A Comprehensive Prototyping-Platform for Hardware-Software Codesign

Andreas Koch
Tech. Univ. Braunschweig (E.I.S.), Gaußstr. 11, D-38106 Braunschweig, Germany
koch@eis.cs.tu-bs.de

Abstract

We present a flexible, yet cost-effective prototyping platform for hybrid hardware/software systems. Our approach is based on combining off-the-shelf hardware components with custom software to arrive at an encompassing solution. We address the hybrid nature by tightly coupling a conventional processor with configurable logic on a single PCI expansion card.

1. Introduction

One of the main difficulties in building and evaluating the hybrid solutions created by hardware/software codesign methods is their *systemic* nature: The designer is no longer faced with designing, implementing, and testing a single chip or a single program, but must consider the interplay between numerous interdependent hardware and software components. Simulating such a system is often not feasible because either the required simulation models are not available or the complexity of the resulting encompassing simulation model is so high that the simulation run-times themselves are no longer practical.

In many cases, the required level of detail can only be observed by actually prototyping a sufficiently large part of the system. Due to the this step being on the critical path of a product introduction, techniques for completing this phase as quickly as possible become crucial. While past technology generations could be easily tested using breadboard assemblies, this is no longer practical with current large systems. With the advent of Field-Programmable Gate Arrays (FPGAs), the current generation of prototyping systems [1] [2] is able to emulate circuits of up to 20 Mgates at a cost of \$0.55 to \$0.89 per gate [3].

While these emulators allow the rapid prototyping of very large systems, they are economically infeasible for smaller design teams and do not address the problem of efficiently executing the interplay between hard- and software (they lack conventional on-board processors). An approach that is far less costly, but that would still allow the seamless prototyping of such a hybrid system, is quite desirable.

2. Solution

Our solution for these requirements leverages state-of-the-art FPGA technology to reach the gate capacities needed for practical testing. This dispenses with the need of partitioning a larger circuit across a sea of smaller FPGAs and the resultant increase in complexity and speed. Thus, we can achieve logic capacities in the 1-3 Mgate range for \$0.008 to \$0.01 per gate.

To cover the software angle, the prototyping platform must contain a sufficiently powerful conventional microprocessor that is tightly coupled to the reconfigurable logic. In order to easily implement software on the system, code running on this CPU must have access to a full set of OS resources (e.g., C library, memory management, etc.), but must be unencumbered by OS constraints that would hinder access to the hardware (convoluted driver models, high interrupt latencies).

3. Hardware Architecture

Instead of custom designing an architecture fulfilling these requirements (as we did before, e.g., in [4] [5]), the hardware of our current prototyping environment is composed by combining two components off-the-shelf (COTS), an approach that makes it applicable to a wide variety of prototyping scenarios. Each of the separate parts adds features critical for arriving at an encompassing solution. Figure 1 shows a schematic of the major hardware features.

3.1. ACE2card

The Lavalogic (formerly known as TSI TelSys, Inc.) *ACE2*card (shown in the lower half of Figure 1) was initially designed to act as a key component in satellite communications equipment. To allow reuse of the same hardware when dealing with different communications protocols, the card offers sufficient configurable gate capacity to accommodate the timing-critical portions of a variety of protocols.

