

The Importance of RTOS in Designing Network Upgradable Systems

There are both engineering and business merits in building a field upgradable product. Xilinx, the leader in programmable logic and Wind River the leading provider of commercial RTOS have teamed up to provide a systems approach for creating Internet Reconfigurable Logic applications. The basic elements in this joint effort are the RTOS from Wind River and an application program interface specifically designed to leverage the RTOS and allows designer to create reliable reconfiguration software specific for their target applications.

THE NEED FOR INTERNET RECONFIGURABLE LOGIC (IRL™)

There are both engineering and business merits in building a field upgradable product. As an example, consider a telecommunications product that shares its network with many different devices. Even though the engineering team thoroughly tests a product against a suite of test vectors in the lab, when the product enters the marketplace it may encounter unpredictable interactions with another vendors network device. For a product designed with a fixed function ASIC this means a new design spin. It is well known that errors or problems discovered at this late stage of development are the most costly to correct.

Basing a product on reconfigurable Xilinx FPGAs offers a level of protection in both functionality and investment. Issues discovered after fielding an FPGA product can be resolved with hardware upgrades, applied in a manner similar to software upgrades. By combining reconfigurable logic and a real-time operating system (RTOS) a designer can meet the demands of a network upgradable product. Network upgradable hardware is called Internet Reconfigurable Logic, IRL, by Xilinx. Xilinx, the leader in programmable logic and Wind River the leading provider of commercial RTOS have teamed up to provide a systems approach for creating IRL applications. The basic elements in this joint effort are the RTOS (VxWorks) from Wind River and an application program interface (API) specifically designed to leverage the RTOS and allows designer to create reliable reconfiguration software specific for their target applications. This API is called PAVE and is available from Xilinx. Let's first look at the RTOS component.

DIVIDE AND CONQUER WITH RTOS

An RTOS offers many features for enabling a robust field upgradable product. An IRL type application has multiple tasks to accomplish. These tasks can include the simple reconfiguration of the FPGA, the reception of new logic over the network, protection against network drop out, power failure, etc. When attacking these multiple tasks it is best accomplished by the "divide

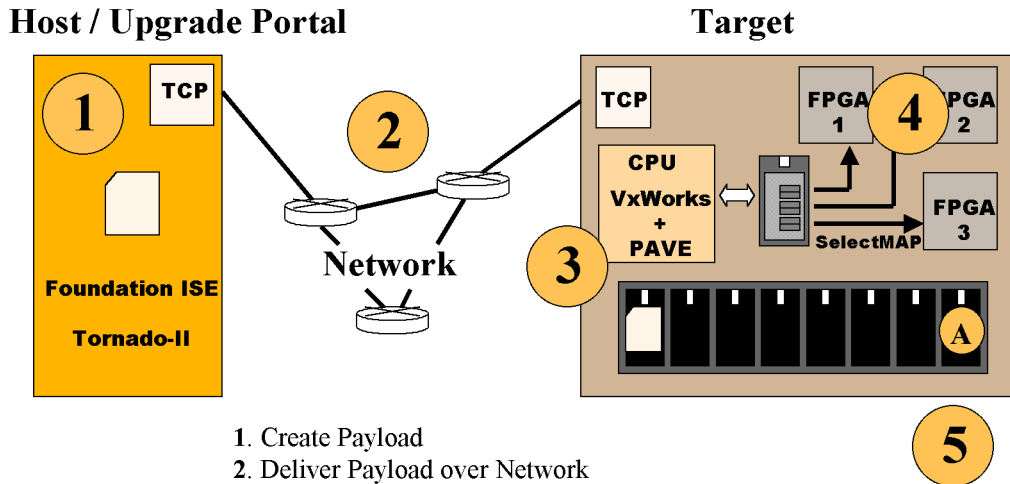
and conquer" approach which an RTOS provides. Instead of creating one large monolithic code segment, developers can partition a job into logical units that are individually tested and combined to provide a flexible solution.

