IPSJ Transactions on System LSI Design Methodology Vol.10 63–70 (Aug. 2017)

[DOI: 10.2197/ipsjtsldm.10.63]

Short Paper

Towards Open-HW: A Platform to Design, Share and Deploy FPGA Accelerators in Low Cost

Qian Zhao[1,a)] Motoki Amagasaki[1] Masahiro Iida[1] Morihiro Kuga[1] Toshinori Sueyoshi[1]

Received: November 28, 2016, Accepted: January 25, 2017

Abstract: Field-programmable gate array (FPGA) is a promising technology for the implementing of high-performance and power-efficient cloud computing by serving dedicated hardware as co-processor to accelerate loads on CPUs. However, developing an FPGA-based system is challenging because the complexity of the hardware and software co-design. In this paper, we propose a platform named hCODE to simplify the design, share, and deployment of FPGA accelerators. First, we adopt a shell-and-IP design pattern to improve the reusability and the portability of accelerator designs. Second, we implement an open accelerator repository to bridge hardware development and software development on one platform. On the hCODE platform, hardware developers can provide designs that follow hCODE specifications, which allowing software engineers to easily search, download, and integrate accelerators in their applications without caring about the hardware details.

Keywords: FPGA, open-source hardware, hardware acceleration

1. Introduction

The process of constructing an information and communication society constantly requires more and higher-performance computing devices with the constrain of lower power consumption. Although the performance per Watt of present CPUs and GPUs is hitting a wall, many recent studies have demonstrated the power-efficiency advantages of field-programmable gate arrays (FPGAs), which allows dedicated hardware to be implemented by programming after manufacture. However, designing of hardware on FPGAs is costly because circuits are traditionally developed by using low-level hardware description languages (HDLs). Although a variety of convenient high-level synthesis (HLS) tools [1], [2] that translating C-like programs into HDLs have been introduced by academic community and FPGA vendors recently, a deep understanding on hardware is still required for the production of high-performance circuits with these tools. In order for FPGAs to be adopted in real world data-center applications such as big data, machine learning, and web services, in particular, the methodology as well as tools that efficiently organizing both hardware and software designs are required.

We believe co-designing of hardware and software is difficult in large and complex projects for two reasons that were not fully discussed yet. First, there is no clear division between the hardware design flow and the software design flow. Co-designing of hardware and software under current methods requires developers to have knowledge of both hardware and software development, which is difficult especially when architecting large projects involving tens to hundreds of people. Therefore, an engineering methodology clarifying roles and responsibilities of developers is needed, which allows the hardware developers to concentrate on accelerator design and software developers to benefit without caring about the details of the hardware.

Second, there is no sophisticated open-source platform for hardware developers to publish their designs and from which software developers can directly obtain accelerators out of the box. Although many valuable efforts have been done on frameworks [3], [4], [5], [6] and high-performance accelerators for FPGAs, which are commonly not easy to be obtained, evaluated and reused because of having different interfaces, different working devices, and managed in different places (difficult to search). In the past two decades, software developers have solved the very similar problem under the open-source model, which has significantly reduced development costs by enabling the sharing and reuse of source code and knowledge. An open-source hardware platform is therefore expected to be an indispensable component of hardware and software co-design.

These problems consequently prevent the value delivery from the hardware community to the software community, then become difficulties for FPGAs to be adopted in large and complex projects. In this paper, we propose a hardware and software co-design platform named the Heterogeneous Computing Oriented Development Environment (hCODE). The hCODE is the first open-source platform approach for simplifying the design, share, and deployment of FPGA accelerators. As a solution of the described problems, the main contributions of hCODE are as follows.

- *A shell-and-IP methodology.* By following the *shell + IP = accelerator* design pattern from traditional FPGA frameworks, hCODE defines three roles in the platform: the shell developer, the IP developer and the accelerator user.

[1] Kumamoto University, Kumamoto 860–8555, Japan
[a)] cho@arch.cs.kumamoto-u.ac.jp

Responsibilities of each role and interfaces between roles are clearly defined so that the collaboration cost of hardware and software engineers can be reduced.

- *An online repository and manager.* The hCODE repository can host many lightweight shells. Each shell only implements necessary blocks for a specific device or interface. An IP could be portable between shells by following the hCODE design rules, which also allows for easy searching, downloading, auto-assembling the IP with a compatible shell, and finally compiled with the manager tool.