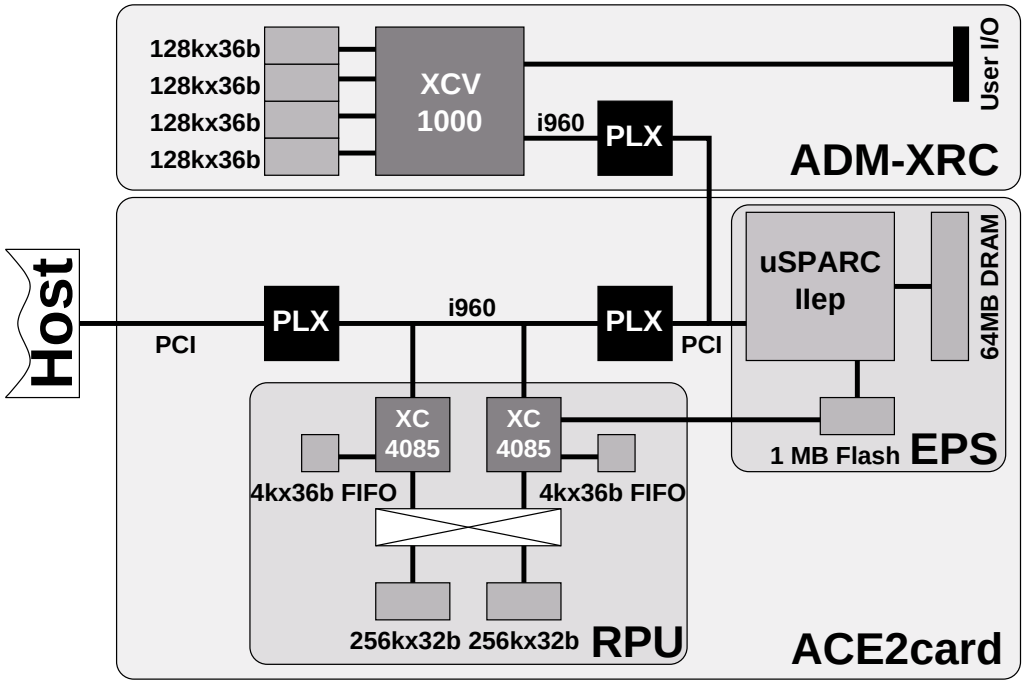

Figure 1: Hardware architecture

The major distinguishing feature between the *ACE2*card and other FPGA-supporting platforms, e.g., [6] [7], is the on-board presence of a conventional RISC processor. This tight integration allows the realistic prototyping of hybrid hardware/software solutions, unencumbered by the overhead of relying on the host computer for software execution.

One of the main components of the *ACE2*card is a conventional computer, called the Embedded Processor Subsystem (EPS). It its is based on a SUN microSPARC-IIep processor [8], an implementation of the SPARC V8 specification [9]. This RISC runs at 100 MHz and has access to 64 MB of EDO DRAM and 1 MB of user-programmable Flash memory (holding the boot firmware). Additionally, the chip also provides a 33 MHz 32-bit PCI master interface (including interrupt management), an implementation of the SPARC reference MMU, and various real-time timers and counters.

The PCI bus is used to communicate with a PMC [10] expansion connector and the on-board reconfigurable processing unit (RPU). On the *ACE2*card, this consists of two Xilinx XC4085XL [11] FPGAs having a capacity of up to 170,000 gates. Each of the FPGAs has access to a dedicated 256k x 32b bank of SRAM memory. Using a fast crossbar, the banks can be switched between the FPGAs on a per-cycle basis. Also available are four 4k x 9b FIFOs per FPGA to use as temporary buffers, e.g., in stream-based computation.

In order to simplify the user logic, the RPU and the EPS do not communicate directly over the PCI bus (complex protocol, multiplexed data and address lines, fixed clock speed). Instead, a PLX 9080 [12] PCI I/O accelerator bidirectionally converts the PCI bus to/from a non-multiplexed 32b bus similar to the one used on the Intel i960 processor. While this bus has a much simpler protocol (considerably easier to implement in user logic), it still achieves PCI performance levels. As an additional plus, the i960 bus (also called "local bus") runs asynchronously to the PCI bus at any speed from 500 kHz to 33 MHz, thus matching the design-dependent FPGA clock-speeds to the fixed PCI clock.

The *ACE2*card attaches to the host as a full-length PCI card. The host PCI bus is converted to the local bus using another PLX 9080. By appropriately configuring the PLX registers, transparent access from the host is possible not only the RPU, but also to the EPS.

While the *ACE2*card in itself already provides a very useful platform for evaluating HW/SW codesigns (especially due to the tightly coupled processor), and is in fact already being used as target for an experimental fully automatic HW/SW compile-flow [13], it has limitations in its original form: The XC4085XL FPGAs are no longer close to the state-of-the-art in FPGA architecture. They are limited with regard to sheer logic capacity (current FPGAs reach up to 3.2 Mgates [14]) as well as to configuration speed and partial configuration ability. The last two features are

important when using FPGAs as flexible compute engines. For this application, the lack of busmastering capabilities for the FPGAs on the *ACE2*card is also annoying. E.g., in the experimental compile flow, data has to be copied explicitly to the RPU SRAMs before it can be processed by the FPGAs. A more homogeneous memory model allowing the RPU direct access to the entire memory space would be desirable instead.

3.2. ADM-XRC

The Alphadata ADM-XRC card is a daughtercard following the PCI Mezzanine Card (PMC) standard [10]. It is shown in the upper half of Figure 1 and uses the *ACE2*card as a motherboard.

In contrast to the *ACE2*card, the ADM-XRC concentrates on providing a state-of-the art Xilinx Virtex or VirtexE FPGA connected to four fast memory banks. Our current configuration uses an XCV1000 FPGA with a capacity of up to 1 million gates accessing four 128k x 36b banks zero-bus latency (sometimes also called "zero-bus turnaround") SRAMs.

As with the RPU on the *ACE2*card, the ADM-XRC relies on a PLX 9080 to convert the PCI bus to the simpler i960 bus. Now, however, the FPGA has full master access to the bus and can transparently access data residing, e.g., in the EPS DRAM. As before, this arrangement also enables the asynchronous operation of the variable-clock speed FPGA from the PCI bus. For further extension or debug connections, the ADM-XRC also offers 34 pins of user-programmable IO in the form factor of a SCSI-2 connector.

