A Unified Hardware/Software MPSoC System Construction and Run-Time Framework

Sam Skalicky*†, Andrew G. Schmidt†, Sonia Lopez* and Matthew French†
*Department of Computer Engineering
Rochester Institute of Technology
{sxs5464,slaeec}@rit.edu
†Information Sciences Institute
University of Southern California
{aschmidt,mfrench}@isi.edu

***Abstract*—With the continual enhancement of heterogeneous resources in FPGA devices, utilizing these resources becomes a challenging burden for developers. Especially with the inclusion of sophisticated multiple processor system-on-chips, the necessary skill set to effectively leverage these resources spans both hardware and software expertise. The maturation of high level synthesis tools and programming languages aim to alleviate these complexities, yet there still exist systematic gaps that must be bridged to provide a more cohesive hardware/software development environment. High level MPSoC design initiatives such as Redsharc have reduced the costs of entry, simplifying application implementation. We propose a unified hardware/software framework for system construction, leveraging Redsharc's APIs, efficient on-chip interconnects, and run-time controllers. We present system level abstractions that enable compilation and implementation tools for hardware and software to be merged into a single configurable system development environment. Finally, we demonstrate our proposed framework with Redsharc, using AES encryption/decryption spanning software implementations on ARM and MicroBlaze processors and hardware kernels.**

I. Introduction

As FPGAs continue to incorporate higher performing heterogeneous resources along with entire multiprocessor system-on-chips (MPSoCs) the demand to exploit these capabilities increases. However, even experienced hardware developers can spend an exhausting amount of time and energy fine-tuning a design to maximize for performance or power. Even as tools and languages mature, such as High Level Synthesis (HLS) and OpenCL, the entire system's performance can hinge on memory bandwidth utilization, bus/interconnect topologies, or how hardware accelerator cores communicate with software processes. With the configurable nature of FPGAs, developers have almost endless design flexibility, requiring more effort to produce a high performing functional system.

System-level development needs a unified hardware/software framework to help orchestrate harmonious development and implementation such that a final application will effectively run on the target platform. Such a framework should not aim to reinvent existing approaches or tools, but leverage and supplement them with the necessary infrastructure to enable effective system creation. We present, the Reconfigurable Data-Stream Hardware Software Architecture (*Redsharc*), which provides a cohesive hardware/software build and run-time environment for FPGAs, allowing developers to design systems of simultaneously executing kernels in software or hardware communicating across a seamless interface.

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001-11-C-0041. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement A (Approved for Public Release, Distribution Unlimited).

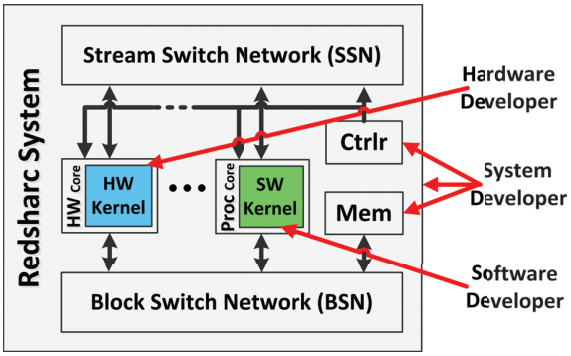


Figure 1: Redsharc development provides build infrastructure enabling developers to focus efforts on kernel implementations

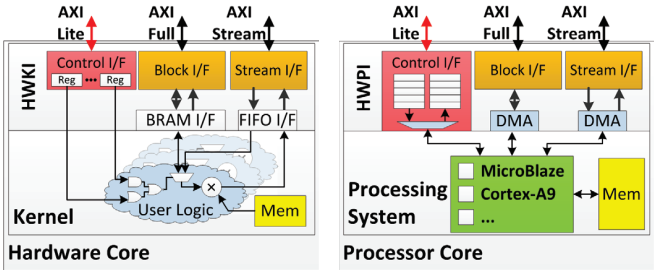
We introduce a *build framework*, taking hardware/software implementations and hardware specifications, that automatically compiles, synthesizes, and generates the heterogeneous hardware/software MPSoC. The system produced is fully functional, requiring no additional user input to setup or configure the design. To demonstrate the framework, four systems have been assembled targeting Xilinx Virtex-7 and Zynq-7000 devices for both software only and hardware/software co-design integration. Results include the run-time for Redsharc's control kernel to manage execution of AES kernels in software and hardware, spanning both MicroBlaze and ARM processors, along with the full application run-time.