The prototype of hCODE is currently open-sourced on Github (https://github.com/hCODE-FPGA/hCODE) for evaluation. All shells and IPs for the case study are also available on the hCODE platform. The rest of this paper is organized as follows. Section 2 introduces related works. A quick overview of hCODE is given in Section 3. Section 4 describes the proposed hCODE platform in detail. Evaluation of the proposed hCODE platform and case studies are shown in Section 5. Section 6 gives conclusions and discusses future work.

2. Related Works

OpenCores is an open-source hardware website that offers a variety of open-source intellectual properties (IPs) [7]. However, OpenCores does not provide any functions beyond online storage. In contrast, modern software package managers provide an online repository, version control, dependency control, and even allow automatic integration of third-party packages into a user project [8]. These convenient functions contribute to building a software development ecosystem and significantly reduced design costs. However, managing a hardware and software co-design project is more challenging when considering portability and scalability on different devices. The hCODE implements a hardware design manager based on the popular CocoaPods [8] from the iOS development community, while also providing necessary supports for reducing hardware and software co-designing costs.

Packaging common modules into a reusable shell framework is a traditional practice to improve hardware design productivity. Although FPGA vendors provide fundamental IPs in their development tools, it still requires efforts to utilize these IPs in specific projects. Third-party frameworks like RIFFA [3] and XILLYBUS [5] have done a lot of works on hardware and drivers in order to provide an easy-to-use interface based on the official PCIe IP core. Reference [6] introduced a complete open-source framework that provides PCIe, Ethernet, and DRAM interfaces. Microsoft also introduced a shell-and-role hardware architecture in Ref. [4]. However, no framework can efficiently fit all application scenario. General purpose frameworks are complexly implemented for more functionality and flexibility, which may result in a larger area and a lower frequency for FPGA designs. Therefore, we implement an online repository as well as a manager in hCODE. The hCODE repository can host many lightweight shells that only implements necessary blocks for a specific device or interface. And these shells can be easily searched and reused with the hCODE manager tool.

There are other works related to improving hardware and software co-design cost on FPGAs. In commercial, Xilinx SDAccel [9] and Altera [10] both released design tools and libraries by extending OpenCL. The vendors have greatly improved the FPGA design efficiency by linking HLS compiler, official OpenCL compatible co-design framework, and IP libraries together. These tools as well as third-party libraries are not open-sourced nor free at present. However, the hCODE approach has the potential to integrate open-source IPs with these commercial flow in future. Other works implement customized multi-core processors on FPGA [11], [12], [13], so that advanced compiler techniques can be employed to utilize high-performance hardware. Because hCODE is an accelerator platform, both dedicated accelerator and multi-core processor on FPGA could be IPs and benefit from the platform.

From the software point of view, most current co-design flow follows embedded or OpenCL programming manner based on C-like languages. In order to allow more high-level (like Java) developers utilize accelerators easier, Refs. [3], [5] provide communication drivers, while [14] extends the OpenCL technique to Java. However, both ways require software developers to understand hardware. In hCODE, we suggest to package an IP following the interface of software library, so that it can be integrated into software application seamlessly.

3. hCODE Overview

The hCODE platform contains design method and tools in order to simplify the design, share, and deployment of FPGA accelerators. As **Fig. 1** shows, on the hCODE platform, hardware design and software design are organized in an online accelerator repository. In this section, we first introduce the fundamental design pattern of hCODE, and then demonstrate the hCODE using a quick example.

3.1 The Shell-and-IP Design Pattern

The hCODE implements the shell-and-IP design pattern and defines participant roles to reduce the design costs of accelerators. It is a common methodology to partition a hardware design into an application-independent shell part and an application-logic IP part [3], [4], [5], [6], as shown in **Fig. 2**. The shell part provides common modules and functions such as communication, memory control, and fault-tolerance. With this method, a developed shell can be reused in different IP projects, or an IP can be integrated into compatible shells for difference scenarios. An IP and a shell are linked to produce an accelerator.

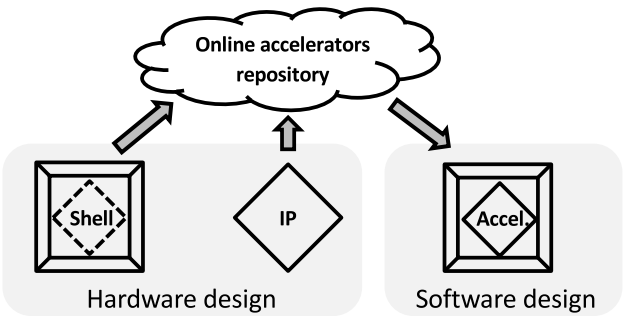

Fig. 1 hCODE platform.

