Reconfigurable Computing Middleware for Application Portability and Productivity

Robert Kirchgessner (speaker)
Alan George, Herman Lam

NSF Center for High-Performance Reconfigurable Computing (CHREC)
Dept. of Electrical and Computer Engineering, University of Florida
Gainesville FL, USA
Email: {kirchgessner, george, hlam}@chrec.org

June 6th, 2013
Motivation

- Reconfigurable computing (RC) offers significant *power* and *performance* advantages over fixed-logic CPUs and GPUs
- Lack of standards between FPGA platforms limits RC application and tool *portability* and developer *productivity*
  - Vendor-specific hardware interfaces and software APIs
  - Per-platform support packages for many HLS tools
- Desire to execute RC applications on heterogeneous systems including *cloud computing* platforms

Objective: Enable RC application portability and improve developer productivity using an extensible framework

Approach:
- Develop an *RC middleware* which abstracts away vendor-specific hardware interfaces and software APIs, enabling an *application-centric development environment*
Presentation Outline

- Introduction
- Approach
  - Overview
  - Layered RC Middleware Architecture
  - RC Middleware Toolset
  - Design-flow Example
- Experiments & Results
  - Overhead Analysis
  - Portability Analysis
  - Productivity Analysis
- Conclusions & Future Work
- Q&A
RC App. Development Cycle

- RC application design follows a *formulation, design, translation, execution* cycle
  - Developer should focus on formulation and design
  - Significant effort spent on design translation

- Design and translation are typically tightly coupled tasks
  - Design for specific target platform
  - Limits design portability and removes focus on *ideal* application partitioning

*Handled by RC Middleware framework*
RC App. Development Cycle

HDL Applications:
- Developers design applications targeting specific platform
  - Platform-specific configuration, vendor-specific API and programming models

HLS-tool Applications:
- Two approaches: *platform-agnostic interfaces* and *platform-support packages (PSPs)*
  - Tool vendors cannot provide support for every platform
  - Typically left as challenge for developer

I have to re-code that again?!
RC Middleware (RCMW) Overview

- Layered model consists of different levels of abstraction, each with *extensible features*.
- RCMW toolset handles application *mapping* and *translation* to a target platform.

- Abstract away host- and vendor-specific interfaces
- Maps application to target platform resources
- User/tool RTL and software source code

Developer creates application and describes using XML

Maps user application to target platform, generating HDL/SW top-levels

Stores platform and IP source and XML metadata
Translation Layer

Hardware:

- Converts vendor-specific interfaces into standardized RCMW interface
  - Continuous-burst/FIFO
- Provides separate user-customizable host interface (aka user bus)
  - Address space, data-type etc.

Software:

- C++11 API wrapping vendor API or interfacing with RCMW driver
- Each platform provides implementation of *Board* interface
  - Encapsulates physical layout of platform
  - FPGA objects, Memory objects
- Memory interface exposes template *read(…), write(…)* functions
  - Support any data type or user-defined structure
  - Re-entrant

All RC Middleware interfaces require flow-control due to unknown underlying platform hardware and mapping.
Presentation Layer

Hardware:

- Maps requested application interfaces to physical platform
  - Instantiates arbitration and interface control logic & buffers

Software:

- Provides application-specific instance of Application interface
  - Encapsulates user-requested interface objects
  - Application::bind(…) maps application onto target platform at runtime

Generated by RCMW toolset based on user XML and target platform
Application Layer

Hardware:
- Consists of user or tool-generated RTL code
- Uses interfaces provided by presentation layer
  - Lower-level details abstracted away
- Top-level interface exposing requested resources generated by toolset

Software:
- Consists of user implementation of `Application::execute()` function using provided application interfaces
  - All interfaces are thread-safe
  - Developer must ensure code is re-entrant
- RCMW API library provides many useful classes:
  - Timing
  - Generic buffer object
User-level API

- Provides functions for enumerating available boards, launching Application instances on specific board
  - Supports multiple Application instances running on single platform, assuming developer code is re-entrant

Full API specification available in PDF

```c
int main(int argc, char *argv[])
{
    /* some code excluded */
    timer.push();
    Board board = BoardFactory::create();
    diff_s = timer.pop();

    MultAddStream instance0, instance1;
    instance0.setMultVal(1);
    instance0.setAddVal(1);
    instance1.setMultVal(2);
    instance1.setAddVal(1);

    ApplicationLauncher launcher;
    launcher.launch(instance0, board);
    launcher.launch(instance1, board);
    launcher.join();

    return 0;
}
```
RCMW Toolset Overview

