RAMP: Architecture, Language & Compiler

http://ramp.eecs.berkeley.edu
Greg Gibeling, Andrew Schultz & Krste Asanovic
gdgib@berkeley.edu
6/21/2006

Outline

RAMP Architecture (1)

- A framework for system emulation
  - Massively parallel (digital hardware) systems
  - Orders magnitude performance enhancement
  - Leverage existing designs
  - Allow community development
    - Share designs, validate experiments, etc…
- Flexible, cross platform designs
  - Requires proper structure
    - Support for automatic debugging
    - Automatic glue logic/code generation
  - Based on the “target model”

RAMP Architecture (2)

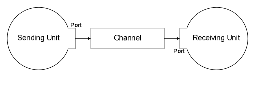
- Target
  - The system being emulated
    - Actually only a model of the system being emulated
    - Can be a cycle accurate model
  - Must conform to the RAMP target model
- Host
  - The system doing the emulation
  - May include multiple platforms
    - Hardware – BEE2, XUP, CaLinx2
    - Emulation – Matlab, ModelSim
    - Software – C++, Java

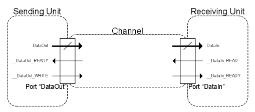
RAMP Architecture (3)

- Fundamental Model
  - Message passing
  - Distributed event simulator
  - Message passing system generator
    - Cross platform
    - Shared development effort
    - Easy to develop, debug and analyze
  - Similar Formalisms
    - Petri Nets
    - Process Networks
    - Research: Click, P2, Ptolmey, Metropolis, etc.…

RAMP Target Model (1)

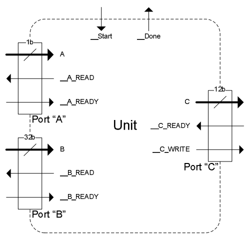
- Units communicate over channels
- Units
  - 10,000+ Gates
    - Processor + L1
  - Implemented in a “host” language
- Channels
  - Unidirectional
  - Point-to-point
  - FIFO semantics
  - Delay Model





Target Model - Units

- Inside edge
  - Ports connect units to channels
    - FIFO signaling
    - Hardware or Software
  - Target cycle control
    - __Start
    - __Done
    - Allows for variable timing, and timing accurate simulation

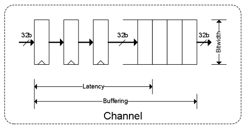


Target Model – Channel (1)

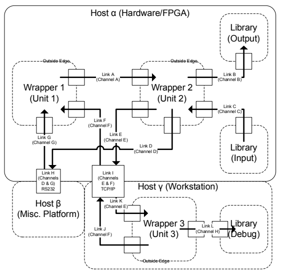
- Channel semantics
  - Arbitrary message size
    - The messages are statically typed
  - Ordered delivery
  - Debugging through monitoring & injection
  - Provides for cross-platform simulations

Target Model – Channel (2)

- Channel Params
  - Only used for timing accurate simulations
  - Bitwidth
  - Latency
  - Buffering
- Fragments
  - Smaller than messages
  - Convey the simulation time through idles

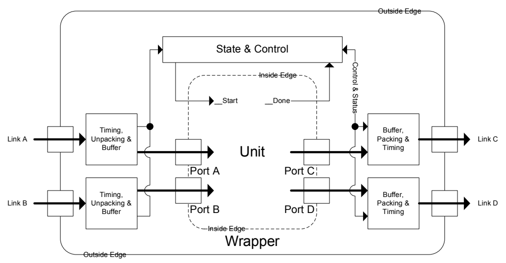


Host Model



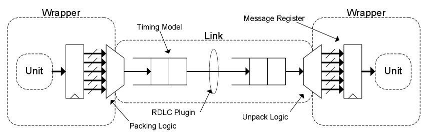
- Cross platform
  - Units implemented in many languages
  - Library units for I/O
  - Links implement channels
- Links
  - Any communication
  - Less defined

