Customizing Controller Instruction Sets for Application-Specific Architectures

Jian Li, David Dickin and Lesley Shannon
School of Engineering Science
Simon Fraser University
Burnaby, British Columbia, Canada
Email: {jla193, drdickin, lshannon}@sfu.ca

***Abstract*—Previous work has proposed the "Systems Integrating Modules with Predefined Physical Links" (SIMPPL) architectural framework as one possible method to shorten the design cycle by utilizing a light weight programmable controller (SIMPPL Controller) as the system-level interface.**

This paper presents a study of how much improvement in area, power, and performance can be achieved through the customization of the SIMPPL Controller's instruction set. Furthermore, we have created a tool to automatically generate the HDL for SIMPPL Controllers with a user specified instruction set. Our study on an FPGA platform has shown that using a customized SIMPPL Controller with a minimal instruction set results in: an area reduction of 42%, a performance increase of 16%, and a power reduction of 10%.

***Keywords*-Systems-on-Chip, SoC Design Methodologies, Application-Specific Architectures, Application-Specific Instruction-set Processors, IP Reuse, Design Automation**

I. Introduction

For System-on-Chip (SoC) design, reusing previously designed Intellectual Property (IP) cores has been one of the popular methods to reduce system design time. Ideally, this would be akin to reusing previously designed software functions. Unfortunately reusing hardware IP is more complicated because each IP may have different forms of inter-module communication (serial/parallel, broadcast/addressed, etc.), and integrating them into the new systems may require significant redesign of the original cores. In fact, the complexity of these system integration challenges may take up to 30% of the overall SoC design time [1].

This led to the proposal of modelling SoCs as *Systems Integrating Modules with Predefined Physical Links* (SIMPPL), which has been demonstrated to reduce system integration time to less than 5% of the total design time [2]. The SIMPPL architectural framework models a SoC as a network of *Computing Elements* (CEs) interconnected via point-to-point links (usually FIFOs), and may be used for SoCs implemented on Field Programmable Gate Arrays (FPGAs) or as Application-Specific Integrated Circuits (ASICs).

Figure 1 illustrates the three components of a hardware CE in terms of their separate modules: **the Processing Element** (PE) comprises the datapath, or the actual hardware IP core being reused, in the CE; **the SIMPPL Controller** provides a programmable inter-CE communication interface to receive and transmit instruction packets; and **the**

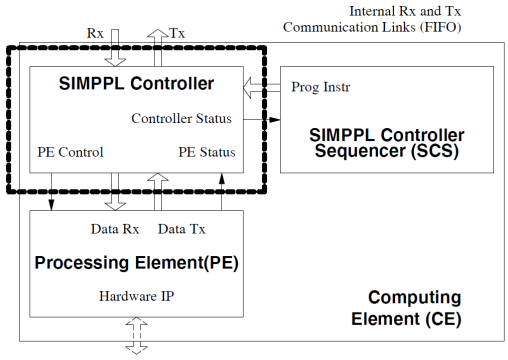

Figure 1: The CE abstraction components.

SIMPPL Control Sequencer (SCS), in combination with instruction packets received by the CE via the receive link, determines how the PE is used with the overall system. The benefit of the CE abstraction is that the PE's functionality remains unchanged, the designer need only change the "program" (SCS) executed by the SIMPPL Controller, and the instruction packets received from other CEs.

The overhead of the CE abstraction is encapsulated by the SCS and the SIMPPL Controller as they provide the SoC interface circuitry for the PE. However, the CE's operations may be dictated completely by instruction packets received from other CEs and not to use an SCS. Therefore, the fixed overhead of using the CE abstraction is incurred from the SIMPPL Controller itself.

This paper presents a study of how much improvement in area, power, and maximum operating frequency can be achieved through the customization of the SIMPPL Controller's instruction set for individual PEs. Furthermore, we have created a tool to automatically generate the HDL for SIMPPL Controllers with a user-specified instruction set. Our study shows that if only a minimal instruction set is required, the application-specific version of the SIMPPL Controller can reduce area by 42%, power consumption by 10%, and increase the operating frequency by 16% relative to the full SIMPPL Controller on an FPGA platform.

The remainder of this paper is organized as follows. Section II summarizes the related work and the SIMPPL Controller architecture and instruction set. Section III discusses the SIMPPL generator and Section IV provides the experimental setup and results of our evaluation of the

application-specific SIMPPL Controllers. Section V concludes the paper and discusses possible areas for future work.

II. Background

This section provides an overview of the related work of customizing SIMPPL Controller instruction set and an overview of the original architecture and instruction set.

