yunfeng.huang [☺] tu-dortmund.de
mikail.yayla [☺] tu-dortmund.de

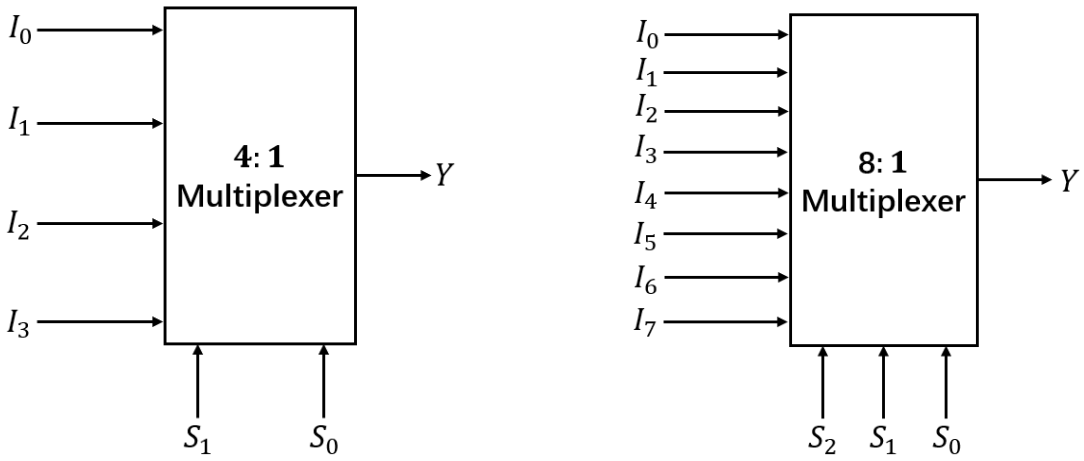
Hardwarepraktikum
Wintersemester 2020

Übungsblatt 4

Bearbeitung ab Samstag, 5. Dezember 2020

4.1 Multiplexer in VHDL

Die Ausgänge von kombinatorischen Schaltungen werden nur durch ihre Eingangssignale bestimmt. Sie arbeiten also *ohne Gedächtnis*. Zudem sind zyklusfrei, wodurch Änderungen an den Eingangssignalen sofort (abgesehen von Schaltverzögerung) am Ausgang wirksam werden. Sogar die komplexesten Systeme sind aus vielen einfachen kombinatorischen Schaltungen gebaut. Einer der häufigsten solcher Bausteine ist der *Multiplexer*, der dazu genutzt wird um aus mehreren Eingabesignalen ein Ausgabesignal anhand eines Auswahlsignals zu wählen. Beispielsweise repräsentiert Abbildung 1(a) einen 4:1 Multiplexer, der das Ausgabesignal Y dadurch bestimmt, dass er einen der Eingänge I_0, I_1, I_2 und I_3, anhand der Auswahlsignale S_1 und S_0, auswählt. Ebenso wählt der 8:1 Multiplexer, der in Abbildung 1(b) gezeigt ist, ein Signal der Eingänge I_0 bis I_7, durch die Auswahlsignale S_2, S_1 und S_0, als Ausgabesignal Y aus.



(a) 4:1 Multiplexer (b) 8:1 Multiplexer

Abbildung 1: Zwei Multiplexer.

Aufgaben:
Multiplexer können auch aus Gruppen von Eingangssignalen der Länge n auswählen. In dieser Aufgabe soll ein 4:1 Multiplexer implementiert werden, der vier 3-Bit Eingangssignale besitzt. In der Wahrheitstabelle 1 wurden die Eingänge dieses Multiplexers auf I_0(000), I_1(001), I_2(010) und I_3(011) gesetzt. Implementiert diesen 4:1 Multiplexer gemäß der Wahrheitstabelle 1, sodass nur genau eines der Signale I_0, I_1, I_2 und I_3 gleichzeitig ausgewählt werden kann. Dies soll anhand der Auswahlsignale S_1S_0 (zwei Bits) geschehen.

a. (2 Punkte) Verwendet eure logischen Gatter (AND, OR, NOT), die ihr in Übung 2 erstellt habt, um diesen 4:1 Multiplexer zu entwerfen. Eure Gatter könnt ihr dabei verändern (z.B. die Anzahl der Inputs erhöhen). Zeichnet erst das **Schaltbild** des 4:1 Multiplexers. Implementiert danach den Multiplexer in VHDL.

b. (1 Punkte) Schreibt eine Testbench für eure Implementierung, in welcher die Fälle S_1S_0 gleich 00, 10, 01 und 11 überprüft werden.

