LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11e)

DS449 October 19, 2011 **Product Specification**

Introduction

The LogiCORE™ IP FSL V20 Fast Simplex Link (FSL) Bus is a uni-directional point-to-point communication channel bus used to perform fast communication between any two design elements on the FPGA when implementing an interface to the FSL bus. The FSL interface is available on the Xilinx MicroBlaze™ processor. The interfaces are used to transfer data to and from the register file on the processor to hardware running on the FPGA.

Features

- Implements a uni-directional point to point FIFO-based communication
- Provides a mechanism for unshared and non-arbitrated communication. This can be used for fast transfer of data words between a master and a slave, thus implementing the FSL interface.
- Provides an extra control bit for annotating transmit data. This bit can be used by the slave-side interface for different purposes, such as decoding the transmit word as a control word, or using the bit to indicate the start or end of frame transmission.
- FIFO depths can be as low as 1K and as high as 8K.
- Supports synchronous and asynchronous FIFO modes. This allows the master and slave side of the FSL to clock at different rates.
- Support for SRL16 and dual port LUT RAM or block RAM based FIFO implementation.

LogiCORE IP Facts		
Core Specifics		
Supported Device Family	Spartan®-3, Spartan-3E, Spartan-6, Spartan-3A/3A DSP/3AN, Automotive Spartan-3/3A/3A DSP/ 3E, Virtex®-4, Virtex-5, Virtex-6	
Resource Used		
	Min	Max (1)
Slices	21	451
LUTs	3	362
FFs	36	34
Block RAMs	0	17
Provided with Core		
Documentation	Product Specification	
Design File Formats	VHDL	
Constraints File	N/A	
Verification	N/A	
Instantiation Template	N/A	
Additional Items	N/A	
Design Tool Requirements		
Xilinx Implementation Tools	ISE® 13.3	
Verification	N/A	
Simulation	Mentor Graphics ModelSim v6.6a and above	
Synthesis	XST	
Support		
Provided by Xilinx, Inc.		

1. Maximum size in slices of FFs and LUTs is obtained using parameter options, C_ASYNC_CLKS=1, C_FSL_DEPTH=128, and C_USE_CONTROL=1. Maximum block RAM size is obtained for parameter options C_ASYNC_CLKS=1, C_FSL_DWIDTH=32, C_FSL_DEPTH=3182, and C_USE_CONTROL=1.

Functional Description

The Fast Simplex Link (FSL) V20 Bus is shown in Figure 1.

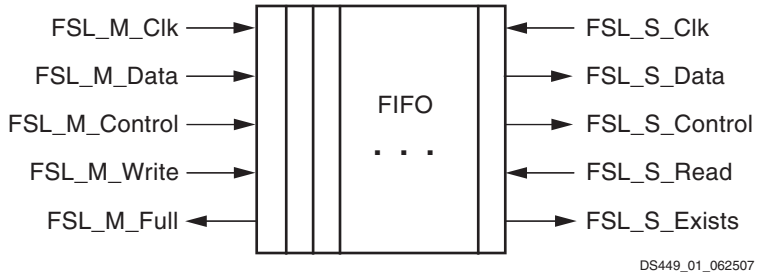

Figure 1: **FSL V20 Bus Block Diagram**

FSL V20 Bus Core I/O Signals

The I/O signals fand their function for the FSL V20 core are provided in Table 1.

