is used to transfer data over the SPI interface in theEFB. The transfer can be performed in a polling (i.e., blocking) mode or an interrupt-driven (i.e., non-blocking) mode. In case of polling mode, control is transferred to the application upon completion (successful or otherwise) of the transfer. In case of interrupt-driven mode, control is transferred to the application immediately after setting up the transfer. By default, the interrupt-driven mode uses Lattice-provided interrupt handler. The customer can override this handler by implementing a handler in application code and then registering it with the LatticeMico EFB context (see function MicoEFB_RegisterSPIISR). Table 55 describes the parameter in the MicoEFB_SPITransfer function syntax.

Table 55: Description of Parameters of MicoEFB_SPITransfer Function Syntax

Parameter	Data Type	Description
ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.
isMaster	unsigned char	Is SPI configured to be a Master (1) or a Slave (0).
slvIndex	unsigned char	The device SPI can communicate with up to 8 external SPI slave devices. This argument identifies which slave device the transfer is intended for. NOTE: Only useful when SPI is configured to be a Master.

Table 55: Description of Parameters of MicoEFB_SPITransfer Function Syntax (Continued)

insertStart	unsigned char	Is the transfer a new transfer (1), or the continuation of an existing transfer (0). NOTE: Only useful when SPI is configured to be a Master.
insertStop	unsigned char	Should the slave chip select line be de-asserted (1) or left asserted (0) at the end of transfer. NOTE: Only useful when SPI is configured to be a Master.
txBuffer	unsigned char *	Pointer to the array that contains the data to be transmitted.
rxBuffer	unsigned char *	Pointer to the array that will store the data that is received. The application is responsible for allocating memory for this array.
bufferSize	unsigned int	The number of byte-pairs to be transferred in the current transaction. For every byte that is transmitted, a byte is received (note that SPI is a full duplex protocol).
irqmode	unsigned int	Is the transfer to be performed in polling (i.e., blocking) or interrupt-driven (non-blocking) mode.

MicoEFB_SPIISR Function

```
void MicoEFB_SPIISR (MicoEFBCtx_t *ctx) ;
```

This function implements the interrupt handler for the SPI interface in the EFB. It is the default implementation. Table 56 describes the parameters in the MicoEFB_SPITransfer function syntax..

Table 56: Description of Parameters of MicoEFB_SPIISR Function Syntax

Parameter	Data Type	Description
ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.

```
unsigned int MicoEFB_SPIXferDone (MicoEFBCtx_t, *ctx) ;
```

This function is used to query whether the interrupt-driven (i.e., non-blocking mode) SPI transfer has been completed or still in progress. Table 57 describes the parameters in the MicoEFB_SPIXferDone function syntax and Table 58 describes the return values..

Table 57: Description of Parameters of MicoEFB_SPIXferDone Function Syntax

Parameter	Data Type	Description
Ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.

.

Table 58: Description of the Return Values of MicoEFB_SPIXferDone Function

Value	Description
1	The SPI transfer is complete.
0	The SPI transfer is still in progress.

MicoEFB_RegisterSPIISR Function

```
void MicoEFB_RegisterSPIISR (MicoEFBCtx_t *ctx, SPIDesc_t *spi)
;
```

This function can be used by the customer to register a user-implemented interrupt handler for the SPI. Once the customer defines own interrupt handler, the functions MicoEFB_SPIISR and MicoEFB_SPIXferDone are no longer useful. Table 59 describes the parameters in the MicoEFB_RegisterSPIISR function syntax..

Table 59: Description of the Parameters in the MicoEFB_RegisterSPIISr Function Syntax

Parameter	Data Type	Description
ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.
spi	SPIDesc_t *	Pointer to the user-implemented SPI interrupt descriptor. It contains a pointer to the user-implemented SPI interrupt handler and a user-defined and implemented data structure that is shared by the program and interrupt handler.

MicoEFB_I2CRead Function

```
char MicoEFB_I2CRead (MicoEFBCtx_t *ctx,
        unsigned char i2c_idx,
        unsigned char isMaster,
        unsigned char buffersize,
        unsigned char *buffer,
        unsigned char insert_start,
        unsigned char insert_stop,
        unsigned char address,
        unsigned int irqmode) ;
```

This function is used to read data over the I^{2}C interface in the EFB. The transfer can be performed in a polling (i.e., blocking) mode or an interrupt-driven (i.e., non-blocking) mode. In case of polling mode, control is transferred to the application upon completion (successful or otherwise) of the transfer. In case of interrupt-driven mode, control is transferred to the application immediately after setting up the transfer. By default, the interrupt-driven mode uses Lattice-provided interrupt handler. The customer can override this handler by implementing a handler in application code and then registering it with the LatticeMico EFB context (see functions MicoEFB_RegisterI2C1ISR and MicoEFB_RegisterI2C2ISR). Table 60 describes the parameter in the MicoEFB_I2CRead function syntax..

Table 60: Description of Parameters of MicoEFB_I2CRead Function Syntax

Parameter	Data Type	Description
ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context
isMaster	unsigned char	Is I^2C configured to be a Master (1) or a Slave (0)
i2c_idx	unsigned char	Use Primary (1) I^2C or Secondary (2) I^2C
address	unsigned char	The I^2C address of the slave from which data is to be read. NOTE: Only useful when I^2C is configured as a slave.
insert_start	unsigned char	Insert START at the start of the current transaction. 1 means insert START, 0 means otherwise. NOTE: Only useful when I^2C is configured as a master. NOTE: Refer to I^2C protocol specifications for more details on a REPEATED START condition.
insert_restart	unsigned char	Insert REPEATED START at the start of the current transaction. 1 means insert REPEATED START, 0 means otherwise. NOTE: Only useful when I^2C is configured as a master. NOTE: Refer to I^2C protocol specifications for more details on a REPEATED START condition.
insert_stop	unsigned char	Insert STOP at the end of current transaction. 1 means insert STOP, 0 means otherwise. NOTE: Only useful when I^2C is configured as a master. NOTE: Refer to I^2C protocol specifications for more details on a STOP condition.
Buffer	unsigned char *	Pointer to the array that will store the data that is received. The application is responsible for allocating memory for this array.
buffersize	unsigned int	The number of bytes to be read in the current transaction.
Irqmode	unsigned int	Is the transfer to be performed in polling (i.e., blocking) or interrupt-driven (non-blocking) mode.

