Gizgate: An Object-Oriented Gateway for Hardware/Software Codesign on the CMC Rapid-Prototyping Board

D. Sharp, W. B. Gardner and M. Serra

VLSI Design & Test Group, Dept. of Computer Science, Univ. of Victoria, Victoria, B.C.
{dsharp1, wgardner, mserra}@csr.uvic.ca

Abstract
The Canadian Microelectronics Corporation (CMC) has made available to Canadian universities a Rapid-Prototyping Board (RPB) for research into field-programmable logic and embedded systems, including hardware/software (HW/SW) codesign. Earlier, we developed IntraGiz, an object-oriented driver for the RPB’s FPGAs. Now this poster describes an additional software layer, Gizgate, which projects the HW/SW interface from the RPB onto remote workstations. It allows a remote program to utilize the FPGAs exactly as if it were running on the RPB’s microprocessor, thereby enabling any workstation on the network to utilize reconfigurable hardware.

1. Introduction

Canadian universities have received a new tool for research in configurable computing, the Rapid-Prototyping Board (RPB) [1,2]. As shown in Figure 1, the board features an MC68306 16-bit microprocessor (MPU) with 2Mb of RAM, and two Xilinx XC4010 reprogrammable FPGAs, each connected to 128Kb of RAM and a variety of off-board connectors. Uses envisioned for this research tool include prototyping of embedded system designs [4] and experiments in hardware/software codesign [3,6]. Since the MPU has large on-board RAM (expandable to 16Mb), dynamic reprogramming of the FPGAs from a number of stored configurations is also possible.

Each RPB user, or would-be codesigner, has to face the issue of managing the board’s HW/SW interface. From the SW standpoint, this means manipulating the MPU’s bus and parallel ports, which are wired to the FPGAs’ I/O pads. In order to facilitate this, we created an object-oriented driver in C++ for the MPU. This package, called IntraGiz [5], makes it easy for an MPU program to communicate with a HW configuration in the FPGA, and provides building blocks for interface cosynthesis.

The work presented in this poster builds on the IntraGiz driver layer. Simply put, it allows the RPB’s reprogrammable HW to be treated as a remote device in a distributed system, rather than just a local device bound to the RPB’s MPU. In effect, the HW/SW interface on the RPB is thus projected onto any remote workstation. This opens up a richer development environment around the RPB, with new possibilities for its use.
The new SW layer is called Gizgate.² It needs to be understood in the context of the IntraGiz driver, so IntraGiz is summarized first in Section 2. Section 3 then outlines the functionality of Gizgate, while Section 4 explains its internal design. The poster will incorporate diagrams from these three sections.

While the work presented here is geared for the CMC board, the concepts and design decisions are applicable to other HW/SW codesign environments.

2. IntraGiz: an object-oriented FPGA driver

IntraGiz was created for the same reason as most drivers: to make it easier and safer for software programs to use a hardware device. Instead of grappling with complex, error-prone low-level coding of 68306 bus and parallel port I/O, an IntraGiz user simply instantiates objects from its class hierarchy (see Figure 2) that represent physical components of the RPB’s HW/SW interface. These classes were intentionally created to provide primitive operations that are close to the hardware. The user can then choose between direct use of the primitives alone, or “layered use” based on creating higher level abstractions which in turn use the direct methods.

The following portion of code illustrates the use of some IntraGiz objects:

```c
try{
    FPGAbus bus;
    OutputPin flag( PA3 );
    InputPins code( PB7 | PB6 | PB4 );
    FPGArupt done( IRQ3, handler );

    bus(select, addr) << value;
    flag.set();
    code >> newCode;
    done.enable();
}
```

In this sample, C++ stream output syntax (<<) is used to send a 16-bit value to the FPGA via the `bus` object, at the same time setting the chip select and address lines to the values `select` and `addr`. Parallel port pin “PA3” has been set up as a flag for the FPGA, and the `set` method is used to place a ‘1’ on this line. The third object, `code`, represents a bundle of parallel port pins, PB4, PB5, and PB7, over which a 3-bit value is input. The `done` object specifies a function handler to be called for IRQ2 interrupts, enabled via the `enable` method. If IntraGiz detects a software usage error—e.g., PA3 is not wired for output, or if the arguments `select`, `addr`, and `value` were too large—it throws the `FPGAerror` exception.

A programmer accesses IntraGiz by including the class definitions from “IntraGiz.h” and linking with the libIG.a library of IntraGiz methods. GNU g++ version 2.7.2 or higher (with exception support) for the “m68k” target is used to cross-compile.

3. Gizgate: a distributed FPGA gateway

The function of Gizgate is to take the HW/SW interface that exists on the RPB and project it onto any network host, by means of the IntraGiz objects. The conceptual model for Gizgate is shown in Figure 3, with the IntraGiz-only model on the left. On the right side, a layer of communication software has been inserted between the user software and the IntraGiz methods on the RPB, so that the user software can be rehosted elsewhere. This is done in a transparent fashion, leaving the “virtual execution path” intact.

To be specific, an off-board user includes the same “IntraGiz.h” file and writes precisely the same program that would work directly on the board. The program is compiled with g++ for the host computer, and linked with Gizgate, rather than IntraGiz, methods. Gizgate translates each call to an IntraGiz constructor or method into an inter-process message which it transmits to its board-side server module. That server is linked with the regular IntraGiz library and carries out the request. Since IntraGiz is essentially a master/slave protocol, no activities originate from the FPGA except enabled interrupts. These are conveyed back to the host computer by Gizgate and result in calls to the specified handler function.