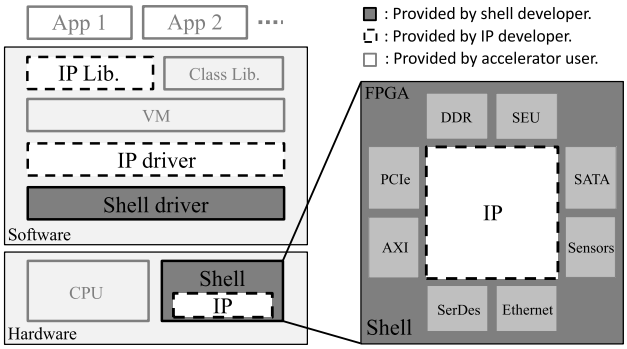

Fig. 2 Shell-and-IP hardware framework.

Table 1 List of main hCODE commands.

Command	Function
hcode setup	Set up the hCODE environment. Download a copy of hCODE SPEC repo. to ~/.hcode.
hcode list/search	List projects/Search projects with a *QUERY* string.
hcode ip get	Download an IP and select a compatible shell.
hcode ip make	Compile the accelerator automatically.
hcode repo	Add or remove private repositories.
hcode fpga	Scripts kit for FPGA management.
hcode ip create	Create a project with a specified shell.

Based on this design pattern, we define three roles on the hCODE platform: the shell developer, the IP developer and the accelerator user. As Fig. 2 shows, the shell developer provides a shell hardware design and a shell driver. An IP developer provides the IP design, IP driver, and IP library based on a shell. Please note that making software-friendly accelerators is an important principle of the hCODE. Therefore, IP developers have the responsibility to provide libraries for the ease of use. For instance, if a sorter IP is packaged as the same interface with *Arrays.sort(int[] a)* of Java core class library, a Java engineer can switch the sorting implementation between the software class and the accelerator seamlessly without caring about hardware details.

3.2 A Quick Tour

The hCODE manager is a command line tool developed from the CocoaPods [8]. Implemented commands are a collection of tools and scripts for the operation of the online repository as well as simplifing FPGA development. **Table 1** gives a list of the main commands in hCODE. Next, we demonstrate basic usages using a tour of obtaining an accelerator.

3.2.1 hcode setup

This command is used to perform environment initialization. A *.hcode* folder will be created under the user's home directory. Then the hCODE master SPEC repository that holding all projects' specification (SPEC) files is cloned to this folder.

3.2.2 hcode list/search

The *hcode list* command lists names of all projects that holding on the hCODE platform, while *hcode search* is used to search projects by with keywords. For instance, *hcode search vc707* returns projects related to the vc707 board. At present, the search command perform full-text search through the SPEC repository. This is sufficient to find target projects by a keyword such as the name of an accelerator, a board, an FPGA device or an interface.

3.2.3 hcode ip get

An accelerator user uses *hcode ip get* command to download the required IP. First, the IP is downloaded from its original repository. Then, hCODE lists all compatible shells for the user to choose. Then the selected shell is obtained and all components to produce an accelerator is ready. The hCODE provides three mechanisms to match compatible shells for an IP, which is discussed in Section 4.3.

3.2.4 hcode ip make

Finally, *hcode ip make* command performs the configuration and the compilation of the accelerator. This process configures and compiles the IP, integrates the product of the IP to the shell project, and at last compiles the combined project to generate a bitstream file. The IP driver compilation and installation are also performed in this stage. More details are explained in Section 4.4.

There are other commands relate to the ip creation, ip sharing, repository management and FPGA management. More details are given in hCODE project page on Github.

4. hCODE Implementation

The hCODE inherits modern version control and repository management features from the CocoaPods [8]. In addition, we implement the shell-and-IP support to provide scalability and portability for hardware designs while do not setting too much limitations on developer's implementation freedom. In this section, we discuss the mechanism of reduction of the the design, share, and deployment costs of FPGA accelerators in details.

4.1 The SPEC File and Online Repository

A JSON format SPEC file with the name *hcode.spec* is necessary to describe a hardware design. **Figure 3** shows SPEC file example parts for a shell and an IP design. Basic information sections such as the design name, project type, author, summary, license, source for the two type of designs are the same. The *name* section describes the project name. The hCODE requires the name of shell and IP projects start with the *shell-* and *ip-* prefixes, respectively. The *type* section indicates the type of the project, which can be a value of *shell* or *ip*. The *source* section shows the Git URI and tag version of the IP, which is important for hCODE to know when cloning the project. The remainder parts will be explained in the following shell and IP sections.

