

Implementing Low-Cost DDR2 Interfaces with Spartan-3A FPGAs

Xilinx provides complete memory interface solutions to help you get to market faster.

by Adrian Cosoroaba
Marketing Manager
Xilinx, Inc.
adrian.cosoroaba@xilinx.com

Applications can generally be classified into two categories: high performance, where getting the highest bandwidth is paramount; and low cost, where the cost of the system is more important. For high-end applications, Xilinx offers Virtex™-5 FPGAs, which are capable of meeting the highest bandwidth requirements.

However, not all systems are pushing memory performance limits. DDR SDRAMs and DDR2 SDRAMs, with data rates running below 400 Mbps per pin, are adequate for most low-cost systems. For these applications, Xilinx offers the Spartan™-3 Generation, comprising Spartan-3, Spartan-3E, Spartan-3A, and Spartan-3AN FPGAs.

In this article, I'll explain the architecture of a DDR2 SDRAM memory interface implementation with Spartan-3 Generation FPGAs and how Xilinx® tools and hardware-verified reference designs can help you build your own memory interface.

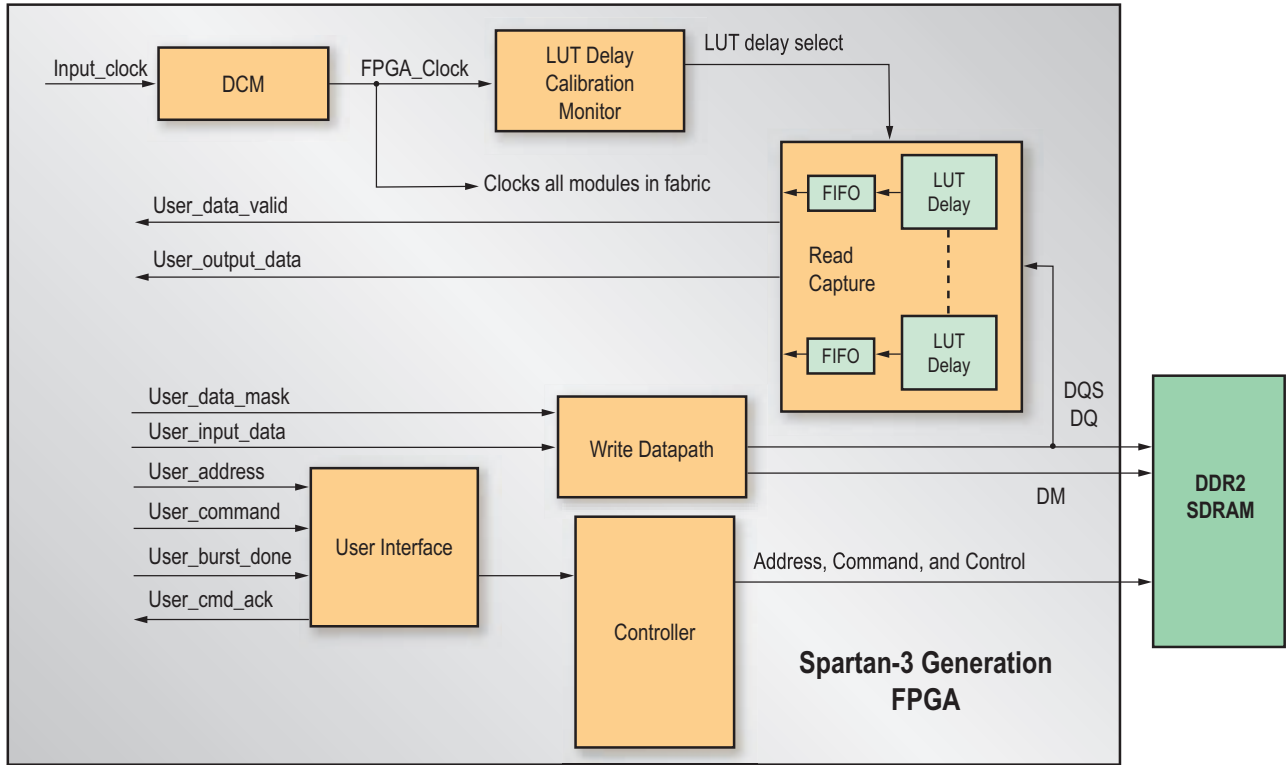


Figure 1 – DDR 2 SDRAM interface implementation for Spartan-3 Generation FPGAs

Memory Controller and DDR2 Interface

Three fundamental building blocks comprise a controller and memory interface for an FPGA-based design: the read and write data interface; the memory controller state machine; and the user interface, which bridges the memory interface design to the rest of the FPGA design (Figure 1).

In Spartan-3 Generation FPGAs, the user interface is a handshaking-type interface. When you send a command (read or write) along with the address and data for write, the user-interface logic responds with the “user_cmd-ack” signal when the next command can follow.

The functional blocks implemented in the fabric are clocked by the output of the digital clock manager (DCM), which also drives the LUT (look-up table) delay calibration monitor. This calibration circuit is used to select the number of LUT-based elements required to delay the read data strobe (DQS) with respect to read data (DQ), in order to properly align it for capture inside the FPGA.