Table 1: **FSL_V20 I/O Signals**

Signal Name	MSB:LSB	I/O	Description
FSL_Clk		I	Input clock to the FSL bus when used in the synchronous FIFO mode (C_ASYNC_CLKS = 0). The FSL_Clk is used as the clock for both the master and slave interfaces
SYS_Rst		I	External system reset
FSL_Rst		O	Output reset signal generated by the FSL reset logic. Any peripherals connected to the FSL bus can use this reset signal to operate the peripheral reset.
FSL_M_Clk		I	Port that provides the input clock to the master interface of the FSL bus when used in the asynchronous FIFO mode (C_ASYNC_CLKS = 1). All transactions on the master interface use this clock when implemented in the asynchronous mode
FSL_M_Data	0:C_FSL_DWIDTH-1	I	Data input to the master interface of the FSL bus
FSL_M_Control		I	Single bit control signal that is propagated along with the data at every clock edge. Transmission of the control bit occurs when C_USE_CONTROL is set to 1 (default). If the control bit is not used by the slave, C_USE_CONTROL can be set to 0 to save area
FSL_M_Write		I	Input signal that controls the write enable signal of the master interface of the FIFO. When set to 1, the values of FSL_M_Data and FSL_M_Control (if C_USE_CONTROL = 1) are pushed into the FIFO on a rising clock edge. Note FSL_M_Write is not gated with FSL_M_Full. The state of the FIFO will be undefined when writing to a full FIFO.
FSL_M_Full		O	Output signal on the master interface of the FIFO indicating that the FIFO is full. This signal can be used for hand-shaking and synchronization between the master and slave connected through an FSL bus
FSL_S_Clk		I	Port that provides the input clock to the slave interface on the FSL bus when used in the asynchronous FIFO mode (C_ASYNC_CLKS = 1). All transactions on the slave interface use this clock when implemented in the asynchronous mode
FSL_S_Data	0:C_FSL_DWIDTH-1	O	Data output bus onto the slave interface of the FSL bus

Table 1: **FSL_V20 I/O Signals *(Cont'd)***

Signal Name	MSB:LSB	I/O	Description
FSL_S_Control		O	Single bit control that is propagated along with the data at every clock edge by the FSL bus when C_USE_CONTROL is set to 1
FSL_S_Read		I	Input signal on the slave interface that controls the read acknowledge signal of the FIFO. When set to 1, the values of FSL_S_Data and FSL_S_Control are popped from the FIFO on a rising clock edge. Note FSL_S_Read is not gated with FSL_S_Exists. Data read is undefined when reading an empty FIFO.
FSL_S_Exists		O	Output signal on the slave interface indicating that FIFO contains valid data. This signal can be used by the FSL slave peripheral for hand-shaking and synchronization with the FSL master peripheral
FSL_Has_Data		O	Indicates FSL buffer has data
FSL_Full		O	Indicates FSL buffer is full
FSL_Control _IRQ		O	Is asserted when FSL_S_Control and (FSL_Has_Data or FSL_S_Exists) are asserted

FSL V20 Core Parameters

The parameterizable features in the FSL V20 FPGA design are shown in Table 2.

Table 2: **FSL_V20 Parameters**

Feature/Description	Parameter Name	Allowable Values	Default Value	VHDL Type
Specify clocking modes of FIFO as synchronous or asynchronous	C_ASYNC_CLKS	0, 1	0	integer
Use BRAMs to implement FIFO	C_IMPL_STYLE	0, 1	0	integer
FSL bus width	C_FSL_DWIDTH	>0	32	integer
FSL FIFO depth C_ASYNC_CLKS=0 C_ASYNC_CLKS=1 and C_IMPL_STYLE=0 C_ASYNC_CLKS=1 and C_IMPL_STYLE=1	C_FSL_DEPTH	 1- 8192(1) (2) 16-128(3) 512-8192(3)	 16 16 512	integer
Propagate control bit	C_USE_CONTROL	0, 1	1	integer
Level of external reset	C_EXT_RESET_HIGH	0 = Low-true external reset 1 = High-true external reset	1	integer
Period of FSL_S_CLK in ps.	C_READ_CLOCK_PERIOD	>0	0	integer

1. C_FSL_DEPTH in the range 2–15 result in a FIFO depth of 16 when C_ASYNC_CLKS=0.
2. C_FSL_DEPTH can have any value higher than 16
3. C_FSL_DEPTH must be a 2^n value

Parameter Descriptions

C_FSL_DEPTH

Specifies the depth of the FIFO implemented by the FSL bus. The depth can be as low as 1 or as high as 8192. The depth specified is dependent on the implementation scheme of the FIFO. When the parameter C_ASYNC_CLKS is set to 0, the depth allowed is between 1 and 8192 (2–15 will result in a FIFO depth of 16). When the parameter C_ASYNC_CLKS is set to 1 and C_IMPL_STYLE is set to 0 (LUT RAM), the depth allowed is between 16 and 128. When the parameter C_ASYNC_CLKS is set to 1 and C_IMPL_STYLE is set to 1 (block RAM), the depth allowed is between 512–8192. The depth must be 2^n when C_ASYNC_CLKS is set to 1, but can have any allowed value when C_ASYNC_CLKS is set to 0.

