**Ausgabe der Aufgabe: 27. September 2017**  **Eingabefrist: 23. Oktober 2017, 24:00**

**Hausaufgabe 1 kann bis zum Ende der Nachholungswoche (15. Dezember 2017, 24:00) mit der Einzahlung des Extraprozessengebührs eingegeben werden**

Die Aufgabe wird in elektronischer Form, in PDF Format durch Hausaufgabeneingabeportal <https://hf.mit.bme.hu/> eingegeben.

Registration ist erforderlich! Erreichbare Punkte: max. 15, IMSC Punktzahl: 5

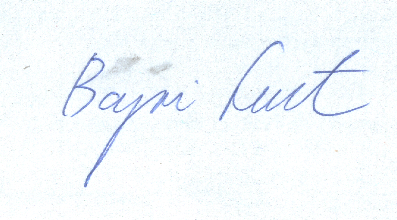
Persönliche Digit-Kodes sind am Ende dieses Dokuments verfügbar!

**Digitaltechnik Hausaufgabe 1**

**Entwurf sequentieller Schaltnetze**

|  |  |  |
| --- | --- | --- |
| **Name:**  Bajczi Levente | **NEPTUN:**  XAO5ER | **Email:**  levente.bajczi@edu.bme.hu |
| **Lehrkreis:**  Deutschkurs | **Jahrgang:**  2017 | **DIGITKODE:**  2745136 |

Die Aufgaben wurden selbständig, ohne unerlaubtes Hilfsmittel und ohne unmittelbare Mitwirkung der Anderen gelöst

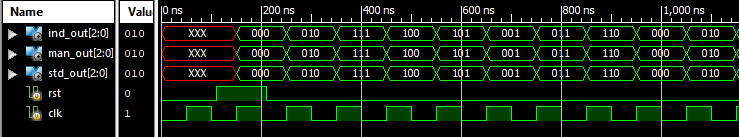
(A feladatokat önállóan, meg nem engedett segédeszközök használata és mások közvetlen közreműködése nélkül oldottam meg:)

……………………………………………………

Unterschrift (Aláírás)

**Zusammenfassung der Ergebnisse**

Die drei Arten von endlichen Zustandsautomaten zahlen nach der Reihenfolge, die durch meinen Digitkode angegeben wurde. Das gemeine Simulationsdiagramm sieht folgenderweise aus:



Anhand der Ergebnisse ist es klar, dass die Hausaufgabe gut gelöst wurde, weil die drei Ausgänge schauen die richtige Werte, was ich (anhand meines persönlichen Digit-Kodes) erwartete. Die drei verschiedene Weise, wodurch die Aufgabe gelöst wurde, waren alle sehr interessant zu entwickeln, aber am meisten gefällt mir die letzte, weil eine durch einer Binärzähler durchgeführte Lösung mich andernfalls nicht eingefallen hätte.

## HA1\_1

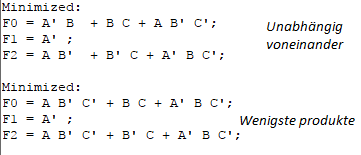
Die erste Weise war um eine Zustandsübergangstabelle orientiert. Meine Digit-Kode ergibt die folgende Tabelle:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aktuell | Folgezustand |  | Aktuell | Folgezustand |
| 0 | **2** |  | **0 – 000** | **2 – 010** |
| 2 | **7** |  | **1 – 001** | **3 – 011** |
| 7 | **4** |  | **2 – 010** | **7 – 111** |
| 4 | **5** |  | **3 – 011** | **6 – 110** |
| 5 | **1** | **4 – 100** | **5 – 101** |
| 1 | **3** |  | **5 – 101** | **1 – 001** |
| 3 | **6** |  | **6 – 110** | **0 – 000** |
| 6 | **0** |  | **7 – 111** | **4 – 100** |

Daraus die Karnaugh-Tabellen sind (Seien a, b, und c der dritten, zweiten und ersten Ziffern des aktuellen Zustandes in Binärform):

