



OpenFPGA CoreLib Core Library Interoperability Effort



M. Wirthlin¹, D. Poznanovic², P. Sundararajan³, A. Coppola⁴, D.
Pellerin⁵, W. Najjar⁶, R. Bruce⁷, M. Babst⁸, O. Pritchard⁹, P.
Palazzari¹⁰, G. Kuzmanov¹¹

¹Brigham Young University, ²SRC Computers, ³Xilinx Corporation, ⁴OptNgn Software,
⁵Impulse, ⁶UC Riverside, ⁷Nallatech, ⁸DSPLogic, ⁹Altera Corporation, ¹⁰Ylichron, ¹¹TU
Delft

Overview

- Background and Motivation
- OpenFPGA CoreLib Project Goals
- Related Hardware Circuit Reuse Efforts
- XML & IP-XACT
- Example
- Status and Future Work

FPGA Design Methods

RTL

- Register Transfer Level
 - VHDL, Verilog, etc.
- “Low-Level” Circuit Design
 - Focus on implementation
 - Time consuming
 - High quality circuits
- High Performance
 - Meet timing constraints
 - Meet resource constraints
 - Utilized specialized resources

HLL

- High-Level Languages
 - C, C++, Java, SystemC, etc.
 - Graphical Programming
- “High-Level” Algorithm Design
 - Focus on algorithm
 - Implementation details ignored
 - Possibly lower quality circuits
- High Productivity
 - Easy to modify algorithm
 - Fewer Implementation details
 - Less design time

High Performance “Cores”

- High-performance circuit “cores” needed for high-performance
 - Perform performance critical functions (FFT, etc.)
 - Hand crafted, efficient implementations
 - Manage a complex resources (MGT, memories, etc.)
- Cores often described in “low-level” language
 - VHDL, EDIF, raw bitstreams, etc.
 - Provide high-performance, hand crafted circuit “functions”

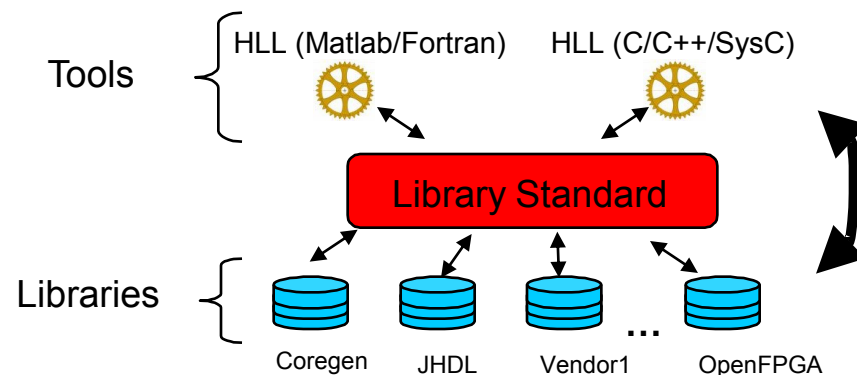
High Performance “Cores”

- High-performance circuit “cores” needed for high-performance
 - Perform performance critical functions (FFT, etc.)
 - Hand crafted, efficient implementations
 - Manage a complex resources (MGT, memories, etc.)
- Cores often described in “low-level” language
 - VHDL, EDIF, raw bitstreams, etc.
 - Provide high-performance, hand crafted circuit “functions”
- Challenge:
 - How do we integrate high-performance cores into high-level languages?
 - How do we reuse high-performance cores with more than one high-level language?



Project Goals

1. Develop a standard for circuit cores
2. Develop a standard for circuit libraries
3. Encourage the use of these standards
4. Create a synergistic environment
5. Create Libraries



Hardware Reuse

- Reuse of hardware circuits is very important for hardware engineers
 - Reduce design cost of large single chip systems
 - Amortize cost of circuit cores over multiple chips
 - Purchase high-quality circuits from “IP” specialists
- Hardware reuse is more difficult than software reuse
 - Complex interfaces
 - Signal timing, signal types, communication protocol
 - Physical implications
 - Circuit area, timing, power, placement, etc.
 - Challenging verification
 - How do you know it works?



