RAMP: Architecture, Language & Compiler

http://ramp.eecs.berkeley.edu
Greg Gibeling (gdgib@berkeley.edu)
Andrew Schultz, Krste Asanovic & John Wawrzynek
1/11/2007

Outline

- Introduction
- Target Model
- Host Model
- Practical Information
- Status & Looking Ahead

Quick Introduction to RDL

- The “RAMP Description Language” (RDL)
  - Hierarchical structural netlisting langauge
  - Describes message passing distributed event simulations
  - System level: contains no behavioral spec.
- Tradeoffs
  - Costs
    - Use of the RAMP target model
    - Area, time and power to implement this model
  - Benefits
    - Abstraction of locality & timing of communications
    - System debugging & power tools
    - Determinism, sharing and research
  - Goal: trade costs for benefits as needed

RAMP Vision

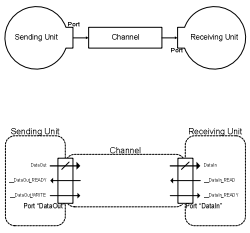
- FPGAs as a Research Platform
  - About 8 CPUs can fit in Field Programmable Gate Array
  - 256-CPU system from ~32 FPGAs? (8 BEE2s)
- Architecture Reasearch community does logic design
  - Create out-of-the-box, massively parallel system
  - Processors, Caches, Coherency, Ethernet Interfaces, Switches, Routers…
- RAMP: A framework for system emulation
  - Massively parallel digital hardware systems, orders magnitude faster than software
  - Leverage existing designs and shared development among researchers
- Flexible, cross platform designs
  - Requires proper structure
  - Based on the RDL “target model”

RAMP Architecture

- RAMP Design Framework
- Target
  - The system being emulated
  - Must conform to the RAMP target model
- Host
  - The system hosting the emulation
  - May include multiple platforms
    - Hardware – BEE3, BEE2, XUP, CaLinx2
    - Emulation – Matlab, ModelSim
    - Software – C++, Java

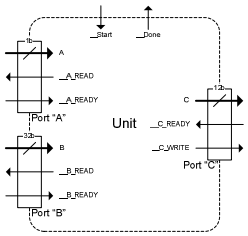
Target Model (1)

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



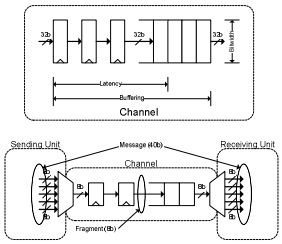
Target Model (2)

- Inside edge
  - Ports connect units to channels
    - FIFO signaling
    - Hardware or Software
  - Target cycle control
    - __Start
    - __Done



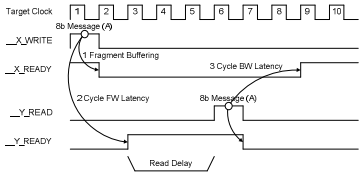
Target Model (3)

- Channel Params
  - Only used for timing accurate simulations
  - Bitwidth
  - FW Latency
  - Buffering
  - BW Latency
- Fragments
  - Smaller than messages
  - Indivisible message piece, which can be carried by a channel



Target Model (4)

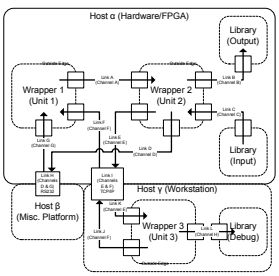
- Simple Example (More Later)
  - Channel parameters <8, 2, 1, 3>
  - A single 8bit Message



Target Model (5)

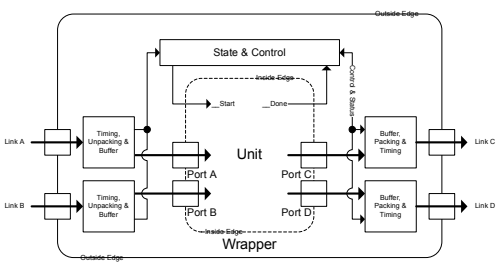
- Non-universal Model
  - Busses, lossy channels, multicast networks modeled as units
  - No global reset
- Emulation & abstraction are not free
  - Time, area and power are all spent
  - Particularly noticeable for DSP or control-free systems
- Existing Systems
  - Can be used as a single unit
  - May be split, but this will require design changes

Host Model (1)



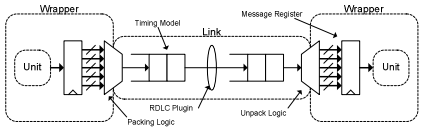
- Cross platform
  - Units implemented in many languages
  - Library units for I/O
  - Links implement channels
- Links
  - Any communication
  - Less defined
  - Plugins