C_USE_CONTROL

This parameter specifies whether the control bit is propagated along with the data bit. When set to 1, the control bit is transmitted from the master to the slave interface. When set to 0, the control bit transmitted to the slave is 0. When propagation of control bit is not required, setting this bit to 0 enables reduction in the area of the FSL bus.

C_ASYNC_CLKS

This parameter specifies whether the FIFO in the FSL bus is implemented as a synchronous or asynchronous FIFO. When set to 1, the FSL implements an asynchronous FIFO. In this case, the clock ports FSL_M_Clk and FSL_S_Clk are used as the master and slave clocks, respectively. If set to 0, the FSL is implemented as a synchronous FIFO. In this case, the clock port FSL_Clk is used for both the master and slave interfaces.

C_IMPL_STYLE

This parameter specifies the style of implementation of the FIFO of the FSL. If set to 1, the FIFO is implemented using block RAMs. If set to 0, the FIFO is implemented using LUT RAMs.

Note: This parameter affects timing. *When C_IMPL_STYLE=1, there is a one-cycle fall-through latency from a write to an empty FIFO before FSL_S_Exists goes High.*

If C_IMPL_STYLE = 0 and C_ASYNC_CLKS = 1 (using asynchronous LUT RAM FIFO), implementation tools do not normally include the asynchronous set and reset path through flip-flops (DFFs). This can result in under-constrained designs when using high clock frequencies. To ensure that the tools include these paths in the timing analysis, the constraint "ENABLE=sr_reg_o;" must be added to the top-level constraints file (UCF). This constraint includes an all asynchronous path using asynchronous set/reset to DFFs for the whole design. However, this could create false critical paths for other parts of the design that need to be handled.

C_READ_CLOCK_PERIOD

When the parameter C_ASYNC_CLKS is set to 1 and C_IMP_STYLE = 0, the parameter C_READ_CLOCK_PERIOD must also be set. This parameter defines the period of FSL_S_CLK and is used for generating timing constraints for the asynchronous path through LUT RAMs

Parameter - Port Dependencies

When the parameter C_ASYNC_CLKS is set to C_ASYNC_CLKS = 0, the asynchronous write and read clock ports, FSL_M_Clk and FSL_S_Clk, are not used in the design. The synchronous clock FSL_Clk is used as the clock port. When C_ASYNC_CLKS = 1, the synchronous clock port, FSL_Clk is not used. The asynchronous read and write clock ports, FSL_S_Clk and FSL_M_Clk, are used.

Reset Descriptions

After the external reset (SYS_Rst) has been deasserted, the FSL_V20 logic stays in a reset state for an additional 17 clock cycles. Exit of the reset state will coincide with the deassertion of the FSL_Rst output signal.

Register Descriptions

Not applicable.

Interrupt Descriptions

The signals FSL_Has_Data, FSL_Full and FSL_Control_IRQ have interrupt properties that make them easily connected to an interrupt controller, such as when asserted an interrupt is generated.

