Field Upgradable Hardware and Software Systems

Enabling Technology from GoAhead Software and Xilinx

Market Challenges

The biggest embedded design challenges faced by manufacturers today are shortened development cycles and software testing (1998 EE Times Study). As development teams race to meet product deadlines, there is not enough time to add all the required features and still test for bugs before a device ships. An additional challenge is the lifetime of the product itself. Intense competitive pressures require better performance, more functionality and different form factors, often at a lower or at least equivalent cost. In some markets products must be delivered that meet new and evolving standards such as various communication protocols. Waiting too long for standard completion can result in a non-viable market position.

When a device does ship with bugs and/or a sub-optimal feature set manufacturers typically use one or more of the following options to upgrade these devices in the field. Each of these methods has its drawbacks.

1. End users download upgrades from the manufacturer's web site.
   This is time consuming and problematic for end-users with no technical skills.
2. CDs are mailed to end users to install upgrades.
   Producing CDs is expensive and not all upgrades are applied in a timely manner, if at all.
3. Support personnel make service calls to perform upgrades for the end users.
   Current rates for field service costs can reach $100+ per hour.
4. The product is recalled.
   A fatal bug could not only be a financial nightmare, but could also severely damage credibility in the marketplace.

If a device is designed with a fixed architecture, then meeting performance, feature, form factor, cost and evolving standards requirements usually results in a costly and timely redesign.

Xilinx and GoAhead Software have partnered to provide enabling technology and solutions that addresses these market challenges.

Upgradable Software Systems

Designing and deploying upgradable software systems addresses a portion of these challenges and problems. This technique has been used for a number of years. For example, embedded processor designs are often upgraded by downloading new firmware. This technique has also been used in the modem industry to enable product delivery under one communication protocol and through a future firmware upgrade support evolving standards such as occurred during the 56K standard development.

Upgradable Hardware Systems

Since their inception designers have been using Field Programmable Gate Arrays (FPGAs) in systems that have exploited the advantages of field programmability and quicker time-to-market. One of the primary roadblocks to realizing even more benefits to field programmability has been how to effectively manage and communicate with such devices once deployed.

Since the FPGA functionality is customizable, the types of systems that utilize FPGAs tend to vary considerably. In addition, the method by which the FPGA is programmed can also differ from system to system. To support remote field updates a system must have some sort of communication channel or connection across which data can be transferred. This could be a cable, modem connection, laser, infrared or radio transmitter / receiver interfaces, etc.

The main task in updating the functionality of the FPGA is replacing the bitstream used to program the FPGA. The best methodology for updating the bitstream is dependent on what resources the application has available to it and how these resources are connected.

Delivery Challenges

There are many challenges to the actual delivery of upgrades to remote systems, especially across an open network. Managing the upgrades on a per target basis, the transmission and communications mechanisms, and security are all areas that must have well engineered solutions. In the past, companies that wished to take advantage of upgradable systems have had to develop, deploy, and support proprietary systems. Not only does this requirement significantly increase development time, it locks out a large number of companies who do not possess the needed resources for such an endeavor.

Managing Upgrades

Managing the upgrade can be a challenge and the intelligence and amount of data tracking required can result in data explosion if the number of units to be upgraded is large. The common approach to the problem has been the implementation of proprietary, server-side solutions to manage the updating. The requirements for such a system can be immense, as the server must track each target unit and its individual state. This creates a scalability problem as well as issues if the device becomes or must become periodically untethered from the network.

Data Transmission

The type of communication channel that the update information travels across will affect the speed, security and integrity of the data that is used to update the FPGA. If a communication interface is already being used for sending and receiving data in the system, it can usually also be used by the designer to perform the remote update. Some of the issues that should be considered when evaluating an interface's suitability for doing remote upgrades are listed below.

The prevalence of the Internet makes this communication channel highly desirable as the infrastructure can be employed freely. Any large-scale private network is cost prohibitive to deploy and limits the ability to deploy field upgradable systems for a large set of potential applications. An open network such as the Internet, however, presents its own issues in the area of security.

Security

Security in delivering the upgrade, especially across unsecured networks, is of paramount importance. There are essentially four primary components that must be considered: validation, verification, integrity, and fault tolerance.