Partitioning a hardware design across high volume commercially available components is an excellent strategy to achieve time to market requirements and avoiding the ever-increasing ASIC NRE costs. If a design includes Xilinx FPGAs even more flexibility is available to the designer. On the software side, an operating system is used to provide services and functionality over which a developer can partition his/her design. The concept of design partitioning is not new. Engineers, of all types, commonly draw block diagrams to break apart a problem into a set of manageable subsystems.

It is experience, attention to detail, and good mentoring that provides the designer with skills to sensibly partition a system. The tools used to achieve this mastery also play an important role in an engineer's skill set and the quality of the resulting products. Though not generally thought of as a tool in the same sense as compilers or debuggers, an operating system is an extremely important tool that aids design partitioning. An RTOS like VxWorks provides a multi-tasking environment with a variety of inter-task communication mechanisms, interrupt management, file systems, and networking support. Each of the RTOS features allows a product to be partitioned using a known set of pre-tested code. Constructing an upgradable system presents many opportunities to include RTOS provided functionality.

ANALYZING THE UPGRADE SEQUENCE

Let's consider the upgrade sequence for a typical upgradable system. This system consists of a Host where payloads are held in some sort of database and delivered to the upgradable Target system over a network. The payloads are used to upgrade both hardware and software components within a system. Before beginning our journey tracing the path of a pay-



1. Create Payload
2. Deliver Payload over Network
3. Validate Payload
4. Upgrade Target System (Software & FPGAs)
5. Provide for Rollback to Know Good Configuration "A"

Figure 1. Block diagram of upgrade sequence.

load, we need to see how payloads are created. In this example we use Wind River System's Tornado-II development environment and Xilinx's Foundation ISE tools to create the software and hardware portions of the payload. The compilers, debuggers, and simulators within Tornado-II are used to create loadable software modules.

The upgradable hardware designs are created within the Foundation ISE tools through VHDL/Verilog design entry and simulation, synthesis, place and route, followed by generating the FPGA bitstream. This generated bitstream along with the loadable software module are combined to create the payload. Now that a payload exists we store it on the Host system until it is required by the Target system. Payloads can be kept in any structure that allows for future retrieval. This structure varies from the directory structure imposed by a file system to a mirrored transaction oriented database. The amount of complexity introduced at the Host depends on the redundancy, journaling, and reporting capabilities desired. An RTOS could be used at the Host to provide determinism for transaction processing, but usually a soft real-time OS is deployed with multiple processors at the Host.

SOCKET LEVEL COMMUNICATIONS

Having the ability to store and retrieve payloads at the Host we need to communicate with the Target system to initiate or fulfill upgrade requests. Though many communications choices exist, selecting a TCP/IP or UDP based protocol leverages the considerable body of programs supporting the internet. (This is the method used by PAVE as well.) This choice also makes it likely that an RTOS will provide a supported communications stack. VxWorks for example provides support for FTP, TFTP, SNMP, direct socket connections and a variety of other network protocols. Developing the Target side communications code to interact with the Host is simplified by using RTOS communications APIs.

Once received by the Target, the payload undergoes several phases of checks before being incorporated by the Target e.g. payload decompression, decryption, and authentication. By using a prevalent RTOS the application code base supporting the RTOS is large, permitting the designer of an upgradable system a selection of off-the-shelf solutions. When an embedded system's RTOS is home-grown, each validation phase listed above entails a development cycle if the proprietary OS doesn't already contain the functionality. More importantly, a widely deployed commercial RTOS will be tested and ironed out over a larger user base, leading to a more robust system.

MANAGING NON-VOLATILE STORAGE

Once the payload has passed all the required checks, it must be stored on the Target system until the proper time to apply the upgrade occurs. Here again we will use the RTOS to perform the desired actions. Storage of the incoming payload is handled by the Target's file system. This file system may be on a local RAM disk, a local hard drive, flash file system or a networked disk. Keeping a known good copy of the component being upgraded in non-volatile storage is a necessary strategy for a rollback in the event of an upgrade failure. The RTOS makes accessing the upgrade payload as simple as opening a file regardless of where the file system is located. As to when the upgrade occurs, application code uses OS services.

