LatticeSC MPI/System Bus

April 2010 Technical Note TN1085

Introduction

The embedded system bus on the LatticeSC™ ties all of the programmable elements together in a bus framework. There are two types of interfaces on the system bus, master and slave. A master interface has the ability to perform actions on the bus such as writes and reads to and from a specific address. A slave interface responds to the actions of a master by accepting data and address on a write and providing data on a read. The system bus has a memory map which describes each of the slave peripherals that is connected on the bus. Using the addresses listed in the memory map, a master interface can access each of the slave peripherals on the system bus. Any and all peripherals on the system bus can be used at the same time. Table 1 lists all of the available user peripherals on the system bus after device power-up.

Table 1. System Bus User Peripherals

Peripheral	Abbreviation	Interface Type
Micro Processor Interface	MPI	Master
User Master Interface	UMI	Master
User Slave Interface	USI	Slave
Serial Management Interface (PLL, DLL, User Logic)	SMI	Slave
flexiPCS™ Interface	PCS	Slave
Direct FPGA Access	DFA	Slave

The peripherals listed in Table 1 can be added when the system bus module is created using IPexpress™ (isp-LEVER® IPexpress).

Figure 1 illustrates the existing peripherals on the system bus. The gray boxes are available only during configuration. The Status and Config box refers to internal system bus registers. Refer to TN1080, LatticeSC sysCONFIG™ Usage Guide for configuration options.

This document describes all the interfaces listed in Table 1 in detail to help the user utilize the desired functions of the system bus.

Figure 1. LatticeSC System Bus Interfaces

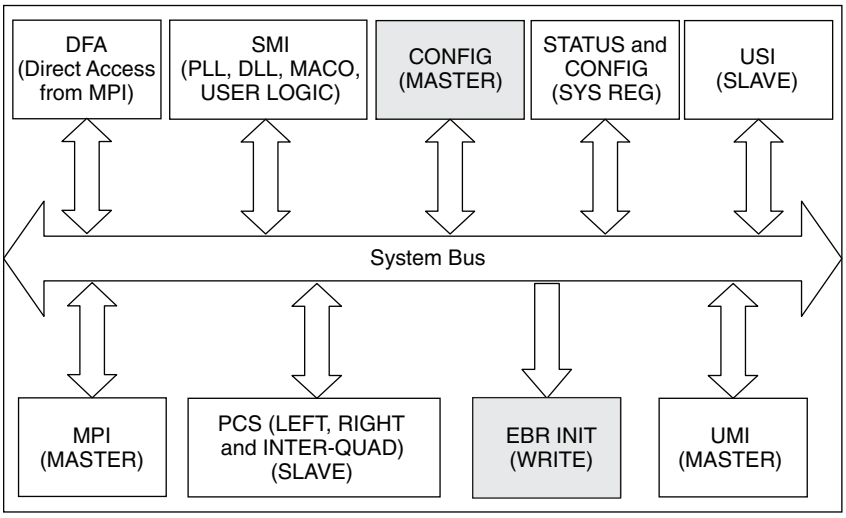
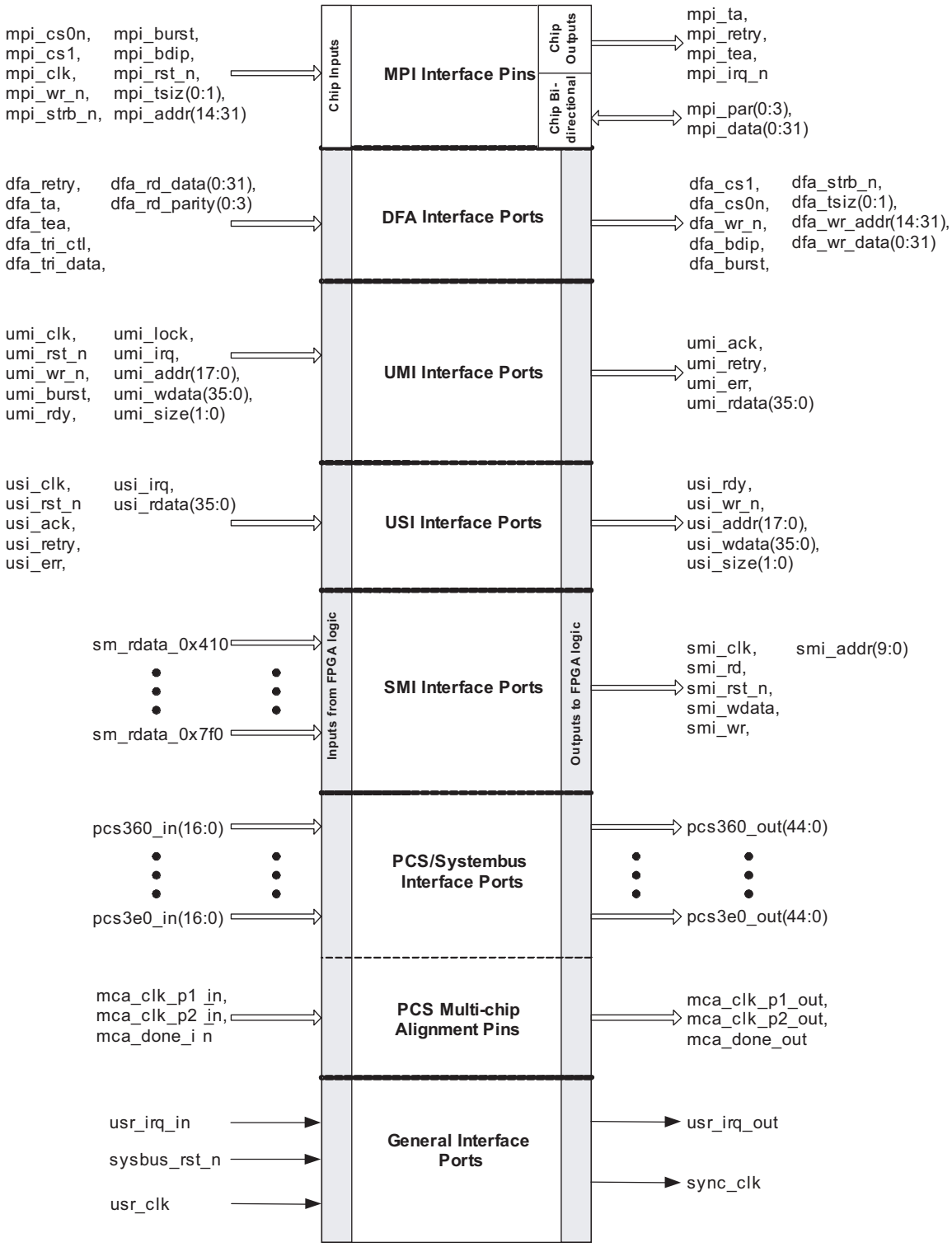


Figure 2 shows a complete system bus I/O diagram with all possible peripherals enabled in IPexpress. These interfaces and their corresponding I/Os are discussed in detail throughout the rest of this document.

Figure 2. System Bus I/O Diagram



Definitions

PCS	Physical Coding Sublayer
MACO	Masked Array for Cost Optimization
MSB	Most Significant Byte
MSb	Most Significant Bit
LSB	Least Significant Byte
LSb	Least Significant Bit
MPI	Microprocessor Interface
UMI	User Master Interface
USI	User Slave Interface
SMI	Serial Management Interface
DFA	Direct FPGA Access
EBR	Embedded Block RAM
PLL	Phase-Lock Loop
HDL	Hardware Description Language
DLL	Delay Lock Loop
ASB	Application Specific Block
PPC	PowerPC

System Bus

Multi Master Capability

The system bus is a multiple master bus. This means that there can be more than one bus master present on the bus at the same time. A single bus arbiter controls the traffic on the bus by ensuring only one master has access to the bus at any time. This bus arbiter monitors a number of different requests to use the bus and decides which request is currently the highest priority. The configuration logic has the highest priority and overrides all normal user interfaces. By default, all master interfaces have equal priority when requesting the embedded system bus, and a fair round-robin scheme is used to rotate arbitration priority. Optionally, one can specify a priority of low=1, medium=2, or high=3 for each master interface in IPexpress. As a result, if more than one master interface is waiting for the system bus, an interface with higher priority will be granted the bus over one with a lower priority. If two requesting interfaces share the same priority, the round-robin scheme will rotate arbitration priority.

System Bus Address Range

The system bus decodes 18 bits of byte addressable memory (256 Kbytes of addresses) in the device address space.

Table 2 describes the address range for the entire device address space.

Table 2. Entire System Bus Address Range

Begin Address	End Address	Size (B=Bytes)	Description
0x00000	0x0003F	64 B	System bus registers (see Table 20).
0x00040	0x003FF	960 B	Reserved.
0x00400	0x007FF	1 KB	64 Serial Memory Interfaces. For access to PLL/DLL configuration memory and user control/status registers in FPGA (64 SMIs, 128 bits each).
0x00800	0x2FFFF	190 KB	User slave interface (USI) baseline. For pre-configuration usage of address range 0x10000-0x2FFFF, see TN1080, LatticeSC sysCONFIG Usage Guide.
0x30000	0x364FF	25,856 B	Left PCS slave interface. See the LatticeSC/M Family flexiPCS Data Sheet for more detail.
0x36500	0x37FFF	6,912 B	Reserved.
0x38000	0x3E4FF	25,856 B	Right PCS slave interface. See the LatticeSC/M Family flexiPCS Data Sheet for more detail.
0x3E500	0x3EEFF	2,560 B	Reserved.
0x3EF00	0x3EFFF	256 B	Inter-Quad PCS slave interface. See the LatticeSC/M Family flexiPCS Data Sheet for more detail.
0x3F000	0x3FFFF	4 KB	Reserved.

System Bus Clock (HCLK)

The system bus is a synchronous element that has an internal clock. This clock is sometimes referred to as the HCLK. Each of the peripherals on the system is also synchronous and has an interface clock. This peripheral clock is the clock that the FPGA designer sources to the peripheral to clock data in and out of the user interface and eventually onto the system bus. The system bus itself needs to be sourced by a clock (HCLK). This main system bus clock is the clock on which all of the traffic will run through the system bus. As traffic is passed to and from each peripheral, a domain change will occur from the system bus clock domain to the peripheral domain. This domain change is handled inside the system bus. The internal system bus HCLK maximum frequency specification can be found in the LatticeSC/M Family Data Sheet.

Note that the system bus interface drives the SYNC_CLK output port, which is synchronous to HCLK.

The system bus clock can be driven from several sources: CCLK, MPI or USER. At power-up, before the bitstream is loaded into the LatticeSC (pre-configuration) the system bus clock source is selected via the mode pins. After the bitstream has been successfully loaded into the LatticeSC, the system bus clock source is selected based on an IPexpress option.

CCLK

Before a bitstream is loaded and during device configuration and reconfiguration the system bus clock defaults to the configuration clock (CCLK). The CCLK is either an input or output of the LatticeSC, depending on the configuration mode pins state. The CCLK is driven internally on the system bus from the configuration logic block. There is no input or output pin associated with the CCLK on the system bus module. Using the CCLK is for configuration only and cannot be simulated.

MPI

If the user has selected to program the device via the MPI, then the microprocessor clock will drive the system bus during configuration. In post configuration, the MPI option is set via the bitstream (IPexpress option) and will select the clock on the MPI to drive the system bus. The MPI clock corresponds to the input MPI_CLK on the system bus module.

USER

The USER clock allows the FPGA design to source a clock to drive the system bus clock. This USER clock is provided on the USR_CLK input port of the system bus. Selecting the USR_CLK as the source of HCLK is an IPexpress option.

The USER clock is driven from the FPGA design and thus can be driven by any signal in the user's design. If the USER clock is derived in any way from the output of a PLL, changing the PLL control register will not be allowed. When changing the PLL control register, the output clock from the PLL will stop for a period of time. If this clock stops, the USER clock will stop and thus the HCLK will stop. If the HCLK stops, the system bus will lock. The PLL control register will not be able to be changed and the FPGA will lock. To prevent this condition, do not drive this signal with a PLL output or do not change the PLL control register for this PLL.

Oscillator

The LatticeSC internal oscillator drives the CCLK during master mode bitstream configuration. To use the oscillator after configuration, set the source of the system bus clock to USER in IPexpress. Also instantiate the OSCA oscillator block in the design to drive the system bus USER clock pin. Various frequency rates can be selected for the oscillator via sysCONFIG. More information on the internal oscillator frequency rate selection can be found in TN1080, LatticeSC sysCONFIG Usage Guide. In simulation, however, the oscillator clock model driving the system bus HCLK has a fixed 100MHz frequency.

System Bus Interrupts

The system bus has the ability to generate and accept interrupt signals from several peripherals. These interrupts can then be used to alert either on-chip modules or external modules. An internal interrupt can be generated from the FPGA design, the configuration block during device configuration or from the PCS block. The interrupt cause register (0x00010) contains a bit for each source of an interrupt. When one of the peripherals sets an interrupt, the particular bit in the cause register will bet set to a 1. To clear the interrupt cause bit, a master interface will need to write a 1 to the particular bit to clear it. Note that clearing the interrupt cause bit does not remove the source of the interrupt.

Some peripherals on the system bus can pass interrupts found in the interrupt cause register to an interrupt output signal on their interface. For a peripheral to pass interrupts to its interrupt output signal, its corresponding interrupt enable register must be properly provisioned. There is an interrupt enable register for user (0x00012), and MPI (0x00013). When a bit is set to a 1 in the interrupt enable register, an interrupt will be passed to the corresponding interrupt output signal (active low MPI_IRQ_N output for MPI and active high USR_IRQ_OUT for user). When the interrupt output signal for the peripheral indicates an interrupt was created, a master should read the interrupt cause register to determine the source of the interrupt. Table 3 summarizes the interrupt enable register and output signal for both MPI and user.

Table 3. Interrupt Enable Registers and Outputs

Interface	Interrupt Enable Register	Interrupt Output Signal	Notes
USER	0x00012	USR_IRQ_OUT	Only the UMI can write to 0x00012
MPI	0x00013	MPI_IRQ_N	Only the MPI can write to 0x00013

The interrupt cause register (0x00010), user interrupt enable register (0x00012) and MPI interrupt enable register (0x00013) are designed such that each bit corresponds to the same interrupt source across all three registers. Bit 1 of all three registers, for example, always corresponds to a PCS interrupt source. The correlation between bit number and interrupt source is illustrated in Table 4.

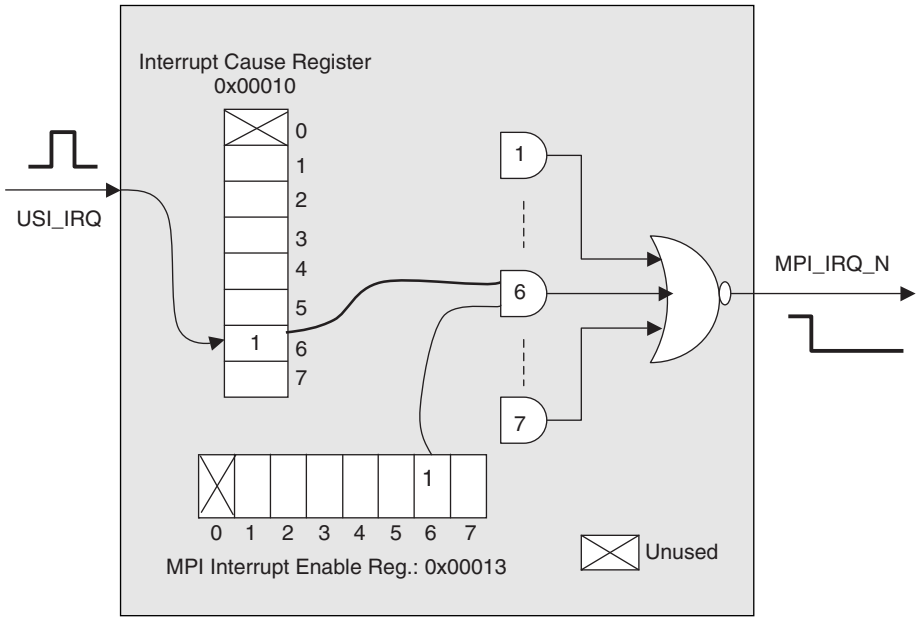
Table 4. Correlation Between Bit Number and Interrupt Source

Interrupt Source/ Signal	Unused	PCS	MPI	CFG_ERR	CGF_DATA	UMI_IRQ	USI_IRQ	USR_IRQ_IN
Interrupt Cause Register (0x00010)	Unused	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
USER Interrupt Enable Register (0x00012)	Unused	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
	When any of the bits in 0x00012 is set high, a rising edge on the interrupt source/signal (as reflected in the corresponding interrupt cause bit in 0x00010) causes signal USR_IRQ_OUT to go high.							
MPI Interrupt Enable Register (0x00013)	Unused	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
	When any of the bits in 0x00013 is set high, a rising edge on the interrupt source/signal (as reflected in the corresponding interrupt cause bit in 0x00010) causes MPI_IRQ_N to go low.							

For more information on the interrupt cause register and interrupt enable registers, refer to the system bus memory map in Table 20.

For example (as illustrated in Figure 3), to pass interrupts created by the USI interface to MPI_IRQ_N, the MPI must write a 1 to bit 6 of 0x00013. When the USI sends an interrupt via a USI_IRQ pulse (larger than one USI_CLK period), bit 6 of the interrupt cause register (0x00010) will go to a 1. This, combined with the 1 on bit 6 of 0x00013, will then drive MPI_IRQ_N pin low to indicate an interrupt to the microprocessor. The microprocessor should then read the interrupt cause register to determine that the interrupt came from the USI. Assuming no more interrupts are generated, the MPI can clear bit 6 of 0x00010 by writing a 1 to it. This also sets the MPI_IRQ_N signal back to 1.

Figure 3. USI_IRQ to MPI_IRQ_N Interrupt



System Bus General Signals

Table 5 describes the general interface signals on the system bus. These signals are not part of any of the specific peripherals mentioned in Table 1.

Table 5. System Bus General Interface Signals

Signal	Direction	Description	Notes
SYSBUS_RST_N	Input	Active low system bus reset	Resets Internal system bus logic
USR_IRQ_IN	Input	Active high user interrupt	
USR_IRQ_OUT	Output	Active high user output interrupt	Signal responds to interrupt when interrupt enable register 0x00012 is set by UMI.
USR_CLK	Input	USER clock	This interface signal is generated only when the USER clock is selected as the system bus clock source (in ispLEVER IPexpress).
SYNC_CLK	Output	Clock synchronous to HCLK (system bus clock)	

Address/Data Bus Ordering

The system bus handles bus transfers of address and data at all of its interfaces. The orientation of the address bus and the data bus may be different, depending on which peripheral is accessed. Internally on the bus, the address bus is always oriented the same way, whereas the data bus is dependent on the driving master/slave interface.

Address Bus Ordering

There are 18 address bits on the system bus. These address lines can be driven by any of the master interfaces on the system bus. Except for the DFA interface, the address bus on a slave interface will always be provided to the slave with bit 0 as the LSb. For the master interfaces (MPI, UMI) the address bus is dependent on the master. The UMI will use bit 17 as the MSb and bit 0 as the LSb. The MPI will match the PowerPC address bus orientation, which is bit 0 as MSb and bit 31 as LSb, even though only bits 14 through 31 (18 bits) are available on the LatticeSC MPI interface (see Table 21). More information on this address bus and its connections is found in the MPI section of this document. Table 6 lists the address bus bit ordering of different interfaces.

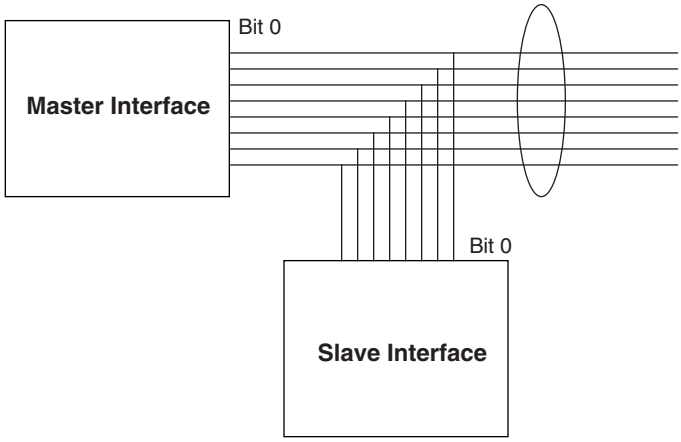
Table 6. Address Bit Ordering for Different Master/Slave Interfaces

	Address Bus Bit Ordering ADDRESS[MSb:LSb]	
Interface	MSb	LSb
MPI	14	31
UMI	17	0
USI	17	0
DFA	14	31
SMI	9	0

Data Bus Bit Orientation

The data bus on the system bus is 36 bits wide: 32 bits for data and four bits for parity. On the system bus, the data bits are passed unchanged through the peripherals. So the orientation of the data on the system bus is dependent on the data driving master/slave interface. This means that bit 0 on the master interface will be bit 0 on the slave interface, bit 1 will be bit 1, etc. So if bit 0 is the LSb on the master interface then bit 0 will be the LSb on the slave interface as shown in Figure 4. Care must be taken when accessing an address location to make sure data bits are used properly in terms of LSb and MSb.

Figure 4. System Bus Data Bus Bit Mapping



Different data interfaces on the system bus do not all follow the same rule in terms of bus bit orientation. The value of the data bus can be interpreted with bit 0 being either the MSb or LSb. For example, for single byte accesses, the MPI interface is always oriented with bit 0 being the MSb (DATA[0:7]), whereas the UMI interface is oriented with bit 0 being the LSb (DATA[7:0]). Table 7 shows the data bus bit orientation for all user accessible interfaces, in addition to the MPI.

Table 7. Data Bus Bit Orientation for Different System Bus Interfaces

	Data Bus Bit Ordering DATA[MSb:LSb]					
	8-Bit		16-Bit		32-Bit	
Interface	MSb	LSb	MSb	LSb	MSb	LSb
MPI	0	7	0	15	0	31
UMI	7	0	15	0	31	0
USI	Follows Master					
DFA	0	7	0	15	0	31

Internal Data Bus

The internal data bus is labeled D(35:0). Bits D(31:0) are used to carry the data between peripherals. Bits D(35:32) are used to carry parity to the peripherals. Except for MPI, parity is not checked internally by any of the peripherals on the system bus. Checking parity on the MPI is enabled via bit 5 of system bus address 0x0000A (MPI_PAR_CHK), and only checks the MPI parity on a write operation. An MPI parity error on a write always results in an MPI interrupt to the system bus interrupt cause register (0x00010, bit 2). Even or odd parity is set in isp-LEVER IPexpress. Internal system bus registers will generate parity for read operations. UMI/USI parity is only passed through the system bus and its interfaces. Parity is mapped to byte lanes as shown in Table 8. The orientation of the data bus D(31:0) depends on the master interface driving the data bus, but the parity bits are always stored on D(35:32).

Table 8. Internal Data Bus Bit Mapping

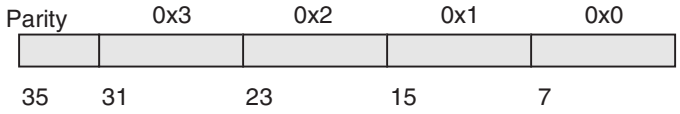
Internal Data Bus	MPI	User Master Interface	Description
D(35)	MPI_PAR(3)	UMI_W/R_DATA(35)	Parity bit for D(31:24)
D(34)	MPI_PAR(2)	UMI_W/R_DATA(34)	Parity bit for D(23:16)
D(33)	MPI_PAR (1)	UMI_W/R_DATA(33)	Parity bit for D(15:8)
D(32)	MPI_PAR(0)	UMI_W/R_DATA(32)	Parity bit for D(7:0)
D(31:24)	MPI_DATA(31:24) 32-bit LSB bit 31 = LSb	UMI_W/R_DATA(31:24)	Data byte 31:24 - LSB of the MPI in 32-bit mode.
D(23:16)	MPI_DATA(23:16) bit 23 = LSb	UMI_W/R_DATA(23:16)	Data byte 23:16
D(15:8)	MPI_DATA(15:8) 16-bit LSB, bit 15 = LSb	UMI_W/R_DATA(15:8)	Data byte 15:8 - LSB of the MPI in 16-bit mode
D(7:0)	MPI_DATA(7:0) MSb=0 8-bit LSb=7 16-bit/32-bit 0:7=MSB	UMI_W/R_DATA(7:0)	Data byte 7:0 - MSB of the MPI in 16/32-bit mode. In MPI 8-bit mode bit 0 is MSb and bit 7 is the LSb

32-bit Data Bus

When a master interface is configured for 32-bit operation, all 32 bits (D(31:0)) are used to carry the data. If the master interface is using parity, then the four parity bits will be carried on D(35:32) using the parity byte mapping previously described. Again, the orientation (MSb,LSb) of the data bus depends on which master/slave interface drives data onto the system bus. When using a 32-bit data bus the address resolution will be limited to 32-bit boundaries. Data will be carried on the bus such that the lowest address maps to byte lane D(7:0) and the highest address maps to byte D(31:24).

For example, a 32-bit read at address 0x00000 will read addresses 0x00000 - 0x00003. Address 0x00000 data will be on D(7:0), address 0x00001 data will be on D(15:8), address 0x00002 data will be on D(23:16), and address 0x00003 data will be on D(31:24) as shown in Figure 5.

Figure 5. 32-bit Data Bus

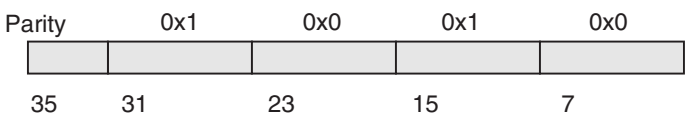


16-bit Data Bus

For 16-bit operation of a master interface, the data will be carried on D(15:0). The slave interface will only need to read data from D(15:0). The MSB and LSB of the data will be determined by how the data was driven onto the bus by the master/slave interface. When using a 16-bit data bus the address resolution will be limited to 16-bit boundaries. Data will be carried on the bus such that the lowest address maps to byte lane D(7:0) and the highest address maps to byte D(15:8). When using MPI, the same data will also be automatically replicated on D(31:16). When using UMI/USI, the user needs to replicate the data to the system bus on D(31:16).

For example a 16-bit read at address 0x00000 will read addresses 0x00000 - 0x00001. Address 0x00000 data will be on D(7:0) and address 0x00001 data will be on D(15:8) as shown in Figure 6.