- Validation
  Validation in the field upgrade arena is the ability to validate that the upgrade sender is a valid sender and the upgrade recipient is a valid recipient.
- Verification
  Verification of the upgrade refers to the ability to verify that the target has successfully received the upgrade.
- Integrity
  Integrity of the upgrade is the ability to ensure that the contents of the upgrade itself have not been altered.

- Fault Tolerance
  This refers to the ability to recover from faults during both the transmission of the upgrade from the server to the target as well as the ability to recover from the application of the upgrade itself.

Design Challenges

A field upgradable system requires a well-thought design. The introduction of field upgradable hardware as well as software/firmware/data opens up even more possibilities for the system architecture. What was previously done only in firmware for future upgrade reasons, can also be done now in hardware using Xilinx FPGAs and CPLDs.

Designing systems that do support field upgrades can also provide new revenue prospects. After the initial product is released, new hardware features can be developed, sold and distributed inexpensively to existing customers in much the same way that new versions of software can be distributed today. In addition a standard "off-the-shelf" application can be developed so features can be swapped in and out depending on what the end-user purchases or needs.

Potential Product Areas

The types of systems that could benefit from being field updateable are wide-ranging. Almost any system that has some type of connectivity to the "outside world" could potentially benefit from being designed to support field updates. Some typical types of applications include:

- Internet Appliances
  - Set Top Box
  - Security Systems
  - Smart Phones
  - PDAs
- Network Equipment
  - ATM Switch
  - Cellular Base Station
  - Home Networking Systems
- Satellite Communications System
- Data Communications (Wired & Wireless)
  - HDTV
  - Video, Image Processing
  - Encryption
- Military Communications
  - Surveillance
  - Radar, Sonar

GoAhead Software and Xilinx – Enabling the Solution

The GoAhead Software and Xilinx partnership leverages the GoAhead commercial software solution, GoAhead FieldUpgrader™, with the Xilinx Online and Xilinx Internet Reconfigurable Logic, IRL™, initiatives to provide the enabling technology for field upgradable systems. This enabling technology provides the backbone for both the system design and the delivery and management mechanisms to successfully design and deploy upgradable products.

GoAhead FieldUpgrader

GoAhead FieldUpgrader consists of three parts: GoAhead DeviceStudio™, GoAhead UpgradeAgent™ and GoAhead UpgradeServer™. DeviceStudio is a development environment used to configure the UpgradeAgent. Once configured, the UpgradeAgent is embedded in the target device. Each agent is unique to the particular operating system that is resident on the target device. UpgradeServer is used to create and publish upgrades and is usually located on a server at the manufacturer's site.

When the need for an upgrade arises, the manufacturer publishes the upgrade with UpgradeServer. The target device containing the UpgradeAgent polls the UpgradeServer at regular pre-determined intervals to look for upgrades. When an upgrade is available, the device downloads the upgrade and applies it as specified by the manufacturer. This process is illustrated in Figure 1.

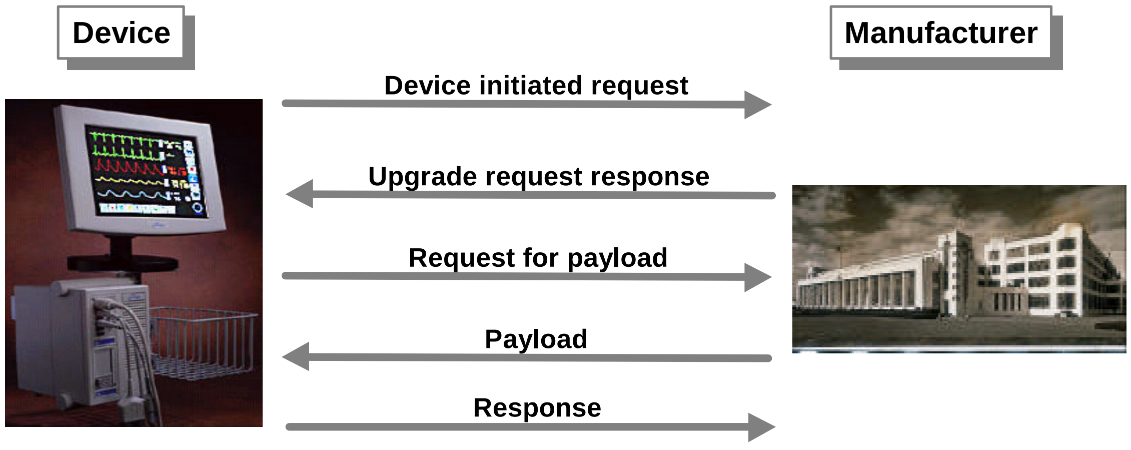

Figure 1

Security

To upgrade devices over the Internet, it is critical that the devices only accept upgrades from the authorized, designated server and the server cannot be spoofed. It is also important to ensure that the upgrade payload is not modified en-route to the target device. GoAhead FieldUpgrader and the UpgradeServer use a variety of techniques to authenticate the upgrade and prevent modifications along the way.

