XAPP412 (v1.0) June 29, 2001

Architecting Systems for Upgradability with IRL (Internet Reconfigurable Logic)

Summary

Internet Reconfigurable Logic (IRL™) is a system design methodology to enable the remote upgrade of hardware, while insuring the reliability of the upgrade. FPGAs, which are “Field Programmable” are inherently capable of changing their functionality with a new bitstream. IRL takes advantage of this capability by delivering new bitstreams and software drivers to the remote hardware.

This application note will describe the basic concepts of an IRL-enabled system, detail design considerations for building an IRL system and give a high level description of the PAVE Framework, the Xilinx API and development framework that enables embedded systems to be upgraded.

Introduction

The advent of Xilinx FPGAs, Flash Memory devices and ubiquitous networks provide the means to store bitstreams and then upgrade them once the hardware has been shipped to the final customer. Architecting your system for IRL will allow you to upgrade software, drivers, firmware, and hardware remotely.

Reasons for enabling your system for field upgradability include:

- Interoperability - Products frequently have to interoperate with other vendor’s products, but there is no reasonable way to test all the possible interactions prior to shipping the product. If the system is IRL-enabled, interoperability issues can be resolved at a minimal cost.
- Time To Market - The hardware can be shipped sooner with a subset of the full functionality. Features that would have taken too long to add prior to the initial release can be added after shipment.
- Design Corrections - In the event a flaw in the product appears after it ships to the final customer, it can be corrected without the need for returns, recalls, field service, and the accompanying customer dissatisfaction
- Performance Upgrades - The performance of the system can be upgraded as the engineering team has time to tune the algorithms and data paths.

IRL Concepts

What is IRL?

Internet Reconfigurable Logic is a system design methodology that enables modification and upgrading of hardware and software in a target system across a network without the need for a service technician or user to directly perform the change. This methodology, when applied to the design process, creates products that are IRL-enabled. IRL can enable upgrades of multiple systems simultaneously, and the ability to go back to a previous configuration if necessary.

A typical IRL-enabled system might include a

- A 32-bit processor based design with TCP/IP networking connectivity. An industry standard example of this is the **Single Board Computer** (SBC), as typically seen in CompactPCI and VME implementations.
- **Real Time Operating System** (RTOS) such as the WindRiver® Systems’ VxWorks®
- Xilinx PAVE (**P**LD **A**PI **V**xWorks **E**mbedded) Framework

When an upgrade is available, it would be sent to the target, where the PAVE API would perform the upgrade. For example, a system, when IRL-enabled, might be able to autonomously upgrade itself and recover from a power failure during this upgrade.

Elements of an IRL system

Creating an IRL-enabled system requires certain hardware and infrastructure components that will allow the remote modifications to occur. As shown in Figure 1 below, there are several elements to an IRL System.

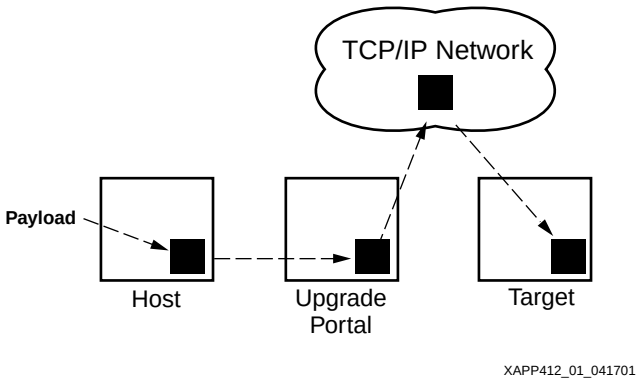

Figure 1: **Block Diagram of Internet Reconfigurable Logic System**

The **Host** is where hardware/software design environment resides and where the FPGA bitstreams and application software are created. This would include the Xilinx design tools, the RTOS build environment (such as WindRiver Systems' Tornado®) where your software applications are developed, and the PAVE System Integration Framework (SIF), which ties all of these efforts together.

Once the upgrade is created, it is assembled into a **Payload** that is sent to the system to be upgraded. The PAVE Framework includes utilities that allow generation of the payload for the build environment on the Host.

The **Upgrade Portal** is the computer your Target communicates with to obtain the payload. This could reside in your domain, or your end customers could operate it.