Figure 6. 16-bit Data Bus

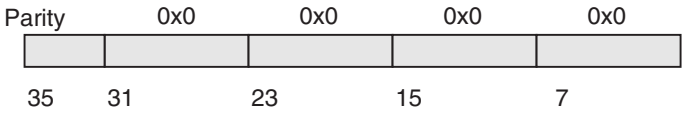


Note that a 16-bit data access (from MPI or UMI) to the PCS slave registers is not supported. Please refer to the PCS section of this document for more information.

8-bit Data Bus

For 8-bit operation of a master interface the data will be carried on D(7:0). The slave interface will only need to read and write data on D(7:0). When using MPI, the same data will also be automatically replicated on D(15:8), D(23:16), and D(31:24). When using UMI/USI, the user needs to replicate the data to the system bus on D(15:8), D(23:16), and D(31:24), as shown in Figure 7. The MSb and LSb of the data will be determined by the master/slave interface.

Figure 7. 8-Bit Data Bus



Synchronous/Asynchronous Mode

The MPI, UMI and USI interfaces on the system bus can be generated (in ispLEVER IPexpress) either synchronously or asynchronously to the system bus HCLK domain.

By default, all three interfaces are generated asynchronously to the system bus HCLK domain. Asynchronous mode is required when the clock clocking the peripheral interface is assumed to run asynchronously (different frequency/phase) to HCLK. In this case, all three peripherals (MPI, UMI, USI) include an asynchronous FIFO to decouple the peripheral clock domain from the internal HCLK domain. This FIFO introduces extra cycles of delay during a read or a write access through the system bus.

In ispLEVER IPexpress, it is possible to independently bypass the MPI, UMI and USI asynchronous FIFOs. This mode, referred to as synchronous mode, reduces the amount of access latency as a result of bypassing the FIFO. For MPI, this mode only requires selecting the MPI clock as the source of the system bus clock. For UMI and USI, a specific ispLEVER IPexpress option to make the peripheral synchronous to the system bus clock needs to be selected. UMI and USI modes also require that the peripheral clock (UMI_CLK or USI_CLK) be driven by SYNC_CLK in synchronous mode. Table 9 describes how to ensure that each of MPI, UMI or USI interface clocks is made synchronous to HCLK.

Table 9. Setting Synchronous clocks to HCLK

Interface	Interface Clock	How to Make Synchronous to HCLK
MPI	MPI_CLK	Select MPI_CLK as source of system bus clock in ispLEVER IPexpress.
UMI	UMI_CLK	Make UMI synchronous to system bus in ispLEVER IPexpress. Connect SYNC_CLK to UMI_CLK port.
USI	USI_CLK	Connect SYNC_CLK to USI_CLK port.

System Bus Time Out

Time out is a programmable feature that allows the system bus to interrupt current operation on the bus. There are two types of time out mechanisms that the system bus can exercise: Wait State time out, and Grant time out. In each case, an index is used to specify the length of time in HCLK cycles after which the system bus times out. Table 10 shows how an index relates to HCLK cycles. For example, if the index is at 5 and HCLK is running at 50 MHz, the system bus will timeout after 2^{10} * 20 ns ~= 20 microseconds. Note that a 0 index value means the system bus never times out.

Table 10. Time Out Index vs. HCLK Cycles Before Time Out

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HCLK Cycles	Never Time Out	2^2	2^4	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}	2^{28}	2^{31}

Wait State Time Out
This type of time out is designed to prevent the master on the bus from waiting indefinitely for the termination phase of an access (MPI_TA or UMI_ACK). Without a time out mechanism, the system bus could hang indefinitely. The Wait State time out index is independently set in both simulation and bitstream generation. Setting a value greater than 0 is recommended to avoid unintentional hanging of the system bus.

In simulation, the index is defined as a programmable parameter (WS_TIME_INDEX). The default value for WS_TIME_INDEX is 4 (256 cycles). The value of the parameter can be modified in the HDL code for simulation purpose. In Verilog, for example, assuming the system bus instance is “systembus_i”, the index can be set to 5 as follows:

defparam systembus_i.SYSBUSA_INST.SYSBUSA_sim_inst.WS_TIME_INDEX = 4’d5;

During bitstream generation, the index is defined as a sysCONFIG switch (WaitStateTimeOut). The default value of WaitStateTimeOut is 5 (2^{10} HCLK cycles). Please see TN1080, LatticeSC sysCONFIG Usage Guide for more information.

When the system bus times out as a result of a Wait State condition, the system bus asserts a transfer error (MPI_TEA or UMI_ERR) along with the termination of access signal (MPI_TA or UMI_ACK).

Grant Time Out
This type of time out is designed to prevent one master from indefinitely owning the system bus when another master is also trying to access it. The Grant time out can only be used when the master currently accessing the system bus has not locked it.

In simulation, the index is defined as a programmable parameter (GRANT_TIME_INDEX). The default value for GRANT_TIME_INDEX is 0 (never time out). The value of the parameter can be modified in the HDL code for simulation purpose. In Verilog, for example, assuming the system bus instance is “systembus_i”, the index can be set to 5 as follows:

defparam systembus_i.SYSBUSA_INST.SYSBUSA_sim_inst.GRANT_TIME_INDEX = 4’d5;

During bitstream generation, the index is defined as a sysCONFIG switch (GrantTimeOut). The default value of GrantTimeOut is 0 (never time out). Please see TN1080, LatticeSC sysCONFIG Usage Guide for more information.

When the system bus times out as a result of a Grant condition, the system bus first waits for the current transfer to end, then grants ownership of the bus to the other master.

System Bus Peripherals

The following section discusses each of the peripherals on the system bus in detail. Some of the peripherals on the system bus (e.g. Configuration) do not have FPGA design user ports and are fully contained inside the system bus. These peripherals’ registers are described in the memory map and do not have any user inputs or outputs on the system bus.

MPI

LatticeSC devices contain an embedded microprocessor interface (MPI) that can be used to interface any LatticeSC device to any MPC860/MPC8260 PowerPC microprocessor or compatible interface through the PowerPC peripheral bus. The MPI acts as a bridge between an external PowerPC processor and the embedded system bus.

Externally, the MPI acts as a slave peripheral interface through which a PowerPC can access the embedded features of the device. Internally the MPI acts as a master peripheral interface on the system bus to initiate data transfers as directed by the external processor.

Note: The MPI interface is not available on all LatticeSC packages. Also, some LatticeSC die/package combinations restrict the maximum allowable MPI DATA bus size. For full information on MPI restrictions for a die/package combination, please refer to the Pinout Information section of the LatticeSC/M Family Data Sheet.

The MPI is one element on the embedded system bus illustrated in Figure 1. The system bus provides multi-master/multi-slave communication between the MPI and the status and configuration interface, SMI, UMI, USI, and one or more PCS interface blocks as needed in each specific LatticeSC device.

The MPI is available prior to and optionally after configuration of the programmable logic in the LatticeSC. The MPI can be used prior to device configuration to identify, test, initialize, and download configuration data into the device. After configuration of the programmable logic, the MPI can be used to read back the configuration and internal status data, control parameters in the PLLs and the DLLs, access status and control registers for an embedded PCS block (if present), and interact with the user's design configured in programmable logic.

The MPI peripheral has two distinguishing characteristics from any other system bus interface:

- The MPI can be turned on at power-up, before device configuration, allowing the MPI to access the system bus control, status and configuration registers. All other peripherals can be present only after device configuration.
- When the user instantiates an system bus with MPI in a design, a register bit setting can keep MPI enabled even after the device is un-programmed.

MPI Interface

Table 11 shows the MPI signals used by the PowerPC to perform transactions with the LatticeSC device. The MPI is a fixed block on the LatticeSC array and the interface signals are mapped to dedicated pins on the LatticeSC device. PowerPC pins to LatticeSC pins mapping can be found in Appendix A.

Externally the MPI implements a 36-bit PowerPC bus slave, which internally drives the system bus as a master. Data bus width is selectable among 8 bits, 16 bits, and 32 bits with parity of 1, 2 or 4 bits, respectively (one parity bit for each active byte). Note that the MPI does not generate parity, but simply passes it from the PowerPC bus to the system bus and vice versa. The MPI interface can check parity on write, however, by setting bit 5 of system bus address 0x0000A (MPI_PAR_CHK), Any write parity error results in an MPI interrupt to the system bus interrupt cause register (0x00010, bit 2).

Table 11. MPI Signals to PowerPC Bus

Name	I/O	Description
MPI_RST_N	I	Resets the MPI interface on the system bus
MPI_CLK	I	This is the clock from the PowerPC. This clock input will clock the MPI. This clock may optionally clock the main system bus clock if selected. The MPI clock can be driven up to 66MHz operation.
MPI_TSIZ[0:1]	I	Transfer size ([0:1]: 00-double word, 10-word, 01-byte) The MPI_TSIZ pins connect directly up to the PowerPC TSIZ1 and TSIZ0. These pins select the size of the PowerPC data transaction. This is the transfer size of the data transaction from the microprocessor's perspective. This is different from the MODE pin selected size discussed later.
MPI_WR_N	I	Transfer type (0-write, 1-read) This signal indicates to the MPI whether the transaction initiated by the microprocessor is a read or a write. If the transaction is a read, data will be provided to the microprocessor from the address specified. If the transaction is a write, data will be written to the address specified by the microprocessor inside the LatticeSC device. This signal connects to the PowerPC RD//WR signal.
MPI_BURST	I	Indicates that a burst transfer is in progress when low. This signal informs the MPI that the PowerPC is performing a burst transaction using the PowerPC burst pin.
MPI_BDIP	I	Burst Data In Progress. This signal from the PowerPC will go low on the first clock of data during a burst and go high on the last clock of data of the burst transfer.

Table 11. MPI Signals to PowerPC Bus

Name	I/O	Description
MPI_STRB_N	I	This active low signal indicates the start of a transactions or the strobe. This pin is connected to the TS pin of the PowerPC.
MPI_CS0N/ MPI_CS1	I	Chip selects for active high (MPI_CS1) and active low (MPI_CS0N). Both of these chip selects must be active for the LatticeSC system bus logic to be selected. Typically CS1 is connected to logic 1 and CS0n is connected to a PowerPC CS pin. When selecting the DFA interface instead of the system bus, the values of these signals are inverted (MPI_CS1=0, MPI_CS0N=1).
MPI_ADDR[14:31]	I	The PowerPC address bus is 32 bits wide. The LatticeSC devices only support 18 bits of address space. The LatticeSC uses the least significant bits of the PowerPC address space using address bits 14:31.
MPI_DATA[0:31]	I/O	The PowerPC data bus can be up to 32 bits wide. Bit 0 is the MSb and bit 31 is the LSb. For multi-byte transfers, the most significant byte ([0:7]) has the lowest address. The PowerPC data pins D[0:31] connect directly to the LatticeSC pins MPI_DATA(0:31). Data pins not used by virtue of selecting 8-bit or 16-bit data widths are available as general-purpose user I/O. *Note: The MPI interface is not available on all LatticeSC packages. Also, some LatticeSC die/package combinations restrict the maximum allowable MPI DATA bus size. For full information on MPI restrictions for a die/package combination, please refer to the Pinout Information section of the LatticeSC/M Family Data Sheet.*
MPI_PAR[0:3]	I/O	Parity can be up to 4 bits wide (one bit per byte of data) depending on the data bus size. Parity connects directly to the PowerPC DP[0:3] pins. Parity pins not used by virtue of selecting 8-bit or 16-bit data widths are available as general-purpose user I/O.
MPI_TA	O	This active low signal indicates the transfer acknowledge from the LatticeSC device. This pin is connected to the TA pin of the PowerPC. For a single MPI write transaction the MPI_TA will come back on the next clock. See the Consecutive Writes with the MPI section of this document. For an MPI read transaction the MPI_TA will come back after the target slave responds. The difference in time it takes to complete a transaction depends on the slave that is accessed. Other dependencies are the clock rate of the system bus (HCLK), clock rate of the slave interface, and slave acknowledge protocol.
MPI_TEA	O	This active low signal indicates a transfer error acknowledge during the current transaction. More information on cause of this error can be found under the MPI Exceptions section of this document.
MPI_IRQ_N	O	Active-low interrupt request from the LatticeSC device. See the System Bus Interrupts section of this document.
MPI_RETRY	O	Active-low request for processor to relinquish the bus and retry the cycle. Exception signal indicating the LatticeSC device is not ready to accept the requested transaction. More information on the cause of this error can be found under MPI exceptions.
MODE[3:0]	I	MPI data width ([3:0]: 1010, 1011, 1110 = 8, 16, 32 bits, respectively) The MODE pins are used to select the type of bitstream configuration the LatticeSC device will utilize. More information on the MODE pins can be found in the Enabling the MPI section of this document. These pins do not appear on the system bus HDL model out of ispLEVER IPexpress.

MPI Single Beat Data Transfers

Any data transfer on the PowerPC bus has four phases: arbitration, address, data and termination. During the arbitration phase the processor initiates the transaction with a MPI_STRB_N pulse. The MPI samples the address and control inputs during the address phase, receives or provides data and asserts transfer acknowledge during the data phase, and de-assert signals during the termination phase. Figure 8 shows the PowerPC bus timing for a 32-bit wide data single beat MPI write transfer. Figure 9 shows the PowerPC bus timing for a 32-bit wide data single beat MPI read transfer. MPI_DATA [0:31]=0x11223344 is written/read to/from address 0x00004 in each case.

Figure 8. MPI Single Write Data Transfer Timing

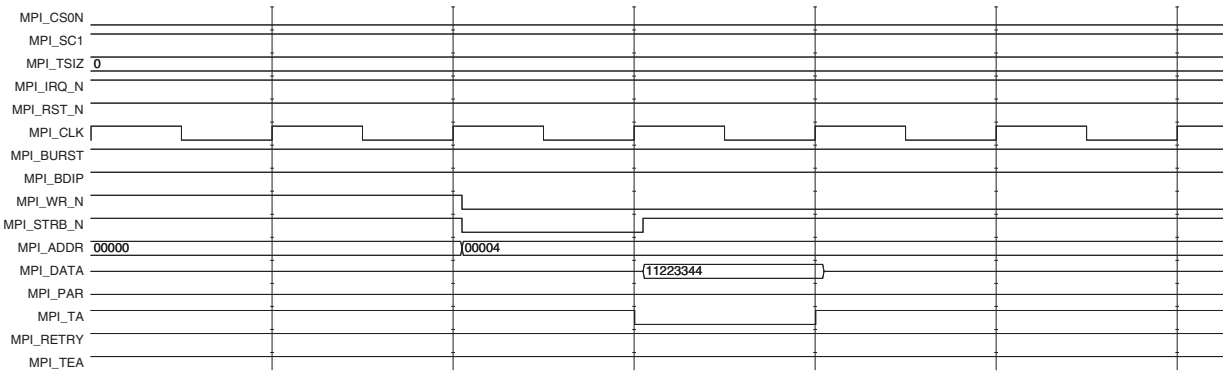
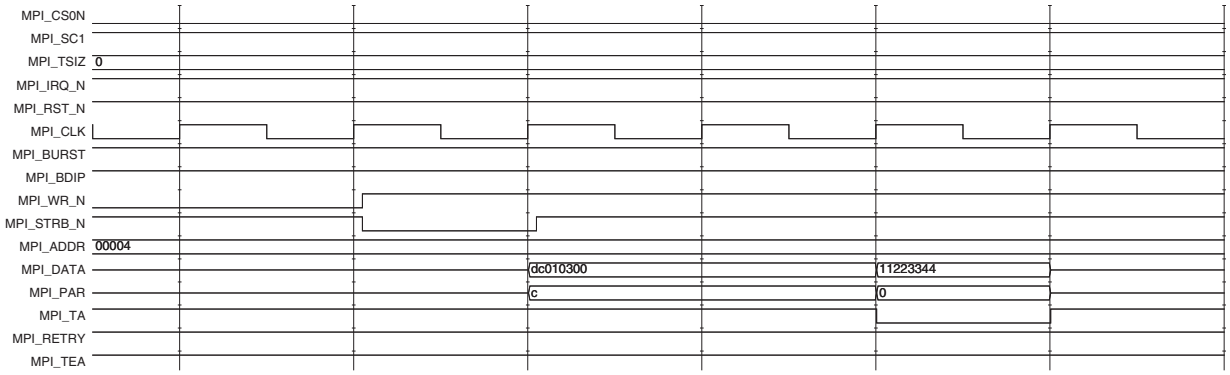


Figure 9. MPI Single Read Data Transfer Timing



MPI Burst Transfers

The MPI will support burst transfers of exactly 4 beats (32-bit width), 8 beats (16-bit width) or 16 beats (8-bit width), depending upon the selected data bus width. Burst transfers can be of any size that is compatible with the selected data bus width given the limitation that the MPI will handle 4, 8, or 16 beats as indicated.

The burst mechanism uses MPI_BURST to indicate that the transfer is a burst transfer and MPI_BDIP to indicate the duration of the burst.

Along with the address and transfer control signals, the PowerPC asserts MPI_BURST during the address phase of the transfer. In the data phase, the microprocessor asserts MPI_BDIP until the next to the last word is sent/received. The MPI continues to send/receive data until it detects MPI_BDIP de-asserted at the rising edge of MPI_CLK while MPI_TA is asserted.

Figure 10 shows the signal timing for a 32-bit wide data 4-beat burst write:

- The first write access is to 0x08000-0x08003 (MPI_DATA [0:31]=0x11111111).
- The second write access is to 0x08004-0x08007 (MPI_DATA [0:31]=0x22222222).
- The third write access is to 0x08008-0x0800B (MPI_DATA [0:31]=0x33333333).
- The fourth write access is to 0x0800C-0x0800F (MPI_DATA [0:31]=0x44444444).

Figure 10. MPI Burst Write Transfer Timing

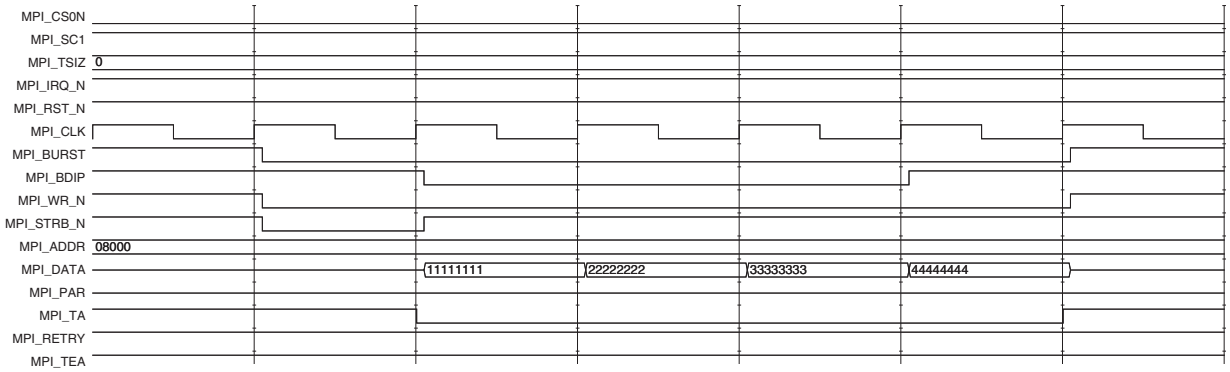
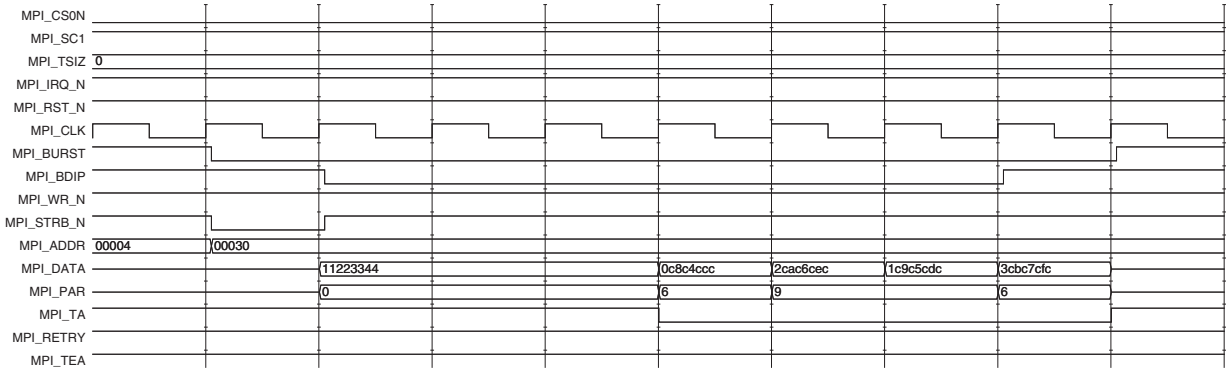


Figure 11 shows the signal timing for a 32-bit wide data 4-beat burst read:

- The first read access is to 0x00030-0x00033 (MPI_DATA [0:31]=0x0C8C4CCC).
- The second read access is to 0x00034-0x00037 (MPI_DATA [0:31]=0x2CAC6CEC).
- The third read access is to 0x00038-0x0003B (MPI_DATA [0:31]=0x1C9C5CDC).
- The fourth read access is to 0x0003C-0x0003F (MPI_DATA [0:31]=0x3CBC7CFC).

Figure 11. MPI Burst Read Transfer Timing



Consecutive Writes with the MPI

The MPI uses a single write post buffer implementation. This means that on each MPI write, the MPI_TA comes back on the next clock cycle to terminate the PowerPC transaction. Internally on the MPI and system bus, the data is not yet transmitted to the target. It will take several HCLK clock cycles before the target receives the data and terminates the transaction. Until this termination takes place any additional writes from the MPI will issue a retry (MPI_RETRY).

During consecutive write cycles, the number of PowerPC clock cycles (MPI_CLK) it takes until the next write can take place without a retry is variable. The variables include the source of the HCLK, whether the MPI interface is in synchronous or asynchronous mode, the target being accessed, and the target's termination protocol and clock. The user should add appropriate delay based on their board and system behavior.

MPI Exceptions

Three signals, MPI_TEA, MPI_RETRY and MPI_TA are monitored during the termination phase of a transfer. A normal termination is indicated when MPI_TA is asserted and both MPI_TEA and MPI_RETRY are de-asserted. If either MPI_TEA or MPI_RETRY are asserted, an MPI bus exception is indicated.

MPI_TEA is asserted for one MPI_CLK cycle to indicate either an internal system bus error, or a transfer with MPI_TSIZ larger than the data bus size, or physical data size selected by the MODE[3:0] inputs.

MPI_RETRY is asserted for one MPI_CLK cycle when the MPI is busy to request that the PowerPC relinquish the bus and reissue the current transfer. A retry is issued when the following occurs:

- The MPI gets a read transaction while its write FIFOs are not empty.
- The MPI gets a write transfer while its write FIFOs are full.
- The MPI receives a retry indication from the embedded system bus during a read transfer.

For burst transfers, the MPI should issue retry before acknowledging the first data phase; if MPI_RETRY is asserted after the first data phase of a burst transfer, it should be treated as a transfer error (MPI_TEA).

MPI Interrupts

The MPI logic on the system bus can generate an MPI interrupt to the system bus as a response to certain MPI write accesses. When an MPI interrupt occurs, it sets the MPI_IRQ bit of the Interrupt Cause Register (bit 2 of 0x00010). The interrupt can then be passed to either the MPI or USER output interrupt (as discussed in the system bus Interrupts section of this document).

There are three possible causes for an MPI interrupt during an MPI write access:

- The MPI logic on the system bus times out as a result of too many system bus slave RETRY responses.
- The system bus slave generates an internal error response to the access.
- The MPI logic on the system bus detects a parity check error (when MPI write parity checking is enabled).

MPI Synchronous vs. Asynchronous MPI Mode Latency

Table 12 shows the best case MPI to system bus access latency for both synchronous and asynchronous modes. The latency is defined as the number of cycles the MPI_TA signal stays high after MPI_STRBN goes low.

Table 12. MPI Sync/Asynchronous Best Case Latency[1]

Synchronous Mode		Asynchronous Mode				
Single	**Burst**	**Single Read**	**Single Write**	**16-Beat Burst**	**8-Beat Burst**	**4-Beat Burst**
2	3	7	6	7	7	7

1. Note that the first access in a series of MPI write cycles has zero latency. Subsequent write cycles may cause retries. When retries are present, the write latency is then the earliest cycle without a retry from the original MPI_STRB_N low.

Enabling the MPI

There are three conditions under which the MPI interface can be enabled: at power-up, via a register bit setting, and via user instantiation. If the MPI is not utilized at all in the LatticeSC design, the dedicated MPI pins can be used as general I/O pins for the user's design.

MPI at Power-up: To enable the MPI at power-up, prior to device configuration, the external MODE pins (see Table 13) must be set to specify one of the three MPI configuration modes, as specified in TN1080, LatticeSC sysCONFIG Usage Guide. If the MPI is not used, all of the MPI pins (MPI_TA, MPI_DATA, etc.) are tri-stated to a pull-up resistance during configuration. In any of the three MPI configuration modes, the MPI can access the registers in the system bus, PCS left, PCS right, and PCS inter-quad address ranges shown in Table 2.

Table 13. LatticeSC MPI Device Configuration Modes

M3	M2	M1	M0	Description
1	0	1	0	MPC860 8-bit
1	0	1	1	MPC860 16-bit
1	1	1	0	MPC860 32-bit

Data bus width is determined at power-up by the value presented on the MODE pins during the low-to-high transition of the INIT signal. The MODE pins select the data size of the transaction including the parity bits. MPC860 8-

bit mode will also use MPI_PAR(0) along with the data. MPC860 16-bit will use MPI_PAR(0:1) and MPC860 32-bit will use MPI_PAR(0:4). Unused MPI_PAR bits will be 3-stated to a pull-up resistance during configuration. All of the other MPI signals connect directly to the PowerPC bus. Pad locations vary depending upon device type, size, and package.

The data bus width selected by the MODE[3:0] pins is not related to the transfer size specified by MPI_TSIZ[0:1]. The bus width selected by the MODE[3:0] pins determines how many data pins are used by the MPI, while the transfer size is determined by the master interface on the PowerPC bus. The transfer size used by an external master must not exceed the selected data bus width; otherwise, the higher-order data bits will be lost and a bus exception is issued by the MPI.

In order to configure the LatticeSC device at power-up via MPI, the MPI PowerPC interface requires two different file types:

- The first file is a PowerPC PROM image for device configuration. ispVM® converts a sysCONFIG-generated bitstream to a PROM image.
- The second file is a configuration-time initialization file generated by sysCONFIG. This file contains initialization values for EBRs, PCS auto-configuration, and SMI memories. This initialization file is not needed for device configuration modes other than MPI.