A. Related Work

Previously proposed architectural frameworks and on-chip communication structures that are similar to SIMPPL include Berkely's SCORE [3] architecture, which divides system computations into fixed-size pages and uses streams as a data abstraction for passing data between pages. Like SIMPPL, the streams use a point-to-point communication link; however, no physical connection is defined. Adaptive System-on-Chip (aSOC) [4] utilizes a point-to-point communication architecture where each module's communication interface is tailored in hardware to optimize performance. SIMPPL does not optimize the physical implementation of each module's communication interface. Instead, the SIMPPL interface is common between all modules to ease the integration process. The SIMPPL Controller and SCS are used to customize each PE's communication scheme to work in the overall system.

Several well-defined IP design methodologies have been proposed [5][6] to ensure that cores with fixed functionality and fixed interfaces have high reusability in future designs. However, they do not address the situation where a core's functionality is required but the system-level interaction requirements have changed. VSI Alliance proposed the Open Core Protocol (OCP) [7] to enable the separation of a core's functionality from its communications by using a socket interface for IP. The socket allows a designer to connect their core to the many bus types supported by the standard. The OCP approach is similar to the SIMPPL model, except that SIMPPL uses a direct communication model for all on-chip communications.

The term Application-Specific Instruction-set Processor (ASIP) design was originally referred to automatically generating complete Instruction-Set Architectures (ISA) for specific applications to improve system performance, resource usage, etc[8]. However, ASIP design is extremely challenging because it requires a complete software and hardware tool flow to support programming and designing processors with customized instruction sets [9] [10]. Therefore, ASIP research started focusing on extending instructions of generic processors to avoid the complexity of (re)designing a complete hardware and software tool-chain [11].

The AutoTIE system [12] from Tensilica is one example of the ASIP solutions that enhances a base processor with specific ISA extensions. AutoTIE first generates various combinations of ISA extension configurations based on application profiling and analysis, and then performs architecture explorations to select the best suited implementation. Our work differs from AutoTIE and previous ASIP research as the focus is on customizing light weight controllers that provide the interface and system-level control of custom IP to reduce the overhead of the SIMPPL framework as opposed to customizing traditional processor ISAs.

Table I: Full SIMPPL Controller instruction set

Instruction Type	Addr Field	Data Field
Reset		
No-op		
Bypass		X
Register Initialization	X	
Register Arithmetic		
Immediate Data Transfer		X
Immediate Data Transfer + Immediate Address	X	X
Immediate Data Transfer + Indirect Addressing	X	X
Immediate Data Transfer + Autoincrement	X	X

B. The SIMPPL Controller

The SIMPPL Controller (highlighted in Figure 1) functions as a programmable system interface that processes instruction packets received from other CEs and generates instruction packets for processing by other CEs [2]. Table I lists the complete instruction set supported by the SIMPPL Controller. All data transfer instruction formats allow data read, data read request and data write operations. The SIMPPL Controller also includes an optional debugging infrastructure for debugging a new PE's integration. The "debug" version (as opposed to the normal "execute" version) of the controller is equipped with additional state logic that can be used to monitor the run time status of the controller and PE [2]. Morever, we also investigated the possibility of customizing the "full" execute version of the controller (Full Controller) for PEs that only produced (Producer Controller) or consumed (Consumer Controller) data to reduce system overhead. For example, an ADC only requires a Producer Controller as the PE has no facility for reading in data, whereas a DAC requires a Consumer Controller has it cannot generate data.

III. Creating custom SIMPPL Controllers

The SIMPPL Controller Generator, `simpplgen`, generates application-specific SIMPPL Controllers by removing unused logic. The SIMPPL Controller has a traditional three-stage micro-controller architecture, consisting of: 1) *fetch*, 2) *decode*, and 3) *execute*. The pseudo code in Figure 2 shows the *decode* and *execute* stages of the controller (marked as Decode Block and Execution Block respectively in the Figure 2). If an instruction is removed from the controller's ISA, then its corresponding "`if`" clause can be removed from the Decode Block. If none of the instructions in the controller's ISA use a specific flag, then both the flag and the corresponding action become unnecessary logic.

```
if (opcode == INST0) {  /* Decode Block */
  flag_0 <= 1;
  flag_1 <= 1;
  flag_2 <= 0;
   ...
  flag_n <= 0;
}
 ...

if (flag_0 == 1) {      /* Execution Block */
  action_A; clear flag_0;
}
if (flag_1 == 0) {
  action_B; clear flag_1;
}
...
```

Figure 2: Pseudo Code for Decoding and Executing SIMPPL instructions.

	flag_0	flag_1	flag_2	…	flag_n
~~INST0~~	1->0	1->0	0	…	0
INST1	0	0	1	…	1
…	…	…	…	…	…
INSTm	0	0	1	…	1

Figure 3: The matrix used in `simpplgen`.