An RTOS by its nature provides extremely low-level timing control of applications and inter-task communications. Applying an upgrade implies the ability to shut-down or drive the application being upgraded into a state where it can accept the upgrade. VxWorks provides application code with the ability to pass user defined messages among tasks through message queues. Once the running task receives a "Shutdown for Upgrade" message it begins its cleanup process and prepares to be replaced. Replacing code, calls

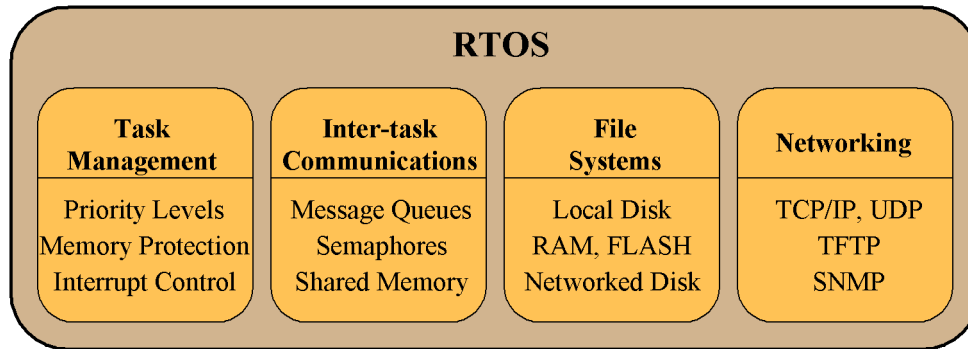


Figure 2. RTOS functionality useful for partitioning upgradable system design.

upon the RTOS's ability to dynamically load a software module. Once loaded, memory protection within an RTOS is used to insure a misbehaving upgrade doesn't crash the Target system. It is highly desirable that code monitoring the upgrade process remain inviolate. An RTOS such as VxWorks makes this possible. Semaphores are another RTOS provided mechanism used to synchronize events to insure an orderly upgrade. One task can "block" on a semaphore, waiting for a second task to place the semaphore in a state which "unblocks" the first task allowing it to resume its course of action.

On the topic of blocking we have to be careful not to end up in a deadlock situation where all tasks involved are blocked waiting for each other. Preventing this deadlock through careful coding may not be enough to avoid this situation. Most RTOSes allow tasks to be assigned a single priority level from a range of levels. Multiple tasks with differing priorities compete for processor cycles and it is possible for a lower priority task to block out a higher priority task in a situation known as priority inversion. An RTOS such as VxWorks can automatically raise the priority of a low priority task to that of the waiting higher priority task, allowing the lower priority task to relinquish the resource the higher priority task is blocked on, thereby allowing the higher priority task to get the processor. (See Figure 2)

RAPID DEVELOPMENT ENVIRONMENT

In order to make it easier to leverage these RTOS features and to actually program the FPGA, the PAVE API provides a common development framework for these applications. And the combination of VxWorks and PAVE allows an application developer to directly reconfigure Xilinx FPGAs through the SelectMAP or JTAG ports of the FPGA. The low level bit manipulation necessary to program the FPGA is taken care of by PAVE, unburdening the developer of explicit knowledge of the signaling at the hardware level. Performing the hardware upgrade and incorporating new functionality within the Xilinx FPGA by abstracting the hardware into an object method invocation or function call is a powerful concept.

We have followed the journey of a payload from Host to Target and on our way encountered many opportunities to invoke time saving services of an RTOS as well as the PAVE API. Building a network upgradable

product with Xilinx FPGAs and VxWorks , gives a designer the ability to rapidly respond to internal marketing requests as well as competitive product placements ■

This article is a co-production of Punit Kalra from Xilinx and Stuart Newton from Wind River.

Punit is the Technical Marketing Engineer within the Xilinx Internet Reconfigurable Logic (IRL) Solutions group. He holds a BSEE from the University of Illinois at Urbana-Champaign and an MSEE from the University of Colorado at Boulder.

Dedicated Systems
Encyclopaedia

<http://www.dedicated-systems.com>

Dedicated Systems
Encyclopaedia

**RTOS BUYER GUIDE ONLINE
&
CALENDAR OF EVENTS**

at the Dedicated Systems Encyclopaedia web site:

<http://www.dedicated-systems.com>