More information on MPI configuration of the LatticeSC devices can be found in TN1080, LatticeSC sysCONFIG Usage Guide.

MPI via User Instantiation: To enable the MPI for use after device configuration and during normal operation, the user must instantiate the system bus with an MPI peripheral in the LatticeSC design. The system bus element along with the MPI and other peripherals is created using ispLEVER IPexpress.

MPI via a Register Bit Setting: When the user enables the MPI, as described in the MPI via User Instantiation section of this document, setting the MPI_USR_ENABLE bit (control register 0x00008, bit 2) will keep the MPI mode enabled even after the device is un-programmed. Un-programming the device still sets the programming mode to MPI (as defined by the user), independently of what the MODE pins are set to.

User Master Interface (UMI)

The User Master Interface (UMI) allows the FPGA design to perform transactions on the system bus. Through the UMI, the FPGA design has access to any and all of the slave peripherals on the system bus. Signals for the UMI are listed in Table 14.

Table 14. UMI Interface Signals

Signal	Type	Description
UMI_CLK	I	The main clock for the user master interface. This clock only clocks the interface registers. A domain change is made from the UMI_CLK domain to the system bus clock domain in asynchronous mode. The user master interface and the system bus can be made synchronous through an ispLEVER IPexpress option (synchronous mode). A frequency preference on this signal will constrain all of the inputs and outputs of the UMI.
UMI_RST_N	I	This active-low reset resets all of the controls in the user master interface. This should be pulsed once before any transactions can occur on the UMI. This is typically connected to the same reset as the GSR and pulsed at power up.
UMI_WDATA[35:0]	I	36-bit write data bus to system bus. This data bus is oriented with bits 35:32 used for parity while bits 31:0 are used for data. Parity is not calculated by any of the peripherals on the system bus, it is only carried. For less than 32-bit writes, the data must be replicated on the UMI_WDATA bus to fill the entire 32 bits. For a 16-bit transaction, bits 15:0 must be replicated on 31:16. For an 8-bit transaction, bits 7:0 must be replicated on 31:24, 23:16 and 15:8.
UMI_RDATA[35:0]	O	36-bit read data bus from system bus. This data bus is oriented with bits 35:32 used for parity while bits 31:0 are used for data. Parity is not calculated by any of the peripherals on the system bus, it is only carried.

Table 14. UMI Interface Signals (Continued)

Signal	Type	Description
UMI_ADDR[17:0]	I	This 18-bit address bus is used to select the address that a user master transaction will target. Bit 17 of the UMI_ADDR bus is the MSb while bit 0 is the LSb.
UMI_WR_N	I	This bit indicates whether the current transaction is a read or a write cycle. A low value indicates the current transaction is a read. Data is expected on the UMI_RDATA bus. A high value indicates the current transaction is a write. Data is driven onto UMI_WDATA bus.
UMI_LOCK	I	This active-high signal will be used to request ownership of the system bus.
UMI_BURST	I	Indicates this operation is a burst transfer.
UMI_SIZE[1:0]	I	Data width for transfer ([1:0]: 10-double word, 01-word, 00-byte, 11-invalid). Selects the size of the data transfer being done on the UMI. For 32-bit transactions all bits 31:0 will carry valid data. For 16-bit transactions only bits 15:0 will carry valid data. For 8-bit transactions only bits 7:0 will carry valid data.
UMI_RDY	I	Data strobe indicates address and data ready. Should be driven high when valid address is provided for read and valid address and data is present for write.
UMI_ACK	O	Acknowledge from master interface to indicate that it is ready for another operation.
UMI_RETRY	O	Asserted for one UMI_CLK cycle when the UMI is busy to request that the master relinquish the bus and reissue the current transfer. A retry is issued when the following occurs: • The UMI gets a read transaction while its write FIFOs are not empty. • The UMI gets a write transfer while its write FIFOs are full. • The UMI receives a retry indication from the embedded system bus.
UMI_ERR	O	Bus error response is asserted for one clock cycle when UMI_RDY is asserted. Indicates an internal system bus error.
UMI_IRQ	I	User logic master interface interrupt request. Active-high signal to indicate an interrupt to the system bus. This signal will map to the interrupt cause register UMI bit.

Locking the UMI

The system bus is a multi master bus and the UMI_LOCK signal will request ownership of the bus. In a multi master system bus application the UMI must be locked to guarantee uninterrupted multiple transactions through the UMI. Ownership of the bus is granted based on the priority of the master interface. If two masters request the bus at the same time, the interface with higher priority will obtain the bus. Once the UMI has locked the bus, the UMI_ACK will stay high indicating the interface is ready for a transaction.

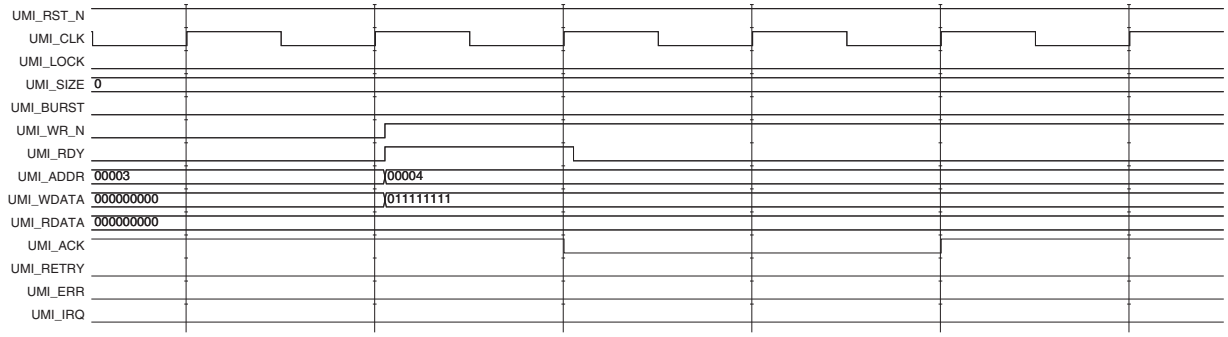
To release the lock on the system bus, a UMI_RDY pulse must be given after UMI_LOCK is driven low. The UMI_RDY pulse will allow the system bus to sample UMI_LOCK and release the bus lock.

For a single UMI transaction, or non-continuous transactions where ownership of the bus is not required, UMI_LOCK is not necessary.

UMI Single Access

A typical 8-bit wide data, single access write transaction is shown in Figure 12. UMI_WDATA [7:0]=0x11 is written to 0x00004. For this description the UMI_LOCK signal is not used to lock the system bus.

Figure 12. Single Access Write from User Master Interface



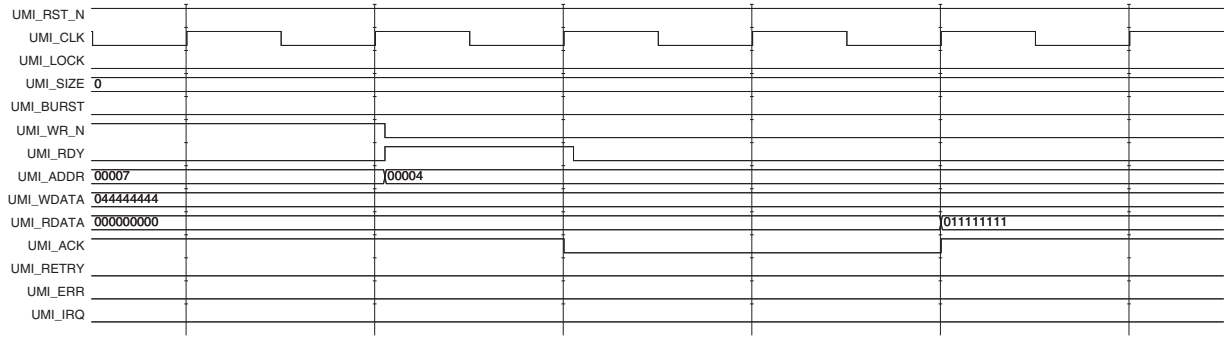
A single write access is initiated with the assertion UMI_RDY with valid data and address on UMI_ADDR and UMI_WDATA. UMI_RDY is the gating signal for valid data and address and is to be asserted for every single access write transaction. On the next cycle, UMI_ACK is de-asserted. When the write transaction is complete, UMI_ACK is asserted by the UMI. UMI_ACK will go low during the transfer for two cycles in synchronous mode and 6-7 cycles in asynchronous mode at best conditions.

UMI_ACK is a synchronous signal and stays asserted after the single write is completed. One cycle before UMI_ACK goes high again, the UMI_ERR or UMI_RETRY will go high if the current transfer gets ERROR or RETRY response from the system bus. The master interface itself may or may not do any retry by itself depending on configuration. UMI_ACK, UMI_ERR and UMI_RETRY will remain high until UMI_RDY is asserted again to begin the next transfer.

Consecutive single writes may be performed with the assertion of UMI_RDY along with new data and address. UMI_WR_N is to be asserted for the entire period of the transaction. UMI_ACK gets de-asserted on the clock cycle after UMI_RDY and gets asserted when the transaction is complete and ready for the next transaction.

A typical 8-bit wide data, single access read transaction is show in Figure 13. UMI_RDATA [7:0]=0x11 is read from 0x00004. Again, the UMI_LOCK signal is not used to lock the system bus.

Figure 13. Single Access Read from User Master Interface



A single access read is initiated with the assertion of UMI_RDY with a valid read address on UMI_ADDR. UMI_RDY is the gating signal for valid address and is to be asserted for only one UMI_CLK cycle for every single read transaction. On the next cycle, UMI_ACK is de-asserted. When the read data is ready at the UMI_RDATA ports, UMI_ACK is asserted by the UMI. UMI_ACK is a synchronous signal and stays asserted after the single read is completed. UMI_ACK will go low during the transfer for two cycles in synchronous mode and 6-7 cycles in asynchronous mode at best conditions.

In case of an ERROR or RETRY response from the system bus, UMI_ERR or UMI_RETRY will be asserted one cycle before UMI_ACK goes high. UMI_ERR and UMI_RETRY will be de-asserted once UMI_ACK goes low again at the onset of the next transaction.

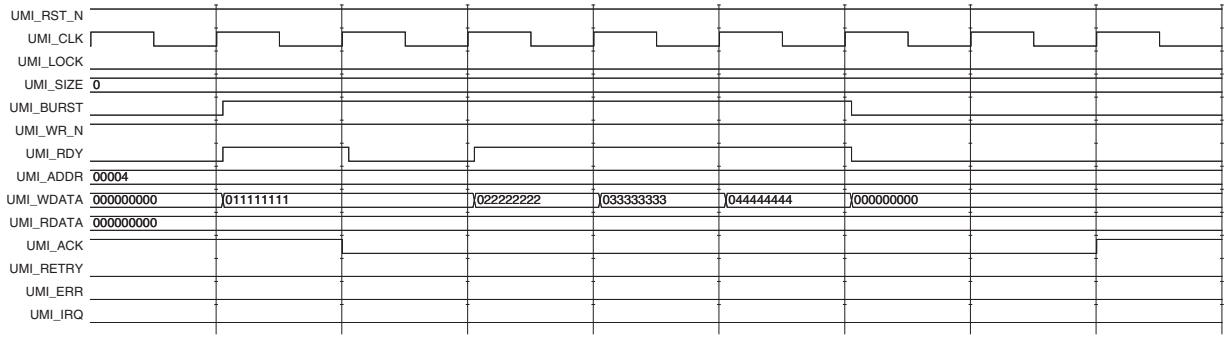
Consecutive signal reads may be performed with the assertion of UMI_RDY along with the new address. UMI_WR_N is to be de-asserted for the entire period of the transaction. On the next UMI_CLK with UMI_RDY high, UMI_ACK is de-asserted. UMI_ACK is re-asserted when the transaction is complete and ready for the next transaction. Signal UMI_RDY should only be high for one UMI_CLK cycle.

UMI Burst Access

Burst access is initiated by asserting the UMI_BURST signal along with UMI_WR_N. The UMI handles bursts that are exactly four beats deep. It employs a 4-beat deep FIFO that is 36 bits wide. It handles and generates burst writes and reads that are four deep regardless of the size of the bus chosen by UMI_SIZE. All address and data mapping is retained across the master interface.

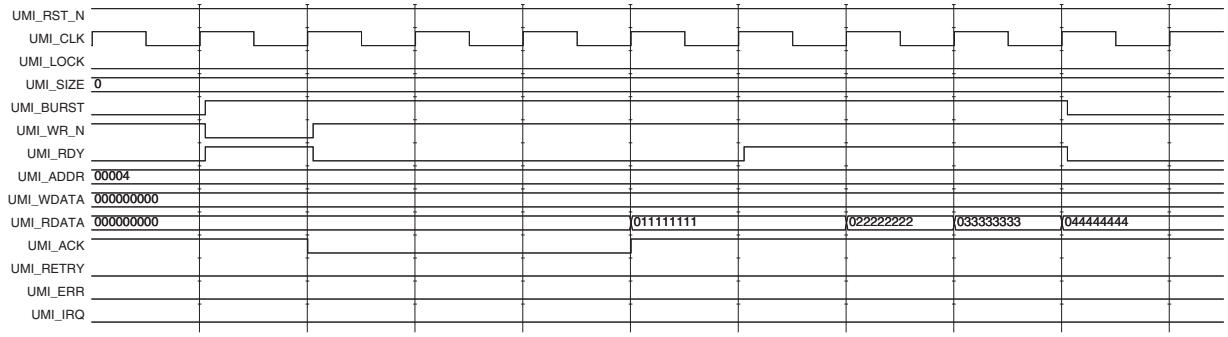
For a write burst, when UMI_ACK is high, a high on UMI_WR_N, UMI_RDY and UMI_BURST for one cycle will result in an exact 4-beat burst write transfer. UMI_ACK will go low after this cycle until the transfer is finished. The UMI_ADDR and the first of the 4-burst data on UMI_WDATA have to be valid for this cycle. The UMI_WR_N and UMI_ADDR/UMI_SIZE may be valid for only this one cycle. However, UMI_RDY has to be high for three more cycles. Note that it is not necessary for UMI_RDY to be high for four continuous cycles and there could be lows in between, but UMI_BURST must be high during this period. If UMI_BURST is de-asserted before the fourth high on UMI_RDY, UMI_ERR will be asserted one or more cycles before UMI_ACK is asserted and the burst transfer is aborted. One cycle before UMI_ACK goes high again, the UMI_ERR or UMI_RETRY may also go high if the transfer gets ERROR or RETRY response from the slave or the master interface loses mastership during the 4-beat transfer. UMI_ACK will go low before going high again for six cycles in synchronous mode and 10-11 cycles in asynchronous mode at best conditions. Figure 14 illustrates an 8-bit wide data 4-beat UMI burst write cycle.

Figure 14. UMI BURST WRITE



A read burst access is initiated with the assertion of input high on UMI_RDY and UMI_BURST and input low on UMI_WR_N for one cycle. This results in a 4-beat (4-byte, 4-halfword or 4-word) burst read transfer. UMI_ACK will go low after this cycle. The UMI_ADDR/UMI_SIZE must be valid for this cycle, too, and the UMI_ADDR is the beginning address for four data reads. There may be a wrap operation on burst address calculation. The transfer finishes after UMI_ACK goes high again for four more active cycles, which may not be continuous four cycles. One of the four read data on UMI_RDATA is valid for each active cycle of UMI_ACK. Each of the four active cycles terminates after a cycle of high on UMI_RDY. Note that it is possible for UMI_ACK to be low between active cycles due to burst FIFO operation. It is necessary for UMI_RDY to be high for three cycles during UMI_ACK high, but UMI_BURST has to be always high until the third active cycle. If UMI_BURST is de-asserted before the third active cycle, UMI_ERR will be asserted one or more cycles before UMI_ACK is asserted and the burst transfer is aborted. UMI_ACK will go low before going high again for three cycles in synchronous mode and eight to nine cycles in asynchronous mode at best conditions. Figure 15 illustrates an 8-bit wide data 4-beat UMI burst read cycle.

Figure 15. UMI Burst Read



Note that in both Figure 14 and Figure 15:

- The first write/read access is to 0x00004 (UMI_W/RDATA [7:0]= 0x11).
- The second write/read access is to 0x00005 (UMI_W/RDATA [7:0]= 0x22).
- The third write/read access is to 0x00006 (UMI_W/RDATA [7:0]= 0x33).
- The fourth write/read access is to 0x00007 (UMI_W/RDATA [7:0]= 0x44).

User Slave Interface (USI)

The USI allows the implementation of a slave peripheral in FPGA design. Through the USI, the FPGA design responds to any master peripherals on the system bus, as long as the transaction address falls within the USI address range (see Table 2). The system bus USI includes the signals listed in Table 15.

Table 15. USI Interface Signals

Signal	Type	Description
USI_CLK	I	User slave interface clock. This clock will only clock the USI. A domain transfer will occur from the USI to the system bus (HCLK) in asynchronous mode. The user slave interface and the system bus can be made synchronous through an ispLEVER IPexpress option (synchronous mode). A frequency preference on this signal will constrain all of the inputs and outputs of the USI.
USI_RST_N	I	This active-low reset resets all of the controls in the user slave interface. This should be pulsed once before any transactions can occur to the USI. This is typically connected to the same reset as the GSR and pulsed at power-up.
USI_WDATA[35:0]	O	36-bit write data from the system bus. This data bus is oriented with bits 35:32 used for parity while bits 31:0 are used for data. Parity is not calculated by any of the peripherals on the system bus; it is only carried. The MSb and LSb will be based on the driving master. For 32-bit access, bits 31:0 are used. For 16-bit access, bits 15:0 are used. The same data on 15:0 will be present on 31:16 as well. For 8-bit access bits 7:0 are used and the same data will be present on D(31:24), D(23:16), and D(15:8) as well.
USI_RDATA[35:0]	I	36-bit read data bus to system bus. This data bus is oriented with bits 35:32 used for parity while bits 31:0 are used for data. Parity is not calculated or checked by any of the peripherals on the system bus; it is only carried. The MSb and LSb must be selected based on the accepting master and application. For 32-bit access, bits 31:0 are used. For smaller than 32-bit transfers the data must be replicated on all byte lanes. For 16-bit access, bits 15:0 and 31:16 are used. For 8-bit access, bits 7:0, 15:8, 23:16, and 31:24 are used with the same read data on all bytes.
USI_ADDR[17:0]	O	This 18-bit address bus provides the address where a slave transaction will operate. Bit 17 of the USI_ADDR bus is the MSb while bit 0 is the LSb.
USI_WR_N	O	Indicates whether the current transaction is a read or a write. 1 indicates a write transaction and the USI_WDATA should be captured. 0 indicates a read transaction and data should be placed on USI_RDATA.

Table 15. USI Interface Signals (Continued)

Signal	Type	Description
USI_SIZE[1:0]	O	Transfer size ([1:0]: 10-double word, 10-word, 00-byte, 11-invalid). Indicates the size of the current transaction. For a 32-bit transaction all of the data bits 31:0 will be valid. For 16-bit transaction only bits 15:0 will be valid. For 8-bit data only bits 7:0 will be valid.
USI_ERR	I	Active-high error response from the user to the USI when data size is not consistent with USI_SIZE, or the address on the USI_ADDR is out of range for the application.
USI_ACK	I	Active-high acknowledge for read operations. User drives USI_ACK high to terminate the transaction. Signal USI_ACK is passed through the system bus to eventually terminate the transaction from the driving master.
USI_RDY	O	Active-high ready response from the USI, for either read or write transactions. Signal USI_RDY goes high when an address is ready on USI_ADDR.
USI_RETRY	I	Signal available for the user to drive when a transaction is not ready to be processed. This retry signal is then sent to the originating master interface.
USI_IRQ	I	Active-high interrupt pin to indicate an interrupt to the system bus. This signal will map to the interrupt cause register USER_SLAVE bit (0x00010 bit 6).

User Slave Transfer Errors

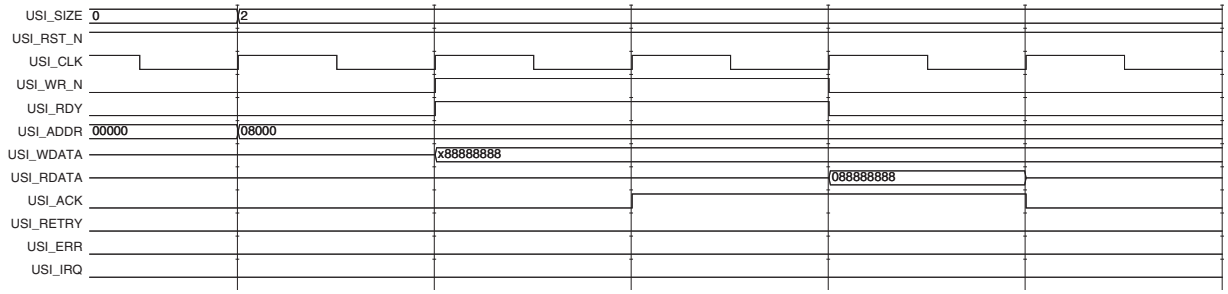
The user logic slave interface responds with a transfer error to the system bus under the following circumstances (this is an internally generated error from the USI and is separate from the USI_ERR signal. The resulting error to the master interface will be either a MPI_RETRY or UMI_RETRY for the MPI and UMI.):

- USI_ERR is high during the transaction.
- USI_RST_N is low during the transaction.
- The device is in bitstream configuration.
- The address does not conform to the USI_SIZE specified. The setting for USI_SIZE will dictate the granularity of the address available from the USI. If USI_SIZE is selected for 8-bit mode, then all addresses can be selected. If USI_SIZE is selected for 16-bit (word) mode, then only addresses on word boundaries can be selected. For example addresses 0x00000, 0x00002, 0x00004, etc. are valid addresses. 0x00001, 0x00003, 0x00005, etc. are not valid addresses for word accesses. The same holds true for 32-bit (double word) mode: 0x00000, 0x00004, 0x00008, etc. are valid while 0x00001, 0x00002, 0x00003, 0x00005, 0x00006, 0x00007, 0x00009, etc. are not valid.

USI Single Access

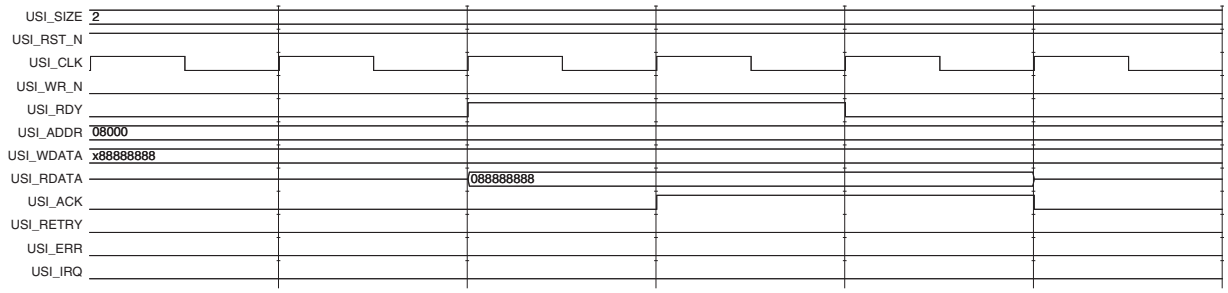
A typical operation of a 32-bit data single access write transaction is illustrated in Figure 16. The USI_WR_N signal goes high to indicate that a write is taking place. The synchronous signal USI_RDY signal indicates the availability of valid address and data on the USI_ADDR and USI_WDATA ports, respectively. The USI inserts additional wait states until the user asserts the USI_ACK signal. For write operations, if the data received by the slave interface can be accepted by the user slave in one cycle, it is acceptable to leave USI_ACK asserted continuously during USI_WR_N high (write operation). If the USI takes more than one USI_CLK cycle to terminate the transaction, then USI_ACK should be driven high when the transaction is complete.

Figure 16. Single Access Write at USI



A typical operation of a 32-bit data single access read transaction is illustrated in Figure 17 The USI_WR_N signal is low to indicate that a read is taking place. The synchronous USI_RDY signal indicates the availability of valid address on the USI_ADDR. The user must then respond with read data on the USI_RDATA bus and assert USI_ACK. The USI inserts additional wait states until the user asserts the USI_ACK signal.

Figure 17. Single Access Read at USI



In both Figure 16 and Figure 17, address 0x08000 is accessed, and USI_W/R_DATA[31:0] =0x88888888.

USI Burst Access

Burst accesses from MPI or UMI are treated similar to single accesses. There is no burst FIFO in the USI and thus all burst accesses are handled as back-to-back signal accesses with wait states. It is possible for the slave to handle bursts of user design defined lengths. During a burst access the USI_ADDR bus will increment with each USI_RDY high following a USI_ACK high cycle during the burst transfer. Figure 18 shows a 32-bit data USI 4-beat burst write access, whereas Figure 19 illustrates a 32-bit data USI 4-beat burst read access.

Figure 18. 4-beat USI Burst Write

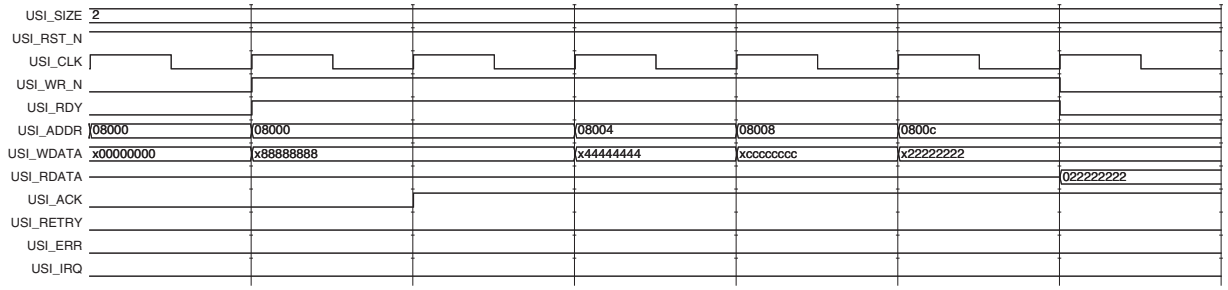
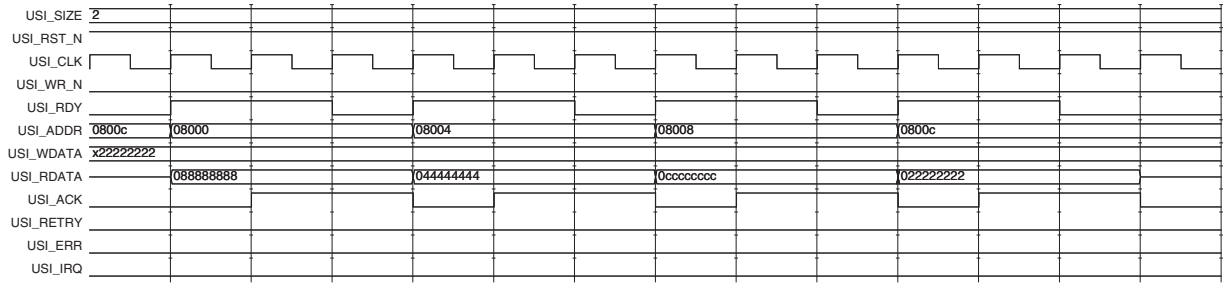


Figure 19. 4-beat USI Burst Read



Note that in both Figure 18 and Figure 19:

- The first write/read access is to 0x08000 (USI_W/RDATA [31:0]= 0x88888888).
- The second write/read access is to 0x08004 (USI_W/RDATA [31:0]= 0x44444444).
- The third write/read access is to 0x08008 (USI_W/RDATA [31:0]= 0xCCCCCCCC).
- The fourth write/read access is to 0x0800C (USI_W/RDATA [31:0]= 0x22222222).