GoAhead FieldUpgrader utilizes the Digital Signature Standard (DSS), which specifies a Digital Signature Algorithm (DSA) appropriate for applications requiring a digital rather than written signature. The DSA digital signature is a pair of large numbers represented in a computer as strings of binary digits. The digital signature is computed using a set of rules (i.e. the DSA) and a set of parameters such that the identity of the signatory and integrity of the data can be verified.

The DSA provides the capability to generate and verify signatures. Signature generation makes use of a private key and the payload message to generate a digital signature. Signature verification makes use of a public key that corresponds to, but is not the same as, the private key. Each user possesses a private and public key pair. Public keys are assumed to be known to the public in general while private keys are never shared. Anyone can verify the signature of a user by employing that user's public key.

Only the possessor of the user’s private key can perform signature generation. A hash function is used in the signature generation process to obtain a condensed version of the payload message called a message digest. The message digest is then input to the DSA to generate the digital signature. The digital signature is sent to the intended verifier along with the signed data message. The verifier of the message and signature verifies the signature by using the sender's public key. The same hash function must also be used in the verification process

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.

Data Integrity

The message digest (secure hash algorithm) and the digital signature discussed above in the security section ensures that the upgrade payload received at the target device matches the original upgrade payload published at the upgrade server. This ensures that the message was not corrupted or modified en-route to the device.

Additionally, 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 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 using the digital signature verification algorithm.

In the combination GoAhead/Xilinx solution, the data integrity is also checked during the actual programming of the Xilinx FPGAs. Xilinx has a CRC checksum built into the programming step.

Fault Tolerant

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 be made fault tolerant in the system architecture. The proper architecture will be introduced later in this paper. The concept is to use redundant non-volatile storage – 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.

What systems does it support?

GoAhead FieldUpgrader supports the following platforms:

- GoAhead UpgradeServer
  Windows NT4.0 or Linux 2.2 and Microsoft Internet Explorer 4.x or greater or Netscape 4.x or greater.
- GoAhead DeviceStudio
  Windows NT4.0 or Linux 2.2 and Microsoft Internet Explorer 4.x or greater or Netscape 4.x or greater.
- GoAhead UpgradeAgent
  Windows 95/98/NT (x86, PPC), Windwos CE (x86, SH), VxWorks 5.3.1 or greater (x86, PPC, ARM, MIPS), or Linux 2.2 (x86)

  They key features of the UpgradeAgent for system design consideration are:
  - TCP/IP stack and connection
  - Real-time clock
  - 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
  - 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

Xilinx Programmable Logic

Xilinx FPGAs are based on SRAM technology and are customized by loading configuration data into internal memory cells. Because FPGAs are SRAM-based, it is very easy to re-program them an unlimited number of times. Updating the functionality of the FPGA only requires the designer to include a mechanism for updating the configuration bitstream.

The configuration bitstream is the programming file for the FPGA. This bitstream is created using the Xilinx development tools. Design entry can be performed using a variety of standard methods such as HDL or schematic entry.

Demonstration Design

The demonstration design utilizes the GoAhead FieldUpgrader solution to upgrade a Xilinx Spartan FPGA. The Xilinx Spartan FPGA is connected to a popular embedded system – a single-board computer running WindRiver's VxWorks® real-time operating system. This type of system exists in many types of application and environments from industrial control stations to remote monitoring systems. Figure 2 provides a diagram of the demonstration system.

