Portable Library Development for Reconfigurable Computing Systems

Proshanta Saha, Esam El-Araby, Miaoqing Huang, Mohamed Taher, Tarek El-Ghazawi,
The George Washington University

Chang Shu, Kris Gaj,
George Mason University

Alan Michalski, and Duncan Buell
University of South Carolina

Acknowledgment

- SRC
  - Dan Poznanovic
  - David Caliga
  - Jeff Hammes
  - Paul Gage
- Cray
  - Dave Strenski
  - Greg Woods
- SGI
  - Brian Larson
  - Chris Lindahl
  - Matthias Fouquet-Lapar
  - Alan Mayer
  - Bruce Losure
  - Dick Riegner

Outline

- Introduction
- Background
- Creating Portable Libraries for RCs
  - Domain Analysis
  - Resolving Porting Concerns
  - Standardizing Interface
  - Benchmarking and Testing
  - Establishing Mechanisms for Collaboration
  - Distribution and Licensing
- Case Study
- Concluding Remarks

Introduction

- Constantly changing technology in RC domain pose significant challenges
  - Lack of standard interfaces make porting applications to new architectures difficult
  - Developers face constantly changing design languages and tools

- Code re-write even in the same application domain prevalent
  - Lack of standard optimized libraries for RC systems
  - Hardware implementations difficult to generalize

Outline

- Introduction
- Background
- Creating Portable Libraries for RCs
  - Domain Analysis
  - Resolving Porting Concerns
  - Standardizing Interface
  - Benchmarking and Testing
  - Establishing Mechanisms for Collaboration
  - Distribution and Licensing
- Case Study
- Concluding Remarks

Background

- Library efforts taken on by several groups
  - Chip manufacturers
    - Xilinx, Altera, Chameleon, etc.
  - System and board vendors
    - SRC, SGI, Cray, Nallatech, AMS, Starbridge, etc.
  - Independent Software Vendors (ISV)
    - Impulse, Mitrionics, DSPLogic, Celoxica, etc.
  - RC Users and Enthusiasts
    - OpenCores, etc.

- Porting to current architectures and platforms is non-trivial
  - Different data/memory access standards
  - Varying number, type, and size of memory banks
  - Tool chain support not uniform across platforms

Outline

- Introduction
- Background
- Creating Portable Libraries for RCs
  - Domain Analysis
  - Resolving Porting Concerns
  - Standardizing Interface
  - Benchmarking and Testing
  - Establishing Mechanisms for Collaboration
  - Distribution and Licensing
- Case Study
- Concluding Remarks

Creating Portable Libraries for RCs

- Key concerns for portable library development
  - Library scope and utility
  - Performance vs. Portability
  - Porting concerns and standardization
  - Collaboration and Intellectual Property (IP)
  - Distribution format
  - Licensing worries

Domain Analysis

- Limiting scope of the library
  - Domain selection
    - Biomedical, Cryptography, Image and Signal processing, etc.
  - Target applications
    - Impact on scientific community
  - Analyzing application characteristics
    - IO Bound, Memory Bound, etc.

- Finding balance between performance and re-use
  - Analyzing applications for overlap in functionality
  - Analyzing utility of reusable core
  - Effort required to by end user to instantiate efficient cores

Addressing Porting Concerns

- Design cores with limited hardware assumptions
  - Cores should meet a minimum 100MHz (10ns) time constraint
  - Little to no assumption of hard cores should be made
    - DSPs, PowerPC, FP units etc.
    - Conservative assumption of BRAM size
    - No assumption to the types of multipliers (MULTS, etc)
- Development language
  - Design cores in HDL to ensure portability and limit duplication of efforts
  - Assume netlist support universal across platforms
- Hardware Core Documentation
  - Hardware assumptions taken (ex. multiplier size, BRAM size)
  - Latency, clock speed, pipeline length, delays, optimization clues
- Limitations
  - Universal portability is unrealistic, even from same vendor
  - Designing cores to work around current hardware drawbacks will limit portability

Standardize Interface Definition

- Create interface definition guidelines
  - Improve usability, readability, debugging, etc
  - Increase collaboration
- Define handshaking protocols
  - Busy, Wait, Done, Bypass ports
- Build in performance measurement mechanism
  - Timer probes
- Modularize hardware cores
  - Standardize bus interface
  - Ease partial reconfiguration efforts

Benchmarking and Testing

- Benchmarks are important to identify portability vs. performance trade off
  - End users can select hardware cores that best suit their needs
  - Allows end users and contributors to predict performance on target hardware
- Tests are essential to hardware library builds
  - Sanity tests provide quick verification and sufficient for small errors
  - Regression tests are more thorough and can identify problems:
    - Tool chain
    - Cross library compatibility
    - Performance
  - Frequent builds can identify compatibility issues

Collaboration

- Source control software selection
  - Project size
  - Collaboration type (multiple sites, etc)
  - Features required
  - Vendor compatibility

- Access control and security
  - Protecting user contributions
  - Protecting early access contributions
  - Protecting IPs