PCS Interface

The system bus PCS interface is a slave interface. The details and protocol of the PCS interface are specific to the LatticeSC device. The PCS interface supports 8-bit and 32-bit system bus master (MPI, UMI) data accesses.

PCS Interface Signals

Up to eight possible PCS interfaces can be generated on a system bus in ispLEVER IPexpress. This corresponds to the eight possible different PCS quad locations on the largest LatticeSC device. Each system bus PCS interface connects to a unique PCS quad in the LatticeSC device. The base address of each PCS is used in defining the PCS interface port names, as illustrated in Table 16. On smaller LatticeSC devices, not all PCS interfaces are available. Please refer to the LatticeSC/M Family flexiPCS Data Sheet for information on the number of PCS quads available for a given die size and package combination. The PCS address range is defined in Table 2.

Table 16. PCS Interfaces on the System Bus

PCS Quad Location	PCS Base Address in HEX	Device Side	System Bus Interface Ports	
			PCS -> System Bus	System Bus -> PCS
360	36000	Left	pcs360_in[16:0]	pcs360_out[44:0]
361	36100	Left	pcs361_in[16:0]	pcs361_out[44:0]
362	36200	Left	pcs362_in[16:0]	pcs362_out[44:0]
363	36300	Left	pcs363_in[16:0]	pcs363_out[44:0]
3E0	3E000	Right	pcs3E0_in[16:0]	pcs3E0_out[44:0]
3E1	3E100	Right	pcs3E1_in[16:0]	pcs3E1_out[44:0]
3E2	3E200	Right	pcs3E2_in[16:0]	pcs3E2_out[44:0]
3E3	3E300	Right	pcs3E3_in[16:0]	pcs3E3_out[44:0]

Enabling multi-device alignment under the system bus PCS section of ispLEVER IPexpress also creates the multi-chip alignment (MCA) inputs (mca_clk_p[12]_in, and mca_done_in) and outputs (mca_clk_p[12]_out and mca_done_out). The MCA inputs are driven by another LatticeSC chip. The MCA outputs drive another LatticeSC chip. These I/O have no system bus function and are used when PCS multi-device alignment is desired. Please refer to the LatticeSC/M Family flexiPCS Data Sheet for more information on multi-device alignment.

PCS Interrupts

Both left and right PCS quads can generate interrupts to the system bus. Since a PCS interrupt can be generated from any of the eight possible PCS quads, inter-quad interface register 0x3EF0B (see Table 17) contains one interrupt bit for every PCS quad location. When any of the bits in 0x3EF0B is set as a result of a PCS interrupt, the interrupt is passed to the system bus via bit 1 of 0x00010 (PCS interrupt bit in interrupt cause register). The interrupt can then be sent to an output interrupt (MPI or USER as discussed in the System Bus Interrupts section of this document). In the reverse direction, when an output interrupt is detected, a read access of 0x00010 showing a 1 on bit 1 indicates a PCS interrupt, and a read access of 0x3EF0B indicates which PCS quad (could be more than one) initiated the interrupt. Furthermore, PCS quad interface register 0x80 can be read to determine if any channel interface register(s) on a given quad channel initiated the interrupt. More information on quad interface register 0x80, as well as quad and channel interface interrupt registers can be found in the Register Map section of the LatticeSC/M Family flexiPCS Data Sheet.

Table 17. PCS Interrupt Source Register 0x3EF0B

Bit Position	Bit 0	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
PCS Quad Source of Interrupt	360	361	362	363	3E0	3E1	3E2	3E3
	Left				Right			

SMI Interface

The SMI (Serial Management Interface) peripheral on the LatticeSC system bus allows any master on the bus to access any element associated with an SMI slave. SMI slaves include LatticeSC PLLs, DLLs (with built-in SMI interfaces), in addition to user-defined control/status registers in the FPGA.

The SMI address range is defined in Table 2. There are a total of 64 possible SMI peripherals. Each peripheral corresponds to one of 64 base addresses in the range (0x00400-0x007F0). Each interface targets 16 bytes (128 bits) of memory. For example, the interface associated with base address 0x00410 targets addresses 0x00410 to 0x0041F. Note that 0x00400 is the default base address for all PLLs and DLLs in the LatticeSC device. This base address can be modified by changing the SMI_OFFSET attribute. For simulation, SMI_OFFSET is assigned via a parameter line. For Map, Place and Route, SMI_OFFSET is assigned via a preference line.

For example, a PLL of type EHXPLLA and module name MYPLL is generated in IPexpress. The top level of a Verilog design has an instance name PLL1 for MYPLL.

To assign an SMI_OFFSET of 0x420 (base address=0x00420) to PLL1 for simulation, the following line is added in the Verilog design:

```
defparam PLL1.MYPLL_0_0.SMI_OFFSET= 12'h420;
```

To assign an SMI_OFFSET of 0x420 to PLL1 for Map, Place and Route, the following line is added in the Map, Place and Route preference file:

```
ASIC "PLL1/MYPLL_0_0" TYPE "EHXPLLA" SMI_OFFSET="0x420";
```

SMI Interface Signals

Table 18 shows the SMI signals generated as part of the system bus I/Os. Note that, because base address 0x00400 is the default base address for PLLs and DLLs, there is no read data port for that interface.

Table 18. SMI Interface Signals

Name	I/O	Description
SMI_CLK	O	SMI Clock. This clock runs at 1/4 the frequency of HCLK.
SMI_WR	O	Active-high dedicated SMI write signal
SMI_RD	O	Active-high dedicated SMI read signal
SMI_RST_N	O	SMI active-low reset. This reset signal is derived from the system bus reset (SYSBUS_RST_N).
SMI_WDATA	O	SMI Data Write signal. This signal is broadcast to all SMI peripherals.
SMI_RDATA_0XNN0	I	This is the SMI read data signal corresponding to interface 0x00NN0. NN can have any value from 41 to 7F, corresponding to one of 63 possible SMI read interfaces (there is no read port for interface 0x00400). Therefore, there could be up to 63 possible SMI read data ports on the SMI interface.
SMI_ADDR [9:0]	O	SMI address. Since the SMI address range is from 0x00400 to 0x007FF, this corresponds 1024 different byte addresses. Hence, the SMI_ADDR only needs to be 10 bits wide. SMI_ADDR= 0x00X corresponds to base 0x0040X in system bus address range. SMI_ADDR= 0x01X corresponds to base 0x0041X in system bus address range. . . . SMI_ADDR= 0x3FX corresponds to base 0x007FX in system bus address range. Where X= is in the range (0-F)

SMI Single Beat Data Transfers

Figure 20 illustrates an SMI write transfer, whereas Figure 21 shows an SMI read transfer. The SMI_CLK is sourced from the system bus at a fourth of the frequency rate of HCLK. Each assertion on read/write strobes SMI_RD and SMI_WR will initiate a byte of data transfer (one bit per SMI_CLK cycle). SMI_ADDR[3:0] determines which of the 16 bytes of the targeted MSI interface is being accessed.

To ensure zero hold time, SMI_RD, SMI_WR, and SMI_WDATA are synchronized to the falling edge of SMI_CLK.

As shown in Figure 20, a byte write cycle starts with the assertion of a single cycle SMI_WR pulse on the falling edge of SMI_CLK along with the first SMI_WDATA bit (D00) to be written. The remaining data bits (D01 to D07) will then be written on each subsequent falling edge of SMI_CLK.

As shown in Figure 21, a byte read cycle starts with the assertion of a single cycle SMI_RD pulse on the falling edge of SMI_CLK. The first bit of SMI_RDATA_0XNN0 (D00) is expected at the following rising edge. The remaining data bits (D01 to D07) will then be received on each subsequent rising edge of SMI_CLK.

Note that, because SMI_CLK is four times slower than HCLK, and the SMI interface only transfers one bit of data per SMI_CLK cycle, MPI and UMI accesses to SMI through the system bus will have a considerably longer latency than accesses to other slave interfaces. For example, one system bus simulation shows that with MPI in synchronous mode, the system bus takes 46 MPI clock cycles to complete an SMI read access. Therefore, the timeout parameter of the PowerPC bus needs to be programmed with the large SMI latency in perspective.

Figure 20. SMI Write Timing Diagram

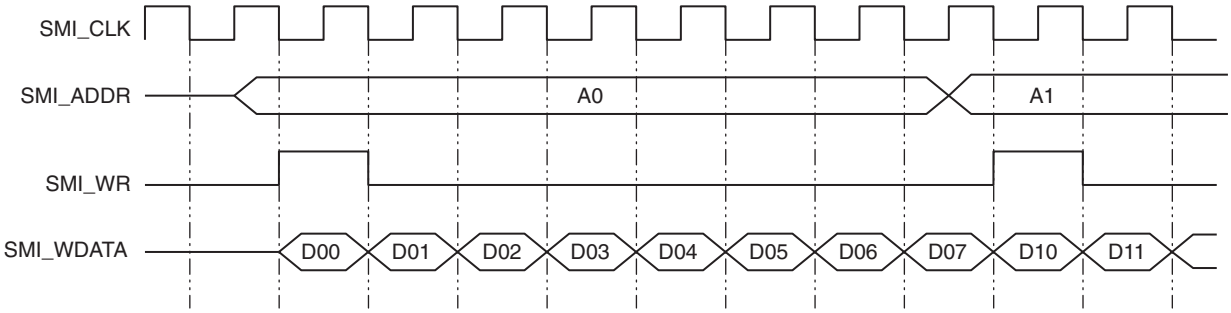
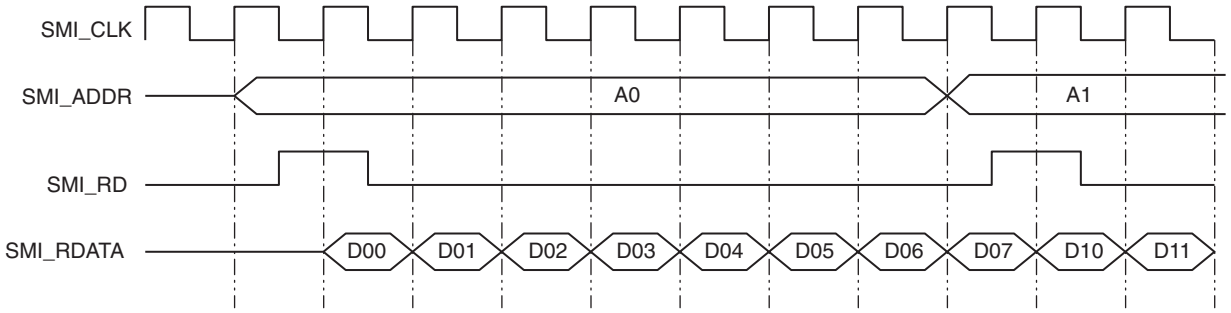


Figure 21. SMI Read Timing Diagram



SMI Burst Access

Burst accesses from a master are treated similar to single accesses. All burst accesses are handled as back-to-back signal accesses.

Direct FPGA Access (DFA) Interface

The LatticeSC device allows for the MPI interface to directly access a DFA controller that the user can implement in FPGA logic. Most of the latency through the system bus is eliminated in this mode, as most of the system bus logic is bypassed to provide direct access from the MPI to the DFA interface. The DFA controller acts as a slave to the MPI. An ispLEVER IPexpress option creates the DFA interface as the system bus model is being configured. Cre-

ating the DFA interface also results in the creation of the DFA ports as part of the system bus model. Creating a DFA interface is only possible once an MPI interface has already been enabled (also in ispLEVER IPexpress). The DFA data bus size reflects that of the MPI.

DFA Interface Signals

Table 19 shows the DFA signals generated as part of the system bus I/Os.

Externally the MPI implements a 36-bit PowerPC bus slave, which internally drives the DFA as a master. Data bus width is automatically generated as 8 bits, 16 bits or 32 bits with parity of 1, 2 or 4 bits, respectively depending on how the MPI was configured.

Table 19. DFA Interface Signals

Name	I/O	Description
DFA_TSIZ[0:1]	O	Transfer size ([0:1]: 00-double word, 10-word, 01-byte). These ports select the size of the DFA data transaction. These ports reflect the condition of MPI_TSIZ[0:1].
DFA_WR_N	O	Transfer type (0-write, 1-read). This signal indicates to the DFA controller whether the transaction initiated by the MPI is a read or a write. If the transaction is a read, data will be provided to the MPI from the address specified. If the transaction is a write, data will be written to the DFA address specified by the MPI. This port reflects the condition of MPI_WR_N.
DFA_BURST	O	Indicates that a burst transfer is in progress when low. This signal informs the DFA that the MPI is performing a burst transaction using the MPI burst pin. This port reflects the condition of MPI_BURST.
DFA_BDIP	O	Burst Data In Progress. This signal from the MPI will go low on the first clock of data during a burst and go high on the last clock of data of the burst transfer. This port reflects the condition of MPI_BDIP.
DFA_STRB_N	O	This active low signal indicates the start of a transactions or the strobe. This port reflects the value of MPI_STRB_N.
DFA_CS0N/ DFA_CS1	O	DFA selects for active low (DFA_CS1) and active high (DFA_CS0N). Both of these chip selects must be active for the DFA controller logic to be selected (DFA_CS1=0, DFA_CS0N=1). These ports reflect the condition of MPI_CS0N/MPI_CS1.
DFA_ADDR[14:31]	O	The PowerPC address bus is 32 bits wide. The LatticeSC devices only support 18 bits of address space. The LatticeSC uses the least significant bits of the PowerPC address space using address bits 14:31. These ports reflect the condition of MPI_ADDR[14:31].
DFA_WR_DATA[0:31]	O	The DFA WR data bus can be up to 32 bits wide (depending on MPI configuration). Bit 0 is the MSb and bit 31 is the LSb. For multi-byte transfers, the most significant byte has the lowest address. These ports reflect the condition of MPI_DATA[0:31] during a write cycle.
DFA_RD_DATA[0:31]	I	The DFA RD data bus can be up to 32 bits wide (depending on MPI configuration). Bit 0 is the MSb and bit 31 is the LSb. For multi-byte transfers, the most significant byte has the lowest address. During a read data cycle, this bus is passed to MPI_DATA[0:31] as long as DFA_TRI_DATA=0.
DFA_RD_PARITY[0:3]	I	Read Parity. Can be up to four bits wide (one bit per byte of data) depending on the MPI data bus size. Parity is passed to MPI_PAR[0:3] on a read access when DFA_TRI_DATA=0.
DFA_TRI_CTL	I	This DFA signal controls the state of MPI_TA, MPI_TEA and MPI_RETRY on the MPI interface. When DFA_TRI_CTL is 1, these MPI outputs are tri-stated. When DFA_TRI_CTL is 0, DFA_TA, DFA_TEA, and DFA_RETRY are passed on to MPI_TA, MPI_TEA, and MPI_RETRY respectively. The FPGA DFA controller logic should always drive this signal to 0 during the data phase of a DFA access (DFA_TA=0).
DFA_TRI_DATA	I	This DFA signal controls the state of MPI_DATA/MPI_PAR during a read access. When DFA_TRI_DATA is 1, MPI_DATA/MPI_PAR are always tri-stated during a read cycle. When DFA_TRI_DATA is 0, DFA_RD_DATA/DFA_RD_PARITY are passed on to MPI_DATA/MPI_PAR on a read cycle. DFA_TRI_DATA=0 should coincide with DFA_TA=0
DFA_TA	I	This active low signal indicates the transfer acknowledge from the DFA. This port is translated to the MPI_TA pin of the MPI as long as DFA_TRI_CTL=0.

Table 19. DFA Interface Signals (Continued)

Name	I/O	Description
DFA_TEA	I	This active low signal indicates a transfer error acknowledge during the current transaction. This port is translated to the MPI_TEA pin of the MPI as long as DFA_TRI_CTL=0.
DFA_RETRY	I	Active-low request for processor to relinquish the bus and retry the cycle. Exception signal indicating the LatticeSC device is not ready to accept the requested transaction This port is translated to the MPI_RETRY pin of the MPI as long as DFA_TRI_CTL=0.

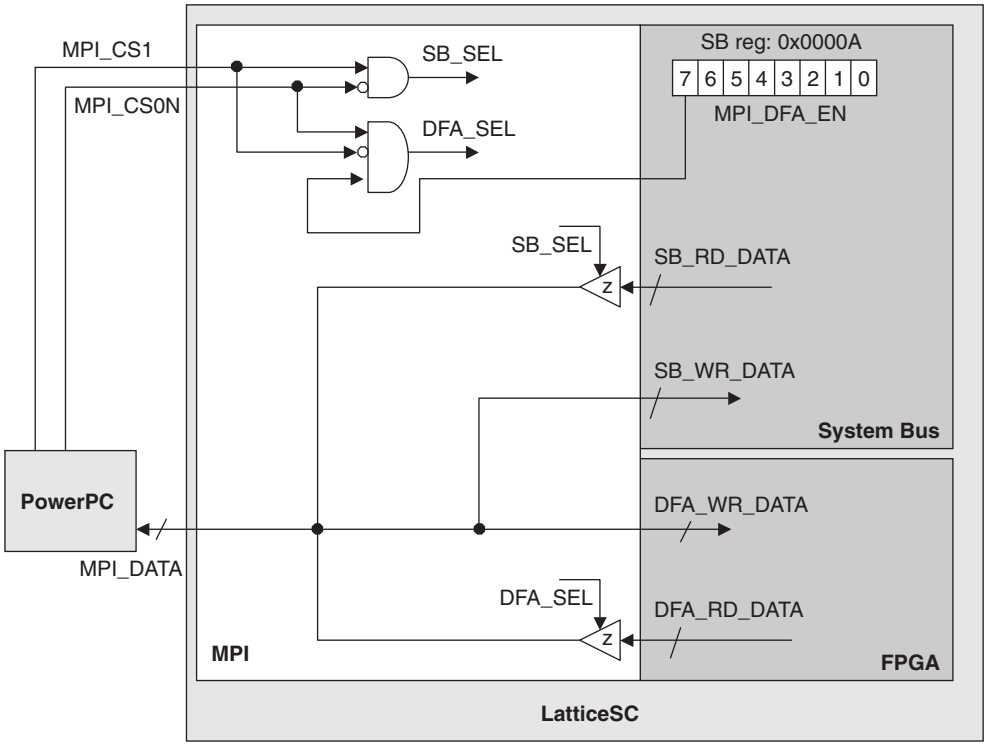
Enabling the DFA Interface

In addition to enabling the creation of the DFA interface signals in ispLEVER IPexpress, the DFA interface needs to be enabled for both writing and reading accesses. This requires setting two parameters:

- MPI_DFA_EN bit (reg. 0x0000A, bit 7): Setting this bit to 1 allows the MPI interface logic to interpret MPI_CS1=0/MPI_CS0N=1 as a valid MPI interface selection (DFA access) in addition to MPI_CS1=1/MPI_CS0N=0 (system bus access).
- MPI_CS0N/MPI_CS1N: Setting these MPI signals to 1 and 0 respectively serves two purposes.
  - Through the DFA_CS0N/DFA_CS1 interface outputs, this combination of values informs the FPGA DFA controller that it is being accessed.
  - This combination of values enables the MPI logic to select data from the DFA interface during a read cycle (as opposed to selecting read data from the system bus logic when MPI_CS0N=0 and MPI_CS1=1).

Figure 22 illustrates from a logical standpoint (not actual hardware depiction) how the MPI, DFA and system bus data busses are affected by MPI_CS0N/MPI_CS1 and MPI_DFA_EN. Control signals are not shown in this figure for simplicity.

Figure 22. DFA/System Bus Data Selection



DFA Single Beat Data Transfers

Any data transfer on the DFA bus has four phases: arbitration, address, data, and termination. During the arbitration phase the MPI initiates the transaction with a DFA_STRB_N pulse. The DFA controller in the FPGA samples

the address and control inputs during the address phase, receives or provides data and asserts transfer acknowledge during the data phase, and de-assert signals during the termination phase. Figure 23 shows the bus timing for an 8-bit wide data, single beat write transfer. Figure 24 shows the DFA bus timing for an 8-bit wide data, single beat read transfer. Both figures access address 0x00000 (DATA [0:7]=0x11).

Figure 23. DFA Single Beat Write Data Transfer Timing

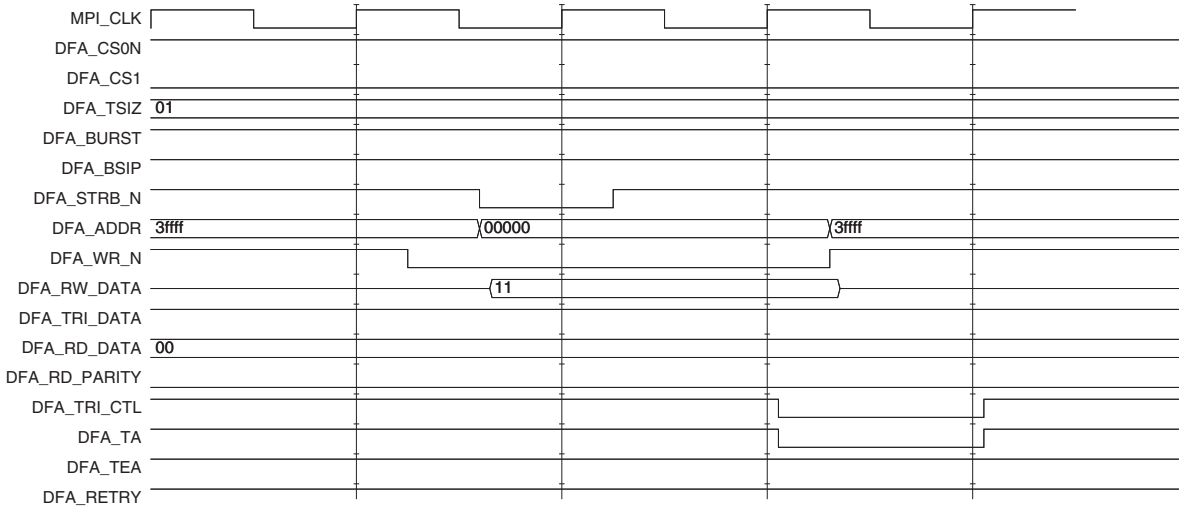
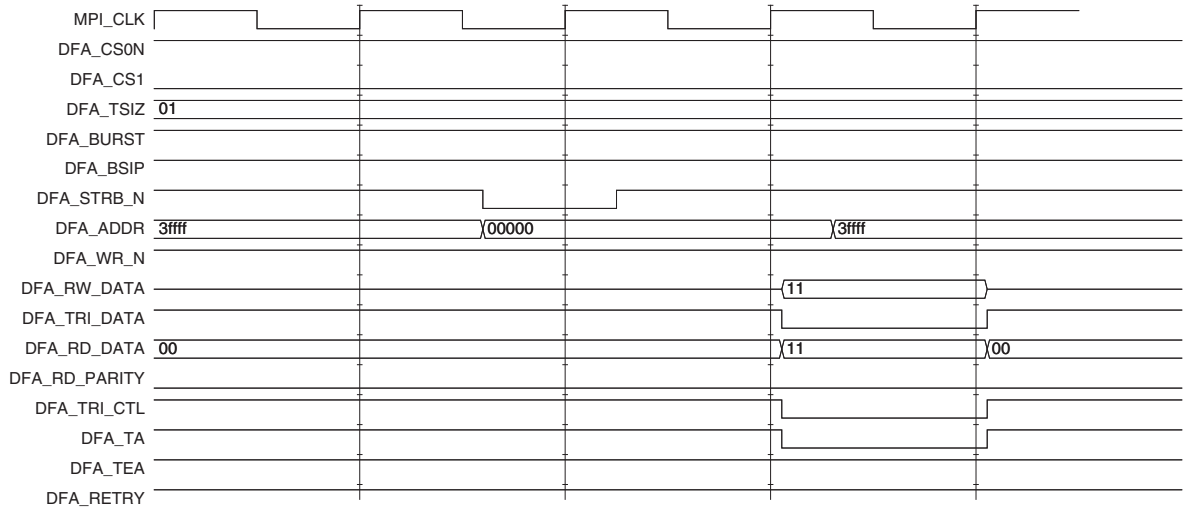


Figure 24. DFA Single Beat Read Data Transfer Timing



DFA Burst Access

DFA burst accesses are similar to MPI burst accesses, except that DFA_BURST and DFA_BDIP are now outputs of the DFA interface (they reflect the state of MPI_BURST and MPI_BDIP respectively). DFA_TRI_CTL and DFA_TRI_DATA are also driven by the FPGA DFA logic as in the case of DFA single accesses.

System Bus Memory Map

Table 20 describes the internal register map within the system bus. Memory maps for a specific block, such as PCS, are found in the block-specific data sheet. The system bus memory map is structured with bit 0 as the MSb to match the PowerPC/MPI bus orientation shown in Table 7. All power-up default values are 0x0 unless otherwise specified.

