How to Add Features and Fix Bugs - *Remotely*

Here's what you need to consider when designing a Xilinx Online application.

by Tom Branca, Applications Engineer, Xilinx, tomb@xilinx.com

Having the ability to remotely update hardware with new features or the latest bug fix can accelerate your time-to-market, extend the useful life of existing systems, and significantly cut production, maintenance, and support costs. If you plan for remote updates during your initial specification and design process, your systems can easily reap all the benefits of the Xilinx Online capability.

The Remote Field Upgrade Process

FPGAs are SRAM-based, so you can reconfigure them an unlimited number of times. To use this capability for field upgrades, you must include a mechanism for updating the configuration bitstream. To support remote field updates a system must have some sort of communication channel across which a replacement bitstream can be transferred; this could be a cable or modem connection; or a satellite, infrared or radio interface.

To initiate a remote field update, a command is sent across the communication interface to the communication processor, signaling to the system that the FPGA needs to be updated. The communication processor can be as complex as a microprocessor, or as simple as a CPLD.

Once the communication processor knows that an update is required, it can reconfigure the FPGA directly using the slave, Boundary-Scan, or peripheral FPGA configuration modes. Alternatively the processor can update a non-volatile memory bank (typically an EEPROM or Flash memory) and then simply initiate a standard FPGA reconfiguration cycle.

Planning for Remote Upgrade

Once you have made the decision to take advantage of remote field updates, a number of issues must be addressed.

Data Transmission

The type of communication channel will affect the speed, security, and integrity of the data that is used to update the FPGA. A communication interface already being used for sending and receiving data in the system (for normal operation or for firmware type updates) can usually be reused to perform the FPGA remote update.

Data Integrity and Verification

It's important that you verify the integrity and reliability of the update data before the FPGA configuration process even begins. Xilinx FPGAs have a cyclic redundancy check (CRC) built into each frame of the configuration data so that an error in the bitstream will cause the FPGA configuration to fail. You should design the system to be able to detect transmission errors, and request a re-send of the data, if necessary.

Security

If FPGA update information is sent over an unsecured network, design security may be an issue. However, it is practically impossible to decipher a configuration bitstream, to extract

(Continued)

information on the functionality of a design or make intelligent modifications to it. Xilinx keeps the specifications of the bitstream a closely guarded secret. If you feel the need for an additional level of security to keep your update data confidential, you can also use encryption.

Compression

As FPGA densities increase, the amount of data required to configure a device increases, and for larger designs compression can be beneficial. This would require some kind of additional software or hardware support to manage the data compression and extraction.

Planning for Adding New Features and Bug Fixes

Designing your system so that it can implement the current functionality and still be flexible enough to meet the requirements for future design revisions requires some advanced planning.

Choosing the Optimal FPGA Density

When you choose an FPGA for a remotely updateable application you should consider future expandability and compatibility. To determine the optimal FPGA density, you must consider the device resource requirements of the current design and that of any potential future design enhancements. An advantage of Xilinx FPGAs is that for any device in a given architecture, in the same package, all device sizes are footprint compatible. This gives you the ability to select and work with a specific device and still have the flexibility to easily change to a larger or smaller device before going to production.

Configuration Memory

You can choose either volatile or non-volatile memory for storing FPGA configuration data. The advantage of non-volatile (EEPROM or Flash) memory is that when reconfiguring the FPGA after a power cycle, the system will still have the most recent configuration data in memory and will not require extra clock cycles to update its configuration data across the communication interface.

Most FPGA configuration modes require the entire bitstream to be loaded into the FPGA during the configuration cycle. Some Xilinx FPGAs (including the Virtex family) allow partial configuration. The Xilinx data book has more information on the different configuration modes available for specific device architectures.

Design Expandability

If additional connections between the FPGA and other devices are required in future revisions, then defining the interface between devices will need to be done during the initial product revision. You can create test programs that toggle these future I/O connections, so you can test specific interfaces, without actually completing the design. Or, you can perform an EXTEST using the FPGA's built-in Boundary-Scan functionality.

Archiving the Design for Future Updates

It is critical to not only archive all source, implementation, and constraint files that were used to create the existing revision of the design, but to also document the entire flow used to create the final bitstream. Also, depending on the expected life of the application, it may be a good idea to archive the tools (save the CDROM) used in the creation of the bitstream. This will make it much easier to update the devices in the field, even years from now.

Conclusion

With a little planning, your next design can easily have the ability to be remotely fixed or upgraded, a feature that your customers will find very appealing.