Laborprotokoll Informationstechnologien

TKS 2004, Sommersemester 2004/05

Kombinatorische Beispiele

Übung 1a

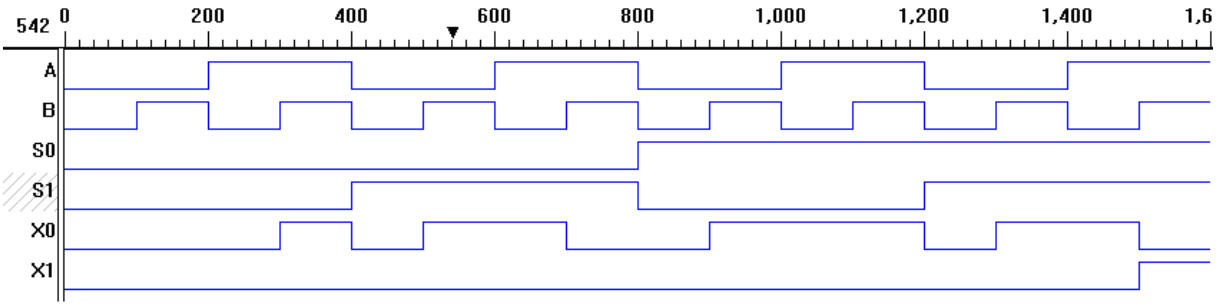
Übungsziel: Programmierung einer 2-Bit-ALU mit Funktionsgleichungen

Quellcode (ABEL):

```
MODULE ZweiBitALU

TITLE 'ZweiBitALU'

A,B pin 2,3;                  //Operanden
S0,S1 pin 4,5;                //Funktionswahl
X0,X1 pin 17,18 istype 'com'; //Ausgänge

equations

WHEN (S0 == 0) & (S1 == 0) THEN {
  X0=A & B;
  X1=0; //Kein Ueberlauf
  }

WHEN (S0 == 0) & (S1 == 1) THEN {
  X0=A $ B;
  X1=0; //Kein Ueberlauf
  }

WHEN (S0 == 1) & (S1 == 0) THEN {
  X0=A # B;
  X1=0; //Kein Ueberlauf
  }

WHEN (S0 == 1) & (S1 == 1) THEN {
  X0=A + B;
  X1=0; //Kein Ueberlauf
  WHEN (A == 1) & (B == 1) THEN //1 + 1 = 10 -> Ueberlauf
    X1=1;      //Undefiniert da Ueberlauf
  }

test_vectors

([S0,S1,A,B]->[X0,X1])

[0,0,0,0]->[.X.,.X.];
[0,0,0,1]->[.X.,.X.];
[0,0,1,0]->[.X.,.X.];
[0,0,1,1]->[.X.,.X.];
[0,1,0,0]->[.X.,.X.];
[0,1,0,1]->[.X.,.X.];
[0,1,1,0]->[.X.,.X.];
[0,1,1,1]->[.X.,.X.];
[1,0,0,0]->[.X.,.X.];
[1,0,0,1]->[.X.,.X.];
[1,0,1,0]->[.X.,.X.];
[1,0,1,1]->[.X.,.X.];
[1,1,0,0]->[.X.,.X.];
[1,1,0,1]->[.X.,.X.];
[1,1,1,0]->[.X.,.X.];
[1,1,1,1]->[.X.,.X.];

END
```

Erläuterungen: Kommentare teilweise im Quelltext. S0 und S1 bestimmen die Art der Operation, die durchgeführt werden soll (und, oder, xor bzw. +), A und B sind die Operanden. Wir haben daher 4 Eingänge (A, B, S0 und S1) und zwei Ausgänge: X0 bzw. X1. X0 ist das Ergebnis der jeweiligen Rechenoperation, X1 signalisiert einen Überlauf. Dieser tritt hier nur in einem Fall auf: eine Addition von 1 und 1 (vgl. Kommentar im Quelltext). Über eine WHEN-Bedingung wird dieser Spezialfall herausgegriffen und X1=1 gesetzt. In allen anderen Fällen ist X1 0, da kein Überlauf auftreten kann.

