|  |
| --- |
| https://www.pointer.de/hs_bild.php?iid=208&lid=477 |
| Projektaufgabe 3 |
| Dokumentation zu „Testbench“ |
|  |
|  |
|  |



|  |
| --- |
|  |





21.12.2017

Jeremy Buchert, Samuel Daurat, Arthur Dischli, Xavier Voltzenlogel

Elektrotechnik und Informationstechnik 3nat

Semester 7

Inhaltsverzeichnis

[1 Einführung 2](#_Toc501634056)

[2 Aufgabenstellung 3](#_Toc501634057)

[2.1 Bedienung 3](#_Toc501634058)

[2.2 Programmablauf / Programmstruktur 4](#_Toc501634059)

[3 Entwurf der Schaltung 5](#_Toc501634060)

[3.1 Gesamtsystem 5](#_Toc501634061)

[3.2 Teilsysteme 5](#_Toc501634062)

[3.2.1 DetectFallingEdge 5](#_Toc501634063)

[3.2.2 Clock Divider 6](#_Toc501634064)

[3.2.3 Buzzer 6](#_Toc501634065)

[3.2.4 ConvertIntToBCD 6](#_Toc501634066)

[3.2.5 StateMachine 7](#_Toc501634067)

[3.2.6 Counter 8](#_Toc501634068)

[4 Simulation 9](#_Toc501634069)

[4.1 Simulation der Teilsysteme 9](#_Toc501634070)

[4.1.1 Detect Falling Edge 9](#_Toc501634071)

[4.1.2 Clock divider 9](#_Toc501634072)

[4.1.3 Buzzer 9](#_Toc501634073)

[4.1.4 Convert Int to 7 Segment 9](#_Toc501634074)

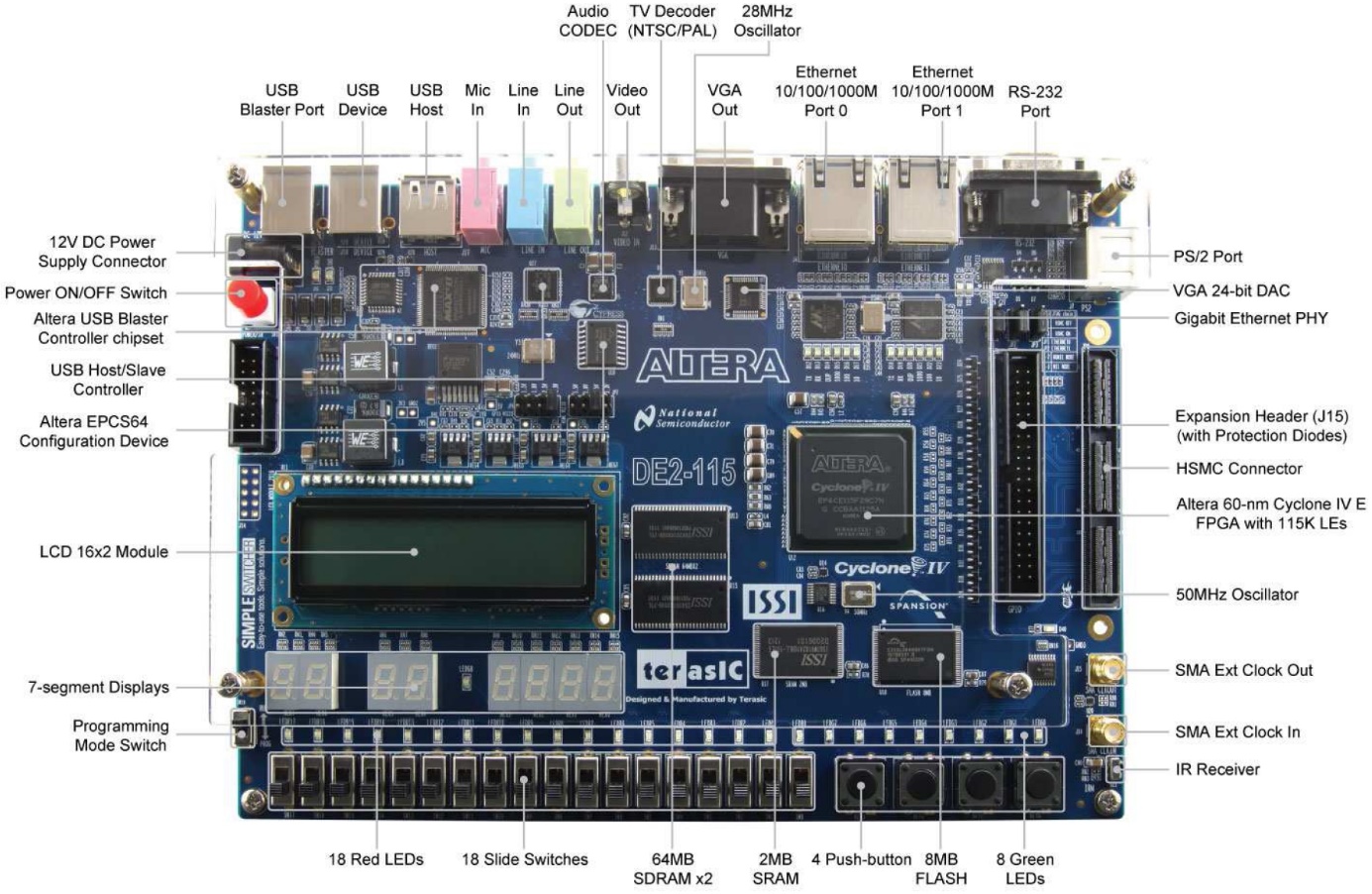
[4.1.5 State Machine 10](#_Toc501634075)

[4.1.6 Counter 10](#_Toc501634076)

[5 Inbetriebnahme Gesamtsystem / Fazit 11](#_Toc501634077)

# 1 Einführung

In diesem Labor geht es darum, bereits in VHDL erstellte Module mit einer Testbench auf ihre korrekte Funktion zu testen.



