Verificação Funcional de Sistemas Digitais II, 15 de outubro de 2019

FPV FOR VENDING MACHINE

1. INTRODUCTION

This report presents the formal verification of properties for a vending machine, which is specific to sell three kinds of sandwiches. The formal verification was performed using the SystemVerilog Assertions language and the Cadence's Jasper verification tool.

In this work, it was adopted the formal verification methodology proposed in the textbook of the course, which suggests starting by defining the verification goals, then the main properties to be verified and the exit criteria to be considered to stablished that the verification is completed.

A. Vending Machine Specification

It was chosen the version 4 of the vending machine specification, which consists of a hand drafted state diagram, briefly described by some few sentences and some simulated and captured waveforms.

The vending machine is specified just by means of a state transition diagram, coded in the VHDL language (t_04.vhd). According to this diagram:

i. starting from a state ACTION, the vending machine can accept *n* times one coin by going to a state SOMA, when a signal m100 is equal to 1, incrementing of 1 a count variable, and going back to ACTION;
ii. from state ACTION, it is possible to go to states SGREEN, SATUM, or SBACON, depending on the signals rgreen, ratum, or rbacon to be equal to 1, respectively; as well as, the count variable is greater than 2, 3 or 4, respectively;
iii. from states SGREEN, SATUM, and SBACON the vending machine goes to state NULO, unconditionally;
iv. from state ACTION, it is also possible to go to state NULO, when signal dev is equal to 1, or two of the rgreen, ratum, and rbacon became 1 at the same time;
v. from state NULO, it is possible to go to state DEVOLVE when signal count is greater than zero, where the count is decremented of one, and then back to state NULO;
vi. at all states, a signal busy receives the value 1, except at ACTION state, where busy is equal to 0;

2. FORMAL PROPERTIES VERIFICATION

In order to perform the formal verification of properties for the vending machine, according to the methodology suggested in the textbook, we started by identifying the cover points, then the assumptions, and finally the assertions.

A. Cover Points

To cover all states of the vending machine, the following cover point was inserted in the SVA specification (t_04.sva file):

```
cover_states: generate for (i = ACTION; i<= DEVOLVE; i++)
  begin :g1
    state1: cover property (ea == i);
  end
endgenerate
```

This simple cover property `cover_states` allows to check if all states of the vending machine are somehow reachable.

B. Assumptions

To take some properties as assumed, in order to avoid checking for unnecessary conditions in the vending machine, the following assumptions were included in the SVA specification (t_04.sva file):

```
only_on_the_action_state: assume property (
      (ea != action) and (m100 == 0 && dev == 0) and
      (sgreen == 0 && satum == 0 && sbacon == 0) and
      (busy == 1)
  );

on_reset_only: assume property (
      (reset == 1) and (pe == "000" and count == "00000")
);
```

The `only_on_the_action_state` assumption specifies that all the signals m100, dev, sgreen, satum, and sbacon must be equal to “0” and busy equal to “1” when the current state is not the action state.

The `on_reset_only` assumption specifies that all next state signal and the count signal are both set to zeros.

C. Assertions

In the followings it is presented each of the specified assertions included in the SVA specification (t_04.sva file):

C.1 Assert that just one of the types of sandwich can be chosen at any given time.

```
request_onehot: assert property ( @(posedge clock)
             (r_green == 1 && r_atum == 0 && r_bacon == 0) or
             (r_green == 0 && r_atum == 1 && r_bacon == 0) or
             (r_green == 0 && r_atum == 0 && r_bacon == 1)
);
```

C.2 Only one of the grants can be 1 at a given time.

```
grant_onehot: assert property (  @(posedge clock)
             (green == 1 && atum == 0 && bacon == 0) or
             (green == 0 && atum == 1 && bacon == 0) or
             (green == 0 && atum == 0 && bacon == 1)
);
```

C.3 Each sandwich should be released according to the request and quantity of coins.

```
release_sanduba_sg: assert property ( @(posedge clock)
   (r_green) and (count > 1) ##1 (sgreen && !satum && !sbacon)
);

release_sanduba_sa: assert property ( @(posedge clock)
   (r_atum) and (count > 2) ##1 (satum && !sgreen && !sbacon)
);

release_sanduba_sb: assert property ( @(posedge clock)
   (r_bacon) and (count > 3) ##1 (sbacon && !sgreen && !satum)
);
```

C.4 Assert that just one of the kinds of sandwich can be chosen at any given time.

```
transition1a: assert property ( @(posedge clock)
    (ea == action and r_green == 1 and count > 1) ##1
           ( ea == sgreen and green == 1 and busy == 1 )
);
transition1b: assert property ( @(posedge clock)
    (ea == action and r_atum == 1 and count > 2) ##1
           ( ea == satum and atum == 1 and busy == 1 )
);
transition1c: assert property ( @(posedge clock)
    (ea == action and r_bacon == 1 and count > 3) ##1
           ( ea == sbacon and bacon == 1 and busy == 1 )
);
transition2: assert property ( @(posedge clock)
    (ea == action and m100 == 1) ##1 (ea == soma and busy == 1
                                     and count > 1)
);
transition3: assert property ( @(posedge clock)
    (ea == action and dev==1) ##1 (ea == nulo and busy == 1)
);
transition4: assert property ( @(posedge clock)
    (ea == soma) ##1 (ea == action and busy == 0)
);
transition5: assert property ( @(posedge clock)
    (ea == nulo and count > 0) ##1
           (ea == devolve and d100 == 1 and busy == 1)
);
transition6: assert property ( @(posedge clock)
    (ea == devolve) ##1 (ea == nulo and busy == 1)
```

```
  );
  transition7: assert property ( @(posedge clock)
      (ea == nulo and count == 0) ##1 (ea == action and busy == 0)
  );
  transition8: assert property ( @(posedge clock)
      (ea == sgreen or ea == satum or ea == sbacon) ##1
             (ea == nulo and busy == 1)
  );
)
```

3. VERIFICATION COMPLETENESS

The specified assertions cover all the possible state transitions, according to the provided state transition diagram of the vending machine.

4. MAIN ISSUES AND THEIR CORRECTIONS

It was noted that without specified some basic assumptions, the formal property verification tool (Jasper) reported many unreachable states. This was corrected just by including the `only_on_the_action_state` and `on_reset_only` assumptions in the SVA specification.

5. DIFFICULTS AND LESSONS LEARNED

I found out that one of the main difficulties in this kind of design activity, it was just to identify the adequate property types and right features of the SVA specification language to use for each case to be considered in the design under test, in order to guarantee that the specification is “correct” (or at least satisfactory).