Host Model (2)

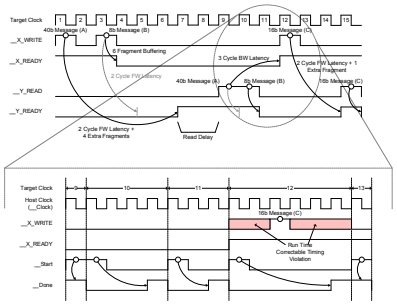


Host Model (3)

- Links - Typically Three Components
  - Packing & Unpacking
  - Timing Model
  - Physical Transport



Host Model (4)

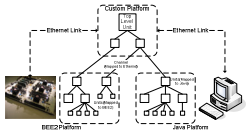


RDL (1)

- "RAMP Description Language"
  - General message passing system description language
  - "Netlisting" language
  - Easy to describe or modify a system
  - Allows specification of partitioning
- Extensible Compiler
  - Links & Languages
  - Plugins & Generators
- Simplified Design
  - Complex interconnect is painful in HDLs
  - Advanced datatypes are usually not present in HDLs

RDL (2)

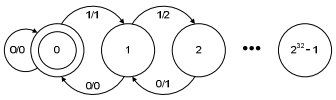
- RDL Target Constructs
  - Channels, Messages and Port types
  - Units include instances, inputs, outputs and connections
- RDL Host Constructs
  - One platform per board or computer
  - Platforms include an implementation language
  - Hierarchy allows for, eg. A board with many FPGAs



- Parameters
  - Per-instance
  - Compiled transparently to output (e.g. Verilog)
  - Support inference (& unit identity in RDLC3)

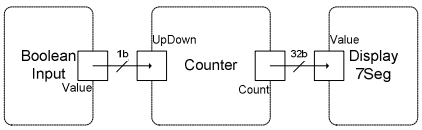
RDL Example (1)

```
unit <width = 32> {
    input  bit<1>          UpDown;
    output bit<$width>     Count;
}                          Counter;
```



RDL Example (2)

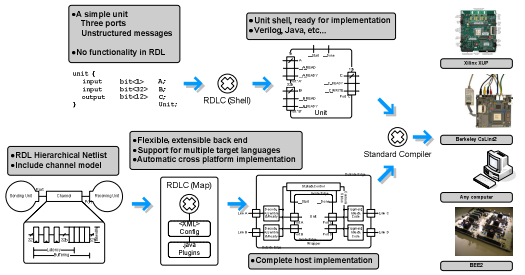
```
unit {
    instance IO::BooleanInput   BooleanInput(InChannel);
    instance Counter<32>        Counter;
    instance IO::Display7Seg    Display7Seg(.In(OutChannel));

    channel                     InChannel { -> Counter.UpDown };
    channel                     OutChannel { Counter.Count -> };
}                               CounterExample;
```



RDL Example (3)

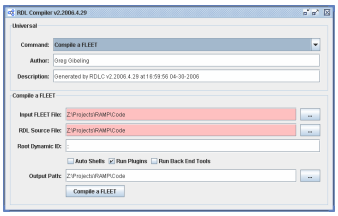
```
platform {
    language                        "verilog";

    plugin "Virtex2ProEngine"<"AJ15", "iostandard",
    "LVCMOS25">                     Engine;

    plugin "ModuleLibrary"<"ModuleLibrary.xml">
                                    Library;
}                                   XUP;

map {
    platform XUP                    BasePlatformInst;
    unit CounterExample             BaseUnitInst;
}                                   XUPCounter;
```


RDLC2 Toolflow (1)

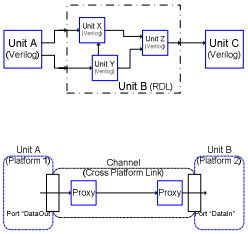


RDLC2 Toolflow (2)

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

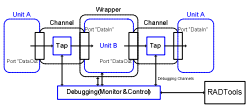


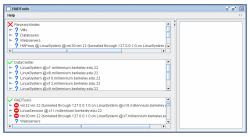
Advanced RDL (1)



- RDL in RDL
  - Useful in complex systems
- "Zero Delay"
  - Easy within FPGA
  - Approximated with a link implemented in RDL

Advanced RDL (2)