Host Model – Wrapper



Host Model - Link

- Typically Three Components
  - Packing & Unpacking
  - Timing Model
  - Physical Transport
    - Generated by RDLC2 plugins



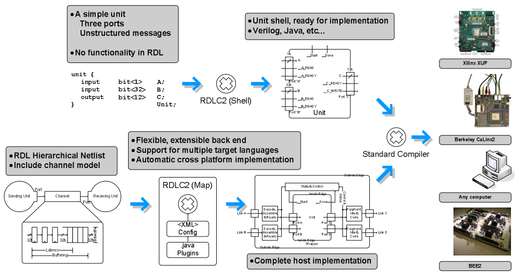
RDL (1)

- "RAMP Description Language"
  - General message passing system description language
  - "Netlisting" language
    - Does NOT include leaf unit behavior
- Compiler is highly extensible
  - Links
  - Other toolflows
  - External signals
  - Memories, etc…

RDL (2)

- Why RDL?
  - Allows specification of partitioning
    - Regular communication
    - Enables cross platform system design
  - RDL is a research enabler
    - Ties together EXISTING designs
    - Allows sharing of work & results
  - Saves a lot of work
    - Complex interconnect is painful in HDLs

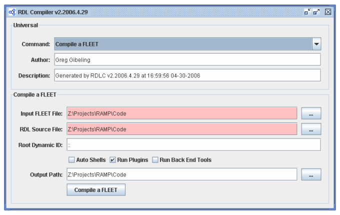
RDLC2 Toolflow (1)



```
unit {
   input    bit<1>    A;
   input    bit<32>   B;
   output   bit<12>   C;
}                     Unit;
```

RDLC2 Toolflow (2)

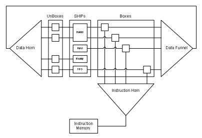
- Help
  - rdlc2 –help
  - Explains commands
  - Includes all the options
- GUI
  - rdlc2 –gui
  - Easy to use
  - Includes error message display



Command: Compile a FLEET
Author: Greg Gibeling
Description: Generated by RDLC v2.2006.4.29 at 16:59:56 04-30-2006
Input FLEET File: Z:\Projects\RAMP\Code
RDL Source File: Z:\Projects\RAMP\Code
Root Dynamic ID: ::
Auto Shells [✓ Run Plugins] Run Back End Tools
Output Path: Z:\Projects\RAMP\Code
Compile a FLEET

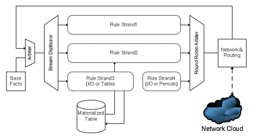
FLEET Builder Application (1)

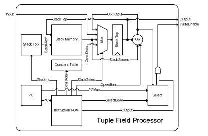
- FLEET
  - A one instruction computer (Move)
  - Highly concurrent
  - Location and operation are tied
  - Includes network builder
  - Includes assembler generator



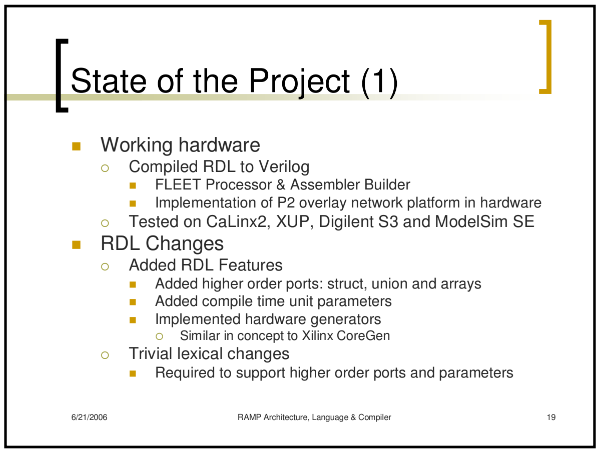
```
initial codebag Accumulate {
move (0) -> Adder.Adder;
move [] IntegerInput.Output
    -> Adder.Addend;
move [] Adder.Sum ->
    Display.Input, Adder.Adder;
};
```