Table 20. System Bus Register Map

Absolute Address	Type R=Read W=Write	Bits	Name	Description
0x00000 - 0x00003	R	0:31	DEVICE_ID	32-Bit Manufacturer and Device ID Code: The manufacturer identification code register contains a unique code for each LatticeSC device in the family. The code is comprised of: • ID[0:11]: This is company identification provided by JEDEC. The value for this field is always 0xEA8. • ID[12:19]: Size Identification. The value is the binary number of columns of CIB/PLC in the device. As an example, the 5S25 device has 72 columns of CIB/PLC so its size identification is 0x48. Refer to the LatticeSC/M Family Data Sheet for the full list of size identification values for different LatticeSC devices. • ID[20:23]:Reserved. • ID[24:27]: Series Identification. LatticeSC is 0x5ID • [28:31]: Reserved to 0x0. For example, the 5S25 part ID CODE is (ID[0:31]): 5S25 (72 columns): 0xEA848X50 (X=don't Care). The system bus simulation model will always show a 5S25 device ID. Manufactured devices will have the proper device ID as part of the silicon.
0x00004 - 0x00007	R/W	0:31	SCRATCH_PAD	32-bit scratchpad register. Free register used for debugging purposes.
0x00008- 0x0000A	**R/W**	**Control Registers**		
0x00008	R/W	0:1	RDBK_SIZE	These two bits specify the number of valid bytes in the read back data register (0x00018) during a readback operation. [0:1]: • 00 - 1 byte • 10 - 2 bytes • 01 - 4 bytes
	R/W	2	MPI_USR_ENABLE	Active high. Enables the MPI interface to the user. Used to keep the MPI available during a reconfiguration process. During reconfiguration, the mode pins are not sampled to check for MPC mode. Set this bit before reconfiguration to keep MPI available.
	R/W	3	REPEAT_RDBK	Active high. Inhibits auto-increment of the readback address (0x00014) when the readback data register (0x00018) is read.
	R/W	4	SYS_RD_CFG	Active-high. Initializes the readback logic.
		5		Unused
	R/W	6	UMI_RST	Active high. Asserts system bus reset. Can only be set by user master. Writing to this register bit is only active if "Systembus Reset by User Master" is set in the IPexpress GUI for system bus.
	R/W	7	MPI_RST	Active high. Asserts system bus reset. Can only be set by MPI. Writing to this register bit is only active if "Systembus Reset by MPI" is set in the IPexpress GUI for system bus.

Table 20. System Bus Register Map

Absolute Address	Type R=Read W=Write	Bits	Name	Description
0x00009	R/W	0		Reserved
	R/W	1	UMI_LOCK	Active high, locks the internal system bus for use by the UMII. Used for multi-cycle operations that must retain bus ownership. Only the UMI can write this bit.
	R/W	2	MPI_LOCK	Active high, locks the internal system bus for use by the MPI. Used for multi-cycle operations that must retain bus ownership. Only the MPI can write this bit.
		3		Unused
	R/W	4	PRGM_UMI	UMI configuration request. Active high, forces reconfiguration of the FPGA logic using the mode specified on the MODE pins during initial power up. Has to be released before sending the bitstream.
	R/W	5	PRGM_MPI	MPI configuration request. Active high, forces reconfiguration of the FPGA logic using the mode specified on the MODE pins during initial power up. Has to be released before sending the bitstream.
	R/W	6	SYS_DAISY	Enables bitstream daisy chaining when configuring more than one device via MPI.
	R/W	7	SYS_GSR	Active high, asserts the global set/reset.
0x0000A	R/W	0:3	EBR_EXP	For pre-configuration usage of these bits, see TN1080, LatticeSC sysCONFIG Usage Guide.
		4		Unused
		5	MPI_PAR_CHK	Enables MPI to check parity errors for write transfers if MPI parity bus is enabled
	R/W	6		Reserved. Must be written to ‘0’.
	R/W	7	MPI_DFA_EN	When set to 1, this bit enables any fabric DFA controller to use the MPI outputs pads.
0x0000B				Unused

Table 20. System Bus Register Map

Absolute Address	Type R=Read W=Write	Bits	Name	Description
0x0000C-0x0000D	**R**	**Status Registers**		
0x0000C	R	0:1	ERR_FLAG	In the event of an error during device configuration these bits will indicate the nature of the error. [0:1]: • 00 - no error • 01 - checksum error indicates one or more corrupt bits in bitstream • 10 - device ID error indicates bitstream does not match target • 11 - framing/alignment error. Data bits may be reversed.
	R	2	INIT_N	Reflects the state of the INIT I/O pad.
	R	3	DONE	Reflects the state of the DONE I/O pad.
	R	4	CFG_DATA_LOST	Indicates that some initialization data was lost because configuration encountered long wait states or too many retries when initializing EBRs/ASB
	R	5:6	CFG_BUSI_ERR	If an internal system bus error occurs during configuration the error is captured in these two bits. The address of the first error in captured in the bus error address register (0x00024). [bit5 bit6]: 00 - no errors 01 - invalid response code 10 - one error occurred 11 - multiple errors occurred
	R	7	RDBK_AOR_ERR	Active-high readback address out of range error alarm.
0x0000D	R	0:1	WDATA_SIZE	Reflects the HSIZE [0:1] size during write transfers to the configuration data register
	R	2	EBR_BIT_ERR	A 1 indicates the bitstream of system bus configuration contains errors in the initialization data for EBRs.
	R	3	ASB_BIT_ERR	A 1 indicates the bitstream of system bus configuration contains errors in the initialization data for ASB.
		4:5		Unused
	R	6	RDATA_RDY	Active-high indicates that data is pending in the readback data register (0x00018).
	R	7	WDATA_ACK	Active-high indicates that the configuration logic is ready for data to be written into the configuration data register (0x0001C).
0x0000E - 0x0000F				Unused

Table 20. System Bus Register Map

Absolute Address	Type R=Read W=Write	Bits	Name	Description
0x00010	**Interrupt Cause Register**			
		0		Unused.
	R	1	PCS_IRQ	Active-high, interrupt request from the PCS interface. Write 1 to clear this bit.
	R	2	MPI_IRQ	Active-high, interrupt request from the MPI. Write 1 to clear this bit.
	R	3	CFG_ERR_IRQ	Active-high, indicates that the ERR_FLAG bits in the status register were changed during device configuration.
	R	4	CFG_DATA_IRQ	Active-high, interrupt request from the configuration logic requesting another word/byte of data. Write 1 to clear this bit.
	R	5	UMI_IRQ	Active-high, interrupt request from the User Master interface (UMI_IRQ). Write 1 to clear this bit.
	R	6	USI_IRQ	Active-high, interrupt request from the User Slave interface (USI_IRQ). Write 1 to clear this bit.
	R	7	USER_IRQ	Active-high, interrupt request from the USR_IRQ_IN signal. Write 1 to clear this bit.
0x00011				Unused
0x00012	**USER Interrupt Enable Register** (Only UMI can write to this register)			
		0	EN_IRQ_USER	Unused
	R/W	1		Logic 1 enables the PCS interrupt bit from address 0x00010 to generate an interrupt to the USR_IRQ_OUT port.
	R/W	2		Logic 1 enables the MPI interrupt bit from address 0x00010 to generate an interrupt to the USR_IRQ_OUT port.
	R/W	3		Logic 1 enables the CFG_ERR interrupt bit from address 0x00010 to generate an interrupt to the USR_IRQ_OUT port.
	R/W	4		Logic 1 enables the CFG_DATA master interrupt bit from address 0x00010 to generate an interrupt to the USR_IRQ_OUT port.
	R/W	5		Logic 1 enables the USER_MSTR interrupt bit from address 0x00010 to generate an interrupt to the USR_IRQ_OUT port.
	R/W	6		Logic 1 enables the USER_SLAVE interrupt bit from address 0x00010 to generate an interrupt to the USR_IRQ_OUT port.
	R/W	7		Logic 1 enables the USER_IRQ interrupt bit from address 0x00010 to generate an interrupt to the USR_IRQ_OUT port.

Table 20. System Bus Register Map

Absolute Address	Type R=Read W=Write	Bits	Name	Description
0x00013	**MPI Interrupt Enable Register** (Only MPI can write to this register)			
	R/W	0	EN_IRQ_MPI	Unused.
	R/W	1		Logic 1 enables the PCS interrupt bit from address 0x00010 to generate an interrupt to the MPI_IRQ_N pin.
	R/W	2		Logic 1 enables the MPI interrupt bit from address 0x00010 to generate an interrupt to the MPI_IRQ_N pin.
	R/W	3		Logic 1 enables the CFG_ERR interrupt bit from address 0x00010 to generate an interrupt to the MPI_IRQ_N pin.
	R/W	4		Logic 1 enables the CFG_DATA master interrupt bit from address 0x00010 to generate an interrupt to the MPI_IRQ_N pin.
	R/W	5		Logic 1 enables the USER_MSTR interrupt bit from address 0x00010 to generate an interrupt to the MPI_IRQ_N pin.
	R/W	6		Logic 1 enables the USER_SLAVE interrupt bit from address 0x00010 to generate an interrupt to the MPI_IRQ_N pin.
	R/W	7		Logic 1 enables the USER_IRQ interrupt bit from address 0x00010 to generate an interrupt to the MPI_IRQ_N pin.
0x00014 - 0x00017	R/W	0:13	CFG_RDBK_ADDR	Configuration memory readback address register (14 bits). Bits [14:31] are reserved.
0x00018 - 0x0001B	R/W	0:31	CFG_RDBK_DATA	Configuration memory readback data register
0x0001C - 0x0001F	R/W	0:31	CGG_DATA	Configuration data register
0x00020 - 0x00023	R	0:31	TRAP_ADDR	24-bit trap address register. Configuration trapped in the address bus is stored here when bitstream initialization for EBR/ASB is lost
0x00024 - 0x00027	R	0:31	BUSI_ERR_ADDR	Bus error address register contains the address of the first configuration error. Indicated by the CFG_ERR bits of register 0x0000C.
0x00028 - 0x0002B	R	0:31	READ_WORD1	Read Only Word #1. Read Only Register with content defined via IPexpress
0x0002C - 0x0002F	R	0:31	READ_WORD2	Read Only Word #2. Read Only Register with content defined via IPexpress
0x00030 - 0x00033	R	0:31	READ_WORD3	Read Only Word #3. Read Only Register with content defined via IPexpress
0x00034 - 0x00037	R	0:31	READ_WORD4	Read Only Word #4. Read Only Register with content defined via IPexpress
0x00038 - 0x0003B	R	0:31	READ_WORD5	Read Only Word #5. Read Only Register with content defined via IPexpress
0x0003C - 0x0003F	R	0:31	READ_WORD6	Read Only Word #6. Read Only Register with content defined via IPexpress

Creating the System Bus in HDL

IPexpress Flow

After creating a LatticeSC project in ispLEVER Project Navigator, one can start an IPexpress (ispLEVER IPexpress) session by either selecting **Tools-> IPexpress** or by clicking on the ispLEVER IPexpress icon from the Tools Toolbar section. The ispLEVER IPexpress session can then be used to configure the system bus and generate an HDL description for synthesis and simulation purpose. This includes but is not limited to:

- Selecting a name for the system bus and a Project Path where the HDL output will be generated.
- Selecting the HDL output format (Verilog/VHDL).
- Customizing the desired master and slave interfaces (example: MPI, UMI, USI).
- Setting general options.
- Enabling flexiPCS ports on the system bus and defining flexiPCS multi-quad and multi-chip alignment. Please refer to TN1145, LatticeSC flexiPCS/SERDES Design Guide for more information.
- Generating the output files.

Help on using ispLEVER IPexpress to configure a system bus module is available from within the ispLEVER Project Navigator.

System Bus Simulation Model

The HDL description for the System Bus that IPexpress generates includes a black box of a simulation model (SYSBUSA). A precompiled simulation model for SYSBUSA exists for every simulator supported by Lattice. The simulator should point to this model during simulation (either explicitly in a user script, or implicitly when the user creates and simulates a Verilog/VHDL design within Project Navigator). It will take about 15 microseconds for the the SYSBUSA simulation model to power-up in a simulation.

Autoconfig Files

Both the system bus and the flexiPCS use autoconfig files to initialize memory maps. IPexpress automatically generates the system bus autoconfig file based on flexiPCS multi-quad and multi-chip alignment selection. Please refer to TN1145, LatticeSC flexiPCS/SERDES Design Guide for more information on using the system bus autoconfig file.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
February 2006	01.0	Initial release.
August 2006	01.1	Translated all VHDL code to Verilog.
		Incorporated actual “orcastra” JTAG model + driver.
January 2007	01.2	Added section about SYSBUSA simulation model.
February 2007	01.3	Clarified MPI availability/size restrictions on certain die/package combinations.
		Added information on flexiPCS options in IPexpress.
		Added information on autoconfig file.
April 2008	01.4	Removed information about internal oscillator option for System Bus clock source.
		Corrected UMI_PCS_ASYNC example. Source of System Bus clock is USER from OSCA block in sc_orcastra module, not internal oscillator.
April 2010	01.5	Appendix B, added link to LatticeSC design files page on the Lattice web site.

Appendix A. PowerPC and LatticeSC MPI Pins Connections

The MPI interface on the system bus implements a synchronous slave interface to Motorola PowerPC860 bus with up to 32-bit data/4-bit parity. Figure 25 shows how the MPI interface is connected to the PowerPC bus to act as a bus slave. Table 21 shows the MPI data and address busses' pin-to-pin connection to the PowerPC address and data busses.

Figure 25. MPI Connection to PowerPC

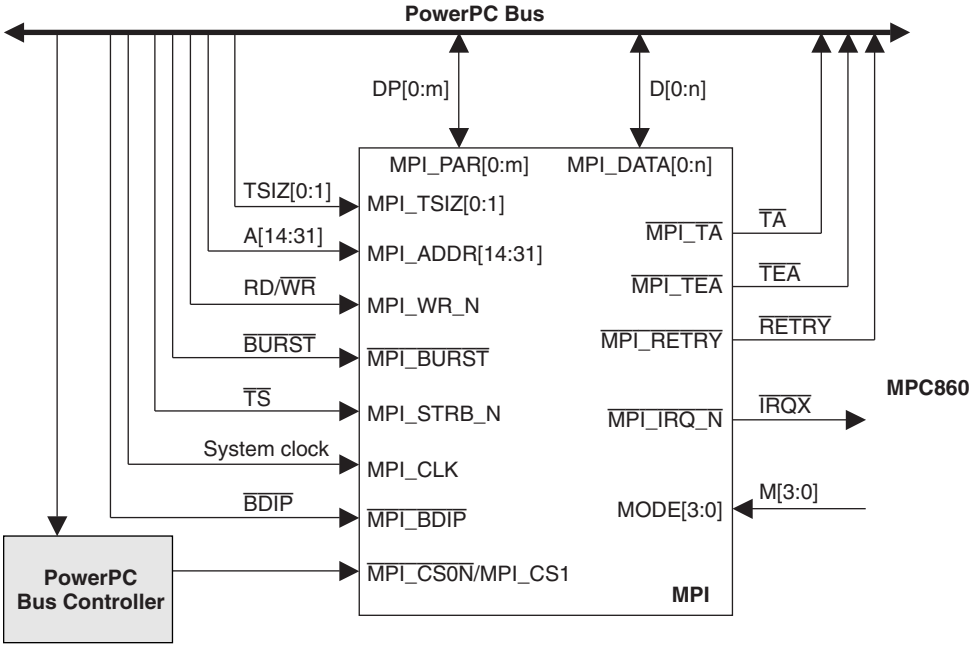


Table 21. PowerPC and MPI DATA/ADDRESS Pin Connections

PPC Address	LatticeSC MPI Address Pin	PPC Data Pin	LatticeSC MPI Data Pin
31	31	0	0
30	30	1	1
29	29	2	2
28	28	3	3
27	27	4	4
26	26	5	5
25	25	6	6
24	24	7	7
23	23	8	8
22	22	9	9
21	21	10	10
20	20	11	11
19	19	12	12
18	18	13	13
17	17	14	14
16	16	15	15
15	15	16	16

Table 21. PowerPC and MPI DATA/ADDRESS Pin Connections (Continued)

PPC Address	LatticeSC MPI Address Pin	PPC Data Pin	LatticeSC MPI Data Pin
14	14	17	17
13	NC	18	18
12	NC	19	19
11	NC	20	20
10	NC	21	21
9	NC	22	22
8	NC	23	23
7	NC	24	24
6	NC	25	25
5	NC	26	26
4	NC	27	27
3	NC	28	28
2	NC	29	29
1	NC	30	30
0	NC	31	31

Appendix B. Generation and Simulation Examples

This section covers several examples of generating and simulating the system bus module. Each example involves one master and one slave interface. Note that all the examples below use 8-bit wide data read and write accesses. Table 22 lists each example covered, as well as a brief description for each. A compressed file containing all these examples is available on the Lattice web site by navigating to the LatticeSC product page and clicking on the Design Files link.

Table 22. List of System Bus Examples

Example Name	Master	Slave	Description
MPI_USI_SYNC	MPI	USI	System bus is clocked by the MPI clock. USI synchronous to system bus.
MPI_USI_ASYNC	MPI	USI	System is bus clocked by the MPI clock. USI asynchronous to system bus.
MPI_PCS	MPI	PCS	System bus is clocked by the MPI clock. PCS in Gigabit Ethernet Mode. Demonstrates interrupt from PCS to MPI.
MPI_DFA	MPI	DFA	System bus is clocked by the MPI clock. Demonstrates interrupts from USR_IRQ_IN to MPI.
MPI_SMI	MPI	SMI	System bus is clocked by the MPI clock. Also accesses the system bus read-only registers.
UMI_SMI_SYNC	UMI	SMI	System bus is clocked by the USER clock. UMI synchronous to system bus. Demonstrates interrupt from UMI_IRQ and USI_IRQ_IN to USR_IRQ_OUT.
UMI_PCS_ASYNC	UMI	PCS	System bus is clocked by the USER clock. The oscillator drives both the USER clock and the UMI clock. PCS runs auto-configuration file. PCS in Gigabit Ethernet Mode. Demonstrates interrupt from PCS to UMI.

Note that for each example above, the ModelSim® script files vcom_SC_top_SE.do and vcom_SC_top_OEM.do are used to compile and simulate the example with the SE and Lattice versions of ModelSim respectively, as later described in each example's simulation section. The scripts must be modified to point to the current ispLEVER installation directory. Find the line shown below in the ModelSim scripts and modify the path to the proper location based on your system.

```
set isplever_dir <isplever_root_dir>
```

Where `<isplever_root_dir>` is the path to the root directory of your ispLEVER software installation.

Example:

```
set isplever_dir C:/ispTools
```

MPI_USI_SYNC

The MPI_USI_SYNC example consists of a design with the following blocks:

- A system bus (systembus) with MPI and USI interfaces. The system bus is generated such that the USI is synchronous to the system bus clock.
- A sw_ctrlstat block to interface to the system bus USI interface. This block contains 4 R/W 8-bit wide user slave interface (USI) registers at addresses 0x00800, 0x08000, 0x10000 and 0x2FFFF respectively.
- A top-level design (SC_top) to stitch all the lower blocks together.

For simulation purposes, a test bench is developed around SC_top with the following elements:

- A PowerPC model (MPU) to connect to the MPI I/O of SC_top for MPI R/W accesses.
- Drivers for all clocks and resets.
- An instance of SC_top with appropriate connections to top-level signals.

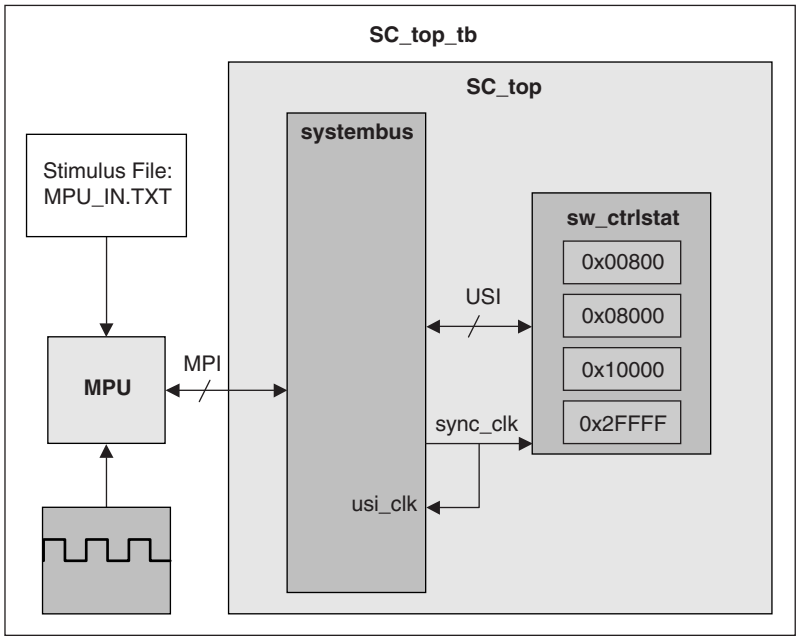
- A top-level test bench (SC_top_tb) to stitch all the lower blocks together.

The features of the example are:

- System bus clocked by MPI clock.
- USI synchronous to system bus.

Figure 26 shows the design structure including both SC_top and SC_top_tb.

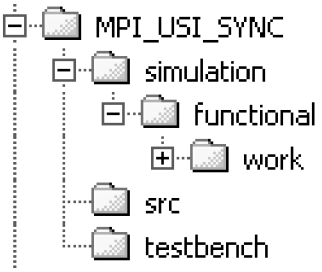
Figure 26. MPI_USI_SYNC Design Structure



MPI_USI_SYNC Directory Structure

Figure 27 shows the directory structure of the MPI_USI_SYNC project.

Figure 27. MPI_USI_SYNC Directory Structure



The “src” sub-directory contains the following design files:

- **systembus.v**: Verilog description of systembus. This file will be regenerated again using IPexpress.
- **sw_ctrlstat.v**: Verilog description of sw_ctrlstat
- **registers.v**: Verilog description of USI registers under sw_ctrlstat.
- **SC_top.v**: Verilog description of SC_top.

The test bench directory contains the following test bench files:

- **mpu.v**: Verilog model of PowerPC interface block to connect to the MPI I/O of SC_top for MPI R/W accesses.
- **SC_top_tb.v**: Verilog model of top-level test bench.

The “simulation/functional” directory contains the following MTI ModelSim files:

- **vcom_SC_top_SE.do**: do file to compile/simulate design and test bench files in SE version of ModelSim
- **vcom_SC_top_OEM.do**: do file to compile/simulate design and test bench files in Lattice version of ModelSim
- **wave.do**: Waveform file called by compile/simulation do file
- **mpu_in.txt**: stimulus file for R/W transactions for the test bench mpu.v file

Generating System Bus for MPI_USI_SYNC

To generate the system bus model, follow these steps:

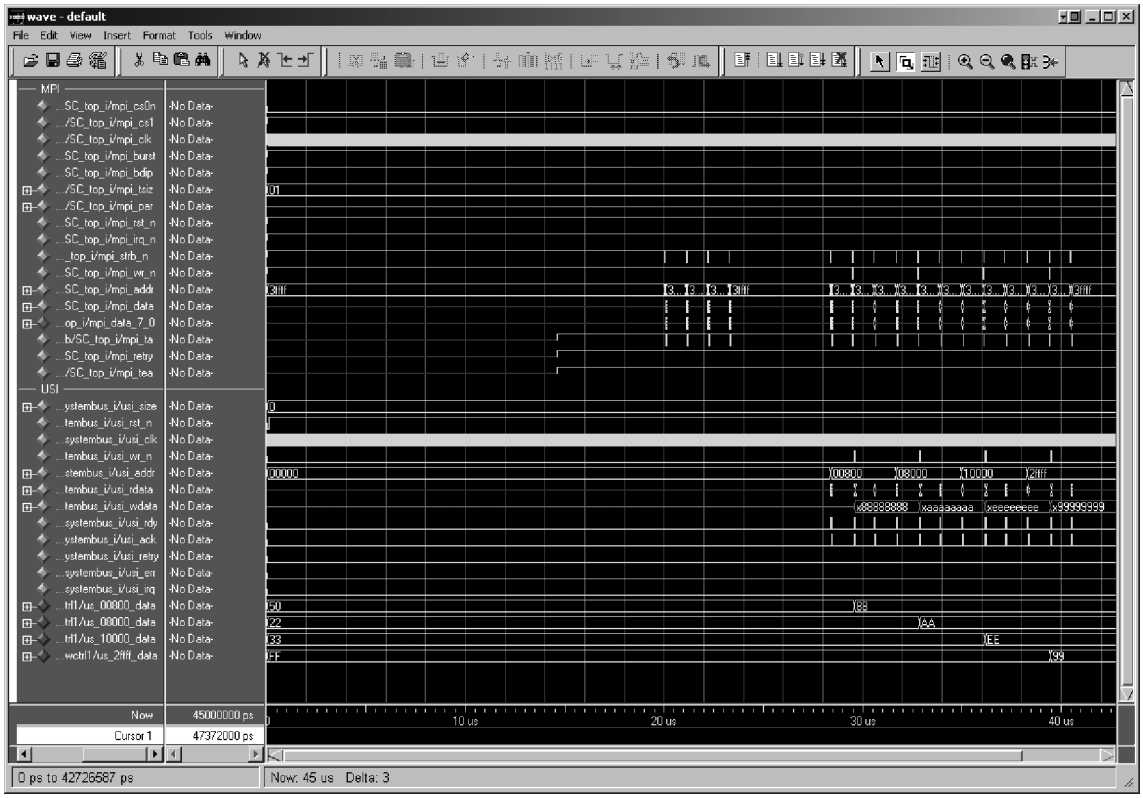
- Start a new ispLEVER Project Navigator session with a LatticeSC device.
- Launch IPexpress.
- Select Module -> Architecture_Modules -> System_Bus and set the File Name to “systembus”. Also make sure that the Design Entry is set to “Schematic/Verilog HDL”. The “Project Path” should point to “MPI_USI_SYNC/src”.
- Click on “Customize”
- In the new window:
  - Enable the MPI Interface
  - Set the MPI Bus Width to 8.
  - Enable Parity on the MPI interface
  - Set the system bus clock source to MPI.
  - Enable the User_Slave Interface.
  - Make the User Slave Synchronous to Systembus Clock
- Click on the “Generate” button (this will re-create systembus.v in “MPI_USI_SYNC /src”).

Running the MPI_USI_SYNC Simulation

To run the MPI_USI_SYNC simulation:

- Start an MTI ModelSim session.
- In ModelSim, change to the “MPI_USI_SYNC /simulation/functional” directory.
- Run the vcom_SC_top_SE.do or vcom_SE_top_OEM.do file using the Execute Macro menu option.
- Note that after running this step, you might encounter errors related to obsolete compiled libraries, which would require refreshing these libraries.
- After running the compile/simulation do file, a waveform similar to Figure 28 will appear and the simulation will run for about 45µs.