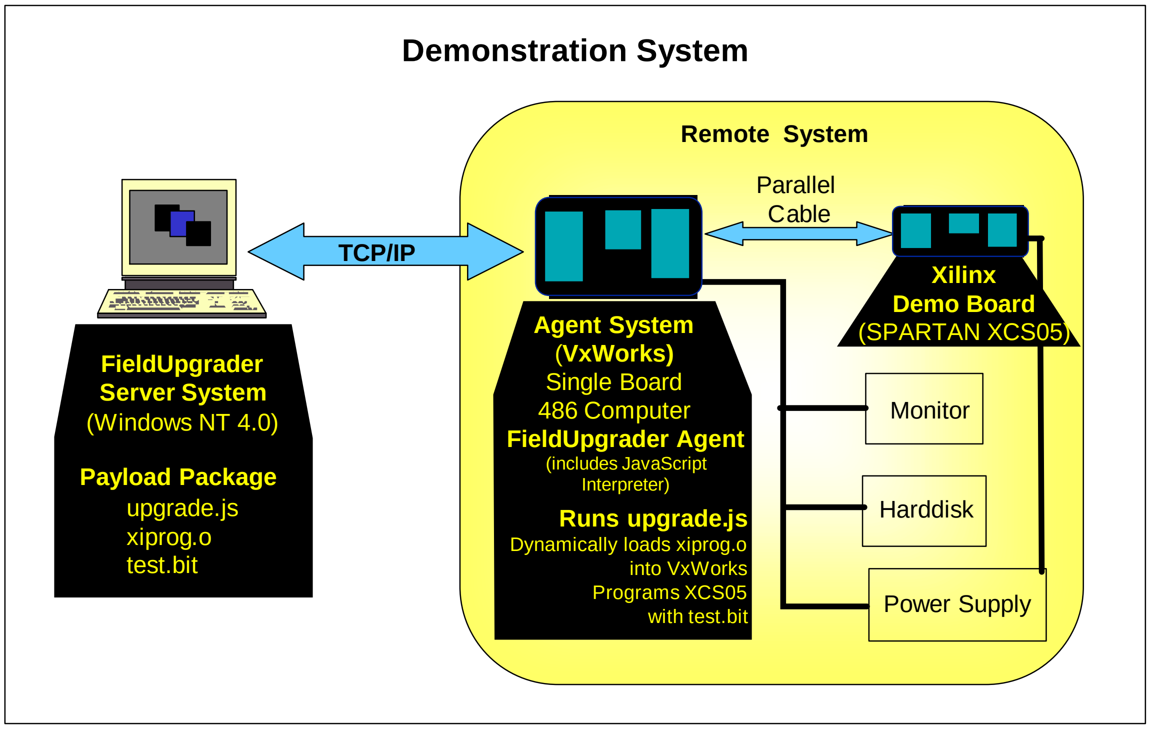

Figure 2

Processor

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

Xilinx demonstration board

The Xilinx demonstration board has two devices on it: an XC3020A and a Spartan XS05. The Spartan XS05 is a 5,000 system gate FPGA that is used in high volume applications and is a core component of Xilinx's ASIC alternative strategy.

Parallel Cable III

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

VxWorks on Field Upgradable System

The single board, Intel x86 processor-based system runs the VxWorks® Real-Time Operating System (RTOS) from WindRiver Systems. VxWorks is the most widely used RTOS for embedded systems.

The GoAhead UpgradeAgent is a real-time embedded application that utilizes 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.

GoAhead FieldUpgrader 2.2

The three applications that comprise GoAhead FieldUpgrader are used in the demonstration design.

DeviceStudio

DeviceStudio is used to create and configure the UpgradeAgent that runs on the VxWorks system.

UpgradeAgent

The UpgradeAgent runs on the VxWorks system and is configured to poll the UpgradeServer once each minute to check for ugrades. 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 running Microsoft NT4.0. The UpgradeServer waits until the UpgradeAgent on the VxWorks system contacts it and requests an upgrade. The UpgradeServer checks any defined policies and if the policy is valid. If there is an available upgrade, the server sends the upgrade payload to the VxWorks systems across the TCP/IP network.

In the demonstration design, the IP Policy is utilized which verifies that the UpgradeAgent is running on a listed valid IP address.

The UpgradeServer is used to create, configure and publish the upgrade module. The upgrade module consists of the components that the GoAhead UpgradeAgent utilizes to perform the upgrade, in this case program the FPGA with new functionality. This module comprises the upgrade payload that is sent to the UpgradeAgent. The components of the upgrade module are:

- Upgrade.js
  This is a JavaScript file that the UpgradeAgent will run after the download of the upgrade payload is complete. This script performs two functions:
  - Dynamically loads the Xiprog.o VxWorks object code into the operating system.
  - Calls the function within the Xiprog.o object code (*ppProgramXilinxFPGA*) that reprograms the FPGA.