P2/Overlog Application



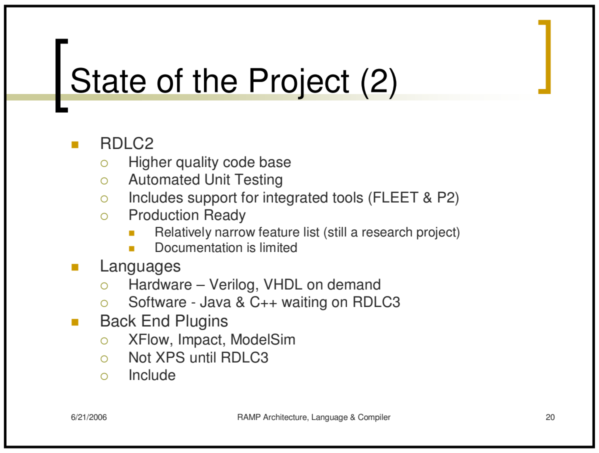


- Overlay Networks
  - Overlog (datalog) spec is compiled as in a DB query planner
  - Creates distributed tuple processors
  - We did a hardware implementation
  - Includes an ASIP



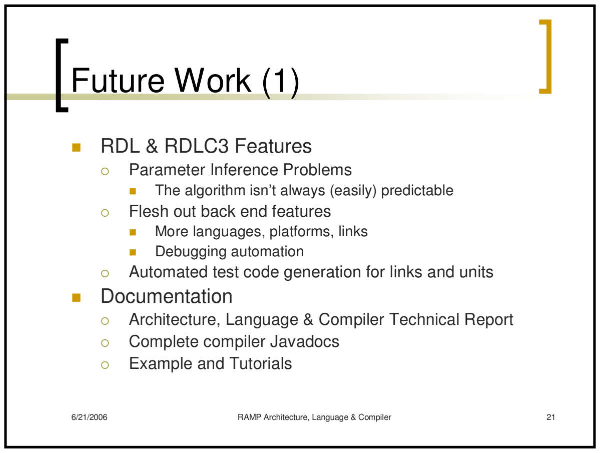
State of the Project (1)

- Working hardware
  - Compiled RDL to Verilog
    - FLEET Processor & Assembler Builder
    - Implementation of P2 overlay network platform in hardware
  - Tested on CaLinx2, XUP, Digilent S3 and ModelSim SE
- RDL Changes
  - Added RDL Features
    - Added higher order ports: struct, union and arrays
    - Added compile time unit parameters
    - Implemented hardware generators
      - Similar in concept to Xilinx CoreGen
  - Trivial lexical changes
    - Required to support higher order ports and parameters



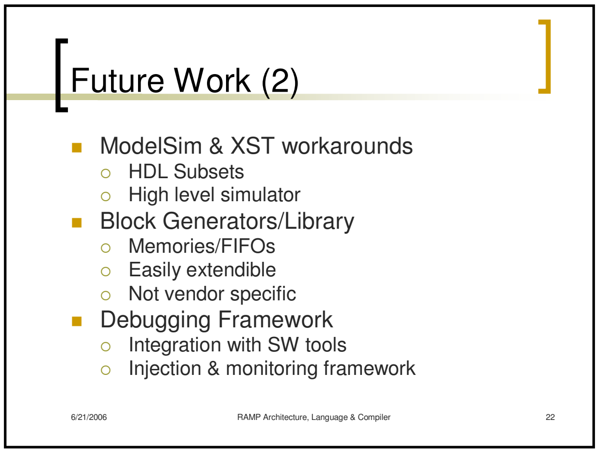
State of the Project (2)

