Universität Hamburg, Fachbereich Informatik

Arbeitsbereich Technische Aspekte Multimodaler Systeme (TAMS)

Praktikum der Technischen Informatik

T1 – 4

Programmierbare Logik

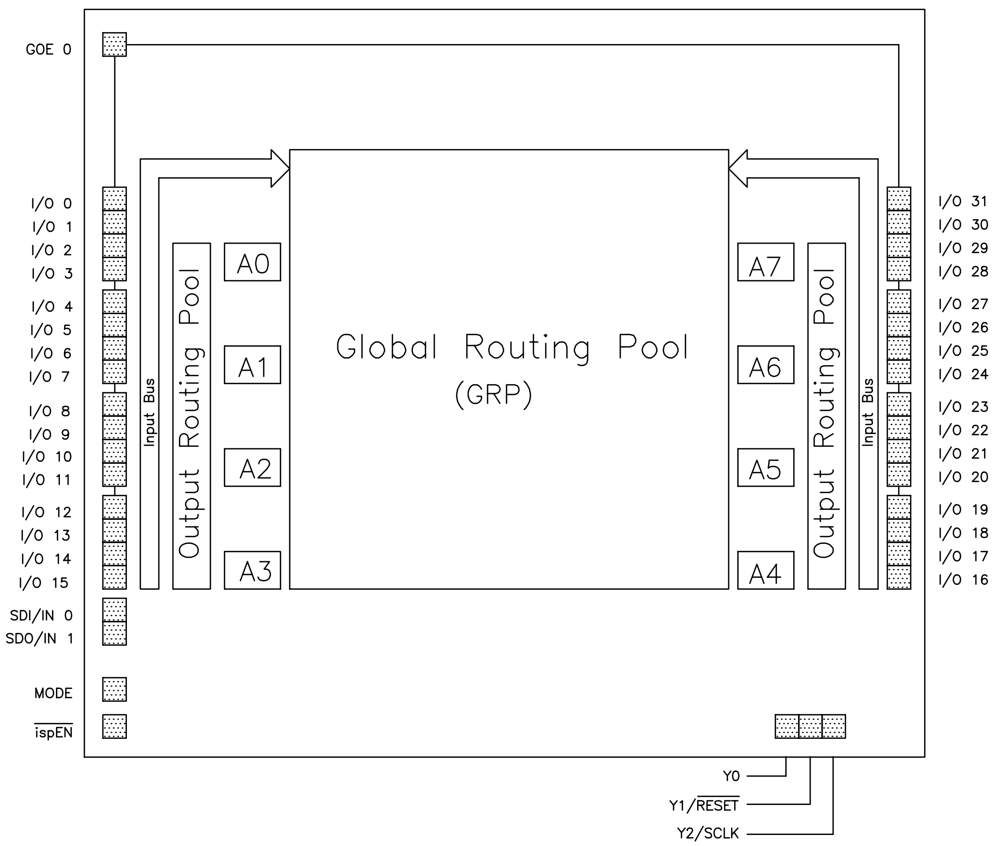
Name: ....

Bogen erfolgreich bearbeitet: ....

FPLDs

In den letzten Versuchen hatten wir mit mehr oder minder großem Verdrahtungsaufwand versucht, digitale Schaltungen zu entwerfen und dann aufzubauen. Dabei hatte sich gezeigt, dass sich selbst kleine Schaltungen mit unseren Mitteln nur recht schwer realisieren lassen. In diesem Versuch wollen wir uns das Leben etwas leichter machen und zur Realisierung unserer Schaltungen programmierbare Bausteine der Firma Lattice verwenden. Es sind dies Weiterentwicklungen der aus der T-Vorlesung (hoffentlich) bekannten PLAs und PALs, also Bausteinen, die sich mit einer vom Benutzer vorgegebenen Logikfunktion programmieren lassen. Im Gegensatz zu PALs lassen sie sich aber wieder löschen und mit einer anderen Funktion neu programmieren. Zudem benötigt man zu ihrer Programmierung kein besonderes Gerät, sondern nur ein Kabel, das man an den Druckerport eines PCs steckt. Man spricht dann auch von einem **F**ield **P**rogrammable **L**ogic **D**evice.

Das folgende Bild zeigt den Aufbau des Bausteins vom Typ ispLSI 2032, der bis auf eine kleinere Zahl von GLBs prinzipiell ebenso aufgebaut ist wie der Baustein vom Typ ispLSI 1016, den wir in diesem Versuch verwenden werden.



