NTNU – Trondheim
Norwegian University of
Science and Technology

DEPARTMENT OF ELECTRONIC SYSTEMS

EXAM IN COURSE TFE4171 DESIGN OF DIGITAL SYSTEMS II

Contact: Kjetil Svarstad
Tel.: 458 54 333
Examination date: May 15, 2017
Examination time (from - to): 0900-1300

Permitted support material: C–Specified printed and hand-written support material is allowed. A specific basic calculator is allowed.

Other information: **Maximum number of points** per task and sub-task are given in the text. **Maximum number of points** totally: **60**.

The **final grade** is calculated by summing your points from this exam with the exercises scaled to 20 points and the optional term project (if delivered and result contributes positively). Sum is then scaled to 100.

NB: This exam must be **passed** to pass in total. It is not sufficient that the total grade is a pass grade (E or better), the grade on the exam itself must also be E or better.

Language: English
Number of enumerated pages: 14
Additional pages in enclosures: 0

Controlled by:

Dato Sign

Intentionally left blank

Problem 1 Multiple choice (18 points)

Answer by circling the answer alternative you believe is the correct answer. You are awarded 3 points for a correct answer and 0 points if you do not answer. If your answer is wrong or you circle more than one alternative, you will get -1.5 point. If you need to change your answer, cross out your circled number and circle the right one.

a) (3 p) Which statement is NOT true

1. Cover statements can not contain a FAIL action
2. Cover statements can not contain a reset condition
3. Both `cover property` and `cover sequence` are legal statements.

Answer:	1	2	3

```
assign sig1 = 1;
assign sig2 = 1;
always_comb a1: assert #0 (sig1 == sig2);
```

b) (3 p) Consider the above SVA code snippet. Assuming `sig1` and `sig2` are initially zero, which of the following is a possible result?

1. Depending on execution order, the assertion `a1` may fail due to a simulation race.
2. The assertion `a1` will vacuously pass.
3. The assertion `a1` will never fail.
4. Depending on execution order, the assertion `a1` may fail due to a glitch.

Answer:	1	2	3	4

```
assert property (@(posedge clk) always s_eventually p);
```

c) (3 p) The assertion above means that the signal p must be:

1. asserted at every clock tick in the future
2. asserted at least once in the future
3. asserted at the current time and at least once in the future
4. asserted infinitely often in the future

Answer:	1	2	3	4

d) (3 p) Which of the following statements is true for a given Kripke model:

1. CTL formulas can be associated with a set of paths in which they are valid.
2. CTL formulas can be associated with a set of states in which they are valid.
3. CTL formulas can be associated with a set of inputs for which the formula is valid.
4. CTL formulas can be associated with a set of outputs that fulfil the formula.
5. CTL formulas can always be associated with a set of state transitions in a Kripke model.

Answer:	1	2	3	4	5

e) (3 p) Let a, b, c be CTL formulas. Which of the following is NOT a CTL formula:

1. $a \vee \mathrm{EX}a$
2. $a \vee b$
3. true
4. $\mathrm{E}(a \;\mathrm{U}\; (\mathrm{AF}b))$
5. $\mathrm{AF}(a\mathrm{X}b)$

Answer:	1	2	3	4	5

f) (3 p) Which of the following statements is true:
CTL formulas can be evaluated by

1. computing the set of reachable states starting from the initial state
2. using the fixed point characterizations of the CTL operators to iteratively compute the state sets associated with the formula
3. using a SAT-solver to jump out of local fixed points
4. ruling out false negatives obtained in the reachable state set

Answer: 1 2 3 4

Problem 2 SystemVerilog Assertions (12 points)

a) (3p) Describe in short at least 3 new features in System Verilog that is not a part of standard Verilog.

