Übung 7 Rechnerstrukturen

Aufgabe 7.1:

X1	X2	X3	X4		F1	F2	F3	F4
			1					
		1						
		1	1					
	1							
	1		1					1
	1	1					1	
	1	1	1				1	1
1								
1			1				1	
1		1				1		
1		1	1			1	1	
1	1							
1	1		1				1	1
1	1	1				1	1	
1	1	1	1		1			1

X1-X4 sind alle möglichen Binär-Werte.

F1-F2 ist das Ergebnis der Multiplikation von (X1X2) und (X3X4).
Dabei funktioniert die Multimpliation folgendermaßen:

Wenn bei X2 eine 1 steht, wird X3-X4 an sie Stellen F3-F4 geschrieben.
Wenn bei X1 eine 1 steht, wird X3-X4 an sie Stellen F2-F3 geschrieben.
Steht an beiden Stellen eine 1, wird X3-X4 and die Stellen F2-F3 *und* F3-F4 geschrieben und addiert.

Unsere PLA -Matrix wird nun folgendermaßen gebildet:
Wir „kippen“ das Diagramm nach rechts. Im nun oberen (bisher linken) Bereich ersetzen wir alle 1 durch 2 und alle 0 durch 3:

0 ↓ →
1 ↓ ↓→ OR
2 ↓→ AND →
3 ↓→ NAND →

Schicken wir nämlich jetzt einen Binärcode (z.B. 1100) rein, geht genau dann eine 1 nach unten, wenn die passende „Maske“ (in diesem Fall 2233) gefunden wird.

Im unteren Bereich werden diese Einsen genau dann nach rechts weitergereicht, wenn in der betreffenden Zeile eine Eins steht. So bekommen wir also rechts die Ausgabe der korrekt multiplizierten Zahl (in 4 Bits.)

Alle Fälle, bei denen die Multiplikation 0 ergibt, brauchen gar nicht aufgeführt zu werden, da die Ausgabe eh 0 ergibt, wenn keine passende „Maske“ in der Eingabe gefunden wird.

b) nun suchen wir noch passende **Karnaugh-Diagramme**.

Hierzu malen wir für jede Funktion ein eigenes Diagramm. Wir schauen in unserer Tabelle oben, wo bei der entsprechenden Funktion eine 1 (rechts!!) stehen haben, und tragen dann an der Stelle eine 1 ins Karnaugh-Diagramm, die links durch x1x2x3x4 gegeben ist:

F1

x3x4 \ x1x2	00	01	11	10
00				
01				
11			1	
10				

F2

x3x4 \ x1x2	00	01	11	10
00				
01				
11				1
10			1	1

F3

x3x4 \ x1x2	00	01	11	10
00				
01			1	1
11		1		1
10		1	1	

F4

x3x4 \ x1x2	00	01	11	10
00				
01		1	1	
11		1	1	
10				

Wir erhalten nun folgende Minterme: (wir schauen einfach, dass wir alle 1en bekommen):
Hierbei achten wir diesmal nicht darauf, möglichst kurze Termen zu bekommen, sondern möglichst wenig verschiedene: (da jeder Term ja eine Spalte im PLA sein muss):

F1 = x1x2x3x4
F2 = **x1x2x3x4** + x1x3x4
F3 = x1x3x4 + x2x3x4 + **x1x2x3x4** + x1x2x3x4
F4 = x2x4

Wir brauchen also 7 verschiedene Spalten für das PLA.
Für jede 1 steht nun eine 2, für jede 0 (NOT) eine 3, und wo es egal ist steht eine 0.

In der ODER-Ebene (unten) werden diese Ausgaben nun kombiniert, und wir erhalten das Ergebnis.
BEACHTE: Dabei werden nicht nur im Fall **2-3-2-2** zwei F's ausgegeben, sondern auch bei anderen Zahlen, da hier auch Spalten mit Nullen „ansprechen" (Nullen sind ja praktisch *, also alles ist erlaubt.)
BEISPIEL: auf 3-2-2-2 spricht die vorletzte *und* die letzte Stelle an.

	1	1	1	1	1	1	1		
X1	2	**2**	2	2	0	3	0		
X2	2	**3**	0	0	2	2	2		
X3	2	**2**	2	3	2	2	0		
X4	2	**2**	3	2	3	2	2		
	1								F1
		1	1						F2
		1		1	1	1			F3
							1		F4

Aufgabe 7.2

Zuerst suchen wir alle möglichen Kombinationen von Kanten, bei denen die Bedingung „Dreieck“ erfüllt ist:

K0	K1	K2	K3	K4	K5
1	1			1	
		1	1	1	
1			1		1
	1	1			1

Da wir jeweils noch eine weitere beliebige Kante hinzunemen dürfen, fügen wir diese noch jeweils ein:

K0	K1	K2	K3	K4	K5
1	1	1		1	
1	1		1	1	
1	1			1	1
1		1	1	1	
	1	1	1	1	
		1	1	1	1
1	1		1		1
1		1	1		1
1			1	1	1
1	1	1			1
	1	1	1		1
	1	1		1	1

Nun haben wir schon fast unsere **PLA.Matrix**: nur noch 0→3 und 1→2 , und fertig! ☺

Das ganze umkippen, unten eine Zeile mit 1-en dran, und wir bekommen rechts immer eine 1 raus, wenn eine der Spalten „passt“.