MicoEFB_I2CWrite Function

```
char MicoEFB_I2CWrite (MicoEFBCtx_t *ctx,
        unsigned char i2c_idx,
        unsigned char isMaster,
        unsigned char buffersize,
        unsigned char *buffer,
        unsigned char insert_start,
        unsigned char insert_stop,
        unsigned char address,
        unsigned int irqmode) ;
```

This function is used to write data over the I^2C interface in the EFB. The transfer can be performed in a polling (i.e., blocking) mode or an interrupt-driven (i.e., non-blocking) mode. In case of polling mode, control is transferred to the application upon completion (successful or otherwise) of the transfer. In case of interrupt-driven mode, control is transferred to the application immediately after setting up the transfer. By default, the interrupt-driven mode uses Lattice-provided interrupt handler. The customer can override this handler by implementing a handler in application code and then registering it with the LatticeMico EFB context (see functions MicoEFB_RegisterI2C1ISR and MicoEFB_RegisterI2C2ISR). Table 61 describes the parameter in the MicoEFB_I2CWrite function syntax..

Table 61: Description of Parameters of MicoEFB_I2CWrite Function Syntax

Parameter	Data Type	Description
Ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context
isMaster	unsigned char	Is I^2C configured to be a Master (1) or a Slave (0)
i2c_idx	unsigned char	Use Primary (1) I^2C or Secondary (2) I^2C
Address	unsigned char	The I^2C address of the slave from which data is to be read. NOTE: Only useful when I^2C is configured as a slave.
insert_start	unsigned char	Insert START at the start of the current transaction. 1 means insert START, 0 means otherwise. NOTE: Only useful when I^2C is configured as a master. NOTE: Refer to I^2C protocol specifications for more details on a REPEATED START condition.
insert_restart	unsigned char	Insert REPEATED START at the start of the current transaction. 1 means insert REPEATED START, 0 means otherwise. NOTE: Only useful when I^2C is configured as a master. NOTE: Refer to I^2C protocol specifications for more details on a REPEATED START condition.

Table 61: Description of Parameters of MicoEFB_I2CWrite Function Syntax (Continued)

insert_stop	unsigned char	Insert STOP at the end of current transaction. 1 means insert STOP, 0 means otherwise. NOTE: Only useful when I^2C is configured as a master. NOTE: Refer to I^2C protocol specifications for more details on a STOP condition.
Buffer	unsigned char *	Pointer to the array that contains the data to be transmitted.
buffersize	unsigned int	The extra number of bytes to be transferred in the current transaction. NOTE: If user input is N, EFB will transfer N+1 bytes of data.
Irqmode	unsigned int	Is the transfer to be performed in polling (i.e., blocking) or interrupt-driven (non-blocking) mode.

MicoEFB_I2C1ISR and MicoEFB_I2C2ISR Functions

```
void MicoEFB_I2C1ISR (MicoEFBCtx_t *ctx) ;
void MicoEFB_I2C2ISR (MicoEFBCtx_t *ctx) ;
```

These functions implement the interrupt handler for the I^2C 1 and 2 interface in the EFB. These are the default implementation. Table 62 describes the parameters in the MicoEFB_I2CISR function syntax..

Table 62: Description of Parameters of MicoEFB_I2C1ISR Function Syntax

Parameter	Data Type	Description
Ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.

MicoEFB_I2C1XferDone and MicoEFB_I2C2XferDone Functions

```
char MicoEFB_I2C1XferDone (MicoEFBCtx_t, *ctx) ;
char MicoEFB_I2C2XferDone (MicoEFBCtx_t, *ctx) ;
```

This function is used to query whether the interrupt-driven (i.e., non-blocking mode) I^2C transfer has been completed or still in progress. Table 63 describes the parameters in the MicoEFB_I2C1XferDone and MicoEFB_I2C2XferDone function syntax and Table 64 describes the return values.

Table 63: Description of Parameters of MicoEFB_I2C1XferDone Function Syntax

Parameter	Data Type	Description
Ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.

Table 64: Description of the Return Values of MicoEFB_I2C1XferDone and MicoEFB_I2C2XferDone Function.

Value	Description
1	The I^2C transfer is complete.
0	The I^2C transfer is still in progress.

MicoEFB_RegisterI2CISR and MicoEFB_RegisterI2C2ISR Functions

```
void MicoEFB_RegisterI2C1ISR (MicoEFBCtx_t *ctx, I2CDesc_t
*i2c) ;
void MicoEFB_RegisterI2C2ISR (MicoEFBCtx_t *ctx, I2CDesc_t
*i2c) ;
```

These functions can be used by the customer to register a user-implemented interrupt handler for the I^2C 1 and 2. Once the customer defines own interrupt handler, the functions MicoEFB_I2C1ISR, MicoEFB_I2C1XferDone, MicoEFB_I2C2ISR, and MicoEFB_I2C2XferDone are no longer useful. Table 65 describes the parameters in the MicoEFB_RegisterI2C1ISR and MicoEFB_RegisterI2C2ISR function syntax..