  The contents of the Upgrade.js file are:

  ```
  handle = loadModule("xiprog.o");
  callFunction(handle, "ppProgramXilinxFPGA", "test.bit");
  ```

- Xiprog.o
  This file contains the VxWorks object code that consists of the functionality to download the Test.bit file from storage across the parallel cable and reprogram the FPGA. The FPGA is programmed using the *slave-serial* mode.
- Test.bit
  This file is the Xilinx Spartan bitstream file that contains the new functionality to reprogram the FPGA with.

Demonstration Upgrade Process

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

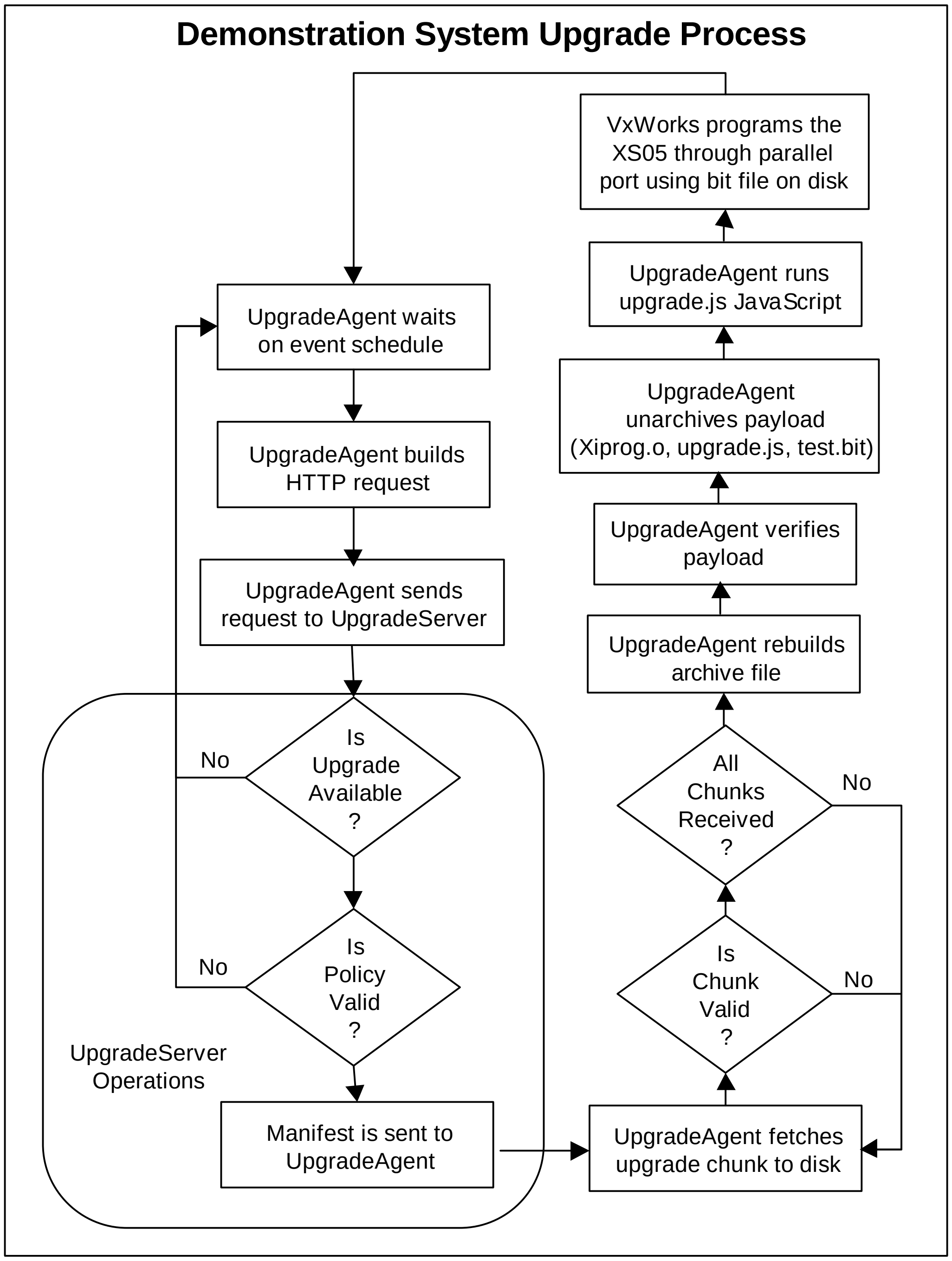
Demonstration System Upgrade Process
UpgradeAgent waits on event schedule
UpgradeAgent builds HTTP request
UpgradeAgent sends request to UpgradeServer
Is Upgrade Available ?
No
Is Policy Valid ?
No
UpgradeServer Operations
Manifest is sent to UpgradeAgent
UpgradeAgent fetches upgrade chunk to disk
Is Chunk Valid ?
No
All Chunks Received ?
No
UpgradeAgent rebuilds archive file
UpgradeAgent verifies payload
UpgradeAgent unarchives payload (Xiprog.o, upgrade.js, test.bit)
UpgradeAgent runs upgrade.js JavaScript
VxWorks programs the XS05 through parallel port using bit file on disk

