Cover Story

Creating FIELD UPGRADABLE Hardware Systems Using Enabling Technology from GoAhead Software

A complete system for managing and remotely upgrading your FPGA designs in the field.

by Greg Brown, Xilinx Software Product Manager for Internet Applications, greg.brown@xilinx.com
and by Mike Akers, GoAhead Software FieldUpgrader Product Manager, mikea@goahead.com

For many years designers have been using Xilinx programmable logic devices (PLDs) to reap the benefits of fast time-to-market. Now, a second major advantage is made possible through the partnership of GoAhead Software and Xilinx—the ability to upgrade your designs via a network after they have been deployed in the field. This can result in dramatic field service cost savings and it helps "future-proof" your product.

The Solution for Field Upgrading

The new Xilinx and GoAhead Software solution combines the GoAhead commercial software (GoAhead FieldUpgrader™) with the Xilinx Internet Reconfigurable Logic methodology (IRL™) to provide the enabling technology to help you create and manage field upgradable systems. This technology provides the backbone for network-based system designs as well as the delivery mechanisms to successfully deploy upgradable products.

GoAhead FieldUpgrader

The GoAhead FieldUpgrader software consists of three parts:

- **GoAhead DeviceStudio™** - a development environment used to create the UpgradeAgent software.
- **GoAhead UpgradeAgent™** - the software that is embedded in the target device. Each agent is unique to the particular operating system that is resident on the target system. The target system containing the UpgradeAgent polls the UpgradeServer at regular pre-determined intervals to look for FPGA upgrades. When an upgrade is available, the device downloads the upgrade to a predetermined storage location. This process is illustrated in Figure 1.
- **GoAhead UpgradeServer™** - used to create and publish upgrades and is usually located on a server at the manufacturer's site. When the need for an FPGA upgrade arises, the manufacturer publishes the upgrade with UpgradeServer.

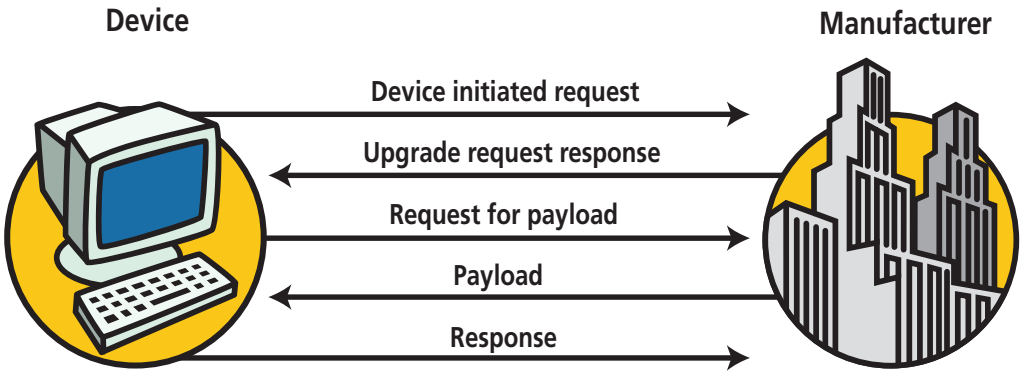

Figure 1 - The remote upgrade process.

Data Security and Integrity

To upgrade devices over a network, it is critical that the devices only accept upgrades from the authorized, designated server. It is also important to ensure that the upgrade payload is not modified or corrupted. GoAhead FieldUpgrader and the UpgradeServer use a variety of techniques to authenticate the upgrade and prevent modifications along the way.

GoAhead FieldUpgrader uses the Digital Signature Standard (DSS), which specifies a Digital Signature Algorithm (DSA). The DSA provides the capability to generate and verify signatures using public and private keys. In addition, the device-initiated approach provides built-in security; the device does not accept externally initiated upgrades. The location and port of the upgrade server is configured into the target device and the device will only allow upgrades from this designated upgrade server. This method ensures that the upgrade payload received at the target device matches the original upgrade payload published at the upgrade server and guarantees that the message was not modified en-route to the device.

The payload of the upgrade is broken into smaller chunks to make the transfer more efficient and re-startable. GoAhead FieldUpgrader calculates a 32-bit CRC checksum for each of these individual chunks. If the target device detects a checksum error, it discards that chunk and re-fetches it from the upgrade server. Once all of the chunks have been downloaded, the device reconstructs the payload and performs the higher-level data integrity check.

Data integrity is also checked during the actual programming of the Xilinx FPGAs; there is a CRC checksum built into the programming step.

Fault Tolerance

The UpgradeAgent is fault tolerant during the transmission of the upgrade. The Agent knows the contents of the payload based on the manifest list and will re-initiate the upgrade to fetch any missing chunks if the connection with the server is lost.

The application of the upgrade to the Xilinx device can also be made fault tolerant in the system architecture, which is shown in Figure 4. The concept uses redundant non-volatile storage areas—one to hold the current, known good system, and the other to hold the upgrade. The system can always fall back to the known good configuration if there are any issues associated with upgrading the system.