There is a central hCODE SPEC repository on Github that holds public information for all of the projects in the hCODE platform, as shown in **Fig. 4**. The folders in this repository are organized in a structure consisting of project name, project version, and project specification file. This repository is synchronized to local when a user executing *hcode setup* command. By indexing this repository, hCODE can perform project searching, downloading, and shell-and-IP matching for platform users.

Shell and IP developers are allowed publishing their designs on the hCODE platform by submitting SPEC files to the SPEC repository. This flow is shown in Fig. 4. First, a developer has to fork the SPEC repository from the hCODE master branch to a private repository. Second, the developer creates a sample *hcode.spec* with *hcode spec create* command, makes necessary modifications on it, and pushes entire design into the private

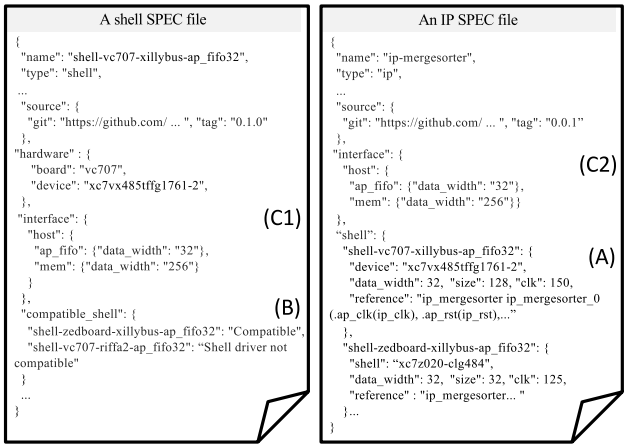

Fig. 3 hCODE SPEC file examples.

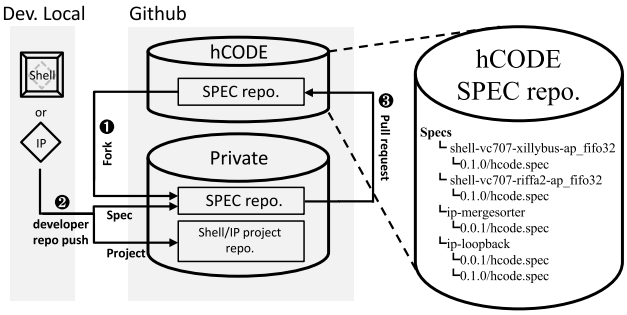

Fig. 4 Repository and design publish flow.

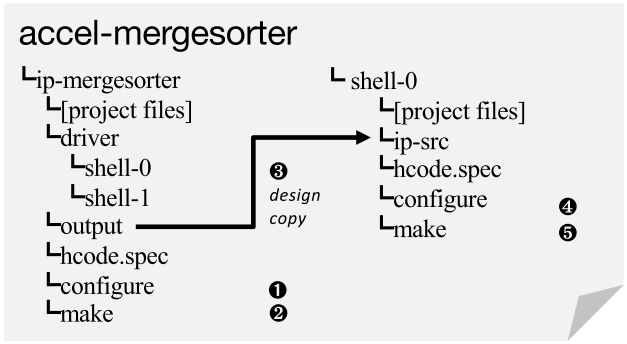

Fig. 5 Accelerator project and compile flow example.

repository with *hcode repo push* command. At last, a pull request to the hCODE master repository has to be sent on Github. In order to ensure the platform quality, we will evaluate developers' requests. After confirmation, the project is merged and becomes public to all hCODE users.

4.2 The Shell Design

In order to reduce the framework overhead, a shell design should provide the minimum reusable functions for IPs of a certain application scenario. Flexibility for difference devices, interfaces and functional blocks can be implemented by providing several shells on hCODE platform. An IP developer can use the *hcode search* command to find the shell that exactly fit a project.

The right side of **Fig. 5** shows a shell project structure. The *ip-src* is a directory to place the IP source codes. A *configure* script should be provided by a shell developer for minor changes such as clock frequency adjustment and IP reference code modification. This script simply reads parameters and make modifications on the shell project source code. The *make* script should provide three functions for IP integration and compiling: removing IPs files in *ip-src* folder from the shell project (*-removeip* argument), adding IPs files in *ip-src* folder to the shell project (*-addip* argument), and executing FPGA vendor's tool for compilation (no argument).