Figure 28. ModelSim Waveform for MPI_USI_SYNC



Description of MPI_USI_SYNC Signals in the ModelSim Waveform

The ModelSim waveform is divided into two sections:

- MPI signals, covering all the MPI signals at the SC_top interface
- USI signals

Description of the MPI_USI_SYNC Simulation Scenario

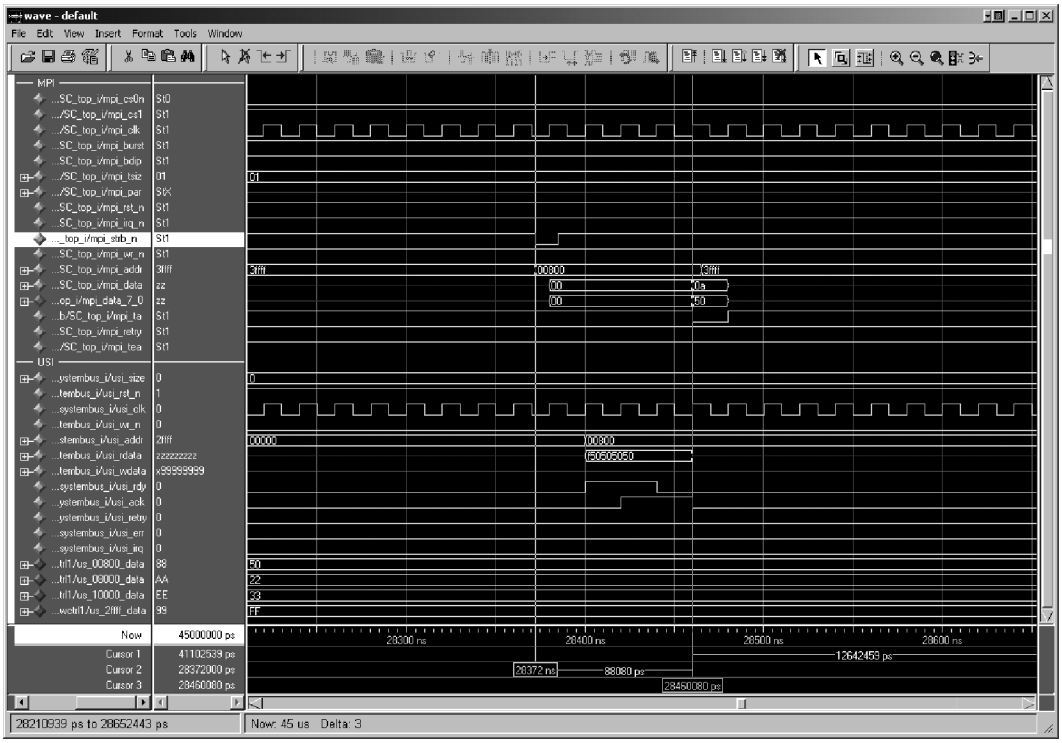
The read and write accesses to the different interfaces on the system bus are initiated by the “mpu_in.txt” file. A read or write command is reflected on the MPI Interface.

The simulation can be divided into the following phases:

- 0-20µs: the test bench waits for the LatticeSC chip and system bus to power-up.
- 20-25µs: the test bench reads the system bus ID register (0x00000-0x00003).
- 28-41µs: the test bench reads the initial value for each of the USI registers defined (0x00800, 0x08000, 0x10000 and 0x2FFFF). The test bench then writes a data value to each USI address, and reads it back. The initial value, and data written for each register is as follows:
  - 0x00800: initial data[7:0]=0x50, written data[7:0]=0x88
  - 0x08000: initial data[7:0]=0x22, written data[7:0]=0xAA
  - 0x10000: initial data[7:0]=0x33, written data[7:0]=0xEE
  - 0x2FFFF: initial data[7:0]=0xFF, written data[7:0]=0x99

Because the USI interface is synchronous to the system bus clock, the delay time from an active MPI_STRB_N to an active MPI_TA when reading USI 0x00800 register from the MPI is only about 89ns (on a 20ns MPI_CLK period). This is shown in Figure 29.

Figure 29. MPI to USI read access for MPI_USI_SYNC



MPI_USI_ASYNC

The MPI_USI_ASYNC example consists of a design with the following blocks:

- A system bus (systembus) with MPI and USI interfaces. The system bus is generated such that the USI is not synchronous to the system bus clock.
- A “sw_ctrlstat” block to interface to the system bus USI interface. This block contains 4 R/W 8-bit wide user slave interface (USI) registers at addresses 0x00800, 0x08000, 0x10000 and 0x2FFFF respectively.
- A top-level design (SC_top) to stitch all the lower blocks together.

For simulation purpose, a test bench is developed around SC_top with the following elements:

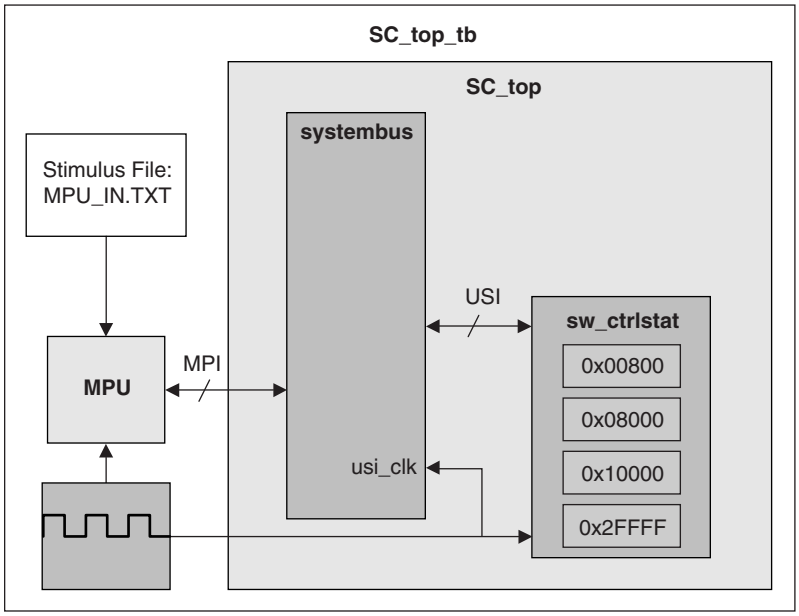
- A PowerPC model (MPU) to connect to the MPI I/O of SC_top for MPI R/W accesses.
- Drivers for all clocks and resets.
- An instance of SC_top with appropriate connections to top-level signals.
- A top-level test bench (SC_top_tb) to stitch all the lower blocks together.

The features of the example are:

- System bus clocked by MPI clock.
- USI asynchronous to system bus. Clocked from test bench.

Figure 30 shows the design structure including both SC_top and SC_top_tb.

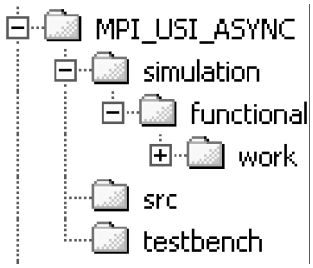
Figure 30. MPI_USI_ASYNC Design Structure



MPI_USI_ASYNC Directory Structure

Figure 31 shows the directory structure of the MPI_USI_ASYNC project.

Figure 31. MPI_USI_ASYNC Directory Structure



The “src” sub-directory contains the following design files:

- systembus.v: Verilog description of systembus. This file will be regenerated again using IPexpress.
- sw_ctrlstat.v: Verilog description of sw_ctrlstat
- registers.v: Verilog description of USI registers under sw_ctrlstat.
- SC_top.v: Verilog description of SC_top.

The test bench directory contains the following test bench files:

- mpu.v: Verilog model of PowerPC interface block to connect to the MPI I/O of SC_top for MPI R/W accesses.
- SC_top_tb.v: Verilog model of top-level test bench.

The “simulation/functional” directory contains the following MTI ModelSim files:

- vcom_SC_top_SE.do: do file to compile/simulate design and test bench files in SE version of Modelsim
- vcom_SC_top_OEM.do: do file to compile/simulate design and test bench files in Lattice version of ModelSim

- wave.do: Waveform file called by compile/simulation do file
- mpu_in.txt: stimulus file for R/W transactions for the test bench mpu.v file

Generating System Bus for MPI_USI_ASYNC

To generate the system bus model, follow the following steps:

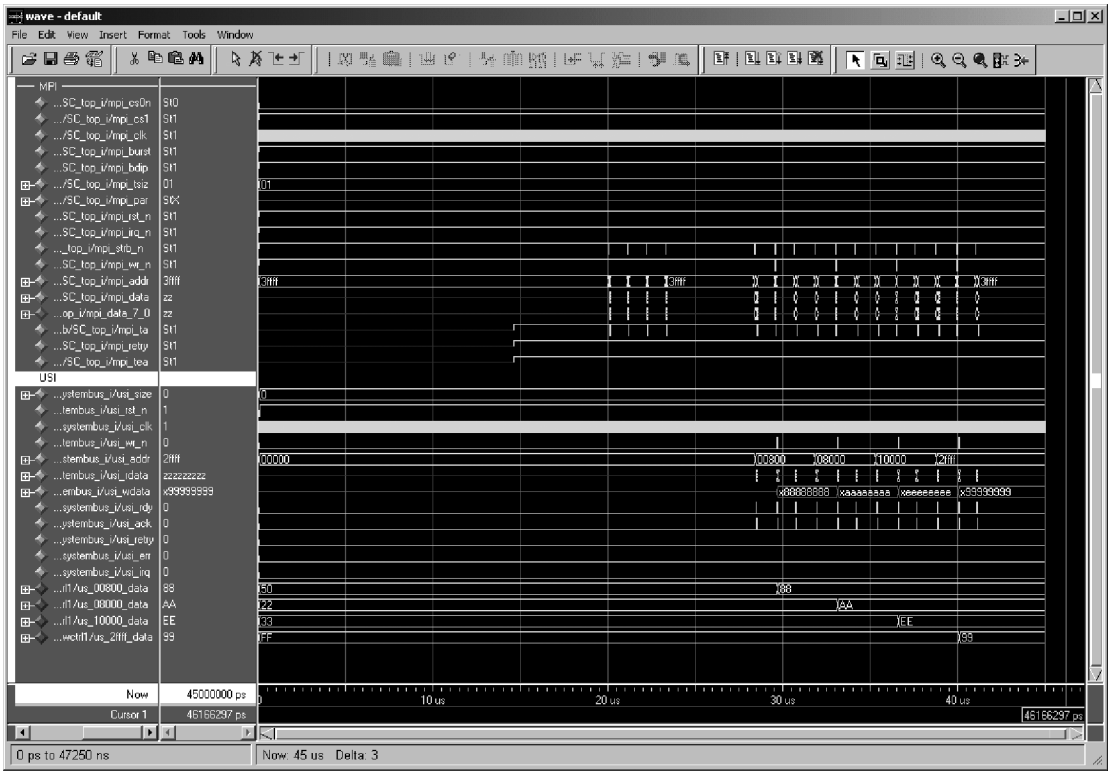
- Start a new ispLEVER Project Navigator session with a LatticeSC device.
- Launch IPexpress.
- Select Module -> Architecture_Modules -> System_Bus and set the File Name to “systembus”. Also make sure that the Design Entry is set to “Schematic/Verilog HDL”. The “Project Path” should point to “MPI_USI_ASYNC/src”.
- Click on “Customize”
- In the new window:
  - Enable the MPI Interface
  - Set the MPI Bus Width to 8.
  - Enable Parity on the MPI interface
  - Set the system bus clock source to MPI.
  - Enable the User Slave Interface.
- Click on the “Generate” button (this will re-create systembus.v in “MPI_USI_ASYNC /src”).

Running the MPI_USI_ASYNC Simulation

To run the MPI_USI_ASYNC simulation

- Start an MTI ModelSim session.
- In ModelSim, change to the “MPI_USI_ASYNC /simulation/functional” directory.
- Run the vcom_SC_top_SE.do or vcom_SE_top_OEM.do file using the Execute Macro menu option.
- Note that after running this step, you might encounter errors related to obsolete compiled libraries, which would require refreshing these libraries.
- After running the compile/simulation do file, a waveform similar to Figure 32 should appear and the simulation should run for about 45µs.

Figure 32. ModelSim Waveform for MPI_USI_ASYNC



Description of MPI_USI_ASYNC Signals in the ModelSim Waveform

The ModelSim waveform is divided into two sections:

- MPI Interface signals, covering all the MPI signals at the SC_top interface
- USI signals

Description of the MPI_USI_ASYNC Simulation Scenario

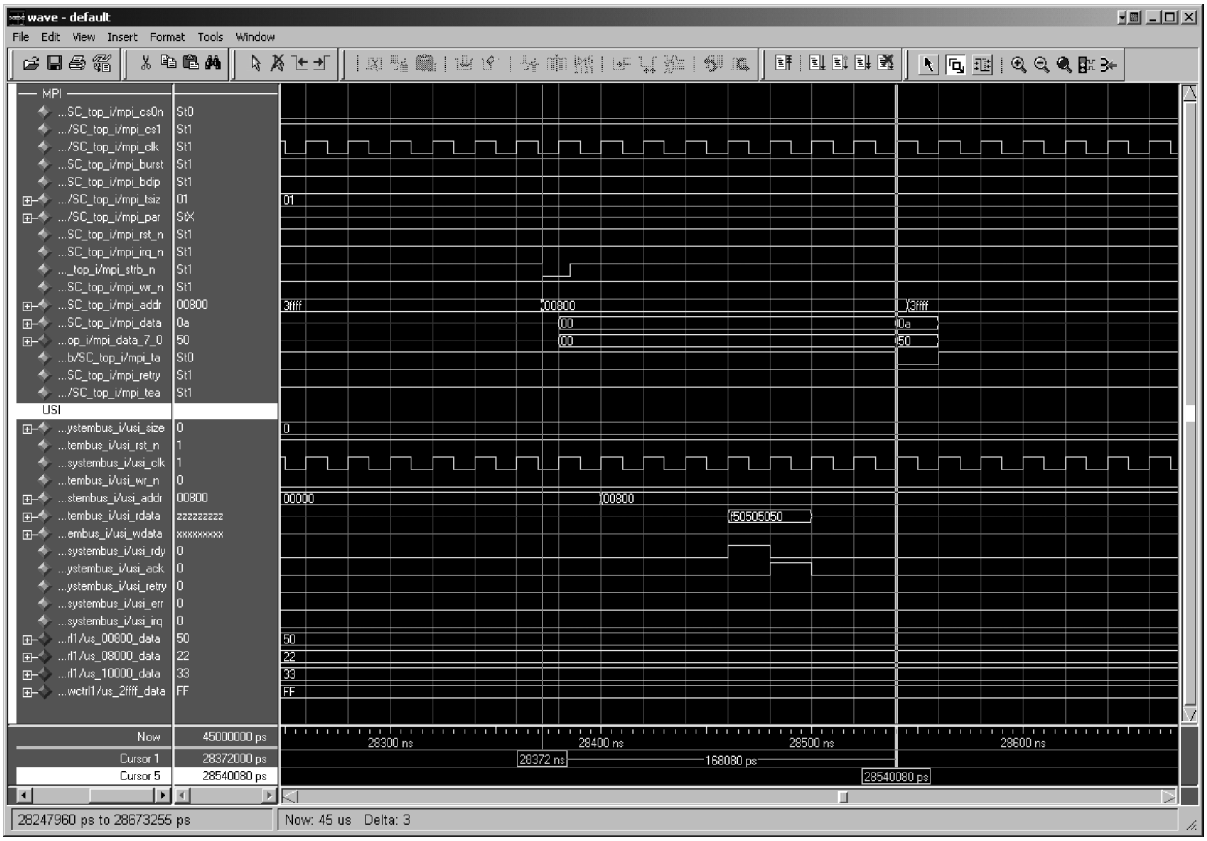
The read and write accesses to the different interfaces on the system bus are initiated by the “mpu_in.txt” file. A read or write command is reflected on the MPI Interface.

The simulation can be divided into the following phases:

- 0-20µs: the test bench waits for the LatticeSC chip and system bus to power-up.
- 20-25µs: the test bench reads the system bus ID register (0x00000-0x00003).
- 28-41µs: the test bench reads the initial value for each of the USI registers defined (0x00800, 0x08000, 0x10000 and 0x2FFFF). The test bench then writes a data value to each USI address, and reads it back. The initial value, and data written for each register is as follows:
  - 0x00800: initial data[7:0]=0x50, written data[7:0]=0x88
  - 0x08000: initial data[7:0]=0x22, written data[7:0]=0xAA
  - 0x10000: initial data[7:0]=0x33, written data[7:0]=0xEE
  - 0x2FFFF: initial data[7:0]=0xFF, written data[7:0]=0x99

Because the USI interface is asynchronous to the system bus clock, the delay time from an active MPI_IRQ to an active MPI_TA when reading USI 0x00800 register from the MPI is about 169ns (with a 20ns MPI_CLK and USI_CLK period). This is shown in Figure 33. This delay is about twice that of the MPI_USI_SYNC one shown in Figure 29.

Figure 33. MPI to USI Read Access for MPI_USI_ASYNC



MPI_PCS

The MPI_PCS example consists of a design with the following blocks:

- A system bus (systembus) with MPI and PCS interfaces.
- A PCS block (PCS0) with channel 0 configured in Gigabit Ethernet mode.
- A pattern generator (cjpat_gen), which reads data content from a distributed ROM (MYROM) and sends it to the TX FPGA data interface of PCS0.
- A PLL block (PLLDIV2) used by the cjpat_gen block.
- A top-level design (SC_top) to stitch all the lower blocks together.

For simulation purpose, a test bench is developed around SC_top with the following elements:

- A PowerPC model (MPU) to connect to the MPI I/O of SC_top for MPI R/W accesses.
- Drivers for all clocks and resets.
- An instance of SC_top with appropriate connections to top-level signals.
- A top-level test bench (SC_top_tb) to stitch all the lower blocks together. The test bench connects the PCS0 hdout* pins back to the hdin* pins to form an external loop-back connection (TX back to RX)

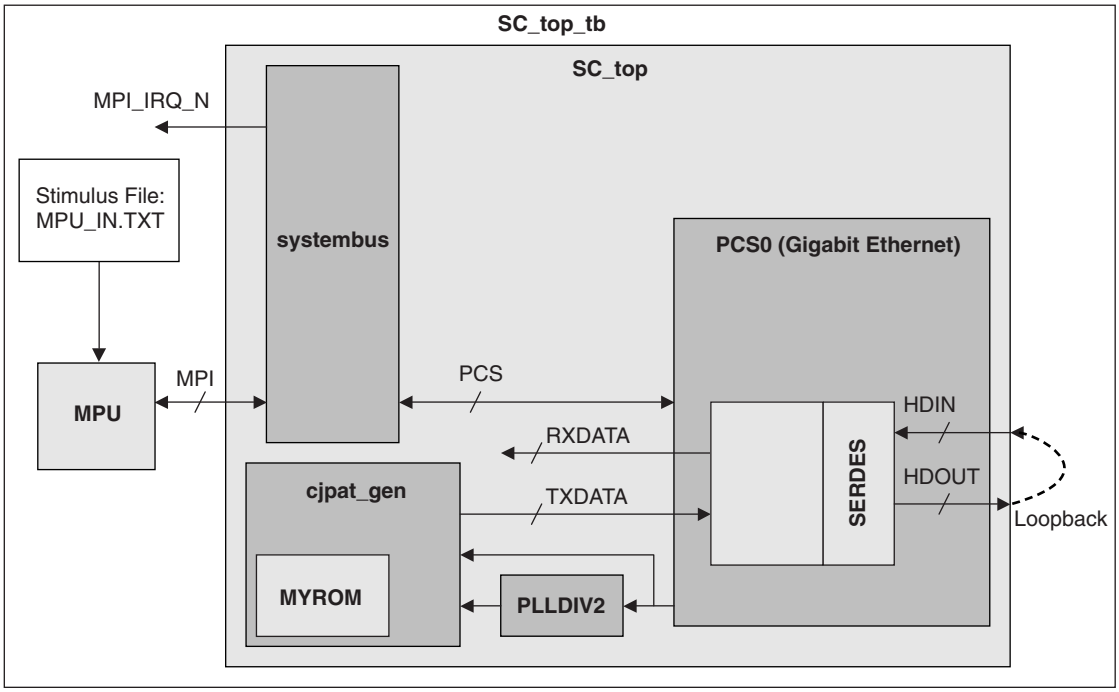
The features of the example are:

- System bus clocked by MPI clock.
- PCS channel 0 generated in Gigabit Ethernet Mode.

- Demonstrates Interrupt from PCS to MPI. The interrupt is enabled when the link state machine of channel 0 reaches a synchronized state.

Figure 34 shows the design structure including both SC_top and SC_top_tb.

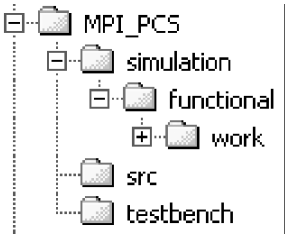
Figure 34. MPI_PCS Design Structure



MPI_PCS Directory Structure

Figure 35 shows the directory structure of the MPI_PCS project.

Figure 35. MPI_PCS Directory Structure



The “src” sub-directory contains the following design files:

- systembus.v”: Verilog description of systembus. This file will be regenerated again using IPexpress.
- PCS0.v: Verilog description of PCS0.
- cjpat_gen.v: Verilog description of cjpat_gen.
- PLLDIV2.v: Verilog description of PLLDIV2.
- MYROM.v: Verilog description of MYROM.
- SC_top.v: Verilog description of SC_top.

The test bench directory contains the following test bench files:

- mpu.v: Verilog model of PowerPC interface block to connect to the MPI I/O of SC_top for MPI R/W accesses.
- SC_top_tb.v: Verilog model of top-level test bench.

The “simulation/functional” directory contains the following MTI ModelSim files:

- vcom_SC_top_SE.do: do file to compile/simulate design and test bench files in SE version of ModelSim
- vcom_SC_top_OEM.do: do file to compile/simulate design and test bench files in Lattice version of ModelSim
- wave.do: Waveform file called by compile/simulation do file
- mpu_in.txt: stimulus file for R/W transactions for the test bench mpu.v file

Generating System Bus for MPI_PCS

To generate the system bus model, follow the following steps:

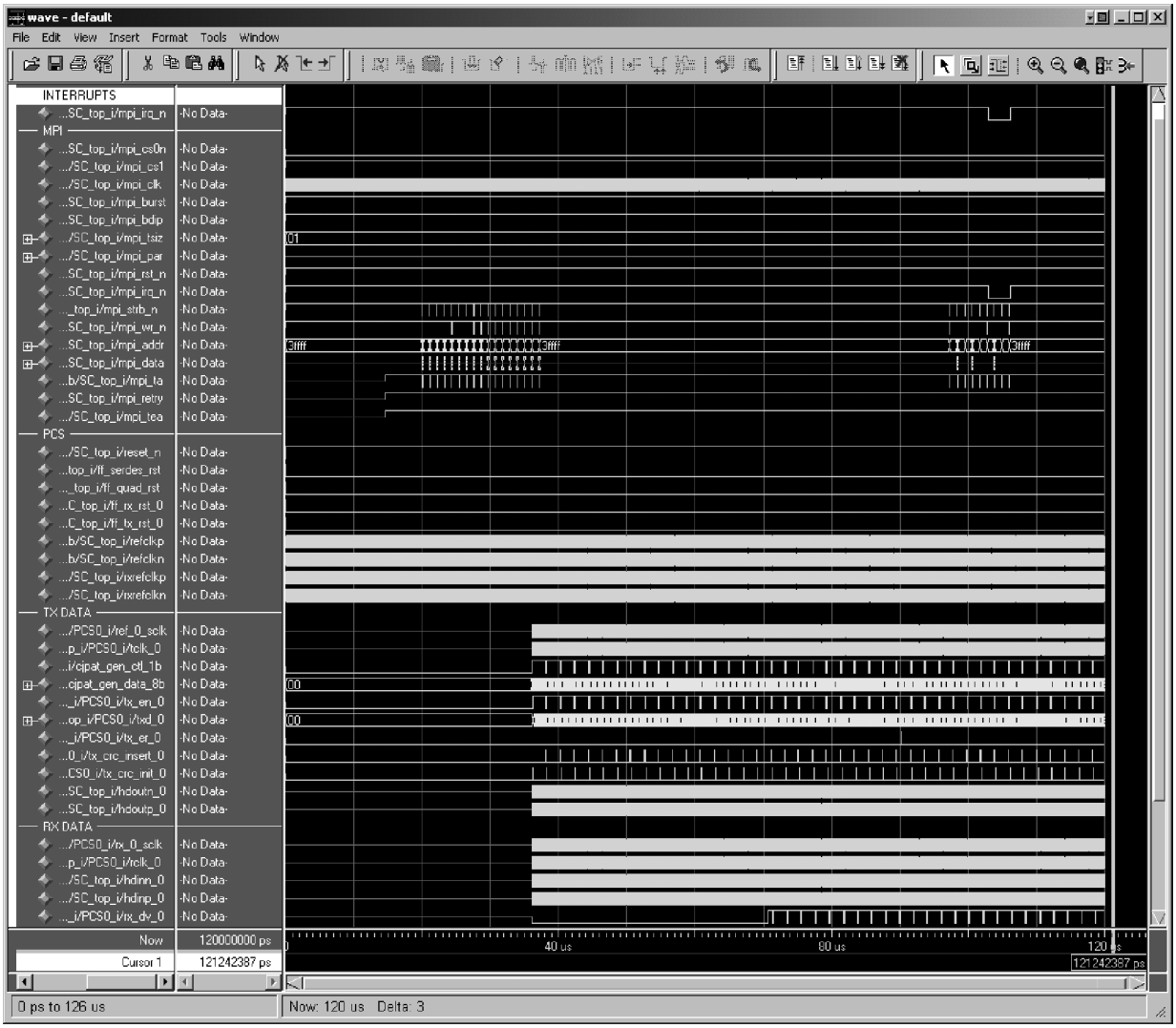
- Start a new ispLEVER Project Navigator session with a LatticeSC device.
- Launch IPexpress.
- Select Module -> Architecture_Modules -> System_Bus and set the File Name to “systembus”. Also make sure that the Design Entry is set to “Schematic/Verilog HDL”. The “Project Path” should point to “MPI_PCS/src”.
- Click on “Customize”
- In the new window:
  - Enable the MPI Interface
  - Set the MPI Bus Width to 8.
  - Enable Parity on the MPI interface
  - Set the system bus clock source to MPI.
  - Enable the PCS 360 Interface.
- Click on the “Generate” button (this will re-create systembus.v in “MPI_PCS /src”).

Running the MPI_PCS Simulation

To run the MPI_PCS simulation

- Start an MTI ModelSim session.
- In ModelSim, change to the “MPI_PCS /simulation/functional” directory.
- Run the vcom_SC_top_SE.do or vcom_SE_top_OEM.do file using the Execute Macro menu option.
- Note that after running this step, you might encounter errors related to obsolete compiled libraries, which would require refreshing these libraries.
- After running the compile/simulation do file, a waveform similar to Figure 36 should appear and the simulation should run for about 120µs.

