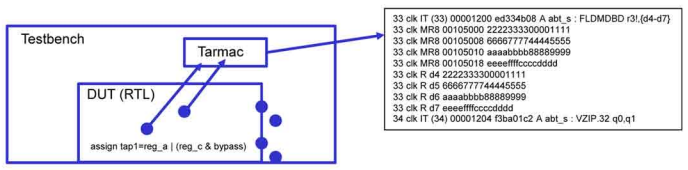
Testpoint Synthesis Using Symbolic Simulation

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

avery design systems

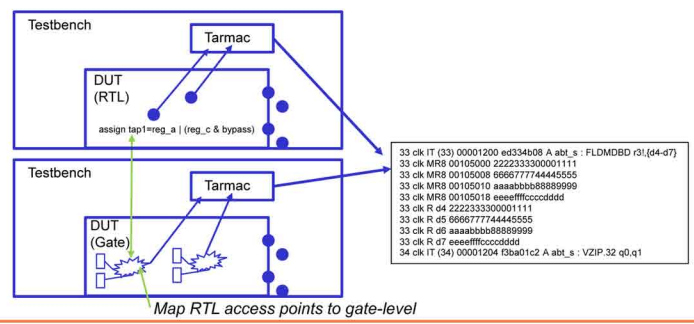
Motivation

- 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 decompose 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

Use Model (Cont’d)

- Next tool automatically reads gate-level netlists to identify registers and block inputs for testpoint reconnection
  - 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 module 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_funct_arb** tb.dut.usb_funct.u2
    - Gate: `define **hier_funct_arb** hier_funct_arb_tb_conn
- Now run gate-level simulation using tool outputs
  - Replace RT-level hierarchy `define macros with tool generated macro file, tool generated modules and redefined macro defines

Example

- Testbench test access points:
  - $display("funct.arb.wsel= %b", `hier_funct_arb.wsel);
  - $display("funct.arb.sram_adr= %b", `hier_funct_arb.sram_adr);
  - `define hier_funct_arb tb.dut.usb_funct.u2 (RTL)
- Macro table file
  - `hier_funct_arb,usbf_top,.u2,tb.dut.usb_funct,tb.dut.usb_funct,usbf_top
- Tool generates new `defines for all hierarchies with mapped test access points
  - `define hier_funct_arb hier_funct_arb_tb_conn
  - Save in “tb_conn_new_define.vh”

Example (Cont’d)

- Testbench changes to run gate-level simulation
  - Add new top-level module “hier_funct_arb_tb_conn” to compile list
  - Use tool generated macro defines (`include “tb_conn_new_define.vh”)
- All global references will be resolved by the tool-generated modules
- Tool generated module for reconnection

```
module hier_funct_arb_tb_conn;
reg [14:0] sram_adr;
reg wsel;
// Assignment for usbf_top.u2.sram_adr
assign sram_adr = xid13241 ;
// Assignment for usbf_top.u2.wsel
assign wsel = ~(tb.dut.usb_funct.u1_u2_word_done_r_reg.n0 |
(~tb.dut.usb_funct.u1_u2_mack_r_reg.n0 & xid11658)) & ((tb.dut.usb_funct.u2_wack_r_reg.n0 &
~(tb.dut.usb_funct.u1_u2_word_done_r_reg.n0 | (~tb.dut.usb_funct.u1_u2_mack_r_reg.n0 &
xid11658))) | xid12378) ;
……
```


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