As Fig. 3 shows, a shell SPEC file must contain three determined sections. The *hardware* section describes the development board and the target FPGA device. The *interface* section indicates the interface providing for IPs. Figure 3 gives an IP-host interface example of a 32-bit Xilinx ap_fifo port and a 256-bit memory array port. Shell developers are free to describe the interface of their shells here. For instance, appending a *ddr* interface for DDR access or a *serdes* interface for high-speed communication. This information is helpful for IP developer to find a shell with desired interface. The *compatible_shell* section shows the name of other shells that may compatible with this shell. The *interface* and the *compatible_shell* sections are used by hCODE manager to find compatible shells for an IP; this mechanism is introduced in the next section.

4.3 The IP Design

A main target of the hCODE platform is providing methodology and tool for the scalable and portable IP designing. The scalability is implemented by the IP developer; and the portability is implemented by the hCODE and shells. A structure of an IP project instance is shown in the left side of Fig. 5. The specified sections of an IP SPEC file is the *shell* section, which describes parameters for implementing this IP on a certain shell.

4.3.1 Scalability

Most IP designs can be parameterized for scaling. For example, the merge-sorter demo IP can be scaled from a 4-input small tree to a tree of any input size by using an IP generator, because the tree structure rule does not change with size. As the same with the shell project, an IP developer has to prepare a *configure* script file, which reads parameters and perform necessary changes on the IP project. And a *make* script is also necessary to generate final HDL files and prepare these files in the *output* folder. The hCODE does not limit implementation of these scripts, an IP developer are free to use any method for the IP generation, such as a novel IP generator and the HLS, as long as final product HDLs can be found in the *output* folder.

4.3.2 Portability

The portability is the most attractive feature for IP developers. The hCODE can find compatible and possibly compatible shells for an IP. With this feature, an IP developed on one device has the potential to be implemented on other devices with compatible shells. The hCODE provides three methods for the shell-and-IP matching task.

First, using the *shell* section in the IP SPEC file to specify shells (Fig. 3-A). This method requires IP developers determine shells as well as IP parameters clearly. With these information, hCODE can automatically compile an accelerator by passing these parameters to *configure* scripts of the IP and the shell. Making a complete *shell* list is difficult, however, this work will

Table 2 Interface between shell and IP.

	Signal	IO	Description
Read FIFO	dout[data_width-1:0]	I	Data from shell FIFO
	empty_n	I	shell FIFO empty
	read	O	read enable
Write FIFO	din[data_width-1:0]	O	Data to shell FIFO
	full_n	I	shell FIFO full
	write	O	write enable
Mem (Xillybus only)	mem[255:0]	I	direct memory map signals

be possible if accepting contributions from the open-source community. On the software side, the IP developer have to prepare drivers for different shells under the driver folder. The *make* script reads the *shell* parameter, then compiles and installs driver of a selected shell.

Second, the hCODE suggests shells using the *compatible_shell* section (Fig. 3-B). A compatible-shell table is created inside hCODE by analyzing this section from all shells, and then suggestions can be made by returning compatible-shells of the determined shells from the first method. This feature is especially useful to new devices. Existing IPs can be ported to the new device if the vendor implementing compatible-shells and claiming compatibility with existing shells.

Third, the hCODE suggests shells with the same *interface* section (Fig. 3-C1,C2) of an IP. An IP is possibly implementable on the shell with the same *interface* it required. At present, hCODE performs this feature with simple content matching on this section.

Please note, only the first method provides information to the hCODE for an automatic accelerator integration. The last two methods require the user to perform the shell-and-IP integration manually. However, we believe accelerator users and shell developers have the motivation to contribute for enriching the *shell* list for valuable IPs.

4.4 The Accelerator Generation

As introduced in the Section 3.2.3, hCODE gives shells choosing options when downloading an IP. After shells and IPs are prepared, the accelerator project directory should be like Fig. 5 shows. By executing the *hcode ip make* command, the automatic compiling flow starts. First, hCODE loads the selected *shell* section parameters from the IP SPEC file, executes *configure* and *make* scripts of the IP project with these parameters. Second, generated IP HDL files are copied from the IP *output* folder to the shell *ip-src* folder. Then, the *configure* and *make -addip* of shell are executed to prepare the shell project. At last, the shell *make* script is called to generate the accelerator bitstream. Please note, because hCODE only defines the project directory structure and the script command interface, this flow is compatible with various design tools.

5. Case Study

In this section, we use several workable shells and IPs as the case study to evaluate the proposed hCODE platform. We have ported two host-IP communication shells from the Xillybus [5] framework and one shell from the RIFFA 2.0 framework [3]. For the application, we first use a simple loopback IP that can be automatically integrated with all shells to examine the hCODE accelerator compilation flow. Then an AES accelerator is implemented to show the portability between different shell frameworks and devices. At last, we use a merge-sorter IP to show the software integration that allowing a Java program utilize the accelerator in low cost.

