HARDWARE AND SOFTWARE DEVELOPMENT AND INTEGRATION IN AN FPGA EMBEDDED PROCESSOR BASED CONTROL SYSTEM MODULE FOR THE ALS*

J. Weber, M. Chin, C. Timossi, E. Williams, LBNL, Berkeley, CA 94720, U.S.A.

Abstract

The emergence of Field Programmable Gate Arrays (FPGAs) with embedded processors and significant progress in their development tools have contributed to the realization of system-on-a-chip networked front-end systems [1]. Embedded processors are capable of running full-fledged Real-Time Operating Systems (RTOSs) and serving channels via Ethernet while high speed hardware functions, such as digital signal processing and high performance interfaces, run simultaneously in the FPGA logic [2].

Despite significant advantages of the system-on-a-chip implementation, engineers have shied away from designing such systems due to the perceived daunting task of integrating software to run a RTOS with custom hardware. However, advances in embedded development tools considerably reduce the effort required for software/hardware integration.

This paper will describe the implementation and integration of software and hardware in an FPGA embedded processor system as illustrated by the design of a new control system module for the Advanced Light Source (ALS) at Lawrence Berkeley National Lab (LBNL).

INTRODUCTION

One of the greatest difficulties faced by the accelerator control system (CS) is in devising a scheme for interfacing the CS to the various types of accelerator instrumentation. The natural tendency is to mandate a "supported" interface, compelling the instrumentation engineer to create a compatible design that may not be optimized for the instrumentation task. In addition, organizational boundaries often exist between the control system group and instrumentation engineers (who may be outside vendors) that increase the chance of hardware compatibility problems during commissioning. There is general agreement, however, that the instrumentation and the attached CS element should be co-located.

One of the most common interface strategies is to use a bus-based system (e.g. VME, PCI). Typically a crate is deployed containing a single-board computer (SBC) card with an Ethernet interface to the control system, and interface cards for analog and digital I/O. More integrated designs may include a custom motherboard with a processor module and I/O modules connected via an onboard bus, or even designing the processor and I/O chips onto a single board. However, as the performance demands on instrumentation interfaces increase, the bus-based interface becomes a bottleneck in the data transfer chain from instruments to the CS. With the evolution of FPGAs with embedded processors, the combination of high speed converter interfaces, high speed data processing, and control system interfaces can now be realized in a single chip solution.

As recently as 5 years ago, the embedded design tools for implementing a processor-based FPGA design were quite primitive and difficult to use. The learning curve for embedded design tools and the lack of practical examples deterred engineers from including this powerful technology in their designs. The progression of these tools, example designs, and vendor support has led to the development of highly integrated design processes, including pushbutton board support package (BSP) generation for several RTOSs, extensive libraries of standard peripheral controllers (i.e. Ethernet, RS-232, CoreConnect™ buses, serial interfaces, etc.), and templates for designing bus compatible custom peripheral controllers (cores).

The ALS Control Systems and Instrumentation groups collaborated to create the ALS Mini IOC [3] based on the FPGA embedded processor technology, including an EPICS [4] control system interface integrated with magnet power supply control hardware.

ENGINEERING COLLABORATION

From previous experience designing the PEP-II Transverse Feedback Electronics [5], it was clear at the outset of the Mini IOC design that the ability of the control systems group and the instrumentation group to collaborate effectively would be critical to the success of this project. For this to work, tasks had to be divided between hardware and software engineers efficiently. Figure 1 shows the final development flow for the Mini IOC. In this figure, hardware and firmware tasks are shown in orange, specification documentation in yellow, and software development and configuration in green. When used, development tools are shown in parentheses. The arrows indicate dependencies of tasks on the completion of other tasks. The starred tasks are not necessarily required depending on available system memory, and will be discussed further in the "Bootloading Control System Software" subsection of this paper.

Design Process

The design process began with the board level design, factoring in requirements of the control system interface specification. For the Mini IOC, this task included

*This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231

selecting the FPGA (a Xilinx® Virtex™-4 with embedded PowerPC®), I/O interfaces, and other physical hardware [3]. Custom cores had to be designed in the FPGA firmware to control the selected hardware, and a specification was written on how to control them from the processor. The custom cores and standard cores were then integrated into the top level design and connected to internal buses to make them accessible to the processor. An initial build of the RTOS kernel (VxWorks) for the customized hardware provided a starting point for the software development effort.