Answer:

b) (4p) Explain the problem of *vacuity/vacuous success/vacuous pass*. Use the following property as an example. Discuss how this problem can be solved by the verification engineer by changing the property description. Mention also how some tools remedy this problem.

```
property pr_r_q;
  @(posedge clk) req |-> ##2 gnt;
endproperty;

assert property (pr_r_q) $display($stime,,,"%m PASS");
  else $display($stime,,,"%m FAIL");
```

Answer:

c) (2p) Show a timing diagram or signal trace that matches the following sequence:

```
(a ##2 b) ##0 (c ##1 d)
```

Answer:

d) (3p) Show by timing diagram or signal trace matching sequences for the two properties p1 and p2. Explain why they are different or equivalent.

```
p1: assert property (@(posedge clk) a |-> ##1 b);
p2: assert property (@(posedge clk) a |=> ##2 b);
```

Answer:

Problem 3 (6 points)

For the Kripke state diagrams in Figure 1:

a) (3p) Mark the states in which the properties hold.

b) (3p) State whether or not the property holds for the system, explain your answer

Answer:

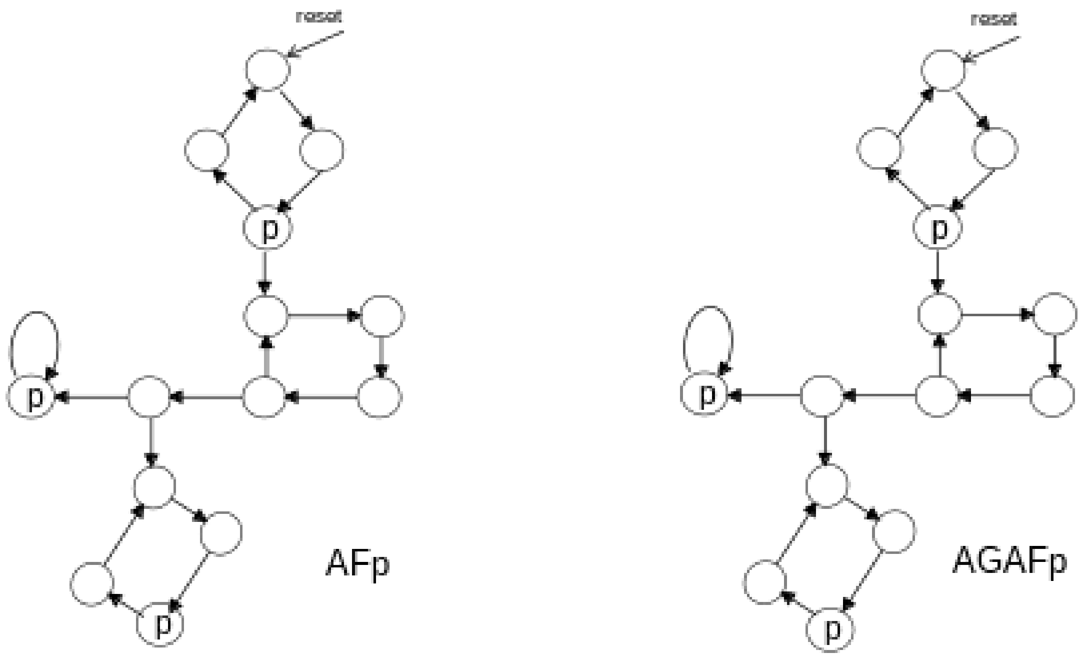

FIGURE 1 – Kripke state diagrams

Problem 4 (12 points)

Consider the design in Figure 2, consisting of two finite state machines. One is a counter. After reset, it counts from 0 to 5 with every clock tick. Once x=5, the value does not change any more until the next reset. The other component implements the finite state machine of Moore type shown in Figure 3

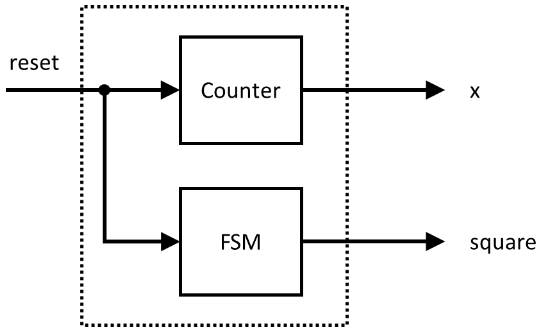

FIGURE 2 – System of 2 FSM's

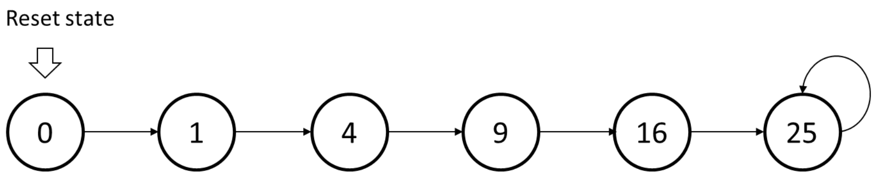

FIGURE 3 – Moore FSM

The numbers shown in the nodes are both, the state of the FSM and its output in that state, name "square". For the design, obviously, after reset, at any point in time the output "square" represents the square of the output "x".

a) (3p) Write an SVA property that expresses that, at all times, square = x^2.