Bus Operation

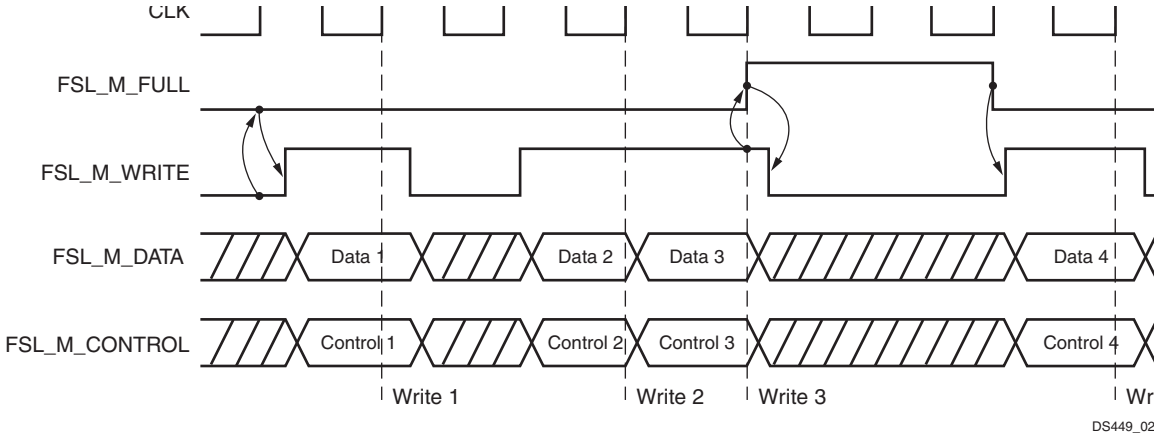

Figure 2: **FSL Write Operation**

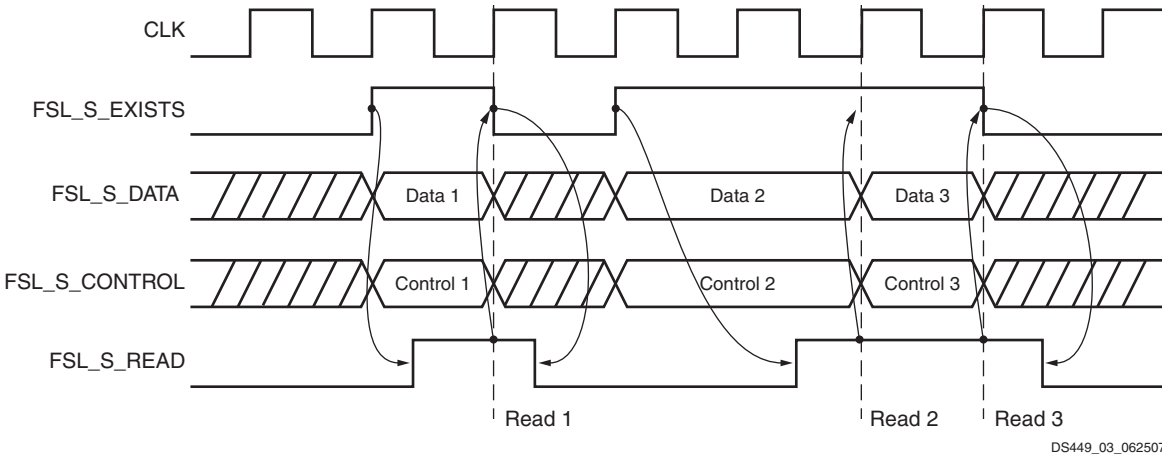

Figure 3: **FSL Read Operation**

Write Operation

The write to the FSL bus is controlled by the `FSL_M_Write` signal. The following sequence of operations indicate a write operation on the FSL bus. When the data in `FSL_M_Data` and control bit in `FSL_M_Control` are ready to be pushed into the FIFO, the `FSL_M_Write` signal is set to 1 for one clock cycle. This pushes the Data and Control signals onto the FIFO. If the FIFO is not implemented with block RAMs, the data becomes available to the slave FSL interface as `FSL_S_Data` and the control becomes available as `FSL_S_Control` after the write clock edge. There is a once-cycle delya if using block RAMs when the FIFO length is zero. Further, the `FSL_S_Exists` signal is set to 1 to indicate that data exists in the FIFO.