During a read transaction, the DDR2 SDRAM device sends the DQS and associ-

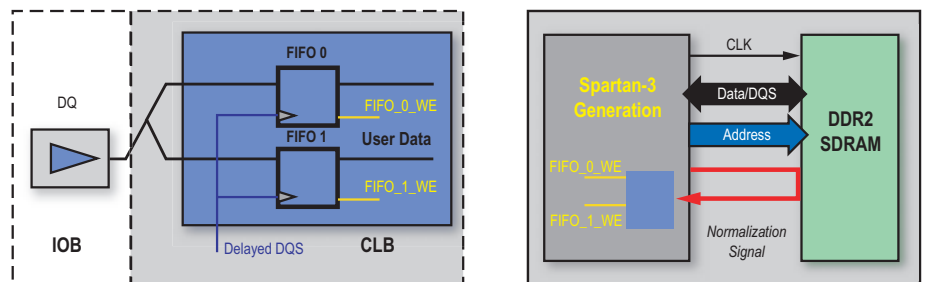


Figure 2 – DQ capture FIFO implementation for Spartan-3 Generation FPGAs

ated data to the FPGA edge-aligned with the DQ. Capturing the DQ is a challenging task, because data changes at every edge of the non-free-running DQS strobe.

DQ capture is implemented using the LUTs in the configurable logic blocks (CLBs). The DQ capture implementation uses a tap-delay mechanism based on LUTs. The DQS clocking signal is delayed to provide enough of a timing margin. The capture of DQ is implemented in dual-port LUT-based RAM (Figure 2). The LUT RAM is configured as a pair of FIFOs; each data bit is input into the rising-edge (FIFO_0) and

falling-edge (FIFO_1) FIFOs. These 16-entry-deep FIFOs are asynchronous, with independent read and write ports.

The transfer of DQ from the DQS clock domain to the memory controller clock domain occurs through these asynchronous FIFOs. Data can be simultaneously read out of both FIFO_0 and FIFO_1 in the memory controller clock domain. FIFO read pointers are generated in the FPGA internal clock domain. The generation of write enable signals (FIFO_0 WE and FIFO1_WE) occurs using the DQS and an external loop-back or normalization signal.

The external normalization signal is driven to an input/output block (IOB) as an output; it is then taken as an input through the input buffer. This technique compensates for the IOB, device, and trace delays between the FPGA and the memory device.

The write data interface generates and controls write data commands and timings. The write data interface uses IOB flip-flops and the DCM's 90-, 180-, and 270-degree outputs to transmit a DQS strobe properly aligned to the command and data bits, per DDR and DDR2 SDRAM timing requirements.