The rest of this paper is organized as follows. Section II discusses the relevant related works. Section III present enhancements to Redsharc that support our detailed build and construction flow, which is presented in Section IV. Next, example system implementations in Section V are demonstrated. Section VI summarizes our contributions and describes our continuing efforts for the future.

II. Background and Related Work

This paper presents our efforts to develop a unified hardware/software MPSoC system construction framework, providing a higher level development abstraction for the user. We introduce a set of system configuration APIs that extend the existing Redsharc APIs, and present a generation framework to allow rapid and efficient system assembly. The field is full of useful tools and techniques spanning from vendor provided base system builders, to high level synthesis, and OpenCL language support. Redsharc does not aim to supplant existing tools, but instead supplements them to create a more cohesive development and run-time environment in order to better address system level issues.

Tools to enable simpler hardware system construction, such as Xilinx Vivado and Altera Quartus II, enable hardware



(a) Hardware Core *(b) Processor Core*

Figure 2: Redsharc's hardware abstractions and interfaces for kernels and processor compute cores

designers to construct their systems without writing HDL. High Level Synthesis (HLS) tools such as Xilinx Vivado HLS and Catapult-C [1] among others [2] produce hardware cores from high level languages, and have been maturing significantly over the past few years. Other efforts have focused on improvements to the design process for SoCs by implementing a programming model based on Pthreads to abstract the hardware/software boundary. ReconOS [3] and hthreads [4] have demonstrated the feasibility of abstract programming models in heterogeneous systems. Extensions to hthreads include a vendor neutral soft processor design flow [5], an OpenCL approach [6], and extensions for partial reconfiguration in the HEMPS project [7]. OpenDF presents a toolset for a dataflow programming model, which encourages the use of a more stream/data centric model and can be applied to FPGAs [8]. These toolsets stop short of providing a build and run-time infrastructure for system-level design.

III. Redsharc Enhancements

Redsharc was developed to address the challenges associated with communication between tasks running in software on a processor and those running as a hardware accelerator in the FPGA's programmable fabric. These tasks are referred to as *kernels*, or the compute functionality running in software or hardware. Kernels communicate through Redsharc's abstract API, which supports both streaming data transfers and random access to blocks of data. These transmissions occur over the proven, validated, and configurable Redsharc on-chip networks. The API enables developers to compartmentalize kernels and also to rapidly migrate from software to hardware implementations without changing existing kernel code. Our previous work describes the API in more detail, providing examples and performance results [9][10][11].

Improved Interfaces. Redsharc provides several interfaces to simplify kernel development. The *hardware kernel interface* (HWKI), as shown in Figure 2a, provides stream and block support to hardware kernels in a simple to use interface. The HWKI has been updated with AXI4 interfaces to provide easier communication to leverage AXI's full and streaming interfaces, without requiring a developer to manage the signaling in their design. The *hardware processor interface* (HWPI), as shown in Figure 2b, has been newly added to abstract the connections from any hard or soft processor core, and provide DMA components to interface between the processor cores and the Redsharc system. The HWPI provides a simple mechanism for vendor agnostic development since different FPGA vendors have support for different soft and hard processors (PowerPC, ARM, MicroBlaze, NIOS, etc). The *software kernel interface*

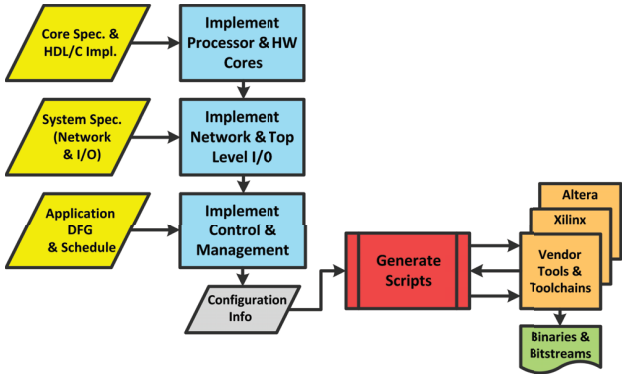


Figure 3: Redsharc build framework flow diagram

(SWKI), allows software kernels to run on any hard or soft processor core and is a collection of APIs that mimic hardware for seamless hardware/software communication. All of which provides a substantial step towards increasing the level of abstraction for the developer.

