Internet-based Tool for System-on-Chip Integration

David Lim, Christopher E. Neely, Christopher K. Zuver, John W. Lockwood
Applied Research Laboratory
Department of Computer Science and Engineering
Washington University
1 Brookings Drive, Campus Box 1045
Saint Louis, MO 63130
lockwood@arl.wustl.edu

ABSTRACT

A tool has been created for use in a design course to automate integration of new components into a System-On-Chip (SoC). Students used this tool to implement a complete SoC Internet firewall, which was prototyped and tested using a field-programmable gate array (FPGA). Common components of the framework were completed as machine problem assignments throughout the first half of the semester. During the second half of the semester, students worked in small groups to design extensible modules, which included additional packet filters, a packet encryption engine, and replacement schedulers, to enhance the functionality of the SoC firewall. The integration tool described in this paper was used to manage project submissions and to automatically generate synthesized designs for student testing and also project evaluation.

INTRODUCTION

Growth in silicon capacity has made it possible to implement complete networking systems within a single Integrated Circuit (IC). Such systems consist of multiple modular components interconnected by common infrastructure. The Reconfigurable System-On-Chip (SoC) design course at Washington University in Saint Louis explores the challenges of designing and testing a complete Internet firewall [1]. Students in the course designed, synthesized, and simulated components using modern Computer Aided Design (CAD) tools. The machine problem assignments were written to familiarize the students with the CAD tools and VHDL constructs that define the functionality of an Internet firewall. The baseline firewall transmits or drops packets based on the source, destination, and ports fields of the packet header. The first enhanced design included a payload filter that used a regular expression engine to search the contents of packet payloads. The second enhanced design included a flow buffer and queue manager to circumvent Denial of Service attacks.

For final projects, students devised new packet processing modules to augment the function of the

enhanced firewall. They attached these new modules to interfaces provided by the SoC framework. Final projects were required to be interoperable with the SoC framework and to fit with other group projects on the same FPGA.
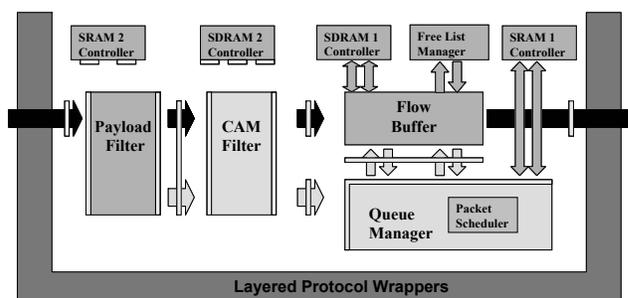

Figure 1. SoC Internet Firewall Extensible Interface

Several challenges hinder the implementation of SoC projects, and so an integration tool was developed to address these issues. The first challenge was to allow changes to be made to one module without affecting the other parts of the system. The integration server acts as a repository for all the modules and allows all of the modular components to integrate into a top-level design. The second challenge arose from the fact that the input/output ports of each module must be connected to the proper ports on the top-level design. If done manually, this task is time-consuming and mistake prone. The integration server automatically wires together the circuit. The third challenge stems from the fact that resources on the chip are limited. The modules for a SoC have to fit within the lookup tables, flip-flops and RAMs of one FPGA. The integration tool tracks the resource utilization of a top-level design with multiple modules and shows how they would fit into the targeted FPGA.

INTEGRATION SERVER

The integration server can be viewed as two logical servers: the module collector and SOC bitfile generator. Each is controlled via a web-based user interface. The collector reads in newly created modules from the web and stores them in the modules database. The bitfile generator creates a bitfile of the complete SOC firewall using a

subset of the modules in the database. A diagram of the integration server is shown in Figure 2.
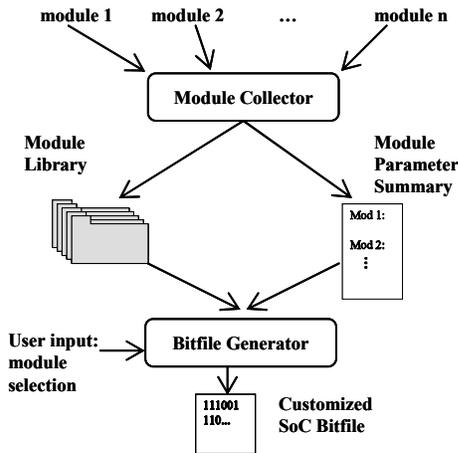