Table 65: Description of the Parameters in the MicoEFB_RegisterI2C1ISR and MicoEFB_RegisterI2C2ISR Function Syntax

Parameter	Data Type	Description
ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.
i2c	I2CDesc_t *	Pointer to the user-implemented I^2C interrupt descriptor. It contains a pointer to the user-implemented I^2C interrupt handler and a user-defined and implemented data structure that is shared by the program and interrupt handler.

MicoEFB_RegisterPCSxISR Functions

```
void MicoEFB_RegisterPCS0ISR (MicoEFBCtx_t *ctx, PCS0Desc_t
*pcs) ;
void MicoEFB_RegisterPCS1ISR (MicoEFBCtx_t *ctx, PCS1Desc_t
*pcs) ;
void MicoEFB_RegisterPCS2ISR (MicoEFBCtx_t *ctx, PCS2Desc_t
*pcs) ;
void MicoEFB_RegisterPCS3ISR (MicoEFBCtx_t *ctx, PCS3Desc_t
*pcs) ;
void MicoEFB_RegisterPCS4ISR (MicoEFBCtx_t *ctx, PCS4Desc_t
*pcs) ;
```

These functions can be used by the customer to register a user-implemented interrupt handler for PCS 0, PCS 1, PCS 2, PCS 3 and PCS 4 respectively.

Table 66 describes the parameters in the MicoEFB_RegisterPCSxISR function syntax, where *x* is numbers 0 through 4.

Table 66: Description of Parameters in the MicoEFB_RegisterPCSxISR Function Syntax

Parameter	Data Type	Description
ctx	MicoEFBCtx_t *	Pointer to the EFB context representing a valid EFB context.
pcs	PCSxDesc_t *	Pointer to the user-implemented PCSx (where x is from 0 through 4) interrupt descriptor. It contains a pointer to the user-implemented PCSx interrupt handler and a user-defined and implemented data structure that is shared by the program and interrupt handler.

LatticeMico8 Microcontroller Software Support

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

Device Driver

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

Type Definitions

This section describes the type definitions for the EFB device context structure. This structure, shown in Figure 13, contains the EFB 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 EFB 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 67 describes the parameters of the EFB device context structure shown in Figure 13

Device Context Structure

Figure 13 shows the EFB device context structure for MachXO2/Platform Manager 2.

Figure 13: EFB Device Context Structure (MachXO2/Platform Manager 2)

```
struct st_MicoEFBCtx_t {
    const char *   name;
    size_t   base;
    unsigned char   intrLevel;
    unsigned char   i2c1_en;
    unsigned char   i2c2_en;
    unsigned char   spi_en;
    unsigned char   spi_irqen;
    unsigned char   timer_en;
    unsigned char   ufm_en;
    unsigned int   ufm_addr;
    unsigned int   ufm_mem;
} MicoEFBCtx_t;
```

Figure 14 shows the EFB device context structure for MachXO3L.

Figure 14: EFB Device Context Structure (MachXO3L)

```
struct st_MicoEFBCtx_t {
    const char *   name;
    size_t   base;
    unsigned char   intrLevel;
    unsigned char   i2c1_en;
    unsigned char   i2c2_en;
    unsigned char   spi_en;
    unsigned char   spi_irqen;
    unsigned char   timer_en;
} MicoEFBCtx_t;
```

Table 67 describes the EFB device context parameters for MachXO2/Platform Manager 2.

Table 67: EFB Device Context Parameters (MachXO2/Platform Manager 2)

Parameter	Data Type	Description
name	const char *	component 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
i2c1_en	unsigned char	Primary I^{2}C enabled
i2c2_en	unsigned char	Secondary I^{2}C enabled
spi_en	unsigned char	SPI enabled
spi_irqen	unsigned char	SPI interrupt enabled
timer_en	unsigned char	Timer enabled

Table 67: EFB Device Context Parameters (MachXO2/Platform Manager 2) (Continued)

Parameter	Data Type	Description
ufm_en	unsigned char	UFM enabled
ufm_addr	unsigned int	UFM Start Address
ufm_mem	unsigned int	UFM memory size

Table 68 describes the EFB device context parameters for MachXO3L.

Table 68: EFBDevice Context Parameters (MachXO3L)

Parameter	Data Type	Description
name	const char *	component 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
i2c1_en	unsigned char	Primary I^2C enabled
i2c2_en	unsigned char	Secondary I^2C enabled
spi_en	unsigned char	SPI enabled
spi_irqen	unsigned char	SPI interrupt enabled
timer_en	unsigned char	Timer enabled

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 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 options 1 or 2.

__MICO_NO_INTERRUPTS__

This preprocessor macro definition disables code and data structures within the device driver that allow the EFB to be used in an interrupt driven mode. It is not defined by default.

__MICOEFB_NO_I2C_INTERRUPT__

This preprocessor macro definition disables code and data structures within the device driver that allow the I^{2}C to be used in an interrupt driven mode. It is not defined by default.

__MICOEFB_NO_SPI_INTERRUPT__

This preprocessor macro definition disables code and data structures within the device driver that allow the SPI to be used in an interrupt driven mode. It is not defined by default.

__MICOEFB_NO_TC_INTERRUPT__

This preprocessor macro definition disables code and data structures within the device driver that allow the Timer/Counter to be used in an interrupt driven mode. It is not defined by default.

__MICOEFB_NO_UFM_INTERRUPT__

This preprocessor macro definition disables code and data structures within the device driver that allow the UFM to be used in an interrupt driven mode. It is not defined by default.

__MICOEFB_NO_UFM_ADDR_CHECK__

MachXO2/Patform Manager 2 only. This preprocessor macro definition disables code and data structures within the device driver that validate the UFM Address when calling any UFM function. It is not defined by default.