The **Network** shown in Figure 1 can be any TCP/IP based network: an Intranet, a local network, a Virtual Private Network (VPN) or even the public Internet. The type of network used will depend on the security requirements and the connectivity available at the location of the final product. PAVE can perform a basic TCP/IP socket connection; any additional protocols for security or other purposes would need to be added by the developer.

The **Target** system is the system that needs the hardware and/or software upgrade. This is the product shipped to your customers and which resides remotely. This IRL-enabled target system will, at a minimum, have a processor running the user's application, the PAVE API (part of the PAVE Framework), the RTOS runtime client (such as WindRiver Systems' VxWorks), and an FPGA. The processor handles communication with the network and has connectivity to the FPGA. The PAVE API is called to perform the upgrade by the user embedded application.

A typical payload structure is shown in Figure 2. Since changes in hardware usually imply new software drivers, these are included in the payload structure, so the drivers can be upgraded concurrently with the hardware. The applications that run on the target can be upgraded as well.

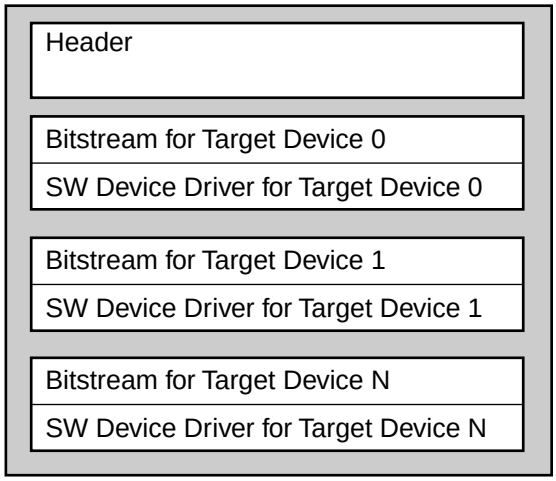

Figure 2: **Payload Diagram**

Expanding on the block diagram in Figure 1, an IRL system in the field could look similar to Figure 3. Here we have a target processor, a system or peripheral bus, and the FPGA(s). The processor is running the user's application, PAVE API, and the WindRiver RTOS. The Upgrade portal is running a PAVE client that communicates with the PAVE Server running on the target. The payload passes from the host to the target, via the upgrade portal and the Internet. Once it arrives at the target, the PAVE Server and API perform the required functions to upgrade the system.

