Embedded Systems

Duration : 1h30 **All documents allowed**

The two parts are independent.
Informal explanations in plain English will be appreciated a lot, and it is compulsory to justify all answers.
The number of points associated with each question is only an indication and might be changed slightly.

Part I - Modeling Time and Concurrency, Validation (10 points)

We consider a simple hardware device having three operating modes: `A, B, C`. The initial mode is `A`. The only possible mode changes are: from `A` to `B`, from `B` to `C` and from `C` to `A`. Moreover, each mode change takes some time: `tAB` (resp. `tBC, tCA`) unit of time from `A` to `B` (resp. `B` to `C`, `C` to `A`). Finally, the device will break one day if is used in such a way that the time spent in state `B` is less than `minB` units of time.

This device can be used by some software, issuing commands to ask for mode changes. The commands are: `gotoA`, `gotoB` and `gotoC`. The software can also wait for some time using the function `wait (T)` where `T` is the number of units of time.

▷ **Question 1 (3 points) :**

— Give an example software code that uses the device correctly and explain why.
— Give several examples of software codes that use the device incorrectly, explain why, and what would be the effects of the bug.

▷ **Question 2 (3 points) :**

Represent the behavior of the device by an automaton, **capturing all the elements of the above description**. Justify carefully the type of automaton you use, and why. Explain how each element of the above description is taken into account.

Now we would like to perform formal validation of the system (in order to check the way the software uses the hardware device). We have to reason in terms of the set of *global states* of the system made of: the hardware device, and the software.

▷ **Question 3 (3 points) :**

— Take one of your software examples from question 1, and describe its behavior as an automaton. Justify the states very carefully.
— Give the set of global states that have to be checked in order to be sure that the software uses the hardware correctly.

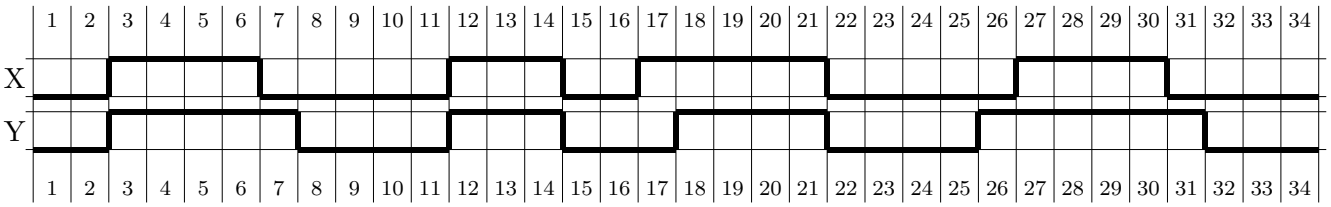
▷ **Question 4 (1 point) :**

What kind of formal verification tool can be used to help validate the correct use of the hardware by the software?

Part II - Synchronous Approach (10 points)

The goal of this exercice is to write Lustre observers that check whether two Boolean flows can be considered as "almost equal". There is not a single, obvious definition of what "almost equal" means; we then consider two possible definitions.

First definition: levels. Intuitively a "level" is a Boolean flow that is true for a while, then false for a while, etc. Two levels can be considered as "almost equal" if they never differ during more than a given amount of time. To simplify, we decide that they must not differ more than one instant. The following timing-diagram gives an example of two flows satisfying this property:



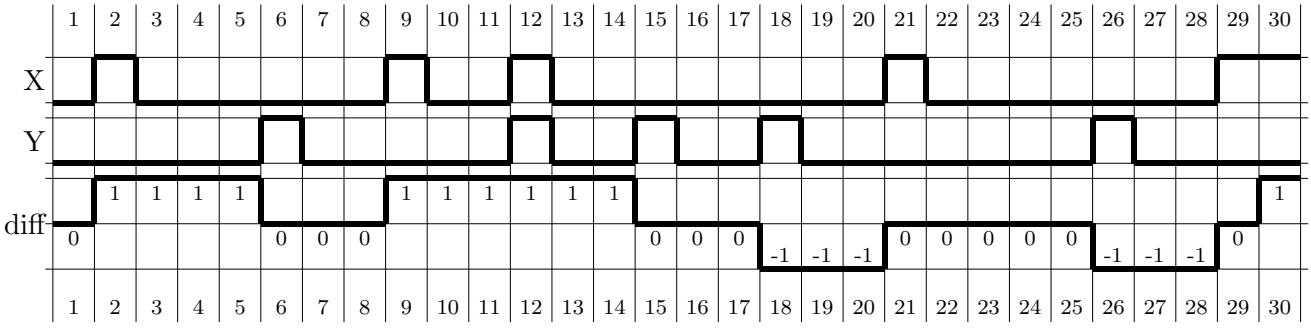
▷ **Question 5 (5 points) :**

Write a Lustre observer whose header is:

```
node levels_almost_equal(X, Y: bool) returns (ok: bool);
```

such that the output `ok` remains always true if and only if `X` and `Y` are almost equal according to the "level" definition above.

Second definition: clocks. Intuitively a "clock" is a Boolean flow that is true for one instant, from time to time. Two clocks are "almost equal" if they never differ of more than two occurrences. More precisely, let `diff` be the difference between the number of occurrences of `X` and the number of occurrences of `Y`, `diff` must always remain between -1 and $+1$. The following timing-diagram gives an example of two flows that have this property (together with the corresponding integer flow `diff`).



▷ **Question 6 (5 points) :**

Write a Lustre observer whose header is:

```
node clocks_almost_equal(X, Y: bool) returns (ok: bool);
```

such that the output `ok` remains always true if and only if `X` and `Y` are almost equal according to the "clock" definition above.