The 'function-like' macro definitions are used in the LatticeMico8 software drivers to access the component's Register Map in order to perform certain operations. All 'function-like' macro definitions take input parameters that are used in performing the operations encoded within the macro. Table 69 describes the 'function-like' macros available in the LatticeMico8 EFB driver header file 'MicoEFB.h'. Table 70 through Table 74 also show how each macro can be used by the software developer in the application code.

Table 69: C Preprocessor Function-like Macros For EFB

Macro Name	Second Argument to Macro	Description
MICO_EFB_READ_IRQR	The 8-bit value read from the IRQ register.	This macro reads a character from the Interrupt Register.
MICO_EFB_WR_IRQR	The 8-bit value to be written to the IRQ register.	This macro writes a character to the Interrupt Register.
Note: The first argument to the macro is the EFB address.		

Table 70: Preprocessor Function-like Macros For SPI

Macro Name	Second Argument to Macro	Description
MICO_EFB_SPI_READ_CR0	The 8-bit value read from the Control Register 0.	This macro reads a character from the Control Register 0.
MICO_EFB_SPI_WRITE_CR0	The 8-bit value to be written to the Control Register 0.	This macro writes a character to the Control Register 0.
MICO_EFB_SPI_READ_CR1	The 8-bit value read from the Control Register 1.	This macro reads a character from the Control Register 1.
MICO_EFB_SPI_WRITE_CR1	The 8-bit value to be written to the Control Register 1.	This macro writes a character to the Control Register 1.
MICO_EFB_SPI_READ_CR2	The 8-bit value read from the Control Register 2.	This macro reads a character from the Control Register 2.
MICO_EFB_SPI_WRITE_CR2	The 8-bit value to be written to the Control Register 2.	This macro writes a character to the Control Register 2.
MICO_EFB_SPI_READ_BR	The 8-bit value read from the Clock Pre-scale Register.	This macro reads a character from the Clock Pre-scale Register.
MICO_EFB_SPI_WRITE_BR	The 8-bit value to be written to the Clock Pre-scale Register	This macro reads a character from the Clock Pre-scale Register.
MICO_EFB_SPI_READ_CSR	The 8-bit value read from the Master Chip Select Register	This macro reads a character from the Master Chip Select Register.
MICO_EFB_SPI_WRITE_CSR	The 8-bit value to be written to the Master Chip Select Register.	This macro writes a character to the Master Chip Select Register.
MICO_EFB_SPI_READ_RXDR	The 8-bit value read from the Receive Data Buffer.	This macro reads a character from the Receive Data Buffer.
MICO_EFB_SPI_WRITE_TXDR	The 8-bit value to be written to the Transmit Data Buffer.	This macro writes a character to the Transmit Data Buffer.
MICO_EFB_SPI_READ_SR	The 8-bit value read from the Status Register.	This macro reads a character from the Status Register.

Table 70: Preprocessor Function-like Macros For SPI (Continued)

Macro Name	Second Argument to Macro	Description
MICO_EFB_SPI_READ_IRQSR	The 8-bit value read from the Interrupt Request Register.	This macro reads a character from the Interrupt Request Register.
MICO_EFB_SPI_WRITE_IRQSR	The 8-bit value to be written to the Interrupt Request Register.	This macro writes a character to the Interrupt Request Register.
MICO_EFB_SPI_READ_IRQENR	The 8-bit value read from the Interrupt Request Enable Register.	This macro reads a character from the Interrupt Request Enable Register.
MICO_EFB_SPI_WRITE_IRQENR	The 8-bit value to be written to the Interrupt Request Enable Register.	This macro writes a character to the Interrupt Request Enable Register.
Note: The first argument to the macro is the EFB address.		

Table 71: C Preprocessor Function-like Macros For I^2C

Macro Name	Second Argument to Macro	Description
MICO_EFB_I2C_READ_CR	The 8-bit value read from the Control Register.	This macro reads a character from the Control Register.
MICO_EFB_I2C_WRITE_CR	The 8-bit value to be written to the Control Register.	This macro writes a character to the Control Register.
MICO_EFB_I2C_READ_CMDR	The 8-bit value read from the command Register.	This macro reads a character from the command Register.
MICO_EFB_I2C_WRITE_CMDR	The 8-bit value to be written to the command Register.	This macro writes a character to the command Register.
MICO_EFB_I2C_READ_PRESCALE_LO	The 8-bit value read from the lower byte of Clock Pre-scale Register.	This macro reads a character from the lower byte of Clock Pre-scale Register.
MICO_EFB_I2C_WRITE_PRESCALE_LO	The 8-bit value to be written to the lower byte of Clock Pre-scale Register.	This macro writes a character to the lower byte of Clock Pre-scale Register.
MICO_EFB_I2C_READ_PRESCALE_HI	The 8-bit value read from the upper byte of Clock Pre-scale Register.	This macro reads a character from the upper byte of Clock Pre-scale Register.
MICO_EFB_I2C_WRITE_PRESCALE_HI	The 8-bit value to be written to the upper byte of Clock Pre-scale Register.	This macro writes a character to the upper byte of Clock Pre-scale Register.
MICO_EFB_I2C_READ_RXDR	The 8-bit value read from the Receive Data Buffer.	This macro reads a character from the Receive Data Buffer.
MICO_EFB_I2C_WRITE_TXDR	The 8-bit value to be written to the Transmit Data Buffer.	This macro writes a character to the Transmit Data Buffer.
MICO_EFB_I2C_READ_SR	The 8-bit value read from the Status Register.	This macro reads a character from the Status Register.