Figure 36. ModelSim Waveform for MPI_PCS



Description of MPI_PCS Signals in the ModelSim Waveform

The ModelSim waveform is divided into four sections:

- Interrupt signals
- MPI signals
- PCS signals, including TX and RX
- System bus PCS interface

Description of the MPI_PCS Simulation Scenario

The read and write accesses to the different interfaces on the system bus are initiated by the “mpu_in.txt” file. A read or write command is reflected on the MPI Interface. The simulation can be divided into the following phases:

- 0-20µs: the test bench waits for the LatticeSC chip and system bus to power-up.
- 20-24µs: the test bench reads the system bus ID register (0x00000-0x00003).
- 24-26µs: the test bench writes data[0:7]=0x11 to system bus scratch pad register 0x00004 and reads it back.

- 26-37µs: the test bench configures the PCS0 block registers such that:
  - Channel 0 is in Gigabit Ethernet Mode.
  - CRC insertion is enabled for both TX and RX.
  - As a result, the PCS0 interface clocks start toggling around 36µs. This enables the TX data generation from cjpat_gen to PCS0. Also, around 70µs, the PCS0 recovered data starts toggling properly. This data should be the receive equivalent of the TX data since the PCS0 block is in external loop-back.
- 90-91 µs: as part of the data flow into the TX direction, an error is inserted through the tx_er_0 signal. As a result, the error propagates to the rx_er_0 signal in the receive direction.
- 96-107 µs: this period of time shows how an interrupt from the PCS can be propagated to the MPI_IRQ_N output. During this phase:
  - Interrupts from the PCS to the MPI_IRQ_N output are enabled by writing data[0:7]=0x40 to 0x00013
  - The system bus interrupt cause register (0x00010) is read. The value is data[0:7]=0x00, indicating that no interrupts were received by the system bus.
  - The PCS0 quad interface register 0x84 (0x36084) is read. The value is data[4:7]=0x8, indicating that the link state machine for channel 0 is synchronized.
  - The PCS0 quad interface register 0x1C (0x3601C) is read. The value is data[0:7]=0x00, indicating that the interrupt enable bit corresponding to a synchronized link state machine for channel 0 is off.
  - The PCS0 quad interface register 0x85 (0x36085) is read. The value is data[0:7]=0x00, indicating that the interrupt bit corresponding to a synchronized link state machine for channel 0 is off.
  - The PCS0 quad interface register 0x1C (0x3601C) is written to data[0:7]=0x08. to turn on the interrupt enable bit corresponding to a synchronized link state machine for channel 0. This enables the interrupt from 0x36085, bit 4, which asserts MPI_IRQ_N to a ‘0’ value.
  - The PCS0 quad interface register 0x85(0x36085) is read. The value is data[0:7]=0x08, indicating that the interrupt bit corresponding to a synchronized link state machine for channel 0 is ON.
  - The system bus interrupt cause register (0x00010) is read. The value is data[0:7]=0x40, indicating that an interrupt was received by the system bus from the PCS.
  - Interrupts from the PCS to the MPI_IRQ_N output are disabled by writing data[0:7]=0x00 to 0x00013. This de-asserts MPI_IRQ_N to a ‘1’ value.

MPI_DFA

The MPI_DFA example consists of a design with the following blocks:

- System bus (systembus) with MPI and DFA interfaces.
- DFA interface block (dfa_8bit). It allows DFA accesses to write to and read from a 256X8-distributed RAM (ram_dfa0_8bit). Note that the dfa_8bit is designed such that, when selected, it will:
  - Respond normally to write and read accesses to ram_dfa0_8bit in the address range: 0x00000-0x000fd
  - Respond with data[0:7]=0xfe, and an active low TEA (DFA_TEA/MPI_TEA) when address=0x000fe is read
  - Respond with data[0:7]=0xff, and an active low RETRY (DFA_RETRY/MPI_RETRY) when address=0x000ff is read
- Top-level design (SC_top) to stitch all the lower blocks together.

For simulation purposes, a test bench is developed around SC_top with the following elements:

- PowerPC model (MPU) to connect to the MPI I/O of SC_top for MPI R/W accesses.
- Drivers for all clocks and resets.
- An instance of SC_top with appropriate connections to top-level signals.
- A top-level test bench (SC_top_tb) to stitch all the lower blocks together.

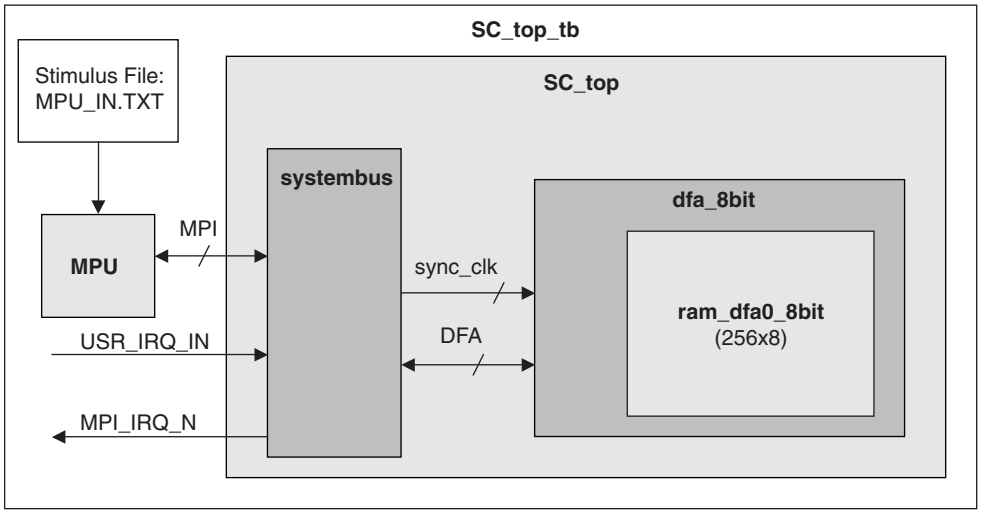
The features of the example are:

- System bus clocked by MPI clock.

- Demonstrates Interrupts from USR_IRQ_IN to MPI.

Figure 37 shows the design structure including both SC_top and SC_top_tb.

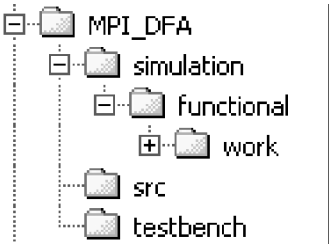
Figure 37. MPI_DFA Design Structure



MPI_DFA Directory Structure

Figure 38 shows the directory structure of the MPI_DFA project.

Figure 38. MPI_DFA Directory Structure



The “src” sub-directory contains the following design files:

- systembus.v”: Verilog description of systembus. This file will be regenerated again using IPexpress.
- dfa_8bit.v: Verilog description of dfa_8bit.
- ram_dfa0_8bit.v: Verilog description of ram_dfa0_8bit.
- SC_top.v: Verilog description of SC_top.

The test bench directory contains the following test bench files:

- mpu.v: Verilog model of PowerPC interface block to connect to the MPI I/O of SC_top for MPI R/W accesses.
- SC_top_tb.v: Verilog model of top-level test bench.

The “simulation/functional” directory contains the following MTI ModelSim files:

- vcom_SC_top_SE.do: do file to compile/simulate design and test bench files in SE version of ModelSim
- vcom_SC_top_OEM.do: do file to compile/simulate design and test bench files in Lattice version of ModelSim
- wave.do: Waveform file called by compile/simulation do file
- mpu_in.txt: stimulus file for R/W transactions for the test bench mpu.v file

Generating System Bus for MPI_DFA

To generate the system bus model, follow the following steps:

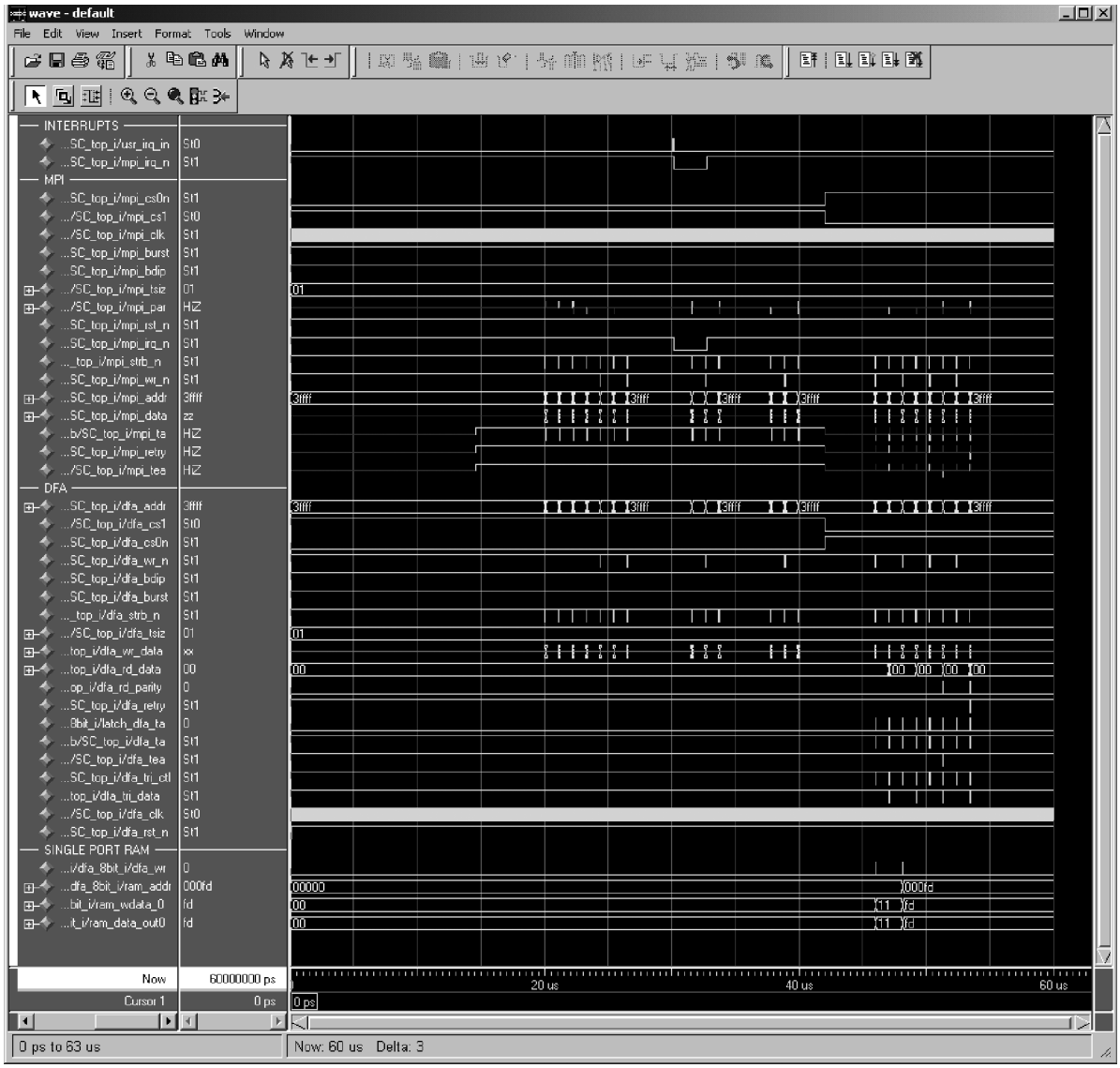
- Start a new ispLEVER Project Navigator session with a LatticeSC device.
- Launch IPexpress.
- Select Module -> Architecture_Modules -> System_Bus and set the File Name to “systembus”. Also make sure that the Design Entry is set to “Schematic/Verilog HDL”. The “Project Path” should point to “MPI_DFA/src”.
- Click on “Customize”
- In the new window:
  - Enable the MPI Interface
  - Set the MPI Bus Width to 8.
  - Enable Parity on the MPI interface
  - Set the system bus clock source to MPI.
  - Enable the DFA Interface.
- Click on the “Generate” button (this will re-create systembus.v in “MPI_DFA /src”).

Running the MPI_DFA Simulation

To run the MPI_DFA simulation

- Start an MTI ModelSim session.
- In ModelSim, change to the “MPI_DFA /simulation/functional” directory.
- Run the vcom_SC_top_SE.do or vcom_SE_top_OEM.do file using the Execute Macro menu option.
- Note that after running this step, you might encounter errors related to obsolete compiled libraries, which would require refreshing these libraries.
- After running the compile/simulation do file, a waveform similar to Figure 39 should appear and the simulation should run for about 60µs.

Figure 39. ModelSim Waveform for MPI_DFA



Description of MPI_DFA Signals in the ModelSim Waveform

The ModelSim waveform is divided into 4 sections

- Interrupt signals
- The MPI signals
- The DFA signals
- The Single Port RAM signals

Description of the MPI_DFA Simulation Scenario

The read and write accesses to the different interfaces on the system bus are initiated by the “mpu_in.txt” file. A read or write command is reflected on the MPI Interface. The simulation can be divided into the following phases:

- Initial values: MPI_CS0N=0,MPI_CS1=1, and system bus register 0x0000A contains 0x00. This will enable the MPI interface to select the system bus for both write and read accesses.
- 0-20µs: the test bench waits for the LatticeSC chip and system bus to power-up.
- 20-24µs: the test bench reads the system bus ID register (0x00000-0x00003).
- 24-26µs: the test bench writes data[0:7]=0x11 to system bus scratch pad register 0x00004 and reads it back.
- 26-34µs: this period of time shows how an interrupt from the USR_IRQ_IN can be propagated to the MPI_IRQ_N output. During this phase:
  - Interrupts from USR_IRQ_IN to the MPI_IRQ_N output are enabled by writing data[0:7]=0x01 to 0x00013.
  - A USR_IRQ_IN pulse is generated. This creates an interrupt to the system bus and asserts MPI_IRQ_N to a '0' value.
  - The system bus interrupt cause register (0x00010) is read. The value is data[0:7]=0x01, indicating that an interrupts was received by the system bus from USR_IRQ_IN.
  - The system bus interrupt cause register (0x00010) written to data[0:7]=0x01, to clear the interrupt bit from USR_IRQ_IN. This de-asserts MPI_IRQ_N to a '1' value.
  - The system bus interrupt cause register (0x00010) is read. The value is data[0:7]=0x00, indicating that no more interrupts are received by the system bus.
- 37-40µs: the test bench reads the content of system bus register 0x0000A, writes data[0:7]=0x01 to it and reads back the value. The 0x01 value enables the DFA interface.
- 42µs: MPI_CS0N=1,MPI_CS1=0. At this point, MPI interface is selecting the DFA interface for both write and read accesses.
- 46-48µs: the test bench writes data[0:7]=0x11 DFA address 0x00000 and reads it back successfully
- 48-50µs: the test bench writes data[0:7]=0xfd DFA address 0x000fd and reads it back successfully
- 50-52µs: the test bench writes data[0:7]=0x00 DFA address 0x000fe and reads the address back. At that point, the DFA interface responds with MPI_TEA (active low) and data[0:7]=0xfe. This behavior reflects the proper operation of dfa_8bit.
- 52-54µs: the test bench writes data[0:7]=0x00 DFA address 0x000ff and reads the address back. At that point, the DFA interface responds with MPI_RETRY (active low) and data[0:7]=0xff. This behavior reflects the proper operation of dfa_8bit.

MPI_SMI

The MPI_SMI example consists of a design with the following blocks:

- A system bus (systembus) with MPI and SMI interfaces.
- Two identical PLL (PLL1 with SMI offset 0x410, and PLL2 with SMI offset 0x420) to interface to the system bus SMI.
- A top-level design (SC_top) to stitch all the lower blocks together.

For simulation purpose, a test bench is developed around SC_top with the following elements:

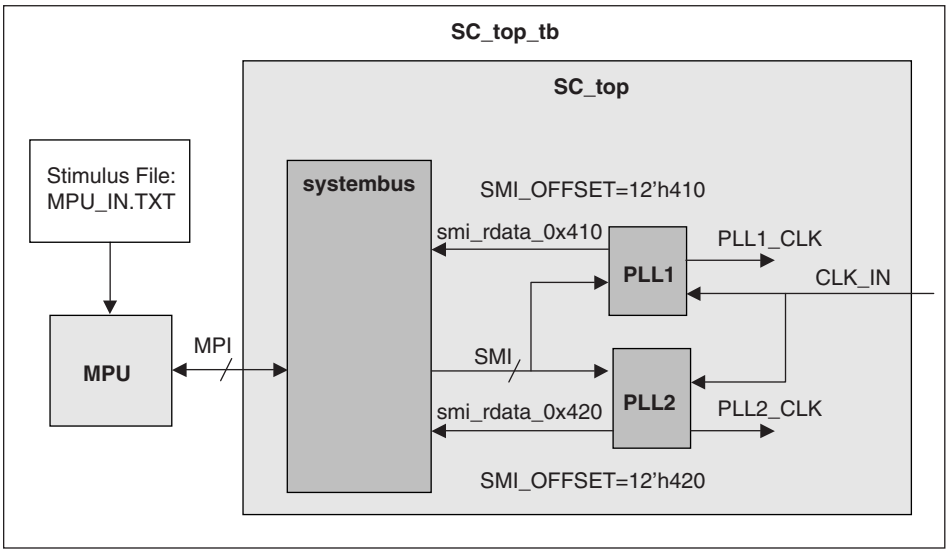
- A PowerPC model (MPU) to connect to the MPI I/O of SC_top for MPI R/W accesses.
- Drivers for all clocks and resets.
- An instance of SC_top with appropriate connections to top-level signals.
- A top-level test bench (SC_top_tb) to stitch all the lower blocks together.

The features of the example are:

- system bus clocked by MPI clock.
- system bus Read Only Registers are configured and read-back in test bench.

Figure 40 shows the design structure including both SC_top and SC_top_tb.

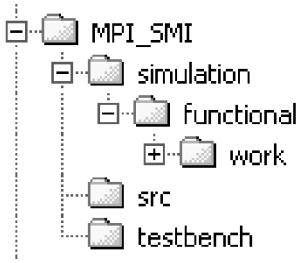
Figure 40. MPI_SMI Design Structure



MPI_SMI Directory Structure

Figure 41 shows the directory structure of the MPI_SMI project.

Figure 41. MPI_SMI Directory Structure



The "src" sub-directory contains the following design files:

- systembus.v": Verilog description of systembus. This file will be regenerated again using IPexpress.
- pll.v: Verilog PLL model for PLL1 and PLL2.
- SC_top.v: Verilog description of SC_top.

The test bench directory contains the following test bench files:

- mpu.v: Verilog model of PowerPC interface block to connect to the MPI I/O of SC_top for MPI R/W accesses.
- SC_top_tb.v: Verilog model of top-level test bench.

The “simulation/functional” directory contains the following MTI ModelSim files:

- vcom_SC_top_SE.do: do file to compile/simulate design and test bench files in SE version of ModelSim
- vcom_SC_top_OEM.do: do file to compile/simulate design and test bench files in Lattice version of ModelSim
- wave.do: Waveform file called by compile/simulation do file
- mpu_in.txt: stimulus file for R/W transactions for the test bench mpu.v file

Generating System Bus for MPI_SMI

To generate the system bus model, follow the following steps:

- Start a new ispLEVER Project Navigator session with a LatticeSC device.
- Launch IPexpress.
- Select Module -> Architecture_Modules -> System_Bus and set the File Name to “systembus”. Also make sure that the Design Entry is set to “Schematic/Verilog HDL”. The “Project Path” should point to “MPI_SMI/src”.
- Click on “Customize”
- In the new window:
  - Enable the MPI Interface
  - Set the MPI Bus Width to 8.
  - Enable Parity on the MPI interface
  - Set the system bus clock source to MPI.
  - Enable the SMI Interface, and turn on both 0x410 and 0x420 read interfaces.
  - Set the system bus Read Only Words values to:
    - Read Only Word 1: 00101011001010100010100100101000
    - Read Only Word 2: 00101111001011100010110100101100
    - Read Only Word 3: 00110011001100100011000100110000
    - Read Only Word 4: 00110111001101100011010100110100
    - Read Only Word 5: 00111011001110100011100100111000
    - Read Only Word 6: 00111111001111100011110100111100

This will set register addresses 0x00028-0x0003F to data values equal to the address values (e.g., content of address 0x00028 is data[7:0]=0x28, content of 0x0003F is data[7:0]=0x3F).

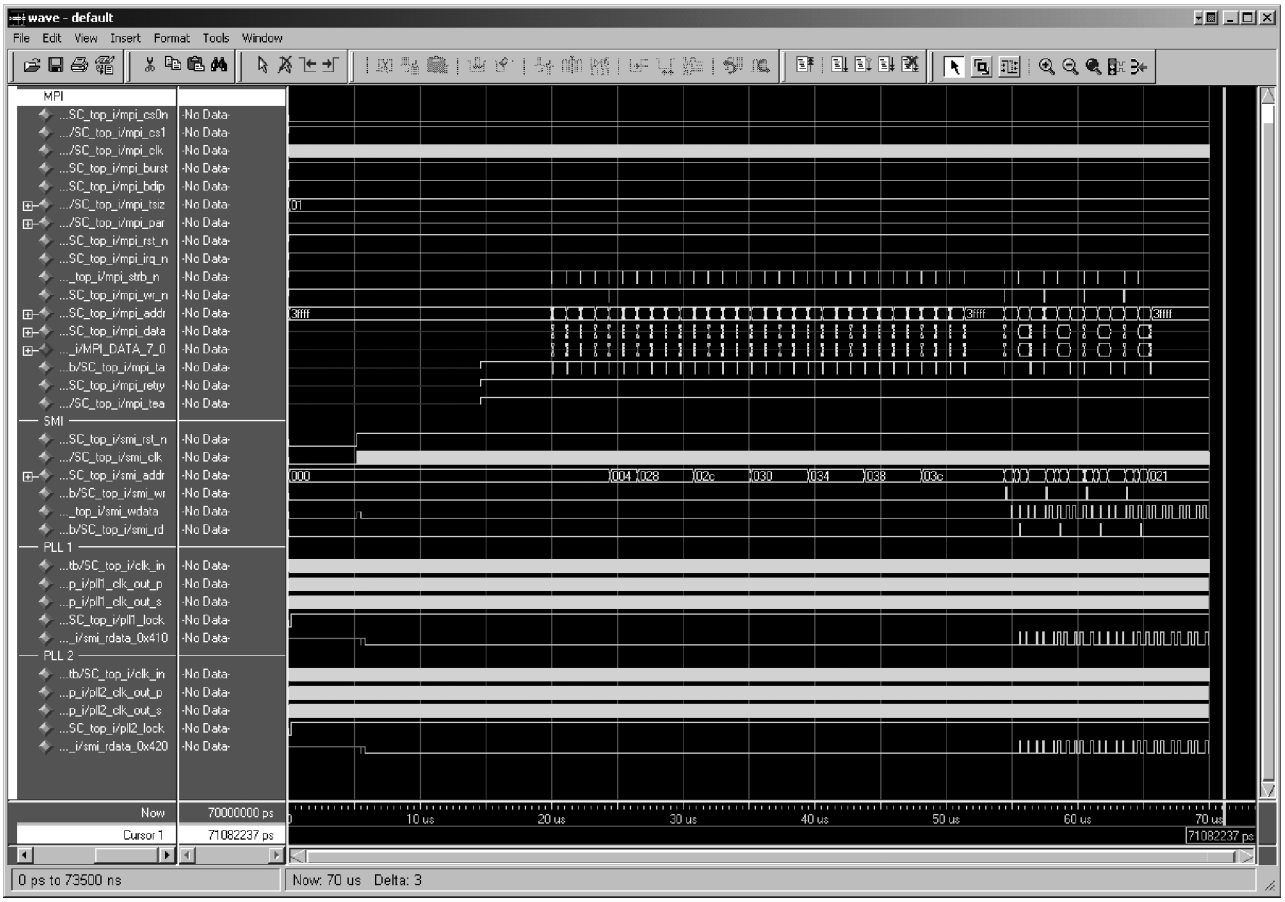
- Click on the “Generate” button (this will re-create systembus.v in “MPI_SMI /src”).

Running the MPI_SMI Simulation

To run the MPI_SMI simulation:

- Start an MTI ModelSim session.
- In ModelSim, change to the “MPI_SMI /simulation/functional” directory.
- Run the vcom_SC_top_SE.do or vcom_SE_top_OEM.do file using the Execute Macro menu option.
- Note that after running this step, you might encounter errors related to obsolete compiled libraries, which would require refreshing these libraries.
- After running the compile/simulation do file, a waveform similar to Figure 42 should appear and the simulation should run for about 70µs.

Figure 42. ModelSim Waveform for MPI_SMI



Description of MPI_SMI Signals in the ModelSim Waveform

The ModelSim waveform is divided into two sections:

- The MPI signals
- SMI, PLL1 and PLL2 section, covering the SMI signals and the PLL outputs

Description of the MPI_SMI Simulation Scenario

The read and write accesses to the different interfaces on the system bus are initiated by the “mpu_in.txt” file. A read or write command is reflected on the MPI Interface. The simulation can be divided into the following phases:

- 0 to 20µs: the test bench waits for the LatticeSC chip and system bus to power-up.
- 20 to 24µs: the test bench reads the system bus ID register (0x00000-0x00003).
- 24 to 26µs: the test bench writes data[0:7]=0x11 to system bus scratch pad register 0x00004 and reads it back.
- 26 to 52µs: the test bench reads the six Read Only Words (0x00028-0x0003F). The values read on MPI_DATA[7:0] should be identical to MPI_ADDR for each of these read transactions
- 54 to 59µs: PLL1 is accessed via SMI by:
  - Writing address 0x00410 to data[7:0] 0x04 (divide output by 2)
  - Reading address 0x00410
  - Writing address 0x00410 to data[7:0] 0x1C (divide output by 8)
  - Reading address 0x00410

- 59 to 65µs: PLL2 is accessed via SMI by:
  - Writing address 0x00420 to data[7:0] 0x04 (divide output by 2)
  - Reading address 0x00420
  - Writing address 0x00420 to data[7:0] 0x1C (divide output by 8)
  - Reading address 0x00420

UMI_SMI_SYNC

The UMI_SMI_SYNC example consists of a design with the following blocks:

- A system bus (systembus) with UMI and SMI interfaces.
- Two identical PLL (PLL1 with SMI offset 0x410, and PLL2 with SMI offset 0x420) to interface to the system bus SMI.
- An sc_orcastra block to interface between the UMI on the system bus and an external PC JTAG interface. The sc_orcastra block translates PC JTAG instructions to UMI format.
- A top-level design (SC_top) to stitch all the lower blocks together.