Currently, the user manually specifies the SIMPPL instruction set they want to support as an input file read by `simpplgen`. `simpplgen` then removes the logic for the instructions that are "deselected" by the user from the full SIMPPL ISA, specifically, the circuitry responsible for decoding and executing these unused instructions. This "unnecessary" logic is detected via a matrix, which is used to relate the instructions to the corresponding series of flags that they set. Figure 3 illustrates how the matrix is used to remove the instruction *INST0*. In the Full SIMPPL Controller, both *flag_0* and *flag_1* are decoded for *INST0* only, and marked with a '1' to indicate that their corresponding actions must be performed to complete the instruction. If the user indicates that *INST0* is to be removed, then `simpplgen` updates the corresponding matrix entries from '1' to '0'. Since both *flag_0* and *flag_1* are only decoded for *INST0* (i.e. none of the other *INST* entries set these two flags to '1'), then `simpplgen` can remove the logic related to these flags without affecting other parts of the controller.

IV. Experimental Setup and Results

The resource usage and maximum clock frequency numbers, as well as the power consumption for each customized controller were obtained using version 10.1.3 of ISE CAD flow on a Virtex 4 LX40-12 device. The Modelsim™ simulator from Mentor Graphics [13] was used for design debugging. This section presents a comparison of the results of using a minimal SIMPPL Controller ISA versus the original SIMPPL Controller ISA.

To quantify the minimal overhead that can be incurred by using the SIMPPL Controller interface, an ISA supporting

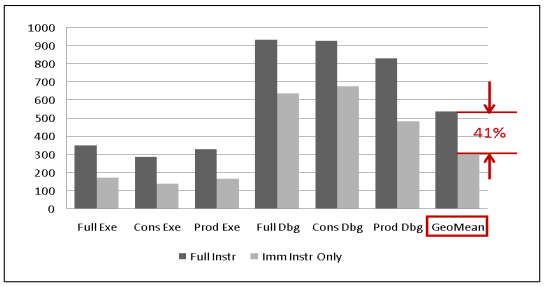

Figure 4: Number of 4-LUTs used for different SIMPPL Controller configurations

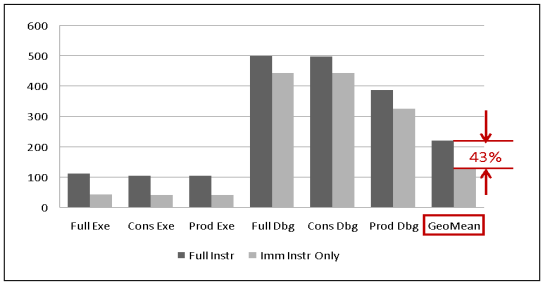

Figure 5: Number of flipflops used for different SIMPPL Controller configurations

only the immediate data tranfer instructions and the no-op instruction is used, see bolded instructions in Table I. This ISA does not support addressing schemes, eliminating these components from the datapath and reducing the decode and execute stage logic. To evaluate this minimal ISA, `simpplgen` is used to generate custom versions of the six original SIMPPL Controller configurations (execute/debug, full/consumer/producer), as discussed in Section II-B.

Figures 4 and 5 illustrate the different controllers' Look-Up-Table (LUT) and Flipflop resource usage, respectively, on a Virtex 4 LX40-12 device. The dark grey columns indicate the resource usage of the original controllers from previous work [2], which support the full instruction set for the given configuration (full/producer/consumer, execute/debug). The lighter grey bars indicate the resource usage of the corresponding SIMPPL Controller configurations based on the minimal ISA generated by `simpplgen`. The final two columns in both Figures 4 and 5 is the geometric mean of the LUT and flipflop resource usage over all six configurations for the original SIMPPL Controllers versus those generated by `simpplgen` from the minimal ISA.

Overall, the LUT usage of the different SIMPPL Controller configurations is reduced by 41% on average and the number of flipflops is reduced by 43% on average. Although the debug versions require significantly more resources, they are only to be used during design phase, and if only the "Execute" versions of the SIMPPL Controllers are considered, LUT and flipflop resource usage is reduced on average by 51% and 61%, respectively.

Figure 6 summarizes the maximum clock frequency ob-

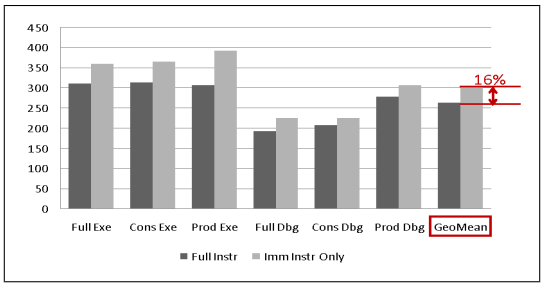

Figure 6: Maximum Clock Frequency (MHz) for different SIMPPL Controller Configurations