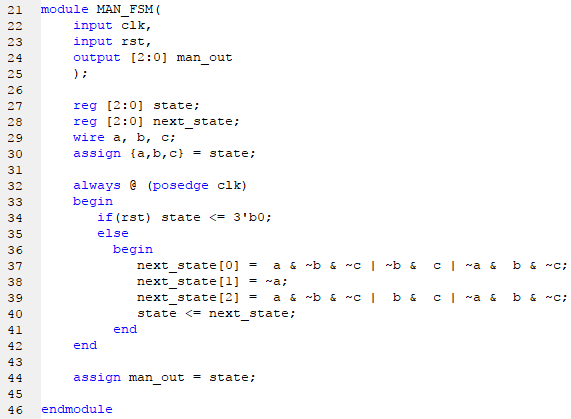
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Folgezustand[0]:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 00 | 01 | 11 | 10 | | 0 | 0 | 1 | 0 | 1 | | 1 | 1 | 1 | 0 | 0 | | Folgezustand[1]:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 00 | 01 | 11 | 10 | | 0 | 1 | 1 | 1 | 1 | | 1 | 0 | 0 | 0 | 0 | | Folgezustand[2]:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 00 | 01 | 11 | 10 | | 0 | 0 | 0 | 1 | 1 | | 1 | 1 | 0 | 1 | 0 | |

Die Ausgänge anhand dieser Tabellen sind:

Nach Minimisierung (zuerst durch die Einkreuzung von Primimplikanten):

Für wenigsten *product-terms* getestete Ausgänge, mithilfe von **Logic Friday**:

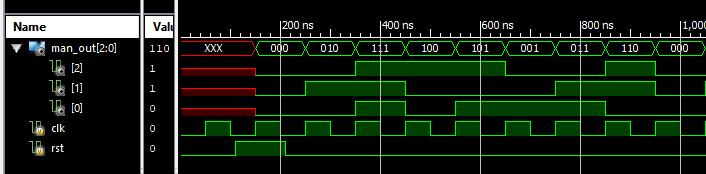
Dieses Ergebnis zeigt, dass zwei Elemente können mehr als einmal verwendet werden (A\*\B\*\C und \A\*B\*\C). Diese Erscheinung bedeutet eine Verkleinerung in der Anzahl der nötigen Elemente in diesem Netzwerk.

Mit diesen logischen Ausdrücken habe ich die Module entwickelt, die bei jeder steigenden Taktflanke den nächsten Zustand (für jedes Bit) anhand des aktuellen Zustandes berechnet. Diese Implementation folgt den Moore-Modell, weil die Ausgänge nur von dem aktuellen Zustand abhängen.

In der Implementation bei jeder steigenden Taktflanke wird jedes Bit anhand dieser Formeln berechnet (oder bei RESET wird jedes Bit nulliert), dann zu den Ausgängen ausgeführt.

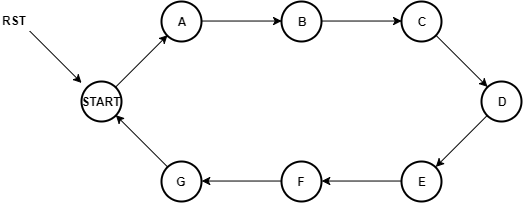
Die Code meiner Implementation ist hier zu sehen:

Das Ergebnis der Simulation:



## HA1\_2

Die zweite Weise wurde durch dem Allgemeinen FSM-Planungsmethodik durchführbar (mit den Schritten angeschaut):

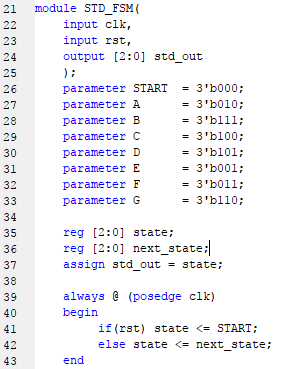
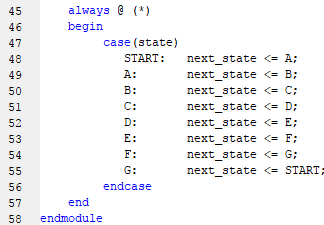
1. Die Zustände definieren:

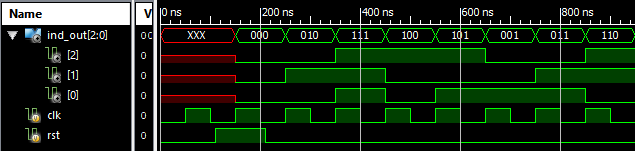
|  |  |
| --- | --- |
| START | 0 – 000 |
| A | 2 – 010 |
| B | 7 – 111 |
| C | 4 – 100 |
| D | 5 – 101 |
| E | 1 – 001 |
| F | 3 – 011 |
| G | 6 – 110 |

1. Entscheidung darüber, dass ich den **Moore-Modell** brauche (ähnlich zu HA1\_1, nur der aktuelle Zustand soll die Ausgänge beeinflussen).
2. Der Zustandsdiagram zeigt, wie diese Zustände einander folgen sollten.