Figure 3

Building a Commercial System

There are several potential field upgradable architectures that the concepts presented in the demonstration system apply to. For the sake of brevity, only one will be expanded upon – the single board computer running an RTOS with programmable logic, as is shown in Figure 4.

Architecture

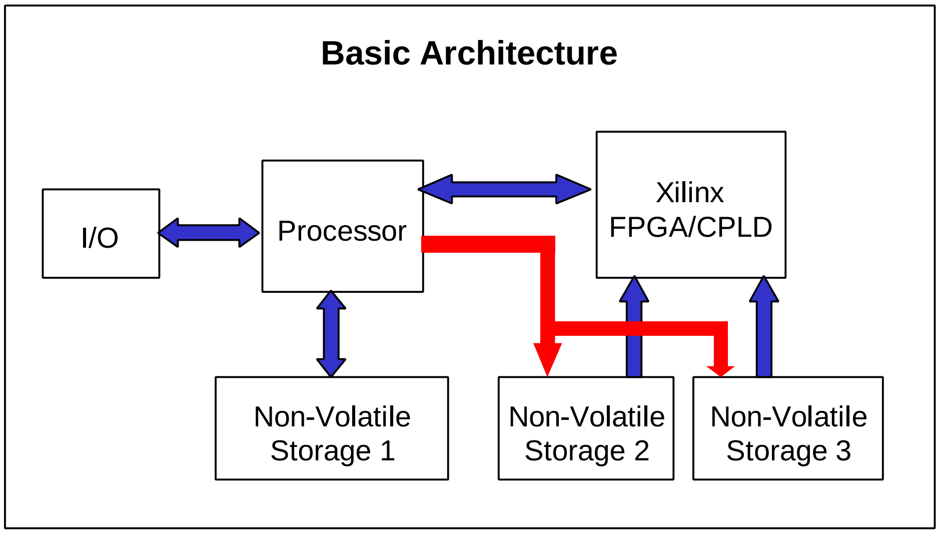

Figure 4

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

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

- Processor Program Memory (Non-Volatile Storage 1)
  This memory area contains firmware for the microprocessor.
- FPGA Configuration Storage (Non-Volatile Storage 2)
  This memory area 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)
  This memory area 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 so that there is always a last known-good configuration for the FPGA. This is handled by the redundancy of the Non-Volatile Storage 2&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 in 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 now ready to contain the next upgrade. Possible technology selections for the non-volatile memory are the Xilinx XC18V00 in-system programmable configuration PROMs, FLASH cards and EEPROMs.

The mechanisms to configure the FPGA also need to be defined during the system design phase. Xilinx supports several different programming solutions depending upon the device used. The programming solution choice should take into account the software implications, as software will have to modified and/or written for the target platform. Some of the possible configuration modes in a field upgradable system are:

- Slave Serial
  In this mode, the FPGA receives configuration data in bit-serial from a serial PROM or other source of serial configuration data. Multiple FPGAs can be daisy chained for configuration from a single source.

The clock signal used by the FPGA for configuration is generated by an external source, such as a microprocessor or CPLD.