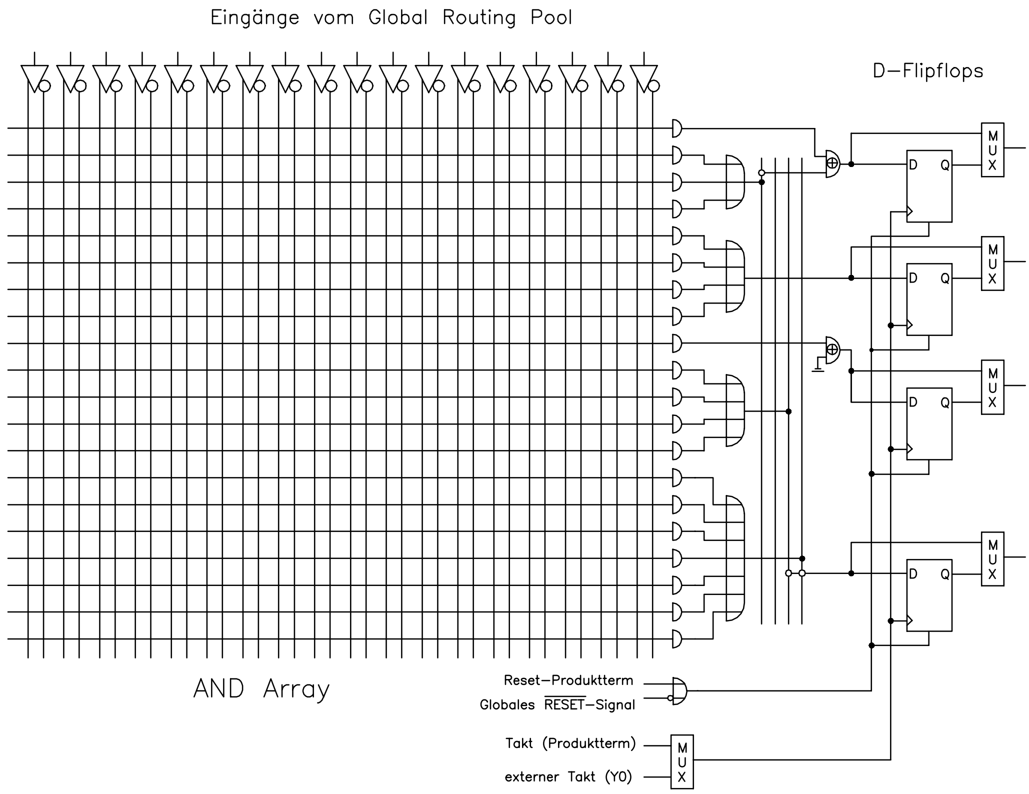
Die externen Anschlüsse des Bausteins haben dabei folgende Bedeutung:

- I/O 0 bis I/O 31 sind Anschlüsse des Bausteins, die sich wahlweise als Ein- bzw. Ausgang verwenden lassen.
- SDI/IN 0, SDO/IN 1, MODE, $\overline{\text{ispEN}}$ und SCLK dienen zur Programmierung des Bausteins.
- Y0 dient als Takteingang für die im Baustein enthaltenen D-Flipflops (s.u.)
- Y1 lässt sich bei entsprechender Programmierung als low-aktiver Rücksetzeingang für die internen Flipflops verwenden.
- GOE 0 dient als Enable-Eingang für die Ein/Ausgänge, d.h. über dieses Signal lassen sich die Pins I/O 0 - I/O 31 in einen hochohmigen Zustand bringen (Tristate-Ausgänge).

Wie auf dem Bild zu sehen, führen von den I/O-Pins Leitungsbündel zum sog. Global Routing Pool, der im wesentlichen zur internen Verdrahtung der vom Benutzer programmierten Schaltung dient. Ebenso gibt es zwei Output Routing Pools, in denen die Ausgänge der Schaltung verdrahtet werden.

Die eigentlichen Logikfunktionen befinden sich in mit A0 bis A7 bezeichneten Blöcken, den sog. Generic Logic Blocks (GLB). Das Bild auf der nächsten Seite zeigt in etwas vereinfachter Form eine der Möglichkeiten, die innere Struktur eines GLB zu konfigurieren.