The timing diagram in Figure 2 shows four write operations on the FSL bus. At the first clock edge the master checks the `FSL_M_Full` signal and sees that it is not set. This allows the master to set the `FSL_M_Write` signal and put the `FSL_M_Data` and `FSL_M_Control` on the bus. At the next clock edge, the data is read by the bus and transferred into the FIFO. Writes 2 and 3 show back-to-back write operations. At write 3 the FIFO is full, which sets `FSL_M_Full`. This forces the master to drop `FSL_M_Write`. After a read takes place, `FSL_M_Full` goes low and the master can issue another write. A read also takes place at write 4; otherwise, the `FSL_M_Full` would have gone high again.

The `FSL_M_Write` is not gated with `FSL_M_Full`; the state of the FIFO is undefined when writing to a full FIFO.

Read Operation

The read side of the FSL bus is controlled by the `FSL_S_Read` signal. The following sequence indicates a read operation on the FSL bus. When data is available in the FSL bus (`FSL_S_Exists` = 1), the data in `FSL_S_Data` and the control bit in `FSL_S_Control` is immediately available to be read by the slave on the FSL bus. After the slave completes the read operation, `FSL_S_Read` has to be set to 1 for one clock cycle acknowledging that a Read has successfully been completed by the slave. After the clock edge where the read takes place, the `FSL_S_Data` and `FSL_S_Control` are updated with new data, and `FSL_S_Exists` and `FSL_M_Full` are updated. Figure 3 illustrates the timing diagram for three read operations from the slave side of the FSL bus. Two writes take place between read 1 and read 2.

The `FSL_S_Read` is not gated with `FSL_S_Exists`; data read is undefined when reading an empty FIFO.

Bus Usage

FSL Peripheral Interconnect Mechanism

FSL peripherals can be created as a master or a slave to the FSL bus.

A peripheral connected to the master ports of the FSL bus pushes data and control signals onto the FSL. All peripherals that act as a master to the FSL bus should create a bus interface of the type MASTER for the bus standard FSL in the Microprocessor Peripheral Description (MPD) file. Further, the peripheral must form default connections to all master ports of the FSL bus and must follow the FSL bus write operation timing requirements as specified in Figure 2.

A peripheral connected to the slave ports of the FSL bus reads and pops data and control signals from the FSL. All peripherals that are a slave to the FSL bus should create a bus interface of the type SLAVE for the bus standard FSL in the MPD file. Further, the peripheral must form default connections to all slave port of the FSL bus and must follow the FSL bus read operation timing requirements as shown in Figure 3.

An example MPD of a simple peripheral having a master interface `FSL_OUT` and a slave interface `FSL_IN` follows.

```
BEGIN my_fsl_peripheral
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
BUS_INTERFACE BUS = FSL_IN,  BUS_STD = FSL, BUS_TYPE = SLAVE
BUS_INTERFACE BUS = FSL_OUT, BUS_STD = FSL, BUS_TYPE = MASTER

## Ports
PORT CLK = "", DIR = IN, SIGIS=CLK
PORT RESET = "", DIR = IN
PORT FSL_S_READ    = FSL_S_Read,    DIR=out, BUS=FSL_IN
PORT FSL_S_DATA    = FSL_S_Data,    DIR=in,  VEC=[0:31], BUS=FSL_IN
PORT FSL_S_CONTROL = FSL_S_Control, DIR=in,  BUS=FSL_IN
PORT FSL_S_EXISTS  = FSL_S_Exists,  DIR=in,  BUS=FSL_IN
PORT FSL_M_WRITE   = FSL_M_Write,   DIR=out, BUS=FSL_OUT
PORT FSL_M_DATA    = FSL_M_Data,    DIR=out, VEC=[0:31], BUS=FSL_OUT
PORT FSL_M_CONTROL = FSL_M_Control, DIR=out, BUS=FSL_OUT
PORT FSL_M_FULL    = FSL_M_Full,    DIR=in,  BUS=FSL_OUT

END
```

MicroBlaze Soft Processor FSL Interfaces

The put and get instructions of the MicroBlaze processor can be used to transfer the contents of a MicroBlaze processor register onto the FSL bus and vice-versa. The FSL bus configuration of the MicroBlaze processor can be used in conjunction with any of the other bus configurations. See *UG081, MicroBlaze Processor Reference Guide* for more information about using FSL from the MicroBlaze processor.