Auswahlsignal		Ausgabe
S_1	S_0	Y
0	0	I_0(000)
0	1	I_1(001)
1	0	I_2(010)
1	1	I_3(011)

Tabelle 1: Die Wahrheitstabelle eines 4:1 Mux.

In den weiteren Aufgabenteilen geht es um einen 8:1 Multiplexer 1(b), mit 3-bit Eingangssignalen I_0(000), I_1(001), I_2(010), I_3(011), I_4(100), I_5(101), I_6(110) und I_7(111). Implementiert diesen 8:1 Multiplexer anhand der Wahrheitstabelle 2, sodass genau eines der 3-Bit Signale von I_1 bis I_7 ausgewählt werden kann. Dies soll anhand der Auswahlsignale $S_2S_1S_0$ (drei Bits) geschehen.

Auswahlsignal			Ausgabe
S_2	S_1	S_0	Y
0	0	0	I_0(000)
0	0	1	I_1(001)
0	1	0	I_2(010)
0	1	1	I_3(011)
1	0	0	I_4(100)
1	0	1	I_5(101)
1	1	0	I_6(110)
1	1	1	I_7(111)

Tabelle 2: Die Wahrheitstabelle eines 8:1 Mux.

c. (2 Punkte) Verwendet zwei der 4:1 Multiplexer, die ihr im vorherigen Aufgabenteil entworfen habt um diesen 8:1 Multiplexers zu entwerfen. Eure Gatter könnt ihr dabei verändern (z.B. die Anzahl der Inputs erhöhen). Zeichnet erst das **Schaltbild** des 8:1 Multiplexers. Implementiert danach den 8:1 Multiplexer in VHDL.

d. (1 Punkte) Schreibt eine Testbench und testet eure Implementierung mit $S_2S_1S_0$ jeweils gleich 000, 010, 001, 011, 100, 110, 101 und 111.

e. (1 Punkte) Minimiert die folgende Funktion (Hinweis: Idempotenz),

$$Y = (A \wedge B \wedge C) \vee (A \wedge B \wedge \overline{C}) \vee (A \wedge \overline{B} \wedge C)$$

f. (2 Punkte) Überlegt ob diese Funktion mit nur eurem 8:1 Multiplexer implementierbar ist. Wenn es möglich ist, zeichnet das Schaltbild eures Entwurfs und Implementiert es in VHDL.

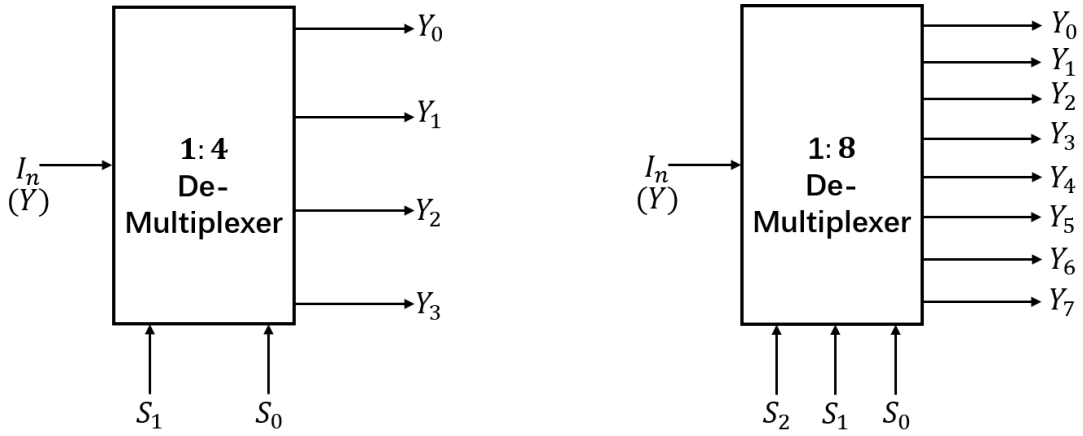
g. (1 Punkte) Testet eure Implementierung mit ABC jeweils gleich 000, 111, 110, 101 in einer Testbench.

h. (2 Punkte) Überlegt ob diese Funktion mit nur eurem 4:1 Multiplexer implementierbar ist. Wenn es möglich ist, zeichnet das Schaltbild eures Entwurfs und Implementiert es in VHDL.

i. (1 Punkte) Testet eure Implementation mit ABC jeweils gleich 000, 111, 110, 101.