Expanding Network Capabilities. To facilitate rapid construction of MPSoC systems while still providing high performance communication, Redsharc provides a block switch network (BSN) and a stream switch network (SSN) to transfer data via modes as needed by the application. We abstract the complications with bus mastering, addressing, and signaling from the kernel and system developers by providing a simple API representing FIFO and BRAM transactions for streams and blocks accordingly. In this work the networks have been extended to leverage the industry standard AXI4 full, streaming, and lite interfaces along with support for alternative on-chip interconnect topologies.

Independent Run-Time Controller. Leveraging the HWPI, Redsharc can utilize a single processor to act as the run-time controller core, responsible for system management tasks such as starting and stopping each kernel in the system among others. The controller also configures the networks' communication paths as needed for the application. Both the BSN and SSN each have a single AXI4 control port to provide register interfaces to configure the BSN and SSN for data communication between cores. The control interfaces are standardized for hardware and processor cores. The HWKI provides registers for starting, stopping, resetting and checking the state machine in the hardware. The HWPI includes an AXI Mailbox that provides two FIFOs in each direction to allow control commands to be issued from the control core to the processor core, and responses and notifications from the processor core back to the control core. With the HWPI, Redsharc can now instantiate a light-weight soft processor for controlling the system, rather than splitting execution on a larger, more capable processor that should be allocated for software kernels, such as the ARM Cortex-A9 cores.

IV. MPSoC Build Framework

The construction of MPSoCs can incur long development time when dealing with memory interfaces, PCIe or other high-speed transceiver IP blocks, and low-level signaling for buses or on-chip interconnect protocols. Redsharc aims to provide both software and hardware designers a simplified development environment, shifting the focus from system

Table I: Redsharc System API

Function Name	Arguments	Description
initSystem	int numProcs	Initializes the system for the number of processors specified.
addCore	int coreID int numKernels type coreType config cfg	Adds a core to the system with given ID, number of simultaneous kernels it can execute, with the specified type.
setCapabilities	int coreID config cfg int numCaps type types[]	Sets the capabilities (kernels that it can execute) for the core.

design and integration to application and kernel development. We extend Redsharc to support integration of HLS tools to rapidly implement hardware accelerated kernels along with automatic system control for user defined scheduling policies. Redsharc now incorporates a vendor-agnostic system development environment and build framework, illustrated in Figure 3.

Implement Processor & HW Cores: To start, the user provides core specifications and kernel implementations in a simplified form to enumerate available kernels and compute resources. These kernels utilize Redsharc's APIs, allowing the system to generate hardware cores or software tasks via the HWKI or SWKI. The user provided kernel source code is combined with Redsharc's libraries to generate a collection of IP, spanning both hardware cores and software to run on specified processors. The interfaces and kernels are provided as input to the next stage.

Implement Network & Top Level I/O: In the second stage the cores are connected together based on the network specifications, provided as input by the user. The network topology is specified by the user for both the BSN and SSN, such as crossbar, bus, or mesh topologies. In this stage the networks are configured and VHDL source files are generated for the system. Any custom top-level I/O specified by the user will be passed directly through to the necessary cores. Memory controllers are also connected directly to the networks to provide high bandwidth, low latency connectivity for the system. The overall configuration is passed to the third stage to assist in managing connections at run-time.

Implement Control & Management: In the final stage the controller core is added to the design. The controller uses the specific system configuration information to control the connections between hardware cores during run-time. At the end of this stage the configuration information about the newly specified Redsharc system is ready to be passed into the Redsharc Generate script, producing the necessary vendor specific project files used to compile, synthesize, and implement the design into software binaries and hardware bitstreams.

Specifying System Configuration begins by determining the number of cores and types of cores to implement in the system. Then, leveraging the Redsharc System API a developer can quickly assemble, generate, and test the system on the device. This approach allows for rapid development and testing along with providing vendor-agnostic implementations for ease of platform migration. The Redsharc System API is shown in Table I. Through this API the developer can easily specify the important characteristics of the system to implement. The build framework can then take this specification and produce an implemented system or simulation.

