|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Document no. | | | | | TFE4171 – DDS 2 |
| TFE4171-Ex01-0 | | | | |
| Revision | | Date | | |
| draft | | 12.11.2014 | | |
| Revision By | |  | | |
| Hope / Grindheim | |  | | |
| ­ | | | | |
| TFE4171 | | | | |
| Exercise I | | | | |
| ­ | | | | | |
| Author(s):  S.Grindheim  B.Hope |  | |  |  | |
| Revision Notes | * 1.0: Initial release | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
| ***Bengt Erik Hope / Svein Grindheim © 2014 Bergen, NORWAY*** | | | | | |

**🗎Page intentionally left blank**

* Preface

## Abstract

*This is a short and informal paper answering questions listed in*

*Exercise 1 in the NTNU course TFE4171* – Design of Digital Systems 2.

Contents

[1.1 Abstract i](#_Toc410005393)

[**Section 1** Lab 1 3](#_Toc410005394)

[**Section 2** Lab 2 5](#_Toc410005395)

[2.1.1 Control Hazards 8](#_Toc410005396)

[2.2 Architecture Components 9](#_Toc410005397)

[2.2.1 Program Counter 9](#_Toc410005398)

[2.2.2 32-bit Full Adder 9](#_Toc410005399)

[2.2.3 Register File 9](#_Toc410005400)

[2.2.4 Sign extender 9](#_Toc410005401)

[2.2.5 Multiplexers 10](#_Toc410005402)

[2.2.6 ALU 11](#_Toc410005403)

[2.2.7 Bit shift module 11](#_Toc410005404)

[2.3 Instruction Set & Encoding 12](#_Toc410005405)

[**Section 3** Results 13](#_Toc410005406)

[3.1 Simulation Results 16](#_Toc410005407)

[3.2 Hardware Deployment 21](#_Toc410005408)

[**Section 4** Discussion 24](#_Toc410005409)

[4.1 Design Constraints 24](#_Toc410005410)

[4.2 Possible Design Improvements 24](#_Toc410005411)

[**Section 5** Conclusion 25](#_Toc410005412)

[**Section 6** Bibliography 26](#_Toc410005413)

[ Appendices A](#_Toc410005414)

[Appendix A RTL Schematic of the MIPS Processor A](#_Toc410005415)

# Lab 1

1. Perform task 2 in the instructions. Report how you solved it.

In order to bind the design module (*dut.v*) to the property module (*dut\_property.v*) the following line was added to the test module (*test\_dut.v*):



1. Perform task 3 (no\_implication) in the instructions. Answer the two questions for the no\_implication case:

The following code snippet represents the property (*pr1*) used in b):



* Q: WHY IS THERE A FAIL -AND- A PASS AT TIME (70) ??

We notice that there is no implication operator in property pr1! We get vacuous FAILS because all we are ‘asking’ is that *req* be true at the positive clock edge and that two clks later *gnt* also must be true. We are not ‘asking’ to check the sequence only “if *req* is true at an positive edge of clk”. Hence, every clock that *req* is not true **the property FAILS**. This is also the case at time 70.

Time 70 is a special case though, because in addition to the vacuous FAIL, we get a PASS. **The property PASSES** because at time 30 *req* is HIGH so the property looks for *gnt* HIGH at 70. It does find *gnt*=HIGH at 70, so it PASSES. However, since *req*=LOW at 70, it also FAILS.

* Q: WHY ARE THERE 2 FAILs AT TIME (130) ??

At time 130 we have two FAILS because:

1. *req* is LOW (the reason for this was explained above) at the positive clock edge @ time 130
2. *gnt* is evaluated false @ time 130, even though the antecedent *req* was asserted HIGH two clock cycles earlier (time 90).
3. Perform task 4 (implication) in the instructions. Answer the two questions for the implication case:

The following code snippet represents the property (*pr1*) used in c):



* Q: WHY ARE THERE 2 PASSES AT TIME 70 ?

Now that we have an implication operator, the *antecedent (req)* must be true in order to evaluate the *consequent* (*gnt*). With this we have fixed the vacuous FAIL whenever *req* is LOW. However, we will now see a vacuous PASS when *req* is LOW instead. The reason for this is given by the LRM[[1]](#footnote-1) ‘‘*If there is no match of the antecedent sequence\_expr, then evaluation of the implication succeeds vacuously and returns true*’’.