Impulsdiagramm:



Übung 1b

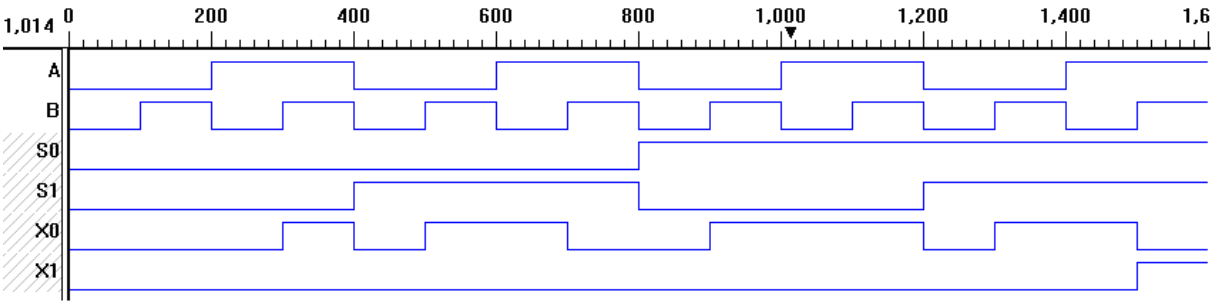
Übungsziel: Programmierung einer 2-Bit-ALU mit Wahrheitstabellen

Quellcode (ABEL):

```
MODULE ZweiBitALU

TITLE 'ZweiBitALU'

A,B pin 2,3;                  //Operanden
S0,S1 pin 4,5;                //Funktionswahl
X0,X1 pin 17,18 istype 'com'; //Ausgänge

TRUTH_TABLE

([S0,S1,A,B]->[X0,X1])

[0,0,0,0]->[0,0];
[0,0,0,1]->[0,0];
[0,0,1,0]->[0,0];
[0,0,1,1]->[1,0];
[0,1,0,0]->[0,0];
[0,1,0,1]->[1,0];
[0,1,1,0]->[1,0];
[0,1,1,1]->[0,0];
[1,0,0,0]->[0,0];
[1,0,0,1]->[1,0];
[1,0,1,0]->[1,0];
[1,0,1,1]->[1,0];
[1,1,0,0]->[0,0];
[1,1,0,1]->[1,0];
[1,1,1,0]->[1,0];
[1,1,1,1]->[0,1]; //Ueberlauf

test_vectors

([S0,S1,A,B]->[X0,X1])

[0,0,0,0]->[.X.,.X.];
[0,0,0,1]->[.X.,.X.];
[0,0,1,0]->[.X.,.X.];
[0,0,1,1]->[.X.,.X.];
[0,1,0,0]->[.X.,.X.];
[0,1,0,1]->[.X.,.X.];
[0,1,1,0]->[.X.,.X.];
[0,1,1,1]->[.X.,.X.];
[1,0,0,0]->[.X.,.X.];
[1,0,0,1]->[.X.,.X.];
[1,0,1,0]->[.X.,.X.];
[1,0,1,1]->[.X.,.X.];
[1,1,0,0]->[.X.,.X.];
[1,1,0,1]->[.X.,.X.];
[1,1,1,0]->[.X.,.X.];
[1,1,1,1]->[.X.,.X.];

END
```

Erläuterungen: Gleiche Funktionsweise wie in Übung 1a, allerdings mit vorgegebenen Wahrheitstabellen anstatt mit Funktionsgleichungen realisiert. Der Überlauf, welcher auch hier auftreten kann und in X1 gespeichert wird, ist im Quelltext mit einem entsprechenden Kommentar gekennzeichnet.

Impulsdiagramm:



Übung 2

Übungsziel: Programmierung eines kompletten Siebensegmentdekoders

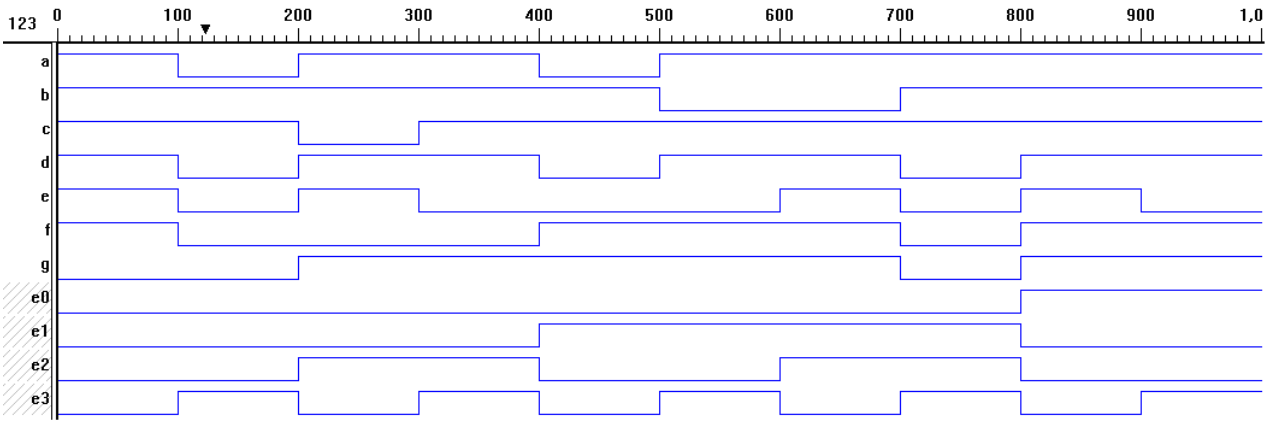
Quellcode (ABEL):

```
MODULE siebenseg

TITLE 'Siebensegmentdekoder'

e0,e1,e2,e3 pin 2,3,4,5; //Eingänge
a,b,c,d,e,f,g pin 13,14,15,16,17,18,19 istype 'com';       //Ausgänge

TRUTH_TABLE

([e0,e1,e2,e3]->[a,b,c,d,e,f,g])

[0,0,0,0]->[1,1,1,1,1,1,0]; //0
[0,0,0,1]->[0,1,1,0,0,0,0]; //1
[0,0,1,0]->[1,1,0,1,1,0,1]; //2
[0,0,1,1]->[1,1,1,1,0,0,1]; //3
[0,1,0,0]->[0,1,1,0,0,1,1]; //4
[0,1,0,1]->[1,0,1,1,0,1,1]; //5
[0,1,1,0]->[1,0,1,1,1,1,1]; //6
[0,1,1,1]->[1,1,1,0,0,0,0]; //7
[1,0,0,0]->[1,1,1,1,1,1,1]; //8
[1,0,0,1]->[1,1,1,1,0,1,1]; //9

test_vectors

([e0,e1,e2,e3]->[a,b,c,d,e,f,g])

[0,0,0,0]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,0,1,0]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,0,1,1]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,0,0]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,0,1]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,1,0]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,1,1]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[1,0,0,0]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[1,0,0,1]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];

END
```

Erläuterungen: Da der Siebensegmentdekoder zehn mögliche Zustände verarbeiten kann (0-9) müssen vier Eingänge (2^4=16) verwendet werden; die sechs Zustände, die übrig bleiben, werden ignoriert. Die Eingänge werden mit e0-e3 benannt. Die sieben Ausgänge werden entsprechend der Bezeichnungen der Segmente benannt (a-g).

Da das Programm über eine Wahrheitstabelle realisiert wird genügt die Spezifikation, wie es auf bestimmte Kombinationen der Eingangssignale zu reagieren hat. Die entsprechenden Zeilen in denen dies geschieht sind durchnummeriert (Kommentare), um erkennen zu können, welcher Dezimalzahl die Kombination der Eingangssignale entspricht.
Das Impulsdiagramm der Testvektoren für die 10 definierten Zustände sieht folgendermaßen aus:

Impulsdiagramm:



Übung 3

Übungziel: Programmierung eines 2 Bit-Komparators

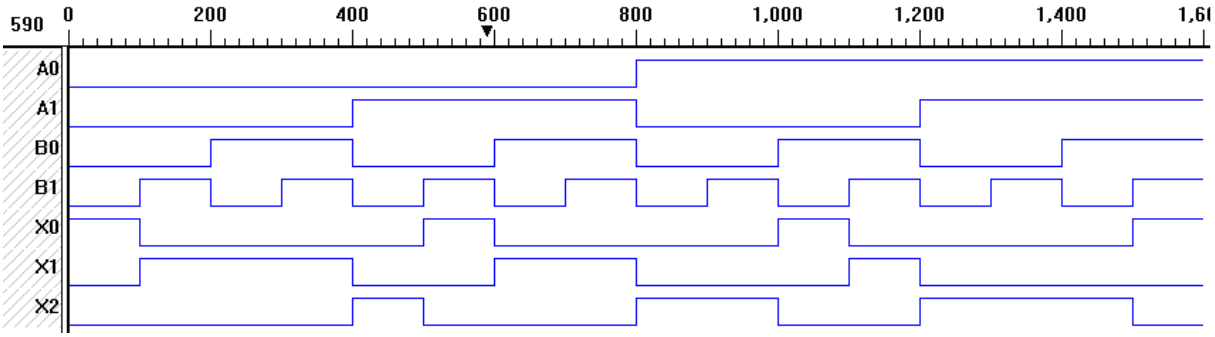
Quellcode (ABEL):

```
MODULE compare

TITLE 'compare'

A0, A1, B0, B1 pin 2,3,4,5;

X0,X1,X2 pin 15,16,17 istype 'com';

//Pin 15: gleich
//Pin 16: kleiner
//Pin 17: größer

TRUTH_TABLE

([A0,A1,B0,B1]->[X0,X1,X2])

[0,0,0,0]->[1,0,0]; //gleich
[0,0,0,1]->[0,1,0]; //kleiner
[0,0,1,0]->[0,1,0]; //kleiner
[0,0,1,1]->[0,1,0]; //kleiner
[0,1,0,0]->[0,0,1]; //größer
[0,1,0,1]->[1,0,0]; //gleich
[0,1,1,0]->[0,1,0]; //kleiner
[0,1,1,1]->[0,1,0]; //kleiner
[1,0,0,0]->[0,0,1]; //größer
[1,0,0,1]->[0,0,1]; //größer
[1,0,1,0]->[1,0,0]; //gleich
[1,0,1,1]->[0,1,0]; //kleiner
[1,1,0,0]->[0,0,1]; //größer
[1,1,0,1]->[0,0,1]; //größer
[1,1,1,0]->[0,0,1]; //größer
[1,1,1,1]->[1,0,0]; //gleich

test_vectors

([A0,A1,B0,B1]->[X0,X1,X2])
```

```
[0,0,0,0]->[.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.];
[0,0,1,0]->[.X.,.X.,.X.];
[0,0,1,1]->[.X.,.X.,.X.];
[0,1,0,0]->[.X.,.X.,.X.];
[0,1,0,1]->[.X.,.X.,.X.];
[0,1,1,0]->[.X.,.X.,.X.];
[0,1,1,1]->[.X.,.X.,.X.];
[1,0,0,0]->[.X.,.X.,.X.];
[1,0,0,1]->[.X.,.X.,.X.];
[1,0,1,0]->[.X.,.X.,.X.];
[1,0,1,1]->[.X.,.X.,.X.];
[1,1,0,0]->[.X.,.X.,.X.];
[1,1,0,1]->[.X.,.X.,.X.];
[1,1,1,0]->[.X.,.X.,.X.];
[1,1,1,1]->[.X.,.X.,.X.];

END
```

Erläuterungen: Für jeden möglichen Zustand wurde der Einfachheit halber ein separater Ausgang (Pins 15-17) verwendet; welcher Zustand welchem Pin entspricht ist in den Kommentaren unter der Deklaration der Pins vermerkt.
Der Komparator wurde über eine Wahrheitstabelle realisiert – die Kommentare am Ende der Zeilen verdeutliche, welcher der Ausgänge bei entsprechendem Auftreten der Eingangszustände folgt.

Impulsdiagramm:



Sequentielle Beispiele

Übung 1a

Übungsziel: Programmierung eines 3 Bit-Binärzählers mittels Wahrheitstabelle

Quellcode (ABEL):

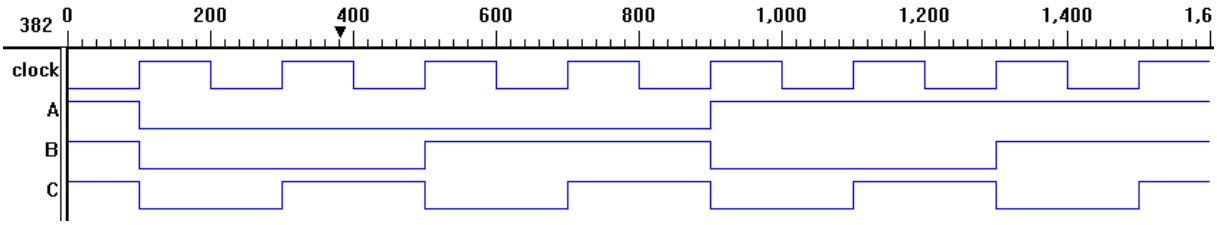
```
MODULE dreibin

TITLE 'dreibin'

A,B,C pin 12,13,14 istype 'reg';

clock pin 1; //Takt

EQUATIONS

[A,B,C].clk=clock;

TRUTH_TABLE

([A,B,C]:>[A,B,C])

[0,0,0]:>[0,0,1];
[0,0,1]:>[0,1,0];
[0,1,0]:>[0,1,1];
[0,1,1]:>[1,0,0];
[1,0,0]:>[1,0,1];
[1,0,1]:>[1,1,0];
[1,1,0]:>[1,1,1];
[1,1,1]:>[0,0,0];

test_vectors

//Takt muss verändert werden!
([A,B,C,clock]->[A,B,C])

[0,0,0,0]->[.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.];

[0,0,1,0]->[.X.,.X.,.X.];
[0,0,1,1]->[.X.,.X.,.X.];

[0,1,0,0]->[.X.,.X.,.X.];
[0,1,0,1]->[.X.,.X.,.X.];

[0,1,1,0]->[.X.,.X.,.X.];
[0,1,1,1]->[.X.,.X.,.X.];

[1,0,0,0]->[.X.,.X.,.X.];
[1,0,0,1]->[.X.,.X.,.X.];

[1,0,1,0]->[.X.,.X.,.X.];
[1,0,1,1]->[.X.,.X.,.X.];

[1,1,0,0]->[.X.,.X.,.X.];
[1,1,0,1]->[.X.,.X.,.X.];

[1,1,1,0]->[.X.,.X.,.X.];
[1,1,1,1]->[.X.,.X.,.X.];

END
```