Links ist die von einen PLA bekannte UND-Matrix. Hier lassen sich neunzehn verschiedene Produktterme bilden. Diese können dann verodert werden. Dazu gibt es in jedem GLB ein ODER-Gatter mit drei, zwei ODER-Gatter mit vier und ein ODER-Gatter mit sieben Eingängen. Weiter stehen zwei Exklusiv-Oder-Glieder zur Verfügung. Die Ergebnisse der logischen Verknüpfungen können dann über einen Multiplexer direkt aus dem GLB herausgeführt oder auf den D-Eingang eines der vier in jedem GLB vorhandenen vorderflankengesteuerten D-Flipflops geschaltet werden. Wie zu sehen lässt sich sowohl der Takt als auch das Reset-Signal aller Flipflops innerhalb eines GLB als Produktterm definieren. Wir werden hiervon aber keinen Gebrach machen.



Die Beschreibungssprache

In diesem Abschnitt soll der hier relevante Teil der Eingabesprache zur Schaltungsbeschreibung erläutert werden. Dazu sollen die Schaltungen aus Versuch 1.2, 2.1 und 3.1 als Beispiele dienen.

In Versuch 1.2 hatten wir einen Multiplizierer mit Overflowflag entworfen. Eine vollständige, allerdings bewusst unnötig umständliche, Beschreibung dieser Schaltung für den ispLSI 2030 könnte dann wie folgt aussehen:

```
LDF 1.00.00 DESIGNLDF;           // Ueberschrift
DESIGN MULTIPLIZIERER;           // Name des Designs

PART ispLSI1016-80LJ44;          // Verwendeter Baustein

SYM GLB  A0  1;               // Beschreibung des GLBs A0
SIGTYPE   TERM1  OUT;
SIGTYPE   TERM2  OUT;
EQUATIONS
TERM1= Do1 & Do3;                // Die Logikgleichungen
TERM2= Do2 & Do4;                // fuer zwei Hilfsterme
end;
END;
```

```
SYM GLB  A1  1;               // Beschreibung des GLBs A1
EQUATIONS
Di1= TERM1;
Di2= (Do1 & Do4) # (Do2 & Do3);
Di3= TERM2;
Di4= TERM1 & TERM2;
end;
END;

SYM IOC  IO0  1  ;         // Beschreibung der I/O-Pins
XPIN IO XDo1 LOCK 15;      // Dem Signal mit Namen XDO1 wird
                           // Pin 15 des Chips zugewiesen
IB11 (Do1,  XDo1);         // Intern heisst das Signal DO1
END;

SYM IOC  IO1  1  ;
XPIN IO XDo2 LOCK 16;
IB11 (Do2, XDo2);
END;

SYM IOC  IO2  1  ;
XPIN IO XDo3 LOCK 17;
IB11 (Do3, XDo3);
END;

SYM IOC  IO3  1  ;
XPIN IO XDo4 LOCK 18;
IB11 (Do4, XDo4);
END;

SYM IOC  IO8  1  ;           // Dasselbe fuer die Ausgaenge
XPIN IO XDi1 LOCK 6;         // der Schaltung
OB11 (XDi1, Di1);
END;

SYM IOC  IO9  1  ;
XPIN IO XDi2 LOCK 5;
OB11 (XDi2, Di2);
END;

SYM IOC  IO10  1  ;
XPIN IO XDi3 LOCK 4;
OB11 (XDi3, Di3);
END;

SYM IOC  IO11  1  ;
XPIN IO XDi4 LOCK 3;
OB11 (XDi4, Di4);
END;
END;                                  // LDF DESIGNLDF
```

Die Schaltung ist in unserem Beispiel auf zwei GLBs aufgeteilt worden, was eigentlich nicht notwendig gewesen wäre; bei nur vier Ausgängen hätte auch schon ein GLB ausgereicht. “//” dient als Beginn eines Kommentars.

Da TERM1 und TERM2 Ausgangsleitungen eines GLBs sind, die nicht auf einen I/O-Pin des gesamten Bausteins gehen wie Di1 - Di4, müssen sie über eine SIGTYPE-Anweisung mit dem Schlüsselwort OUT besonders deklariert werden, damit man in einem anderen GLB auf sie Bezug nehmen kann.

Erlaubt sind bei der Eingabe der Logikgleichungen folgende Operatoren

!	für eine Negation
&	für eine UND-Verknüpfung
#	für eine ODER-Verknüpfung
$	für eine EXOR-Verknüpfung
!$	für eine Äquivalenz-Verknüpfung

Klammern sind erlaubt. Die obigem Beispiel hätte man somit z.B. die Gleichung für Di2 mit NANDs auch wie folgt schreiben können:

```
Di2= !( !(Do1 & Do4) & !(Do2 & Do3));
```