Hence, whenever we see *req* LOW we get a vacuous pass which triggers the PASS action block, and we get the PASS display as a result.

* Q: WHY IS THERE A PASS -and- A FAIL AT TIME 130 ?

1. There is a vacuous PASS at time 130 when *req* is LOW because of the reasons discussed above.
2. There is a FAIL at time 130 because *gnt*=LOW two clock cycles after the antecedent *req* was asserted to HIGH (time 90).
3. Perform task 5 (implication\_novac) in the instructions. Do you get the expected result?

The following code snippets represents the properties (*pr1* and *pr2* respectively) used in d):

 

When running the simulator we get the expected result (no vacuous PASS/FAILS). This is because of the following lines in the property definitions:

 

Here it is used a cover statement as well as an “else” statement for the action block of the assertion. What this does is that it asks: ‘‘did you exercise this condition’’ or ‘‘did you cover this property’’. So; if an assertion never fires, it can be bacuse:

* 1. you don’t have any bug
  2. you never exercised the condition in the first place

Regarding the cover statement: If the condition gets exercised but does not fail, we get that indication through the PASS action block we wrote with the cover statement.

# Lab 2

1. Perform task 2 (overlap) in the instructions. Answer the following questions for the overlap case:

The code snippet below illustrates the property *pr1* with its sequence *sr1*. The property is utilized by the cover (*creqGnt*) and assert (*reqGnt*) statements in a).



* Q: WHY DOES THE PROPERTY FAIL at 30?

The assert property FAILS at time 30 because *cstart* is HIGH while *req* is LOW. As defined by the sequence *sr1*: *req* needs to be HIGH when *cstart* is asserted HIGH @ positive clock edge.

* Q: WHY DOES THE PROPERTY PASS at 90?

The cover property passes because the requirements set by *sr1* is fulfilled. That is; *cstart* and *req* is set HIGH two clock pulses before (time 50) *gnt* goes HIGH.

* Q: WHY DOES THE PROPERTY PASS at 150?

The cover property passes because the definitions in *sr1* is met: *gnt* goes HIGH two clock pulses after *cstart* and *req* was assertet HIGH (time 110).

* Q: WHY DOES THE PROPERTY FAIL at 170?

The assert property fails at time 170 since *req* is asserted HIGH when *cstart* is LOW.

Also, two clock cycles ago (time 130) *cstart* and *req* was sampled HIGH at the positive edge of clock. Because the antecedent (*cstart*) is held high over several clock pulses (time 110 through 150) the consequent will always be reevaluated at each clock pulse; we must ask our self if this is something we really want? Maybe it would be more sensible to evaluate the antecedent with a $rose(…) or $fell(…) statement.

* Q: WHY DOES THE PROPERTY FAIL at 190?

The assert property fails because *gnt* remains unchanged even though the antecedent was HIGH two clock pulses ago (time 150). This issue was also discussed above…

1. Perform task 3 (non\_overlap) in the instructions. Answer the following questions for the non\_overlap case:

The following code snippet represents the property (*pr1*) used in b):



* Q: WHY DOES THE PROPERTY FAIL at 70?

The antecedent, *cstart*, is asserted HIGH at 50 and the consequent is evaluated. *req* is false at the subsequent rising clock edge(70) and the assertion is fired.

* Q: WHY DOES THE PROPERTY PASS at 90?

*cstart* is asserted HIGH at 30 and the consequent is evaluated. The cover property passes at time 90 as *req* and *gnt* are true at 50 and 90 respectively as specified in the property *pr1*.

* Q: WHY DOES THE PROPERTY FAIL at 170?

*cstart* is asserted HIGH at time 110 and the consequent is evaluated. The assertion property fails at 170 as *req* is true at 130 but *gnt* is false at 170.

* Q: WHY DOES THE PROPERTY FAIL at 190?

*cstart* is asserted HIGH at 130 and the consequent is evaluated. The assertion property fails at 190 as *req* is true at 150 but *gnt* is false at 190.

* Q: WHY DOES THE PROPERTY PASS at 210?

*cstart* is asserted HIGH at 30 and the consequent is evaluated. The cover property passes at 90 as *req* and *gnt* are true at 50 and 90 respectively as specified in the property *pr1*.

1. **L**anguage **R**eference **M**anual [↑](#footnote-ref-1)