- Generates top-level HDL and software interface
  - User specifies application-description XML, top-level HDL and SW, and target platform
- Handles mapping of application to target platform

rcmw-gen --cost “greedy”
--application MultAddStream.xml
--target “PROCStar III”
Developer specifies application resources via XML description

```
<resources>
    <fpga id="fpga0">
        <sources>
            <fileset path="src">
                <file type="vhd" toplevel="true">MultAddStream.vhd</file>
            </fileset>
        </sources>
        <resources>
            <registers>
                <register direction="write" hold="false" width="1">go</register>
            </registers>
            <memory id="memory0" size="256MB">
                <interfaces>
                    <interface id="read" type="sequential" direction="read">
                        <port type="address" width="32" direction="out">address</port>
                        <port type="data" width="32" direction="in">data</port>
                        <port type="enable" width="1" direction="out">enable</port>
                        <port type="ready" width="1" direction="in">ready</port>
                        <port type="size" width="32" direction="out">size</port>
                        <port type="go" width="1" direction="out">go</port>
                    </interface>
                </interfaces>
            </memory>
        </resources>
    </fpga>
</resources>
```
RC Middleware Design Flow

1. Developer selects target platform from RCMW database

2. Developer selects target platform from RCMW database

- PROCStar III
- PROCStar IV
- M501
- H101
- PCIe-385n
- AC3611S
**RC Middleware Design Flow**

1. Developer executes RCMW tool chain with target platform

2. XML

3. RCMW Toolset

Select minimum cost mapping using user-definable cost function

E.g.: \( \text{Cost} = w_1 \times \text{area} + w_2 \times \text{num\_component} \)

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping A</td>
<td>100</td>
</tr>
<tr>
<td>Mapping B</td>
<td>50</td>
</tr>
<tr>
<td>Mapping C</td>
<td>100</td>
</tr>
<tr>
<td>Mapping D</td>
<td>200</td>
</tr>
</tbody>
</table>

The numbers above are just an example.
RC Middleware Design Flow

1. Generate top-level SW and hardware interfaces ready for compilation

2. RCMW Toolset

3. XML

4. HDL SW
## Supported Platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Supported Platforms</th>
<th>Underdevelopment; Partially supported or stability issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCStar III</td>
<td>4 Altera Stratix III E260</td>
<td>256MB SDRAM + 2x2GB DDR-II SODIMMs per FPGA</td>
</tr>
<tr>
<td>PROCStar IV</td>
<td>4 Altera Stratix IV E530</td>
<td>512MB SDRAM + 2x4GB DDR-II SODIMMs per FPGA</td>
</tr>
<tr>
<td>M501</td>
<td>1 Xilinx Virtex-6 LX240T</td>
<td>512MB DDR-III SDRAM</td>
</tr>
<tr>
<td>H101</td>
<td>1 Xilinx Virtex-4 LX100</td>
<td>512MB DDR-II SDRAM 4x4MB DDR-II SRAM</td>
</tr>
<tr>
<td>PCIe-385n</td>
<td>1 Altera Stratix V A7</td>
<td>2x4GB DDR-III SDRAM</td>
</tr>
<tr>
<td>AC3611S</td>
<td>1 Altera Stratix IV GX530</td>
<td>N/A</td>
</tr>
</tbody>
</table>

PCIe x8 | PCIe x8 | PCIe 2.0 x8 | 64-bit/133 MHz PCI-X | PCIe 3.0 x8 | PCIe 2.0 x8
Overhead Analysis

- Examined performance and area overhead
  - Compared wall-clock transfer performance of native vendor API and RC Middleware API from host to platform
  - Compared required number of clock cycles for increasing read/write transfer sizes for hardware interfaces
  - Examined relative area of vendor, RCMW and application components for several applications

Altera bitfiles generated using Quartus 11.1 sp1. Xilinx bitfiles generated using Xilinx ISE 14.1. Tests conducted using GiDEL driver version 8.9.3.0 and Pico driver version 5.2.0.0. Software compiled using GCC v4.7.2 with flags “-std=c++0x –O3”
Overhead Analysis

![Graph showing RCMW Interface Performance Overhead for different transfer sizes and performances in Read and Write. The graphs compare Overhead percentages across various data sizes for M501, PS3, and PS4.](image-url)
Overhead Analysis

RCMW area overhead depends on application configuration and target platform.
Portability Analysis

- Demonstrated single-source kernels and applications executing on three platforms without modification
  - RCMW toolset executed for each application/platform
  - Bitfile generated using vendor tool chains
  - Design parameters and problem size same for all platforms

Alterna bitfiles generated using Quartus 11.1 sp1. Xilinx bitfiles generated using Xilinx ISE 14.1. Tests conducted using GiDEL driver version 8.9.3.0 and Pico driver version 5.2.0.0. Software compiled using GCC v4.7.2 with flags “-std=c++0x –O3”
## Portability Analysis

Application characteristics and platform strengths determine performance.