For integration with the *ACE2*card, we had to develop code (running on the EPS) that attached the ADM-XRC to the PCI bus and mapped its memory regions into the microSPARC-IIep address space.

By combining both hardware components in this fashion, we obtain an off-the-shelf platform having the strengths of the *ACE2*card (embedded processor, easy access from host), and use the ADM-XRC to compensate its weaknesses (large logic capacity, state-of-the-art FPGA, homogeneous memory model).

4. Software Architecture

Hardware is only one part of a prototyping *system*, software (while often neglected) forms the other half. When assembling our prototyping environment, this was the area that required the most effort to arrive at a usable, tightly integrated solution.

For the *ACE2*card, the vendor offers host-side drivers (Solaris and Windows NT) that map the various devices (memories, PLXs, FPGAs) into the host filesystem. E.g., the EPS DRAM can be accessed using an `open()` system call, `write()` and `read()` calls will then exchange data between the DRAM and the host.

However, for applications actually executing on the EPS, the support was far more rudimentary: Their only means of communication used the PLX9080 mailbox registers to simulate four "virtual serial ports", which are then also mapped to host devices. While useful for debugging, these high latency and low bandwidth channels are unsuitable for any practical I/O needs. No support was included for such critical operations as FPGA access and interrupt processing on the EPS. Furthermore, not even the basic C library functions (memory management, math, signal handling, etc.) were available. These restrictions severely hindered the porting of conventional C code to the EPS.

Since the ADM-XRC is intended as a general purpose extension to all PMC-compatible environments, it did not include any *ACE2*card-specific software. Only small C fragments illustrating the transformation of Virtex bitstreams into a downloadable form and the actual configuration sequence were provided.

Our approach to removing these limitations is described in the next sections and illustrated in Figure 2.

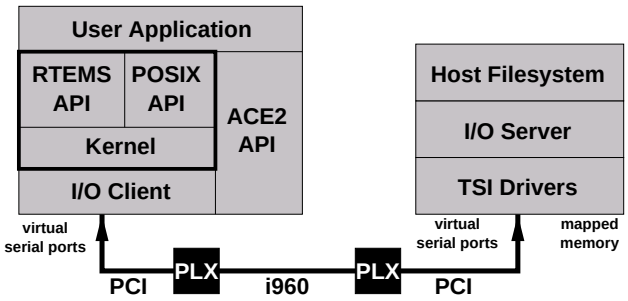

Figure 2: Software architecture

4.1. RTEMS

As a first step in presenting EPS applications with a more familiar and complete run-time environment, we ported the RTEMS operating system [15] to the EPS. RTEMS is a preemptive multi-threading real-time operating system freely available for a wide variety of boards and processors. It is sufficiently lightweight (e.g., no virtual memory, efficient direct hardware access, very short interrupt handling latencies) that it remains suitable for small embedded system. RTEMS furthermore includes a flexible model for I/O drivers and a POSIX-compliant standard C library.

The port to the EPS was facilitated by the fact that processor-level support for the SPARC V7 architecture was already in the RTEMS 4.0.0 code base. At the low level of

this base port, we had to add EPS specifics such as memory-management, cache control, interrupt handling, and real-time clock access. For testing, we mapped the RTEMS console to the *ACE2*card virtual serial ports.

At this stage, it was now possible to execute conventional C programs on the EPS. I/O, was limited to interaction on the virtual serial port, though.

4.2. Host I/O Access

While the limited I/O capabilities just described might be sufficient for small embedded systems, our aim of using the *ACE2*card/ADM-XRC combination as a target for automatic HW/SW compilation requires a higher degree of host integration. Specifically, many of the applications need transparent read/write access to files residing on the host filesystem.

While ad-hoc methods of transferring this data could be used, we implemented a reusable mechanism providing full access to host files and devices. It relies on a custom RTEMS driver that forwards all I/O operations on non-local devices to a server program running on the host.

This communication occurs by setting up a parameter block in the EPS DRAM and sending an I/O request to the host using one of the virtual serial ports. The I/O server is awakened and then uses the TSI host-side device driver to retrieve the parameter block from the mapped-in EPS DRAM. Next, the I/O operation is actually performed and any read data transferred back to the EPS through the shared memory.

In this manner, an application running on the EPS can access all data on the host (even devices, network mounted volumes, pipes, etc.). Furthermore, since all three of the standard I/O streams are also routed using this mechanism, it is even possible to transparently pipe data from a host application through the EPS and back to another host application without any user intervention.

4.3. Hardware API

Instead of simply reading and writing hardcoded memory locations for access to the FPGAs, a dedicated set of routines provides these operations in an easy-to-use and portable manner.