Aufgabe 7.3: Wolf-Kohlkopf-Ziege-Problem

Zuerst erstellen wir eine Wertetabelle:

1.) EINGABE: Nummer der Aktion, binär codiert (1 bis 7)
2.) AUSGABE: **Was macht er?** (hin oder zurück und was nimmt er mit)
3.) AUSGABE: Nummer der nächsten Aktion, also 1.) + 1

HIN	**ZURÜCK**	**Codiert nach Tabelle**	**Zustand Nummer**
→ Ziege		**0 10**	1
	← leer	**1 00**	2
→ Wolf		**0 11**	3
	← Ziege	**1 10**	4
→ Kohlkopf		**0 01**	5
	← leer	**1 00**	6
→ Ziege		**0 10**	7

Dann haben wir ein Diagramm:

Oben: 1 1 1 1 1	000	**010**	001
	001	**100**	010
	010	**011**	011
	011	**110**	100
	100	**001**	101
	101	**100**	110
	110	**010**	111
	Links: Input		

In der **ersten Spalte** stehen die Nummern der Zustände, von 1 bis 7.
In der **dritten Spalte** steht die Folgenummer, also jeweils +1.

In der **mittleren Spalte** steht die nächste Aktion (hin oder zurück, und mit was.)

Für das **PLA** lassen wir dieses Diagramm einfach nach rechts auf die Seite kippen.
In der ersten Spalte (bzw. oberen Zeile jetzt) müssen wir alle Nullen durch „3“ und Einsen durch „2“ ersetzen.

WIESO? → von **oben** kommen ja immer 1en, von **links** kommt unsere Eingabe (z.B. 110): es kommt in dieser Spalte dann genau dann unten eine 1 raus, wenn für jede 1 eine 2 und für jede 0 eine 3 steht, da

- „2“ die 1 von oben per AND mit dem Signal von links verknüpft (gibt ja 1 wenn beides 1), und
- „3“ die Signale von oben und links per NAND verknüpft
  (gibt ja genau 1, wenn ein Signal (das von oben) 1 ist, und das andere (das von links) 0.

Die neue Zahl wird nun per Delay an die **alte Zahl** gegeben, wo dann genau *diejenige* Spalte eine 1 nach unten gibt, die auf die Eingabe (Index, z.B. 011) passt.

Aufgabe 7.4:

	AscII	Voraussetzung Zustand	Neuer Zustand	
B	01100010	**00	1000	
O	11101111	1000	0100	Z1 = b war schon da
M	01101101	0100	0010	Z2 = bo ...
B	01100010	0010	0001	Z3 = bom ...
*	********	0001	0001	Z4 = ALARM!!!

Für das PLA wird das ganze jetzt wieder „aufgestellt“, oben kommen 1en rein, links oben unser ASCII-Code und links unten der zurückgeführte „neue Zustand“.

Übung 8

Aufgabe 8.1

a) jedes beliebige Programm ausführbar
b) Daten und Code im gleichen Speicher
c) Speicher, CPU (Rechenwert und Steierwerk), I/O
d) Datenprozessor:
ALU (Rechenwert),
Akku A;
Linkreguster L,
Multiplikatoregister MR (für 2. Multiplikator, 1. im Add-Akku),
MBR (Memory Buffer Register), (Bindeglied zwischen RAM und Akku)
e) Befehlsprozessor:
Befehlsregister (aktueller Befehl),
Speicheraddressregister (Adresse des nächsten anzusprechenden Speicherplatzes),
PC (nächster Befehl),
Decodierer (entschüsselt Befehl),
Steuerwerk
f) Unterscheidung Daten - / Adressbus: MUSS nicht sein, aber:
- unterschiedliche BIT-Zahl,
- parallel Adresse und Daten übetragbar, so also schneller.
g) SISD: Single Instruction Single Data.
h) Maschinencode, Assemblersprachen (1:1-Übersetzung)
, ohne Parameter: kein-Adress-Befehl. (??) Mehr-Adress-Befehle: 2 Parameter.
i) 2-Phasen-Konzept: 1. Holen/Decodieren, 2. Ausführen. Beide Phasen können parallel ausgeführt werden. Problem gelöst: zwischen Befehlen und Daten unterschieden. (vgl. Vorlesung: 4 versch. Phasen)
j) Bus zwischen CPU und Speicher langsamer als Berechnungen der CPU möglich.