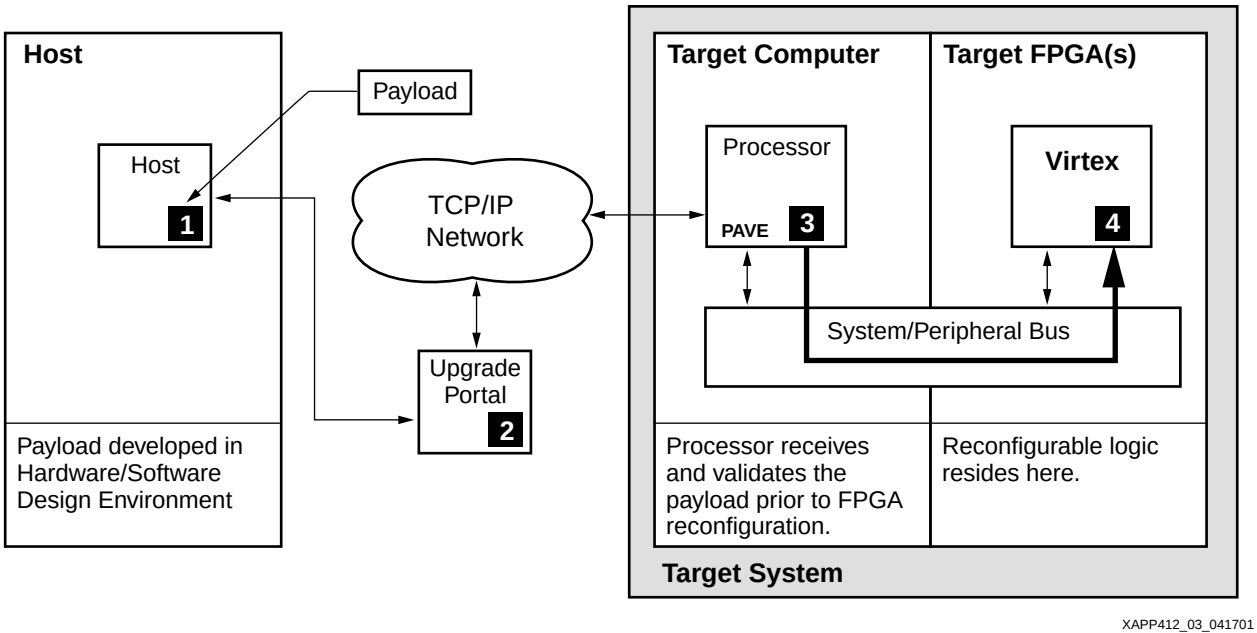

Figure 3: **Fielded IRL System**

Host, Upgrade Portal, and Network Concepts

The beginning of the upgrade process is the creation of new FPGA designs and accompanying software drivers, followed by testing in an appropriate environment. Once the upgrade is ready to be sent to the field, the developer uses the utilities supplied with PAVE to create the payload.

After the payload has been assembled, the developer would publish it out to the Upgrade portal, similar to how files are published for internet delivery. Once the payload has been published to the upgrade portal, there are two main means to deliver the payload to the target system.

Push (see Figure 4) is similar to broadcasting; the payloads are sent by the upgrade portal to each target system. This allows the Upgrade portal to control the upgrade process and ensure all systems have been upgraded.

Pull (see Figure 5) is similar to FTP; the target system contacts the upgrade portal to see if new upgrades are available. If so, the payload is pulled off the portal by the target.

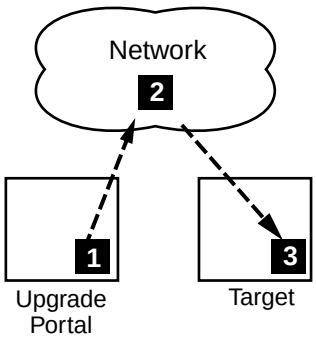

Figure 4: **Pushing a payload to the target**

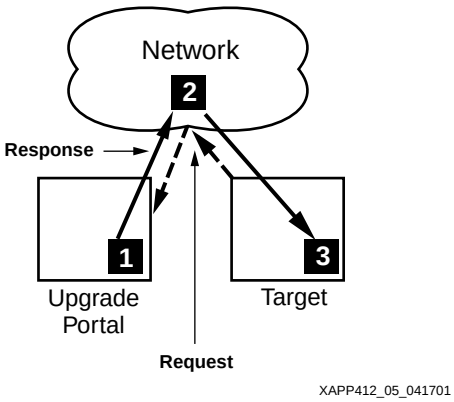

Figure 5: **Pulling a payload from the upgrade portal**

Careful consideration of using push vs. pull should be done to ensure that upgrades do not interfere with the end user's operation of the system. The operator of a high-availability system, such as the telecommunication services, might run the upgrade portal; in this case push would offer complete control over the process. A user of a low-cost consumer product would not have control of the upgrade portal. This user might prefer to have the option of upgrading or not; in this case pull would be the best choice. If the upgrades are not free, the upgrade portal may need to authenticate the user to ensure the upgrade was purchased.

Target Software Concepts

Figure 6 is a model of the software stack that runs on the target. At the highest level is the user applications. Running concurrently with the application is the PAVE API and server that caches the payload, and then performs the upgrade.

On the second level, the PAVE API provides system calls for the customer C++ applications to perform the reconfiguration process. The customer applications and API both interface directly with the RTOS.

The third level is the WindRiver RTOS. VxWorks is the run-time component of the Tornado II embedded development platform and acts as the operating system "kernel" on your target system. PAVE works directly with the VxWorks RTOS.

The **Board Support Package** (BSP) in level four in the stack is required to interface the desired processor to the RTOS. Each different SBC running an RTOS will need a Board Support Package to abstract the processor from the RTOS. The BSP used must match the RTOS and the embedded processor combination used in your system. PAVE assumes the existence of the BSP.