Flexibility By Design - Pull or Push?

There are some applications that may require a "push" method of upgrading. The term "push"

entails the directing and initializing of upgrades from a central server. There are cases or management decisions that may dictate this alternative to providing upgrades.

The FieldUpgrader software primarily uses a device-initiated or "pull" methodology. However, it was developed to be flexible as well, and can be enhanced to allow a "push" methodology. At a pre-determined time, the UpgradeAgent may simply open a communications port to listen for "hello" messages that are broadcasted or multicasted from the UpgradeServer. If the UpgradeAgent responds after verification, the UpgradeServer sends a request packet to the system to begin the upgrade process. The UpgradeAgent then makes the request for an upgrade from the secure UpgradeServer. This methodology solves the problems associated with firewalls and security. The requests may pass through firewalls and the device port remains open for only the specified upgrade period.

System Support

GoAhead FieldUpgrader supports the following platforms:

- GoAhead UpgradeServer, GoAhead DeviceStudio:

  Windows NT4.0, Linux 2.2, Microsoft Internet Explorer 4.x or greater, Netscape 4.x or greater
- GoAhead UpgradeAgent:

  Wind River VxWorks 5.3.1 or greater (x86, PowerPC, ARM, MIPS), Windows 95/98/NT (x86, PowerPC), Windows CE (x86, Hitachi SH) or Linux 2.2 (x86)

The key requirements of the UpgradeAgent are:

- TCP/IP stack and connection.
- Real-time clock.
- File system (optional for VxWorks).
- Provision of the main software module's C source code (main.c) and a linkable library to enable the integration of the UpgradeAgent into custom applications.
- Support for embedded JavaScript calls with the ability to:
  - Control the upgrade process
  - Load and call C functions to extend functionality
  - Send additional request data to GoAhead UpgradeServer
  - Access GoAhead UpgradeAgent environment variables

Demonstration Design

The demonstration design utilizes the GoAhead FieldUpgrader to upgrade a Xilinx XCS05 Spartan FPGA, which is connected to a popular embedded system—a single-board computer running Wind River's VxWorks® real-time operating system. This type of system exists in many applications and environments from industrial control stations to remote monitoring systems. Figure 2 shows a diagram of the system.

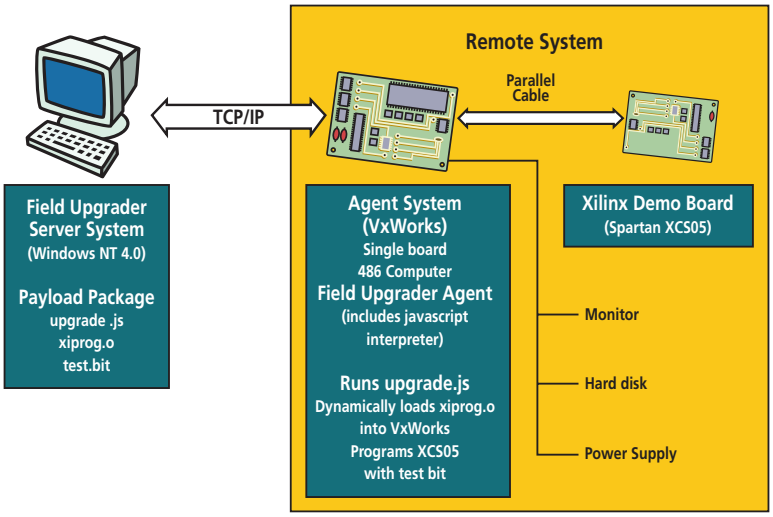

Figure 2 - Demonstration system.

This demonstration system includes the following components:

Processor - The processor is a single-board, Intel 486-based system. The board has 2MB of RAM, integrated VGA controller, an IDE controller, and a parallel interface. A hard disk is used for non-volatile storage other than the BIOS.

Xilinx demonstration board - This board has two devices on it: an XC3020A and a Spartan XS05. The Spartan XS05 is a 5000 system gate FPGA that is used in high volume, low-cost applications.

Xilinx Parallel Cable III - This cable runs between the single board computer and the Xilinx demonstration board. This cable is used to provide the communication between the computer system and the Xilinx FPGA.

Wind River VxWorks - The VxWorks® Real-Time Operating System (RTOS) from Wind River Systems is the most widely used RTOS for embedded systems.

The GoAhead UpgradeAgent - This is a real-time embedded application that uses VxWorks' dynamic loading capability to load and execute other real-time embedded applications. In the demonstration design, this feature of VxWorks is used to dynamically load the object code that programs the Xilinx FPGA. This was strictly an architectural choice. An alternative and equally valid approach is to have the object code that programs the Xilinx device be resident on the VxWorks system and not include it as part of the upgrade process, or use a combination of the two approaches.

The UpgradeAgent is configured to poll the UpgradeServer once each minute to check for upgrades. When one exists, it downloads the upgrade payload and executes the instructions included in the payload. For demonstration purposes, two payloads are published on the UpgradeServer: one to upgrade the bitstream and one to downgrade to the original bitstream. This enables the system to toggle between the two FPGA configurations.