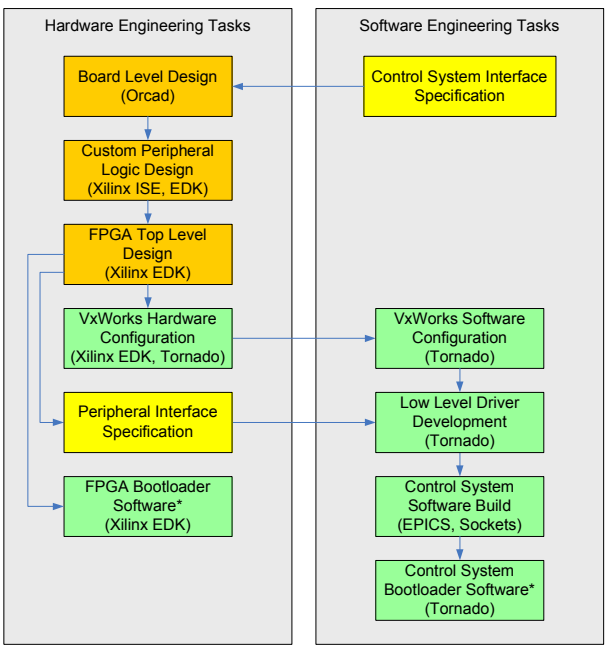

Figure 1: Division of Mini IOC design tasks between hardware and software engineers.

While software engineering worked towards a final VxWorks kernel and low level drivers, hardware engineering developed bootloader software that would start on FPGA power-up. The tasks of designing custom peripherals through writing low level drivers for them was iterated as each custom peripheral was developed so that the software engineering development could begin before the firmware design was complete. Once all the low level drivers were complete, they were integrated into the EPICS control system interface.

DESIGN STRATEGY

A significant goal of this project was to create a CS friendly instrumentation platform that could be leveraged for future designs [6]. Rather than standardizing the physical hardware, we created a virtual base design that consists of a standard control system interface and a subset of standard peripherals (i.e. Ethernet, RS-232, memory controllers) with a pre-defined method for controlling custom peripherals. This virtual design can be merged with a customized hardware/logic design that is optimized for specific instrumentation tasks. Figure 2 illustrates the firmware design of the Mini IOC. Cores shown in yellow were provided by Xilinx®, blue cores were designed by LBNL using Xilinx bus interface templates.

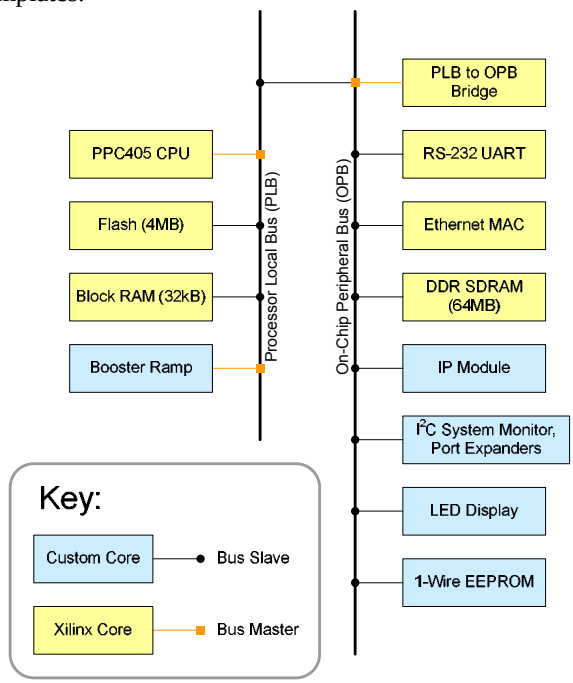

Figure 2: ALS Mini IOC FPGA firmware block diagram.

In an embedded processor based design, one of the challenges is to merge control system requirements and instrumentation requirements efficiently into an FPGA with shared resources. For the Mini IOC, this meant merging an EPICS control system interface with a booster ramp controller including four channels each executing a ramp table of 10,000 16-bit points in 100us intervals [3].

Embedded Resource Management

Since hardware and software must coexist in the FPGA, careful attention must be paid to resource allocation and the interconnection of the various system components to avoid collisions. However, the flexibility of embedded systems allows designers to consider multiple implementations of some system functions, enabling effective resource management at design time and into the future.

In the case of the Mini IOC, one resource allocation that underwent changes during the design process was the storage of the booster ramp tables. The preliminary design specification required two ramp channels, each with a 10,000 point ramp table of 16-bit values, which required 40kB of memory. Since the selected FPGA contains about 70kB of block RAM, this memory was initially selected for ramp table storage. This left nearly half (about 30kB) of the block RAM available for bootloading software and other logic functions. Moreover, block RAM is a flexible resource and could be easily incorporated into the booster ramp controller logic.

As design requirements were finalized, this specification changed to four 10,000 point ramp channels (for future expansion), with the ability to download a new

ramp table to each channel before the next ramp cycle (once per second, maximum). To accommodate the ramp table download, we chose to store a secondary ramp table for each channel, doubling the amount of ramp table data. With the final specifications, the Mini IOC now needed to store 160kB of ramp table data—four times the preliminary requirement, and more than could be stored in the FPGA block RAM.