Distribution and Licensing

- Two approaches to library distribution
  - Vendor specific distribution release
    - Immediately usable in target architecture
    - Optimized performance on target architecture
  - Source code distribution release
    - Requires end users to create necessary wrappers for target architecture
    - Optimization depends on users knowledge of target architecture
- Choosing a licensing agreement
  - Open source license
  - Non-profit use license
  - Proprietary license
- License compatibility
  - Confusing precedence rules on open source licenses
  - Bundling packages with different licensing agreements

Outline

- Introduction
- Background
- Creating Portable Libraries for RCs
  - Domain Analysis
  - Resolving Porting Concerns
  - Standardizing Interface
  - Benchmarking and Testing
  - Establishing Mechanisms for Collaboration
  - Distribution and Licensing
- Case Study
- Concluding Remarks

Case Study

- Platforms used in Case Study:
  - SRC6
  - SGI RC100 RASC
  - Cray XD1
- Library Packages
  - Platform specific binary release for SRC6
  - Platform independent source code release
  - Available at http://hpc.gwu.edu/library

Testbed Specifications

Platform	Number of FPGAs	FPGA Type	Maximum Frequency	RC System vs. Workstation		
				Cost	Power	Size
SRC6	4	XC2V6000	100MHz	200x	3.64x	33.3x
Cray XD1	6	XC2VP50	200MHz	100x	20x	95.8x
SGI RC100	6	XC4LX200	200MHz	400x	11.2x	34.5x

Domain Analysis

- Application domains selected in study include:
  - Secret Key Ciphers
  - Binary Galois Field Arithmetic
  - Elliptic Curve Cryptography
  - Long Integer Arithmetic
  - Image Processing
  - Bioinformatics
  - Matrix Arithmetic
- Library scope narrowed with assistance from domain experts and RC community input
  - Applications chosen based on impact on domain
  - Application characterization (I/O bound, Memory bound, etc) also played key role in selection
- Core breakdown based on:
  - Domain application analysis (overlap among applications, etc)
  - Effort required by developer to create efficient designs using re-usable core from library
  - Core utility (number of applications that can benefit from core)
  - Resource utilization

Portability Concerns

- Library cores developed in either Verilog or VHDL
  - SRC6 Carte allows user macros to be defined in HDL
  - SGI RC100 Core Services provides top level modules to support user application written in HDL
  - Cray XD1 ufpapps provides scripts to support user application written in HDL
- Library cores meet a minimum clock speed of 100Mhz (10ns)
  - SRC6 supports 100MHz
  - SGI RC100 and Cray XD1 support clock speeds upto 200MHz

Portability Concerns (2)

- Assumptions made on platforms
  - Availability of basic hard core arithmetic units and modest BRAM size
  - No assumptions of hard cores such as PPC, DSP, Rocket IO, etc
  - No assumptions made on memory banks and peripherals
    - Assumes vendor cores handle data transfer and I/O to/from peripherals
- A minimum set of files provided for each library core helps porting efforts:

```
.v/.vhd          for Verilog/VHDL source code
.c/.cpp          for Host C/C++ source code
.blk             for black box interface file
.tex/.PDF        for LaTeX/PDF documentation
test.c/.cpp      for test code
Makefile         for Makefile
```

- Separate auxiliary library necessary
  - Namespace clashes for performance and timing functions
  - Overloading not supported in C language
- Choosing a flexible and standard documentation format proved to be difficult
  - LaTeX chosen for developer documentation to allow for automated updates
  - PDF chosen for end user documentation

Collaboration

- Source control systems explored
  - Concurrent Versions System (CVS)
  - Subversion (SVN)
- CVS chosen primarily because of vendor support
  - SRC Carte provides library build process
    - Libraries built can be directly called without special setup procedures
    - Regression tests and library builds can be stream lined
- Access control is enforced into three categories
  - Development branch for latest alpha code
  - Main library branch for release quality code
  - IP and legacy branch for contributed code

Benchmark and Testing

- Testing proved to be essential in RCLib build process
  - Short sanity tests allowed for quick verification
  - Longer regression tests were essential:
    - The number of tools in the tool chain requires thorough compatibility tests
      - Synplicity, Xilinx PAR, C/C++ compilers, GNU Multi-precision (GMP) library, LiDIA library, OpenSSL library
    - The number of libraries created, eight in total, required cross compile tests for compatibility
      - Utilizing Binary Galois Field (GF2m) libraries with Elliptic Curve Cryptography Library(ECC) for example showed name space clashes in auxiliary functions (performance and timing)
  - Frequent builds allowed for vendor library and user library compatibility checks

Benchmark and Testing (2)

- Benchmarking provided a performance gauge for RCLib
  - Allows developers to gauge if library core resource utilization and performance
  - Allows end users to see if upgrading to newer library release will provide additional benefits

- Benchmarking provided in two formats
  - A quick benchmark to ensure that the general functionality of the core passes
  - A thorough benchmark to ensure proper handling of all functionality, data types, and performance are tested