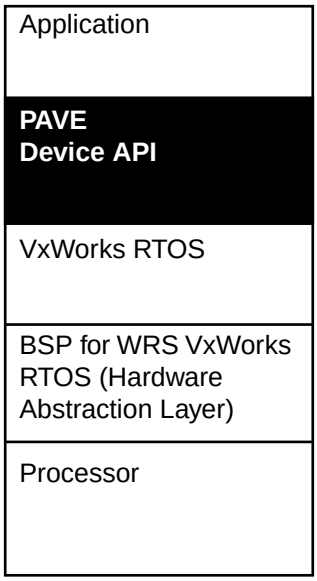

Figure 6: **Target software stack**

Target Hardware concepts

Processor Coupling

In the embedded market, processors have a bus known as the **Processor Local Bus** (PLB) that is directly fed from the processor and an **Embedded System Bus** (ESB), such as PCI, that usually requires a bridge or host chip to interface from the system bus to this secondary bus. The PLB varies depending on the processor and is not a standardized bus like PCI. The Embedded System Bus is not to be confused with the term "system bus", widely used in PC architectures to refer to the PLB. Connecting to the processor through an ESB is considered to be **Loosely coupled** and connecting through the PLB is considered to be **Tightly coupled**. In Figure 7 we see an example of these two different processor couplings.

Until recently, advanced processors (32-bit) could only be accessed through bridge chips supplied by the processor vendor. This would lead to a multi-chip connection, which added performance bottlenecks, consumed board space and power, and added cost to the design. Now, with programmable logic, it's possible to directly access the processor local bus, eliminating this series of chips, which is enhancing the importance of tight coupling to the PLB in newer designs.

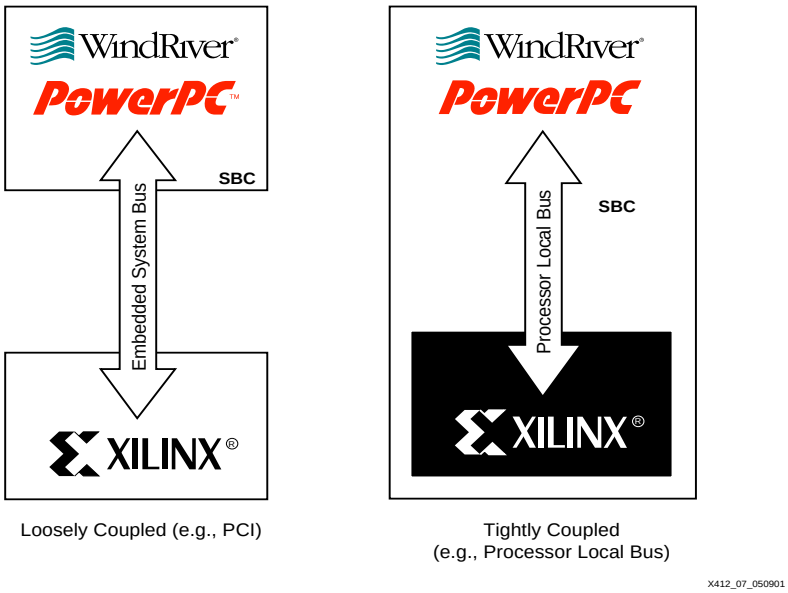

Figure 7: **Processor coupling**

Double Buffering

FPGA bitstreams are frequently stored on flash devices (including Xilinx XC1800 series devices), which can experience problems if the power fails while being written. IRL involves designing your hardware so that it is impervious to power failures during the upgrade process. The goal is to never have a piece of hardware that fails to operate.

For the IRL hardware to meet this requirement, it should have a **Double Buffer** design. One example method could consisting of a **Default** configuration that is always available and a second configuration that can store the upgrade.This Default configuration is never upgraded or changed except at the factory. Addition of the second storage location allows upgrades to occur, since the Default can not be changed. Double buffering ensures the hardware can be reliably upgraded.

Rollback is the ability to revert to a previous upgrade (possibly the Default). In a system that has space for more than two configurations, (e.g. using a commodity flash chip), it could rollback to a known good upgrade that was previously installed.

IRL Examples

Having examined the concepts that make up the IRL design methodology, let's examine a few practical examples of how to implement an IRL-enabled target system using PAVE.

Basic IRL-enabled System

Figure 8 shows an IRL-enabled system with a processor, an FPGA, and multiple FPGA configuration storage areas. The Processor communicates with the FPGA and, after configuration, can perform an update of the upgrade PROM. A register in the bridge address space receives the new bitstream and writes it out to the PROM via the JTAG controller.