4.2 Demultiplexer in VHDL

Der Demultiplexer erfüllt die umgekehrte Aufgabe des Multiplexers. Während der Multiplexer dazu benutzt wird, eine Ausgabe aus vielen Eingaben auszuwählen, leitet der Demultiplexer ein einziges Eingabesignal auf eine von mehreren Ausgaben weiter.
Beispielsweise repräsentiert Abbildung 2(a) einen 1:4 Demultiplexer, dessen Eingabesignal I_n eigentlich das Ausgangssignal Y des 4:1 Multiplexers aus Abbildung 1(a) ist. Die Aufgabe dieses 1:4 Demultiplexers ist es, die Eingabe I_n auf den korrekten Ausgang (Y_0, Y_1, Y_2 oder Y_3), je nach Auswahlsignal S_1S_0, weiterzuleiten. Ebenso ist es die Aufgabe des 1:8 Demultiplexers die Eingabe I_n, aufgrund der Auswahlsignale S_2, S_1 und S_0, auf den korrekten Ausgang (Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6 oder Y_7) weiterzuleiten.



(a) 1:4 Demultiplexer (b) 1:8 Demultiplexer

Abbildung 2: Zwei Demultiplexer.

Aufgaben:
Nehmt an, dass das Eingabesignal I_n des 1:4 Demultiplexers in Abbildung 2(a) ein 3-Bit Signal ist. Implementiert einen 1:4 Demultiplexer anhand der Wahrheitstabelle 3, sodass I_n, aufgrund der Auswahlsignale S_1 und S_0, auf den richtigen Ausgang weitergeleitet wird.

Eingabe	Auswahlsignal		Ausgabe			
I_n	S_1	S_0	Y_0	Y_1	Y_2	Y_3
I_n	0	0	I_n	0	0	0
I_n	0	1	0	I_n	0	0
I_n	1	0	0	0	I_n	0
I_n	1	1	0	0	0	I_n

Tabelle 3: Die Wahrheitstabelle eines 1:4 Demux.

a. (2 Punkte) Verwendet die logischen Gatter (AND, OR und NOT), die ihr in Übung 2 erstellt habt, um diesen 1:4 Demultiplexer zu entwerfen. Eure Gatter könnt ihr dabei verändern (z.B. die Anzahl der Inputs erhöhen). Zeichnet erst das **Schaltbild** des 1:4 Demultiplexers, den ihr entwerfen wollt. Implementiert den 1:4 Demultiplexer danach in VHDL.

b. (1 Punkte) Testet eure Implementierung in einer Testbench mit S_1S_0 jeweils gleich 00, 10, 01 und 11.

Nehmt nun an, dass das Eingabesignal I_n des 1:8 Demultiplexers in Abbildung 2(b) ein 3-Bit Signal ist. Implementiert diesen 1:8 Demultiplexer anhand der Wahrheitstabelle 4, sodass I_n, aufgrund der Auswahlsignale S_2, S_1 und S_0, auf den richtigen Ausgang weitergeleitet wird.

c. (2 Punkte) Verwendet zwei der 1:4 Demultiplexer, die ihr im vorherigen Auftrag erstellt habt, um diesen 1:8 Demultiplexer zu entwerfen. Zeichnet erst das **Schaltbild** des 1:8 Demultiplexers den ihr implementieren wollt. Implementiert den 1:8 Demultiplexer danach in VHDL.

d. (1 Punkte) Testet eure Implementierung in einer Testbench mit $S_2S_1S_0$ jeweils gleich 000, 010, 001, 011, 100, 110, 101 und 111.