Table 71: C Preprocessor Function-like Macros For I^{2}C (Continued)

Macro Name	Second Argument to Macro	Description
MICO_EFB_I2C_READ_IRQSR	The 8-bit value read from the Interrupt Request Register.	This macro reads a character from the Interrupt Request Register.
MICO_EFB_I2C_WRITE_IRQSR	The 8-bit value to be written to the Interrupt Request Register.	This macro writes a character to the Interrupt Request Register.
MICO_EFB_I2C_READ_IRQENR	The 8-bit value read from the Interrupt Request Enable Register.	This macro reads a character from the Interrupt Request Enable Register.
MICO_EFB_I2C_WRITE_IRQENR	The 8-bit value to be written to the Interrupt Request Enable Register.	This macro writes a character to the Interrupt Request Enable Register.

Note: For the primary I^{2}C, the first argument to the macro is EFB address. For the secondary I^{2}C, the first argument to the macro is EFB address plus 0x0a.

Table 72: C Preprocessor Function-like Macros with Two Arguments for Timer/Counter

Macro Name	Second Argument to Macro	Description
MICO_EFB_TIMER_READ_CR0	The 8-bit value read from the Control Register 0.	This macro reads a character from the Control Register 0.
MICO_EFB_TIMER_WRITE_CR0	The 8-bit value to be written to the Control Register 0.	This macro writes a character to the Control Register 0.
MICO_EFB_TIMER_READ_CR1	The 8-bit value read from the Control Register 1.	This macro reads a character from the Control Register 1.
MICO_EFB_TIMER_WRITE_CR1	The 8-bit value to be written to the Control Register 1.	This macro writes a character to the Control Register 1.
MICO_EFB_TIMER_READ_CR2	The 8-bit value read from the Control Register 2.	This macro reads a character from the Control Register 2.
MICO_EFB_TIMER_WRITE_CR2	The 8-bit value to be written to the Control Register 2.	This macro writes a character to the Control Register 2.
MICO_EFB_TIMER_GET_CNT	The 16-bit Counter Value.	This macro gets the current counter value.
MICO_EFB_TIMER_GET_TOP	The 16-bit Current Top Counter value .	This macro gets the current top counter value.
MICO_EFB_TIMER_GET_OCR	The 16-bit Current Compare Counter Value	This macro gets the current top compare counter value.
MICO_EFB_TIMER_GET_ICR	The 16-bit Current Capture Counter Value.	This macro gets the current capture counter value.
MICO_EFB_TIMER_SET_TOP	The 16-bit Top Counter Value to be set .	This macro set the Top Counter Value.
MICO_EFB_TIMER_SET_OCR	The 16-bit Compare Counter Value to be set .	This macro set the Compare Counter Value.

Table 72: C Preprocessor Function-like Macros with Two Arguments for Timer/Counter (Continued)

Macro Name	Second Argument to Macro	Description
MICO_EFB_TIMER_READ_SR	The 8-bit value read from the Status Register.	This macro reads a character from the Status Register.
MICO_EFB_TIMER_READ_IRQSR	The 8-bit value read from the Interrupt Request Register.	This macro reads a character from the Status Register.
MICO_EFB_TIMER_WRITE_IRQSR	The 8-bit value to be written to the Interrupt Request Register.	This macro writes a character to the Interrupt Request Register.
MICO_EFB_TIMER_READ_IRQENR	The 8-bit value read from the Interrupt Request Enable Register.	This macro reads a character from the Interrupt Request Enable Register.
MICO_EFB_TIMER_WRITE_IRQENR	The 8-bit value to be written to the Interrupt Request Enable Register.	This macro writes a character to the Interrupt Request Enable Register.
Note: The first argument to the macro is the EFB address.		

Table 73: C Preprocessor Function-like Macros with Two Arguments for UFM (MachXO2/Platform Manager 2 only)

Macro Name	Second Argument to Marco	Description
MICO_EFB_UFM_WRITE_CR	The 8-bit value to be written to the Control Register.	This macro reads a character from the Control Register.
MICO_EFB_UFM_READ_CR	The 8-bit value read from the Control Register.	This macro writes a character to the Control Register
MICO_EFB_UFM_WRITE_TXDR	The 8-bit value to be written to the Transmit FIFO Data Buffer.	This macro writes a character to the Transmit FIFO Data Buffer
MICO_EFB_UFM_READ_SR	The 8-bit value read from the Status Register.	This macro reads a character from the Status Register.
MICO_EFB_UFM_READ_RXDR	The 8-bit value read from the Receive FIFO Data Buffer.	This macro reads a character from the Receive FIFO Data Buffer.
MICO_EFB_UFM_WRITE_IRQSR	The 8-bit value to be written to the Interrupt Request Register.	This macro writes a character to the Interrupt Request Register.
MICO_EFB_UFM_READ_IRQSR	The 8-bit value read from the Interrupt Request Register.	This macro reads a character from the Interrupt Request Register.
MICO_EFB_UFM_WRITE_IRQENR	The 8-bit value to be written to the Interrupt Request Enable Register.	This macro writes a character to the Interrupt Request Enable Register.
MICO_EFB_UFM_READ_IRQENR	The 8-bit value to be written to the Interrupt Request Enable Register.	This macro writes a character to the Interrupt Request Enable Register.

Table 74: C Preprocessor Function-like Macros With One Argument for Timer/Counter

Macro Name	Description
MICO_EFB_TIMER_STOP	This macro stops the timer.
MICO_EFB_TIMER_RESET	This macro resets the timer
MICO_EFB_TIMER_START	This macro starts the timer.

Functions

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

MicoEFBInit Function

```
void MicoEFBInit (MicoEFBCtx_t *ctx);
```

This is the EFB initialization function. It disable all interrupts (should be enabled by user as required) and stops the timer.

Table 75 describes the parameter in the MicoEFBInit function syntax.

Table 75: MicoEFBInit Function Parameter

Parameter	Description
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.

MicoEFBISR Function

```
void MicoEFBISR (MicoEFBCtx_t *ctx);
```

This is the EFB Interrupt handler. Each EFB component has it's own interrupt handler and must be implemented by the developers in user code to reflect their application behavior.