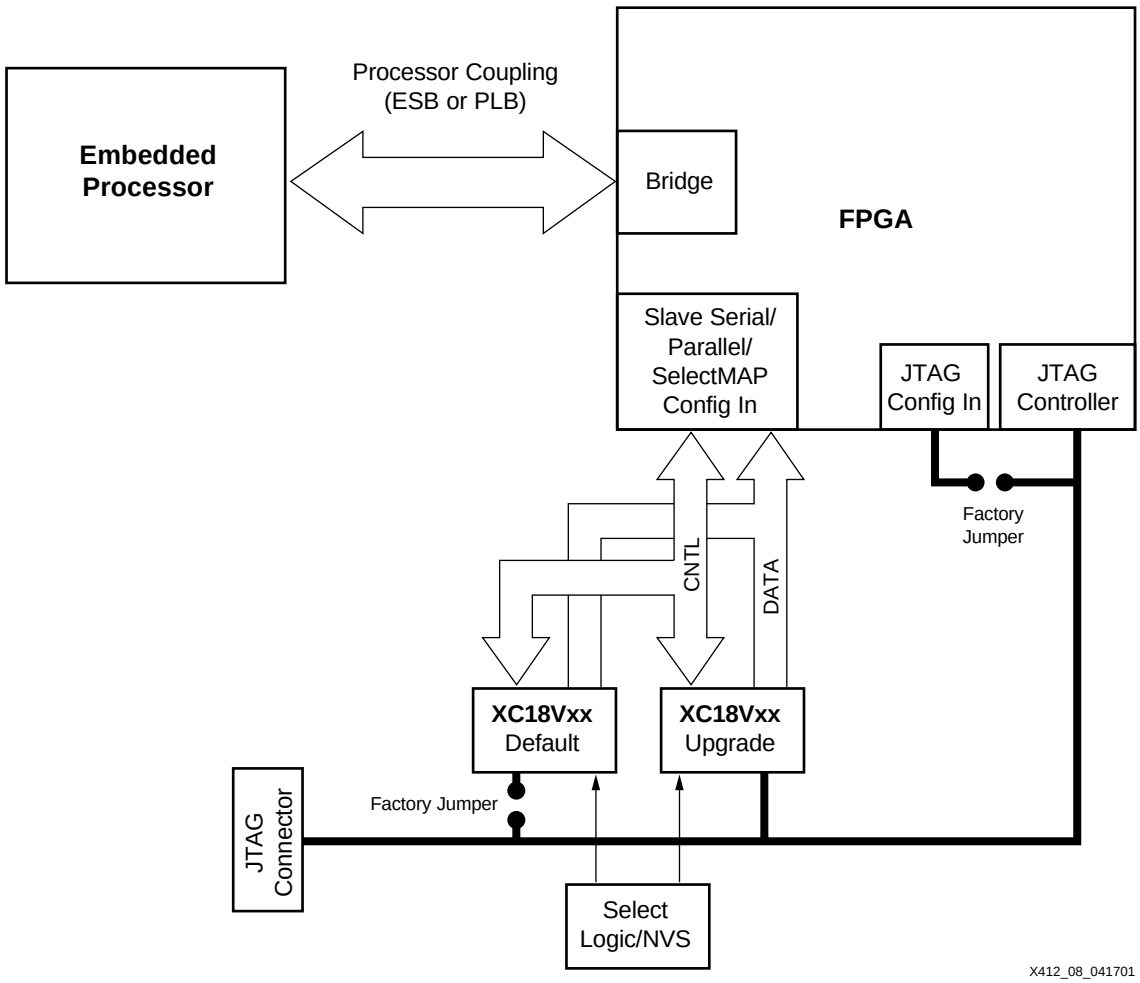

Figure 8: **Example two PROM system**

The PROM marked "Default" is the known good configuration from the factory. The default should never be upgraded in the field as it provides a baseline configuration that the hardware can revert to in case of failure of the upgrade process. This protects the hardware against power failures, customer or technician mistakes, and any other failure mode that would render the hardware inoperable (and non-upgradable). By preventing the end user from updating this PROM, he will always have a fallback position in the event the upgrade fails. The factory jumpers on the Default PROM's JTAG lines physically prevent the changing of this PROM, except during the manufacturing process. The upgrade PROM can be changed through the JTAG controller in the FPGA. With only two storage locations, the new upgrade always overwrites the old upgrade.