UpgradeServer - The UpgradeServer is running on a laptop PC using Microsoft NT4.0. It waits until the UpgradeAgent on the VxWorks system contacts it and requests an upgrade. If there is an available upgrade, the server sends the upgrade payload to the VxWorks system across the TCP/IP network.

Demonstration Upgrade Process

The upgrade process used in the demonstration design is illustrated by the flow diagram in Figure 3.

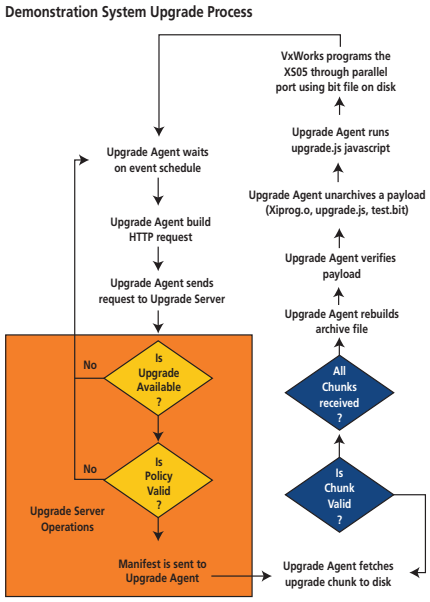

Figure 3 - Demonstration system upgrade process.

Building a Commercial System

There are several potential field upgradable architectures to which the concepts presented here apply. However, only one is described here—the single board computer running an RTOS with programmable logic, as shown in Figure 4.

In Figure 4, the processor must be one supported by the GoAhead UpgradeAgent. This can

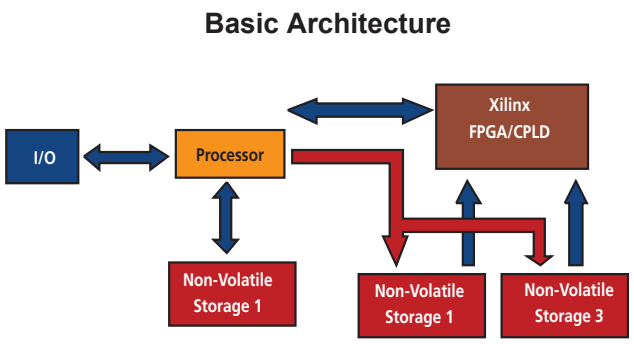

Figure 4 - Basic system architecture.

be either an Intel X86 (or compatible), PowerPC, ARM, or MIPS processor running VxWorks as the RTOS (or a supported Microsoft Windows or Linux configuration as previously listed).

The non-volatile storage can be divided into three functions:

- Processor Program Memory (Non-Volatile Storage 1) - contains firmware for the microprocessor.
- FPGA Configuration Storage (Non-Volatile Storage 2) - contains configuration data for the FPGA. The system can be designed so that this storage area contains the initial configuration data.
- FPGA Configuration Storage (Non-Volatile Storage 3) - contains configuration data for the FPGA. The system can be designed so that this storage area contains the first upgrade configuration data.

The system should be designed to always have a last known-good configuration for the FPGA. This is handled by the redundancy of the Non-Volatile Storage 2 and Non-Volatile Storage 3. In a usual sequence of events, the FPGA will be initially configured from Non-Volatile Storage 2. When an upgrade is requested, it is downloaded into Non-Volatile Storage 3. The FPGA is then reconfigured from this location. The program controlling the programming of the FPGA then switches the functions of these two storage areas as Non-Volatile Storage 3 is now the known-good configuration and Non-Volatile Storage 2 is ready to contain the next upgrade. Possible selections for the non-volatile memory are EEPROMS, FLASH memory cards, or the Xilinx XC18V00 in-system programmable configuration PROMs. (See page 62.)

Software

There are two basic software components that need to be written for the target processor architecture:

- Firmware to copy the Xilinx programming file to non-volatile storage.
- Firmware to program the FPGA.

Another approach is to use the Java programming language either with the firmware or as a complete application (Xilinx supplies a Java API for the Boundary Scan configuration mode). A Java application can therefore be written (and called by the UpgradeAgent) to program the FPGA. On Windows platforms, there are several Java run-time engines (the virtual machine) available. On a VxWorks platform, the application would need to be developed using Personal JWorks™ from Wind River.

Conclusion

The purpose of the GoAhead and Xilinx partnership is to provide the enabling technology and solutions for a variety of field upgradable systems. GoAhead FieldUpgrader provides the management, deployment, communications, and security backbone, while Xilinx provides reconfigurable hardware devices and programming technologies. Together, you have the building blocks needed to enable the rapid development and deployment of a new generation of cost-saving, life-extending, field-upgradable systems. Σ

For more information regarding how the Spartan-II family addresses traditional ASIC and ASSP designs, please see the article on page 49.

For more information see: www.xilinx.com/xilinxonline/partners/goaheadhome.htm