For simulation purposes, a test bench is developed around SC_top with the following elements:

- A JTAG_PC interface model to connect to the JTAG I/O of SC_top for PC JTAG accesses
- Drivers for all clocks and resets.
- An instance of SC_top with appropriate connections to top-level signals.
- A top-level test bench (SC_top_tb) to stitch all the lower blocks together.

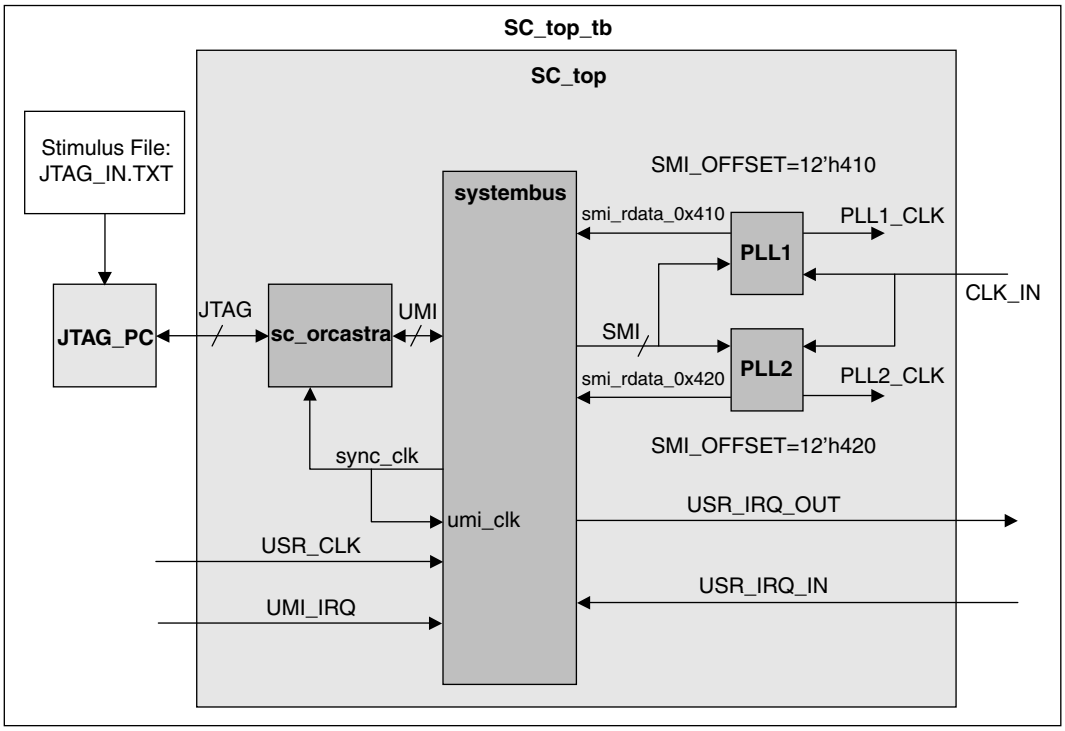
The features of the example are:

- System bus clocked by USR_CLK.
- UMI synchronous to system bus.
- Demonstrates Interrupt from UMI_IRQ and USI_IRQ_IN to USR_IRQ_OUT.

Figure 43 shows the design structure including both SC_top and SC_top_tb.

Note that the USR_CLK signal is used to clock the system bus HCLK. Also, the UMI_CLK is sourced from SYNC_CLK (HCLK). The UMI_IRQ, USI_IRQ_IN, and USR_IRQ_OUT are also brought to primary I/Os.

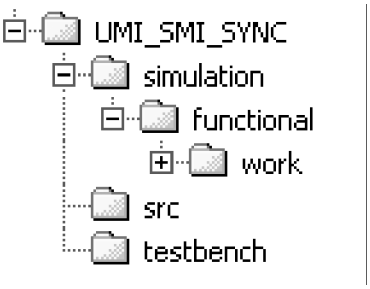
Figure 43. UMI_SMI_SYNC Design Structure



UMI_SMI_SYNC Directory Structure

Figure 44 shows the directory structure of the UMI_SMI_SYNC project.

Figure 44. UMI_SMI_SYNC Directory Structure



The "src" sub-directory contains the following design files:

- systembus.v: Verilog description of systembus. This file will be regenerated again using IPexpress.
- sc_orcastra.v: Verilog description of sc_orcastra.
- pll.v: Verilog PLL model for PLL1 and PLL2.
- SC_top.v: Verilog description of SC_top.

The test bench directory contains the following test bench files:

- JTAG_PC.v: Verilog model of JTAG_PC interface block to connect to the JTAG I/O of SC_top for JTAG accesses.
- SC_top_tb.v: Verilog model of top-level test bench.

The "simulation/functional" directory contains the following MTI ModelSim files:

- vcom_SC_top_SE.do: do file to compile/simulate design and test bench files in SE version of ModelSim
- vcom_SC_top_OEM.do: do file to compile/simulate design and test bench files in Lattice version of ModelSim
- wave.do: Waveform file called by vsim_SC_top.do
- jtag_in.txt: stimulus file for R/W transactions for the test bench JTAG_PC.v file

Generating System Bus for UMI_SMI_SYNC

To generate the system bus model, follow the following steps:

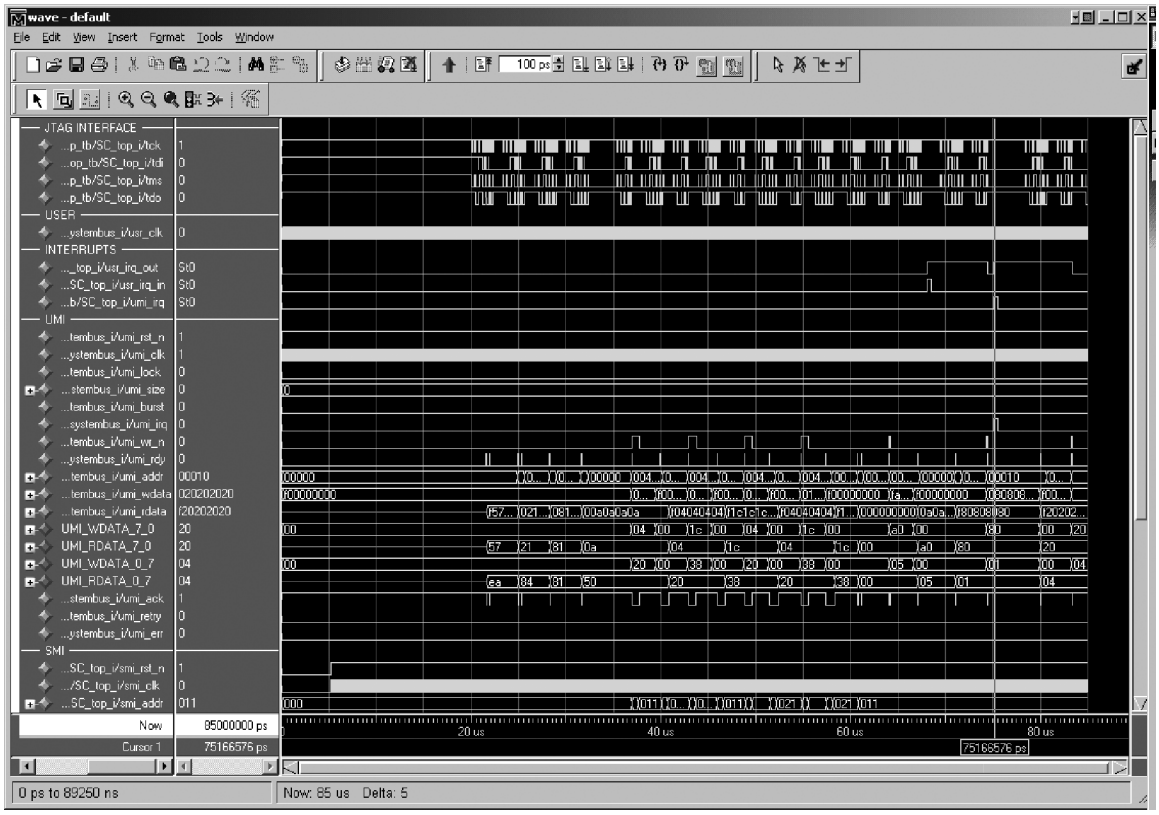
- Start a new ispLEVER Project Navigator session with a LatticeSC device.
- Launch IPexpress.
- Select Module -> Architecture_Modules -> System_Bus and set the File Name to “systembus”. Also make sure that the Design Entry is set to “Schematic/Verilog HDL”. The “Project Path” should point to “UMI_SMI_SYNC/src”.
- Click on “Customize”
- In the new window:
  - Enable the UMI Interface
  - Set the UMI to synchronous mode (UMI Sync. To Systembus).
  - Enable the SMI Interface, and turn on both 0x410 and 0x420 read interfaces.
  - Set the system bus clock source to USER.
- Click on the “Generate” button (this will re-create systembus.v in “UMI_SMI_SYNC /src”).

Running the UMI_SMI_SYNC Simulation

To run the UMI_SMI_SYNC simulation

- Start an MTI ModelSim session.
- In ModelSim, change to the “UMI_SMI_SYNC /simulation/functional” directory.
- Run the vcom_SC_top_SE.do or vcom_SE_top_OEM.do file using the Execute Macro menu option.
- Note that after running this step, you might encounter errors related to obsolete compiled libraries, which would require refreshing these libraries.
- After running the compile/simulation do file, a waveform similar to Figure 45 should appear and the simulation should run for about 85µs.

Figure 45. ModelSim Waveform for UMI_SMI_SYNC



Description of UMI_SMI_SYNC Signals in the ModelSim Waveform

The ModelSim waveform is divided into four major sections:

- JTAG signals
- Interrupt signals
- UMI signals
- The SMI, PLL1 and PLL2 section, covering the SMI signals and the PLL outputs

Description of the UMI_SMI_SYNC Simulation Scenario

The read and write accesses to the different interfaces on the system bus are initiated by the “mpu_in.txt” file. A read or write command is reflected on the PC (and UMI) Interface. The simulation can be divided into the following phases:

- 0 to 20µs: the test bench waits for the LatticeSC chip and system bus to power-up.
- 20 to 36µs: the test bench reads the system bus ID register (0x00000-0x00003).
- 36 to 48µs: PLL1 is accessed via SMI by:
  - Writing address 0x00410 to data[7:0] 0x04 (divide output by 2)
  - Reading address 0x00410
  - Writing address 0x00410 to data[7:0] 0x1C (divide output by 8)
  - Reading address 0x00410
- 48 to 60µs: PLL2 is accessed via SMI by:
  - Writing address 0x00420 to data[7:0] 0x04 (divide output by 2)
  - Reading address 0x00420

  - Writing address 0x00420 to data[7:0] 0x1C (divide output by 8)
  - Reading address 0x00420

- 60 to 68µs: The EN_IRQ_USER register (0x00012) is written with data[7:0]=0xA0, such that interrupts from both USR_IRQ_IN and UMI_IRQ are enabled to USR_IRQ_OUT.
- 68 to 75µs: this period of time shows how an interrupt from the USR_IRQ_IN can be propagated to the USR_IRQ_OUT output. During this phase:
  - A USR_IRQ_IN pulse is generated. This creates an interrupt to the system bus and asserts USR_IRQ_OUT to a '1' value.
  - The system bus interrupt cause register (0x00010) is read. The value is data[7:0]=0x80, indicating that an interrupts was received by the system bus from USR_IRQ_IN.
  - The system bus interrupt cause register (0x00010) is written with data[0:7]=0x80, to clear the interrupt bit from USR_IRQ_IN. This de-asserts USR_IRQ_OUT to a '0' value.
- 75 to 85µs: this period of time shows how an interrupt from the UMI_IRQ can be propagated to the USR_IRQ_OUT output. During this phase:
  - A UMI_IRQ pulse is generated. This creates an interrupt to the system bus and asserts USR_IRQ_OUT to a '1' value.
  - The system bus interrupt cause register (0x00010) is read. The value is data[7:0]=0x20, indicating that an interrupts was received by the system bus from UMI_IRQ.
  - The system bus interrupt cause register (0x00010) is written with data[0:7]=0x20, to clear the interrupt bit from UMI_IRQ. This de-asserts USR_IRQ_OUT to a '0' value.
  - The system bus interrupt cause register (0x00010) is read. The value is data[7:0]=0x00, indicating that no more interrupts are received by the system bus.

UMI_PCS_ASYNC

The UMI_PCS_ASYNC example consists of a design with the following blocks:

- A system bus (systembus) with UMI and PCS interfaces.
- A sc_orcastra block to interface between the UMI on the system bus and an external PC JTAG interface. The sc_orcastra block translates PC instructions to UMI format.
- A PCS block (PCS0) configured in Gigabit Ethernet mode.
- A pattern generator (cjpat_gen), which reads data content from a distributed ROM (MYROM) and sends it to the TX FPGA data interface of the PCS block.
- A PLL block (PLLDIV2) used by the cjpat_gen block.
- A top-level design (SC_top) to stitch all the lower blocks together.

For simulation purposes, a test bench is developed around SC_top with the following elements:

- A JTAG_PC interface model to connect to the JTAG I/O of SC_top for JTAG accesses
- Drivers for all clocks and resets.
- An instance of SC_top with appropriate connections to top-level signals.
- A top-level test bench (SC_top_tb) to stitch all the lower blocks together. The test bench connects the PCS0 hdout* pins back to the hdin* pins to form an external loop-back connection (TX back to RX)

The features of the example are:

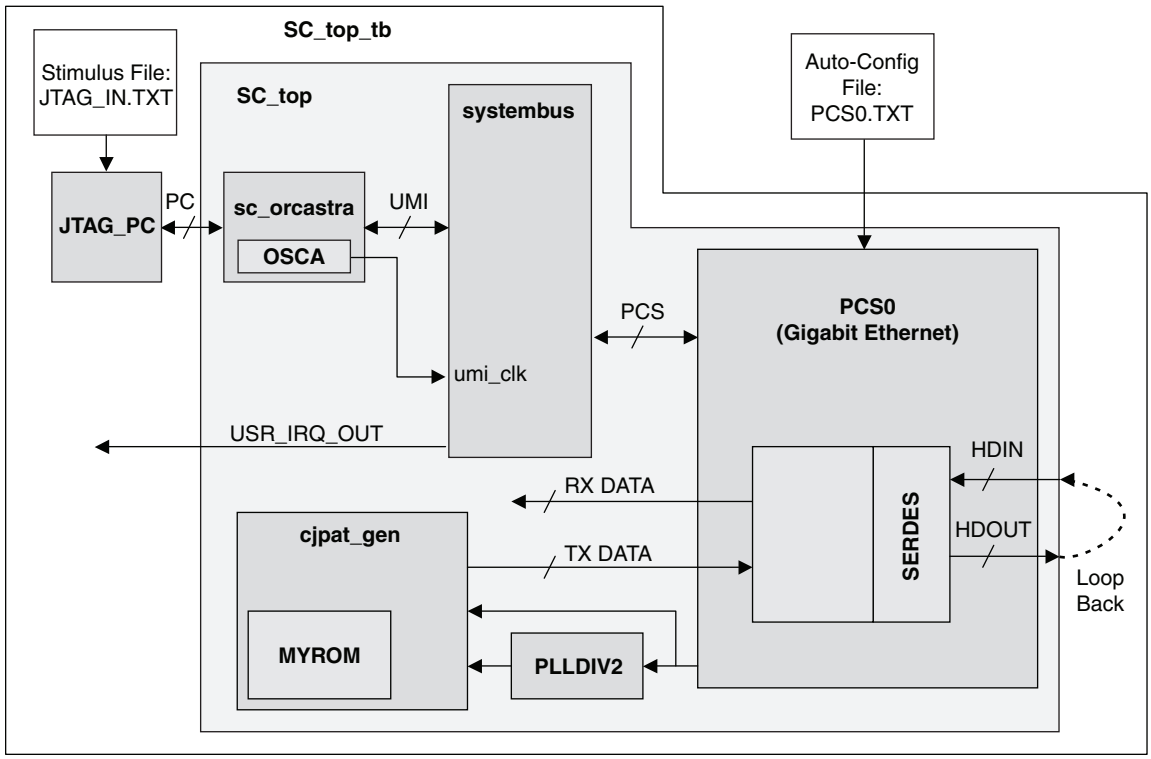
- System bus clocked by the USER clock input pin.
- OSCA oscillator in sc_orcastra module drives both UMI clock and USER clock.
- PCS is configured using an auto-configuration file (PCS0.txt). MTI ModelSim automatically loads this file during

simulation.

- PCS channel 0 generated in Gigabit Ethernet Mode.
- Demonstrates Interrupt from PCS to UMI. The interrupt is enabled when the link state machine of channel 0 reaches a synchronized state.

Figure 46 shows the design structure including both SC_top and SC_top_tb.

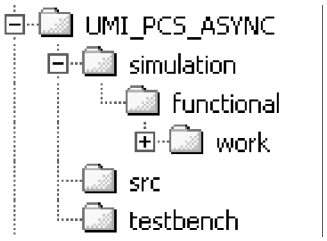
Figure 46. UMI_PCS_ASYNC Design Structure



UMI_PCS_ASYNC Directory Structure

Figure 47 shows the directory structure of the UMI_PCS_ASYNC project.

Figure 47. UMI_PCS_ASYNC Directory Structure



The “src” sub-directory contains the following design files:

- systembus.v: Verilog description of systembus. This file will be regenerated again using IPexpress.
- sc_orcastra.v: Verilog description of sc_orcastra.
- PCS0.v: Verilog description of PCS0.
- cjpat_gen.v: Verilog description of cjpat_gen.

- PLLDIV2.v: Verilog description of PLLDIV2.
- MYROM.v: Verilog description of MYROM.
- SC_top.v: Verilog description of SC_top.

The test bench directory contains the following test bench files:

- JTAG_PC.v: Verilog model of JTAG_PC interface block to connect to the JTAG I/O of SC_top for JTAG accesses.
- SC_top_tb.v: Verilog model of top-level test bench.

The “simulation/functional” directory contains the following MTI ModelSim files:

- vcom_SC_top_SE.do: do file to compile/simulate design and test bench files in SE version of ModelSim
- vcom_SC_top_OEM.do: do file to compile/simulate design and test bench files in Lattice version of ModelSim
- wave.do: Waveform file called by compile/simulation do file
- jtag_in.txt: stimulus file for R/W transactions for the test bench JTAG_PC.v file.
- PCS0.txt: Auto-configuration file. MTI ModelSim automatically loads this file during simulation.

Generating System Bus for UMI_PCS_ASYNC

To generate the system bus model, follow the following steps:

- Start a new ispLEVER Project Navigator session with a LatticeSC device.
- Launch IPexpress.
- Select Module -> Architecture_Modules -> System_Bus and set the File Name to “systembus”. Also make sure that the Design Entry is set to “Schematic/Verilog HDL”. The “Project Path” should point to “UMI_PCS_ASYNC/src”.
- Click on “Customize”
- In the new window:
  - Enable the UMI Interface
  - Set the system bus clock source to OSC.
  - Enable the PCS 360 Interface.
  - Click on the “Generate” button (this will re-create systembus.v in “UMI_PCS_ASYNC /src”).

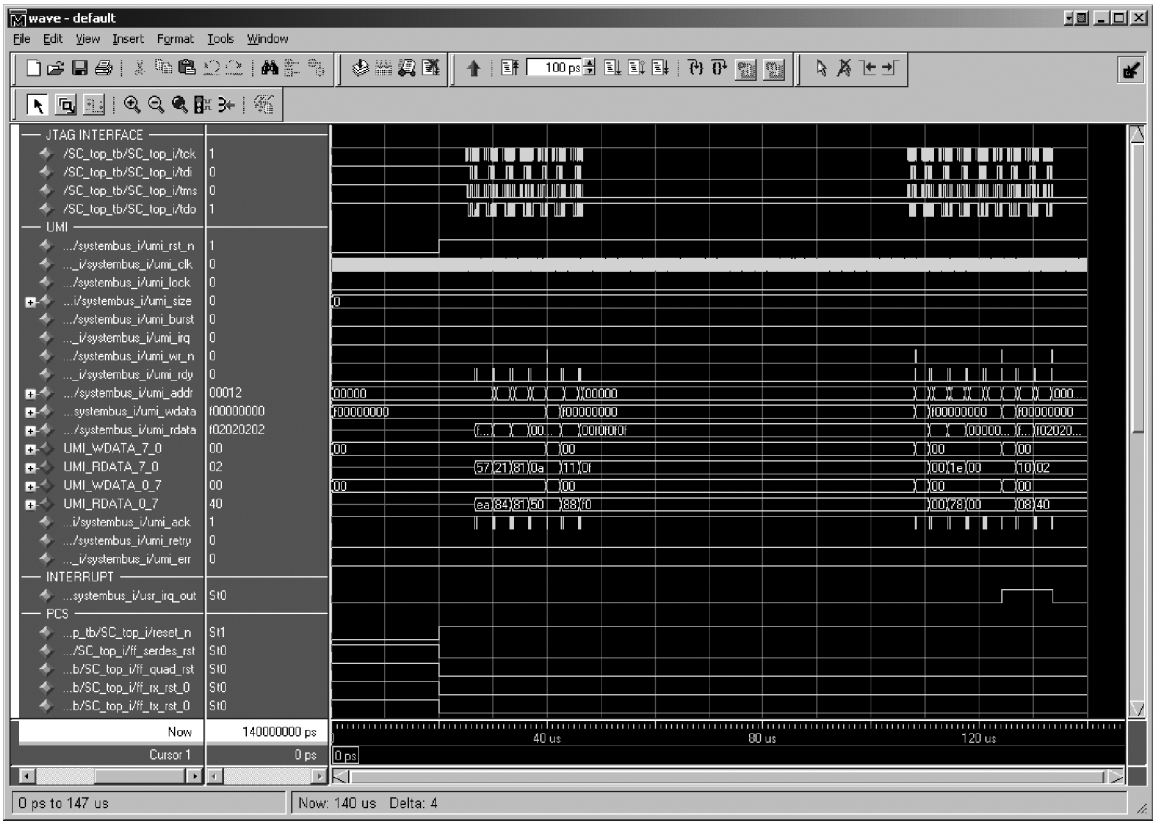
Running the UMI_PCS_ASYNC Simulation

To run the UMI_PCS_ASYNC simulation:

- Start an MTI ModelSim session.
- In ModelSim, change to the “UMI_PCS_ASYNC /simulation/functional” directory.
- Run the vcom_SC_top_SE.do or vcom_SE_top_OEM.do file using the Execute Macro menu option.
- vcom_SC_top_SE.do or vcom_SE_top_OEM.do.

- Note that after running this step, you might encounter errors related to obsolete compiled libraries, which would require refreshing these libraries.
- After running the compile/simulation do file, a waveform similar to Figure 48 should appear and the simulation should run for about 140µs.

Figure 48. ModelSim Waveform for UMI_PCS_ASYNC



Description of UMI_PCS_ASYNC Signals in the ModelSim Waveform

The ModelSim waveform is divided into 5 sections

- JTAG signals
- UMI signals
- USR_IRQ_OUT signal
- The PCS signals, including TX and RX
- The system bus PCS interface

Description of the UMI_PCS_ASYNC Simulation Scenario

At the onset of the simulation, the simulator will automatically load the PCS0.txt (containing the memory initialization sequence of PCS0). Then, system bus read and write accesses to the different interfaces on the system bus are initiated by the “pc_in.txt” file. A read or write command is first reflected on the PC (and UMI) Interface. The simulation can be divided into the following phases:

- 0: PCS0.txt configures the PCS0 block registers such that:
  - Channel 0 is in Gigabit Ethernet Mode.
  - CRC insertion is enabled for both TX and RX.

- When reset_n is removed, this enables the TX data generation from cjpat_gen to PCS0. Also, around 71µs, the PCS0 recovered data starts toggling properly. This data should be the receive equivalent of the TX data since the PCS0 block is in external loop-back. As part of the data flow into the TX direction, an error is inserted through the tx_er_0 signal (around 100µs). As a result, the error propagates to the rx_er_0 signal in the receive direction.
- 26 to 38 µs: the test bench reads the system bus ID register (0x00000-0x00003).
- 38 to 44 µs: the test bench writes data[7:0]=0x11 to system bus scratch pad register 0x00004 and reads it back.
- 45 µs: The PCS0 quad interface register 0x84 (0x36084) is read. The value is data[4:7]=0x0, indicating that the link state machine for channel 0 is not yet synchronized.
- 106 to 140 µs: this period of time shows how an interrupt from the PCS can be propagated to the USR_IRQ_OUT output. During this phase:
  - Interrupts from the PCS to the USR_IRQ_OUT output are enabled by writing data[0:7]=0x40 (data[7:0]=0x02) to address 0x00012.
  - The system bus interrupt cause register (0x00010) is read. The value is data[0:7]=0x00, indicating that no interrupts were received by the system bus.
  - The PCS0 quad interface register 0x84 (0x36084) is read. The value is data[4:7]=0x8 (data[7:4]=0x1), indicating that the link state machine for channel 0 is now synchronized.
  - The PCS0 quad interface register 0x1C (0x3601C) is read. The value is data[0:7]=0x00, indicating that the interrupt enable bit corresponding to a synchronized link state machine for channel 0 is off.
  - The PCS0 quad interface register 0x85 (0x36085) is read. The value is data[0:7]=0x00, indicating that the interrupt bit corresponding to a synchronized link state machine for channel 0 is off.
  - The PCS0 quad interface register 0x1C (0x3601C) is written to data[0:7]=0x08 (data[7:0]=0x10). to turn on the interrupt enable bit corresponding to a synchronized link state machine for channel 0. This enables the interrupt from 0x36085, bit 4, which asserts signal USR_IRQ_OUT to a ‘1’ value.
  - The PCS0 quad interface register 0x85 (0x36085) is read. The value is data[0:7]=0x08 (data[7:0]=0x10), indicating that the interrupt bit corresponding to a synchronized link state machine for channel 0 is ON.
  - The system bus interrupt cause register (0x00010) is read. The value is data[0:7]=0x40 (data[7:0]=0x02), indicating that an interrupt was received by the system bus from the PCS.
  - Interrupts from the PCS to the USR_IRQ_OUT output are disabled by writing data[0:7]=0x00 to 0x00012. This de-asserts USR_IRQ_OUT to a ‘0’ value.