Hardware Reuse Efforts



opencores.org

- Open source repository of circuit cores
- Cores developed for *wishbone* bus interface



OPC-IP

- Specifies a common standard for core interface
- Provides tools/infrastructure for integrating cores



SPIRIT Consortium

- Develops specs for *describing* circuits for re-use
- Created IP-XACT v1.2 spec (used in this work)



Virtual Socket Interface Alliance (VSI)

- Provide standards, docs, and methods for SoC design
- Facilitate circuit protection and transfer

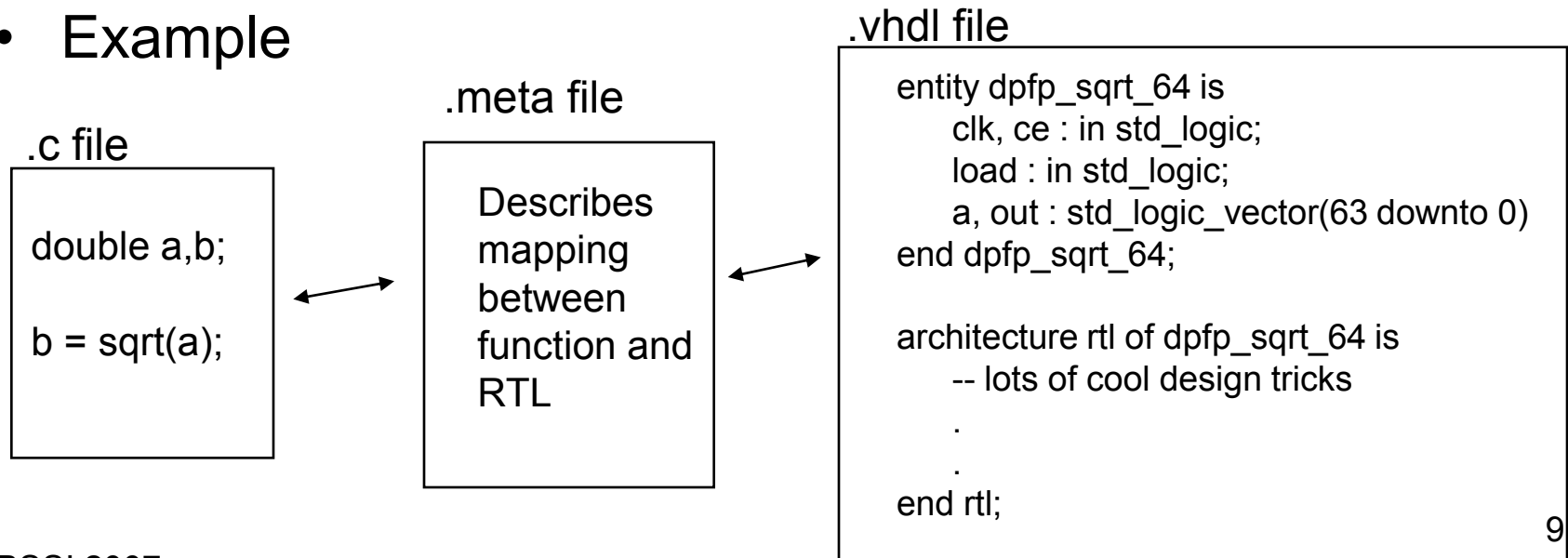


Si2

- Facilitate interoperability of EDA tools
- Provides common libraries and interoperability tools

Reuse with Existing HLL Tools

- Techniques used for importing external cores
 - Function call interface
 - New language semantics or PRAGMA statements
 - Overloading standard operators
 - New graphical library elements
 - Custom instructions
- Example

