Assignment 3: Requirements and Verification

1DT059: Model-based Design of Embedded Software

November 8, 2022

In this assignment you produce Simulink/Stateflow models and text. All models (and code) should be well-structured and readable, and use named constants wherever appropriate. Your solutions to each exercise exercise should be implemented in a .slx-file and/or .pdf-file that is named after the exercise. Include your name at the top of all submitted files.

> **Assignments are to be solved by students individually.** You can (and are encourage to) discuss ideas and concepts with fellow students, but it is absolutely forbidden to share or copy (even parts of) solutions, lines of code, or similar, from each other, or from models found on internet.

Problem 1 Bridge Crossing (30p)*:*
Solve the Bridge crossing problem using Simulink Design Verifier

> A group of four people must cross a bridge. It is dark, and the bridge is fragile. To cross it, one needs to bring a torch, and the bridge can carry at most two persons simultaneously. The group of people have only one torch, meaning that the torch must follow the persons that cross the bridge back and forth across the bridge. The four persons walk at different speeds; it takes them 1, 2, 5, and 10 minutes to cross the bridge, respectively. Since the bridge carries at most two persons, at most two persons can cross in one "move", which takes time equal to that which is needed by the slower person crossing. E.g., if the first move is that the 2-minute person and the 5-minute person cross, then that move takes 5 minutes. Thereafter one of them must go back with the torch before any other person can cross, etc. What is the shortest time they need to cross the bridge?

Make a structured Simulink/Stateflow, using which SLDV can find the solution to this problem. This means that your model should **not** provide any hint about how the group should cross (i.e., in which order the persons should move across). It should only model the above rules of the problem. *Finding* a successful shortest-time plan which gets all persons across the bridge should be done by the Property-proving engine in Simulink Design Verifier. A possible stragety is for the model to contain an integer "parameter" n, and "ask" SLDV whether there is a way to cross the bridge in at most n minutes, and then repeat this "question" for different values of n.

Your solution should include

- A model, which SLDV analyzes to find a shortest time, and a model which allows SLDV to deduce that ther is no shorter-time plan. A description of how the information provided by SLDV is interpreted as a "schedule" for crossing the bridge in a shortest time.
- An explanation of your model and how it captures the rules of the problem.

Problem 2 Requirements for Elevator (30p)*:*
In this problem, the context is the elevator controller of Assignment 2. For the elevator system (consisting of controller and simple elevator model) that you constructed in Assignment 2, you should formulate requirements. Each Requirement (numbered R1 – R6 below) should be formalized by a Simulink/Stateflow monitor which has relevant signals of the elevator controller as input, and with an output signal that can be sent to a `Proof Obligation` or an `Assertion` block. In this problem, the focus is on designing the monitor so that it precisely captures the formulated requirement, i.e., it should signal a violation in exactly all scenarios where the requirement is violated. Therefore, do not test your requirements together with your elevator model. Instead, use the signal builder block to generate both violating and non-violating input scenarios and check that your monitor signals a violation for the violating scenarios. The requirements are as follows.

R1: The doors are never open when the elevator is not at some floor. This requirement can be checked by a monitor whose input is the signal to the lamp which denotes that doors are open, and the signal *position* from the elevator.

R2: When doors are opened, they remain open for at least 5 seconds.

R3: The elevator stops at floor A only if the button for floor A has been pressed previously. Here, you can choose a suitable value for A, such as 2 or 3.

R4: The elevator stops at floor A only if the button for floor A has been pressed previously and after the last time that the elevator stopped at and left floor A.

R5: If the button at floor A is pressed after the last time (if any) that the elevator stopped at and left floor A, and also the button at floor B hat not been pressed since the last time (if any) that the elevator stopped at and left floor B, then thereafter the elevator will stop at floor A before stopping at floor B. Choose concrete values (floors) for A and B.

R6: After a button-push for floor A, the elevator arrives within d time units to floor A. Choose concrete values for A and d.

For each requirement, you should hand in at least one violating and one non-violating scenario, in the form of a signal-builder block.

Problem 3 Testing Elevator Requirements (30p):
In this problem, Simulink Design Verifier (SLDV) is used to generate test suites for the elevator controller. The steps are as follows.

1. Exclude the elevator model from your design since the focus is only on the controller.

2. Run SLDV on your controller under three coverage criteria: *Decision*, *Condition/Decision*, and *MCDC*. If the tool reports that the model is *incompatible* to test generation, you need to find the reason and update the controller to resolve incompatibility.

3. After test generation completes, the number of *unsatisfied* objectives is reported. Unsatisfied objectives usually mean dead logic in your design. Find the cause of unsatisfied objectives, if any, and try to fix it if it is rooted in wrong/dead logic in your design.

4. Consider the test suite generated under the "Condition/Decision" coverage criterion. For each test case (at most four): (a) specify what the test case is doing in terms of the elevator functionality; (b) write the expected outputs (of the controller); (c) run the test case on the controller model[1] and compare the observed output with the expected ones. If the observed output does not conform to the expected one, explain why this is the case. If this is due to an issue in the controller, try to fix the defect.

Below are some points to consider in test generation

- Let SLDV run for at least an hour on each coverage criterion if needed.
- You should remove the user interface blocks ("buttons" etc.) by open input blocks, for which the generated test suite will define sequences of values.
- You can focus the test generation by restricting the values of inputs (e.g., to [0,1]).

Deliverables: (1) The report generated by SLDV for each coverage criterion (the pdf file); (2) justification of unsatisfied objectives (if any); (3) the items asked in item 4 above. In addition, you need to specify any changes made in the controller (e.g. to make the model compatible to test generation, fix detected defects ...).

Problem 4 Test Generation for Microwave Oven (10p):
Consider the controller you designed for a microwave oven in Assignment 2. For each of the following coverage criteria, write test-case(s) that satisfy it. Describe the test case in terms of a set of inputs and the expected output. Specify inputs and outputs as signals (i.e., values during the considered time interval). Make sure that you specify all inputs, even those that are not changed during the test case.

a) "All-boundaries" coverage with respect to the "cooking power" input (i.e., assume the other inputs to be constant).

[1] You may run test cases by "Open harness model"

b) “All-boundaries” coverage for Start button (the button is considered as an input with two possible values).

c) “All-boundaries” coverage for Stop button.

Submission

Solutions (all files) to this assignment are to be submitted via the Student Portal by **Wednesday, November 23, 2022**