5.1 Conditions

In order to examine the scalability and portability of the hCODE platform, we used two totally different hardware systems. One is a desktop platform with an Intel Core i7-2600K processor, 16 GB DDR3 memory, and a vc707 FPGA evaluation board plugged into a Gen2 PCIe interface with 8 lanes. The desktop OS is Ubuntu 14.04. A Xillybus shell and a RIFFA shell are ported for the desktop environment. The other platform is a DIGILENT Zedboard, which is based on the ZYNQ programmable system-on-a-chip and has 512 MB DDR3 memory. The OS on the Zedboard is Ubuntu 12.04 and a Xillybus shell is prepared for it. **Table 2** shows the interface between shell and IP. The hardware-design CAD system is Xilinx Vivado 2015.4.

5.2 Loopback

The loopback IP simply echoes data from the input FIFO port back to the output FIFO port (Table 2), which is used to test the design flow and evaluate shell performance. This IP is implemented with Vivado HLS. We made the *configure* script for the data width and clock frequency parameters scaling. These parameters are specified in the *shell* section of the IP SPEC file, so that the IP can be compiled with all three shells automatically with hCODE.

Table 3 shows the implementation results. Comparing different shells and devices are helpful for the shell selection. Throughput dependent application should consider *shell-vc707-riffa2-ap_fifo128* for the high performance. However, because Xillybus includes ZYNQ FPGA shells, a broader range of devices are supported (such as Zedboard). In addition, drivers are also different. An IP developer has to include the RIFFA library to communicate with the hardware, while the Xillybus provides a set of convenient device files that can be accessed by common file operation from any language. Therefore, two drivers for RIFFA and Xillybus shells are prepared in the loopback IP *driver* folder for the software portability.

5.3 AES Encryption

The AES is an encryption algorithm that widely adopted in

Table 3 Implementation results of the loopback IP.

Shell Name	CLK (MHz)	LUT Used	BRAM Used	Throughput* (Max MB/s)
shell-vc707-xillybus-ap_fifo32	250	6965(2%)	11(1%)	800**
shell-vc707-riffa2-ap_fifo128	250	10487(3%)	44(4%)	3675.0
shell-zedboard-xillybus-ap_fifo32	200	4628(9%)	6(4%)	300**

*The total reading and writing band width of the PCIe bus.
**Official data is used. Because according to the Xillybus document, a loopback IP cannot show the maximum throughput of Xillybus [5] PCIe core for design reasons.

applications from embedded system to data center. We used an open-source AES-128-ECB core [15] to examine the portability of IPs on hCODE.

The original AES core has an interface of FIFOs for I/O data streaming and a 128-bit key signal port. This interface fits Xillybus shells, however, the RIFFA shell only has FIFO ports. We implemented an FIFO interface adapter for the original core with Vivado HLS. The data width of the adapter can be scaled to 32 or 128, so that the AES IP is implementable with all three shells. We also provided drivers for Xillybus shell and RIFFA shell in the IP project. We first developed this IP on *shell-vc707-riffa2-ap_fifo128*. Then in the *compatible_shells* section of *shell-vc707-riffa2-ap_fifo128*, we claimed that this shell is compatible with *shell-vc707-xillybus-ap_fifo32* and *shell-zedboard-xillybus-ap_fifo32*. Therefore, the hCODE can find all three shells when implementing the IP. Because the AES IP does not have scaling parameters, not much manual operations are required for the accelerator compiling.

The evaluation is performed on two hardware environments and three shells by encrypting a 8 MB data file. On the desktop, a pure Java evaluation of the same AES algorithm from javax.crypto.Cipher (OpenJDK1.8) package showed a 110.4 MB/s throughput. On the other hand, the throughput of *shell-vc707-xillybus-ap_fifo32* and *shell-vc707-riffa2-ap_fifo128* implemented accelerators were 559.2 MB/s (5×) and 932.1 MB/s (8.4×), respectively. On the zedboard, the *shell-zedboard-xillybus-ap_fifo32* implemented IP showed a 56.3 MB/s throughput, which achieved 264 times the speed of Java (OpenJDK1.6) on this low-end ARM processor. This case shows the hCODE can speedup hardware and software co-design on both high-performance and embedding systems under the same platform.

5.4 Merge-sorter

Sorting is a fundamental task in computer science. Software sorters are implemented in almost all programming environments. Therefore, we built a scalable merge-tree sorter IP to explain how to integrate an accelerator with software in low cost.