Als nächstes Beispiel soll das RS-Flipflop aus Versuch 2.1 betrachtet werden. Dazu ist zu bemerken, dass das verwendete Entwicklungsprogramm keine direkte Rückkopplung eines Ausgangs auf einen Eingang eines GLBs zulässt, sondern ein besonderes Konstrukt fordert. Die folgende Beschreibung ist also falsch:

```
SYM GLB  A0  1;                 // falsche Beschreibung
EQUATIONS                       // eines RS-Flipflops
Di1= !(Do1 & Di2);
Di2= !(Do2 & Di1);
end;
END;
```

Richtig muss es lauten (die Extension “.pin” ermöglicht es, in einem Schaltnetz Ausgänge auf Eingänge von GLBs rückzukoppeln):

```
SYM GLB  A0  1;                // richtige Beschreibung
EQUATIONS                      // eines RS-Flipflops
Di1= !(Do1 & Di2.pin);
Di2= !(Do2 & Di1.pin);
end;
END;
```

Als letztes Beispiel soll das Schaltwerk aus Versuch 3.1 betrachtet werden. Eine mögliche Beschreibung lautet:

```
SYM GLB  A0  1;          // Beschreibung eines
SIGTYPE  Di1  REG OUT;   // Schaltwerks DI1 ist Ausgang
                         // eines D-Flipflops
SIGTYPE  Di2  REG OUT;   // DI2 ebenso
EQUATIONS
Di1.CLK=  CLOCK;         // Das D-FF wird mit dem
Di2.CLK=  CLOCK;         // Signal CLOCK getaktet
Di1=  Di2;               // Oder auch: Di1.d= Di2.q;
Di2= !Di1;               // Oder auch: Di2.d= !Di1.q;
end;
END;
```

Man beachte, dass bei Schaltwerken die Extension ".pin" bei der Rückkopplung des Ausgangs eines Registers auf einen Eingang nicht notwendig und auch nicht erlaubt ist.

Die Design-Software

Um in folgenden Versuchen zu einem programmierten isp-Baustein zu gelangen, sind folgende Schritte zu durchlaufen:

- Eingabe der Beschreibung
  Dazu kann man z.B. mit dem Windows-Editor seine Beschreibung eingegeben. Die Beschreibung sollte dabei die Extension .LDF haben.
- Einlesen der Beschreibung in die Design-Software
- Aufruf eines Programmteils zum "Verifizieren" des Designs, "Routen" (Verdrahten) der GLBs über den Global Routing Pool und Aufruf Erzeugen einer sog. Fuse-Map in Form eines JEDEC-Files
- Programmierung des Bausteins

Nun zu den einzelnen Schritten:

Einlesen des Designs

Klicken Sie in der Menue-Leiste **File** und dann **Import LDF** an. Achten Sie bitte auf mögliche Fehlermeldungen im "Message-Fenster".

Verifizieren des Designs

Klicken Sie in der Button-Leiste rechts oben auf **All**. Achten Sie bitte auf mögliche Fehlermeldungen.

Laden des Designs in den Baustein

- Klicken sie auf den Button **DOWNLOAD**

- Wählen Sie den Button **SCAN**. Das Programm sollte dann den Baustein auf ihrer Platine richtig erkennen.
- Wählen Sie mit **BROWSE** die richtige JED-Datei aus.
- Wählen Sie im Menuepunkt **Command** den Eintrag **Run Operation**

Praktische Hinweise und Restriktionen

Die Schaltungen für die folgenden Versuche sind bereits fest verdrahtet, so dass wirklich nur noch die Beschreibung für den ispLSI-1016 erstellt werden muss. Verdrahtet sind dabei alle acht digitalen Ein- bzw. Ausgänge Di1-Di8 und Do1-Do8. Wichtig ist dabei, dass der Ausgang Do5 für den Takt der internen Flipflops reserviert ist.

Für die einzelnen Versuche gibt es bereits vorgefertigte Rümpfe, in den inbesondere schon die Definition der I/O-Pins enthalten ist. Sie brauchen also nur noch die Beschreibung für das Innenleben der GLBs (Logikgleichungen und Signaldeklarationen) zu erstellen. Achten Sie besonders darauf, dass in einem Schaltwerk ein D-Flip-Flop als solches deklariert werden muss und immer einen Takt braucht (siehe Seite 7). Bei den einzelnen Versuchen finden Sie die Namen der Beschreibungsrümpfe und Angaben zu den vorverdrahteten Ein- und Ausgängen.