# 2 Aufgabenstellung

Im ersten Teil sollen vier Testbenches geschrieben werden, welche unterschiedliche Module aus den Projektaufgaben 1 & 2 testen.

In einem zweiten Teil soll schließlich eine große Testbench geschrieben werden, welche den kompletten Timer aus Projektaufgabe 2 testet.

# 3 Entwurf der Testbenches

## 3.1 BCD-Decoder

Die erste Testbench wird den BCD zu 7-Segment Decoder auf seine funktionsweise prüfen.

Dadurch dass der Code des Decoders rein kombinatorisch arbeitet, muss man in dieser Testbench nicht mit einer Clock arbeiten, sondern kann alles mit „Wait for xx time“ programmieren.

Ablaufen wird der Test folgendermaßen:

# 

Das Ergebnis der Simulation wird mithilfe von assert Befehlen im Simulationsfenster angezeigt.

Hier einmal ein Beispiel bei welchem der BCD-Decoder einwandfrei funktioniert:

run 500ns

# \*\* Note: Number 0: passed

# \*\* Note: Number 1: passed

# \*\* Note: Number 2: passed

# \*\* Note: Number 3: passed

# \*\* Note: Number 4: passed

# \*\* Note: Number 5: passed

# \*\* Note: Number 6: passed

# \*\* Note: Number 7: passed

# \*\* Note: Number 8: passed

# \*\* Note: Number 9: passed

# \*\* Note: DONE!

# Time: 320 ns Iteration: 0 Instance: /testbench\_decoder

Und hier ein Beispiel, bei welchem der BCD-Decoder ein Fehler aufweist:

Man sieht hier gut, dass im Fehlerfall eine erweiterte Analyse ausgeführt wird, so dass der Benutzer herausfinden kann welches Digit an der 7-Seg-Anzeige denn genau falsch angesteuert wird.

\*\* Note: Number 0: passed

# \*\* Error: Number 1 : FAILED

# \*\* Error: Position 0: FAILED

# \*\* Note: Position 1: passed

# \*\* Note: Position 2: passed

# \*\* Note: Position 3: passed

# \*\* Note: Position 4: passed

# \*\* Note: Position 5: passed

# \*\* Note: Position 6: passed

# \*\* Note: Number 2: passed

# \*\* Note: Number 3: passed

# \*\* Note: Number 4: passed

# \*\* Note: Number 5: passed

# \*\* Note: Number 6: passed

# \*\* Note: Number 7: passed

# \*\* Note: Number 8: passed

# \*\* Error: Number 9 : FAILED

# \*\* Error: Position 0: FAILED

# \*\* Note: Position 1: passed

# \*\* Note: Position 2: passed

# \*\* Note: Position 3: passed

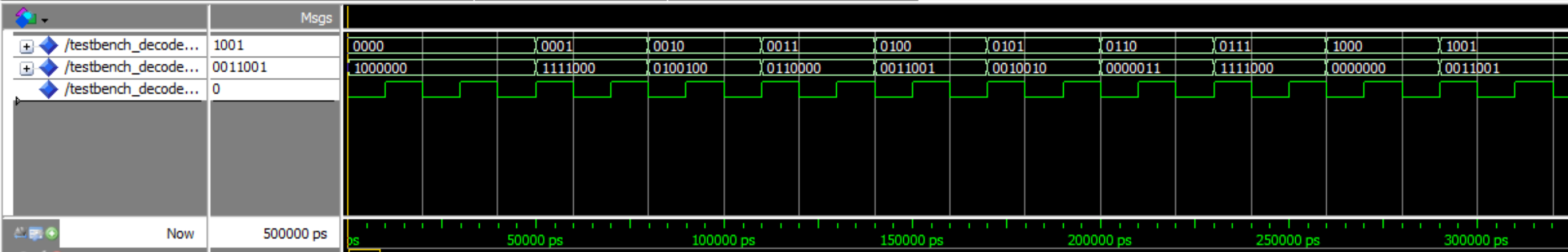
# \*\* Note: Position 4: passed

# \*\* Note: Position 5: passed

# \*\* Note: Position 6: passed

# \*\* Note: DONE!

Weiterhin kann man zur manuellen Analyse auch das Wave-Fenster benutzen, welches hier im Fehlerfall so aussieht:



# 4 Simulation

Mithilfe von ModelSim wurde jedes Einzelsystem simuliert um es auf seine Funktionsfähigkeit zu testen, bevor das System als komplette Einheit in Betrieb genommen wird.

# 5 Inbetriebnahme Gesamtsystem / Fazit

Zuletzt wurde das Gesamtsystem in Betrieb genommen und ausführlich an der Hardware getestet.

Ein großes Problem hierbei war, dass die Statemachine zufällige Ausgangskombinationen aktivierte, die laut dem Programmcode eigentlich gar nicht existieren dürften. Nach mehreren neu Programmierten Statemachines und vielen neuen Simulationen und Hardwareimplantierungen wurde schließlich der Grund des Problemes gefunden:

Nach Einsatz einer Statemachine, wie sie in Quartus als Codebeispiel existiert (statt drei Prozessen mit next\_mode nur zwei Prozesse, bei welchem mode direkt geändert wird) funktionierte die Statemachine normal.

Ein paar kleinere Bugs später, war das System voll einsatzbereit und konnte in der Hardware auf seine Zuverlässigkeit getestet werden.