Among the operations supported are the decompression and fast loading of configuration bitstreams, the retrieval of address mappings for the FPGAs and their associated memories, and the locking and synchronization of FPGA-based computation with RTEMS threads. Additionally, the package also encapsulates board specifics such as interrupt handling and programming the variable clock for each of the i960 busses. All of these functions can operate regardless of whether the target FPGA is in the RPU or on the ADM-XRC.

4.4. Tools

Since the microSPARC-IIep of the EPS is fully compatible with other SPARC V8-based computers (e.g., the SUN SparcStation5), existing tool chains can be used to target the EPS with only slight adjustments. In our case, we are relying on the GNU suite of C compiler, assembler, and linker for the main flow. The resulting executables are then transformed into a binary format suitable for downloading to the *ACE2*card using the GNU binutils package. Our standard compile flow wraps the binary in an envelope that automatically starts the I/O server on the host, performs the download, starts the application, and establishes contact with the I/O client on the *ACE2*card. Thus, running a program on the EPS is accomplished transparently by simply typing its name on the host command line.

For debugging, one of the virtual serial ports on the EPS is used by the *ACE2*card firmware to implement the GNU GDB remote debugging protocol. In this manner, EPS applications can be comfortably debugged from the host using GDB and enhancements like DDD.

For hardware creation, we employ conventional logic synthesis tools starting from Verilog HDL or an experimental compiler translating C into a hybrid HW/SW application. In both cases, the Xilinx M2 EDA tools are used as a back-end for creating the bit-stream files, which are then compressed and converted into ELF object files that are directly linkable into the EPS application. This approach is preferable over the standard solution of converting the bit-stream files into a C program (hex dump) which is then compiled and assembled before linking: For a current medium-capacity FPGA such as the Virtex 1000, bitstreams are 770KB in length. The resulting C file using the conventional approach would thus have a length of ca. 3.8MB. Especially when multiple configurations need to be integrated in this fashion, the compile and assembly times become unacceptable. Compare this with our way of compression and ELF generation: A 770KB bit-stream is turned into a 12KB linkable ELF object in a fraction of a second. This bit-stream can be decompressed back into a format suitable for downloading in less than 100ms on the EPS.

5. Conclusion

In this work we described an approach to obtaining a very flexible platform for prototyping hybrid hardware/software systems. We achieve our aims of tight hardware/software integration, large hardware capacity, and ease-of-use through the combination of two off-the-shelf

hardware products with the addition of a powerful yet lightweight software layer. The system has proven very successful both as a conventional prototyping environment as well as the target for an automatic hardware/software compilation system.

All of the custom-developed software (e.g., PCI configuration code, RTEMS port, I/O system, hardware API, and tools) is available on request from the author.

References

[1] Cadence Design Systems, Inc., "CoBALT", `http://www.quickturn.com/products/cobalt.htm`, 1999

[2] Aptix Corp., "System Explorer MP4", `http://www.aptix.com`, 1999

[3] Goering, R., " Quickturn boosts emulation to 20M gates", EE Times, No. 1036, 23 Nov 1998

[4] Koch, A., Golze, U., "A Universal Co-Processor for Workstations" in *More FPGAs*, ed. by Moore, W., Luk,W., Oxford 1994, pp. 317-328

[5] Koch, A., Golze, U., "Practical Experiences with the SPARXIL Co-Processor", *Proc. Asilomar Conference on Signals, Systems, and Computers*, 11/1997

[6] Virtual Computer Corp., "H.O.T. II Development System", `http://www.vcc.com`, 1998

[7] Annapolis Micro Systems, Inc., "WILDSTAR High-Speed DSP Boards", `http://www.annapmicro.com`, 1999

[8] Sun Microelectronics, "microSPARC-IIep User's Manual", `http://www.sun.com/sparc`, 1997

[9] Weaver, D.L., Germond, T., "The SPARC Architecture Manual, Version 8", Prentice-Hall, 1992

[10] IEEE, "Draft Standard Physical and Environmental Layers for PCI Mezzanine Cards: PMC", IEEE Standard P1386.1, 1995

[11] Xilinx, Inc., "The Programmable Logic Data Book", `http://www.xilinx.com`, 1998

[12] PLX Technology, "PCI 9080 Data Book", `http://www.plxtech.com`, 1998

[13] Harr, R., "The Nimble Compiler Environment for Agile Hardware", *Proc. ACS PI Meeting*, *`http://www.dyncorp-is.com/darpa/meeting/acs98apr/Synopsys\%20for\%20WWW.ppt`*, Napa Valley (CA) 1998

[14] Xilinx, Inc., "Virtex-E 1.8V Field-Programmable Gate Arrays", `http://www.xilinx.com`, 1999

[15] On-Line Applications Research Corporation, "RTEMS - Real-Time Executive for Multiprocessor System", `http://www.rtems.com`, 1998