The Select Logic and Non-volatile storage (NVS) is to determine which PROM should be used and use the default if a configuration error occurs during the loading of the upgrade. In it's simplest form, it would attempt to load the upgrade PROM, monitor the DONE line of the FPGA, and if it failed, automatically revert to the default PROM. Adding a small NVS device, such as a Dallas Semiconductor DS2430A (scratchpad EEPROM) would allow specifying which PROM to boot from initially. This NVS could allow a more sophisticated approach of choosing among multiple upgrades. The select logic could be a CPLD or even something simpler, but, like the default, it should not be modifiable outside the factory (unless there is a double buffer for the CPLD configuration).

In the event of a configuration fault, the select logic should be able to detect this and attempt to configure the FPGA with the Default bitstream. If the bitstream in the upgrade buffer is

corrupted or non-existent, the FPGA DONE signal will not go high. In this case the select logic should attempt to load the default bitstream.

IRL in a Bridge System

Figure 9 shows an IRL system with a bridge and the two PROM model discussed in the last example. The bridge FPGA initializes off the PROMs; subsequently the target FPGA can be configured from the processor through the bridge. In the previous example the FPGA was both the bridge and the target. The interface in this case could be with either the ESB or PLB. A register in the Bridge interface would accept the configuration data sent from the processor and pass it on to the target via either the SelectMAP or JTAG controllers.

Use of a bridge in your system is not an IRL requirement; this example may or may not apply to your design. This figure is an example of how you could perform double buffering, but not the only way.