Using this simple API, a developer can specify the number

```
1  //allocate space for the top level system configuration data structure
2  struct system_config* config = initSystem(2);
3  //add a processor core to the system that can execute two kernels
4  addCore(0,2,MB,config);
5  kernel_t t0[] = {AES_ENCRYPT, AES_DECRYPT, AES_CHECK};
6  setCapabilities(0, config, 3, t0);
7  //add a hardware core to the system
8  addCore(1,1,HW,config);
9  kernel_t t1[] = {AES_ENCRYPT};
10 setCapabilities(1, config, 1, t1);
11 //initialize the system state to idle
12 initSystemState(config);
```

Listing 1: Example System Implementation

of cores and the capabilities of each core. The Redsharc System API does not require the developer have any HDL, networking, or even any FPGA knowledge since all cores are connected to the BSN, SSN and control core. An example system containing one processor core and one hardware core is shown below in Listing 1. This system is being constructed to perform AES encryption and decryption. The processor core is able to perform any of the two kernels in addition to an *AES_CHECK* kernel for validation. The hardware core is only capable of performing encryption. Notice that both the processor core and hardware core can perform encryption, allowing the scheduling policy to dictate where to assign encryption kernels as needed for best performance.

System Generation with Redsharc incorporates Python scripts to organize and assemble vendor specific project files before leveraging the vendor tools to compile and implement the design. The Redsharc API provides the common interface for the tools to identify and connect cores together with the BSN and SSN networks, create Xilinx, Altera, or Achronix project files for synthesis and implementation. While the build framework does not expedite vendor tool flow execution, it does reduces the complexity for a designer such that backend scripts can ease overall system development.

V. Building a System: An Example

To demonstrate the Redsharc build framework, four systems have been constructed targeting Xilinx Virtex-7 and Zynq-7000 devices for both software only and hardware/software co-design integration. These systems target both hard and soft processor cores utilizing compilation tool chains from two different vendors (ARM: armcc and armlink, Xilinx: gcc).

A four kernel encryption application has been run on each system, consisting of reading some data from memory (the generate kernel), encrypting and transmitting the packet, receiving a packet and decrypting it, and performing a check to validate the same plaintext data was decrypted correctly. The application was implemented entirely in software initially, and executed on the single processor core systems. Then the encryption kernel was marked to be implemented in hardware and the system regenerated. Then the application was executed again utilizing both the processor and hardware cores.

Hardware Kernel Generation. The same C code that was provided for the software encryption kernel was synthesized using Vivado HLS to produce a hardware kernel. The only directives that are automatically applied by the build framework by default specify a handshake interface for the control of the kernel. Vivado HLS will produce a standard memory interface for any arguments at the top function level. For the encryption kernel, these were the plaintext, key, and resulting ciphertext.

These three memory interfaces were connected directly to the BRAM interfaces available in the HWKI.

Processor Core Generation. To show the simplicity of the system API, the processor core's architecture was specified to be either MicroBlaze or ARM. Both processor cores were connected to the external DDR memory on the board through their cache hierarchies. Both processor cores were connected to a HWPI with a single DMA controller connected to the BSN and another DMA controller connected to the SSN.

Networks and Data Transfer. Two methods of data transfer were used in these experiments: writing into BRAM blocks through the BSN, and through virtual blocks created within the processor core's DDR memory. When two software kernels that are assigned to the same processor core need to communicate there is no reason for that data to leave the cache hierarchy and enter into the on-chip network. Instead, memory allocation commands were implemented in the processor control protocol to allow blocks to be created in a processor's memory. Since data transfer through the BSN passes through the DMA controller the control core only needs to provide a pointer in the processor's local address space to direct it where to write data. Whether this is an address in DDR or the address of the DMA controller, it is transparent to the software kernel, allowing the execution to be finely tuned dynamically by the control kernel to optimize performance during run-time.

Experimental Results. The system initialization procedure first boots the processor core followed by the control core. The boot time for the control core was 90ms compared to the compute cores: 56μs boot time for the MicroBlaze processor core and 30μs for the ARM core. Table II shows execution times for selected operations.

The four kernel AES application initially executes on a PC using the Redsharc API to provide a performance baseline of 112.3ms. Migrating to the FPGA and executing in the a dual MicroBlaze system on the Virtex-7 device resulted in an execution time of 103.6ms. Accelerating the encryption kernel as a hardware kernel achieved an entire application runtime of 98.2ms. This speedup was achieved by making a simple change to direct the build framework to produce a hardware core from the software implementation, requiring no HDL development by the user.