Design Implementation

Design Tools

The FSL V20 design is implemented in VHDL. XST is the synthesis tool used for synthesizing the FSL V20 design. The NGC netlist output generated by XST is then input to the Xilinx ISE tool suite for FPGA implementation.

Target Technology

The target technology is any of the FPGA families listed in the Supported Device Family field of the LogiCORE IP Facts table on the first page of this data sheet.

Device Utilization and Performance Benchmarks

The resources used by the Fast Simplex Link (FSL) Bus change based upon the architecture and the parameter configuration chosen for the bus. The number of slices used ranges between 21 for the smallest configuration, and up to 451 for the largest configuration. The number of block RAMs used depends on the width, depth, and implementation style selected to implement the bus.

Specification Exceptions

Not applicable.

Reference Documents

1. *UG081, MicroBlaze Processor Reference Guide*

Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
5/20/04	1.0	Initial Xilinx release.
8/05/04	1.1	Reformat of document and corrections in Table 2
8/17/04	1.2	Updated for Gmm; reviewed/corrected trademarks and supported device family list
4/4/05	1.3	Updated for EDK 7.1.1 SP1; updated trademarks; made minor edits.
8/03/05	1.4	Converted to new DS template; updated and re-imported images.
10/1805	1.4.1	Updated to incorporate CR205787; updated cross references.

Date	Version	Revision
11/11/05	1.5	In Allowable Parameters Combinations: In C_FSL_DEPTH: *C_ASYNC_CLKS is set to 0*, was *C_SYNC_CLKS is set to 1*; In *C_IMPL_STYLE*, last sentence was: *This parameter affects timing: When the FiFO is empty, there is a one cycle delay before FSL_S_Exists goes high*.
07/12/06	1.6.1	Minor correction on page 7; Incorporated CR232700
02/11/06	1.7	CR419924; Asynchronous mode together with BRAM implementation of FIFO now supported; CR415462 and CR304088, For Asynchronous mode, C_ASYNC_CLKS=1, FIFO_DEPTH < 16 is not allowed; added Spartan-3A and Virtex-5 FPGAs to supported device listing; FSL_Control_IRQ signal: added FSL_Control_IRQ, FSL_Has_Data and FSL_Full possible to automatically connect to interrupt controller; CR426522: Write Operation when FULL Flag asserted; CR426524: Reset Descriptions section added
05/23/07	1.8	CR432809: Ambiguity in what happens for FSL_S_Read when FSL_S_EXISTS is low and for FSL_M_WRITE when FSL_S_FULL is high; CR432811: Allowable values for C_FSL_WIDTH; CR432812: FSL_Full is specified as an input instead of output; CR433775: Async FSL pcore is causing timing violations between FSL pcore and the Microblaze processor; CR438690: FSL v2.10.a can't handle back to back read, fixed; CR439091: FSL v2.10.a assigns exists flag too early, fixed; MicroBlaze Soft Processor FSL Interface section updated; supported Target Technology updated
6/25/07	1.9	Changed FSL_S_EXISTS signal direction in Figure 1; updated legal footer.
6/24/09	2.0	Updated for 11.2.
4/19/10	2.1	Created version 2.11c for 12.1 release; incorporated CR513242 by adding *C_FSL_DEPTH in the range 2-15 will result in a FIFO depth of 16 when C_ASYNC_CLKS=0* as table note 1 in Table 2; incorporated CR524441 with minor edits.
3/1/11	2.2	Created version 2.11d for 13.1 release; clearified C_FSL_DEPTH values for different configurations, added table note 2 and 3 in Table 2
6/22/11	2.3	Fixed pointer issue for large synchronous FIFOs
10/19/11	2.4	Added explanation of why "ENABLE=sr_reg_o" needs to be added for asynchronous FIFO using LUTRAM Updated notice of disclaimer Xiinx tools support for 13.3 release

Notice of Disclaimer