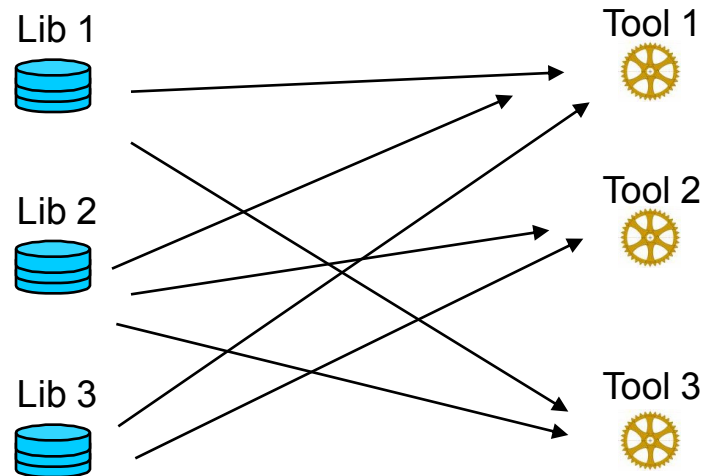


Reuse with Existing HLL Tools

- Techniques used for importing external cores
 - Function call interface
 - New language semantics or PRAGMA statements
 - Overloading standard operators
 - New graphical library elements
 - Custom instructions
- Commercial
 - Impulse-C
 - Dime-C
 - Carte
 - Reconfigurable Computing Toolbox
 - C2H
- Research
 - ROCCC
 - Trident
 - DWB
 - CHiMPS

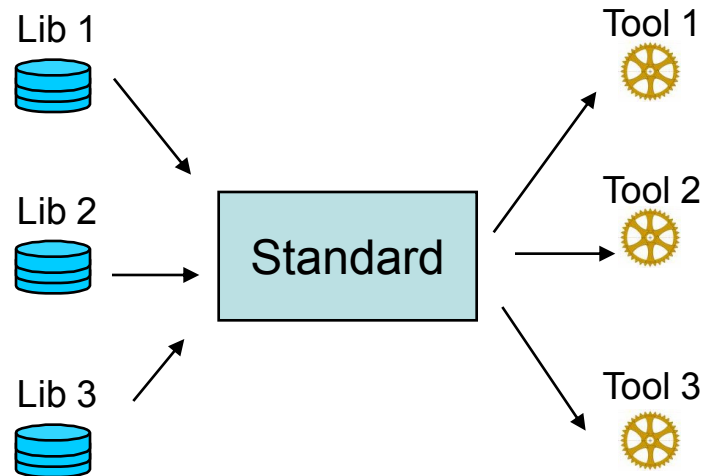
Reuse Challenges

- Each tool has a custom method for importing core
- Importing external core for new tool requires extra work
- Difficult to use same core in multiple tools
- Difficult to reuse libraries among tools



Reuse Goal

- Each core and library conforms to a standard
- Each tool recognizes this standard
- Easy to use same core in multiple tools
- Easy to reuse libraries among tools



IP Interface Standard Requirements

- Structural Information
 - Signals (name, bit width, properties)
 - Signal types (higher level type information)
- Timing Interface
 - Signal arrival times
 - Signal response times
- Control interface
 - Interface protocol specification
 - Handshaking requirements
- Parameterization
- Estimation interface
- External tool/generator interface

XML Circuit Meta Description

- Exploit XML infrastructure to describe cores and libraries
 - Define custom XML schema for describing cores
 - Create XML descriptions of reusable cores
 - Define all details of complicated circuit interface
 - Provide multiple views of core
 - Package cores into reusable libraries
- Benefits
 - Many tools available for manipulating/viewing XML
 - Several HLL tools already use XML to describe cores
 - Existing techniques for publishing cores with XML (IP-XACT)

IP-XACT



- XML meta description of reusable IP
 - XML schema defining tags for specifying reusable IP
 - Defines interface, configuration, and generation of IP
 - Used primarily for defining bus-based IP in SoC design
 - Current standard intended for RTL-level IP
- Created by the Spirit Consortium
 - Non-profit organization with over 60 company members
 - Led primarily by ARM and Mentor Graphics
 - Being considered as an IEEE Standard (P1685)
- Compatible and complementary with other IP-reuse activities
 - Opencores, OCP-IP, etc.