Erläuterungen: Um sequentielle Ausgänge zu kennzeichnen ist “istype ’reg’“ in der Pin-Deklaration notwendig. In der Wahrheitstabelle müssen die Bindestriche durch Doppelpunkte ausgetauscht werden.
Die Testvektoren müssen den Takt beinhalten, da die Flipflops im GAL auf eine steigende Taktflanke reagieren. In der Simulation wird daher das Impulsdiagramm nur dann richtig angezeigt, wenn der Takt wechselt.

Da der Takteingang des GAL16V8 an Pin 1 liegt muss auch die Clock-Deklaration im oberen Teil des Quelltextes auf Pin 1 gelegt werden. Legt man Clock per Programm auf z.B. Pin 2 wird der Pin beim Kompilieren automatisch angepasst.
Die einzig notwenig Gleichung ist hier ist die Zuweisung des Taktsignals an jene Flipflops, die den Status des Zählers speichern müssen.
Beim Schaltungsaufbau wurde – aus welchen Gründen auch immer – der Zustand 000 übersprungen. Die Ursache dafür konnte nicht gefunden werden; wir mutmaßen, dass die fehlenden Vorwiderstände dafür verantwortlich sind.

Wichtig: OE (Output Enable, Pin 11) muss auf Ground liegen (invertierter Eingang) damit der Zähler zu zählen beginnt.

Impulsdiagramm:



Übung 1b

Übungsziel: Programmierung eines 3 Bit-Binärzählers mittels Statusmaschine

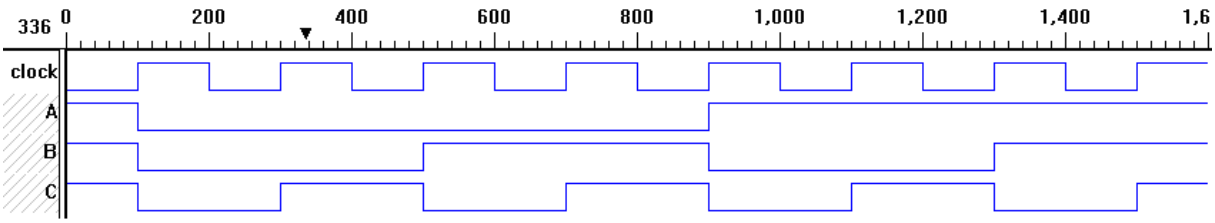
Quellcode (ABEL):

```
MODULE dreibin

TITLE 'dreibin'

A,B,C pin 12,13,14 istype 'reg';

clock pin 1; //Takt

sreg = [A,B,C];

//Zustände
Z1=[0,0,0];
Z2=[0,0,1];
Z3=[0,1,0];
Z4=[0,1,1];
Z5=[1,0,0];
Z6=[1,0,1];
Z7=[1,1,0];
Z8=[1,1,1];

EQUATIONS

[A,B,C].clk=clock;

STATE_DIAGRAM sreg;

State Z1: goto Z2;
State Z2: goto Z3;
State Z3: goto Z4;
State Z4: goto Z5;
State Z5: goto Z6;
State Z6: goto Z7;
State Z7: goto Z8;
State Z8: goto Z1;

test_vectors

//Takt muss verändert werden!
([A,B,C,clock]->[A,B,C])
```

```
[0,0,0,0]->[.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.];

[0,0,1,0]->[.X.,.X.,.X.];
[0,0,1,1]->[.X.,.X.,.X.];

[0,1,0,0]->[.X.,.X.,.X.];
[0,1,0,1]->[.X.,.X.,.X.];

[0,1,1,0]->[.X.,.X.,.X.];
[0,1,1,1]->[.X.,.X.,.X.];

[1,0,0,0]->[.X.,.X.,.X.];
[1,0,0,1]->[.X.,.X.,.X.];

[1,0,1,0]->[.X.,.X.,.X.];
[1,0,1,1]->[.X.,.X.,.X.];

[1,1,0,0]->[.X.,.X.,.X.];
[1,1,0,1]->[.X.,.X.,.X.];

[1,1,1,0]->[.X.,.X.,.X.];
[1,1,1,1]->[.X.,.X.,.X.];

END
```

Erläuterungen: Unter den Pindeklarationen werden die möglichen Zustände des Zählers definiert – hier mit Z1 bis Z8 bezeichnet. Im Statusdiagramm (state_diagram-Abschnitt) wird festgelegt, was passiert, wenn sich die Statusmaschine in einem Zustand befindet und die Taktflanke, auf die die Flipflops im GAL reagieren, kommt. In diesem Fall ist das relativ einfach: die Statusmaschine muss einfach nur in den nächsten Zustand übergehen.