There are other aspects of the design, including the overall controller state machine logic generation and user interface. To make the complete solution easier for designers, Xilinx developed the Memory Interface Generator (MIG) tool.

Controller Design and Integration with the MIG Software Tool

Integrating all of your building blocks including the memory controller state machine is essential for the completeness of your design. Controller state machines vary with memory architecture and system parameters. State machine code can also be complicated – and a function of many variables – such as:

- Architecture (DDR, DDR2)
- Number of banks (external or internal to the memory device)
- Data bus width
- Memory device width and depth
- Bank and row access algorithms

Finally, parameters like data-to-strobe ratios (DQ/DQS) can add further complexity to the design. The controller state machine must issue the commands in the correct order while considering the timing requirements of the memory device.

The complete design can be generated with the MIG software tool, freely available from Xilinx as part of the ISE™ software CORE Generator™ suite of reference designs and IP. The MIG design flow is very similar to the traditional FPGA design flow. The benefit for design-

ers is that with this software tool, there is no need to generate RTL code from scratch for the physical layer interface or the memory controller.

To set system and memory parameters, you use the MIG's graphical user interface (GUI). For example, after selecting the FPGA device, package, and speed grade, you can select the memory architecture and even pick the actual memory device or DIMM (dual inline memory module). This same GUI provides a selection of bus widths and clock frequencies. Other options provide control of the CAS (column access strobe) latency, burst length, and pin assignments.

In less than a minute, the MIG tool generates RTL and UCF files (the HDL code and constraints files, respectively). These files are generated using a library of hardware-verified reference designs, with modifications based on your inputs. The

The final stage of the design is to import the MIG files in the ISE project, merge them with rest of your FPGA design files (followed by synthesis and place and route), run additional timing simulations if necessary, and perform hardware verification. The MIG software tool generates a batch file with the appropriate synthesis, map, and place and route options to help you optimally generate the final bit file.

Hardware Verification and Development Boards

Hardware verification of reference designs is an important final step to ensure a robust and reliable solution.

Xilinx has fully verified the implementation of a DDR2 SDRAM memory interface for Spartan-3A FPGAs in hardware. We implemented a DDR2 SDRAM design using a low-cost Spartan-3A starter kit board. Our design uses the on-board 16-

Xilinx FPGA	Spartan-3 FPGAs	Spartan-3E FPGAs	Spartan-3A FPGAs
Development Board	SL361	Starter Kit 3E	Starter Kit 3A
Memory Interfaces Supported	DDR	DDR	DDR2

Table 1 – Low-cost development boards for memory interfaces

output files are categorized in modules that apply to different building blocks of the design, such as user interface, physical layer, and controller state machine.

You also have complete flexibility to further modify the RTL code. Unlike other solutions that offer “black-box” implementations, the code is not encrypted, providing complete flexibility to change and further customize the design. After any optional code changes, you can perform additional simulations to verify the functionality of your overall design.

The MIG tool also generates a synthesizable test bench with memory checker capability. The test bench is a design example used in the functional simulation and hardware verification of the Xilinx reference design. The test bench issues a series of writes and read-backs to the memory controller. You can also use it as a template to generate your own custom test bench.

bit-wide DDR2 SDRAM memory device and the XC3S700A-FG484 FPGA. The reference design uses only a small portion of the Spartan-3A device's available resources: 13% of IOBs, 9% of logic slices, 16% of BUFG MUXs, and one of the eight DCMs. This implementation leaves plenty of resources for other functions.

Xilinx has verified memory interface designs for various Spartan-3 Generation FPGAs. Table 1 shows hardware-verified memory interfaces for each of the Spartan-3 Generation FPGA development boards.

Conclusion

You can speed up your memory interface and controller designs with low-cost Spartan-3 Generation FPGAs, the Memory Interface Generator (MIG) tool, and Xilinx development boards.

For more information and details on memory interface solutions, visit www.xilinx.com/memory.