SoC Chapter 7: Formal Verification

Modern SoC Design on Arm
End-of-Chapter Exercises

Q1 What is declarative proof?

Define the following classifications of programming languages and systems: declarative, functional, imperative, behavioural and logic. What class are the following languages: Prolog, SQL, Verilog, C++, Spec- man Elite, PSL and LISP?

Q2 Assertions over an RTL design.

The synchronous subsystem in Figure 1 has three inputs: clock, reset and start. It has one output called Q. It must generate two output pulses for each zero-to-one transition of the start input (unless it is already generating pulses). Give an RTL implementation of the component. Write a formal specification for it using PSL or SVA. Speculate whether your RTL implementation could have been synthesized from your formal specification.

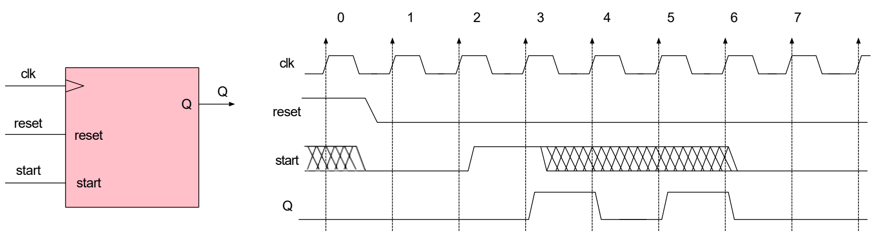

Figure 1: A pulse generator: schematic symbol and timing waveforms.

Q3 Model checking a FIFO

Create a formal glue shim like the one in Figure 2 to check the correctness of a FIFO component.

Q4 Model checking a RAM

Create a similar formal proof of the correctness of a RAM, showing that writes to different locations do not interfere with each other.

Q5 Sequential Equivalence Checking

Prove the equivalence of the two designs in Figure 3 by naming each state in each design and defining a minimal FSM whose states are each labelled with the list of states in each input design that they model.

Q6 Dynamic Validation using Formal VIP

In the book it says 'Implement the checker described in the `bus-checker` folder of the additional material' and that content is pasted below on this exercise sheet.

1a: Design at the gate-level an arbiter for three customers and a single resource and say what basic type

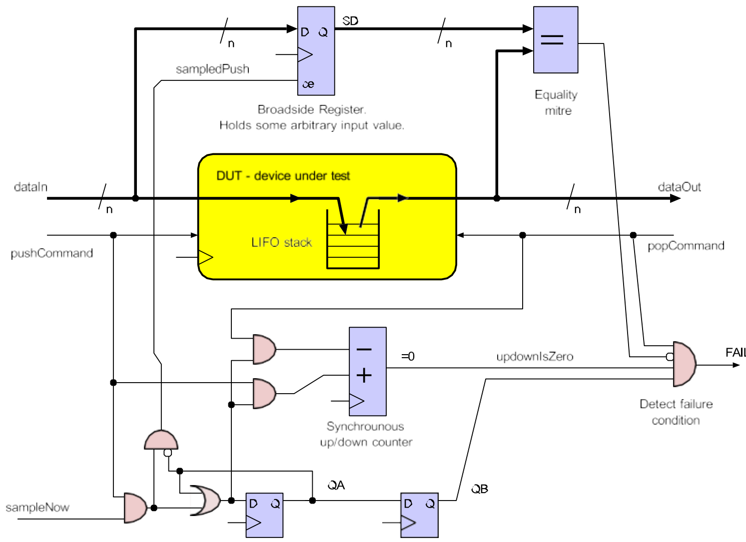

Figure 2: A harness around a data path component (a LIFO stack).

of arbiter it is. (You do not need to include details of the resource or customers.)

1b: Each customer will interact with the arbiter using a protocol, typically using one request and one grant signal. Give a formal specification of this protocol using a state transition diagram. For a synchronous protocol, explain how the concept of the clock is embodied in the diagram.

[Step 2 - Simple protocol safety checking using hardware monitors.]

2a: (easy) Give a completely separate RTL or gate-level design that is a monitor (or checker) for the following safety property: 'At all times, never is the resource granted to more than one requester'. This checker should be a component (e.g., separate RTL module) that has as many inputs as is needed to monitor the

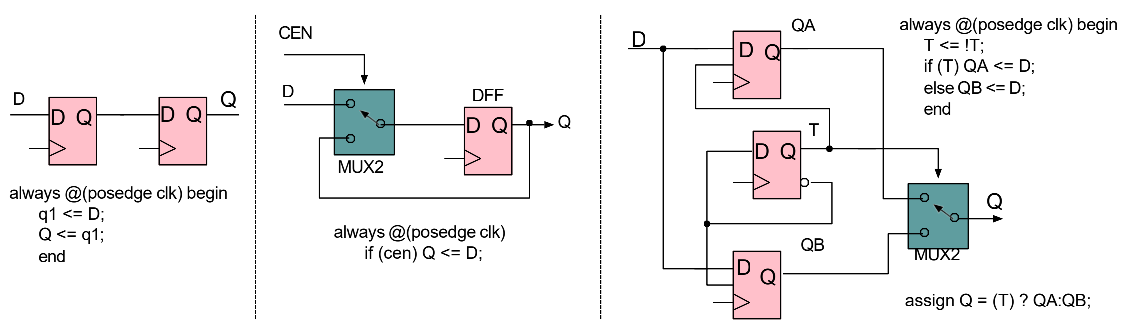

Figure 3: A two-bit shift register (left) with a conventional design. By using a clock-enabled flip-flop (centre), an alternative implementation is possible (right). The state encoding is totally different, but the observable black-box behaviour is identical.

necessary nets in the system (e.g. all connections to the arbiter) and an output that is asserted in any state where the assertion is violated.

2b: (harder) Similarly, design an RTL or gate-level protocol checker that could be instantiated for each connection between a customer and the arbiter that checks each instance of the request/grant protocol is being properly followed. Do you have, or can you envision, a request/grant hardware protocol that has no illegal behaviours? What is allowed to happen in your system if a customer wants to give up waiting for the resource (known as ’baulking’)?

2c: For your particular request/grant protocol design, if you extended 2a to also check that no grant is issued without a request would this be a state or path property checker?

Step 3 - Liveness Checking Machine

3. Give a completely separate RTL or gate-level design that is a monitor/checker of the following liveness property: "Whenever reset is not asserted, when a request is made for the resource, it will eventually be granted".

Step 4 - Formal Logic Implementations

4a. Using PSL, SVA or a similar assertion language, give an assertion that checks the safety property of step 2a above.

4b. Give a similar temporal logic assertion that asserts that the liveness property of step 3 above is never violated.