The simple description above reveals what is basically a remote method invocation type of OO distributed system

²The RPB was developed by the Department of Electrical and Computer Engineering, University of Toronto, under the auspices of CMC. The MPU monitor prompt is “Gizmo2>”, the name Gizmo being a holdover from a 68000-based processor board used in their computer architecture courses. This explains the “Giz” in our names IntraGiz and Gizgate.
communication. The following section describes Gizgate’s internal design in more detail.

4. Gizgate architecture

4.1. Components

Figure 4 illustrates the four major software components of Gizgate. The red dotted rectangles enclose processes running on one or more UNIX hosts; the RPB is shown in the dashed blue rectangle. Shaded blocks indicates Gizgate/IntraGiz software.

Walking through the components from the upper left Application Host, the user’s C++ application is linked together with IntraGiz stubs constituting the Gizgate adapter component. When such an application runs on any host, its Gizgate adapter establishes a socket connection (solid red arrows) with the Gizgate Remote Server. The Remote Server process arbitrates access to the RPB resource and forwards IntraGiz object transactions to the RPB Gateway.

The purpose of the Gizgate RPB Gateway is twofold. First note that the RPB is physically connected to the RPB Host by means of the serial “Host” line. (There is a second serial “Terminal” line, not shown, which provides the command line interface for the MPU’s ROM monitor, FBUG. This is typically configured as a remote device in a UNIX “tip” session on the RPB Host.) During an MPU download operation, the FBUG monitor expects the Host line to be connected to a UNIX command prompt. However, when Gizgate is in operation it takes over the Host line for its own use. Therefore, the Gateway must recognize when it is talking with FBUG and transparently emulate a UNIX shell.

Second, when Gizgate transactions are in progress, the Gateway is responsible for translating between the UNIX socket-based messages used by the Remote Server and an economical text-based format used on the slow serial line. When transactions arrive in the board-side RPB Server, methods are invoked on actual IntraGiz objects corresponding to the stubs in the currently connected Application Host. This results in communication with the FPGAs.

Strictly speaking, the Application Host need not be running a UNIX operating system. The Gizgate Adapter component could be rehosted under another OS that provides TCP/IP sockets. In any case, the important result is that the IntraGiz-related portion of the user’s client code is location transparent. However it runs on the remote host, it can be recompiled for the RPB’s MPU without modification and it will operate the same, aside from Gizgate communication delays.
4.2. Internal design

A more detailed view of the flow of data and control within Gizgate is shown in Figure 5. The dotted rectangles represent remote software processes, while the large magenta dashed rectangle encompasses the processes on the RPB, including the FPGA synthesized hardware.

In the leftmost block labeled Gizgate Client, the IntraGiz proxy objects are represented by blue circles “cs,” for “client stub.” Instead of IntraGiz methods, the objects invoke an alternate Gizgate implementation which marshals the arguments and sends message packets to their counterpart Router Clients (“C,” in the Remote Server process via network sockets (blue arrows). The messages are multiplexed by the Gizgate Remote Server through an RPB Server Client (one per Gizgate Client process), and are sent off, one at a time, to the RPB Gateway process (in the middle). The Gateway process acts as a bridge between the network TCP protocol used on the host side and the proprietary serial (magenta) serial line protocol used to communicate with the RPB. The Gateway process strips away non-essential data from the message packet, and it sends and accepts minimal messages across the serial connection. RPB transactions are timed by the Gateway. If a response is not forthcoming, the default action is to return an abort message to the Gizgate client. This results in throwing an exception in the user application program.

When a message arrives in the Gizgate RPB Server (right side), its target IntraGiz object “I,” is determined. The specified method is invoked with its unmarshaled arguments, which causes I/O to be performed on the FPGA. A return message is constructed from the input data, if applicable, or, in any case, a positive acknowledgment. This is sent back to the originating client stub object via the RPB Gateway and Remote Server.

5. Performance

IntraGiz itself is very efficient. Apart from the constructors and destructors, the actual I/O methods are just a short line of C++ code, representing a handful of machine instructions. Multiple-pin methods have a packing or unpacking loop.

Since with Gizgate we have inserted a substantial layer of communications between the remotely hosted software and IntraGiz on the RPB, it is clear that “speed” was never a primary motivation. However, the worst part of the time penalty is incurred from the existing 19.2 Kbaud serial line connecting the RBP and its host. With the proper software in the MPU, the board’s external connectors could be used to attach it directly to the network or to a workstation bus extender.

6. Conclusions

Adding the Gizgate layer on top of IntraGiz yields a number of benefits for RPB users. First, Gizgate allows the application developer to get access to richer development and debugging facilities on a remote host before subsequently moving the application code to the MPU. Secondly, some applications may need access to disk files or other operating system services that only exist on a PC or workstation. With Gizgate, these can be utilized along with the FPGAs, for example, to read files of test vectors and write test logs. Thirdly, Gizgate facilitates timesharing of a scarce resource—the RPB—among multiple users, and can offer them access to hardware configurations. In the future, Gizgate can be enhanced to manage a library of “configuration,” swapping FPGA configurations on demand.

With respect to codesign, we see value in having an approach to HW/SW interface construction, in this case the building blocks given by IntraGiz and Gizgate. Building with a high level OO layer, as opposed to MPU assembly instructions, insures ease of coding and avoids many possi-
ble logical errors. The OO paradigm enables complete control of all hardware details, while enhancing the productivity of engineering a codesign prototype. We believe that the careful design of the interface portion in a codesign project is essential to the ultimate success of the HW/SW interaction.

7. References


