RTLgen - Functional Specification Document

Nitij Mangal

January 2009

1 Overview

RTLgen is an EDA tool which provides a perl based development environment to generate RTL. It currently support verilog and will be extended to VHDL over time. The tool is an aid to the designer as it off loads all syntax and language specific tasks so that the designer can focus on the core functionality and logic of the design. RTLgen can currently perform the following tasks

- Create port lists and declarations
- Create net declarations
- Create sensitivity lists for combinational and sequential always blocks
- Create instance declarations and connections

2 Tool flow

RTLgen allows a designer to call high level construct in a pseudo RTL (RTL like language) which are processed by the tool to output actual RTL code.The tool will work in three stages

1. Stage1: Parse the pesudo RTL
2. Stage2: Analyze the design
3. Stage3: Output full blown RTL

1. **Stage1: Parse the pesudo RTL**
   The tool uses a Verilog parser built on top of Hardware::Verilog and Parse::RecDescent packages from CPAN. The verilog parser has been enhanced to parse the following pseudo RTL constructs.

   (a) ::Module *moduleName*;
   (b) ::Clk *clkList*;
   (c) ::Reset *rstList*;
   (d) ::Ports;
   (e) ::Nets;

   (f) ::Wires;
   (g) ::Regs;
   (h) ::Always *seq -clk 'clk' -rst 'rst'—comb*;
   (i) ::Instance *-f filename -m module instanceName*;
   ::s/string1/string2/;

   The tool will store the following in a internal datastructure.

   (a) module name
   (b) declared ports
   (c) declared nets - wires and regs
   (d) instances and already connected ports
   (e) always blocks and sensitivity lists
   (f) all nets used as lvalues (count and context)
   (g) all nets used on right side of expressions (count and context)

   Casey please add details od DB here

2. Stage2: Analyze the design In this stage RTLgen will do the following tasks

   (a) For each net, count number of source and sinks and the context of source or sink and cataegorize each net as input, output, wire or reg. Compare against the already declared ports, wires and regs, if any and check the type and width are declared correctly. Keep track of undeclared ports, wires and regs.
   (b) For each always block, create sensitivity list
   (c) For each instance, read the file which has the module defined, recursively call RTLgen if it is also **.ehv** file, and keep track of portlist of the instance and regexp replace for connectivity.

3. Stage3: Output full blown RTL

   (a) **::Module *moduleName*;**
   expand portlist
   (b) **::Clk *clkList*;**
   (c) **::Reset *rstList*;**
   (d) **::Ports;**
   expand input, output and inout declarations
   (e) **::Nets;**
   wires and regs declarations
   (f) **::Wires:**
   wires declarations

(g) **::Regs:**
regs declarations

(h) **::Always *seq -clk 'clk' -rst 'rst'—comb*:**
expand sensitivity list, use default clock and reset if not specified for a seq block

(i) **::Instance *-f filename -m module instanceName*:**
expand instance connectivity

3 Detailed description of supported funtionality

- put here

4 RTLgen pseudo RTL language

5 Examples

6 Next Steps

1. add support for VHDL