Testpoint Synthesis Using Symbolic Simulation

Kai-hui Chang, Yen-ting Liu, and Chris Broy

Motivation

- Testbenches may rely on testpoints to obtain necessary information
- Testpoints can be registers, wires, or combinations of both
- For example, TARMAC for ARM processors
- After synthesis, registers are mostly preserved, but wires can be gone
- Testpoints that access wires must be reconstructed from the netlist for the testbench to work
- This process can be tedious, error prone and time-consuming

Solution

- Testpoint synthesis
  - Uses symbolic simulation to generate driving equations of design objects
  - Automatically extracts the logic functions of RTL wires and decomposes them based on registers and inputs of the block
- A flow has been implemented to reconnect RTL test points to gate-level netlists

Use Model

- User provides hierarchical references that need to be converted from RTL to gates
  - Assume define macros are used for hierarchical references for easier mapping between RTL and gates
  - For example
    - RTL: "DUT_CORE.var1" for "top.dut.core.var1"
    - Auto mapped to "DUT_CORE.var1" for gate-level
    - "DUT_CORE.tb_conn.var1"
- User provides a table for the information needed by the tool
  - Macro name, RTL module name, additional RTL hierarchy under module, RTL hierarchy, gate-level hierarchy, gate-level module name
  - We need "additional" RTL hierarchy under module because the block may be flattened in the netlist
- Tool automatically reads RTL modules to decompose the testpoints to registers and primary inputs of modules

Example (Cont'd)

- Next tool automatically reads gate-level netlists to identify registers and block inputs for testpoint reconstruction
  - A Perl script is provided for matching RTL/gate-level signals based on the naming convention of the synthesis tool and options
  - The matching template needs to be revised the first time a design uses the tool
- Tool outputs
  - A new macro for each hierarchy macro that contains testpoints which need to be mapped
  - Macro file that redefine hierarchy defines for all mapped variables
  - RTL: define `hier_func Arb` `ib_dut` `usb_functional` `ib`
  - Gate: define `hier_func Arb` `hier_func Arb` `ib_conn`
- Now run gate-level simulation using tool outputs
  - Replace RT-level `hier_func Arb` `macro` with tool generated macro, tool generated modules and redefine macro definitions

Example

- Testbench test access points:
  - `o$display("func.arb.sram.adr= %d", hier_func Arb.sram.adr);`
  - `r$display("func.arb.sram.adr= %d", hier_func Arb.sram.adr);`
  - `define hier_func Arb tb_dut` `usb_functional` `ib` (RTL)
- Macro table file
  - `hier_func Arb` `usb_functional` `ib` `tb` `usb_functional` `ib` `usb_functional` `ib`
- Tool generates new `define` for all hierarchies with mapped test access points
  - `define hier_func Arb hier_func Arb` `ib` `conn` `save in "tb_conn_new_define.vh"

Handling Synthesis Optimizations

- Problem: Some FF outputs may be inverted by synthesis tool
  - Solution: Compare RTL/gate-level waveforms to determine polarity of FFs
    - Read RTL/gate-level waveforms at reset deassertion
    - If the values are different, then invert the gate-level FF output
- Problem: Some inputs are optimized away in a block
  - Solution1: Decompose at a higher-level of hierarchy where the signal still exits
  - Solution2: Search for ports of the block with similar names and compare RTL and gate-level waveforms

Discussions

- Testpoint synthesis using symbolic simulation is more efficient than using logic synthesis
  - Word-level operations are preserved, thus reducing simulation time
  - Word-level optimizations can greatly simplify driving equations
- The proposed solution can handle most cases but may not fully reconstruct a few signals due to special synthesis optimizations
  - Without output from synthesis tools, it is difficult to achieve 100% reconstruction
  - In our experience, only one or two testpoints need to be reconstructed manually for the given designs, which can already significantly save engineers' time