Impulsdiagramm:



Übung 2

Übungsziel: Programmierung eines BCD-Zählers mittels Statusmaschine zur Ausgabe auf einer Siebensegmentanzeige

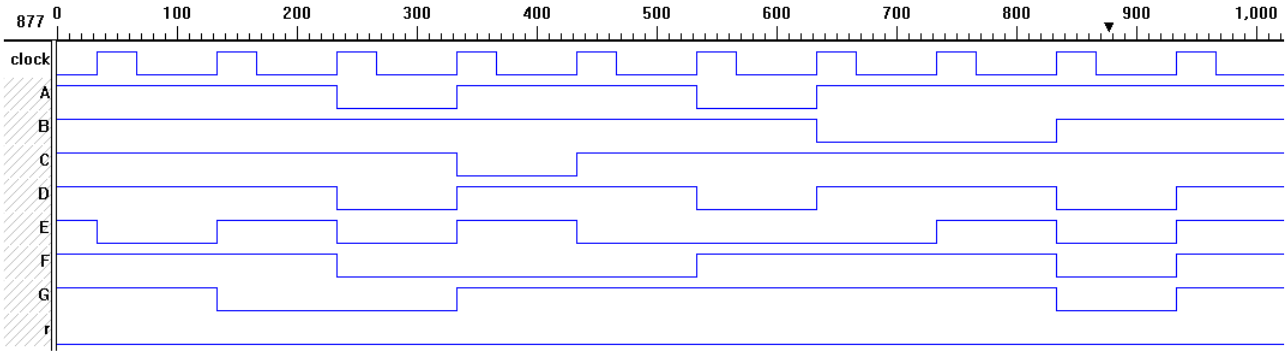
Quellcode (ABEL):

```
MODULE bcdz

TITLE 'bcdz'

A,B,C,D,E,F,G pin 12,13,14,15,16,17,18 istype 'reg';

clock, r pin 1,2; //Takt und Umkehrung (wenn r=1)

sreg = [A,B,C,D,E,F,G];

//Zustände
Z0=[1,1,1,1,1,1,0];
Z1=[0,1,1,0,0,0,0];
Z2=[1,1,0,1,1,0,1];
Z3=[1,1,1,1,0,0,1];
Z4=[0,1,1,0,0,1,1];
Z5=[1,0,1,1,0,1,1];
```

```
Z6=[1,0,1,1,1,1,1];
Z7=[1,1,1,0,0,0,0];
Z8=[1,1,1,1,1,1,1];
Z9=[1,1,1,1,0,1,1];

EQUATIONS

[A,B,C,D,E,F,G].clk=clock;

STATE_DIAGRAM sreg;

State Z0:
  if (!r) then goto Z1;
  else goto Z9;
State Z1:
  if (!r) then goto Z2;
  else goto Z8;
State Z2:
  if (!r) then goto Z3;
  else goto Z7;
State Z3:
  if (!r) then goto Z4;
  else goto Z6;
State Z4:
  if (!r) then goto Z5;
  else goto Z5;
State Z5:
  if (!r) then goto Z6;
  else goto Z4;
State Z6:
  if (!r) then goto Z7;
  else goto Z3;
State Z7:
  if (!r) then goto Z8;
  else goto Z2;
State Z8:
  if (!r) then goto Z9;
  else goto Z1;
State Z9:
  if (!r) then goto Z0;
  else goto Z0;

test_vectors

//Takt muss verändert werden!
([r,A,B,C,D,E,F,G,clock]->[A,B,C,D,E,F,G])

[0,1,1,1,1,1,1,0,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,0,1,1,0,0,0,0,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,1,0,1,1,0,1,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,1,1,1,0,0,1,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,0,1,1,0,0,1,1,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,0,1,1,0,1,1,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,0,1,1,1,1,1,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,1,1,0,0,0,0,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,1,1,1,1,1,1,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];
[0,1,1,1,1,0,1,1,.c.]->[.X.,.X.,.X.,.X.,.X.,.X.,.X.];

END
```