Achten Sie bei Ihrer Schaltungsbeschreibung bitte besonders auf folgende Punkte:

- Füllen Sie einen GLB nicht mit zu vielen logischen Funktionen, sondern verteilen Sie ihr Design lieber über mehrere GLBs.
- Achten Sie darauf, dass es in jedem GLB nur vier Ausgänge zum Global Routing Pool gibt, d.h. innerhalb eines GLBs dürfen maximal vier Leitungen als SIGTYPE OUT deklariert werden bzw. externe Ausgänge des Bausteins sein.
- In jedem GLB gibt es nur vier Register (D-Flipflops), d.h. innerhalb eines GLBs dürfen maximal vier Signale als SIGTYPE REG deklariert sein.
- Die in den Beschreibungsrümpfen verwendeten Signalnamen Do1 bis Do8 und Di1 bis Di8 sind natürlich wenig aussagekräftig. Es steht Ihnen frei, in Ihrem Design andere Namen zu verwenden. Dann müssen Sie aber auch die Deklaration der externen Anschlüsse, d.h. die SYM IOC-Deklarationen entsprechend ändern. Wenn Sie z.B. dem Ausgang mit Namen Di1 den neuen Namen Q0 zuordnen wollen wird aus

  ```
  SYM IOC  IO8  1  ;
  XPIN IO XDi1 LOCK 6;
  OB11 (XDi1, Di1);
  END;
  ```

  die neue Deklaration

```
SYM IOC   IO8   1   ;
XPIN IO XDi1 LOCK 6;
OB11 (XDi1, Q0);
END;
```

Alles andere sollte unverändert bleiben.

- Auf der Platine befindet sich ein RESET-Knopf mit dem sich alle D-Flipflops rücksetzen lassen, so dass sie eine 0 am Ausgang liefern.
- Wenn Sie einen Automaten (Schaltwerk) entwerfen, der nicht 2^n verschiedene Zustände hat, könnte es günstig sein, einem der Zustände eine Zustandscodierung mit lauter Nullen zuzuweisen, da sich dieser Zustand durch Drücken des RESET-Knopfes immer erreichen lässt. Sonst muss man sich Gedanken darüber machen, was geschieht, wenn sich der Automat zu Anfang in einem ungültigen Zustand befindet.

Vor den Aufgaben noch ein Wort der Warnung: Die Erfahrung zeigt, dass man für die Aufgaben 4.3 und 4.4 inklusive Aufstellen der Gleichungen (mindestens) je eine Stunde braucht. Man sollte sich also ernsthaft überlegen, was man schon zuhause vorbereiten kann, um nicht in unnötige Schwierigkeiten zu geraten.

Versuch 4.1: Modifizierter Multiplizierer

Beschreiben Sie noch einmal den Multiplizierer aus Versuch 1.2. Dabei soll es jetzt aber kein Overflow-Flag geben, sondern ein echtes vier Bit breites Ergebnis erzeugt werden. Programmieren Sie einen ispLSI-1016 mit Ihrer Beschreibung und testen Sie sie mit dem schon bekannten Programm WINETPS. Dazu gibt es wie auch für die folgenden Versuche eine vorgefertigte Rumpfdatei, in die Sie noch noch ihre Logikgleichungen und gegebenenfalls die Deklarationen für die von Ihnen verwendeten Flipflops eintragen müssen.

Rumpfdatei: `C:\SORC\LDF\MUL.LDF`
Vordefinierte Ein-/Ausgänge: Do1-Do4 Eingänge der Schaltung, Di1-Di4 das Ergebnis der Multiplikation.

Notieren Sie hier Ihre Gleichungen:

Di1=

Di2=

Di3=

Di4=

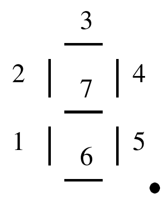
Aufgabe gelöst:	Gruppe:	TeilnehmerIn: .... vom Betreuer auszufüllen

<u>Versuch 4.2:</u> Ansteuerung einer Sieben-Segment-Anzeige

Auf einer Sieben-Segment-Anzeige sollen Buchstaben dargestellt werden. Diese sollen in vier Bit codiert sein und es gelte:

A - 0000, B - 0001, C - 0010, ... , P - 1111

Eine Sieben-Segment-Anzeige (SSA) besteht aus sieben Leuchtsegmenten und einem Punkt, die wie folgt angeordnet sind:



Für den Buchstaben A sollten also z.B. die Segmente 1, 2, 3, 4, 5 und 7 leuchten, der Punkt aber nicht. Offenbar lassen sich aber nicht alle oben angegeben Buchstaben sinnvoll auf einer SSA darstellen (z.B. M); für diese Buchstaben soll der Punkt leuchten, die Ansteuerung der Segmente ist beliebig. Andere Darstellungen sind mehrdeutig (z.B. D und O). Hier sollte das O dargestellt werden und das D wie ein ungültiger Buchstabe behandelt werden.

