|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Document no. |  | | | | | | | TFE4171 – Design of Digital Systems 2 |
| TFE4171-Ex4-01 |
| Revision | | Date | | | | |
| 1.0 | | 13.03.2015 | | | | |
| Revision By | |  | | | | |
| Hope / Grindheim | |  | | | | |
| ­ | | | | | | | |
| Laboratory Report | | | | | |  | |
| Exercise IV | | | | | |  |  |
|  | | | | | | |
| ­ | | | | | | | | |
| 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 fourth exercise in the NTNU course TFE4171 – Design of Digital Systems 2.*

Contents

[Assignment 5 1](#_Toc414410610)

[**Task 5.1** 1](#_Toc414410611)

[**Task 5.2** 1](#_Toc414410612)

[**Task 5.3** 1](#_Toc414410613)

[**Task 5.4** 2](#_Toc414410614)

[**Task 5.5** 2](#_Toc414410615)

[**Task 5.6** 3](#_Toc414410616)

[Assignment 6 4](#_Toc414410617)

[**Task 6.1** 4](#_Toc414410618)

[Assignment 7 6](#_Toc414410619)

[**Task 7.1** 6](#_Toc414410620)

[**Task 7.2** 6](#_Toc414410621)

[**Task 7.3** 7](#_Toc414410622)

[**Task 7.4** 7](#_Toc414410623)

[**Task 7.5** 7](#_Toc414410624)

[Assignment 8 9](#_Toc414410625)

[**Task 8.1** 9](#_Toc414410626)

[**Task 8.2** 11](#_Toc414410627)

[**Task 8.3** 12](#_Toc414410628)

# 

## 

Below follows a state diagram for the main controller in the serial receiver:

Figure

## 

Below is the property reset – with its corresponding assertion and sequence, which proves that right after applying the reset sequence the controller is in state IDLE and the counter is reset to 0.



Code Excerpt

## 

Below is the property called stay\_in\_idle – with its corresponding assertion, which proves that the controller stays in state IDLE if no start bit arrives.



Code Excerpt 2

## 

Below is the property read\_byte – with its corresponding assertion, which proves that a byte is transmitted when applicable and that the controller returns to IDLE after transmission.



Code Excerpt 3

## 

We now restrict the OneSpin tool to use only the basic IPC solver. This will make OneSpin generate a counterexample to the read\_byte property assertion (…had to struggle a bit to find this error as I would have put […] t**##**0 cnt\_s **==** 3'b0 **and t##**0 cnt\_en **==** 1'b0 **implies** […] directly in the cause part – which would make the property pass when asserting it in both configurations.)



Code Excerpt 4

## [[1]](#footnote-1)

We now

# 

## 

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 Except

After running checks in OneSpin based on the assertions listed in CODE EXCEPT 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 Except

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 Except

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 excepts 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 Except

## 

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 Except

## 

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



Code Except

## 

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 Except

## 

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 Except

**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 Except

# 

## 



Code Except

## 

Code Except

## 

Code Except

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

**Asserting the properties**

The properties in this assignment is asserted as follows:



Code Except

1. There is a typo in the exercise text where Task 5.6 is number as Task 5.5 [↑](#footnote-ref-1)