Runtime Filesystem Support for Reconfigurable FPGA Hardware Processes in BORPH

Hayden Kwok-Hay So
Dept. of Electrical and Electronic Engineering
University of Hong Kong, Hong Kong
*hso@eee.hku.hk**

Robert Brodersen
Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA94720, U.S.A.
rb@eecs.berkeley.edu

Abstract

This paper presents the design of BORPH's file system layer for FPGA-based reconfigurable computers. BORPH provides user FPGA designs that execute as hardware processes access to the general file system using familiar UNIX file I/O semantics. Such capability provides FPGA designers an intuitive interface not only for regular file I/O, but also for representing streaming hardware/software and hardware/hardware communication using UNIX pipes. Design trade-offs among system manageability, user usability and application performance are explored. A case of mixed hardware/software video processing is presented as a proof-of-concept.

1. Introduction

The Berkeley Operating system for ReProgrammable Hardware (BORPH) is a general purpose, multi-user operating system for FPGA-based reconfigurable computers[3, 4]. By extending various conventional software-centric UNIX semantics to reconfigurable computers, it aims to make FPGA systems more accessible to even non-FPGA experts. This paper presents a brief overview of the file system layer design of BORPH.

In a reconfigurable computer managed by BORPH, user FPGA applications, called *gateware* designs, execute as special *hardware processes*. A hardware process behaves in the same way as any normal UNIX process except its "program" executes on reconfigurable hardware instead of using up CPU time slices.

BORPH's hardware process concept allows gateware designs to execute independent of any controlling software. This enables an FPGA-centric compute model for reconfigurable computing, which also calls for a gateware-centric data I/O model. Among all types of gateware initiated data I/O, general UNIX file system access is one of the most important. In a UNIX system, many important entities such as sockets, pipes, and even device driver handles are all represented using the same file abstraction. Providing file system access to gateware therefore opens up many novel communication scenarios that are previously impossible when FPGA designs are run as mere accelerators of software.

Furthermore, BORPH extends the usual semantics of a UNIX pipe (FIFO) to represent hardware/software and hardware/hardware data stream. In particular, when two hardware processes are connected through a pipe, the kernel sets up direct connection such that they may communicate with cycle-accurate streaming data. The switching between regular file access and cycle accurate data streaming is handled by the kernel. Therefore, without recompilation, the same gateware design may engage in different modes of file access by a simple run time file redirection.

2. Design Issues

This section compares the differences between file system access by standard software and gateware designs.

2.1. High Speed Data Access

Even in a traditional software system, any kernel overhead on data I/O has significant impact on the over application performance. In the case of BORPH, since gateware designs run at "hardware speed," the effect of such software kernel overhead is much more prominent. It is especially true as the performance of most gateware designs are I/O bounded.

Our current implementation has partially addressed this problem by optimizing the case when two hardware processes are connected by a UNIX pipe. In that case, a direct hardware connection is setup between them to allow uninterrupted cycle-accurate data streaming.

2.2. Dynamic Switching Between Different File Access Modes

The optimization for hardware/hardware streams described in previous subsection creates a challenge to BO-

*This work was funded in part by C2S2, the MARCO Focus Center for Circuit & System Solutions, under MARCO contract 2003-CT-888.

RPH's interface design. In a UNIX system, during run time, the target of a file operation may dynamically be altered by means of I/O redirection.

For example, assuming `f1`, `f2` are both implemented as gateware, the following UNIX command will setup a direct high-speed connection between `f1` and `f2` that bypass OS intervention.

```
bash$ decode video.in | f1 | f2 > video.out
```

However, say for debugging purposes, one may save the output of `f1` easily using I/O redirection as follow:

```
bash$ decode video.in | f1 > debug.out
```

BORPH shields the user from all complexities involved in switching between the two modes described above. Therefore, without recompilation, the same gateware design may engage in either mode of file operations, making it easy to use on one hand, yet not degrading performance on the other hand.

2.3. Blocking vs. Nonblocking File Access

In standard UNIX systems, during the period in which the OS kernel services a file access system call, the calling user process is usually blocked until the system call commences. At that time, the calling process is resumed at the point where it was suspended, having the illusion that the system call has completed instantaneously. However, unlike software programs, a gateware application exhibit a parallel computation model in which every part of the design execute concurrently. Therefore, it is unwise for the kernel to block the entire hardware process using techniques such as clock gating because it will inevitably result in unacceptable performance penalties.

As a result, all hardware system calls are executed asynchronously to the user gateware designs in our current implementation. User designs must handle signals from the kernel that indicate the progress of a system call.

2.4. Concurrent File System Access

Since gateware designs compute in parallel, it is possible that multiple portions of the same design demand OS services concurrently. As oppose to the "logical" concurrent system calls generated by multi-threaded software applications, gateware designs may issue file system requests on the exact same hardware cycle. Therefore, the OS kernel must physically be able to service such concurrent requests.

2.5. Partial Result and Error Status

The semantics of standard software file read/write system call state that the OS kernel may return less data or commit less data than what the user has requested. It is useful when the file doesn't have enough data to return, or when the user's disk quota has exceeded. Furthermore, the kernel may return status code that indicates situations such as I/O error or reaching the end of a file.

While such concept of partial data delivery and error reporting is common among software developers, they are seldom considered by gateware designers. Our current implementation retains such software semantics and mandates gateware deigns to handle them gracefully similar to the ways software programs handle them. A set of user-space gateware libraries is provided to hardware-centric designers to ease their development efforts.

3. Example

As a proof-of-concept design, we have implemented various video processing filters using FPGAs on a BEE2 platform[1]. These gateware filters accept the common video streaming format used by the Linux MJPEG tools[2]. As a result, we were able to dynamically construct video processing chains during run time by mixing-and-matching gateware and unmodified software filters from the MJPEG tools as follow:

```
bash$ lav2yuv test.avi | yuvedgdet.bof \
      | mpeg2enc -o output.mpg
```

where `lav2yuv` and `mpeg2enc` are unmodified Linux software and `yuvedgdet.bof` is implemented in FPGA.

4. Conclusion

In this paper, we have presented an overview of the design of BORPH's file system layer. Providing file system access to gateware not only provides an easy-to-understand I/O interface for novel FPGA designers, but it also presents a novel FPGA-centric communication model that is only made possible by BORPH's hardware process model. A prototype system has been built that demonstrated the feasibility of constructing complex applications at run time by combining unmodified Linux software with gateware designs using UNIX pipes.

References

[1] C. Chang, J. Wawrzynek, and R. W. Brodersen. BEE2: A high-end reconfigurable computing system. *IEEE Design & Test*, 22(2):114–125, 2005.

[2] mjpegtools. [Online] http://mjpeg.sourceforge.net.

[3] H. K.-H. So and R. Brodersen. A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. *Trans. on Embedded Computing Sys.*, 7(2):1–28, 2008.

[4] H. K.-H. So and R. W. Brodersen. Improving usability of FPGA-based reconfigurable computers through operating system support. In *Proc. FPL'06*, pages 349–354, 2006.