The large off-chip DDR SDRAM (64MB) had plenty of memory available to store both control system software and ramp tables. The drawback was that the booster ramp controller would have to contend with other activity on the bus and compete with the processor for access to the SDRAM. Testing was done to measure the impact of this additional load on the bus bandwidth and processor performance, and it was determined that the impact was minimal, since the transfer was only 8 bytes of ramp table data every 100 us. As a result, we were able to accommodate changes in the specification without modifying the physical hardware or the control system interface.

SOFTWARE DEVELOPMENT

The main challenge in software development is generating a custom BSP and building a properly configured RTOS kernel that runs on the embedded platform. The Xilinx® Embedded Development Kit (EDK) supports automatic BSP generation for VxWorks, and a Xilinx® tutorial [7] describes how to configure VxWorks to run on the embedded PowerPC®, which allowed us to get VxWorks up and running quickly on the Mini IOC. Some additional kernel configuration was required to support EPICS on VxWorks. A simple sockets interface was used to download ramp table data.

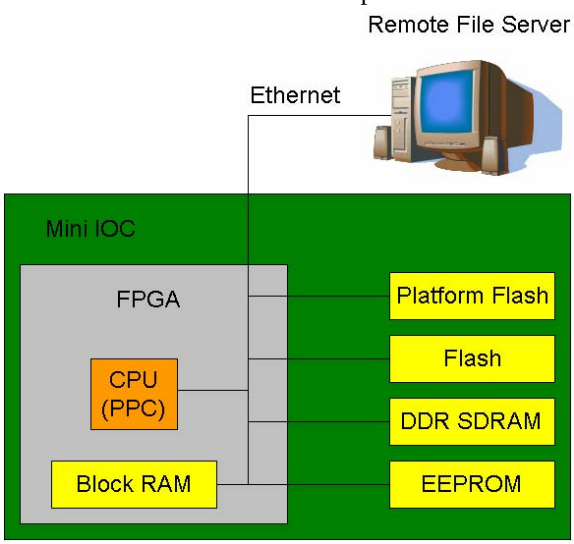

Figure 3: ALS Mini IOC Software Bootloading Components.

Bootloading Control System Software

Due to limitations of the memory available in the system, the Mini IOC requires a complex multi-step boot process to configure itself for Ethernet communication and to start EPICS. Figure 3 shows the connection of components in the boot sequence.

After power-up, the FPGA loads it's configuration from the Platform Flash, which includes the EDK bootloader, and begins executing it. The bootloader copies the VxWorks bootrom from Flash to DDR SDRAM and begins execution. The VxWorks bootrom reads unique Ethernet configuration parameters, such as the MAC address, from the EEPROM, and configures the Ethernet interface. The bootrom then obtains additional IOC-specific parameters from the BOOTP server based on its MAC address. This extra step allows us to swap Mini IOCs for servicing without having to reconfigure their Flash by simply updating the central BOOTP database. The bootrom then downloads the VxWorks kernel (common to all Mini IOCs) from the remote file server to SDRAM and starts VxWorks. VxWorks executes a startup script to download EPICS code, and generic and IOC-specific database settings from the remote file server. EPICS is then started to perform the IOC's control functions.

CONCLUSION

FPGAs with embedded processors allow designers to integrate CS and instrumentation requirements into a single chip using a virtual design template. Employing this template as a standard for accelerator CS instrumentation interfaces allows greater flexibility in hardware designs. Accelerator CS and Instrumentation groups must collaborate on the embedded design for this strategy to work effectively. A benefit of the initial collaboration is that the experience gained by both groups can be leveraged towards future designs.

REFERENCES

[1] L. Doolittle, "Embedded Networked Front Ends – Beyond the Crate," ICALEPCS'03, Gyeongju, Korea, October 2003.

[2] J. Weber, M. Chin, "Using FPGAs with Embedded Processors for Complete Hardware and Software Systems," BIW'06, Batavia, IL, May 2006.

[3] J. Weber, et al, "ALS Mini IOC: An FPGA Embedded Processor Based Control System Module for Booster Magnet Ramping at the ALS," these proceedings.

[4] L. R. Dalesio, et al, "The Experimental Physics and Industrial Control System Architecture," ICALEPCS'93, Berlin, Germany, 1993.

[5] J. Weber, et al, "PEP-II Transverse Feedback Electronics Upgrade," PAC'05, Knoxville, TN, May 2005.

[6] M. Chin, C. Timossi, "The Use of FPGAS as a Platform for Distributed Control Systems," ICALEPCS'05, Geneva, Switzerland, October 2005.

[7] "ML310: Creating a VxWorks BSP and System Image from the Base Design," http://www.xilinx.com.