- Slave Parallel (Spartan II) or Express (SpartanXL)
  These modes are similar to Slave Serial, except that data is processed one byte per clock cycle instead of one bit per clock cycle. These modes are the fastest programming modes for the supported devices.
- SelectMAP (Virtex)
  This mode is similar to the two Spartan modes previous presented, and is the fastest programming mode for the Virtex devices. Byte-serial data is written to the FPGA and the controlling signals (clock, chip select, write and busy) are provided by an external source. This mode can also read configuration data out of the FPGA.
- Boundary-scan
  In this mode, configuration is done using the IEEE Standard 1149.1 Test Access Port (TAP). For more information on how to utilize a microprocessor/controller to program a FPGA using JTAG, please refer to the pointer in the *Additional Information* section titled *Xilinx In-System Programming Using an Embedded Microcontroller.*

  Xilinx also provides a Java API for use with boundary scan programming. As the systems GoAhead supports also have Java Virtual Machines available, please refer to the pointer in the *Additional Information* section titled *Xilinx Online*.

Software

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

- Firmware to copy the Xilinx programming file to non-volatile storage
  This firmware is storage architecture dependent. The firmware needs to understand how to move the file from either the file system used by the RTOS or, new with GoAhead FieldUpgrader 2.2, its in-memory file system to the target storage device. In the demonstration design, this function is not needed as the storage device is the hard disk and the FPGA is programmed from the bit file on the disk.

- Firmware to program the FPGA
  The Xiprog.o file represents this firmware in the demonstration system. Its functions are to control the programming of the FPGA. This program must read the configuration file from non-volatile storage and feed it to the FPGA using one of the previously discussed configuration modes.

  Another approach is to use the Java programming language either with the firmware or as a complete application. As mentioned, Xilinx released a Java API for the boundary scan configuration mode. A Java application can therefore be written that the UpgradeAgent calls that would 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 WindRiver.

Conclusion

The GoAhead and Xilinx partnership's purpose is to provide the enabling technology and solutions for a variety of field upgradable systems. The types of target systems are processor based running either Windows CE/95/98/NT, Linux 2.2, or the VxWorks RTOS. GoAhead FieldUpgrader provides the management, deployment, communications and security backbone. 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, product life-cycle extending, field upgradable systems.

Additional Information

GoAhead

http://www.goahead.com
http://www.goahead.com/fieldupgrader/fieldupgrader.htm
http://www.goahead.com/fieldupgrader/fup_ds8.pdf “FieldUpgrader Datasheet”
http://www.goahead.com/fieldupgrader/GoAead FieldUpgrader White Paper.doc “FieldUpgrader White Paper”
http://www.goahead/com/fieldupgrader/Creating Self-Managing Internet Devices.doc “Self-Managing Internet Devices”

Xilinx Support

http://www.support.xilinx.com/apps/config.htm
http://www.support.xilinx.com/xapp/XAPP178.pdf “Configuring Spartan-II FPGAs from Parallel EPROMs“
http://www.support.xilinx.com/xapp/XAPP176.pdf “Spartan-II FPGA Family Configuration and Readback”
http://www.support.xilinx.com/xapp/XAPP138.pdf “Virtex Configuration and Readback”
http://www.support.xilinx.com/xapp/XAPP13.pdf “Configuring Virtex FPGAs from Parallel EPROMs with a CPLD”
http://www.support.xilinx.com/xapp/XAPP122.pdf “The Express Configuration of Spartan-XL FPGAs”
http://www.support.xilinx.com/xapp/XAPP098.pdf “The Low Cost, Efficient Serial Configuration of Spartan FPGAs”
http://www.support.xilinx.com/xapp/XAPP058.pdf “Xilinx In-System Programming Using an Embedded Microcontroller”
http://www.support.xilinx.com/appnotes/bus_conf.pdf “Configuring FPGAs Over a Processor Bus”
http://www.support.xilinx.com/xcell/xl32/xl32_10.pdf “Silicon Xpresso and Internet Reconfigurable Logic”
http://www.support.xilinx.com/xcell/xl32/xl32_13.pdf “Internet Appliances”
http://www.support.xilinx.com/xcell/xl31/xl31_6.pdf “New Internet Reconfigurable Logic for Creating Web-enabled Devices”
http://www.support.xilinx.com/partinfo/databook.htm#SPROM “XC1800 Family of ISP Configuration PROMs”

Xilinx Online

http://www.support.xilinx.com/xilinxonline/index.htm