|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Document no. |  | | | | | | | TFE4171 – Design of Digital Systems 2 |
| TFE4171-Ex3-01 |
| Revision | | Date | | | | |
| 1.0 | | 13.03.2015 | | | | |
| Revision By | |  | | | | |
| Hope / Grindheim | |  | | | | |
| ­ | | | | | | | |
| Laboratory Report | | | | | |  | |
| Exercise III | | | | | |  |  |
|  | | | | | | |
| ­ | | | | | | | | |
| Author(s):  B.Hope  S.Grindheim | |  | |  |  | | | |
| Revision Notes | * 1.0: Initial release | | | |
|  |  | | | |
|  |  | | | |
|  |  | | | |
|  |  | | | |
| ***Bengt Erik Hope / Svein Grindheim © 2015 Bergen, NORWAY*** | | | | | | | | |

**🗎Page intentionally left blank**

* Abstract

*This document attempts to answer the questions as listed in the third exercise in the NTNU course TFE4171 – Design of Digital Systems 2.*

Contents

[Assignment 1 1](#_Toc414041238)

[**Task 1.1** 1](#_Toc414041239)

[General methodology of formal property checking using OneSpin 1](#_Toc414041240)

[Analyzing the D-flipflop 1](#_Toc414041241)

[**Task 1.2** 1](#_Toc414041242)

[Assignment 2 2](#_Toc414041243)

[**Task 2.1** 2](#_Toc414041244)

[Assignment 3 4](#_Toc414041245)

[**Task 3.1** 4](#_Toc414041246)

[**Task 3.2** 4](#_Toc414041247)

[**Task 3.3** 5](#_Toc414041248)

[**Task 3.4** 5](#_Toc414041249)

[**Task 3.5** 5](#_Toc414041250)

[Assignment 4 7](#_Toc414041251)

[**Task 4.1** 7](#_Toc414041252)

[**Task 4.2** 9](#_Toc414041253)

[**Task 4.3** 10](#_Toc414041254)

# 

## 

### General methodology of formal property checking using OneSpin

To generate the hardware module for a DUT[[1]](#footnote-1), we elaborate and compile the design in the FV tool OneSpin. When done we can switch to its module verification mode and load the source file (which in our case is written in SVA) containing the properties we wish to use (assert) to formally verify that the module implements the intended behavior. We are now able to tell OneSpin to prove all the assertions specified in the SVA-file. As we do this OneSpin will tell us the status and validity of the proof – e.g. did the assertion pass and is the proof up to date? If any of our assertions fail counterexamples are generated and we can analyze the design using OneSpins debugger.

The debugger provides us with many analytical tools – such as the waveform viewer. The debugger helps indicate problem areas in the design by, for example, highlighting parts of the timing diagram depicting erroneous behavior in the waveform viewer. In this way we will evaluate results from assertions and make the necessary corrections.

### Analyzing the D-flipflop

Analyzing the d-flipflop using the provided assertions we find that the property behavior 1 holds because it properly describes the behavior of a standard D-flipflop. Behavior 2 however describes a function that a D-flipflop does not offer and rightly fails as this is not a bug in the DUT, but rather a faulty property.

## 

To verify that the JK-flipflop has the following functionality:

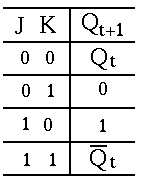


Table 1

We added the following properties and assertions to **jkff.sva**:



Code Excerpt 1

OneSpin report all assertions holds.

# 

## 

In this assignment we analyze an ATM controller which is defined by the **atm.vhd** module for errors. The specification of the design is as follows:

1. A cell is never corrected and dismissed at the same time.

2. An error-free cell is neither corrected nor dismissed.

3. All cells with multiple-bit errors are dismissed.

4. A first erroneous cell coming in is corrected if the error is a single-bit error and not a multiple-bit error.

5. A second erroneous cell is always dismissed.

On the next page follows our complete ‘atm\_property\_suite’ module where all assertions as described in the specification of the design are checked. Please refer to comments in the code for details.



Code Excerpt 2

After running checks in OneSpin based on the assertions listed in Code Excerpt 2, the tool reported a counterexample which is caused by an error in the ATM controller design. The assertion *a\_behavior5* was triggered, meaning that there is a problem with point 5 in the ‘Specification of the design’ list. Alas, a second erroneous cell is ***not*** always dismissed.

The error was traced to line 23 in the supplied **atm.vhd** file:



Code Excerpt 3

The design does not take into consideration the current state when deciding whether to dismiss an erroneous cell. Therefore we made the following addition to the design:



Code Excerpt 4

Now OneSpin report that all assertions hold.

# 

For solving the tasks in this assignment, we used the design provided in subdirectory 03 in the work directory. We will show code excerpts from the complete SVA file containing all assertions and corresponding properties for each task. The complete file is listed after this in order to show the assertions.

## 

In order to prove the correct reset behavior we note that the arbiter should IDLE and the outputs (grant\_o) should be set to 0. We can now write the following property *behaviour1*:



Code Excerpt 5

## 

We write the property *behaviour2* to ensure the first competing request after reset is granted to master 0. We also need a constraint to the environment of the arbiter, namely we *assume* that resource is available on reset. We name this property *assumption*.



Code Excerpt 6

## 

We write the property *behaviour3* to ensure that if there is no request there will be no grant.



Code Excerpt 7

## 

We write the property *behaviour4* to prove that when the resource is free there will be only a single request from one master and that it IDLEs again 2 cycles later.



Code Excerpt 8

## 

We write the property *behaviour5* to verify the correct behavior; if two masters request access simultaneously, the one that did not receive the previous grant is granted access and arbitration shall switch between the two.



Code Excerpt 9

**The complete *arbiter\_property\_suite* module**

On the next page the complete SVA file is listed. Here we see how the assertions is implemented.



Code Excerpt 10

# 

## 



Code Excerpt 11

## 

Code Excerpt 12

## 

Code Excerpt 13

“The property may not hold if you use an expression like “PC == PC + 2”. Instead, if you write it in the form “PC == PC + 16’d2” it may work. Why?”

The reason behind this is that if you don’t explicitly specify the bit width it will default to a size of 32 bits or larger depending on the machine. This may cause undesirable effects during property checking, a lesson that was learned the hard way.

**Asserting the properties**

The properties in this assignment is asserted as follows:



Code Excerpt 14

1. **D**evice **U**nder **T**est [↑](#footnote-ref-1)