For the IP development, we selected the *shell-vc707-xillybus-ap_fifo32* shell for few reasons. First, Xillybus can provide enough throughput for the implemented merge-tree. Second, the 32-bit FIFO is convenient for Int32 numbers operation. And last, we need the mem port for the ease of controlling. After determining the shell, we implemented and tested a 4-number merge-tree IP using Vivado HLS based on the shell. We then extended this small IP to a scalable IP generator script, which can produce merge-tree HLS code of any input size according to a given argument.

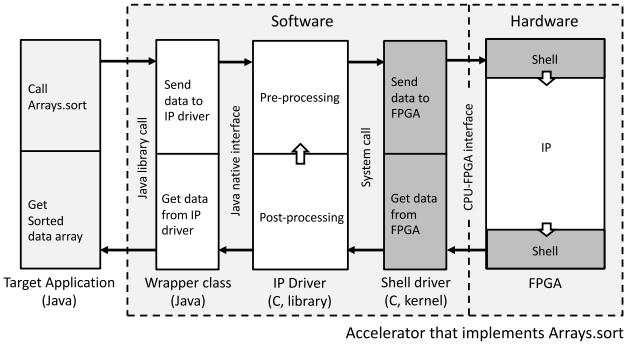

Fig. 6 Example of the hCODE acceleration data flow.

The data flow of the whole system is shown in **Fig. 6**. IP drivers and Java class wrappers are developed. The merge-tree IP driver receives data, recursively sends the data to the FPGA until all data is sorted, and then finally returns the sorted data to the application. The Java class wrapper passes data between the target application and the IP driver, which implements the same interface as the *Arrays* class of the OpenJDK core class library. Therefore, the whole accelerator can seamlessly replace the original *Arrays* class. With the help of hCODE, a software developer can easily search, download and compile the proposed accelerator. After installation, the original software *Arrays* class can be replaced with the hardware Arrays class by using the *-Xbootclasspath* option of Java HotspotVM, so that the target application can be accelerated without modification.

In the acceleration evaluation, we compared the original Java *Arrays.sort*, the merge sorter implemented in C and the merge-tree accelerator. The *Arrays.sort* method in OpenJDK 8 implements a high performance dual-pivot quick-sort algorithm, which has similar performance to the C merge sorter. The 2^{21} Int32 numbers take only three rounds of sorting by the 128-number merge-tree. The proposed accelerator was 2.0 times the speed of Java, and 1.8 times the speed of C. The achieved throughput was 470 MB/s.

6. Conclusion and Future Works

In this paper, we proposed the hCODE platform for high-efficiency low-cost hardware and software co-design. This is achieved by implementing the methods and tools for running an open-source platform for accelerators. In addition, a hardware open-source methodology is proposed for reducing the software integration costs of accelerators. The evaluation showed

hardware acceleration can be easily achieved with hCODE. The hCODE platform is still under development at present. In future work, we intend to continue to provide more useful features and documents for hCODE and to contribute to the open-source community. More useful open-source IPs are also in development for the online repository.

References

[1] Xilinx: Vivado Design Suite User Guide, High-Level Synthesis, Xilinx UG902 (v2015.1) (2015).
[2] Calagar, N., Brown, S. and Anderson, J.H.: Source-Level debugging for FPGA high-Level synthesis, *IEEE International Conference on Field-Programmable Logic and Applications* (*FPL*) (2014).
[3] Jacobsen, M., Richmond, D., Hogains, M. and Kastner, R.: RIFFA 2.1: A reusable integration framework for FPGA accelerators, *ACM Trans. Reconfigurable Technology and Systems*, Vol.8, No.4, Article 22 (2015).
[4] Putnam, A. et al.: A reconfigurable fabric for accelerating large-scale datacenter services, *ACM/IEEE 41st International Symposium on Computer Architecture* (*ISCA*), pp.13–24 (2014).
[5] XILLYBUS Ltd., available from ⟨http://xillybus.com⟩.
[6] Vipin, K., Shreejith, S., Gunasekera, D., Fahmy, S.A. and Kapre, N.: System-level FPGA device driver with high-level synthesis support, *Proc. 2013 International Conference on Field Programmable Technology* (*ICFPT*), pp.128–135 (2013).
[7] OpenCores, available from ⟨http://opencores.org⟩.
[8] Cocoapods, available from ⟨https://cocoapods.org⟩.
[9] Xilinx SDAccel, available from ⟨http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html⟩.
[10] Altera SDK for OpenCL, available from ⟨http://dl.altera.com/opencl/⟩.
[11] Caspi, E., Chu, M., Huang, R., Yeh, J., Wawrzynek, J. and DeHon, A.: Stream Computations Organized for Reconfigurable Execution (SCORE), *International Workshop on Field-Programmable Logic and Applications* (*FPL*), pp.605–614 (2000).
[12] Peck, W., Anderson, E., Agron, J., Stevens, J., Baijot, F. and Andrews, D.: Hthreads:A Computational Model For Reconfigurable Devices, *IEEE International Conference on Field-Programmable Logic and Applications* (*FPL*), pp.1–4 (2006).
[13] Ma, S., Ding, H., Huang, M. and Andrews, D.: Archborn: An open source tool for automated generation of chip heterogeneous multiprocessor architectures, *2015 International Conference on ReConFigurable Computing and FPGAs* (*ReConFig*), pp.1–6 (2015).
[14] AMD Aparapi, available from ⟨http://developer.amd.com/tools-and-sdks/opencl-zone/aparapi/⟩.
[15] Salah, A.: available from ⟨http://opencores.org/project,aes-128_pipelined_encryption⟩, overview, (2016).