A Zynq system with ARM processor cores further accelerated this execution, running in 90.79ms. Since the size of the data being transferred was only 128-bytes for the plaintext, key, and ciphertext the data transfer rate achieved was only 35.5 MBps using a 32-bit wide datapath. The focus of this work is not strictly on HLS accelerated performance, but the ability to easily migrate from a pure software implementation to multiple hardware accelerated kernels, with minimal changes to software kernels in the system. Further performance could be obtained by implementing custom hardware kernels in place of those generated by Vivado HLS or by applying further performance enchancing directives such as loop unrolling or pipelining.

Table II: System performance with Redsharc build framekwork

Operation	Elapsed Time
MB Control Bootup	90ms
MB Control Send Message	1.5μs
MB Control Receive Message	1.6μs
ARM Compute Send Message	0.9μs
ARM Compute Receive Message	0.9μs
ARM Compute 128-Byte Block Transfer	3.6μs
ARM Compute AES generate kernel	6.5μs
ARM Compute AES encrypt kernel	73.8μs
ARM Compute AES decrypt kernel	298.3μs
ARM Compute AES check kernel	6.7μs
Hardware Compute AES encrypt kernel	42.9μs

VI. Conclusion

This paper presents a build framework for unified hardware/software MPSoC system construction. The framework incorporates the Reconfigurable Data-Stream Hardware Software Architecture (Redsharc) infrastructure to reduce system design efforts. Incorporating Redsharc's hardware and software APIs, stream and block switch networks, and controller core, the build framework allows developers to spend more time on kernel development rather than spending significant time and effort to manage memory accesses, on-chip interconnect protocols, and other system-level development complexities. The framework has been demonstrated on several running systems and showcases the capability to incorprate both vendor tools, high level synthesis tools, and Redsharc's run-time system across both MicroBlaze and ARM based systems.

References

[1] C. Economakos and G. Economakos, "FPGA Implementation of PLC Programs Using Automated High-Level Synthesis Tools," *IEEE International Symposium on Industrial Electronics*, June 2008.

[2] J. I. Villar, J. Juan, M. Bellido, J. Viejo, D. Guerrero, and J. Decaluwe, "Python as a Hardware Description Language: A Case Study," *Southern Conference on Programmable Logic*, Apr. 2011.

[3] E. Lubbers and M. Platzner, "Reconos: An rtos supporting hard-and software threads," *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 441–446, 27-29 Aug. 2007.

[4] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden, "Programming Models for Hybrid FPGA-CPU Computational Components: A Missing Link," *IEEE Micro*, vol. 24, no. 4, July 2004.

[5] E. Cartwright, A. Fahkari, S. Ma, C. Smith, M. Huang, D. Andrews, and J. Agron, "Automating the Design of mLUT MPSoPC FPGAs in the Cloud," *International Conference on Field Programmable Logic and Applications*, Aug. 2012.

[6] S. Ma, M. Huang, and D. Andrews, "Developing Application-specific Multiprocessor Platforms on FPGAs," *International Conference on Reconfigurable Computing and FPGAs*, Dec. 2012.

[7] E. Cartwright, A. Sadeqian, S. Ma, and D. Andrews, "Achieving Portability and Efficiency over Chip Heterogeneous Multiprocessor Systems," *International Conference on Field Programmable Logic and Applications*, Sept. 2014.

[8] S. S. Bhattacharyya, G. Brebner, J. W. Janneck, J. Eker, C. von Platen, M. Mattavelli, and M. Raulet, "OpenDF: A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems," *SIGARCH Computer Architecture News*, vol. 36, no. 5, June 2009.

[9] A. Schmidt, W. Kritikos, R. Sass, E. Anderson, and M. French, "Merging Programming Models and On-chip Networks to Meet the Programmable and Performance Needs of Multi-core Systems on a Programmable Chip," *International Conference on Reconfigurable Computing and FPGAs*, Dec. 2010.

[10] W. Kritikos, A. Schmidt, R. Sass, E. Anderson, and M. French, "Redsharc: A Programming Model and On-Chip Network for Multi-Core Systems on a Programmable Chip," *International Journal of Reconfigurable Computing*, 2012.

[11] S. Skalicky, A. G. Schmidt, and M. French, "High Level Hardware/Software Embedded System Design with Redsharc," *International Workshop on FPGAs for Software Programmers*, Sept. 2014.