Answer:

b) (4p) For the sequence of natural square numbers it holds that the distance between two consecutive squares grows by two. For example, (4-1)=3, (9-4)=5, (16-9)=7, The following SVA property checks this relationship.

```
property sva_squares_difference;
  !reset ##1 !reset ##1 !reset |->
    (square - $past(square, 1) ==
    $past(square, 1)-$past(square, 2) + 2);
endproperty
```

An interval property checker returns the counterexample in Figure 4 to this property.

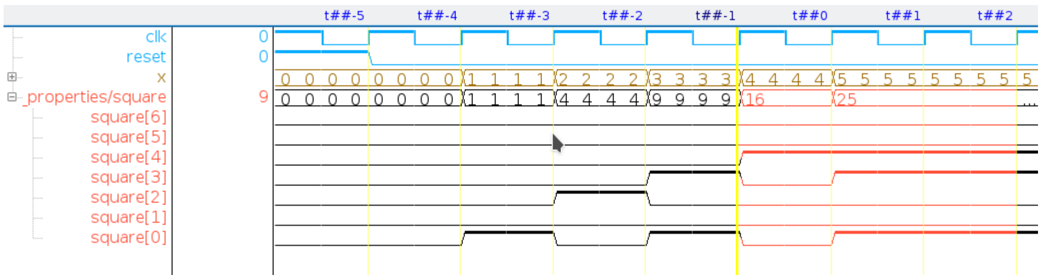

FIGURE 4 – Counterexample

- Is this a true or a false counterexample?
- How can the problem be fixed?

Answer:

c) (5p) The following operation property expresses that when the FSM begins in state 0, a sequence of numbers x and their squares will be produced by the design, as shown.

```
property square_sequence;
  t ##0 square == 0 implies
    t  ##0 x == 0 && square == 0 and
    t  ##1 x == 1 && square == 1 and
    t  ##2 x == 2 && square == 4 and
    t  ##3 x == 3 && square == 9 and
    t  ##4 x == 4 && square == 16 and
    t  ##5 x == 5 && square == 25;
endproperty
```

An interval property checker returns the counterexample in Figure 5 to this property.

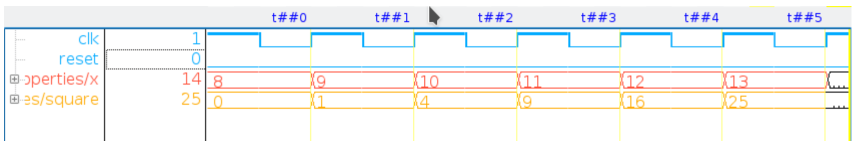

FIGURE 5 – Second counterexample

- Is this a true or false counterexample?
- How can the problem be fixed?

Answer:

Problem 5 SystemC (12 points)

a) (2p) How many times will the `countme` method in the SystemC module code below be executed, and what will be the output? Explain your answer.

```
class count : sc_module {
  SC_CTOR(count) {
    c = 0;
    SC_METHOD(countme);
  }
  void countme () {
    c++;
    cout << c << " ";
  }
  int c;
};
int sc_main () {
  count count_inst;
  sc_start (100);
}
```

Answer:

b) (6p) Explain how the use of event modelling (use of `sc_event`), dynamic sensitivity lists, interface methods, and Transaction Level Modelling with “local time” can improve system modelling, HW/SW codesign, and simulation performance.

Answer:

c) (4p) SystemC like many Hardware Description Languages uses Discrete Event Simulation as the semantics (“meaning”) of the modelling code. Such a simulator can be either *preemptive* or *non-preemptive*. Which one is the case for SystemC, and explain what that means.

Answer: