Using RDLC to Launch a FLEET

http://ramp.eecs.berkeley.edu
Greg Gibeling
gdgib@berkeley.edu
4/30/2006

Outline

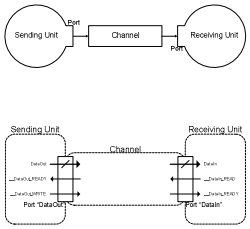
- RAMP Architecture & Target Model
- Tools & Toolflow
- RAMP Description Language
- An RDL FLEET
- FLEET Assembly
- Status & Future Work

RAMP Architecture

- 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 Target Model (1)

- Units communicate over channels
- Units
  - Implemented in a "host" language
  - ~1 SHIP for FLEET
  - "Transition"
- Channels
  - Unidirectional
  - Point-to-point
  - FIFO semantics
  - "Place"

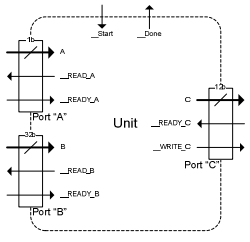


RAMP Target Model (2)

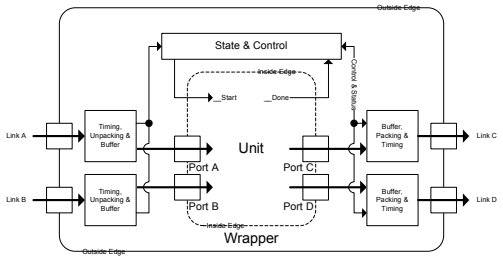
- Transaction style unit semantics
  - Read 0/1 messages from each input
  - Perform some action
  - Write 0/1 messages to each output
  - Units must be latency insensitive
- This affects the way SHIPs are coded

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



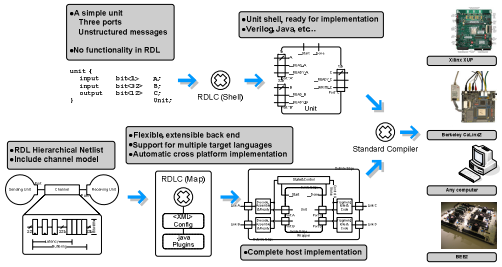
Host Model – Wrapper (2)



RDLC2 Toolflow (1)

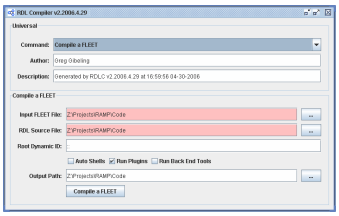
- Development Steps
  - Unit Implementation
    - RDL unit descriptions
    - RDLC generates shell code in a specific language
    - Researcher adds implementation
  - RDL target design
    - Includes Mapping
    - RDLC generates complete implementation code
    - Includes all links, instantiates all unit shells

RDLC2 Toolflow (2)



RDLC Toolflow (3)

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



RDL (1)

- "RAMP Description Language"
- General message passing system description language
  - Hierarchical Netlisting Language
- Compiler includes back-end extensibility
  - Can invoke ModelSim/XFlow back ends
- Does not include behavioral code

RDL (2)

- Hierarchical Namespaces
  - Declarations can be external to a namespace
  - Allows for communal development
- RDL Target Constructs
  - Channels, Messages and Port types
  - Units include instances, inputs, outputs and connections
- RDL Host Constructs
  - Links and Terminal types
  - One platform per board or computer
  - Platforms include an implementation language
- RDL Mappings
  - Hierarchy allows for "compile one, run many"
  - Allows specific units and channels to be precisely mapped

RDL (3)

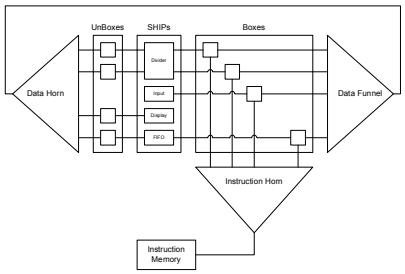
```
unit <width> {
        plugin Platforms::ModelSim"SetParam"<$width, 32>    ModelSimWidth;
        plugin Platforms::XUP"SetParam"<$width, 4>          XUPWidth;
        plugin Platforms::CaLinx2"SetParam"<$width, 32>     CaLinx2Width;
        plugin Platforms::S3"SetParam"<$width, 16>          S3Width;

        instance IO::BooleanInputBoolean   InputX(Value(InChannel));
        instance Counter<$width>           CounterX(InChannel, OutChannel);
        instance IO::DisplayNum<$width>    DisplayNumX;

        channel                  InChannel;
        channel                  OutChannel { -> DisplayNumX.Value };
} CounterExample;

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


A FLEET in RDL (1)



A FLEET in RDL (2)

- ::Maps Namespace
  - Contains all the mappings of the FLEET
  - ModelSim, CaLinx2, XUP and S3
  - ModelSim XE can be downloaded free!
- FLEET Unit
  - Instantiates everything
  - All SHIP connections are automatic

FLEET Assembly (1)

- Aliases
  - `alias FIFO[0].Input 0;`
- FLEET Descriptor
  - `fleet <6, 4, 3, 32, 10, 5>;`
- Moves
  - `move IntegerInput.Output -> Display.Input;`
  - `move [] (1) -> Adder.Adder;`

FLEET Assembly (2)

- Differences from other assembly
  - Semi-colons at line ends
  - [?] for move counts, [] for standing
  - Dotted port notation matches RDL EXACTLY

```
initial codebag Accumulate {
      move (0) -> Adder.Adder;

      move [] IntegerInput.Output -> Adder.Addend;
      move [] Adder.Sum -> Display.Input, Adder.Adder;
};
```


FLEET Assembly (3)

- Programs
  - Simple
    - Displays some output
    - Tests code bag loading
  - Addition
  - Counter
  - Accumulator
    - The demo I have today

FLEET Toolflow

- Assemble a FLEET Image
  - Convert FLEET code to a memory hex image
- Assemble a FLEET Memory
  - Above, plus creating a verilog memory module
- Precompiling a FLEET
  - Generate Verilog from RDL
  - Generate a FLEET file with aliases
- Compile a FLEET and Program
  - Generate a complete, running FLEET
  - One click: Program -> Computer

FLEET Toolflow (2)

- Working & Tested
  - Assemble an Image
  - Compile a FLEET and program
- Working with FPGA tools
  - The XST/Verilog bug
    - Always @ (*) is NOT properly supported
    - XST can't handle this with wire arrays
    - Will be fixed in ISE9.1
    - I'm not going to hack around this (Sorry)

Working with FLEET

- Labs
  - Lab1 – CounterExample
  - Lab2 – FLEET
    - Pretty sketchy for now
- Getting Help
  - http://ramp.eecs.berkeley.edu
  - RDLC Users Mailing List

State of the Project

- Working hardware implementation!
  - Compiled RDL to Verilog
  - Tested on CaLinx2, XUP, Digilent S3 and ModelSim SE
  - Runs FLEET programs
- RDL & RDL Compiler
  - RDL2 is stable
  - Working compiler, written in java
  - 125,000 lines of code
  - Definitely going to get cleaned up

Future Work (1)

- RDL & RDLC Features
  - Language Features
    - Generated code and compile time parameters
    - Support for expressions and better parameters
  - Languages, platforms, links
  - Debugging automation
- Documentation
  - Architecture, Language & Compiler Technical Report
  - Complete compiler internals documentation
  - Example and Tutorials

Future Work (2)

- SHIPs
  - Multiplier & Divider
  - Larger library of RDL units will help
  - Data Memory
- FLEET Programs
- Alternate Switch Fabrics
  - Butterfly, Crossbar, etc
  - Multi Trunk
  - Manhattan Grid