Licensing and Distribution

- Distribution formats
  - Platform specific distribution – currently available for SRC6
  - Platform independent distribution – HDL source code, black box files, documentation, test code, and test vectors
- Licensing challenges to packaging, a precedence issue
  - Xilinx soft cores – Xilinx proprietary license
  - SRC library support – SRC proprietary license
  - GMP library – GNU GPL
  - OpenSSL – BSD style license
  - LiDIA – non-commercial license
- Third party and proprietary tools and libraries
  - Users will be provided with a list of pre-requisites
  - Software libraries are checked to ensure for maintenance and availability
  - Soft core libraries from vendors and chip manufacturers are assumed to be available and installed
- Licensing chosen is LGPL to allow for the library to be binary compiled with third party libraries

Portable Libraries Developed

http://hpc.gwu.edu/library

RCLib Cores

Secret Key Ciphers	
Application	**Cores**
IDEA	encryption
	decryption
	breaking
DES	encryption
	decryption
	breaking
RC5	key scheduling
	encryption
	decryption
	breaking

Elliptic Curve Cryptography	
Application	**Cores**
Scalar Multiplication	Normal Basis
	Polynomial Basis
Project to Affine	Normal Basis
	Polynomial Basis
Point Addition	Normal Basis
	Polynomial Basis
Point Doubling	Normal Basis
	Polynomial Basis

RCLib Cores (2)

Binary Galois Field Arithmetic		
Application		**Cores**
Polynomial Basis	Trinomial	Squaring
		Multiplication
		Inversion
	Pentanomial	Squaring
		Multiplication
		Inversion
	Special Field	Squaring
		Multiplication
		Inversion
	NIST	Squaring
		Multiplication
		Inversion
Normal Basis	NIST	Squaring
		Multiplication
		Inversion

Image Processing	
Application	**Cores**
Wavelet	Discrete Wavelet Transform
	Inverse Discrete Wavelet Transform
	Correlation and Histograming
2-D Convolution Operators	Gaussian Filter
	Smoothing Filter
	Sharpening Filter
	Blurring Filter
	Prewitt Filter
	Sobel Edge Filter
	Median Filter
Buffering	Line Buffer

RCLib Cores (3)

Long Integer Arithmetic	
Application	**Cores**
1024 bit	Montgomery Multiplier w/ Carry save adder
	Montgomery Multiplier w/ Carry propagate adder
	Modular Exponentiation w/ Carry save adder
	Modular Exponentiation w/ Carry propagate adder
1536 bit	Montgomery Multiplier w/ Carry save adder
	Montgomery Multiplier w/ Carry propagate adder
	Modular Exponentiation w/ Carry save adder
	Modular Exponentiation w/ Carry propagate adder
2048 bit	Montgomery Multiplier w/ Carry save adder
	Montgomery Multiplier w/ Carry propagate adder
	Modular Exponentiation w/ Carry save adder
	Modular Exponentiation w/ Carry propagate adder
3027 bit	Montgomery Multiplier w/ Carry save adder
	Montgomery Multiplier w/ Carry propagate adder
	Modular Exponentiation w/ Carry save adder
	Modular Exponentiation w/ Carry propagate adder

RCLib Cores (4)

Bioinformatics	
Application	**Cores**
Smith-Waterman	Scoring
	Maximum Score Search

Matrix Arithmetic	
Application	**Cores**
Bit-Matrix	Bit-Matrix Multiplication
	Bit-Matrix Transpose

Sorting	
Application	**Cores**
Sorting	Bitonic Sorting
	Stream Sorting
	Odd-Even Sorting
	Heap Sorting
	Quick Sorting
Scheduling	Sorting Scheduler

Library Performance

Performance on SRC6

Application	Speedup	Savings		
		Cost	Power	Size
Smith Waterman	1138	6x	313x	34x
DES Breaker	6757	34x	1856x	95.8x
IDEA Breaker	641	3x	176x	19x
RC5 Breaker	1140	6x	313x	34x

Performance on SGI RC100

Application	Speedup	Savings		
		Cost	Power	Size
Smith Waterman	8723	22x	779x	253x
DES Breaker	38514	96x	3439x	1116x
IDEA Breaker	961	2x	86x	28x
RC5 Breaker	6838	17x	610x	198x

Performance on Cray XD1

Application	Speedup	Savings		
		Cost	Power	Size
Smith Waterman	2794	28x	140x	29x
DES Breaker	12162	122x	608x	127x
IDEA Breaker	2402	24x	120x	25x
RC5 Breaker	2321	23x	116x	24x

Concluding Remarks

- In this work we detailed the proposed methodology for developing hardware library cores
- We outlined the issues related to creating a portable hardware library considering:
  - domain analysis, code reuse, packaging, interface definition, portability, collaboration, benchmarking and testing, distribution and licensing
- Our case study showed that the process requires careful consideration and planning when collaborating with:
  - A large group of developers
  - Different system vendors
  - Different chip vendors
  - Different users
- Future support may include newer architectures from:
  - SRC, Cray, SGI
  - And potentially from DRC, Altera, etc.
- Work is also underway to define a unified layer between the user application core and the vendor interface services