<table>
<thead>
<tr>
<th>Kernel/Application</th>
<th>M501</th>
<th></th>
<th>PROCStar III</th>
<th></th>
<th>PROCStar IV</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Utilization</td>
<td>Time</td>
<td>Utilization</td>
<td>Time</td>
<td>Utilization</td>
<td>Time</td>
</tr>
<tr>
<td>1D Conv. FXD</td>
<td>43.3 ms</td>
<td>26%</td>
<td>39.3 ms</td>
<td>14%</td>
<td>38.9 ms</td>
<td>7%</td>
</tr>
<tr>
<td>2D Conv. FXD</td>
<td>16.0 ms</td>
<td>57%</td>
<td>13.2 ms</td>
<td>44%</td>
<td>16.3 ms</td>
<td>25%</td>
</tr>
<tr>
<td>Image Segmentation</td>
<td>1.73 s</td>
<td>64%</td>
<td>1.41 s</td>
<td>53%</td>
<td>1.39 s</td>
<td>26%</td>
</tr>
<tr>
<td>Needle-Distance</td>
<td>161 ms</td>
<td>48%</td>
<td>194 ms</td>
<td>38%</td>
<td>203 ms</td>
<td>21%</td>
</tr>
<tr>
<td>OpenCores AES128</td>
<td>32.6 ms</td>
<td>32%</td>
<td>25.3 ms</td>
<td>22%</td>
<td>24.3 ms</td>
<td>11%</td>
</tr>
<tr>
<td>OpenCores FIR</td>
<td>21.5 ms</td>
<td>28%</td>
<td>24.5 ms</td>
<td>15%</td>
<td>24.0 ms</td>
<td>8%</td>
</tr>
<tr>
<td>OpenCores JPEGEnc</td>
<td>23.9 ms</td>
<td>29%</td>
<td>15.3 ms</td>
<td>15%</td>
<td>19.6 ms</td>
<td>8%</td>
</tr>
<tr>
<td>OpenCores SHA256</td>
<td>73.3 ms</td>
<td>26%</td>
<td>64.1 ms</td>
<td>12%</td>
<td>63.3 ms</td>
<td>5%</td>
</tr>
<tr>
<td>Smith-Waterman</td>
<td>96.0 ms</td>
<td>23%</td>
<td>116 ms</td>
<td>11%</td>
<td>119 ms</td>
<td>6%</td>
</tr>
<tr>
<td>Sum of Absolute Diff.</td>
<td>18.7 ms</td>
<td>86%</td>
<td>14.7 ms</td>
<td>75%</td>
<td>19.1 ms</td>
<td>38%</td>
</tr>
</tbody>
</table>

Not meant as platform-performance comparison.
Productivity Analysis

- Same developer mapped each kernel to platform using vendor-specific method and RCMW toolset
  - SHA256, JPEGEnc., FIR Filter, AES128, 3DES
- For lack of a better metric, analyzed productivity by comparing SLoC and HLoC coded by developer, and total development time
  - Included all developer-written code, no comments

Altera bitfiles generated using Quartus 11.1 sp1. Xilinx bitfiles generated using Xilinx ISE 14.1. Tests conducted using GiDEL driver version 8.9.3.0 and Pico driver version 5.2.0.0. Software compiled using GCC v4.7.2 with flags “-std=c++0x -O3”
Productivity Analysis

- On average, RCMW required 65% less SLoC, 41% less HLoC and 53% less development time than GiDEL-based implementations
  - Increased SLoC due to user-level buffer management, thread safety
  - Increased HLoC due to additional CDC, additional code to map and interface kernels to platform interfaces

Altera bitfiles generated using Quartus 11.1 sp1. Xilinx bitfiles generated using Xilinx ISE 14.1. Tests conducted using GiDEL driver version 8.9.3.0 and Pico driver version 5.2.0.0. Software compiled using GCC v4.7.2 with flags “-std=c++0x -O3”
Productivity Analysis

On average, RCMW required 66% less SLoC, 59% less HLoC and 69% less development time than Pico-based implementations.

- Increased SLoC due to initialization, stream configuration and buffer management.
- Increased HLoC due to additional CDC, and controller code to interface with AXI interconnect.

Altera bitfiles generated using Quartus 11.1 sp1. Xilinx bitfiles generated using Xilinx ISE 14.1. Tests conducted using GiDEL driver version 8.9.3.0 and Pico driver version 5.2.0.0. Software compiled using GCC v4.7.2 with flags “-std=c++0x –O3”
Productivity Analysis

- RCMW was designed to reduce required coding by developer by providing application-specific interfaces
  - Interface customizability and integrated CDC reduced HLoC
  - RCMW API features reduced SLoC

Altera bitfiles generated using Quartus 11.1 sp1. Xilinx bitfiles generated using Xilinx ISE 14.1. Tests conducted using GiDEL driver version 8.9.3.0 and Pico driver version 5.2.0.0. Software compiled using GCC v4.7.2 with flags “-std=c++0x -O3”
Conclusions & Future Work

- Developed an extensible middleware framework to enable application *portability* and improve *productivity*
  - Less than 1% performance and area overhead in common case
  - Demonstrated single-source portability using 10 apps/kernels
  - Presented evidence indicating productivity improvement

**Future Work:**

- Explore application-specific optimizations
- Perform larger productivity case studies
- Expand features to include multi-FPGA and multi-Board support for scalability
- Investigate approaches for IP portability
Thanks for listening!

Questions
References


References