Erläuterungen: Das Prinzip ist ähnlich dem aus Übung 1b, mit dem Unterschied, dass der Zähler auch umkehrbar ist, d.h. dass ein zusätzlicher Eingang existiert, der – je nach Beschaltung – ein Vorwärts- oder Rückwärtszählen verursacht.
Zum Verkürzen des Quelltextes bei den Testvektoren wurde der wechselnde Takt durch ".c." ersetzt – ABEL simuliert den Takt dann selbst und erspart in diesem Fall zehn weitere Programmzeilen.
Die Zustände wurden so definiert, dass die Ausgänge des GALs (hier Pins 12-18) gleich auf eine Siebensegmentanzeige gelegt werden konnten. Durch diese Vorgehensweise war es möglich, den Zähler problemlos in ein einziges GAL zu packen.
Der Einfachheit halber wurde bei den Testvektoren nur das Vorwärtszählen demonstriert, der Aufbau der Schaltung zeigt, dass das Rückwärtszählen ohne Probleme möglich war.

Impulsdiagramm:



Übung 3a

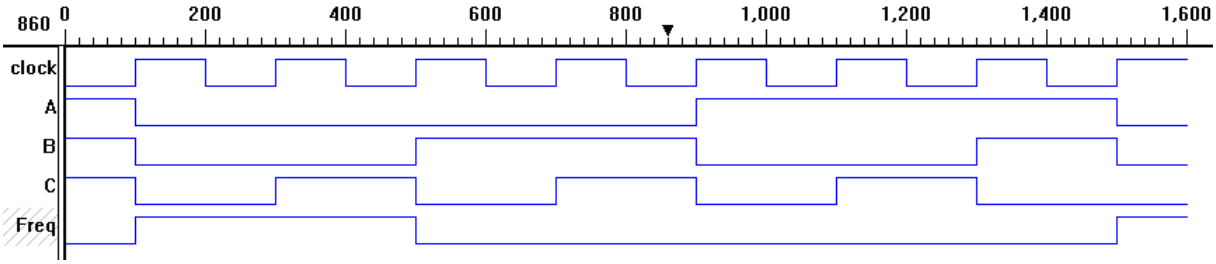
Übungziel: Programmierung eines Frequenzteilers 1:7, Tastverhältnis 2:5 (Hi:Lo) mittels Wahrheitstabelle

Quellcode (ABEL):

```
MODULE freqt_a

TITLE 'Frequenzteiler'

A,B,C pin 12,13,14 istype 'reg'; //Zähler
Freq pin 15 istype 'com'; //Frequenzausgang mit 2:5

clock pin 1; //Takt

EQUATIONS

[A,B,C].clk=clock;

TRUTH_TABLE

([A,B,C]:>[A,B,C]->Freq)

[0,0,0]:>[0,0,1]->1; //2x High
[0,0,1]:>[0,1,0]->1;
[0,1,0]:>[0,1,1]->0; //5x Low
[0,1,1]:>[1,0,0]->0;
[1,0,0]:>[1,0,1]->0;
[1,0,1]:>[1,1,0]->0;
[1,1,0]:>[0,0,0]->0; //Nur 7 Zustände, daher zu 000 zurück

test_vectors

//Takt muss verändert werden!
([A,B,C,clock]->[A,B,C,Freq])

[0,0,0,0]->[.X.,.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.,.X.];
[0,0,1,0]->[.X.,.X.,.X.,.X.];
[0,0,1,1]->[.X.,.X.,.X.,.X.];
[0,1,0,0]->[.X.,.X.,.X.,.X.];
[0,1,0,1]->[.X.,.X.,.X.,.X.];
[0,1,1,0]->[.X.,.X.,.X.,.X.];
[0,1,1,1]->[.X.,.X.,.X.,.X.];
[1,0,0,0]->[.X.,.X.,.X.,.X.];
[1,0,0,1]->[.X.,.X.,.X.,.X.];
[1,0,1,0]->[.X.,.X.,.X.,.X.];
[1,0,1,1]->[.X.,.X.,.X.,.X.];
[1,1,0,0]->[.X.,.X.,.X.,.X.];
[1,1,0,1]->[.X.,.X.,.X.,.X.];
[0,0,0,0]->[.X.,.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.,.X.];

END
```

Erläuterungen: Der Frequenzteiler wird mittels eines modifizierten 3-Bit-Binärzählers realisiert, der nur von 0 bis 6 zählt und dann wieder bei 0 beginnt (7 Zustände). Zu beachten bei der Programmierung ist hier nur eines: in der Deklaration der Wahrheitstabelle müssen die sequentiellen und die kombinatorischen Glieder voneinander getrennt werden. Da der Frequenzausgang (derjenige, an dem dann das Taktverhältnis 2:5 auftritt) kombinatorisch und der (interne Zähler) sequentiell ist, müssen die beiden voneinander getrennt geschrieben werden (Beispiel: `[0,0,1]:>[0,1,0]->1;`). Beim Aufbau ist generell auf die Verwendung der richtigen Vorwiderstände zu achten.