Figure 2. Integration Server

MODULE COLLECTOR

When a module is ready to be integrated into the SoC firewall, the collector reads the top-level VHDL and EDIF files that define the structure and implementation of the module. The collector verifies that the uploaded files are correctly formatted and parses the VHDL files to identify all of the ports on the module. For each interface, the collector prompts the user for the corresponding interface on the top-level design. Next, the collector prompts the user to select how each port should be mapped on each interface. The collector then records the port mapping for the newly created module, generates a bitfile of only the new module integrated into the baseline SoC, records device utilization as determined by the place and route tools, sends the bitfile to the user for testing. It also stores the newly created module in the modules library, and updates module parameters in Module Summary File (MSF). The MSF contains summary of all available modules and their corresponding device utilizations.

In essence, the module collector transforms the time-consuming and mistake-prone task of individually wiring ports on the newly created module (in VHDL) into the relatively simple task of selecting where a module fits within a well-defined SoC. As long as the user adheres to one of the provided modular interfaces, they need not worry about the lower-level details of integrating a newly created module into the SoC.

BITFILE GENERATOR

To allow the synthesis of custom firewalls, the generator parses the MSF created by the collector and displays a menu of available modules. Every time a module is selected, the generator automatically tracks the logic (slice) utilization and memory utilization. Once the selections are made, the generator builds a bitfile for the

SoC firewall with the selected subset of modules. First, it verifies that the resource utilization of the selected modules combined with the baseline firewall does not exceed the available resources on the FPGA. Next, it maps all ports on the requested extensible modules using the port mapping stored by the module collector. Finally, it generates a bitfile with all the desired extensible modules.

A screen shot of a request form for an actual design run is shown in Figure 3. Every time a user loads this page, the SoC bitfile generator re-generates the page by parsing the MSF, and displays a list of all of the available modules. The bitfile generator tracks the device utilization and displays it on the form. In this screenshot, five modules are available for integration. With none of the extensible modules selected, 39% of the block Rams are utilized and 43% of the logic slices are used. As modules are selected, the values listed at the bottom of the page are automatically updated.
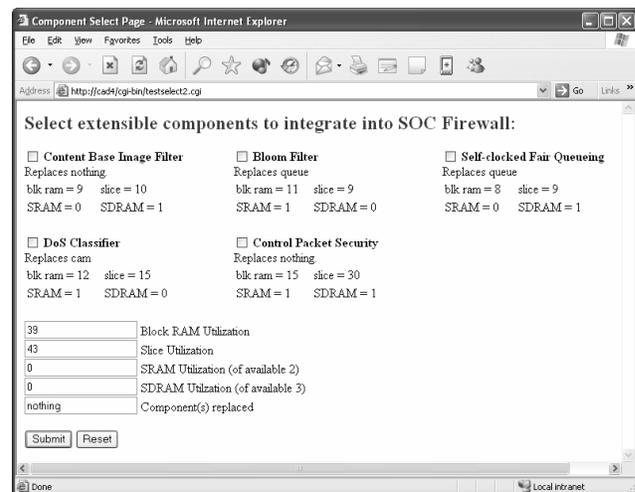

Figure 3. Screenshot of Interface for Bitfile Generator

CONCLUSIONS

The integration server was used to synthesize FPGA-based firewalls for a SoC class. Multiple modules for filtering Internet packets were developed and integrated into the FPX platform. All the extensible modules were developed concurrently throughout one semester. The integration server combined modules into a single FPGA that implemented the single-chip firewall.

REFERENCES

[1] John W. Lockwood, Christopher Neely, Christopher Zuver, "CS536 Course Website," Washington University, http://www.cs.wustl.edu/~lockwood/class/cs536/index.html

[2] John W. Lockwood, Jon S. Turner, David E. Taylor, Field Programmable Port Extender (FPX) for Distributed Routing and Queuing, ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000), Monterey, CA, February 2000, pp. 137-144.