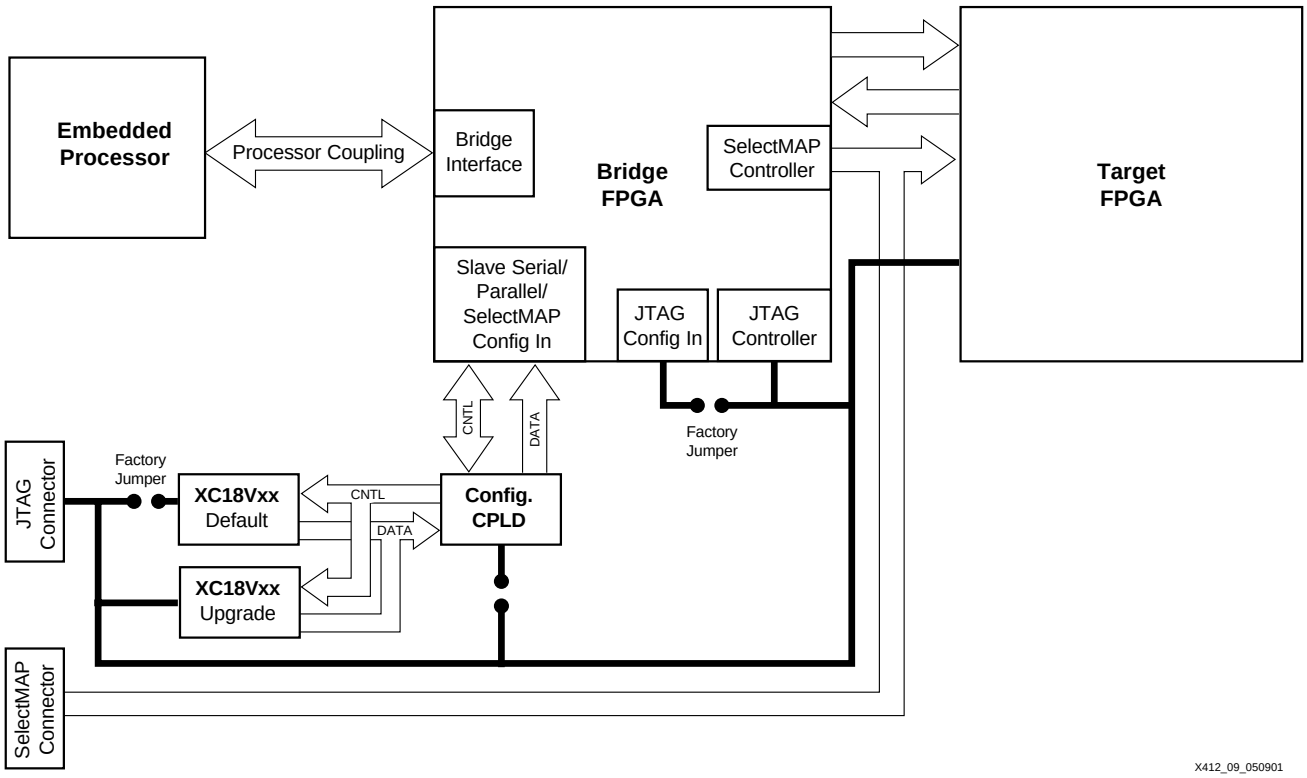

Figure 9: **System with Bridge and Target FPGAs**

In a programmable bridge system the processor cannot directly send configuration data prior to the initial configuration of the bridge FPGA. All of the aforementioned details on insuring a known good configuration still applies to this bridge. For the target FPGA in this diagram, the processor is able to send configurations directly to it from the processor's data storage. In this case, two means of configuration supported under PAVE are shown, SelectMAP and JTAG. The select logic used by the bridge is a CPLD that is acting as a mux for the two PROMs.

General IRL System Considerations for Bridges

Communication between the Processor and the target FPGA occurs through a bridge. The bridge facilitates the interface to the processor through the specified interface (e.g. ESB, PLB). Most processors require a separate chip (a Bridge) to support an ESB. When using a bridge chip, the processor is not directly mastering the bus to the FPGA. A few processors do have direct ESB support on chip. These are considered to have the bridge built-in; this bridge would be non-upgradable.

Most SBCs do not provide direct access to the PLB via a plug-in form factor. In the case of a CompactPCI system, a form factor known as PCI Mezzanine Card (PMC) is typically used. A PMC card loosely coupled to the processor could be on the SBC board, or a PMC carrier in the same chassis. A tight coupling would be the processor local bus (PLB), such as the PowerPC 405GP peripheral bus that is fed directly from the processor. The upgrade to the FPGA passes through this coupling and into the FPGA; this data is then updated into the appropriate storage area.

Memory usage for storing bitstreams

Building on the models in last two examples, this next example adds additional memory space for bitstream storage. Figure 10 shows a loosely coupled PMC system with configuration flash in addition to the two PROMs. This flash chip is a standard commodity flash, which are available in varying sizes. Depending on the design, the flash chip could store additional Bridge bitstreams, while depending on the Processor to supply the configuration to the target FPGA, or target FPGA configurations could be stored there as well. Flash chips are able to store much larger amounts of configuration data, and this could translate to multiple upgrades or support for the largest FPGAs.