Qian Zhao received his B.E. degree in the College of Automation and Electronic Engineering from Qingdao University of Science and Technology, China, in 2007. Further, he received his M.E. and D.E. degrees in Computer Science and Electrical Engineering from Kumamoto University in 2011 and 2014, respectively. He is now a postdoctoral researcher at Kumamoto University. His research interests include architecture and design methods of reconfigurable computing systems. He is a member of IEICE.

Motoki Amagasaki received his B.E. and M.E. degrees in Control Engineering and Science from Kyushu Institute of Technology, Japan in 2000, 2002, respectively. He was a design engineer at NEC Micro Systems Co., Ltd. from 2002 to 2005. He received his D.E. degree from Kumamoto University, Japan, in 2007. He had been an assistant professor in the Department of Computer Science at Kumamoto University until 2007. He has been an assistant professor in the Faculty of Advanced Science and Technology at Kumamoto University since 2016. His research interests include reconfigurable system and VLSI design. He is a member of IPSJ, IEICE and IEEE.

Masahiro Iida received his B.E. degree in Electronic Engineering from Tokyo Denki University in 1988. He was a research engineer at Mitsubishi Electric Engineering Co., Ltd. from 1988 to 2003. He received his M.E. degree in Computer Science from Kyushu Institute of Technology in 1997. Further, he received his D.E. degree from Kumamoto University, Japan, in 2002. He was an associate professor at Kumamoto University until 2015, and during 2002–2005, he held an additional post as a researcher at PRESTO, Japan Science and Technology Corporation (JST). He has been a professor in the Faculty of Advanced Science and Technology at Kumamoto University since January 2016. His current research interests include high-performance low-power computer architectures, FPGA computing, VLSI devices and design methodology. He is a senior member of IPSJ and the IEICE, and a member of IEEE.

Morihiro Kuga received his B.E. degree in Electronics from Fukuoka University in 1987 and M.E. and D.Eng. degrees in Information Systems from Kyushu University in 1989 and 1992. From 1992 to 1998, he was a lecturer at the center for Microelectronic Systems, Kyushu Institute of Technology. He has been an associate professor of computer science at Kumamoto University since 1998. His research interests include parallel processing, computer architecture, reconfigurable system, and VLSI system design. He is a member of IPSJ and IEICE.

Toshinori Sueyoshi received his B.E. and M.E. degrees in Computer Science and Communication Engineering from Kyushu University in 1976 and 1978 respectively. From 1978 to 1987, he was a research associate at Kyushu University, where he received D.E. degree in 1986. From 1987 to 1989, he was an associate professor of Information System at Kyushu University. From 1989 to 1997, he was an associate professor of Artificial Intelligence at Kyushu Institute of Technology. Since 1997 he has been a professor of Computer Science at Kumamoto University. His primary research interests include computer architecture, reconfigurable computing, parallel processing. He served as Chair of the Technical Committee on Reconfigurable Systems of the IEICE, Chair of the Technical Committee on Computer Systems of the IEICE, Chair of the IEEE Computer Society Fukuoka Chapter, Chair of the IEEE CAS Society Fukuoka Chapter, and Director of the IPSJ Kyushu Chapter. Currently he also serves as Director-Elect of the IEICE Kyushu Chapter. He is a fellow of IEICE and a senior member of IPSJ.

(Recommended by Associate Editor: *Tetsuo Hironaka*)