Example – UART on AMBA Bus

```
-->
<spirit:component xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-1
e" xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.2
Y:\SCHEMA-2\1.2\index.xsd">
  <spirit:vendor>spiritconsortium.org</spirit:vendor>
  <spirit:library>Leon2</spirit:library>
  <spirit:name>uart</spirit:name>
  <spirit:version>1.2</spirit:version>
  <spirit:busInterfaces>
    <spirit:busInterface>
      <spirit:name>APBCLK</spirit:name>
      <spirit:busType spirit:vendor="AMBA" spirit:library="AMBA2" spirit:name="APB" spirit:version="r0p0"/>
      <spirit:system>
        <spirit:group>APB_CLK</spirit:group>
      </spirit:system>
      <spirit:connection>required</spirit:connection>
      <spirit:signalMap>
        <spirit:signalName>
          <spirit:componentSignalName>clk</spirit:componentSignalName>
          <spirit:busSignalName>PCLK</spirit:busSignalName>
        </spirit:signalName>
      </spirit:signalMap>
    </spirit:busInterface>
    <spirit:busInterface>
      <spirit:name>APBReset</spirit:name>
      <spirit:busType spirit:vendor="AMBA" spirit:library="AMBA2" spirit:name="APB" spirit:version="r0p0"/>
      <spirit:system>
        <spirit:group>APB_RESET</spirit:group>
      </spirit:system>
      <spirit:connection>required</spirit:connection>
      <spirit:signalMap>
        <spirit:signalName>
          <spirit:componentSignalName>rst</spirit:componentSignalName>
          <spirit:busSignalName>PRESETn</spirit:busSignalName>
        </spirit:signalName>
      </spirit:signalMap>
    </spirit:busInterface>
    <spirit:busInterface>
      <spirit:name>ambaAPB</spirit:name>
      <spirit:busType spirit:vendor="AMBA" spirit:library="AMBA2" spirit:name="APB" spirit:version="r0p0"/>
      <spirit:slave>
        <spirit:memoryMapRef spirit:memoryMapRef="ambaAPB"/>
      </spirit:slave>
      <spirit:connection>required</spirit:connection>
      <spirit:signalMap>
        <spirit:signalName>
          <spirit:componentSignalName>pse1</spirit:componentSignalName>
          <spirit:busSignalName>PSELx</spirit:busSignalName>
        </spirit:signalName>
        <spirit:signalName>
          <spirit:componentSignalName>penable</spirit:componentSignalName>

```

Raw XML

Example – UART on AMBA Bus

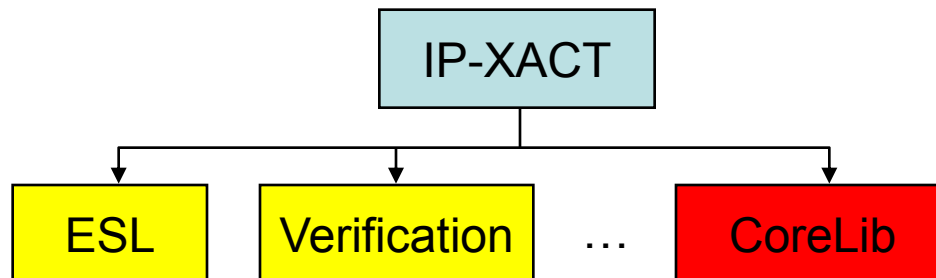
spirit:component	
xmlns:spirit	http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.2
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.2 □ □ Y:\SCHEMA~2\V1.2\index.xsd
spirit:vendor	spiritconsortium.org
spirit:library	Leon2
spirit:name	uart
spirit:version	1.2
spirit:busInterfaces	
spirit:busInterface	
spirit:busInterface	
spirit:busInterface	
spirit:name	ambaAPB
spirit:busType	
spirit:slave	
spirit:connection	required
spirit:signalMap	
spirit:signalName	
spirit:signalName	
spirit:signalName	
spirit:componentSignalName	paddr
spirit:busSignalName	PADDR
spirit:signalName	
spirit:componentSignalName	pwrite
spirit:busSignalName	PWRITE
spirit:signalName	
spirit:componentSignalName	pwdata
spirit:busSignalName	PWDATA
spirit:signalName	
spirit:componentSignalName	prdata
spirit:busSignalName	PRDATA
spirit:busInterface	
spirit:memoryMaps	
spirit:memoryMap	
spirit:name	ambaAPB
spirit:addressBlock	
spirit:model	
spirit:componentConstruct	