Bestimmen Sie die logischen Funktionen für die Ansteuerung der SSA, programmieren Sie einen ispLSI-1016 mit Ihrer Beschreibung und testen Sie sie (Menue **Digital/Einzelmessung**).

Den Segmenten 1-7 sind dabei durch die Vordrahtung schon fest die Ausgänge Di1-Di7 und dem Punkt der Ausgang Di8 zugeordnet. Die Eingänge der Schaltung liegen auf Do1-Do4.

Eine Rumpfbeschreibung steht in der Datei `C:\SORC\LDF\LETS.LDF`. Beachten Sie dabei, dass dabei die einzelnen Buchstaben schon vordefiniert vorliegen, so dass Terme wie Di8= D # K # M # N möglich sind.

Notieren Sie hier ihre Gleichungen:

Aufgabe gelöst: Gruppe: .... TeilnehmerIn: ....
vom Betreuer auszufüllen

<u>Versuch 4.3:</u> Würfel

Es soll ein digitaler Würfel entworfen werden, d.h. eine Schaltung die auf einem Feld von sieben Leuchtdioden

```
1 ○       ○ 2
     4
3 ○   ○   ○ 5

6 ○       ○ 7
```

zyklisch folgende Ausgaben erzeugt:

○ ○	○ •	○ •	• •	• •	• •
○ • ○	○ ○ ○	○ • ○	○ ○ ○	○ • ○	• ○ •
○ ○	• ○	• ○	• •	• •	• •
1	2	3	4	5	6

Offenbar werden die LEDs 1 und 7, 2 und 6 sowie 3 und 5 immer gleich angesteuert, so dass sich die Zahl der Ausgänge der Schaltung etwas reduziert.

Entwerfen Sie einen Automaten (ein Schaltwerk) für die Ansteuerung der LEDs. Der Automat habe weiter einen Eingang S. Wenn S= 0 ist, wird die Zustandsfolge zyklisch durchlaufen; wenn S= 1 ist, bleibt der Automat im aktuellen Zustand.

Programmieren Sie einen ispLSI-1016 mit dem Automaten und testen Sie ihn. Falls Ihnen unklar ist, wie man Flip-Flops definiert, sehen Sie sich noch einmal das Beispiel auf Seite 7 an. Für die LEDs ist folgende Zuordnung bereits fest vorverdrahtet:

Diode	Name
1, 7	Di4
2, 6	Di5
3, 5	Di6
4	Di7

Dem Eingang S ist Do1 zugeordnet und der Name CLOCK dem Takt der Flipflops, der fest auf Do5 vorverdrahtet ist. Die Zustände ihres Automaten sollten Sie zu Testzwecken auf die Ausgänge Di1-Di3 legen, falls Sie sich für eine Zustandscodierung mit drei Bit entscheiden.

Die Rumpfbeschreibung steht in der Datei `C:\SORC\LDF\WUERF.LDF`.

Notieren Sie hier ihre Gleichungen für ihr Überführungs- und das Ausgabeschaltnetz:

Aufgabe gelöst:	Gruppe:	TeilnehmerIn: ....
		vom Betreuer auszufüllen

Von den beiden folgenden Versuchen können Sie sich einen zur Bearbeitung aussuchen:

Versuch 4.4.a: Ampelschaltung

Die Ampelschaltung aus Versuch 3.5 soll etwas erweitert werden. Nachgebildet werden soll eine Fußgängerampel, d.h. eine Schaltung, die zyklisch folgende Ausgaben liefert:

Zustand	Autoampel	Fußgängerampel
A	grün	rot
B	gelb	rot
C	rot	rot
D	rot	grün
E	rot	grün
F	rot	rot
G	rot, gelb	rot
H	grün	rot

Entwerfen Sie einen Automaten für die Ansteuerung der Ampeln. Der Automat habe dabei einen Eingang G (den Druckknopf der Fußgängerampel). Wenn G= 0 ist und sich der Automat im Zustand A befindet, bleibt er im Zustand A. Sonst wird die Zustandsfolge zyklisch durchlaufen, d.h. es wartet ein Fußgänger an der Ampel.