- RDLC2
  - Higher quality code base
  - Automated Unit Testing
  - Includes support for integrated tools (FLEET & P2)
  - Production Ready
    - Relatively narrow feature list (still a research project)
    - Documentation is limited
- Languages
  - Hardware – Verilog, VHDL on demand
  - Software - Java & C++ waiting on RDLC3
- Back End Plugins
  - XFlow, Impact, ModelSim
  - Not XPS until RDLC3
  - Include



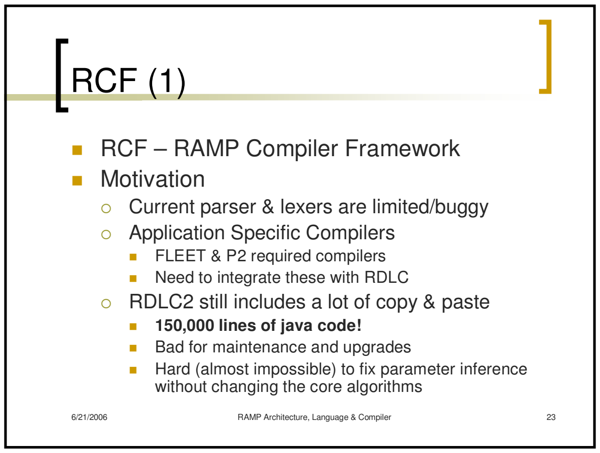
Future Work (1)

- RDL & RDLC3 Features
  - Parameter Inference Problems
    - The algorithm isn't always (easily) predictable
  - Flesh out back end features
    - More languages, platforms, links
    - Debugging automation
  - Automated test code generation for links and units
- Documentation
  - Architecture, Language & Compiler Technical Report
  - Complete compiler Javadocs
  - Example and Tutorials



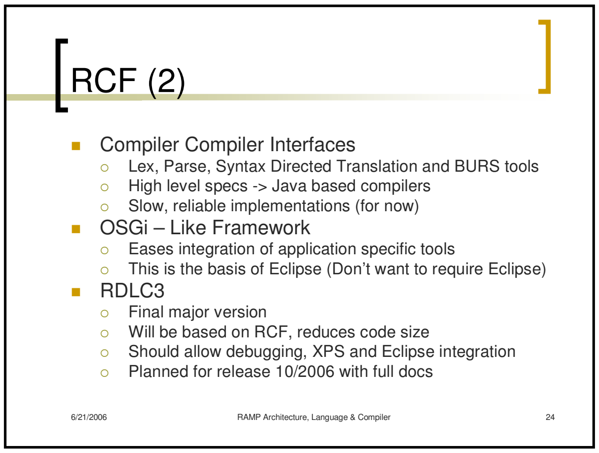
Future Work (2)

- ModelSim & XST workarounds
  - HDL Subsets
  - High level simulator
- Block Generators/Library
  - Memories/FIFOs
  - Easily extendible
  - Not vendor specific
- Debugging Framework
  - Integration with SW tools
  - Injection & monitoring framework



RCF (1)

- RCF – RAMP Compiler Framework
- Motivation
  - Current parser & lexers are limited/buggy
  - Application Specific Compilers
    - FLEET & P2 required compilers
    - Need to integrate these with RDLC
  - RDLC2 still includes a lot of copy & paste
    - **150,000 lines of java code!**
    - Bad for maintenance and upgrades
    - Hard (almost impossible) to fix parameter inference without changing the core algorithms



RCF (2)

- Compiler Compiler Interfaces
  - Lex, Parse, Syntax Directed Translation and BURS tools
  - High level specs -> Java based compilers
  - Slow, reliable implementations (for now)
- OSGi – Like Framework
  - Eases integration of application specific tools
  - This is the basis of Eclipse (Don't want to require Eclipse)
- RDLC3
  - Final major version
  - Will be based on RCF, reduces code size
  - Should allow debugging, XPS and Eclipse integration
  - Planned for release 10/2006 with full docs