Table 76 describes the parameter in the MicoEFBISR function syntax.

Table 76: MicoEFBISR Function Parameter

Parameter	Description
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.

This function and the following component interrupt handler will not be declared if user specifies __MICO_NO_INTERRUPTS__ preprocessor.

MicoEFB_I2C1ISR Function

```
void MicoEFB_I2C1ISR (MicoEFBCtx_t *ctx);
```

This function will not be declared if user specifies __MICOEFB_NO_I2C_INTERRUPT__ preprocessor. This is the primary I^2C Interrupt handler.

Table 77 describes the parameter in the MicoEFB_I2C1ISR function syntax.

Table 77: MicoEFB_I2C1ISR Function Parameter

Parameter	Description
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.

MicoEFB_I2C2ISR Function

```
void MicoEFB_I2C2ISR (MicoEFBCtx_t *ctx);
```

This function will not be declared if user specifies __MICOEFB_NO_I2C_INTERRUPT__ preprocessor. This is the secondary I^2C Interrupt handler.

Table 78 describes the parameter in the MicoEFB_I2C2ISR function syntax.

Table 78: MicoEFB_I2C1ISR Function Parameter

Parameter	Description
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.

MicoEFB_SPIISR Function

```
void MicoEFB_SPIISR (MicoEFBCtx_t *ctx);
```

This function will not be declared if user specifies __MICOEFB_NO_SPI_INTERRUPT__ preprocessor. This is the SPI Interrupt handler.

Table 79 describes the parameter in the MicoEFB_TimerISR function syntax.

Table 79: MicoEFB_TimerISR Function Parameter

Parameter	Description
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.

MicoEFB_TimerISR Function

```
void MicoEFB_TimerISR (MicoEFBCtx_t *ctx);
```

This function will not be declared if user specifies __MICOEFB_NO_TC_INTERRUPT__ preprocessor. This is the Timer Interrupt handler.

MicoEFB_SPITransfer Function

```
char MicoEFB_SPITransfer (MicoEFBCtx_t *ctx,
              unsigned char isMaster,
              unsigned char slvIndex,
              unsigned char insertStart,
              unsigned char insertStop,
              unsigned char *txBuffer,
              unsigned char *rxBuffer,
              unsigned char bufferSize);
```

This function initiates a SPI transfer that receives and transmits a configurable number of bytes. Table 80 describes the parameters in the MicoEFB_SPITransfer function syntax.

Table 80: MicoEFB_SPITransfer Function Parameters

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid `MicoEFBCtx_t*` structure representing a valid EFB instance.	
unsigned char	master or slave	master or slave 1 = master 0 = slave
unsigned char	Assert chip select at start of transfer	1 = insert 0 = do not insert
unsigned char	Deassert chip select at end of transfer	1 = insert 0 = do not insert
unsigned char	Bytes to be transmitted	min 1 and max 256
unsigned char	Bytes to be received	
unsigned char	Number of bytes to transfer	0 refers to 1 byte. 255 refers to 256 bytes

MicoEFB_SPITxData Function

```
char MicoEFB_SPITxData (MicoEFBCtx_t *ctx,
            unsigned char data);
```

This function initiates a byte transmission. Table 81 describes the parameters in the MicoEFB_SPITxData function syntax.

Table 81: MicoEFB_SPITxData (MicoEFBCtx_t *ctx Function Parameters

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t* structure representing a valid EFB instance.	
unsigned char	Data Bytes to be transferred	

Table 82 describes the values returned by the MicoEFB_SPIRxData Function.

Table 82: Values Returned by the MicoEFB_SPITxData Function

Return Value	Description
0	Successful writes

MicoEFB_SPIRxData Function

```
char MicoEFB_SPIRxData (MicoEFBCtx_t *ctx,
            unsigned char *data);
```

This function initiates a byte receive. Table 83 describes the parameters in the MicoEFB_SPIRxData (MicoEFBCtx_t *ctx function syntax.

Table 83: MicoEFB_SPITxData (MicoEFBCtx_t *ctx Function Parameters

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t* structure representing a valid EFB instance.	
unsigned char	Data Bytes received	

Table 84 describes the values returned by the MicoEFB_SPIRxData Function.

Table 84: Values Returned by the MicoEFB_SPITxData Function

Return Value	Description
0	Status register contents after receive of data.

MicoEFB_I2CStart Functions

```
char MicoEFB_I2CStart (MicoEFBCtx_t *ctx,
            unsigned char i2c_idx,
            unsigned char read,
            unsigned char address,
            unsigned char restart);
```

This function initiates a START command provided the I^{2}C master can get control of the bus.

Table 85 describes the parameters in the MicoEFB_I2CStart function syntax.

Table 85: MicoEFB_I2CStart Function Parameters

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t* structure representing a valid EFB instance.	
unsigned char	I^{2}C index	Value is 1 or 2
unsigned char	Read or write operation	0 = Write 1 = Read
unsigned char	Slave address	
unsigned char	Is this a 'repeated start' event or a new 'start' event?	1 = (repeated start) 0 = (new start)

Table 86 describes the values returned by the MicoEFB_I2CStart function.

Table 86: Values Returned by the MicoEFB_I2CStart Function

Return Value	Description
0	Successful writes
-1	Failed to receive ack during addressing
-2	Failed to receive ack when writing data.
-3	Arbitration lost during operations

MicoEFB_I2CWrite Function

```
char MicoEFB_I2CWrite (MicoEFBCtx_t *ctx,
        unsigned char i2c_idx,
        unsigned char slv_xfer,
        unsigned char buffersize,
        unsigned char *data,
        unsigned char insert_start,
        unsigned char insert_restart,
        unsigned char insert_stop,
        unsigned char address);
```

This function performs block writes. In addition it also allows the user to optionally:

1. Initiate a START command prior to performing the block writes if the I^{2}C is an I^{2}C master.

2. Initiate a STOP command after performing the block writes if the I^{2}C is an I^{2}C master.
3. Hold the SCL line low (i.e. clock stretching) after performing the block writes if the I^{2}C is an I^{2}C slave.