Eingabe	Auswahlsignal			Ausgabe							
I_n	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
I_n	0	0	0	I_n	0	0	0	0	0	0	0
I_n	0	0	1	0	I_n	0	0	0	0	0	0
I_n	0	1	0	0	0	I_n	0	0	0	0	0
I_n	0	1	1	0	0	0	I_n	0	0	0	0
I_n	1	0	0	0	0	0	0	I_n	0	0	0
I_n	1	0	1	0	0	0	0	0	I_n	0	0
I_n	1	1	0	0	0	0	0	0	0	I_n	0
I_n	1	1	1	0	0	0	0	0	0	0	I_n

Tabelle 4: Die Wahrheitstabelle eines 1:8 Demux.

4.3 Kodierer und Dekodierer in VHDL

Ein andere häufig genutzte kombinatorischen Schaltung ist der *Kodierer*. Während ein Multiplexer nur eines der Eingabesignale als Ausgabesignal auswählt, verarbeitet ein Kodierer - oft auch binärer Kodierer genannt - all seine Eingänge einzeln und wandelt sie in ein entsprechendes kodiertes Ausgangssignal um. Abbildung 3 zeigt einen 4:2 Kodierer, der, wie in Wahrheitstabelle 5 dargestellt, ein 4-Bit Eingabesignal $I_0I_1I_2I_3$ in ein kodiertes 2-Bit Signal Y_1Y_2 umwandelt.

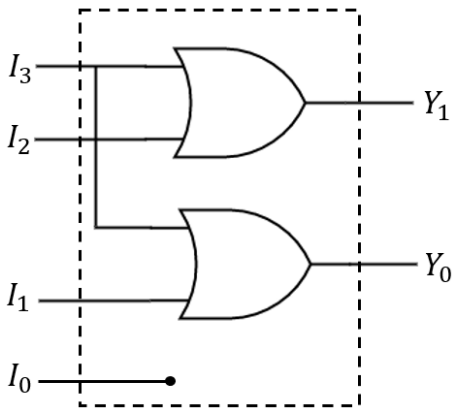


Abbildung 3: 4:2 Kodierer.

Eingabesignale				Ausgabesignale	
I_3	I_2	I_1	I_0	Y_1	Y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Tabelle 5: Die Wahrheitstabelle eines 4:2 Kodierers.

Ein Problem eines solchen einfachen Binärkodierers ist, dass die Ausgabe mit hoher Wahrscheinlichkeit falsch ist, falls mehrere Eingangssignale den logischen Wert "1" besitzen. Wenn zum Beispiel I_2 und I_1 gleichzeitig den logischen Wert "1" haben, so generiert der Binärkodierer die Ausgabe "11", statt "01" oder "10".
Dieses Problem kann man vermeiden wenn verschiedenen Eingängen Prioritäten zugewiesen werden. Sobald mehrere Eingabesignale den logischen Wert "1" besitzen, bestimmt das System den Eingang mit der höchsten Priorität und gibt die entsprechende kodierte Ausgabe aus. Wenn an I_2 eine logische "1" anliegt, so werden die Ausgabesignale Y_1Y_0 nur durch I_2 bestimmt, unabhängig von den welche logischen Werten die an I_1 und I_0 anliegen. Diese Arbeitsweise ist in Tabelle 6 dargestellt.
Aufgaben:

a. (1 Punkte) Entwerft anhand der Wahrheitstabelle 6 einen 4:2 Prioritätskodierer indem ihr das Schaltbild zeichnet.

Eingabesignale				Ausgabesignale	
I_3	I_2	I_1	I_0	Y_1	Y_0
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

Tabelle 6: Die Wahrheitstabelle eines Prioritätskodierers.

b. (2 Punkte) Implementiert den 4:2 Prioritätskodierer in VHDL und testet eure Implementierung in einer Testbench mit $I_3I_2I_1I_0$ jeweils gleich 0001, 0010, 0100 und 1000.

c. (1 Punkte) Dekodierer, die das kodierte Signal zurück in seine originale Form übersetzen, implementieren die umgekehrte Funktion des Kodierers. Entwerft einen 2:4 Dekodierer nach der Wahrheitstabelle 7 indem ihr das Schaltbild zeichnet.

d. (2 Punkte) Implementiert den 2:4 Dekodierer in VHDL und testet eure Implementierung in VHDL mit I_1I_0 jeweils gleich 00, 01, 10 und 11.

Eingabesignale		Ausgabesignale			
I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Tabelle 7: Die Wahrheitstabelle eines 2:4 Dekodierers.