Erstellen Sie eine Beschreibung für den Automaten und testen Sie ihn. Für die Ampel ist folgende Zuordnung bereits fest vorverdrahtet:

Diode	Name
Fußgänger grün	Di8
Fußgänger rot	Di7
Auto rot	Di4
Auto gelb	Di5
Auto grün	Di6

Die Zustände Ihres Automaten können Sie sich z.B. auf Di1-Di3 ansehen. G ist auf Do1 vordefiniert und der Takt auf dem Ausgang Do5 hat den Namen CLOCK.
Eine Rumpfbeschreibung befindet sich in der Datei `C:\SORC\LDF\FUAM.LDF`

Notieren Sie hier ihre Gleichungen:

Aufgabe gelöst:	Gruppe:	TeilnehmerIn: .... vom Betreuer auszufüllen

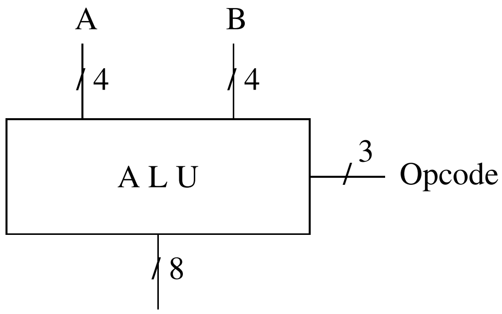
Der folgende Teil ist **optional**, falls Sie noch Zeit haben:

Offenbar ist das obige Verhalten für eine Fußgängerampel noch nicht recht befriedigend: Da wir mit vorderflankengesteuerten Flipflops arbeiten, wird der Eingang G auch nur zur Vorderflanke des Taktes ausgewertet. Ein Fußgänger müsste also den Druckknopf der Ampel gedrückt halten, bis im Zustand A eine Vorderflanke kommt. Wünschenswert wäre es, die Schaltung würde sich während der Zustände A, F, G und H merken, wenn der Knopf gedrückt wird und zwar unabhängig davon, was auf dem Takteingang geschieht. Zum "Merken" braucht man dabei offenbar ein geeignetes Flipflop.

Erweitern Sie ihr Design um eine entsprechende Komponente, übersetzen und testen Sie es.

<u>Versuch 4.4.b:</u> Entwurf einer einfachen ALU

Eine **A**rithmetical **L**ogical **U**nit (ALU) ist ein Schaltnetz, das abhängig von Steuersignalen verschiedene arithmetische und logische Operationen durchführen kann und außerdem Flags, d.h. Statusinformationen über das Ergebnis, liefert. Wir wollen hier eine einfache Vier-Bit-Alu entwerfen, die in Abhängigkeit von einem drei Bit breiten Opcode zwei Vier-Bit-Operanden in Zweier-Komplement-Darstellung verknüpfen kann und die vier Flags Sign, Zero, Carry und Overflow liefert:

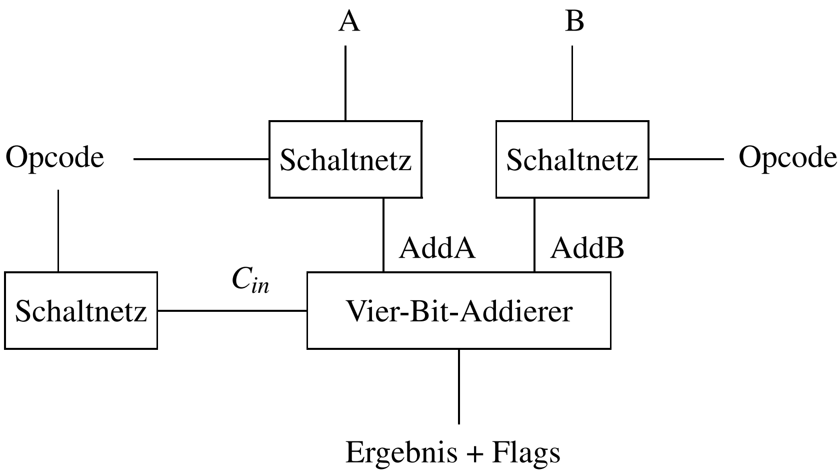


Unsere ALU mit den Eingängen A und B und dem Ausgang E soll folgende Operationen durchführen können, die sich so als Befehle auch in fast jedem Mikroprozessor finden:

Opcode	Operation	Ergebnis
0 0 0	Durchschalten (MOV)	E= B
0 0 1	Dekrementieren (DEC)	E= B $-$ 1
0 1 0	Inkrementieren (INC)	E= B + 1
0 1 1	Addieren (ADD)	E= A + B
1 0 0	Einerkomplement (COM)	E= $\overline{\text{B}}$
1 1 0	Zweierkomplement (NEG)	E= $-$B
1 1 1	Subtraktion (SUB)	E= A $-$ B

Zur Erinnerung: COM(0000)= 1111, aber NEG(0000)= 0000. Der Opcode 101 bleibt unbenutzt; die ALU kann dann ein beliebiges Ergebnis liefern.