Table 87 describes the parameters in the MicoEFB_I2CWrite function syntax.

Table 87: MicoEFB_I2CWrite Function Parameters

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t* structure representing a valid EFB instance.	
unsigned char	I^{2}C index	Value is 1 or 2
unsigned char	I^{2}C master or slave	0 = master 1 = slave
unsigned char	Number of bytes to be transferred	min 1 and max 256
unsigned char	Buffer containing the data to be transferred	
unsigned char	Master: Insert Start (or repeated Start) prior to data transfer	1 = insert 0 = do not insert
unsigned char	Master: Insert Stop at end of data transfer. Slave: Stretch clock at end of transfer.	1 = insert 0 = do not insert
unsigned char	Master: Repeated Start inserted prior to data transfer (this argument is valid only is 'insert_start' is 1	1 = insert 0 = do not insert
unsigned char	Slave address	

Table 88 describes the values returned by the MicoEFB_I2CWrite function.

Table 88: Values Returned by the MicoEFB_I2CWrite Function

Return Value	Description
0	Successful writes
-1	Failed to receive ack during addressing
-2	Failed to receive ack when writing data.
-3	Arbitration lost during operations

MicoEFB_I2CRead Function

```
char MicoEFB_I2CRead (MicoEFBCtx_t *ctx,
        unsigned char i2c_idx,
        unsigned char slv_xfer,
        unsigned char buffersize,
        unsigned char *data,
        unsigned char insert_start,
        unsigned char insert_restart,
        unsigned char insert_stop,
        unsigned char address);
```

Table 89 describes the parameters in the MicoEFB_I2CRead function syntax.

Table 89: MicoEFB_I2CRead Function Parameters

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t* structure representing a valid EFB instance.	
unsigned char	I^2C index	Value is 1 or 2
unsigned char	I^2C master or slave	0 = master 1 = slave
unsigned char	Number of bytes to be transferred	min 1 and max 256
unsigned char	Buffer to put received data in to	
unsigned char	Master: Insert Start (or repeated Start) prior to data transfer	1 = insert 0 = do not insert
unsigned char	Master: Repeated Start inserted prior to data transfer (this argument is valid only is 'insert_start' is 1	1 = insert 0 = do not insert
unsigned char	Master: Insert Stop at end of data transfer. Slave: Stretch clock at end of transfer.	1 = insert 0 = do not insert
unsigned char	Slave address	

Table 90 describes the values returned by the MicoEFB_I2CRead function.

Table 90: Values Returned by the MicoEFB_I2CRead Function

Return Value	Description
0	Successful reads.
-1	Failed to receive ack during addressing
-2	Failed to receive ack when writing data.
-3	Arbitration lost during operations

MicoEFB_TimerStart Function

```
void MicoEFB_TimerStart (MicoEFBCtx_t *ctx,
              unsigned char mode,
              unsigned char ocmode,
              unsigned char sclk,
              unsigned char cclk,
              unsigned char interrupt,
              unsigned int timerCount,
              unsigned int compareCount);
```

This function sets up timer configuration and starts timer. Table 91 describes the parameters in the MicoEFB_TimerStart function syntax.

Table 91: MicoEFB_TimerStart Function Parameters

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t* structure representing a valid EFB instance.	
unsigned char	Timer mode	0 - Watchdog 1 - Clear Timer on Compare (CTC) Match 2 - Fast PWM 3 - Correct PWM
unsigned char	Timer counter output signal's mode	0 - Always zero 1 - Toggle on TOP match (non-PWM modes) Toggle on OCR match (Fast PWM mode) Toggle on OCR match (Correct PWM mode) 2 - Clear on TOP match (non-PWM modes) Clear on TOP match, set on OCR match (Fast PWM mode) Clear on OCR match when CNT incrementing, set on OCR match when CNT decrementing (Correct PWM mode) 3 - Set on TOP match (non-PWM modes) Set on TOP match, clear on OCR match (Fast PWM mode) Set on OCR match when CNT incrementing, clear on OCR match when CNT decrementing (Correct PWM mode)
unsigned char	Clock source selection	0 - WISHBONE clock (rising edge) 2 - On-chip oscillator (rising edge) 4 - WISHBONE clock (falling edge) 6 - On-chip oscillator (falling edge)

Table 91: MicoEFB_TimerStart Function Parameters (Continued)

Parameter	Description	Note
unsigned char	Divider selection	0 - Static 0 1 - sclk/1 2- sclk/8 3 - sclk/64 4 - sclk/256 5 - sclk/1024
unsigned char	interrupt	1 = Enable interrupts 0 = Disable interrupts
unsigned char	Timer TOP value	maximum 0xFFFF
unsigned char	Timer OCR (compare) value	maximum 0xFFFF

Note

The MicoEFB_ UFMCmdCall Function, MicoEFB_ UFMSetAddr Function, MicoEFB_ UFMErase Function. MicoEFB_ UFMRead Function, and MicoEFB_ UFMWrite Function apply only to MachXO2/Platform Manager 2.

MicoEFB_ UFMCmdCall Function

```
char MicoEFB_UFMCmdCall(MicoEFBCtx_t *ctx,
        unsigned long opcode,
        unsigned char enable);
```

Table 92 describes the parameter in the MicoEFB_ UFMCmdCall function syntax.