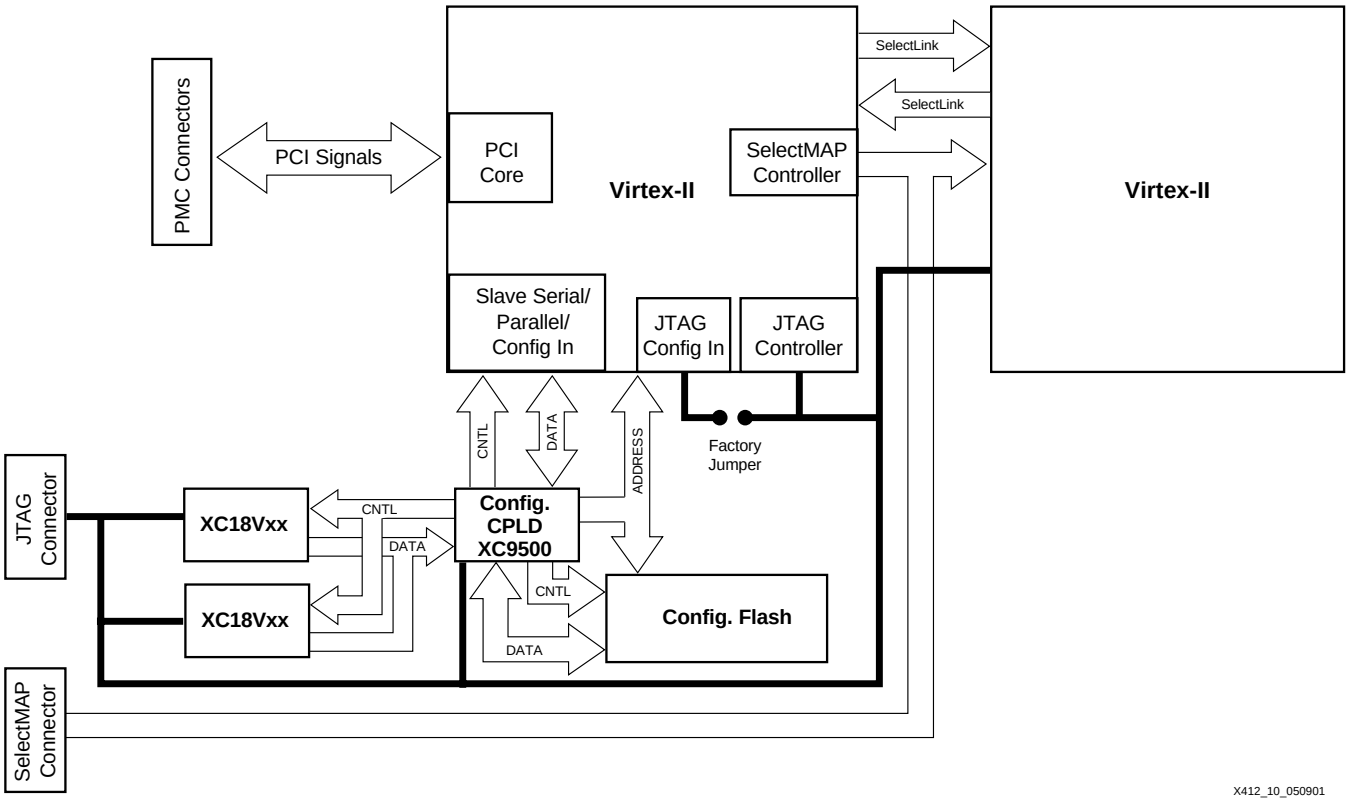

Figure 10: **PMC example with Bridge, PROMs, and Flash**

In this case, the CPLD is considered to be a thin device, basically a data mux with the majority of the logic in the FPGA. The address lines feeding from both the FPGA or CPLD to the flash chip would allow it be controlled from either chip.

In this example the default could reside in the Configuration flash or in a PROM; thus no jumpers are shown on the PROMs. If this is the case, it's the responsibility of the system designer to ensure fail-safe operation.

Use of PAVE in IRL Systems

The PAVE Framework is an embedded applications software development framework that can be employed to facilitate the development of reconfigurable embedded applications.

Object Oriented Hardware

The PAVE Framework and its components are a collection of C++ classes and object models that abstract an implementation of a Xilinx FPGA, called the IRL-enabled Device implementation. PAVE treats the programmable hardware as an object within the system, similar to software objects used in C++. As a result, applications that are written using PAVE tend to be highly object oriented, modular, and extremely upgradable. You can change a single module without replacing the whole framework.

SelectMAP and JTAG support

For PAVE 1.0, the programming interfaces supported are SelectMAP and JTAG, via the configuration register contained in your design, typically in the bridge. When compiling the design under the PAVE, you define the location of this and any other user registers in the device memory map. PAVE will encapsulate this programming interface and generate C++ source and header files and associated project files based on your design definition.

Available Development Platforms

Several development platforms that can be used for IRL are available today:

Motorola

The Motorola MCP750 SBC has the following features:

- MPC750 Power PC processor
- A PMC slot
- Ethernet connection
- Compact Flash

Motorola Computer Group can be contacted at:

http://www.mcg.mot.com

Alpha Data

The Alpha Data ADM-XRC is a PMC card that allows reconfiguration of the FPGA across a bridge. Details can be found at:

http://www.alphadata.co.uk/dsheet/adm-xrc.html

WInd River Systems

Wind River Systems makes the Tornado-II RTOS development platform.

http://www.windriver.com

Xilinx

Xilinx offers IRL training and the PAVE Framework.

http://www.xilinx.com/xilinxonline

Summary

With minor hardware and software changes, you can enable your systems for IRL and add much value for both you and your customers. The addition of IRL to your product will extend it's life and simplify support and distribution models. With IRL, you could manufacture a single physical version of your hardware and ship multiple different hardware versions. And your customers will appreciate the speedy, hassle-free upgradability of your products.

The Xilinx PAVE Framework provides a powerful software framework that allows designers to easily integrate IRL into their designs. The object oriented nature of PAVE eliminates the need to handle low level issues with JTAG or SelectMAP programming, allowing the designer to focus on the end-user's application.

Future revisions of the PAVE Framework will bring additional functionality to your IRL-enabled design. The modular nature of PAVE will allow you to add new features without disturbing your current application framework.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
6/29/01	1.0	Initial Xilinx release.