Impulsdiagramm:



Übung 3b

Übungziel: Programmierung eines Frequenzteilers 1:7, Tastverhältnis 2:5 (Hi:Lo) mittels Statusdiagramm

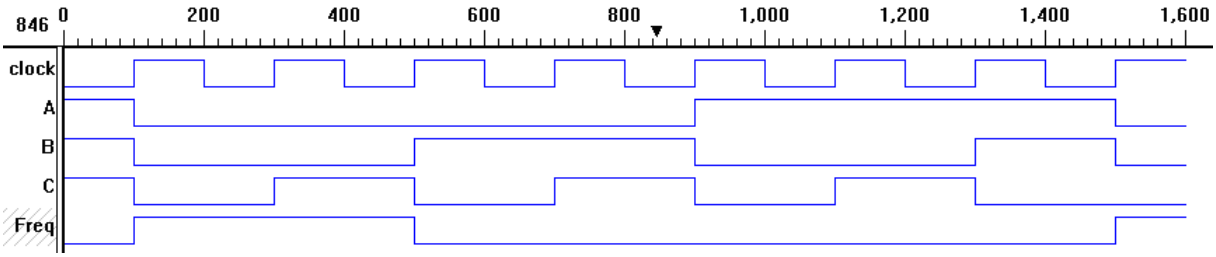
Quellcode (ABEL):

```
MODULE freqtb

TITLE 'Frequenzteiler'

A,B,C pin 12,13,14 istype 'reg'; //Zähler

Freq pin 15 istype 'com'; //Frequenzausgang (2:5)

clock pin 1; //Takt

sreg = [A,B,C];

//Zustände

Z1=[0,0,0];
Z2=[0,0,1];
Z3=[0,1,0];
Z4=[0,1,1];
Z5=[1,0,0];
Z6=[1,0,1];
Z7=[1,1,0];

EQUATIONS

[A,B,C].clk=clock;

STATE_DIAGRAM sreg;

State Z1: Freq = 1; goto Z2; //2x High
State Z2: Freq = 1; goto Z3;
State Z3: Freq = 0; goto Z4; //5x Low
State Z4: Freq = 0; goto Z5;
State Z5: Freq = 0; goto Z6;
State Z6: Freq = 0; goto Z7;
State Z7: Freq = 0; goto Z1; //Nur 7 Zustände, daher hier zurück zu 000

test_vectors

//Takt muss verändert werden!
([A,B,C,clock]->[A,B,C,Freq])
```

```
[0,0,0,0]->[.X.,.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.,.X.];
[0,0,1,0]->[.X.,.X.,.X.,.X.];
[0,0,1,1]->[.X.,.X.,.X.,.X.];
[0,1,0,0]->[.X.,.X.,.X.,.X.];
[0,1,0,1]->[.X.,.X.,.X.,.X.];
[0,1,1,0]->[.X.,.X.,.X.,.X.];
[0,1,1,1]->[.X.,.X.,.X.,.X.];
[1,0,0,0]->[.X.,.X.,.X.,.X.];
[1,0,0,1]->[.X.,.X.,.X.,.X.];
[1,0,1,0]->[.X.,.X.,.X.,.X.];
[1,0,1,1]->[.X.,.X.,.X.,.X.];
[1,1,0,0]->[.X.,.X.,.X.,.X.];
[1,1,0,1]->[.X.,.X.,.X.,.X.];
[0,0,0,0]->[.X.,.X.,.X.,.X.];
[0,0,0,1]->[.X.,.X.,.X.,.X.];

END
```

Erläuterungen: Im Prinzip ist auch dieser Frequenzteiler nichts anderes als ein modifizierter 3 Bit-Binärzähler, der nur von 0 bis 6 zählt. Unterschied zur Programmierung per Statusdiagramm (Übung 3a): die 'freq'-Variable (Taktausgang) kann nicht über den Zustand direkt definiert sein; vor dem Sprung zum nächsten Status muss sie per Zuweisung (z.B. `Freq = 0;`) gesetzt werden.

Impulsdiagramm:



Die restlichen Übungen konnten aus Zeitgründen nicht mehr durchgeführt werden. Alle hier beschriebenen Beispiele wurden aufgebaut und auf ihre Funktionalität überprüft. Anmerkungen zu den Schaltungen befinden sich – sofern vorhanden – unter den Übungsnummern.