ECEN689 Fall 2018 Dependable Learning System

Final Exam

Chun-Hao Huang 627002917

I. MODEL CHECKING TOOL

In this report, I use EBMC, which is a model checking tool for hardware designs, to verify the properties of the design.

II. DESIGN UNDER VERIFICATION

The design under verification is a synchronous FIFO memory that can store 10 8-bit words. The FIFO status flags include full and empty. A separate register is used to keep track of how many words are in the FIFO. When the FIFO is empty, it can only do write operation. When the FIFO is full, it can only do read operation. Based on the design, I write three assertions to check the properties of the design. The first property to verify is that full and empty cannot be asserted at the same time. The second property is that FIFO should keep full asserted until a read operation occurs. The third property is that FIFO should keep empty asserted until a write operation occurs. These properties can be formulated in LTL formulas.

FIFO is full: *fifo_is_full*
FIFO is empty: *fifo_is_empty*
Write enable signal is on: *write*
Read enable signal is on: *read*

- Property 1: fifo_no_full_and_empty
  $\varphi_1 = \square\neg(fifo\_is\_full \wedge fifo\_is\_empty)$
  ```
  assert property (!(Full && Empty));
  ```
- Property 2: fifo_remain_full_until_read
  $\varphi_2 = \square(fifo\_is\_full \cup read)$
  ```
  assert property ((Full & !rd_en & !reset) |=> Full);
  ```
- Property 3: fifo_remain_empty_until_write
  $\varphi_2 = \square(fifo\_is\_empty \cup write)$
  ```
  assert property (Empty & !wr_en |=> Empty);
  ```

Figure 1: Symbol View of FIFO

III. SIMULATION RESULTS

The following picture is the snapshot of the tool output. The model checking tool first checks the base case and reports the counterexample if the properties are violated. In the next step, the tool checks the step case. Our results show that all of three properties hold in the design.

```
using default bound 1
Parsing FIFO.sv
Converting
Type-checking Verilog::FIFO
Induction Base
General constraints
Initial state
Transition relation
Transition 0->1
Transition 1
Solving with propositional reduction
Checking FIFO.property.1
UNSAT: No counterexample found within bound
Checking FIFO.property.2
UNSAT: No counterexample found within bound
Checking FIFO.property.3
UNSAT: No counterexample found within bound
Induction Step
General constraints
Transition relation
Transition 0->1
Transition 1
UNSAT: inductive proof successful, property holds
General constraints
Transition relation
Transition 0->1
Transition 1
UNSAT: inductive proof successful, property holds
General constraints
Transition relation
Transition 0->1
Transition 1
UNSAT: inductive proof successful, property holds

** Results:
[FIFO.property.1] always !(FIFO.Full && FIFO.Empty): SUCCESS
[FIFO.property.2] always FIFO.Full && !FIFO.rd_en && !FIFO.reset |=> FIFO.Full: SUCCESS
[FIFO.property.3] always FIFO.Empty && !FIFO.wr_en |=> FIFO.Empty: SUCCESS
```

Also, I try to modify the code and write a bug in the design. This bug makes the empty and full signal trigger at the same condition. The tool outputs the counterexample trace. The results show that property 1 is violated. From the counterexample trace, we can observe that the full and empty signal are high simultaneously.

```
[FIFO.property.1] always !(FIFO.Full && FIFO.Empty): FAILURE
Counterexample:

Transition system state 0
----------------------------------------------------
  FIFO.Data_In = 0 (00000000)
  FIFO.wr_en = 0
  FIFO.rd_en = 0
  FIFO.clk = ?
  FIFO.reset = 1
  FIFO.Data_Out = 0 (00000000)
  FIFO.Full = 1
  FIFO.Empty = 1
  FIFO.memory = { 0, 2, 1, 0, 32, 4, 16, 1, 8, 1 }
  FIFO.rd_ptr = 19 (0000010011)
  FIFO.wr_ptr = 63 (0000111111)
  FIFO.depth_cnt = 9 (0000001001)
```