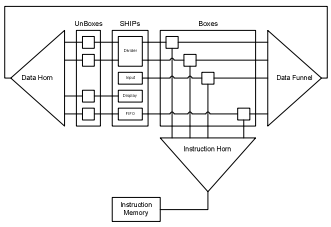


- Debugging
  - Monitoring
    - All communication is over channels
    - Real time can be paused or slowed down
  - Injection
    - Greatly eases testing
    - Fault Injection
  - Co-Simulation

Advanced RDL (3)

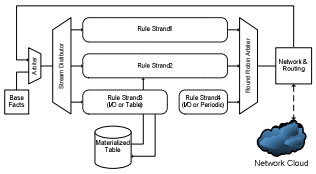
- Power Estimation
  - Methodology
    - Node activity determines power in CMOS
    - Unit activity likely tracks node activity
  - Implementation
    - RDL to RDL transformation
    - In system computation

FLEET Builder Application




```
unit <IWidth> {
    input Integer<$IWidth>
    Adder;
    input Integer<$IWidth>
    Addend;
    output Integer<$IWidth>
    Sum;
} Adder;

unit {
    input Instruction In;
    output Operation[$NPorts]
    Out;

    plugin
    "FLEET.HornBuilder"
    <$NPorts> HornBuilder;
} InstructionHorn;
```

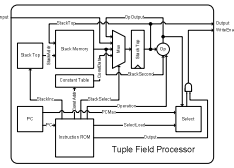

P2/Overlog Application (1)



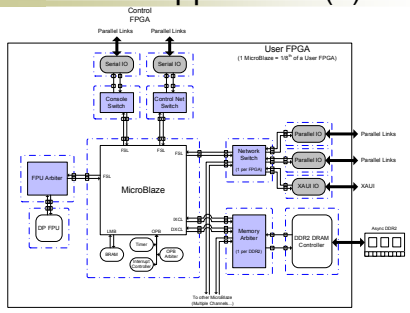
- Overlay Networks
  - Overlog (datalog) spec is compiled as in a DB query planner
  - Creates distributed tuple processors
  - We did a hardware implementation
  - Includes an ASIP

P2/Overlog Application (2)

- TFP
  - Parameterized Tuple Field Proc.
  - Includes hardware builder
  - Includes auto-generated assembler
  - Simple stack architecture



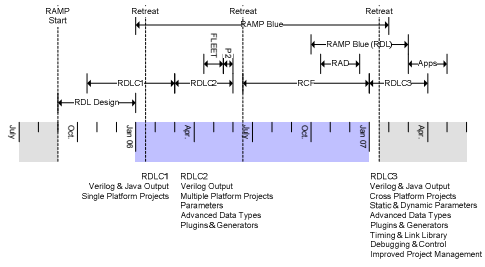
RAMP Blue Application (1)



RAMP Blue Application (2)

- RDL Related Lessons
  - RDLC parser has poor error recovery
  - Documentation incomplete
    - Platform level hardware support
    - Parameters & inference
  - Project management
    - Which compiler tool is “in charge”
  - Big thanks to Jue Sun

Timeline



Conclusion

- Success
  - Compiled RDL to Verilog (and Java)
  - Tested on CaLinx2, XUP, Digilent S3 and ModelSim SE
- Language Changes in v2
  - Added RDL Features
  - Trivial lexical changes
- RDLC2
  - Generator plugins: FLEET & P2
  - Back end plugins: XFlow, Impact, ModelSim
  - Ready for widespread use
- RDLC3
  - Integration of RDLC2 with RCF libraries
  - RDL Debugging & RADTools
- Thanks
  - Andrew Schultz, Krste Asanovic & John Wawrzynek
  - Nathan Burkhart, Alex Krasnov & Jue Sun

Future Work

- RDL & RDLC3 Features
  - Parameter Inference Problems
  - Flesh out back end features
    - More languages, platforms, links, timing
    - Debugging automation
  - Better project management
- Documentation
- Debugging & Power
  - Major cross-platform design
  - Based on RDL -> RDL Transformation
  - Debugging, Monitoring and Visualization
  - Distributed power calculations

An Open Call

- The need for users
  - ~3 people here have really used RDL
  - RAMP Blue & White are counting on it
  - The final major version is in progress
- Requirements
  - Actual hands-on experience
  - Specific applications
  - Specific requirements

Project Ideas

- Language backends for RDLC3
- Unit Generators
  - Memory controller, network switch
  - Should allow better parameterization (in many ways)
- Port FLEET & P2 from RDLC2 to 3
- Processor Generators
- Multi-FPGA PAR
- Distributed Systems
- SimuLink/DSP Designs & Unit Library