Eclipse IP-XACT Plug-In View

Several tools available for manipulating and creating IP-XACT meta descriptions

IP-XACT Extensions

- IP-XACT standard not sufficient for CoreLib effort
 - Limited to bus-based cores and HDL data types
 - Limited support for custom interfaces
 - IP-XACT supports a variety of *extensions* for new functionality
- Develop extensions to IP-XACT for CoreLib
 - Higher level data types (floating point, fixed point, etc.)
 - Temporal interface (timing and protocol)
 - Specifying behavior (arithmetic functions)
 - Advanced memory architectures



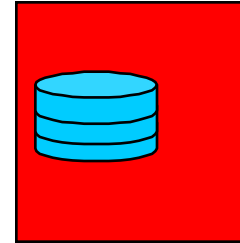
Square Root Example

```
double a,b;
```

```
b = sqrt(a);
```



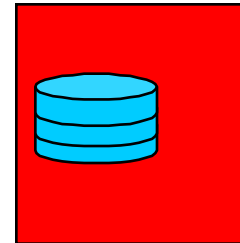
Vendor Library



```
b = my_fast_sqrt(a);
```



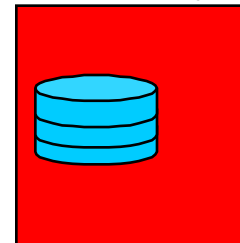
User Library



```
b = vendor_small_sqrt(a);
```



GPL Library

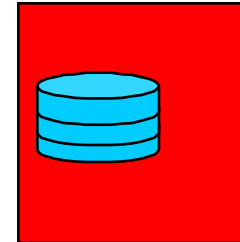


Square Root Example

```
x = [3.2; 1.0; 2; 5];  
y = [1.1, -2.3, 4, .4];  
a = det(x * y);
```



Vendor Library

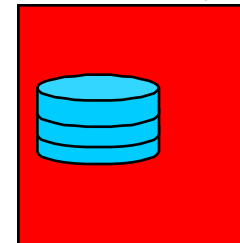


```
b = sqrt(a);
```

```
b = my_sqrt(a);
```



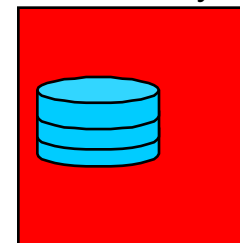
User Library



```
b = vendor_sqrt(a);
```



GPL Library



Status

- Completed survey of previous work
- Gathered information about tools
- Proposed framework around IP-XACT
- Experimenting with IP-XACT
- Gathering freely available cores