Auf den ersten Blick sieht es so aus, als hätte man ein Schaltnetz mit elf Eingängen zu entwerfen, was mit KV-Diagrammen ziemlich hoffnungslos erscheint. Es wird sich aber zeigen, dass sich der Entwurf wesentlich vereinfachen lässt. Es genügt nämlich für die Operationen, die unsere ALU können soll, einen Vier-Bit-Addierer zu bauen, was ziemlich trivial ist, und kleinere Schaltnetze vor die Eingänge dieses Addierers zu setzen.



Man beachte hier auch den Eingang C_{in}, der dieselbe Funktion hat wie beim Vier-Bit-Addierer aus Versuch 1.6, den Sie sich noch einmal ansehen sollten.

Z.B. lässt sich der MOV-Befehl so realisieren, dass man AddA= 0, AddB= B, $C_{in} = 0$ setzt, für den COM-Befehl setzt man AddA= 0, AddB= $\overline{\text{B}}$, $C_{in} = 0$; für den INC-Befehl hat man sogar zwei Möglichkeiten: AddA= 1, AddB= B, $C_{in} = 0$ und AddA= 0, AddB= B, $C_{in} = 1$. Sie sollte die für Ihren Entwurf günstigere dieser beiden Möglichkeiten auswählen.

Vervollständigen Sie folgende Tabelle:

Opcode	Operation	AddA	AddB	C_{in}
0 0 0	MOV	0	B	0
0 0 1	DEC			
0 1 0	INC			
0 1 1	ADD			
1 0 0	COM	0	$\overline{\text{B}}$	0
1 1 0	NEG			
1 1 1	SUB			

Bestimmen Sie jetzt AddA, AddB und C_{in} als logische Funktion von A, B und dem Opcode. Dabei hängt AddA offenbar nicht von B, AddB nicht von A und C_{in} nur vom Opcode ab.

Programmieren Sie Ihren Baustein entsprechend mit Ihrer ALU und testen Sie sie.

Zur Vereinfachung sind in den GLBs B0 und B1 schon zwei Zweibit-Addierer als Makros definiert, die folgendes Aussehen haben

```
ADDF2(e0, e1, Cout,
      a0, a1, b0, b1, Cin)
```

Dabei ist **e0, e1** das Ergebnis der Addition (niederwertiges Bit e0), Cout der Carry aus der Addition, **a0, a1** und **b0, b1** die zu addierenden Zweitbit-Zahlen (niederwertiges Bit a0 und b0) und **Cin** ein Carry aus einer möglichen vorhergehenden Addiererstufe.

Die Eingänge A und B sollten Sie auf die Ausgänge Do1-Do8 legen, den Opcode können Sie über die Schalter T1-T3 einstellen. Eine Rumpfdatei liegt unter dem Namen `C:\SORC\LDF\ALU.LDF` vor.

Notieren Sie hier ihre Gleichungen:

Aufgabe gelöst:	Gruppe:	TeilnehmerIn: .... vom Betreuer auszufüllen

Der folgende Teil ist **optional**, falls Sie noch Zeit haben:

Was noch fehlt, sind die Flags, d.h. Statussignale, die die ALU liefert. Wir wollen hier folgende Flags realisieren:

- **Sign** ist 1, wenn das Ergebnis E (im Zweierkomplement) eine negative Zahl darstellt.
- **Zero** ist 1, wenn gilt E= 0.
- **Carry** ist 1, wenn man bei der Operation einen Übertrag aus der höchstwertigsten Stelle heraus erhält. Z.B. liefert 0001 + 1111 einen Carry, 0111 + 0111 nicht, ebenso sollte 1 - 2 (0001 - 1110) einen Carry liefern, 2 - 1 aber nicht.
- **Overflow** ist 1, wenn das Ergebnis E wegen eines Überlaufs falsch ist. Z.B. liefert 0111 + 0111 = 1110 einen Overflow, weil sich in unserem Zahlensystem das Ergebnis von 7 + 7= 14 nicht mehr in vier Bit darstellen lässt, sondern man das falsche Ergebnis -2 erhält.

Entwerfen Sie Schaltnetze, die diese Flags zumindest für die Operationen ADD und SUB korrekt liefern und erweitern Sie Ihre Schaltung entsprechend. Die ersten beiden Fälle sind fast trivial, beim Carry und insbesondere beim Overflow-Flag muss man etwas nachdenken.