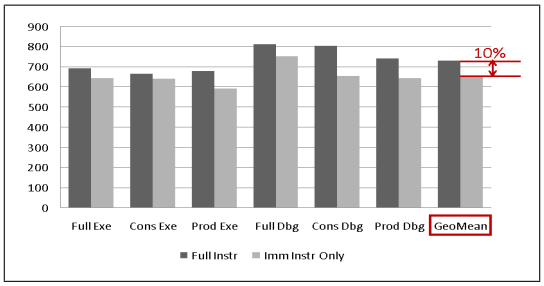

Figure 7: Estimated Power Consumption

tained for each of the SIMPPL Controller configurations, with a similar column format. On average, the autogenerated SIMPPL Controllers have a 16% increase in operating frequency. As before, by considering only the Execute SIMPPL Controllers, an average increase of 20% in the operating frequency is observed.

An operating frequency of 194 MHz is used for the power estimation as this is the maximum operating frequency of the Full Debug SIMPPL Controller, the slowest of all the SIMPPL Configurations (see Figure 6). Figure 7 presents the power consumption estimation results using the same format as the preceding figures. The geometric mean of the reduction in power consumption across all six configurations is 10%. Interestingly, the average drop in power consumption for only the "Execute" SIMPPL Controllers is 8%, less than the overall average. We expect that this is partially due to the placement and routing of the SIMPPL Controllers on the device. As the controllers use such a small percentage of the device (<5%), the CAD flow need not used as compact a placement. This would result in using more of the routing fabric, which would not be reflected in the LUT and flipflop resource usage and yet still consume more power.

V. Conclusions and Future Work

This paper presents a study of how customizing the SIMPPL Controller architecture with an application-specific instruction set can greatly reduce the the overhead of using the SIMPPL framework. Specifically, customizing the controllers with a minimal instruction set can reduce resource usage by 42% and power consumption by 10% while increasing the operating frequency by up to 16%. The paper also introduces `simpplgen`, a CAD tool for automatically generating application-specific SIMPPL Controllers based on a user specified ISA.

Present work is focused on adding an error checking function to the generator to ensure that the user input file has a valid format and specifies a valid ISA. The autogenerated versions of the SIMPPL Controllers are also being synthesized to ASIC standard cell libraries to see how this might reduce the overhead of using the SIMPPL SoC framework for ASICs. Finally, the current structure of the `simpplgen` suggests that it may be possible to allow users to incorporate new instructions into the ISA. A feasibility study is currently underway, with the hope that this will be possible in the future. A long term goal for this project is to allow users to compile a high-level language description of the system-level CE interactions into the appropriate SCS's and their corresponding application-specific SIMPPL Controllers.

References

[1] (2003) MEDEA+ EDA roadmap, executive summary europe. [Online]. Available: http://www.medea.org/

[2] L. Shannon and P. Chow, "SIMPPL: an adaptable soc framework using a programmable controller ip interface to facilitate design reuse," *IEEE Trans. VLSI Syst.*, vol. 15, no. 4, pp. 377–390, 2007.

[3] E. Caspi, M. Chu, R. Huang, J. Yeh, J. Wawrzynek, and A. DeHon, *Stream Computations Organized for Reconfigurable Execution (SCORE)*, ser. Lecture notes in computer science. Springer Berlin / Heidelberg, 2000, pp. 605–614.

[4] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier, "An architecture and compiler for scalable on-chip communication," *IEEE Trans. VLSI Syst.*, vol. 12, no. 7, pp. 711–726, 2004.

[5] W. Savage, J. Chilton, and R. Camposano, "IP Reuse in the System on a Chip Era," in *Proc. of 13th Int'l Symposium on System Synthesis*, 2000, pp. 2–7.

[6] G. Martin, "Design methodologies for system level IP," in *Proc. of the Conf. on Design, Automation and Test in Europe*, Paris, 1998, pp. 286–289.

[7] VSI Alliance. [Online]. Available: http://www.vsia.org

[8] B. K. Holmer, "Automatic design of computer instruction sets," Ph.D. dissertation, UC Berkeley, 1993.

[9] J. Van Praet, G. Goossens, D. Lanneer, and H. De Man, "Instruction set definition and instruction selection for ASIPs," in *Proc. of the 7th Int'l Symp. on High-level Synthesis*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 11–16.

[10] I. Huang and A. Despain, "Synthesis of application specific instruction sets," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 6, pp. 663–675, Jun. 1995.

[11] K. Atasu, L. Pozzi, and P. Ienne, "Automatic application-specific instruction-set extensions under microarchitectural constraints," in *Proc. of Design Automation Conf.*, 2003, pp. 256–261.

[12] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," in *Proc. of the 2003 Int'l Conf. on Compilers, Arch. and Synthesis for Embedded Systems*, 2003, pp. 137–147.

[13] ModelSim users manual. Mentor Graphics Inc.