Table 92: MicoEFB_ UFMCmdCall Function Parameters (MachXO2/Platform Manager 2 only)

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.	
unsigned char	UFM 4 bytes opcode	
unsigned char	Enable the close frame signal	1: enable 0: disable

MicoEFB_ UFMSetAddr Function

```
char MicoEFB_UFMSetAddr (MicoEFBCtx_t *ctx,
        unsigned int address);
```

Table 93 describes the parameter in the MicoEFB_ UFMSetAddr function syntax.

Table 93: MicoEFB_ UFMSetAddr Function Parameters (MachXO2/Platform Manager 2 only)

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.	
unsigned int	UFM 14 bits memory address	

Table 94 describes the values returned by the MicoEFB_ UFMSetAddr Function.

Table 94: Values Returned by the MicoEFB_ UFMSetAddr Function (MachXO2/Platform Manager 2 only)

Return Value	Description
0	Sucess
-1	Invalid address

MicoEFB_ UFMErase Function

```
char MicoEFB_UFMErase (MicoEFBCtx_t *ctx);
```

Table 95 describes the parameter in the MicoEFB_ UFMErase function syntax.

Table 95: MicoEFB_ UFMErase Function Parameters (MachXO2/Platform Manager 2 only)

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.	

Table 96 describes the values returned by the MicoEFB_ UFMErase Function.

Table 96: Values Returned by the MicoEFB_ UFMErase Function (MachXO2/Platform Manager 2 only)

Return Value	Description
0	Erase successfully
-1	Erase fails

MicoEFB_ UFMRead Function

```
char MicoEFB_UFMRead (MicoEFBCtx_t *ctx,
        unsigned char *data,
        unsigned char pagesize);
```

Table 97 describes the parameter in the MicoEFB_ UFMRead function syntax.

Table 97: MicoEFB_ UFMRead Function Parameters (MachXO2/Platform Manager 2 only)

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.	
unsigned char	Data Bytes received	
unsigned char	Number of Page to read	Max Page is 16

Table 98 describes the values returned by the MicoEFB_ UFMRead Function.

Table 98: Values Returned by the MicoEFB_ UFMRead Function (MachXO2/Platform Manager 2 only)

Return Value	Description
0	read successfully
-1	invalid page size
-2	Read from invalid memory

MicoEFB_ UFMWrite Function

```
char MicoEFB_UFMWrite (MicoEFBCtx_t *ctx,
        unsigned char *data);
```

Table 99 describes the parameter in the MicoEFB_ UFMWrite function syntax.

Table 99: MicoEFB_ UFMWrite Function Parameters (MachXO2/Platform Manager 2 only)

Parameter	Description	Note
MicoEFBCtx_t*	Pointer to a valid MicoEFBCtx_t structure representing a valid EFB instance.	
unsigned char	Data Bytes transfer	

Table 100 describes the values returned by the MicoEFB_ UFMWrite Function.

Table 100: Values Returned by the MicoEFB_ UFMWrite Function (MachXO2/Platform Manager 2 only)

Return Value	Description
0	Write successfully
-1	Write to invalid memory

Software Usage Example

This section provides an example of using the EFB. The example is shown in Figure 15 and assumes the presence of an EFB component named “efb” and a UART component named “uart”.

Figure 15: Example of Using EFB

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

void MicoEFB_I2C1ISR (MicoEFBCtx_t *ctx)
{
  return;
}

void MicoEFB_I2C2ISR (MicoEFBCtx_t *ctx)
{
  return;
}

void MicoEFB_SPIISR (MicoEFBCtx_t *ctx)
{
  return;
}

void MicoEFB_TimerISR (MicoEFBCtx_t *ctx)
{
  return;
}

static unsigned char GetCharacter(MicoUartCtx_t *pUart)
{
  char c;
  MicoUart_getC (pUart, &c);
  return(c);
}

static void SendCharacter(MicoUartCtx_t *pUart, char c)
{
  MicoUart_putC (pUart, c);
  return;
}

/************************************
 * main program                     *
 ************************************/
int main()
{
  MicoEFBCtx_t *efb = &efb_machxo2_efb;
  size_t efb_address = (size_t) efb->base;

  MicoUartCtx_t *uart = &uart_core_uart;
  size_t uart_address = (size_t ) uart->base;
```

Figure 15: Example of Using EFB (Continued)

```
    unsigned char iter;
    unsigned int snapshot;
    for (iter = 0; iter < 2; iter++) {
      MicoEFB_TimerStart (efb, 0, 0, 0, 2, 0, 0xFFFF, 0x0);
      MicoSleepMicroSecs (100*(iter+1));
      MICO_EFB_TIMER_GET_CNT (efb_address, snapshot);
      MICO_EFB_TIMER_STOP (efb_address);

      // Print snapshot
      SendCharacter (uart, (char)(snapshot>>8));
      SendCharacter (uart, (char)(snapshot));
      SendCharacter (uart, '\n');
    }

      return(0);
  }
```

Revision History

Component Version	Description
1.0	Initial release.
1.1	▶ Support added for new UFM features: Read, Write, Command Call, Set Address and Erase. ▶ Support added for I^2C with repeated start command. ▶ Content added to optimize the I^2C and SPI by inserting assembly code. ▶ Fixed issues with I2C_Read in master mode. ▶ Updated document with new corporate logo.
1.2	▶ Added support for User-Managed Timer Reset mode. ▶ Added I/O port ufm_sn to Table 8, UFM I/O Port (MachXO2/ Platform Manager 2 Only).
1.3	▶ Added LatticeMico32 microprocessor software support.
1.4	▶ Added support for a new preprocessor option, "__MICOEFB_NO_UFM_ADDR_CHECK__", which allows the UFM function to bypass the UFM address checking.
1.5	▶ Fixed issues with EFB instance name.
1.6	▶ Added support for Platform Manager 2 and MachXO3L devices.

.

Trademarks