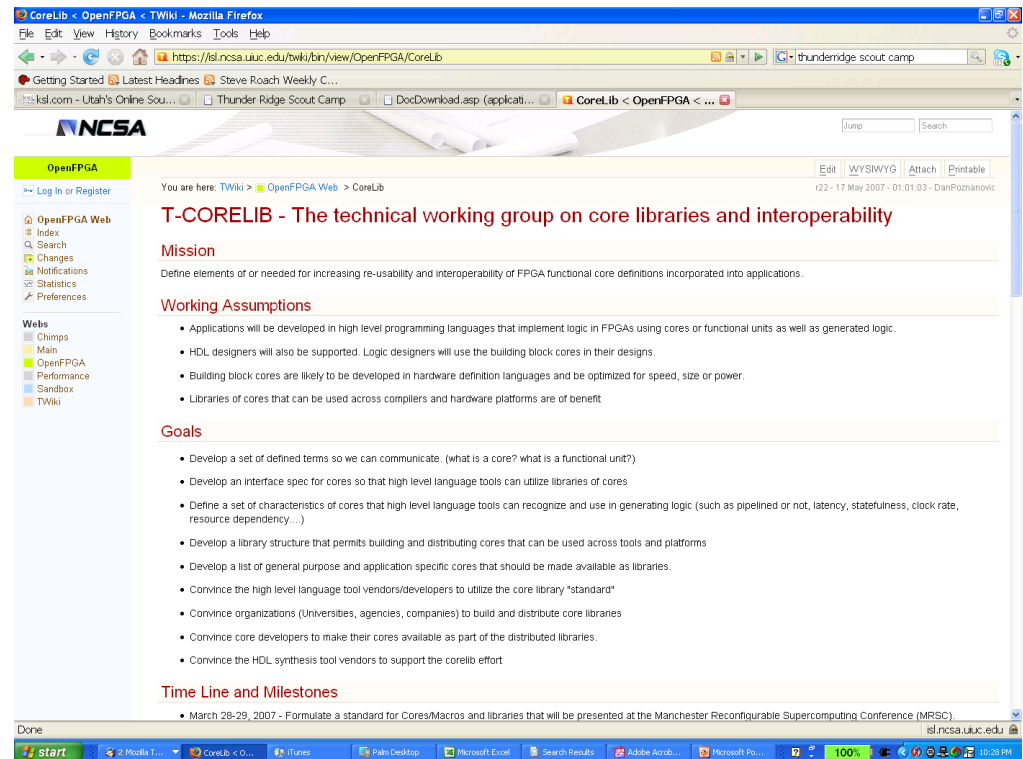
Future Work

- Develop sample IP-XACT wrappers for cores
- Propose and demonstrate extensions
- Solicit feedback from vendors
- Encourage use of standards

CoreLib Wiki

<https://isl.ncsa.uiuc.edu/twiki/bin/view/OpenFPGA/CoreLib>

- Related Circuit Reuse Activities
- Tools and Compilers
- Examples
- HLL Compiler Requirements
- Interface Standards



The screenshot shows a Mozilla Firefox browser window displaying the CoreLib Wiki page. The address bar shows the URL <https://isl.ncsa.uiuc.edu/twiki/bin/view/OpenFPGA/CoreLib>. The page content includes the NCSA logo, a navigation menu, and the main text of the wiki page. The main text is titled "T-CORELIB - The technical working group on core libraries and interoperability" and includes sections for "Mission", "Working Assumptions", "Goals", and "Time Line and Milestones".

NCSA

OpenFPGA

You are here: TWiki > OpenFPGA Web > CoreLib

T-CORELIB - The technical working group on core libraries and interoperability

Mission

Define elements of or needed for increasing re-usability and interoperability of FPGA functional core definitions incorporated into applications.

Working Assumptions

- Applications will be developed in high level programming languages that implement logic in FPGAs using cores or functional units as well as generated logic.
- HDL designers will also be supported. Logic designers will use the building block cores in their designs.
- Building block cores are likely to be developed in hardware definition languages and be optimized for speed, size or power.
- Libraries of cores that can be used across compilers and hardware platforms are of benefit

Goals

- Develop a set of defined terms so we can communicate. (what is a core? what is a functional unit?)
- Develop an interface spec for cores so that high level language tools can utilize libraries of cores
- Define a set of characteristics of cores that high level language tools can recognize and use in generating logic (such as pipelined or not, latency, statefulness, clock rate, resource dependency...)
- Develop a library structure that permits building and distributing cores that can be used across tools and platforms
- Develop a list of general purpose and application specific cores that should be made available as libraries.
- Convince the high level language tool vendors/developers to utilize the core library "standard"
- Convince organizations (Universities, agencies, companies) to build and distribute core libraries
- Convince core developers to make their cores available as part of the distributed libraries.
- Convince the HDL synthesis tool vendors to support the corelib effort

Time Line and Milestones

- March 28-29, 2007 - Formulate a standard for Cores/Macros and libraries that will be presented at the Manchester Reconfigurable Supercomputing Conference (MRSC)