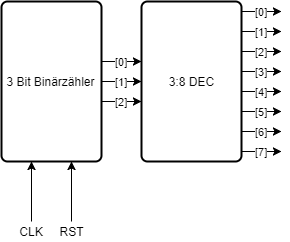
Mithilfe von diesen Punkten die Entwicklung des Modules war eigentlich einfach: für jeden Zustand definierte ich einen Parameter, dessen Name START für 0, und A-G für die Reste war (Sehen Sie die Tabelle oben für die Zuordnungen).

Bei jeder steigenden Taktflanke wird der Zustand aktualisiert mit dem Wert von dem nächsten Zustand, der bei anhand der Zustandsdiagram „berechnet“ wird (bei RESET wird START als Zustand ausgegeben).

Die Code meiner Implementation ist hier zu sehen:

Das Ergebnis der Simulation:

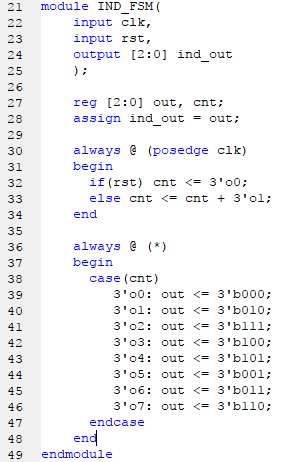
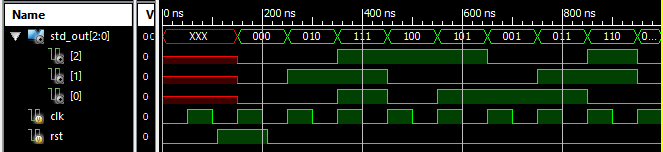
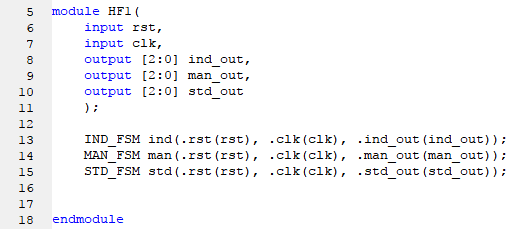
## HA1\_3

Die dritte Weise ähnelt die zweite darin, dass die Zustände nach ihren „Indizes“ behandelt werden, aber anstatt Parametern (START, A-G) benutzen wir jetzt einen 3:8 Dekoder in einem Busmultiplexer, anhand dessen und eines Binärzählers (der die Eingänge von dem Dekoder generiert) den Wert von dem nächsten Index berechnet und zu den Ausgängen ausgeführt wird. (Zum Beispiel, wenn der Binärzähler den Wert **1012** nach den Dekoder „schickt“, wählt die Dekoder den **1012**-ten (5-ten) Index aus, und deshalb wird der aktuelle Zustand **1** als Wert haben):

*Diese 8 Ausgänge definieren welche 3-Bit Vektor sollte ausgewählt werden (als Indizen).*

Die Implementation dieses Modules folgt diese Logik Schritt für Schritt: ein a*lways*-Block zählt in 3 Bits von 0 bis 7 bei jeder steigenden Taktflanke (dann wegen Überlauf wieder von 0) und ein anderer ist der Busmultiplexer, dessen Eingänge die Ausgänge des Binärzählers sind und als Ausgang gibt den Wert des aktuellen Zustands aus (mit einer case-Statement in Verilog durchgeführt). RST nulliert die Zahl in dem Binärzähler.

Die Code meiner Implementation ist hier zu sehen:



Der Topmodule, die diese Einheiten zusammenhielt, instanziiert jede Submodule, damit sie funktionellen können. Die Code dafür:

Das Ergebnis der Simulation: